Assignment #1: Generating Mazes and Finding Solutions

This assignment is intended primarily to 1) introduce you to/refresh you on Python 3 and 2) investigate search algorithms for pathfinding. Pathfinding is a classical problem in artificial intelligence. You may already be familiar with algorithms like breadth-first search, depth-first search, or even A* search. The software provided with this assignment allows you to create your own mazes and then implement search algorithms to find solutions to your mazes. By using Python3 and tkinter, the application creates a GUI to visualize your maze.

You are allowed to work with a partner on this assignment. Only one submission per group is needed, just make sure that both your names are on the submitted PDF.

Source code for this assignment is provided on Blackboard. If you want to put your code on Github, I ask that you please put it in a private repository.

Learning Objectives

1) Learn/Refamiliarize yourself with Python3
2) Investigate maze generation algorithms
3) Investigate classical AI pathfinding algorithms

Installing Python

In this course you will primarily be working in Python3. If you are not familiar with Python3, this assignment will help give you an overview of Python. I suggest that you also look into installing a virtual environment for this course in order to keep the packages you install separate from your system installation of Python. https://realpython.com/intro-to-pyenv/#virtual-environments-and-pyenv has been my go to for installing and managing virtual environments in Python3.

If you are just learning Python3, feel free to check in with me in office hours and we can discuss good sources to look at for tutorials based on your prior programming experience. Considering you might be coming from Java, one of the many Java to Python guides might be helpful. For a very brief crash course try: https://medium.com/nestedif/cheatsheet-python-for-java-developers-98f75c94a1a

Generating Mazes

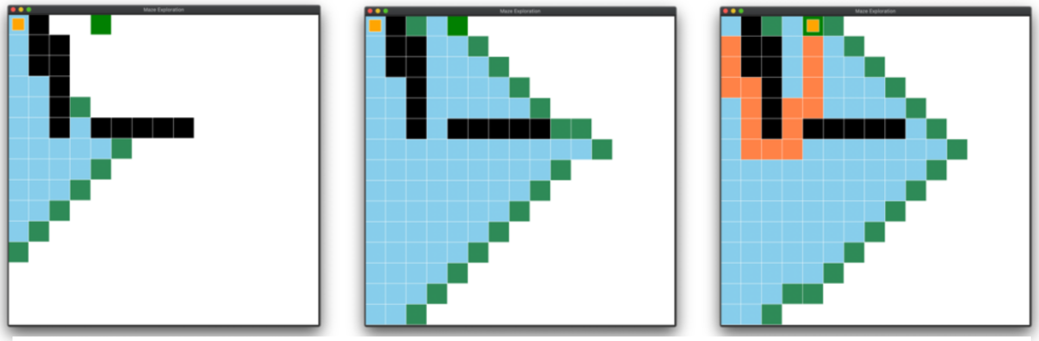
In the first half of this assignment you are to implement a maze generator. While we might think of mazes as traditional, wall-based designs, you could also pursue a “maze” that appears more organic with blobs, islands, or other natural shapes generated by something like Perlin’s noise. No matter which algorithm you choose, generate at least three unique mazes and attach screenshots of them to your submission to show your maze generator in action.

You will find a sample maze generator in the file maze_generator.py. Specifically look at the create_walls function as this is where you likely will generate your obstacles. The world specification can be found in the sample world in the worlds folder under world_enhanced.dat. The parser for the world file can be found in world_spec.py.

Solving Mazes

The second half of this assignment is implementing and visualizing maze solving algorithms. You need to implement two separate maze solvers one being Tremaux's algorithm. Note that your other cannot be bread-first or depth-first search since I have provided the breadth first search implementation as an example. If you want a depth first search note that removing one character from the BFS code turns it into DFS. A* might be a good second algorithm, or one of the variants.

You may choose to create a copy of the Maze_Solver.py file or implement both in the same file with two separate functions. I leave that up to you. For both solvers, visualize their search process across your three mazes by providing a three image sequence of the search in process for each maze and annotating the images with a description of the primary features of the search. See the example for BFS below.



BFS progresses by adding cells to a queue and then visiting them in a first-in, first-out manner. This leads to a potentially large search space as the frontier grows over time. Sparse mazes exacerbate this challenge. The final path to the goal is efficient.

Deliverables

A PDF writeup addressing the following:

- Maze Generator:

1) Summary of the algorithm
2) Why did you choose it? (Some possible prompts)
    a. What unique elements of the algorithm drew your interest?
    b. Did you look at other algorithms but choose this instead?
    c. Were there any challenges in implementing the algorithm?
3) Three screenshots of generated mazes.

- Pathfinding
    1) Summary of each algorithm
    2) Key differences between the two algorithms
    3) Why did you choose each algorithm? (Some possible prompts)
        - What unique characteristics drew you to them?
        - What challenges did you encounter with this algorithm?
        - What Python data structures helped you implement this algorithm?
    4) Three screenshots of each algorithm with annotation describing the search.

This assignment is worth 50 points, or what amounts to 10% of your final grade.

Note: Writeup will be graded on content as well as proper formatting and grammar. Do not rush this phase!