Othello, Part 2 – Adding AI

Eckel, TJHSST AI1, Fall 2020

Background & Explanation

The time has come to add AI to Othello and see how well your code can do! This sheet contains two assignments; each is worth 10 points. They can be done in either order, though it's probably easier to do the Othello server first.

Here's how this works. Your code **is not running the whole game**. There is moderator code that runs the game, and runs two different strategies to get the moves for the two players. Each move occurs when the moderator code runs one player's strategy, sending it the current board and token. After a certain number of seconds, the moderator code will **kill the process** running the strategy for that player. It will then take the most recently decided move and use that to play.

There are descriptions on the course website for how to get information from / send moves to the server, and for how to do so with my grading scripts. You'll want to read up on both of them at some point, but for now, regardless of the particular input/output protocol, here's what your code needs to do:

1) Accept a board state when your code is run. (This will be accomplished differently for your server submission and your submission to my grader, but either way – your code gets the current board. It doesn't play a game from the beginning.)
2) Run an iterative deepening minimax (or I suppose negamax) from there. Ie, it will look one move in the future and determine the best move from that information. Then, start over, look two moves into the future, and determine the best move from that information. Etc, etc.
3) When each iterative deepening call reaches its max depth, you'll need to return a score for that board state. As the code won't be able to search all the way to the end, you will need to find a way to score mid-game states based on how good they seem to look. More advice about this is on the next page!
4) Continue the iterative deepening process indefinitely! Your code doesn't need to stop running. It just needs to keep printing/setting the best answer it has been able to find so far, then running the next search, until the moderator code kills the process.
5) Then, on the next turn, your code will be run once again and receive a new board. Each turn is a **separate run of your code**, so no data can be kept between runs.

You're only **required** to do a few things to get credit for your AI.

- You must implement alpha-beta pruning.
- You must have an AI that modifies its strategy in some way as the game progresses (see the next page).
- For credit on the first assignment, you must beat two particular strategies on the Othello server in a game with at least a 2 second time limit, take screenshots, and submit them to me.
- For credit on the second assignment, you must submit your code to me, and it must take 75% of the tokens against a random player across 10 games with a 2-second time limit.

However, there's a lot you can do to improve an Othello AI. OW is available for trying more advanced techniques (whether they improve your AI's performance or not) or for just making a better AI (taking 90% against random).

Advice for a good AI

The minimax/negamax process would already be perfect if we could see all the way to the end, like in Tic-Tac-Toe, but we can't. So, the quality of your AI here is determined by two things:

1) How far into the future it is able to look
2) How good it is at determining whether any given mid-game state is favorable or unfavorable

For #1, the answer is alpha-beta pruning. **You are required to implement alpha-beta pruning** to speed up your code's processing of the possibility tree. (Believe me, if you weren't, you wouldn't succeed anyway!)

For #2, the answer is a little more complicated. Here are the general principles:

- An obvious answer, you might think, is "whoever has the most tokens is probably winning right now". This is absolutely not true, at least not early in the game. Play a game of Othello online against a good AI opponent and you'll see what I mean; it's quite likely you have a lot of tokens for the first two thirds of the game, and then at the end the situation shifts and you lose in the last few moves.
- Instead, it turns out that early on in the game the most important factor is **mobility**. A good AI for Othello will restrict its opponent's options, so on its own turns it has more available moves than its opponent. A simple measure of (# of available moves for black) – (# of available moves for white), perhaps multiplied by a coefficient, makes a good early game AI. Black strictly prefers more positive values; white, more negative.
- At a certain point, the game will near the corners. **Corners are very important locations because they can't be recaptured**. So, your AI will probably want to add some number to the score for each corner captured by black and subtract for each corner captured by white. (Again, black prefers more positive values, white more negative.)
    - Similarly, the three squares adjacent to each corner are bad locations to move, because they allow the opponent to potentially take a corner; perhaps add that to your mid-game score (more positive if WHITE takes corner-adjacent squares, more negative if BLACK does.)
    - Feel free to extend this logic to other squares that seem like good captures (edges?)
- Near the end of the game, it becomes plausible for your A/B pruning to reach the end of the game. If your search reaches the end of the game, it can definitively score whether that board is victory or defeat! It is **extremely important** that the numerical score representing victory or defeat should be **much** larger in either direction – positive or negative – than any possible score from your mid-game guessing. You never want to be in a situation where, numerically, your code thinks that any estimated score of a mid-game board looks like a better option than a guaranteed victory.
    - For the assignment requiring you to take 75% of the tokens against random, you might also want to modify the victory number by the total amount of tokens taken at the end of the game (ie, victory for black is 1000000 + how many total black tokens – how many total white tokens, something like that) so that your code chooses the more overwhelming victories when given the chance.

