CS461 Artificial Intelligence
Homework 2

Spring 2023

Due: April 5, 2023, 23:59

Instructions

- You will submit your Python codes with.py format to Moodle. Only modify the specified parts (highlighted by `*** YOUR CODE HERE ***` comments) 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 the comments.
- You will submit your codes individually or in groups of two. Please also include the name and the student ID of each 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 `Python 3.6` 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 work on other versions of Python.
- Only one member should upload a .zip file including the mentioned python files.
- 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. Submit your own work.
- Your codes will also be evaluated in terms of efficiency. Make sure that you do not have unnecessary loops and obvious inefficient calculations in your code.
- We follow no extension policy. However, you will lose 25% 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: Barış Bilgin Şenol (bilgin.senol@bilkent.edu.tr) and Navid Ghamari (navid.ghamari@bilkent.edu.tr).

0 Python Environment Setup

This section describes the Python environment that will be used during the grading of your homework. You do **not** have to follow the same setup as long as you meet the requirements listed in the instructions and get proper grades. However, it is recommended to follow the same setup to avoid inconsistencies that you may encounter while working on the assignment.

The Python environment will be managed by `conda`. If you have not used it before, take this as a chance to get used to that good practice.

1. Download and install Anaconda [1]. Miniconda is also fine if you prefer a minimal installation [2].
2. Search for the `Anaconda Prompt` on your computer. It will launch a command line interface with `base` environment already activated.
3. Create a new `Python 3.6` environment by entering the following command on the Anaconda Prompt. This will create a new `conda` environment named `cs461-hw2` and install `Python 3.6` along with some base packages. Refer to [3] for further information on Conda basics and managing environments.

   ```
       conda create --name cs461-hw2 python=3.6
   ```

4. Confirm the prompt "Proceed ([y]/n)?" by pressing enter.

5. Enter the following command to the Anaconda Prompt. This will activate the environment you just created. You should see the environment name `cs461-hw2` in parentheses at the beginning of the line.

   ```
   conda activate cs461-hw2
   ```

6. Enter the following command to verify the Python version.

   ```
   python --version
   ```

7. You should see an output similar to the one below:

   ```
   Python 3.6.9 :: Intel Corporation
   ```

8. That is it! Run the necessary checks shown under each question in the assignment within this environment. Also, don't forget to select your interpreter as this environment in your code editor of choice.

1 Multi-Agent Pacman

You are now familiar with the Pacman environment from your first assignment [4]. 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 five questions in this assignment including a reflex agent, three 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.

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 the foods and the other agents (Ghosts) to find the best path to avoid being killed by Ghosts and looking 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 this 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 the games. You can evaluate your implementation by running the following command in your active Python environment:

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

Note that your code will be graded using 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
```

Question 2 [20 Points]

Minimax. The second question is to implement an agent based on the **Minimax** adversarial search algorithm. You will complete the `MinimaxAgent` class in the `multiAgent.py` file. 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 of Pacman and each of the Ghosts. You have access to the depth of the tree and also to 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
```

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 compared 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 again by an autograder. You should use `GameState.generateSuccessor` when necessary, as you did in Question 2. 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 prune in equality, the provided autograder is implemented assuming that pruning is not performed in equality.

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. Expectimax 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. You will complete the `ExpectimaxAgent` class in the `multiAgent.py` file.

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

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

Please note that this algorithm fails in some cases. The correct implementation is expected to fail in some cases.

Question 5 [20 Points]

Evaluation Function. Finally, you should implement a new evaluation function which evaluates the states rather than actions. You will implement the `betterEvaluationFunction` function in the `multiAgent.py` file.

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. (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 homework 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] Anaconda web page. `https://www.anaconda.com/`. Accessed: 2023-03-21.

[2] Miniconda documentation. `https://docs.conda.io/en/latest/miniconda.html`. Accessed: 2023-03-21.

[3] Conda cheat sheet. `https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf`. Accessed: 2023-03-21.

[4] Berkeley project web page. `https://inst.eecs.berkeley.edu/~cs188/sp23/projects/proj3/`. Accessed: 2023-03-21.