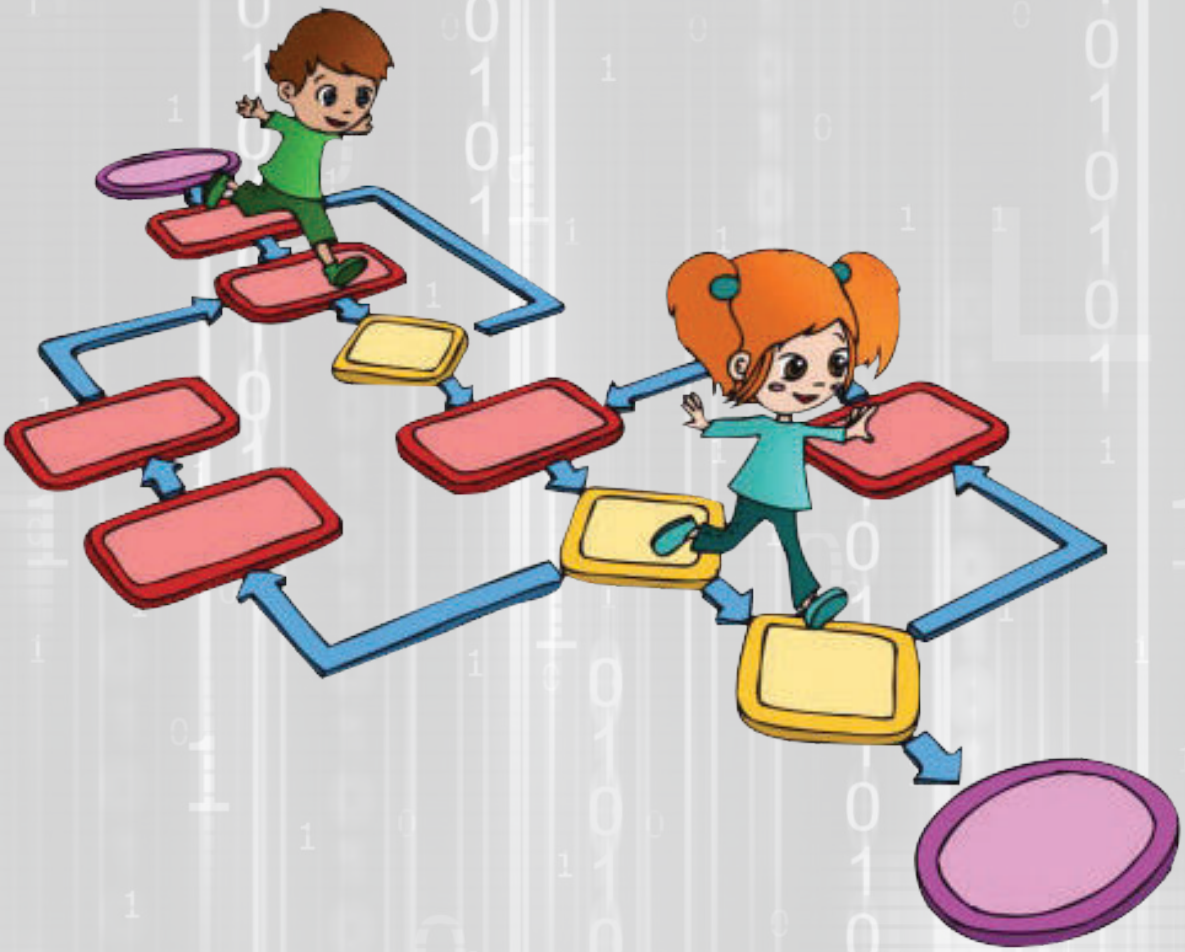
Claire Calmet, Mathieu Hirtzig and David Wilgenbus

1,2,3...
CODE!

Teaching computer science at primary and middle school



FONDATION La main à la pâte
POUR L'EDUCATION A LA SCIENCE

ISTIC
INTERNATIONAL SCIENCE, TECHNOLOGY AND INNOVATION CENTRE FOR SOUTH-SOUTH COOPERATION UNDER THE AUSPICES OF UNESCO

Claire Calmet, Mathieu Hirtzig and David Wilgenbus

1,2,3...
CODE!

Teaching computer science at primary and middle school

ISTIC
INTERNATIONAL SCIENCE, TECHNOLOGY AND INNOVATION CENTRE FOR SOUTH-SOUTH COOPERATION UNDER THE AUSPICES OF UNESCO

KEMENTERIAN TENAGA, SAINS, TEKNOLOGI, ALAM SEKITAR DAN PERUBAHAN IKLIM
MINISTRY OF ENERGY, SCIENCE, TECHNOLOGY, ENVIRONMENT & CLIMATE CHANGE

FONDATION La main à la pâte
POUR L'EDUCATION A LA SCIENCE

Éditions Le Pommier

The Original Edition
Copy preparation: Valerie Poge
Page layout: Marina Smid

ISBN: 978-2-7465-1106-4
8, rue Ferou-75006 Paris
www.editions-lepommier.fr

The English Edition
Translated from French by: Niamh O'Brien / Sans Contresens
Edited by: Mathieu Hirtzig, Fondation La main à la pâte
Project Advisors: Dato' Dr. Sharifah Maimunah Syed Zin and Academician Dato' Ir. (Dr.) Lee Yee Cheong
Project Coordinator: Mohd Azim Noor
Published by: International Science, Technology and Innovation Centre for South-South Cooperation under the auspices of UNESCO (ISTIC)

Perpustakaan Negara Malaysia Cataloguing-in-Publication
Calmet, Claire 1,2,3…Code! : Teaching computer science at primary and middle school / Claire Calmet, Mathieu Hirtzig and David Wilgenbus
ISBN 978-967-13199-2-5
1. Computer Science. 2.Government publications-Malaysia
I. Hirtzig, Mathieu II. Wilgenbus, David III. Title 004

PREFACE

Daniel Rouan
Member of the Académie des sciences
President of the Foundation *La main à la pâte*

It is a real pleasure for me to write an introductory note for the English version of the handbook “1, 2, 3… codez !”.
First because its amazing success in France demonstrates the quality and the usefulness of its content that deserves a broader international audience; therefore this translation is an excellent idea that I do warmly support.

Second, from a more personal perspective, I had the opportunity to put into practice a few of the activities proposed in the handbook with my grand-daughters aged 6 and 10, and I have been extremely pleased to discover how their interest was immediately captured and how their progress was obvious thanks to the well-constructed approach proposed by the authors. I should confess that, inspired by the book, my Christmas gift to the two girls was a pair of programmable tiny robots; and guess what? the least passionate to play programming the robots was not the grand-father!
Indeed, we are in a digital world and nobody will avoid this new dimension of everyday life, especially the young people who will be more and more immersed in it in the future, professionally and in many aspects of their activities.

Obviously, the school must take with an extreme seriousness the question of young children getting into the science of computing and programming. There are several reasons for that, both practical and ethical.
First it is essential that students have an early understanding and practice of what is actually computer science, so that they can become active players in the world of tomorrow instead of remaining only consumers of what is designed and built elsewhere, as stated by Gérard Berry in his preface of the second volume of “1, 2, 3… codez !”.
Second, by becoming familiar with the way information can be manipulated and distorted, the young people will be better equipped to deal with a flow of information where fake news and intrusion into private life are more and more actual threats.

This is what motivated the Foundation *La main à la pâte*, which was created by the French Academy of science to promote and improve science teaching through IBSE, when the project “1, 2, 3… codez !” was launched. I am convinced that the goal was indeed reached.

Clearly, computing science does not mean to be just familiar with surfing the web and using spreadsheets or word processing programmes. In most of the questions we ask, beginning with the word “how?”, the answer is an algorithm based on a few well defined principles. That is the reason why the concept of algorithm and programming is at the heart of “1, 2, 3… code!”. Activities are proposed in both plugged and unplugged forms; though using just paper and pencil, unplugged activities are not the less effective and captivating for children.

The handbook, intended for the teachers, proposes two types of content: on one hand a series of progressions for the classroom, with well-tested sequences and their attached hand-outs, and on the other hand a historical, scientific and educational background with a number of references and links, so that teachers can enlarge their vision of the question. This smart combination probably explains the success of the book.

I finally would like to warmly thank ISTIC and especially its former President Dato' Lee Yee Cheong, who initiated this translation of the handbook and mobilised the resources that were needed to accomplish this work in good conditions. The result is to my opinion a superb product that should be circulated worldwide, for the benefit of millions of children with, this is my deep wish, no exclusion of any social or gender category.

May 2018

FOREWORD

Dato' Dr. Samsudin Tugiman
Chairman
of ISTIC Governing Board

The publication of the English version of 1,2,3…Codez! is yet another milestone for ISTIC in its efforts to enhance the capacity of science educators particularly those teaching computer science education in the English speaking developing countries. This third publication by ISTIC on classroom resources in science education which is translated into English from French is again meant to be used by teachers in the classroom. The publications of the two earlier books "Discoveries in Islamic Countries" which focused on the contribution of science discoveries to modern science and the second book "When the Earth Rumbles" provide exciting activities and simulation of scientific principles related to earth movements through hands-on activities using IBSE as an approach.

Digital technology is increasingly being used in teaching and learning. Online learning and use of computer software have become a feature in many classrooms. The introduction of computer science in the school curriculum is therefore imperative as we move into the digital world. While it is important for students to be competent in the use of computers to enhance learning, it is equally important to teach students on problem solving through logical and creative thinking. Hence the rationale for including learning to write and read code which teaches logic and programming algorithms which tells the computer the exact step to take in the teaching of computer science in schools. This book precisely does that. As one goes through the book, one will realise that teaching computer science covers the logical thinking and creativity through both the 'unplugged' activities which involve teaching computing concepts without computers by using Inquiry-Based Science Education and 'plugged' activities which makes use of computers and robots.

As the world is fast moving into the digital age, developing countries cannot afford to be left out of the benefits of this new revolution.

I would like to take this opportunity to thank the La main a la Pate Foundation for giving the consent to translate this book into English and to supervise in the translation of the book. Special thanks goes to Dr. David Wilgenbus, one of the authors who introduced the book and the philosophy behind it. I also would like to thank our Honorary Chairman, Dato' Ir. (Dr.) Lee Yee Cheong for his persistence and perseverance in ensuring the translation on this book takes off. I am grateful to the Ministry of Science, Technology and Innovation for the financial assistance to the publication.

Finally it is ISTIC's hope that the book will reach out to teachers in the teaching of coding, programming and algorithm in computer science towards developing the future creators of technology.

May 2018

Authors and acknowledgements

"1,2,3...Code!", a teaching project designed by the *La main à la pâte* foundation, was produced with the support of the French Institute for Research in Computer Science and Automation (Inria) and the association France IOI.
The project was also financed by the French government's *Investissements d'Avenir* program — via the "Class'Code" project — Google, Microsoft and Mobsya. The authors and the *La main à la pâte* foundationexpress their thanks.

Coordination
David Wilgenbus (*La main à la pâte*)

Design and writing
Claire Calmet (*La main à la pâte*)
Mathieu Hirtzig (*La main à la pâte*)
David Wilgenbus (*La main à la pâte*)

With the support of:
Gilles Dowek (Inria)
Mathias Hiron (France IOI)
Florent Masseglia (Inria)
Elena Pasquinelli (*La main à la pâte*)
Pierre-Yves Oudeyer (Inria)
Martin Quinson (Inria)
Didier Roy (Inria)

Illustrations
Gabrielle Zimmermann (*La main à la pâte*)

The authors would also like to thank the teachers across France who tested this project in their classrooms, as well as the teacher trainers who assisted them. Their feedback was invaluable in the drafting of this teaching manual. Sincere thanks to Laurence Bensaid, Manuel Binet, Jean-Christophe Bizot, Cédric Blacha, Anne-Sophie Boullis , Olivier Cogis, Typhaine Collignon, Marik Cosson, Christelle Crusberg, Jeanne Daufin, Nicolas Demarthe, Catherine Dicky, Murielle Ducroo, Kévin Faix, Caroline Fayard, Marc Fouré, Olivier Gagnac, Vanessa Guionie, Anna Halatchev, Anne-Marie Lebrun, Catherine Le Frapper, Martine Lizambert, Anne Marigiano, Jessica Mazoyer, Anne-Hélène Montfort, Nathalie Pasquet, Pascale Priez, Fatima Rahmoun, Richart Terrat, Nicolas Thiéry, Carole Vinel, Emmanuelle Wilgenbus and Gabrielle Zimmermann.

Lastly, the authors thank Sophie de Quatrebarbes and Thierry Viéville (Inria) for their much-appreciated advice and support.

Contents

Introduction

The Benefits of Teaching Young Children Computer Science

All learning should enable children, adolescents (and adults!) to understand the world they live in and prepare them to play an active role in it. Recent transformations in how we communicate, our leisure activities, social interactions and production tools, for example, are intimately linked to advances in computer technology, to the extent that today we talk about the "digital world". This new world bears many hopes for technological progress, job creation, but also raises concerns, particularly in the field of ethics and privacy.

From preparing children for the jobs of the future, helping them understand the things and networks around them — so that they are not passively subjected to them but able to act on them — to making them aware of civic challenges and encouraging cooperation and developing their creativity, everyone should be taught to use computers, from as early an age as possible.

A consensus has begun to develop in recent years, uniting the scientific community and economic actors and policymakers. France introduced an optional course for final year high school students on computer and digital sciences, and now computer science is being taught in primary and middle schools too.

New Curricula Create an Enabling Environment

In September 2016, and for the first time in France, computer science made an appearance in the national education system's curricula for primary school and middle school.
While it is not yet identified as a separate subject, computer science is no longer restricted to simply using digital tools — as has been the practice for three decades through the ICT approach — and is now considered as a group of specific concepts and methods[1].
In addition to learning about these concepts and methods, computer science provides an excellent opportunity to conduct active lessons — either through inquiry-based learning or by projects — and therefore develops cross-curricular skills such as decision-making, reasoning, discussion, working independently, collaboration, etc.

Principles of the "1,2,3...Code!" Project

The "1,2,3...Code!" project created by the *La main à la pâte* foundation with support from the scientific community (in particular INRIA, the French Institute for Research in Computer Science and Automation) aims to introduce students and teachers to computer science, from kindergarten to end of 6th grade.[2]
It offers plugged activities (requiring a computer, tablet or automated device) that introduce programming basics and unplugged activities (computer science without a computer) that allow the teacher to address fundamental concepts in computer sciences (algorithms, languages,

1 See page 43 for more details

2 This teachers' handbook is designed for children in classes between the first and third Levels (from kindergarten to 6th grade). A second tome is dedicated to middle school (fourth Level).

how information is shown, etc.).
These activities are organized in ready-to-use progressions specially designed for each Level, with an emphasis on a multidisciplinary approach and active learning such as inquiry-based learning or project learning.

These progressions are easily adaptable to both computer-equipped (or with tablets or automated devices) or non-equipped classes.

Tools for the teacher and the student

The “1, 2, 3...Code!” project is based on two teaching manuals, amongst which the present one includes:

- 3 progressions for the class (Levels 1, 2 and 3)
- Ready-to-use lessons, tested in the classroom, divided into themed sequences for each Level;
- Handouts to be photocopied;
- Teaching and scientific insights to guide the teacher in carrying out the project;
- A bibliography for the teacher and for the students.

This teaching manual is accompanied by a dedicated website that offers a selection of resources available for download (specifically for coding) and a forum designed to assist classes throughout the project. The website is presented on page 342.

SCIENTIFIC BACKGROUND

Scientific Background

Just a century ago, computers did not exist. Today there are billions of them. These computers and other digital systems, namely networks, telephones, televisions, personal music devices, cameras and robots have changed the way we:

- Design and manufacture objects,
- Exchange information,
- Keep a trace of our past,
- Access knowledge,
- Practice science,
- Create and distribute works of art,
- Organize businesses,
- Govern countries,
- etc.

Computers have transformed everything because they are versatile. It is thanks to this object, the computer, that we can use computer assisted design software, modelling and simulation software, emailing and instant messaging software, file exchange software, video and music reading programs, digital mixing consoles, computer numerical control systems, encyclopedias, online classes, databases, blogs, forums and even digital archives. This versatility can be seen in the number of tools that computers have replaced: looms, calculators, typewriters, telephones, televisions, cameras and multimedia players, to name just a few.

In all of these applications, computers process information. That is, they systematically apply operations to symbolic objects. Looking up a word in a dictionary, encrypting and decrypting a secret message, addition and multiplication of two numbers, producing timetables for students in a high school or for airline pilots, calculating the area of a farming plot or even counting a tarot player's points are all examples of information processing.

A systematic process that enables information to be processed is called an algorithm. Computers are in fact algorithm-execution machines and computer science is the field of scientific, technological and industrial activity concerned with the automatic processing of information by machines.

This scientific insight on computer science begins with the history of the discipline and then identifies the four fundamental concepts: algorithm, language, machine, and information.

A brief history of computer science

The story of computer science is the coming together of two schools of thought: the first focusing on the notion of algorithm and the second on that of machine. In the 1940s, these two notions were brought together to create the first computers, or algorithm-execution ("computing") machines. Other schools of thought that focused on language and information concepts were added to this mix.

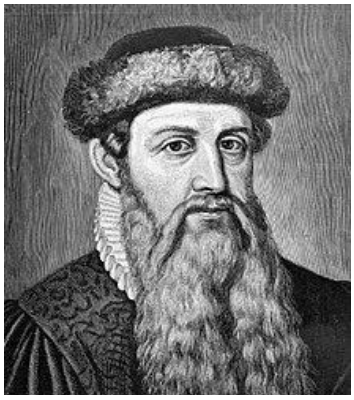
The history of the algorithm dates to the beginnings of writing, as the first traces of writing discovered were accounting calculations. The scribes of Mesopotamia therefore knew algorithms to add and multiply numbers, as well as for more complex operations such as compound interest calculations and inheritance taxes. It is possible to go even further back in history if we expand the concept of an algorithm to include systematic methods that are applied to things other than information, which means the production methods for bronze or pottery, cooking recipes and even the process for weaving fabrics can be seen as algorithms. The story of the machine is a very old one too, even if we restrict it to the machines that operate using information. The machines of Heron of Alexandria, cathedral bells, calculators invented by Schickard, Pascal and Leibniz, and the Jacquard loom all contained elements of information processing.

The history of computer science is a prolific one, spread across several centuries. What follows is a selection of historical figures associated with its story and who made great strides in its development. This historic approach helps explain computer science concepts in a way that is appealing and easy to grasp. However, the story must be accompanied with a word of caution, as not all contributions can be singularly attributed to one person. Discovery was an incremental process, with each "inventor" adding to what was produced by their predecessors and often their contemporaries. This caveat is intended to point out that these people, while geniuses in their own right, were not the sole inventors of these contributions, but we can evidently appreciate the major importance of their work as building blocks in a process.

Muhammad ibn Musa al-Khwarizmi

Al-Khwarizmi was a Persian scholar believed to have been born in the 8th century, circa 780. He lived in an empire under the reign of the Caliph Al-Ma'mun, which encouraged the study of science and the arts. With other scholars of his era, he worked in the House of Wisdom, a school founded by the Caliph in Baghdad for the study of geometry, algebra and astronomy. He began by translating Greek and Indian scientific and philosophical manuscripts. With this work, he passed on the knowledge of these civilizations, such as the decimal positional number system, to his peers. Al-Khwarizmi then went on to write his own scientific texts, specifically his book "Kitâb al-jabr wa al-muqâbala" ("The Compendious Book on Calculation by Completion and Balancing"), in which we can hear the sound "al-jabr", the origin of the modern word "algebra". His desire was to provide the tools that would solve the problems of daily life. It is best if we take Muhammad Al Khwarizmi's own words to explain how his work was aimed to facilitate calculations "that people need for inheritance, donations, sharing, judgements, businesses and in every transaction between them on land surveyings, digging canals, geometry and other things related to its aspects and arts (...)." The mark that Al-Khwarizmi left in history is closely tied to the history of mathematics, thanks to his work in algebra and geometry. But his name is also intrinsic to computer science, because

each procedure that he describes to solve a problem is a series of instructions to manipulate numbers. What's more, Al-Khwarizmi's name was transcribed as "Algorismi" in Latin, which later produced the word "algorithm."



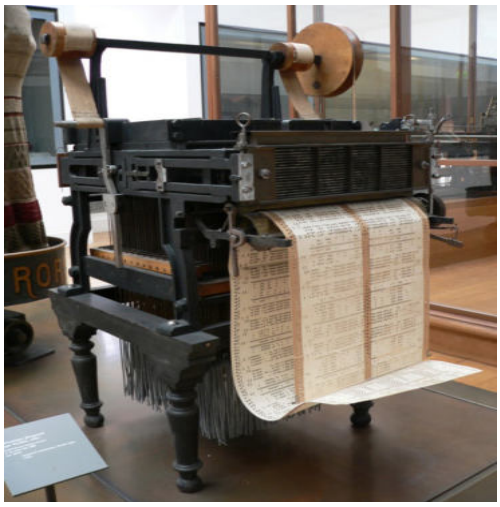
Johannes Gutenberg

Johannes Gutenberg was born in around 1400 in Mainz, Germany. He trained as a goldsmith, and usde is metalworking experience to help further his invention: the printing press. However, the story of the printing press goes further back than Gutenberg. This story, which demonstrates the importance for humanity of ensuring that great works withstand the ages and are disseminated, dates back to the 7th century. Prints were previously made using woodblocks to reproduce texts and images on pages made of paper or fabric. Processes that meant identical copies of pages could be reproduced cut down on manual labor and saw great progress. A key moment of progress was undoubtedly the invention by Bi Sheng of movable type, the first traces of which were discovered in China circa 1040. They were first made of terracotta and then metal began to be used. In Europe, Gutenberg applied a principle common to the history of inventions: he perfected existing techniques and combined them. He created a metal alloy that proved highly suited to printing: type metal. He came up with the "printing press," which was more efficient than existing techniques for applying the page to the platen. Lastly, he developed an ink that was thicker and more suited to printing. His invention encountered great success and rapidly spread across Europe.

Printing, although it did not directly influence computer science, has some interesting ties to the subject. Firstly, among printing innovations, let us look at movable type. This allowed printers to fill a page with their choice of characters, which was reversible, whereas previously the full page had to be printed and could not be changed. The platen could therefore be reconfigured. Printers could also provide a plate for another press which could then (re)produce the same page. In a way, simply rendering a machine configurable could be seen as a distant precursor of programming, which appeared much later in the Jacquard loom, Babbage's machine, and lastly in computers. Another significant connection to computer science can be seen in the cultural revolution that the printing press initiated. An estimated one billion books are thought to have been printed in the 18th century, which completely changed access to knowledge and therefore social equilibrium.

Joseph Marie Jacquard

Joseph Marie Jacquard was a French inventor and mechanic, born in 1752 in Lyon. After working in his father's workshop as a weaver he therefore grew familiar with looms, which enabled fabric to be produced. The loom was widespread at the time and a source of employment, but it was a complex machine to maneuver. Several workers were needed to pass the shuttles (which spooled the threads) back and forth at the right time. Jacquard wanted to automate the loom to make it both easier to use and more efficient. He adopted the punched paper tape system that Basile Bouchon added to the loom and Jacques Vaucansons' proposals for an automated process.



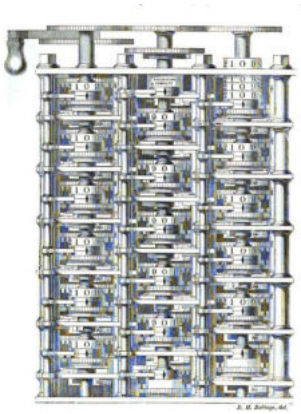
Jacquard mechanism (extracted from a loom) on display at the Musée des Arts et Métiers in Paris

By improving and combining these two techniques, he contributed to the weaving loom's expansion. The punched card system offers a solid medium which stores the instructions for passing the shuttle. The series of punched cards are placed in the loom, and each card stores instructions for a different step of the fabric production. When a punched card passes through the loom, the holes in the card change the position of the threads, guiding the shuttles through. Each punched hole line corresponds to a different step, and a specific design on the fabric. By increasing the number of punched cards, it is possible to write the instructions to obtain the complete design on the full width of the fabric. Just like Gutenberg's press, the loom became configurable. A set of cards can be identically copied and used on another loom, which produces the exact same fabric with the same design. The same loom with a different set of punched cards produces another fabric with a different design.

Jacquard's invention meant more fabric could be produced much faster. It was extremely successful, with over 20,000 looms built. Since it uses punched cards to give instructions to a configurable machine, this invention is considered to be one of the computer's ancestors. Jacquard's work is a good example of the usual innovation process. Jacquard sought inspiration in the most advanced technologies of his era, for example with Basile Bouchon's punched paper tape. He adapted and improved several inventions to combine them into a single mechanism, easier to install and to use. The major progress made by Jacquard with the loom meant it could be operated by a single weaver, instead of six previously. These workers, known as "Canuts," subsequently launched a revolt due to the lack of work often attributed to this invention.

Charles Babbage

Charles Babbage, born in 1791, was a British mathematician and inventor. At that time, long before the invention of GPS, sea navigation was guided by observing the positions of stars in the sky and comparing them with their expected position at that date. This method enabled sailors to determine the boat's position on the globe. This required calculating the map of the sky for each day and for many weeks ahead. Unfortunately, these calculations were time-consuming and complex, and mathematicians who perform them were at risk of error, which would cause accidents at sea. In 1821, Charles Babbage presented a proposal for a machine to the London Royal Astronomical Society, which would speed up calculations and make them more reliable: "the difference engine." The aim of this machine was to automatically solve, through mechanical processes, complex calculations which



Drawing of Babbage's difference engine

would predict the stars' positions. Unfortunately for Charles Babbage, his machine required extremely fine and precise mechanical parts, which were difficult to manufacture at that time. He did not succeed in building his machine, but this did not prevent him from inventing a second, even more powerful machine. Babbage wanted this second machine to do more than just calculate the position of the stars. He wanted it to be able to execute any task that would be "described" to it. To do this, inspired by Jacquard's work, he designed the "task description" (a series of calculations) to be performed with punched cards. Babbage may not have fully realized the potential of his invention. It was Ada Lovelace who clearly understood everything that this "analytical engine" would make possible. Unfortunately for Babbage, who ran out of funding, the second machine was not made either. Nonetheless, this machine incorporated all the fundamentals of a computer, which could be programmed by punched cards that gave it the instructions to execute, as was the case for the first computers that appeared a century later.

Ada Lovelace

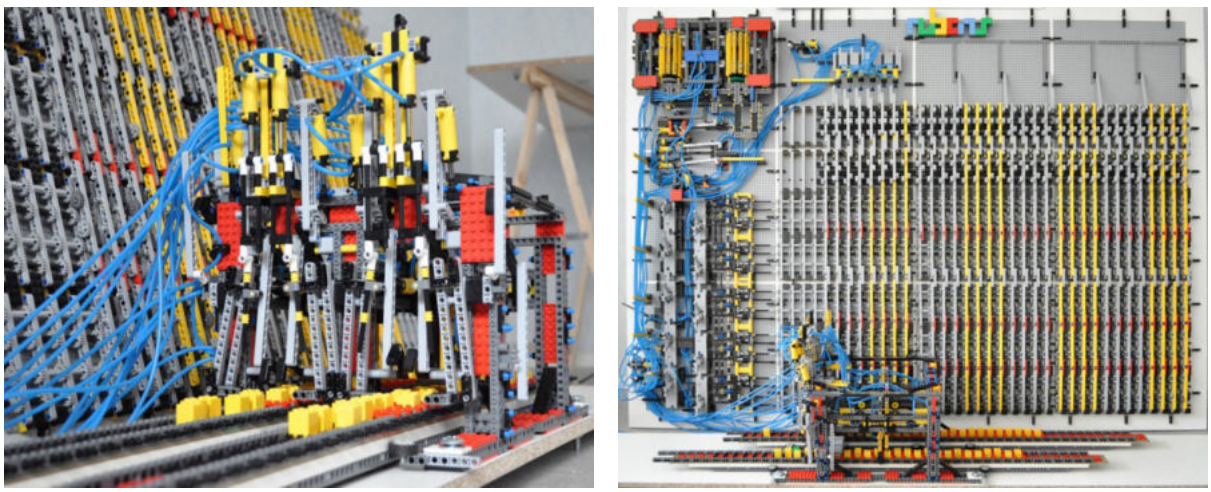
Augusta Ada King (Ada Lovelace), was a British mathematician born on December 10, 1815 in London. She was the daughter of Lord Byron, the British poet, whom she never knew. Her mother, who wanted Ada to have a high-level science education, ensured she studied mathematics in particular. This was quite unusual for a young lady during that time. When she was 17, Ada met Charles Babbage and was passionate about his work on the difference engine, followed by the analytical engine. Ada Lovelace and Charles Babbage formed an excellent scientific duo. Ada wrote up thorough documentation on Babbage's machine and supported him throughout his financial difficulties. He helped Ada deepen her mathematical knowledge, giving her recommendations to study with the most esteemed mathematicians in the kingdom.

While Babbage was planning for his machine to be used for digital calculations, Ada had already imagined a more universal use for it. At the time, numerical calculations were already completed automatically by machines such as the Pascaline: this was a series of wheel dials that could carry out addition and subtraction only. Babbage saw his machine as a sort of "programmable calculator": first, you could explain to the machine what calculation it had to do (with instructions, i.e. the "program") then provide it with the values to which it would apply these instructions.

Ada, a visionary, understood the potential that this machine had, even more than Babbage himself did. In her manuscripts, she describes the possibility for this machine to manipulate numbers, but also letters or any other type of symbol. She explains how this machine could be programmed for tasks other than numerical calculation, such as musical composition. In her notes, Ada describes all of the instructions that must be given to Babbage's machine so that it can perform a specific calculation: the series of successive terms in the Bernoulli sequence. In doing so, she became the first person in the world to write a computer program. The first computer programmer was indeed a woman!

Alan Turing

Alan Turing was a British mathematician and computer scientist, born in 1912.
He is often presented as the founding father of computer science — although it is undoubtedly a vain attempt to seek such a founding father to something that was created collectively.
He made many contributions to early computer science. In the 1930s, he came up with one of the first precise definition of an algorithm. His aim was to study what was "calculable."
Multiplication, for example, is calculable because it can be performed in a certain number of steps. But what are calculable problems? This was a difficult question, that was a popular topic of study in the 1930s, and Turing focused specifically on it. While studying the issue, he came up with an abstract machine — or a machine that was not intended to be really built — but which he would use to structure his reasoning. Imagining how it functioned enabled him to better study what was calculable and what was not. This machine was described very simply. It used a tape which contained a series of boxes that stored data (zeroes and ones).
The machine was able to read what was in the boxes, move from left to right on the tape, and write in a box. With this machine, Turing showed that it was possible to replace any other machine, by writing the "program" of the replaced machine on the tape (for example, multiplication) followed by the "data" to which this program would be applied. This method of combining program and data in a machine was a key step and today exists in all computers.
During World War II, Turing played a crucial role in building electromechanical machines that could decrypt the secret codes used by the German military, without knowing the key. After the war, he helped design the first British computers in the National Physical Laboratory, followed by the University of Manchester.



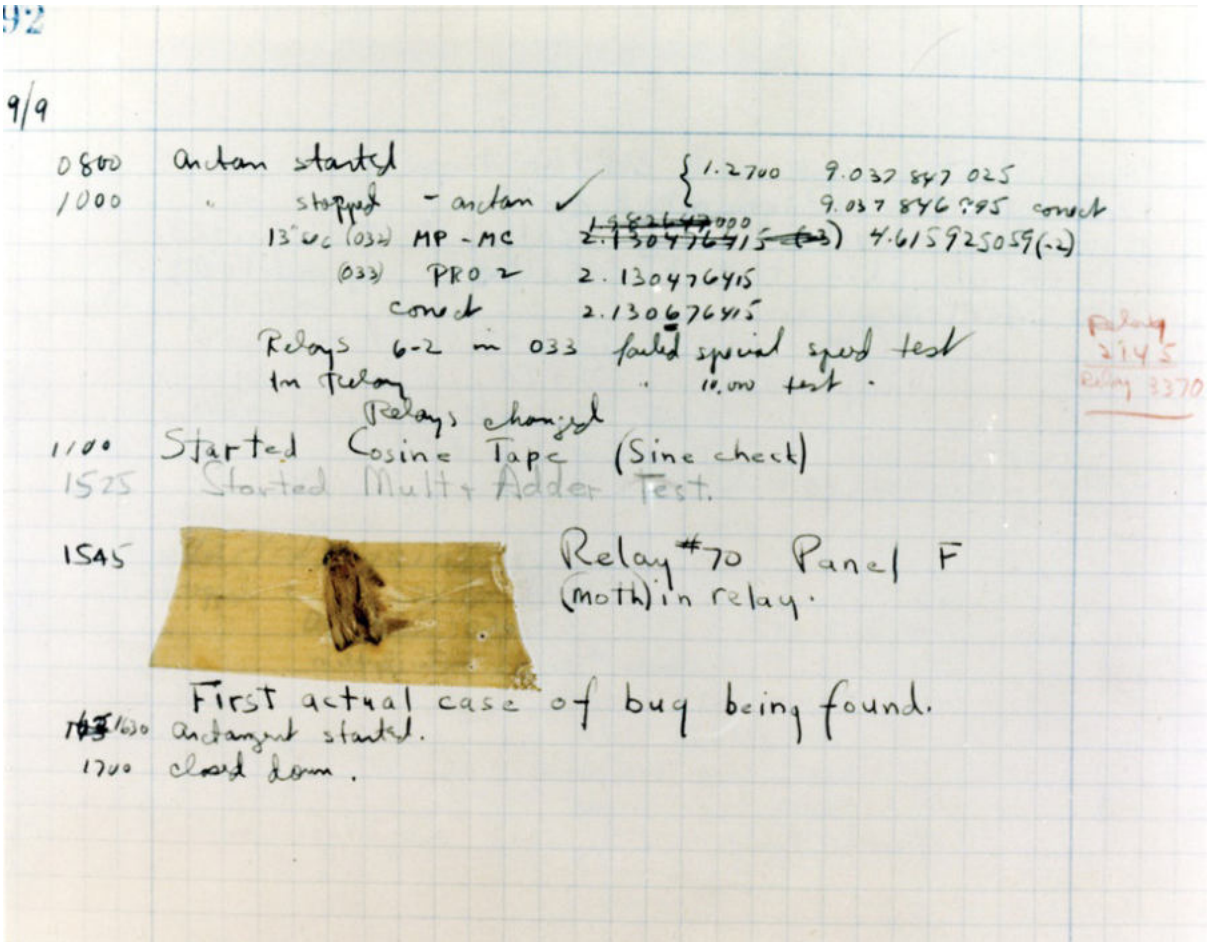
Lego® Turing Machine (Rubens project, ENS Lyon): overall view (left) and detail of the reading and writing head (right).

He also came up with many other ideas about the phenomenological nature of intelligence, on morphogenesis, the approximate solving of differential equations with a computer, and program proofs. The last example raises the following issue: we can conduct tests on a program to check its stability and accuracy. If we find an error, then we try and fix the program; but if we do not

find an error using this method, does this mean there is no error or that we were unable to find it? Is there mathematical proof that the program is truly without error?
Sentenced to chemical castration because of his homosexuality, considered a crime in England at that time, Alan Turing committed suicide at the age of 41.

Grace Hopper

Grace Murray Hopper was an American mathematician, born in 1906. During her childhood, she showed a great interest in science and technology (for example, by disassembling the alarm clocks in the house one by one, until she could put them all back together at the age of 7). Her parents encouraged her taste for the sciences while ensuring that her choices were nurtured, rather than following tradition in terms of the education of young American girls. Grace therefore became one of the few women of her era to study at Yale University and obtain a doctorate in mathematics.
During World War II, she joined the navy as a lieutenant and joined the team working on Mark 1, the first computer in the United States. Grace Hopper was a member of the first group that could program that machine. Mark 1 received instructions and data from punched cards, as was planned for Babbage's machine, and the Jacquard loom nearly two centuries before. In the 1950s, Grace Hopper believed strongly in the idea that computer science should be accessible to as many people as possible, not just computer specialists and mathematicians. She argued that punched cards were hampering the development of computers and began working on a programming language project, which would be close to English and make program writing easier. The difficulty was to design a program capable of transforming high-level instructions (in a language similar to English that would be easily understood by a human) into low-level instructions (the computer's language). Today, this type of program is called a "compiler", and its invention is partly thanks to Grace Hopper. Her programming language, known as COBOL, became the most widespread between 1960 and 1980.
Grace Hopper is not just famous for making computers more accessible to non-specialists. One day, when she was programming her computers with punched cards, one of them failed. When she went to find what happened, she found a moth stuck in one of the holes of a punched card. She stuck the it into her report, with a comment indicating that the insect was at fault for the breakdown. The term "bug" had already been used by scientists, such as Thomas Edison, to identify an error due to the undesired interference by one of them in their experiments. By applying it to the failed execution of a computer program, Grace turned this into a household term to describe this situation.



Grace Hopper discovers a “bug” in 1947

A brief history of the Internet

The idea of connecting numerous computers in a network emerged in the U.S. in the 1960s. In 1961, researcher Leonard Kleinrock developed a way to share data between computers by breaking the data into packets, sending them along a variety of different paths, and piecing them back together in the right order at the receiving end. This is the principle behind the method still used today to send data — such as emails — on the Internet (see illustration page 29). The following year, Joseph Licklider promoted the computer as a communication and resource-sharing tool, and proposed building a network of interconnected computers. Licklider joined the Advanced Research Projects Agency (ARPA), a state body founded in 1958 to guarantee the United States' military and technological superiority, following the U.S.S.R.'s launch of Sputnik I. He became the director of ARPA's Information Processing Techniques Office, and it was while working there that he proposed building an experimental network using the principle of data transmission by packets. This network, named Arpanet, was deployed in 1969. Only four computers were connected, exchanging data at a speed of 50,000 bits per second (6 kb/s).

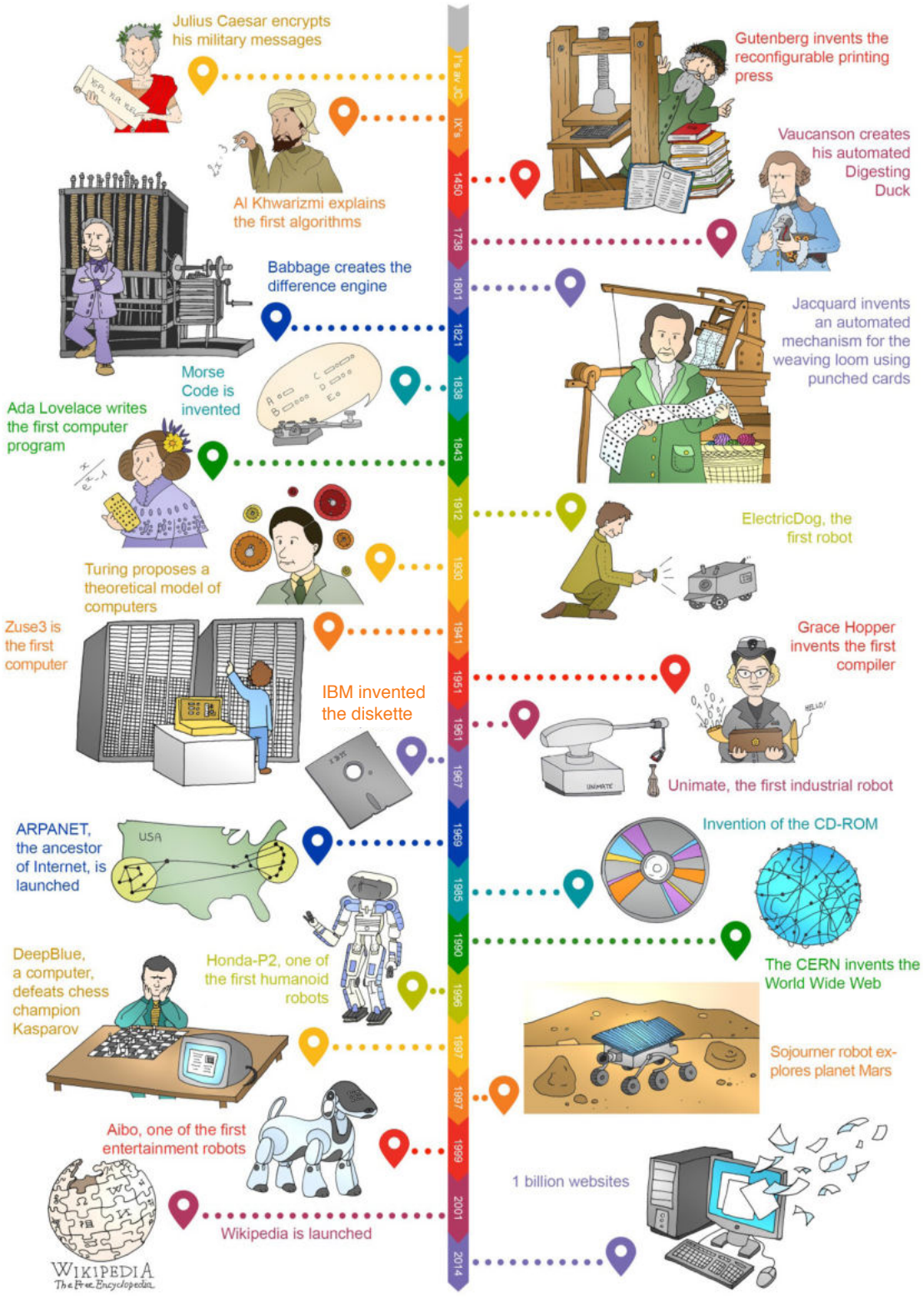
The first Arpanet users developed programs that enabled messages to be sent from one computer to another via the network. The first email, sent in 1972, already featured the character "@" to separate the addressee's name from their address. The key functions of electronic mailboxes — sort, reply, forward — were invented at this time.

The first public demonstration of Arpanet took place in 1972, at an international conference. It was a great success which contributed to the project's expansion, the creation of other networks and inspired the idea of interaction between networks. Such interaction required modifications in the way Arpanet ran: the solution developed was the TCP/IP protocol, still in use today on the "network of networks" that is the Internet.

In 1989, Tim Berners-Lee, a CERN network (Cernet) user, wanted to make browsing the network easier by using a browser and hypertext links. He proposed creating a web that would allow users to switch between contents via multiple paths. This project, approved by the CERN, was named the World Wide Web. It is one of the Internet's networks.

Projects began to abound from the 1990s on, surrounded by stock market speculation, until the dot-com bubble burst in 2000. The first web browser with a graphic interface was launched in 1993. It was soon replaced by others, which were in turn replaced. The Yahoo! Internet directory was created in 1994, before becoming an international portal. Google began as a start-up in 1998, followed by e-Bay and Amazon, today the giants of the Internet.

Timeline



The lesson “Review: Defining computer science” (page 328) allows students to draw up a similar timeline using documentary study on the history of computer science.

Algorithms, languages and programs

The ambiguity of the word “code”

“Code” is currently a buzzword. For example, it is used in the title “Read, Count, Write and Code” in the introduction to computer science in school curriculums. As this word can be ambiguous, we shall discuss it here.

The first of these ambiguities is the easiest to clarify. Sometimes, the word “code” is used to refer to the modification of a message to render it unreadable to whoever is not equipped with the “secret code.” At a later stage in this handbook, this action will be referred to as “encryption,” which is the act of “encrypting” a message to render it unreadable to whoever does not possess the “encryption key,” which allows the message to be “encrypted” and also “decrypted” so that it may be read. In general, the word “code” refers to the action of giving a machine, usually a computer, instructions. This activity can also be described using the term “programming.” The instructions are then interpreted by this machine which executes them to produce a result.

Coding can also be used to represent information with symbols. For example, writing a text in binary, using zeroes and ones. Depending on the situation, we shall use the terms encode or decode. In this teaching handbook, “to encode” is the meaning of the verb “to code”, even though the word is commonly used to refer to programming.

The word “code” has a second ambiguity, which is unrelated to its definition but rather the perceived notion of computer science through the activity of programming. Computer science is much more than just programming, although programming is perhaps the most visible activity in this field. In popular culture, it evokes the power of the programmers, and the distinction between those who deal with machines as users (even tacitly) and those who make them operable. An example is the often-exaggerated image of the programmer in a film rapidly producing lines of code. Another is the image of a screen with full pages of code scrolling in the background of a report or documentary on computer scientists, most likely because this creates an impressive, or highly specialized, effect. It is possible that this image contributes to spreading the notion that computer science is hard to access and reserved for an elite or minds that are predestined for it. Fortunately, this is not the case. Of course, coding is entirely related to computer science, but it is just one of a series of cogs in the system. Coding does of course render concepts visible. But these concepts can be created and developed outside of any programming, using reasoning as the main driver. This reasoning, together with the mechanical nature of computers and their functioning, produces computational thinking, and this is how the algorithms that enable much of our world to run today are developed.

What is an algorithm?

There is no need to prove that algorithms are everywhere. When we enter a request in a search engine, several algorithms are at work to find the most relevant pages possible. When thousands of financial market operations are completed per second by a single trader, it is not the human trader giving the orders, but a high-frequency trading algorithm. When a government wants to analyze vast masses of data concerning human activities in order to detect suspicious behavior,

they use algorithms[3]. Algorithms play a major role in our lives, often without us realizing.
To illustrate the algorithm, the recipe example is quite useful. It is, after all, a series of instructions to apply in order to obtain a result. Analogies between a recipe and an algorithm can be taken even further. A recipe can be shared around the world, or it can be kept a secret. It can be hand-copied or passed on verbally. However, until it is incarnated by actions in the kitchen, it remains a simple concept. In computer science, this concept is an algorithm. It can be hand-written, to varying degrees of precision, combining explanations and mathematical symbols. If it remains in this format without being applied, it is knowledge that yields no results. The power of an algorithm can be seen when it is interpreted and concretely executed, either by a human or a machine.
Speed of execution is another issue. A simple algorithm can be interpreted by a human being by following the steps and writing down the intermediate results, if any. One example is the division algorithm that students can systematically apply once they have learned it. They can then execute any division by hand by rigorously applying the steps of the algorithm. They write down intermediate values and start the operation again until the conditions to stop the calculation are achieved. For a more complicated algorithm, this can be difficult and for most of the algorithms that are currently being executed around us, it is impossible to envisage. Machines can execute these algorithms at increasingly faster speeds every year. For this to be possible, at least two conditions must be met:

1. Create and write an algorithm that achieves the desired result.
2. Translate this algorithm into a program that the machine can interpret.

These two activities are central to producing digital objects. Programming may be produced at different levels of quality, depending on the programmer's experience and knowledge of the language used. A program can be written several ways while following the same algorithm. Certain programmers know their preferred languages so well that they can work extremely efficiently, writing programs several times faster than programs written by beginners. However, this is not always enough to obtain efficient programs.
This is because algorithms can also be written several ways. This is simply due to the fact that they can be designed from several perspectives. An algorithm might be particularly well-designed but it can also be basic and use the wrong options. As we will shortly discuss, a poorly designed algorithm cannot be compensated for with efficient programming or a high-performance computer. A well-designed algorithm, on the other hand, even if it is programmed by a beginner on an outdated computer, can offer radically superior performance. This conception of the method, or the reasoning that produces the most efficient algorithm possible, is a crucial element in computer science. The corresponding discipline is called algorithmics. Let's now look at some of the basic aspects of this field, in light of the brief history of computer science that we have just discussed.

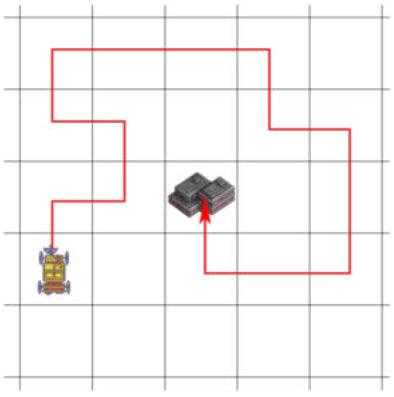
Languages

The first computers were programmed with punched cards. This is what Babbage had planned for his analytical engine. As we have already mentioned, Grace Hopper wanted to make programming more accessible and saw the limits of punched cards as an interface between the machine and those who programmed it. To push beyond these limits, she contributed to

3 The relevance of algorithms from this perspective is not the issue here, but algorithms are indeed envisaged to process the vast quantities of data that cannot be analyzed "by hand".

the creation of compilers. This involved writing a program in a language closer to human language. The challenge was translating programs into the language of the machine so that it could execute the corresponding actions. This translation became known as “compiling”. The program that executes the compiling is a compiler. To compile a program, there are many challenges to overcome, such as:

- Recognizing and separating words and language elements to create units that can be subsequently used. For example, the line of code result=a+3 must enable the variable result, the operator =, the variable a, the operator +, and the number 3 to be recognized.
- Detecting inconsistencies in the language and ensuring the program is correct from a syntax and grammatical point of view (i.e. ensure that the program respects the rules of the language in which it is written).



Which language can be used to command this robot?
(extract from teaching progression in Level 3, page 207)

- Detecting semantic inconsistencies. For example, if two variables have the same name, it will be impossible to translate the program for the machine, as there is an ambiguity that makes compilation impossible.
- Effectively generating code, in the machine’s language, from the elements constructed in the previous steps.

Compilers were a major advance in the history of computer science. It is obviously much easier to use words, numbers and operators to program, rather than enter the instructions one by one for the machine on a punched card. Today this contribution is seen in the multitude of programming languages that exist, such as *Scratch*, the language we have chosen to use to introduce students to programming. These languages can vary greatly. In the teaching project proposed for Level 3 students, one lesson (page 207) illustrates the diversity of languages that can be used to guide a robot: autocentered and allocentered languages can be used (the Pixees website offers a robot game, available at https://pixees.fr/?p=724).
Programming languages have extremely limited and highly specific vocabulary, with no ambiguity whatsoever. In our example, the robot can be guided with four allocentered (or absolute) instructions: North (meaning, “go forward one step North”), South, East, and West. In an autocentered (relative) language, the robot can respond to three instructions: forward (take one step forward straight ahead of you), right (meaning, “turn 90° to your right”), and left. We can even further simplify the left instruction to right-right-right, which would reduce the vocabulary needed to just two elements. It is possible to switch from one language to another, but it is not an easy exercise, and attempting it in the classroom is a good way of showing how important compilers are!
Once the major contribution of compilers — and therefore programming languages — is established, all that’s left is to program! To do this, the best way is to begin by properly designing the algorithm before translating it into a program (and the compiler follows by translating this program for the machine). Let’s look at how these famous algorithms are created, what they’re made up of and most importantly how we can produce them!

Algorithms, reasoning and computational thinking

We know today that all algorithms can be written with a few basic instructions that are assembled using control structures.
The most common basic instruction is attributing a variable, for example, a=3, where the execution involves attributing the variable a to the value of 3. This idea of attributing a variable therefore requires understanding the concept of variable: a box that contains a value that can be used and modified while the program is running.
The number of times a video was viewed on a streaming platform is an example of a variable. Every time the video is viewed, the "number of views" variable is modified (for example, the instruction number of views = number of views +1 means we can attribute the new value, which is one more than previously, to the variable "number of views".)
Other basic instructions are displaying a message on a screen, moving a sprite, etc.
These basic instructions are then assembled with control structures, which enable complex instructions to be created by assembling simpler ones. The main instructions are:

- the sequence: perform one instruction followed by another;
- the test, which enables a specific instruction to be performed, depending on the condition. For example, in a washing machine, if the weight of the laundry is less than two kilograms, start the energy-saver program, otherwise start the standard program.
- the loop, which enables an instruction to be repeated several times.

But other control structures exist, for example in event-driven programming (particularly in *Scratch*), an instruction is triggered following an event.
The first computers were capable of executing algorithms made up of these elements. These machines are therefore capable of executing any algorithm in the world. They are universal machines. We can be certain that for the pioneers of computer science, seeing algorithms come to life in universal machines must have been exciting.
Just like these pioneers, beginner programmers discover the capabilities of their computers progressively. They begin with simple, well-known algorithms such as mathematical operations (addition, multiplication, etc.). Then they move on to more complex algorithms, reproducing mathematical sequences such as the Fibonacci sequence or the Bernoulli numbers. It was in fact while writing a program to calculate the Bernoulli numbers on Babbage's analytical engine that Ada Lovelace became the first person in the world to write a computer program.
There are lots of other programs. The universal machine allows us to apply known algorithms, but also lets us look for algorithms to solve problems that we do not yet know how to solve. However, does simply discovering an algorithm necessarily mean that this algorithm will always provide a solution? To delve deeper into this question, we will look at "computational thinking."
Computational thinking is a body of knowledge and thought processes used to understand the world we live in and help us guide our actions. It covers a vast range of concepts, and we will try to explain some of them here[4].
Everyone, every day, uses algorithms — sometimes without really realizing. We have established strategies, habits and reasoning that we apply to our daily activities. In a supermarket on shopping day, we can choose to walk up and down every aisle checking if it contains the product on our shopping list. But why would not we make life easier by writing this list beforehand, according to our knowledge of the store, so that the products are listed in the right order? Going to work, we can plan our journey depending on the day of the week, because everybody

4 The article "What is Computational Thinking" on Computer Science for Fun http://www.cs4fn.org/computationalthinking/ offers a description of what this expression covers.

knows that on Tuesdays and Thursdays, Kolmogorov Street is completely jammed, and to be avoided. Lastly, while looking up a word in the dictionary, we open it at the most likely spot depending on the position of the first letter of the word in the alphabet (near the beginning for the letter "C," near the middle for the letter "P," etc.). Then, by trial and error, we flick backward or forward a few pages, according to the letter at the start of the words on the opened page. When the first letter of the word is found, we start again with the second letter, and so on.
Computers can help us process these daily tasks. We must iterate the problem, then describe an algorithm based on the four elements mentioned above (instructions, loops, tests, and variables) that can solve the problem. So that these algorithms can be automatically executed, they have to be translated into programs so that a machine can run the instructions. Part of computational thinking means rationalizing these processes so that a machine is capable of executing them.
We will now have a look at the kind of problems we can solve using computational thinking and algorithms. There are problems that can be solved with algorithms inspired by human reasoning. These are algorithms such as drawing up the shopping list in the order of the store's aisles. Many algorithms are created this way. The person that develops the algorithm turns their own thinking process into a written process. It is like explaining a method to someone while making sure that there will be absolutely no error in how it is interpreted. There are also problems that can be solved with algorithms inspired by human reasoning or practices, but which are made stricter and more general by adding a study to the approach. Let's take the example of looking for a word in the dictionary. There is a very similar exercise which consists of looking for one's name or surname in a list (for example, to see if we have been successful in an exam). Let's try and iterate the way this task is carried out.
Imagine an unsorted list of a few dozen names. Each person will take a few seconds to find their name (or make sure they're not on the list). Next, if the same list is provided in alphabetical order, the exercise is much quicker.

The difference in speed is because when the list is sorted, a much more efficient method can be used than with an unsorted list. When it is not sorted, we look all over at random, or else we read through a list systematically from start to end, checking names one by one. However, when the list is sorted, we can see the place where the names that begin with the same letter as ours are found. Followed by the second letter, and so on.
This approach works, but it can be perfected. If there is a very large number of objects (like in a dictionary, or a country telephone directory with millions of entries), we can make huge leaps forward or backward to save time. There is an extremely efficient algorithm for searching a sorted list, the most efficient known algorithm for this problem, and it is called a "binary search". To find an item in a sorted list, we cut the list in the middle. If the item was in the middle, we'll find it; otherwise we know in which half of the sorted list it is to be found. We can therefore start afresh with the binary search on the remaining half-list. And so on. At every stage, the list we are searching is twice as short.
A list of 20 names can be searched in four attempts at most. The number of operations in the worst case can be found by the number of times that 20 can be divided by 2 before we arrive at 1 (the first division can be rounded down if there is an uneven number of lines). Without a binary search, in the worst-case scenario 20 iterations would be necessary to successively read each name from first to last in order to realize our own was not there (the algorithm testing all thoroughly all possibilities is called "brute force"). The power of the binary search can be quickly seen when the lists are very long. A list of 300,000 items can be searched in

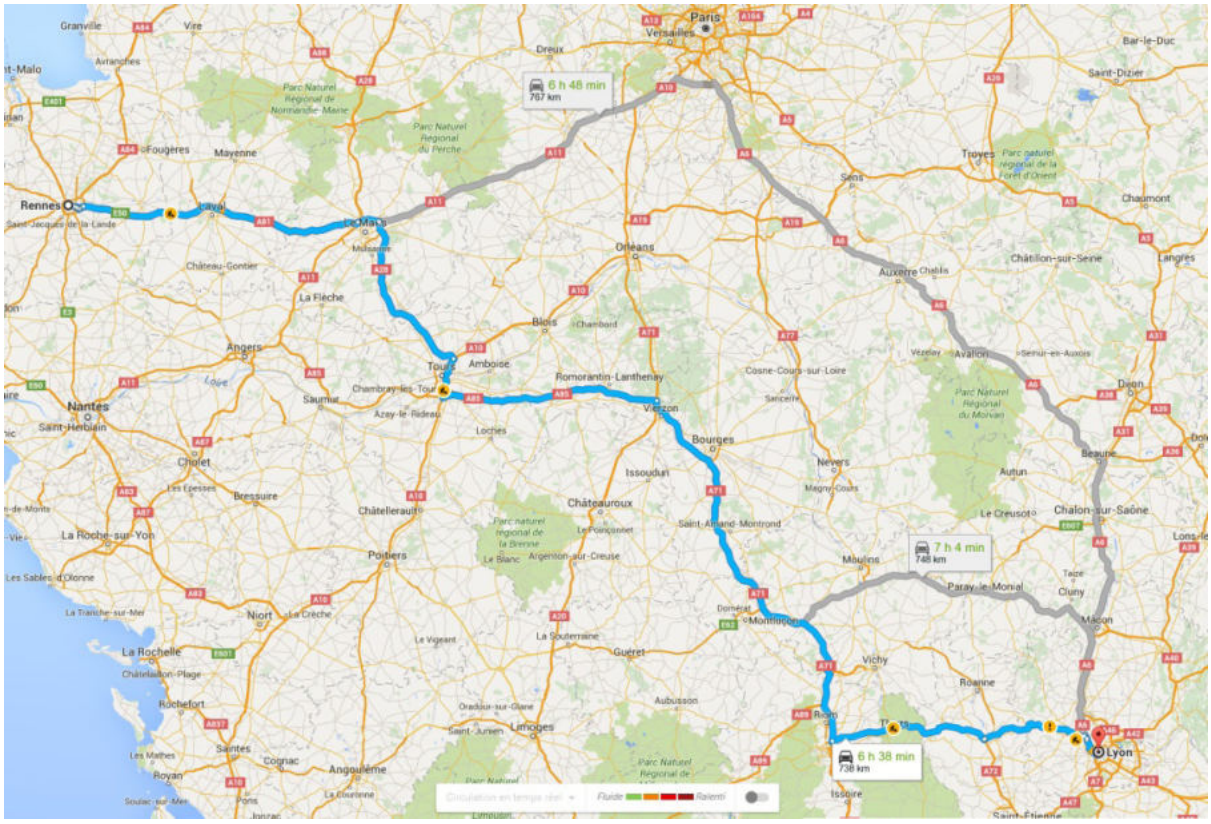
just 19 operations. Let us now imagine that each reading operation in the table requires a millisecond on the computer. The following table gives an estimation of the application time of both algorithms in the worst case:

Number of names on the list	300	3,000	30,000	300,000	3,000,000
Algorithm 1: brute force	0.3 s	3 s	30 s	300 s	3,000 s
Algorithm 2: binary search	0.008 s	0.012 s	0.015 s	0.018 s	0.022 s

Comparison of time taken in the worst case to find a name in a sorted list using two different algorithms. The simplest algorithm (reading the list from beginning to end) takes 10,000 times longer to execute if the length of the list is multiplied by 10,000 (going from of a second to one hour); the binary search only takes 1/100 of a second longer.

There are numerous approaches to developing algorithms. We can only mention a few of them here. However, it is important to look at one type of problem in particular. We know how to write a correct algorithm, but for this problem there is no known solution. Or, we should say, that the solution cannot be obtained within a reasonable amount of time. This is one of the fundamental aspects of algorithmics. For example, consider the "traveling salesman problem" that we talk about in one of the unplugged lessons for Level 3 students (page 267). This involves finding the order in which a traveling salesman must visit a number of cities so that he spends as little time as possible on the road. If there are only two cities to visit, the answer is easy, with just one journey possible. With three cities, there are six possible journeys to compare (the number six is reached by distinguishing "identical" journeys that can be travelled in opposing directions, for example Lyon-Bordeaux-Marseille and Marseille-Bordeaux-Lyon). With four cities, 24 journeys are possible. With 20 cities, there are already over two trillion possibilities. A computer would take several billion years to test all these possibilities to find the best one! This problem is found in several applications, such as the mail carrier's (or garbage collectors' or snow-clearing vehicles') route: in which order should the streets be covered? The same problem applies to factories where articulated arms (or robots) screw nuts and bolts to a plate in different locations: in which order should the nuts and bolts be screwed to minimize movements and therefore save energy and prolong the lifespan of the robot? When the number of "objects" (cities to visit, nuts and bolts to screw, etc.) rises, the number of possibilities explodes. This is called a combinatorial explosion or exponential growth. Beyond a relatively small number of objects, the solution to all these problems is simply unknown.
Of course, to reduce calculation time, we can envisage using a more powerful computer or a more efficient language. The new program, written by an experienced programmer in a more efficient language, may be ten times or even one hundred times faster. But when the estimated calculation time is 10 billion years, it may be reduced to "only" a hundred million years. Just for the sake of reasoning, we could even imagine finding a computer architecture, programming language and a system that lets us calculate the exact solution to the journey across 20 cities in a minute. What would happen if the initial problem changed slightly? Let's add another city: it will take one minute for each of the 21 cities, therefore 21 minutes to find the solution. Now let's add a second city: the super-computer will be working for seven hours. A third city? The

calculation goes beyond seven days. This imaginary machine, completely outside the realm of possibility, will struggle with the addition of just a few extra items.
However, there is hope, because there are several methods to deal with these problems, but they do not calculate the exact solution. They offer a solution that appears suitable, which can be improved on in several operations, and when it is deemed too difficult to improve yet again, the calculation stops there — a good enough effort has been made. In computer science, an approach that offers a satisfactory, but not optimal, result is called a heuristic. In a way, this comes down to applying the old saying, “perfect is the enemy of the good.” Knowing how to recognize these problems for which we cannot calculate a solution within a reasonable amount of time is a major aspect of algorithmics. If a program takes a long time to perform a task, it may not be due to a bug or poor language programming. It may simply be because the algorithm is correct, just like the program, but it may take a long time to perform the task.



This suggested route between Nantes and Lyon is not the best possible route. Finding it would take billions of years of calculations. It is a heuristic, a compromise between the quality of the route and the calculation time necessary to find it. The best route is probably not much shorter than this one. © GoogleMaps

Of course, this does not mean that we can do without experienced programmers who are extremely familiar with their preferred languages, nor does it mean technological progress is irrelevant. Many would be happy to see their favorite software improved to be more efficient by wiser programming or a faster processer. But advances in algorithmics are essential to reach better knowledge of the nature of problems we deal with and how they can or cannot be solved.

What is artificial intelligence?

The term “artificial intelligence” (AI) is used to describe algorithms and machines that are equipped with forms of “intelligence.” It also refers to the field of study that studies these algorithms, which was created in the 1950s. The machines studied in AI are extremely diverse, both in terms of their mechanisms and functions and also their designers’ aims.

Examples of such algorithms are those that allow computers to beat the world’s greatest chess players or assist mathematicians in automatically proving certain theorems. These algorithms require the clever manipulation of symbols, and the aim of their designers is to build efficient machines capable of assisting us in daily tasks such as searching for information on the Internet or solving logistical problems. While they are extremely efficient for specific tasks, these algorithms function very differently to the human brain and are not capable of independently adapting to new tasks (without the help of an engineer that reprograms them).

Other work aims to build algorithms that create models of human cognitive and neural processes, in order to help better understand them. Some research labs create models of the learning and development processes of children’s sensory and motor skills, or language acquisition, often by experimenting on robots. However, the mechanisms that are used still have cognitive and adaptation capacities that are far removed from those of young children, and there are still fundamental comprehension and cognition problems to be solved.

In recent years “Big Data” (vast quantities of data) has become increasingly important in the development of new forms of artificial intelligence. For example, researchers instructed a program to view thousands of images of a cat; the program then independently constructed the “concept” of a cat. Automatic translation also falls under this heading. Google’s translation algorithm, for example, learns gradually as it translates.

Computing objects: computers, robots, networks and more

As we have previously mentioned, computer science was born with the emergence of machines capable of executing algorithms. These "universal" machines are what we call computers today.

What is a computer?

A computer is a machine capable of automatically processing information and memorizing it, and which can be programmed. Data manipulation is a result of arithmetic and logic. While the word "computer" is often associated with the image of our personal computers, with screens and keyboards, computers in the wider sense are today present in many everyday objects in the form of "embedded" processors in these objects[5]: telephones, tablets, electronic watches, cars, planes, traffic lights, cameras and video cameras, televisions, household appliances, interactive games, hearing aids, thermostats and robots, which we'll talk about later on.

From mechanical computers to microprocessors

Today, most computers are electronic machines. However, the first computers were mechanical. A pioneer computer scientist, Charles Babbage designed the first computer in the early 19th century, in the form of a machine made of a complex system of mechanical cogs and gears (see page 7). Calculation instructions were given to this machine in the form of punched cards, following a principle similar to the programming of mechanical looms. Babbage's engine was capable of completing loops and conditional branching in particular, which meant deciding on the follow-up instructions to be executed depending on the result of a calculation.
The first electromechanical computers appeared during World War II. They were used to calculate missile ballistics. These computers used electricity to activate and deactivate mechanical relays in order to carry out calculations. Fully electronic computers appeared shortly after, based on the use of vacuum tubes. The Colossus in Great Britain was an example, and it was used to decode secret messages sent within the German Army (see page 9 on Alan Turing). Vacuum tubes were then replaced by transistors, whose small size, low energy consumption and sturdiness meant they were widely used for integrated circuits. The constant miniaturization of integrated circuits resulted in the emergence of microprocessors in the late 1950s.

Vacuum tube from 1947

PNP transistor from 1953

Intel© 4040 microprocessor from 1974. Contains 3,000 transistors.

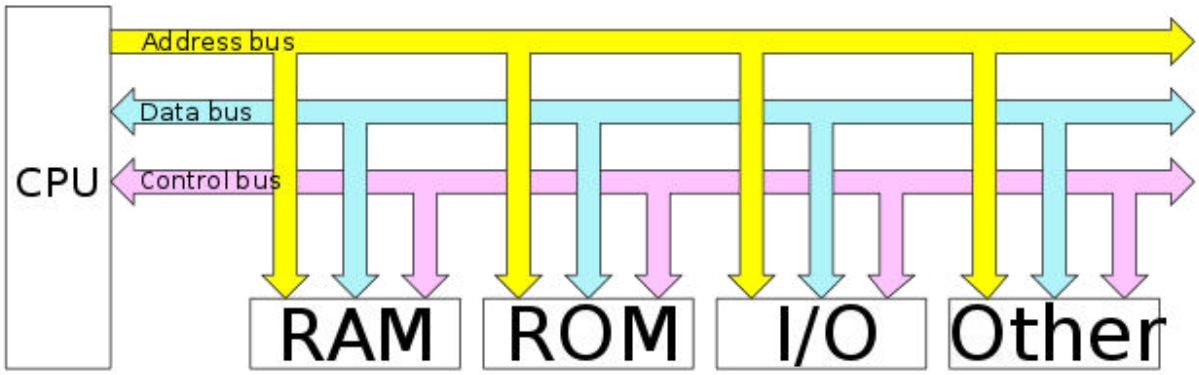
Intel© Core i7 microprocessor from 2014. Contains 2.6 billion transistors.

5 There are around ten times more embedded computers than "classic" computers (with screen and keyboard etc.) in the world.

Today, research labs are seeking alternatives to electronic circuits for the computers of the future. In particular, they are studying how to use the properties of matter to accelerate certain types of calculations (such as quantic states in quantum computers), or how to use certain complex molecules (such as DNA in DNA computing).

Computer components

Most current computers are based on four entities: the control unit, the arithmetic logic unit (ALU), the memory (RAM – random access memory – and ROM – read only memory) and the input and output devices. These entities, often in turn made up of several components, are interconnected via electrical conductors (copper wires and cables) which are generically referred to as "busses." A bus transmits information following a specific code, the "communication protocol."
The control unit is the conductor, directing all of the other entities. It reads and decodes the instructions and data in the memory, transforms it into signals that activate the other entities (for example, it may send the ALU a calculation to perform, retrieve the result and store it in a specific location in the memory).



The central processing unit (CPU) is often used to refer to the control unit, the ALU and the special purpose registers, used to memorize the location of the next instruction in the memory.
A computer's memory can be imagined as a set of compartments where we can store and write numbers (encoded according to binary code: an elementary circuit switched on is coded with "1" and an elementary circuit switched off is coded with "0"). Each compartment has an address which enables the other components to locate it.
Computers are often connected to input and output devices, or peripheral devices, which allows them to exchange information with the user. In addition to keyboards, mice, hard disks, screens, speakers and printers, the peripheral devices of most everyday embedded computers are sensors (for example, of light, movement, heat, GPS signal, etc.), actuators (for example for engines, valves, radiators, etc.) and especially other computers, located inside or outside the same object. For example, a personal computer or telephone contain several computers for sound or image processing and which communicate between themselves, as well as with other external computers, connected to the Internet network.

Robots

Among the objects that have embedded computers, “robots” play an increasingly important role in science, society and the economy. Robots are everywhere: in factories and fields, in the depths of the ocean and in space, in gardens and in living rooms. What’s more, they have become a part of our culture and some of them contribute to the changing vision that we have of ourselves.

Some examples of robots. From left to right, and top to bottom: mechanical arms used in the automotive industry; humanoid robot designed to recognize and replicate emotions; robot explorer on Mars and military robot designed to carry heavy loads across hilly terrain.

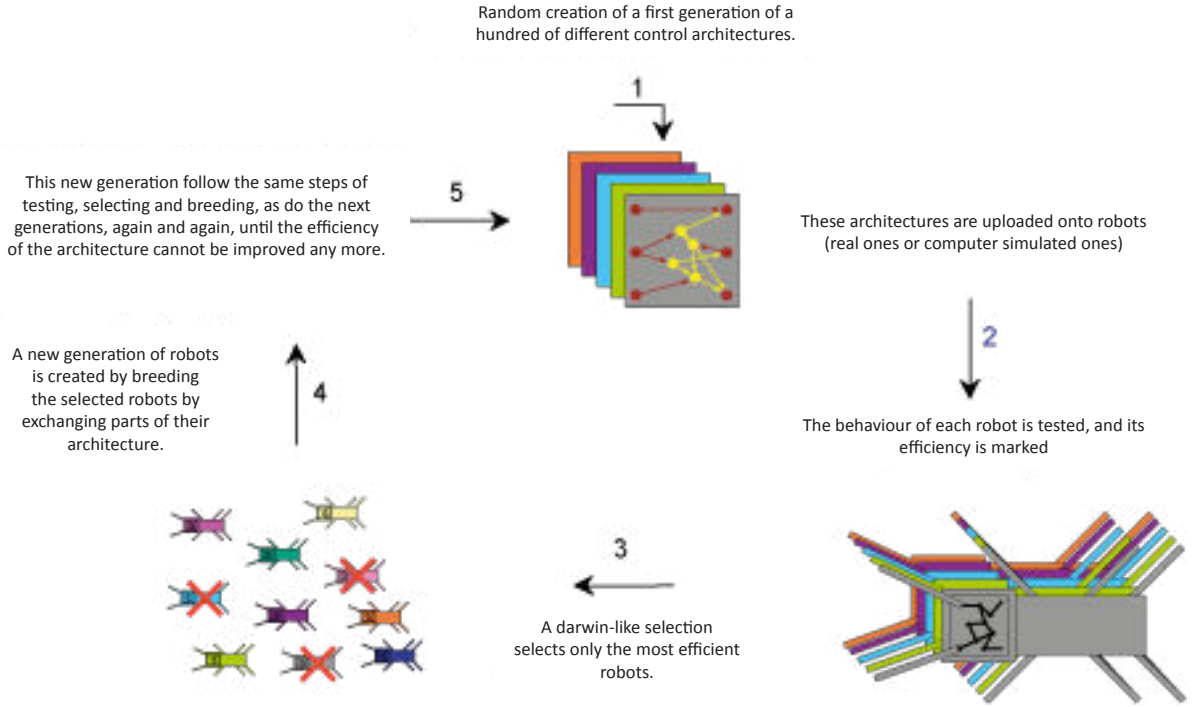
From a technological standpoint, a robot is a machine equipped with sensors (contact, distance, color, force, etc.) which allow to sense its environment, with engines that let it move and operate in this environment, and a system that controls what the robot performs depending on what it perceives. A fundamental characteristic of robots which distinguishes them from automatons is this feedback between perception and action. Automatons, such as those invented by Jacques de Vaucanson and Pierre and Henri-Louis Jaquet-Droz in the 18th century, are not robots because their movements do not depend on what happens around them: they had no sensors and their sequence of actions was fully predetermined by the program.

In practice, this definition of a robot covers a vast diversity of machines: programmable articulated arms in automotive plants, cars (in driver assist mode) and planes that today are vastly automated, vacuum cleaners that do the housework alone, certain electronic toys, and

biomimetic robots in the shape of animals (monkeys, fish, etc.) that are sometimes seen in research labs.
This diversity is not just in terms of shape, but also in terms of use, which implies a diversity of functioning logics. Robots can therefore be designed to fall into one of two specialized fields: autonomy or adaptation and/or learning capacities.

Autonomy: There are robots that can function without human guidance, and others whose behavior is either influenced by a human or almost completely controlled by a human. For example, in a factory, the robots working on a production line and which always repeat the same gesture, often do so autonomously. However, the robots used in nuclear power plants to operate in high radiation zones are generally remote controlled by a human who tells them where to go and what to do after each action.

Adaptation and learning: The behavior of certain robots is programmed as non-modifiable from the outset by the programmer, whereas others are capable of acquiring new behaviors and skills through their experience. Their behavior evolves according to the history of their interactions with the environment. Some robots are therefore capable of recognizing objects in images or even learning to walk while experimenting and assessing different strategies on their own. These adaptation mechanisms are made possible by "learning algorithms," which are based on the automatic detection of regularities in the data flow received by the robot, and on the "optimization" methods that enable the progressive and iterative refinement of the parameters to solve a problem.



Example of the iterative process that allows "generations" of a robot to learn to perfom a task themselves (for example, movement).

With these learning algorithms, certain robots are capable of inventing solutions and behaviors that are not pre-set by their designer, and even of independently selecting objectives that are not pre-programmed. For example, it is possible to program a robot by instructing it to look for new situations so that it can increase its knowledge of the world around it. So, some

algorithms mean these machines are equipped with forms of learning and creativity. However, these capacities and the performances of these algorithms are today, and probably for the foreseeable future, very weak in comparison to the adaptation and reasoning capacities of many animals, and especially humans.

As we have seen above, these various functioning methods meet a number of needs: there are a great many reasons, and therefore functions, for which robots are built and used. We could consider three categories of functions: working and exploring; human assistance and modelling the cognitive and behavioral mechanisms of living things.

Working and exploring

Most robots currently in service in the world are industrial robots, of which there are around nine million. Very early, companies took an interest in these machines for two reasons:

- Firstly, robots can be used to replace human workers for repetitive, menial tasks that require low skill levels, such as assembling, painting or soldering parts;
- Secondly, these machines are able to conduct production line work much more quickly and efficiently than humans.

The first industrial robot, Unimate, appeared in 1961. It was an articulated arm installed in a General Motors car factory, and it handled heavy foundry parts. In the 1970s, the use of robots in industry took off. Today, robots are present in every industry and are no longer restricted to the automotive sector. For example, in the agriculture and agrifood sectors, robots can be seen in the fields harvesting fruits and vegetables: some of them cut, squeeze and bottle; others sort and fill cartons; and other group the products into pallets. In some airports, fleets of robots transport baggage and load them into the holds.

Robots are not only useful in industry for simple, repetitive tasks; they are also used to work in environments that are dangerous for humans. The nuclear industry is a typical example. Whether they are autonomous or partially remote-controlled, nuclear plant robots can move around in confined, radioactive areas, they can handle dangerous substances and perform maintenance on other machines. Another example is the oil industry: underwater robots, for example, are used to check the condition of ships' hulls to prevent accidents and identify vessels ready for decommissioning.
Lastly, robots are crucial for exploring places where humans cannot go, first and foremost space and solar system bodies. In 1996, the first mobile robot landed on the moon, onboard the *Surveyor* probe. Next came the Soviet Lunokhod, followed by the series of U.S. Mariners. In 1997, a robot landed on Mars: *Sojourner* was propelled by the energy it captured using solar panels. It sent thousands of pictures to Earth, and was much adored by the general public. *Sojourner*'s navigation system was partly autonomous, because at such a distance from Earth, it is extremely difficult to operate by remote control in real time. In 2004, a new robot mission attracted the whole world's attention: *Spirit* and *Opportunity*, equipped with spectrometers and an arm which allowed them to dig into the surface, offering proof that water flowed on Mars.

Assisting humans with everyday tasks

While the 20th century saw the rise of worker and explorer robots, at the dawn of the 21st century, another great family of robots began to take off: human assistance robots. In our homes, robotic assistance for household chores is becoming increasingly widespread. Robotic

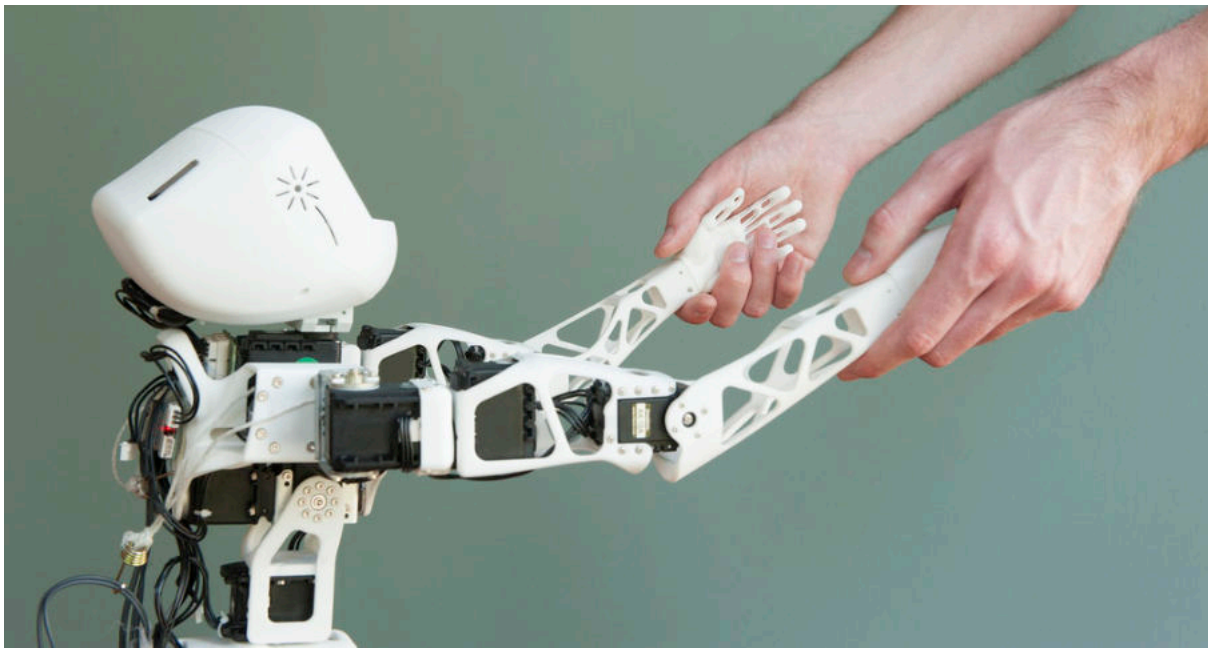
assistants are also used in stores and in the workplace. The medical field has been particularly transformed by robotics: while surgery assistant robots have been used daily for around 15 years, today there are new uses emerging. Robots can assist people with physical or cognitive difficulties, for example to help them get up and sit down, to stimulate them cognitively when they have memory problems, or to play a facilitator role in communicating with family or medical teams. Tomorrow, researchers will introduce even smaller robots to operating theaters: miniaturized endoscopic capsules capable of exploring intestinal tracts, arteries and veins to help the surgeon diagnose an illness. In the past few years, robotic prosthetic hands and entire arms have emerged, which can be used for amputees.

Robotic prosthetic hand©BeBionic

Modelling living things and cognition

Robots have become essential tools for studying and making models of the complex systems in the field of life sciences. In particular, robots today are used in research labs to understand how living things adapt to their natural environment, in terms of behavior and deployment of cognitive abilities. This behavior is caused by the dynamics of interactions between the brain, the physical body and the environment, dynamics that are being constantly redefined because the brain changes with each new interaction. Embedded in robots are a “brain” (or programs that allow it to process the information acquired by the robot according to specific rules), “sensory organs” (sensors) and “motor systems” (actuators). In this way, just like living things, robots can change and be changed by the physical environment that they operate in. Their “brain” is therefore modified too, because the robot acquires new skills that can be reused in later interactions.

Researchers can study the complexity of the brain-body-environment interaction, by conducting experiments that are possible on robots but impossible on living things — such as, for example, “switching off” a part of their artificial brain to see how the behavior is modified, or by altering body parts. Some research labs also study motor control, visual perception, spatial awareness and even speech and language learning and development mechanisms in humans. In these projects, interactions with the neurosciences, biology, psychology and even ethology play a central role.



Poppy, an open source humanoid robot. It was developed by Inria for research on cognition and for educational projects. For more information: https://www.poppy-project.org

Networks

Today, most computers are interconnected, and rare are those disconnected from a network. We connect them to be able to exchange information, whether this is sending an email or downloading information stored on the Wikipedia computers to read them.

We also interconnect computers so they can work together on tasks that are too vast to be performed on a single machine. Now, there are computer "farms" that group together thousands, even millions, of machines — which, individually, perform to standard levels — in the same, air-conditioned hangar, to conduct increasing numbers of calculations per second.

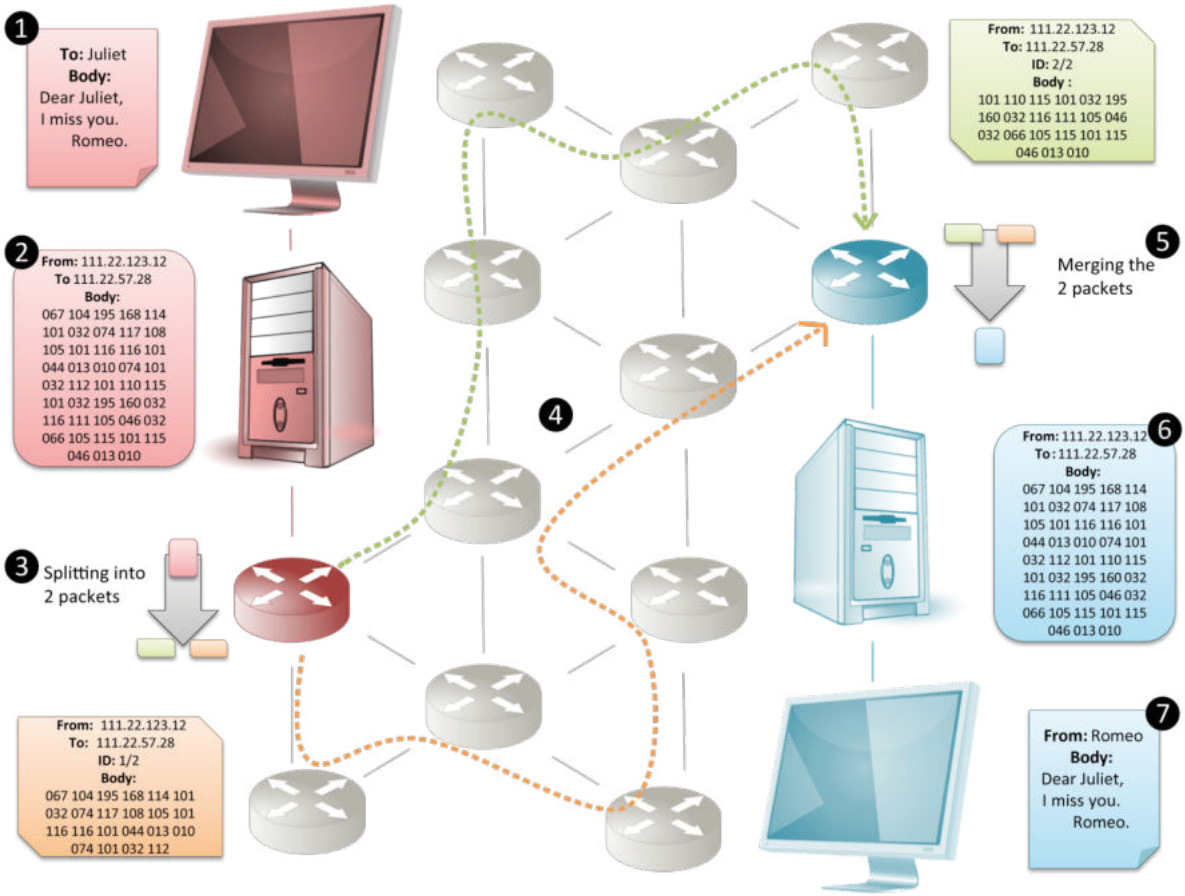
The first long-distance networks appeared at the end of the 1960s. There were only around 15 interconnected machines in 1971, a thousand in 1984, a million in 1992 and today there are several billion. The first email was sent in 1972, the first website made in 1991 and the first tweet posted in 2006.

