CSCE 451 Assignment 2 – Maze Search

Due: July 22, 11:00pm

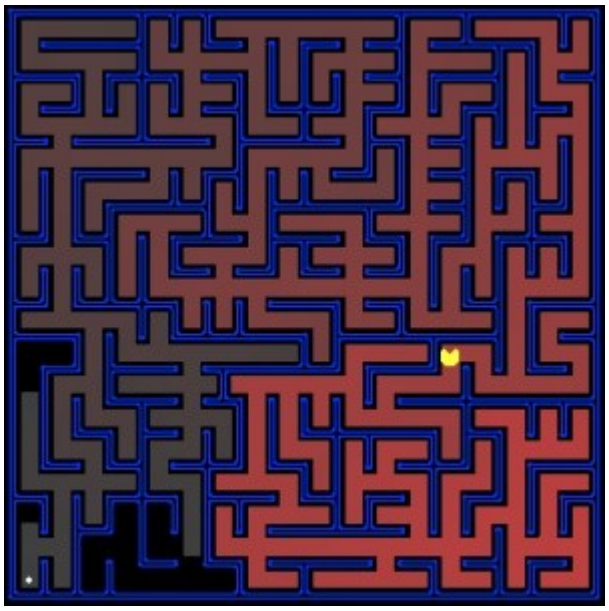
1- The board

In our first Project, we built a game board using a matrix. Can you come up with an alternative way that make the process arguably easier?

What is the advantage of using this "alternative method" compare to the HM1 method? Please explain your answer in your report file.

Note: For this project it does not matter which way you use to represent your matrix.

2- Basic Pathfinding



This assignment is simplified and updated from Berkeley CS 188 projects

To begin with, you will consider the problem of finding a path through a maze from a given start state to a given goal state. This scenario is illustrated in the figure above, where the start position is indicated by the "Pacman" icon and the goal state is a dot. The maze layout will be given to you in a simple text format, where '%' stands for walls, 'P' for the starting position, and '.' for the goal (see sample maze file). For this assignment, all step costs are equal to one.

Implement the following search algorithms for solving different mazes:

- Depth-first search;
- Breadth-first search;
- A* search.

For A* search, use the Manhattan distance from the current position to the goal as the heuristic function.

Run each of the above algorithms on the small maze, medium maze, big maze, and the open maze. For each problem instance and each search algorithm, report the following:

- The solution and its path cost;
- Number of nodes expanded;
- Maximum size of the fringe (or frontier)

Display the solution by putting a '.' in every maze square visited on the path (example solution to the big maze).

Note: For any input maze above, if there are more than one solution only show the best solution.

Tips

- Make sure you get all the bookkeeping right. This includes handling of repeated states (in particular, what happens when you find a better path to a state already on the fringe) and saving the optimal solution path.
- You will be graded on the correctness of your solution, not on the efficiency and elegance of your data structures. For example, I don't care whether your priority queue or repeated state detection uses brute-force search, as long as you end up expanding (roughly) the correct number of nodes and find the optimal solution. So, feel free to use "dumb" data structures as long as it makes your life easier and still enables you to complete the assignment in a reasonable amount of time.

Grading Requirements:

You will need to turn in the following

1- A report in PDF or Word format. The report should briefly describe your implemented solution and fully answer the questions for every part of the assignment. Your description should focus on the most "interesting" aspects of your solution, i.e., any non-obvious implementation choices and parameter settings, and what you have found to be especially important for getting good performance. Feel free to include pseudocode or figures if they are needed to clarify your approach. Your report should be self-contained and it should (ideally) make it possible for me to understand your solution without having to run your source code.

2- The name of the report file should be lastname_firstname_assignment1.pdf.

3- Your source code compressed to a single ZIP file. The code should be well commented, and it should be easy to see the correspondence between what's in the code and what's in the report. **Please do not send your code in Jupyter sheets. Make your code in .py file.**

4- The name of the code archive should be lastname_firstname_assignment1.zip.