CS440/ECE448 Fall 2023, Mp 4: A* Search for Grid Search

Due: Monday, October 2, 11:59pm

I. Overview

In this assignment you will be implementing the A* search algorithm for solving GridSearch. You can safely reuse some of the codes you wrote for MP3 (e.g. A* search) to solve this problem. You will see both the power of A* as an algorithm that applies to arbitrary discrete state spaces, and the power of heuristics to speed up search.

II. Getting Started

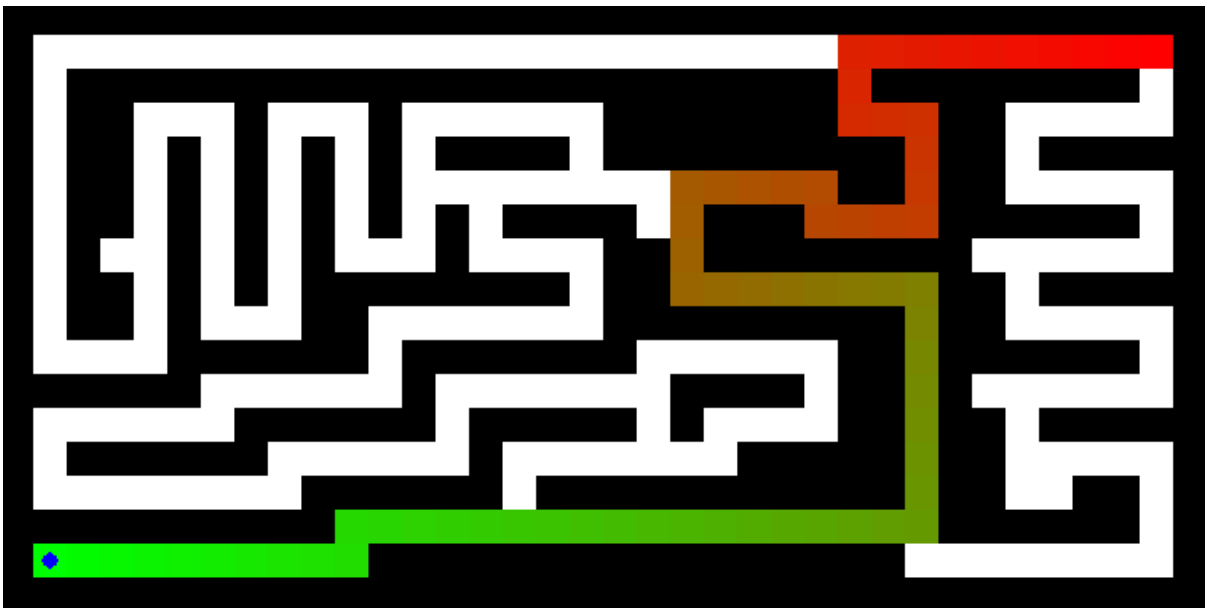
To get started on this assignment, download the template code. The template contains the following files and directories:

- `search.py`. You will edit and submit this - your implementation of A* goes here.
- `state.py`. You will edit and submit this - your implementation of each problem's State goes here.
- `utils.py`. Some utilities we provide.
- `maze.py`. Some utilities for reading and working with grid mazes.
- `main.py`. You will run this file to test your code.
- `data/mazes`. Files containing example mazes to solve

Please ONLY submit `search.py` and `state.py`.

For each of the remaining parts of the assignment you will find TODOs in `search.py` and `state.py` where you need to write your own code. For example, for part V you will find TODOs marked `# TODO(V)`. We've provided many comments and instructions in the code under those TODOs.

V. Single Goal Grid Search



In grid search we find a path through a 2D maze from some starting location to a single goal location. Each state is a discrete (x,y) location in the maze, and the goal is an additional location in the maze. From any location you can transition to a neighboring location assuming there is no obstacle there (colored black). In the visualization above green indicates the end of the path and red indicates the beginning.

Finally we come to GridSearch.