In the old telephone networks, two telephones had to be physically connected so they could communicate. Switchboard operators performed these connections, which meant you needed a line for every conversation.

Most modern computer networks follow a more efficient approach, called *packet switching*. The information is broken into packets and sent one after the other. At each network node (which is connected to a chain), routers receive these packets and pass them onto the next, until they reach their destination.

Many forms of technology have been proposed to exchange these packets between computers, at risk of turning computer networks into something resembling the Tower of Babel. Fortunately, the Internet allows all the computers connected to it to communicate despite their differences. To achieve such a feat, scientists and engineers applied a standard information technology approach: the problem is broken down into sub-problems, which are resolved separately, and the partial solutions are combined. This way, the problem of exchanging information between computers is broken into layers.

This breakdown is what makes a global-scale network possible: the technologies are interchangeable within each layer, and the whole lot work together thanks to standardized interfaces between the layers. Internet Protocol (IP) is the name of a protocol that lets any local networks connect among themselves. This is why the Internet is described as the “network of networks.”



Romeo uses his email program to send an email to Juliette (1). This message is first of all encoded in ASCII (2) and then in binary. A router breaks it down into several packets (3: in reality, there are lots more packets!). Each packet travels across the network via several other routers (4). The packets are finally put back together (5), and the message is transmitted to Juliette’s computer (6), before being decoded and displayed on the screen (7).

Internet or Web?

In everyday language, and often in the media, there is no difference between these two terms. Wrong!
Internet is the "network of networks" and the Web is just one of the networks that the Internet connects (see page 12 for the history).
The World Wide Web, invented in 1990 at CERN in Geneva, is a network of pages connected solely by hypertext links (they are easily recognizable because their Uniform Resource Locator — URL, or their address — often begins with "www."). But there are other networks on the Internet, such as, for example:

- Electronic mail (email) services, which use Simple Mail Transfer Protocols (SMTP), Post Office Protocol version 3 (POP3), Internet Message Access Protocol (IMAP), etc.
- Instant messaging services, or chat, use Internet Relay Chat (IRC) protocol, Extensible Messaging and Presence Protocol (XMPP), etc.
- The Usenet forum (newsgroups) servers use the Network News Transfer Protocol (NNTP).

How information is represented[6]

The notion of representing an object, either tangible or abstract, using another is an ancient one, because all languages are based on representing objects using sounds.
As there are too many objects for every single one to be represented by a different sound, they are represented by a series of sounds, selected from a small collection. For example, standard French uses thirty-six sounds, or phonemes, to make up words. In a language, a message is a finite sequence of units chosen in a finite set. This notion, however, is not entirely satisfactory in spoken languages, where the volume, delivery and intonation, etc. of the message contribute to its meaning, just as much as the sequence of units which form the message.

How much information does a text contain?

Writing, and in particular its use to express languages, eliminates these accessory elements, but it is a regression — at first — because the earliest written languages associated a different symbol with every object. It was not until the alphabet was invented that the idea of associating a sequence of units taken from a small set (the alphabet) emerged, rather than assigning a unit to each object. The French alphabet, for example, contains around 110 characters: 42 lower and upper-case letters (the 26 basic letters, 13 accented vowels, the c cedilla and 2 tied letters), 10 numbers and around 20 punctuation marks.
The number of sequences of n characters that can be formed by selecting them from an alphabet of "k" elements is k^n. There are k possibilities for the first character, k possibilities for the second... and k possibilities for the nth character. The total number of possibilities is therefore k * k * ... * k (n times) i.e. k^n.

6 The "1,2,3...Code!" project proposes several activities for Level 2 and 3 students to work on how information is represented (text and images), in particular in the lessons 1.2 and 1.3 page 115, 123(Level 2) and 1.2, 1.3, 1.4, 3.1, 3.2 and 3.3, (page 213, 219, 225, 296, 304, 309)(Level 3).

The greater the number, the more information is contained in each of these sequences. For example, a 4,000-character text chosen from the 110 characters of the French alphabet contains more information than a 2-character text chosen from three.
The unit of information is called a bit.
For historic reasons, we often use another unit of data, the byte, which is equal to eight bits, and its multiples: the kilobyte (KB) equal to a thousand bytes, or eight thousand bits; the megabyte (MB) equal to 1 million bytes; the gigabyte (GB) equal to one billion bytes; the terabyte (TB) equal to one trillion bytes; and the petabyte (PB) equal to one quadrillion bytes.
The quantity of information contained in a character of the alphabet is roughly a byte. In a page of 2,000 characters there are around two kilobytes; in a book of 600 pages roughly a megabyte; a small library containing one thousand books contains roughly a gigabyte. The collection of printed books at the National Library of France (BNF), which holds 14 million volumes (excluding images and films, etc.) is approximately one terabyte (14 terabytes to be precise). A terabyte is also the capacity offered by a disk that costs approximately 50 euros. A petabyte is the size of one hundred National Libraries. Every year, CERN (European Organization for Nuclear Research) produces around 15 petabytes of data.

Binary: A two-letter alphabet

How is a number represented?

An entire sector of information technology focuses on data representation in the form of a sequence of symbols from a finite alphabet, often an alphabet that only contains two characters: **0** and **1**. So, to represent all numbers in an alphabet that contains just these two characters, we can associate a sequence of four 0s or 1s with each digit, for example:

- 0: 0000
- 1: 0001
- 2: 0010
- 3: 0011
- 4: 0100
- 5: 0101
- 6: 0110
- 7: 0111
- 8: 1000
- 9: 1001

Intuitively, to represent a number, the first idea we might have is to place the representation of each of the digits next to each other. For example, the number 13 would be represented by placing the representation 0001 of the digit 1 with the representation 0011 of the digit 3: 00010011. This rather long notation is not used in practice for numbers.
Another, shorter representation is found by representing the number in the base two system. The number 13, for example, which is broken down into a unit, zero packets of two units, one packet of two packets of two units and one packet of two packets of two packets of two units (1 + 4 + 8) is written from right to left: 1-0-1-1, or 1101 from left to right.

$$\begin{array}{rl} 0 & R1 \\ 2\overline{)1} & R1 \\ 2\overline{)3} & R0 \\ 2\overline{)6} & R1 \\ 2\overline{)13} & \end{array}$$

To write the number 13 in base two, we must break it down as a sum in powers of two. To do this, we need to perform a series of divisions by two, until we obtain a quotient equal to 0. The digits (the remainders of the divisions) must then be placed in order (units on the left).

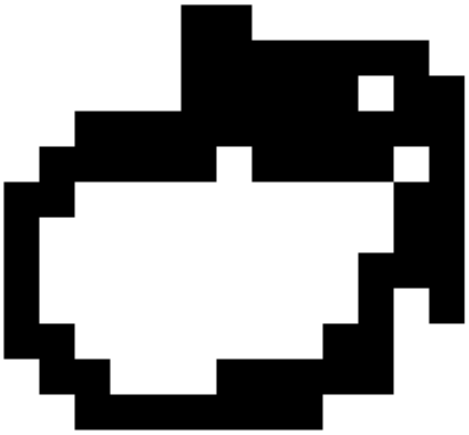
A succession of 64 “0 and 1” characters allows us to represent the numbers 0 to 2^{64} - 1 = 18,446,744,073,709,551,615. The question of number representation does not stop there, however, because negative and decimal numbers have their own representation method.

How is a text represented?

The representation of textual characters — and texts which are successions of textual characters — is also an extremely interesting question. An initial way of representing textual characters was by associating a group of 7 “0 and 1” characters to each character of a 95-character alphabet (26 lower-case, 26 upper-case, 10 digits and 33 other symbols, including punctuation marks). The textual character “a” was represented by 1100001, “b” by 1100010, and so on. A word was simply represented by aligning the representation of these characters, or its letters. The word “Le,” for example, was represented by aligning the 1001100 representing the character “L” and the 1100101 representing the character “e”: 10011001100101.
But this representation, called American Standard Code for Information Interchange (ASCII), was suitable for English and not languages like French, which use diacritical marks such as accents and cedillas, for example. The ASCII representation was therefore extended several times, until it became Unicode, which provides for the roughly 110,000 characters used by the various scripts of the world, including non-alphabetic scripts such as Chinese.

How can an image be represented?

Representing images is more difficult, because this requires breaking down the image into pixels, and coding the shade of each pixel. In a black-and-white image, each pixel is represented by a single “0 or 1”: 1 for black, 0 for white.



0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 1 1 1 1 1 0 1 1 1 1 0 1 0
0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0
0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0
0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0
0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0
0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0
0 0 0 1 1 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Left: a black-and-white, 16x16 pixel image. Right: binary coding for this image. To make it easier to read, we have arranged the bits on a 16x16 grid.

In a grayscale image, each pixel is typically represented by a number between 0 and 255, in turn represented in base-two by a group of eight 0s or 1s. 0 = 00000000 if it's black, 1 = 00000001 if it's dark gray, ... 254 = 11111110 if it's light gray, 255 = 11111111 if it's white. Of course, the decision to use 255 shades of gray is arbitrary and other formats do exist.
In a color image, each pixel is typically represented by three numbers, all of them between 0 and 255, which indicates the quantity of red, green and blue in this pixel. For example, a bright pink pixel is obtained by combining 249 units of red, 66 units of green and 158 units of blue. The number 249 is represented in base-two by 11111001, the number 66 by 1000010 and the number 158 by 10011110, and thus a bright pink pixel is represented by the succession of 18 0s and 1s: 111110011000010100111110. An image, naturally, is represented by aligning the representations of each of its pixels. An image made up of two bright pink pixels is represented by: $111110011000010100111101111100110000101001111 0$.

Organizing the information

These principles allow us to represent small objects: a number, a text, an image, a sound or a video, for example. We can, for example, use them to save a single text, image, sound or video on a computer disk. However, very often we want to save several of that type of object on such a disk. This leads us to other principles which, rather than representing the information, will allow us to organize it.
A disk is a space on which it is possible to store a large number of 0s and 1s, generally eight trillion for a disk with a terabyte. In order to store several texts and images, we can begin by breaking this space down into several zones called "files," which allow us to store a text, an image, a sound or a video, for example. But when these files begin to pile up, we need to put them into folders, and these folders into other folders. Essentially, we structure them like a tree. However, finding a file in such a tree quickly becomes difficult, and so we have invented other principles to organize large volumes of information. The first is the use of hyperlinks. Such a link allows us to indicate, in the first file, the name and location of a second file on the same disk, or on another.
So, we can indicate in the first file that the second is found at the address "disk3:archives/2018/

letter.txt," which means its name is "letter.txt" and it is located in the 2018 folder, which is found in the "archives" folder on a disk called "disk3." This way, by using a browser to read the first file, we just need to click on the link to access the second. This method of organizing information using hyperlinks, across a network of disks and computers, is the idea behind the Web (see above, page 30).

This organization of information in files, then folders, and finally by adding hyperlinks is in itself insufficient when we need to store a lot of information, and there are other ways of organizing large quantities of information. The first, which is suitable for structured information, is to create a database. For example, an address book is a small database, made up of quintuplets (5 pieces of data), each containing a surname, first name, address, telephone number and email address. We can check this database by entering a request, in which we ask for the list of quintuplets that contain for example a surname, or a first name, or even a telephone number. For less structured information, we can leave an assortment of files on one or several disks and let the research engine index them, then access these files by entering a request in the search engine, which will then indicate where all files containing, for example a certain word, can be found. This method of organizing information requires less work that structuring them in a database, but yields poorer quality results. Looking for Mrs. Sellers' phone number by entering a request in a search engine leaves us at risk of coming across several texts containing the word "seller" but are merely referring to the profession.

Converting and manipulating data[7]

Representing and organizing information enables them to be sent from one place to another (which means they can travel through space), archive them to reproduce them later (which means they can travel through time), but most importantly they can be converted. Several algorithms enable specific information to be converted; for example, the multiplication algorithm only applies to numbers. But three types of algorithms can be applied to any data: compression, correction and encryption algorithms.

Compression

We saw earlier that a bright pink pixel is represented by 111110011000010100111110 and a monochrome image of a million pink pixels can be obtained by aligning this block of 24 0s and 1s a million times. Instead of producing an extremely repetitive text of 24 million characters, we can use a more concise representation that consists in indicating that the block 111110011000010100111110 must be repeated a million times. This is what we call compression.

Correction

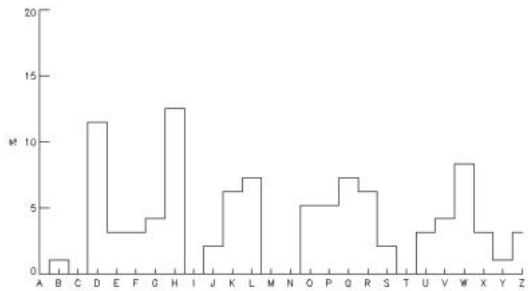
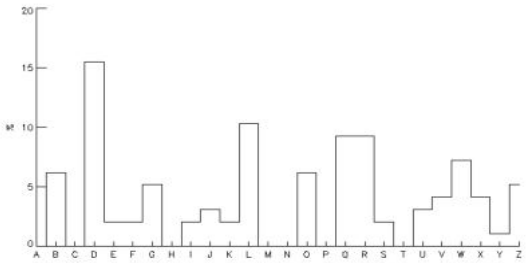
Error correction algorithms find and sometimes correct an error in a piece of information, for example an error introduced when the information was sent. A simple method, although one which is costly in memory space, is to repeat each unit of information three times. For example, 101 is converted into 111000111. If an error occurs, for example in the third triplet: 111000110, it is easy not only to find this error, but also correct it: as there is a majority of 1s

7 The "1,2,3...Code!" project offers two activities that deal with correction and encryption in Level 3 (Lesson 3.4 and 3.5, page 316 and 324).

in 110, the initial letter was most likely a 1. This correction algorithm only fails when two errors occur in the same triplet.

Encryption

Lastly, encryption algorithms render a text unintelligible, except to the people who hold the key. A simple, but not very effective method, was used by Julius Caesar to communicate with his armies. It consisted in shifting each letter a set number of places forward or backward in the alphabet. If we shift the letters three places in the phrase “VENI VIDI VICI”, we obtain “YGPL YLPL YLEL”. To decrypt this message, we need to know the key. Here, it is the number of places that we need to shift each letter back to find the original message. As there are only 23 letters in Latin, there are only 23 keys possible, and so the message is not very hard to break (or cryptanalyze), which means decrypting it without knowing the key.
Frequency analysis helps us find the key quickly: in a given language, all letters have a known frequency (in English, the letter E is by far the most frequent letter; the letters T, A, O, I, N, S, H, Rare quite frequent. At the other end of the spectrum, the letters Z, Q, and X are hardly every used). Although in some specific cases, the most frequent letter directly gives us the code key, it is often necessary to use a more comprehensive histogram, which provides the frequencies of all letters. To figure out the key, we just need to shift the histogram of the ciphered text until it best corresponds to the histogram of the language considered. In the three examples below, the encryption key remains the same (+3, the same as the one Julius Caesar used): even if there are some minor discrepancies in the histograms, we can see a similar theme (even for the extract of Georges Perec’s “A Void”, which he wrote without the letter E).

Ciphered message (key+3)	Frequency analysis	Deciphered message
D KXPDQ EHLQJ ZLOO DOZDBV VXSSRVH WKDW, WKH PRUH KXPDQ D URERW LV, WKH PRUH DGYDQFHG, FRPSOLFD-WHG, DQG LQWHOOL-JHQW KH ZLOO EH.		A human being will always suppose that, the more human a robot is, the more advanced, complicated, and intelligent he will be. (The Robots of Dawn, I. Asimov)
D JDS ZLOO BDZQ, DFKLQJOB, GDB EB GDB, LW ZLOO WXUQ LQWR D FRORVVDO SLW, DQ DEBVV ZLWKRXW IRX-QGDWLRQ, D JUDGXDO LQYDVLRQ RI ZRUGV		A gap will yawn, achingly, day by day, it will turn into a colossal pit, an abyss without foundation, a gradual invasion of words... (A Void, G. Perec)

There are other, more complex methods aside from a monoalphabetic substitution cipher. During World War II, the German cipher Enigma (broken by A. Turing and his team, see page 9) was also based on substitutions, but the encipher key

changed regularly throughout the text. Historically, ciphering methods that were not monoalphabetic substitutions were also used, in particular during wartime (for example, the ADFGVX cipher during World War I).
Even better ciphering methods exist and are used every day to protect our credit card numbers and other secrets that we only share with a small number of people.

Computer science and social challenges

Current scientific, economic and social challenges

The building of computers from the 1940s first and foremost transformed the world of research. By enabling us to simulate complex physical phenomena, such as climate changes, and by allowing us to process great quantities of data, such as the human genome, computers have transformed the scientific method. Computers and networks have simultaneously transformed how information circulates in companies. The appearance of computers and networks in households transformed the way we communicate in our private lives, with our friends and families.

We sometimes talk about the "digital world" to refer to the world we live in, characterized by the omnipresence of computerized objects. They are everywhere, both visible (computers, tablets, telephones, etc.) and invisible. In "embedded" computer systems (on planes, trains, cars, pacemakers and household appliances, for example), computerized objects are increasingly replacing mechanical or electrical systems.

Data access, through the Internet, is now extremely easy and fast. When data is digitized, it takes up hardly any space at all, costs very little and can be instantly duplicated. As such, computers are said to have perfect memory, able to collect enormous quantities of data and conserve it for long periods. A simple exercise can give us an idea of this perfect memory: the National Library of France holds 14 million volumes. Let's say that every volume includes 500 pages and every page 2,000 characters (for this exercise, we will forget about images) and a character is represented by a byte. The "size" of this library is therefore:

14,000,000 × 500 × 2,000 = 14,000,000,000,000 bytes

or 14 trillion bytes, which is 14 terabytes. But a 10 terabyte disk costs a few hundred euros. We can therefore see how, thanks to digital technologies, the storage capacity of a small company can rapidly exceed the size of the National Library of France (BNF).

From an economic standpoint, the computer industry is the star of the digital world, as it produces all this equipment and software with high value added, alongside research in computer science, as progress in theory goes hand in hand with technological progress. The digital world destroys employment: carrying mail from one end of the Earth to another required armies of mail carriers, which today are no longer necessary as mail is electronic. But it is also a source of employment, because we have to develop new applications and new software incessantly to keep up with rapidly changing needs. And finally, the third aspect of the economic importance of computers, which we do not necessarily think of straight away, is professional training. Everyone who handles computers and software, directly or indirectly (scientists, technicians, salespersons, lawyers, teachers, architects, artists, etc.) require training, either initial training or professional development. Professional training and guidance in computer science is of increasing importance to businesses in the digital sector.

Despite these opportunities, it is evident that France and Europe are behind in terms of North America and Asia. With the exception of a few niche markets (computer services, research, robotics, etc.) our countries have for too long relegated computers to the rank of a tool, teaching people how to use a particular program (which often becomes obsolete in a few years) rather than teaching them computational thinking, an essential step that fosters the emergence of

innovation. In recent years, we have seen efforts being made to address this delay, and a desire to compensate for it, mainly through education.

Computers and ethics

The digital world offers unrivalled liberties to express oneself, access information, do training, communicate and discuss. But wherever there are so many liberties, the risks and dangers are great. Young people should be assisted, by being shown the boundaries of this new world.
Accessing free, instant information would have seemed like a utopia to the builders of the Library of Alexandria. However, as we saw above, an inexpensive hard disk can contain almost as much data as the National Library. But if there is a large quantity of data accessible, is it good quality? How much can we rely on the information found on the Web? How can we maintain a critical eye, check our sources (also accessible on the Internet), fact-check accounts? In the classroom, we could, for example, have the students produce a web page so that they realize the fact that they can write whatever they like on the Web, and therefore, if they can, anybody else can (if they know how). The information that we find on the Web was placed there by people like them.
Among the digitized information stored here and there, we can also find personal data. Our social security information is stored on a server in France, and we cannot do anything about it. Our bank details are stored, most likely on a server abroad, and we cannot do anything about that either. However, our holiday photos are posted on Facebook, because we posted them ourselves. In these three cases, we must be cautious: to what degree is our personal data secure? Who can access this astronomical collection of personal information? Remember that computers have absolute memory and that every activity can be stored and processed extremely quickly. In 2008, Google launched the project “Google Flu”, collecting all searches with the keywords “flu”, “fever”, etc., region by region. Google was able to describe in real time the progression of the epidemic, and even able to predict it ahead of time. This type of information is of course extremely valuable to health authorities, but also to pharmaceutical companies. Should this type of information be made public, or could it be privately kept and sold to the highest bidder? The flu example can be applied to many other topics, and since Google stores not just search requests but also the IP address of the computer that sends it, it is not very difficult to convert this seemingly anonymous data collecting into a form of espionage. And what about all the applications that use and abuse geolocation, meaning we can be located to the closest meter? All of these tools, which we use to make our lives easier, have hidden dangers. Every country must therefore provide informed legislation that provides for the security of our personal data, and also the right to be forgotten. This requires legal recognition of digital infractions, research in computing cryptology and virology, security training, and more. This is already quite a challenge at the national level, due to the lack of “digital common sense” and a thorough knowledge of the risks involved, so it becomes even harder at the international level. Networks are supranational, and laws change from one country to another. The sale of certain products such as drugs are authorized in one country and not another. Similarly, the publication of certain texts — defamatory, or blasphemous, for example — is authorized in some countries and not others. Which laws govern the publication of a text: the laws of the country where the computer that hosts the text is located, or the laws of the country where the computer used to access the text is found?
Educating young people, but also adults (both citizens and legislators) is essential to ensure that these new tools are used in an enlightened and respectful way.

Educational Background

Making the distinction between Computer Science and Information and Communications Technology (ICT) in Education

ICT in education: Important, but insufficient

Over the last three decades, the presence of computer science in education focused on teaching students how to use tools such as computers, tablets and some commonly used programs such as word processors, spreadsheets, web browsers, etc.
This user-centered approach is the approach taken by Information and Communications Technology (ICT) in Education. It is useful because it enables children to acquire useful skills in a number of school subjects and in everyday life, including researching information, communicating and performing calculations.

Going further than simple usage: Understanding computer science

However, a usage-centered approach simply prepares the student to use standard tools without understanding the underlying principles of how they work. Students are not equipped to adapt existing tools to their specific needs, nor to adapt themselves to the rapid development of these tools, and they are not trained to invent and create new tools. These are the skills that they need to become active rather than passive users of the digital world around them.
Computer science is not just a set of tools: it is a science, with its own history and concepts, an overview of which can be found in the Scientific Background section, pages 3 - 38. As a science, computer science is a fully-fledged educational subject just like physics or biology. Indeed, the skills used in computer science, just like in the other sciences, help children work on their own, take initiative, develop reasoning and creativity, etc.
The concepts behind computer science will remain long after the tools of today, and being familiar with these concepts will be of permanent value to students throughout their professional and private lives. As an example, we believe it is more educational to guide these students to discover how images are encoded, through unplugged activities (manually pixelating images) and plugged activities (creating small images on the computer) rather than simply training them to manipulate digital images with complex software used like black boxes. These programs could be put to better use if the students understood the underlying key ideas such as information coding, pixels, resolution and compression.
The same is true for key ideas in algorithmics, programming and robotics. Understanding them gives meaning to their use. Students that are taught these concepts will never look at their image processing program, spreadsheet or any other software — even a household robot — the same way.
To sum up, approaching computers as a science does not exclude ICT, but to stop at ICT appears to us to be a very narrow view of computer science.

How should this teaching manual be used?

A turnkey project...

This handbook offers a number of sequences for Levels 1, 2 and 3 (from pre-school to 6th grade) that may be considered "ready-to-use". The description of the sequences and lessons is specific enough to allow a teacher, even with very little knowledge of computer science, to confidently lead a class. For every lesson, the duration of the activity is given, alongside the necessary equipment, initial problem, potential difficulties and the conclusions expected.
All the lessons were prepared by a multifunctional team of teachers, trainers and scientists. They were tested in around thirty classes of varying profiles (rural/urban environments, privileged and disadvantaged, with young and experienced teachers, multi-level classes, etc.).

...A project you can grasp, and adapt to your class!

Understanding the teaching project first requires reading up on it. It is therefore essential to try out the activities — both plugged (requiring a computer, tablet or robot) and unplugged — beforehand.
For each Level, there are several sequences that include plugged and unplugged activities, and which sometimes provide alternatives depending on equipment available (computer, tablet or robot). These sequences are based on an inquiry-based or a project method: it is possible that the students deviate from the outline provided for the class. The teacher must, if necessary, adapt their progression, and sometimes construct a new one, depending on the time they wish to spend on the lesson and the students' past experience.
In addition, this teaching guide — which covers the first three educational Levels — can be broken down into further projects within each Level, or as a project to help students transition from fourth to fifth grade, for example.
To do this, the conceptual scenarios suggested for each Level provide an overall perspective of the key ideas being targeted, working in tandem with the lesson summaries which provide an overview of the types of activities proposed.

Support options available in addition to this teaching guide

To provide support when introducing computer science to elementary and middle schools, the *La main à la pâte* foundation and its partners offer a variety of services to teachers and trainers:

- Training courses on demand:
  - The *La main à la pâte* team hold training sessions at the request of the local educational authorities (district authorities, boards of education, teacher training colleges, etc.). More information is available on the project website (see page 342).
- Distance and blended learning
  - The Class'Code consortium, which brings together several institutions and associations, offers blended learning that combines a MOOC with classroom support on teaching computer science (http://classcode.fr, in French).
  - France IOI offers a variety of resources on its website (http://www.france-ioi.org/) as well as online courses and distance support.

- Distance support
  - The project website (see page 342) is designed as a teaching support tool, where teachers can ask questions and provide feedback about their classroom activities and more.

How to teach computer science

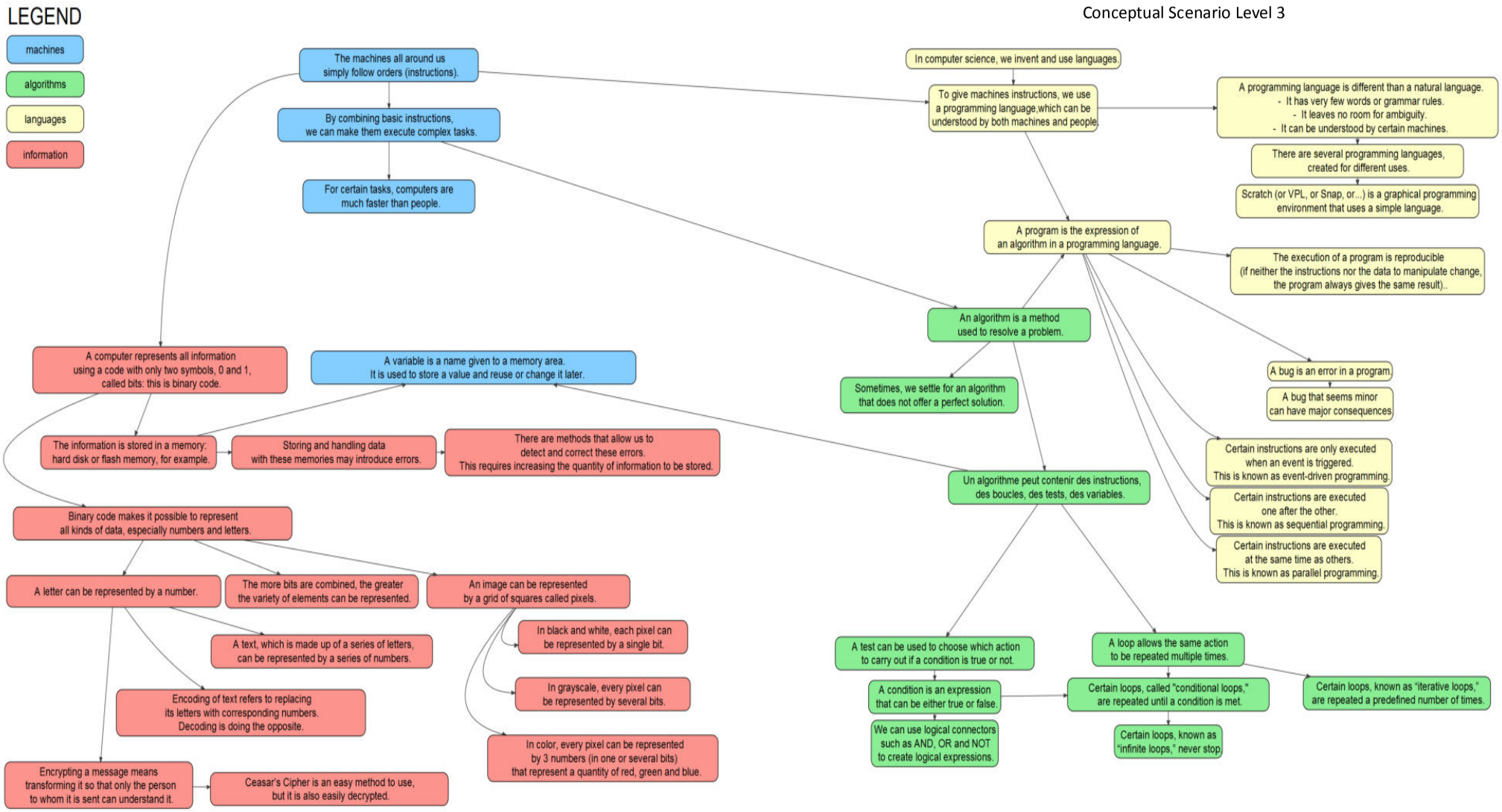
Choosing lesson content

Teaching computer science is often understood as teaching computer programming. As we shall see, programming is essential, but far from sufficient when teaching this subject.

- It is essential that students grasp the basics of programming early on.
  Firstly, because it is practically impossible to understand what a program is without having written a few yourself.
  Secondly, because providing students with a basic grasp of programming is essential if they are to have an active learning approach.
  A student that does not know how to program and is trying to understand Caesar's Cipher, for example, will rapidly become passive. They might understand the principles of this ciphering method, and that other people use it to encrypt and decrypt messages, but they may not be able to do it themselves.
  However, as soon as they understand the basics of programming language, they can write a program themselves with a few lines, which enables them to encrypt and decrypt messages and fully grasp this method through an active approach.
- However, being familiar with computer science does not stop at knowing how to program, just like being familiar with electricity does not stop at knowing how to produce an electrical circuit. It is therefore essential, when preparing a computer science lesson, to balance the various objectives:
  - Students must understand algorithms and the principles that enable them to create algorithms, such as the binary search algorithm (in middle school).
  - Students must understand what a programming language is, how different it is from a natural language, the similarities with musical notation and with the language with which we can express numbers with Arab or Roman numerals, etc.
  - Students must understand that computer objects are data represented as symbols (text, image, sound, etc.) and how this data can be compressed, encrypted, corrected, etc.
  - Students must understand that there are a vast number of machines that process information: computers, telephones, robots, networks, etc. They must also realize that beneath this diversity, there is a fundamental unity. All of these machines process information by executing algorithms, expressed as programs in a programming language.

The "1,2,3...Code!" project allows students to grasp the basics of the four key concepts in computer science, mentioned above (machine, algorithm, language, information).
These concepts may be broken down into more basic key ideas, which can be seen in the project's conceptual scenarios.
On the page opposite is an example of a conceptual scenario, created for Level 3 (4th, 5th and 6th grades. The associated lessons are described on page 202).

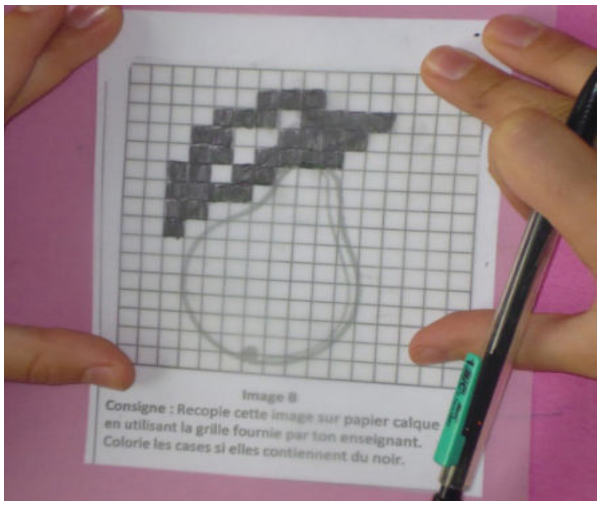
"1,2,3...code!"
Conceptual Scenario Level 3

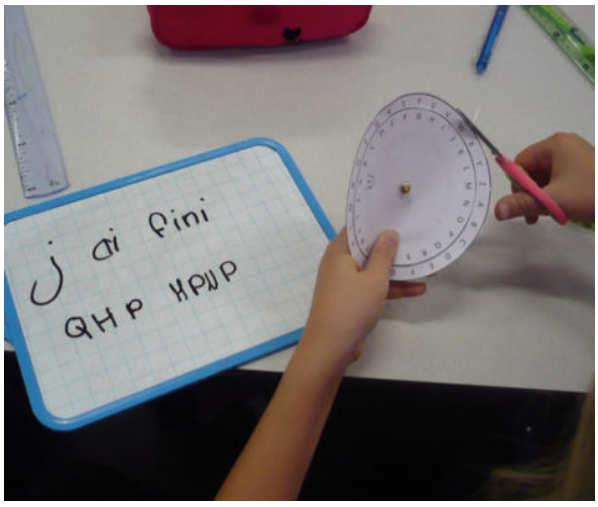


The other scenarios can be seen on page 56 (Level 1) and page 108 (Level 2).

The importance of unplugged activities

This teaching guide offers a selection of unplugged or "disconnected" activities, referred to as such as they require neither computer nor robot. There are several benefits to this approach. To begin with, the activities proposed in this guide demonstrate that several significant key ideas in computer science can easily be dealt with without a computer. This has practical advantages because no costly equipment is required, thus all schools can offer them.
In addition, lessons using a machine require extra preparation time so that the teacher can confront unexpected difficulties met in class (see pages 229 and thereafter for practical advice).





A few examples of disconnected activities in Levels 1, 2 and 3 address key ideas in language and algorithms, coding information (text or image) and encryption.
There are several advantages for teachers starting a computer science project with unplugged activities. During a plugged activity, the machine requires a degree of precision and attention to detail that gets in the way of students understanding the major principles. In practice, some students also have trouble listing to instructions or interacting with each other while using computers, as the screen attracts all their attention.[8]

8 On the topic of screen time, we recommend the excellent teaching project, "Screens, the brain and the child", published on the *La main à la pâte* website (https://www.fondation-lamap.org/en/international-resources).

An unplugged computer science activity is closer to a typical class activity, which makes it less surprising to students and teachers, simplifies working in groups or as a class, while avoiding the minor technical issues that are unrelated to the ideas being studied.
These unplugged activities must however be complemented with plugged activities wherever possible, because becoming proficient in computer skills is the ultimate goal of studying computer science. Some students may feel frustrated by class activities described as computer science, but which never call for computer use.

The educational benefit of robotics

Robotics provides a connection between the digital world and the physical world. It is one of the primary fields of application of algorithmics and programming that is often more motivating and reassuring that a simple computer screen. It is also an opportunity to introduce students to a major aspect of modern technology, illustrating the incorporation of computer science within physical objects.
Several studies have shown the highly positive impact of robotics on learning computer science concepts, among others[9]. The impact is even greater when it is paired with active learning where students experiment themselves, using a scientific inquiry-based method and a cooperative approach (see page 47).
Learning the scientific method, discussion and debate are reinforced by the tangible aspect of robotics. Students are motivated when handling a physical object, and this contributes to greater comprehension. The cooperative aspect provoked by robotics activities also provides students with a new space for expression. The possibility of testing programs on physical objects, confirming or disproving hypotheses by providing them with tangible expression helps students disassociate the notion of error from intellectual sanction and gives error a positive status, seen as a step in the learning process.

9 See, for example "IniRobot: a pedagogical kit to initiate children to concepts of robotics and computer science", D. Roy et al., RIE 2015. https://hal.inria.fr/hal-01144435

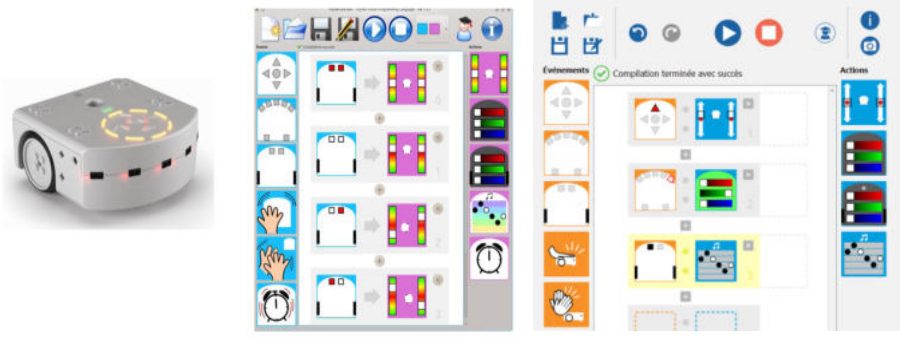
Thymio II, a robot designed for education

This teaching guide includes a selection of robotics activities using the robot Thymio II. Why this robot rather than another?

Thymio II is a small robot (11 x 11 x 5 cm) that is autonomous, mobile and robust, specifically designed for education and particularly suitable from pre-school to middle school. Thymio II uses open source equipment and software and has a rich, ergonomic user interface. It is ready to use right out of the box, you do not need to assemble anything to make it work, while the Lego spaces on top and on the wheels allow it to be transformed with extra pieces. Input functions include five proximity sensors at the front, two at the back and two sensors on the top that measure the floor's brightness so that it can distinguish zones of different colors (to follow a black line, for example). There are five capacitive touch buttons on the top, a microphone, a triple-axis accelerometer to measure inclines and shocks, an infrared sensor for an external remote controller and a thermometer. Output functions include two engines to drive the wheels, 39 LEDs on the body, on top, underneath and on the sides, allowing the robot to use a large variety of color combinations. Thymio also has a speaker, a sound synthesizer and a micro SD slot to store music files, for example. All the sensors and actuators mentioned above are accessed by programming.
The robot can be programmed as follows:

- By a highly intuitive visual programming interface accessible to beginners and even children who cannot yet read;
- By a textual programming interface using simplified script language;
- By visual/textual programming software such as *Scratch*, Snap! and Blockly. For an example of how to program Thymio using *Scratch*, click here: https://dm1r.inria.fr/t/piloter-un-thymio-ii-avec-scratch/161

The combination of sturdiness, the variety of sensors and actuators, the user-friendly interface, the possibility of entirely visual programming and programming with many other languages make Thymio a robot that is remarkably suited to education.



Thymio II (left) and the VPL 1.4 programming environment (right).

Teaching computer science through active learning: inquiry-based and project-based learning

The multidisciplinary teaching project, "1,2,3...Code!" focuses on student activity through questioning, experimenting, observing, trial and error, programming and discussion. This "active learning" can take several slightly different forms, in particular the inquiry-based approach and the project method.

Computer science, unlike the natural sciences (physics, biology, etc.), is not a study of a pre-existing world, but rather a world created by man. While the inquiry-based approach applies to both the study of natural objects and that of artificial objects, studying a "science of artificial objects", such as computer science, requires the learner to build these objects themselves. This is why the inquiry-based approach must be used together with the project method.

Some lessons in the "1,2,3...Code!" project aim for concepts to be grasped by the students and clearly follow an inquiry-based approach, as may be used when teaching the "traditional" sciences. Other lessons focus specifically on skills development (which does not exclude the comprehension of concepts). This is the case with programming lessons using *Scratch*. These lessons follow the project method. Robotics lessons are in between. The robot can result in the completion of a project, but can also be the subject of inquiry itself.

What is an inquiry-based approach?

The inquiry-based approach, which has been promoted by *La main à la pâte* for 20 years, is now well established in primary schools. While acknowledging that a fixed model of this approach is oversimplifying, we can nonetheless identify three general phases:

- Questioning, initiated by the teacher or the students, that gives rise to the forming of theories;
- Research, which may be an experiment, an observation or documentary study;
- Structuring of knowledge which in turn leads to more questioning, more research, etc.

The following paragraphs provide a brief overview of the main phases of an inquiry-based approach.

The questioning phase

The variety of answers to this questioning, comparisons and discrepancies will lead to a problem that the students have to solve. The teacher's role is to lead the discussion that will lead the students to become aware of the problem and what they are trying to find out or demonstrate. To do this, the teacher encourages communication among the students and guides them in their thinking: *"What would you say is the answer? What do you think of this?"*

Forming theories

Using their experience or their knowledge, the students provide explanations that they believe to be plausible: they are the students' theories. Through inquiry and documentary research,

experimenting and/or modelling, students will be able to confirm whether their theories are right or wrong. The research phase is set off by the need to test the credibility of a theory.

The students can form ideas or theories (what they think they know, what they think they understand and can explain about a certain phenomenon) either individually or as a group:

- In writing, in the form of:
  - a drawing or a figure with a key;
  - a reasoned text;
  - a list drawn up collectively;
- Orally, through a group debate among the students.

The research phase

During this phase, guided by the teacher, the student works alone or in a group to find solutions to the problem raised. This involves testing the theories they have chosen. The teacher makes sure that the research methods have been found by the students themselves, they must not simply execute the teacher's instructions. The teacher may help them if they get stuck, for example by showing them the available materials.
When experimenting, modelling or direct observation are not possible, documentary research and even interviewing an "expert" (which may be the teacher) will enable the students to prove or disprove the theories previously produced.
The teaching guide "1,2,3...Code!" offers a large selection of inquiry-based methods. Here are examples of each type:

- Experimenting: use pixel grids of varying cells widths to test the effect of image resolution on the ability to recognize the object represented;
- Modelling: use a deck of cards to manipulate variables;
- Documentary research: study the history of computer science;
- Programming: create a video game; pilot a robot;
- Observations: disassemble a Thymio robot to see what it is made of.

Structuring knowledge

We have seen how questioning plays an essential part throughout the inquiry, whether this involves stating the problem, interpreting the result of an experiment, or comparing opinions, etc. Sometimes students need to go back and forth several times from questioning to research before finding a solution and construct new knowledge.
During the collective oral phase, the class builds their shared knowledge. Discussion plays an essential role. This group discussion must not be seen as a dialogue between the students and teacher, but as a dialogue amongst the students, facilitated by the teacher.
The entire class contributes to produce a group lesson recapitulation, that they all agree on and which summarizes what they have learned and understood. This conclusion also allows them to get some perspective on the activity they have completed so that they can begin to generalize and conceptualize. Precise vocabulary is key at this stage. The group lesson recapitulation is often a written text, but can be added to with other forms of presentation such as graphics, figures and timelines.
The class conclusion is a consensus reached, but this does not mean that it's valid! We can all be wrong! An often forgotten, but essential step in the inquiry-based method is comparing the

knowledge created in the class (our conclusions) with the common knowledge (the scientists' knowledge). This comparison can be conducted using books, documents and even with the teacher, who is also a guardian of the common knowledge.
In the teaching module, "1,2,3...Code!", model conclusions are presented at the end of each lesson. These are of course examples (based on the tests carried out in class) designed to guide the teacher. It would be a shame to use these conclusions as they are. We recommend that the teacher allow the students to draw up their own conclusions, based on the work they did in class.

The teacher's role in the inquiry-based method

While the students' activity is essential and prioritized, the teacher plays a dual role that is key. They are no longer simply the bearer of knowledge, but also the one that guides the students on a path towards constructing knowledge by themselves and acquiring hard and soft skills.
To do this, the teacher relies on the knowledge they have on their students' abilities as well as the current level of class progression. They must be attentive to the general atmosphere, and the pace of work for each student or group, provide support or spark the students' reflection when necessary, playing a moderator role ("*What do you think*?" "*Do you agree with what was said*?", etc.).The teacher decides if the class should move on to another activity, when refocusing or during the generalization. For these reasons, the teacher is described as the class "tutor".
The teacher also plays an intermediary role, establishing whether the "facts" observed are indeed facts, and that they comply with the "official" science (the science of the experts). They also decide, providing an explanation, whether the students' suggestions can be accepted for consideration or studied through experiments; lastly, as an expert or reference point, the teacher bears witness on the scientific exactness of the results of the class's work. For this reason, they are the scientific "mediator" for the class.

Sciences and language proficiency

Oral and written communication is present throughout the project "1,2,3...Code!". The science notebook, in particular, is a precious tool, and their use is worth mentioning in detail.
Writing helps create distance, clarify and formulate thoughts, making them comprehensible to all. Students that are new to the inquiry-based approach tend not to write naturally. This activity therefore requires practice, which will be effective if the students understand its usefulness. All writings, in all their many forms (drawing, figures, legends, descriptive or explanatory texts) contribute to the learning process.

- Students write for themselves
  Writing helps the students to act (select an experiment, make decisions, plan, anticipate results), memorize (keep a record of observations, research, readings, look back on a previous activity) and understand (organize, sort, structure, compare to previous writings, reformulate collective writings).
- Students write for others
  Writing lets the students share what they have understood, question the other students and also people outside the class (other classes, family, etc.), explain what they have done or understood, recapitulation, etc.

The science notebook can be organized in two parts: individual and collective.
Individual writings represent the student's personal space, where they write down their initial

answers to the questions raised, describe the activities that enable them to answer these questions, jot down what they anticipate, and draft their reports. These writings may be in the form of texts, as well as figures, drawings and graphs. They serve as a driver of thinking and a record of the action. As such, for the teacher they are a way of monitoring progress and the personal progress of each child. It is important that the teacher does not play an authoritative role in the student's personal writings (by correcting mistakes, for example). They may, however, help the child structure them gradually. Written texts that start out with little preparation or structure will develop gradually with a description of the experiments (list of equipment, procedure, figure or drawing), noting of results and their interpretation, and conclusions.
Collective writings are the result of the students comparing ideas and suggestions. They then become "valid" writings and must respect spelling and syntax rules, and be enhanced with specific vocabulary.

Evaluating concept acquisition

How can we evaluate the knowledge and skills acquired by the students throughout a project such as this one? The answer to this question will primarily depend on how this evaluation will be used. Is it a question of checking that the students have properly grasped a particular key idea at the end of the project so that they can be graded, for example? Or, is it more about collecting indicators of their level of comprehension throughout the inquiry approach, which will help the teacher adapt their progression method?
The first case is referred to as summative evaluation. Considering the length of the teaching module, the diversity of levels it is designed for and the variety of possible paths, we cannot include a summative evaluation procedure here. However, examples of evaluations conducted in the classroom are available on the project website (see page 342).
In order to be precise, reliable and useful, the evaluation of knowledge, skills and attitudes can be complemented with regular observation of the student's behavior, individual and group work and the lesson recapitulation written in their experiment notebook.
This type of gradual evaluation allows the teacher to adapt the progression. If the teacher then notices that some students are struggling with a key idea, they can spend a few minutes or an entire lesson on another activity. This detour will allow the teacher to deal with the concept that has been poorly grasped by some students in a different way, without boring the others.
The science notebook can be an excellent tool for formative evaluation, if the students use it systematically for writing down their thoughts about the problem studied (ideas, conceptions, projections, suggestions or theories), explain how they are going to solve the problem (the experiment procedure, for example), report their results, explain what they have understood individually, as a conclusion, before preparing and writing up a collective recapitulation with the class.

What is the project method?

Project-based learning is a fully-fledged concept of active learning. First described in the early 20th century (initially by John Dewey, who also described inquiry-based learning), it was for a long time confined to primary education before gradually spreading to secondary and higher education.
The teaching guide "1,2,3...Code!" proposes two sequences that are fully based on this approach; they are the sequences dedicated to programming in *Scratch Junior*, for Level 2 (see page 143) or in *Scratch*, for Level 3 (see page 229).

Key aspects of PBL:

Students work on a complex task (a problem on which they conduct an investigation to solve it; a question to which they must research the answer). The purpose of the project method in "1,2,3...Code!" is to enable students to learn to program based on an open question, rather than repetitive exercises (exercises are however necessary, at the beginning, to grasp the programming environment);

- The task may be a practical, hands-on one, leading to the students producing something tangible; or it may be conceptual, to justify a point of view (for example, a philosophy workshop). In the case of the "1,2,3...Code!" project, the aim is to produce something concrete: programming a video game.
- The period spent on this project in the classroom allows the students, as a group or individually, to conduct a real investigation to answer the questions or solve the challenges presented in the task. These questions are often open-ended, in that they do not necessarily have a single, pre-established answer. During the project, students mostly work in groups and their productions are personal creations rather than something that already exists.
- The project is managed by the class group and not the teacher alone (who organizes and directs but takes no decisions without the students);
- This complex task may be broken into more basic tasks where the students have a high degree of independence and get actively involved.

There are many benefits to this approach:

The student is aware of what they are doing, and why they are doing it.

- Due to its practical nature, the project is particularly motivating. It is important that task engages the students and that they make sense of *their point of view*, so that they want to solve the problem. It is also important that they see it as manageable, and the challenge is not beyond their reach. If it is too easy, it's pointless; if it is too hard, they become discouraged. The fact that they are living an "authentic" experience (they are not just solving a school exercise, they're creating a real video game) is a source of great satisfaction.
- Learning the concepts is made easier by the fact that they are given context and have meaning. The question, "*how will we count the score or the number of lives in our video game?*" will lead students to the key idea of variable and the actions that go with it: *How do we create a variable? Why must we initialize it? How do we do that? How can we change its value? How can we use this value in another part of the program (for example, when the number of lives is zero, must the game end)?* etc. A wide variety of knowledge and skills are used throughout the project, without any arbitrary disciplinary boundaries.
- Students change their perspective of error, which becomes an integral part of the learning process. They are highly encouraged to proceed through trial and error. Indeed, the project consists of programming a game: making a mistake does not matter, there are no possible malfunctions (a program with errors will not damage the computer) and the verdict is evident instantly. This is something the students will particularly appreciate: they know straight away if what they suggested worked or not, and the machine tells them, not the teacher nor classmate. The ease with which they try it out

and the lack of serious consequences of a mistake facilitates their independence and decision-making. Independence is genuine in a programming project. There are often several methods (some more elegant than others) to solve a problem.

- The project enables the students to use several cross-functional skills, such as decision-making, planning, etc. Although it is possible to take a class on project management, it is a field where experience is key. We learn how to successfully complete a project by doing it.
- Collective intelligence is showcased. Cooperation is essential, because if the project is well chosen, each student will, at some point, face a task that they are unable to solve alone. Starting off the year with a project is an excellent way to create social ties in the class and help students bond with an activity that they work on together. In addition, because it has a concrete, "non-academic" objective, it helps bring students with difficulties out of their shell, as they often quite withdrawn.

Avoid interruptions to the project

It can be very tempting for the teacher to take advantage of this project and the difficulties encountered by students and gradually introduce more "traditional" lessons. This is sometimes justified (helping students grasp a new key idea, conceptualize, and get some perspective) but teachers must be careful, as these interruptions take away from the project's value. They must not occur too often. For this reason, in our programming sequence for Level 3, we propose some activities that can help students grasp certain key ideas (variable, loop, logical operator, etc.), by making sure that 1) they are optional and 2) they are practiced at another time, outside the class time given to the project, so that they do not interrupt the project.

The teacher's roles in project-based learning

Just like with inquiry-based learning, project-based learning places the focus on student activity. This, however, does not mean the teacher does not have a role to play; quite the opposite!
The teacher ensures the project is scientifically and educationally beneficial, they help define the project and ensures that its objectives are attainable by the students. They anticipate the various steps and transform them into simplified tasks that enable the students to be even more independent.
When necessary, the teacher supervises the distribution of tasks between the students.
The teacher is also a facilitator: they refocus the activity or the discussion on the problem to be solved, reminds the class of the objective, and in principle tries not to express their opinion on the students' suggestions in order to encourage them. Similarly, they make sure that every student participates in the project. In concrete terms, in our programming sequence for Level 3, we recommend that the teacher put the students into pairs to work, but with each pair switching places every 10-15 minutes so that everyone gets their turn to control the keyboard and mouse.
Finally, the teacher is a regulator of the class discussions, according each student time to speak and helping them formulate the summary. They can also play a role of advisor/expert by offering possible solutions or informing them of something they did not know if they find themselves stuck.

Pedagogical module: Class Activities

Level 1 Activities

Overview

The activities module for Level 1 includes two sequences. The first is entirely unplugged (done without a computer, tablet or robot and using only motor skills development materials) while the second is a plugged activity (using robots).

- The first sequence (entirely unplugged) lets students invent and use a language to program a sprite's movements. Little by little, they enrich this language with new instructions, tests and loops.
- The second sequence (entirely plugged) introduces students to the basics of robotics: understanding that a robot can interact with its surroundings (by manipulating a Thymio robot); see page 82 for more information).

A review lesson is available on page 96: it can be taught after Sequence 1 or Sequence 2.

Lesson summary

Sequence 1: Playing robot

	Lesson	Title	Page	Summary
	Lesson 1	Moving an object around a grid	58	Students learn how to give precise orders to a sprite to control its movements around a grid.
	Lesson 2	Challenge: Programming a sprite's movements along a route	63	By combining instructions from the previous lesson, students design a program to create a complex route for a sprite.
	Lesson 3	Formative assessment: Other routes, other programs	65	The students write and interpret programs for other routes.
	Lesson 4	Conditional routes: Treasure hunt	69	Students enrich their programming language with conditional constructs (if–then statements).
	Lesson 5	(Optional) A route of any length: Loops	77	When routes become long or complex, students begin to understand the importance of simplifying a program: they discover that loops can be used to avoid repetitions.

Sequence 2: Playing with robots

	Lesson	Title	Page	Summary
	Lesson 1	Introduction to the Thymio robot	82	Students are introduced to the Thymio robot and learn how to manipulate it.
	Lesson 2	Colors and behaviors	86	Students learn that Thymio has several modes and can behave differently depending on the chosen mode.
	Lesson 3	Thymio in Investigator mode	89	Students discover Thymio's turquoise mode and prepare a route that Thymio can follow alone.
	Lesson 4	Challenge: Get Thymio through a maze	93	Students build a maze and must find all possible ways to get Thymio through it.

Conceptual scenario: "Level 1 computer science"

The key ideas covered during these two sequences for Level 1 can be organized as follows.

Sequence 1: Playing robot

	Lesson	Title	Page	Summary
	Lesson 1	Moving an object around a grid	58	Students learn how to give precise orders to a sprite to control its movements around a grid.
	Lesson 2	Challenge: Programming a sprite's movements along a route	63	By combining instructions from the previous lesson, students design a program to create a complex route for a sprite.
	Lesson 3	Formative assessment: Other routes, other programs	65	The students write and interpret programs for other routes.
	Lesson 4	Conditional routes: Treasure hunt	69	Students enrich their programming language with conditional constructs (if–then statements).
	Lesson 5	(Optional) A route of any length: Loops	77	When routes become long or complex, students begin to understand the importance of simplifying a program: they discover that loops can be used to avoid repetitions.

The class can then continue on with Sequence 2, page 81 (if robots are available) or the Review lesson, page 96.

Lesson 1 - Moving an object around a grid

Summary	Students learn how to give precise orders to a sprite to control its movements around a grid.
Key ideas *(see Conceptual scenario, page 56)*	"Machines" • The machines all around us simply follow "orders" (instructions) "Languages" • To command machines, we invent and use languages. • A program is written in a language that both humans and machines can understand. "Algorithms" • A program is a combination of instructions.
Inquiry-based methods	Experimentation
Equipment	For the class: • A sprite • A token • A poster on A3 or A2 size paper with a 3x4 grid • Three copies of Handout 1, page 62 (laminated, if desired) • Magnets to keep everything on the whiteboard/chalkboard
Glossary	Instruction, program
Duration	30 min

Foreword

This sequence is aimed at very young students. For students at the beginning of Level 1, the activities will most often be done orally as a class, while students at the end of Level 1 can do them in small groups, creating drawings of what they have learned. For older students, (Level 2), the activities can be done in groups with written conclusions. Several variations of the same activities are suggested to accommodate these different age groups.
The goal is to move an object (the "sprite") from a starting point to a destination point. Setting the scene for this activity is very important to pique the students' interest. Any object or stuffed animal can be used as the sprite. To create a convincing reason why the sprite wants to go to the destination point, another object is added (the "reward"). The teacher can choose any reward depending on the choice of sprite; for example, a teddy bear will look for a pot of honey, a pixie will go to pick a flower, etc.
Must this activity be done on a whiteboard or table? Both options are possible, but the "program strip" (see following lessons) or instruction cards must fit on the same support. If the grid is hung on the board with the magnetic sprite on it, the instruction cards must also be placed on the board with magnets. The same applies if you are working on a table. While using arrows may be easy for adults, this is not the case for children. For example, what does an upward-pointing arrow on the board mean for a sprite lying flat on a table? Go up (in altitude)? Move away from the student (confusing if the student is not directly in front of the grid)? Move towards the top of the grid (which is what we expect)?

Can students play the sprites themselves in the classroom? It is possible, but care must be taken to ensure that no errors or misconceptions are introduced. For practical reasons, it will probably be difficult to place giant instruction cards on the ground where they can be read. However, for the reasons mentioned above, it is inadvisable to hang them on the wall; students will be able to read them, but they may not understand them. It is possible to invent a new language for this activity: instructions can become "move one square towards the cafeteria," "move one square towards the playground," etc. However, it may be difficult to apply the logic to later lessons.
Where possible, we suggest using a sprite that does not need to face a particular direction. Here, the focus is on instructions for movement (go right, go left, etc.) without complicating matters with issues of directionality (turn a quarter turn to your right, etc.). However, if the class chooses a sprite with front, back, right and left sides, face the sprite in a single direction (e.g., front towards the top of the grid) and keep it facing this way at all times. This means that a move to the right would correspond to a "step right" for the sprite, and not a turn to the right followed by a step forward. The other option would be to face the sprite to the right, because at this age, most children have become accustomed to "reading" from left to right. So, saying "move forward" to go right would not pose any problems.

Starting the activity

The teacher presents the grid to the class and places the sprite on one of the squares. They tell the class that they must give the sprite orders so it can move around the grid.

Experiment: Giving orders to the sprite

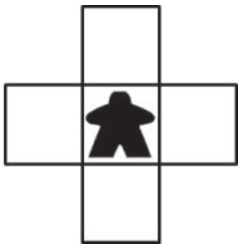
Orally: As a class, the students suggest orders to give to the sprite: *"Walk," "Go," "Go over there,"* etc. There are numerous possibilities.

- Written: In groups, ask the students to find four orders (written or drawn) to control any move on the grid.

First, place a reward on the grid that the sprite must collect. Place it two or three squares away on the same row or same column as the sprite. If the students' orders are vague (*"Go!"*), the teacher asks, *"Go where?"* The same order ("go up", for example) repeated two or three times is sufficient. The formulation "Go three squares up" also works.

Teaching notes

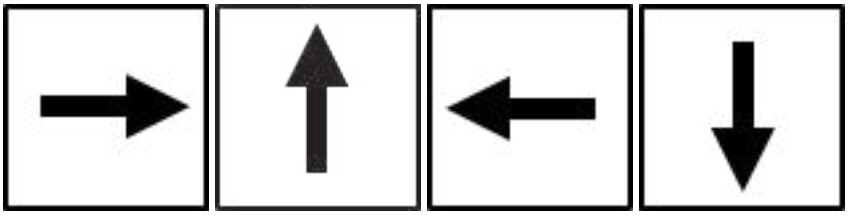
- For younger students, the grid should be extremely simple: the squares can either be aligned (the grid is one dimensional and students give instructions with a number of times to go right or left) or placed in a cross configuration, where from the center square there is only one square up, one square to the right, one square down and one square to the left.



The reward can also be placed at a diagonal from the sprite (not for younger students). The students may suggest the sprite move diagonally, but the teacher then explains that it cannot: it can only move to one of the four squares that share a border with the square it is in. Students must combine two orders, such as “go up” and “go right.”
The teacher then asks what four orders the sprite can obey. (If students suggest eight, remind them that diagonal movements are not allowed.)

Group discussion

The teacher puts the class’s different suggestions of orders for the sprite up on the board. The class discusses these suggestions and chooses the signals they want to use.
In the following lessons, we’re using a simple signaling system of arrows to indicate the direction the sprite should move. The teacher can ask students to draw arrows on cards or use the arrows from Handout 1, which will need to be cut out (and laminated, if desired).

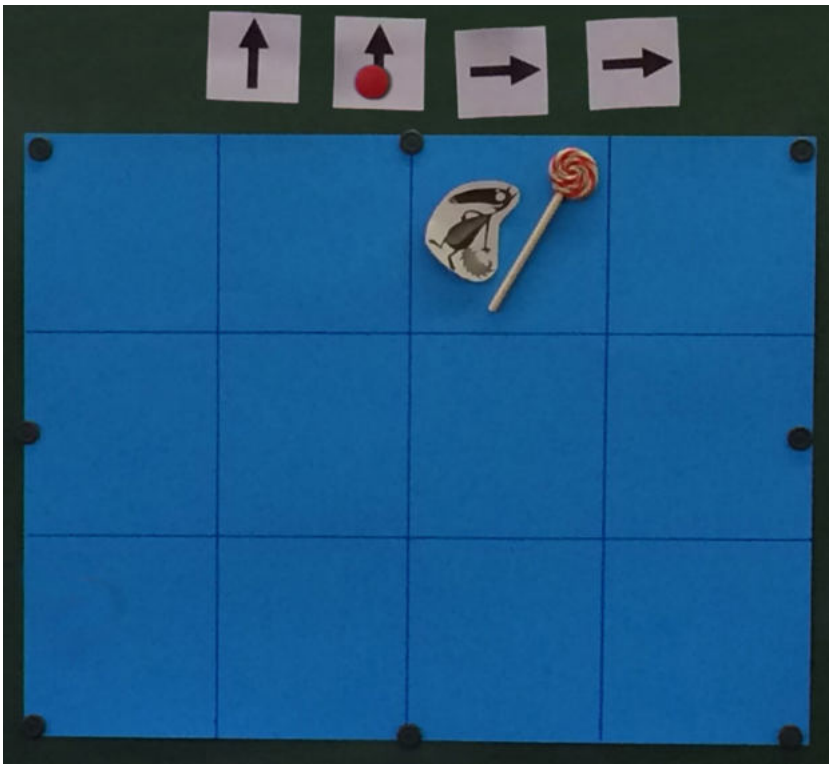


The teacher introduces a new glossary term: the orders given to the sprite on the cards are “instructions.” The teacher asks the students to explain what each card means. Each card corresponds to the movement of the sprite from one square following the direction of the arrow.

Teaching notes

- This method of giving instruction is called “allocentric”: if the grid is placed in a particular direction, the instructions have no effect on the direction the sprite is facing. In the class, these instructions can be reworded, such as “move one square towards the whiteboard,” “move one square towards the door,” etc. Later, in geography class, the four cardinal directions can be used. For younger students, more context can be given for the grid by drawing a faraway environment with various colors: “go towards the red mountain,” “go towards the blue sea,” “go towards the green forest,” “go towards the yellow desert,” etc. Contextualizing the environment can be useful, especially at the beginning for younger students, in helping them learn what arrows mean. However, for ages five and up, we suggest using just arrows. This helps students learn spatial awareness.

- For the sake of convenience, we will call the four cards above “instruction cards.”



Preschool class, Jessica Mazoyer (Paris).

Role play

The teacher hangs a long roll of blank paper above the grid on the board. This is the "program strip" on which the instruction cards will be placed, side by side from left to right, to be followed. The teacher adds the first instruction card to the program strip and places a magnetized token on it: the class moves the sprite on the grid. The teacher then adds another instruction card after the first one, moving the token onto this new card (the token indicates the instruction being carried out). There is no need to remember the previous instructions or prepare the following instructions ahead of time. The teacher then adds another instruction, followed by another. The class reads and applies the instructions one by one, moving the token along the program and the sprite on the grid.

Conclusion

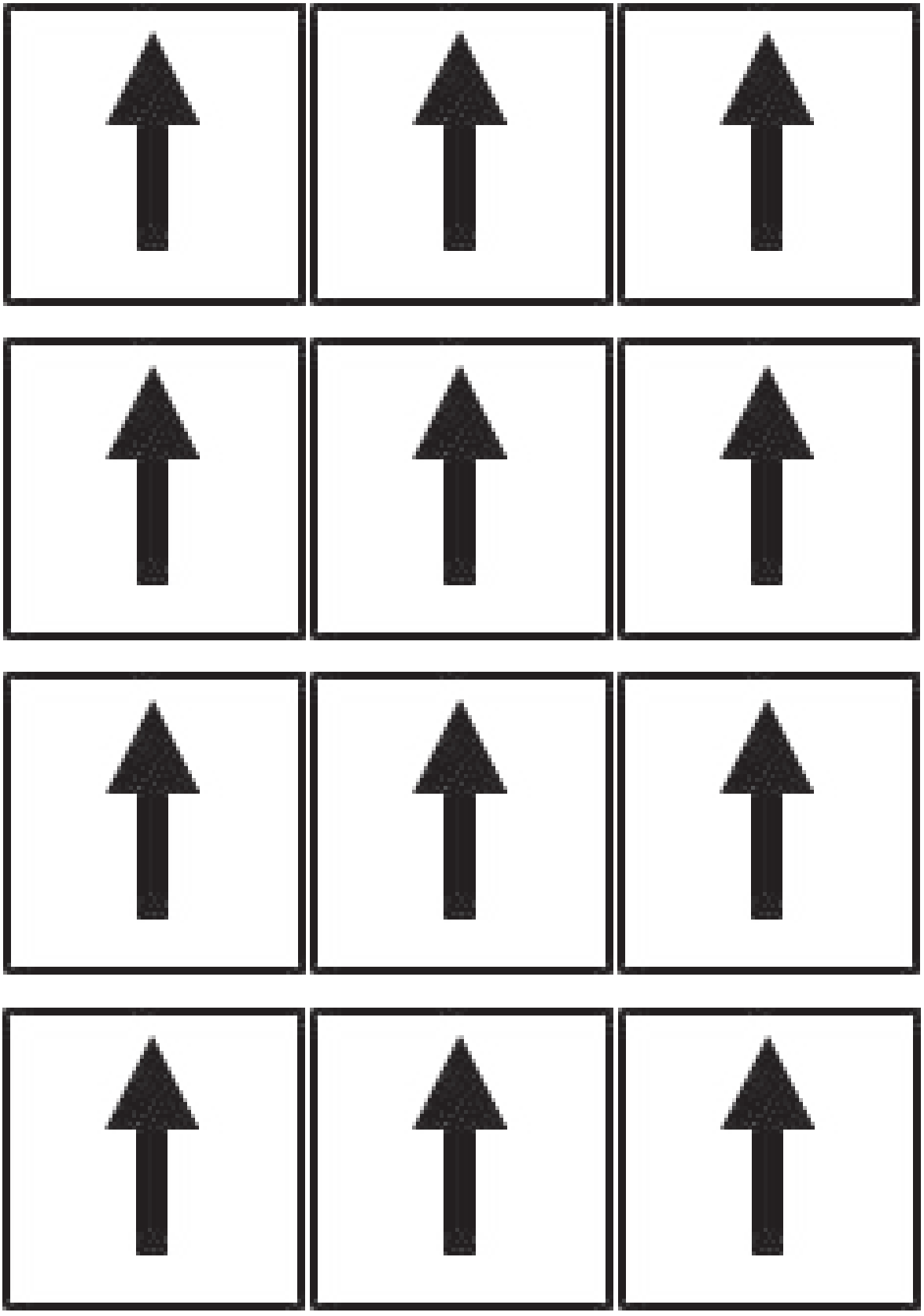
The class summarizes together what they learned in this lesson:

- *To move the sprite, you can give it simple orders, called "instructions."*
- *By combining instructions, you can write a program.*

Further study

The teacher places the sprite in the center of the grid. Half the class secretly hides the reward under the grid, writing the program to find it from the square where the sprite is placed. The teacher asks the other half of the class to find the reward by following the suggested program. The two groups then switch roles.

HANDOUT 1
Instruction cards for the sprite



Lesson 2 - Challenge: Programming a sprite's movements along a route

Summary	By combining instructions from the previous lesson, students design a program to create a complex route for a sprite.
Key ideas *(see Conceptual scenario, page 56)*	"Machines" • The machines all around us simply follow "orders" (instructions) • By combining several simple instructions, we can perform a complex task "Languages" • To command machines, we invent and use languages • A program is written in a language that both humans and machines can understand. "Algorithms" • A program is a combination of instructions.
Inquiry-based methods	Experimentation
Equipment	For each group: • A sprite • A poster on A3 or A2 size paper with a 3x4 grid • Several copies of the instruction cards from Handout 1 (copied or drawn by the students during the previous lesson)
Glossary	Program, language
Duration	30 min

Preparation

Before this lesson, the teacher should prepare or have the students prepare several copies of the instruction cards from Handout 1 (in all, six copies of each instruction card will be required for the entire sequence).

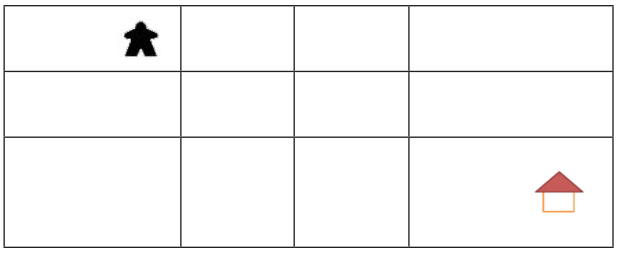
Teaching notes

- For preschool students, plan to have a set of instruction cards and a grid for each student (or student pairs). Kindergarten students can begin to work in groups of four.

Starting the activity

As a class, review the conclusions from the previous lesson: *By giving instructions, you can move the sprite where you want around the grid*. The teacher reminds the students what they did at the end of the lesson: compile the instruction cards one after another, without removing any of them. The teacher introduces the term "*program*": a program is a set of instructions.
The teacher presents the grid to the class and places the sprite on one of the corner squares. They ask the class to create a program that will help the sprite go home at the opposite corner of the grid (the sprite and the house are each on a grid square).

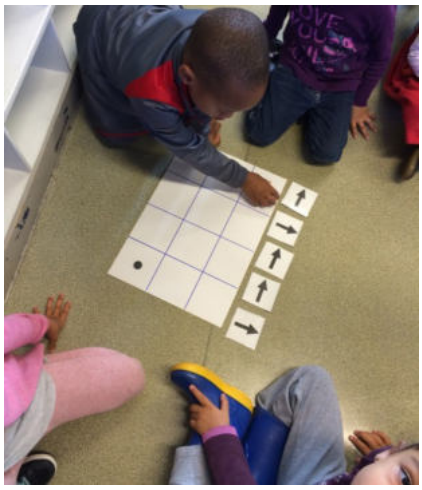
Start



Destination

Experiment: Create a program for the sprite (in groups)

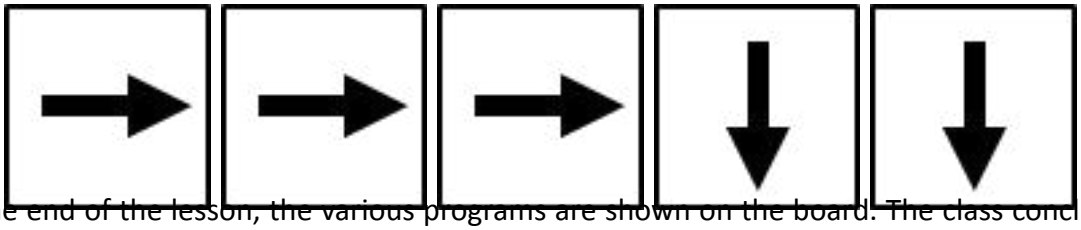
The students are split into small groups, each with its own sprite, a grid, a program strip and enough instruction cards (four copies of each) to program the sprite's movements. The teacher asks them to find two different routes to guide the sprite home. The students combine their instruction cards and test their routes to see if the program works.



On the left: Caroline Fayard's kindergarten class; on the right: Jessica Mazoyer's preschool class, Paris.

Group discussion

The teacher asks each group to present one of their programs. There are multiple possibilities. For example:



At the end of the lesson, the various programs are shown on the board. The class concludes that sometimes there are many ways to get the same result.

The teacher explains that these cards form a language that (in our game) can be understood by both the sprite and people: it is a *"programming language."*

Conclusion

The class summarizes together what they learned in this lesson:

- *By combining several simple tasks, we can perform a complex task.*
- *A program is written in a language that the sprite and students can understand.*

Lesson 3 - Formative assessment: Other routes, other programs

Summary	The students write and interpret programs for other routes.
Key ideas *(see Conceptual scenario, page 56)*	"Machines" • The machines all around us simply follow "orders" (instructions) • By combining several simple instructions, we can perform a complex task "Languages" • To command machines, we invent and use languages • A program is written in a language that both humans and machines can understand. "Algorithms" • A program is a combination of instructions.
Inquiry-based methods	
Equipment	For the class: • Handout 2 (a copy for the class or for each student, depending on the chosen method) For each student pair • Handout 3
Glossary	
Duration	30 min

Foreword

This formative assessment aims to verify that students have understood the key ideas covered in previous lessons. It can be done orally as a class or individually. If doing it individually, plan to have one copy of Handout 2 for each student as well as one copy of Handout 3 per student pair.

Exercise 1: Executing a program

The teacher reviews the key ideas covered previously: a program is a sequence of instructions given to the sprite in a specific language. They suggest practicing on new programs and routes. The teacher then passes out Handout 2 to the students. The exercise consists in executing each of the programs step by step to find where the sprite finally ends up. The sprite should start the exercise from the corner of the grid and continue on with each new program from the square where it ended up. To set the scene for the exercise, the teacher can tell students that the pixie/sprite must go to pick a flower (green route), then get water (blue route) before going home (red route).

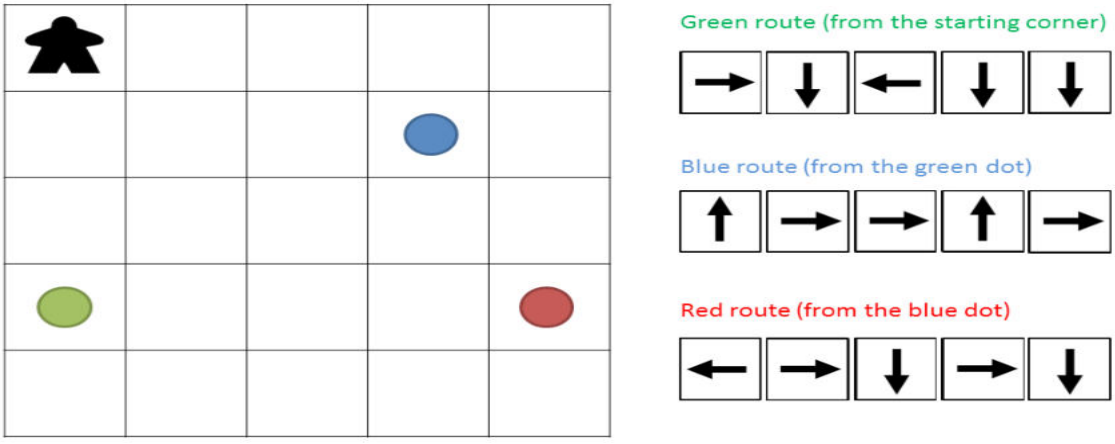
Teaching notes

- For preschool students, plan to have tokens to use as the sprite and another token to keep track of where students are on the program strip. At this age, children have

a hard time following both the sprite's movement and the program instructions with their finger.

- To make sure all students start from the right point at each step, it is best to correct any errors for each route before starting the next one.
- The red route is the most difficult, because it asks students to go back from where they came from, which can be problematic for certain students (they may wonder, "Why go back?"). Do this activity last or use it as an optional exercise.

The answers for the first exercise are as follows:

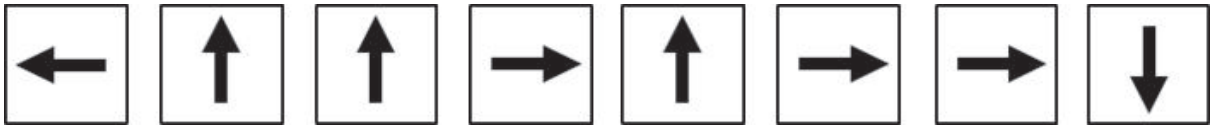


The class takes the time to do each program step by step to check the final position.

Exercise 2: Writing a program

The teacher passes out Handout 3, which asks students to write a program that will take the sprite to a destination point with an obstacle to avoid. As in the previous lesson, there are several possible programs.

One example is:



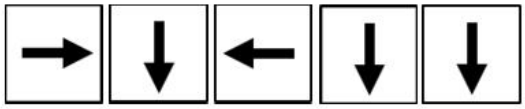
If students find the exercise difficult, the teacher can have them do additional similar exercises before moving on.

Further study

What students learn from this lesson (right, left, moving square by square) can be applied to other activities. For example, board games use these types of instructions to move tokens along a grid (e.g., Snakes and Ladders) or in motor skills development activities.

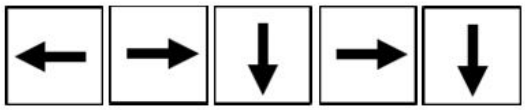
HANDOUT 2
One program, more programs (1/2)

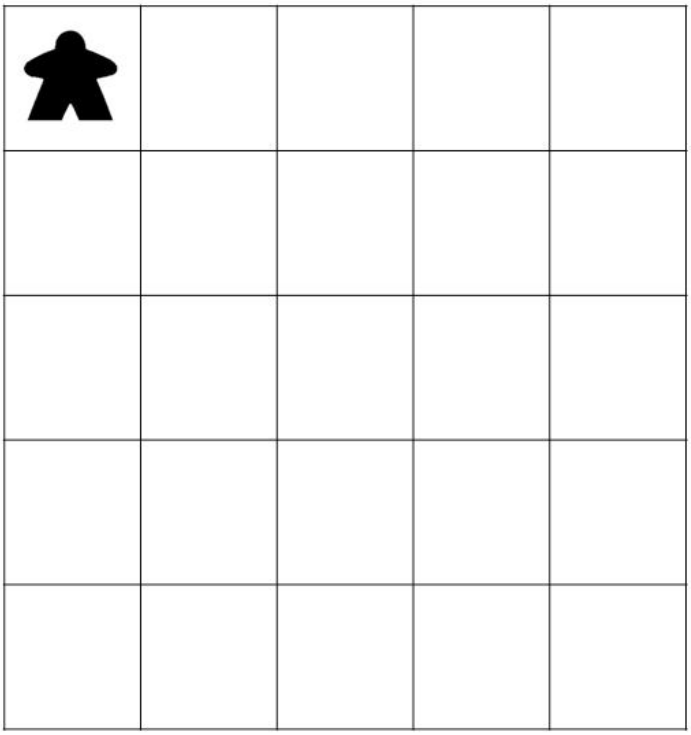
Green route (from the starting corner)



Blue route (from the green dot)

Red route (from the blue dot)

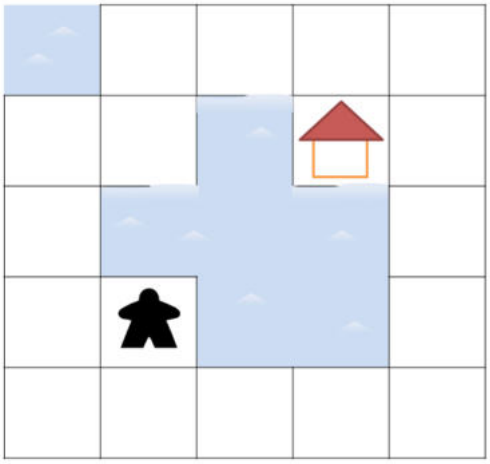




Instruction: From the upper left corner, the sprite must first follow the Green program. Color the square of its final position green. From there, it must follow the Blue program. Color the square of its final position blue. Lastly, starting from the blue position, it must follow the Red program. Color the square of its final position red.

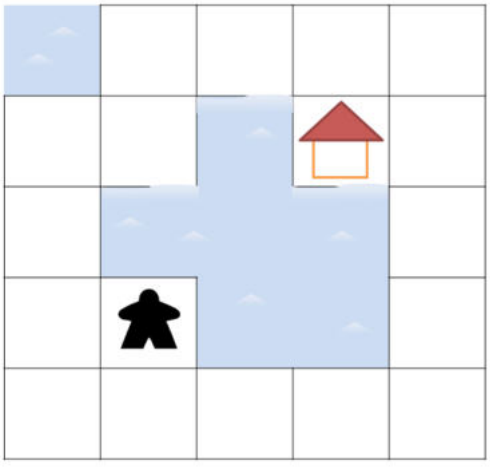
HANDOUT 3
One program, more programs (1/2)

PROGRAM



✂ **Instruction**: Write a program that will take the sprite home. Be careful: it must not fall in the water!

PROGRAM



Instruction: Write a program that will take the sprite home. Be careful: it must not fall in the water!

Lesson 4 - Conditional routes: Treasure hunt

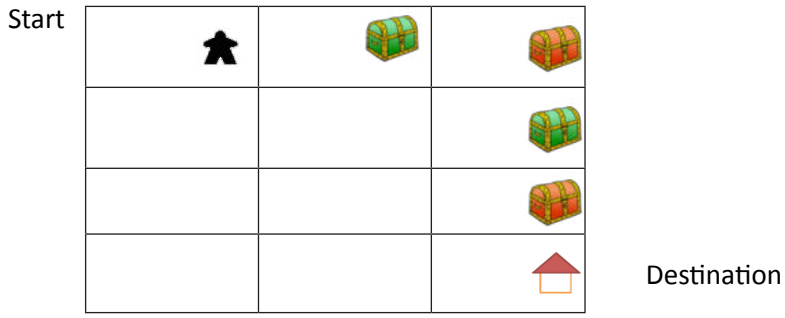
Summary	Students enrich their programming language with conditional constructs (if–then statements).
Key ideas *(see Conceptual scenario, page 56)*	"Algorithms" • A program is a combination of instructions. • In a program, tests say which instruction should be done when a condition is met.
Inquiry-based methods	Experimentation
Equipment	For the class: • A sprite • A poster on A3 or A2 size paper with a 3x4 grid • Several copies of the instruction cards from Handout 1 • Treasure chest cards from Handout 4 and Handout 5 • New instruction cards: Handout 6 and two copies of Handout 7
Glossary	Conditions, tests
Duration	45 min, can be split into two sessions.

Preparation

To fill in the grid used during the previous lessons, the teacher creates or has students create tokens with treasure chests on them . The treasure chests can be green, red or neutral (gray). On the back of the red chests, draw a monster. On the back of the green chests, draw a reward. The corresponding cards are found on Handout 4 (cut on the solid lines and fold on the dotted lines). After folding, the coins are hidden behind the green chests and the skulls are on the back of the red chests.

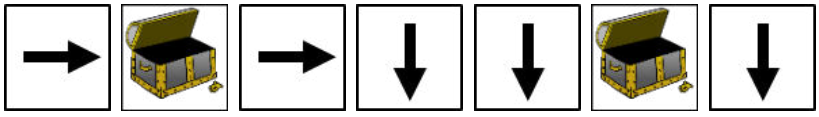
Starting the activity

The teacher takes the grid from the first lesson and adds green and red treasure chests along the routes (Handout 4). For example:



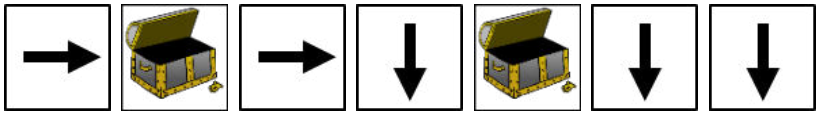
The teacher presents the treasure chest cards and explains the game rules: If a sprite opens a green chest, it gets a reward. If it opens a red chest, the monster inside will scare it and it will have to go back to the starting point. The teacher asks the class a simple question: "Using

the programming language we already used, will the sprite know how to open the treasure chests?" No – it only knows how to move around. The teacher then introduces a fifth programming language glossary term: "Open the chest" (the corresponding card is found on Handout 6). To emphasize that this card is essential to opening the chest (without this instruction card, the chest cannot be opened), the teacher suggests doing this first route as a class to help the sprite safely gather all the rewards and make it to the final destination. The teacher even gives students a program (which deliberately contains an error):



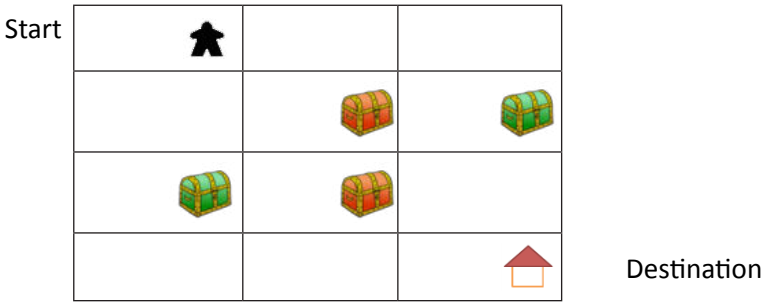
Open the first green chest, then build suspense by moving by the first red chest without opening it, but then forget to open the second green chest, and instead open the second red chest. This demonstration will help students remember that simply being on the same square as a chest does not mean it can be opened.

