Homework 4: Pacman versus the world

Due October 2nd at 10pm

Part 1: Basic search

The UC Berkeley AI course has a set of projects that revolve around the Pacman game. Pacman is a game in which your character, seemingly a sentient block of cheese, tries to (1) eat lots of pellets and (2) avoid being eaten by ghosts.

The Berkeley projects come with a Python library that implements a GUI to run Pacman on your computer. Your goal is to fill in some of the functions related to search, which implement strategies for the Pacman agent to use during the game.

Refer to the UC Berkeley assignment for instructions on how to run the Pacman game simulation: UC Berkeley Project 1.

Download the starter code using the link on the course website. If you unzip this file, you will find a folder called singleagent.

In Part 1, you will implement basic search strategies that will enable you to find a path to a goal (for instance, a game block with a particularly tasty item, like a strawberry). You will fill in missing functions in **search.py**.

You can check your work by running the autograder.py file that is provided. You are not required to implement all parts of the original UC Berkeley assignment, so you should expect to see zeros for questions that you have not attempted.

Task 1: Breadth-first search

Implement breadth-first search by completing the breadthFirstSearch function. I have already implemented the depthFirstSearch function for you. Much of the code will be similar, although you will need a different data structure.

Before starting your implementation, take a look at the data structures provided in util.py. You should also look at the searchProblem class at the top of the search.py file, so that you understand what methods a problem object has.

Task 2: A* search

In A* search, your agent is guided by a heuristic. **Implement A* search for Pacman by completing the aStarSearch function in search.py.**

The search problem is very similar to breadth-first search. The difference is that now your data structure needs to be one that is ordered according to the heuristic that you are using to guide

search. (You do not need to write your own heuristic.)

I found it useful to make a helper function called cost that, given a node, a problem, and a heuristic, returns the estimated cost of the node. Remember that in A* search, the estimated cost of a state includes both the cost to reach the current state, and the heuristically estimated cost of reaching the goal from the state.

When you are trying to determine the next action to take, you may find it useful to break this down into three components: the cost of reaching the current state, the cost of taking the action, and the estimated cost of reaching the goal from the state that you will reach if you take that action.

Part 2: *You Look Like a Thing and I Love You* reflections

Question 1 (10 points)

Pick a game that you enjoy playing and research whether an AI player has been built for it. How well does the AI player do? What kind of techniques does it use? If the AI player has surpassed human performance, when did this happen?

You can pick a board game or a video game.

Question 2 (10 points)

Consider the word game Wordle: `https://www.nytimes.com/games/wordle/index.html`. If you were building an AI program to solve this game, what approach would you take? Discuss whether or not the following approaches could be applied: constraint-solving, uninformed search (BFS/DFS), and A* search.