Turn-Based AI with Othello, Part 1: Modeling

Eckel, TJHSST AI1, Fall 2020

Background & Explanation

The culminating assignment in gaming AI will be a tournament that pits our various Othello artificial intelligences against each other. This will be awesome. But of course, before we get there, we need to effectively model and play Othello! The first part of this assignment is just to get all the internal modeling working and play a game pitting two players against each other that always choose random moves.

A good primer on the Othello rules is here: https://www.wikihow.com/Play-Othello

I'd also recommend googling an Othello game you can play online and playing a couple of rounds against an AI.

Some important notes:

- The board is always 8x8.
- Black always moves first.
- All eight directions are available for flipping tiles – up, down, left, right, and all four diagonals.
- If a tile traps enemy tiles in two or more different directions at once, they are all captured.
- If someone can't make a valid move, play passes back to the other player. If neither player can move, the game is over.
- A player cannot intentionally pass if a legal move is available. A player must make a move.

Eventually, when we're submitting these to the tournament server, the following conventions will apply. You do NOT have to store your board internally in this way! If something else makes more sense to you, that's fine! You just have to read in input and send out output according to these conventions, so if you do something else, you'll need to write code to convert back and forth.

On the server, the board will be conceptualized as a length-64 string – an 8x8 grid. Empty squares are ".", Black is "x" and White is "o". This 8x8 grid is stored as length-64 string in row major order.

The initial starting configuration looks like **THIS** in the middle**:**

```
ox
xo
```

On the server, locations will be referenced by integer index on this string. Once we get to this part of the assignment, you will communicate the move you select to the server using this integer index. Again, it is not required that you track indices this way internally, but that'll be the goal eventually.

First: Model Othello

1. Write a possibleMoves() function that takes two arguments – a board and a token – and returns a list of all possible squares that can be played into by that token. This is harder than you think. I strongly recommend working out a plan for this on paper before you code. In in-person years, I require that students discuss and write a pseudocode plan in partners – that's a good idea here as well. Bare minimum, you should create some board states on paper, figure out the available moves by hand, and make sure your code matches. This might seem excessive, but if something doesn't work weeks from now and it turns out to be a problem with your first method, you'll feel pretty silly, and this is not an uncommon occurrence on this lab!

2. Write a move() function that takes three arguments – a board, a token, and a valid move position – and returns a board where the token has been played to the indicated position, including attendant token flips. This method does not need to account for invalid input. You should also test this thoroughly. For instance, make sure that blank spaces between the new move and a matching token remain blank.

Get Your Code Ready to Turn In

Use your above functions to create a script that works as follows:

1. Accept two command-line inputs: a 64-character puzzle string formatted as given, and then a 1-character string that is either "x" or "o", representing the player that will move next.
2. Output a list of available moves. Just print the list of integers. Make sure the list is in sorted ascending numerical order or my grader won't read it correctly.
3. For each move, in ascending numerical order, make that move and print the resulting board as a 64-character string, one board per line.
4. Do not print anything else at any point, including extra newlines.

Sample Run

Here's an example to clarify my input/output requirements.

```
> othello_model.py ...........o.......o.......oo....ooooo...xx.x................... x
[20, 21, 24, 25, 26, 30]
...........o.......ox......xx....oxoxo...xx.x...................
...........o.......o.x.....ox....ooxoo...xx.x...................
...........o.......o....x..oo....xoooo...xx.x...................
...........o.......o.....x.oo....xoooo...xx.x...................
...........o.......o......xoo....oxxoo...xx.x...................
...........o.......o.......oo.x..oooox...xx.x...................
```

THIS IS NOT, BY ITSELF, SUFFICIENT TO TEST YOUR CODE; there are so many different special cases in Othello!

- Most notably, a move often flips tiles in two different directions at once. There are two moves like this in the test case above, at indices 20 and 26, so those are good ones to double check. But you'll want to test situations like that in multiple directions.
- You'll also want to test that both "x" and "o" work, of course.
- Finally, the other problem I often see is that the checks sideways/diagonally sometimes cross from one row to the next. If you have this situation, on two separate lines:

  ```
  .......o
  ox.....o
  ```

  …you want to make sure that it won't accidentally loop across lines and make this move for x:

  ```
  ......xx
  xx.....o
  ```

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.
- Your code does all of the following:
  - Follow the numbered lists above, precisely as written.

For **resubmission**:

- Complete the specification correctly.