As a class, the students suggest how to correct the error:

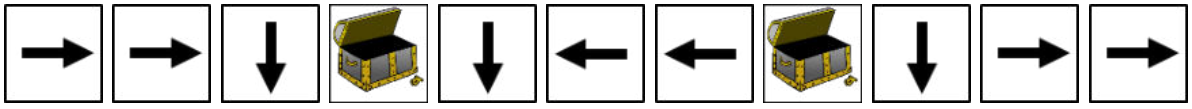


Experiment: Collect all the rewards while avoiding the monsters (as a class)

Following this practice exercise, the teacher traces out a new route, such as this one:



The teacher then asks the class to write a new program that will let the sprite safely collect all the rewards and return home. The class finishes this example with a program like this one:

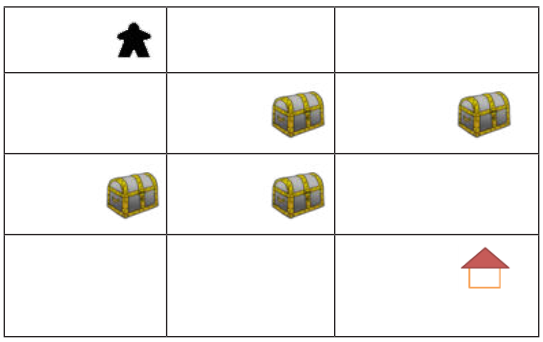


Preschool and kindergarten classes, Laurence Bensaid (Paris).

Experiment: Gathering all the coins along an unmarked route

This time, the teacher shows the class a similar route, but with one difference: the treasure chests are not red or green, but rather gray. Under each treasure chest card with a gray chest is hidden a colored treasure chest card (green or red), which indicates whether the chest contains coins or a monster. *"The sprite already knows where the treasure chests are, but does not know what color they are. What does it do?"*

Start



Destination

The class discusses the fact that the sprite must go to all the squares with a chest, but it needs to check whether the chest is red or green *before* opening it.

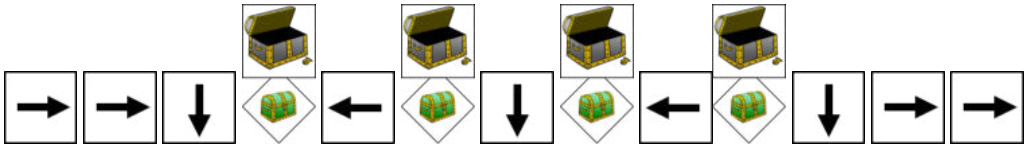
The class begins by trying to verbalize the necessary instruction.

IF the chest is green THEN it must be opened.

Scientific notes:

- The teacher can explain that if the condition is not met, nothing needs to be done: IF the chest is green, THEN it must be opened, OTHERWISE it is not opened.
- When the sprite is on the same square as a red chest, it still obeys this instruction and does not open the chest. This is neither an error nor disobedience.

The teacher then suggests a new instruction card (found on Handout 7). This card is a test; it includes a condition (here, *"Is the chest green?"*) and the instruction (here, *"Open the chest"*) to follow if the condition is met.
During the exercise, when the sprite asks the question, the teacher removes the gray treasure chest card and reveals the chest's real color.
The class should improve the previous program with this new instruction to help the sprite safely collect all the coins and arrive at the destination.
The final program created by the class may be similar to this one:



The instruction appears four times, once for each chest, because it is impossible to know ahead of time where the green chests are.

Conclusion

The class summarizes together what they learned in this lesson:

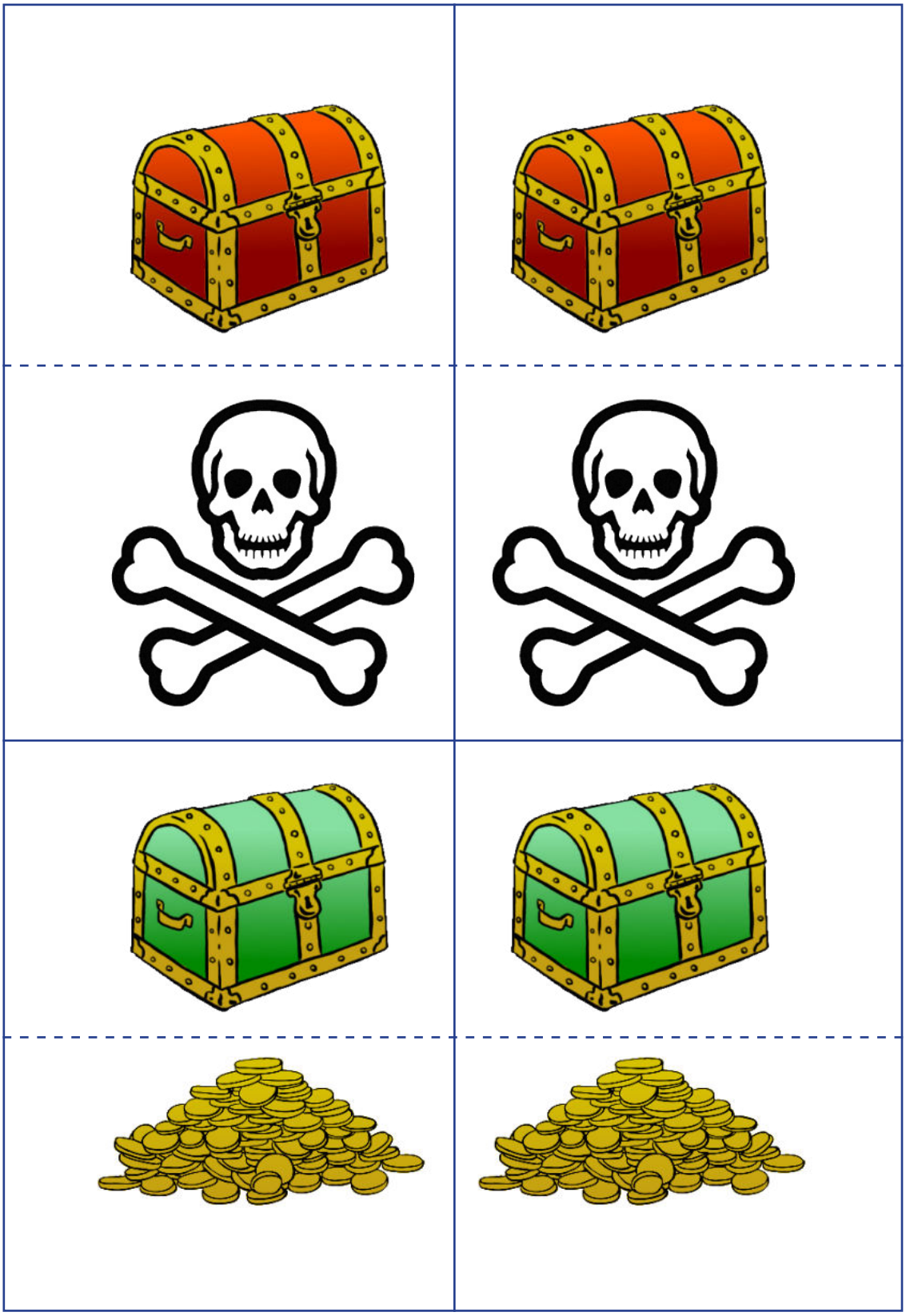
- *In a program, tests say which instruction should be done when a condition is met.*

Further study

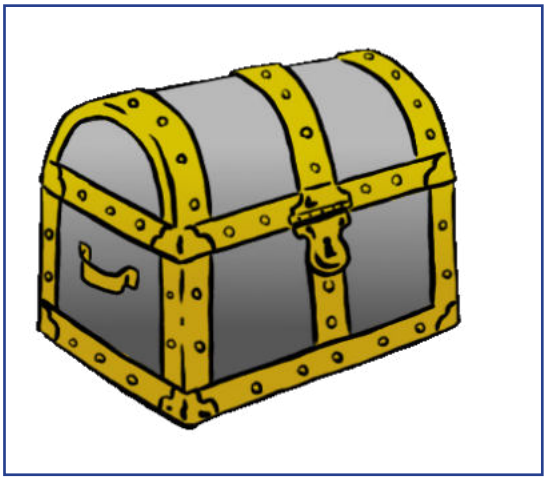
Suggest other routes, asking how many rewards the sprite will collect using the program.

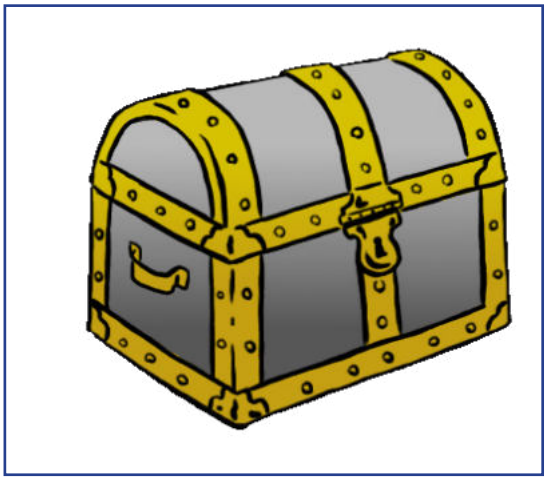
- For kindergarten classes: You can ask the students to create an "OTHERWISE" card: for example, "IF the chest in green, THEN the sprite opens it to get the reward, OTHERWISE it buries the chest so its friends will not accidentally open it."

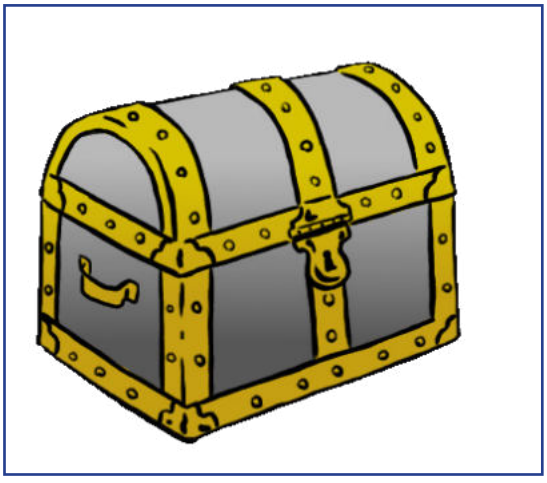
HANDOUT 4
The sprite's treasure chests: colored version

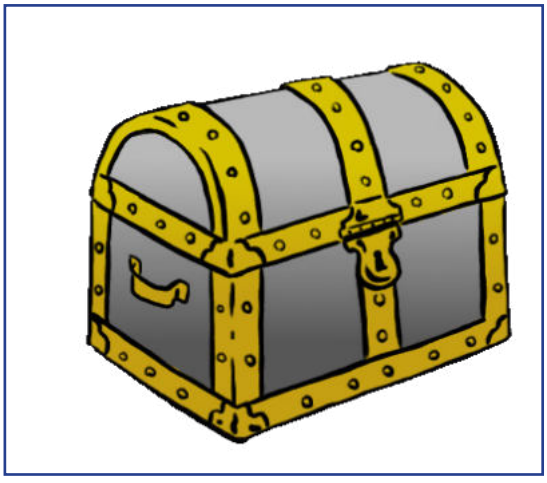


HANDOUT 5
Sprite's instructions: Conditional cards

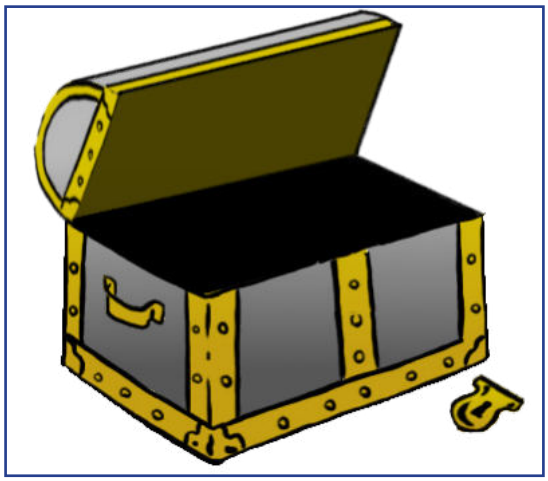


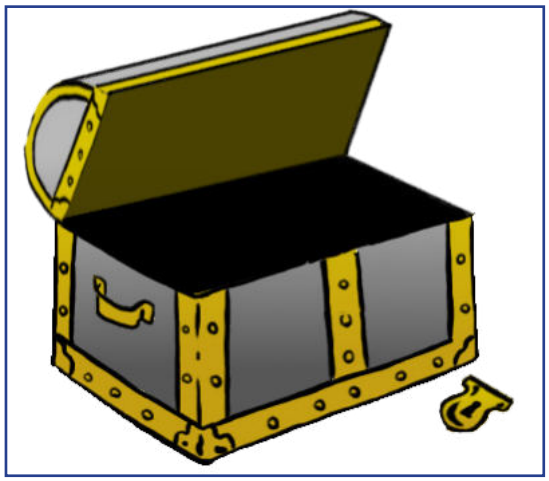


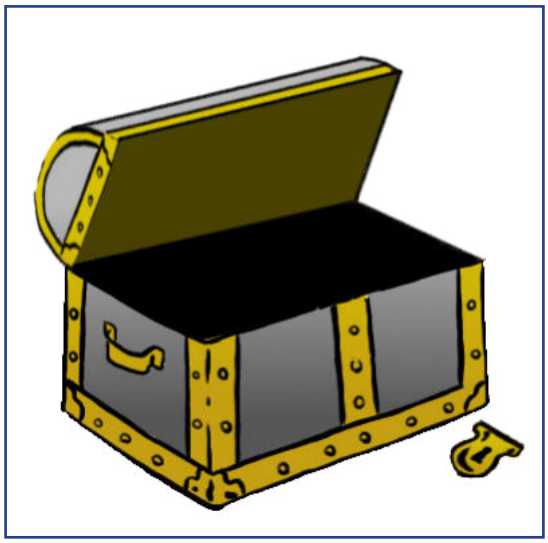


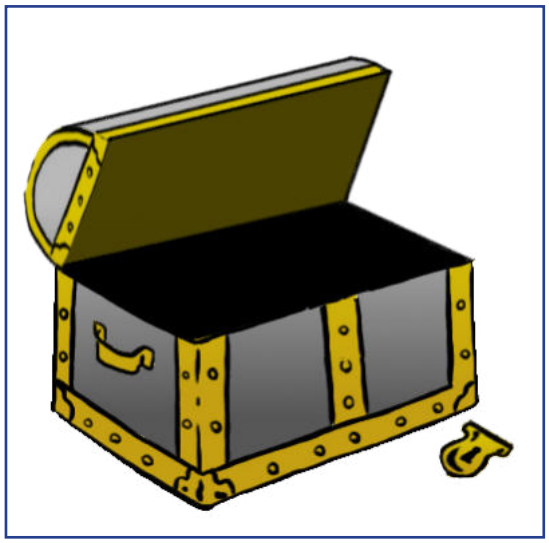


HANDOUT 6
Sprite's instructions: Conditional cards

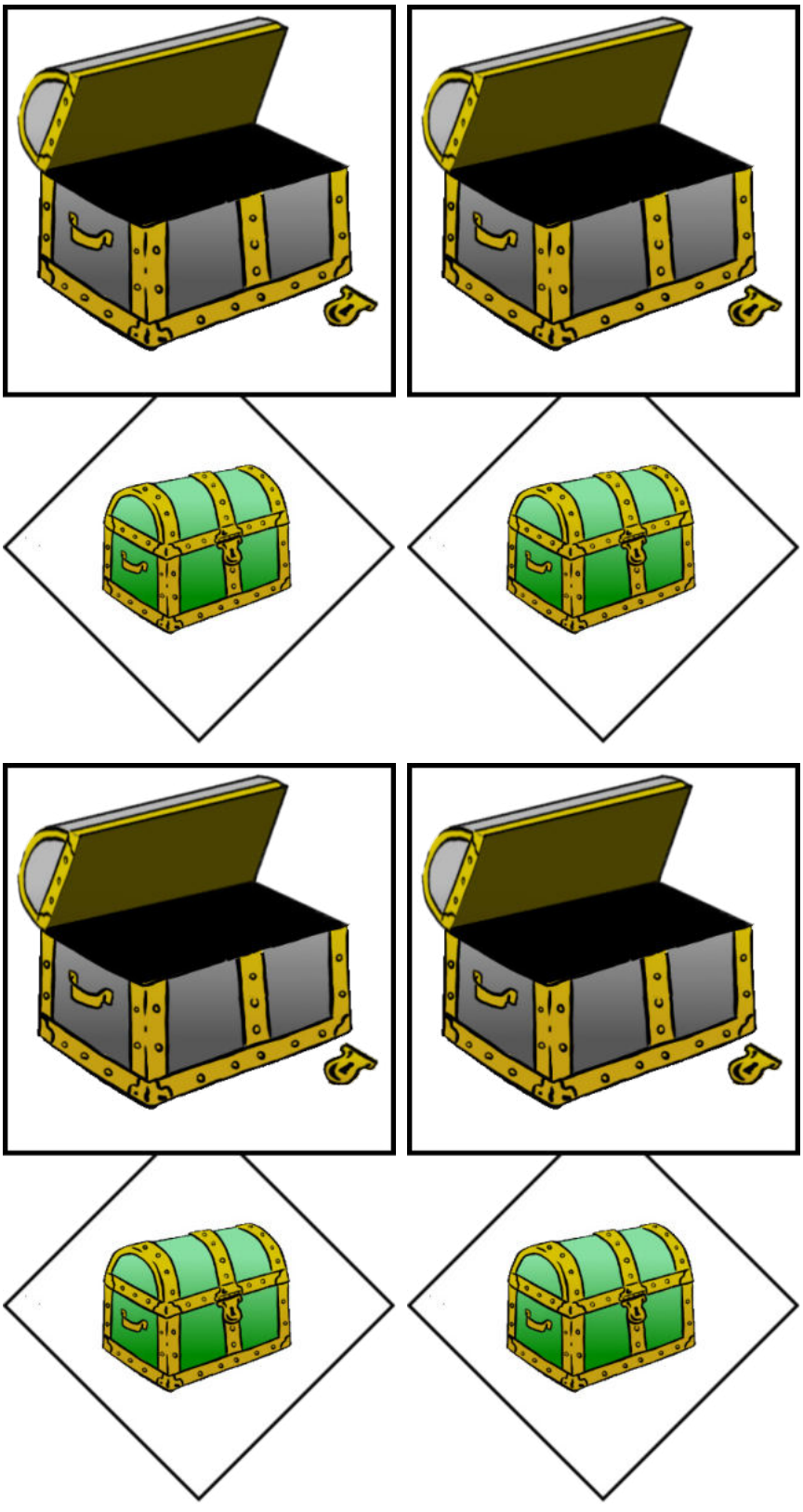








HANDOUT 7
Sprite's instructions: Conditional cards

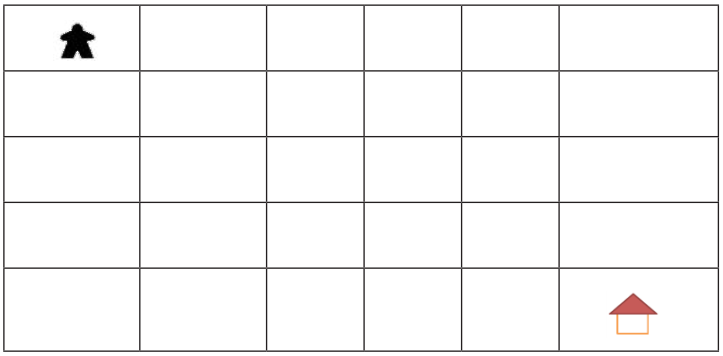


Lesson 5 - (Optional) A route of any length: Loops

Summary	When routes become long or complex, students begin to understand the importance of simplifying a program: they discover that loops can be used to avoid repetitions.
Key ideas *(see Conceptual scenario, page 56)*	"Algorithms" • A program is a combination of instructions. • In a program, loops are used to repeat the same instruction several times.
Inquiry-based methods	Experimentation
Equipment	For the class: • A sprite • A poster on A3 or A2 size paper with a 6x5 grid • Instruction cards: Handout 1, Handout 7 (previous lessons) • Treasure chest cards: Handout 4, Handout 5 (previous lessons)
Glossary	Loop
Duration	30 min

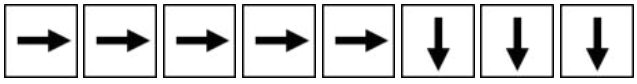
Starting the activity

The teacher shows the class a new route, which is bigger than the previous ones (six columns by five lines) and without any treasure chests.



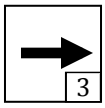
They ask the students to create a program that will take the sprite to the "destination" square. This simple exercise is very quick. Among the students' suggestions, the "simplest" ones are those using straight lines rather than routes with stair-stepping or detours.

Route example:



Although simple, this program requires numerous cards: the teacher asks the students how they might be able to shorten the program. If necessary, they can tell students that there are a lot of repetitions. Rather than using the same card several times, is there a way to show on the card that the instruction will be repeated several times? The class makes and discusses various suggestions.

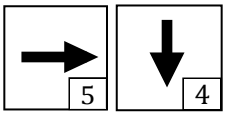
We suggest writing down on the card the number of times it will be applied. This type of notation has the advantage of being compatible with the design of the loops in the *Scratch Junior* software, which will be used in Level 2 (see page 142).



This card means “Move three squares to the right”

The teacher explains the key idea of *“loop,”* which is the repetition of a single instruction to simplify the writing (and understanding) of a program.

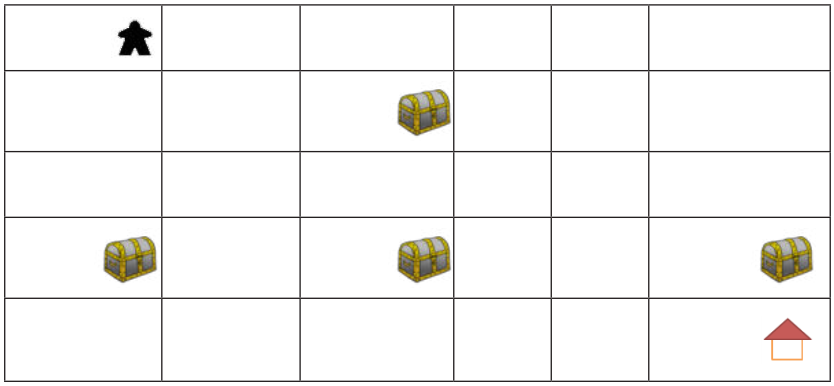
The class uses loops to simplify the previous program, which becomes:



The class then looks to see which of the previous programs they suggested can be simplified using loops.

Exercise and review

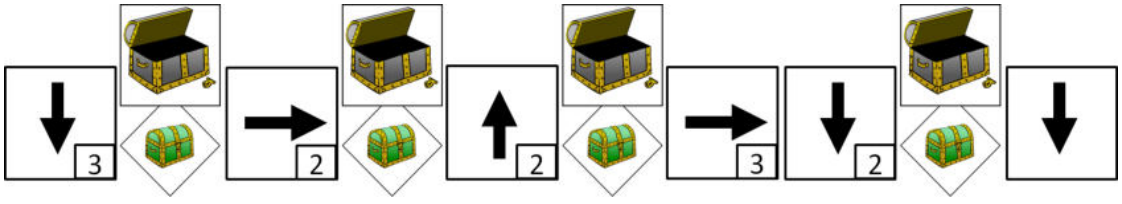
The teacher shows students the same route on which they have placed treasure chests:



The aim is to create a program to take the sprite to the destination square while collecting all the rewards (using tests, as in the previous lesson).

Depending on students’ ages, this exercise can be done in small groups or as a class.

One possible solution is:



Kindergarten class, Caroline Fayard (Paris).

Conclusion

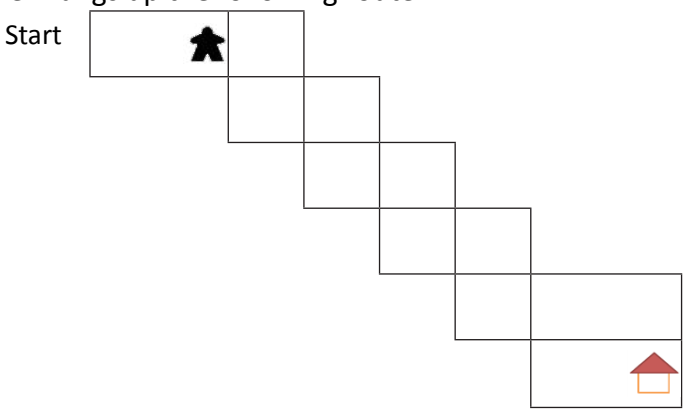
The class summarizes together what they learned in this lesson:

- *In a program, loops are used to repeat the same instruction several times.*

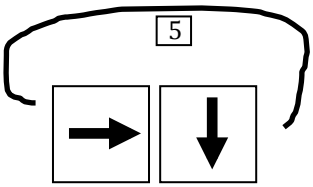
Further study (Level 2)

Older students can continue working on loops, especially to help them understand that several instructions can be included in a single loop. You can ask students to write a program using the fewest instruction cards possible to reach the destination, resulting in a route such as the one that follows.

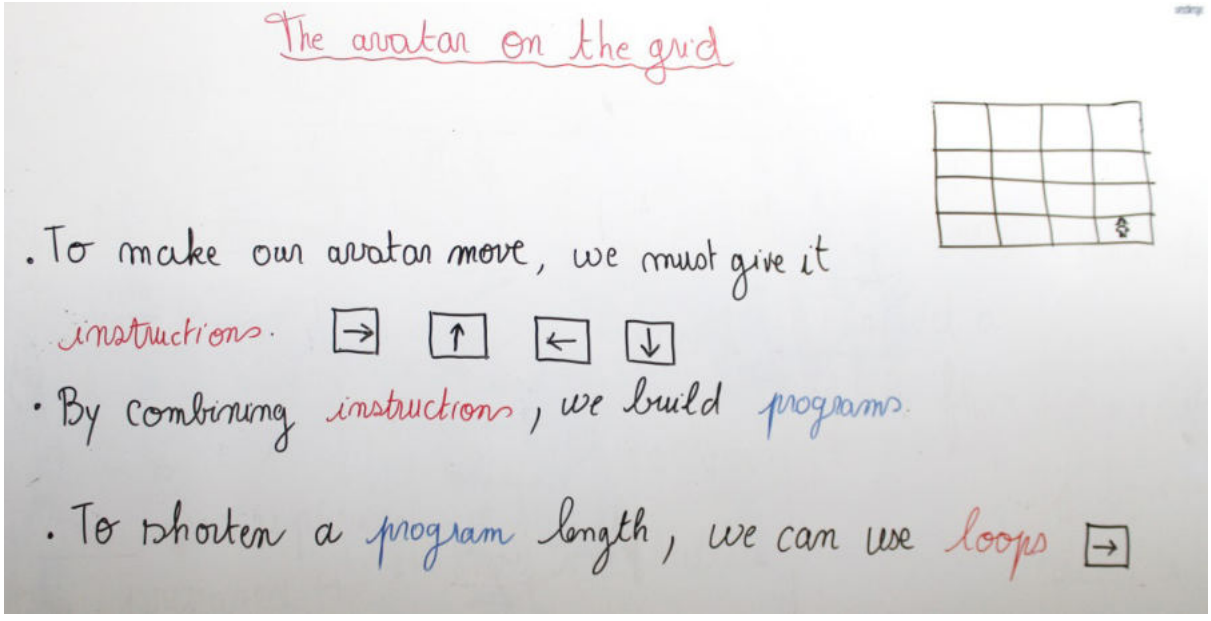
The teacher hangs up the following route:



The class begins by writing a program to describe the route. After learning about loops, the students try to apply this idea to their route. They need to figure out how to make a loop with two instruction cards. The teacher can suggest the following:



This notation is also compatible with the instructions used in the *Scratch Junior* language.



Kindergarten class, Caroline Fayard (Paris)

If the class has robots, it can continue with the next sequence. If not, wrap up this sequence with the Review lesson, page 96.

Sequence 2: Playing with robots

Introductory note: Robotics for preschoolers and kindergarteners

- Working with robots is extremely beneficial, not only to teach basic computer concepts (algorithms, machines, programs, etc.) and robotics (sensors, actuators, interactions with surroundings, etc.) but to help children develop cognitive and language skills as well. Additionally, manipulating a physical object is a strong motivator for students. However, despite the appeal, not all classes have access to robotics technology due to equipment costs (especially given that several robots are needed for each class).
- There are numerous educational robots, but few can be adapted to suit the needs of preschools, such as options that are affordable, sturdy and easy to use with a range of behaviors and interactions, etc. We have based our sequence on the Thymio 2 robot (which we simply call Thymio) as it has all these features. Of course, other options are available (more basic, such as Bee-Bot, or more sophisticated and expensive, such as Nao). If robots other than Thymio are used, the sequence will need to be adjusted accordingly.[10]

The class can then go to the Review lesson on page 96.

10 In 2016, a Thymio 2 robot cost about €150. We suggest working with at least two robots in class (and more if possible). A list of retailers that sell Thymio is available here: https://www.thymio.org/en:thymiobuy

Lesson 1 - Introduction to the Thymio robot

Summary	Students are introduced to the Thymio robot and learn how to manipulate it.
Key ideas *(see Conceptual scenario, page 56)*	"Robot" • A robot can perform actions: move, make a sound, produce light, etc.
Inquiry-based methods	Observation, experimentation
Equipment	For each group: • A Thymio robot, with its batteries charged For each student: • 2 sheets of A4 paper For the teacher: • Handout 8, page 85 • A2 size poster or flip chart
Glossary	Thymio
Duration	Two 30-minute time slots

Starting the activity

The teacher asks the entire class to explain what a "robot" is. To help them verbalize an answer, the teacher hands out a sheet of A4 paper to each student and tells them to draw a robot. After 15 minutes, students hang the drawings on the board and discuss them. The teacher also prepares the poster that will be used to summarize the robot characteristics.

Kindergarten class, Anna Halatchev (Paris)

The first observation is the general robot shape. The robots students imagine are nearly always humanoid and angular with lots of lights and buttons. They are often huge, move around on legs, wheels or track rollers, and can be grouped into two categories:

- Warrior robots: Armed with blades, guns, canons, crossbows and lasers, they destroy everything in their path.
- Utility robots: They clean, travel, dance, repair cars, cook, etc.

The teacher gradually fills in the poster: robot uses, means of locomotion, shapes, sizes, tools, etc. The poster will be used again at the end of the sequence to better define what a robot is.

Experiment: Discovering Thymio (in groups)

This second part of the lesson can be done right after the previous one or saved for another day depending on students' concentration levels.
The teacher splits the class into several groups and has them stand or sit around large flat surfaces (on the classroom floor or large tables, etc.). They give each group a robot (turned off). The teacher presents the Thymio robot and asks the students to explore it.
The teacher gives them a few minutes to familiarize themselves with the robot on their own. They will quickly figure out that it must be turned on to work (if they do not, ask them to press the middle button for three seconds) and that it can move around, make music and change color.

Group discussion

At the end of this activity, the students explain how they turned Thymio on. They also explain how, using the arrows on the top of the cover, they could make it change color and play music. They describe how they were able to turn it off.

Teaching notes

- Handout 8 (page 85) is for the teacher: It explains the commands, sensors and actuators for Thymio and the different operating modes.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *Thymio turns on using the middle button*
- *Thymio can change color*
- *Thymio can make sounds*

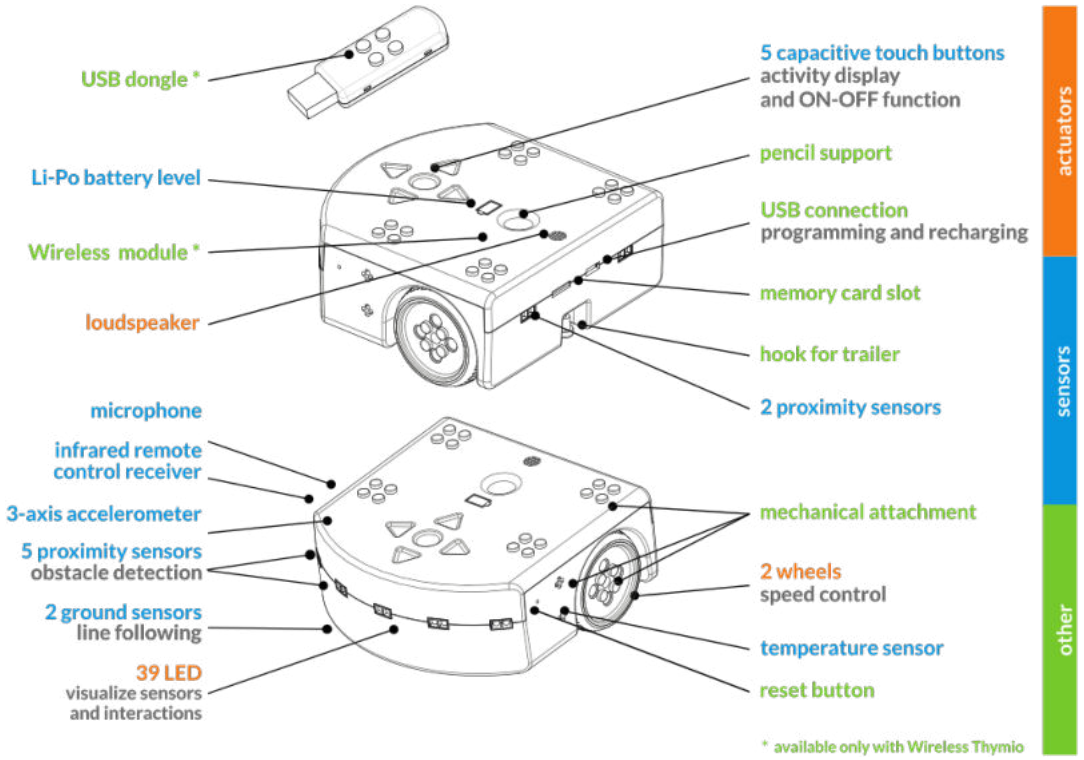
On a sheet of A4 paper, the students draw their Thymio.

Kindergarten class, Caroline Fayard (Paris)

Teaching notes

- The drawing of Thymio may be the first observational drawing students have ever done. Some may find it difficult to decide which angle to start drawing from, while others may start drawing right away. Some of the drawings may be very detailed. Drawing helps students learn to observe things carefully: *How are the buttons positioned? How can the shape be described in words?* and so on. This exercise has students use a number of skills.

HANDOUT 8
About Thymio



To turn Thymio on, touch the round button that lies in the center of the four arrow buttons until the robot makes a sound and turns green. It takes just a few seconds.

To turn Thymio off, proceed as for switching it on. Touch the round button until the robot no longer shows any color. It will also make a sound.
(Source: https://www.thymio.org/en:thymiostarting)

Thymio is pre-programmed with six behaviors. These behaviors are always available on the robot. To choose Thymio's behavior, simply start the robot and select a color using the arrow buttons, then press the middle button to confirm. When the behavior is active, the middle button lets you go back to the behavior selection menu.

Mode	Color	Behavior
Friendly	Green	Thymio follows obstacles that move in front of it.
Explorer	Yellow	Thymio randomly explores its surroundings and avoids obstacles.
Fearful	Red	Thymio avoids obstacles in front of or behind it.
Investigator	Turquoise	Thymio follows a dark track against a light background drawn on the floor.
Obedient	Purple	Thymio is manually oriented using the arrows on the cover.
Attentive	Blue	Thymio reacts to sounds: depending on the number of hand claps it hears, it can turn, move forward, stop, move in a circle.

Lesson 2 - Colors and behaviors

Summary	Students learn that Thymio has several modes and can behave differently depending on the chosen mode.
Key ideas *(see Conceptual scenario, page 56)*	"Robot" • A robot can perform actions: move, make a sound, produce light, etc. • A robot has sensors that let it perceive its surroundings.
Inquiry-based methods	Observation, experimentation
Equipment	For each group: • A Thymio, with its batteries charged. For each student: • The Thymio drawing from the previous lesson. For the teacher: • Handout 8, page 85 (used in the previous lesson) • An A3 or A2 size flip chart.
Glossary	Sensor, wheels
Duration	30 min

Preparation

Just before the lesson, the teacher turns the Thymio robots on and sets them to different modes (green, yellow, red and purple). Note that it is best to select the yellow mode at the last minute, otherwise it will move around the table on its own.

Starting the activity

Each group tries to understand how Thymio is behaving when displaying a particular color. The teacher starts off the experiment by turning on the Thymio robots that are to be in yellow mode.

Experiment: Which behaviors correspond to which colors? (in groups)

Aside from the yellow mode, the other modes do not immediately start Thymio moving. If the students do not think of it on their own, suggest they place obstacles near the robot (a hand, an object, etc.).
When the Thymio robots begin to move (green and red modes), ask the students to figure out which part of its body allows the robot to detect obstacles and have them identify the distance sensors. They can make the connection between the robot's actions and the sensor indicators that light up. For example, in green mode, if a sensor detects an object, the indicator light turns

red and Thymio begins following the object. The teacher can then officially introduce the term "sensor" to talk about these components.

The purple mode will likely be the most difficult to understand. The teacher can tell the students that the on/off button is also a sensor. The arrows could also be considered sensors.

Teaching notes

- For a smoother observation process, a few rules should be established from the start:
  - One student handles Thymio at a time
  - After each manipulation, leave a few minutes to observe and understand the effects
  - Leave some space around Thymio so it can move (students quickly tend to sit very closely around it, which overstimulates the sensors and does not give it enough space to move around)

Kindergarten class, Caroline Fayard (Paris)

Group discussion

Each group tells the rest of the class about its robot and explains its behavior by showing which Thymio sensors interacted with its surroundings (obstacle detection or pushing buttons):

- Yellow Thymio "moves on its own" by "avoiding obstacles."
- Green Thymio tends to follow objects, like a hand, placed in front of it.
- Red Thymio moves away from objects placed in front of, behind or beside it.
- Purple Thymio moves forward or turns depending on the arrows that are pressed.

As a group, try to give each behavior a name (e.g., *friendly, fearful, explorer*, *obedient*). The teacher concludes the group discussion by asking how Thymio moves. The students will quickly point out the wheels.

Scientific notes:

- Sensors are components that let a robot perceive its surroundings (including a person's actions).
- The actuators are components that let the robot interact with its surroundings (here, by moving).

Exercise: Playing with the other modes

The students swap robots to explore the other modes.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *Thymio can be in different modes, each indicated by a different color, which determine the robot's behavior.*

On their sheet of A4 paper, the students complete their drawing of Thymio, labeling the sensors and wheels.

On the board, the teacher describes the four initial modes, labeling each with a color, the adjective used to describe the mode (and/or an icon chosen by the class to designate the behavior, such as smileys).

A fifth line should be prepared ahead of time to describe the turquoise mode, which will be covered in the next lesson. This experiment will require some advanced preparation by the teacher, who will likely want to prepare the routes for Thymio to follow (see next page).

Although Handout 8 describes six behaviors, only five of them will be explored by the students. We do not recommend working in blue mode. Because Thymio reacts to sounds, the classroom can quickly become chaotic.

Lesson 3 - Thymio in Investigator mode

Summary	Students discover Thymio's turquoise mode and prepare a route that Thymio can follow alone.
Key ideas *(see Conceptual scenario, page 56)*	"Robot" • A robot can perform actions: move, make a sound, produce light, etc. • A robot has sensors that let it perceive its surroundings.
Inquiry-based methods	Observation, experimentation
Equipment	For each group: • A Thymio, with its batteries charged • Large sheets of white drawing paper, black paint, small paint rollers (4 cm wide) For the teacher: • Handout 8, page 85 (used in Lesson 2.1) • The A3 poster created during the previous lesson
Glossary	Sensor, routes
Duration	30 min

Foreword

There are two different ways to approach this lesson.

- The first (described here) is for the teacher to prepare a route that Thymio will be able to follow. In half an hour, the students can both explore the turquoise mode and quickly draw conclusions about the concept of "sensor" while describing the types of routes that work well.
- A second option (the lesson variation) will take longer (one to two hours, done partly during art class) because students will have to learn about the sensors and explore how sensitive they are by creating routes in various materials that work well (or not). They will analyze the shape as well as the color and materials of the routes Thymio sees.

Preparation

The day before the lesson, the teacher can prepare the black route sections (straight, curved, etc.) using a paint roller and poster or acrylic paint. The route should be around 4 cm wide. Be sure to try it out ahead of time with the robot to make sure it works! Just before the lesson, the teacher turns on the Thymio robots and places them in turquoise mode.

Starting the activity

Each group observes that Thymio turns around and around when placed on the table. The teacher explains that Thymio is looking for a route and the students are going to give it one.

Experiment: Drawing routes for turquoise Thymio (in groups)

Students will glue route sections provided by the teacher onto their pieces of drawing paper. The routes can be straight, curved, open or closed. A track in a figure-8 shape is simple and provides an interesting experience.
When the route is ready, students can place their Thymio (still in turquoise mode) on their paper near the route. They will notice that the robot follows the route all on its own.

Kindergarten class, Caroline Fayard, (Paris). Note that rough textured paper is not ideal for this type of route (see the list of suitable materials below).

Group discussion

Each group shows the class its route and describes how Thymio followed it. The teacher writes down the features of the routes that worked well on the board:

- Continuous routes (at each break between two segments, Thymio turns back the way it came)
- Gentle curves (Thymio has trouble managing hairpin turns)

Whether the route is open or closed or features intersections, Thymio is able to follow it: It either turns around at the end of an open route, continues along closed routes or generally goes straight at an intersection.

The teacher mentions the term "sensor" again so that students understand how the robot was able to "see" the route. When students lift their Thymio up, they can see two sensors under the frame at the front of the robot.

The teacher asks students how they know these are the sensors that let Thymio "see" the route. The class comes to a consensus with a short experiment: cover the sensors with a piece of paper taped to the underside of the robot. Now, Thymio cannot "see" any route, which confirms the initial hypothesis.

Scientific notes:

- The two sensors on the robot's frame detect the presence or lack of a route (black or another dark color versus white or another light color). For example, if the right sensor detects white but the left sensor detects black, Thymio will turn left to follow the route, which is certain to be a left turn. If both sensors detect white, Thymio turns around in a circle until it finds a route. If both sensors detect black, Thymio moves straight forward, which will also happen if both sensors are blinded by sticky tack.

As a class, students describe this behavior with an adjective (e.g., *investigator*, because it investigates a route, older children may suggest *explorer*). Avoid using the term *follower*, because the green *friendly* mode can follow a hand placed in front of it.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *Turquoise Thymio can follow routes drawn in black on a white background.*

On the board, the teacher completes the poster from the previous lesson, adding a description of the fifth mode and labeling it with the color, the adjective used to describe it (and/or an icon chosen by the class to designate the behavior, such as a smiley).

Further study

- Kindergarten students may want to try out other routes with other shapes to see how turquoise Thymio reacts to more complex routes.

Variation

In this variation, the teacher lets the students create their own routes. This takes time (i.e., letting the paint dry) and results can vary. Here are a few things to keep in mind with regards to the materials and shapes children can use:

- Materials:
  - What works: black Bristol board, ink, acrylic, poster paint, garbage bags
  - What does not work: textured paper, tissue paper, felt
- Shapes that work:
  - Continuous routes
  - Gentle turns
  - A smooth surface (Thymio's movement can be hampered by irregularities in texture, glue blobs, folds, etc.)

The routes on the right (in poster/acrylic paint) worked well, but Thymio was unable to detect the one on the left (textured paper). The route in pencil (graphite, in the middle) is being tested. Kindergarten class, Caroline Fayard (Paris).

After trying out different materials, kindergarten or Level 2 students can extend their study by creating a black route on a black background. A route on Bristol board will be seen by Thymio even if glued onto a black textured background.

Lesson 4 - Challenge: Get Thymio through a maze

Summary	Students build a maze and must find all possible ways to get Thymio through it.
Key ideas *(see Conceptual scenario, page 56)*	"Robot" • A robot is a machine that can interact with its surroundings. • A robot can perform actions: move, make a sound, produce light, etc. • A robot has sensors that let it perceive its surroundings.
Inquiry-based methods	Observation, experimentation
Equipment	For each group: • A Thymio, with its batteries charged For the class: • Large sheets of white drawing paper, black paint, small paint rollers (4 cm wide) • Black fabric ribbons (4 cm wide): These should be tested as results can vary depending on the fabric (cotton, silk and wool work) • Objects that can be easily moved around and used as obstacles for Thymio (cubes, books, etc.) For the teacher: • Handout 8, page 85 (used in Lesson 2.1) • The poster created during the previous lessons • A stopwatch
Glossary	Sensor, wheel, route, maze
Duration	30 min

Preparation

Before the lesson, the teacher sets up a maze using cubes, books, etc. on the classroom floor. Obstacles should be between 5 and 6 cm high to be detected properly by the side sensors and heavy enough that Thymio will not move them by accident if it runs into them. The path should be around 20 cm wide and the turns not too sharp. The maze can be open (with an entrance and an exit) or closed. The teacher places the Thymio robots (turned off) on the tables for each group.

Starting the activity

The teacher presents the maze to the class. Their challenge is to have Thymio run the maze. (For open mazes, place Thymio at the entrance and have it go out the exit; for closed mazes, have the robot do a complete round through the maze.) Each group can suggest a method and apply it to see if their strategy works.

Experiment: Guiding Thymio through a maze (in groups)

If necessary, the teacher can bring out the poster and describe the five behaviors seen during the previous lessons:

- Can you use the green mode? What do you do? (Yes, you can: You have to guide Thymio step by step around the maze, using your hand or an object placed just in front of the robot.)
- Can you use the red mode? (Same as above, but you have to push it)
- Can you use the yellow mode? (Yes, the robot explores and avoids hitting the maze walls to eventually reach the exit.)
- Can you use the purple mode? (Yes, you have to guide it step by step using the forward, right and left buttons, but you have to be quick and precise to do it correctly.)
- Can you use the turquoise mode? (Yes, you have to draw a black route or place a black ribbon inside the maze.)

Teaching notes

- You could also use the blue mode, but since Thymio reacts to sound in this mode, the exercise can be very noisy!

If all the groups come up with the same ideas, the teacher can use the above questions to suggest that certain groups try other possibilities to cover all five options.

Students prepare a route for turquoise Thymio. Kindergarten class, Caroline Fayard (Paris)

Group discussion

Each group tries out its solution. One possibility is for the teacher to time how long it takes Thymio to run the maze.

After doing at least five tests with all five modes, the class will notice that Thymio is able to get through the maze more or less by itself.
If the teacher timed the different exercises, the class can create a podium, ranking the modes from fastest to slowest. If they were not timed, students can vote on the mode they found to be the fastest, easiest, most entertaining, most leisurely, etc.

Conclusion

The class summarizes together what they learned in this lesson:

- *Thymio can always get out of a maze, either alone or with the help of a person.*

Further study

Kindergarten students may want to try other mazes and shapes to see how Thymio reacts in more complex environments. This will give them a chance to handle Thymio more as to create their own mazes.

Review: What is a robot?

Summary	Whether they worked with an «unplugged» sprite or a Thymio robot, students learn to define what a robot is: a machine that can interact with its surroundings.

Key ideas
(see Conceptual scenario, page 56)
	"Robot"
	• A robot is a machine that can interact with its surroundings.
	• A robot has sensors that let it perceive its surroundings.
	• A robot can perform actions: move, make a sound, produce light, etc.
	• A robot has a computer that decides which actions to take in which situations.
	• If you compare a robot to an animal, you can say that:
	  ○ Its sensors are like sensory organs
	  ○ Its motors are like muscles
	  ○ Its computer is like a brain
	  ○ The parts taken together are like a body

Inquiry-based methods	Observation, discussion

Equipment	For classes that have done Sequence 2
	• For the teacher:
	  ○ A Thymio robot
	  ○ A screwdriver
	  ○ Handout 9, page 101
	• For students:
	  ○ The drawing of Thymio created at the start of Sequence 2 (Lesson 2.1 and 2.2)
	
	For classes that have done Sequence 1, Sequence 2, or both
	• For students:
	  ○ Handout 10
	  ○ Handout 11
	• For the class:
	  ○ Poster(s) created at the start of the sequence

Glossary	Sensor, motor, computer, robot
Duration	45 minutes, split into two sessions

Foreword

This review lesson can be done at the end of Sequence 1, during which students will have done unplugged activities, or at the end of Sequence 2, during which students will have handled robots (here, Thymio).
This lesson is divided into three parts:

- A transition for classes having just completed Sequence 1
- A transition for classes having just completed Sequence 2
- A section applicable to both to help the class fully define what a robot is.

Teachers who have taught both sequences may choose to do only one of the two transitions, most likely the one related to the sequence they most recently finished. In any event, it provides an opportunity to re-hang all of the posters created during the lessons.

Transition for classes having just completed Sequence 1

Starting the activity

Students created programs that made it possible to guide the sprite (when the route is known ahead of time) and have it collect rewards. The teacher asks the students to discuss the problem of an unknown route, i.e., the maze. If you want the sprite to be able to get out of the maze alone, without telling it step by step what to do or where to go, what do you do?

Discussion: sensors, motors, programs

Covering the idea of "test" again, the students can consider conditional constructs such as *"IF there is an obstacle in front of a sprite, THEN the sprite turns right."* This is a good approach.

The next question to ask is *"How does the sprite know when there is an obstacle?"* If the students have a hard time answering, help them by making a connection with living beings: *What would a dog do to get out of a maze? A person?* Students will bring up the senses, such as sight, smell, touch, etc., which leads to the key idea of *"sensor."* You can mention auditory, optic, olfactory and tactile sensors and the like. The test that students did (*"Is the chest green?"*) requires an optical sensor. Obstacle detection happens through sensors.

Group discussion

The class summarizes what they have covered: with sensors, the sprite can observe its surroundings. With a program, you can tell the sprite what to do depending on the circumstances.

The teacher adds that to move on its own, the sprite would need muscles and feet, legs, etc. In mechanics, we would say it needed *"motors."*

Teaching notes

- With older students, the need for motors can also be a discussion topic.

In this case, the final question will be: *What do we call objects that have sensors, motors and programs?*

Transition for classes having just completed Sequence 2

Starting the activity

The teacher shows the class a Thymio (turned off). They ask them to imagine what might be inside. The students should be able to repeat the terms "robot," "sensors," "motors," and "wheels" that were already covered. If they have a hard time coming up with ideas, the teacher can guide them using questions such as:

- *"What makes Thymio's wheels turn?"*
- *"How does Thymio get energy to move or turn on its lights?"*
- *"Do we need to fill it up with gas or feed it?"*
- *"How does it decide which direction to go in when it detects an obstacle?"*

Observation: What is inside a Thymio robot? (entire class)

The teacher tells the students that certain parts of the Thymio robot can be taken apart to see what is inside. After removing several screws, the teacher can show the students the robot's electronic components. Because the robot is fragile in this state, it is best that the teacher alone handles it.

The teacher points out and names the various parts:

- The sensors and the red lights that light up automatically when the sensor detects something.
- The electric wires that link the sensors to small black boxes (microprocessors) that act as a computer for Thymio: they are what let it decide what it must do when the sensors detect something.
- The "ambiance" lights that turn Thymio a different color based on its mode.
- The two motors, connected to wheels, that obey orders from the microprocessors.
- The battery that gives Thymio energy and that can be recharged.

Teaching notes

- Students will likely not understand the importance of microprocessors and/or the program if they have not completed Sequence 1. They will understand why the wheels are important for moving around and the sensors for detecting obstacles, but the interpretation and decision-making aspects will be unclear. To help them, have a student pretend to be a robot – i.e., obey without asking questions. Tell them to walk straight. The student will walk straight towards the back wall and will begin to worry if they do not receive another order. Rather than run into the wall, they will stop on their own. The teacher can then ask why they disobeyed. Their eyes saw the wall, and their brain told their legs to stop to not get hurt. The computer is the robot's brain.

The teacher then asks the students to explain what Thymio is.

Review: What is a robot?

Documentary study

Next (or during a second time slot), the teacher passes out Handout 10 and Handout 11 and asks the students to sort the objects, without telling them how many categories to create. It is possible that the students will instinctively separate the humanoid robots into one category and the non-humanoid robots into another, but they may also do it by shape or color.

Once they have done an initial sorting, the teacher adds the robot drawings from the first lesson. The class concludes that all these objects are part of the same broad category, "robots." Despite their different shapes, they all have sensors, motors and computers. While they do not look alike (and may not necessarily look like humanoids), they all work in a similar way. Our definition of a "robot" is: *A machine with sensors, motors and a computer, which can perceive its surroundings and take action accordingly.*

Teaching notes

- You can explain to students that we project our preconceived ideas about robots onto them. When we see a humanoid robot, we think it will be "smart" because its shape resembles a person. But in reality, they are often not any more sophisticated than a robotic vacuum cleaner.

Scientific notes:

- What is the difference between an automated machine and a robot? The question can be asked when we, as adults, consider certain machines (such as a machine tool). Originally, they were programmed to reproduce a movement; however, this did not make them robots but rather automated machines. An automated machine is programmed to always repeat the same movement ("bend the arm to 45°," "descend the drill," "drill down 5 cm," "straighten the arm for 45 seconds"), but does not have sensors. If the joint is blocked, the machine will still try to perform the other actions; if the drill is no longer fitted with a drill bit, it will drill empty space, etc. Mechanical arms used today are robots: a pressure sensor confirms that it is in contact with the plate to drill, a gauge tells it if there is enough oil in its joints, an actuator confirms the joint rotation angle, and a program tells it how to adapt or stop if a parameter changes. Many current technological devices are robots. If your toaster knows when to eject your toast before the bread burns, that means it is equipped with sensors and a program.

- Below are the special features of the robots on Handout 10 and Handout 11:
    - Mechanical arms have sensors to control their gestures and consumable levels.
    - Baxter is equipped with shape recognition to know which objects to pick up from a conveyor belt.
    - BigDog adapts the way it walks to the terrain to be able to continue moving forward despite obstacles in the way.
    - When in a group, Eporo robots imitate schools of fish to travel together, without causing traffic jams or accidents.
    - Robots can be used to help scientists explore locomotion mechanisms, such as the Harvard Ambulatory Micro-Robot, which walks on several legs (it is available in a centipede-inspired version), the Honda P2 for bipedal walking, RoboBee for flight or the G9 robotic fish for swimming.
    - Han explores emotion recognition and reproduction via subtle facial movements.
    - Roomba is a vacuum cleaner that moves around the room on its own and goes back to its docking station when the batteries run down. It closely resembles Thymio's yellow mode.
    - The Sojourner rover is one of many robotic solar system explorers (the first was Lunokhod 1, sent to the moon in 1970).

Group discussion

To reinforce this key idea, the teacher can compare robots to animals:

- Its sensors are like sensory organs
- Its motors are like muscles
- Its computer is like a brain
- The parts taken together are like a body

Conclusion and lesson recapitulation activity

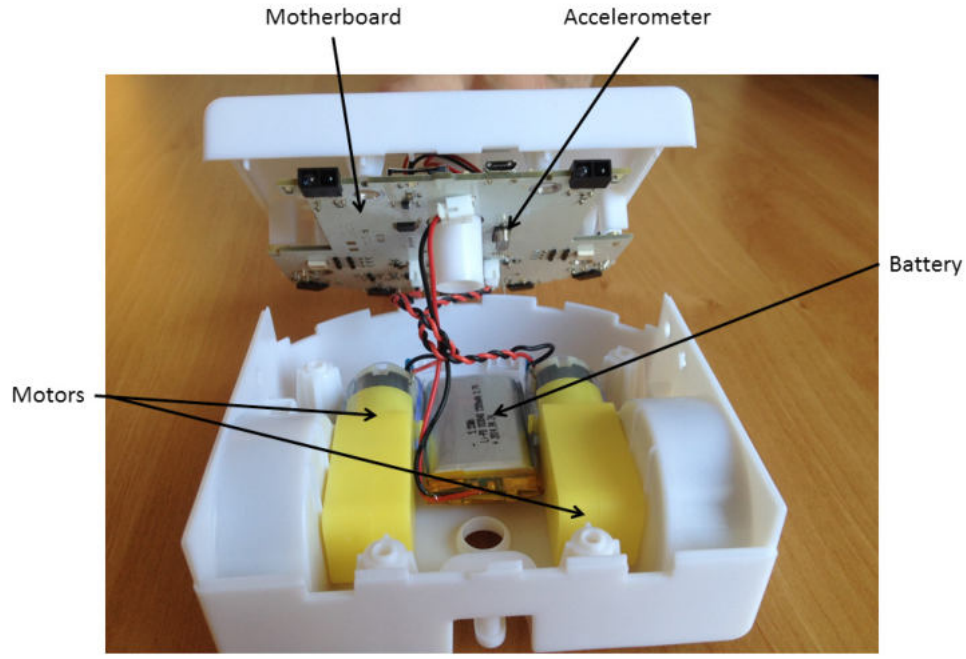
The class summarizes together what they learned in this lesson:

- *A robot has a computer, sensors and actuators that are all interconnected.*

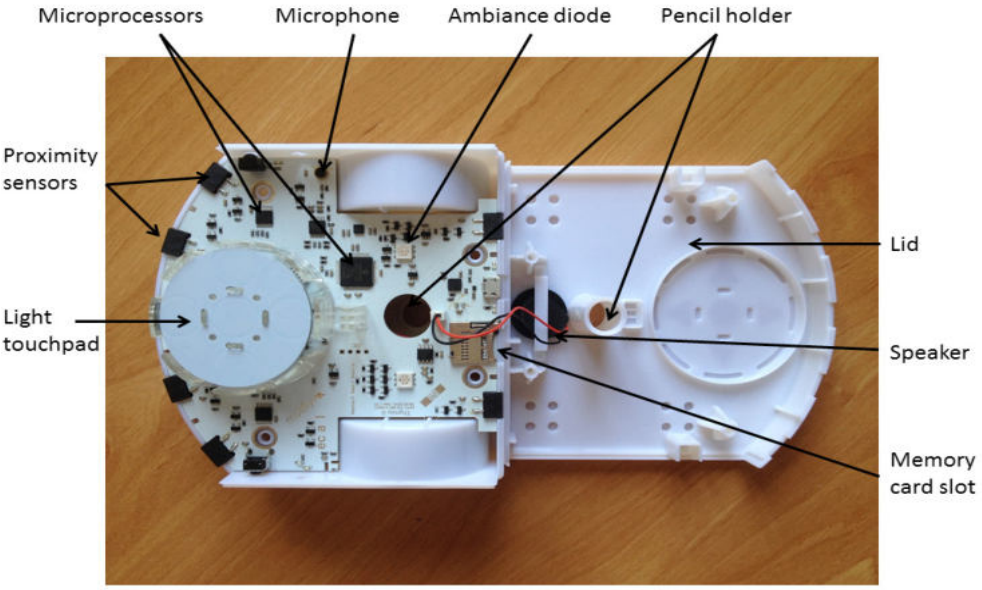
Further study

- Have students draw more robots. Some will draw humanoid androids again, while others will draw from science fiction. See how many draw cubic robots this time.
- Suggest a “philosophical” workshop on this question: Are machines smart?

HANDOUT 9
Thymio dissection

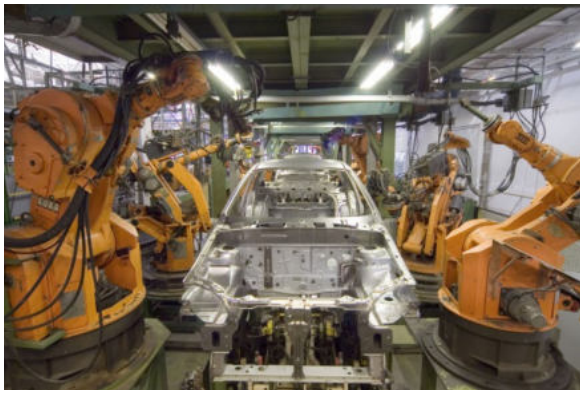


The chassis: the battery (in the middle) feeds the two motors (yellow) that turn the wheels



The motherboard, which holds the infrared sensors, central light touchpad, microprocessors and diodes

HANDOUT 10
What do these things have in common? (1/2)

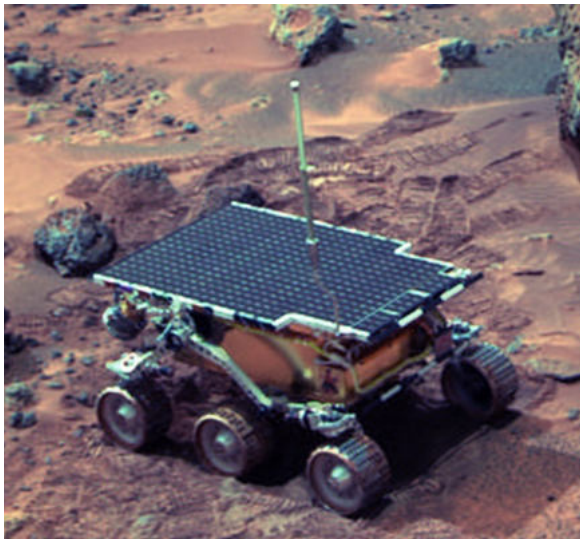


Mechanical arms

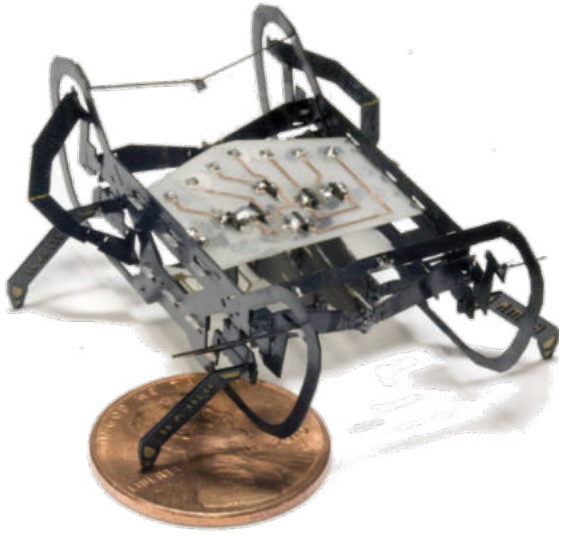
Roomba©

Han

Eporo©



Sojourner©

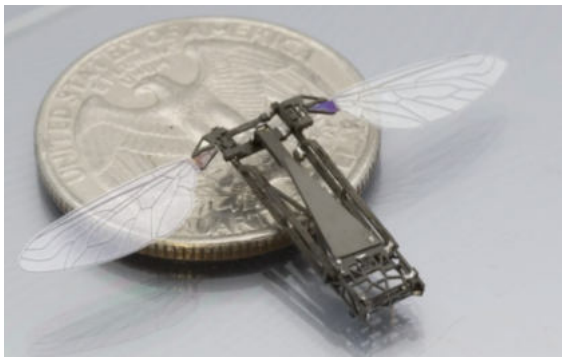


HAMR (© Harvard University)

HANDOUT 11
What do these things have in common? (2/2)

Baxter©

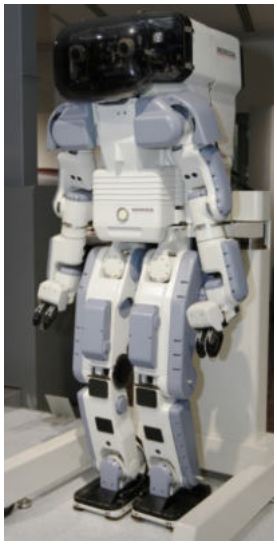
G9 fish



RoboBee©

Thymio

BigDog©



Honda P2©

Level 2 Activities

Overview

The theme that ties the Level 2 activities together is an adventure.

We suggest splitting the activities into two sequences:

- The first is entirely unplugged (done without a computer but with experimentation and documentary equipment). This sequence sets the stage for an adventure during which a hero must successfully overcome various challenges before he can return home. The students will learn about the key ideas of algorithm, language, and data representation (texts and images).
- The second is entirely plugged (requiring a tablet, computer or robot). This second sequence can be any of the following:
    - Sequence 2 involves programming on a tablet (using Scratch Junior). This sequence (which we recommend as the default option) tells the hero's adventure through animation. The students will be introduced to programming by using a graphic environment designed for young children.
    - Sequence 2a is a variation of Sequence 2, for classes without tablets but which do have computers (using Scratch rather than Scratch Junior). It should be noted that if the school is equipped to choose between the above two options, we strongly suggest opting for Sequence 2 (Scratch Junior), which is both better suited to this age group and easier to use.
    - Sequence 3 has students program a robot (Thymio) using the same concepts as programming a computer or tablet, but applied to a physical object (the robot). Please note: contrary to Sequence 2 or 2a, in this sequence, programming the robot is completely unrelated to the adventure told in Sequence 1.

Lesson summary

Sequence 1: The adventure

	Lesson	Title	Page	Summary
	Lesson 1	The hero's journey	110	The hero awakes in an unknown world in the great outdoors. A journey awaits where he must travel down the mountain he finds himself on. Students must guide him with conditional constructs.
	Lesson 2	Decoding a message	115	At the end of this perilous journey, the hero must solve a riddle carved on a tree trunk. Students understand that it is a coded message. To help the hero, they must decode the message to understand its meaning.
	Lesson 3	Programming a route	119	The hero cannot reach the treasure at the bottom of the sea, but he finds a small submarine. The students must invent a language to pilot it remotely.
	Lesson 4	Summoning the magician	123	The hero must summon a magician by asking for help from the birds. To do this, he must create a drawing on the ground using white and black rocks. The students learn how to pixelate an image in black and white.
	Lesson 5	(Optional) Following a recipe	129	Thanks to the magician, the hero will be able to create the magic recipe. The students must analyze the structure of the recipe to find the elements for an algorithm.
	Lesson 6	(Optional) Building a magic key	134	The hero can return home. Before he leaves, the magician gives him a magic key that will let him come back. The students must describe the algorithm that will let him duplicate this key.

Sequence 2: Telling the adventure with Scratch Junior

	Lesson	Title	Page	Summary
	Lesson 1	Getting started with *Scratch Junior*	139	The students are introduced to *Scratch Junior*, an easy-to-use graphic programming environment for children ages 5 to 8. They explore the ways to control a character's movements.
	Lesson 2	The first episode: Choosing the hero and controlling his movements	146	Students tell an episode of their hero's adventure. While they do so, they learn the new functionalities of *Scratch Junior* (deleting a character, importing a new character, choosing a setting) and are exposed to the key ideas from the previous lessons (set of instructions, event).
	Lesson 3	Simplifying a program by using loops	149	The students continue learning to use *Scratch Junior* by exploring the instruction «repeat...,» which is a loop. They practice anticipating what a program given to them will do, combining loops and movement instructions. Finally, they revise their initial program by replacing the repeated instructions with loops.
	Lesson 4	Coordinating several scripts	153	Students tell a new episode of their hero's adventure, with more autonomy than in the first lessons. They discover new functionalities in *Scratch Junior* and deepen their understanding of what a set of instructions and a program are.
	Lesson 5	Predefined loops and infinite loops	157	Students tell a new episode of their hero's adventure. They reinforce the key ideas from the previous lessons, namely predefined loops, and learn about infinite loops.

	Lesson 6	Adding recorded dialogues to the program	160	Students learn to record character dialogues.
	Lesson 7	Producing the final episode autonomously	162	Students work on their own to tell the last episode of their hero's adventure. They cover the key ideas from the entire sequence and finish their program.

Sequence 2a: Alternative with Scratch

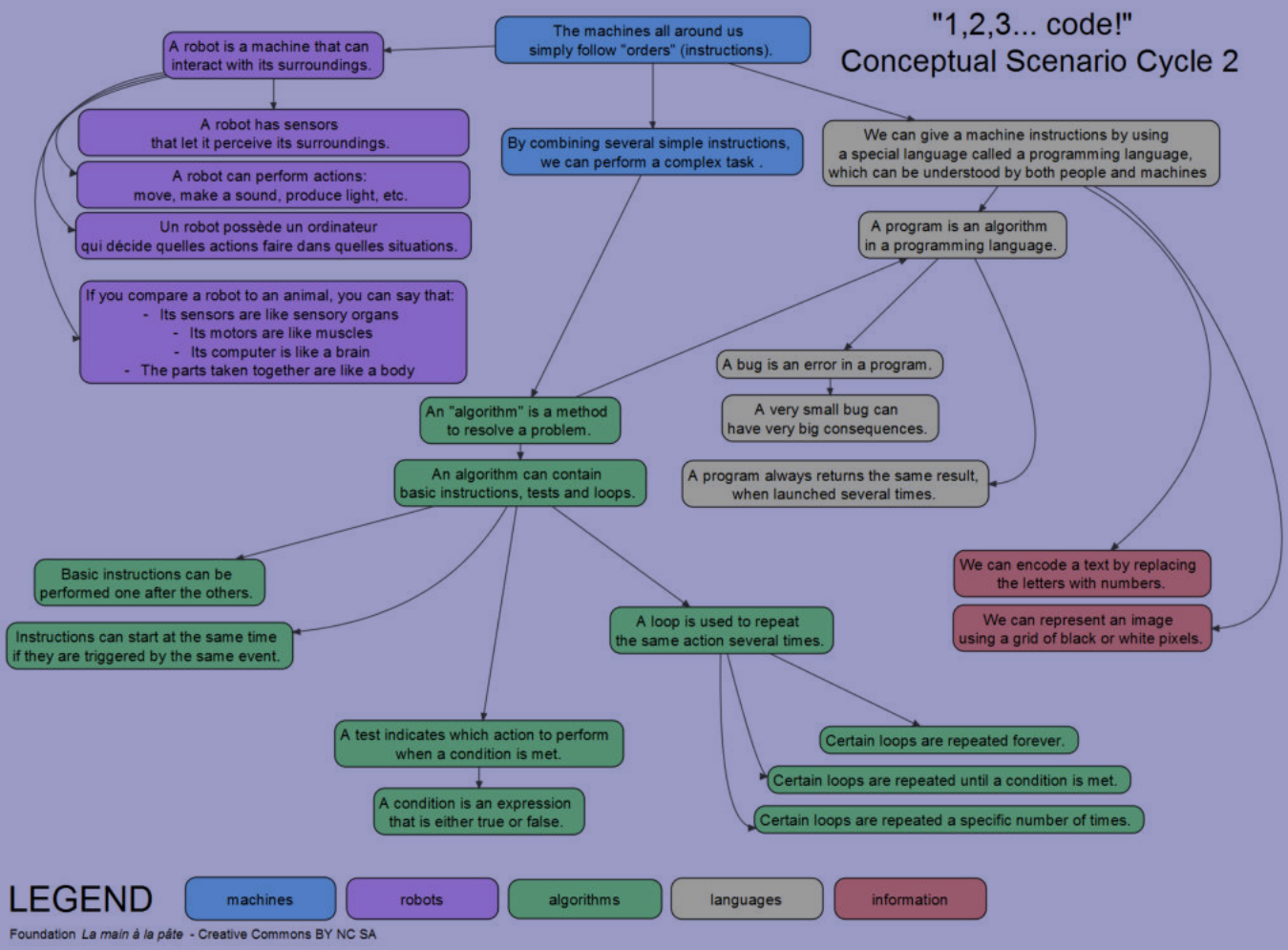
	Lesson	Title	Page	Summary
	Lesson 1	Introduction to the *Scratch* programming environment	166	Students are introduced to *Scratch*, an easy-to-use graphic programming environment.
	Lesson 2	Making a character move	168	Students explore the ways to control a character's movements.
	Lesson 3	Choosing the hero and controlling his movements	170	Students tell the first episode of their hero's adventure, where he comes out of the forest and follows the river to the sea. During this time, they cover the key ideas from the previous lesson (set of instructions, event), learn about the idea of initialization and use predefined loops («repeat…»).
	Lesson 4	Programming several sprites	174	The students tell another episode of the hero's adventure, where he sees the treasure at the bottom of the sea and gets help to retrieve it. To do this, students learn to load a new stage, add a sprite and cover the key programming ideas from the first two lessons.
	Lesson 5	Coordinating the first two episodes	177	Students must figure out how to make the two first episodes continue one after the other. To do this, they learn the key idea of message: a message can be sent during an instruction, and when the message is received, it can trigger one or more instructions.
	Lesson 6	Different types of loops	179	Students tell the next episode of the hero's adventure: the octopus goes to the bottom of the sea to get the treasure and bring it back to the surface. They reinforce the key ideas from the previous lessons, namely predefined loops, and learn about infinite loops.
	Lesson 7	Producing the final episode autonomously	181	Students work on their own to tell the last episode of their hero's adventure. They cover the key ideas from the entire sequence and finish their program.

Sequence 3: Robotics with Thymio

	Lesson	Title	Page	Summary
	Lessons 1, 2, 3	Introduction to Thymio in Level 2	183	Students are introduced to the Thymio robot and familiarize themselves with it. After exploring the various pre-programmed modes, they have Thymio run a maze. They gradually formulate a simple definition of what a robot is. (Adaptation of the four first lessons of the «Robotics in Level 1» sequence, pages 82 and on)
	Lesson 4	Programming Thymio (1/2)	188	To go into more depth with Thymio, students discover the Aseba/VPL programming environment. The graphic interface lets them design their own programs for Thymio.
	Lesson 5	Understanding sensors to program Thymio	193	VPL programming for Thymio is event-driven: students will learn how to use Thymio's sensor status to trigger precise actions.
	Lesson 6	Programming Thymio (2/2)	196	Students take on small challenges to create their own VPL programs for Thymio.
	Lessons 7 and 8	Obstacle course for Thymio	198	Students must reproduce Thymio's yellow «explorer» mode. First, they write the program. Then, they test their program in a real maze.

Conceptual scenario: Level 2 computer science

The key ideas covered during these four sequences for Level 2 can be organized as follows.



Sequence 1: The adventure

Lesson 1 - The hero's journey

Summary	The hero awakes in an unknown world in the great outdoors. A journey awaits where he must travel down the mountain he finds himself on. Students must guide him with conditional constructs.
Key ideas *(see Conceptual scenario, page 108)*	"Algorithm" • An «algorithm» is a method to resolve a problem. • A test indicates which action to perform when a condition is met. • A condition is an expression that is either true or false.
Inquiry-based methods	Observation, experimentation
Equipment	For each student • Handout 12 • Handout 13 (for second and third grades only) For the class • A video projector or A3 size printout (or poster) of Handout 12
Glossary	Conditions, tests
Duration	1 hour

Foreword

The teacher explains to the students that in the following lessons, they will follow the adventures of a hero or heroine and must help them solve riddles to be able to return home (for simplicity's sake, we will just say "hero" from here on).

Starting the activity

When he wakes up, a hero finds himself at the top of a mountain. He has no idea how he got there and does not recognize the forest or valley below. He does not recognize the birds, either. He is far from home. He sees a clearing below and decides to go there.

Create instructions using conditional constructs (in groups or as a class)

The teacher passes out Handout 12 and projects the route the hero must follow to reach the clearing at the foot of the mountain. To help him, the students must create a set of instructions for the hero to follow exactly to get there safe and sound. The instructions must be IF–THEN statements. For example:

IF the hero sees a cliff, THEN he must climb down it.

Depending on students' ages, this activity can be done orally as a class or in small groups.

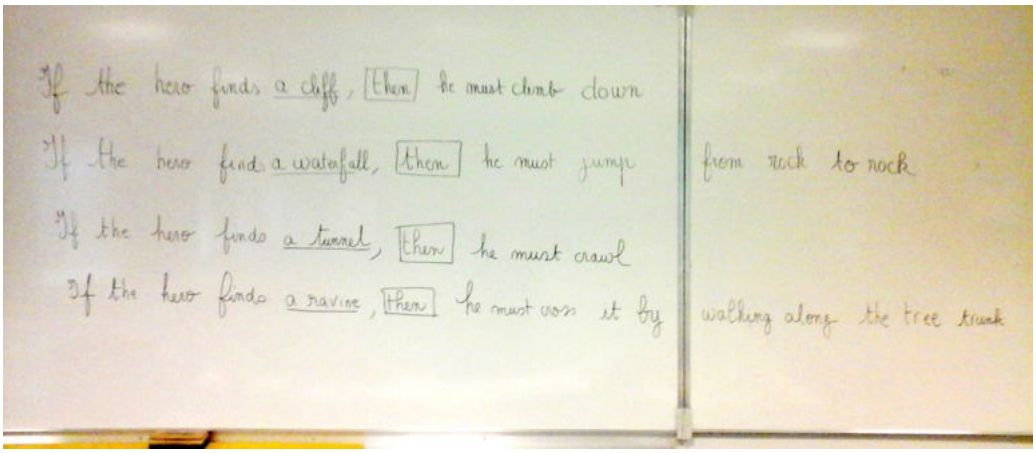
- If done as a class (which is typical in preschool), the class should decide together which situations or obstacles the hero might come across (a river, a ravine, a cliff, a tunnel,

etc.), and for each situation, they should come up with an instruction to overcome the obstacle. As the students dictate the instruction, the teacher writes a sentence on the board.

- If done in groups, the students work on their own, with the teacher suggesting they first create a list of obstacles (checked after 15 minutes) and then the instructions for the hero. Second graders should fill out Handout 13, while third graders can designate someone to write down the instructions. Handout 13 can then be used as a summary in their science notebook.

During the group discussion, the teacher can introduce a new computer term. A method used to resolve a problem is an "algorithm." In this activity, the algorithm is reflected in the series of "tests": a "condition" (*"IF the hero sees a cliff"*) followed by one or more instructions to follow if the condition is met (*"THEN he must climb down it"*). At every stage of his journey, the hero must check that all the program conditions are met and obey all applicable instructions to the letter.

The teacher asks the students to compare this algorithm with another instruction to give the hero: "Return home." Here, the problem is very complex and no there are no explanations on how to resolve it. If the hero does not know how to do it, the instruction will not help him. An algorithm is built from "basic" instructions the hero knows how to do.



Third grade class, Emmanuelle Wilgenbus, Antony

Exercise: Creating other conditional constructs

The teacher asks students to create other instructions following the same rule (being as explicit as possible). For example, they can imagine the hero being in another environment, such as a hostile jungle, ice floe, futuristic city, etc.

They can also ask students to use conditional constructs to explain the algorithms they see every day, such as in sports, grammar class or the school's rules (what you must do in such-a-such situation), etc.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

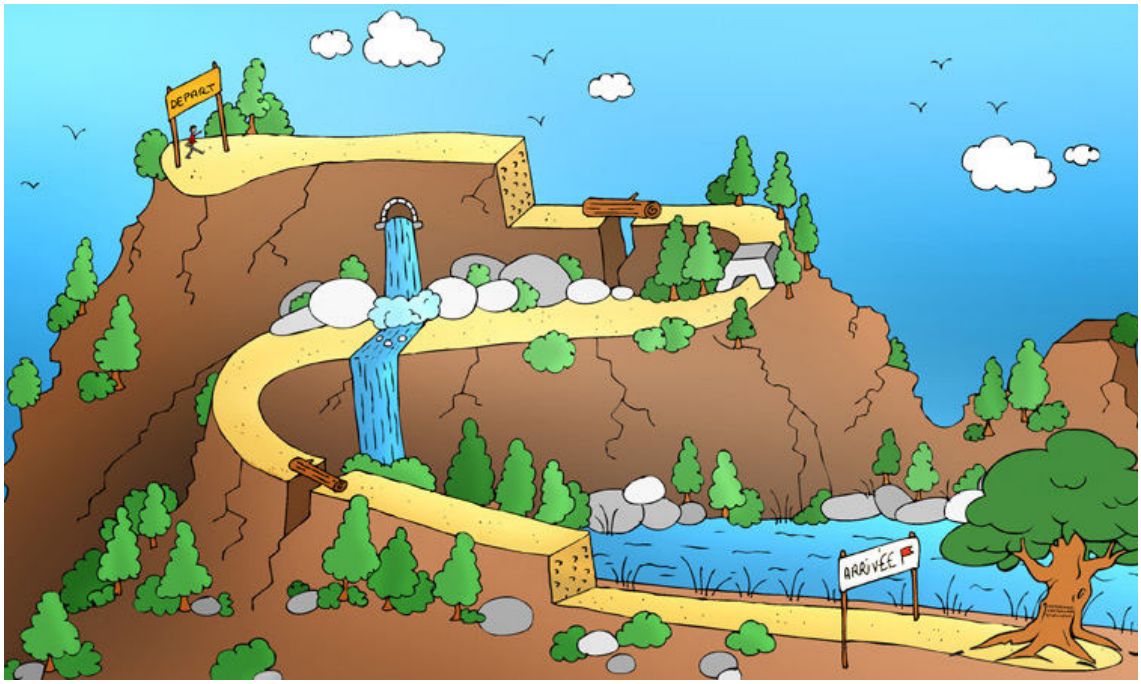
- *An "algorithm" is a method to resolve a problem*
- *A test indicates which action to perform when a condition is met*
- *A condition is an expression that is either true or false*

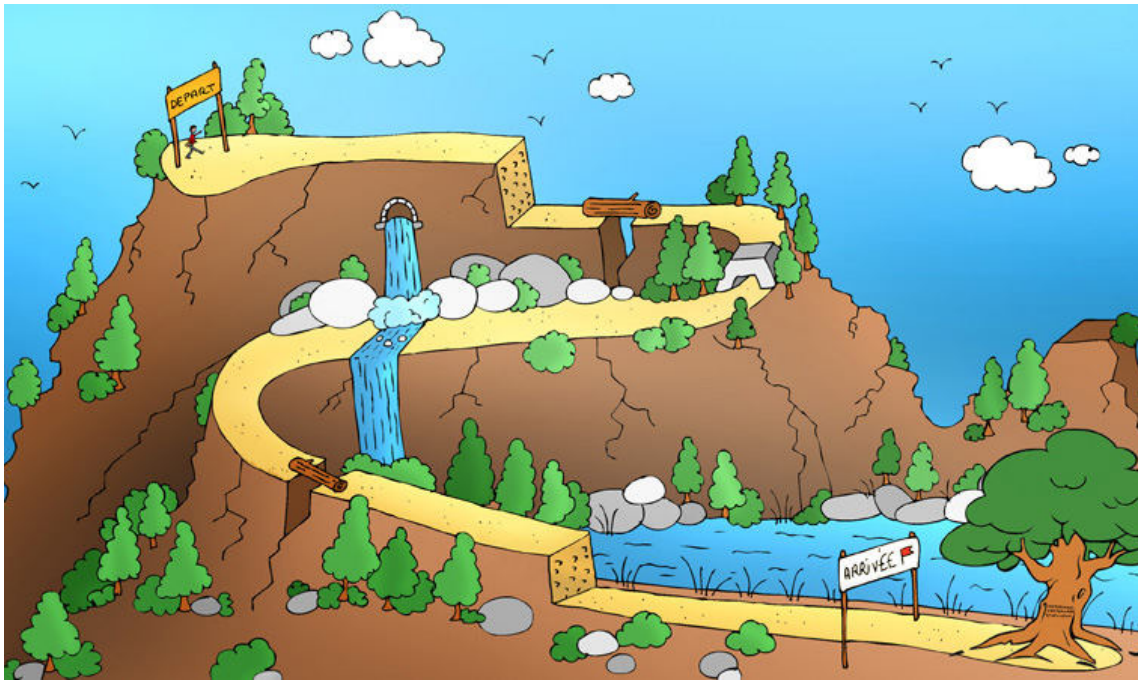
The students write down these conclusions in their science notebooks.

Further study

Using gym equipment, the teacher can create another obstacle course for the hero, using obstacles, tunnels, hoops, steps, etc. The aim of the exercise is to create instructions using the "IF–THEN" structure so students can safely run the course.

HANDOUT 12
The hero's route





HANDOUT 13
Instructions for the hero

Instruction: In the boxes on the left, write down the obstacles the hero might find. Then, in the boxes on the right, write in the instructions to help him get past the obstacles.

IF the hero finds	*a ravine*	THEN he must	*cross it by walking along the tree trunk.*
IF the hero finds		THEN he must	
IF the hero finds		THEN he must	
IF the hero finds		THEN he must	
IF the hero finds		THEN he must	

Lesson 2 - Decoding a message

Summary	At the end of this perilous journey, the hero must solve a riddle carved on a tree trunk. Students understand that it is a coded message. To help the hero, they must decode the message to understand its meaning.
Key ideas *(see Conceptual scenario, page 108)*	"Information" • We can code a text by replacing the letters with numbers.
Inquiry-based methods	Observation, experimentation
Equipment	For each student (second or third grades) • Handout 14, page 118 For the class • Handout 14 (projected on the board)
Glossary	Coding, decoding
Duration	30 min

Starting the activity

In the previous lesson, the hero was able to safely come down the mountain. When he comes to the clearing, he sees a long message carved on a tree trunk that he can not understand.

Experiment: decoding an encoded message (in groups)

The teacher projects the first half of Handout 14 on the board: it is the message carved on the tree trunk. The teacher asks the students what they think about it. They can not read what is written, although it looks like a text written in another language. Each symbol looks like numbers, which will make it easy to name them. Perhaps, to understand the message, students need only to find a correspondence between the symbols and the letters of our alphabet. The teacher introduces the terms "encode" and "decode."

Teaching notes

- For first graders, the entire activity should be done on the board as a class. Second and third graders can write their answers down on paper in pairs.
- To make decoding easier and put more emphasis on the method rather than the result, punctuation has not been encoded.
- In common language, the terms "coding," "encoding," and "encrypting" are often misused or used interchangeably (see vocabulary note in the Level 3 sequence 1, page 213). Here, we are talking about encoding because we are interested in representing alphabet characters using numbers, which is used in computer programming even when information is not confidential. "Encryption" refers to changing a message so that it cannot be understood by unintended recipients.
- In "traditional" coding, "A" must be encoded using "01" because all coding symbols must be the same length (see note in the Level 3 sequence 1, page 213). However, at this age as students are beginning to learn to count, they are told that numbers do not start with a zero (except for zeros, of course). This is why the encoding here is done within boxes to help clearly separate the symbols.