You need to decide how to numerically weight all of these ideas – mobility, value of certain spaces, etc – against each other. Feel free to experiment and submit to the Othello server as many times as you like!

Required Assignment: Beat PW1 and PW2

Go to othello.tjhsst.edu. Read the submission instructions. Also, read the explanation on the course website for submitting to the grader.

You must beat the AI players PW1 and PW2. These AI players were coded by Dr. White a couple of years ago specifically to be strategies that could be overcome with good use of the standard techniques, but not beatable with slow or incorrect implementations. You must beat **both** AIs, in games with at least a 2-second time limit.

When you beat each AI, take a screenshot. Make sure your screenshot shows the time limit. You need both screenshots for credit. You only have to beat each AI once.

Specification

Submit both screenshots to the link on the course website.

This assignment is **complete** if:

- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order.
- You submit **both** screenshots, showing victories against both AI players.
- The screenshots **clearly show** that the time limit was set to 2 seconds or higher.

For **resubmission**:

- Complete the specification correctly.

The Tournament

After first semester is over, all the strategies on the Othello server will be entered into a tournament to see whose is the best. This is purely for fun. If you do not wish to participate, please let me know and we'll delete your submission before the tournament runs!

Required Assignment: Take Most of the Tokens in Several Games Against Random

This is the part you submit to me, and where I check all of your requirements. Read the explanation on the course website for submitting your code to me. **Note you can't just submit the same code as the code you sent to the server**; please make sure you read up on the requirements.

To get credit here, you need to capture the majority of the total tokens in a series of several games against random. It's quite possible the AI you used for part 1 will do this just fine; it's also possible you'll need to modify it. (For example, now all victories aren't equal – victories with more pieces captured are better. How does that change your late game minimax implementation?)

This is a little harder to test on your own, but in general if you're winning big victories against a random player on the Othello server, you're probably in pretty good shape. Just use that strategy but swap the output to the specification above and submit.

Specification

Submit a single Python script to the link on the course website.

This assignment is **complete** if:

- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order.
- After ten games against a totally random opponent, five going first and five going second, you have captured 75% of the total tokens or more. (Any leftover blank spaces in a round do not count towards the percentage.)
- **IMPORTANT:** Your code contains the following two comments:
    - The word "`PRUNING`", in all capital letters, must be in a comment that identifies where your code implements Alpha/Beta Pruning, so I can see you have done so.
    - The word "`CHANGES`", in all capital letters, must be in a comment that explains where your code's heuristic changes and when in the game that change is triggered. Explain what change is made. Place the comment near the code where the change happens.

For **resubmission**:

- Complete the specification correctly.

Specification for Outstanding Work: Completely Destroy Random

You will automatically get Outstanding Work credit if your code takes more than 90% of the tokens in its series of games against a random player. You do not need to do a separate submission to a separate link.

Specification for Outstanding Work: Add Advanced AI Strategies

Submit a single Python script to the link on the course website.

This assignment is **complete** if:

- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order.
- Your code implements **two or more** of the following advanced techniques:
    - An opening book
    - Negascout
    - Some kind of measure of piece stability / edge building
    - Bit boards
    - Use of the time hoarding option available on the server
    - Any other strategy you can find, as long as you inform me in advance and I agree
- **IMPORTANT:** Your code contains a separate comment containing "`ADVANCED`" in all capital letters next to the implementation of *each* of the above strategies, briefly explaining how it works. I reserve the right to ask for further explanation if I don't understand your writing.
- Finally, you do not plagiarize. (I restate this here because it's more tempting than usual for this assignment.) To be specific, you **can** do these things:
    - Find internet videos, tutorials, explanations, or how-tos about any of the above strategies
    - Read pseudocode / explanations in words for how to apply those strategies to Othello

  You **cannot** do this:
    - Find a Python implementation of one of those techniques and copy it, either directly or by rewriting it yourself

For **resubmission**:

- Complete the specification correctly.

Specification for Outstanding Work: Ultimate Tic-Tac-Toe

Go to http://bejofo.net/ttt and play a few games; learn how the game Ultimate Tic-Tac-Toe works.

Implement the game. You figure out how the board is displayed and how the user interface works, but make it not be terrible, please. When I run your code, I should see a brief explanation of how to choose move locations before the game begins.

Then: write an AI that beats me when I play against it.

It's that simple.

(Note: I am not very good at this game; this is not as intimidating as it sounds.)

Submit a single Python script to the link on the course website.