Artificial-Intelligence-Lab4

叶璨铭，12011404@mail.sustech.edu.cn

Lab Practice

In Lab3, you have implemented 3 algorithms, DFS, BFS and UCS.

In Lab4 this week, we are going to go through GBFS(Greedy best first search) and A star.

Greedy Best First Search

Here is a map, where the blue point is the starting point, and the green point is the goal point. The problem for the agent to solve is to find a path that reaches the goal state.

You need to complete the method `GBFS` in `agent.py`. The you can see the effect like this:

 GBFS

A star Search

 Astar0

Coding Instruction

```
├──gif
├──test_cases
├──utils
├---agent.py
├---main.py
├---plotting.py
```

The Project is composed of 3 folders and 3 files.

- The main part you should finish is `agent.py`， in which the path searching algorithms will be implemented.
- If you finishes the algorithms, you can run `main.py` to test your program.
- `plotting.py` and `utils` contains some tools to help main.py and agent.py to visualize the running process of the algorithm you implemented.

- `test_cases` includes world configuration files suffixed after `toml`, a popular markup language that is equivalent to `json` but is more human friendly. The `main.py` will read different test cases configuration and load different maps described by `.png` file to test your algorithms.

Now let's look deeper into `agent.py`

agent.py

In class `ProblemSolvingAgent` you can see this method.

```
def solve_by_searching(self, obstacles, start_pos, goal_pos, algorithm='DFS'):
```

The method is the API of the agent, which will be invoked by the `main.py`.

`solve_by_searching` will dispatch the task to different methods below, so you don't need to change it.

What you need to do is to complete the following methods.

```
def GBFS(self, obstacles, start_pos, goal_pos):
    path, visited = [], []
    return path, visited
def Astar(self, obstacles, start_pos, goal_pos):
    path, visited = [], []
    return path, visited
```

The data types and meanings of the input output variables are described in the Python doc of `solve_by_searching`. Read them carefully!

```
 """Let the agent solve problem by searching path on the graph.
        Args:
            obstacles (list of bi-tuples):
                Obstacles represents the graph information of the grid map,
                by a list of points called obstacles.
                At any coordinate, you are allowed to move to
                any node nearby that is not in the obstacles.
                When coding, you can use self.neighbours(obstacles, node)
            start_pos (bi-tuples): the position of initial state.
            goal_pos (bi-tuples): the position of goal state.
            algorithm (str, optional): The strategy applied by the agent.
                Defaults to 'DFS'.
        Returns: tuple (path, visited)
            path (list of bi-tuples): the path chosen by the algorithm
                to navigate from initial position to the goal position
            visited(list of bi-tuples): the position checked by the agent
                during the searching process.
        """
```

There is also some APIs to help your programming, among which they most important one is `neighbours_of`

```
def neighbours_of(self, obstacles, node):
        """_summary_

        Args:
            obstacles (_type_): _description_
            node (_type_): _description_
        Returns: iterable generator of tuple(neighbour, moving_cost)
            neighbour(bi-tuple): a position near to the node.
            moving_cost(float): the cost the agent has to pay to move from node to
neighbour.
        """
```

This is the same as `adj()` of a graph represented by `adjecency list` you learnt in `DSAA course`. For example,

```
# 2x2 grid map graph is (0, 0)<-sqrt(2)->(1,1)
(0,0)<-1->(1, 0) (1, 1)<-1->(1,0)
# obstacles are [(0, 1)]
self.neighbours_of(obstacles, (0, 0)) == [((1,1), sqrt(2)), ((1,0), 1)]
```

Frequently asked questions

How to prove that Astar remains optimal?

Reference

[^1]: “zhm-real/PathPlanning: Common used path planning algorithms with animations.” GitHub - zhm-real/PathPlanning: Common used path planning algorithms with animations. (accessed Sep. 21, 2022). [^2]: “Intelligent-Robot-Course/Homework: The homework for this course.” GitHub - Intelligent-Robot-Course/Homework: The homework for this course (accessed Sep. 21, 2022). [^3]: A. Felner, “Position Paper: Dijkstra’s Algorithm versus Uniform Cost Search or a Case Against Dijkstra’s Algorithm,” p. 5. [^4]: Stuartj. Russell, PeterNorvig, 诺维格, 罗素, 祝恩, and 殷建平, “人工智能:一种现代的方法,” 清华大学出版社, 2013, doi: 9787302331094.