**Video outline – Maze game**

**Module 6-7** : Games

**Length**: max 5 min

**Voice-over** : George?

Sub-titles: English

**Objective**

* Learn about the use of programming language
* Creating a sequence of instructions or ‘algorithms’
* Practice the process of writing code, testing and debugging

**Bring**

* **Chalk**
* **Cards**
* **Paper + pencils**

**Target Audience**

* Lower and upper secondary learners

**Elements in the video**

* Programming language
* Algorithm
* Testing
* Debugging

|  |  |  |
| --- | --- | --- |
| **Animation/Imagery** | **Voice Over** | **Notes/shoot location** |
| George explaining | Through unplugged Coding activities, students can develop computational thinking skills through concepts such as algorithms, testing and debugging without using computers or other devices. So unplugged activities help students to understand the basics of coding without even touching a digital device! The only thing you need is creativity and imagination! Let’s get started! | Multimedia studio |
| * Voice over * We see the maze game in the scratch interface | In this video we will discover the unplugged activity ‘Maze’. In the Maze game in the Scratch interface the aim is to move a sprite from the start of a maze to the end without touching the walls.  In this unplugged activity, it is the learner who moves through the Maze without touching the walls. | Scratch interface  A picture containing chart  Description automatically generated |
| * Voice over * We see an outdoor space with a gid (8x8) * We see pen and paper * text with supplies needed floats in | **Preparation**  What do you need for this activity?   * An 8x8 grid marked on the ground. This could be:   + Drawn with chalk on concrete big enough for people to stand in each square   + On flooring with a built-in grid pattern (such as square tiles)   + Masking tape (painter’s tape)/ropes on a suitable surface * Cards or obstacles such as stones, branches,… to mark the maze * Something to write down the code (this could be cards, chalk, pen and paper,…) | Outside - School – 8x8 grid (without the learners) |
| * Voice over * Shot of group 1 drawing the grid * Shot of group 2 (who are looking to the other side, covering their ears?) * Text ‘step 1’ flies in | **How does it work?**  Step 1. The facilitator of the game makes the maze by putting some obstacles in the grid. Then, divide the learners in 2 groups. Every group choses **one robot**. **The others are developers**. The Robots can move to the starting point of the maze. | Outside – school |
| * Voice over * Text ‘step 2’ flies in * Text ‘algorithm’ * Shot of learners writing code | Step 2: Then the developers write down an **algorithm** for the robot to move through the Maze. An algorithm means a sequence of instructions. They can use following commands: Forward, turn left and turn right. | Outside - school |
| * Voice over * Shot of the used programming language * Text ‘step 3’ flies in * Text ‘programming language’ flies in | Step 3: The development team decides what symbols they will use to write the instructions. This means they decide on the “**programming language**”. Examples of programming languages:   * 1. Arrows (↑, ↰, ↱)   2. Letters such as ‘F’ for ‘forward’, ‘L’ for ‘turn left’.   3. Other…   In our example we will use arrows printed on cards. If you do not have cards you can write down the code with chalk on the ground or pen and paper. | Outside – school – 8x8 grid |
| * Voice over * Shot of a developer reading the code * Shot of a robot performing the code * Text ‘step 4’ * Text ‘**running the programme**’ * Text ‘**test’** | Step 4: One team start reading the commands to their robot. In coding language this means ‘**running the programme**’.  A key point in this activity is that the instructions are all written before they are **tested**. We don't allow anyone to give additional instructions to the robot; they must follow exactly what is written. This is what happens when programming: you write instructions for a program, and when you run the program, they are all executed without the programmer intervening. A programmer needs to picture in their mind what would happen when they are writing the instructions; during testing they will find out if what they pictured was correct! | Outside – school – 8x8 grid |
| * Voice over * Text ‘step 5’ * Shot of ‘error’ and debugging * The word ‘**debug**’ flies in | Step 5: If the robot makes a mistake, the developers say `Error Error` and the other team takes turn. Meanwhile team 1 gets time to **debug** their code. Debugging means find the mistake and improve the code. It’s normal for computer programs to need debugging - a bug is simply when it doesn’t do what was intended. If the developer team is ready with debugging and has a new algorithm ready, they can try again.  Whoever gets to the other side of the maze first wins the game. | Outside – school – 8x8 grid |
| * Voice over * Text ‘step 6’ * Text ‘**multiple levels’** * Shot of picking up ball/stones * Shot jumping over ball/stones | Step 6: After a while you can add **multiple levels** to the game. You can do this by making the maze more difficult by adding more objects (ball, stones, leaves). Picking up some of the objects leads to extra points, or make you return to start. You can also expand the commands with for instance ‘jump over’. | Outside – school – 8x8 grid |
| * George explaining * Words in bold ` fly in’ | **Conclusions**  Creating a sequence of instructions or ‘**algorithms**’ that a computer or robot can follow is an important skill in programming. Like all skills, programming is something you learn through practice, by making mistakes, and learning from them. In this unplugged activity we also learned about **decomposition**. This is splitting up a problem (going from the one side of the grid to the other side) into smaller steps that are achievable to solve. Another important skill that programmers need is to **collaborate** with others, especially when they are working out what the program should do. They also need to be persistent when finding and fixing **bugs**. Bugs happen all the time in programming, so being able to identify where the bug occurs and problem solving how to fix it is incredibly important. It doesn’t matter how experienced you are at programming, there will always be bugs that need to be found, and fixed. That’s why being prepared, and able, to debug is important. | Multimedia studio |
| Interview/ reflection with learners | * What did you learn? * How did you like this game? * What did you learn about coding today? | school |