COMP341 Introduction to Artificial Intelligence
HW2

This programming homework will test your knowledge and your implementation abilities of what you have learned in the adversarial search part of the class. This homework has two parts; a programming part and a written report part. The written report will require you to interpret the output of the programming part.

The programming part of the homeworks will follow the Berkeley CS188 Spring 2014 pacman projects at `http://ai.berkeley.edu/` closely. This homework includes P2, multiagent search.

This homework must be completed individually. Discussion about algorithms and data structures are allowed but group work is not.

However, you might find yourself having trouble implementing the algorithms. I am going allow you to copy or look at someone else's code as long as you credit the person or the website! In this case I will grade you differently, as described in the next part.

Any academic dishonesty, which includes you taking someone else's code but not properly crediting the source, will not be tolerated.

Grading

The code part and the report will have 2:1 weight ratio in your submission (programming 2/3, report 1/3). In order for the report to be graded, corresponding programming parts need to be submitted.

In case you have copied/used someone else's code as stated above, you will only be graded for your report. In this case you will be eligible for up to half the credit, i.e., the report will have 1/2 weight. You should not submit your code in this case **AND** properly credit the source in your report. Reports without proper credit will not be graded.

Finally, we are going to compare your code to previous year's submissions, each other's and other online sources known to us. If your code's similarity level is above a certain threshold, your code will be scrutinized and you might end up getting a 0.

Part 1: Programming

Instructions

You are going to do the 5 programming questions about adversarial search given here: `http://ai.berkeley.edu/multiagent.html`. Feel free to use the forum in the blackboard for general discussion, however remember there is no group work.

You are only required to change *multiAgents.py*. If you have any issues with other parts of the code let your instructor or TA know ASAP, even if you manage to solve your problem. Use the data structures in *util.py* for the autograder to work properly. If you think you have the right answer but the autograder is not giving you any points, try to run it on individual questions. In case that you are sure your algorithm is working properly, but the autograder is not giving you any points do not worry. Let the TA or the instructor know after your submission and we are going to arrange a time for you come in and argue why you think your algorithm is running correctly.

In the previous homework, HW1, you had the option to use the mazeDistance, which gives you the cost to go between 2 locations in the maze taking the walls into account, i.e., the true graph cost. This results in better cost estimates at the expense of more computation. This resulted in lower number of node expansions but potentially made the algorithms run slower than if you used the `manhattanDistance`. For this project, timing will also be considered so use it at your own risk! In this case, you are allowed to submit additinal files (e.g. *search.py* and *searchAgents.py* from HW1).

Hints

There are hints for the programming part, especially about what features to use in your evaluation functions, in the project description website. Read those carefully, as they generally provide good ideas. Note that your evaluation function for question 1 is a good starting point for question 5. As noted in the website, do not waste too much time on both of these questions and only get back to them if you have time left.

Evaluation Functions

The `evaluationFunction` method of the `ReflexAgent` class already shows you how to get important information from the `GameState` class. As an addition, you can use the `getCapsules` method to get the locations of the remaining capsules as a list. These capsules let the pacman eat the ghosts for significant additional points!

You can call the `asList` method of the `Food` class to get the locations of the foods as a list. You can use the built-in `len` function to get the remaining number of foods or capsules.

The `scaredTimes` gives you the remaining time for pacman to eat the ghosts. A 0 value means that the ghosts are not scared so watch out!

You are potentially going to use the distances to the food as a feature (e.g. the distance to the closest food). Your agent, pacman, might get right next to the food and not eat it! This is due to the fact that your evaluation function might return a lower value after eating the food and the next distance to the closest food or the average distance to the foods becomes higher. Make sure your evalation function takes this into account. The same thing might happen with eating the ghosts, pacman might chase the ghost but never actually eat it.

Your pacman can get to a state of *thrashing*, which means that it either stops or keeps moving between two states, until a ghost comes nearby. In your ReflexAgent, you might think of penalizing the *Stop* action. This is not always the best idea. In this case, try to tune your feature weights. Finally, it is okay if you see this behavior as we are not sending a rover to Mars, yet.

