Pac-Man in Maze World

COL106 Assignment 1

August 5, 2024

1 Background

Pac-Man is trapped in a haunted maze and needs your help to move! The maze is filled with ghosts, and Pac-Man needs to find his way to his favourite destination while avoiding them. Your task is to create a navigation system for Pac-Man using only **stacks**.

2 Modelling

Let us assume that Pac-Man is moving in a **2D rectangular grid**.

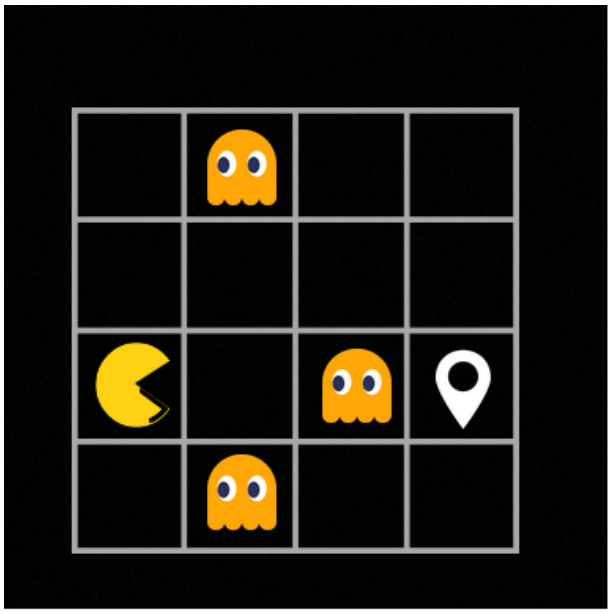

Figure 1: Here's an example of the kind of grids we'd be dealing with. The white symbol represents the destination Pac-Man has to reach

We will represent the vacant cells of the grid (even if they are occupied by Pac-Man) by a 0, and the cells with ghosts by a 1. Here's the grid corresponding to the maze 2 to demonstrate:

0	1	0	0
0	0	0	0
0	0	1	0
0	1	0	0

Note that the grid cells will be numbered from the top right. Here's an example numbering to demonstrate this:

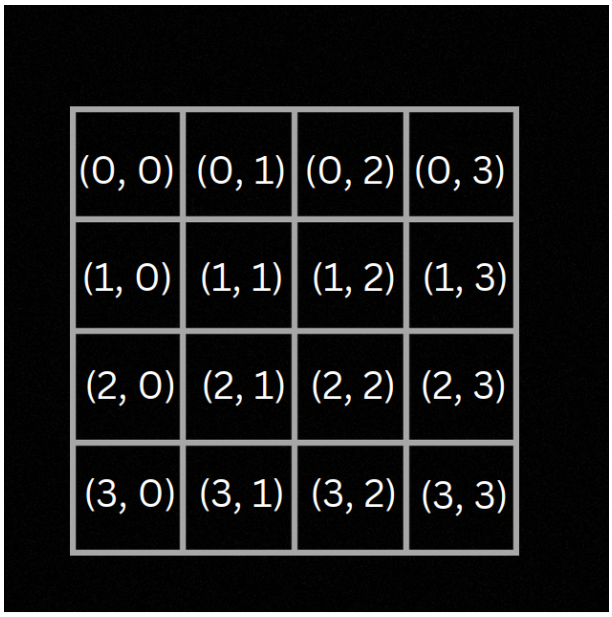

Figure 2: Cells of the grid numbered with their coordinate pairs

We will assume that Pac-Man cannot step outside this grid, and that he cannot step into the cells numbered 1 as well. For example, the path shown below is legal:

