CS461 Artificial Intelligence
Spring 2022
Homework 2

Due: March 7th, 2022, 23:59 PM

Instructions

- You should submit your python codes with .py format to Moodle. Only modify the specified parts (highlighted by "*** YOUR CODE HERE ***") of the provided files, you will lose the whole grade otherwise. There are other parts of the codes you can (but may not need to) change which are explicitly mentioned in comments.
- You will submit your codes in groups of 3 to 5. Please also include the name and the student ID of every member in a text file named 'members.txt'.
- You will submit only two files: `multiAgents.py`, and `members.txt`. Any other file will be ignored.
- You will need python3.6 installed on your computer to run the environment. Your implementation must not change the package/version requirements. Please note that you might face errors running the scripts if you install other versions of python3.
- Only one member should upload a .zip file including the mentioned python files. The file name should be $< Group - ID >$_HW2.zip. You can see your group ID in the shared Excel file on Moodle.
- The codes will be checked for plagiarism using online tools which are difficult to fool. This check will include publicly available implementations for this homework.
- Your codes will be evaluated in terms of efficiency as well. Make sure you do not have unnecessary loops and obvious inefficient calculations in your code.
- We follow no extension policy. However, the entire group will lose 25 percent of the grade per day of late submission up to 2 days.
- The files have been tested before being uploaded to Moodle. However, if you still have problems using and running the codes, you can contact the course TAs: Sina Barazandeh (sina.barazandeh@bilkent.edu.tr) and Maria Raluca Duminica (raluca.duminica@bilkent.edu.tr).

1 Multi-Agent Pacman

You are now familiar with the Pacman environment from your first assignment. In the first assignment, you implemented algorithms for a single Pacman agent to explore the given environment to find the best path to complete the tasks. However, Pacman is originally a multi-agent game in which Ghosts are independent agents chasing the Pacman agent. In this assignment, you will implement algorithms for the multi-agent version of the Pacman environment. There are 5 questions in this assignment including a reflex agent, 3 adversarial searches, and an evaluation function. Since the questions can be implemented and evaluated separately, it is possible to work on multiple questions at the same time.

1.1 Question 1 - 20 Points

ReflexAgent. In this question you will complete the `ReflexAgent` class in the `multiAgents.py` file. The ReflexAgent should consider the positions of both foods and other agents (Ghosts) to find the best path to avoid being killed by the Ghosts and look for food at the same time. In the default function, you are provided with example codes that get information about the environment. You will use these information to make decisions. Your main objective is to complete the evaluation function, but you are also allowed to change the *getAction* function if you need to.

Grading. Your code will be graded by an autograder. However, since the agent will perform differently in every trial, we will run your code 2N times (on a specific layout). You will get 10 points if your agent wins between N and 2N times, and you will get 20 points if your agent wins all of the games. You can evaluate your implementation by running the following command:

```
python pacman.py -p ReflexAgent -l testClassic
```

Note that your code will be graded using a more complicated cases than the basic test case in all questions. You can run the same test cases using the command below:

```
python autograder.py -q q1
```

1.2 Question 2 - 20 Points

Minimax. The second question is to implement an agent based on the **Minimax** adversarial search algorithm. The implemented minimax algorithm should work with any number of ghost, and your implementation should be able to expand the tree to an arbitrary depth given as an input. A tree of depth K involves K movements from Pacman and each of the Ghosts. You have access to the depth of the tree and also the evaluation function to evaluate each state. As a hint, you can implement your algorithm recursively.

Grading. Your code will be graded by an autograder. This evaluation partly depends on when you call the *GameState.generateSuccessor* function, and you will receive a higher grade if your implementation makes better decisions in this regard. You can evaluate your implementation by running the following command:

```
python autograder.py -q q2
```

1.3 Question 3 - 20 Points

Alpha-Beta Pruning. Complete the *AlphaBetaAgent* class in the *multiAgent.py* file implementing the Alpha-Beta Pruning algorithm. Alpha-Beta Pruning is more efficient than the Minimax algorithm, and you should notice a speed-up using this algorithm comparing to Minimax. You can test the speed of your algorithm using the command below. It should not take more than few seconds for each move:

```
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```

Grading. Your code will be graded by an autograder again. Similar to Question 2, you should use *GameState.generateSuccessor* when necessary. Please note that since the grading includes the check for the right number of states, you should perform alpha-beta pruning without reordering children. You can evaluate your implementation by running the following command:

```
python autograder.py -q q3
```

Note that you should not prune in equality. Although you can normally do prune in equality, the provided autograder is implemented assuming that pruning is not performed in equality.

1.4 Question 4 - 20 Points

Expectimax. Here you should implement another adversarial search agent this time based on the Expectimax algorithm. Unlike Minimax and Alpha-Beta Pruning, Expectimax does not assume opponents with optimal decisions, and therefore performs closer to the real case scenarios. Expectiman is a probabilistic agent suitable against non-optimal adversary agents as it can take risks and end up in better states which the optimal agent would not.

Grading. Your code will be graded by an autograder again, and you can evaluate your implementation by running the following command:

```
python autograder.py -q q4
```

Note that this algorithm, like every other algorithm, fails in some cases. The correct implementation is expected to fail in some cases.

1.5 Question 5 - 20 Points

Evaluation Function. Finally, you should implement a new evaluation function which evaluates the states rather than actions.

Grading. Similar to the first question, your code will be graded by an autograder and we will run your code multiple times:

- You will receive 4 points if your agent wins at least once.
- You will receive 3 points for winning half of the times, and 6 points for winning all the time
- You will receive 3 points for an average score of 500 or higher and 6 points for an average score of 1000 or higher
- You will receive 4 points if your algorithm runs in less than 30 seconds when you run it in the `--no-graphics` mode [1]. (The codes will be tested on a server to ensure the equality)
- Note that you should win at least half of the times to receive the score for the third and forth cases.

You can evaluate your implementation by running the command below:

```
python autograder.py -q q5
```

Run the following command to run in the `--no-graphics` mode:

```
python autograder.py -q q5 --no-graphics
```

2 Grading

Although your homeworks will be graded using an autograder with the mentioned details, your implementations will also be checked. In any case, only the correct implementations will be accepted.

References

1. Berkeley project web page https://inst.eecs.berkeley.edu/~cs188/sp22/project2/