159.172 Computational Thinking

Assignment 2: Theseus and the Maze

This assignment is worth 15% of your final mark. It will be marked out of a total of 20 marks. The due date for this assignment is Sunday 20 October. You are expected to work on this assignment *individually* and all work that you hand in is expected to be your *own* work.

In this assignment you will develop an animated application that uses depth-first and breadth-first searches as the basis of the implementation.

Go to Stream and download the files

`mexplorer.py`
`mazeclass.py`

Set up a new project in Eclipse and add these files to it.

`mexplorer.py` is a program that implements the beginnings of a game about finding a way through a maze. When you run this program, you will see a screen, 1000 pixels wide by 500 pixels high, displaying a maze. In our game, the maze is a two-dimensional structure with walls and paths. A character, called Theseus, is placed somewhere in the maze and has to follow the paths to achieve the goals of his missions. The paths can branch, which means that Theseus has choices where to go next. As for his missions, he has to achieve the following:

1. Walk through the maze in a systematic way such that eventually he has visited all locations that are reachable from his initial position.
2. Find his way out of the maze.
3. Find the quickest way out of the maze.

To achieve the first mission, Theseus traverses the maze in a depth-first manner from his current location, and after he has done so, returns to his current position. He marks the locations that he has visited with a special token, and the ones he has revisited with another special token.

For the second mission, Theseus performs a depth-first search in his mind without actually marking locations – he has a good memory of the maze – to find his way to the exit. When doing so, he records the path taken on a stack, which he then uses to build the path to actually walk to the exit.

For the third mission, Theseus traverses the maze in his mind again. To find the shortest path to the exit, he uses a breadth-first search, where he records the cells that he visits in a queue. Each cell added to the queue is in fact a stack with the visited cell at the top and the rest of the stack showing the path to that cell. Once Theseus finds the exit, he can use the stack associated with the exit to build the shortest path to the exit.

mazeclass.py uses a two-dimensional grid (list of lists) to represent the maze. Each list item is a cell with attribute **status**. The values for the status attribute have the following meaning:

- 0
  The location is part of a path and has not been visited by Theseus before.
- 1
  The location is part of a wall and therefore cannot be visited.
- 2
  The location is part of the perimeter of the maze and therefore cannot be visited.
- 3
  The location is part of a path and has been visited by Theseus.
- 4
  The location is part of a path and has been revisited by Theseus (i.e., visited for a second time).
- 5
  The location is part of a path and has been examined by Theseus in his mind.
- 6
  The location is the exit square.

The constructor for a maze uses a two dimensional integer array of status values to build the maze, and places Theseus at a random location in the maze. To test your program using different mazes you can insert different two dimensional integer arrays as values for the **mazegrid** variable in **mexplorer.py**. The code has a number of methods that are used to display the maze, which you can use as is, or alter to your liking.

mexplorer.py includes headers for the three functions that you are supposed to implement:

```
def depthfirsttraversal(i, j):
    ...
```

This function traverses the maze in a depth-first manner and changes the entries in the maze grid appropriately. It also takes care of displaying the maze and providing a time delay after each change. An easy way to do this is by including the following code:

```
the_maze.display_maze(screen)
pygame.time.delay(200)
pygame.display.flip()
```

```
def depthfirstsearch(i, j):
    ...
```

This function uses depth-first search to look for the exit in the maze without initially changing the display of the maze. It changes the entries for the locations in the maze that have been examined from 0 to 5. It also keeps track of the path, which is then displayed step by step at the end of the procedure.

```
def breadthfirstsearch(i, j):
    ...
```

This function uses breadth-first search to look for the shortest path to the exit in the maze without initially changing the display of the maze. It also keeps track of the path, which is then displayed step by step at the end of the procedure.

You can invoke these functions by pressing the 'd, 's' or 'q' keys, respectively. The main game loop takes care of recognizing key presses and allowing to quit the program. The maze is reset to its initial state before each invocation of one of the three functions.

Submission

Submit the assignment in Stream as a single zipped file containing:

1. Your completed code, contained in the file **`mexplorer.py`**. Include **`mazeclass.py`** if you have altered it.
2. A word document containing annotated screen shots demonstrating the behaviour of your program.
3. If you have attempted any extension to the project, submit your extended code as separate code modules, and include a separate word document demonstrating the extended program behavior. This will not (necessarily) contribute to your final mark, but imaginative solutions are welcomed!

Marking Scheme:

- Implementation of each of the three functions - 5 marks each
- Word document containing annotated screen shots- 5 marks
- **Total – 20 marks**

Late submission:

Late assignments will be penalised 10% for each weekday past the deadline, for up to five (5) days; after this no marks will be gained. In special circumstances an extension may be obtained from the paper co-ordinator, and these penalties will not apply. Workload will not be considered a special circumstance – you must budget your time.