CPSC 481 Artificial Intelligence
Project 2 – Heuristics for Pacman

Mode: team of up to four
Due Date: As shown

Continue with Pacman

For this project, you still download the starter code (**search.zip**), unzipping it, and changing to the directory.
The starter code and the autograder are the same as the ones we used in Project 1.

Files to Edit and Submit: You will fill in portions of **search.py** and **searchAgents.py** during the assignment. Please do not change the other files in this distribution or submit any of our original files other than these files.

Evaluation: Your code will be autograded for technical correctness. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation – not the autograder's judgements – will be the final judge of your score. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

Getting Help: You are not alone! If you find yourself stuck on something, office hours, section, and the discussion forum are there for your support; please use them. If you can't make our office hours, let us know and we will schedule more. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

Discussion: Please be careful not to post spoilers.

Question 4: A* search

Implement A* graph search in the empty function aStarSearch in search.py. A* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the

main argument), and the problem itself (for reference information). The nullHeuristic heuristic function in search.py is a trivial example.

You can test your A* implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic (implemented already as manhattan Heuristic in searchAgents.py).

*python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhatta*nHeuristic

Question 5: Finding All the Corners (for graduate students ONLY)

Undergraduate students do NOT need to finish this question.
Graduate students need to finish this question to fulfil university policy on 400-level courses.
"According to the University policy on 400-level courses, *Graduate students are expected to complete additional work or perform better than undergraduate students.*"
If a team has both undergrad. and grad. students, graduate students need to finish Q5 and include the required code file in the submission.

The real power of A* will only be apparent with a more challenging search problem. Now, it's time to formulate a new problem and design a heuristic for it.

In corner mazes, there are four dots, one in each corner. Our new search problem is to find the shortest path through the maze that touches all four corners (whether the maze actually has food there or not). Note that for some mazes like tinyCorners, the shortest path does not always go to the closest food first! Hint: the shortest path through tinyCorners takes 28 steps.

Implement the CornersProblem search problem in **searchAgents.py**. You will need to choose a state representation that encodes all the information necessary to detect whether all four corners have been reached. Now, your search agent should solve:

python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

To receive full credit, you need to define an abstract state representation that does not encode irrelevant information (like the position of ghosts, where extra food is, etc.). **In particular, do not use a Pacman GameState as a search state. Your code will be very, very slow if you do (and also wrong).**

Hint: The only parts of the game state you need to reference in your implementation are the starting Pacman position and the location of the four corners.

Our implementation of breadthFirstSearch expands just under 2000 search nodes on mediumCorners. However, heuristics (used with A* search) can reduce the amount of searching required.

Plz. Ignore questions 3, 6-8 for this project.

Grading for this project:
There is a project 2 rubric attached below project 2 description on Canvas.
Your program needs to pass autograder tests.
You can find the test cases located in "test_cases" folder in search.zip
Autograder will not only test Pacman but also some search graphs in the "test_cases" folder.

Deliverables:
Include team members in a readme file.
Name, graduate student or undergraduate student.
Submit search.py and readme to the submission link.
If Q5 is implemented, you will also need to submit searchAgents.py.

One team only submit ONE copy. Pick a team member who will be responsible for submitting the project.
Other members do not need to submit. Canvas may mark your project as "Missing", but you do not need to worry about it.
I'll contact you if I cannot find your name in any readme files.