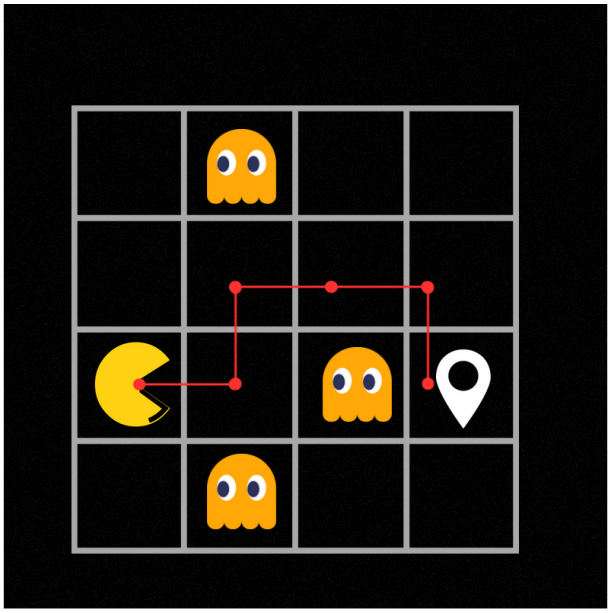

Figure 3: A valid path

And the path shown here is not (as it goes through a cell with a ghost):

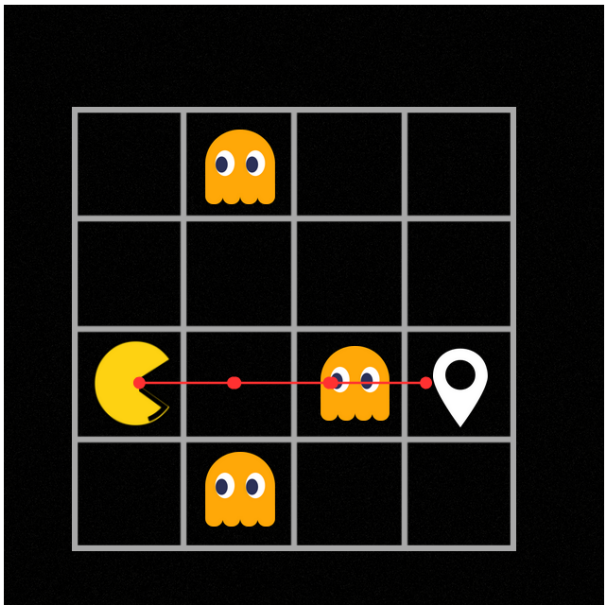

Figure 4: An invalid path

3 Requirements

Here's a description of the classes and functionalities you are expected to implement:

3.1 Maze

Refer to `maze.py` in the starter code. The maze is a 2D array of integers, and is implemented using a list of lists (nested 2D list) in code. The constructor takes the dimensions of this maze as input and initialises with all 0s (so that initially, all cells are vacant). You need to implement the `Maze` class, with these functions

1. `add_ghost(x, y)`: Adds a ghost at the specified coordinates. Formally, after executing this function, the cell at `(x, y)` has value 1 (irrespective of its initial value).
2. `remove_ghost(x, y)`: Removes all ghosts at the specified coordinates. Formally, after executing this function, the cell at `(x, y)` has value 0 (irrespective of its initial value).
3. `is_ghost(x, y)`: Checks if a cell contains a ghost. Return `True` if so, otherwise return `False`
4. `print_grid()`: Print the current maze layout. You are expected to print the values of cells in a row separated by a space, and the rows are to be separated by a newline.

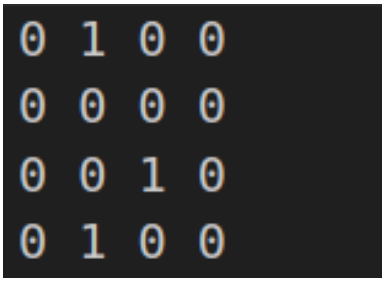

Figure 5: Sample output for print_grid() corresponding to 2

NOTE: It is guaranteed that `(x, y)` will be within the bounds of the grid for `add_ghost(x, y)`, `remove_ghost(x, y)` and `is_ghost(x, y)`. Further, you are NOT allowed to modify the other functions and the attributes present in the class, though you are free to add other methods and attributes.

