CS 4300: Artificial Intelligence

Assignment: A* Search Agent

Create an agent to perform in the MiniGrid - MultiRoom environment, (minigrid source code).

The performance measure used by this assignment to assess the quality of your agent will be the episode total reward, averaged over at least 25 episodes. An agent that doesn't complete some episodes (times out / runs out of memory / crashes), will give given an average score of 0.

After your report and code are reviewed, assignment grades will be assigned. The maximum *possible* score will be controlled by the agent's performance measure. See the table below.

Use the GitHub repository available for this course to store your solutions. Make a directory named `multiroom-astar-search`, and store your agent in `multiroom-astar-search/agent1.py`.

Note that you are to implement an agent that has a *model* of the environment and uses the A-star search algorithm we have discussed. DO NOT make a reinforcement learning agent, or use some other algorithms for these agents.

The model must contain at least these methods:

- ACTIONS(s) -> list of actions allowed in state s
- RESULT(s, a) -> state that results from action a in state s
- GOAL-TEST(s) -> true or false, depending on the state s
- STEP-COST(s, a, s1) -> cost of taking action a in state s and ending up in state s1
- HEURISTIC(s) -> estimated cost of reaching a goal state from state s.

We discussed in class that using graph search would probably be better than tree-like search.

Create a short report, containing these elements:

- An explanation of the heuristics tried, which one was selected as the best, and why it was selected.
- The average performance of your agent on each of the following environments: `MiniGrid-MultiRoom-N2-S4-v0`, `MiniGrid-MultiRoom-N4-S5-v0`, and `MiniGrid-MultiRoom-N6-S6-v0`

Required Submissions

- Code submitted to github.
- A PDF file containing your report submitted to Canvas.

Performance Measure Expectations

MiniGrid-MultiRoom-N6-S6-v0

Average Score	Maximum Possible Grade
a < 0.50	50%
0.50 <= a < 0.60	65%
0.60 <= a < 0.68	75%
0.68 <= a < 0.69	80%
0.69 <= a < 0.70	85%
0.70 <= a < 0.71	90%
0.71 <= a < 0.72	95%
a >= 0.72	100%

Hints and Resources

- `import my_minigrids` to make a variety of environment configurations available. They have the form `MiniGrid-MultiRoom-N{}-S{}-v0`, where N2-N9 and S4-S9 are available. NX will have X rooms, and SY will have rooms at most YxY in size. my_minigrids.py

- As discussed in class, the unique state information is: agent row, agent column, agent direction, and (door row, door column, door state) for every door. Everything else in the environment should never change. This is useful for generating the unique key per state, if you do graph search.

- There is a builtin PriorityQueue in Python. If you use it, I recommend using a class for your nodes, with the `__lt__` operator overloaded to correctly sort nodes for the A-star algorithm.
- Try to write the A-star code such that it could work with any model that supported the methods listed above.
- Use manual_control.py found in the MiniGrid repository to manually run the agent. I made a personal copy of it and modified the code to help me explore the structure of the observations.
- constants.py Can help to interpret the contents of each grid location.
- The observation is a dictionary. The `observation["image"]` contains a 3-D numpy array. The first index is the *COLUMN*, the second index is the *ROW*, and the third index is 0-contents of grid cell, 1-color of grid cell, 2-state of grid cell.
- The Fully Observable environment wrapper is necessary to have the required information for search in the observation.
- In the RESULT method, when moving the agent's location, it's OK to set the contents of the grid cell the agent is moving from to be "empty".