CPSC 383: Explorations in Artificial Intelligence and Machine Learning

Assignment 1: Symbolic AI, Agents, Search, Pathfinding, A*

Weight: 15%

Collaboration

Discussing the assignment requirements with others is a reasonable thing to do, and an excellent way to learn. However, the work you hand-in must ultimately be your work. This is essential for you to benefit from the learning experience, and for the instructors and TAs to grade you fairly. Handing in work that is not your original work, but is represented as such, is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. Cite all sources of code that you hand-in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:

   ```
   # the following code is from
   https://www.quackit.com/python/tutorial/python_hello_world.cfm.
   ```

   Use the complete URL so that the marker can check the source.

2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. **However, you may still get a low grade if you submit code that is not primarily developed by yourself. Cited material should never be used to complete core assignment specifications. You can and should verify and code you are concerned with your instructor/TA before submission.**
3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's screen, then this code is not yours.
4. **Collaborative coding is strictly prohibited**. **Your assignment submission must be strictly your code**. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. **You can not use (even with citation) another student's code.**
5. Making your code available, even passively (e.g. online repository accessible to other students), for others to copy, or potentially copy, is also plagiarism.
6. We will be looking for plagiarism in all code submissions, possibly using automated software designed for the task. For example, see Measures of Software Similarity (MOSS - https://theory.stanford.edu/~aiken/moss/).
7. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize. **The most common penalty is an F on a plagiarized assignment.**
8. For assignments **limited use of generative AI in writing assistance is acceptable. For example, grammar suggestion, or code suggestion tools for programming**. **Programming or text that is largely generative AI produced is not allowed**. Learners are ultimately accountable for the work they submit. Use of AI tools must

be documented in an appendix for the assignment. The documentation should include what tool(s) were used, how they were used, and how the results from the AI were incorporated into the submitted work. Failure to cite the use of AI generated content in an assignment will be considered a breach of academic integrity and subject to Academic Misconduct procedures.

Late Penalty

For late individual assignments, those submitted within 24 hours of the initial deadline will receive 10% off, and within 48 hours will receive 20% off. After 48 hours, no late assignments will be accepted. -10% of 20 marks is -2 marks. -20% is -4.

Goal

Within the provided **AEGIS** system modify the provided **Python agent** code to allow the agent to **path-find** using **A*** to the **SURVIVOR** grid location. The solution must be written by you the student and not use existing libraries.

Technology

AEGIS, Python 3

Submission Instructions

You must submit your assignment electronically using **D2L**. Use the Assignment 1 dropbox in **D2L** for a final codebase electronic submission. In **D2L**, you can submit multiple times over the top of a previous submission. Do not wait until the last minute to attempt to submit. You are responsible if you attempt this, and time runs out. Your assignment must be completed in **Python 3**.

Description

What is AEGIS?

The Goobs have sent an elite space force to occupy AEGIS, the galaxy's central hub dedicated to saving lives across the galaxy. This futuristic space station floats in a serene nebula, and it's the last beacon of hope in the vast and dangerous expanse of space. Equipped with powerful scanners and teleportation gates, AEGIS connects distant worlds and provides a lifeline in the galaxy's most perilous regions. From here, they embark on their journeys, navigating the galaxy's dangers to rescue those in distress and tackle the most formidable dangers.

AEGIS is a simulated agent disaster rescue scenario environment written in Python 3 with an optional Electron-based GUI. AEGIS consists of a simulation controller that manages all communication with agents, executes the simulation, and maintains the current state of the world. It updates the world as events happen. If the optional GUI is used, the GUI displays real-time updates of the world’s state during the simulation. Additionally, the GUI allows users to create their own worlds.

Full documentation is available here and students are expected to read this themselves to complete the assignment: https://aegis-game.github.io/docs/ Tutorials will also introduce students to the AEGIS system.

