Roamer Activity

RoboCup Computational Thinking

Introduces students to object-oriented ideas and computational thinking by analysing a RoboCup game.

Subjects	Age	Roamer Expertise	Student Grouping	Lesson Time	Availability
Computing	9 11 10				

Description

Object-Oriented Programming (OOP) is a set of popular programming languages like Python, Java, C++, LiveCode and many others. Perhaps more importantly, it's an approach to problem-solving and thinking. This is a computing science unplugged[1] task.

In this task, you'll use the PowerPoint presentation to introduce students to the ideas used in OOP. They then use these ideas to analyse and develop strategies for their RoboCup team. This involves them thinking about the positions played by players (goalkeeper, defender, midfielder and strikers) and what gameplays they might want to develop. They'll do this by using the student materials to create UML[2] Class Diagrams.

Class	Player
Attributes	Position: Goalkeeper Name: John
Operations	Patrol Save Penalties Goalkicks Pass

Figure 1 UML Class Diagram for a Goalkeeper

Objectives

Students can:

1. Learn about object-oriented ideas.
2. Analyse a problem.
3. Gain experience of how diagrams help programmers.
4. Learn about top-down and bottom-up programming.

[1] Computer Science Unplugged involves tasks which do not use computers, robots or other electronic gadgets. Playing Turtle is an example.

[2] A UML (Unified Modelling Language) diagrams give computer scientists and programmers a way to analyse problems. They help them to understand the problem, gather information, identify connections, explain and document projects. Programmers use them as a tool to support computational thinking.

Sustainable Learning Objectives

Students can:

1. Improve their computational thinking skills (Cognitive: thinking – types of thinking).
2. Sort out their role in a team (Emotional: relating – cooperation).
3. Work out a team strategy (Social: working – teamwork}.

Related Activities

Roamer RoboCup Activities.

	Activity	**Summary**
1.	Dance Festival[1]	Students new to Roamer take a crash course by programming their robot to dance.
2.	**Forces Unplugged**	Helps children understand the idea of force through their personal, everyday experiences.
3.	**Newton's Laws**	Students create and run a set of Roamer experiments to find out more about forces.
4.	Kick About	Students gain experience of playing RoboCup with a free-for-all game.
5.	Designing Robot Footballers	Students use data from their Kick About game to design, make and test Roamer Kickboards and Team Colours.
6.	Practice Makes Perfect	Students perfect their robot footballer designs and skills.
7.	**RoboCup Computational Thinking**	Introduces students to object-oriented ideas and computational thinking by analysing a RoboCup game.
8.	Developing Match Play Consoles	Students create their gameplay procedures and their RoboCup Consoles (a mobile App for match play).
9.	Final Practice	Students play an attack and defence game controlling Roamer with their Roamer RoboCup Match Consoles.

[1] Optional for those who've never used Roamer before.

Lesson Plan

Don't treat this lesson plan prescriptively: it's for guidance only. We expect you to change it to meet your needs, teaching preferences and above all in response to the creativity and curiosity of your students. Treat our timings as a rough guide.

Preparation

Set Up PowerPoint

1. Download PowerPoint.
2. Set up interactive whiteboard or similar.

Copy Student Worksheets

1. Two per team.
2. For Extension Activity
    a. Scissors
    b. Paper glue.
    c. One sheet of A4 or similar paper.

Activity

PowerPoint Presentation (10 minutes)

1. Get students discussing the ideas.
2. Guide the debate as needed.

Explain Task (5 minutes)

1. Use last slide in PowerPoint to explain task.
2. Get students to state (in their words) what they're going to learn.
    a. Learn how to analyse a problem like a computer scientist.
    b. Learn about object oriented ideas.
3. How will they know when they've succeeded?

Create UML Player Class Diagrams (20 minutes)

1. Students decide what positions each will play in the team.
2. Discuss the gameplays: what do they mean?
3. Write information from the attribute and gameplay list into the Class Templates.
    a. Can you think of other plays or attributes?
    b. Add them to the Class templates.

Assessment

Student Presentation (10 minutes)

1. Each presents their work.
    a. They need to clearly explain what they've done and why.
2. Other students ask the team questions.
3. Make evaluation notes and ask questions to clarify student understanding.

Evaluation

For your reports and the e-Robot research note and enter the following data into the Lesson Evaluation form.