Second grade class, Vanessa Guionie (Bergerac)

The teacher hangs up the entire documentary handout on the board. The students must find the clues to decode this message (second graders should focus on one line at a time, while third graders can decode the entire message). On the board, the teacher fills in the correspondence table based on the group's solutions.

If the students find it difficult to understand how to decode the message, the teacher can make suggestions to point them in the right direction:

- *Which are the shortest words? What words might correspond in English?* The shortest English words are "a" and "I." English also has certain contractions, such as words ending with 'S, 'T, 'M, 'D, as well as certain names that start with O'. Two letter words include of, to, in, it, is, be, as, at, so, we, he, by, or, on, do, if, me, my, up, an, go, no, us, am and the contraction endings 're and 've as well as MC at the beginning of certain last names.
- *Which letter is most common in English?* (Answer: E). *Where might it be here?* In the coded text here, it is the symbol 5. We can assume that "5" is always the letter E in the first message. Incidentally, the letter E is the fifth letter of the alphabet.
- *Because the symbols look like numbers, we can also try replacing the letters of the alphabet on the same row (intuitively, we "want" to replace 1 with A, 2 with B, 3 with C and so on*).

Group discussion

Together, the class decodes the message:

FOLLOW DOWN THE STREAM
BENEATH THE SEA WATER
YOU SHALL FIND THE TREASURE
THAT WILL FULFILL YOUR DREAM

In this code, the letter A is coded using "1," the letter B with "2," the letter E with "5," and so on through Z, which is "26."
The students can encode and decode other messages of their own choosing to share (when doing so, be sure to place the symbols in boxes again).

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *A text can be coded by replacing its letters with numbers chosen ahead of time (for example, 1 can be "A," 2 can be "B," etc.).*

The students write down these conclusions in their science notebooks.

HANDOUT 14
A riddle to decode

Instruction: Decode this message using and completing the correspondence table on the board.

6	15	12	12	15	23	4	15	23	14	20	8	5	19	20	18	5	1	13					
2	5	14	5	1	20	8	20	8	5	19	5	1	23	1	20	5	18						
25	15	21	19	8	1	12	12	6	9	14	4	20	8	5	20	18	5						
20	8	1	20	23	9	12	12	6	21	12	6	9	12	12	25	15	21	18					

The message carved on the tree

1	2	3	4	5	6	7	8	9

1	1	1	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8

1	2	2	2	2	2	2	2
9	0	1	2	3	4	5	6

Correspondence table to decode the message on the tree trunk

Lesson 3 - Programming a route

Summary	The hero cannot reach the treasure at the bottom of the sea, but he finds a small submarine. The students must invent a language to pilot it remotely.
Key ideas *(see Conceptual scenario, page 108)*	"Machines" • The machines all around us simply follow "orders" (instructions). • By combining several simple instructions, we can perform a complex task . "Languages" • We can give a machine instructions by using a special language called a programming language, which can be understood by both people and machines. • A program is an algorithm in a programming language. • A bug is an error in a program. • A very small bug can have very big consequences.
Inquiry-based methods	Experimentation
Equipment	In pairs • Handout 15, page 122 • A pawn (toy, figurine) to be the submarine For the class • Handout 15, page 122, projected on the board • A silhouette with a magnet (or a tack) for the submarine
Glossary	Programming language, instruction, bug
Duration	1 hour

Starting the activity

After following the river, the hero ends up at the sea. On the beach, he sees a dock and goes closer. When he looks into the water, he can see a treasure! But he cannot reach it. However, he sees a small submarine that can be controlled by voice. He will need to explain how to go get the treasure.

Experiment: Inventing a language to guide the submarine (in pairs)

The teacher projects Handout 15 on the board: the scene shows the bottom of the sea, where there is a maze of coral the submarine must navigate through to reach the treasure. In pairs, the students must come up with a set of instructions to describe the route to take. The teacher introduces the term "program" to describe the set of simple instructions that can be performed by a machine.
The conditions are: the submarine cannot move more than one square at a time and it cannot

move diagonally. The students can try to reproduce their route by moving their pawn, making sure to carefully follow all of the instructions.

Group discussion

The teacher asks one of the groups to present their program to the class. To check their programs, students carefully follow the instructions to move the submarine silhouette around the board. If the method works, the teacher goes back to the board and asks if other students have other suggestions.

There are (at least) two languages to command the submarine. We can use "absolute" directions (go towards the surface, go west/towards the dock, etc.) or "relative" directions that depend on the direction the submarine is facing (turn right, go forward, turn left, go back, etc.). Note: it is best to cut instructions such as "go forward one square to the right" into two separate instructions: 1) turn right (while staying in the same square), *then* 2) move forward one square.

Teaching notes

- The first approach to spatial processing (North, West, etc.) is called "allocentric" while the second (right, left, etc.) is called "autocentric".
- Students do not need to know these terms as they will not be used in later lessons. Being able to tell the difference between these two methods is not the aim of this lesson, and students will often mix terms from both.
- A third approach (rarer) can also be suggested: assigning coordinates to each square (A1, A2, B1) and, like in a game of Battleship, code movements by giving the name of square of departure and arrival. For example, "Go from A1 to A2." Please note: The direction "A1 to A2 is not ambiguous because these squares are adjacent. However, "A1 to B7" is ambiguous (and therefore not satisfactory) as there are several ways to move from square A1 to square B7. We will not go into further detail about this method in later lessons.

It is likely that different teams will suggest the two different methods. If this is not the case, the teacher can introduce the other method during the group discussion.

Allocentric languages (or <<absolute>>)	Autocentric language (or <<relative>>)
• Rocks (meaning «move forward one square towards the rocks»), bottom, bottom, bottom, rocks, surface, rocks • East, down, down, down, east, east, up, east	• Forward (meaning «move forward one square right in front of you»), dive, dive, dive, forward, forward, up, forward

Teaching notes

- If this task is easy for students, you can have them program the submarine's return trip, remembering to include the instruction "grab," meaning "grab the treasure", so they do not return empty-handed.

The class remarks that the submarine needs only a very basic language to be controlled (with very few different words). The teacher explains that machines like computers, robots, etc. can be programmed using special languages, called "programming languages," which are much simpler than natural languages such as English, French, etc.

This group discussion is also when the idea of "bug" can be covered. As students present their programs, there will likely be some that have left out or made a mistake in an instruction. When this happens, even if the class knows the result will be incorrect, the teacher can complete the program all the way through to see where the submarine ends up.
A single error can have very serious consequences. An error in an autocentric language can take you farther from the goal than an allocentric language error. However, in both cases, this is a bug and there are two things to take note of. First, the goal is not met, so the failure is just as serious in both cases. Second, if the pirate who left the treasure at the bottom of the sea also set booby traps, you do not want to make a mistake – even a little one.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *A program is a set of instructions in a special language that people and machines can understand.*
- *A bug is an error in a program. A very small bug can have very big consequences.*

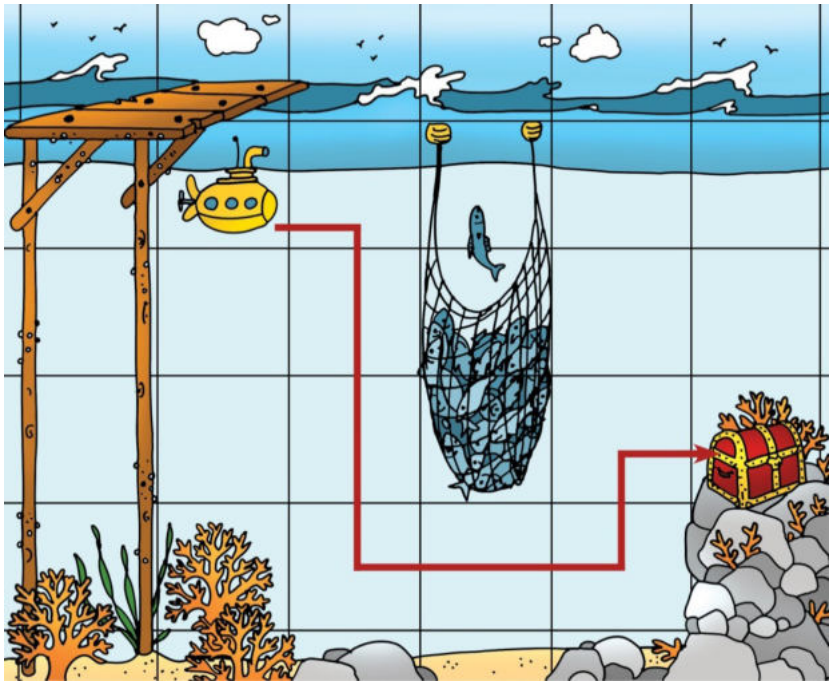
The students write down these conclusions in their science notebooks.

Further study

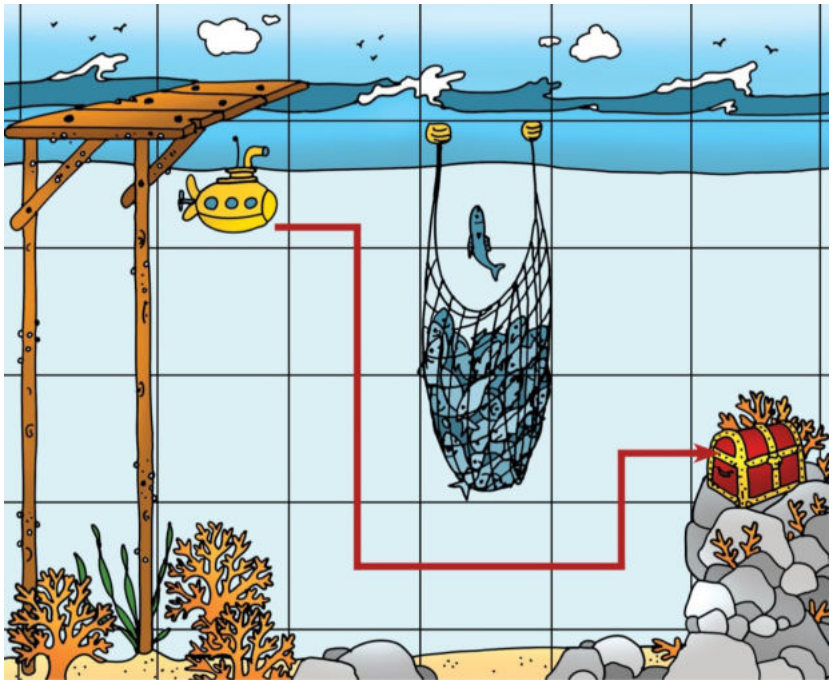
To illustrate how to break complex tasks down into simple instructions, the teacher can do the following activity. They want to do something that seems easy (e.g., drink a glass of water, eat a cookie) but will not obey until given very basic and perfectly explicit tasks. The students must explain without being vague what the teacher must do (i.e., "program" the teacher as if they were a machine): "raise your hand," "raise your elbow," "bring your hand close to the glass," "grab the glass lightly," "bring it towards your mouth," "open your lips," etc. The teacher mimes the orders as they hear them. Obviously, the degree of detail allowed in "simple" instructions is up to the teacher while the students begin to understand how difficult it is to break tasks down without ambiguity for a machine to do a task that people find easy.

HANDOUT 15
The coral maze

Instruction: Write a program (set of instructions) so the submarine can follow the route to collect the treasure. But be careful: it can only move one square at a time and cannot move diagonally.



Instruction: Write a program (set of instructions) so the submarine can follow the route to collect the treasure. But be careful: it can only move one square at a time and cannot move diagonally.



Lesson 4 - Summoning the magician

Summary	The hero must summon a magician by asking for help from the birds. To do this, he must create a drawing on the ground using white and black rocks. The students learn how to pixelate an image in black and white.
Key ideas *(see Conceptual scenario, page 108)*	"Information" • We can represent an image using a grid of black or white pixels.
Inquiry-based methods	Observation, experimentation
Equipment	For each student • Handout 16, page 126 • Handout 17, page 127 (or on tracing paper) For each group • Handout 18, page 128
Glossary	Pixel
Duration	1 hour

Starting the activity

In the treasure chest from the bottom of the sea (previous lesson), the hero finds a parchment that describes a recipe for a magic cake that will let him return home. But without the ingredients or utensils, the hero cannot make it. The parchment tells him about a magician who can help him. To contact him, the hero must send a message to the birds, who can find the magician.

Experiment: Pixelating an image (in pairs)

Because the birds in this country do not speak the same language as the hero, there is only one solution to communicate with them: draw something on the ground to get their attention. The hero can use large rocks in black or white to create his message. He can use them to outline something on the ground that the birds can see from above.

The teacher gives students Handout 16 and the top part of Handout 17 (7x7 grids). Using white or black rocks placed on a 7 x 7 grid (called "pixels"), students must create a basic rendering of the magician's hat. Each grid square must be either entirely black or entirely white, which corresponds to using a black rock or a white one.

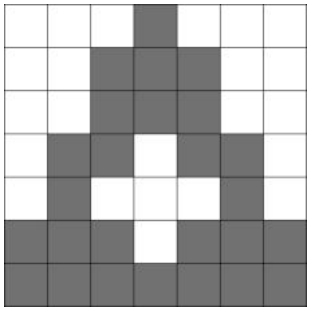
Scientific notes:

- The squares of this image are called "pixels" (a portmanteau of the words *picture* and *element*). Here, students are pixelating an image. Pixelated images come in various formats: black and white, grayscale and colored images. This Level 2 lesson will only deal with black and white images. Other formats are covered in Level 3 activities (see Sequence 3: Sending news, pages 295 and on).

Teaching notes

- Printing out the magician's hat in black and white allows students to trace the outline by placing Handout 16 under Handout 17 (which can be printed on tracing paper if desired).
- Regardless of how students transfer the outline, it is important to remind them that they are not meant to copy the hat exactly. The squares must be either entirely black or entirely white, and students cannot divide the squares by tracing extra details to make their image look more like the original drawing.

Group discussion

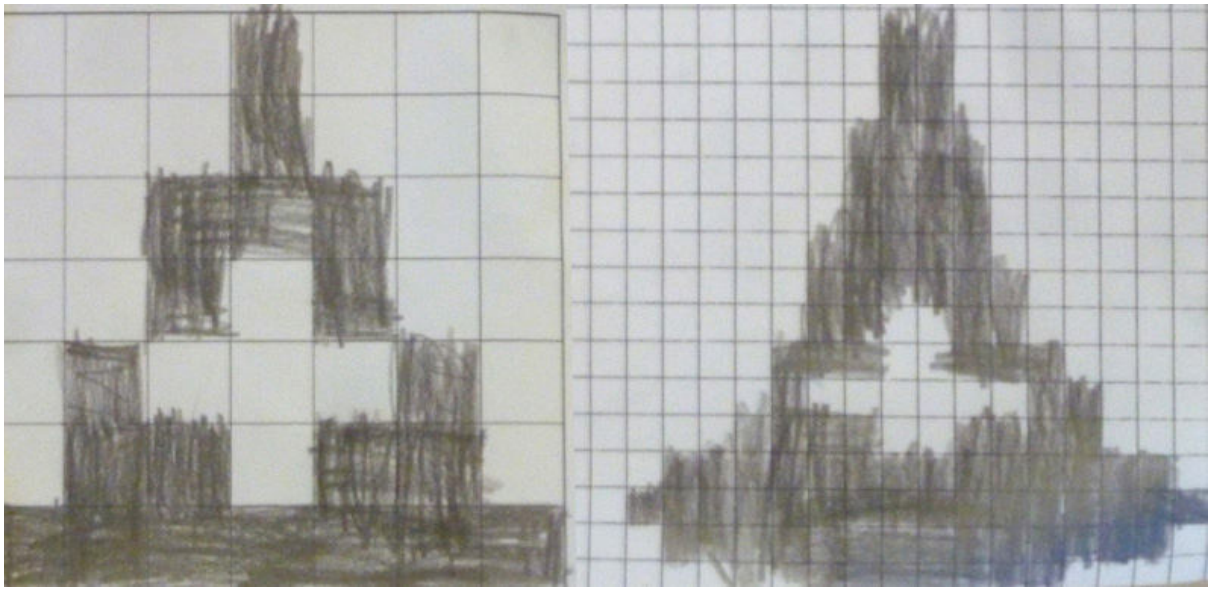


The students compare the pixelated images they created. Opposite is an example of one possible result, but there are a range of possibilities.

While the images may be hard to recognize at close range, when viewed from the other side of the classroom, the image clearly looks like a hat.

Experiment: Improving the pixelated image (third grade)

Older students will find that the pixelated image is not precise enough. The teacher can ask them to find ways to improve the result. Students will likely come up with two possibilities: either using rocks (pixels) in other colors, or using more rocks. The second option, which has long been the actual approach, is a good segue to the idea of "resolution": by increasing the number of grid cells, we can refine the image and make it easier to identify (but the number of pixels increases very quickly: a 7x7 grid has 49 pixels; when doubled, we can draw 196 pixels, and so on).
The teachers gives students a second, more detailed grid with 14x14 pixels (bottom half of Handout 17). They must repeat the activity: pixelate the initial image on this new grid. Again, make sure the students do not simply trace the original drawing.



On the left, the magician's hat pixelated on a 7x7 grid. On the right, the same hat pixelated on a more detailed grid. Third grade class, Emmanuelle Wilgenbus, Antony

Conclusion and lesson recapitulation activity

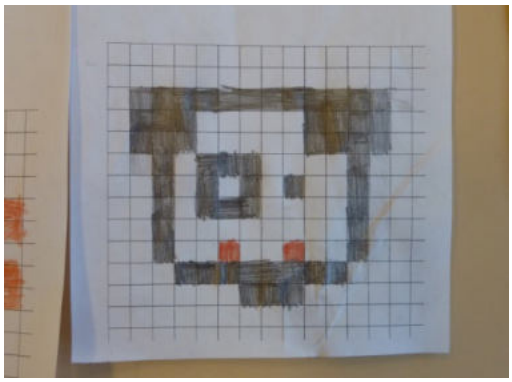
The class summarizes together what they learned in this lesson:

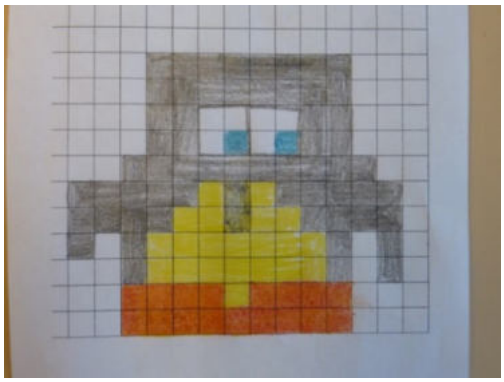
- *We can represent an image using a grid of black or white pixels.*

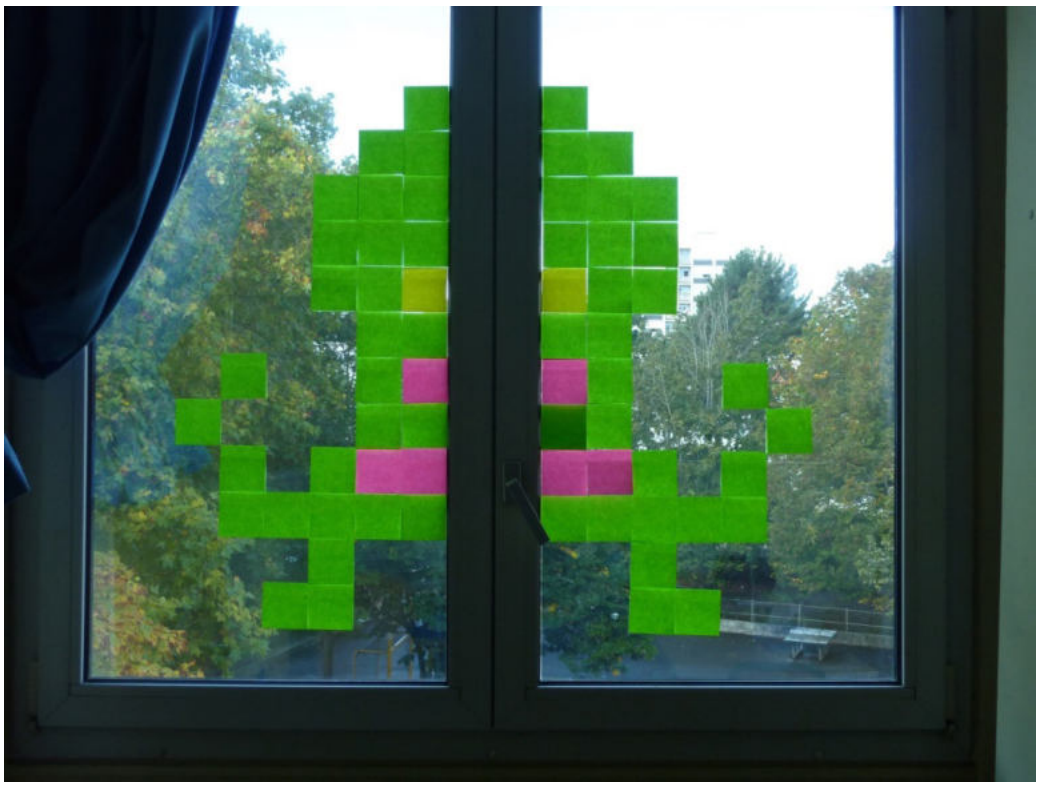
The students write down these conclusions in their science notebooks.

Further study

In art class, the students can create the grid themselves with posters or clay, for example. They may also want to pixelate other drawings using the same principle. We highly encourage students to reinforce the idea of pixilation in this way. Handout 18 also provides additional images that work very well for this type of activity.

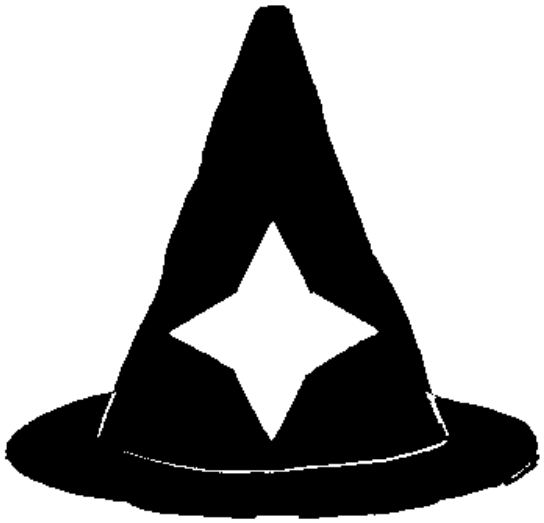




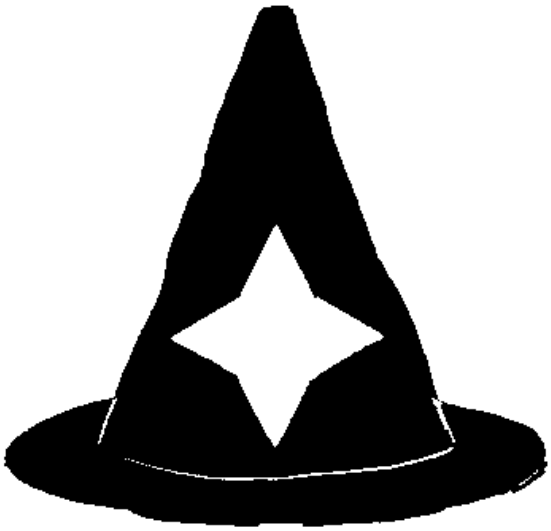


Third grade class, Emmanuelle Wilgenbus, Antony

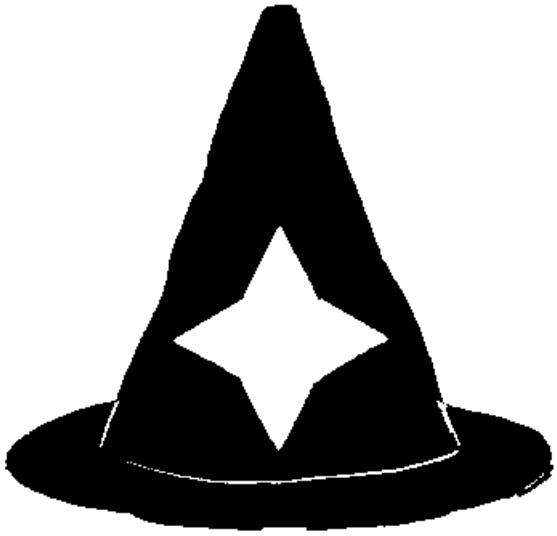
HANDOUT 16
Summoning the magician: The hat to draw



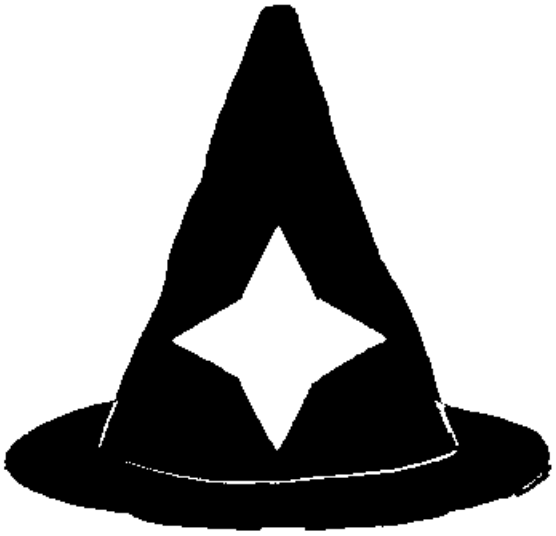
Instruction: Fill in certain grid squares in black to copy the magician's hat. Each square must be entirely black or entirely white.



Instruction: Fill in certain grid squares in black to copy the magician's hat. Each square must be entirely black or entirely white.

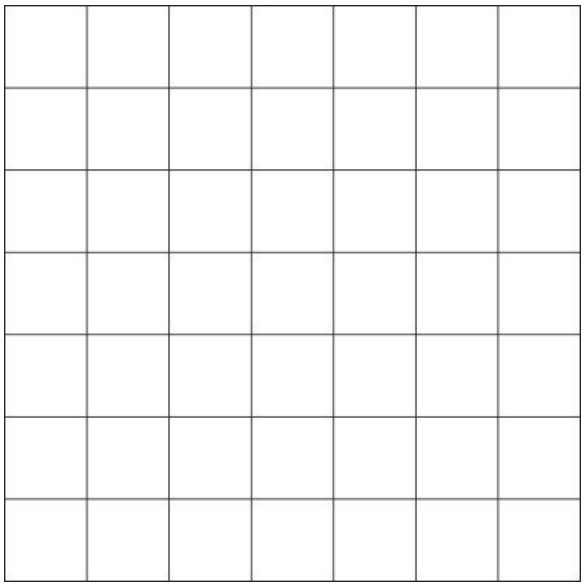


Instruction: Fill in certain grid squares in black to copy the magician's hat. Each square must be entirely black or entirely white.

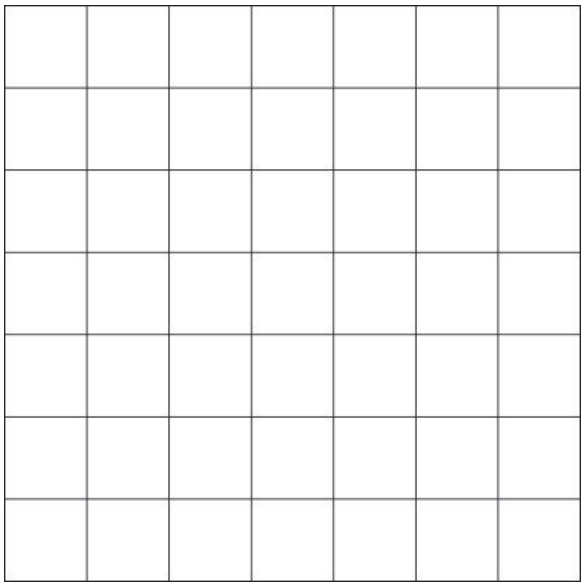


Instruction: Fill in certain grid squares in black to copy the magician's hat. Each square must be entirely black or entirely white.

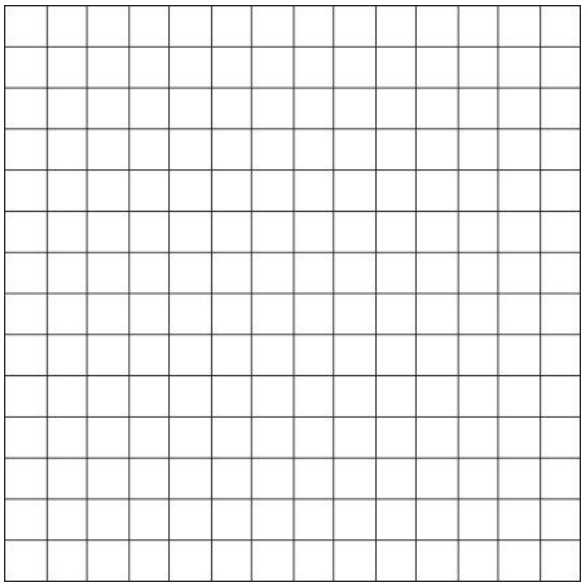
HANDOUT 17
Summoning the magician: Alternate grids



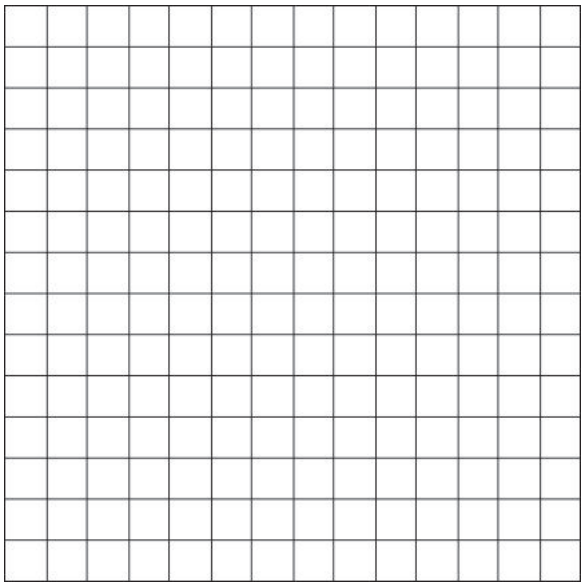
Instruction: Fill in certain grid squares in black to copy the magician's hat. Each square must be entirely black or entirely white.



Instruction: Fill in certain grid squares in black to copy the magician's hat. Each square must be entirely black or entirely white.

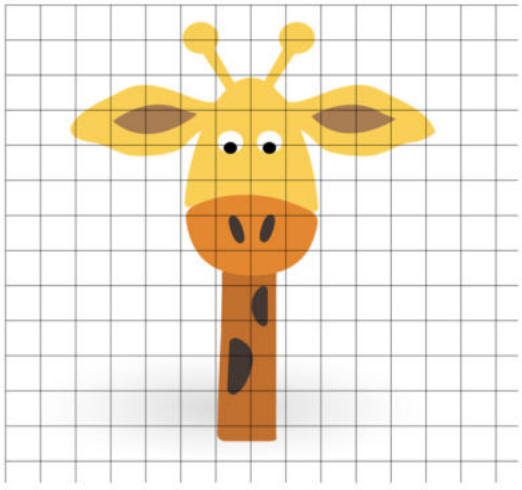


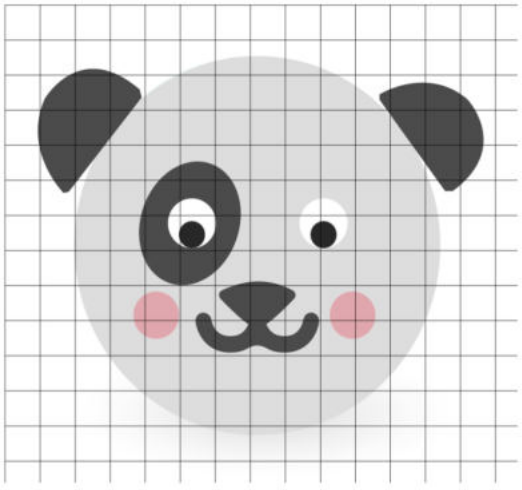
Instruction: Fill in certain grid squares in black to copy the magician's hat. Each square must be entirely black or entirely white.

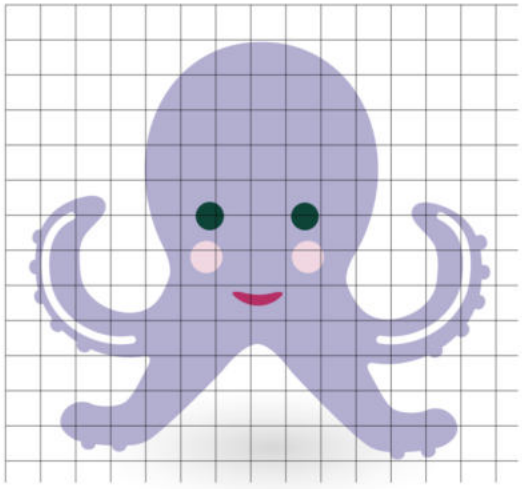


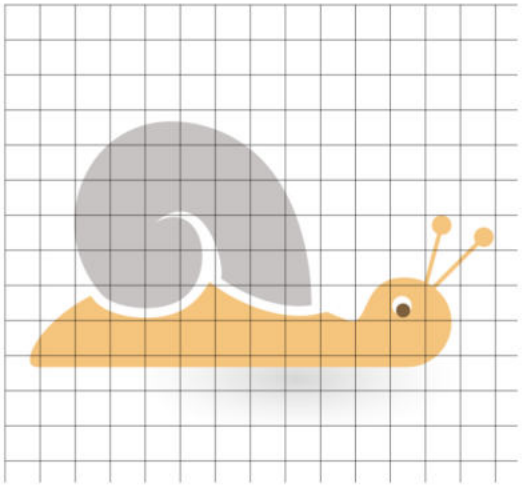
Instruction: Fill in certain grid squares in black to copy the magician's hat. Each square must be entirely black or entirely white.

HANDOUT 18
A few suggestions for the <<Post-It art>>









Instruction: Using the grid, pixelate your chosen image, then create it using Post-Its.

Lesson 5 - (Optional) Following a recipe

Summary	Thanks to the magician, the hero will be able to create the magic recipe. The students must analyze the structure of the recipe to find the elements for an algorithm.
Key ideas *(see Conceptual scenario, page 108)*	"Machines" • By combining several simple instructions, we can perform a complex task. "Algorithm" • An «algorithm» is a method to resolve a problem. • An algorithm can contain basic instructions, tests and loops. • A test indicates which action to perform when a condition is met. • A condition is an expression that is either true or false. • A loop is used to repeat the same action several times. • Certain loops are repeated a specific number of times. • Certain loops are repeated until a condition is met.
Inquiry-based methods	Observation, experimentation
Equipment	For each student • Handout 19, page 132 For each group: • Handout 20, page 133
Glossary	Algorithm, instruction, loop, test, condition
Duration	1 hour

Foreword

This lesson is based on a rather long text that covers measurement concepts (g, mL, cL) that will not yet have been addressed. This exercise should be done in third grade classes, or perhaps a second grade class at the end of the year. However, younger students can do the same work with simpler recipes, even if all structures in the following recipe may not be covered.

Starting the activity

After being summoned by the birds, the magician appears from a huge cloud of white smoke. The hero hands him the parchment from the treasure chest. The magician understands that the recipe will send the hero back home. With a wave of his magic wand, the magician makes all of the ingredients and utensils appear to make the magic recipe.

Observation: Is a recipe an algorithm? (as a class)

The teacher gives students Handout 19 and also hangs a copy of the recipe hero found on the board. When he eats a cake made from this recipe, he will immediately return home. Students read the recipe individually to become familiar with the recipe text.
Once students have read and understood the recipe, the teacher divides the class into groups and gives each group Handout 20. The instruction is simple: the groups must categorize the parts of the recipe according to any criteria of their choosing.
During the group discussion, students explain why they grouped certain parts of the text together and excluded others. Gradually, the class identifies four different structure categories in the recipe:

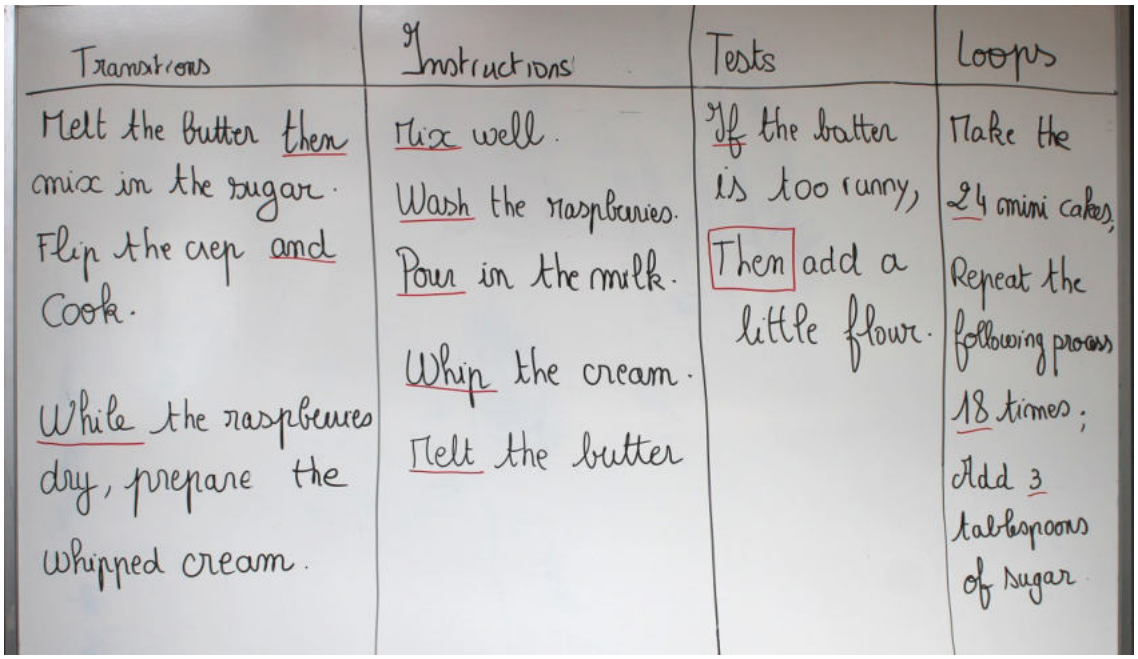
- Instructions: *"Melt the butter"*
- Tests: "***If*** *the batter is too runny,* ***then*** *add a little flour."*
- Repetitions, which are called "loops" in computer language (this concept is covered in more detail in Lesson 1.6 page 134): *"Repeat* ***18 times****"*
- Sequences: *"Melt the butter* ***then*** *mix****," "While*** *the raspberries dry, prepare the whipped cream****."***

Teaching notes

- If the students have trouble, the teacher can gradually guide them with questions such as "*Are there any instructions?*", "*What about tests?*" (these ideas are covered in Lesson 1.1 page 110), or "*Are the instructions separate steps or are they sequences?*"
- The conditional structure "IF–THEN" is quickly identified this activity. The teacher can also tell students that instructions are often verbs (imperative tense), while sequences use conjunctions such as *then*, *and*, etc.

Scientific notes:

- Sequences can be divided more precisely:
    - Temporal sequence: *"Melt the butter* ***then*** *mix"*: These are "sequential instructions" (they are done one after the other).
    - Independent instructions: ***While*** *the raspberries dry, prepare the whipped cream."* These are "parallel" instructions (they are done simultaneously and independently)
    - Event-driven instructions: ***"If*** *the matter is too runny"* (they are only done if a certain event happens)
- In this recipe, it is also possible to identify several types of loops:
    - Iterative loops (*"Repeat* ***18 times"****)*
    - Conditional loops (*"Cook on high heat* ***until*** *the batter is golden brown"*)



Transitions	Instructions	Tests	Loops
Melt the butter then mix in the sugar. Flip the crep and Cook. While the raspberries dry, prepare the whipped cream.	Mix well. Wash the raspberries. Pour in the milk. Whip the cream. Melt the butter	If the batter is too runny, Then add a little flour.	Make the 24 mini cakes. Repeat the following process 18 times; Add 3 tablespoons of sugar

Third grade class, Emmanuelle Wilgenbus, Antony

Application: Making the recipe

Following the quantities in the recipe, the teacher can do a cooking activity with the class and let them eat what they make.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson.

- *Making a cake is a complex task.*
- *By combining several simple instructions, we can perform a complex task.*
- *The method for making a cake is called an algorithm or a recipe.*
- *An algorithm can contain basic instructions, tests and loops.*
- *A test indicates which action to perform when a condition is met.*
- *A loop is used to repeat the same instruction several times.*
- *Certain loops are repeated a specific number of times.*
- *Certain loops are repeated until a condition is met.*

The students write down these conclusions in their science notebooks.

HANDOUT 19
The recipe for the magic cake

Ingredients (for 24 mini cakes):

- *180 g flour*
- *2 eggs*
- *1 teaspoon vanilla*
- *400 ml 2% milk*
- *40 cl heavy cream*
- *100 g corn starch*
- *100 g granulated sugar*
- *30 g butter*
- *72 raspberries*
- *powdered sugar for dusting*

Preparation:

Melt the butter, then mix in half of the sugar and the eggs in a large bowl. Pour in the milk, then add the vanilla and corn starch. Mix well. Little by little, add the flour, stirring until the batter is smooth and there are no lumps. If the batter is too runny, then add a little flour.

To make the crepes, repeat the following process 18 times: pour a ladleful of batter on a grease pan, cook on high heat until the batter is golden brown, flip the crepe and cook the other side for one minute.
Next, cut each of the 18 crepes into 4 mini crepes using a glass or a biscuit cutter.

Wash the raspberries. While the raspberries dry, prepare the whipped cream. Whip the cream until it forms stiff peaks. Add 3 tablespoons of sugar and whip for another minute. Be careful not to over whip the cream or it could turn into butter.

Make the 24 mini cakes: In a small bowl, put one mini crepe, then a spoonful of whipped cream, another mini crepe, then another spoonful of whipped cream, then a last mini crepe. Decorate with 3 raspberries. Sprinkle with powdered sugar.

HANDOUT 20
The recipe for the magic cake: Expert

Melt the butter, then mix in half of the sugar...

Pour in the milk...

Mix well.

...repeat the following process 18 times...

Flip the crepe and cook the other side

Wash the raspberries.

While the raspberries dry, prepare the whipped cream.

If the batter is too runny, then add a little flour.

Make the 24 mini cakes...

Decorate with 3 raspberries.

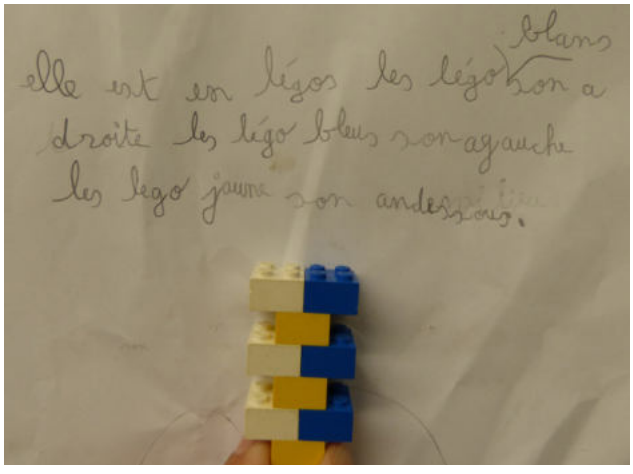
Sprinkle with powdered sugar.

Lesson 6 - (Optional) Building a magic key

Summary	The hero can return home. Before he leaves, the magician gives him a magic key that will let him come back. The students must describe the algorithm that will let him duplicate this key.
Key ideas *(see Conceptual scenario, page 108)*	"Languages" • A program is an algorithm in a programming language.. "Algorithm" • An "algorithm" is a method to resolve a problem. • A loop is used to repeat the same action several times. • Certain loops are repeated a specific number of times.
Inquiry-based methods	Observation, experimentation
Equipment	For each group: • Lego®-style building blocks.
Glossary	Coding, decoding
Duration	1 hour

Preparation

Please note that this lesson requires a large number of Lego®-style building blocks in standard colors and shapes. Each group will have a "key" made from three to six blocks that will then be copied (up to around six), so you will need to have at least 100 to 200 blocks for the entire class. Before the lesson, the teacher makes one or two "keys" using Lego®-style building blocks. They should be stacked in repetitive patterns, such as in the example below. Repeating identical patterns emphasizes the idea of "loop" in this lesson.



Second grade class, Vanessa Guionie, (Bergerac)
"The key is made of Legos. The white ones are on the right, the blue ones on the left, the yellow ones below."

Starting the activity

The magician was able to help the hero make the magic recipe to return home. Before leaving, the magician gives the hero a magic key so he can come back to this world as often as he wants. The hero then takes a bite of the cake, closes his eyes, and wakes up at home. In his hand is the magic key that he wants to share with his friends. But how can he describe it to them so that they can make their own?

Observation: Describe and reproduce the magic key (in groups)

The teacher gives each group one of the keys made beforehand. In groups, the students must identify their key's basic pattern as well as the number of repetitions so they can completely recreate it.
So that students describe their key well, give them the following scenario: "*Imagine that we're on the phone and you have to explain to me how to make the key. You have to describe everything, because I can not see what you're looking at and you can not see what I'm doing*."

Teaching notes

- Students may come up with the idea to number the studs on the Lego® blocks to make the description easier. This takes more time, but is more precise.

Third grade class, Emmanuelle Wilgenbus, (Antony)

Group discussion

When they think they have a good, unambiguous description of their key, students from each group show the class their key and describe it. The teacher helps them revise their "program" and "instructions" by introducing the term "loops."

Experiment: Creating and describing a new key (in groups)

One student from each group makes a new key using the same principle as the one made by the teacher: a basic pattern reproduced two, three or four times. Without showing it to the rest of the group, the student must then describe the pattern and the loop so the others can reproduce a copy of their key. The teacher makes sure the student does not show the original to the others and does not correct them if they make any mistakes. The student must come up with an unambiguous description of their key by being precise, clear and concise.
Finally, the students compare the original key with their copies. If there are any differences, they must try to understand why.

Exercise: Creating more complex keys (in groups)

The teacher hands out a second, more complex key to each group. It features a simple pattern that combines two loops: for example, “ABC ABC ABC D ABC ABC ABC D.” The class must come up with a description of the basic pattern (ABC ABC ABC D) that is applied twice, and which in fact includes a triple loop with a smaller basic pattern (ABC).
Following this example, students should create more complex keys with nested loops, being sure to describe it so that the others can copy it properly.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *An algorithm is a method used to resolve a problem.*
- *A loop is used to repeat the same instruction several times.*
- *Certain loops are repeated a specific number of times.*

The students write down these conclusions in their science notebooks.

Sequence 2: Telling the adventure with Scratch Junior

This sequence deals with **programming** an animation to tell the hero's adventure from Sequence 1.

Scratch Junior, an ideal environment for learning to code

There are a variety of tools for learning to program, but few of them are suited to young elementary students who are still learning to read. To do this sequence as early as first grade, we chose to base it on the ***Scratch Junior***[11] programming environment, where all programming elements are represented by drawings, without any text other than numbers.
Scratch Junior has other advantages, such as being free, easy to use and very complete. Moreover, students who have used *Scratch Junior* during elementary school will feel comfortable continuing learning in *Scratch* grades four and up (see Sequence 2 in the Level 3 activities, from page 229).

What can you do if your school has only computers but not tablets?

Scratch Junior is only available for tablets. This is easier for younger children because they do not need to know how to type, use the mouse or browse through a tree structure of folders and files. Tablets are ideal for programming in elementary school while computers are more relevant for grades four and up.
However, it is still possible to do programming in grades three and under. If your school does not have tablets, you can adapt this sequence to the more elaborate *Scratch* version, which runs on computers (on Windows, Mac and Linux operating systems). To make the adaptation easier, we suggest doing an alternative sequence using *Scratch* instead of *Scratch Junior* (Sequence 2a: Telling an adventure with *Scratch*, page 164).

What can you do if your class has a robot?

We have another alternative to *Scratch Junior* if your class has a Thymio robot (Sequence 3, page 182). Programming a robot uses the same concepts as programming on a computer; they are simply applied to a physical object. The Thymio robot is programmed using visual programming language (VPL), which, although not as rich or simple as *Scratch Junior*, is a very good tool for learning to program from first grade.

Working in half-groups

Ideally, **there should be a tablet for every student pair.**
To make this possible and make it easier to manage the class during the programming activities (during which the teacher will have to work closely with students), half the class can work independently on exercises not requiring assistance from the teacher while the other half of the class can work on the project. Then, the teacher can have the groups switch roles (A and B in the following lessons).

11 The *Scratch Junior* application can be downloaded at http://www.*Scratch*jr.org/ in two versions, one for Android tablets (tap the GET IT ON Google play icon) and one for Apple tablets (tap the Download on the App Store icon).

Doing the project yourself first

It is essential for the teacher to take two or three hours of their own time BEFORE the first programming lesson to get familiar with *Scratch Junior* and carry out the tasks the students will have to perform during the project.
Otherwise, they may not be able to help the students when they need it. It is very easy (simply follow the instructions in this sequence) and even quite amusing.

	Lesson	Title	Page	Summary
	Lesson 1	Getting started with *Scratch Junior*	139	The students are introduced to *Scratch Junior*, an easy-to-use graphic programming environment for children ages 5 to 8. They explore the ways to control a character's movements.
	Lesson 2	The first episode: Choosing the hero and controlling his movements	146	Students tell an episode of their hero's adventure. While they do so, they learn the new functionalities of *Scratch Junior* (deleting a character, importing a new character, choosing a setting) and are exposed to the key ideas from the previous lessons (set of instructions, event).
	Lesson 3	Simplifying a program by using loops	149	The students continue learning to use *Scratch Junior* by exploring the instruction «repeat...,» which is a loop. They practice anticipating what a program given to them will do, combining loops and movement instructions. Finally, they revise their initial program by replacing the repeated instructions with loops.
	Lesson 4	Coordinating several scripts	153	Students tell a new episode of their hero's adventure, with more autonomy than in the first lessons. They discover new functionalities in *Scratch Junior* and deepen their understanding of what a set of instructions and a program are.
	Lesson 5	Predefined loops and infinite loops	157	Students tell a new episode of their hero's adventure. They reinforce the key ideas from the previous lessons, namely predefined loops, and learn about infinite loops.
	Lesson 6	Adding recorded dialogues to the program	160	Students learn to record character dialogues.
	Lesson 7	Producing the final episode autonomously	162	Students work on their own to tell the last episode of their hero's adventure. They cover the key ideas from the entire sequence and finish their program.

Lesson 1 - Getting started with Scratch Junior

Summary	The students are introduced to *Scratch Junior*, an easy-to-use graphic programming environment for children ages 5 to 8. They explore the ways to control a character's movements.
Key ideas *(see Conceptual scenario, page 108)*	«Machines» and «Languages» • We can give a machine instructions by using a special language called a programming language, which can be understood by both people and machines. • An «algorithm» is a method to resolve a problem. • A program is an algorithm in a programming language.
Equipment	For the class • (Recommended) A video projector system to show the teacher's tablet screen to the entire class. For each pair or small group • An Android or OS tablet with the *Scratch Junior* application installed[1]. For each student • Handout 21, page 145 (this handout will be used again in the other lessons)
Glossary	Program, script, character, stage, instruction, event
Duration	1 hour

Starting the activity

The teacher explains to the students that they are going to use a tablet to tell the main episodes of their hero's adventure. To do this, they must program the tablet, i.e., tell it what to do. They will have to use a special language, called a *programming language*, which can be understood by both the students and the tablet. The language they are going to use is called *Scratch Junior*. Today, they are going to learn to use *Scratch Junior* and next time they will begin working on their stories.

Teaching notes

- You learn to program by programming, not by watching someone program. There are benefits to working on a problem with others, but it is also important that all students have a chance to do programming exercises individually. We suggest putting students into small groups with tablets (ideally, two students per tablet) and to have one student handle the tablet at a time and then having them switch roles every five to ten minutes.
- As explained on page 138, we recommend splitting the class in half. Make sure that the student groups using one tablet name their files with easily identifiable names and do not delete others' work. We suggest that class groups A and B save their programs using file names that start with A and B, respectively.
- Additionally, tablets should be identified (e.g., numbered) so that students can continue working on the same tablet during every lesson.

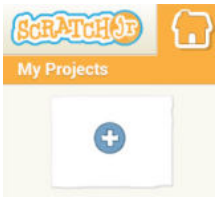
Launching Scratch Junior and basic presentation (as a class)

The teacher projects the tablet screen on the board to show students the basics of how to use *Scratch Junior*.

Starting the software

To start *Scratch Junior*, tap once on the following icon from the list of applications installed on the tablet:

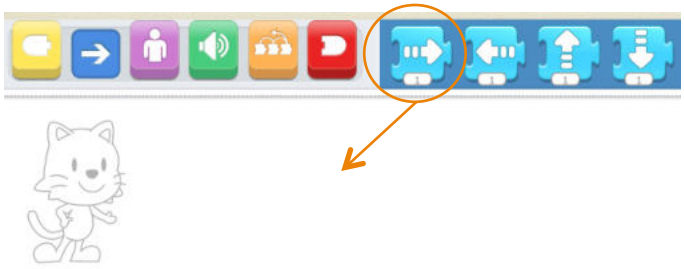
You come to a home screen with two icons: a house and a question mark. If you tap the house, you are taken to a new page that shows everything that has been created on the tablet with *Scratch Junior*. Tap the "+" icon below to create a new program:



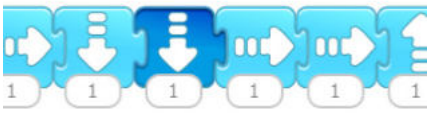
You then go to the screen shown on Handout 21. Students will program the tablet from this screen.

Principle of a sequence of instructions

In the center of the screen, there is a gray rectangle with a drawing of a cat in the middle. This is where the story will be told. To program the story, *instructions* (small colorful puzzle pieces) are placed in the large white strip at the bottom of the screen (*programming area*). The silhouette of the cat, to the left of the programming area, indicates that the instructions placed here are for the cat. If, for example, you use your finger to drag the circled icon below to the programming area (called a *drag and drop*) and then tap the icon, you will see the cat move slightly to the right:



You can add other instructions and connect them to the first one, then tap the group of instructions. You can see the cat follow all of the grouped instructions in order, from left to right. When an instruction is being executed, the puzzle piece becomes darker:



A block of instructions within the programming area is called a *script*. Several scripts can coexist and be executed simultaneously. All of the scripts together are called a *program*.

Teaching notes

- This presentation can be divided into two phases, especially for younger children. The teacher can show how to start the software and the students can do it immediately; then the teacher can explain the principle of a sequence of instructions with the example of having the cat move, and the students then experiment with it on their own.
- On the home screen, the question mark will take you to a presentation video of *Scratch Junior*. This video is very useful for helping teachers learn to use the software, but it is a little too fast for very young students. We suggest showing the video in steps; here, for example, show only the first ten seconds.
- If possible, it is preferable for the teacher to have a demonstration tablet connected to a projector so the class can see the screen and to show students a few actions at an appropriate pace. It should be noted that depending on the tablet brand, this may be rather complex and require third-party applications (which are not always free). The site below explains the process: https://blog.tripplite.com/how-to-connect-a-tablet-to-a-dvi-monitor-flat-screen-tv-or-hdmi-projector/

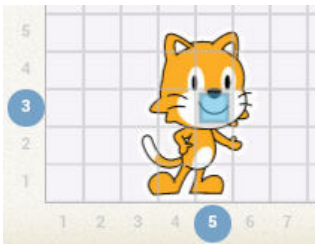
Getting started with Scratch Junior (in small groups, ideally in pairs)

Students are divided into as many groups as there are tablets available. In each group, one student is designated to handle the tablet for the first few minutes (see teaching notes at the start of this lesson). Students will then take turns handling the tablet. The student opens the *Scratch Junior* software, creates a new story and practices making the cat move, as explained to the class.

The teacher gives students time to explore the environment and let everyone see what the instructions do (for example, five minutes for each student in the group): moving the cat up, down, right and left, turning both directions, hopping and returning to the starting positing, combining several movement instructions.

The teacher brings up the grid (icon) and shows the class that when the cat is moved with a finger, a blue square moves with him:

This square indicates the cat's position: here, for example, the cat is in the grid square where the third row from the bottom intersects with the fifth column from the left, which should then be written down (row 3, column 5). The teacher asks the students to bring up the grid on their tablets and gives them a few challenges.



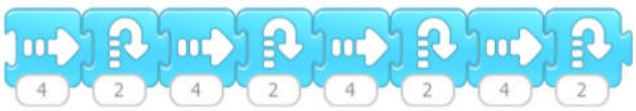
Challenges: Controlling the cat's movements (ideally in pairs)

The challenges gradually get more difficult. Here are a few examples of possible challenges and the programs to solve them:

- Exercise A: Move the cat three squares to the right, then two squares up:
- Exercise B: Move the cat across the screen from left to right, with a hop every four steps (cat starts from column 3):
- Exercise C: Have the cat move around the entire stage (cat starts from the square row 3, column 3).

Second grade class, Vanessa Guionie (Bergerac)

For each challenge, a student presents their solution to the class using the demonstration tablet. Some students may have noticed that the number "1" that appears on the movement instructions can be replaced by a bigger number and that it controls the number of squares the cat moves. If so, the student sends one of these students to show the class their solution to the first challenge. If not, explain this feature at the end of the first challenge by showing two scripts in the programming area and compare what they do by tapping on them. The teacher asks the students which they prefer and tells them that if there are a lot of instructions for a character, it is best to opt for the script that takes up less space and to remove the other one by dragging it out of the programming area.
The answers to the exercises are:

- Exercise A:  or
- Exercise B: 

- Exercise C: 

Teaching notes

- If the cat reaches the edge of the screen, you can move it wherever you like with a drag and drop.

Activity: Triggering an event (ideally in pairs)

The teacher shows students how to switch to full screen mode (icon  , upper bar). The class observes two things: first, the grid disappears (it is used only to facilitate programming movements), and second, you can no longer launch scripts because they no longer appear and you cannot tap them. When you tap the green flag, the cat goes back to the starting position, but does not execute the movements. This problem can be resolved as follows: tap the yellow icon to bring up the commands and choose the green flag:

Next, from the programming area, connect this instruction to the left side of the instructions to trigger, like in Exercise A:

Now, a “tap on the green flag” (called an *event*) triggers the movement.
Students add the command instruction to their program and try out what it does in full screen mode. The teacher tells them that the program is always executed in the same way. They show how to exit full screen mode (icon ).
The teacher asks the students to explore on their own what another event means, shown by the following icon  :

This will require (although the teacher does not say so) students to create a script, with the first instruction being this icon, and they must figure out how to trigger this script without tapping directly on the instruction block.
During the group discussion, the students share their conclusion: the event represented by the finger touching a character is a “tap on the character.” The teacher explains that all instructions in *Scratch Junior* are drawings to be easily understandable and that they should try out instructions to see what they do.
The teacher shows the class how to save the programs: tap the upper right corner of the screen, where you can see the corner of an orange book. Enter the file name in the blank field (have students use basic names, such as “AP1” for Practice program 1, class group A), then validate with the icon  .

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson.

For younger students:

- *I give orders to a tablet using a programming language. The tablet always obeys the orders.*

For older students:

- *We can give a machine instructions by using a special language called a programming language.*
- *Instructions are put together in a program so the machine can perform them.*
- *If you launch the same program several times, it will always do the same thing.*
- *In* Scratch Junior*, the programs have one or several instruction blocks, called scripts. The scripts are triggered by events, such as "tap on the flag" or "tap on a character."*

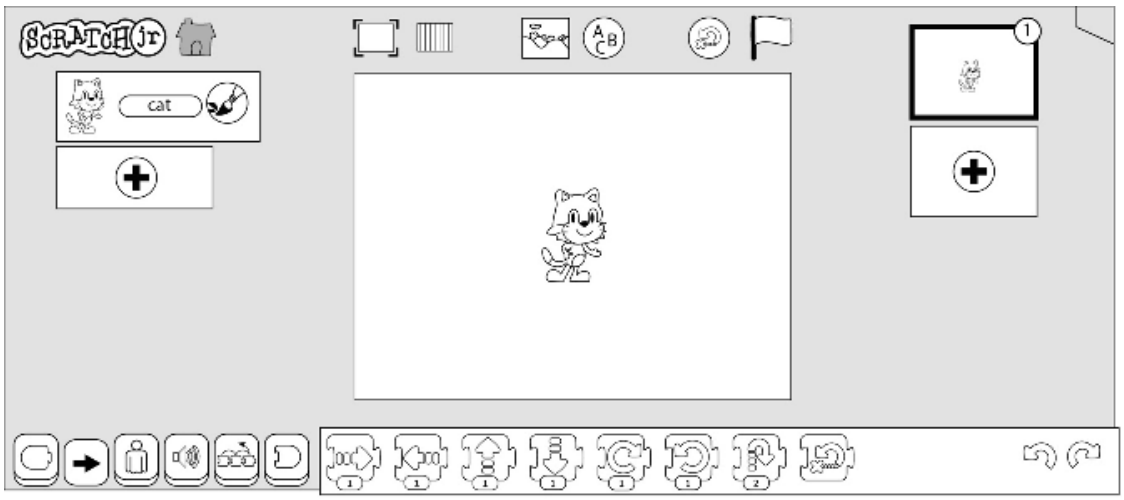
The students write down these conclusions in their science notebooks.

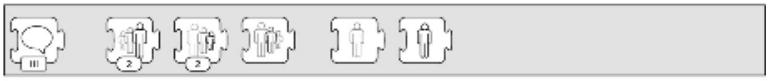
They then complete Handout 21: Using the correct color, they color in the instructions they discovered during this lesson and add a few keywords: full screen, grid, start flag, programming area, movement instructions, return to the starting position, turn, hop, events.

Teaching notes

- This handout will be used again in later lessons: the students will color in the instructions as they learn them.

HANDOUT 21
The Scratch Junior programming screen





Lesson 2 - The first episode: Choosing the hero and controlling his movements

Summary	Students tell an episode of their hero's adventure. While doing this, they learn new *Scratch Junior* functionalities (deleting a character, importing a new character, choosing a backdrop).
Key ideas *(see Conceptual scenario, page 108)*	Same as Lesson 2.1, page 139
Equipment	Same as Lesson 2.1, page 139
Glossary	Instruction, event
Duration	45 min

Starting the activity

The teacher asks the students to remember the adventures of their hero or heroine: s/he mysteriously woke up at the top of mountain and was able to make their way to a clearing. On a tree trunk, they found a message and decoded it. This message had the hero follow the river to the sea and find a treasure on the sea floor. The episode the students are going to tell today using *Scratch Junior* is where the hero follows the river.

Choosing a hero and a backdrop (ideally in pairs)

Students must do the following: Replace the cat with another character (the hero or heroine of the story), then replace the gray stage with a clearing and a river. Next, they must place the hero on the riverbank (near the orange butterfly) and control his movements so he walks along the river in the direction of the current. The hero must start by following a "stair-type" path (the aim is to prepare for the introduction of loops in the next lesson), then go right.
After a few minutes, students who successfully complete the first part of the assignment present their solution to the class on demonstration tablet: to delete the cat, you have to press on the cat rectangle in the upper left part of the screen (character and object area) for several seconds until a red X appears (see below), then tap this X.

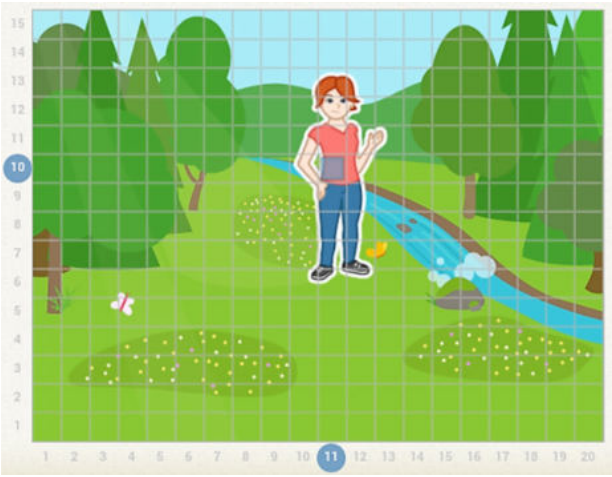
Next, to select another character, tap the icon to see the options, scroll through the images to find the hero the group wants to choose, ta hero and validate at the top right (icon).

Second grade class, Vanessa Guionie (Bergerac)

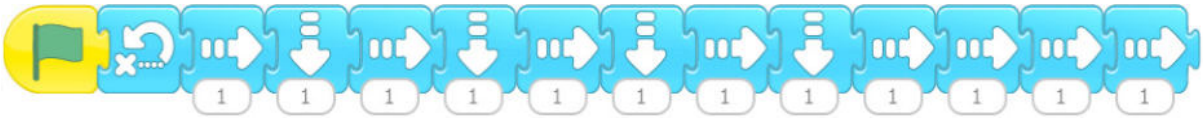
If certain groups have a hard time, students who have found the solution can show them how they did it:

- You can see the stage options by clicking on the icon  at the top of the screen, then select and validate a stage just like for the characters.
- You can place the character wherever you like on the stage with a drag and drop. This position is where the character will come back to when given the following instruction:

- To make the character walk along the river, several movement instructions must be combined in the programming area. Depending on the chosen character, its size and starting point, the instructions will differ slightly. For example, if the character is a child or teen with the default size, and the character starts in square (row 10, column 11) as here:



The program below can be used to get the expected result:



The same program can also be written in a more concise manner (however, it can still be improved, and this will be covered in the next lesson):

The teacher reminds the students how to save the programs and gives them a basic file name format, such as “AH1” for group A, Hero story (reminder: the practice programs for group A with a cat character are called AP1, AP2, etc.), then validate using the icon

Teaching notes

- If students notice the loops (in the orange instructions), tell them to remember how they did it for the next lesson and to not share it with the rest of the class.
- To ensure that students have clearly identified the main objective of the lesson, which is to learn to program, we suggest not drawing the stages or characters but to use the preset options. The scenario for Sequence 1 (pages 110 and on) was designed accordingly. However, if extending the lesson to an art or ICT class, you can use the paint editor interface in *Scratch Junior* to customize the characters and stages.

Conclusion and lesson recapitulation activity

During this lesson, students will not have covered all of the new computer concepts, but will have reinforced what they have already seen. Accordingly, they simply fill in their *Scratch Junior* worksheet, coloring in the instructions they learned during this lesson with the appropriate colors and adding a few keywords: choosing a stage, choosing a character, character list.

Lesson 3 - Simplifying a program by using loops

Summary	The students continue learning to use *Scratch Junior* by exploring the instruction «repeat...,» which is a loop. They practice anticipating what a program given to them will do, combining loops and movement instructions. Finally, they revise their initial program by replacing the repeated instructions with loops.
Key ideas *(see Conceptual scenario, page 108)*	"Algorithms" • A loop is used to repeat the same action several times. • Certain loops are repeated a specific number of times.
Equipment	Same as Lesson 2.1, page 139 Plus, for each student • Handout 22, page 152
Glossary	«Repeat...» loop
Duration	1 hour

Starting the activity

The teacher tells the class they are going to take a break from the story to learn a new type of instruction that they can use in their stories later.
They put the saved program from the end of the previous lesson on the board (either with the projector or written):

This program has only 11 basic instructions, but it takes up a lot of space in the programming area. If they want to keep telling the story, the program may become very long and complicated. The teacher tells them that today they are going to learn to identify instructions that are repeated to simplify them. To do this, they will use an orange instruction which is available after tapping on the icon  :

This instruction is like a bridge above one or more instructions. The students will have to experiment to understand what it means. They will then be ready to simplify the script that controls their hero's movements.

Activity (ideally in pairs)

The teacher gives the students two assignments (one after the other):

- They must first use the "orange bridge" instruction to make the cat move in a stair step direction (one step right, one step up, one step right, one step up, etc.). The cat must start in square (row 3, column 3) and "climb 7 steps" when you tap on the green flag.
- Once the first assignment is completed, the must then use the "orange bridge" instruction to have the cat move all around the stage three times in a row (starting from square (row 3, column 3).

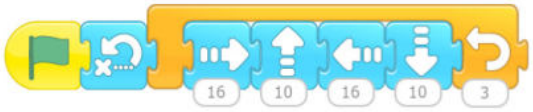
Teaching notes

- You can have students place the two previous scripts in the same programming area and have them triggered by different events: “tap on the flag” and “tap on the cat.”
- An alternative option is to create a new character and assign one script to the cat and the other to the new character, both triggered by whichever events the students want.

Group discussion

For the first challenge, a correct program that follows all directions is:

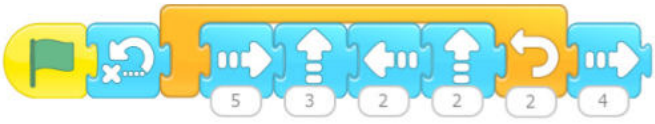
The teacher asks the students if they figured out what the orange instruction that looks like bridge does. The class concludes that this instruction requires the instruction block it encompasses to be repeated the number of times indicated on the instruction. The class gives this instruction a name, such as “Repeat...” If the students do not talk about loops on their own, the teacher reminds them about this term by referring to the previous sequence (Lessons I.5, page 129 and I.6, page 134). The teacher explains that they can include a loop each time the instructions are repeated.
For the second challenge, a correct program that follows all directions is:



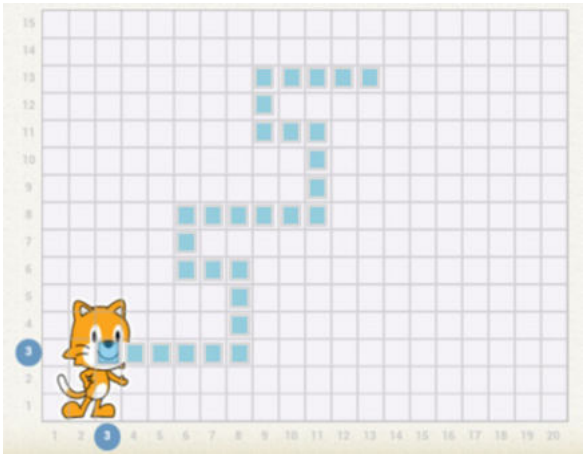
The students save their second practice program as “AP2” (for class group A) and “BP2” (for class group B).

Exercise (individually within small groups)

The teacher puts up a simple program on the board using loops and movement instructions:



They ask the students what the character will do if they start the program. The students must trace the character’s movements on a grid (Handout 22, page 152).
During the group discussion, the students compare their answers (especially the destination row and column), and check them as a class by launching the program. The path followed by the cat is the following:



Simplifying the program controlling the hero's movements (ideally in pairs)

To finish the lesson, the students apply what they learned to simplify their program from Lesson 2.1 (page 139), which controls the hero's movements. The original program (given above) becomes:

The teacher reminds the students to save their modified program, without changing the file name (e.g., AH1 for group A).

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson.

For younger students:

- *A loop is used to repeat the same instruction several times.*

For older students:

- *A loop is used to repeat the same instruction several times.*
- *Certain loops are repeated a specific number of times.*

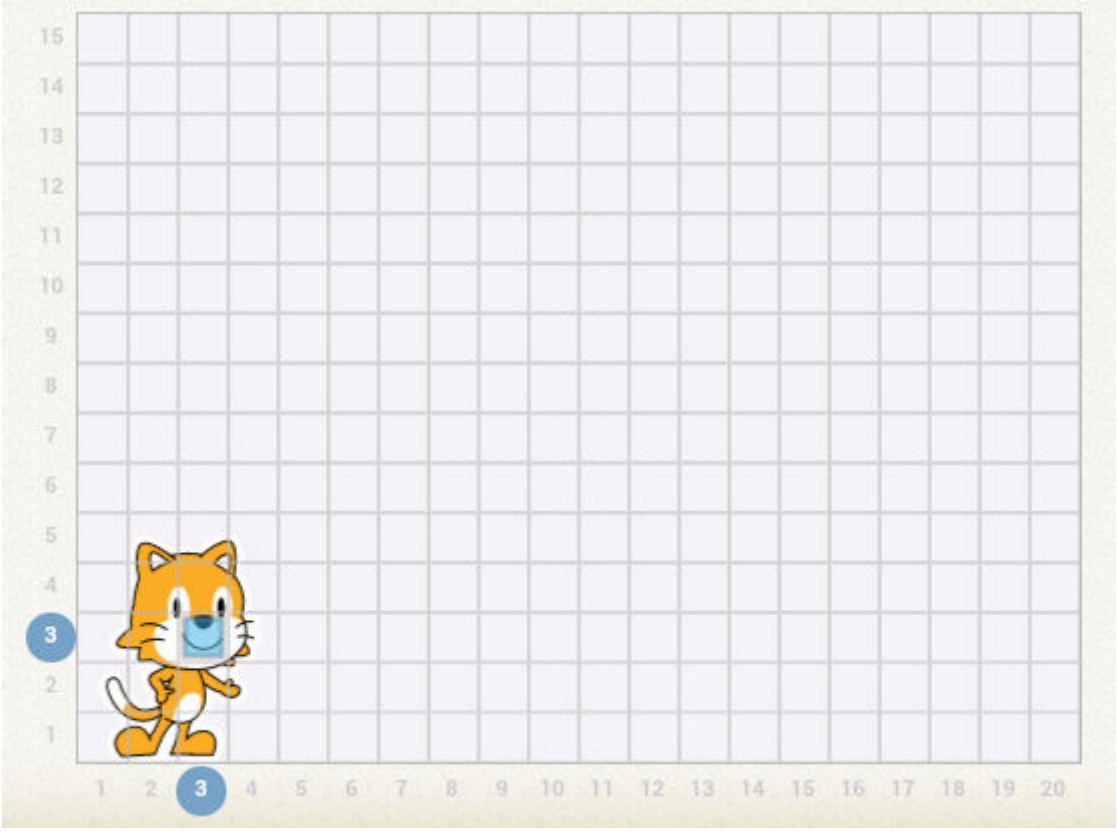
The students write down these conclusions in their science notebooks.
The students then complete their *Scratch Junior* worksheet: they color in the instruction they learned in this lesson in orange and add a few words: "Repeat..." loop, number of repetitions.

HANDOUT 22
Understanding a program with a loop

Instruction

The cat is in square (row 3, column 3). It follows the instructions of the following program:

Draw its entire path on the grid below and circle the square where he ends up:



Lesson 4 - Coordinating several scripts

Summary	Students tell a new episode of their hero's adventure, with more autonomy than in the first lessons. They discover new functionalities in *Scratch Junior* and deepen their understanding of what a sequence of instructions and a program are.
Key ideas *(see Conceptual scenario, page 108)*	«Algorithms»: • Basic instructions can be performed one after the others. • Instructions can start at the same time if they are triggered by the same event.
Equipment	Same as Lesson 2.1, page 139
Glossary	Sequence of instructions
Duration	45 min

Starting the activity

The teacher tells the students that the hero has arrived at the sea after following the river. Today, the students are going to program another episode of his adventure, where he uses a submarine to find the treasure at the bottom of the sea.

Teaching notes

- *Scratch Junior* does not have a submarine in its preset characters. The students can replace it with a marine animal, such as a seahorse (which we have done here), or you can use time during an art or ICT class to let them draw their own submarine using the integrated paint editor tool in *Scratch Junior*.

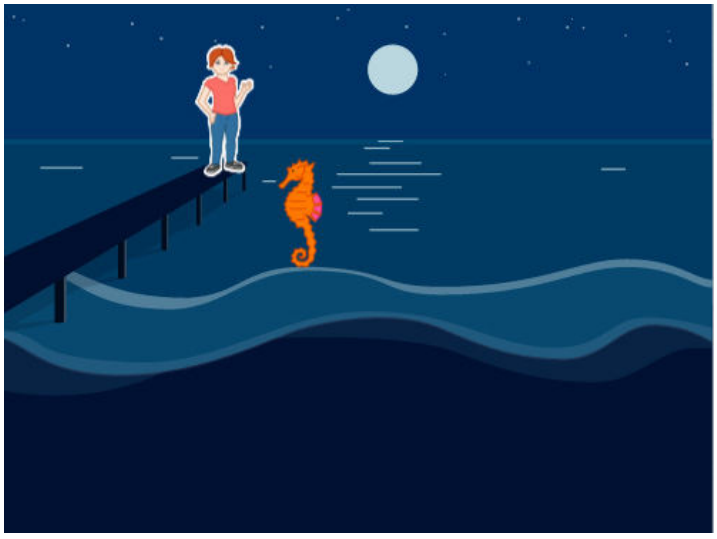
Activity (ideally in pairs)

The teacher gives the students their programming assignment: In the same program from the last lesson (AH1 or BH1, depending on the group), they must add a new stage (the characters are listed on the left, stages on the right) – the seashore, at night, with a large dock. The students must import two characters: the same hero as in the clearing and a seahorse.

Using this stage, they must then:

- Place the hero on the dock, near the edge of the water.
- Place the seahorse in the water, near the other side of the dock, but hidden (this instruction can be found in the list of purple instructions).
- Control the hero's movements to the end of the dock.
- Make the seahorse appear and come towards the hero when the hero gets to the end of the dock.

When the programs have been executed, the scene should look like this:



The teacher lets students play around with the software, offering as little help as possible so problems emerge.

Second grade class, Vanessa Guionie (Bergerac)

If necessary, show certain commands to the class using the demonstration tablet, but the first steps are very similar to what students already saw in Lesson 2.1 (page 139) and should not cause too much difficulty.

To add a new stage, click on the icon on the right side of the screen in the stages area, then select and validate it. To delete the cat that initially appears in the scene and add two new characters (the hero and seahorse), follow the same steps as in Lesson 2.2 (page 146). To position the characters, use a finger to drag them where you want them to go.

Once the character is added to the program, you can have it appear and disappear whenever you want. To do this, use the following two pink instructions: Generally speaking, the students will notice that the pink instructions control the appearance of the characters.

The main difficulty lies in having the seahorse appear when the hero gets to the end of the dock. There is no preset event called "the hero gets to the end of the dock" to trigger the program that makes the seahorse appear. The students may suggest several options:

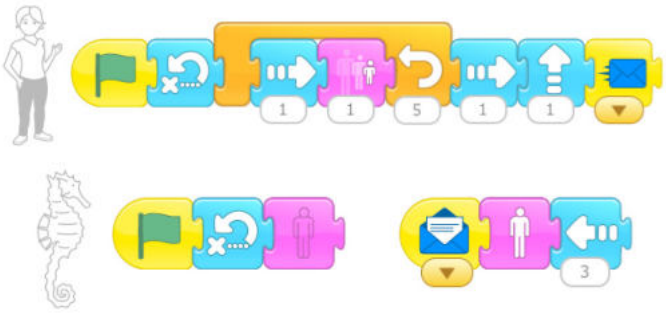
- One is to trigger the program that makes the seahorse appear and the program that controls the hero's movement with the same event, but adding a delay to the start of the seahorse program with the following orange instruction:

The delay can be set by modifying the "10" value (this is not 10 seconds, but the time it takes to make 10 steps, so you have to play around with it to get the seahorse to appear at the right time).
The teacher tells the students that if they modify the hero's program, they'll have to adjust the delay as well, which takes more time.

- Another option is to control the hero's movement using a "finger tap" command, and to control the seahorse with a "tap on the green flag" command.
- The teacher gives students a final option: the program to make the hero move sends a message at a particular time, and when the message is received, it triggers the seahorse to appear. The yellow instructions below will give you this result.
"Send blue message" event (there are five other possible message colors):

Trigger a script with the "Blue message received" event:

You will get, for example, the following programs for the hero and seahorse, respectively:



Note that tapping the green flag triggers two scripts: moving the hero and hiding the seahorse. Also note that the messages are sent "publicly." In other words, several scripts can be triggered by sending the same message.

Teaching notes

- It is rather difficult to make the character walk precisely along the dock. Tell the students that they should do their best but that it does not need to be perfect.

Group discussion

During the group discussion, the teacher goes over any difficulties students had, particularly regarding timing.

They then switch to full screen mode to show the class that you can go from one episode of the story to another by using the small arrows   . To launch each episode, you have to tap the green button. Finally, the hero is not always where he should be when you get to screen 2. The teacher shows a red instruction that automates moving from one stage to another:

It should be placed as the last instruction for the hero, on stage 1, and command the transition to stage 2. The students add this instruction to their program, then save it as AH2 (group A) or BH2 (group B).

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

For younger students:

- *Scripts are triggered by events. Within a program, a single event can trigger several scripts at once.*

For older students:

- *Within a script, basic instructions are executed one after another, from left to right.*
- *Within a program, several scripts can start at the same time, if they are triggered by a single event.*

Students write down these conclusions in their science notebook. They then complete their *Scratch Junior* worksheet by coloring in the instructions they learned during the lesson and writing down the key words: pause, appearance and disappearance, sending and receiving messages, changing stages.

Lesson 5 - Predefined loops and infinite loops

Summary	Students tell a new episode of their hero's adventure. They reinforce the key ideas from the previous lessons, namely predefined loops, and learn about infinite loops.
Key ideas *(see Conceptual scenario, page 108)*	«Algorithms»: • Certain loops are repeated forever.
Equipment	Same as Lesson 2.1, page 139
Glossary	Infinite loop
Duration	1 hour

Starting the activity

The teacher reminds the students that the hero, who cannot reach the treasure he can see at the bottom of the sea on his own, must guide a submarine, represented here by a seahorse. The seahorse brings the treasure to the surface. Today, the students are going to program this episode of the hero's story.

Activity (ideally in pairs)

The teacher gives students their programming assignment: in the same program as the last time, they must add a new stage that appears once the seahorse is visible in the sea. In this scene, they must add a "decorative" animal of their choice. It will not participate directly in the story and its movements will repeat through the scene. For a script to repeat forever, they must place the following red instruction at the end of the script:

The students must then make the seahorse appear and go to get the treasure by making back and forth movements to the left and right, descending gradually. The treasure is in a chest at the bottom of the sea. When the seahorse touches it, both go back up to the surface at the same time. To trigger a script when two characters touch, the following yellow instruction must be used:

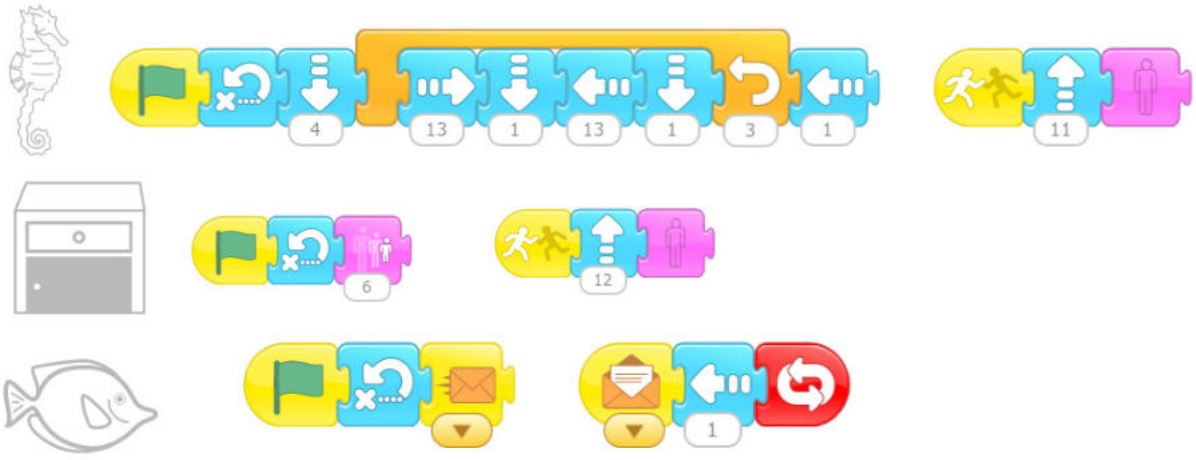
As during the previous lesson, the teacher lets students play around with the software, offering as little help as possible so problems emerge. If necessary, you can show several commands to the class using the demonstration tablet, but students should be increasingly independent. However, you should remind them to save their programs with the names AH3 or BH3 (depending on their group).

Teaching notes

- You can pre-program each tablet with part of the lesson and have students complete or modify the program.

Group discussion

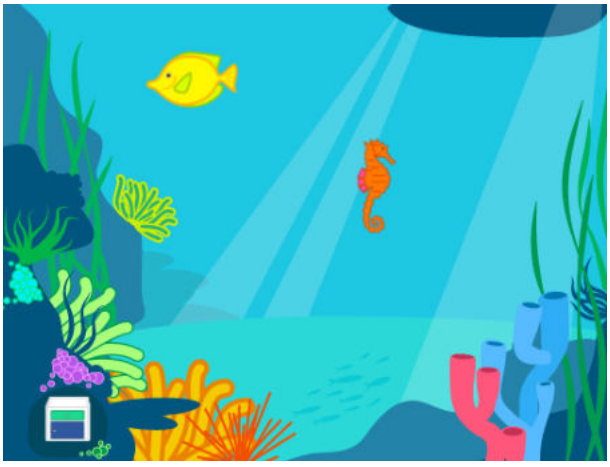
During the group discussion, the teacher goes over any difficulties students had. For example, if the seahorse's ascent is triggered when it touches another character, the "decorative" animal must not touch the seahorse while it descends. The students must choose the seahorse's positions and movements so as to avoid the two coming into contact. Here are a few programs for each character (seahorse, treasure chest and decorative fish) that follow the directions, but students will come up with a variety of solutions.



