Artificial Intelligence CPSC 470/570
PS 2
16 points
Game Playing and Minimax
Due Monday, Feb. 11th, 11:59:59 PM

Some reminders:

- **Grading contact:** Sahib Grover (sahib.grover@yale.edu) is the point of contact for initial questions about grading for this problem set.
- **Late assignments** are not accepted without a Dean's excuse.
- **Collaboration policy:** You are encouraged to discuss assignments with the course staff and with other students. However, you are required to implement and write any assignment on your own. This includes both pencil-and-paper and coding exercises. You are not permitted to copy, in whole or in part, any written assignment or program as part of this course. You are not to take code from any online repository or web source. You will not allow your own work to be copied. Homework assignments are your individual responsibility, and plagiarism will not be tolerated.
- **Students taking CPSC570:** There is no extra section for this assignment. Your assignment is the same as CPSC470.

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 lectures #5 and #6, and are also covered in detail in Chapter 5 of your textbook.

II. Environment setup

Once you get the starter code, you can run this command:

```
$ python3 othello.py random1 greedy
```

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

```
$ python3 othello.py greedy random1 -v | more
```

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

```
$ python3 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 (6 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 moves in a reasonably short amount of time. You will be asked to 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 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 and possible disciplinary action.

IV. Assignment part b: an alpha-beta Othello player (6 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.

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.:

```
$ python3 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 (4 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 (1 point)
- the number of nodes containing states that were generated previously, i.e. duplicated nodes (1 point)
- the average branching factor of the search tree (1 point)
- the runtime of the algorithm to explore the tree up to a depth of *D*, for different values of *D* (1 point)

Provide a short description of the experiments you performed in the text file `[netid]-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, 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:

```
$ python3 othello.py student1 student2
```

(You are permitted to test your code against your classmates, but please remember that looking at their code is forbidden!). We will give discretionary bonus points depending on your performance in the tournaments.

VIII. Submit your assignment

Please zip the following two files:

- `student.py: When you turn this in, make sure you rename student.py to` **`[netid].py`** `For example, if your netid is abc123, then rename the file to abc123.py .`
- `[netid]-README.txt`

Please rename the zipped file to: **[netid]-ps2.zip**

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
- 4 points for written comparison

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