1. What percentage of students thought they met their success criteria?
2. How well did the task engage the students?
3. Note anything the students did or said that of interest.
4. Think about your next steps (if any) to help the students progress.

Teacher's Notes

Subject Comments

Our aim here isn't to teach with extreme rigour the ideas of Object-Oriented Programming (OOP). We do use language which has specific meanings, but we don't need to force this on students. We want to introduce them gently to this way of thinking and lay the foundations for future studies.

In 2006, Jeannette Wing, then President's Professor of Computer Science at Carnegie Mellon University, delivered a seminal paper to 'The Association of Computer Machinery' (Wing, 2006).

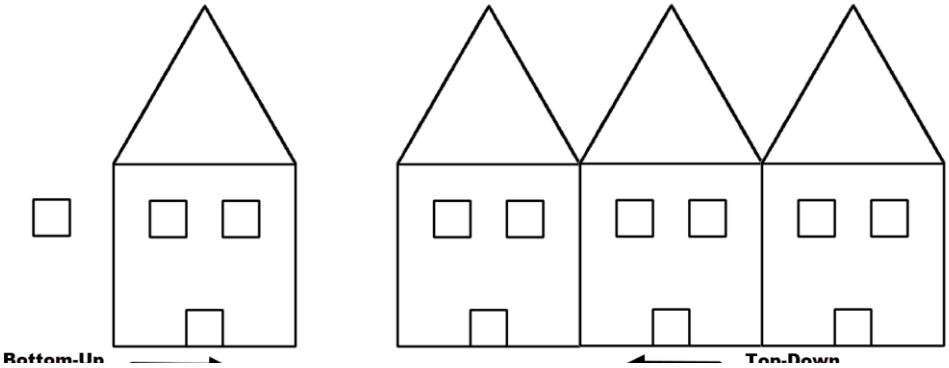
> *"It [Computational Thinking] represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. (Wing, 2006)"*

The idea of computational thinking rapidly spread around the world. What does it mean? The table below shows a simplified version of what's involved.

Computational Thinking	
Skill	Know-How
Decomposition	Breaking complex problems into basic parts.
Abstraction	Getting rid of unnecessary detail.
Generalisation	Finding likenesses and patterns in the problems.
Algorithm	Identifying step-by-step solutions to the problem.

What you'll notice is none of these features involves programming: they're all about sorting out what you need to code. More than anything that's what OOP is a way of sorting out the problem.

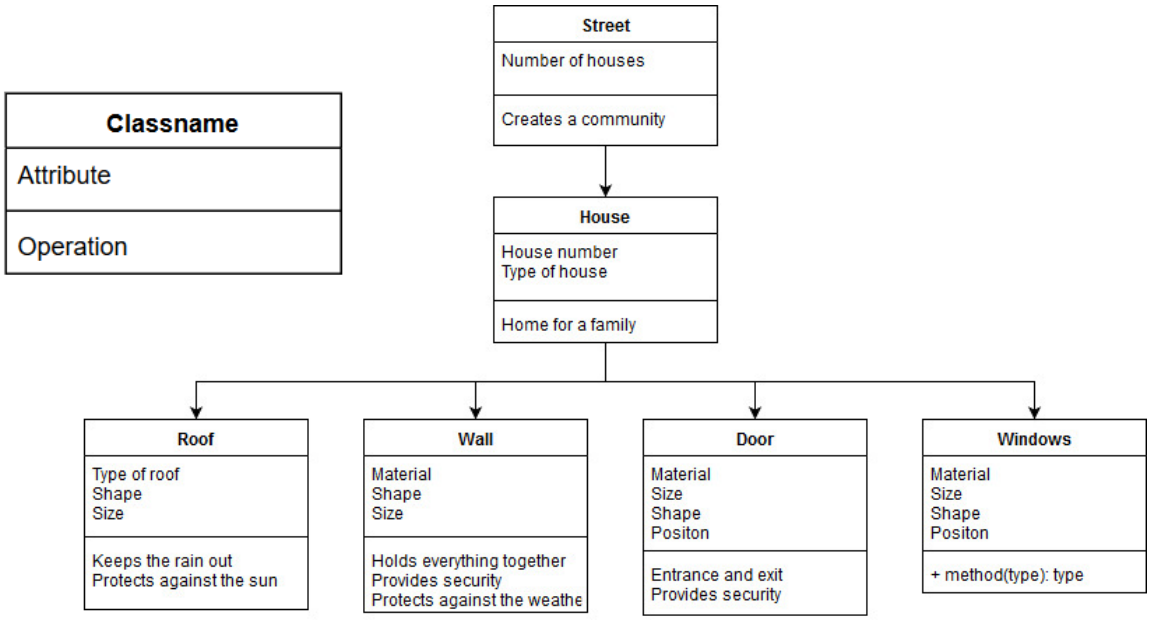
Programming paradigms promote a way of approaching a complex coding task (Harvey, 1997, pp. 162-169). Roamer uses a Logo-like programming language which we classify as a procedural language. In Logo's heyday, we'd talk about bottom-up and top-down problem-solving. You could, for example, code a procedure to make your robot to first draw a square then a house and a street. This is an example of bottom-up programming where the student reuses previous experience (the square procedure) to solve a new problem. The top-down approach breaks the big problem into small easy-to-solve problems. In both cases, we search for a pattern which we can program as a procedure and use repeatedly. To draw the street, we reuse three: the house, square and triangle. We use the scale command to change the size of the square, and the house uses the square and triangle procedures as commands.



