COMP3211 – Fundamentals of Artificial Intelligence

2025 Spring Semester – Assignment 2 Programming Part (18 points)

Maze Navigation Agents

Date Assigned: Mar 9, 2025
Due Time: 23:59 on Mar 23, 2025

How to Submit

- Submit your codes as a zip file named `{YourStudentID}.zip`.
- **Important Note:** Only submit your `search.py` file. There is no need to submit any other files. Your `{YourStudentID}.zip` file should contain only `search.py` to ensure the autograder can correctly recognize it. If you are concerned that the TA might not correctly identify your file, you may add a comment in `search.py` with your name and Student ID.
- Please avoid plagiarism. You must acknowledge individuals who assisted you or sources where you found solutions. Failure to do so will be considered plagiarism.
- **No late submission will be accepted.**

Overview

This assignment requires you to implement two search algorithms to navigate a maze and collect all goal points (`C`). You will implement:

- **Problem 7: Depth-First Search (DFS) Agent (9 pts)**
- **Problem 8: A* Search Agent (9 pts)**

Your goal is to complete the corresponding functions in `search.py`, ensuring the agent successfully navigates the maze using the specified algorithms.

Provided Files

The following files are provided to you:

1. `main.py`

- This is the main program to run the maze navigation using different search algorithms.
- It uses **Pygame** to visualize the search process.
- You can specify which algorithm to run using command-line arguments.
- **Do not modify this file**.

2. `maze_maps.py`

- Contains different maze configurations.
- Each maze consists of:
  - `#` for walls
  - Spaces for open paths
  - `C` for collectible goals
- **Do not modify this file**.

3. `problem.py`

- Defines the `SearchProblem` class, which represents the maze navigation problem.
- The `MazeProblemMultiGoal` class extends `SearchProblem` to handle multiple goal points.
- **Do not modify this file**.

4. `search.py`

- Implements the search algorithms.
- **You need to modify this file** to implement DFS and A* search.
- The following functions are currently placeholders and must be completed:
  - `depth_first_search(problem: SearchProblem)` (for Problem 7)
  - `a_star_search(problem: SearchProblem)` (for Problem 8)
  - `heuristic(goals, state)`, which is used in A* search.
- **Only modify this file**.

Running the Code

To run the program and test your search algorithms, use the following commands in the terminal:

Running the test demo (random walk):

```
python main.py
```

Running DFS:

```
python main.py --algorithm dfs
```

Running A*:

```
python main.py --algorithm astar
```

- The `--map` argument selects a predefined maze from `maze_maps.py`. You can try different values (0, 1, 2) to test different mazes.
- The `--tick` argument controls the speed of visualization.

Problem Requirements

Problem 7: DFS Agent (9 pts)

- Implement **Depth-First Search (DFS)** in `depth_first_search(problem)`.
- Use a **stack** (Last-In-First-Out structure) to explore the search space.
- Maintain a **set of visited states** to avoid re-exploration.
- Yield each visited state for visualization.
- Return a tuple: `(path from start to goal, set of visited states)`.
- If no path is found, return `(None, visited)`.

Problem 8: A* Search Agent (9 pts)

- Implement **A* Search** in `a_star_search(problem)`.
- Use a **priority queue** (heap) to prioritize states based on `f = g + h`, where:
  - `g(n)`: Cost from the start state to the current state.
  - `h(n)`: Heuristic estimate of remaining cost to the goal.
- Implement a heuristic function in `heuristic(goals, state)`.
- Yield each visited state for visualization.
- Return a tuple: `(path from start to goal, set of visited states)`.
- If no path is found, return `(None, visited)`.

Special Requirements

- **Use Python standard libraries only.** No external packages are allowed.
- Make sure your implementation works correctly and efficiently.
- Test your implementations using different maps.

Marking Scheme

For Problem 7 & 8, we will test your agent with three hidden maps (3 points each).

Good luck!