PyPiGame Assignment

In this assignment you will complete the maze game PyPiGame. You will use various programming constructs in the Python programming language.

NOTE: You will **ONLY** be working on the **Maze class** in the **maze.py** file

Please ensure that you have the pygame library installed on your system. Visit https://www.pygame.org/wiki/GettingStarted or https://cs.hofstra.edu/docs/pages/guides/InstallingPygame.html for more instructions.

If you cannot find a particular question look for the following lines in the code:

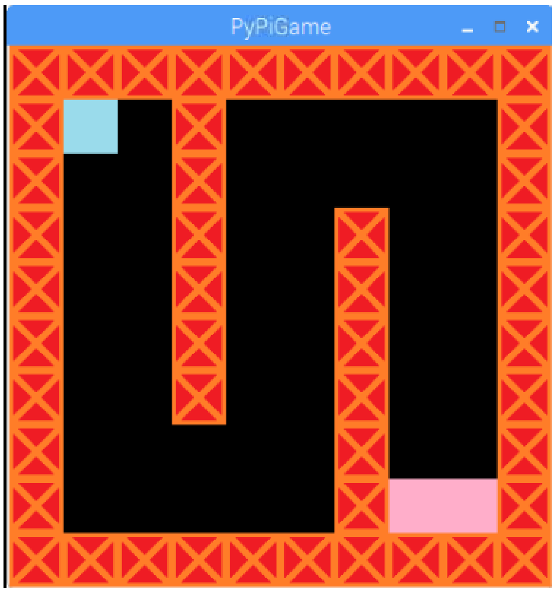
```
###

        # TO DO: QUESTION …

###
```

Try running the assignment. ("python GameApp.py" on the Pi / "python GameAppDesktop.py" on a desktop computer) Do not worry if you get an error that it will not run, this is supposed to happen, we will sort that out in a moment.

Question 1: Variables

1.1 In the **Maze class**, create a variable named **maze_choice** and set it to 1. This variable will be used to set the map which we will play on. Do this in the __init__(self): method. We will use this variable again later.

Try running the assignment now, see it works. You will see a map that sort of looks like this:



Now that we have sorted that issue out, we can move on to displaying a new map which you will create.

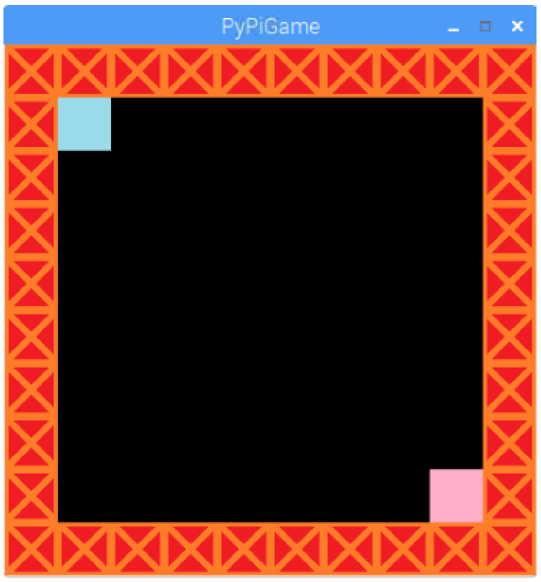
The mazes here is actually a 10x10 grid space stored in a single 1 dimensional array consisting of 100 integers.

1.2 In the same file, scroll down until you can see the **maze_Four(self)** method (**def maze_Four(self):** …) Define a new map using a single dimension array (m = […]). This map should be a clear map with borders and the goal state being in the bottom right corner.
A value of 0 indicates that there will be an open space in that region, a value of 1 indicates a wall will be in that region and a value of 2 indicates that the goal state will be in that region.

NOTE: You can use (copy and paste) any of the mazes defined before this and modify them accordingly to get the desired outcome.

Now set **maze_choice** to 4 and run the program.