Run the following to see mazes and navigate them yourself with the arrow keys:

```
python3 main.py --problem_type=GridSingle --human --maze_file=[path_to_maze_file in data/mazes/grid_single]
```

- **note**: if the red-green gradient is hard for you to see, you can make the visualization use an alternative color scheme by specifying the `--altcolor` option

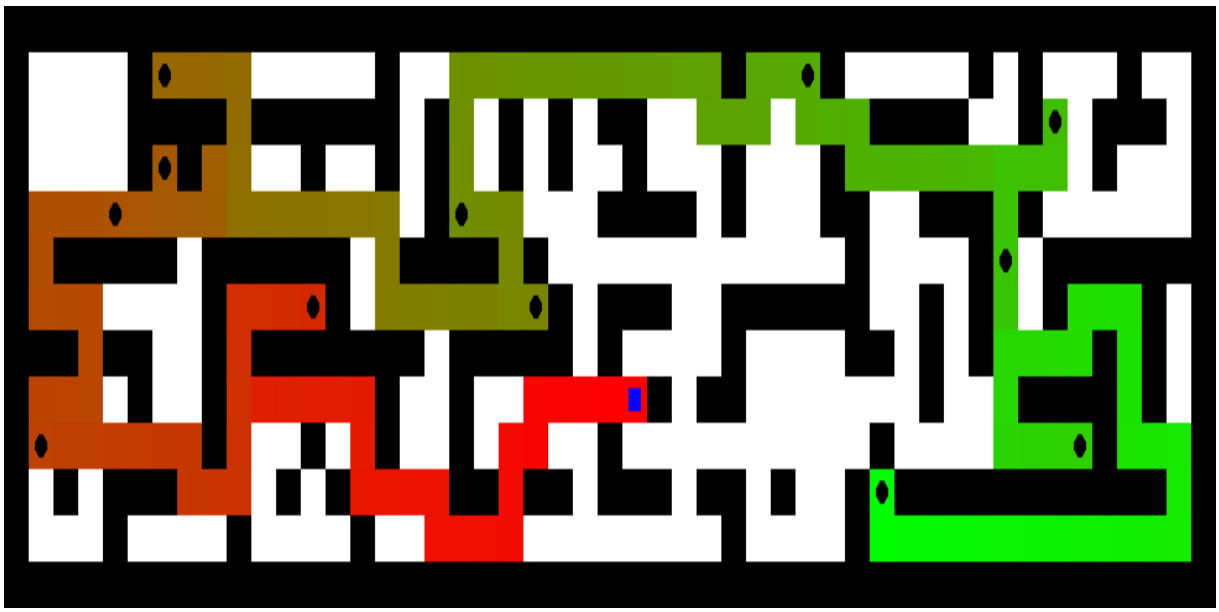
Now you will need to implement the `SingleGoalGridState` class. If you would like to see how the maze is built navigate to `maze.py`, but otherwise we've provided you everything you need and instructions in `state.py` where you have 4 "TODO(V)" to complete. The heuristic we use for grid search is the manhattan distance from the current location to the goal.

You can test your code with:

```
python3 main.py --problem_type=GridSingle --maze_file=[path_to_maze_file in data/mazes/grid_single]
```

If you would like to also visualize the resulting solution in PyGame add the `--show_maze_vis` flag.

VI. Multi Goal Grid Search



Now we consider grid search problems with multiple goals that can be reached in any order

We now generalize single goal grid search to multi goal grid search. There are 5 "TODO(VI)" for you to complete in `state.py`.

Multiple goals requires a new heuristic. The one we use is the Minimum Spanning Tree (MST). Specifically, given a state and a set of goals, the heuristic cost for visiting all the goals is computed as follows:

- Treat the set of goals as nodes in a complete graph
- The edge weights on this graph are the distances between the goals (or approximate distances using manhattan)
- To solve the multi goal search problem (i.e., a version of the Travelling Salesman Problem) we must travel AT LEAST the cost of the MST of this graph
- In addition to the cost of the MST we also need to reach the first goal, therefore our final heuristic is:
  - h(state, goals) = manhattan(state, closest_goal) + MST(goals)

We provide you with most of the code you need to compute this MST heuristic, you can call `compute_mst_cost(self.goal, manhattan)` to compute the cost of the minimum spanning tree for a set of goals. **Note that because computing the mst takes some time, you should store the computed mst values in the cache we provide you.**

You can test your code with:

```
python3 main.py --problem_type=GridMulti --maze_file=[path_to_maze_file in data/mazes/grid_multi]
```

You can also check backwards compatibility by running GridMulti on a maze file with only one goal.

VII Submission Instructions

Submit the main part of this assignment by uploading `search.py` and `state.py` to Gradescope.

Policies

You are expected to be familiar with the general policies on the course syllabus (e.g. academic integrity) and on the top-level MP page (e.g. code style). In particular, notice that this is an individual assignment.