The API is primarily under https://aegis-game.github.io/docs/docs/api/,

and getting started information under https://aegis-game.github.io/docs/docs/getting-started/installation/

The AEGIS simulation is turn-based. For assignment 1 you will have one agent connect to the server. On connection agents are given a simplified rectangular grid view of the world which includes if a grid is safe to move on (**KILLER** grid locations are instant-'death', other grid types are safe to move on) However, grid locations do each have a **MOVE_COST** of energy. If an agent runs out of energy they also 'die' for remainder of simulation. **GET_SURVS()** can be used to find the **LOCATION** of the **ONE SURVIVOR**. It is your goal to save the **ONE SURVIVOR** in the grid without running out of energy as efficiently as possible. Efficiency is to limit the energy use of your agent to the minimal possible given the knowledge available to the agent.

A simulation consists of multiple rounds, with each round being a single time step in the simulated world. During each round each agent can pick one action to take. Agents have energy that is expended each time they use commands. If their energy expiries the agent becomes non-functional.

For assignment 1 your **ONE AGENT** needs only to be concerned with the commands of **MOVE(DIRECTION)** and **SAVE()** (save survivor). These are described at:

https://aegis-game.github.io/docs/docs/api/agent/

MOVE(DIRECTION) allows an agent on its turn to move in 1 of the 9 neighboring grid locations (or **CENTER** to not move).

https://aegis-game.github.io/docs/docs/api/direction/

The result of a **MOVE** action informs an agent of its remaining energy levels and information about the surrounding grid locations around the agent's new location. This information will be stored for you when received.

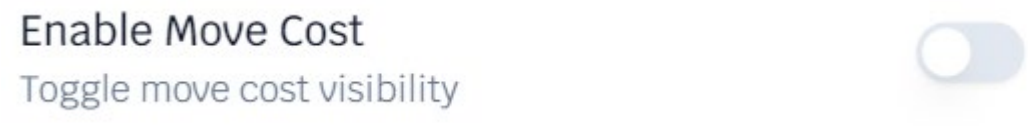
Once the agent moves onto the grid location containing the survivor the **SAVE()** command is used to end the simulation. If there is no survivor at that location the **SAVE()** command will have no effect. Agents must choose to send only one of these two commands each turn of the simulation.

Assignment Challenge

Your agent in AEGIS will begin on a grid location that does not contain the **SURVIVOR**. This is a **SPAWN** grid location. It will be your job to move your agent each round until it is at the

SURVIVOR location and then use **SAVE()** to end the simulation. To accomplish this goal, you will be using A*, a very common path-finding algorithm. A*, as introduced in class, is a graph search algorithm that uses an admissible heuristic to provide an optimal sequence of **DIRECTION** that can be used in **MOVE(DIRECTION)** commands so that your agent reaches the survivor as efficiently as possible. You will be creating an A* solution for this assignment that can solve three versions of the AEGIS path-finding problem. (Note, it is acceptable to just do version 3 and it will solve version 1 and 2. We will have equal tests for each version in case your A* doesn't work as well for a later version challenge.)

1. In version 1, all move costs will be equal (1 energy). Your agent will need to use A* to avoid instant-death **KILLER** grids to navigate to the **SURVIVOR** in the shortest path (fewest **MOVE(DIRECTION)** commands) possible. This minimizes both energy and **MOVE(DIRECTION)** commands. Note, that sometimes the path will not be a straight-forward and require some maze-solving.
   Your agent will know all these costs from the start and you should ensure the **USER SETTING** below is toggle **OFF** (on hides move costs right now). (The default agent does **MOVE(CENTER)** on round 1 to reveal hidden **MOVE_COSTS** at start, but that is unnecessary here and would waste energy. However, this is useful in version 3 so we except it to be retained.)