The initial positions that are compatible with these programs are:

- Seahorse: (row 14, column 4)
- Treasure chest: (row 2, column 3)
- Decorative fish: (row 12, column 18)

An intermediate view of the scene looks like this:



Teaching notes

- Some students will use instructions the class has not yet seen (especially as the *Scratch Junior* worksheet provides a preview of all commands). At this point, when they begin to have a good understanding of the software, it is important to let them express their creativity. But to help them continue learning, the teacher should encourage them to regularly test their programs. It is much more difficult to detect a bug in a program written as a block without tests than in a program that has been tested throughout the writing process.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *A loop is used to repeat the same instruction several times.*
- *Certain loops are repeated forever.*

The students write down this conclusion in their science notebook and complete the *Scratch Junior* worksheet by coloring in the new instructions they learned and writing down a few key words: “touch another character” event, infinite loop.

Lesson 6 - Adding recorded dialogues to the program

Summary	Students learn to record character dialogues.
Key ideas *(see Conceptual scenario, page 108)*	All previously covered key ideas
Equipment	Same as Lesson 2.1, page 139
Duration	45 min

Starting the activity

The teacher tells the students they are going to add sound recordings to their stories.

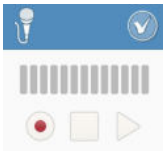
Teaching notes

- To facilitate this lesson, the text can be created ahead of time by the students during a writing class. The recordings can be done during an ICT class (rather than during a science class).

Recording procedure (as a class)

The teacher shows the students how to record a message for a particular character: select the character from the left side of the screen, then go to the green instructions using the icon and tap on the command to create a new recording:

The following mini window opens:



Recording begins as soon as you tap on the red dot and ends when you tap on the gray square (which then flashes orange). You can immediately listen to the recording by tapping on the gray arrow, and start again if necessary by tapping on the red dot. When you are happy with the recording, validate using the usual icon at the top right of the mini window.
Now, the recorded sound will appear in the list of sounds for the active character (here, it is sound no.1):

You can record all sounds for each character separately, making sure to save each recording for the right character. We recommend limiting the number of dialogues, recording all of the sounds first, then adding them to the programs.

Timing the dialogues (ideally in pairs)

Once the recordings are all done and the program has been saved, the students must create a dialogue on their own. They save the new program with the same name as before, either AH3 or BH3.

Teaching notes

- There are several ways to coordinate the dialogues, or more generally, the actions of several characters. The most intuitive way is to add the pause instructions and adjust the duration as you go (the duration of each pause will depend on those in the audio recordings).
- However, adding pause instructions to suit the program timings can be problematic as the pause durations do not adjust automatically if a recording is changed or if instructions are added in a program. Another more comprehensive approach, which was seen in Lesson 2.4 (page 153), consists in sending messages.
- While we would like students to figure out this solution, it is important to let them start by adding pauses if they think of this first.
- We do not recommend adding text to the story as typing in text on a tablet can be cumbersome. However, if the teacher prefers adding text over recordings, the instruction blocks to do so are pink.

Conclusion

To conclude the lesson, students complete the worksheet for *Scratch Junior*, coloring in the instructions they learned during this lesson and adding key words: new sound, pre-recorded sound.

Lesson 7 - Producing the final episode autonomously

Summary	Students work on their own to tell the last episode of their hero's adventure. They cover the key ideas from the entire sequence and finish their program.
Key ideas *(see Conceptual scenario, page 108)*	All previously covered key ideas
Equipment	Same as Lesson 2.1, page 139
Duration	1 hour

Starting the activity

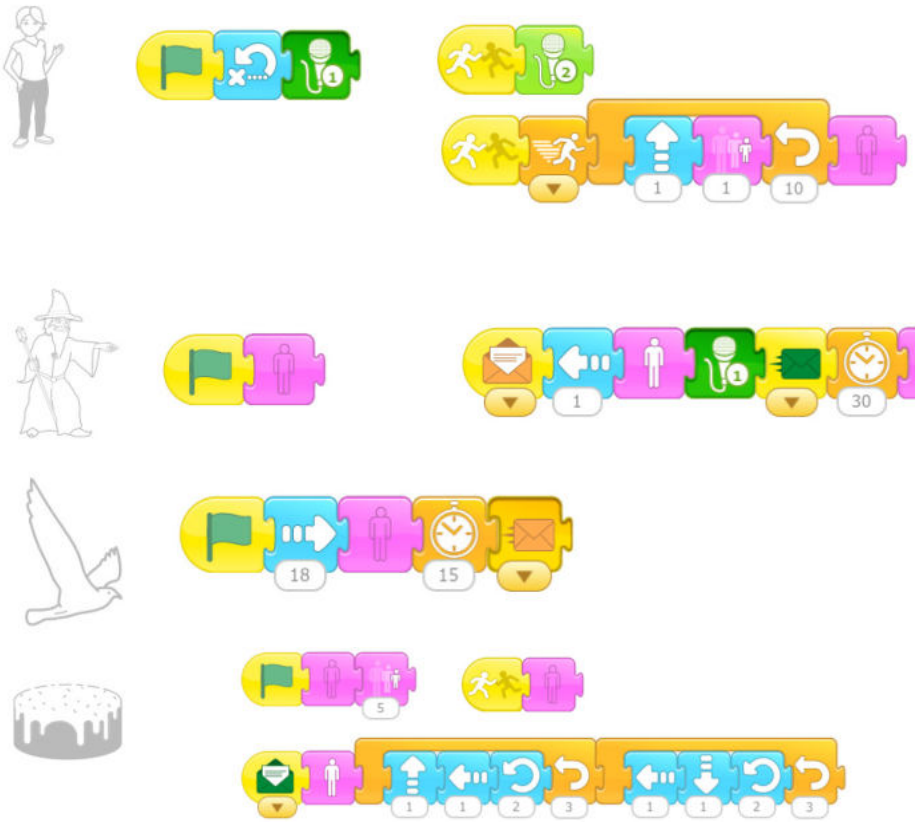
The teacher tells the students that they can tell the last episode almost however they like with *Scratch Junior*. There is only one condition: they must use events to coordinate their characters and use loops. The teacher suggests that they try and not be overly ambitious to start with and to add elements to their story if they have time at the end. In particular, any sound recordings should be done last, with students focusing on structuring their programs first.

Creating the episode (ideally in pairs)

The students talk in pairs about the essential elements to include in the last episode. Which characters? Which stage? Which events to trigger the characters' actions?

Programming the episode

After approval from the teacher, who checks with that the students that their plan is realistic, the students program the last episode. The teacher walks around to all the groups to make sure they are making progress.
The teacher checks that the students remember to link stages 3 and 4.
Here are a few examples of programs that tell the final episode of the story. The selected stage is a beach at sunset (orange colored). The story is a bird that goes to find the magician, who conjures up a cake and sends it to the hero. The hero eats the cake and slowly gets farther away (he returns home).
These programs are already rather complex (only older students will likely manage to produce them without help):



The initial positions that are compatible with these programs are:

- Hero: (row 5, column 3)
- Magician: (row 6, column 17)
- Bird: (row 12, column 2)
- Cake: (row 9, column 10)

Teaching notes

- With *Scratch Junior*, the positioning, disappearance and scaling of characters is rather slow. The characters will appear differently than desired for a couple seconds when the program is launched and when stages change. The more advanced version, *Scratch,* will let students in grades four and up achieve a better result.
- This program can be enhanced with sound messages, sound effects, etc.

Conclusion: Class presentation

To conclude the lesson, each group of students shows their final episode to the class. The students talk about the programs they are most proud of or those that were most difficult to figure out. They can also share any difficulties they could not solve and see if the class can help find a solution.

Further study

Now that students have learned to program on *Scratch Junior*, they can use their skills to create other projects during the year, such as animated cards for Christmas, Mother's Day, etc.

Sequence 2a: Alternative with Scratch

This sequence is an alternative to Sequence 2 (pages 139 and on) and deals with **programming** an animation to tell the hero's adventure from Sequence 1.

Scratch instead of Scratch Junior?

As already discussed above, for grades one through three, it is preferable to learn to program on tablets using *Scratch Junior* (see explanations on page 137). However, if your school does not have tablets but does have computers, you can use ***Scratch***[12] (available at no cost for Windows, Mac and Linux operating systems).

An intermediate sequence for grades three and up

Because this is not our recommended solution, we have created this lesson plan with fewer details. However, it has been successfully tested with several third grade classes (it should not be done with first or second grade classes). Students should already have keyboard and mouse skills before starting this work.

This sequence draws from the scenario in the Level 2 activities (Sequence 2, page 139), but uses the tools recommended for Level 3 activities (Sequence 2, page 229).

The teacher can opt for one of the following:

- Have the class tell the hero's entire story in *Scratch* (as we suggest in Sequence 2 using *Scratch Junior*)
- Have students tell only one episode of the hero's story
- Have different groups each tell a different episode. The programs created by the different groups can be linked together to tell the entire adventure. However, this can only be other by copying the different programs "by hand."

Whichever option is chosen, the class will start with the same introductory steps in *Scratch*, which are a review of the first lessons described in detail for grades four and up (pages 229 and on).

Working in half-groups

Ideally, there should be one computer for each student pair. To do this and make it easier to manage the class during the programming activities (during which the teacher will have to work closely with students), half the class can work independently on exercises not requiring assistance from the teacher while the other half of the class can work on the project. Then, the teacher can have the groups switch roles (A and B in the following lessons).

12 *Scratch* is available either online (without prior installation, but this requires a good internet connection, at the address: https://scratch.mit.edu) or offline (requiring prior installation, after downloading the software at the address: https://scratch.mit.edu/scratch2download).

Preparing the working environment

In order to save time, it is good to prepare the working environment in advance:

- *Scratch* must be installed on all computers (or accessible online)
- A **shortcut to** *Scratch* should be placed on the desktop
- Similarly, a dedicated **folder** for the project (and the class) should be easily accessible, either on the desktop or on a USB flash drive for the group.

Doing the project yourself first

It is very important for the teacher to spend two hours, during their lesson preparation time and before the first programming lesson, to get familiar with *Scratch* and do the same tasks the students will do during the project. Otherwise, they may not be able to help the students when they need it.
To be prepared, simply follow the lessons in this sequence.

	Lesson	Title	Page	Summary
	Lesson 1	Introduction to the Scratch programming environment	166	Students are introduced to *Scratch*, an easy-to-use graphic programming environment.
	Lesson 2	Making a character move	168	Students explore the ways to control a character's movements.
	Lesson 3	Choosing the hero and controlling his movements	170	Students tell the first episode of their hero's adventure, where he comes out of the forest and follows the river to the sea. During this time, they cover the key ideas from the previous lesson (set of instructions, event), learn about the idea of initialization and use predefined loops («repeat…»).
	Lesson 4	Programming several sprites	174	The students tell another episode of the hero's adventure, where he sees the treasure at the bottom of the sea and gets help to retrieve it. To do this, students learn to load a new stage, add a sprite and cover the key programming ideas from the first two lessons.
	Lesson 5	Coordinating the first two episodes	177	Students must figure out how to make the two first episodes continue one after the other. To do this, they learn the key idea of message: a message can be sent during an instruction, and when the message is received, it can trigger one or more instructions.
	Lesson 6	Different types of loops	179	Students tell the next episode of the hero's adventure: the octopus goes to the bottom of the sea to get the treasure and bring it back to the surface. They reinforce the key ideas from the previous lessons, namely predefined loops, and learn about infinite loops.
	Lesson 7	Producing the final episode autonomously	181	Students work on their own to tell the last episode of their hero's adventure. They cover the key ideas from the entire sequence and finish their program.

Lesson 1 - Introduction to the Scratch programming environment

Summary	Students are introduced to *Scratch*, an easy-to-use graphic programming environment.
Key ideas *(see Conceptual scenario, page 108)*	"Machines" and "Languages" • We can give a machine instructions by using a special language called a programming language, which can be understood by both people and machines. • An «algorithm» is a method to resolve a problem. • A program is an algorithm in a programming language.
Equipment	For the class • (Recommended) A computer on which the *Scratch* software has been installed and a video projection system. For each student pair • A computer on which the *Scratch* application has been installed[2]. For each student • Handout 32, page 242 (this handout will be used again in the other lessons)
Glossary	Program, script, character, stage, instruction, event
Duration	1 hour

Foreword

As explained earlier, the first few lessons of this Level 2 *Scratch* sequence reviews certain activities described in detail in the Level 3 activities. We will not rewrite everything here and will frequently refer back to the relevant Level 3 lessons. Before continuing, we strongly recommend that teachers read at least the first three lessons of the Level 3 *Scratch* sequence (and to do the activities themselves ahead of time).

The general scenario of this Sequence 2a is very similar to that for Level 2, Sequence 2, except that we use *Scratch* instead of *Scratch Junior*. Where Scratch instructions are concerned, we will again refer back to the relevant lessons rather than rewrite everything here.

Teaching notes

- You learn to program by programming, not by watching someone program. It is interesting to consider the same problem in pairs (probably more so than to program alone), but it is important to be active. We therefore recommend placing the students in small groups in front of computers (ideally two students per machine) and asking them to "switch over" (pass the keyboard and mouse to their neighbor) every 10-15 minutes.
- If possible, we recommend organizing the class in half-groups, so as to avoid having too many pairs to manage at once. While half the class works on *Scratch*, the other should do something else independently.
- If possible, two *Scratch* lessons should be organized weekly, at least at the beginning.

- This step of getting to know *Scratch* is deliberately very directive (it starts with a demonstration by the teacher). It is the only step presented in this form. All pairs will have to carry out a series of basic tasks. At the end of each task, a group discussion ensures that everyone has understood and knows what to do. The other steps will be less directive, as students become more independent and progress at their own pace.
- To save time, switch on the computers before the lesson begins.

Starting the activity

The teacher explains to the students that they are going to use a computer to tell the main episodes of their hero's adventure. To do this, they must program the computer, i.e., tell it what to do. They will have to use a special language, called a programming language, which can be understood by both the students and the computer. The language they are going to use is called *Scratch*. Today and in the next lesson, they will get familiar with *Scratch*, and from the third lesson, they will start programming the story.

Opening Scratch and getting to know the interface

After a quick presentation of *Scratch*, the students explore the software on their own, then do a few simple exercises. See Level 3 Lesson 2.1 of the *Scratch* Sequence (page 234), and particularly the following tasks:

- Activity 1: Demonstration of *Scratch*, showing the different elements (stages, sprites, backdrops, the script tab) and basic actions (sliding instructions into the programming area, combining instructions as scripts, deleting instructions). The teacher gives students Handout 32, which the students will color in as they progress through the sequence.
- Activity 2: Exploring *Scratch* (page 238). The students learn about the various instruction categories ("motion," "looks," "events," "control," etc.).
- Activity 3: Getting comfortable with the interface through seven short exercises (page 239).

Lesson 2 - Making a character move

Summary	Students explore the ways to control a character's movements.
Key ideas *(see Conceptual scenario, page 108)*	Same as for the previous lesson
Equipment	Same as for the previous lesson Plus, for each student: • Handout 32, page 242
Glossary	
Duration	1 hour

During the previous lesson, students worked with the *Scratch* interface to become more comfortable with its features. They will now learn how to move the sprites where they want them by practicing on the cat sprite.

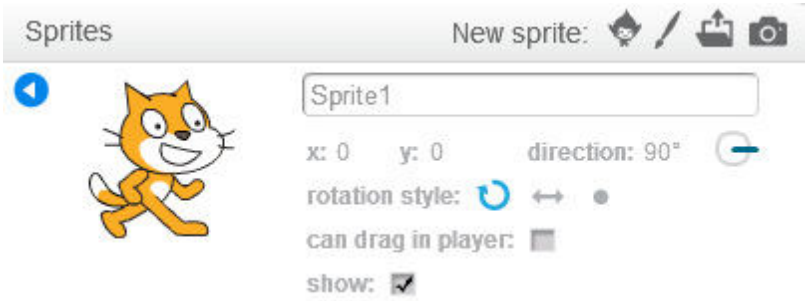
Teaching notes

- At this stage, students will still need assistance. As they use the software, they will become more independent and each pair will advance at their own pace.

Challenges: Controlling the cat's movements (ideally in pairs)

The previous exercises will have taught students how to make the cat move to the right. Now, they need to learn to make it move in any direction.
First, students should have the cat move to the left (see Activity 1, page 236). Having the cat move to the left is slightly more difficult as students must have the cat turn left, then move forward.
They should work independently and experiment while the teacher goes from group to group to help when needed. The teacher can guide them by suggesting they find an instruction to "point in direction".

The students test the effect of a block of two instructions, "point in direction" and "move 10 steps", depending on whether the "point in direction" instruction says 90, -90, 0 or 180.
The teacher should also encourage students to click on the sprite's information icon (the "i" in a blue circle) and to try out the three rotation options.



They can turn the sprite by using the blue “direction” line. Whenever the “point in direction” instruction is used, this blue line is updated. You can exit the information mode by clicking on the small blue “back” arrow in the blue circle.

The students can then figure out how to complete the challenge: by combining the “point in direction -90” and “move 10 steps” instructions.

With these new tools, the students can now answer the initial question of how to move the cat in any direction (see Activity 2, page 238).

Third grade class, Emmanuelle Wilgenbus, Antony

Conclusion and lesson recapitulation activity

During this lesson, students will not have covered all of the new computer concepts, but will have reinforced what they have already seen. Accordingly, they complete their *Scratch* worksheet (see the previous lesson) by using the appropriate color to color in the instructions they learned during this lesson.

Lesson 3 - Choosing the hero and controlling his movements

Summary	Students tell the first episode of their hero's adventure, where he comes out of the forest and follows the river to the sea. During this time, they cover the key ideas from the previous lesson (sequence of instructions, event), learn about the idea of initialization and use loops.
Key ideas *(see Conceptual scenario, page 108)*	*"Algorithms"* • A loop is used to repeat the same instruction several times. • Certain loops are repeated a specific number of times.
Equipment	Same as for the previous lessons The teacher should have prepared the workstations by copying the *Scratch* files (extension .sb2) available on the project website (see page 342).
Glossary	Initialization, «Repeat...» loop
Duration	1 hour

Starting the activity

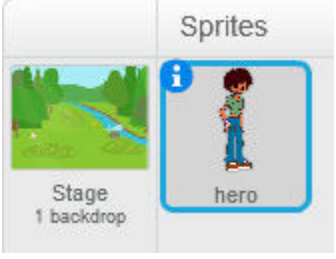
The teacher asks the students to remember the adventures of their hero or heroine: s/he mysteriously woke up at the top of mountain and was able to make their way to a clearing. On a tree trunk, they found a message and decoded it. This message had the hero follow the river to the sea and find a treasure on the sea floor. The episode the students are going to tell today using *Scratch* is where the hero follows the river.

Choosing a hero and controlling his movements (ideally in pairs)

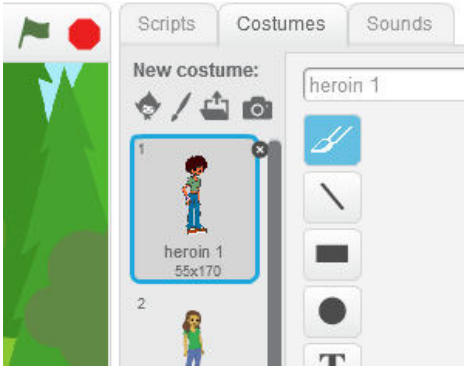
Students open the prepared file "2-2a_river.sb2" using *Scratch*. The stage shows a forest, a clearing and a river so the students can program the episode where the hero follows the river. A sprite is already there: the hero. He is positioned on the left side of the screen at the edge of the forest.

The students begin by choosing the main character of the story from among three heroines and three heroes. To do this, they must:

- Select the "hero" sprite by clicking on him in the "Sprites" area:



- Click on the “Costumes” tab to bring up the six possible characters

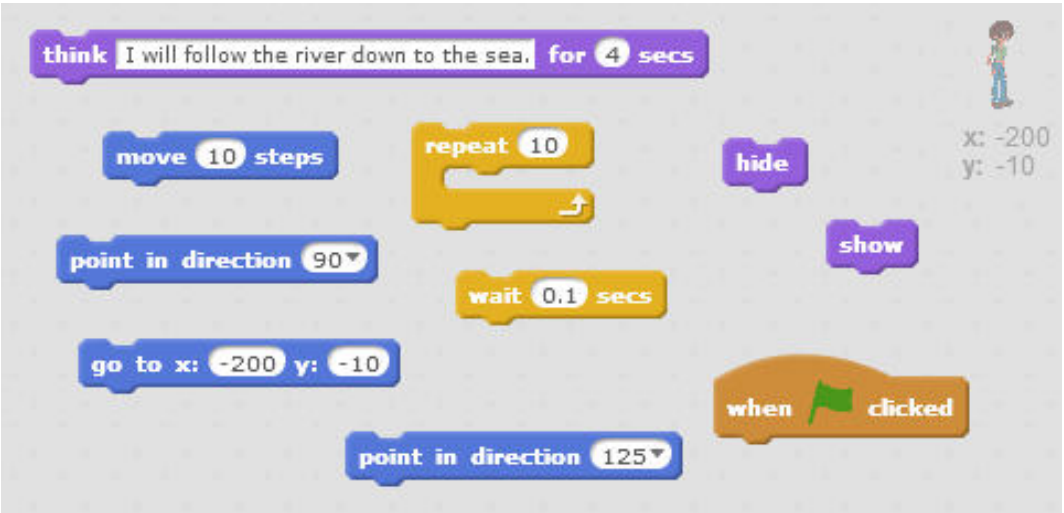


- Drag their chosen costume to the first position and delete any costumes they do not need using the small gray X that appears when they are selected.
- Go back to programming by clicking on the “Scripts” tab.

Once they have completed this task, the students must control the hero’s movements to make him go to the river and follow it to the lower right corner before he then disappears. The movements should be triggered by a click on the green flag.

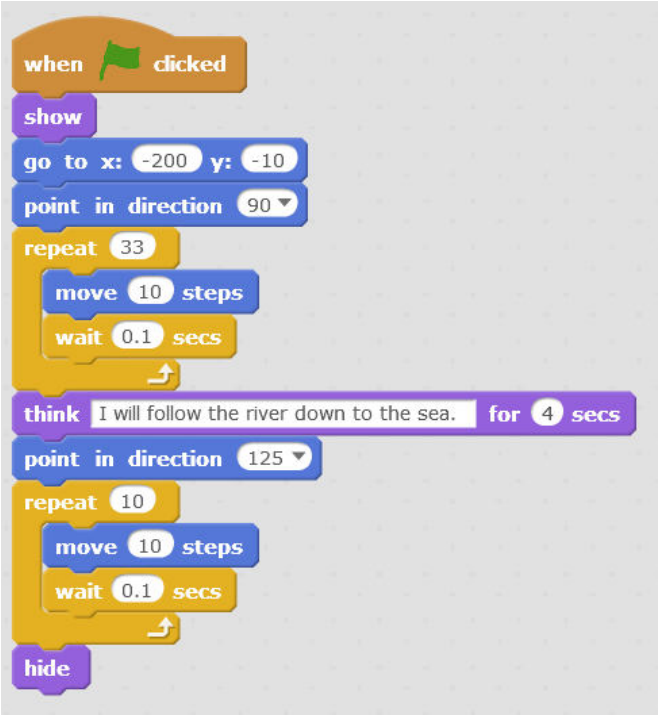
The students should use only the instructions that are already in the character programming area (there may be copies of some of them, in which case they will need to find these instructions from the list in the center of the screen, or do a right-click copy). They do not have to use all the instructions.

The instructions they can use are the following (the values can be changed):



Group discussion

One student pair presents their solution on the demonstration computer. Based on this presentation, the class discusses the difficulties they encountered and share their solutions. There may be as many suggestions as there are pairs, but one option is as follows:



Teaching notes

- We chose six characters ahead of time instead of having students choose from the vast selection of *Scratch* sprites so that the characters all have similar sizes, the same starting direction and to spend more time on programming rather than endless conversations about which character to choose. However, it is possible to let students create their own character during art or ICT class using *Scratch* editing tools, or modify an existing character.
- Students will often think they can control their hero's movements with a single instruction, such as "move 400 steps." But they will see that the movement is nearly instantaneous and not realistic. They will then come up with the idea of making small moves, perhaps using the "repeat..." instruction. However, if students do not use the instruction "wait...secs" between two successive moves, the problem remains the same.
- Students generally think to use the "hide" instruction so that the character disappears at the end of the program, but when they try and relaunch the program, their character does not appear. It stays in "hidden" mode, because no instruction has been given to appear. This problem is easily fixed by adding a "show" instruction at the start of the program. This instruction is an initialization, giving the initial situation, such as the instruction "point in direction 90."
- Similarly, if the students place their character on the edge of the forest by clicking and dragging it onto the stage, without adding this initial position to the program with an instruction, the character will leave from this position the first time, but will not go back when the program is restarted. This is the point of adding the instruction "go to x: … y: …" at the start of the script. This instruction is also an initialization. Note that manually positioning the character does have its use: the instruction "go to x: … y: …" uses the values x and y corresponding to this position, and there is no longer a need to place this instruction in the programming area to use it.
- The teacher guides the discussion to the purpose of the instruction "repeat…" and

brings up the term "loop," discussed in Lessons I.5 (page 129) and I.6 (page 134) in the previous sequence. The teacher asks where the repeat instructions must be placed: they must be placed within the "repeat…" instruction. The teacher points out that in this type of loop, you have to decide ahead of time how many times you want to repeat the instructions: the number of repetitions are defined in advance, or predefined.

The groups can modify their initial suggestion after the group discussion, such as by using a loop if they had not already. The teacher also shows the class how to save the program (File, Save As, location) and imposes the name: A1 or B1 for this first episode, depending on whether the students were in half-group A or B.
The file "2-2a_river_correction_lesson_3.sb2" provides an example of the result after this lesson

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *Certain instructions are used to indicate an initial value: these are initialization instructions, or initializers.*
- *A loop is used to repeat the same instruction several times. Certain loops are repeated a specific number of times.*

The students write down these conclusions in their science notebooks. They then complete their *Scratch* worksheet by using the appropriate color to color in the instructions they learned during this lesson and a few words: predefined loop, pause.

Lesson 4 - Programming several sprites

Summary	The students tell another episode of the hero's adventure, where he sees the treasure at the bottom of the sea and gets help to retrieve it. To do this, students learn to load a new stage, add a sprite and cover the key programming ideas from the first two lessons.
Key ideas *(see Conceptual scenario, page 108)*	"Algorithms" • Instructions can start at the same time if they are triggered by the same event.
Equipment	Same as for the previous lessons
Glossary	
Duration	1 hour

Starting the activity

The teacher asks the students what happens to the hero once he has followed the river to the sea. They remember that the hero sees the treasure at the bottom of the sea and commands a submarine to collect the treasure. The teacher tells the students they will program this episode today.

Changing a stage and adding a sprite (as a class, then in pairs)

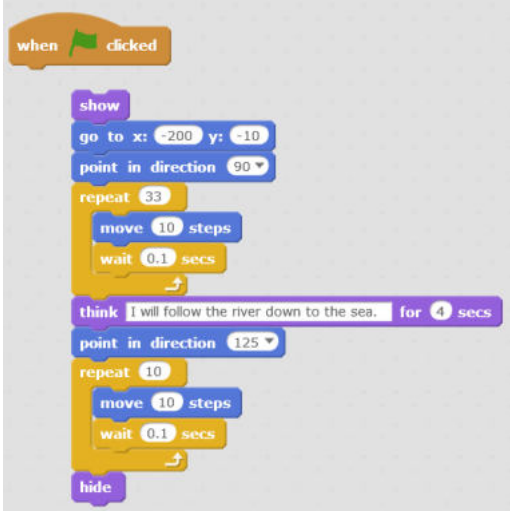
First, the teacher shows the class how to load a new stage on the demonstration computer (see Level 3, Lesson 2.2, Activity 2, page 244). Click on the current stage (the river), click the "backdrops" tab and load a backdrop from a file (the backdrop file we supply is called "landscape_dock"). The students repeat these actions on their own computers.

The teacher then shows them how to load a new sprite from the *Scratch* library. Since there is no submarine, the teacher can choose an octopus, for example, with approval from the students. The students also load the "octopus" sprite on their computers.

The teacher shows the class that the stage and each sprite each have their own programming area. To create a program for a particular sprite (or for the stage), select the sprite (or stage) by clicking the Sprites area (or the Stage area).

The teacher explains that for the time being, the script for the river episode will be inactivated: to do this, separate the entire instruction block for the "When green flag clicked" trigger event, like this:

We will deal with reapplying these instructions later (next lesson). The students inactivate the script.



Programming the episode (in pairs)

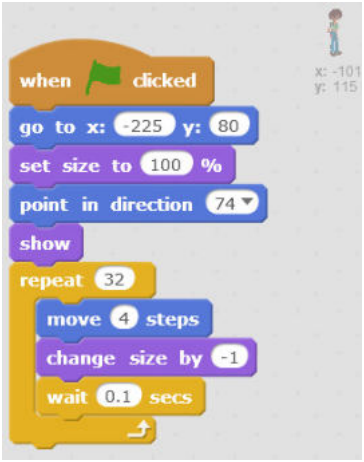
Now, the students will program the new episode: the hero must go to the end of the dock, where he meets the octopus swimming back and forth (left to right). When the hero arrives at the end of the dock, the octopus stops near him. After they talk for a while, the octopus disappears (it goes to find the treasure).

Teaching notes

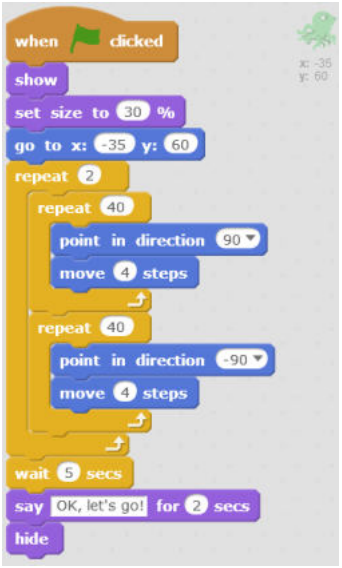
- We suggest letting students find the instructions to use themselves, because most of them have been used in the previous lesson. However, if they encounter any major difficulties, remind them of the useful instructions.

Group discussion

During the group discussion, one group of students presents their solution. Based on this presentation, the class discusses the difficulties they encountered and the solutions they found.
One possible solution is as follows, without the dialogues (to the left, the hero's program and to the right, the octopus's program, as indicated by the sprite shown at the upper right of the programming area):



Hero program



Octopus program

Teaching notes

- Few students will remember to initialize the sprite's position at the start of the program. Cover this idea again if necessary: "How did we make the hero stay visible at the start of the river episode?", "How did we make the hero stay at the edge of the forest at the start of the river episode?" To do this, simply indicate the desired values of the X and Y coordinates using the "go to x: … y: …" instruction).
- Few students will think to choose how the octopus turns. For example, it will end up upside down when it moves left.
- The teacher can introduce the instruction "Glide...secs to x: … y: …" to simplifying the octopus's program (the version above includes nested loops. This gives you:

The file “2-2a_dock_correction_lesson_4.sb2” provides an example of the result after this lesson.

Conclusion and lesson recapitulation activity

No new key ideas were covered during this lesson, but students will have learned about new *Scratch* instructions. They complete their *Scratch* worksheet by coloring in the new instructions and adding a few keywords: selected sprite, loop.

Lesson 5 - Coordinating the first two episodes

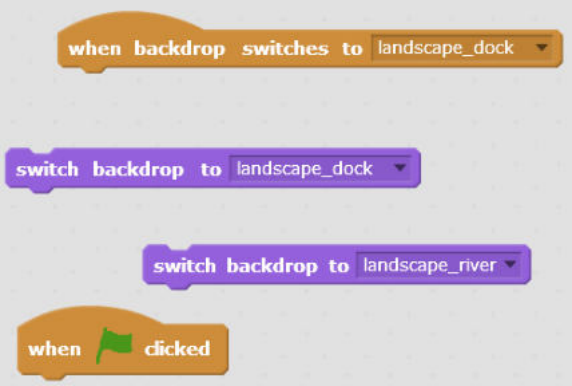
Summary	Students must figure out how to make the two first episodes continue one after the other. To do this, they learn the key idea of message: a message can be sent during an instruction, and when the message is received, it can trigger one or more instructions.
Key ideas *(see Conceptual scenario, page 108)*	"Algorithms" • Instructions can start at the same time if they are triggered by the same event.
Equipment	Same as for the previous lessons.
Glossary	Script, event
Duration	45 min

Starting the activity

The teacher reminds the students that for the time being, the hero's program (the river episode) is deactivated (no event will trigger it). However, now we want the two story episodes to follow one after the other. This is what they will be doing today.

Activity (ideally in pairs)

The teacher tells the students the instructions they must use:

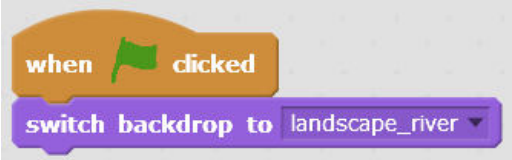


The teacher lets them decide on their own how to include them in their program to get the expected result.

Group discussion

The group discussion lets students discuss any difficulties they encountered and the solutions they found. The easiest solution is the following:

In the stage programming area, the stage is initialized for the beginning of the story:



At the end of the hero's first script (which marks the end of the river episode), add the instruction "Switch backdrop to *Landscape_dock*."
Trigger the hero's script for the river episode with the event "When green flag clicked."
Trigger the hero's and octopus's scripts for the dock episode using the backdrop switch event by placing the instruction "When backdrop switches to *Landscape_dock*" at the start of these scripts.
The initializers must also be added, namely to hide the octopus when the green flag is clicked and resizing the hero to 100%.
The file "2-2a_dock_correction_lesson_5.sb2" provides an example of the result for the first two episodes (when the river and dock episodes are connected).

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *Scripts are triggered by events. Within a program, a single event can trigger several scripts at once.*

Students write down these conclusions in their science notebook. They then complete their *Scratch* worksheet by coloring in the instructions they learned during the lesson and writing down the key words: broadcasting messages, "when I receive message" event.

Lesson 6 - Different types of loops

Summary	Students tell the next episode of the hero's adventure: the octopus goes to the bottom of the sea to get the treasure and bring it back to the surface. They reinforce the key ideas from the previous lessons, namely predefined loops, and learn about infinite loops.
Key ideas *(see Conceptual scenario, page 108)*	"Algorithms": • Certain loops are repeated forever. • Certain loops are repeated until a condition is met.
Equipment	Same as for the previous lessons
Glossary	Infinite loop
Duration	1 hour, plus 30 minutes to connect episodes 2 and 3.

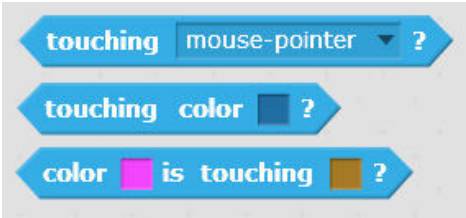
Starting the activity

The teacher reminds the students that the hero, who cannot reach the treasure he can see at the bottom of the sea on his own, must guide a submarine, represented here by an octopus. The octopus brings the treasure to the surface. Today, the students are going to program this episode of the hero's story.

Activity (ideally in pairs)

The teacher gives students their programming assignment: in the same program as the last time, they must add a new scene that appears once the octopus is visible in the sea. In this scene, they must add a "decorative" animal of their choice. It will not participate directly in the story and its movements will repeat through the scene. The teacher shows them the expected result (file "2-2a_sea_ correction_lesson6.b2), which uses new "Control" instructions:

The teacher tells the class about these instructions: they are loops, but you cannot decide ahead of time how many times they will be executed. Some of these loops have a hexagonal field that must be filled in with an instruction in the same shape. Some can be found in the "Sensing" instruction category (as well as in the "Operators" category, but these will not be useful here), such as:



Combinations such as above can be found, and the teacher can describe them to the class and test them on the cat sprite:

The cat sprite moves 10 steps, stops for one second, then starts again until it touches the edge. If it is touching the edge from the start, it does nothing. If it starts near the edge, it will take one or two steps; if it starts far from the edge, it will take more steps.

Once students have understood, the teacher suggests inactivating the scripts for the first two episodes of the story (to do this, simply disconnect them from the trigger event) to they can focus on the current episode. They must try to get a similar result as the one the teacher showed them, but it does not have to be exactly the same.

Group discussion

During the group discussion, the teacher goes over any difficulties students had. The teacher guides the discussion to new loop types: the “repeat forever” loop and the “repeat until ...” loop. If there is enough time, students can revise their programs following the group discussion. If not, the students can make any corrections during the extra 30 minutes dedicated to this lesson to connect episodes 2 and 3.

The file “2-2a_sea_correction_lesson6.sb2” provides an example of the result once the scripts for the first two episodes are reactivated and after the dock and sea episodes are linked.

Teaching notes

- Some students will use instructions the class has not yet seen. At this point, when they begin to have a good understanding of the software, it is important to let them express their creativity. But to help them continue learning, the teacher should encourage them to regularly test their programs. It is much more difficult to detect a bug in a program written as a block without tests than in a program that has been tested throughout the writing process.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *Certain loops are repeated forever.*
- *Certain loops are repeated until a condition is met (for example, a sprite touches another sprite, or a sprite touches a certain color).*

The students write down this conclusion in their science notebook and complete the *Scratch* worksheet by coloring in the new instructions they learned and writing down a few key words: infinite loop, condition, touch a sprite, touch a color.

Lesson 7 - Producing the final episode autonomously

Summary	Students work on their own to tell the last episode of their hero's adventure. They cover the key ideas from the entire sequence and finish their program.
Key ideas *(see Conceptual scenario, page 108)*	Same as for the previous lessons
Equipment	Same as for the previous lessons
Duration	1 or more one-hour lessons

Starting the activity

The teacher tells the students that they can tell the last episode almost however they like with *Scratch*. There is only one condition: they must use events to coordinate their characters and use loops. The teacher suggests that they try and not be overly ambitious to start with and to add elements to their story if they have time at the end.

Creating the episode (ideally in pairs)

The students talk in pairs about the essential elements to include in the last episode. Which characters? Which stages? Which events to trigger the characters' actions?
The students have two available backdrops: the beach at sunset and the hero's bedroom. They can also choose another stage from the *Scratch* library.

Programming the episode

After approval from the teacher, who checks with the students that their plan is realistic, the students program the last episode in a new file separate from the first three episodes. The teacher walks around to all the groups to make sure they are making progress.

Conclusion: Class presentation

To conclude the lesson, each group of students shows their final episode to the class. The students talk about the programs they are most proud of or those that were most difficult to figure out. They can also share any difficulties they could not solve and see if the class can help find a solution. Students will have time to modify their episode after the class discussion.
The file "2-2a_magician_correction_lesson_7.sb2" is an example of a program to tell the last episode (bird, magician and returning home for the hero). This is only one possible option.

Further study

Now that students have learned to program on *Scratch*, they can use their skills to create other projects during the year, such as animated cards for Christmas, Mother's Day, etc.

Sequence 3: Robotics

This sequence deals with programming a **robot** and is an alternative option to Sequence 2 (programming in *Scratch Junior*) and its alternative Sequence 2a, programming in *Scratch*).

Programming a robot instead of programming on a tablet or computer

Programming a robot uses the same concepts as programming on a computer; they are simply applied to a physical object. The Thymio robot can be programmed using Aseba/VPL (visual programming language).
Sequence 3 does not include scripted elements from the adventure from Sequence 1 (pages 109 and on): in this respect, it is completely independent. However, we do not suggest doing it alone but rather after Sequence 1. This will let students use the unplugged activities to conceptualize the key ideas of algorithm and program, which will be covered again here, as well as key ideas specific to robots (sensors, actuators, environmental interaction, etc.). That said, there is no need to have completed Sequence 2 (*Scratch Junior*) or the alternative lesson 2a (with *Scratch*).
This "Level 2 robotics" sequence is highly inspired by the "Level 1 robotics" sequence (pages 81 and on), and in fact repeats certain parts of it: the first three lessons of this sequence are a repeat of the Level 1 lessons, with just a few small differences. The following lessons will give students a more in-depth look at robot programming. For the sake of brevity, we will not recopy all the information here, but simply the key parts of these lessons with the necessary adaptations. We therefore suggest reading the "Level 1 robotics" sequence beforehand.

	Lesson	Title	Page	Summary
	Lessons 1, 2, 3	Introduction to Thymio in Level 2	183	Students are introduced to the Thymio robot and familiarize themselves with it. After exploring the various pre-programmed modes, they have Thymio run a maze. They gradually formulate a simple definition of what a robot is. (Adaptation of the four first lessons of the «Robotics in Level 1» sequence, pages 81 and on)
	Lesson 4	Programming Thymio (1/2)	188	To go into more depth with Thymio, students discover the Aseba/VPL programming environment. The graphic interface lets them design their own programs for Thymio.
	Lesson 5	Understanding sensors to program Thymio	193	VPL programming for Thymio is event-driven: students will learn how to use Thymio's sensor status to trigger precise actions.
	Lesson 6	Programming Thymio (2/2)	196	Students take on small challenges to create their own VPL programs for Thymio.
	Lessons 7 and 8	Obstacle course for Thymio	198	Students must reproduce Thymio's yellow «explorer» mode. First, they write the program. Then, they test their program in a real maze.

Lesson 1, 2, 3 - Introduction to Thymio in Level 2

Summary	Students are introduced to the Thymio robot and familiarize themselves with it. After exploring the various pre-programmed modes, they have Thymio run a maze. They gradually formulate a simple definition of what a robot is. (Adaptation of the four first lessons of the «Robotics in Level 1» sequence, pages 81 and on)

Key ideas
(see Conceptual scenario, page 108)
	"Machines"
	• The machines all around us simply follow "orders" (instructions).
	"Robot"
	• A robot is a machine that can interact with its surroundings.
	• A robot has sensors that let it perceive its surroundings.
	• A robot can perform actions: move, make a sound, produce light, etc.
	• A robot has a computer that decides which actions to take in which situations.
	• If you compare a robot to an animal, you can say that:
	  • Its sensors are like sensory organs
	  • Its motors are like muscles
	  • Its computer is like a brain
	  • The parts taken together are like a body
	"Algorithms"
	• A test indicates which action to perform when a condition is met..

Inquiry-based methods	Experimentation, Observation, Discussion
Equipment	For the teacher: • Handout 8, page 85 (documentary handout from Level 1) • A screwdriver • Drawing paper, black paint and a paint roller (4 cm) • A2 size poster or flip chart For each group: • A Thymio robot • A track printed on A3 paper • Objects that can be easily moved around and used as obstacles for Thymio (cubes, books, etc.) For each student: • Handout 23, page 187
Glossary	Thymio, instruction, condition, test, robot, sensor, motor, program
Duration	2 to 3 one-hour lessons

Foreword

As previously explained, this Level 2 robotics sequence covers the Level 1 introduction to Thymio sequence in two to three lessons (pages 81 and on) with a few differences as described below, and then goes into greater detail with an initial approach to programming a Thymio robot using Aseba/VPL.
We also suggest having students complete Handout 23 individually (instead of doing their own drawings), and creating posters for the entire class.

First lesson: Meeting Thymio

As explained in Lesson 2.1, Level 1 (page 82), the teacher shows the Thymio robots (turned off) to the students, who have been divided into small groups. Students can then play with Thymio on their own to figure out how to turn it on and off and make it change colors.

Students from Nathalie Pasquet's class (Paris)

Then, as in Lesson 2.2, Level 1 (page 86), the teacher gives Handout 23 to the students. Each group will study one of Thymio's colors (green, red, purple, yellow) and describe its behavior, then fill out Handout 23 by connecting the event/action pairs. The following vocabulary is introduced during the group discussion: a *test* is comprised of a *condition* ("***IF** green Thymio detects an object in front of it*") and an *instruction* to do only if the condition is met ("***THEN** it moves forward*").

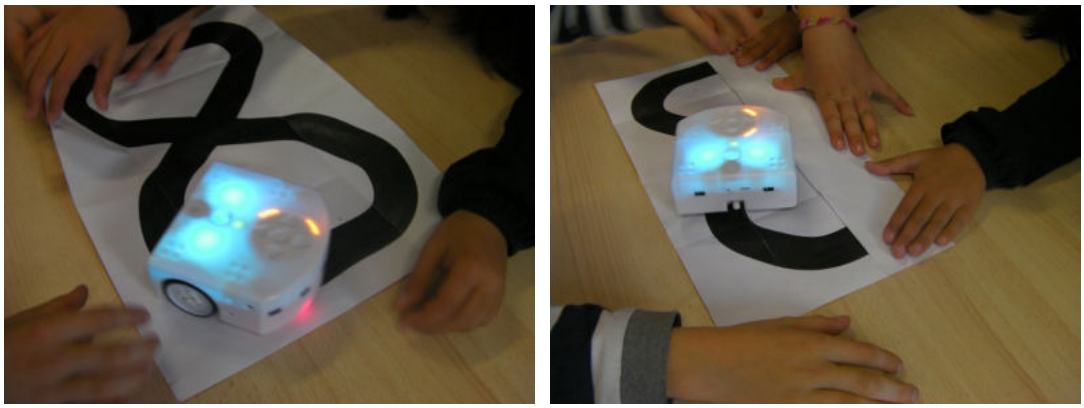
Teaching notes

- Handout 23 is meant to be quite condensed. Accordingly, it does not include some of Thymio's behaviors, which can be explored and described orally with the class:
    - In "red" mode: Thymio behaves differently depending on whether an object is placed "behind it, to the right" or "behind it, to the left." It's up to you to figure out how!
    - In "purple" mode": the behavior of the arrows is not completely straightforward. In fact, moving is more complex than simply going forward, standing still, or going backward. Thymio has three speeds in each direction (forward or backward). Pressing one arrow increases ("forward" arrow) or decreases ("back" arrow) the speed, similar to the speeds on a motorLevel. For example, if Thymio is moving on speed 3 (the fastest), pressing on the "back" arrow does not make it move backward, but rather slows it down one speed. Thymio will then begin moving in speed 2. These speeds can be combined with the instructions "turn right" or

"turn left" to make Thymio turn at different speeds.
    - In "yellow" mode, Thymio's behavior is less exaggerated than on the handout. When an obstacle is placed in front of it, Thymio tries to keep moving forward several times before deciding to go around the obstacle.
- Using the Thymio robot can help deepen students' understanding of a nuance of language, which will recur often in the different lessons: what is the difference between "move" and "move around"? A mechanical arm screwed into the ground cannot move around the room, for example, but its joints allow it to move: it can pivot, bend, etc. Thymio, by moving (or turning) its wheels, can move around.

Second lesson: Get Thymio through a maze

Using Lesson 2.3, Level 1 (page 89), the teacher has the students test out a Thymio mode that was not discussed in the previous lesson: the turquoise mode. The teacher hands out the printed tracks[13] so students can see what turquoise Thymio can do. The students will learn about his "investigator" or "tracker" mode.



Students from Nathalie Pasquet's class (Paris)

Next, the teacher has students do an activity that will touch on all key ideas covered so far: getting Thymio through a maze built with objects such as cubes, books, etc. Each of Thymio's modes students have seen until now can be used for this activity: Green Thymio will follow a hand that guides it through the maze, turquoise Thymio will follow a black ribbon placed on the ground in the maze, etc.

Third lesson: Thymio is a robot

As explained in the review lesson "What is a robot?" (page 96), the teacher begins by taking Thymio apart to describe its components. Little by little, the students will understand that a robot is a machine that can interact with its environment using sensors, actuators, and a program.

13 All of the *Scratch Junior*, *Scratch* and Thymio programming resources are available on the project website. See page 342.

Second grade and third grade class, Anne-Sophie Boullis (Saint-Georges d'Orques)

Conclusion and lesson recapitulation activity

Through these lessons, the class creates a group poster showing what they learned about Thymio. In particular, they will come to the following conclusions, which are to be written down in their science notebook:

- *Thymio turns on using the middle button.*
- *Thymio can change color.*
- *Thymio can make sounds.*
- *Thymio can be in different modes, each indicated by a different color, which determine the robot's behavior.*
- *A robot has a computer, sensors and actuators that are all interconnected.*

HANDOUT 23
Learning abaout Thymio

Instruction: Turn on your Thymio robot and try out the different modes. Find a nickname for each mode. Next, link the event and action pairs based on your observations.

Green Thymio

Nick-name:

IF Thymio detects an object in front of it	●	● **THEN** it turns left
IF Thymio detects an object to the right	●	● **THEN** it turns right
IF Thymio detects an object to the left	●	● **THEN** it moves forward

Red Thymio

Nick-name:

IF Thymio detects an object in front of it	●	● **THEN** it moves backward
IF Thymio detects an object to the right	●	● **THEN** it moves backward and turns right
IF Thymio detects an object to the left	●	● **THEN** it moves backward and turns left
IF Thymio detects an object behind it	●	● **THEN** it moves forward

Purple Thymio (start blocked)

Nick-name:

IF you press on the forward arrow	●	● **THEN** it moves forward
IF you press on the backward arrow	●	● **THEN** it moves backward
IF you press on the right arrow	●	● **THEN** it turns left
IF you press on the left arrow	●	● **THEN** it turns right

Yellow Thymio

Nick-name:

IF Thymio detects an object in front of it	●	● **THEN** it turns left
IF Thymio detects an object to the right	●	● **THEN** it turns right
IF Thymio detects an object to the left	●	● **THEN** it moves backward
IF Thymio detects nothing	●	● **THEN** it moves forward

Lesson 4 - Programming Thymio (1/2)

Summary	To go into more depth with Thymio, students discover the Aseba/VPL programming environment. The graphic interface lets them design their own programs for Thymio.
Key ideas *(see Conceptual scenario, page 108)*	"Machines" • The machines all around us simply follow "orders" (instructions) "Languages" • We can give a machine instructions by using a special language called a programming language, which can be understood by both people and machines. If you launch the same program several times, it will always do the same thing. "Robot" • A robot is a machine that can interact with its surroundings • A robot has a computer that decides which actions to take in which situations. "Algorithms" • A test indicates which action to perform when a condition is met.
Equipment	For each group • A Thymio robot. • A computer (Windows, Mac or Linux OS) with VPL software[3]. For each student: • Handout 24, page 191 • Handout 25, page 192
Glossary	Programming language
Duration	1 hour

Preparation

Download the Aseba/VPL software at https://www.thymio.org/en:start (you can choose the version for your operating system). The software is free of charge and available for Windows, Mac and Linux OS. During installation, simply accept the default settings (and be sure to choose the "For Thymio II" option (Recommended)).

When installation is complete, rename the "Aseba" shortcut "Thymio."

To start VPL	
Method 1	**Method 2**
1. Connect Thymio to the computer using the USB cable (it will turn on) 2. Start Thymio-VPL	1. Start Thymio-VPL (a «Choose an Aseba target» window opens) 2. Connect Thymio to the computer using the USB cable (it will turn on) 3. Check the «Serial port» box, select «Thymio-II Robot,» click «Connect»

To program
1. Write the program 2. Save the program 3. Run the program

If the robot is connected to the computer via a cable and the software is closed out by mistake, the computer may no longer recognize the robot. If this happens, unplug the robot and plug it back in.
Start the "Thymio VPL" software or ask the students to do so.

Starting the activity

In the previous lessons, students will have handled Thymio and learned that a computer commands the robot's actions based on what its sensors detect. Now, students will create their own programs to have Thymio perform other actions. For students to be able to talk to Thymio, the teacher presents the Aseba/VPL programming language.

Experiment: Programming Thymio with VPL (in groups)

This language lets students create programs by describing a series of tests. By combining one card each from the left and right columns, students can create a test.

To help students better understand VPL, the teacher gives students Handout 24. The students begin by getting familiar with the interface and how to create programs with the cards provided. In particular, they will see that the cards from the left column correspond to various events that the sensors can trigger, while the cards from the right column correspond to actions.

Teaching notes

- You learn to program by programming, not by watching someone program. It is interesting to consider the same problem in pairs (probably more so than to program alone), but it is important to be active. We therefore recommend placing the students in small groups in front of computers (ideally two students per machine) and asking them to "switch over" (pass the keyboard and mouse to their neighbor) every 10 minutes.
- For this introductory lesson, only the "red" sensor status is taken into account. If students ask what the "black" status for a sensor in VPL means, the teacher tells them that this will be covered in the next lesson.

The class explores the pre-set programs. The teacher gives students Handout 25. The students must test the four programs one after another and write down what they did in the form of a test.
To test the effect of a program on Thymio, students must:

- Delete the cards already in the central area of the graphic interface by clicking on the corresponding Xs (see Handout 24)
- Place the event card and the action card to be tested in the central area
- Modify these cards if necessary by clicking the buttons and/or moving the cursors
- Start the program by pressing the arrow

- Place Thymio on a flat surface, without unplugging it if possible, to test and observe the effects of the program by manipulating Thymio as much as needed. If the students unplug Thymio, they will have to plug it back in before testing the next program.

Group discussion

The teacher writes down the class's descriptions for the different programs on the board:

- Program 1: IF you press the center button, THEN Thymio moves forward
- Program 2: IF Thymio detects an object in front of it, THEN the cover turns green
- Program 3: IF Thymio detects an object under it, THEN the chassis turns blue
- Program 4: IF you tap on Thymio's cover, THEN it plays music

Scientific notes

- Program 3 uses Thymio's chassis sensors. These are the same sensors that are used for turquoise Thymio's "line-follower" mode. If you put Thymio near the edge of the table, it detects nothing. The same happens when you put it on a black piece of paper. However, it will detect any light surface.
- For Programs 1, 2 and 3, Thymio will not come back alone to its original setting (it does not stop until another instruction is given and will not go back to the original color). This is normal, because this would be another behavior that the one covered here: at the moment, when it detects an event, Thymio changes color or begins to move and it shall continue until another program asks it to do something else. The next lesson deals with getting Thymio to return to its original setting.

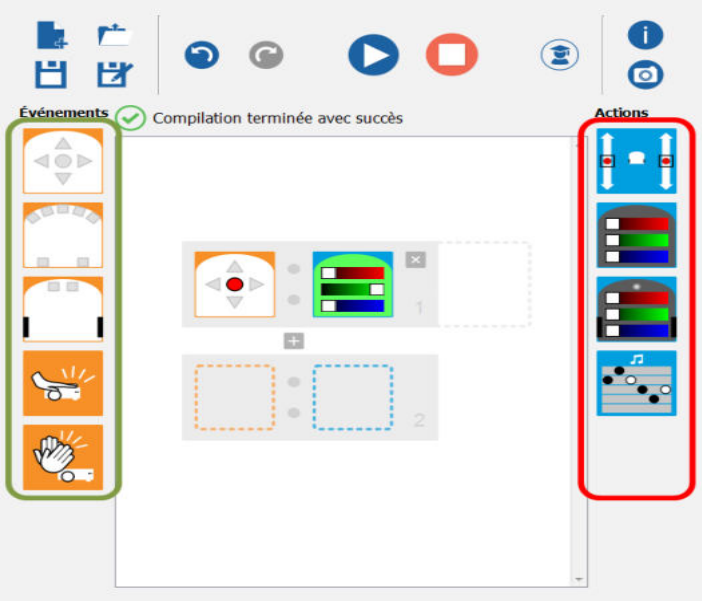
Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *The machines all around us simply follow orders (instructions).*
- *We give instructions to a machine by creating a program, which uses a programming language.*
- *The execution of a program is reproducible (if neither the instructions nor the data to manipulate change, the program always gives the same result).*

HANDOUT 24
Programming Thymio: Introduction to the VPL interface

Instruction: Place two cards in the center and change them to reproduce the program below. Then circle the correct answers based on what happens.



The button  is used to: *Start the program* *Stop the program*

The button  is used to: *Start the program* *Stop the program*

The images in the green box are: *Actions* *Sensors*

The images in the red box are: *Actions* *Sensors*

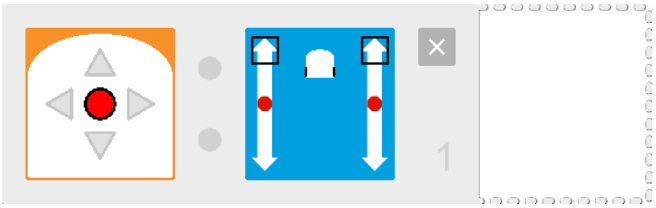
The "+" button outlined in green is used to: *Delete an order* *Add an order*

The button "×" outlined in red is used to: *Delete an order* *Add an order*

HANDOUT 25
Writing your first Thymio programs

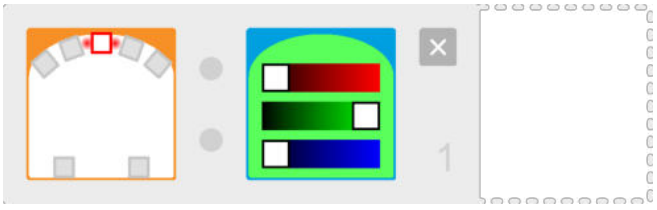
Instruction: Here are four different programs that are created using one event card and one action card. Test them on your Thymio robot, then complete the sentences to describe what each program does.

Program 1:



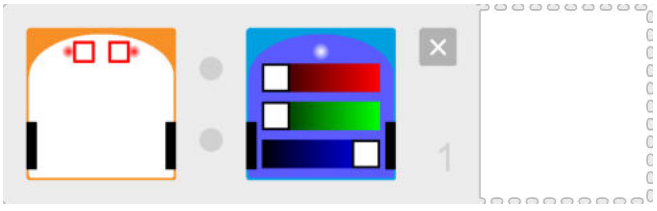
IF	THEN

Program 2:



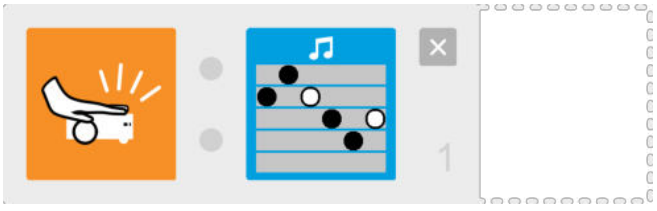
IF	THEN

Program 3:



IF	THEN

Program 4:



IF	THEN

Lesson 5 - Understanding sensors to program Thymio

Summary	VPL programming for Thymio is event-driven: students will learn how to use Thymio's sensor status to trigger precise actions.
Key ideas *(see Conceptual scenario, page 108)*	"Robot" • A robot has sensors that let it perceive its surroundings. "Algorithms" • A test indicates which action to perform when a condition is met.
Equipment	For each group • A Thymio robot. • A computer with VPL software. For each student: • Handout 26, page 195. • (as well as Handout 25 from the previous lesson for further study activities)
Glossary	Sensor, event
Duration	1 hour

Starting the activity

In the previous lesson, the students programmed basic behaviors for Thymio: moving forward and changing color. But they also noticed that Thymio never goes back to its original setting. If it starts moving forward, nothing in its program tells it how or when to stop. The teacher restates this observation: *when a sensor detects something, we say there is an* "event"*; for each event, Thymio verifies in its program if there is a test that gives it instructions to follow. In your opinion, can "detecting nothing" be an event?*

Experiment: Detecting and not detecting (in groups)

The teacher gives students Handout 26. Each group will test the programs on the handout, making sure to delete the previous programs, and answer the questions.

Scientific notes

- For the first time, students will see a program with more than one test (Program 5 has two tests). The two tests must be written one above the other for the program to be complete.

Group discussion

The class understands that VPL lets them write very precise tests, based on whether the sensors detect something (red icon), detect nothing (white icon) or if their setting is not important (gray icon).

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *When a sensor detects something, we call this an event.*
- *A condition can be "an event happened" or "an event has not happened"*

Further study

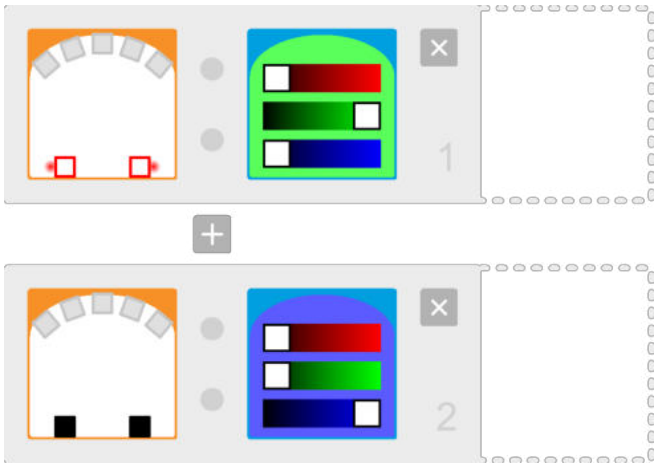
Faster students can apply this new information to complete the programs on Handout 25 (previous lesson):

- Program 1: Add a test to make Thymio stop (e.g., when another button is pressed).
- Program 2: Add a test to make Thymio's cover not be green (e.g., it turns yellow) if it no longer detects anything in front of it.
- Program 3: Add a test to make Thymio's chassis not be blue if it no longer detects anything underneath it.

HANDOUT 26
Testing Thymio's sensors

Instruction: Here are two different programs: Program 5, which has two tests, and Program 6, which has one test. Try them on your Thymio robot, then answer the questions.

Program 5:



Circle the right answer:

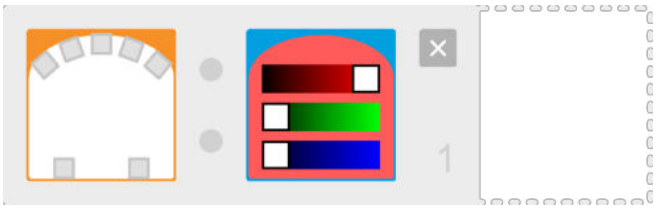
What color is Thymio when your hand is in front of the two rear sensors?

GREEN / BLUE

What color is Thymio when your hand is not in front of the two rear sensors?

GREEN / BLUE

Program 6:



Answer the questions:

What color is Thymio when your hand is in front of the two rear sensors?

. . . . . .

What color is Thymio when your hand is not in front of the two rear sensors?

. . . . . .

Connect the icons to their meanings.

The icon  means ●

The icon  means ●

The icon  means ●

● «IF the sensor detects or does not detect something...»

● «IF the sensor does not detect something...»

● «IF the sensor detects something...»

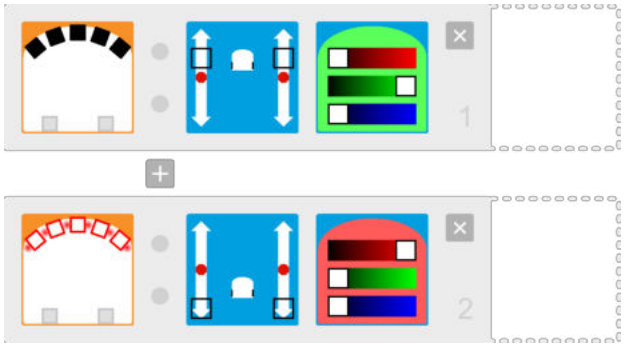
Lesson 6 - Programming Thymio (2/2)

Summary	Students take on small challenges to create their own VPL programs for Thymio.
Key ideas *(see Conceptual scenario, page 108)*	"Machines" • The machines all around us simply follow "orders" (instructions) "Languages" • We can give a machine instructions by using a special language called a programming language, which can be understood by both people and machines. "Robot" • A robot is a machine that can interact with its surroundings. • A robot has a computer that decides which actions to take in which situations. "Algorithms" • A test indicates which action to perform when a condition is met.
Equipment	For each group • A Thymio robot. • A computer with VPL software.
Duration	1 hour

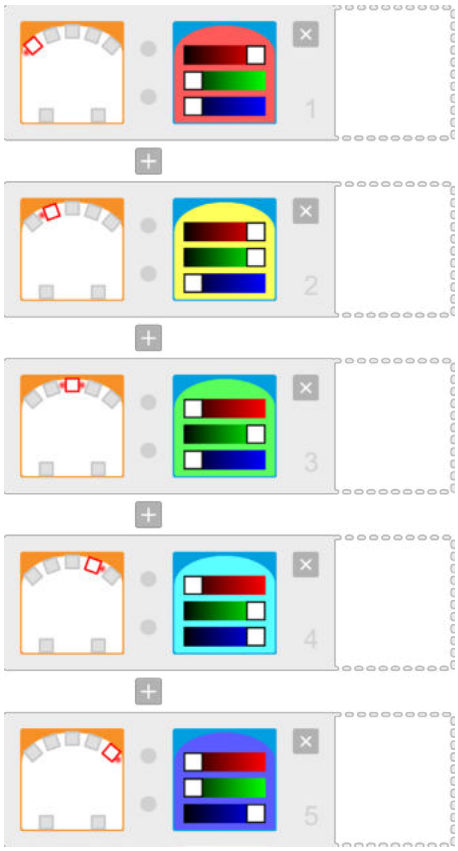
Challenges: Creating new programs for Thymio (in groups)

In this formative assessment lesson, the students review the key ideas they have previously learned. The teacher will have the class do three new challenges. The students will have 20 minutes to create a program for each problem.

Challenge 1: Make Thymio move forward if its front sensor detects nothing and move backward if the sensor detects something. Link one color to each of these movements.

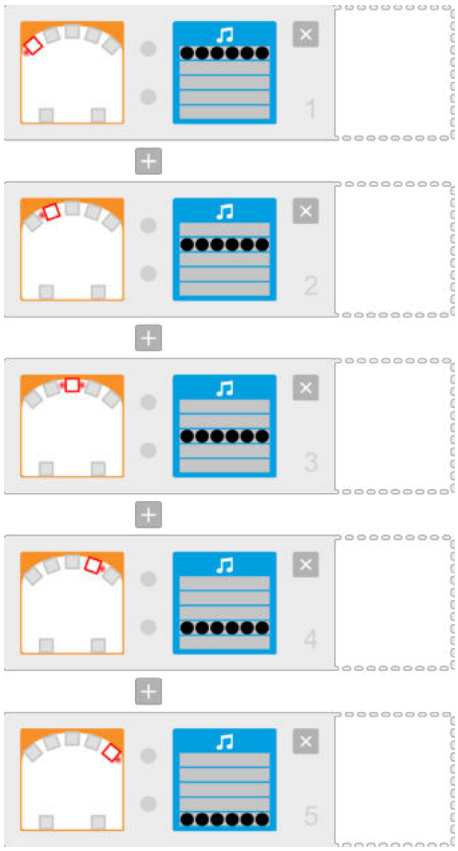


Second grade and third grade class, Anne-Sophie Boullis, Saint-Georges d'Orques



Challenge 2: Create a color selector. Link one color to each of Thymio's front sensors.
For this challenge, the students may realize that there is an order for the tests. For example, if two conditions are met at the same time with concurrent executions, which takes priority? Here, the execution affects Thymio's color: it cannot have two colors at the same time, so which will it choose? Answer: VPL applies the instruction with the highest number. If, for example, you simultaneously activate the "center right" sensor (Instruction no.4: turn Thymio turquoise) and the "right" sensor (Instruction no.5: turn Thymio blue): Thymio will turn blue. (Here, instructions no.5 and 4 are concurrent, but instruction no.5 is executed.)

Challenge 3: Create a musical instrument. Link a sound to each sensor.



Lessons 7 and 8 - Obstacle course for Thymio

Summary	Students must reproduce Thymio's yellow «explorer» mode. First, they write the program. Then, they test their program in a real maze.
Key ideas *(see Conceptual scenario, page 108)*	"Machines" • The machines all around us simply follow "orders" (instructions) • By combining several simple instructions, we can perform a complex task. "Languages" • We can give a machine instructions by using a special language called a programming language, which can be understood by both people and machines. • A bug is an error in a program. "Robot" • A robot is a machine that can interact with its surroundings • A robot has a computer that decides which actions to take in which situations. "Algorithms" • A test indicates which action to perform when a condition is met.
Equipment	For each group • A Thymio robot • A computer with VPL software • (Optional) Handout 27 (page 200)
Glossary	Bug
Duration	2 one-hour lessons

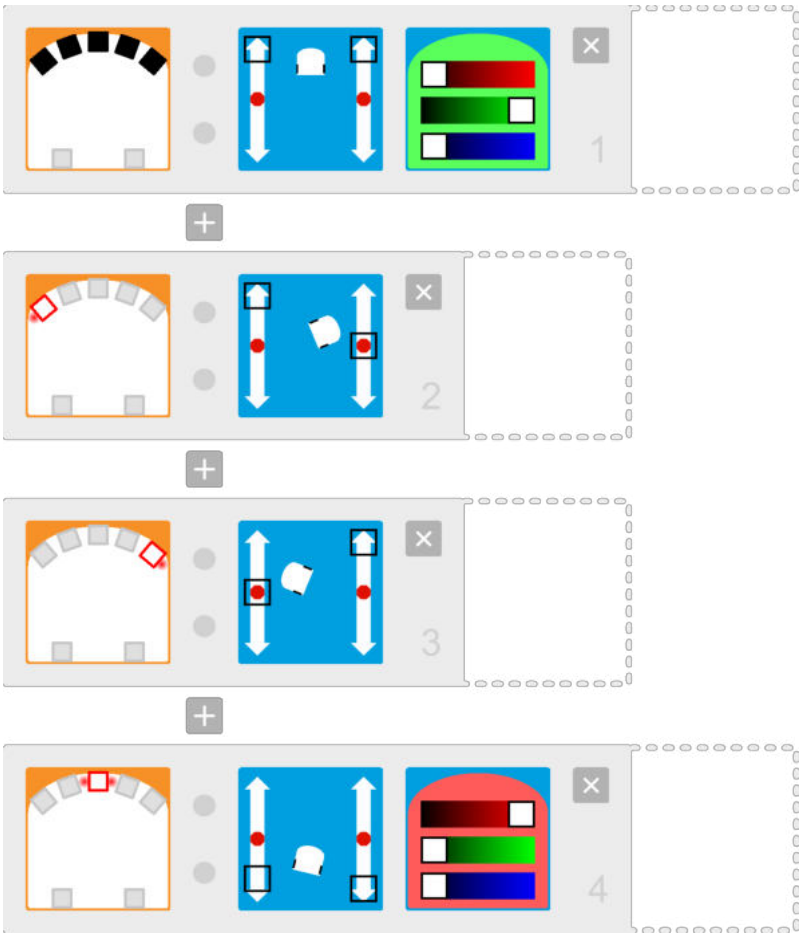
Starting the activity

The teacher reminds the students that Thymio comes with pre-programmed modes. The teacher gives them a challenge: reprogram a similar mode (simplified) to Thymio's yellow mode. The students remember that this is the explorer mode, where Thymio moves forward and avoids obstacles.

Challenge: Reproducing an explorer Thymio (in groups)

Depending on how comfortable the class is with the technology and their ages, this challenge can be done in several ways. For more independent students, the teacher can not pass out Handout 27 and simply use it as a cheat sheet. If students need more support, the teacher can give them the handout to give them more guidance.
In groups or as a class, the various stages of programming need to be broken down: what does Thymio do in its default settings? If it detects an obstacle to the right, what does it do? And to the left? And in front of it? VPL is then used to program the robot and test whether the program works by playing with Thymio on the table.
It is very possible that the lesson will be over by the time the programs are written. The "real world" test can then be done during the next lesson.

One example of a correct program is:



Experiment: A real test for Thymio (in groups)

The class now prepares a huge maze with obstacles that are at least 6 cm tall. All the groups will test their programs at the same time: the robots will interact with the maze and with each other. If possible, the floor of the maze can be covered with drawing paper: each group can insert a marker in the pen holder on Thymio's cover. This will let them see the paths taken by the different robots during the experiment.
The groups load the program they designed during the previous lesson and let their robot run the maze. They can improve their program as problems crop up: the teacher uses this as an opportunity to introduce the term *"bug"* to describe the issues.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *By combining several simple instructions, we can perform a complex task such as running a maze*
- *A bug is an error in a program.*

HANDOUT 27
Programming an "explorer" Thymio

1. 

Create an instruction to make Thymio move forward if its front sensors detects nothing

2. 

Add an instruction to make Thymio turn right when it detects something on the left

3. 

Add an instruction to make Thymio turn left when it detects something on the right

4. 

Add an instruction to make Thymio back up slightly while turning if it detects something in front of it

5. (OPTIONAL)

Add instructions to make Thymio turn red if it detects an obstacle and green if it does not

Activities Level 3

Overview

The activities module for Level 3 alternates unplugged exercises (done without a computer but with experimentation and documentary equipment) and plugged programming exercises (using a computer).

The theme that ties the activities together is the exploration of an unknown planet.

- Sequence 1 (entirely unplugged) focuses on preparing the mission (How will you get around and communicate?) and allows students to become familiar with programming language and encoding information (first using numerals, then using binary code).
- Sequence 2, mainly plugged, gives students a similar mission through a video game they must program. Students learn about the programming environment and define the steps and tasks needed to complete the project as they work at their own pace. Several unplugged activities help students to fully understand the concepts (variables, tests, loops, logical operators) used during the programming activities.
- Sequence 3 (entirely unplugged) returns to the issue of how information is represented to teach students how to code images, learn cryptography techniques for secure data exchanges, etc.

If the classroom is not equipped with computers, Sequences 1 and 3 can be done back to back so students can complete an entirely unplugged IT project, which is an interesting approach in and of itself.

Note: Contrary to the activity modules for Levels 1 and 2, here we do not have a "robot" variation. However, if the classroom is equipped with robots, you can substitute or round off the *Scratch* lessons with robot programming lessons. These types of lessons will introduce students to the ideas of algorithms, programs, tests, events, etc.[14]

14 The Thymio II robot, used in the Level 1 and 2 lessons, can also be used for Level 3. You can even operate it using *Scratch* (this requires installing an extension for *Scratch*, refer to https://www.thymio.org/en:scratchprogramming-asebascratch).

Lesson summary

Sequence 1: Prepare the mission

	Lesson	Title	Page	Summary
	Lesson 1	How to remotely operate a vehicle	207	Students must provide the instructions to remotely operate a vehicle. To do this, they define a programming language and explore the difference between it and a natural language. They are also introduced to the notion of a software bug.
	Lesson 2	How to encode a message with numbers	213	Students must encode a textual message using only numbers. To do this, they make suggestions and then create a correspondence table for the letters and numbers for use by the entire class. They use this table to encode a message that they send as well as to decode a message they receive.
	Lesson 3	How to code information in binary	219	Students must now use only two symbols (0 and 1) to transmit messages. They explore the ways of encoding different information (the four cardinal directions – North/South/East/West, the seven days of the week, etc.) by combining 0s and 1s as an introduction to binary coding.
	Lesson 4	How to encode and decode a binary message	225	Continuing on from the previous lesson, students apply what they have learned to encode a short worded message in binary code, then decode a message in binary code they receive.

Sequence 2: Simulate the mission in Scratch

For this sequence, we reason in “steps” rather than lessons (see note on project pedagogy, at the start of this sequence, pages 229 and on).

	Step	Title	Page	Summary
	Step 1	Introduction to the *Scratch* programming environment	234	The students learn about *Scratch*, a programming environment suited to elementary school students. They learn to launch the program and follow a few simple instructions.
	Step 2	Customizing the environment and saving all work	243	The students learn to customize *Scratch* (sprite and background) and save their work to be used again later. They discuss the different steps they will follow to create their video game.
	Step 3	Operating the rover	246	The students create their first program, which will let them operate the rover using arrows. They learn about the coordinate system.
	Step 4	Gathering resources and managing scores	253	Students complete their program by adding resources they must pick up (new sprites) and creating a variable for their score (this score increases as more resources are picked up). They learn to program conditional constructs (if– then) and use sensors.
	Step 5	Plugged and unplugged activities to better understand certain algorithmic concepts	260	Alongside their programming activity, students deepen their understanding of certain algorithmic concepts introduced durixng Step 4: variables, tests, loops, logical operators and even the notion of algorithm.

	Step 6	Avoiding obstacles and managing player lives	277	Students now add obstacles to avoid (new sprites) and create a variable for the number of «lives». They are again exposed to the ideas of tests, loops and variables seen previously and deepen their understanding of what an event is.
	Step 7	Ending the game: «Game over»	281	Students complete their program by introducing a test on the number of remaining lives: the message «Game over» appears and the program stops when no more lives remain.
	Step 8	Adding challenges	284	Students finalize their video game by adding additional challenges: a countdown, a tornado that goes faster and faster and moves around randomly, etc. The concepts seen during the previous lessons – tests, loops, variables and events – are all reviewed.
	Step 9	Further study in *Scratch*	292	Here are several ideas to explore other functionalities in *Scratch*, such as giving students extra options for their personal projects.

Sequence 3: Sending news

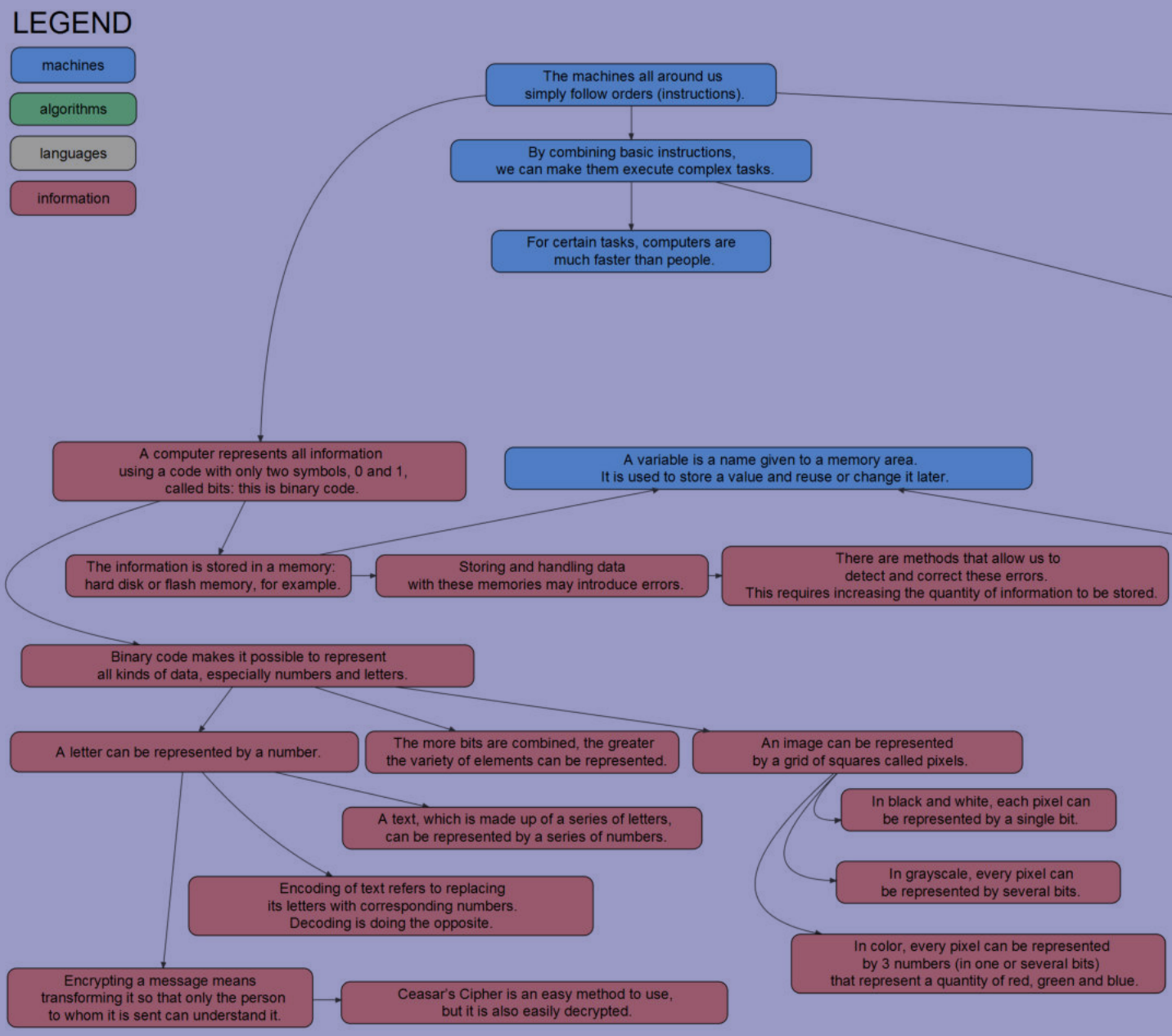
	Lesson	Title	Page	Summary
	Lesson 1	How to send an image	296	Students must figure out how to transfer an image remotely. To do this, they learn that an image can be represented by a pixel grid. They learn about the notion of resolution as they see that the more pixels an image has, the clearer it becomes, but also the slower it is to transfer.
	Lesson 2	How to code a black and white image	304	Students apply what they learned from the previous lesson to coding black and white digital images. They first view a single file in a text editor and an image editor to understand how the information is coded. They then code a small checkerboard themselves.
	Lesson 3	(Optional) How to code a grayscale or color image	309	Students take what they learned in the previous lesson further by learning how to code a gray and color digital image.
	Lesson 4	How to ensure a message is secure	316	To protect their messages, students learn about encryption using a simple algorithm (called Caesar's cipher), which involves shifting the letters of a message.
	Lesson 5	(Optional) How to make sure our data are successfully sent	324	Students learn that it is possible to detect and correct errors introduced when storing or transferring a file by adding the right information. This lets them do a sort of «magic trick.»

Review: Defining computer science

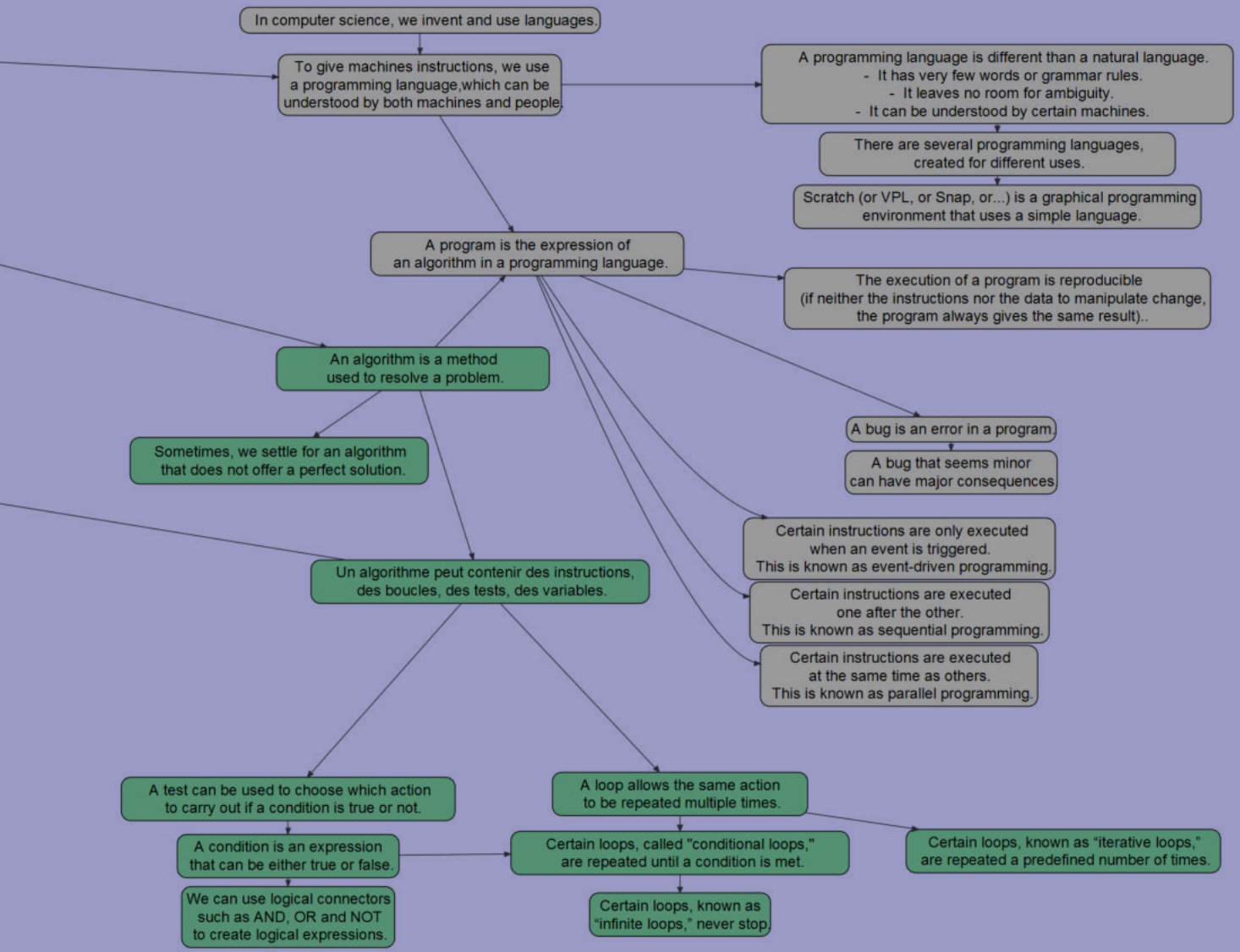
This lesson, page 328, is a review of what computer science is all about using the poster created during the previous sequences. With the help of documentary research, students create a timeline of the key moments in the history of computer science.

Conceptual scenario: << Level 3 computer science >>

The ideas covered during these three sequences for Level 3 can be organized as below.



