Artificial Intelligence
Homework 2
15 points
Game Playing and Minimax
Due Friday, October 6, 11:59:59 PM

I. Introduction

The goal of this assignment is to create an agent that can accurately play Othello using a minimax strategy with alpha-beta pruning. These topics were covered in detail in lecture, and throughout the assignment you should refer back to the corresponding notes in lecture slide 824. The book is also a useful resource.

II. Environment setup

We will take the files in `~/hidden/<YOUR_PIN>/Homework2` as your submission for this assignment, so we recommend that you work remotely on the Zoo using an FTP-compatible text editor like Sublime. If you need help setting up the Sublime FTP plugin, contact the course ULAs.

You can create a copy of the project template with the following command:

```
$ cp -r /c/cs570/assignments/Homework2 ~/hidden/<YOUR_PIN>/
```

You can find several Othello agents in `~/hidden/<YOUR_PIN>/Homework2/engines`. These agents can already play against each other. You can start an Othello game from the `~/hidden/<YOUR_PIN>/Homework2` directory as follows:

```
$ python othello.py random1 greedy
```

This will run random1 against greedy and show you the final board and score. You can also use the `-v` option for more information about the simulated game:

```
$ python othello.py greedy engine1 -v | more
```

To play against an AI agent, you can use the human agent, which will allow you to input moves:

```
$ python othello.py human greedy
```

The `engines` directory also contains a student agent in `student.py`. This file is, by default, a copy of the greedy agent. Throughout this assignment you will be replacing this with your own implementation of minimax search.

III. Assignment part a: a minimax Othello player (40 points)

Using the starter code of the student engine, implement **a player for the Othello game that uses the minimax search algorithm** to select its move.

You will implement the `get_minimax_move` method of `StudentEngine` in `student.py`, which should return a coordinate pair (x, y) that indicates the move that your agent chooses to make. You can implement any helper function you want in the `StudentEngine` class but you shouldn't modify the skeleton of the `get_move` function.

Your agent should be able to beat the random agent consistently. It should also be efficient enough to select its move in a reasonably short amount of time. Document your heuristics and algorithmic choices in your report.

This means that you may have to find a compromise: an algorithm that perhaps doesn't always find the best possible outcome, but that makes a good decision in a reasonable amount of time while using a reasonable amount of memory (you can enforce a cutoff at a fixed depth by defining a class attribute in the `StudentEngine` class).

In writing your Othello player, you may find useful to examine and understand some methods of the `Board` class located in `board.py`. We especially suggest that you take a look at:

- `count` – get the number of pieces for a specified color
- `get_legal_moves` – return a list of all valid moves for the player whose turn it is

You're free to reuse these functions in your code.

Also, note that there are several research papers and web sites that discuss possible heuristics. You're welcome to make use of these as long as they are properly cited in your report. Using any external code is strictly prohibited and can result in a grade of 0 on this assignment.

IV. Assignment part b: an alpha-beta Othello player (40 points)

Now implement `get_ab_minimax_move`. Like `get_minimax_move`, this function should implement the minimax search algorithm, but it should also use **alpha-beta pruning** during search. Alpha-beta pruning makes use of two additional values α (your

best score by any path) and β (the best score of the opponent by any path). Since the utility of any viable path will fall between α and β, it is possible to stop following a branch whenever a value less than α or greater than β is found. For a more detailed discussion of alpha-beta pruning, review lecture slide 824.

To test your implementation of alpha-beta pruning, you can use the `-aB` and `-aW` options to turn alpha-beta pruning on for black (team 1) and white (team 2) respectively, e.g.:

```
$ python othello.py -aB student greedy
```

The above command will simulate the student agent with alpha-beta pruning enabled (i.e. it will call `get_ab_minimax_move` instead of `get_minimax_move`) playing against the greedy agent.

VI. Comparing your implementations (20 points)

Design and conduct some experiments to determine the following statistics on the search process, for both the minimax and alpha-beta players:

- the total number of nodes generated (5 points)
- the number of nodes containing states that were generated previously, i.e. duplicated nodes (5 points)
- the average branching factor of the search tree (5 points)
- the runtime of the algorithm to explore the tree up to a depth of *D*, for different values of *D* (5 points)

Provide a short description of the experiments you performed in the text file `~/hidden/<YOUR_PIN>/README.txt` and discuss your results.

VII. Tournament (bonus points)

A class tournament will give you a chance to test your code against that of other students.

It will consist of several group stages, in which each student will meet all other contestants of his/her group. The initial format will be 16 groups of 16 players, including some players provided by the course staff. The matches will be timed, so that each player will have 30 seconds total of CPU time to use during each game. The less efficient programs will be eliminated after each stage and new groups will be formed consisting of the top 3 agents from the previous groups. A final group stage with the top performers will then determine the champion.

A second tournament will also be conducted, in which your agents will be given less time to select their moves. This will help us to distinguish the most efficient implementations of the algorithms.

It is possible to use `othello.py` to test how your agent will perform against another student agent:

```
$ python othello.py student1 student2
```

We will give discretionary bonus points depending on your performance in the tournaments. The authors of the three top-ranked programs will also be invited to say a few words in class about the secrets of their implementations.

VIII. Submit your assignment

For this and all other assignments, you will submit by making sure all required files are in the appropriate homework folder in your hidden folder on Zoo. It is fine to have other files in the directory as well. We use the timestamp of the required files on Zoo to determine if you submitted on time, so please do not modify these files after the assignment deadline. Please note that we will only grade files on Zoo. For this assignment, please turn in the following files in `~/hidden/<YOUR_PIN>/Homework2`:

- `engines/student.py`
- `README.txt` (this file should not be in the engines subdirectory)

IX. Grading criteria

This assignment will be scored out of a total of 15 points:

- 6 points for part a
- 6 points for part b
- 3 points for written comparison

Bonus points will also be awarded on a discretionary basis for good performance in the class tournament.

X. Late policy

10% off for each day or part thereof.

After three full days, the assignment will be given a score of zero.