Activity Sheet:

Title: Create a Maze Game with the BBC Micro:bit

Objective:

- Understand the concept of perception in AI by using the Micro:bit's accelerometer to control a game character in a maze.
- Develop coding skills to create an interactive game.

Materials:

- BBC Micro:bit with USB cable.
- Computer with MakeCode coding environment installed.

Instructions:

Step 1: Introduction

- Connect the Micro:bit to your computer via USB.
- Open the MakeCode coding environment in a web browser.

Step 2: Create a New Project

- Start a new MakeCode project for your maze game.

Step 3: Understand Perception

- Perception in AI involves sensing and understanding the environment. In this activity, you'll use the Micro:bit's accelerometer to detect tilting motions, allowing the game character to move within the maze.

Step 4: Design and customize Your Maze

In this step, you will design the maze using the grid provided in the MakeCode coding environment. You can customize the maze by adding walls and open pathways to create a challenging puzzle. The maze should have a clear start point and a destination, which is the end point of the game. Keep It Challenging: Consider the difficulty level of your maze. The path from the start to the end should present a challenge to the player. The player should navigate through the maze by tilting the Micro:bit while avoiding walls to reach the destination.

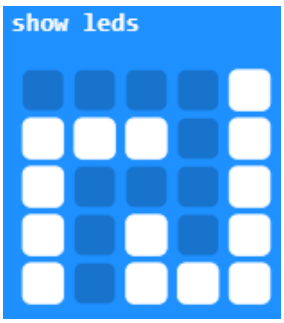

Figure 1 LED grid using the block "show led" in MakeCode

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Create the maze layout using the grid provided in MakeCode. Use the block shapes to represent walls, open pathways, the start (x = 0, y = 0), and the end (x = 1, y = 4) of the maze. Customize the layout to match the provided maze or create your own maze design. You can also create more levels for the player.

Note

The complete set of coordinates x,y for the grid that micro:bit offers are presented in the table below.

Table 1 X, Y coordinates for micro:bit grid

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)

The player's location on the Micro:bit screen will be indicated by a flashing red LED. Solid red LEDs will symbolize walls, while unlit LEDs will signify the maze pathways.

Coordinates are employed to manipulate the Micro:bit LEDs effectively! The x coordinates range from 0 on the left to 4 on the right, while the y coordinates range from 0 at the top to 4 at the bottom. Consequently, the LED at the upper left is denoted as x=0, y=0, and correspondingly, the LED at the bottom right is represented as x=4, y=4.

Step 5: Code the Game

Use the following blocks to program the game's behavior:

Figure 2 Beginning of program in MakeCode

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

First, we need to create a few variables. Recall that variables function as containers that store information. In this case, two variables are necessary to monitor the player's location. One is designated to record the player's x position, while the other is dedicated to tracking the player's y position.

Additionally, we require a variable to monitor the maze level, allowing for the possibility of multiple levels. Another variable is necessary to track the game's status, indicating whether it is active or if it has concluded.

The initial values are set to start at level 1, and gameOn is initialized as True. This is because, upon powering on the Micro:bit, the intention is to commence the game immediately. While the starting point for the player's location can be chosen arbitrarily, it needs to be recalled later when configuring the maze level to ensure the player does not begin inside a wall. For this example, the player is initiated at x=0 and y=0.

Figure 3 First forever loop

Now that the initial variables are in place, let's ensure our player is displayed on the Micro:bit screen!

To achieve a distinctive blinking effect for the player, we'll employ the 'plot x y' block alternated with the 'pause' block within a forever loop. The intention is for the player to continuously blink on and off. When maze walls are introduced, the Micro:bit will overwrite the player each time it draws the walls. By incorporating a pause block here, we ensure that the player won't be immediately re-plotted, resulting in the desired blinking effect.

The utilization of the playerX and playerY variables created earlier is crucial. Why? If numerical values were directly inputted here, it would limit the flexibility to make the player move. The use of variables enables us to modify the values of playerX and playerY, allowing the forever loop to plot the player's new location.

It's essential to note that the pause block operates in milliseconds (e.g., 200 ms = 0.2 seconds), and the blinking speed can be customized by adjusting the duration of the pause.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Figure 4 Second forever loop

Now we need to setup the player's movements (left, rights, up, and down). We will use the two integrated buttons and the logo swipe function.

We'll set the logo up gesture to move up, the logo down gesture to move down, the button A to move left, and the button B to move right.

To accomplish this, we utilize if statements. These statements assess whether a condition is true; if it is, any blocks inside the if block are executed. When we embed an if statement within a forever loop, we continually check if the condition is true.

For player movement, we modify the playerX or playerY variables. It's crucial to remember that decreasing or increasing playerX causes left or right movement, respectively, while decreasing or increasing playerY results in upward or downward movement, respectively. Given that we consistently plot the player's location using these variables, any changes automatically reflect the player's new position.

It's worth noting that a brief 300ms pause is added after each button press. This prevents the Micro:bit from moving the player across multiple spaces rapidly with each button press, as the code runs quickly without the pause.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

n: firstly, displaying the maze

Figure 5 Third forever loop

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

walls on the LED screen; secondly, continually checking if the player collides with a wall (indicating game over); and thirdly, perpetually assessing if the player successfully completes the maze level.

A forever loop is employed. Within this loop, an 'if' statement is used to verify if the level variable equals 1. Consequently, this code segment will only execute when the level variable equals 1. If we want to add more levels, then we should make sure that this variable changes accordingly.

Inside the 'if' statement, the maze walls are displayed using the 'show leds' block. LEDs are illuminated to represent walls, while unlit LEDs denote the maze paths. Caution must be exercised to ensure that the player's starting position, set earlier at x=0, y=0, does not coincide with a maze wall.

The subsequent task involves checking if the player collides with a wall. This is achieved through additional 'if' statements, verifying if the playerX and playerY variables align with the coordinates of a wall in the 5x5 LED grid.

Lastly, the code checks if the player successfully navigates through the maze. In this example, the maze's end is at x=1, y=4. If these conditions are met, a successful melody plays, the player's position is reset to the beginning of the maze, and a smiley face appears on the Micro:bit. If we have added additional levels, then we also need to change the variable level by 1.

Figure 6 Fourth forever loop

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

In case of a game over, we need to implement an action triggered by the 'gameOn' variable indicating a collision with a wall.

Within a forever loop, an 'if' statement is used to assess the value of the 'gameOn' variable. If it equals 'false,' the game over code is executed.

In this instance, a sad melody plays in the background, the 'level' is reset, the player LED is unlit, a sad face is displayed, and the game starts from the beginning.

Step 6: Test Your Game

- Test your game by guiding the character through the maze. Does everything work correctly?

Step 7: Play and Share

- Share your maze game with others. Load it onto your Micro:bit and challenge your friends to complete the maze.

This project allows students to experience the concept of perception in AI by using the Micro:bit's accelerometer to control a character within a maze. Students can design their own mazes, create different levels of difficulty, and share their games with peers for added fun and learning.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.