THE UNIVERSITY OF MELBOURNE
SCHOOL OF COMPUTING AND INFORMATION SYSTEMS
COMP90054 AI PLANNING FOR AUTONOMY

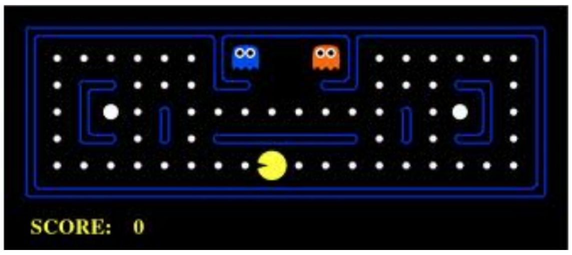
Project 1, 2019

Deadline: Monday 2 September 18:00

This project counts towards 10% of the marks for this subject.
This project must be done individually.

Aims

The aims of this project are to improve your understanding of the various search algorithms and to experience how to derive heuristics, using the Berkely Pac Man framework.



https://inst.eecs.berkeley.edu/~cs188/fa18/project1.html

Your task

Your tasks relate to the assignment at https://inst.eecs.berkeley.edu/~cs188/fa18/project1.html.

Set Repository to Private (0 marks)

Once you have forked the repository, your repository may be viewable by other students in the class. To avoid any issues with academic misconduct, please set your repository to 'private'.You can does this by going to gitlab.eng.unimelb.edu.au, selecting your `comp90054-a1-2019` repository, navigating to the privacy settings using 'Settings' , then 'General', then 'Permissions', and selecting 'private'.

Please do this as soon as you fork the repository.

Practice Task (0 marks)

To familiarise yourself with basic search algorithms and the Pacman environment, it is a good start to implement the tasks at https://inst.eecs.berkeley.edu/~cs188/fa18/project1.html, especially the first four tasks; however, there is no requirement to do so.

Part 1 (2 marks)

Implement the Iterative Deepening Search algorithm discussed in lectures. You should be able to test the algorithm using the following command:

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ids
```

Other layouts are available in the `layouts` directory, and you can easily create you own!

Part 2 (2 marks)

Implement the Weighted A* algorithm discussed in lectures using $W = 2$. You may hardcode this weight into your algorithm (that is, do not pass as a parameter).

You should be able to call your function using the `fn=wastar` parameter from the command line, i.e. you should be able to test the algorithm using the following command:

```
python pacman.py -l bigMaze -z .5 -p SearchAgent
      -a fn=wastar,heuristic=manhattanHeuristic
```

Other layouts are available in the `layouts` directory, and you can easily create you own!

Part 3 (6 marks)

Now we wish to solve a more complicated problem. Just like in Q7 of the Berkerley Pac Man framework, we woud like to create an agent that will eat all of the dots in a maze. Before doing so, however, the agent must eat a Capsule that is present in the maze. Your code should ensure that no food is eaten before the Capsule. You can assume that there is always exactly one Capsule in the maze, and that there will always be at least one path from Pacman's starting point to the capsule that doesn't pass through any food.

In order to implement this, you should create a new problem called `CapsuleSearchProblem` and a new agent called `CapsuleSearchAgent`. You will also need to implement a suitable `foodHeuristic`. You may choose to implement other helper classes/functions. You should be able to test your program by running the following code:

```
python pacman.py -l capsuleSearch -p CapsuleSearchAgent
     -a fn=wastar,prob=CapsuleSearchProblem,heuristic=foodHeuristic
```

An agent that eats the capsule then proceeds to eat all of the food on the maze will receive 3 marks. The remaining 3 marks will be based on the performance of your agent (i.e. number of nodes expanded), as in Q7 of the Berkeley problem. Since you are using the Weighted A* algorithm, however, the number of node expansions required for each grade will vary.

HINT: Think carefully about how you intend to structure your solution before starting to implement it.

NOTE: You should not change any files other than `search.py` and `searchAgents.py`. You should not import any additional libraries into your code. This risks being incompatible with our marking scripts.

Marking criteria

This assignment is worth 10% of your overall grade for this subject. Marks are allocated according to the breakdown listed above, based on how many of our tests the algorithms pass. No marks will be given for code formatting, etc.

Submission

The master branch on your repository will be cloned at the due date and time.

From this repository, we will copy *only* the files: `search.py` and `searchAgents.py`. Do not change any other file as part of your solution, or it will not run. Breaking these instructions breaks our marking scripts, delays marks being returned, and more importantly, gives us a headache.

Note: Submissions that fail to follow the above will be penalised.

Academic Misconduct

The University misconduct policy[1] applies. Students are encouraged to discuss the assignment topics, but all submitted work must represent the individuals understanding of the topic. The subject staff take academic misconduct seriously. In the past, we have prosecuted several students that have breached the university policy. Often this results in receiving 0 marks for the assessment, and in some cases, has resulted in failure of the subject.

Important: As part of marking, we run all submissions via a code similarity comparison tool. These tools are quite sophisticated and are not easily fooled by attempts to make code look different. In short, if you copy code from classmates or from online sources, you risk facing academic misconduct charges.

But more importantly, the point of this assignment is to have you work through a series of foundational search algorithms. Successfully completing this assignment will make the rest of the subject, including other assessment, much smoother for you. If you cannot work out solutions for this assignment, submitting another persons code will not help in the long run.

[1]See `https://academichonesty.unimelb.edu.au/policy.html`