2. In version 2, AEGIS will provide move costs of varying value. It is your job to find the path to the survivor that **minimizes the energy MOVE_COSTS** of the agent. This path may be longer than the shortest quantity of **MOVE(DIRECTION)** commands that ignores **MOVE_COST** energy.
   Your agent will know all these costs from the start and you should ensure the **USER SETTING** below is toggle **OFF** (on hides move costs right now). (The default agent does **MOVE(CENTER)** on round 1 to reveal hidden **MOVE_COSTS** at start, but that is unnecessary here and would waste energy. However, this is useful in version 3 so we except it to be retained.)

3. In version 3, AEGIS will only provide **MOVE_COST** values as the result of **MOVE()** actions (not at the start of the simulation). Each time you move to a new **LOCATION** you will have the 9 adjacent **MOVE_COST** around the agent revealed.
   Your agent will have to solve the path-finding problem without knowing move costs at the beginning. In other words, you will have to assume default move costs, and as your

agent completes **MOVE()** actions and learns more, your agent will have to re-assess the best path to the survivor. Note, this could mean pathfinding in one direction before finding that there was a previously unrevealed wall of 'instant-death' energy **MOVE_COST** you have to back-track from to avoid.
Your agent will **NOT** know all these costs from the start and you should ensure the **USER SETTING** below is toggle **ON** (on hides move costs right now). (The default agent does **MOVE(CENTER)** on round 1 to reveal hidden **MOVE_COSTS** at start.)

Note, your agent should tie-break equal paths choices with the order of DIRECTION -> N, NE, E, SE, S, SW, W, NW, C. It is necessary to follow this tie break for full credit on our test maps.

Solution

A* star is a very well-known algorithm https://en.wikipedia.org/wiki/A*_search_algorithm . **You'll find even better information attending your tutorial on pathfinding for pseudo-code (as well as a participation)**.

It is possible to find other pseudo-code and explanations all over the internet. If you use such a resource (including that Wikipedia link], then you should reference it in your code submission. I do not recommend this over going to tutorials and following the given suggestion. Be vary of pathfinding solutions that depart from the provided recommendation as these may give incorrect answers and lose grades.

You will need to modify the **THINK()** function in the agent to use your A* path-finding. In **THINK()** you will need to solve A* for your World state to find a path towards the one **SURVIVOR** grid location.

Evaluation

The files provided for assignment include 2 example world files for each of the three versions. For version 1 and 2 ("**HIDDEN_MOVE_COSTS":** false). For version 3 ("**HIDDEN_MOVE_COSTS":** true).

You are free to make, and share in discord, other example worlds you make using AEGIS to test your system.

For grading the instructional staff will have 5 world files for each of the versions that will be used to assess students submitted code.

Additional Specification

- Put your **name, date, course, semester, and tutorial** into the comments of the **main.py** of your modified **agent_path**.
- **You must comment your code, provide citations for the source of algorithmic designs, and cite any GenAI code suggestion usage.**
- **Do not rename or modify the provided common files.**
- You should not have any other .py files for assignment 1.
- **You should not import ANY non provided libraries to complete the A* algorithm. Using such could result in grade of 0 for that portion of assignment. (You are allowed to use heapq and math). Failure to follow this restriction will result in 0 grade.**
- Do not change provided code without discussion with instructor. If there is a bug, or something is broken, the instructor should be informed to fix this issue.

Grading

The total grade is out of 20.

Version 1
    5 test maps (success if minimum path is achieved -> least move actions)
Version 2
    5 test maps (success if minimum move cost path is achieved)
Version 3
    5 test maps (success if minimum move cost path is achieved with challenge of having to re-plan when move costs are revealed as agents **MOVE**)
Style/Commenting (out of 5)
    Name/Date/Course/Semester/Tutorial, don't change files, etc.
Bonus
    Variant of version 3 test maps. Your agent needs to be able to notice that it can't reach destination without charging once. If your agent first pathfinds first in this situation to the closest charging grid to its destination that it can reach without running out of energy, and then second to its destination after having charged enough energy to finish the trip (in our test map) you will get a bonus credit of 1 grade point.

Submit the following using the Assignment 1 Dropbox in D2L

1. **main.py**
    a. **Just submit this one file for grading**