PacMan Game

Welcome to my PacMan game! In this game, you will control PacMan, a character who must eat all of the pellets in a maze while avoiding ghosts. The game ends when either all pellets are eaten or PacMan collides with a ghost.

Problem Statement

The problem is that the ghosts need to be smarter. This is done specially by giving one ghost A* pathfinding capabilities, while giving the other 3 ghosts Breadth First Search capabilities.

How to Play

1. Download Pacman.zip and save to your preferred directory.
2. Navigate to the file Pacman.py via command prompt.
3. Use command “python3 Pacman.py” to run game.
4. Press space bar to start game
5. Use the arrow keys on your keyboard to control PacMan's movement.
6. Eat all the pellets in the maze without getting caught by a ghost.
7. If PacMan collides with a ghost, the game is over.
8. After the 3 lives are gone, the game ends.

Features

- Launch Screen: Preview of the Ghosts with instructions to start game. Renamed “PacDom”, well because everyone calls me Dom ☺.
- Maze: The game contains a maze with walls and pellets.
- PacMan: The player controls PacMan's movement with the arrow keys.
- Ghosts: The game features four ghosts that move around the maze.
    - The pink, blue and orange ghost use a Breadth First Search for a 2D array. Their movement is advanced as it is an informed searched based on direction of Pacman.
    - The “Shadow” ghost that goes by the nickname “Doc” is the smartest as it uses the A-Star search for the grid. It uses the heuristic Manhattan distance formula to calculate the total cost of the path from the current location to the target location by adding the heuristic cost to the fixed cost.
- Score: The game keeps track of the player's score as they eat pellets.

- Sound effects: The game includes sound effects for eating pellets, colliding with ghosts and death of Pacman.

Time Complexity Analysis

The A-Star Algoritm is dependent upon heuristic calculations that change with each movement of Pacman. A-Star search algorithm can be expressed as O(b^(d/2)), where b is the branching factor (the number of neighbors that can be reached from each cell), and d is the shortest distance from the start node to the goal node.

The factor of 1/2 is because, in a 2D grid, each cell has up to four neighbors (up, down, left, and right), so the branching factor is typically 4. However, each cell can only be visited once, so the maximum number of cells that can be explored is roughly equal to half of the total number of cells in the grid (worse case being that the start and goal nodes are located in different halves of the grid).

Technologies Used

- Python/Pygames
- Photoshop

Credits

- This game was created by Dominique Kellam.