"1,2,3...code!"
Conceptual Scenario Level 3



Sequence 1: Prepare the mission

	Lesson	Title	Page	Summary
	Lesson 1	How to remotely operate a vehicle	207	Students must provide the instructions to remotely operate a vehicle. To do this, they define a programming language and explore the difference between it and a natural language. They are also introduced to the notion of a software bug.
	Lesson 2	How to encode a message with numbers	213	Students must encode a textual message using only numbers. To do this, they make suggestions and then create a correspondence table for the letters and numbers for use by the entire class. They use this table to encode a message that they send as well as to decode a message they receive.
	Lesson 3	How to code information in binary	219	Students must now use only two symbols (0 and 1) to transmit messages. They explore the ways of encoding different information (the four cardinal directions – North/South/East/West, the seven days of the week, etc.) by combining 0s and 1s as an introduction to binary coding.
	Lesson 4	How to encode and decode a binary message	225	Continuing on from the previous lesson, students apply what they have learned to encode a short worded message in binary code, then decode a message in binary code they receive.

Lesson 1 - How to remotely operate a vehicle

Summary	Students must provide the instructions to remotely operate a vehicle. To do this, they define a programming language and explore the difference between it and a natural language. They are also introduced to the notion of a software bug.

Key ideas
(see Conceptual scenario, page 204)
	"Machines"
	• The machines all around us simply follow orders (instructions).
	
	"Languages"
	• In computer science, we invent and use languages.
	• To give machines instructions, we use a programming language, which can be understood by both machines and people.
	• A programming language is different than a natural language.
	  • It has very few words or grammar rules.
	  • It leaves no room for ambiguity.
	  • It can be understood by certain machines.
	• There are several programming languages, created for different uses.
	• A bug is an error in a program.
	• A bug that seems minor can have major consequences.

Inquiry-based methods	Experimentation
Equipment	For the class • Handout 28, page 212 (optional) • A computer room with an internet connection
Glossary	Programming language, instruction, bug
Duration	1 hour 30 minutes (can be broken into two 45-minute lessons)

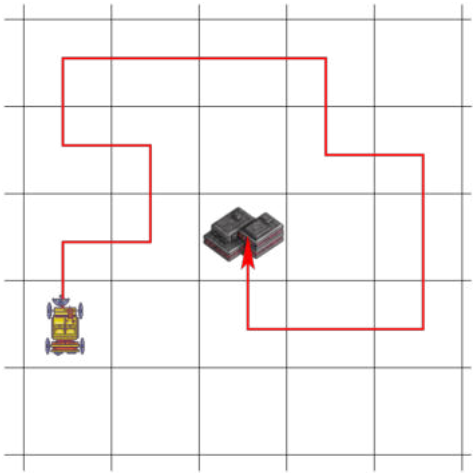
Introductory question: Project presentation

The teacher explains to the class that the project involves simulating an exploration mission on a faraway planet. The class must start by preparing the mission: thinking about how they will get around, communicate, etc. Then, the class will "play" the mission using a simulation program each student will create (a simple video game).
This will be a manned space mission and the team will have a base and a land rover vehicle available when they land on the planet. It is a hostile environment, so someone must always stay at the base during the exploration outings. If the field team members can no longer pilot the rover (for example, if they lose consciousness), the person at the base must be able to remotely drive the rover back to base without needing to talk to the team. Orders are transmitted to the rover as waves, but you must invent a language to give these orders.
The question is what language do you use to operate a rover remotely?

The teacher hangs or puts up on a projector a map of the area to explore (Handout 28). This area is sectioned off on a grid and a route has been drawn to return to base while avoiding dangerous areas. No shortcuts are possible: you must follow the established route in the direction of the arrows.

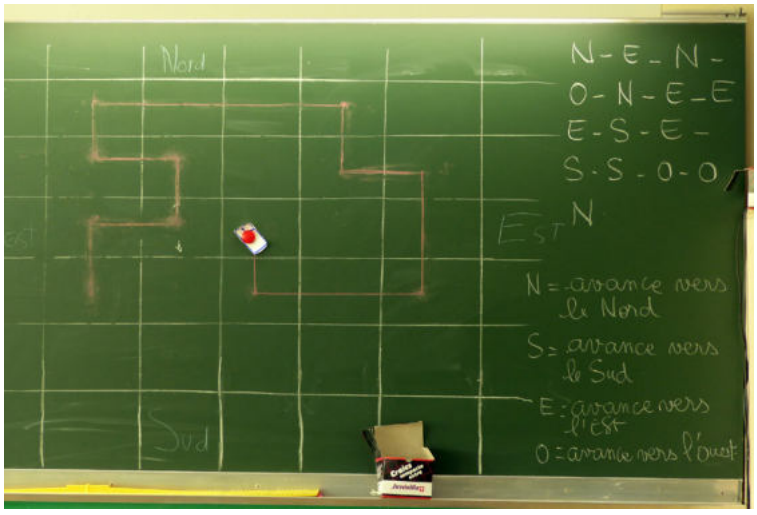
Quest: establish a language (in pairs)

The teacher passes out the Handout 28 to the students, who are split into pairs. They must decide what types of instructions to give to the rover to make it follow the required route and return to base. The movements must be made square by square (not diagonally).



Group discussion

After a few minutes, the class comes together to discuss all the pairs' work. For example, students can draw or show their solutions on a projector and the class verifies together if it is correct by filling out the table (use any object as the rover and have it follow the instructions exactly).



Fourth Grade class, Carole Vinel (Paris)
"N = move northward; S = move southward; E = move eastward; O = move westward"

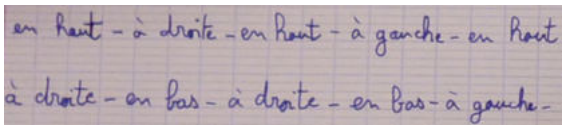
There are (at least) two ways consider giving the instructions. You can either give the rover "absolute" directions (go North, go West, etc.) or relative directions that depend on the rover's actual position (turn right, go forward, turn left, back up, etc.). It is preferable to divide the

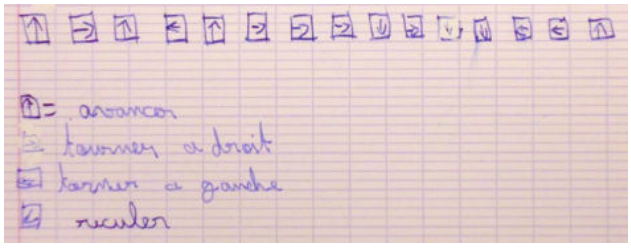
instructions into separate orders. For example, the instruction “Move forward one square to the right” is best given as: turn right (without advancing), then move forward one square.

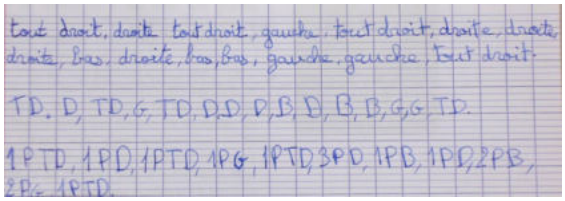
Teaching notes

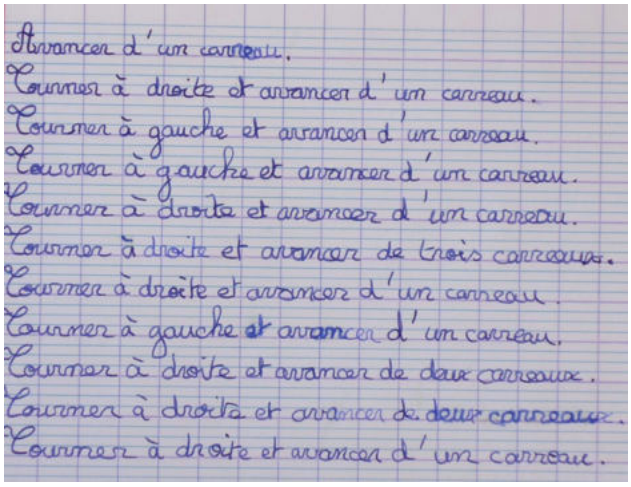
- The first approach to spatial processing (North, West, etc.) is called “allocentric” while the second (right, left, etc.) is called “autocentric”. Students do not need to know these terms as they will not be used in later lessons.
- A third approach (rarer) may also be suggested: assigning coordinates to each square (A1, A2, B1) and, like in a game of Battleship, code movements by giving the name of square of departure and arrival. For example, “Go from A1 to A2”. Please note: The direction “A1 to A2 is not ambiguous because these squares are adjacent. However, “A1 to B7” is ambiguous (and therefore not satisfactory) as there are several ways to move from square A1 to square B7. We will not go into further detail about this method in later lessons.

It is likely that different teams will suggest the two different methods. If this is not the case, the teacher can introduce the other method during the group discussion.









Several student suggestions
top left, allocentric language that does not address the distance moved: “up – right – up – left – up – right – down – right – down – left”;
top right, allocentric language with autocentric keys: “▲=advance ►=turn right ◄=turn left ▼=reverse;
lower left, showing changes made by a student during the research phase: “straight ahead, right, straight ahead, left, straight ahead, right, right, right, down, right, down, down, left, left, straight ahead;
SA, R, SA, L, SA, R, R, R, D, R, D, D, L, L, SA;
1tSA, 1tR, 1tSA, 1tL, 1tSA, 3tR, 1tD, 1tR, 2tD, 2tL, 1tSA”;
lower right, autocentric language completely written out: “Advance one cell. Turn right and advance one cell. Turn left and advance one cell. Turn left and advance one cell. Turn right and advance one cell. Turn right and advance three cells. Turn right and advance one cell. Turn left and advance one cell. Turn right and advance two cells. Turn right and advance two cells. Turn right and advance one cell.”.
Fifth Grade class, Anne-Marie Lebrun (Bourg-la-Reine).

The teacher tells the students that the instructions are written in a special language, with a very limited and unambiguous vocabulary: each instruction is perfectly explicit and must not have more than one possible interpretation. This is called a "programming language."
This language can be further simplified. For example, you do not need to say "Go to the East" or "Go to the right" when you can simply say "East" or "Right" instead (for example, assuming that the meaning of "right" is well defined when you say "move from one square to the right" and not "pivot a quarter-turn to the right").

As a class, the group describes the two languages, e.g.:

Allocentric language (or << absolute >>)	Autocentric language (or << relative >>)
• **North** (means "move one square to the North") • **South** • **East** • **West**	• **Forward** (means "move forward one square in front of you") • **Right** (means "pivot a quarter-turn to the right without advancing") • **Left**

The allocentric language requires four vocabulary words while the autocentric language requires only three. Some students may give the instruction "Back", but the rover will end up in the same square if it backs up one square or goes "Right, Right, Forward". However, with these instructions, it will have changed the direction it is facing. For the rover to be facing its initial direction, the instruction must be "Right, Right, Forward, Right, Right."

It is also possible to reduce the number of words used in the autocentric language. "Left", for example, can be "Right, Right, Right". Here, two words suffice. For more clarity, you may want to keep three or four words, depending on what the students decide.
The teacher tells the students that the grammar is also very basic. There are no genders, plurals, moods or tenses. The only rule here applies to sequencing: when there is a series of two instructions, such as "Right Forward," this means that they must be done one after the other in the order they appear.
For greater clarity when reading and writing the instructions, students can decide (or not!) to introduce an additional rule, such as separating instructions with commas.
Finally, the class notes that these languages do not allow for other actions besides moving around a grid (e.g., you cannot display text or do calculations): programming languages are specialized. The teacher can tell the class that there are other similar languages (with few "words", few grammar rules, little or no ambiguity, etc.), such as music notation.

Introduction to errors

The teacher asks the students what happens if there is an error in the program (for example, if an instruction is left out). A concrete example can be used based on the rover's initial route (Handout 28). What happens if you skip an instruction? Regardless of the language used, the goal is not achieved. An error in an autocentric language can take you farther from the goal than an allocentric language error. However, in both cases, this is a bug and there are two things to take note of. First, the goal is not met, so the failure is just as serious in both cases. Second, if there are obstacles along the route (cracks, etc.) you do not want to make a mistake, however slight. Both bugs are equally problematic.
The class discusses the different possible causes of a bug. It may be from an error in the algorithm (the method), an error in the program (the expression of the algorithm in the chosen

language, such as a syntax error), or an equipment failure (related to a broken part in the machine or an error in transferring the instructions, such as in this activity).

Teaching notes

- The word "bug", first coined by Thomas Edison, began being used in the field of computer science after computer scientist Grace Hopper found the cause of a computer malfunction: an actual insect found in the machine's inner workings. Read more about Grace Hopper on page 10.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *In computer science, we invent and use languages.*
- *To give machines instructions, we use programming language, which can be understood by both machines and people.*
- *A programming language is different than a natural language.*
    - *It has very few words or grammar rules.*
    - *It leaves no room for ambiguity.*
- *There are several programming languages, created for different uses.*
- *A bug is an error in a program.*
- *A bug that seems minor can have major consequences.*

Students write down these conclusions in their science notebook. The teacher prepares a poster called "Defining computer science?". This poster will be completed during the project and will provide a general overview of this field (key ideas of language, algorithm, program, machine, data, etc.). They start by copying what the class learned about the notion of language during this lesson.

Further study (unplugged, Level 4)

This lesson can be extended with an exercise where pairs translate one language into another:

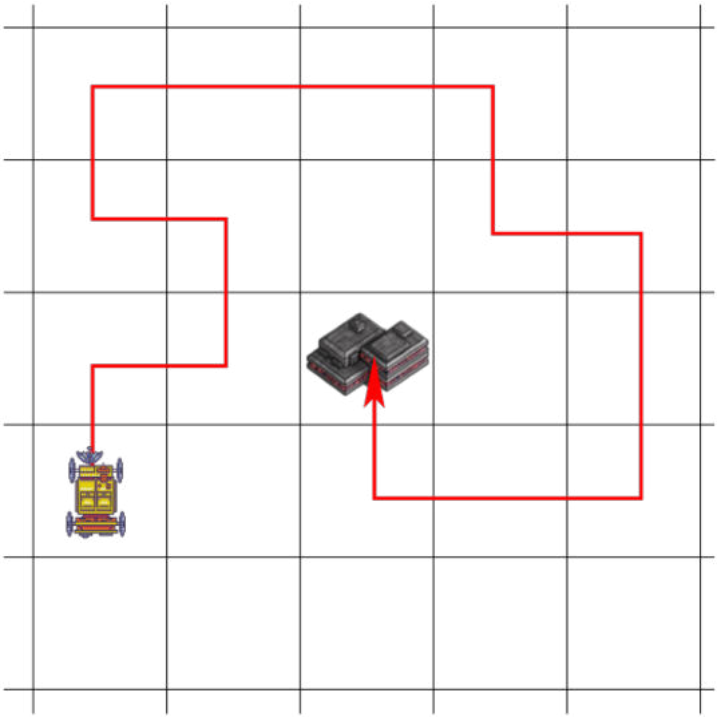
- Translate this series of instructions: North, West, North, East, East, East, South, East, South, West, West, North, North" (allocentric language) into an autocentric expression. Use a piece of grid paper to check that both expressions lead to the same result.
- Translate (for a rover initially facing North) this series of instructions: Right, Forward, Forward, Left, Forward, Left, Forward, Forward, Forward, Forward, Right, Forward into an allocentric expression and check the result on a piece of graph paper.

If the students finish this activity quickly, have them write the following notion down in their science notebooks:

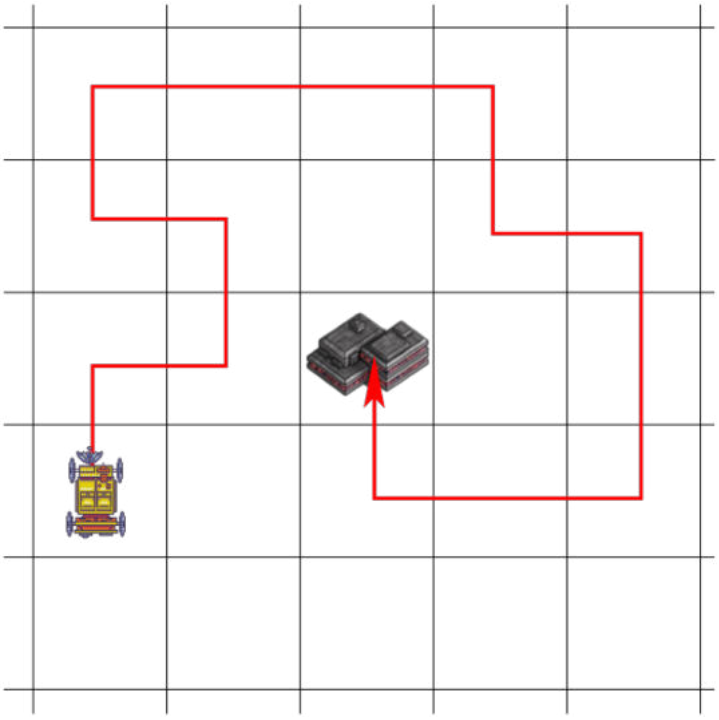
- *We can translate the same instructions from one language to another.*

HANDOUT 28
Operating a vehicle remotely

Instruction: Write a series of instructions to make the vehicle follow the red route square by square to return to base.



Instruction: Write a series of instructions to make the vehicle follow the red route square by square to return to base.



Lesson 2 - How to encode a message with numbers

Summary	Students must encode a textual message using only numbers. To do this, they make suggestions and then create a correspondence table for the letters and numbers for use by the entire class. They use this table to encode a message that they send as well as to decode a message they receive.
Key ideas *(see Conceptual scenario, page 204)*	"Information" • A letter can be represented by a number. • A text, which is made up of a series of letters, can be represented by a series of numbers. • Encoding of text refers to replacing its letters with corresponding numbers. Decoding is doing the opposite.
Inquiry-based methods	Experimentation
Equipment	For the classroom • Handout 29, page 218
Glossary	Correspondence table, encoding, decoding
Duration	1 hour 15 minutes

Scientific notes about the vocabulary before starting

- In common language, the terms "coding," "encoding," and "encrypting" are often misused or used interchangeably. The term "coding," for example, is sometimes used to mean "programming" (or "writing code"), "representing data" (for example, binary coding), or "modifying a message to make it incomprehensible" (as in a secret "code").
- Here, we will use these words according to their scientific definitions:
  - **Coding** means to represent data. During this lesson, students will represent a worded message using numbers. When they convert the letters into numbers, we will talk about **encoding**, and when they do the operation in reverse, we will talk about **decoding**. This is the aim of this lesson as well as the next two lessons on binary code.
  - **Encrypting** a message consists in changing it to make it incomprehensible to any unintended recipients (who do not have the key to decrypt it). This is the aim of Sequence 3.4, page 316.
  - The terms **enciphering/deciphering** (which we will avoid here) are generally used as synonyms for encrypting/decrypting, even more so in the everyday language, but there is a nuance. Cryptanalysis, or decipherment, consists in **breaking the cipher of** an encrypted message (the key is guessed by someone not meant to have access to it).

Introductory question

The teacher explains to the class that the mission control team communicates with the rover and the astronauts using electronic instruments. These instruments can only send and receive messages in numbers, not letters. Accordingly, the entire worded message to be sent must be converted into a series of numbers before being sent (this is the encoding operation), then converted back into a series of letters when received (this is the decoding operation). There cannot be any spaces or commas between the numbers: the numbers are all "stuck" to each other.
The question is: How can you encode worded messages as numbers and then decode them?

Activity: Encoding and decoding strategy (in pairs)

The teacher tells the students that the astronauts want to extend their outing past the initially stated time. However, the wind has picked up. They need to send a message to mission control with a message that starts with:

SEND WEATHER REPORT.

In pairs, the students try to figure out an encoding strategy for this text using numbers. They will not encode the message at this point but must simply figure out ways to do so.

Fifth Grade class, Christelle Crusberg (Champigny-sur-Marne)

When a pair says they have figured out an encoding strategy, the teacher secretly gives each student a slip of paper with a short worded message with a least one space or a period, such as “GO FASTER.”, “VERY WELL.” or “EVERYTHING IS FINE.”. They ask the students to encode the message for their partner, who must then decode it. The teacher reminds the students that the numbers in the encoded message must all be stuck together, without any spaces or punctuation. The two students verify whether the information has been properly transferred. If not, the pair works to identify the problem and change or improve their strategy.

Teaching notes

- The text to encode should contain only letters in ALL CAPS (without any accents), periods and spaces. Accordingly, it is likely that most of the students’ solutions will involve linking each letter to a number in alphabetical order (1 = A, 2 = B, etc. up to 26 for Z) with additional numbers to correspond to periods and spaces (e.g., 27 and 28 or 27 and 0).
- However, some groups may decide to encode using different numbers for uppercase letters (e.g., 1 to 26) and lowercase letters (e.g., 27 to 52). The issue of accented letters and punctuation other than periods may also arise. While the message to encode does not contain any numbers, some groups may try to take this possibility into account. They may decide to link the code numbers 0 to 9 to message numbers 0 to 9, with encoding of letters starting at 10.
- All of these possibilities should be accepted, even if for simplicity’s sake the message given to students will contain only letters and encoding will be identical for all variations of a letter (e.g., *A* and *a* are both encoded using 01).
- Some students may attempt to encode the message by using a less intuitive

correspondence between text and numbers, or even create a system that changes on a regular basis (e.g., one that changes every 10 letters). Tell these students to jot their ideas down for a later lesson (Sequence 3.4, page 316).

Group discussion

During the group discussion, the teacher asks a first pair of students to tell the class their solution. The other groups' solutions are compared (see Teaching notes above), weighing the pros and cons of each. Next, the class comes to a consensus: a one-to-one correspondence table (without any ambiguity during encoding or decoding) between the letters used in the messages and the numbers.
The correspondence table chosen for the next part of the lesson is the following (note that the actual table used can vary from one class to another, especially with regards to spaces and periods):

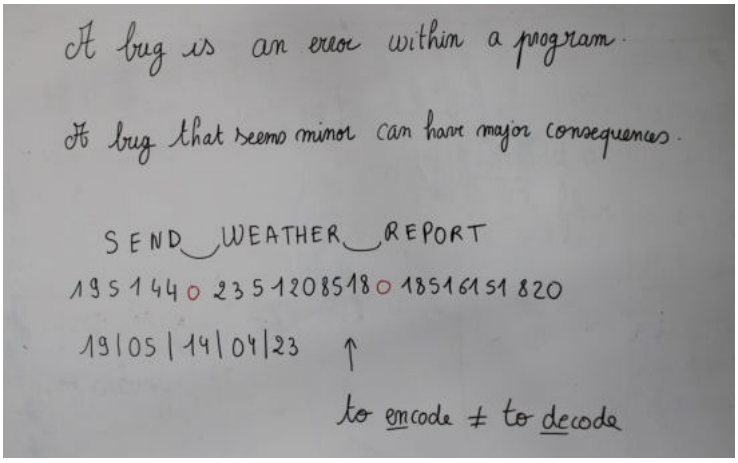
Letter	A	B	C	D	E	F	G	H
Number	01	02	03	04	05	06	07	08

Letter	I	J	K	L	M	N	O	P
Number	09	10	11	12	13	14	15	16

Letter	Q	R	S	T	U	V	W	X
Number	17	18	19	20	21	22	23	24

Letter	Y	Z	.	Space
Number	25	26	27	28

Note that the numbers 1 to 9 (corresponding to letters A to I) were written 01 to 09 so that all numbers used to code the letters are written using the same number of figures (i.e., two). This means that 0221 should be read as 02 21 and decoded as BU; 2201 should be read as 22 01 and decoded as VA. If letters A to I were written using single numbers 1 to 9, a text encoded as 221 could be read as 2 21 or 22 1, which could then be decoded as either BU or VA.



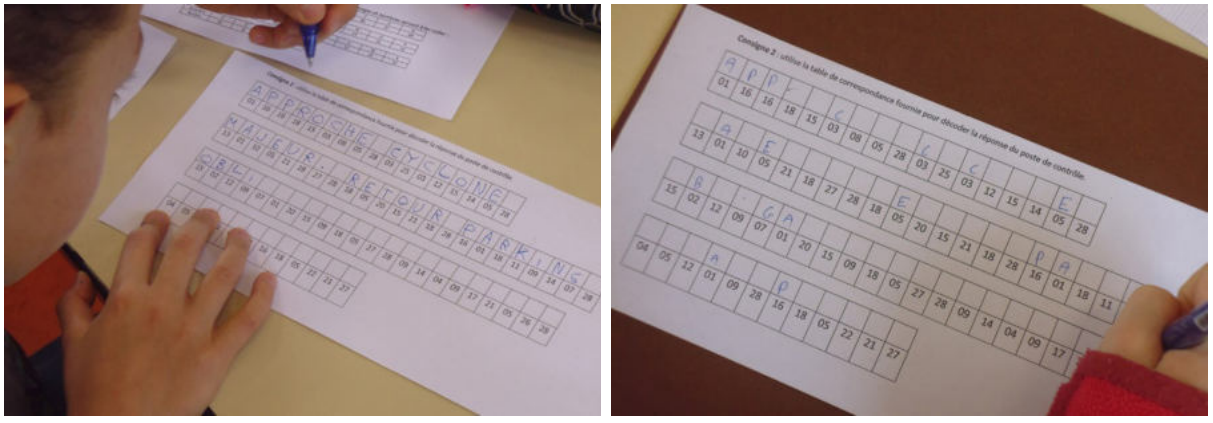
Discussion with students why it is important to encode the messages using two numbers for each letter.
Fourth Grade class, Carole Vinel (Paris)

Exercise - Encoding a message (in groups)

The teacher hands out the top of Handout 29 (correspondence table and Instruction 1) and asks the students to encode the message for mission control (Instruction 1) using the correspondence table chosen by the class.
Once the message is encoded, the teacher posts the results and announces that the message was sent to mission control. They emphasize that the original message and the encoded message contain the same information in two different forms.

Challenge - Decoding a message (in groups)

The teacher tells the class that they have just received a reply from mission control in an encoded message. They now need to decode the reply.
The teacher hands out the bottom of Handout 29 (Instruction 2) and gives the students time to decode the message.
The class shares the decoded message:
ALERT MAJOR CYCLONE. RETURN TO THE BASE. TELL EXPECTED DELAY.
The mission cannot be extended and the rover must return immediately to base.



Two decoding strategies: each letter one by one (left) or every occurrence of a letter in the text (right).
Fifth Grade class, Anne-Marie Lebrun (Bourg-la-Reine)

Teaching notes

- The encoding exercise and the decoding challenge can be done collaboratively: have different students encode and decode different lines of the message and then share their answers together with the class. However, we suggest handing out the entire message to students as this facilitates the logistics of the lesson, allows faster students to have work to do and gives students a complete record of what they did together.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *A letter can be represented by a number.*
- *A text, which is a series of letters, can be represented by a series of numbers.*
- *Encoding of text refers to replacing its letters with corresponding numbers. Decoding is doing the opposite.*

Students write down these conclusions in their science notebook. The teacher adds what the class learned about data and algorithms to the *"Defining computer science?"* poster.

Further study

With older students, the message to decode can be a little longer. For example, the message below is a pangram (all letters of the alphabet are included at least once):
WARNING. MAJOR CYCLONE APPROACHING FAST. CRAZY STRONG WINDS EXPECTED. MUST RETURN TO BASE QUICKLY. ADVISE ETA.

HANDOUT 29
Encode and Decode a message

Correspondence table for the letters in the messages and the code numbers:

Letter	A	B	C	D	E	F	G	H
Number	01	02	03	04	05	06	07	08

Letter	I	J	K	L	M	N	O	P
Number	09	10	11	12	13	14	15	16

Letter	Q	R	S	T	U	V	W	X
Number	17	18	19	20	21	22	23	24

Letter	Y	Z	.	Space
Number	25	26	27	28

Instruction 1: Use the correspondence table to encode the message for mission control.

A	L	E	R	T		M	A	J	O	R		C	Y	C	L	O	N	E	.

R	E	T	U	R	N		T	O		T	H	E		B	A	S	E	.

T	E	L	L		E	X	P	E	C	T	E	D		D	E	L	A	Y	.

Instruction 2: Use the correspondence table to decode the message for mission control.

01	12	05	18	20	28	13	01	10	15	18	28	03	25	03	12	15	14	05	27	28

18	05	20	21	18	14	28	20	15	28	20	08	05	28	02	01	19	05	27	28

20	05	12	12	28	05	24	16	05	03	20	05	04	28	04	05	12	01	25	27

Lesson 3 - How to code information in binary

Summary	Students must now use only two symbols (0 and 1) to transmit messages. They explore the ways of encoding different information (the four cardinal directions – North/South/East/West, the seven days of the week, etc.) by combining 0s and 1s. This is binary coding.
Key ideas *(see Conceptual scenario, page 204)*	"Information" • A computer represents all information using a code with only two symbols, 0 and 1, called bits: this is binary code. • Binary code makes it possible to represent all kinds of data, especially numbers and letters. • The more bits are combined, the greater the variety of elements can be represented.
Inquiry-based methods	Experimentation
Equipment	For each student • Handout 30, page 224
Glossary	List of elements, bit, binary code
Duration	1 hour 30 minutes

Introductory question

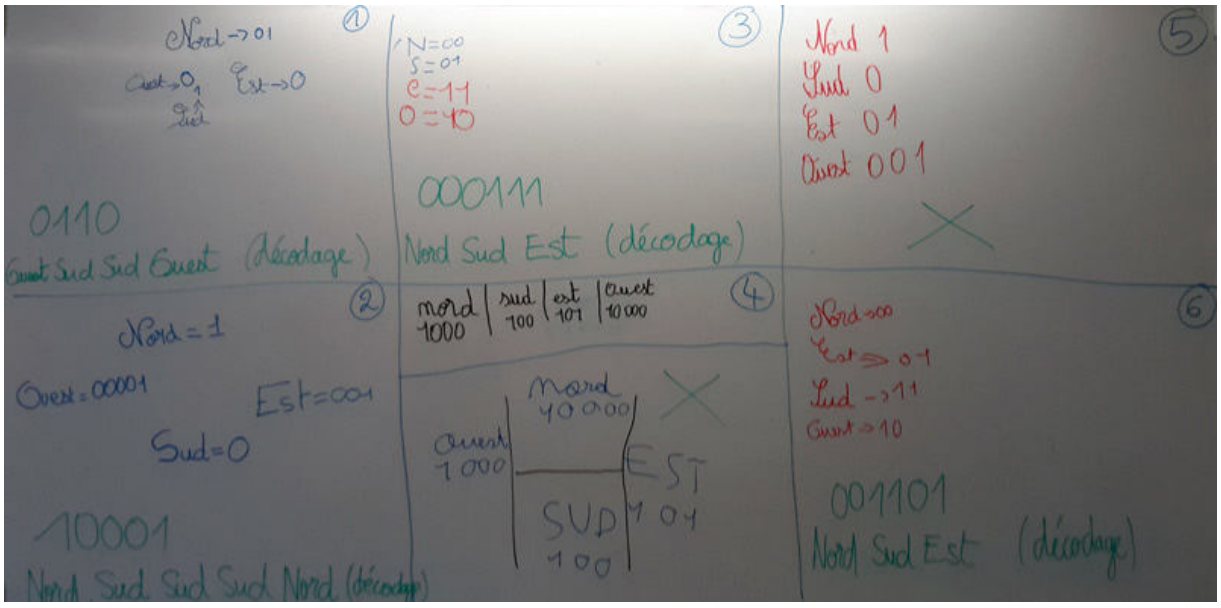
The teacher explains to the class that electronic instruments cannot directly transmit numbers: they transmit flows of light or electrical signals. These signals have only two states of being: NO (no signal) or YES (signal), also referred to as 0 and 1. Students must revise their coding system with this new limitation.
The question is: How do you encode information by using only 0s and 1s?

Activity: Find an encoding strategy using only 0s and 1s (in pairs and as a class)

The teacher reminds the students that the rover can be operated by mission control using the four words North/South/East/West. They ask the students to work in pairs to find a way to transmit these four words using only 0s and 1s.

Student solutions. Fourth/Gifth Grade class, Thyphaine Collignon (Vernaison)

After a few minutes, the groups share their solutions (see Teaching notes below).
The class discusses the suggestions and observes that those that work all use groupings of 0s and 1s.



Remind the group about the importance of encoding each piece of information using the same number of numbers for decoding. Fifth Grade class, Christelle Crusberg (Champigny-sur-Marne)

Teaching notes

- A first solution could be to replace:
  "North" with "00"
  "South" with "11"
  "East" with "01"
  "West" with "10"

  For this type of suggestion, each cardinal direction is coded with a pair of 0s or 1s. All combinations of two 0s and 1s are used. It is important to tell students that each group may have a different cardinal direction linked to a given number pair: all combinations are valid.
- A second solution could be to replace:
  "North" with "1000"
  "South" with "0100"
  "East" with "0010"
  "West" with "0001"

  For this type of suggestion, each cardinal direction is coded with a series of four 0s or 1s. Only certain combinations of four 0s and 1s are used (for example, the combination 0000 is not used).
- A third solution could be to replace:

«North» with:	«South» with:	«East» with:	«West» with:
1 0 0 0	0 0 0 1	0 0 1 0	0 1 0 0

For this type of suggestion, the teacher notes that it is a good idea, but that the 0s and 1s cannot be transmitted unless they are next to each other, so they cannot be spread out. This third suggestion is the equivalent of suggestion 2: for each cardinal direction, four 0s or 1s are transmitted, and only one is a 1. There are four possible positions for the 1, one for each cardinal direction.

Exercise: Choosing the number of 0s and 1s to combine in order to code the days of the week (in groups)

The teacher tells the students that all messages exchanged between the base and the rover have dates. They need to be able to indicate the day of the week (among other information) using 0s and 1s. The teacher asks the students to suggest a way to code the seven days of the week using the fewest combinations of 0s and 1s possible.

8 times Fifth Grade class, Christelle Crusberg (Champigny-sur-Marne)

Group discussion

The group discussion shows that combining three 0s or 1s (as below) is one way (of many equivalent possibilities) to accomplish the task:

000 for Monday
001 for Tuesday
010 for Wednesday
011 for Thursday
100 for Friday
101 for Saturday
110 for Sunday

There is even another combination (111) that has not been used.
The teacher guides the students in forming an oral conclusion using Handout 30 projected and handed out to all students:
We can code the elements of a list using a series of 0s or 1s, also called bits (a contraction of "binary digit").

- With a single bit, we can code all elements on a list that has no more than two elements (black/white or on/off are examples of lists with two elements): we link the 0 to one element and the 1 to the other element on the list (the class writes down list examples with two elements on Handout 30).
- With two combined bits, we can code all elements on a list that has no more than four elements (e.g., North/South/East/West) because there are four different ways to combine pairs of 0s and 1s: "00," "01," "10" and "11" (the class writes down examples of lists than can be coded using 2 bits but not 1 bit on Handout 30: these lists have three or four elements).
- With three combined bits, we can code all elements on a list that has no more than eight elements (e.g., the seven days of the week), because the only eight possible combinations of 0s and 1s are: "000," "001," "010," "011," "100," "101," "110" and "111" (the class writes down examples of lists than can be coded using 3 bits but not 2 bits on Handout 30: these lists have five to eight elements).
- The more bits are combined, the more different elements can be represented: a maximum of 16 elements combining 4 bits, 32 elements combining 5 bits, 64 elements combining 6 bits, *etc.*

Each student writes down on Handout 30 a list that can be coded using 4 bits but not 3 bits (the list should have between nine and 16 elements). Students may suggest the 12 months of the year, the 10 figures 0 through 9, the 12 hours on a clock, the 10 fingers of the hands, the names of nine of their cousins, 16 farm animals, etc.

Challenge (in pairs)

The teacher asks the students to find how many bits they need to combine to be able to code the 26 letters of the alphabet.
After a few minutes, the group discussion will show that four bits is not enough to code the 26 letters (four bits = maximum coding of 16 elements: 2x2x2x2=16), but five bits is (maximum coding of 32 elements: 2x2x2x2x2=32).

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *We can encode the elements on a list using a series of 0s and 1s, also called bits. This is what is called "binary code".*
- *The more bits are combined, the more different elements can be represented: A maximum of two elements with one bit, a maximum of 2x2=4 elements by combining two bits, 2x2x2=8 by combining three bits, 2x2x2x2=16 elements by combining four bits, 2x2x2x2x2=32 bits by combining five bits, 2x2x2x2x2x2x2x2=256 bits by combining eight bits, etc.*

8 times

- *Binary code makes it possible to represent all kinds of data, especially numbers and letters.*
- *A computer represents all types of information using binary code.*

The students write down these conclusions in their science notebook while the teacher adds this information to the *"Defining computer science"* poster.

Further study (for Level 4)

In Level 4, additional attention can be paid to how to easily find all combinations of 0s and 1s for a given number of bits. For example, if you ask students to find the 32 possible combinations of five bits, they will realize it is very difficult to find them all without repeating some of them. The teacher can suggest a recursive approach:

- With just one bit, the list is easy: 0, 1.
- For two bits, simply copy the one-bit list twice: once adding a 0 at the start, once adding a 1 at the start, to get 00, 01, 10, 11.
- For three bits, the process is the same. Copy the two-bit list twice: once adding a 0 at the start, once adding a 1 at the start to get 000, 001, 010, 011, 100, 101, 110, 111. The process is the same for four bits, five bits, etc.

HANDOUT 30
Code the elements of a list in binary

Instruction: In each empty bubble, write an example of a list for which all elements can be coded with one bit, two bits, three bits or four bits.

1 bits:

0 1	

2 bits:

00 01 10 11	

3 bits:

000 001 010 011 100 101 110 111	

4 bits:

0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111	

Challenge: Determine the minimum number of bits you need to combine to code each of the 26 letters of the alphabet.

Lesson 4 - How to encode and decode a binary message

Summary	Continuing on from the previous lesson, students apply what they have learned to encode a short worded message in binary code, then decode a message in binary code they receive.
Key ideas *(see Conceptual scenario, page 204)*	"Information" • Binary code makes it possible to represent all kinds of data, especially numbers and letters.
Inquiry-based methods	Experimentation
Equipment	For each group • Handout 31, page 228
Glossary	List of elements, bit, binary code
Duration	1 hour

Introductory question

The teacher reminds the students that they have received a message from the base that a storm is coming. The base team asked for information: the time rover will return to base.
Today, the students will code their reply in binary.

Activity: Encode a message for the base in binary (as a class, then in groups)

The teacher notes that the message the students must send to the base has only capital letters, spaces and periods (28 types of characters). They ask students to determine the smallest number of bits they need to encode each letter and suggest, if necessary, to look back at their notes from the previous lesson (Handout 30). The class agrees to limit coding to five bits per character.

The teacher hands out the top of Handout 31. They give students five minutes to create a correspondence between characters and five-bit combinations. During the group discussion, the class creates the following correspondence table:

5 bits	00000	00001	00010	00011	00100	00101	00110	00111
Letter	A	B	C	D	E	F	G	H

5 bits	01000	01001	01010	01011	01100	01101	01110	01111
Letter	I	J	K	L	M	N	O	P

5 bits	10000	10001	10010	10011	10100	10101	10110	10111
Letter	Q	R	S	T	U	V	W	X

5 bits	11000	11001	11010	11011	11100	11101	11110	11111
Letter	Y	Z	.	Space	No meaning (these can be used for other punctuation signs if desired)			

The teacher tasks the student groups with using this correspondence table to encode the following text in binary:

TEN MINUTES

They hand out the middle section of Handout 31. The class gets:

Letter	T	E	N		M	I	N	U	T	E	S
5-bit group	10011	00100	01101	11011	01100	01000	01101	10100	10011	00100	10010

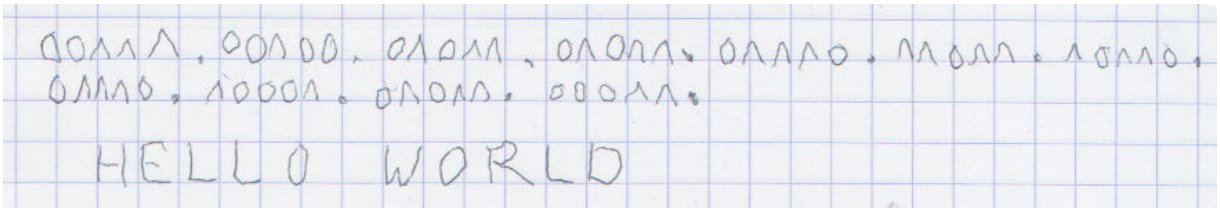
Challenge: Decoding a message sent by base (in pairs)

The teacher gives students the base's final reply (Handout 31) that the students must decode: 0111001010

Dividing the message into five-bit combinations gives students 01110 and 01010, which according to the table corresponds to the letters O and K. The message received from the base is "OK."

Individual exercise: Encoding and decoding binary messages

The teacher gives the students 10 or 15 minutes to encode short messages and to pass them to fellow students to decode. Students enjoy this activity, which helps them consolidate what they learned in class.



CM1 class of Carole Vinel (Paris)

Conclusion

The class reviews the conclusion from the previous lesson, especially with regard to the following idea: binary code makes it possible to represent all types of data, especially texts.

Further study (unplugged)

To help students better understand why electronic instruments often require binary data representation, an analogy can be made using electric circuits that include the same number of light bulbs as the number of bits used for coding. Each bulb is linked to a switch. You can place each switch in an open/closed position (either 0/1 or OFF/ON). These are the only two states possible for a switch. For electronic devices, the electronic components are not light bulbs or switches, but they work in a similar way: they can be powered via electricity or not. It is practical to distinguish between these two states and coding information in binary.

HANDOUT 31
How to encode and decode a binary message

Instruction

Suggest a correspondence table: fill in the table below by linking each five-bit combination to a character:

5 bits	00000	00001	00010	00011	00100	00101	00110	00111
Letter								

5 bits	01000	01001	01010	01011	01100	01101	01110	01111
Letter								

5 bits	10000	10001	10010	10011	10100	10101	10110	10111
Letter								

5 bits	11000	11001	11010	11011	11100	11101	11110	11111
Letter								

This is the correspondence table we will use going forward:

5 bits	00000	00001	00010	00011	00100	00101	00110	00111
Letter	A	B	C	D	E	F	G	H

5 bits	01000	01001	01010	01011	01100	01101	01110	01111
Letter	I	J	K	L	M	N	O	P

5 bits	10000	10001	10010	10011	10100	10101	10110	10111
Letter	Q	R	S	T	U	V	W	X

5 bits	11000	11001	11010	11011	11100	11101	11110	11111
Letter	Y	Z	.	Space	No meaning *(these can be used for other punctuation signs if desired)*			

Instruction

Encode the message below in binary to tell the mission control team that the rover is ten minutes from base:

Worded message:	T	E	N		M	I	N	U	T	E	S
Message in binary code											

Challenge: Direction

The base replied “0111001010”. Decode this message.

Sequence 2: Simulate the mission in Scratch

This sequence is dedicated to programming a video game to simulate our space exploration mission, following on from what was done in Sequence 1.

Sequence 2 is mainly carried out on the computer. However, there are a few unplugged exercises (not using the computer) aimed at fostering understanding of certain concepts such as variables and logical operators. These unplugged exercises should ideally take place outside of time dedicated to programming (e.g., math or English classes), so as to avoid interrupting the project.

Some tips before beginning a programming activity

Programming a video game is highly motivating for students, but a few precautions are necessary to ensure it goes well.

Scratch, an ideal environment for learning to code

There are a number of tools available for learning programming. We have chosen to use the software ***Scratch***[15] on the basis of its remarkable quality, its simplicity of use, it being free of charge, and it having a very active community, including in the education field (elementary and middle school).

This choice is, however, arbitrary. Teachers can teach the same concepts using another environment (such as *Snap* which is very similar to *Scratch* and is also free, or *Kodu* and *Tangara* which must be purchased).

We also propose an alternative to *Scratch* "alone" in classes that have a robot whose programming uses the same concepts but applied to a physical object. The robot *Thymio*, for example, is programmed using visual programming languages like Aseba/VPL and/or *Scratch* (in this regard, see Level 2, Sequence 3, starting from page 182).

Ensure mastery of some basic ICT skills

Programming requires interaction with a computer, meaning some basic skills are needed:

- Using keyboard and mouse (this is generally, but not always, the case taught from grade nine)
- Launching programs by double-clicking on their icon
- Saving work in a file, and saving the file in a folder
- Opening work saved earlier

If certain students do not have these basic skills, they can learn them through this programming project but may end up falling behind during the first exercises.

15 *Scratch* is available either online (without prior installation, but this requires a good Internet connection, at the address: https://scratch.mit.edu) or offline (requiring prior installation, after downloading the software at the address: https://scratch.mit.edu/scratch2download).

Working in half-groups

Ideally, there should be one computer for two students (3 at most). To make that possible, and so as to make it easier to manage classes during programming activities (during which teachers are very busy), we advise working in half-groups: half of the class programs while the other half works on something else independently.

Preparing the working environment

In order to save time, it is useful to prepare the working environment in advance:

- *Scratch* needs to be installed on all computers (or accessible online, see footnote on previous page)
- A shortcut to *Scratch* should be placed on the desktop
- Similarly, a dedicated folder for the project (and class) should be easily accessible, either on the desktop or on a USB flash drive for the group. This folder should contain the files needed for the project (images to import, copies of programs, etc.). We will provide all the files required on the project's dedicated website (see page 342).
- The most advanced users will be able to put the useful files directly into the sub-folders of the Scratch application for example: <Scratch>/Medias/Costumes/MissionMars and <Scratch>/Medias/Background/MissionMars, where <Scratch> is the application's installation path.

Doing the project yourself first

This is probably the most important tip, even if it seems obvious. It is essential for the teacher to take two or three hours of preparation time BEFORE the first class exercise to get used to *Scratch* and carry out the tasks that the students will have to perform during the project. Otherwise, they may not be able to help the students when they need it. This is not difficult (you can follow the process described in this sequence), and it is even quite fun!

Steps for the project

The recapitulation table below lists the different steps and, for each step, the various tasks that need to be carried out to build a video game. If the teacher chooses a different scenario for the game, they will obviously have to adapt their lesson.

Please note: In most Level 3 classes, carrying out the whole project will require six or seven one-hour sessions. Some pairs will produce a more comprehensive and complex game than others, but they will all put together a satisfactory and fulfilling project.

We recommend holding two sessions per week, at least at the beginning of the project, to avoid the students forgetting what they have learned from one session to the next, as programming is a genuinely new activity for them.

Levels of difficulty

To help give an idea of the difficulty of the various steps and activities, we use color-coded symbols:

●	Green activity: easy. All students should manage without difficulty.
●	Blue activity: medium difficulty. Most students should manage on their own, but some may need a little guidance.
●	Red activity: difficult. Most students will need some guidance (more or less depending on ability).
●	Black activity: very difficult. All students will need guidance. These are optional activities.

Unless otherwise stated, all the steps are plugged activities. The durations given are averages.

Step	Title	Page	Activity	
Step 1	**Introduction to the *Scratch* programming environment**	234	Activity 0: Demonstration of the final game by the teacher (5 minutes)	●
			Activity 1: Opening *Scratch* and getting to know its interface (10 minutes)	●
			Activity 2: Exploring *Scratch* independently (15 minutes)	●
			Activity 3: Short exercises (20 minutes)	●
Step 2	**Customizing the environment and saving all work**	243	Activity 1: Changing the sprite (5 minutes)	●
			Activity 2: Changing the backdrop (5 minutes)	●
			Activity 3: Saving your *Scratch* program (5 minutes)	●
Step 3	**Operating the rover**	246	Activity 1: Making the rover move to the left (10 minutes)	●
			Activity 2: Making the rover move in any direction (5 minutes)	●
			Activity 3: Driving the rover using the arrow keys (15 minutes)	●
			Activity 4: Bouncing off the edges (5 minutes)	●
			Activity 5: Reset the position of the rover (5 minutes)	●
			Activity 6: Understanding the X and Y coordinates of the rover (20 minutes)	●

Step 4 	**Gathering resources and managing scores**	253	Activity 1: Importing a resource (ice) in the form of a new sprite (5 minutes)	
			Activity 2: Making the resource say “Well done!” when touched by the rover (20 minutes)	
			Activity 3: Making the resource disappear when touched (10 minutes)	
			Activity 4: Creating a “score” variable (5 minutes)	
			Activity 5: Increasing the score when a resource is gathered (10 minutes)	
			Activity 6: Resetting the score to zero (10 minutes)	
			Activity 7: Making resources reappear in random positions (15 minutes)	
			Activity 8: Importing a new resource (plant) and repeating the same tasks as for the ice (20 minutes)	
Step 5  	**Plugged and unplugged activities to better understand certain algorithmic concepts** This step, which is optional, does not deal with programming the video game and should not interrupt it (it should be done as a parallel activity, such as during a math or language arts class).	260	Activity 1: Formative assessment on the loop concept (plugged activity, 10 to 20 minutes)	
			Activity 2: Set of cards to consolidate the notion of variable (unplugged activity, 1 hour)	
			Activity 3: A card game to work on logical operators (unplugged activity, 1 hour)	
			Activity 4: Understanding that an algorithm is not always perfect: the traveling salesman game (unplugged activity, 1 hour)	
Step 6 	**Avoiding obstacles and managing player lives**	277	Activity 1: Adding new sprites (5 minutes)	
			Activity 2: Creating and initializing a “number of lives” variable (5 minutes)	
			Activity 3: Losing a life when the rover touches the lava (30 minutes)	
			Activity 4: Repeat Activity 3 for the dune (10 minutes)	
Step 7 	**Ending the game: <<Game over>>**	281	Activity 1: Make «game over» appear when there are no more lives (15 minutes)	
			Activity 2: Stopping the game when “game over” appears (15 minutes)	

Step 8	**Adding challenges**	284	Activity 1: Make a countdown appear when the game starts (15 minutes)	●
			Activity 2: Limiting the game duration (15 minutes)	●
			Activity 3: Add a tornado that moves around randomly (15 minutes)	●
			Activity 4: Make the tornado bigger (15 minutes)	●
			Activity 5: Making the tornado go faster and faster (20 minutes)	●
			Activity 6: Simulate a torus world (joining the edges of the backdrop) (20 minutes)	●
			Activity 7: Preventing resources and traps from overlapping (20 minutes)	●
Step 9	**Further study in Scratch**	292	At this stage, the project is complete. Here we offer a few suggestions to explore other *Scratch* features, which could serve to support students' future personal projects.	

Step 1 - Introduction to the Scratch programming environment

Summary	The students learn about *Scratch*, a programming environment suited to elementary school students. They learn to launch the program and combine a few simple instructions.
Key ideas *(see Conceptual scenario, page 204)*	"Machines" • The machines around us merely execute orders (instructions). • By combining basic instructions, we can make them execute complex tasks. "Algorithms" • An algorithm is a method used to resolve a problem. • A loop allows the same action to be repeated multiple times. • Certain loops, known as "infinite loops," never stop. • Certain loops, known as "iterative loops," are repeated a predefined number of times. "Languages" • To give machines instructions, we use a programming language, which can be understood by both machines and people. • *Scratch* is a graphical programming environment that uses a simple language. • A program is the expression of an algorithm in a programming language. • Certain instructions are only executed when an event is triggered. This is known as event-driven programming. • Certain instructions are executed one after the other. This is known as sequential programming. • The execution of a program is reproducible (if neither the instructions nor the data to manipulate change, the program always gives the same result).
Equipment	For the class • Projector • Enlarged version (A3 or A4) of Handout 32 on page 242 For each pair of students • A computer with Internet access or, if there is no good connection available, a computer with *Scratch* preinstalled (see footnote on page 229). For each student • Handout 32, page 242
Glossary	Program, script, sprite, instruction, event
Duration	1 hour

Teaching notes

- You learn to program by programming, not by watching someone else program. It is interesting to consider the same problem in pairs (probably more so than to program alone), but it is important to be active. We therefore recommend placing the students in small groups in front of computers (ideally two students per machine) and asking them to "switch over" (pass the keyboard and mouse to their neighbor) every 10-15 minutes.
- If possible, we recommend organizing the class in half-groups, so as to avoid having too many pairs to manage at once. While half the class works on *Scratch*, the other should do something else independently.
- If possible, two *Scratch* sessions should be organized weekly, at least at the beginning.
- This step of getting to know *Scratch* is deliberately very directive (activities 0 and 1 are actually a demonstration by the teacher!). It is the only step presented in this form. All pairs will have to carry out a series of basic tasks. At the end of each activity, a group discussion ensures that everyone has understood and knows what to do. The other steps will be less directive, as students become more independent and progress at their own pace.
- To save time, switch on the computers before the session begins.

Activity 0: Demonstration of the final game by the teacher (5 minutes)

The teacher explains that the aim is to simulate an exploration mission (seeing as we can not go for real) through a video game we are going to program ourselves.
From their computer, the teacher opens *Scratch* and shows the "final" video game (that they have produced in advance[16]), simply demonstrating the game without explaining how the program works.

Screenshot: final version of the video game produced by the teacher
(Please note: a model program and an export in Flash format are available on the project website, page 342).

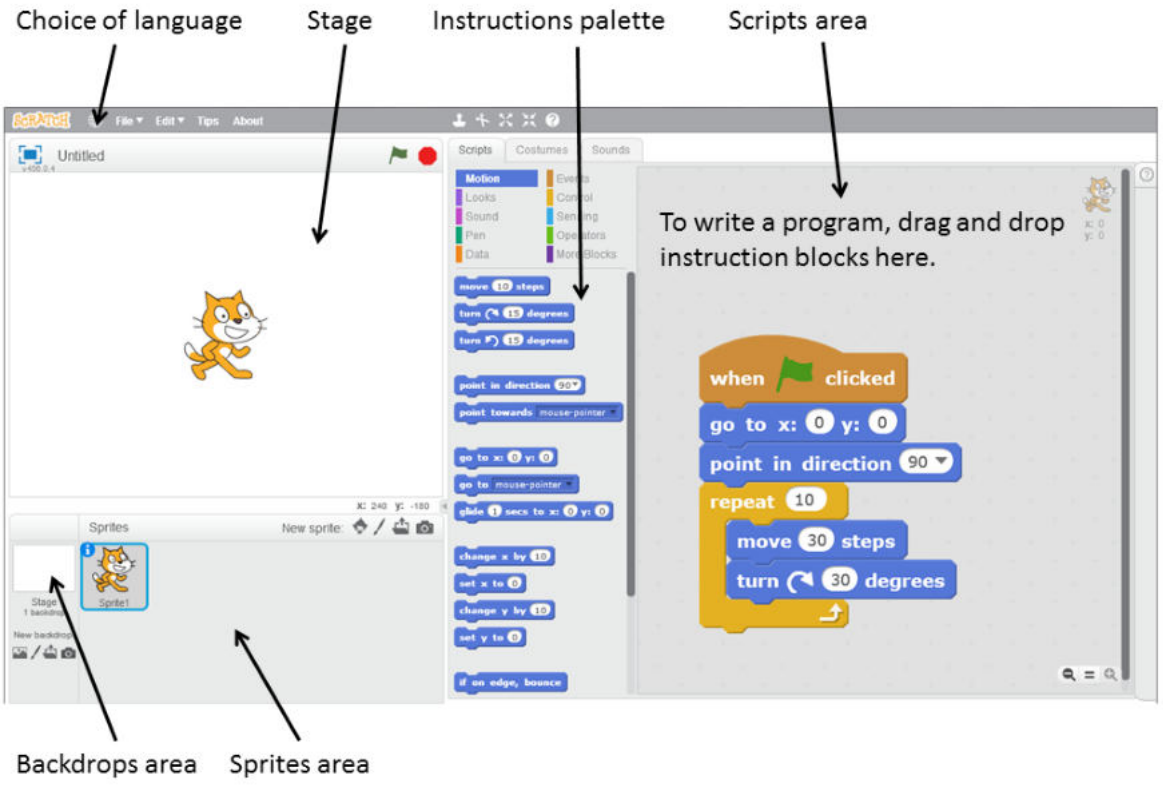
16 Reminder: it is essential for the teacher to complete the exercise before proposing it to the students! All they have to do is follow the steps in the sequence. Teachers with no *Scratch* experience will need about 3 hours to do the whole project, including black activities.

Teaching notes

- This demonstration is very important, as it is highly motivating and reassuring for students: they are actually going to program a "real" video game! It also helps tie in this programming activity with the unplugged activities in Sequence 1.
  "Here is our rover, which we are going to learn to move. It will collect resources to allow humans to live in the base, such as water and food (for each resource collected, the score increases). But watch out for traps! If the rover falls into a trap, it loses a life. Once it has no lives left, the game is over."
- It is important to explain to students that such a game cannot be programmed in a single session but will require several (typically six or seven sessions, depending on their level and ambition).

Activity 1: Opening Scratch and getting to know its interface (10 minutes)

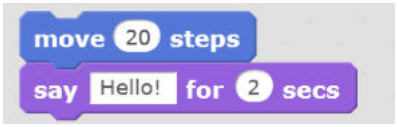
Each pair of students should open *Scratch* by double-clicking on the icon. They should cancel any update windows.
Scratch is usually configured in English the first time it is used on a machine. If necessary, students can change the language very easily by clicking on the globe icon (top left, next to the *Scratch* logo).



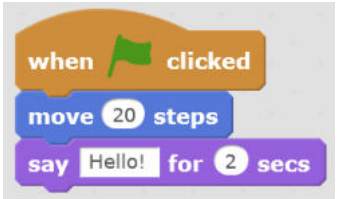
The teacher should explain to the students that *Scratch* is a programming language designed specifically for learners. When you open the application, there is a sprite (a cat) on the screen. You can give it simple instructions.

The teacher carries out a short demonstration (at the end of the session, the students will repeat these exercises).
For example, to ask the cat to move 10 steps, you can simply drag the instruction block "move 10 steps" from the scripts tab into the scripts area. If you then click on the block, you will see that the cat does move 10 steps (1 step = 1 pixel on the screen).

If you want to move 20 steps, you can simply replace "10" with "20" by clicking on the number. If you now want the cat to move 20 steps and then say "Hello!," simply add the new instruction at the end of the program. The instruction "Say Hello!" is located under the "Looks" category of the scripts tab. You can replace the text "Hello!" with any words you like by clicking on it. You can write a program simply by snapping together the instructions in order.



If you want the cat to do that each time you click on the green flag (top right of the stage; the green flag launches the program), then add the instruction "when green flag clicked," which is to be found under the "Events" category on the scripts menu. The result is:



Finally, the teacher should show students how to delete an instruction (or a whole stack of instructions): simply drag the instruction (or stack) from the scripts area towards the scripts tab.
The teacher should very briefly introduce the *Scratch* interface, which includes:

- A **"stage"**: this is where the "game" (or, more generally, the program – *Scratch* is not only for games!) is executed.
- A **"sprites" area**: sprites are the characters or objects that are controlled in the program (they can move, change shape, speak, interact with other sprites, etc.). When you open *Scratch*, only one sprite is displayed on the screen: a cat (other sprites will be added later, and the cat will be deleted).
- A **"backdrops" area**, right next to the sprites: the backdrop is static, unlike the sprites that can move. Be default, the backdrop in *Scratch* is a plain white screen (this will be changed later).
- A **"scripts" tab**, which offers:
  - A **scripts tab** (central column, to the right of the stage). This is where the instructions (or "blocks") we will use to build our program are to be found. There is a wide variety of instructions, grouped by color (e.g. anything to do with the motion of the sprite is in the deep blue category, anything to do with its looks is in the purple category, etc.).
  - A **"scripts" area,** to the right of the scripts tab. This is where the program is written, simply by dragging and dropping instructions from the tab into this area.
- The other tabs (Costumes, Sounds) are not useful for now.

Activity 2: Exploring Scratch independently (15 minutes)

The students have 15 minutes to explore *Scratch* by themselves. For now, they should not try to change the stage or the sprite (that's for the next step, on page 243). They should simply try out simple instructions and order them to see what happens. The teacher should encourage them to explore the various categories of instructions, including:

- "Motion" (deep blue)
- "Looks" (purple)
- "Events" (brown)
- "Control" (gold)

Third-grade class of Emmanuelle Wilgenbus (Antony, France)

Teaching notes

- Establish a rule of taking turns from the outset, so that the same student does not always control the keyboard and mouse.

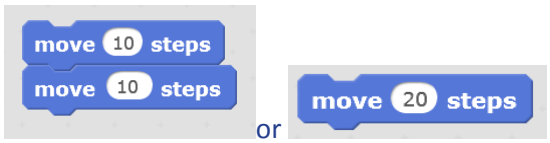
Activity 3: Short exercises (20 minutes)

The teacher should give a series of short exercises (mostly going over what was done previously in the demonstration) for the students to carry out. After each exercise, a quick group discussion will ensure everyone knows how to do the exercise.

Exercise 1 – Make the cat move 10 steps

Exercise 2 – Make the cat move 20 steps

Two possible solutions:

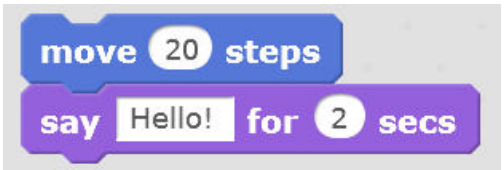


The second solution is better, as it is more elegant and easier to read.

Exercise 3 – Return the cat to the center of the stage

Some students will no doubt get the trick, but most will need to be shown. However, it is essential for them to see this instruction now, as their movements will eventually send the cat off the screen and they will not know how to get it back.

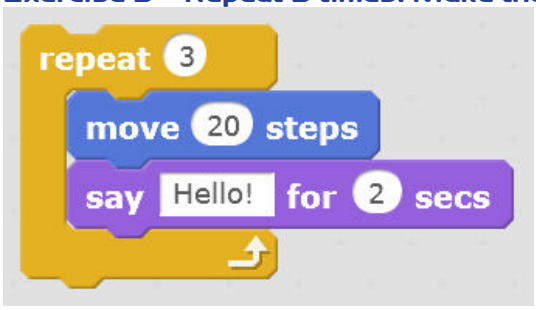
Exercise 4 – Make the cat move 20 steps and say <<Hello>>



Stress that "say" hello means "write" hello here: a speech bubble should appear on the screen with the text "Hello!" inside – we do not want to make the cat talk!

Please note: It is possible to make the cat "talk" (play musical notes or a sound file imported into or recorded in *Scratch*), but it is highly unadvisable in class.

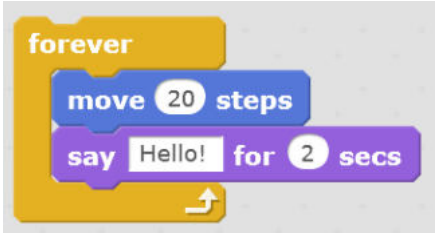
Exercise 5 – Repeat 3 times: Make the cat move 20 steps and say <<Hello>>



Encourage students who do not find what they need to look in the "control" category (gold). They will find a similar instruction "repeat 10" that can easily be changed by replacing "10" with "3." That loop fits around the other instructions. Everything within the loop is executed three times.

The cat stops for 2 seconds between each movement. To reduce this pause, simply shorten the period during which it says "Hello!" (if you write 0.5 instead of 2, the cat will only stop for half a second each time).

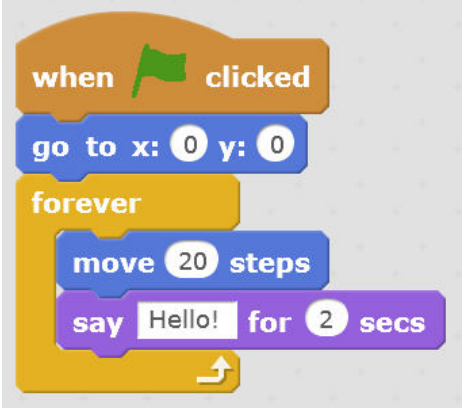
Exercise 6 – Repeat forever: Make the cat move 20 steps and say <<Hello>>



This is very simple to do, using the same principle as the previous exercise but with a different sort of loop.

Exercise 7: Same thing when you click on the green flag

All you really have to do is add the instruction "when green flag clicked" (from the "events" category), but it is even better if you ask the cat to start again from the center of the stage.



This is when to explain the purpose of the **red button** (next to the green flag). Clicking on the red button stops the execution of the program (which would otherwise never stop in this case).

Review and conclusion

The class summarizes together what they learned in this lesson, including by listing the *Scratch* instructions they now all know.

- *move 10 steps*
- *go to x: _ y: _*
- *say Hello! For 2 secs*
- *repeat 10*
- *forever*
- *when green flag clicked*

It can be very useful for students to color the *Scratch* instructions one by one as they discover and understand them. After each new step, you can therefore see the progress of each pair and the whole class.

Handout 32 “Some useful *Scratch* instructions” on page 242 can be photocopied for each student and enlarged for the class. This handout will be enriched later, once students have used tests (page 254), sensors (page 254), variables (page 256) and operators (page 258).

HANDOUT 32
Some useful Scratch instructions

Step 2 - Customizing the environment and saving all work

Summary	The students learn to customize *Scratch* (sprite and backdrop) and save their work to be used again later. They discuss the different steps they will follow to create their video game.
Key ideas *(see Conceptual scenario, page 204)*	Same as previous session
Equipment	Same as previous session

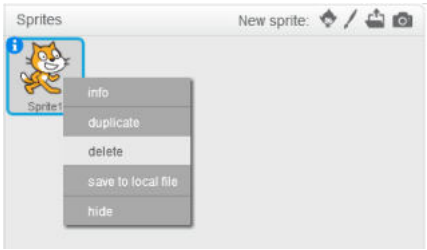
Teaching notes

- The purpose of this teaching guide is to teach Computer Science (more specifically programming, here). We will not, therefore, describe possible extension activities such as those that can be done in art or ICT class to design a personalized rover or backdrop for our video game. In this step, we propose simply to import components that we provide for classes. That has two benefits: it saves time and generates a degree of consistency between the students' programs, making it easier to compare them.
- Of course, the teacher can decide to spend an hour in class with the students designing these items. In this case, you need to be careful about the backdrop: it has to be relatively uniform, as decorative elements (obstacles and resources) will be added later in the form of other sprites.

Activity 1: Changing the sprite (5 minutes)

The teacher explains that it is possible to get rid of the "cat" sprite and create another in its place that fits in better with our space mission project: a rover.

- To delete the cat, right-click on its icon in the sprites area and select "delete."



- There are four different ways to create a new sprite, which are accessible from the "new sprite" toolbar, to the bottom right of the stage (the method we recommend here is described in bold).

	*Choose sprite from library* *Scratch* comes with a library of about 100 predefined sprites. These sprites can be practical for students' projects, but have very mixed styles.
	*Paint new sprite* *Scratch* has an integrated drawing tool that students can use to create their own rover "by hand."
	***Upload sprite from file*** This is the option we recommend here, as the aim of this project is not to teach drawing on the computer, but to teach programming. We recommend teachers provide the files required for the project (in this case, the rover) in a folder that is easily accessible for the students.[4] There are two options for the rover: a "square" rover and a more elongated one. We will be using the latter in the screenshots.
	*New sprite from camera* This can be a very practical tool for personal projects (you can add your own face as a new sprite), but it is not useful for this project.

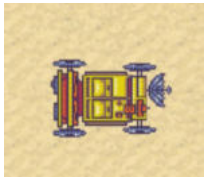
Teaching notes

- We have noticed that there can be problems importing the sprite from a file on certain computers. If the import fails, there is a very simple way to fix the problem: save all ongoing work, close *Scratch*, reopen *Scratch*, and try importing again. After that little workaround, it works!

Activity 2: Changing the backdrop (5 minutes)

Similarly to above, it is possible to change the stage's backdrop, using an image from the library or an image from a file provided by the use, or by painting a backdrop yourself.

We recommend choosing the option "Upload backdrop from file," selecting the file *martian_soil.png* (from the "Stages" folder in the files provided). Here is a preview of the rover, on the chosen backdrop.



Activity 3: Saving your Scratch program (5 minutes)

The teacher explains that the current program has to be saved (even if there is not yet much in there) to avoid having to redo everything during the next step.

Option 1: *Scratch* installed locally	Option 2: *Scratch* used online
Save by opening the "file" menu and then on the option "save." Then browse to the folder used for the project and class (once again, we strongly recommend a Desktop shortcut) and choose a filename. This filename could, for example, contain the students' names, so that they can easily find their own programs later. Import by either double-clicking on the saved file (which opens *Scratch*) or by opening *Scratch* and then clicking on "file" and the option "open."	Save by opening the "file" menu and then on the option "Download to your computer." You can import your file later from the same menu by clicking on the option "Upload from your computer."

Review and conclusion

The class goes over what it has learned to do in *Scratch*: importing sprites and backdrops, and saving and resuming work.

The teacher can show the demonstration of the "final" game again, so as to help the students visualize the activities they still need to complete. For example:

- Driving the rover using arrow keys
- Importing other sprites as resources and obstacles
- Making the player win points for collecting resources, and lose lives when they hit obstacles
- Making resources disappear once they are collected and reappear elsewhere on the screen (at a random position)
- Make a tornado move randomly around the stage
- Make the game end when the player has no lives left (with the text "Game Over" that appears, and all the rest disappearing).

Other activities are also possible:

- Adding a countdown to spice up the game (collect as many resources as possible in a given time)
- Personalize the game by painting your own sprites and backdrop
- Etc.

These steps will be looked at again later, and broken down into basic tasks where necessary. Each pair can move forward at their own pace, as the main thing is to have a playable game at the end of the project.

Step 3 - Operating the rover

Summary	The students create their first program, letting them operate the rover using the arrow keys. They learn about the coordinates system.
Key ideas *(see Conceptual scenario, page 204)*	Same as previous sessions
Equipment	Same as previous sessions Plus, for each student: • a photocopy of Handout 33, page 252

Once each pair has successfully imported their program (which so far contains only the rover and backdrop), the class goes back over the list of steps needed to program the video game. The first thing to do is to drive the rover. The simplest way is to drive the rover using the keyboard arrow keys.

Teaching notes

- The students will again need to be guided during this step. They will then have learned the reflexes they need to be more independent, and each pair can progress at its own pace.

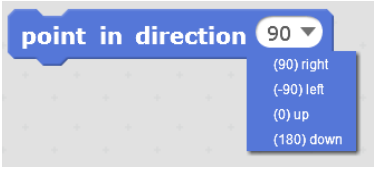
Activity 1: Making the rover move to the left (10 minutes)

The students already know how to move the rover to the right: they just have to tell it to move, as it faces the right by default. Moving it to the left is a little more difficult, as the students first have to ask the rover to point to the left before moving.

They should work independently and feel their way through, with the teacher regularly checking in on groups to ensure nobody is stuck. The teacher can guide them by suggesting looking for a "point" instruction.

Teaching notes

- There are two instructions of this type:
- "Point towards," which does not help us as the only available option when you click on the little arrow is "mouse-pointer" (meaning the sprite would point towards the position of the mouse pointer).
- "Point in direction...," which is what we need here. When you click on the number in the instruction (the default number is generally "90"), a help bubble explains that the angle 0 is the top of the screen, 90 is the right, etc.
  So here, you need to choose <<Point in direction -90>>



In the end, the program to move the sprite towards the left is:

Activity 2: Making the rover move in any direction (5 minutes)

The students should now be able to move the rover in any direction (right, left, up and down) using exactly the same method described above.

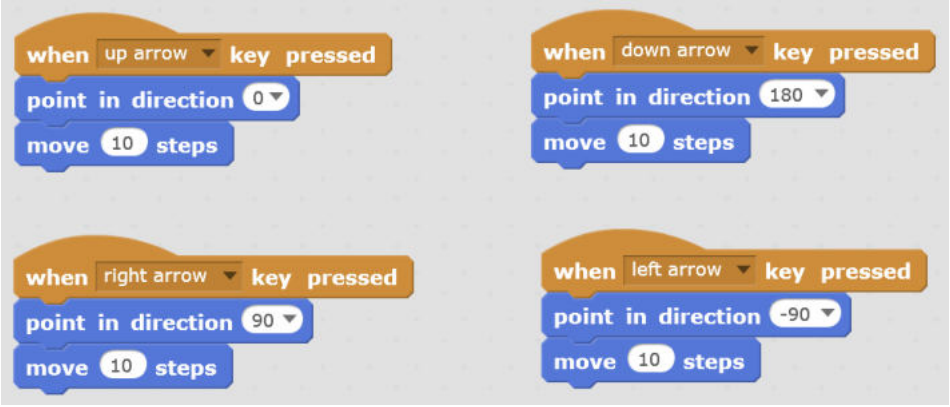
Please note: You now need the instruction “point in direction 90” to tell it to go to the right, as the rover no longer points in that direction by default.

Activity 3: Driving the rover using the arrow keys (15 minutes)

The students will now make the rover move when they press the arrow keys on the keyboard. They should try to work out how independently. Some will remember the instruction “when green flag clicked,” which they saw during the first *Scratch* session. That was an event that triggered an action.

Here too, an event is needed: the action is triggered when a key is pressed. The command **“when (space) is pressed”** is what we need, except “space” should be replaced with one of the arrow keys (right arrow to move to the right). That is done the same way as before:

In the end, the rover's scripts area will contain four scripts, each describing movement in a specific direction. The program may look like this:



Teaching notes

- You can see here that several scripts can co-exist in the same program. Each is executed when the trigger event (here, a key press) is detected.
- Some students may panic, thinking that their program has disappeared following a misstep. That is generally not the case (the program is not deleted). They have simply clicked on the stage (which has its own scripts area, but which is empty because we have not put anything in there yet) instead of the sprite. Sometimes, they have clicked on the sprite, but have clicked on the "costumes" tab instead of the "scripts" tab. All they have to do is click again on the sprite, and then on the "scripts" tab to display the program once more!

Fourth-grade class of Caroline Vinel (Paris)

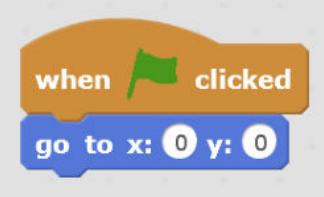
Activity 4: Bouncing off the edges (5 minutes)

The students work out how to make the rover bounce of the edges of the stage. For example, if the rover drives towards the right and reaches the right-hand edge of the screen, it has to bounce back so as not to leave the screen.

That is very easy to do, by adding the instruction “if on edge, bounce” to each of the scripts produced previously. For example:

Activity 5: Reset the position of the rover (5 minutes)

The teacher reminds the students that, when you start the program (green flag), the rover should be situated at the center of the screen. The students easily remember the instructions they saw during the first *Scratch* session.



Teaching notes

- You can now launch the program by clicking on the green flag. If you prefer, you can hide programs during execution by clicking on the “full screen” button at the top left of the stage.
- Always remember to save your work! (see page 245).

Activity 6: Understanding the X and Y coordinates of the rover (20 minutes)

The previous activity demonstrated the X and Y coordinates of the rover, through the instruction “go to X: ... Y: ...” The next steps (resources, traps, etc.) will require students to make use of these coordinates, so it is important to understand how they work.

The teacher asks the students to observe the X and Y coordinates displayed at the bottom right of the stage. They will notice that the coordinates displayed change depending on the position of the mouse.

- What are the values of X and Y when the mouse is at the center of the stage? (answer: X=0, Y=0)
- And when the mouse is at the right-hand edge? (answer: X = 240. Y can have any value, depending on the position of the mouse)
- And when the mouse is at the left-hand edge? (X=-240)
- And when the mouse is at the top edge? (Y = 180) or at the bottom edge? (Y=-180)

Together, the class concludes that X indicates the position on the horizontal axis (imaginary, invisible axis) and that Y indicates the position on the vertical axis (also imaginary).

The students may realize that, in the "motion" category of the scripts tab, many instructions use the X and Y variables. In these cases, it is not a question of the position of the mouse, but of that of the selected sprite. The rover has its own set of X and Y variables.

The teacher can hand out Handout 33 to each student and suggest some little exercises:
- Place the sprite on the stage, at the coordinates X =100, Y = 100
- What happens if you add 50 to X? Where is the sprite now?
- And what if you now set Y to 0? Where is the sprite?

Teaching notes

- To help students understand these coordinates, the teacher can draw a parallel with what the students have already seen in geography: latitude and longitude. Here, the unit is no longer the degree (we are not using angles), but the pixel. Similarly, in a game of battleships, ship locations are identified using two coordinates (a letter and a number). Give or take the unit or symbol, this is exactly the same: identifying the position of a point on a surface, which requires two coordinates as a surface is a two-dimensional space.
- Similarly, it may be useful for certain students to use concrete examples to demonstrate negative numbers. There is no shortage of examples, from dates and temperatures (what is "-10°C"? Is it hotter or colder than "0°C"? And is "-20°C" hotter or colder than "-10°C"?).

Conclusion and lesson recapitulation activity

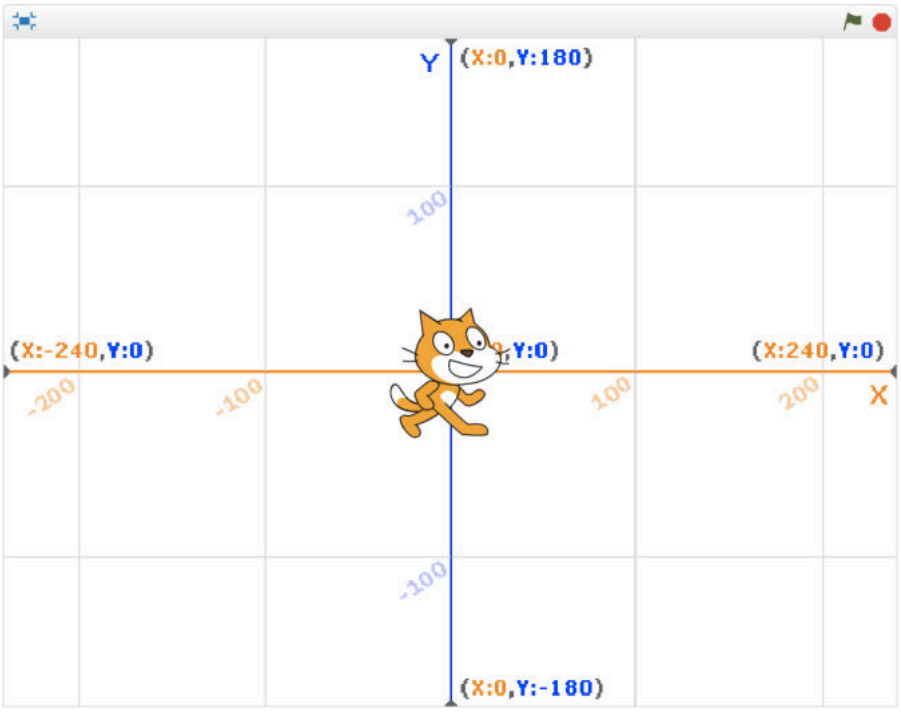
At the end of this session, it is important to recapitulation the new *Scratch* instructions the students have learned to use:
- Point in direction (90)
- When (space) key pressed
- When (green flag) clicked
- Go to (X =…, Y = …)

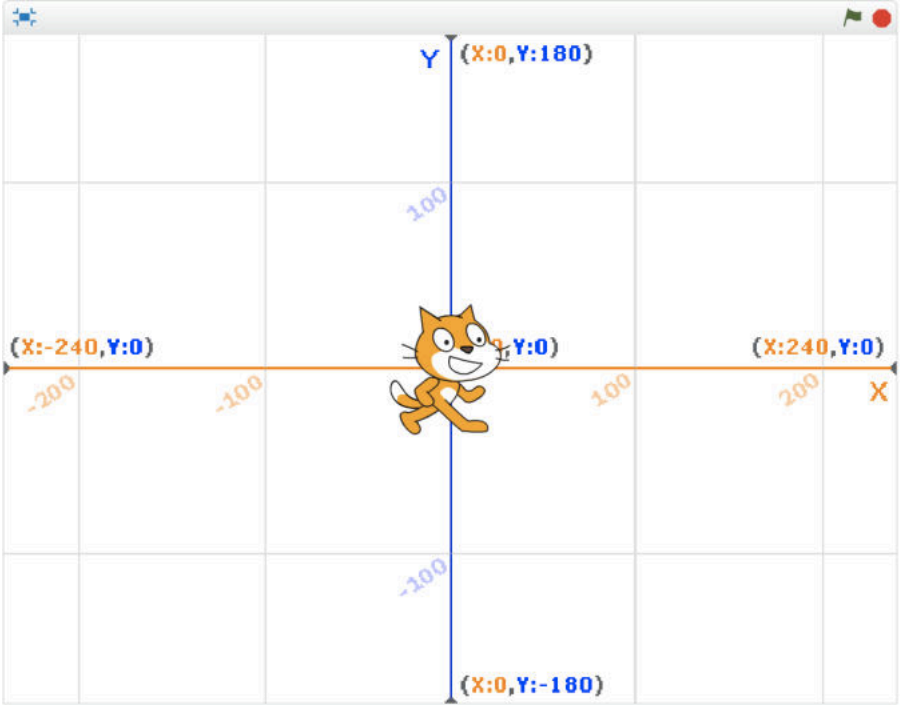
The students should color these instructions on Handout 32 (page 242) which they have already used.
Moreover, this session is an opportunity to step back from programming activities and go back over a few concepts:
- *A program is an algorithm expressed in a special language, known as a programming language. These are comprehensible by both machines and human beings.*

- *The execution of a program is reproducible (if neither the instructions nor the data to manipulate change, the program always gives the same result).*
- *Computers merely execute the instructions they are given, no more, no less.*
- *The position of an item on the screen is identified using two coordinates. In Scratch, they are called X and Y. X varies between -240 and 240, while Y varies between -180 and 180.*

The students write down these conclusions in their science notebooks. The teacher updates the “Information” section of the poster entitled “Defining computer science.”

HANDOUT 33
X and Y coordinates in *Scratch*





Step 4 - Gathering resources and managing scores

Summary	Students complete their program by adding resources to gather (new sprites) and creating a variable for their score (this score increases as more resources are gathered). They learn to program conditional instructions (if ... then) and use sensors.
Key ideas *(see Conceptual scenario, page 204)*	"Algorithms" • A loop allows the same action to be repeated multiple times. • A test can be used to choose which action to carry out if a condition is true or not. • A condition is an expression that can be either true or false. "Machines" • A variable is a name given to a memory area. It is used to store a value and reuse or change it later. "Languages" • Certain instructions are executed at the same time as others. This is known as parallel programming.
Equipment	Same as previous sessions
Glossary	Loop, test, sensor, random number

Teaching notes

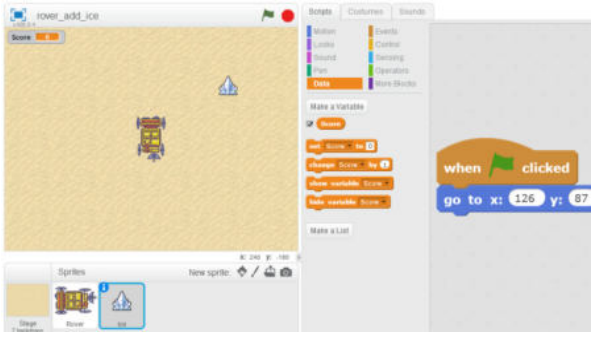
- This is the project's central step, as students will have to discover and use a number of new concepts to manage resources: tests, variables, sensors and operators.
- In order to avoid tackling all these new ideas at once, we suggest splitting the step into several basic tasks. Even then, activity 2 is relatively complex and will require guidance from the teacher.
- From now on, it is unrealistic to expect all students to progress at the same pace (otherwise, the most advanced groups will quickly become bored and distracted). Breaking down learning into steps and tasks helps facilitate the teacher's management of the class: everyone has something to do, whatever stage they're at.
- We recommend putting together pairs with a similar level rather than pairs where a struggling student and a more advanced student work together (as in the latter case, experience has shown that struggling students become passive and let the others do the work).

The teacher explains that the mission has to collect various resources to survive, including water (in the form of ice, on this planet) and food (in the form of plants). The rover will have to pick them up, and a "score" will be used to count the number of resource items collected.

Activity 1: Importing a resource (ice) in the form of a new sprite (5 minutes)

The students return to the program they saved during the previous session and import a new sprite: the ice (image available in the "Sprites" subfolder of the files provided, as previously). The teacher reminds the students to manually set the position of this resource, as they did for the rover in the previous session. They can position the ice anywhere on the stage, so long as the sprites (ice and rover) do not overlap.

For example:



Teaching notes

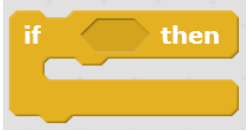
- Here we notice that **each sprite has its own scripts area** (switch between the rover program and the ice program by clicking on the sprite in question). There can be as many programs as there are sprites: all programs are executed in parallel.

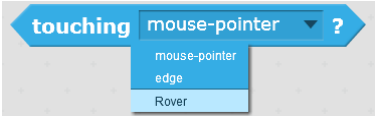
Activity 2: Making the resource say <Well done!> when touched by the rover (20 minutes)

Each pair needs to amend the ice program so that it says "Well done!" when it is touched by the rover. The teacher should let them try to work it out, then check on the groups to guide them if they get stuck.

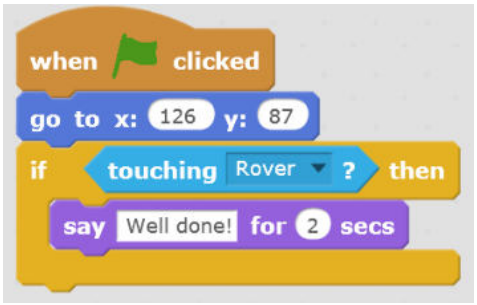
This task involves:

- Knowing how to make the sprite say "Well done!" (all the students know at this stage)
- Knowing how to launch an instruction only once a given condition is fulfilled. That is done via the **"Control"** category, where the instruction **"if ... then"** is to be found
- Knowing how to detect when one sprite touches another. That is done via the **"sensing"** category of the scripts tab (**"touching ... ?"** instruction). Once the instruction has been selected, clicking on the arrow opens a list of the sprites already created. Here, we are in the ice program and want to test whether this sprite is touching the rover, so we click on "rover."





