day75 micro:bit maze
Friday 3/29/24

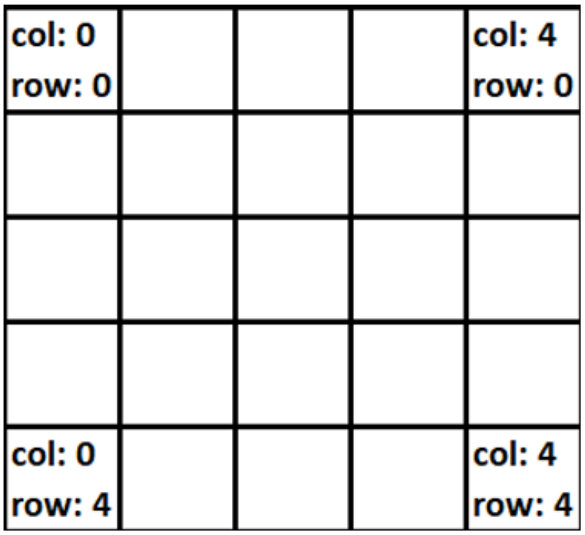
Today we are going to make a maze game.

To make this program work, you'll need to know the rows and columns of the lights on the micro:bit. Here is the layout to the right.

Open the starter code file "maze starter file 2024.py" attached to this assignment on the Google Classroom. Put your name into a comment at the top. Save the file with your name and "maze" in the title in your folder on the computer.

Flash the code to make sure it works. It should display several solid lights and one that is blinking (the bottom left light).

The lights in your image are listed in the lights list with the column first, then the row



Today we're making a maze game, the blinking light is the player, and the solid lights are the walls that you have to get around.

Step 1:

The player starts at column 0, row 4 (the bottom left light). This is tracked using the c and r variables.

Add code to move up if you press the A button. Moving up would mean subtracting one from r.

Use "if button_a.was_pressed():" to check for a press. **Pressing the A button should change the r variable.** Remember, appropriate values for r will be between 0 and 4 inclusive, so make sure you don't make the dot try to draw out of bounds, it will crash.

Test your code by flashing it, then pressing the A button several times. Make sure the blinking dot goes up the left side of the screen and then stops at the top left. For now, don't worry that the dot goes through the solid light at column 0, row 2, we're fixing that in step 2.

Step 2:

Add a check that prevents you from moving up through a wall of the maze. It should just not move if you try to move up through a maze wall. The maze walls are the LEDs that are set to power 9. You can "see" if an LED is lit by checking maze.get_pixel(c, r - 1). If the value is 0, you can move there. If the value is 9, you cannot.

Test your code, make sure your dot goes from (0, 4) to (0, 3) and then no further up because the light at (0, 2) is lit up and thus is considered a "wall".

(continued on next page)

Step 3:

Add code to move right if you press the B button. Also add code to prevent the user from moving right through a maze wall or off the screen to the right. Moving right means increasing the c variable. Make sure that you do not make c larger than 4 and that you do not "move" through walls.
Get all of this working before moving on.

Step 4:

Add code to stop the game and put up a happy face forever if you make it to location
r = 0, c = 4 (the top right.)

Extensions:

Choose one of the following extensions. **Everyone must do one** (or a different one of your choice that you discuss with me before implementing):

1. If you press the A button down for more than 1 second the dot moves down instead of up. Of course, only if the spot below the dot is legal. If you press the B button for more than a second the dot moves to the left, if legal. I used a while button_a.is_pressed(): loop inside the "if button_a.was_pressed():" if statement and the running_time() function to time how long the button was pressed.

2. Make the grid fill with 8 randomly placed wall parts instead of using a fixed grid (so the game is different every time. Make sure the spot at column 4 row 0 is not a wall dot, or you couldn't win. Use a for loop and a single maze.set_pixel(column, row) column to add the random wall parts.

3. Instead of using button A and button B use gestures and the accelerometer to move the dot. Don't move it through walls or off the edges. Make the "shake" gesture restart the game.

4. Use the pin0.is_touched() and pin1.is_touched() methods in addition to button a and button b to move the dot.

5. Add a randomly placed dot that is lit at level 5 that acts as a portal: if you get to that spot, you are automatically teleported to the space right next to the end of the maze.

6. Something else? If you've got a great idea, talk to me, I probably will encourage you to try it.

Call me over to check you off. Also, turn your code in to the Google Classroom when you are done.