Procedural Computing Top-Down and Bottom-Up Strategies.

OOP is another way of thinking about programs. In this method, you break the problem down into a set of independent objects. So we can think of street, house, roof, walls, door and windows. Each object has features called 'attributes' and purpose called 'operations'.

Computer scientists help themselves think about problems and solutions using diagrams called 'Unified Modelling Language' (UML).



UML Class Diagram: **You can create at different levels and show the relationships between them.**

If you Google UML diagrams and check out the images: you'll find many examples which software engineers use when thinking about how to create coding solutions.

The example shows a UML Class diagram for a street - not the Roamer drawing of a street. This effort helps you do the first three parts of the computational thinking task: decomposition, abstraction and generalisation. It's easy to see what object have in common. Using UML is over-doing-it for the street problem, however, this task challenges students to apply these ideas to developing a Game Console for RoboCup.

We've scaffolded the challenge by providing a list of attributes and operations for a team. Working as a team the students use the student materials to create the Class UML diagram. The extension activity challenges the students to create a UML case diagram by adding a team level.

We recommend you use the PowerPoint Presentation to explain Object-Oriented ideas to you students: it focuses on presenting information and asking them questions.

Prior Knowledge

It's useful if students:

1. Know how to write basic Roamer programs.
2. Preferably know about procedures.

Classroom Tips

You'll find the key to success lies in how well you can present and use the PowerPoint slideshow. We suggest you make sure this works well for you and please change it to suit your preferences.

Training Links

This task highlights a key constructionist teaching skill: effective questioning. In the videos, teachers explain how they ask questions and ensure all students think about and discuss the ideas.

How to Ask Good Questions includes videos for:

- Primary School Teachers
- Secondary School Teachers
- Posing questions and handling feedback

The Science of Learning

People use to say computers can't think. Ironic, since the first steps towards computing began with Aristotle recording people's thought patterns. From this starting point, we can trace precomputing thought through a roll-call of the Western intellectual high achievers. Thomas Hobbes, Gottfried Leibniz, George Boole, Gottlob Frege, Bertram Russell and Alfred North Whitehead and Kurt Gödel led to Alan Turing crucial work on computing. This isn't what people normally think of as computational thinking. Yet, any coding effort will plunge programmers into the thought patterns and ideas recognised by these people (Catlin, Education Robots and Computational Thinking, 2013).

Jeannette Wing didn't 'invent' the idea of using coding as a means of improving student thinking and problem-solving skills. Seymour Papert did this in 1967 when he invented Logo, a computer language specifically for learning. In those days people created computer languages for distinct purposes, like FORTRAN for science and engineering and COBOL for business. He wanted Logo to allow four-year-old children to start programming, but also challenge much older students. At first, Papert focused on mathematics. Students trying to, for example, write code to solve maths puzzles had to dig deeper than getting the right answer. They had to understand the mathematical nature of the problem before they could code a solution. However, armed with such powerful tools, students soon started to explore ideas in all subjects.