The desired map should look like this:



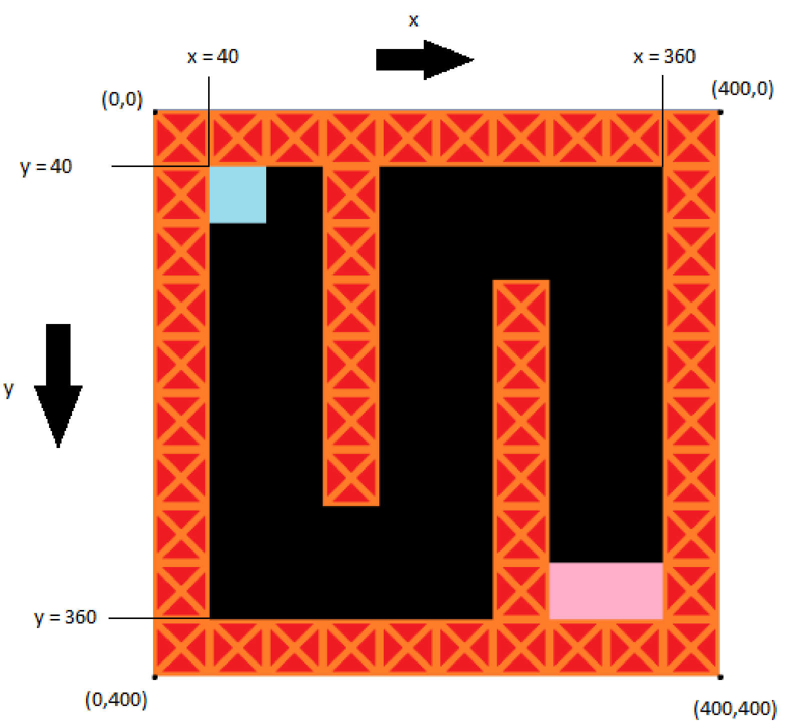
<u>Question 2: Conditional Statements and Boolean expressions</u>

If you move the blue character around using the joystick, you will see that the character will disappear behind the walls instead of 'bouncing' off of them. We will fix that here.

2.1 In the **Maze class**, scroll right to the bottom. You will see the **check_boundaries()** method (**check_boundaries(self, x, y):** …) Check that the x and y values for the player are inside the border using if statements in whichever way you want to (i.e that the player is inside of the border)

Information:

The x-axis runs from left to right and the y-axis runs from top to bottom. Below is an example of the 2D world and where the walls are located. The player will always start at position (40,40). The player is also 40 units in length and width.



Question 3: Loops

This question will demonstrate the concept and importance of loops.

Say for instance we wanted to count the number of walls on the map which we are on (including the borders), we would need something to loop through the entire array while we are counting the walls on the map (the number of 1's in the array). Thankfully we have 2 options for this: the for loop and the while loop.

3.1 In the **Maze class**, scroll right to the bottom. You will see the **count_walls_for()** method (**count_walls_for(self):** …) In this method, use a for loop to count the number of walls on the map and display it to the console.

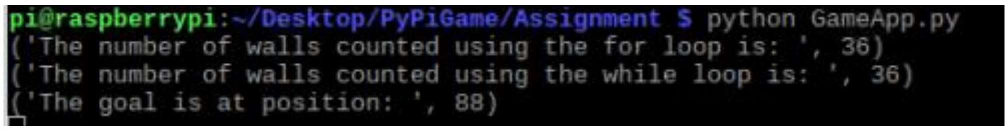
You can also rewrite this for loop you have just coded as a while loop.

3.2 In the **Maze class**, scroll right to the bottom. You will see the **count_walls_while()** method (**count_walls_while(self):** …) In this method, use a while loop to count the number of walls on the map and display it to the console.

Now say for instance we want to know where exactly the goal state is in the map array. If we were to use a for loop and run through the whole array to find it, we will just be wasting time as the goal state can be anywhere and we would have to run through the whole array each and every time. Instead we can just use a while loop to loop through the array, until we find the goal state.

3.3 In the **Maze class**, scroll right to the bottom. You will see the **find_goal()** method (**find_goal (self):** …) In this method, use a while loop to find where the goal is in the map and display it to the console. (*Where the value of map_Arr[…] is equal to 2.*) You do not need to worry about there being multiple goal states, just find the first one.

The output for the question should look similar to this:



```
pi@raspberrypi:~/Desktop/PyPiGame/Assignment $ python GameApp.py
('The number of walls counted using the for loop is: ', 36)
('The number of walls counted using the while loop is: ', 36)
('The goal is at position: ', 88)
```

NOTE: If you reach the goal state and want to play again, you will need to close and re-run the game.