Programming for Multiple Ghosts

Remember that in the class we talked about the multiplayer case, where all the agents are trying to maximize their utility. Think of all the ghosts as trying to minimize your evaluation function, i.e., all the ghosts are *MIN* agents. This means that you need to call the *min-value* function for all of the ghosts. You can do it by passing an agent index to your *value* function. Note that pacman will always be agent 0. Look at the comments in the *multiAgents.py* file to see which functions take an agent index (look for `agentIndex`) as a variable. You are free to define your own functions, either externally or within the classes, but make sure the entry points are in the existing class methods (e.g. the `getAction` methods).

If you are having too much trouble with this part, seek the instructor's help.

Keeping Track of Depth

In this assignment, 1-level of depth is interpreted as including all the agents' actions. In the 2-player version, this means 1-level of depth is completed after you check the actions of both the *MAX* agent and the *MIN* agent. If you have more than 1 ghost, the depth increases after you look at the actions of pacman and all the ghosts.

In the class, we have discussed why we are not able to search as deep as the end of the game. The assignment requires you to implement a depth limit. In a recursive algorithm, it might not be obvious how to implement this. There are multiple ways to do this. You can pass down the current depth, along with your state to your functions.

If you are having too much trouble with this part, seek the instructor's help.

AlphaBeta Pruning

The α and β values are not for the entire search tree but for a certain node. This means that you should not keep the same variables for the entire search.

Pacman Not Eating the Last Food

Due to the setup of the environment, you might not be able to force the pacman to eat the last food for the programming question 5. This is because the action sequences *Stop* and *Move*, where the latter is moving in any legal direction, and *Move* and *Stop* will potentially have the same value at the end of the game depending on the location of the ghosts. Since the *Stop* always comes before the *Move* actions when getting the legal actions, your agent might always chose it first. Keep this in mind if your pacman stops right next to a food! A way to solve this is to force pacman to chose the *Move* action in case two actions have the same value.

Part 2: Report

This part includes answering the following questions based on your program's output on the given pacman tests.

You are expected to answer the questions concisely. This means maximum ten sentences per question. Answers that are longer than 200 words will only be graded for the first 200 words or so. It is okay if you over-generalize, as long as your direction is clear and correct. For some questions, you do not even need to write that much (*Hint*: Q2, Q3 and Q4). Note that some of the answers could already be in the provided link to the Berkeley site!

Written Q1:

What are the features you used in your evaluation function for your reflex agent? Why did you chose them? If you have too many, limit yourself to at most 5. Do you think using the reciprocals of some values is a good idea and why?

Written Q2:

Try the following lines of code:

```
python pacman.py -p MinimaxAgent -l trickyClassic -a depth=2 -f
python pacman.py -p AlphaBetaAgent -l trickyClassic -a depth=2 -f
```

Run both them until 20 seconds (or less if pacman ends up dieing) and see how far the pacman has got. You do not need to write additional time keeping code, a stopwatch should suffice.

In your tests, were you able to see any speed difference between the MinimaxAgent and AlphaBetaAgent, between pacman actions? If so, why and if not why not? Is there any situation you came across that highlights this?

Written Q3:

When you were running the tests in the previous question, did your pacman behave exactly in both cases? Why?

Written Q4:

Now try the same with the ExpectimaxAgent;

```
python pacman.py -p ExpectimaxAgent -l trickyClassic -a depth=2 -f
```

Comment on how fast your code runs. Compare it with the MinimaxAgent and AlphaBetaAgent. Note that this comparison is trickier to do. If you are not able to conclusively see anything, write what you would have expected.

Written Q5:

We are sure that you were able to write a better evaluation function than the one we used for the programming questions 2-4. Did you change anything from your evaluation function for the ReflexAgent? If so, what were the changes? What, if anything, is different in this case? If you have written something entirely different, comment on your new evaluation function

Written Q6:

For both the programming questions 1 and 5, you probably needed to tune your feature weights. If so, comment on how you selected your weights, what did you prioritize and why.

Submission

You are going to submit a compressed archive through the blackboard site. The file can have *zip*, *tar* or *tar.gz* format. This should extract to a folder with your student ID which includes *multiAgents.py*, *report.pdf* and other files you might have. If you have included any extra files, these **cannot** have the same names as the ones already provided to you. If this happens and you get a 0, you get a 0! **DO NOT ARCHIVE THE ENTIRE FOLDER.** Let us know if you have any problems.

We are going to use the same autograder that is available to you. We are also going to compare your code to everything that is available to us. If your code's similarity level is above a certain threshold, your code will be scrutinized and you might end up getting a 0.