3.2 Navigator

Refer to `navigator.py` in the starter code. You need to implement the `Navigator` class, with these functions

1. `find_path(start, end)`: Find the path from `start` (a tuple of the form `(s_x, s_y)` where `s_x` is the x-coordinate of Pac-Man's starting point, and `s_y` is the y-coordinate of Pac-Man's starting point) to `end` (a tuple of the form `(e_x, e_y)` where `e_x` is the x-coordinate of Pac-Man's goal, and `e_y` is the y-coordinate of Pac-Man's goal).
   The output is expected to be the path in the form of a list of coordinate pairs, with the first one being `start` and the last one being `end`.

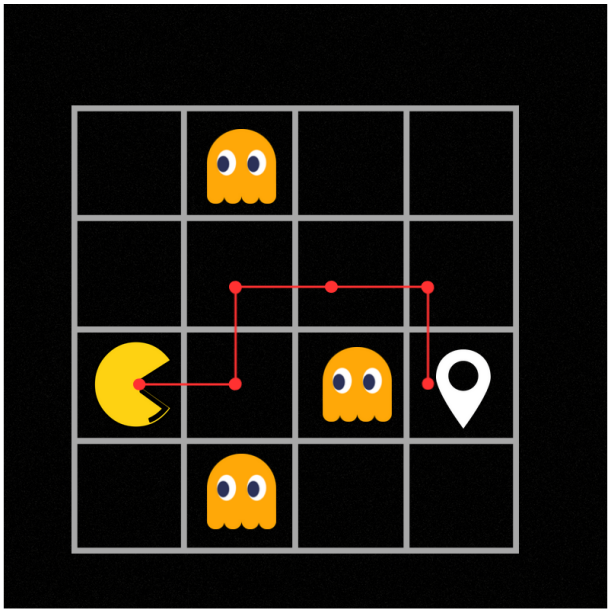

Figure 6: For example, in this maze, a correct output, corresponding to the path in red, is

$$[(2,0),(2,1),(1,1),(1,2),(1,3),(2,3)]$$

3.2.1 Clarifications

1. Note that the `Navigator` class has an attribute of the type `Maze` class, initialised at the time of construction. This is the maze you are supposed to run `find_path` on.
2. A coordinate pair should occur *atmost once* in a path. So, $[(2,0),(2,1),(2,2)]$ is a valid path, but $[(2,0),(2,1),(1,1),(2,1),(2,2)]$ is not.
3. If a path is not found, or if the start and end cells are not vacant, you are expected to raise `PathNotFoundException` (refer `exception.py`)
4. If there are multiple correct paths from `start` to `end`, you can output **any one** of them.
5. Yet again, you are NOT allowed to modify the other functions and the attributes present in the class, though you are free to add other methods and attributes.
6. The time complexity of this function should be $O(n * m)$ where n is the number of rows, and m is the number of columns in the maze.

3.3 Stack

Refer to `stack.py` in the starter code. You can modify the constructor and add any functions of your choice to simplify your code. You are allowed (and expected) to use this class in your `Navigator` class.

4 Points to Ponder

1. To implement the `Stack` class, you can look into growable arrays
2. Think about how you can model maze traversal using a stack. When you want to move to a new cell, you may consider pushing this cell on the stack so that you can store the path taken so far. When you reach a dead end and want to go back, you can try to pop from the stack.
3. How would you keep track of what cells you have visited so far? In particular, you don't want to keep going round in circles. With each cell, maintain a boolean variable which tells you whether you have been to this cell before or not. If you been to a particular cell before, you don't advance to it again (i.e., each cell should be pushed on the stack only once).

5 Submission

1. You are expected to complete this assignment in the next 2 weeks during your lab hours
2. You need to submit a zip file <your-entry-number>.zip (for example, `2021CS10081.zip`) which contains your code. Do NOT modify the names of the existing files.
3. Your submission upon extraction, should yield a folder named <your-entry-number>.
4. This folder should contain 5 files, namely
   (a) `maze.py`
   (b) `stack.py`
   (c) `navigator.py`
   (d) `exception.py` (You are NOT supposed to modify this file)
   (e) `main.py` (You can use this file for testing and debugging your code, changing this file would NOT make a difference to your submission. You can use `python3 main.py` to run and debug your code locally)