The ice sprite program then becomes:

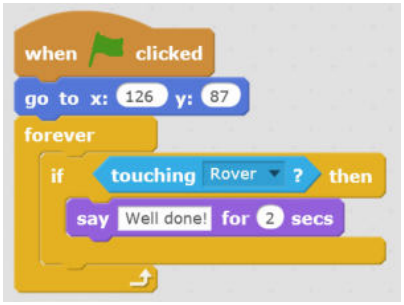


Unfortunately, when you execute the program and the rover is driven towards the ice, it does not work. Why? The class can discuss together what the program does, step by step:

- The ice is placed in the chosen position
- A test is carried out: if the ice touches the rover, then it says "Well done!"
- Then ... nothing.

When reading the program, we notice that the test is only carried out once, when the program is launched (just after setting the position of the ice). But at that moment, the two sprites are not touching. So the "Well done!" message is not displayed, which is normal.
For the program to work properly, the test "if the ice is touching the rover" has to be carried out constantly, so as to trigger the desired action as soon as the condition is met.

To do that, simply place the test within a **"forever"** loop, which is to be found in the **"Control"** category. The ice program becomes:

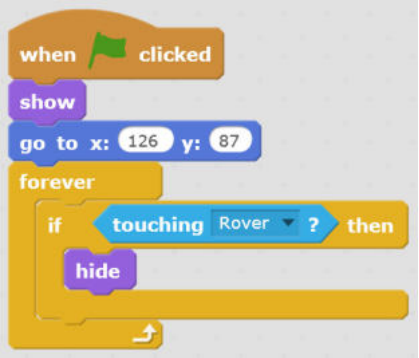


Activity 3: Making the resource disappear when touched (10 minutes)

This very simple task simply requires replacing the instruction "'say Well done!'" with an instruction that makes the sprite disappear. The instruction in question is **"hide"** which is to be found in the **"Looks"** category.

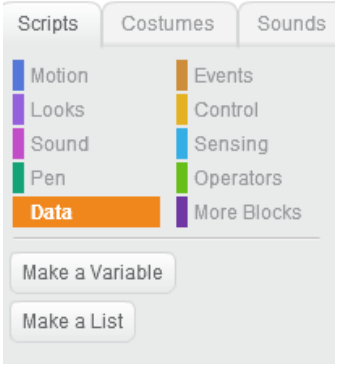
Please note: Once the program is launched, the resource is now always hidden (because we have not told it to come out of hiding!). It is therefore needed to add the instruction **"show"** just after the instruction **"when green flag clicked."**

The ice program becomes:



Activity 4: Creating a <score> variable (5 minutes)

The teacher reminds the students that they need to create a score and increase it each time a resource is touched. The students can explore the various categories of instruction: the one they are looking for is in the orange “Data” category, and called “Make a variable.”



Teaching notes

- The variable created may be accessible either to only one sprite (the one in the program within which it was created) or all sprites. In other programming languages, these are known as local and global variables, respectively.
- For a program to be easy to understand, it is important to give explicit names to the variables you create. This good practice also limits programming bugs. The name of the variable could, therefore, be simply “score.” Some students use names that have no meaning, or which demonstrate confusion between the variable and operations using the variable (for example, they might call the variable “add 1 to score”).
- When the variable is created, it is displayed on the screen, alongside its value. To get rid of this display, simply untick the box to the left of the variable name, in the “data” category.

Here, the aim is to keep a score. This variable no doubt needs to be used by several sprites (the various resources), so it needs to be accessible to all of them.
The teacher should point out to the students that they have already used variables in previous sessions (the X and Y coordinates that give the position of the sprite on the stage). These

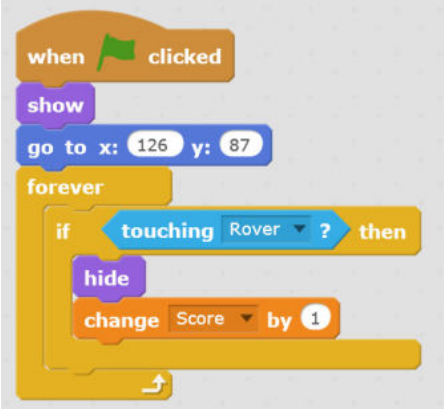
variables were already available and the students used them (carried out tests, set values, etc.) without having to make them.

Activity 5: Increasing the score when a resource is gathered (10 minutes)

The students decide how the score should be increased (for example, it could increase by 1 every time the rover touches a resource).

Then, the students should look for an instruction to increase the score by 1:

They simply need to place it in the ice program, within the test “touching rover?”, just above or below the instruction “hide.”



Teaching notes

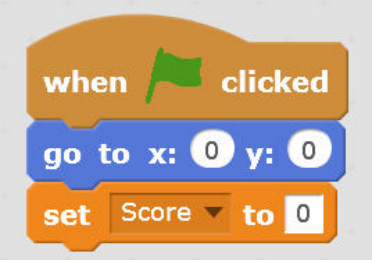
- The variables X and Y, which are pre-existing and relate to the position of the sprite, are available in the “Motion” category, along with the instructions relating to them (setting a value, increasing the value, etc.). The variables created by the user, like the score here, are in the “Data” category. There are two instructions relating to them:
  - “set “score” to (...)”: this command can be used to store the value zero in the “score” variable. The zero can be replaced by any other value.
  - “change “score” by (...)” this command takes the previous value of the variable “score” and adds 1: this new value is now saved in the “score” variable. This is the instruction we need here.
- To get familiar with these instructions, we recommend letting students try them out for a few minutes, with the variables they are using displayed.
- There is an unplugged activity to better understand the use of variables (see page 262).

Activity 6: Resetting the score to zero (10 minutes)

The students test several times what happens when the rover touches the ice. The program seems to work (the ice is present, the rover touches it, the ice disappears and the score is increased by 1). However, if you stop the program and start it again, the score does not return to zero.
To reset the variable to zero, simply add the instruction “set score to 0” at the beginning of the program.

Teaching notes

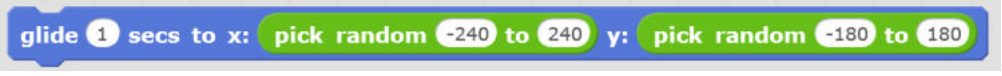
- On the face of it, it should be possible to reset this value in the program of any sprite: the important thing is for it to be done once, and only once. But the score is a variable that will no doubt be used by other sprites (plant, when we add it). There is no reason to choose the ice sprite over a plant one. That is why we recommend resetting the variable in the rover program (which is our “main” program), just below the resetting of its position.
- You can also decide that the main program is that of the backdrop and not that of a sprite. In that case, you can put all the resets in the backdrop, under a “when green flag clicked” instruction.
- When you create a variable, it is a good idea to get into the habit of resetting it straight away.



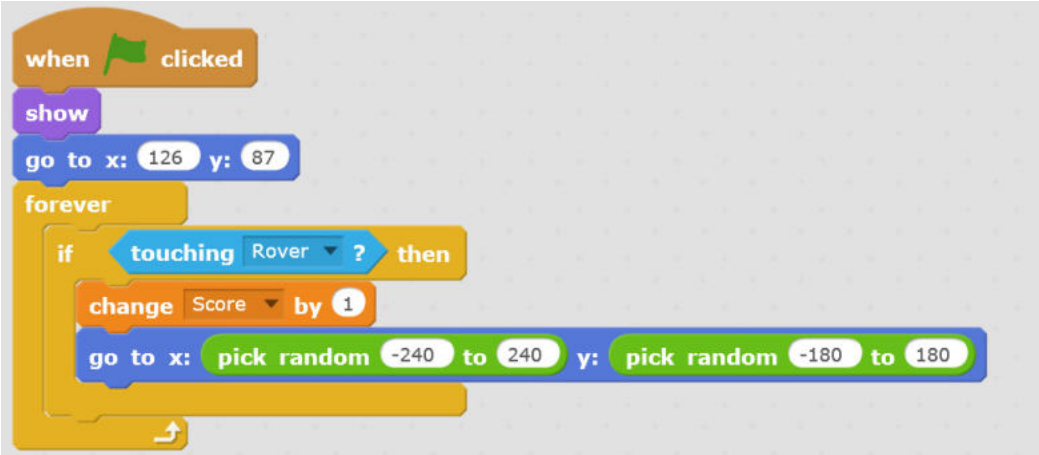
Resetting the position of the rover and the score in the rover program.

Activity 7: Making resources reappear in random positions (15 minutes)

The game can be more fun if the resources reappear after being gathered, but not always in the same place. A random position is best. That means using the instruction “go to,” which the students already know, as well as a new instruction available in the green “Operators” category. This new instruction is called “pick random ... to ....”
As the X axis in *Scratch* varies between -240 and +240, and the Y axis between -180 and +180, the following command is used to position the sprite at a random position on the stage:



There is no longer any need to hide the sprite, as it is simply moved. The ice program becomes:



It is also possible to make the initial position of the resource random rather than fixed (at X=126 and Y=87 in our example).

Teaching notes

- Teacher guidance can be quite free. For example, you could have a student recapitulation the values between which X and Y vary, or find other examples in daily life were random things are needed (dice rolls, card games, national lottery, etc.) .
- Students will then look in the "Operator" category for the instruction that gives a random value between -240 and 240 (for X) and between -180 and 180 (for Y). They should find that without any difficulty.

Activity 8: Importing a new resource (plant) and repeating the same tasks as for the ice (20 minutes)

The students now need to add a second resource (plant) and repeat the work done for the ice:

- Import the "plant" sprite
- Constantly test whether this sprite touches the rover, and if yes:
  - Increment the score by 1
  - Make the sprite reappear elsewhere on the stage (random position)

This activity is very useful for students as it allows them to repeat and consolidate the various concepts already seen.

Conclusion and lesson recapitulation activity

The class summarizes together what they have learned in these various tasks:

- *It is possible to create variables in a program. It is best to reset each variable, meaning to give it a value when the program is launched.*
- *A computer program can generate random numbers. This means that each execution can give a different result.*

The students write down these conclusions in their science notebooks. The teacher updates the poster entitled "Defining computer science."

Step 5 - Plugged and unplugged activities to better understand certain algorithmic concepts

Summary	Alongside their programming activity, students deepen their understanding of certain algorithmic concepts introduced during Step 4: variables, tests, loops, logical operators and even the notion of algorithm.
Key ideas *(see Conceptual scenario, page 204)*	"Algorithms": • A loop allows the same action to be repeated multiple times.. • A test can be used to choose which action to carry out if a condition is true or not. • A condition is an expression that is either true or false. • We can use logical connectors such as AND, OR and NOT to create logical expressions. • Sometimes, we settle for an algorithm that does not offer a perfect solution. "Machines": • A variable is the name we give to a memory area. It is used to store a value and use it later or modify it. "Machines" • For certain tasks, computers are much faster than people.
Inquiry-based methods	Unplugged activities (as well as a plugged activity)

Equipment	For Activity 1 (formative assessment of the loop concept)
	• Computer room
	For Activity 2 (set of cards to consolidate the notion of variable):
	• For each group of 8 students:
	  • 4 whiteboards, 4 markers, 4 cloths
	  • 1 set of cards (cut out) from Handout 34, page 272, and Handout 35, page 273 (for greater durability, use cardstock)
	  • A 6-sided die
	For Activity 3 (game to work on logical operators)
	• For each student:
	  • Handout 36, page 274
	• For each group of 4 students:
	  • Handout 37, page 275
	For Activity 4 (traveling salesman game)
	• (Optional) for each group
	  • An 8 x 8 x 8-inch board (preferably) with around 20 nails driven in (as straight as possible) randomly.
	  • A 2 1/2-yard-long string, tied to one of the nails
	  • A felt-tip pen
	• For each student
	  • Handout 38, page 276

Glossary	Loop, variable, test, condition, logical expression

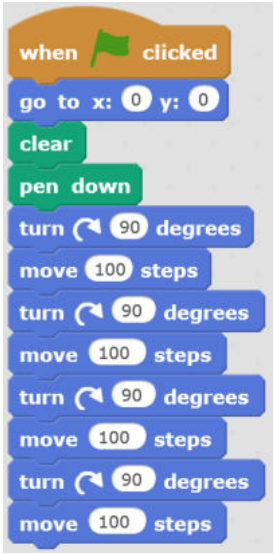
Foreword: When to do these activities

This step is different than the others in that the tasks are not required to program the video game. Rather, they are a series of activities (most are unplugged) designed to review certain algorithmic concepts used during the programming activity.
These activities are completely **independent** of each other and are **optional**: you can continue the project without doing them.
To not lose momentum on the project underway (video game programming in *Scratch*), we suggest doing the unplugged activities suggested here during another lesson slot than that reserved for programming. **Ideally, they should be done during a Language Arts or Math class.**
Activity 1 (plugged) can easily be incorporated into a programming lesson (simply spend ten minutes on it before going back to the project).

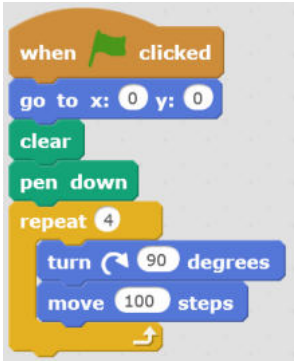
Activity 1: Formative assessment on the loop concept (plugged, 10 to 20 minutes)

It is strongly recommended to do this activity after Step 4, during which students will have dealt with loops several times in *Scratch*. This loop concept can be consolidated through a series of short formative assessment exercises. These will also let students further explore the loops available in *Scratch* (because the ones we use in our program are “infinite” loops, but others do exist!).

All of this is available in Level 2 with *Scratch Junior* (see Lesson 2.3, page 149), you can have students simplify the programming code by using loops. The program below (on the left) has a sprite draw a square. The program on the right gets the same result using a loop. Depending on students’ levels, you can have them create the program on the right by themselves or give them all of the necessary elements unlinked and out of order and have them put everything in order so the program provides the same result as the one on the left.



This sprite draws a 100 pixel square.



This program does exactly the same thing using a loop.

We suggest the teacher to repeat this type of exercise as many times as necessary for students to gain a good grasp of what loops are and how they work.

Activity 2: set of cards to consolidate the notion of variable (unplugged, 1 hour)

This activity tackles the notion of variable through a card game. It is a good idea to let students play this game several times because they begin to develop complex strategies. It can be played during lessons designed to work on mental calculations (scoring), Language Arts (work on card meanings), or tutoring work in small groups of 8 or 16 students. If the activity is done as a class, discuss what all the cards mean before starting. If done with smaller groups, explain the meanings of any specific cards that pose a problem.
The variables dealt with during the game are the players' scores. Certain cards affect these scores. When students are learning to play the game, only cards that make basic changes to scores are included. More complex cards can then be introduced, including those that change scores depending on certain conditions.

Introductory question: Game presentation

Nights are long at the planetary base. Once the explorers have finished their work for the day, they relax by playing indoor sports or board and card games. Their favorite game is a card game the students are going to play. The game is played with four teams (ideally pairs) named A, B, C and D. Set up workspaces with eight students. Each team must try to score as many points as possible as follows:

- Teams A, B, C and D all start the game with 1 point, written on a whiteboard.
- Each team is dealt four cards and the remaining cards are stacked face down (the draw pile).
- The students look at their cards without showing them to the other teams.
- Each team takes a turn and decides to choose one of their four cards. They read it out loud, place it face side up on the table, and follow the instructions. Some of these instructions affect the score of one or more teams. Any changes are noted on the whiteboards.
- When all the cards have been played, the team with the most points wins.

The teacher gives an example on the board: they draw an aerial view of teams A, B, C and D around a table, with their starting scores of 1. Out loud, they read (or have a student read) a card played by team A, asking the class how the scores will change based on the card instruction. The scores are updated with the changes. The teacher then reads (or has a student read) a card played by team B and so on. They continue as long as necessary to make sure the entire class understands.
Alternative: You may decide to set up the game as a joint effort. In this case, the four teams at a workspace work together, without telling each other what cards they have in their hands, to get their final scores as high as possible.

Game with cards 1 to 24

The students play the game with cards 1 to 24 only from Handout 34. The cards that affect the scores explicitly give the name of the teams' scores to change (A, B, C and/or D) and the score changes do not depend on conditions.
During the group discussion, the teacher asks students if their scores stayed the same during the game or if they changed. They introduce the adjective "variable" but not as it relates to computer science. The class discusses the role of the whiteboard (to write down current scores and easily change them).
The teacher can then, if the students have already used *Scratch*:

- Ask them to suggest a name for the scores of the four teams at a workspace, e.g., Score A, Score B, Score C and Score D.
- Show them how to initialize the value of these four variables at 1.
- Introduce the term "variable" as it relates to computer science: a variable is a memory area where you can save a value to reuse or modify later.
- In *Scratch*, begin translating the most basic cards (1 to 8 to start, and more if the class is ready; see paragraph "Translating cards into *Scratch* language" below). The cards are now replaced in the game by their *Scratch* version.

Game with cards 1 to 36

The students play the game again, adding cards 25 to 36 from Handout 35. Some of the new cards designate scores to change based on a player's position (player's own score, score of the person to the right or across from you, etc.). Others introduce conditions (if your score... then...).
The cards are gradually translated into *Scratch* to replace the physical cards with their translation.

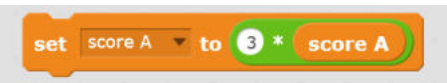
Game with cards 1 to 48

The students play the game again, adding cards 37 to 48 from Handout 35. Two of these new cards deal with random draws. The eight final cards should be filled in by the students.
The translation of the cards continues, without all cards needing to be translated.

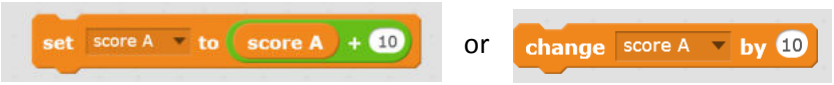
Translating cards into Scratch language

Depending on how work in *Scratch* has progressed, certain cards will have been translated from English into *Scratch* (cards 1 to 8 to start).
For example:

- Card 1 reads 

- Card 5 reads 

- Card 9 has two options:



The translation is divided among different student groups and discussed. Note that certain translations are somewhat complex (especially for cards from No.25) and others are impossible (cards that do not affect the scores or where the player's strategy depends on the context: their score, the scores of other teams, the cards remaining in their hands).
Some cards, such as No.32, require an additional variable to store a value temporarily.

Conclusion and group discussion

The students come together as a group to summarize what they learned from the card game:

- *During several hands, the players scores changed (they are variables). They are memorized at each step of the game. Similarly, the position of the players on the board game, the numbers of pawns they had, etc. were memorized and varied.*
- *In programming language, a variable makes it possible to memorize a value that can vary while a program runs.*

Further study

The students try and find different contexts in which computers used in daily life use variables to store data:

- The time on a digital clock
- The speed of a car displayed on a digital screen
- A bank account balance

They try and find when these variables change or are used. For example, the time on a clock changes every second, and at a certain time, an alarm goes off. A bank account balance changes each time a withdrawal or deposit is made.

Activity 3: A card game to work on logical operators (unplugged, 1 hour)

During the previous step, students used tests (if the rover gathers a resource, the score increases). The teacher explains to the class that the group of words "The rover gathers a resource" is an expression that can be true or false. This type of express is called a "condition".
There are two parts to this activity:

- First, students become familiar with logical expressions by analyzing a scene (Handout 36).
- Next, using vignettes with conditions and logical connectors, they express the condition that must be met for the space station alarm to be triggered (Handout 37).

Logic exercise (individual)

Each student receives Handout 36 and must indicate if the expression for each situation is true or false, or if it is impossible to know.

The answers are:

- Expression 1: TRUE (this is rather clear)
- Expression 2: FALSE (clear as well)
- Expression 3: TRUE (a door is always either open or closed)
- Expression 4: FALSE (a door cannot be both open and closed)
- Expression 5: Impossible to know (nothing tells us the state of the door, unless we can see it at that moment)
- Expression 6a: TRUE (both conditions are met at the same time)
- Expression 6b: TRUE (only one of the conditions needs to be verified, which is the case here)
- Expression 7a: FALSE (the second condition is not met)
- Expression 7b: TRUE (only one of the conditions needs to be verified, which is the case here)

The students' compare their answers to the different expressions on the handout. The teacher makes sure the class agrees for each expression. It is likely that the expressions with an "OR" will be harder for the students to evaluate.

- For expression "A or B" to be true, only one of the two sub-expressions A and B must be true. It is possible, but not necessary, for A and B to both be true.
- For expression "A and B" to be true, both A and B must be true.

If necessary, the class can invent new simple expressions to work with these logical conditions and test if the expressions are true or false using the illustrations on Handout 3 6(or for situations at school). If this is easy for students, present the expressions within a test rather than as isolated examples. For example: "The school bell rings if it's time for recess or if it's time to go back to class or if it's the end of the class."

Manipulating logical expressions: programming the base alarm

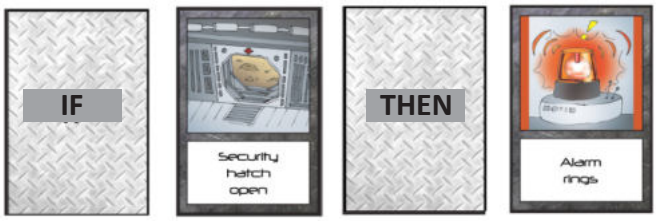
Once students understand this concept, the teacher gives students Handout 37 and divides them into small groups (maximum of four students per group). This handout reviews the previous notions in the context of our scenario (exploring the planet).
The teacher gives the students the following instruction: *Cut out the vignettes, then arrange them into logical expressions. The aim is to describe a condition that will cause the space station alarm to go off.*
For example, the alarm goes off:

- IF "the security door is open"
- OR if "it's night" AND "the rover is not at base"
- OR if "oxygen levels are critical" AND "the base is occupied"

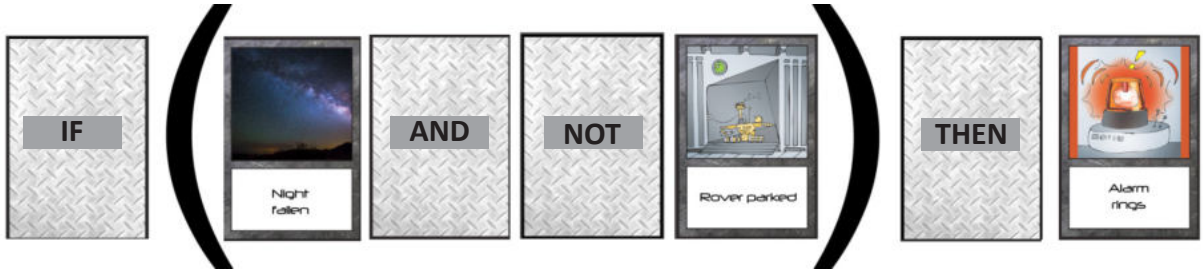
- OR if “energy is low” AND “the base is occupied”
- OR if “the generator is NOT working”
- Etc.

First, the teacher makes sure all the students understand the vignettes, whether with regard to the conditions (cards with drawings) or the logical connectors (IF, THEN, AND, OR, NOT). The NOT is new and should be explained. The fact that the rover is not at base is written: “NOT (the rover is at base).” It may be necessary to have the class think of a few simple examples together, first orally, then using the vignettes for help. Once the students understand the concept, you can let them work on their own to find and write down other conditions.

The first condition is written:



The second condition is written:

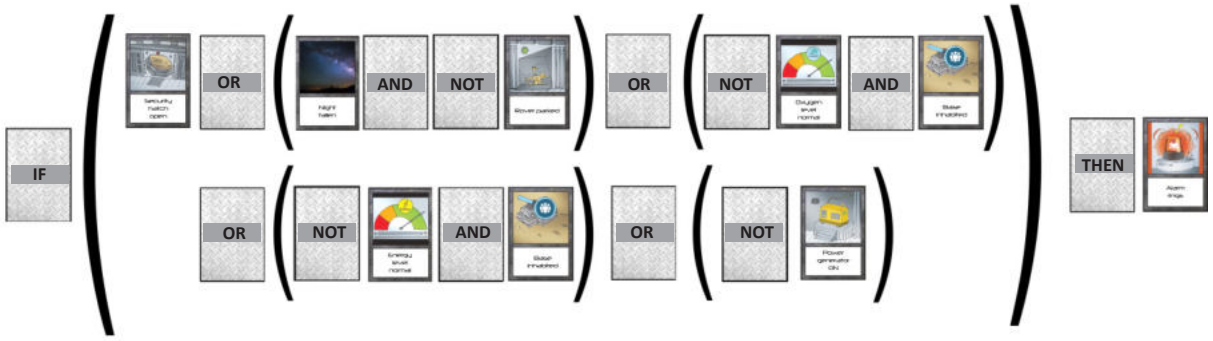


Teaching notes

- The parentheses help make expressions easier to understand and are an integral part of the syntax. Moving parentheses can change what an expression means! Students can place their cards on a blank piece of paper and draw parentheses on the paper.

- Depending on how comfortable students are with the material, the teacher can ask them to write each condition separately or write a single expression that includes all the conditions that will trigger the alarm (all these conditions are connected by “OR”. In this case, it may be needed to print out several copies of Handout 37 so that each group can have more vignettes (especially the logical connectors).

For each of the necessary conditions to trigger the alarm, the teacher makes sure that the students’ various suggestions are presented and discussed.
As a group, the class writes a single expression that includes all conditions. To make the expression easier to read, feel free to add parenthesis and write the expression over several lines:



The class summarizes together what they learned in this lesson:

- *In an algorithm, we can use tests that say which instruction should be done when a condition is verified or not.*
- *A condition is an expression that is either true or false (but not both).*
- *We can use logical connectors such as AND, OR and NOT to create logical expressions.*

Students write down these conclusions in their science notebook. The teacher updates the *"Defining computer science"* poster by copying down what the class learned about the notion of logic during this lesson.

Activity 4: Understanding that an algorithm is not always perfect: the traveling salesman game (unplugged, 1 hour)

Teaching notes

- This activity (optional) is aimed at older students (6th grade and up). It goes into greater detail about the notion of algorithm through a simple problem: finding the shortest route that goes by several locations. The aim is to show that sometimes a problem cannot be resolved, even if there is an algorithm that is supposed to work. You have to settle for an approximate solution, which may be imperfect but good enough.
- Here, there is a simple algorithm to resolve the problem (try all routes and choose the shortest) but this algorithm cannot actually be implemented because the number of possible routes to try quickly becomes massive.

Introductory question

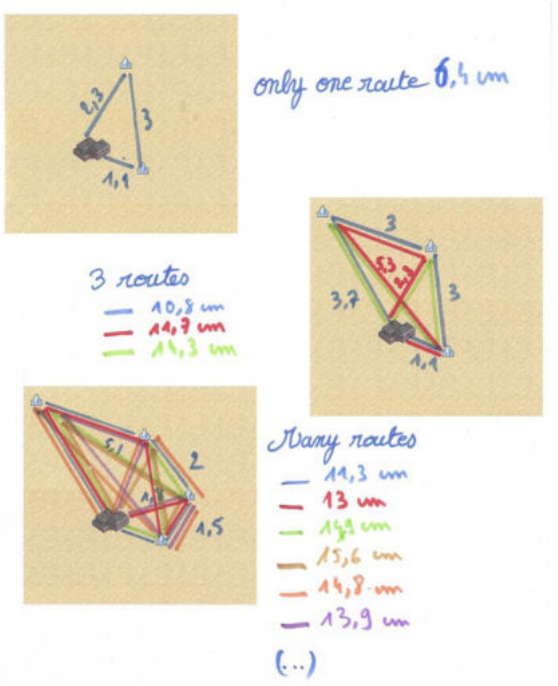
In the previous lessons, students set up their program to be able to add another challenge: the rover must explore the map to recover all possible resources.
The teacher explains that the rover has a limited amount of fuel left and so they need to use as little fuel as possible. This is a "classic" optimization problem: *"You know from the outset the number and location of the resources. You must find the shortest route that still goes by all of these resources and returns to the starting point."*
The teacher asks the students if there is a method (algorithm) that can offer a solution to this problem. The group discussion shows that this question can be broken down into two more specific questions:

- *Does the problem have a solution?* Of the various routes that go by all the resources, is there one that is shorter than the others (or more than one that are equally short)? The answer to this question is yes: there are several routes possible, so there will obviously be one (or more) that is shorter than the others.
- *Is there a method that will allow us to definitely find the shortest route?* The class can suggest a method for finding this route: they simply need to test all the possible routes, measure them and select the shortest one.

Depending on the available materials, the teacher can do a single activity (based on the handout) or two activities (first the handout, then the experiment using the boards with nails).

Finding the shortest route (in groups)

The teacher hands out a copy of Handout 38 to each student (because there are so many routes to test and draw, it is best to give all students a handout). The aim is to find all the possible routes to find two (or three, or four) resources and measure the length of the routes using a ruler. Note: It is assumed that the route only goes by the base twice: at departure and arrival.



Group discussion

The group discussion shows that the number of possible routes quickly increases as the number of resources to gather rises.

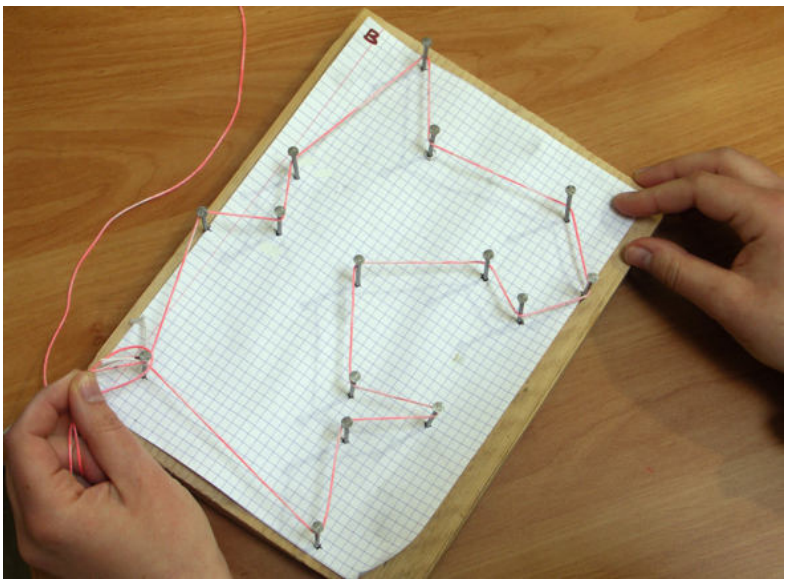
- For two resources: If B is the base and 1 and 2 are resources, there are two possible routes: B12B or B21B. In fact, this is the same route followed in opposite directions.
- For three resources, there are six possible routes, with three that are actually different:
  - B123B and B321B (the same route in both directions)
  - B132B and B231B (the same route in both directions)

  - B213B and B123B (the same route in both directions)
- For four resources, there are 24 possible routes, with 12 that are actually different:
  - B1234B and B4321B (the same route in both directions)
  - B1243B and B3241B
  - B1324B and B4231B
  - B1342B and B2431B
  - B1423B and B3241B
  - B1432B and B2341B
  - B2134B and B4321B
  - B2143B and B3412B
  - B2314B and B4132B
  - B2413B and B3142B
  - B3124B and B4214B
  - B3214B and B4123B

When all duplicates are eliminated, there are "only" 12 possibilities left, which is too many for students to find all of them and draw on a single map.

(Optional) Finding the shortest route (in groups)

If there are enough boards with nails for each group, the teacher hands out the boards and asks the students to find the shortest route possible that has the string go around each nail (and return to the starting point). For each route tested, place a mark on the string to indicate the total length (which should be a short as possible). This activity is similar to the previous one except you need physical objects (nails, string) to let students easily eliminate certain "wrong" routes and intuit what might be the "right" route.



Group discussion

This group discussion highlights several things:

- There are a large number of possible routes (no one will have been able to test them all).
- At the start of the activity, progress is significant (students find shorter and shorter

routes) but then tapers off (despite multiple tests, they cut routes by only a tiny distance).

- Some students may believe they found THE best route. It is not possible to confirm this statement, but you can tell them that the "right" routes are those that do not cross over each other and limit backtracking.

Scientific notes:

- The number of different routes, for *n* resources + 1 base (or n+1 nail, for the second experiment) is *n*! (which is read "factorial *n*"). This number is obtained by multiplying *n* by all lesser whole numbers down to 1. For example:
  - 1! = 1
  - 2! = 2x1 = 2
  - 3! = 3x2x1 = 6
  - 4! = 4x3x2x1 = 24
- If you eliminate all routes that are identical but simply taken in opposite directions, for *n* resources, this results in *n*!/2 number of possible routes.
- This function (which, for *n* number of resources linked to *n*!/2 routes) increases very quickly:
  - 5 resources: 60 routes
  - 6 resources: 360 routes
  - 7 resources: 2,520 routes
  - 10 resources: nearly 2 million routes
  - 20 resources: 1 billion billion possible routes
  - 100 resources: the number of possible routes is 157 figures long!
- This is a "classic" algorithmic problem, known as the "traveling salesman problem." A salesman must visit several customers in an area. What is the shortest route to visit each one once?

The teacher explains to the class that the number of routes increases exponentially (they can explain that for 20 resources, there are more than a billion billion possibilities). They ask the students if they know a device that can find the best route among several options. Some students will likely say a GPS. By replacing resources with cities, the problem the GPS must solve is similar (although it differs slightly – the traveling salesman must go to all the cities, whereas a GPS must find the shortest route from one place to another, using passing through several intermediate points). What do you do when there are thousands and thousands of cities in a country and not just 20?

The class agrees that it is impossible to test all possible routes (even a supercomputer could not do this test in our lifetime!). A GPS computer proceeds in a way similar to the nail/board experiment: it determines a given route and alters it several times to reduce the route's total distance. When it is unable to reduce the distance by a significant amount, it stops looking and suggests the best route from those it explored. To increase its chances, it may choose several routes randomly and alter each of them separately.

The GPS does not necessarily suggest the "best" route possible but a "pretty good" route. It is not surprising that two different GPS devices will suggest two different routes for a single destination because they feature different maps (one may have more details or be more updated than the other) and different algorithms.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *Sometimes a problem is so long or complicated that we have to settle for an algorithm that provides an imperfect solution.*
- *For certain tasks, such as finding the best route from numerous possibilities, computers are must faster than people.*

Students write down these conclusions in their science notebook. The teacher updates the *"Defining computer science?"* poster.

Further study

- Showing students *The Imitation Game*, which tells the story of Alan Turing's life, can illustrate in another context (here, breaking the Nazis' code during the Second World War) that is sometimes impossible to test all possibilities of a problem and that a method must be found to restrict the problem's size by focusing on "interesting" possibilities. Even in the movie, the possibilities well exceeded the abilities of one or several people. To resolve the problem, Turing and his team had to find not only an effective algorithm, but also create a machine to run it: this machine was the precursor of today's computers.
- We can compare the itineraries provided by several webmapping services (Google Maps, Mappy, ViaMichelin, etc.) or several GPS providers and verify that these tools do not suggest THE best route (if they did, they would all suggest the same route), but a "pretty good" route.

HANDOUT 34
Playing cards (1/2)

Card #1 Score A is now 3.	Card #2 Score B is now 4.	Card #3 Score C is now 5.	Card #4 Score D is now 6.
Card #5 Score A is multiplied by 3.	Card #6 Score B is multiplied by 3.	Card #7 Score C is multiplied by 2.	Card #8 Score D is multiplied by 2.
Card #9 Score A increases by 10.	Card #10 Score B increases by 8.	Card #11 Score C increases by 6.	Card #12 Score D increases by 4.
Card #13 Add together scores A and B. The result is the new value of score A.	Card #14 Add together scores B and C. The result is the new value of score B.	Card #15 Add together scores C and D. The result is the new value of score C.	Card #16 Add together scores D and A. The result is the new value of score D.
Card #17 Update scores A and C by exchanging their values.	Card #18 Update scores B and D by exchanging their values.	Card #19 Update scores A and B by exchanging their values.	Card #20 Update scores C and D by exchanging their values.
Card #21 Your score is now 10.	Card #22 Your score is now 10.	Card #23 The score of the team in front of you is now 0.	Card #24 The score of the team in front of you is now 0.

HANDOUT 35
Playing cards (1/2)

Card #25 Divide all even scores by 2. Odd scores are unchanged.	Card #26 All scores multiple of 3 are divided by three. The other scores are unchanged.	Card #27 If your score is 0, then copy the value of the best score.	Card #28 If you have the best score, then change its value to 0. Otherwise, keep your score unchanged.
Card #29 The new value of your score is the sum of your two closest neighbours' scores.	Card #30 Exchange the value of your score with your right-hand neighbour's.	Card #31 If your score is even, then divide it by 2, otherwise add 1 to it.	Card #32 Switch clockwise all the scores (all teams get the values of their right-hand neighbours' scores).
Card #33 Choose the new value of your score amongst all the available scores, including yours.	Card #34 If your score is inferior or equal to 10, multipliy it by itself to find its new value.	Card #35 All the scores superior or equal to 5 lose 5 points.	Card #36 All the scores inferior or equal to 5 increase by 5 points.
Card #37 Keep all the scores unchanged.	Card #38 Steal a total of 10 points from one or more opposing teams, unless the sum of their scores is less than 10.	Card #39 Roll a 6-sided die. The result of the die is the new value of your score.	Card #40 Roll a 6-sided die. The result of the die is the new value of the score of the team in front of you.
Card #41 Your score …	Card #42 Your score …	Card #43 Your score …	Card #44 Your score …
Card #45 All the scores …	Card #46 All the scores …	Card #47 All the scores …	Card #48 All the scores …

HANDOUT 36
Logical expressions

Instruction: For each of the following expressions, say whether it is TRUE, FALSE or if it is impossible to know.

Expression 1: All cats are animals

Expression 2: All animals are cats

Expression 3: Right now, the door to the gymnasium is open or closed

Expression 4: Right now, the door to the gymnasium is open and closed

Expression 5: Right now, the door to the gymnasium is open

Expression 6:

Expression 6a: The dog is on the grass and the cat is in the tree

Expression 6b: The dog is on the grass or the cat is in the tree

---

Expression 7:

Expression 7a: The dog is on the grass and the cat is in the tree

Expression 7b: The dog is on the grass or the cat is in the tree

HANDOUT 37
Securing the base

Instruction: Cut out the vignettes and arrange them to create a condition that will trigger the alarm.

HANDOUT 38
Finding the shortest route

Instruction: The rover must leave the base and gather all resources (here, ice crystals for the water supply) before returning to base. For each of these examples, answer the following two questions:

- How many possible routes are there (assuming the rover drives in a straight line between two points)?
- Which is the shortest route?

Step 6 - Avoiding obstacles and managing player lives

Summary	Students now add obstacles to avoid (new sprites) and create a variable for the number of «lives». They are again exposed to the ideas of tests, lops and variables covered previously and deepen their understanding of what an event is.
Key ideas *(see Conceptual scenario, page 204)*	Same as previous sessions
Equipment	For each student pair • A computer with *Scratch* and the program saved from the previous session

Teaching notes

- It can be useful to occasionally show the "final" game (teacher's version) to keep students motivated and review the next steps. The point is not to give them the solution by having them read the program, but simply do a demonstration of the game.

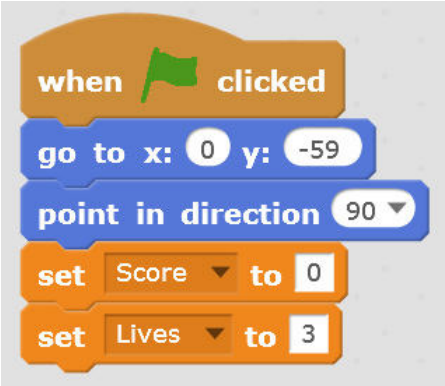
To make the space mission game more interesting, the teacher explains that the students will need to introduce obstacles (a lake of lava, a sand dune) and a new variable: the number of "lives." The rover starts the game with three lives and loses one life on its way back to base each time it touches an obstacle. The game stops when the number of lives reaches 0.

Activity 1: Adding new sprites (5 minutes)

Based on what they learned in previous lessons, the students import the three new sprites (the base, the dune and the lava) and initialize them by setting their position in a fixed location on the screen. They can decide, for example, to place the base in the center, which means they will need to move the rover's initial position (since it is already in the center).
This is not essential, but you may want to discuss the "depth" of the different sprites (rover, base, lava, dune, plants, ice). Deciding which sprite will be in the foreground is done using the instruction "Go to front" available in the "Looks" instruction category.

Activity 2: Creating and initializing a <number of lives> variable (5 minutes)

The students can easily create a new "number of lives" variable with an initial value of three (in the same place as where they initialize the score, for example, in the rover program). The rover program now contains (in addition to the scripts to command it):

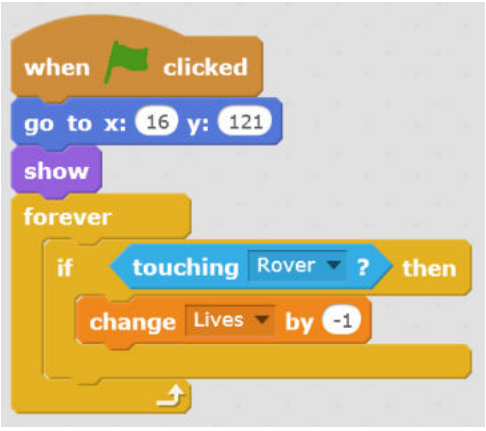


Activity 3: Losing a life when the rover touches the lava (30 minutes)

During the previous lesson, touching a resource increased the value of the “score” variable by one. Here, touching the lava or the dune must reduce the value of the “life” variable by one. We suggest completing the task for one of the obstacles (lava), then starting again for the second one (dune).

Subtracting in Scratch

The students try to figure out how to control/program this reduction. There is no “subtract” command in *Scratch*, only “add.” If necessary, discuss as a class that they must insert the value -1 to subtract one.
The program for the lava is:



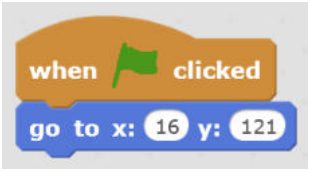
Provisional program for the lava

However, this program has one problem: when the rover touches an obstacle, it touches it for a long time (a few tenths of a second, or several seconds if the user does not move). During this time, the “life” variable continues to decrease. After a few seconds, the score will end up being extremely negative (e.g., -4000). The only way to avoid this is to immediately break contact between the rover and the obstacle.
Because the obstacle is in a fixed position, the rover must move. However, this cannot be done in the program for the lava or dune: it must be in the rover program to command the rover’s position.

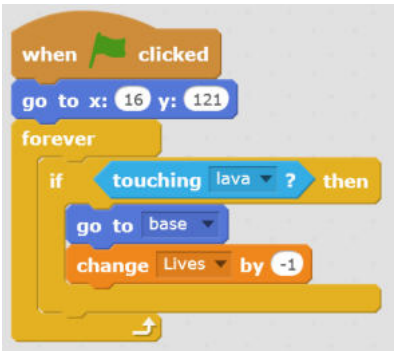
Moving the rover

There are (at least) two solutions to this problem:

- **Solution 1**
  Delete the program created for the lava (except its position initializer) and make a similar one in the rover program. In the rover program, add a command telling it (for example) to go back to base.
  To return to base, the rover can either be told to go to (X=0, Y=0) or to go to the position of the «base» sprite. This final solution is better because it will work even if you decide to place the base somewhere else.

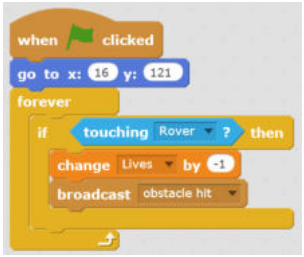


Lava program

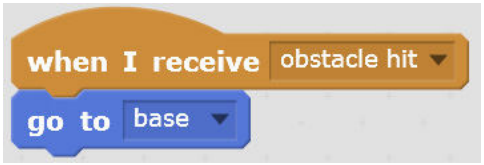


Rover program

- **Solution 2** (more practical for later activities)
  A more elaborate solution consists in keeping the program that was created for the lava and adding a new instruction that will send a message to the other programs (especially the rover program). This message, in the rover program, will trigger an action (return to base).
  Like the names of the variables, the message titles should be explicit. Here, for example, our message is «obstacle hit.»



Lava program



Rover program

The lava program sends a message to the other programs (especially the rover program).
In the rover program, receipt of the message is an event that triggers an action (going to base).

The commands that can be used to send a message or trigger an action when a message is received are found in the “Events” category.

Teaching notes

- A third solution is also possible: programming the rover to bounce off if it hits an obstacle (this way, it does not remain in contact).
- Solution 2 is the best solution if the aim is to emphasize the "event" aspect of this program. Each time an event happens (for example, hitting an obstacle), the program can send a message that will be used by other programs. The idea of event was already introduced in previous lessons ("when the top arrow is touched," "when the green flag is touched," etc.), but this is the first time that students will use this new type of event: receiving a message sent by a program.
- We allow the teacher to decide which method to use based on their desired focus for the lesson: reviewing previous concepts or introducing the new idea of message. Of course, the teacher will want to make this choice based on the ideas students come up with and their understanding of the concepts already covered.
- Please note: sending and receiving a message will be used again at a later time to manage the end of the game (see "Game over," following pages).

Activity 4: Repeat Activity 3 for the dune (10 minutes)

Now that the students have learned to manage one of the obstacles (lava), they must do the same for the other one (dune).
This short exercise reinforces their understanding of what was done previously, namely sending and receiving messages.

Conclusion and lesson recapitulation activity

The students update the list of *Scratch* instructions they know.

Step 7 - Ending the game: "Game over"

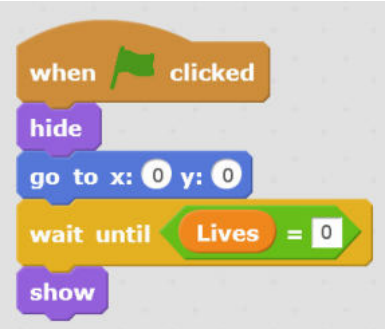
Summary	Students complete their program by introducing a test on the number of remaining lives: the message «game over» appears and the program stops when no more lives remain.
Key ideas *(see Conceptual scenario, page 204)*	Same as previous sessions
Equipment	For each student pair • A computer with *Scratch* and the program saved from the previous lesson

The students must now make the game stop when there are no more lives. This requires several things:

- Make "game over" appear
- Make it so the game can no longer be played (unless it is relaunched)

Activity 1: Make "game over" appear when there are no more lives (15 minutes)

There are several ways to make "game over" appear. The easiest way is to import a sprite that takes the appearance of "game over" (sprite provided). This sprite appears when the number of lives reaches zero.
This is very easy to do using the instruction "Show" in the "Looks" category. Note that you must remember to hide the "game over" sprite when the program is launched!
The program for this "game over" sprite is:



"Game over" sprite program

Teaching notes

- It is also possible to import a "text" sprite from the *Scratch* library. This sprite is called "awesome!". By clicking on the "Costumes" tab, you can change the color and font as well as edit the text.

Activity 2: Stopping the game when "game over" appears (15 minutes)

As the program stands now, users can continue playing even when "game over" appears, which is not the desired outcome. To end the game, there are several strategies:

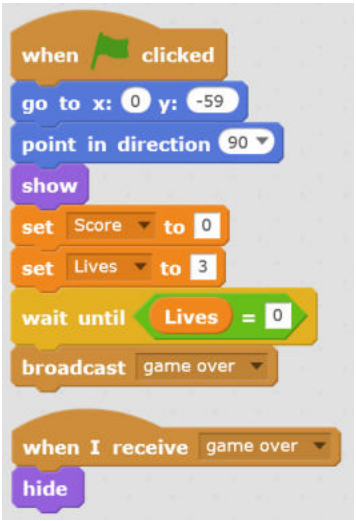
- **Method 1**
  Trigger the end of all programs when the number of lives reaches zero. This is done using the "Stop all" command from the "Control" category.

Rover program

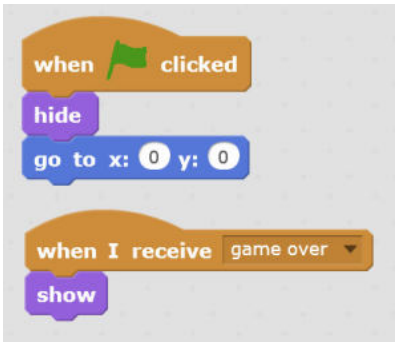
In theory, this should stop the game. However, in practice, a bug in *Scratch* (not yet fixed at time of printing) means that despite using "Stop all," certain programs will continue to run (it is still possible to move the rover). As a result (and because it is visually more appealing), we prefer the second method described below.

- **Method 2**
  Make all the other sprites disappear when the number of lives reaches zero. With this solution, only the "game over" sprite remains on the screen, so the game is truly over. This method involves having one of the programs (e.g., the rover) send a message. This message is called simply "game over."

  All the sprites (except the "game over" sprite) are hidden when they receive this message. Because they are told to hide at the end, you must remember to ask them to appear again when the program is launched.



Rover program

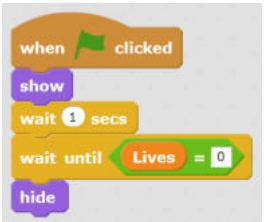


«Game over» sprite program

The final instruction of the rover program (hide when «game over» is received) is also in the other sprites' programs (except the «game over» sprite, which appears at that moment).

Teaching notes

- A simplified variation of Method 2 is available if you do not want to send or receive messages. This variation consists in asking each sprite to hide when the number of lives reaches zero.



  The one-second pause ensures that the "lives" variable in the rover program had the time to be initialized at 3. Otherwise, since its value is 0 when the program launches, the sprites will hide immediately.
- Introducing a pause, even for a fraction of a second, is a workaround for the small bugs linked to the syncing of different programs. *Scratch* gives the illusion of running all programs simultaneously, but in fact, it runs them one after another (very quickly, which is why it appears to be simultaneous). Introducing a pause to send/receive a message is a way to force a program to run before another.

Conclusion and lesson recapitulation activity

The students update the list of *Scratch* instructions they know.

Step 8 - Adding challenges

Summary	Students finalize their video game by adding additional challenges: a time limit, a tornado that goes faster and faster and moves around randomly, etc. The concepts seen during the previous lessons – tests, loops, variables and events – are all reviewed.
Key ideas *(see Conceptual scenario, page 204)*	Same as previous sessions, plus: "Algorithms" • Certain loops, called «conditional loops,» are repeated until a condition is met.
Equipment	Same as previous sessions

Starting the activity

The students will likely have noticed that their game can be played but that it is not very interesting because there are no major difficulties. If you pay attention to the obstacles, the game can go on forever.

The class thinks together about a way to end the game and make it harder. There are several options:

- **Option 1:** Add a time limit. When the time is up, the game stops. The player's score is based on the number of lives they still have and the number of resources gathered.
- **Option 2:** Add a new element to make the game increasingly difficult and eventually end the game. For example, you can introduce a new trap (a tornado, similar to the weather problems in Lesson 3, page 219 and Lesson 4, page 225, on binary coding). The tornado moves faster and faster and its direction is unpredictable. Eventually, it becomes too fast for the player to avoid and the game will end.

Teaching notes

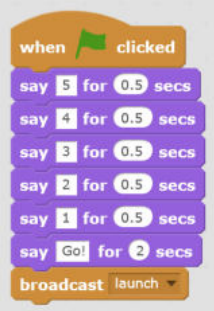
- It is quite possible that the students will think of other ways to end the game. We suggest the teacher to encourage a class discussion to choose one or more options for the students to use (groups can choose the same option if they like).
- The teacher makes sure that students have an idea before getting started. If one option is appealing but unrealistic, it's best to choose another that is more feasible.

Below, we describe the two options above, as well as several others that do not necessarily make the game more difficult but do make the game more visually appealing or avoid certain bugs. The tasks below are of varying difficulty, and can be used independently of each other.

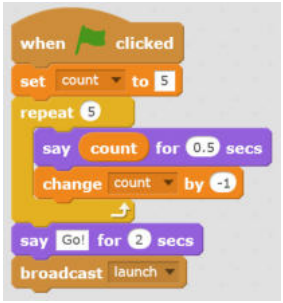
We suggest doing at least Activity 2 or 3 to make the game more interesting.

Activity 1: Make a countdown appear when the game starts (15 minutes)

You can decide that the game only starts after a countdown: *5, 4, 3, 2, 1, go!* This can be done very easily, like this:



Or in a more sophisticated way by using a new variable and a loop:



Depending on students' levels, you can have them use the "easy" method or ask them to use a variable and a loop. For the more complicated version, you can also give them all of the elements but out of order and not linked and ask them to put everything in order for the program to get the same result as the first option.

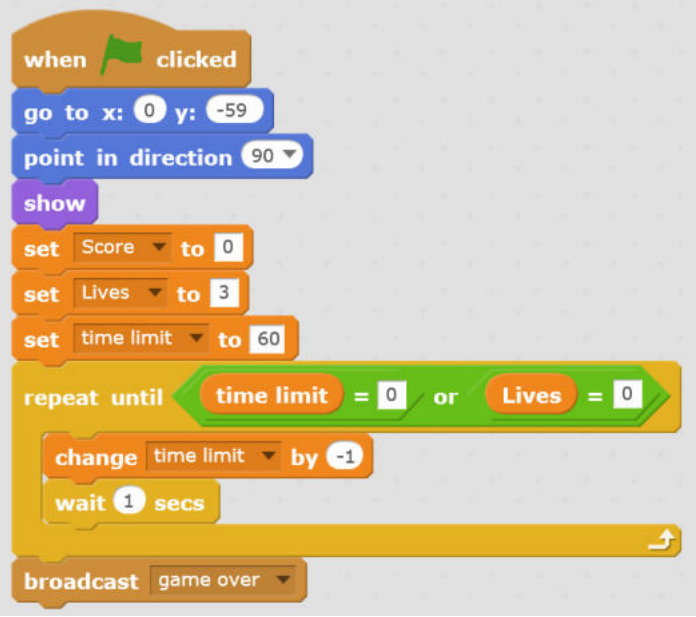
Activity 2: Limiting the game duration (15 minutes)

Teaching notes

- This task is similar to the first, described above, but you must use the more complicated variation (variable and loop) because it is not feasible to write out a countdown by hand, second by second, that will last several minutes.
- If the students have already created a countdown (Activity 1), this task will be easy for them. If not, they will need a little more time and may need some guidance.
- This task reviews the ideas of variables, tests, loops and logical operators.

Limiting the game time is an easy way to make it more interesting. A "time limit" variable must be created, given an initial value and decreased as time goes on by introducing a timeout period to control the speed of the process.
The program stops ("game over" message) when the time limit reaches 0 or when the number of lives reaches 0.

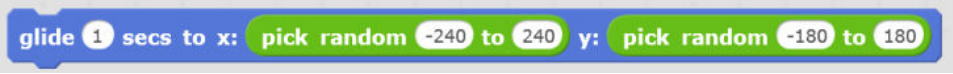
The rover program must be modified to include:



In this example, set the "countdown" to 20 and remove 1 every second. These values can be changed for a shorter or longer time limit.

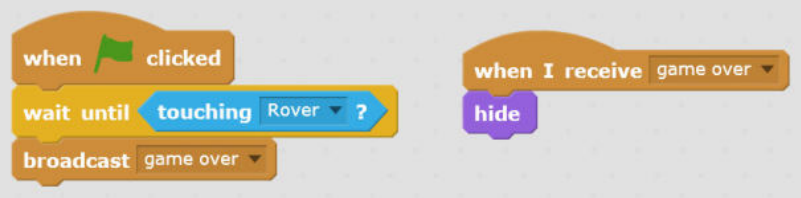
Activity 3: Add a tornado that moves around randomly (15 minutes)

Students already know how to create a sprite from an image and initialize its position. They can, for example, have a tornado start from the lower left corner (X = -230 and Y = -170). They must then figure out how to make it move around randomly. For the movement to be instantaneous, change the X and Y values. However, it is better to see the tornado move around. The command to do this is "Glide...secs to x=… and y=…" from the "Motion" block category. For the tornado to move to any position on the map, simply write the following instruction:



By clicking several times on this instruction, the students can confirm that the tornado moves randomly each time. The movement is always one second, so if the point of arrival is close, it moves very slowly, and if it is farther away, it moves faster.
Now, this instruction needs to be connected to the rest of the tornado program (we have changed the time to "2 seconds" to slow the tornado down).

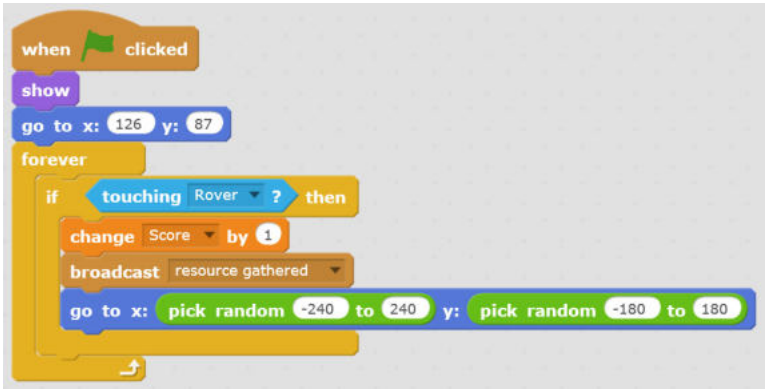
You must remember that if the rover touches the tornado, the game is over. The following instruction is added to the tornado program:



Activity 4: Make the tornado bigger (15 minutes)

To make the game even harder, you can make the tornado get bigger as resources are gathered. At this point in the project, this task should not be too difficult.

- In the resource programs (ice and plants), send a message each time a resource is gathered.



- In the tornado program, make it bigger each time the message “resource gathered” is received. To do this, use the command “Change size by...” from the “Looks” category.

- Because the tornado’s size is modified in the program, the size must be initialized at the start of the program using the command “Resize to 100%” from the “Looks” category.

Activity 5: Making the tornado go faster and faster (20 minutes)

This task, which is very difficult, is intended for older students (middle school). However, some older elementary school students may need a challenge. This should do the trick!

The aim is to make the tornado go faster each time a resource is gathered (ice or plant). To make the tornado go faster, a variable – "speed_tornado" – must be created (initialized at 1, in the same program as for the other variables) and to increase it, such as by 10% at each event (hint: increasing the speed by 10% equates to multiplying by 1.1).

The tornado program is modified as follows:

Note that the value of "speed_tornado" must be taken into account in the program that commands its movement ("Glide" instruction). This can be done by including "speed_tornado" in the glide time calculation as follows:

The greater the "speed_tornado", the faster the movement is: the tornado gets faster and faster, which is the result we want.

Activity 6: Simulate a torus world (joining the edges of the backdrop) (20 minutes)

The game will be more fun if the rover is not blocked by the edges of the backdrop. When it reaches the right edge, it needs to continue along its path and reappear on the left side, and vice versa. It should do the same for the top and bottom edges.

Scientific notes

- A flat surface that is folded so that the left and right edges meet is called a cylinder. If a second fold is added to join the top and the bottom edges, this is called a torus cylinder. It resembles a doughnut.
- Many video games are based on a torus world, even if it does not actually resemble a real planet (on Earth, when you reach the North Pole, you do not switch to the South Pole!).

The teacher lets the students work on their own and guides them if they encounter difficulties, first explaining the algorithm to use: if position X of the rover goes past 240 (far right edge), then it must go to -240 (far left edge). Next, the teacher can show the different blocks required to build the program: a "repeat forever" loop, a, "if…then" control structure, a "higher than"

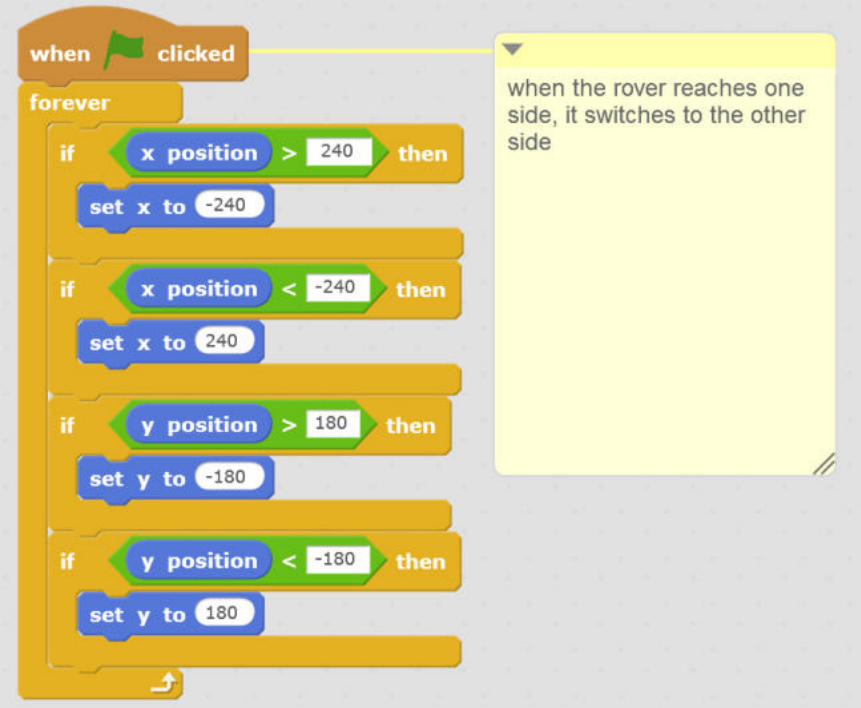
operator, the value for variable X (blue “Position X” block), and the instruction to be able to change this value “blue “Set X to …”). The blocks connect together as follows:

Teaching notes

- Depending on the shape of the sprites, it may be necessary to leave a small margin (instead of setting the edge value at 240, set it at 235).

Once the students have understood how to program the transition from the right to left edges, it is easy to program the transition from left to right and then vertically (Y variable) top to bottom and bottom to top.

The rover script that creates a torus world is:



Teaching notes

- The screenshot above shows that it is possible to add comments to a program. It is good to get into the habit of doing this to make the program clearer for both you and those who will be reading it.
- At this point, the instruction “bounce back if the edge is touched” needs to be deleted in the scripts that pilot the rover using arrows.

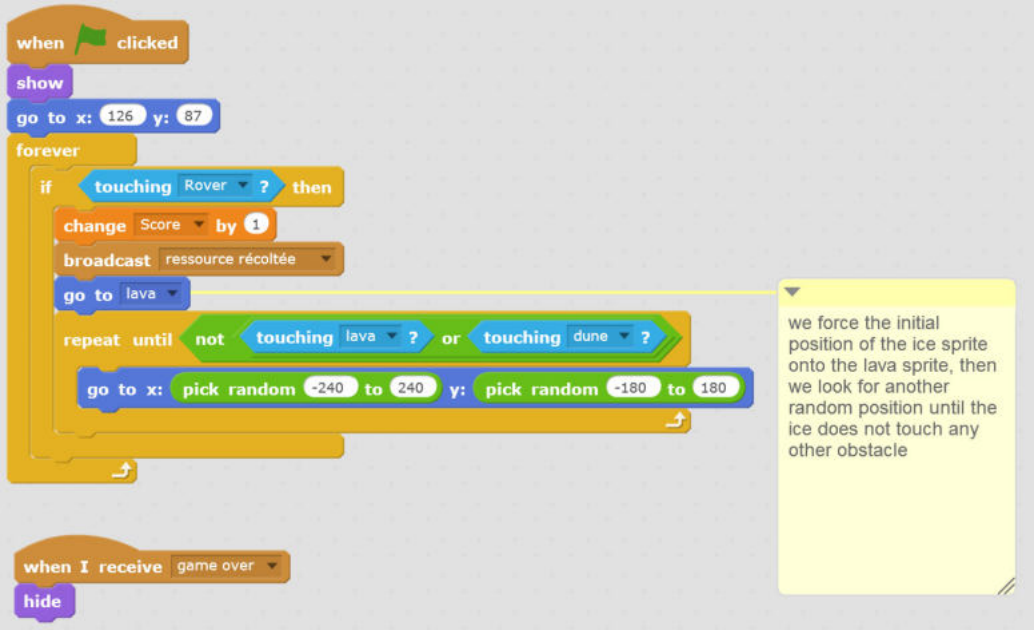
Activity 7: Preventing resources and traps from overlapping (20 minutes)

Like for Activity 5 above, this task is aimed at middle school students or older elementary school students who are ahead of the rest of the class.
When resources (ice, plants) are gathered, they reappear randomly on the stage. It is quite possible that a resource will appear where there is already a trap (dune or lava). This situation must be avoided, as you cannot have two contradictory goals: gathering the resource and avoiding the obstacle.
What do you do to ensure this does not happen? A new position must be chosen randomly as long as the resource is touching a trap. However, because the resource was not initially in contact with a trap, the loop will not run. The trick is to introduce a preliminary step to force the loop to run a first time. The algorithm is:

1. Place the resource anywhere by randomly selecting its coordinates (or, if you prefer, place it on a trap).
2. Then, create the following loop: Until the resource is in position free of any traps, it assigns a new random position.

The students will need to be guided, either by telling them the trick above (after letting them explore on their own) or by giving them the final program and asking them to analyze it to understand what it does and why.

The final program for the ice or plants is:



Conclusion and lesson recapitulation activity

The video game is now interesting enough for students to play it and give each other challenges. The students update the list of *Scratch* instructions they know.
The teacher reviews with the class what they learned during this project, the difficulties they encountered, what they enjoyed (some will likely have already started creating other *Scratch* programs at home), etc.

Teaching notes

- This programming activity was probably new to most students. To make sure they do not forget how to use *Scratch* and to encourage their creativity, we suggest having them create a personal project later in the year. This can be a video game (a fairly complex project, as we saw), an animated card (for Halloween, Christmas, etc.) or interactive surveys. There are many possibilities.
- The aim of the next step is to explore some of the additional functionalities in *Scratch* to give you an idea of other possible activities.

Step 9 - Further study in Scratch

Summary	Here are several ideas to explore other functionalities in *Scratch*, such as giving students extra options for their personal projects.
Key ideas *(see Conceptual scenario, page 204)*	Same as previous sessions
Equipment	For each student pair • A computer with *Scratch* and the program saved from the previous lesson
Duration	1 to 4 lessons

Foreword

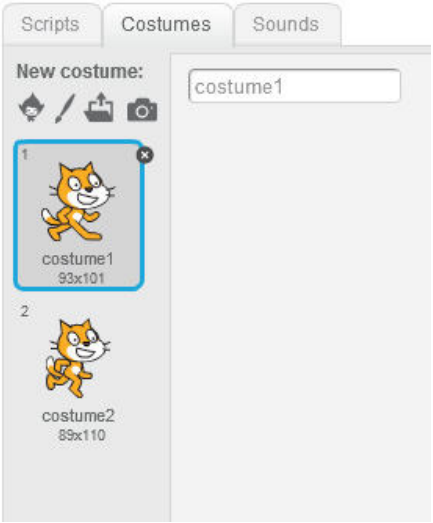
During the previous lessons, students learned about the basic functionalities and concepts of *Scratch*:

- Sprites
- Stages
- Instruction blocks
- Motion
- Looks
- Events
- Loop
- Tests
- Variables
- Calculations and logical operators
- Sensors

However, there are many functionalities that were not covered. Many are very simple and can enhance students' programs they do on their own: games, animated cards, interactive surveys, drawings, etc. We have provided a list below.

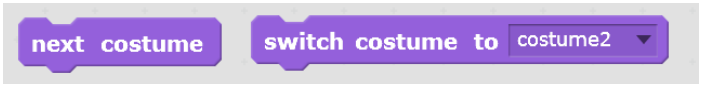
Costumes

Each sprite can have several "costumes" that correspond to its physical appearance. A new costume can be a slight variation of the previous costume (a different arm or leg position, etc.) or very different than other costumes.
To create a new costume, select the sprite you want to change and click the "Costumes" category. This new costume can be based on the old costume, an imported image or a drawing created by the user.

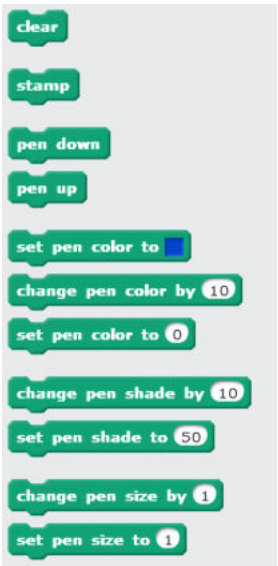


Here, the cat (default sprite in *Scratch*) has two costumes. You can change the costume while making it move forward, which makes it look like it is walking.

The two instructions to change costumes are available in the "Looks" category:



For the previous sequence, for example, you can ask the students to create a new costume for the rover to make it look "burned" or "broken" to be activated when the rover hits an obstacle.



Pen

Each sprite has a "pen" that leaves a line on the ground (the screen) when lowered. The pen's default position is raised and does not leave a line.

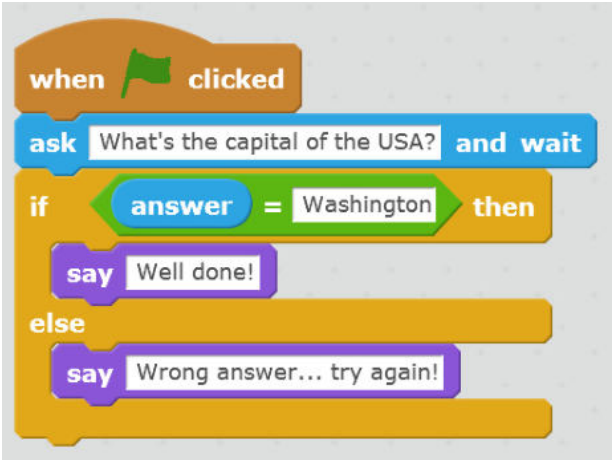
The instructions available from the green "Pen" category let you raise or lower the pen, change the thickness of the shade, thickness, color, etc. of the line.

This functionality means you can create such things as geometric shapes in *Scratch*.

Speech and questions/answers

Each sprite can display text on the screen ("Say" instruction in the "Looks" category). It can also interact with the user by asking questions ("Ask" instruction in the "Sensing" category). It waits for the user's response typed with the keyboard and saves it in a predefined variable called "answer." The answer, like any variable, can be manipulated by the program.
This small program, for example, greets the user by using their first name:

This one asks a question and congratulates the user if the answer is correct.



It is possible (and quite interesting) to use *Scratch* to ask the students to program quizzes or tests in different subjects.

Other ideas

Scratch offers even more possibilities and functionalities, but they may be too complex for elementary school students. For example:

- **Clones**, which can be used to duplicate a sprite in as many copies as desired, each considered to be the same sprite as the original (they obey the same programs). For the project in this sequence, you might, for example, want to display a random number of plants and place them randomly around the stage. The clones are found in the “Control” category.
- **Custom blocks**, which let the user create their own new instruction blocks. This can be very useful if you frequently re-use the same set of instructions. Rather than repeating them, grouping them in a custom block saves time, lightens the code and reduces errors. Custom blocks are available from the indigo menu “More blocks” (create a new block).
- **Sounds**: Using sounds can be very easy (each sprite can play a pre-recorded sound or a note with the pitch and volume adjusted), but it can quickly become cumbersome in a classroom setting.

Sequence 3: Sending News

	Lesson	Title	Page	Summary
	Lesson 1	How to send an image	296	Students must figure out how to transfer an image remotely. To do this, they learn that an image can be represented by a pixel grid. They learn about the notion of resolution as they see that the more pixels an image has, the clearer it becomes, but also the slower it is to transfer.
	Lesson 2	How to code a black and white image	304	Students apply what they learned from the previous lesson to coding black and white digital images. They first view a single file in a text editor and an image editor to understand how the information is coded. They then code a small checkerboard themselves.
	Lesson 3	(Optional) How to code a grayscale or color image	309	Students take what they learned in the previous lesson further by learning how to code a gray and color digital image.
	Lesson 4	How to ensure a message is secure	316	To protect their messages, students learn about encryption using a simple algorithm (called Caesar's cipher), which involves shifting the letters of a message.
	Lesson 5	(Optional) How to make sure our data are successfully sent	324	Students learn that it is possible to detect and correct errors introduced when storing or transferring a file by adding the right information. This lets them do a sort of «magic trick».

Lesson 1 - How to send an image

Summary	Students figure out how to send an image remotely. To do this, they learn that an image can be represented by a pixel grid. The learn about the notion of resolution as they see that the more pixels an image has, the clearer it becomes, but also the slower it is to send.
Key ideas *(see Conceptual scenario, page 204)*	"Information" • An image can be represented by a grid of squares called pixels.
Inquiry-based methods	Observation, experimentation
Equipment	For the class • Magnets or blu-tack to attach finished work to the whiteboard. For each group (four groups, A, B, C and D) • Handheld magnifying glass or microscope • Newspapers • Use large posters if there are no magnifying glasses or microscopes available • Photocopy of Image A, Handout 39 (page 302), for each student in Group A, Image B for Group B, etc. • Handout 40 (page 303) printed or photocopied on a slide or tracing paper and cut into 3 grids. Make sure there are extra grids. For each student • Tracing paper (1/4 of an A4 page) and sharpened pencil, or slide and fine-tipped permanent marker • Sticky tape or paper clips
Glossary	Image, pixel, resolution
Duration	1 hour 30 minutes

Decluttering situation

Introductory question The teacher explains that explorers want to photograph their discoveries, and send the photos to their base. "How can they send their photographs across long distances?" Students make suggestions: by courier, carrier pigeon, Facebook, scan or e-mail.
Even if students do not think of digitizing the photograph, the teacher then asks this question: "But what is an image?"

Research work: Defining an image (in groups)

The teacher hands out newspapers to each group. They ask the students to think about what an image is made of. The students mention materials: paper, cardboard and ink. When this work is mentioned, the teacher hands out the magnifying glasses or microscopes. "Can you describe how the ink looks in these pictures? What color is it?"

Very quickly the students will see that newspaper print is made up of thousands of tiny dots, and that the colors of these dots are in fact very limited. The teacher introduces the word 'pixel' ("picture element") and helps draw a conclusion such as the following: "A photograph is made up of tiny colored dots, called pixels. From a distance, we can not see the pixels, but an image that appears continuous".

Studying magazines through a microscope. Handheld magnifying glasses can be used if the print chosen is of basic quality, such as newspaper. 6th grade class in Paris

Teaching notes

- Depending on the quality of the microscopes and the print (laser versus inkjet, for example), the pixel overlap may make it hard to see them in magazines and photographs. This is why we recommend using newspaper. However, you can test the microscopes before the lesson to see if the pixels can be seen on a material other than newspaper.
- If there are no magnifying glasses available, pixels are visible to the naked eye on large-format advertisement posters. Pixels are not visible the further we are away from them.
- During the following lesson, we will look at the pixels again, but on a computer screen, to conclude that images are made up of tiny, discontinuous spots of various shades (see scientific note below).

Scientific notes

- For technical reasons, pixels on screens are rows of tiny squares (to be more precise, on color screens each square pixel is in fact made up of three rectangular sub-pixels: red, green and blue, from left to right. See below). On paper, the colored dots may overlap (the white in the paper is also used in reproducing colors).
- The pixel colors depend greatly on the device used. On a computer, tablet or smartphone screen, pixels exist in Red, Green and Blue. This is known as RGB printing. For images printed in four colors, these are Cyan, Magenta, Yellow and Black. This is known as CMYB printing. Two-color printing simply requires two complementary ink colors (blue and orange, for example). The combination of these few colors enables a wide variety of colorful creations to be reproduced.

Exercise (in pairs): How many pixels are needed for our image?

The teacher puts the conclusion into context: “To send an image, you need to send all the pixels, one by one.” The teacher suggests an exercise that enables the students to fully grasp this key idea and go further into depth by pixelating an image, i.e. replace it with a pixel grid. The teacher divides the class into four groups, and each group pixelates one of the four images (A, B, C, or D) on Handout 39.

- A copy of the image for their group. Do not forget to remind them that the other groups cannot know which image they have.
- Grid 1 on Handout 40 printed on a slide or tracing paper.
- Plain tracing paper (a quarter of an A4 page) with a pencil or slides (a quarter of an A4 slide) and fine-tipped permanent markers (in this example, we used tracing paper);
- Sticky tape or paper clips.

The students must place the grid over the image, ensuring the image corners are aligned within the crop marks. They place the tracing paper on top, attach the 3 papers with sticky tape or paper clips, and color in (on the tracing paper) the boxes that the image outline crosses.

Sixth grade class in Paris

When they have finished, they write the image letter (A, B, C or D) and the number of the grid (1, in this instance) on their Handout. Each group hands the teacher one or two copies of the pixelated image with 64 pixels, choosing the darkest pencil-colored boxes. The teacher displays the group work in 4 columns (“Image A”, “Image B”, etc.) allowing room for 3 subsequent lines (which will be named “Grid 1”, “Grid 2”, and “Grid 3”). The images pixelated with Grid 1 are unidentifiable.

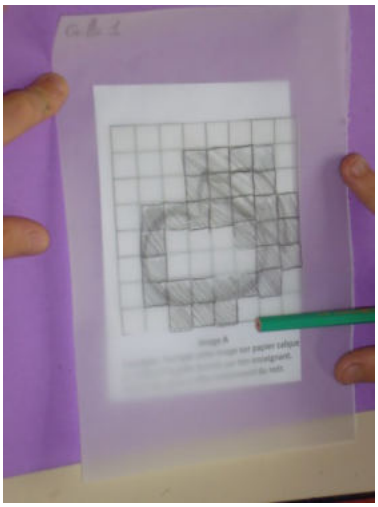
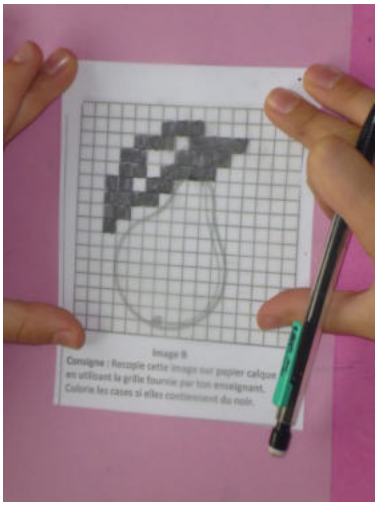


Image A pixelated with Grid 1. Anne-Marie Lebrun's fourth grade class (Bourg-la-Reine)

"How can we make these images clearer so that we can recognize the picture?" The students will come up with two ideas: either by using different shades of gray, instead of just black and white, or by adding more pixels. The first option — if it is suggested spontaneously — should be written on the whiteboard, and studied in further detail at a later stage (Lesson 3.3, page 309). To explore the second suggestion, the teacher hands out the finer grids in Handout 40

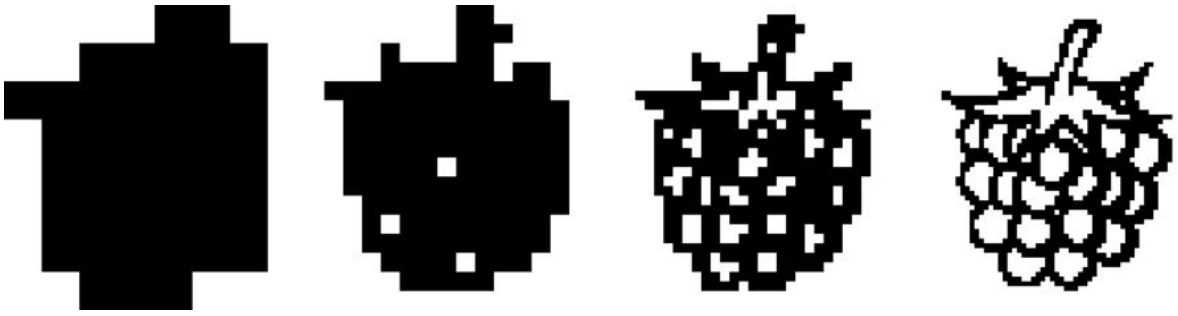


A student works on pixelating Image B with Grid 2. Anne-Marie Lebrun's fourth grade class (Bourg-la-Reine)

Teaching notes

- Hand out Grids 2 and 3 depending on the speed at which students in the same group work, in order to avoid lengthening the lesson time. Each student should complete the exercise at least once.
- An alternative method would be to hand out Grids 1, 2 and 3 to the groups from the start, rather than having the whole class use Grid 1. This saves 15 minutes, but the discussion on how to improve the first result will not be possible.
- Expect to repeat the instructions "color in the boxes fully or leave them blank" and "color in the boxes fully where the image outline passes through" several times. Do not hesitate to show the students how this should be done, using the whiteboard.

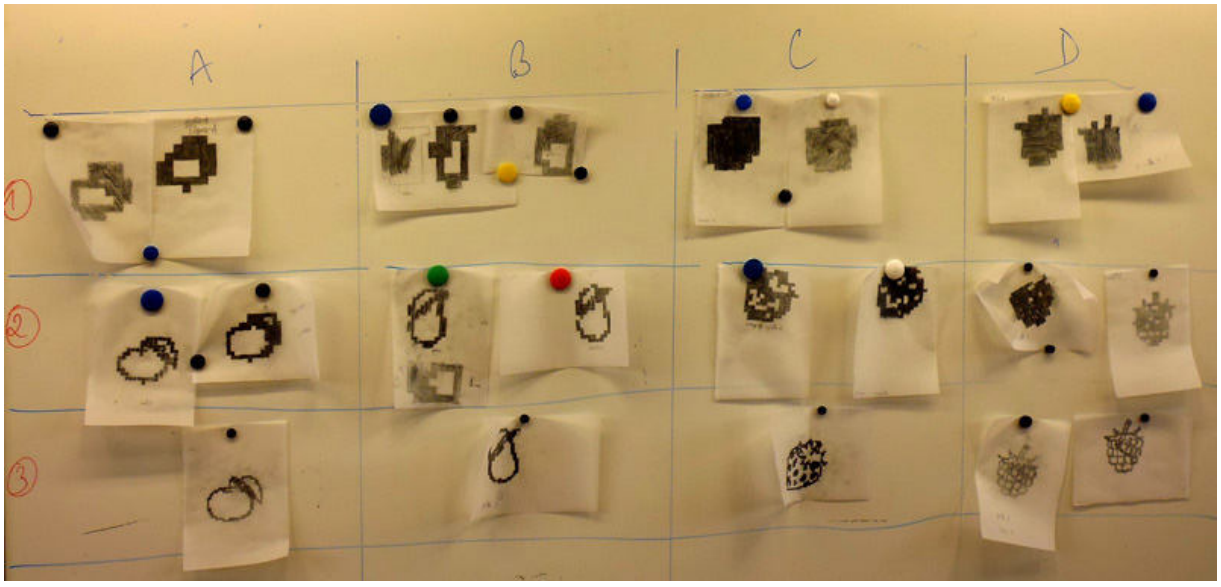
The teacher asks the pairs using Grid 2 to show their result beneath the previous images of their group. If the students from the other three groups appear to identify the picture, the teacher adds a sub-heading with their guess (apple? peach? pear?). Then they ask the pairs using Grid 3 and write down the new guesses.



From left to right, Image D pixelated with Grids 1, 2 and 3. Right: image pixelated with a 64x64=4,096-pixel grid (too long to use in the classroom).

Group discussion

The teacher asks the students if adding more pixels is an effective solution to the issue encountered (i.e. how to render the image intelligible despite the pixelation). The term "resolution" is introduced. "When we increase the number of pixels, the image resolution is higher, and it is easier to see what the picture is."
By comparing pixelated images with varying resolutions, the resolution requirement can be explained. Certain images were identifiable when using Grid 2, whereas for others Grid 3 was needed. The teacher must remind the class that all pixels must be sent one by one to the base before the image can be reproduced. They highlight the necessary compromise between resolution and ease of transfer: "If we had limited means, what resolution would be sufficient?" For each pixelated image, the class discusses and chooses a compromise resolution. For example, the lowest-quality resolution which renders at least three of the four images identifiable, or the resolution which at least shows the difference between the four images.



Images A, B, C and D pixelated using grids 1, 2 and 3. Anne-Marie Lebrun's fourth grade class (Bourg-la-Reine)

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *An image is made up of pixels.*
- *To send an image, you need to send all of its pixels one by one.*
- *The more pixels we use, the closer the pixelated image will be to the original, but it takes up more memory and takes longer to send.*

Students write down these conclusions in their science notebook. The teacher updates the "Information" section of the poster entitled "What is computer science?"

Further study

- Pixelated images can be used for artistic applications. For example, Post-it art (see examples here: http://www.postitwar.com) can be used to create posters or decorate walls with Post-its© that act as pixels. This could be a good opportunity to talk about art history, and pointillism in painting. See also the activity suggested in Level 2: Lesson 1.4 (page 123) and Handout 18 (Page 128).
- This work can extend to include photomosaics. In photomosaics, the pixel is made of an image. Looking closer, you can see the details of a myriad of miniature photos, whereas from a distance an entirely different image is seen. For example, miniature photos may be pictures of the students, and the overall image could be a panoramic image of the school, or a mythical creature or landscape, etc. Free software, such as AndreaMosaic (http://www.andreaplanet.com/andreamosaic) offers the possibility of creating photomosaics.

HANDOUT 39
Images for sending

Image A

Instruction: Copy this image onto tracing paper using the grid your teacher gives you. Color in the boxes if there is black in them.

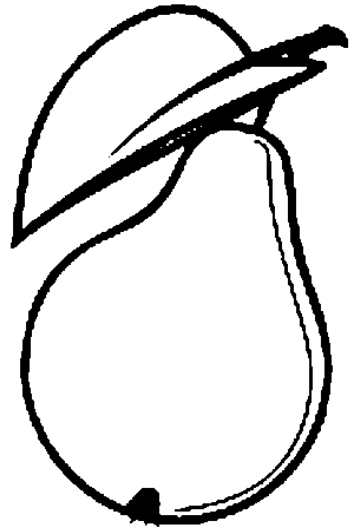


Image B

Instruction: Copy this image onto tracing paper using the grid your teacher gives you. Color in the boxes if there is black in them.