Computer scientists aren't the only ones claiming the power of their subjects' meta-thinking. George Polya, one of Papert's influences, inspired the USA's National Council for Teachers of Mathematics (NCTM) to do the same for maths. They put mathematical thinking at the heart of their 2000 Standards publication (Stewart, 1990). Design Technologists (forerunners of the Maker Movement) proposed how designers thought and worked could help student's problem-solve and study all subjects. The truth is all subjects have thinking levels and ways of organising their work efforts. The disturbing fact is these ways of working are emergent behaviours that grow from experience (Catlin & Woollard, 2014).

In their review of education research, John Bransford and state the importance of learning with understanding. They go on to stress the importance of a "conceptual framework". It's part of computational thinking and as student expertise grows it becomes natural and intuitive rather than a rigid set of rules they must follow. If you insist students follow the rules step-by-step you often get a result that lacks creativity (Bransford, Brown, & Cocking, 2000, pp. 16-17).

Papert explains this tension when he recalls research done by MIT's Sherry Turkle. Both Jeff and Kevin created computer graphics depicting space exploration scenes. Jeff followed a computational thinking approach: Kevin more of an artistic strategy. Kevin doesn't have a plan: he tries something and if he doesn't like it he changes it. Despite this Turkle reports, "Kevin not only succeeded in creating a space scene but like Jeff, he learnt a great deal about

computer programming and mathematics (Turkle, 2005)." You'll notice the adopted an approach natural to them: top down (Jeff) and bottom up (Kevin).

Papert picks up the story, "Kevin is lucky to be in an environment where he is allowed to work in his own style (Papert, 1993). In many schools, he would be under pressure to do things 'properly'… (Papert, 1993, p. 148)." Both Papert and Turkle claim bullying children into thinking in a particular way will adversely affect their intellectual growth. Papert goes further, "Rather than pushing children to think like adults, we might do better to remember they are great learners and be more like them (Papert, 1993, p. 155).

You can introduce primary-school children to schemas like computational thinking and object-oriented ideas, but you don't need to make a fuss over it. Knowledge and organisation need to grow together and for this age group, our main aim is to excite students and give them experiences they can build on more formally in the future.

References and Useful Links

Bransford, J. D., Brown, A. L., & Cocking, R. R. (Eds.). (2000). *How People Learn: Brain, Mind, Experience and School* (Vol. Expanded Edition). Washington DC: National Academy Press.

Catlin, D. (2013, November 24). *Education Robots and Computational Thinking*. Retrieved May 18, 2019, from GO Magazine: http://go.roamer-educational-robot.com/2013/11/24/educational-robots-and-computational-thinking/

Catlin, D., & Woollard, J. (2014). Educational Robots and Computational Thinking. *Teaching Robotics Teaching with Robotics (TRTWR) and Robotics in Education (RIE) 2014 Conference, Padova, Italy Teaching Robotics Teaching with Robotics (TRTWR) - Robotics in Education (RIE) 2014 Conference, Padova, Italy, Italy. 8 pp.* Padova, Italy. Retrieved October 29, 2018, from http://research.roamer-educational-robot.com/2014/07/22/educational-robots-and-computational-thinking/

Harvey, B. (1997). *Computer Science Logo Stylle* (Vol. Vol 3 Beyond Programming). Cambridge: MIT.

Papert, S. (1993). *The Children's Machine.* New York: Basic Books.

Stewart, I. (1990). *How to Solve It* (New Edition ed.). London and New York: Penguin Books.

Turkle, S. (2005). *The Second Self: Computers and the Human Spirit.* (20th Anniversary Edition ed.). Cambridge, MA.

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM, 49*(3), 33-35.

Resources

Teacher Materials

Object Oriented PowerPoint Presentation

Student Materials

Team Sheet

Other Materials

For the extension challenge.

1. Scissors
2. Paper glue.
3. One sheet of A4 or similar paper.

Curriculum Links

National Curriculum England: Computing Programmes of Study

Key Stage 2 – Ages 7 to 11:

Goals: Design and write programs that accomplish specific goals, including controlling or simulating physical systems; solve problems by decomposing them into smaller parts.

Other Countries

Use the online form to send curriculum link information for other countries and we will add the links to the site.

Lesson Evaluation

Use the online form to complete a lesson evaluation.

Evaluation serves three purposes:

1. It gives you a record of student's progress and understanding.
2. It helps you plan personalised strategies for a student's progress.
3. It provides data for the e-Robot research project.

You use the online form to complete a lesson evaluation.