Overview

This program is a pac-man like movement simulation. There are two types of beings, Pac-man and the ghosts. The beings are bounded by the size of the map and the boundaries within it. Pac-man's goal is to traverse the map and eat all the cherries. The goal of the ghost(s) is to eat pac-man.

Pac-man and the ghosts move about the map in either Von Neumann or Moore neighbourhood scheme. There are three different simulation scenarios with different modes in play depending on the beings interacting with the features of the map.

Modes:

- Ghosts chase Pacman
- Ghosts run away from Pacman
- Pacman chase cherries
- Pacman chase ghosts (duration of invincibility after eating a cherry)

Scenarios:

- No Obstacles map: There are no boundaries limiting movement
- Maze map: A maze like map where the beings have to navigate around to find shortest path to their target
- Maze map with portals: A similar maze map but with two sets of portals that connect opposite ends of the map

User Guide

On the command line, type python pacSimulation.py to run the program.
You will be prompted to input a number of things, ghost beings, cherries, timesteps, choice of map and movement scheme.
After the input the program will begin to display the simulation as a plot every time interval.

Traceability Matrix

Feature	Code	Test
Options for map choice, number of beings/objects and movement scheme	pacmanSimulation.py def main() while choice not in [1,2,3,4]: choice = int(input('Please select 1-4'))	Input numbers not within range Input number within range
Read map returns map size and obstacle locations	pacmanSimulation.py def read_map()	def test() Read csv files with numerous line of coordinates
Plot maze scatter	pacmanSimulation.py def plot_maze_scatter(mazeList)	Read csv file to plot coordinates of maze
Read Ghosts returns instances of ghost characters	pacmanSimulation.py def read_ghosts()	
Plot ghost scatter	pacmanSimulation.py def plot_ghost_scatter(ghosts, num)	Tested num with (1-4) to display the number of ghosts during simulation
Spawn location of cherries	pacmanSimulation.py def cherry_spawn_location(col, row, list, num)	col and row parameters assigned to the map size maximum columns and rows List passed in is an obstacle list to prevent cherries spawning on an obstacle Tested num with 1-4 to display the number of cherries created and plotted
Plot cherry scatter	pacmanSimulation.py def plot_cherry_scatter(cherry)	Only current cherry in the cherry list is passed in so 1 cherry is displayed and interacted with at a time
Create a graph data structure	pacmanSimulation.py	Within test()

with VonNeumann/Moore neighbourhood scheme	create_graph_von(col, row, list) create_graph_moore(col, row, list)	Passed in testMaze, testObstacle
Movement Von Neumann scheme	characters.py def bfsMove() pacmanSimulation.py theGraph = create_graph_von()	
Movement Moore scheme	character.py def bfsmove() pacmanSimulation.py theGraph = create_graph_moore()	
Breadth FIrst Search (finding the shortest path in a graph data structure)	pacmanSimulation.py def bfs(graph, current_vertex, target_value)	Within test() Passed in a dictionary with strings for both key and value called some_hazardous_graph Passed in a dictionary with integer tuples for both key and value

Showcase

Introduction

The base code was taken from Prac Test 3. Basic use of class to create characters, their attributes and their methods.
The Matplotlib scatter method displays the maze, the cherries and the characters on the map.
Very similar to Prac Test 3, the beacon is replaced with the cherry and the characters have their respective targets to chase and run away from.

Features:

Map with walls/obstacles: Beings and cherry spawn within the boundaries of the map on an obstacle free cell. Beings also obey boundaries and cannot move onto a cell that is a boundary

Pathfinding algorithm was taken from the well known advanced graph searching algorithm, Breadth FIrst Search(BFS). The idea behind this algorithm is to traverse a graph by visiting all its neighbours first before moving onto the next vertex. By doing so you end up with a path with the minimum number of edges which gives you the shortest path to travel in the graph. The BFS function implemented takes three arguments, the graph, the start vertex and the end vertex. It returns the path (the vertices to travel) from start to end vertex.

Methodology

The setup of the overall program is very simple, there are no parameter sweeps.
After running the program the user will be prompted to enter a number of things all of which stay within a valid range.

Users can use other custom maps as long as the map is in a csv file filled with the map size on the first line and the list of coordinates on the subsequent lines. The csv files follow the usual pattern of (integer),(integer)
Note there is no whitespace after the comma

The program outputs a plot that can be saved at any timestep however as the images display one after the other every time interval, the user would need to either increase the time interval or rewrite the line of code plt.pause() to plt.show() to allow the current state of the simulation to be shown indefinitely.

Results

No obstacles map
Pac man and ghosts are able to move freely within the boundaries of the map.
Pacman and Ghost gang can move in both Von Neumann and Moore neighbourhood schemes.
Ghosts chase pacman while he is not invincible and run away from pacman while he is invincible

Maze (wall type obstacles):

Using the bfs function, both pacman and the ghosts are able to navigate around their boundaries to get to their target. As the function creates a path for them to follow, it prevents the beings from being trapped in a section the map.

Maze with portals:
Both pac-man and ghosts use the portals if it presents a shorter path to get to their target.

Conclusion and Future Work

Everything up until the interaction with obstacles was fairly straightforward. It wasn't difficult to modify the movement of characters in Moore neighbourhood scheme whereas Von Neumann neighbourhood scheme was initially difficult in an obstacle free map. It was difficult to work out the algorithm for characters to move towards an object that was diagonal of their position.

However once a pathfinding algorithm was implemented(Breadth First Search) the algorithm works out the singular path in either Von Neumann or Moore movement scheme. So by following the generated path the characters are able to display movement in either Von Neumann or Moore.

There can be a change from top down perspective to side view perspective (like the 2D Mario games). Characters move sideways on their platform, or fly (depending on the mode).

Another pathfinding algorithm, possibly Depth First Search, can be implemented. As it doesn't look for the shortest path it just finds a path to the target which gives it some unpredictability in movement. This could allow for more leniency in the ghosts chasing pac-man. The ghosts would take longer to get to pac-man which allows him more timesteps to get to his cherry

Perhaps a possible use of 3D plot to display height in z coordinate. This would add an extra layer of code interactivity between objects and their terrain. Certain cells can have certain levels of stacked elevation or even layers of cells to represent floors in a building.

There could be a single floor with obstacles of varying heights. A low height obstacle can be traversed upon (jump onto) where higher height obstacles cannot.

Multiple floors can introduce the idea of movement between floors (ramps/elevators). This could be implemented in a fire evacuation simulation in a building.