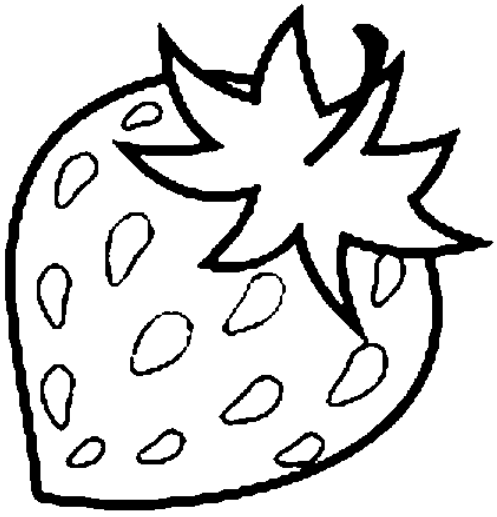


Image C

Instruction: Copy this image onto tracing paper using the grid your teacher gives you. Color in the boxes if there is black in them.

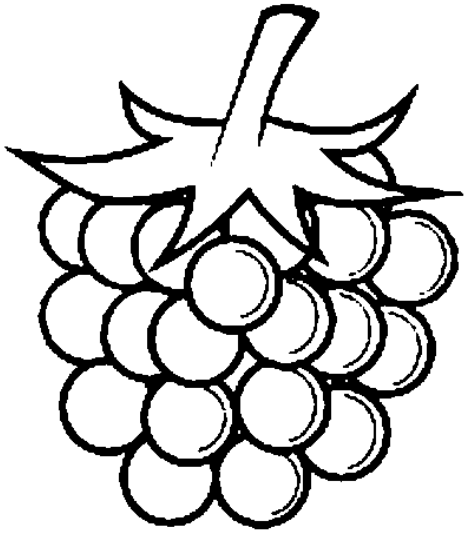
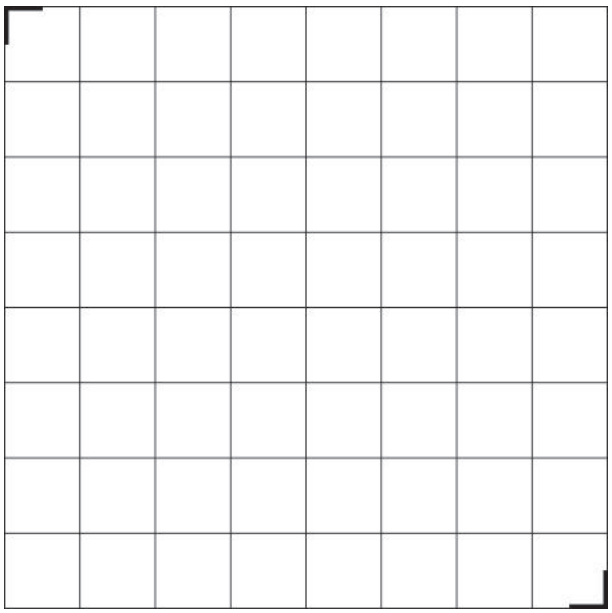


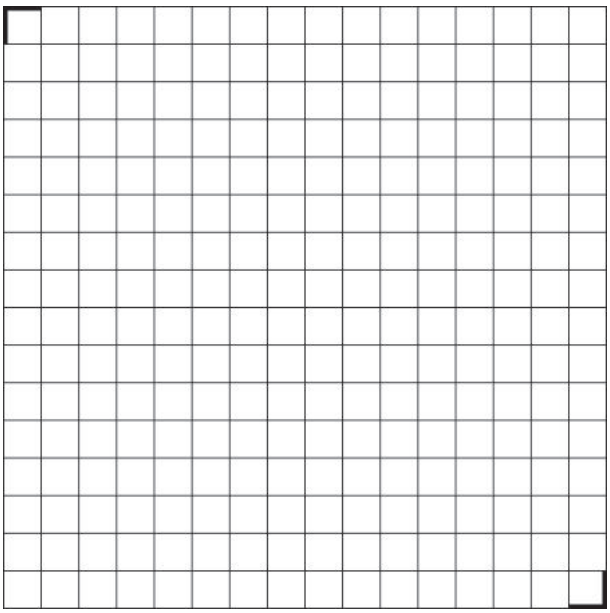
Image D

Instruction: Copy this image onto tracing paper using the grid your teacher gives you. Color in the boxes if there is black in them.

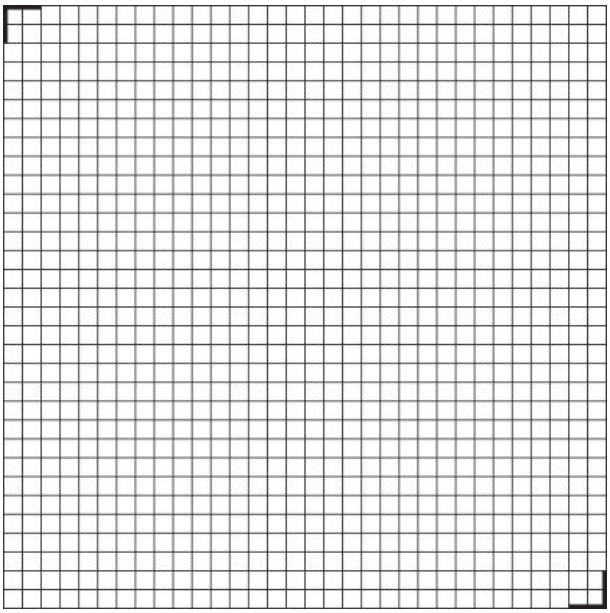
HANDOUT 40
Grids of varying density



Grid 1



Grid 2



Grid 3

Lesson 2 - How to code a black and white image

Summary	Students apply what they learned in the previous lesson by coding black and white images. They first view a single file in a text editor and an image editor to understand how the information is coded. They then code a small checkerboard themselves.
Key ideas *(see Conceptual scenario, page 204)*	"Information" • An image can be represented by a grid of squares called pixels. • A computer represents all information using a code with only two symbols, 0 and 1, called bits: this is binary code. • In black and white, each pixel can be represented by a single bit.
Inquiry-based methods	Observation, experimentation
Equipment	For each pair • Computer with basic text editing program (Notepad, for example) and a basic image editing program and file accessible to students containing the files 3-3.2_research_BnW and 3-3.2_challenge_BnW_blank. • Magnifying glass/microscope • Video projector For the class • Handout 41, page 308, projected on the whiteboard
Glossary	Image, pixel, coding
Duration	1 hour

Teaching notes

- It is highly recommended to use the image editor XnView in its minimal version, because of the following advantages:
  - It is free for educational use;
  - It correctly reads files in PBM format (and PGM and PPM for Lesson 3-3.3);
  - It has a simple interface;
  - Zoom function possible without entering text, using magnifiers "+" and "-";
  - An open image can be refreshed ("Reopen" in the "File" menu or use keyboard shortcut "Ctrl + R"), which facilitates students' trial and error.
- XnView can be downloaded here: http://www.xnview.com/en/xnview

Decluttering situation

Introductory question The students pay attention to the board, where the teacher shows them the picture of the apple they pixelated to 256 pixels (Grid 2) in the previous lesson (Handout 39, page 302, Image A): "*Our explorers want to send this image to the base, without travelling there. How can they do it*?"
In the ensuing discussion, the class suggests several methods and various concepts. The teacher writes the different ideas on the board. In particular, several suggestions should be explored

further, such as “there are white pixels and black pixels”, or “each pixel can be described in Morse code with a flashlight”.
The suggestions are an opportunity to remind students of the key idea of binary code (0 for black pixels, 1 for white pixels) if this key idea was addressed in Lesson 1 (page 219). “*Imagine if the explorers could not send a simple text message (SMS) to describe this image*.” Little by little, the students evoke the possibility of a text composed of 0s and 1s being enough to define the image.

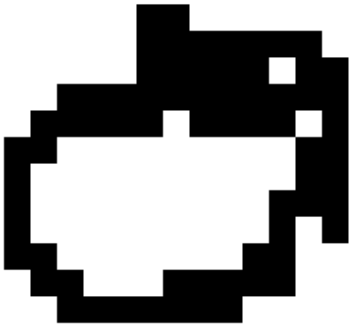
Observation: Understanding how a black and white image is encoded (first as a class, then in pairs)

In order to check if a text written with 0s and 1s is enough to represent a pixelated black and white image, the teacher shows the students how to open the text file 3-3.2_research_BnW.pbm with a basic text editor (Notepad in Windows, for example). The file contains the following:

```
P 1
1 6  1 6
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0
0  0  0  0  0  0  0  1  1  1  1  1  1  1  0  0
0  0  0  0  0  0  0  1  1  1  1  1  0  1  1  0
0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  0
0  0  0  1  1  1  1  1  0  1  1  1  1  0  1  0
0  0  1  1  0  0  0  0  0  0  0  0  0  1  1  0
0  0  1  0  0  0  0  0  0  0  0  0  0  1  1  0
0  0  1  0  0  0  0  0  0  0  0  0  1  1  1  0
0  0  1  0  0  0  0  0  0  0  0  0  1  0  1  0
0  0  1  1  0  0  0  0  0  0  0  1  1  0  0  0
0  0  0  1  1  0  0  0  1  1  1  1  1  0  0  0
0  0  0  0  1  1  1  1  1  1  1  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

The teacher explains that this file is a digital version of the image of the pixelated apple in 16x16 pixels. They ask the class if the grid of 0s and 1s appears to correctly illustrate the apple. It does, as we can see some of the leaf and the outline, visible in the 1s. If the students appear uncertain, the teacher can automatically replace all the zeros with periods (.) in Notepad using Ctrl + H. The apple can then be seen very clearly (this can be quickly undone with the command “Ctrl + Z”).
The teacher asks if there is anything else in the file aside from the grid of 0s and 1s. Students study the first two lines of the file and may suggest that 16 and 16 corresponds to the pixel grid’s dimensions. The teacher explains that these first lines define the format of the data written to

this file: “P1” indicates binary code (0s and 1s only) and the “16 16” line tells us that a table of 16 columns and 16 lines of data will follow. There are 16x16=256 bits (0s and 1s) in the file (description of the actual pixel grid).



The teacher then shows the class how to view this file with an image editor rather than a text editor, such as XnView (see teaching note at the beginning of the lesson). The apple appears very small. To see it properly, you need to zoom in completely. The class notices, by comparing the grid of 0s and 1s and the image of the apple, that the black pixels are encoded with a 1 and the white pixels with a 0.

They immediately practice what the teacher has shown them: they open a file using both a text editor and an image editor (see upper section of Handout 41). They look at the miniature apple using a magnifying glass on the screen before zooming in, then zoom in as much as necessary.

A group summary is written about the file format: P1 on the first line, the number of columns and the number of lines on the second, then a grid of 1s (for black) and 0s (for white) of suitable size on the following lines.

Scientific notes

- The PBM extension on the file means “portable bit map”.
- If, in the pixel grid, there are values other than 0 and 1, they are interpreted modulo 2: 0 produces white, 1 produces black, 2 (and all even values) produces white again, 3 (and all uneven values) produces black, etc.

Challenge in pairs: Digitally encoding a checkered grid

The teacher gives the class a challenge: they must use the text editor to create a file representing a small, black and white checkered grid with 5 lines and 5 columns, with black squares in each corner. The file, which is empty for now, is called 3-3.2_challenge_BnW_blank.pbm.

Then they save the file and open it with the image editor, without closing the text editor, and call the teacher once they have the file opened side by side in both programs, with the checkered grid visible in the image editor.

The students should suggest the following code:

```
P 1
5 5
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
```

Conclusion and lesson recapitulation activity

The class summarizes together what they've learnt in this lesson:

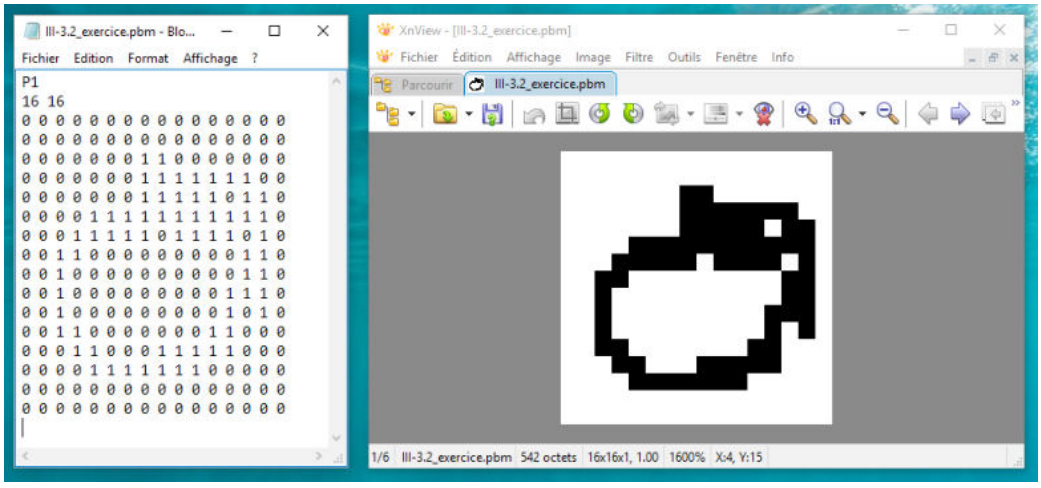
- *The pixels of an image can be represented by numbers.*
- *Each pixel in a black and white image is represented by either a 0 (white pixel) or a 1 (black pixel).*
- *To send a black and white image, we have to encode it and send the coded version, which is decoded upon receiving to reproduce the image.*

Students write down these conclusions in their science notebook. The teacher updates the "Information" section of the poster entitled "Defining computer science".

HANDOUT 41
How to code a black and white image

Research: Understanding how a black and white image is encoded

The same file, named 3-3.2_research_BnW, is opened on the left using a text editor and on the right using an image editor:



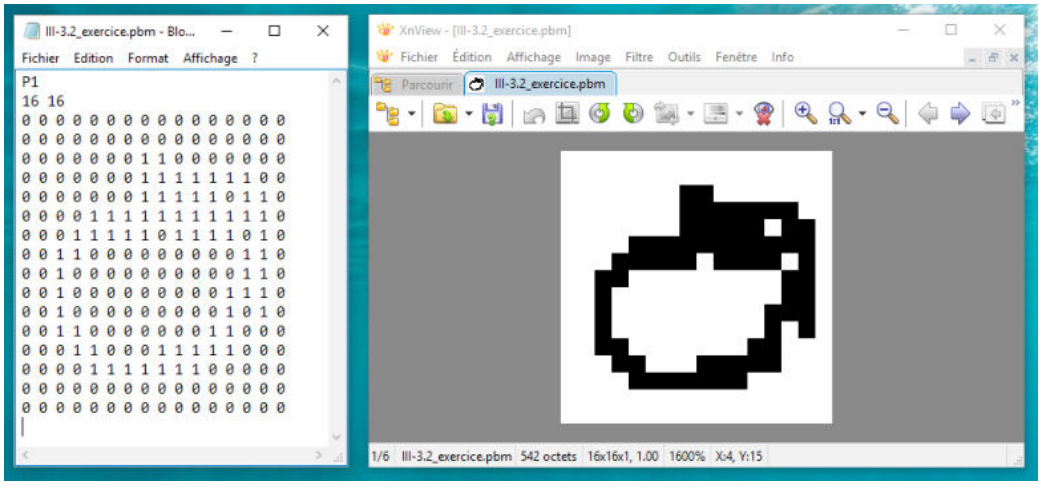
Instructions: Explain what conclusion you can come to on how to encode a black and white image.

Challenge

Encode the small checkered grid in file 3-3.2_challenge_BnW_blank.pbm and view it on the screen.

Research: Understanding how a black and white image is encoded

The same file, named 3-3.2_research_BnW, is opened on the left using a text editor and on the right using an image editor:



Instructions: Explain what conclusion you can come to on how to encode a black and white image.

Challenge

Encode the small checkered grid in file 3-3.2_challenge_BnW_blank.pbm and view it on the screen.

Lesson 3 - (Optional) How to code a grayscale or color image

Summary	Students take what they learned in the previous lesson further by learning how to code gray and color digital images.
Key ideas *(see Conceptual scenario, page 204)*	"Information" • An image can be represented by a grid of squares called pixels. • The more bits are combined, the greater the variety of elements can be represented. • In grayscale, every pixel can be represented by several bits. • In color, every pixel can be represented by 3 numbers (in one or several bits) that represent a quantity of red, green and blue.
Inquiry-based methods	Observation, experimentation
Equipment	For each pair • Computer with basic text editing program (Notepad, for example) and a basic image editing program (XnView, for example) • Folder that can be accessed by students, containing files 3-3.3_research_grayscale, 3-3.3_challenge_grayscale_blank, 3-3.3_research_color and 3-3.3_challenge_color_prefilled. • Hand-held magnifying glass For the class • Video projector • Handouts 42 (page 314) and 43 (page 315) projected on the whiteboard
Glossary	Image, pixel, coding
Duration	1 hour 30 minutes, which can be divided into two 45-minute lessons

Introductory question The teacher returns to the conclusions of the previous lesson: a black and white image can be coded with a pixel grid, by indicating 0 for a white pixel and 1 for a black pixel.
If the students suggested during Lesson 3-3.1 to improve the reproduction of the images of the fruits by coloring in the boxes in lighter or darker shades, the teacher picks up this suggestion, noted on the whiteboard: "When we pixelated the images of the fruits, you suggested not only coloring in the boxes in black or leaving them white, but also coloring in some boxes in gray." Whether or not the suggestion was made by the students, the teacher then opens a grayscale image using the image editor (file 3-3.3_research_grayscale) and zooms in sufficiently so that the pixels are visible. The teacher asks the class if this image has only black or white pixels. The students remark that there are light, medium and dark gray pixels, as well as black and white pixels, which brings them to the first research activity.

Research (in pairs and as a class): Encoding an image in grayscale

The teacher asks if 0s and 1s will be enough to encode the black, white and various shades of gray pixels. The students answer “no”, and suggest adding in other numbers aside from 0 and 1. The teacher asks the students to try and figure out how this image was encoded (they have access to the file 3-3.3_research_grayscale and can open it with the program of their choice). If they have difficulty, the teacher can suggest opening the file using both the text editor and image editor programs (see upper section of Handout 42).

In a few minutes, the class report back together. They will have noticed, by comparing the file opened with the text editor and with the image editor, that:

- P2 is indicated on the first line, and not P1 (the teacher explains that this means that what follows will be encoded in shades of gray).
- On the second line, there is an indication of the number of lines and the number of columns (here 15 15), as was the case for the coding of black and white images.
- There is an extra line where the figure 7 is indicated, the highest figure that appears in the pixel grid that follows.
- The pixel grid contains figures between 0 and 7, i.e. 8 different figures (and not just 0s and 1s as was the case for the coding in black and white).
- The lower the figure, the darker the corresponding pixel (black pixels encoded by 0, increasingly lighter shades of gray pixels encoded with figures 1 through 6, and white pixels encoded by 7). The opposite was true for black and white coding (0 corresponded to white pixels).

If only some or none of these observations were made in pairs, the teacher guides the class. If questions remain, the teacher can encourage the students to make changes to the file in the text editor, and refresh the image in the image editor (File/Reopen), to see what has been changed.

Scientific notes

- Grayscale images are usually coded with a number of shades corresponding to a power of 2 (for example, 8, 16, 32 or 256 grayscale), which corresponds to a code for each pixel in 3 bits, 4 bits, 5 bits and 8 bits, respectively) (see Lesson 1.3 page 219).
- The PGM extension of the image files means “portable grayscale map”.

Challenge (in pairs): Encoding a grayscale

The teacher gives the class the following challenge: they must use the text editor to create a file representing a grayscale, like the one projected on the board (point out the white pixel on the left if it is not visible). The file, which is empty for now, is called 3-3.3_challenge_grayscale_blank.pbm.

Then the students save the file and open it with the image editor, without closing the text editor, and call the teacher once they have the file opened side by side in both programs, with the grayscale visible in the image editor.

A possible code, that most pairs will suggest, is the following:

```
P2
16 1
15
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

To achieve this result, the students will have counted the number of gray levels on the scale (16, in this case) and adapted the third line accordingly. They must also have adapted the format of the pixel grid (16 columns and a single line) and encoded the pixels with values in descending order from 15 to 0, so that the white pixel appears on the left and the black pixel on the right, to match the request of the challenge.

Here is another possible code (among many others):

```
P2
16 1
31
31 29 27 25 23 21 19 17 15 13 11 9 7 5 3 0
```

Research (in pairs and as a class): Coding a color pixel

The teacher reminds the class that images can also be in color. They indicate the file 3-3.3_research_color, which allows the class to code a single pixel in color. The teacher asks the students to try and understand how coding works using this file, as they did for the grayscale example. They suggest only touching the last line to begin, and to look at the result on the screen with magnifying glasses (that they distribute).

The coding suggested in the file, which yields a red pixel, is the following:

```
P3
1 1
7
7 0 0
```

The students can try, for the third line, full colors 0 0 0 (black pixel), 7 7 7 (white pixel), 0 7 0 (green pixel), 0 0 7 (blue pixel), 0 7 7 (cyan pixel), 7 0 7 (magenta pixel), 7 7 0 (yellow pixel) and several other intermediary combinations of their choice (for example, 7 6 2, which produces a golden yellow or 1 5 6 which produces a sky blue).
During the group discussion, the students report that using the magnifying glass, they see little red, green and blue lights, of rectangular shape, aligned repetitively in this order, from left to right on the screen. When 0 0 0 is indicated on the last line of the file, the lights on the screen (of all colors — red, green and blue) are not very bright under the magnifying glass, whereas with 7 7 7, they are very bright. With 7 0 0, only the red lights are bright, etc. The class concludes together that the three values of the last line indicate the level of brightness of the pixel in red, green and blue (the square pixel has therefore 3 rectangular sub-pixels in three shades, R, G and B). The impression we get at a distance from the screen depends on the balance between the brightness in each shade. The color obtained for the following combinations is kept visible on the board, in order to complete the following challenge:

000	777	700	070	007	077	707	770
Black	White	Red	Green	Blue	Cyan	Magenta	Yellow

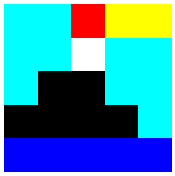
The teacher can point out that, generally, there are 256 levels of red, green and blue possible, and not just 8. They specify that for the next challenge, the class is going to reduce the levels of red, green and blue, to have just two: switched off or switched on, just like for coding in black and white.

Scientific notes

- The ppm file extension means “Portable Pixel Map”.
- Usually, the sub-pixels are encoded in 8 bits, which corresponds to 256 levels of possible intensity (values fro 0 to 255). Therefore, 24 bits are required to describe a pixel in RGB (see Lesson 1.3 page 219).

Challenge: Code a mini-lighthouse in color

The teacher gives the class a final challenge. They have to code a small lighthouse in color, using only two levels of luminance for the sub-pixels: 0 for an “dark” sub-pixel and 1 for a “bright” sub-pixel.



In other words, the file 3-3.3_challenge_color_prefilled to be modified begins with:

```
P3
5 5
1
```

Followed by a grid of 0 (off pixels) and 1 (on pixels) values. The file is pre-completed with the correct number of 0s (15 per line, across 5 lines). A comment line beginning with the hashtag (#) makes finding the blocks of three sub-pixels easier.

The students who complete the challenge correctly suggest the following code:

```
P3
5 5
1
#  #  #  #  #
0 1 1 0 1 1 1 0 0 1 1 0 1 1 0
0 1 1 0 1 1 1 1 1 0 1 1 0 1 1
0 1 1 0 0 0 0 0 0 1 1 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 1
```

This time, there is only one possibility, as the 1-bit code (0 or 1 for each sub-pixel) is obligatory in the challenge.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *The pixels of an image can be represented by numbers.*
- *Each pixel of a grayscale image is represented by a number. If the image has 8 shades of gray, this number is included between 0 (black pixel) and 7 (white pixel).*
- *Each pixel of a color image is represented by 3 numbers which indicate the intensity of red, green and blue of this pixel.*
- *To send a grayscale or color image, we have to encode it and send the coded version, which is decoded upon receiving to reproduce the image.*

Students write down these conclusions in their science notebook. The teacher updates the "Information" section of the poster entitled "Defining computer science"

Further study

The students can practice by taking well-known images (Mario©, Tintin©'s rocket, the sword in Minecraft©, etc.) and, to begin, pixelate them with grids like those used in Lesson 3-3.1 (16x16 grid for example) and then encode the images in a PBM, PGM or PPM file in order to make them visible on a screen.
In science and art, we can also study this topic further by recomposing colors in red-green-blue triplets.

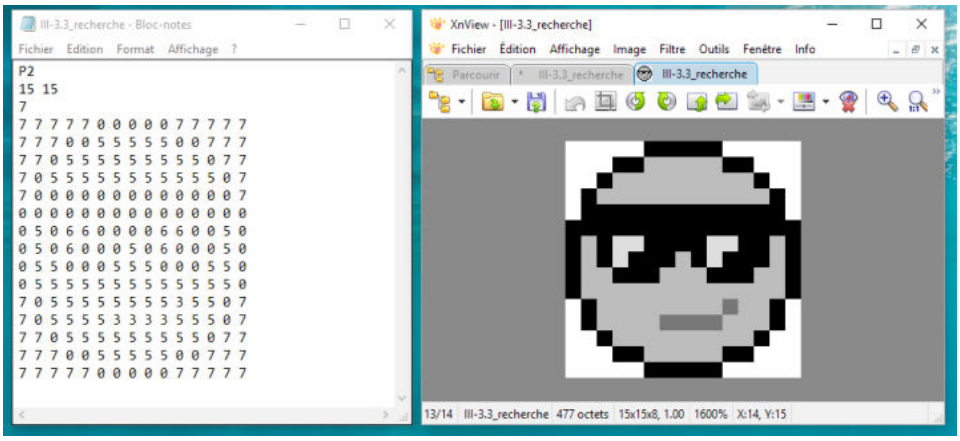
Scientific notes

- Additive color (sometimes called "light color" at school) means adding streams of light colors together in order to recreate all the colors of the visible spectrum. Inversely, subtractive color (sometimes called "material color" at school) combines colored pigments which, when brought together, absorb certain colors of the surrounding light and reflect others.
- By using three pocket torches covered with red, blue and green colored transparent paper (sweet wrappers for example), students can attempt to reproduce various colors, from dark to light: cyan, magenta, yellow and white.

HANDOUT 42
Encoding grayscale images

Research: Understanding how a grayscale image is encoded

The same file, named 3-3.3_research_grayscale, is opened on the left using a text editor and on the right using an image editor:



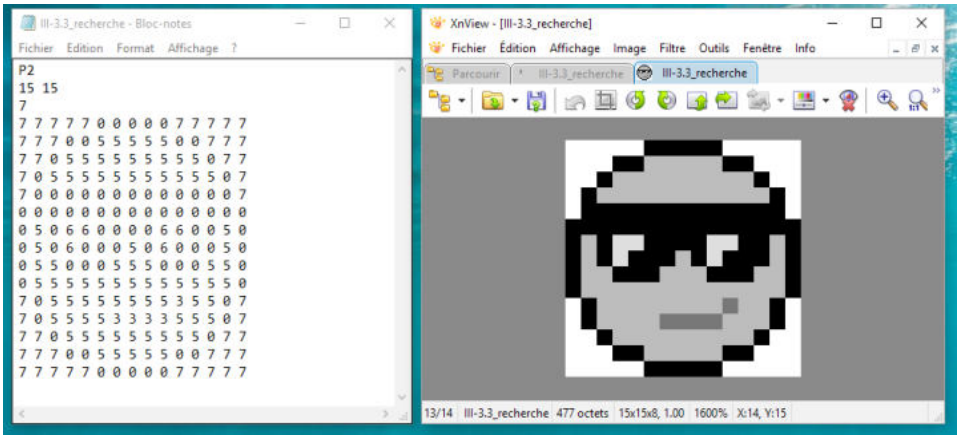
Instructions: Explain what conclusion you can come to on how to encode a grayscale image.

Challenge

Encode this grayscale in the file 3-3.3_challenge_grayscale_blank.pbm and view it onscreen (do not forget the white pixel on the left!).

Research: Understanding how a grayscale image is encoded

The same file, named 3-3.3_research_grayscale, is opened on the left using a text editor and on the right using an image editor:



Instructions: Explain what conclusion you can come to on how to encode a grayscale image.

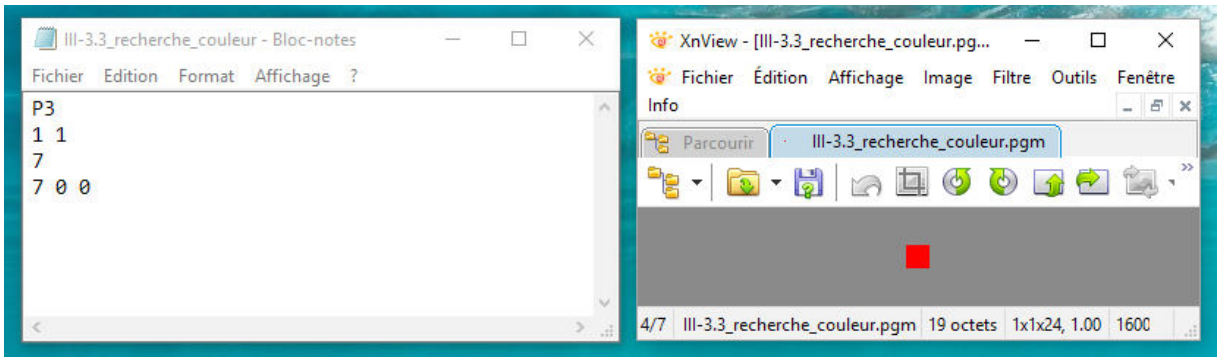
Challenge

Encode this grayscale in the file 3-3.3_challenge_grayscale_blank.pbm and view it onscreen (do not forget the white pixel on the left!).

HANDOUT 43
Encoding color images

Research: Understanding how a colored pixel is encoded

The same file, named 3-3.3_research_color, is opened on the left using a text editor and on the right using an image editor:



Instructions: Replace the figures 7, 0 and 0 in the fourth line of the file by other values between 0 and 7, save, refresh the display in the image editor and observe the changes (use your magnifying glass!). Explain what can be concluded on how a colored pixel is coded.

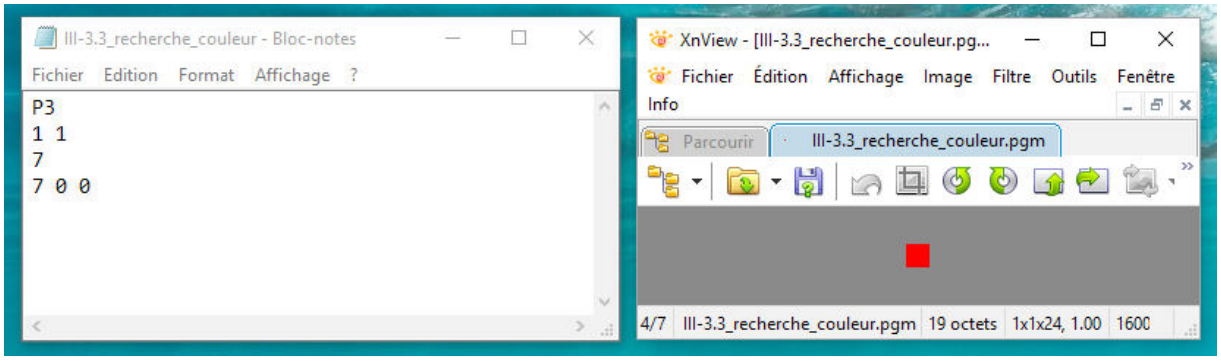
Challenge

Code this lighthouse in the pre-completed file 3-3.3_challenge_color_prefilled.pbm and view it on the screen.

✂----------

Research: Understanding how a colored pixel is encoded

The same file, named 3-3.3_research_color, is opened on the left using a text editor and on the right using an image editor:



Instructions: Replace the figures 7, 0 and 0 in the fourth line of the file by other values between 0 and 7, save, refresh the display in the image editor and observe the changes (use your magnifying glass!). Explain what can be concluded on how a colored pixel is coded.

Challenge

Code this lighthouse in the pre-completed file 3-3.3_challenge_color_prefilled.pbm and view it on the screen.

Lesson 4 - How to ensure a message is secure

Summary	To protect their messages, students learn about encryption using a simple algorithm (called Caesar's Cipher), which involves shifting the letters of a message.
Key ideas *(see Conceptual scenario, page 204)*	"Information" • Encrypting a message means transforming it so that only the person to whom it is sent can understand it. • Caesar's Cipher is an easy method to use, but it is also easily decrypted.
Inquiry-based methods	Experimentation
Equipment	For each student • Handout 44, page 321 • Handout 45, page 322 For each group - Handout 46 (page 323) (only if the teacher chooses to do the extra activity, where students make a tool to encrypt and decrypt a message)
Glossary	Encrypting
Duration	1 hour 30 minutes

Decluttering situation

Introductory question The teacher tells the class about the space mission: "The discovery team needs to communicate regularly with base. The base is also in permanent communication with the Earth. But we have to be sure that the messages can not be intercepted by hostile powers that might spy on our results and endanger the team's safety. What can we do to help them?" The class discusses different ways of maintaining confidentiality: the words "secret language" and "coded language" very quickly emerge. Here, there is a risk that students confuse the "coding" presented in the previous lessons (in the sense of binary code) and the use of the word here (in the "encrypted" sense). *The teacher explains the new term, "encrypted language". Encryption means changing a text to make it less intelligible, so that non-authorized persons cannot access the content of the text.*[17]

Experiment: Decrypt the explorers' message (as a class activity)

The teacher hands out the encrypted sentence on Handout 44 to students: "Here is a message sent by the explorers to the base. Can you decrypt this message?"

17 In terms of vocabulary, see the scientific note in Lesson 1.2 (page 213)

Teaching notes

- To make the encryption simpler and focus on the method rather than the result, we will not include punctuation.

The class quickly figures out that the message is written backwards. By reading from right to left, we can discover the content:

RIVAL TEAM SPOTTED WE MUST CIPHER OUR COMMUNICATIONS

The class will have noticed how easy it was to "break" this first encryption. Encryption in "mirror writing" is therefore not very secure. We will now study a slightly more complicated encryption method, one of the first to be used in history.

Experiment: Decrypt the message from the base (as a class activity)

The teacher gives each student Handout 45. "In response to the alarming message from the explorers, the base replied the following. Can you decrypt this message?"
It is clear, this time, that the message is not encrypted with "mirror writing". If the students struggle to decrypt this message, the teacher can gradually give them pointers, in several ways:

- *What are the shortest words? What might they correspond to in English?* The shortest word in the English language is "a". There are also some two-letter words (to, on, in, as, if, at, etc.).
- *What is the most commonly used letter in a written text in English? (answer: the letter E). What about in this example?*

In the ciphered text, the most commonly used letter is H. We can therefore suppose that "H" systematically replaces all the 'E's in the initial message.

The cipher used here, called the Caesar Cipher, shifts all letters in the alphabet three spaces forward: A become D, B becomes E, C becomes F, E becomes H, X becomes A, Z becomes C. This is also known as cyclic permutation. Decrypted, the message becomes:

UNDERSTOOD LET US USE CAESAR S CIPHER

Scientific notes:

- Caesar's Cipher takes its name from Julius Caesar, who used it for his secret communications during the Gallic Wars.
- The key to this code is the shifting of letters. In Caesar's Cipher, the letters are all shifted by a certain number (the key). In the example used, the key is +3, which means that, to encrypt the message, we just need to move all letters 3 places forward in the alphabet (A becomes D, B becomes E, W becomes Z, X becomes A, Y becomes B, etc.).
- With a key of 0, the letters do not shift, and therefore the encrypted message is identical to the original. With a key of -3, the letters shift in the other direction (A becomes X, B becomes Y, etc.) which is how we decrypt a message encrypted with the key +3.

Experiment: finding other forms of encryption (in groups)

The third part of this lesson lets students reuse the key ideas dealt with thus far. The teacher suggests: “Now, in groups, you must improve Caesar’s Cipher to scramble your messages.” Firstly, the students try to encrypt and decrypt short messages that they invent. Next, the groups exchange encrypted messages and try to break the encryption key to their neighboring groups’ messages.

Scientific notes:

- There are many encryption methods. It is likely that the students’ first attempts will be to change the key of Caesar’s Cipher. A variation on Caesar’s Cipher that may emerge during the experiments would be a variable key that follows a very specific pattern. For example, the first letter of the message might be shifted +1, the second +2, the third +3, and the fourth will be back to +1, the fifth +2, and so on. There are infinite possibilities. The students might also think of deleting spaces, which prevents short words from being spotted and therefore makes identifying the key used in Caesar’s Cipher more difficult.

Group discussion

One student per group stands up to present the encryption method that their group invented. The class then discusses the reasons why their encryptions were easy or difficult to break. This allows student to gradually see that there are several encryption strategies. The class could possibly invent a common encryption and write a short text that you can send around neighboring classes to test its strength.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

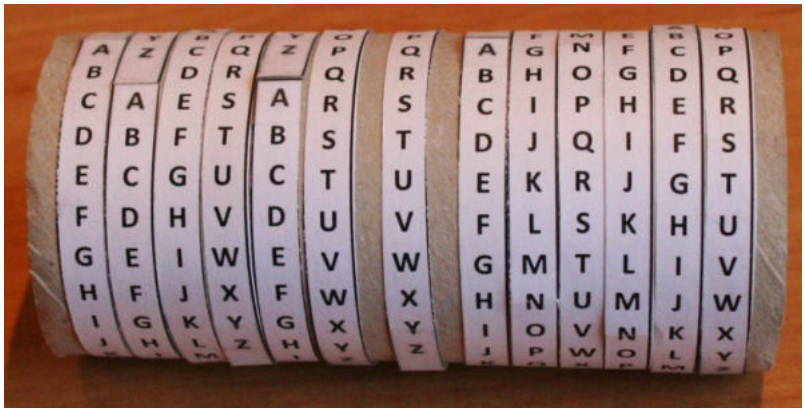
- *Encrypting a message means transforming it so that only the person to whom it is sent can understand it.*
- *Caesar’s Cipher is an easy method to use, but it is also easily decrypted.*

Students write down these conclusions in their science notebook. The teacher updates the “Defining computer science” poster.

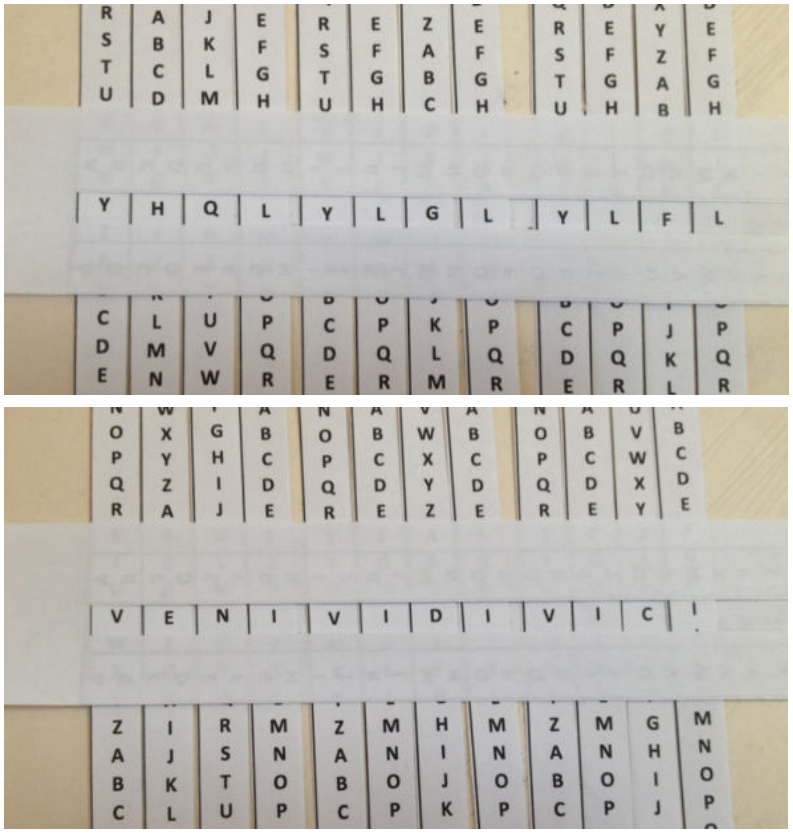
Further study

- Documentary study on Alan Turing is highly appropriate after working on encryption. He directed the team that broke the famous Enigma, the Nazis’ secret code, contributing to the development of the machines that preceded computers. Read more about this topic on page 9.
- Create encryption and decryption tools:
  - First type of tool: a cylinder with wheels that spell out the alphabet. In the image below, strips measuring 138x5 mm are printed with all letters of the alphabet from

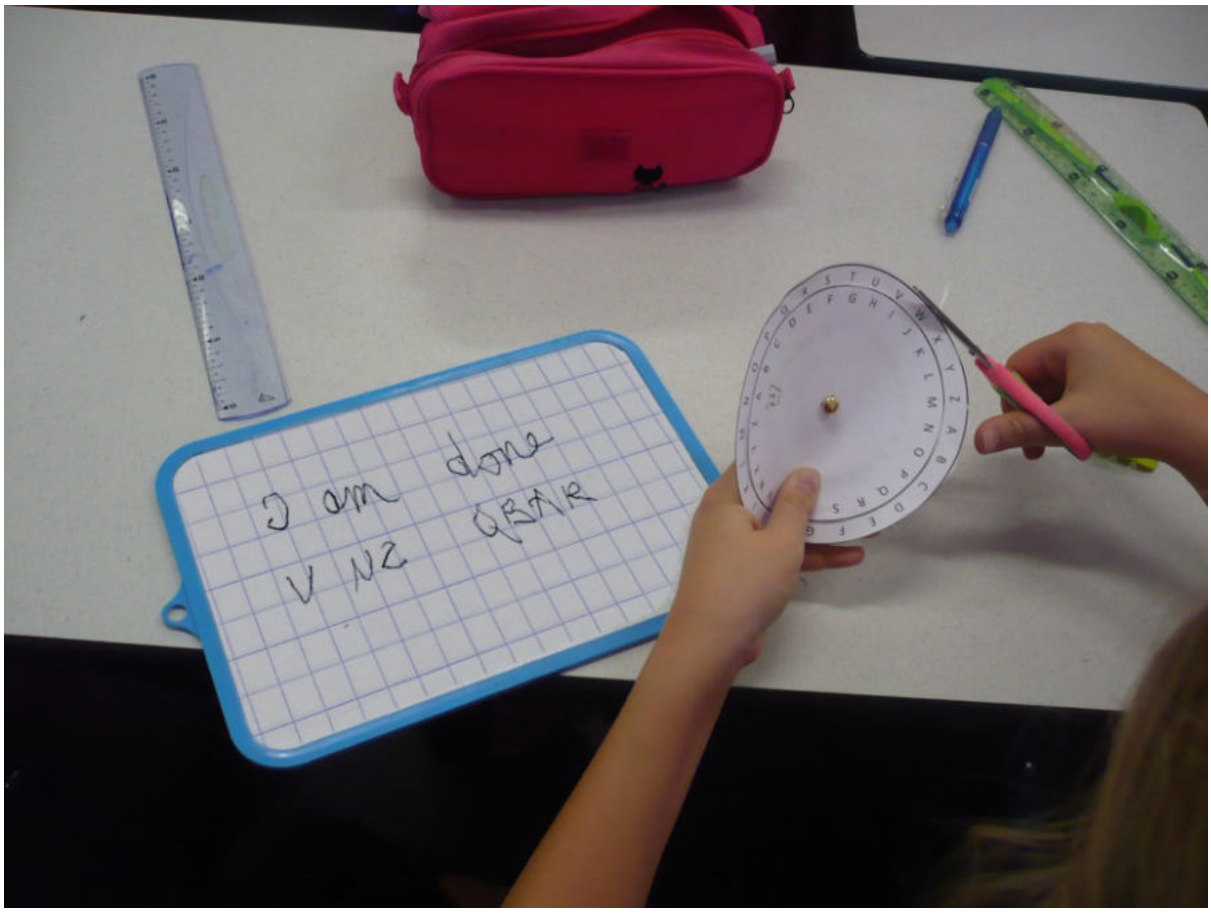
A to Z, then wrapped around a cardboard roll. The strips are taped to themselves, and not to the cardboard, so that the roll can be used as an axis. By turning the wheels, we can rapidly encrypt and decrypt a message. Here, CAESAR'S CIPHER (which can be read on the central line) becomes "FDHVDU V FLSKHU", and so on.



- Second type of tool: a slider system, with strips laid next to each other. Each strip has the alphabet printed twice. Using a ruler, the sliders can be aligned to reveal the message. Then we slide the ruler vertically in one direction and the other to read the encrypted message.



- Third type of tool: two concentric discs attached with a brass fastener. Around the edges of the discs, the alphabet letters are placed. By pivoting one disc, it is easy to quickly encrypt and decrypt any letter.



Kévin Faix's fourth grade class (Le Kremlin Bicêtre)

HANDOUT 44
Simple examples of secure communication

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

✂ -

Instruction: The base has received the following message. Can you decrypt it?
SNOITACINUMMOC RUO REHPIC TSUM EW DETTOPS MAET LAVIR

HANDOUT 45
Another example of secure

Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

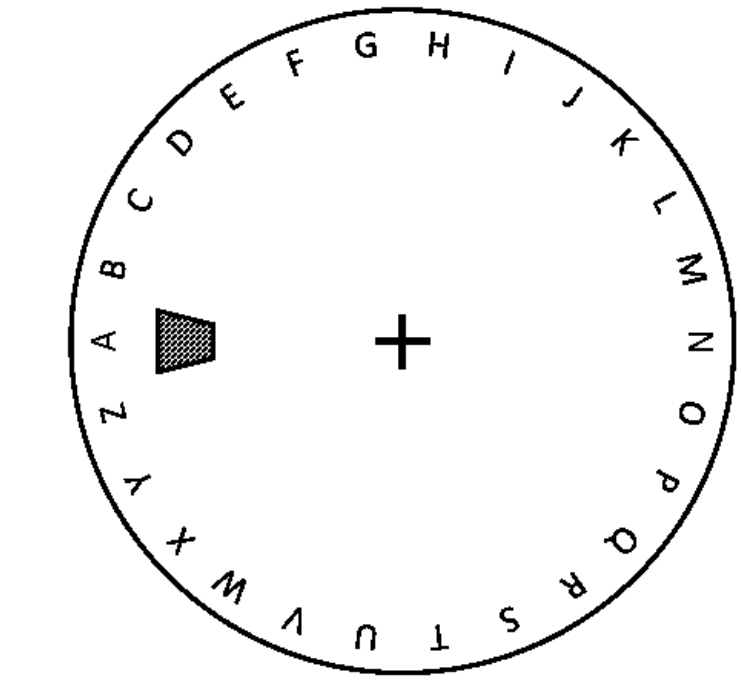
Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

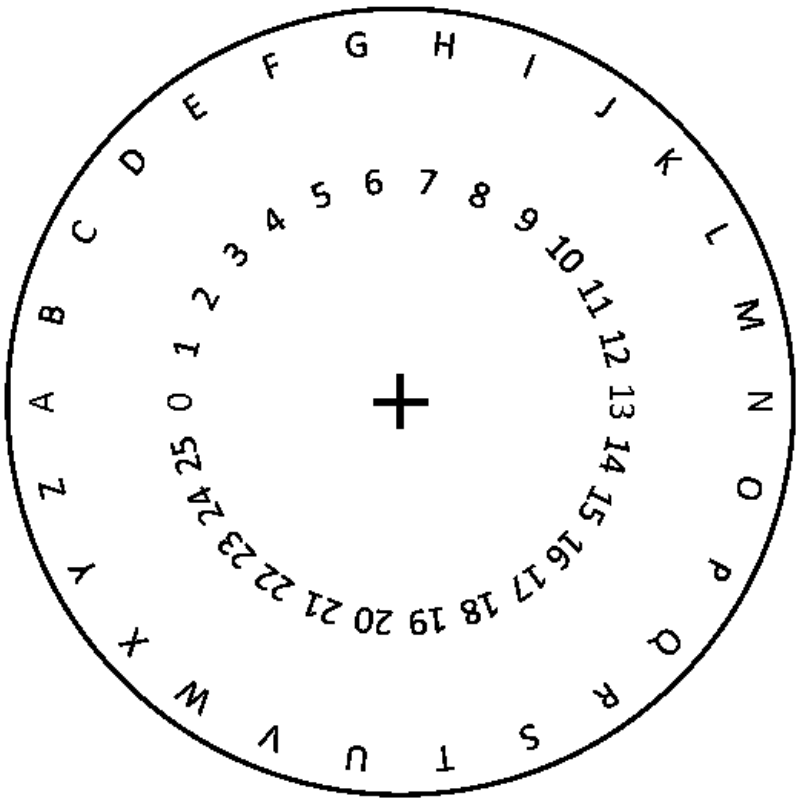
Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

Instruction: The base sends this new message to the explorers. Can you decrypt it?
XQGHUVWRRG OHW XV XVH FDHVDU V FLSKHU

HANDOUT 46
Making encryption and decryption tool



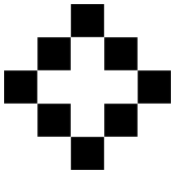


Lesson 5 - (Optional) How to make sure our data are successfully sent

Summary	Students learn that it is possible to detect and correct errors introduced when storing or transferring a file by adding the right information. This lets them do a sort of «magic trick».
Key ideas *(see Conceptual scenario, page 204)*	"Information" • The information is stored in a memory: hard disk or flash memory, for example. • Storing and handling data with these memories may introduce errors. • There are methods that allow us to detect and correct these errors. This requires increasing the quantity of information to be stored.
Inquiry-based methods	Experimentation
Equipment	For each group • A pack of 36 cards (standard playing cards or simply cards with one black side and one white side)
Glossary	Parity bit, data integrity
Duration	1 hour

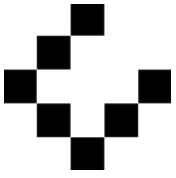
Foreword

It is possible that students will not come up with the method known as "parity checking" themselves. We propose two variations of this lesson. In the first (described step by step, below), this method is formulated together as a class. In the second (a quick description at the end of the lesson), the teacher does a "magic trick", which the students should be able to understand. Introductory question The teacher shows an image similar to that studied at the beginning of Lesson 3.1 (page 296) and its binary code (here, we have removed the heading of the file "P1 5 5" which indicates that it is a black and white image with 5x5 pixels):



0 0 1 0 0
0 1 0 **1** 0
1 0 0 0 1
0 1 0 1 0
0 0 1 0 0

The teacher asks the students what would happen if, when sending, certain errors occurred, that changed the value of certain pixels. They change any value, and ask a student to come to the board and draw the new image. If the teacher has changed the underlined value, the new image is:



0 0 1 0 0
0 1 0 **0** 0
1 0 0 0 1
0 1 0 1 0
0 0 1 0 0

Teaching notes

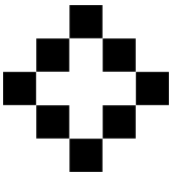
- This initial question is deliberately chosen to be very easy, which enables the class to recapitulation and refresh what they learned in Lesson 3.1 (page 296).

Research (as a class): How to detect an error

The teacher asks the students how we can make sure that the image we receive is indeed the one that was taken. The aim is not yet to correct potential errors, but just to learn how to detect them. Another way of saying it is to ensure the data's integrity.
Some students may have the idea of adding extra information. They could, for example, calculate the sum of the values line by line and add a number that encodes this value. Or, a simpler (and more practical) option would be to see if the number of 1s on each line is odd or even. In that case, at the end of each line a new number is added so that the number of 1s is always even on every line.

- If the number of 1s on the line is already even, then we add a 0 at the end of the line. The new, completed, line still has an even number of 1s.
- If the number of 1s on the line was uneven, then we would add a 1 at the end of the line. The newly completed line now has an even number of 1s.

When applied to the first image, the result is as follows:



```
00100      001001
01010      010100
10001      100010
01010      010100
00100      001001
```

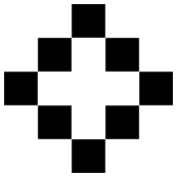
We added a new column so that, on each line, the number of 1s would be even. The teacher closes their eyes and asks a student to change any value without telling them. All they need to do, then, is count, line by line, the number of 1s to find out which line poses a problem.

Scientific notes:

- This method is often used in IT to detect (and correct, see below) errors when copying or sending files.
- The added bit at the end of the line is called a "parity bit".

Research (class activity): How to correct the error

The class points out that they can tell there is an error, but they do not know how to correct it, because we do not know which pixel has been changed. All we know is the number of the line where there is a problem. In order to find out precisely which value has been changed, we need to know the number of the column.
The teacher asks the class how to do it. This time, it is easier for the students to find the answer. They need to do the same as before, but in columns rather than lines. We count the number of 1s in each column and add another line so that, in each column, the number of 1s is always even. When applied to the first image, the result is now as follows:



```
00100
01010
10001
01010
00100
```

```
001001
010100
100010
010100
001001
100010
```

A student can now change a value anywhere and the teacher shows that they are able to detect this value (by cross-referencing the information in the line and the modified column) and then correcting it.

Scientific notes

- The parity bit added in the bottom-right corner allows us to detect (and correct) an error that has occurred, not in the bits of the 5x5 grid (the original image) but in the parity bits themselves.

Exercise (in groups)

Once the method is explained, the teacher divides the class into pairs or groups of four (depending on the equipment available) and hands a set of cards to each pair. The students practice detecting and correcting errors.

- The student being tested covers their eyes while the rest of the group agree on a pattern they make with the cards (each face card is a 1, while each pip card is a 0).
- The student being tested opens their eyes and places the extra cards (“parity bits”).
- The student being tested covers their eyes again while the others turn over 1 card.
- The student being tested should be able, when looking at the parity bits, to figure out which lines and columns were changed, and therefore correct the errors.

Teaching notes

- If the class is not sure whether the student is using the parity method and not just playing by memory to identify and correct the error, they just need to increase the number of cards (i.e. the number of bits) and avoid creating an easily memorized pattern.

Group discussion

The teacher makes sure that all students have understood the parity checking method and asks them what would happen if two errors, for example, occur on the same line. The class realizes that, in this case, the error is detectable because the parity of certain columns has changed, but the error cannot be corrected because we do not know which line has been corrupted.
The class discusses how to minimize this risk. For example, we could increase the number of parity bits (instead of placing one parity bit for every five bits, as we have done here, we can place a parity bit every three bits). The higher the number of parity bits, the easier it is to detect errors. However, on the other hand, the quantity of information to be stored (and later sent) must be increased.
In practice, inserting one parity bit every five bits means increasing the information by 20%, and so the file size also increases by 20%. Depending on the volume of information and the reliability of the storing or sending processes, we can decide if we reduce it to 10% or even 1%: it is a compromise to be made.

Conclusion and lesson recapitulation activity

The class summarizes together what they learned in this lesson:

- *A computer represents all types of information using binary code.*
- *When saving or handling (copying or sending, for example) data, errors can occur.*
- *Certain methods, such as parity checking by adding parity bits, allow us to detect and correct these errors. This requires increasing the quantity of information to be stored.*

Students write down these conclusions in their science notebook. The teacher updates the "Defining computer science" poster.

Variation

This lesson can be conducted differently, by breaking away from the inquiry-based method, where the problem posed has no relation to what was studied in previous lessons. However, this variation can be of value, even if it's just for fun.

- The teacher explains that they will do a "magic trick".[18]
- The class is divided into four groups. Each group has 25 cards on the table, laid out in a pattern of five lines by five columns.
- The teacher goes from table to table, placing extra cards at the end of each line and column, making sure to keep them even (the number of face cards should be even), without explaining to the students what they are doing.
- They then ask each group to change one card (just one!) and then checks each group, revealing the card that was changed.
- The rest of the lesson can be conducted similarly to what was described above, so that the students discover and grasp the teacher's method.

The students can do this magic trick at home — success guaranteed!

Extended study for middle school (4th Level)

The class can extend this study, for example by looking for the four errors that cannot be detected or corrected. They may also study the practical applications of this "parity-checking" method.

- The American Standard Code for Information Interchange (ASCII) is a 7-bit code of alphabet letters and punctuation. Generally, computers use 8-bit (octet) packets. The last bit is used as a parity bit.
- Internet connections use protocols (such as TCP-IP), which use parity bits to check the integrity of data sent.
- An Internet Blog Serial Number, or ISBN, is a book's unique identification code. It also contains a checksum, calculated in a similar way, although it is a little more complex that the social security checksum.
- Barcodes also use similar types of checksums.

18 Note: This variation is inspired by an activity taken from the book Computer Science Unplugged (*classic.csunplugged.org*).

Review: Defining computer science

Summary	This lesson is a review of what computer science is all about using the poster crated during the previous sequences. With the help of documentary research, students create a timeline of the key moments in the history of information science.
Inquiry-based methods	Documentary study
Equipment	For each group • An A3 poster • Two Handouts, which the group chooses from: – Handouts 47 and 48 – Handouts 49 and 50 – Handouts 51 and 52 – Handouts 53 and 54 – Handouts 55 and 56 For each student: • Handout 57 For the class • A large white poster board for the final timeline (or Handout 57 printed on an A2 or A1 poster sheet). • Extra copies of Handouts 47, 49, 51, 53, 55 and 57. • The poster "Defining computer science", filled in throughout the previous lessons
Duration	Two hours, which can be divided into several lessons

Introductory question

The teacher displays on the board the poster that the class gradually completed throughout the lessons. There are several categories on this poster: "Languages", "Algorithms", "Machines", and "Information". To give the poster a historical context, and encourage documentary research, the teacher asks a question that appears simple: "In your opinion, when was computer science invented?" The students will probably say that the first computers appeared in the 20th century. But the teacher will go into more depth on this categorical reply. "Yes, computer science was created in the 20th century, but are the four fundamental components (name and point to them on the poster) as recent?"

Research (documentary study)

To begin, the students work in groups of four: each group studies one of the five collections of images (Handouts 47, 49, 51, 53, 55). Instructions are simple: *In your opinion, what does each image represent?* The students write down what they think.
Next, the teacher hands each group the text bundles that correspond to the images they have already received. The students study the texts, reading them quietly and independently. They

can use a dictionary for the terms they do not understand. They must then create a poster to present to the class. The instructions are as follows:

1. In the texts handed out, find the name of what each image represents
2. Stick the four images in the left column of the poster, and add a caption
3. Opposite each image, in the right-hand column, stick the corresponding text
4. Choose a title for the poster

Ideally, each poster could have a different background color (five colors for five themes), but this is not obligatory.

Scientific notes:

- The handouts explain in simple terms the major discoveries that have led to developments in computer science concepts, and key figures that played a determining role in the history of this science. Further details are provided in the scientific insights, and the teacher can use these notes to go into more depth on a personality that they see as representative.
- As in all simplified timelines, this one is partial (in both senses of the term!) It is often difficult for historians to clearly identify the true inventor of this machine. Often, history remembers the person that had the idea of combining several inventions, ideas, techniques, and concepts that other inventors of their era brought to light (for example, Joseph Jacquard is credited with inventing the punched card, but he used an idea that was originally Jean-Baptiste Falcon's, who in turn was inspired by the punched tape invented by Basile Bouchon, for whom he worked as his assistant). Even more often, it is due to the work of teams that together developed inventions, when history only retains one name (Morse Code was Samuel Morse's idea, but Alfred Vail's work; Turing's Bombe was devised by Rejewski in Poland and completed by Turing and Welchmann in the United Kingdom, etc.).

Group discussion

Each group presents its poster to the class. One by one, the students in the group present one of the four poster images (another option is for the teacher to successively project the images on the board so that they are more easily visible). This group discussion takes about an hour. If this lesson is in two parts, this may be a good point to stop.

Creating a timeline poster

The last step is to prepare a timeline poster together using all the documents provided. Each group collects its poster and the teacher hands each student a copy of Handout 57.
Using the information contained in their poster, the students must complete the blank timeline as best as possible.
Since each group can only partially complete the timeline, the teacher must gather together in a single, large-format timeline all the elements spotted by the class. They ask each group in turn to present a significant milestone, with its date and location. The students explain the words they have discovered. When an image illustrates one of these milestones specifically, it can be stuck on the giant timeline. A color code can also be introduced, to connect each item to the corresponding poster (if the posters were created on different colored backing paper).

The final timeline should look like this:

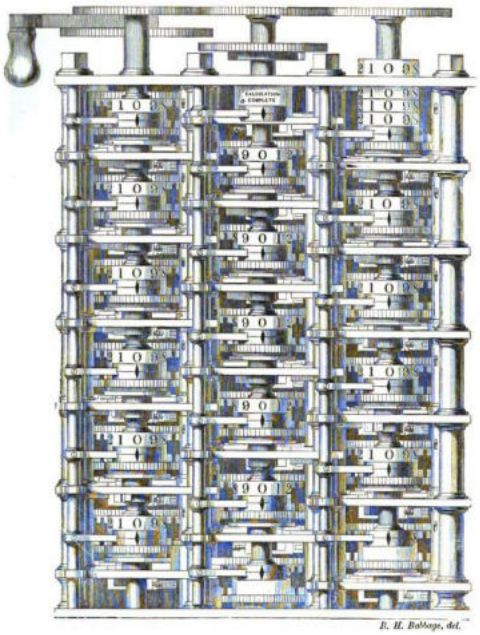
200 BCE	Antikythera mechanism
100 BCE	Julius Caesar encrypts his military messages
9th century	Al Khwarizmi explains the first algorithms
1450	Gutenberg popularizes movable type printing
1801	Jacquard invents a mechanism for the weaving loom
1821	Babbage creates the analytical engine
1838	Samuel Morse and Alfred Vail develop Morse Code
1843	Ada Lovelace writes the first computer program
1912	ElectricDog, the first robot
1930	Turing's theoretical model for a computing machine
1941	Zuse3
1951	Grace Hopper invents one of the first compilers
1961	Unimate, the first industrial robot
1967	IBM invents the floppy disk
1969	ARPANET, the ancestor of Internet, is launched
1985	Invention of the CD-ROM
1990	The CERN invents the World Wide Web
1996	Honda-P2, one of the first humanoid robots
1997	DeepBlue, a computer, defeats Kasparov in chess
1997	Sojourner robot sent to Mars
1999	Aibo, the entertainment robot
2001	Wikipedia is launched
2008	Google launches Google Flu Trends
2012	Facebook reaches a billion members

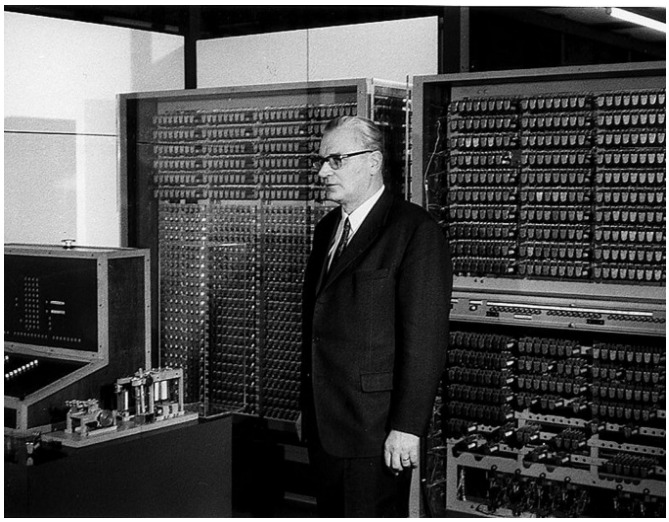
Conclusion

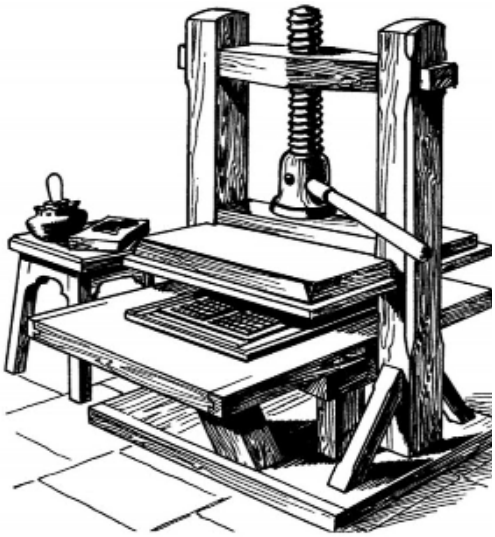
This timeline can help students reach several conclusions. Firstly, some inventions appeared over two thousand years ago: calculating machines and clockwork figures fascinated the kings' courts of Antiquity. While conceptualizations in mathematics and algorithms appeared in the Middle Ages, computer science problems (sums, programming, reproducibility and reconfigurability) began to be addressed in the last three centuries. As the students may have guessed, the first computers were in operation in the middle of the 20th century. Information sending and data sharing (what we now call Internet) closely followed, along with the improvement of automatons to become real robots, capable of interacting with their environment almost autonomously.

Students come up with a common conclusion, which they copy into their science notebooks. *Mathematics and automatons have existed since Antiquity. Technical advances from the 16th century onwards contributed to the invention of the first calculating machines. In the 20th century, electronics enabled the first computers, robots and the Internet to be created. While algorithms have been known and machines produced for a long time, computer science was born in the 1940s when we began to produce machines capable of performing all algorithms.*

HANDOUT 47
The History of Computer Science: Image collection 1



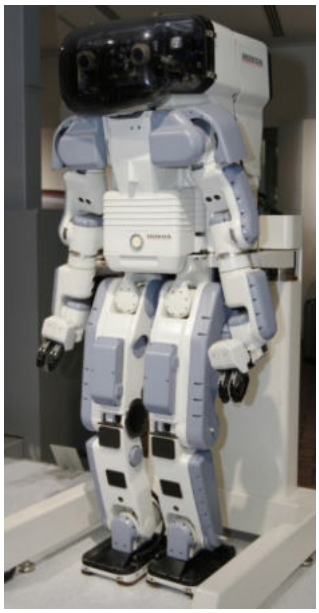


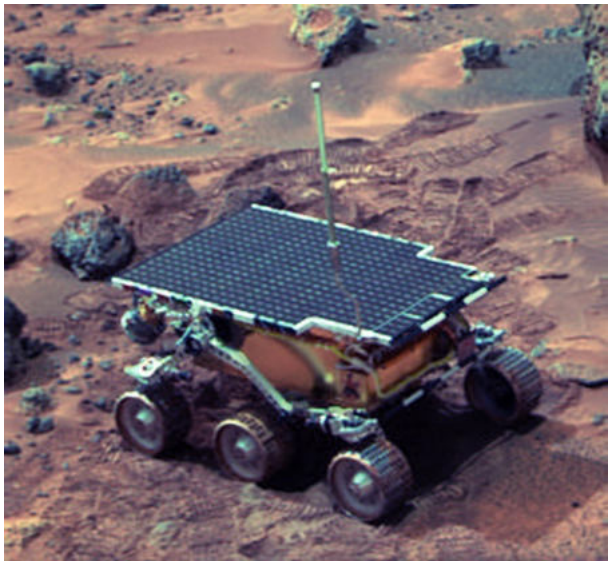


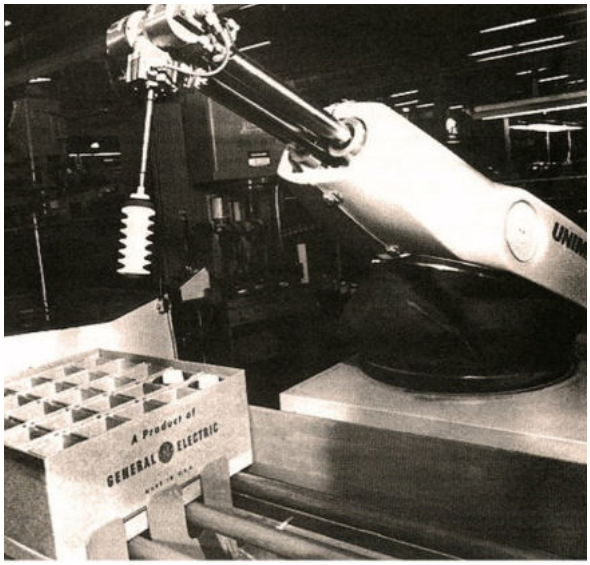
HANDOUT 48
The History of Computer Science: Text selection 1

The oldest known gear mechanism is the Antikythera mechanism, which is believed to have been invented by Archimedes in the 2nd century BC. It is a clock that can forecast the positions of the sun, the moon and the eclipses, by turning a simple hand crank. This masterpiece of mechanics had around thirty cogwheels.	In 1450, Johannes Gutenberg revolutionized printing. He improved the press, the ink, and most importantly, he made it movable, which means the press could be reconfigured as many times as needed. In the same way, in 1801, Joseph Marie Jacquard improved the weaving loom by introducing the punched card. This card contained the instructions for how to create a specific pattern. By using this card on two different looms, the same pattern can be reproduced; by changing the card, the same loom could produce another pattern.
In the 19th century, navigators found their way by looking at the positions of the stars and planets. They needed precise information. In 1821, Charles Babbage presented drawings for a difference machine that would speed up arithmetic. He never built it, but he designed another, more powerful machine: the analytical engine, which could read the calculations to be made on punched paper - this was the first computer concept.	In 1941, the German Konrad Zuse successfully invented the world's first computer, the Zuse3. But it was destroyed in 1944 by the Allied bombings. The race for power had begun, to rival the human brain. In 1997, the computer Deep Blue defeated the world chess champion Gary Kasparov.

HANDOUT 49
The History of Computer Science: Image collection 2







HANDOUT 50
The History of Computer Science: Text selection 2

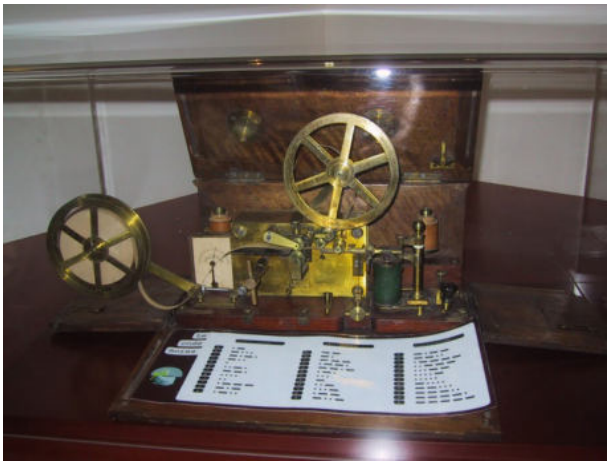
In 1961, Unimate became the first industrial robot. It assembled cars for General Motors. Several variations followed: robots that painted, welded, and fitted. Robots made industry easier, replacing humans for tasks that were too painstaking, or too dangerous.	Robots enabled scientists to explore bionics - the study of muscular movements. They attempted to reproduce human movement: in 1996, the humanoid robot, Honda's P2, could walk up stairs. The versions that followed learned to modify their trajectory while walking, recognize faces and keep their balance.
Robots let us explore where humans cannot survive. Space discovery took off again thanks to robots, which explored the moon and Mars - for example, the rover Sojourner that conquered Mars in 1997. Robots began to be used for entertainment in the new millennium. In 1999, Sony introduced the pet robot Aibo.	From Antiquity, the best clockmakers invented the first automatons: Hero of Alexandria, Leonardo da Vinci, Vaucanson, and others. But automatons do not know how to interact with their environment. Only in 1912 was the first robot invented: Electric Dog. Hammond and Miessner created a little trolley that moved towards sources of light it found.

HANDOUT 51
The History of Computer Science: Image collection 3

HANDOUT 52
The History of Computer Science: Text selection 3

Grace Murray Hopper (1906-1992) was an American mathematician. In 1951, she improved how dialogues function between humans and machines. The computer at that time only obeyed programs in "machine language" (called assembly language). Grace created one of the first compilers, A0-system, which translated a programming language close to English into assembly language.	Alan Turing (1912-1954) was a British mathematician and computer scientist. In the 1930s, he invented a theoretical model for a computing machine, with a calculator and memory where the program and data to be processed were stored. He was also famous for helping break the Enigma code during the Second World War. German military messages, encrypted with 159 trillion possibilities, could then be decrypted in 20 minutes.
Augusta Ada King, Countess of Lovelace (1815-1852), was a British mathematician. In 1833, Ada Lovelace met Charles Babbage and discovered his analytical engine. Where Babbage saw a reconfigurable calculator, Ada saw the potential for the first programmable computer. She wrote the first programs in 1843, for mathematics and also to compose music.	Al Khwarizmi (~780-~850) was a Persian scholar. In "The House of Wisdom" in Baghdad, founded by the Caliph Al-Ma'mun (813-833), he studied geometry and astronomy. In his work "*Kitâb al-jabr wa al-muqâbala*" (where the phonetic of "al-jabr" gave us the word for "algebra"), Al Khwarizmi systematizes decimal notation and algorithms which describe the methods for everyday calculations (addition, multiplication, etc.).

HANDOUT 53
The History of Computer Science: Image collection 4

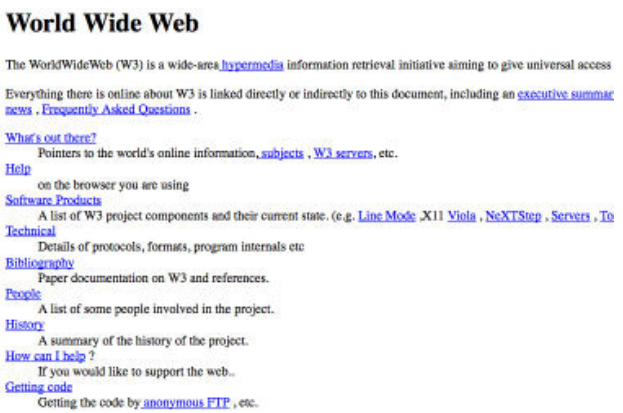


HANDOUT 54
The History of Computer Science: Text selection 4

In the 1st century BCE, Julius Caesar coded his military messages to ensure they remained confidential. To do so, he substituted letters of the alphabet for other letters. The messenger just had to make sure the encrypted message reached its addressee. This method was so simple that it was used even during World War 1 by the Russian Army.	The use of electricity and magnetism to send information across vast distances or between the components of a computer can be applied to writing information to a memory. After 250 years of loyal service, the punched card was replaced by magnetic tape. In 1957, IBM replaced magnetic tape with a magnetic disk, protected from dust by a plastic case. The floppy disk was born.
The compact disc is not a magnetic information carrier. It is a plastic disc, read by a laser beam. Initially, the CD was used to replace vinyl records in the music industry. In 1985, the same principle was used to store any form of digital data. This was the beginning of the CD-ROM. It rapidly led to the disappearance of the floppy disk, before being replaced by DVDs, based on the same optical technology.	Sending information across vast distances without a messenger was the greatest challenge of the 18th century. Chappe's telegraph was complex: it used 92 different signals and relay stations had to be very close to one another. The electrical telegraph simplified that problem. In 1838, Samuel Morse and Alfred Vail invented a code for the telegraph. Their ingenious creation used short codes for the most frequent letters.

HANDOUT 55
The History of Computer Science: Image collection 5

WIKIPEDIA
The Free Encyclopedia

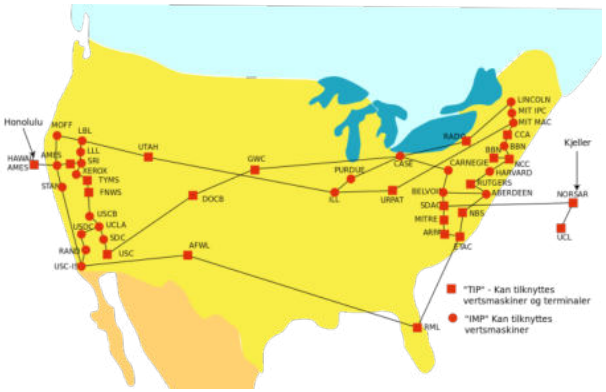


World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summar news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , To
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.



HANDOUT 56
The History of Computer Science: Text selection 5

In 2008, Google analyzed the searches of certain key words entered by its users (over 40,000 per second) to detect the appearance of a flu epidemic and follow its course. The keywords "'flu", "fever" and "cough" are used most often when we are sick. But this posed ethical issues in terms of data privacy. Meanwhile, more and more people joined social media sites. In 2012, eight years after its launch, Facebook had over a billion members!	To share data, it can be useful to connect several computers together. This is how networks were born. The very first computer network was ARPANET. In 1969 the US Army connected some of its computers in the Universities of Los Angeles, Stanford, Santa Barbara and Salt Lake City. This created the basis for the Internet.
At the CERN, in Geneva, Tim Berners-Lee developed a protocol that allowed him to publish pages of text on a network and more importantly, to navigate from one page to another using a link. These links formed the structure of the pages to form a world wide web. On November 13th, 1990, the very first page on the World Wide Web contained 6 links. Since then, the number of websites accessible on the Internet network doubles every six months.	Jimmy Wales founded Wikipedia in January 2001. It is a multilingual, open digital encyclopedia. It can be consulted freely on the Internet, evolves and grows bigger with the contributions of a vast, volunteer community. It is one of the ten most consulted websites in the world!

HANDOUT 57
The History of Computer Science: A Timeline

200 BCE	
100 BCE	
9th century	
1450	
1801	
1821	
1838	
1843	
1912	
1930	
1941	
1951	
1961	
1967	
1969	
1985	
1990	
1996	
1997	
1997	
1999	
2001	
2008	
2012	

The 1, 2, 3 Code! website

Support for the classroom project

The “1,2,3...Code!” project has its own website, designed to be used as a support in the classroom.

The website provides a space for classes to exchange ideas and help each other via a selection of collaborative tools. Each class can record their work’s progress in a blog, upload their multimedia productions and consult and comment on productions by other classes.
A forum allows teachers to ask their colleagues and the project scientific and teaching consultants questions, thereby finding help and support in putting this project into practice.
Lastly, an interactive map allows users to make contact (twinning, visits, penfriends, etc.) with other classes conducting the project, either nearby or further away.

Teachers' area

The teachers’ area is a space for teachers to get information about the project and sign up. This registration provides access to a comprehensive range of scientific and teaching documentation, as well as the collaborative functions mentioned above:

- The teaching module is fully available online.
- The pedagogical and scientific backgrounds are also available online, in more detailed and updated versions than the print.
- The documents used in class can be downloaded (and projected or printed in better quality than photocopies).
- Useful files to conduct the plugged lessons (sprites, prefilled programs or corrected ones).

Project partners

La main à la pâte foundation
www.fondation-lamap.org

The *La main à la pâte* foundation aims to help improve the quality of science and technology education in primary school and middle school, where the curriculum common core provides equal opportunities for all. In France and abroad, the Foundation's activities are oriented towards support and professional development for science teachers. The Foundation is the general coordinator for the "1,2,3...Code!" project, designs pedagogical tools for teachers and disseminates the project in classrooms. It hosts and manages the website dedicated to the project and implements a teachers training plan.

Institut national de recherche en informatique et en automatique **(French national institute for computer science and applied mathematics) – www.inria.fr**

The French national institute for computer science and applied mathematics (INRIA) aims to build a network of skills and talents across French and international research facilities. In addition to research, INRIA's role is to provide scientific mediation. This means helping to educate enlightened citizens, fostering understanding among all of this new dimension of existence created by digital technologies, sparking curiosity through innovative applications, encouraging the participation and involvement of everyone in creating a digital world, and helping to combat the digital divide. INRIA played a major role in the deployment of ISN (computer science and digital technologies) education and scientific culture activities on these topics. INRIA provided scientific expertise in the creation of the teaching guide, "1,2,3...Code!".

France IOI - **www.france-ioi.org**

The association France-IOI works to help as many young people as possible discover and advance in programming and algorithmics. It develops tools and content for this purpose and provides them free of charge to the public. It co-organizes the Castor Informatique competition with INRIA and the Ecole Nationale Supérieure de Cachan, and provides a platform for programming and algorithmics education through practice and at the learner's own pace. France IOI provided scientific and teaching expertise in the creation of the "1,2,3...Code!" project and created online activities for students.

Class'Code - **http://classcode.fr**

Class'Code is an ambitious project initiated by INRIA, bringing together a number of actors in the fields of computer science, education and science culture. Its aim is to design and deploy at a large scale, blended learning (distance and classroom) for educators and all those who want to introduce young people to computational thinking. Class'Code's support to the *La main à la pâte* foundation enabled free, large-scale distribution of the teaching guide "1,2,3...Code!" and permitted teacher and teacher trainer courses to be provided.

Microsoft - **www.microsoft.com**

Microsoft is a technology company founded in 1975 and with operations in 107 countries. In France, Microsoft employs 1,700 people and works with more than 11,000 partner companies. Microsoft conducts various initiatives to help young generations acquire a good understanding of programming, in particular through the Imagine Cup and DigiGirlz. The release of Minecraft Education in June 2016 added to the company's approach to teaching students to code. Lastly, Microsoft contributes to the "1,2,3...Code!" project by supporting the free distribution of the teaching guide and by hosting teacher training courses provided by the *La main à la pâte* foundation.

Google - www.google.com

Google is a global technology leader that aims to improve access for all to information. Google's innovations in research and online advertising have made it one of the top websites and one of the most easily recognized brands in the world. Google is committed to supporting improvements in access to education and enabling all groups to discover computer science. That is why Google supports the *La main à la pâte* foundation for the free distribution of the teaching guide "1,2,3...Code!", as part of the inclusion of computer science in school curricula. This support will also enable the *La main à la pâte* foundationto establish nationwide communities that include teachers, university academics and institutions to promote professional development in computer science among peers.

Educaland - www.educaland.com

Educaland, a brand belonging to the Jeulin company, is a publisher of teaching materials for primary education. In partnership wtih the *La main à la pâte* foundation and Le Pommier publishers, Educaland produced an educational toolkit tailored to the "1,2,3...Code!" project.

Mobsya - www.mobsya.org

Mobsya is an association that aims to promote science and technology among young people. It was created as a result of the collaboration between research institutes who worked on the creation of the Thymio robot. It is in charge of producing, distributing and marketing Thymio and helps showcase the product, either for individuals or for schools, mainly by working together on the development of educational equipment.

Editions Le Pommier - www.editions-lepommier.fr

Le Pommier aims to make the language of science and philosophy accessible. Their books help young and old to understand the world, while marvelling at all it has to offer. Le Pommier contributed to this project by publishing the teaching guide and making it freely available online.

International Science, Technology and Innovation, Centre for South-South Cooperation under the auspices of UNESCO (ISTIC) - www.istic-unesco.org

The International Science, Technology and Innovation Centre for South-South Cooperation under the auspices of UNESCO (ISTIC) was established as a follow up of the Doha Plan of Action which has been adopted by the head of States and Government of the Group of 77 and China. ISTIC is a UNESCO Category 2 Science Centre and acts as an international platform for South-South cooperation in science, technology and innovation and makes use of the network of the G77 plus China and the Organization of the Islamic Conference (OIC). The overall goal of ISTIC is to increase the capacity for management of science, technology and innovation throughout developing countries.

From preparing children for the jobs of the future, helping them understand the things and networks around them — so that they are not passively subjected to them but able to act on them — to making them aware of civic challenges and encouraging cooperation and developing their creativity, everyone should be taught to use computers, from as early an age as possible.
The "1,2,3...Code!" project created by the *La main à la pâte* foundation, INRIA (the French Institute for Research in Computer Science and Automation) and France IOI aims to introduce students and teachers to computer science, from kindergarten to end of 6th grade.
It offers plugged activities (requiring a computer, tablet or automated device) that introduce programming basics and unplugged activities (computer science without a computer) that allow the teacher to address fundamental concepts in computer sciences (algorithms, languages, how information is shown, etc.).

A turnkey resource

This teaching manual includes:

- **3 progressions for the class (Level 1, 2 and 3)**
  - A multidisciplinary approach combining inquiry-based and project-based learning;
  - Ready-to-use lessons, tested in the classroom, divided into themed sequences for each cycle;
  - Handouts to be photocopied;
- Teaching and scientific insights to guide the teacher in carrying out the project;
  A bibliography for the teacher and for the students.

The authors

Claire Calmet is a teacher and teacher educator, in charge of the links with employers and the research at the Foundation *La main à la pâte.*
Mathieu Hirtzig is a webmaster and scientific communicator at the Foundation *La main à la pâte.*
David Wilgenbus is a teacher educator, in charge of the production of pedagogical resources at the Foundation *La main à la pâte.* He is the coordinator of the "1, 2, 3... Code!" project.

FONDATION La main à la pâte
POUR L'EDUCATION A LA SCIENCE

Launched in 1996 by Georges Charpak, Nobel Prize in Physics, with the support of the Académie des sciences and the Ministry of Education, *La main à la pâte* aims at improving the quality of science and technology teaching in primary and middle schools: http://www.fondation-lamap.org

With the support of:

KEMENTERIAN TENAGA, SAINS, TEKNOLOGI, ALAM SEKITAR DAN PERUBAHAN IKLIM
MINISTRY OF ENERGY, SCIENCE, TECHNOLOGI, ENVIRONMENT & CLIMATE CHANGE

FONDATION La main à la pâte
POUR L'EDUCATION A LA SCIENCE

Éditions Le Pommier

ISBN 978-967-13199-0-1
9 789671 319925