# Othello

## Instructions

Open `index.html` or visit [http://annlee.li/othello/](http://annlee.li/othello/) in any browser to play the game.

The game has 3 modes:

- Player vs Player
- Player vs AI
- AI (random) vs AI (minimax)

The game is written in JavaScript and HTML/CSS.

## Methodology

### Search Engine & Pruning

This AI uses minimax search with alpha-beta pruning to discover the best moves.

It is implemented in the function `minimax()` which is a recursive function that passes along the current `node` (state & action), `depth`, `eval` function, if it is a `max` node, current `player` (`player == true` when dark player), and the `alpha`, `beta` values.

It checks for `depth == 0` and for terminal state. Terminal state, `isTerminal()`, is determined by if there are no more moves for the current player and if there are no moves for the other player on the same board. Number of moves is determined by the number of successors returned by the `successor()` function.

It also checks if there are no moves for the current player (but not yet game over). At this point, there is only 1 child in the tree, which is the current board passed with `max` and `player` alternated.

The max and min nodes are found by looking at each successor child node and calling `minimax()` on it. If `maxVal >= beta` or `minVal <= alpha` then it will be pruned.

### Heuristics & Evaluation Function

Weighted squares, mobility and stability are the heuristics used for the AI. The evaluation function used is `evalWithMobilityAndStability()`

#### Weighted Squares

I used this table of values to calculate the weights of each piece.

```
[[12, -5, 3, 2, 2, 3, -5,12],
 [-5,-40,-5,-5,-5,-5,-40,-5],
 [ 3, -5,15, 3, 3,15, -5, 3],
 [ 2, -5, 3, 3, 3, 3, -5, 2],
 [ 2, -5, 3, 3, 3, 3, -5, 2],
 [ 3, -5,15, 3, 3,15, -5, 3],
 [-5,-40,-5,-5,-5,-5,-40,-5],
 [12, -5, 3, 2, 2, 3, -5,12]]
```

This table was constructed by reading Othello strategy guides and consultation with some online tables. It is based on the theory that corners are best, while x-squares (-40) are really bad, c-squares (-5) are bad, and the other ones are not too bad and middle ones are neutral. The values were then tweaked after running simulations against a random AI.

The final score is the value of your pieces minus the value of your opponents pieces.

### Mobility

Mobility is also an important heuristic to take into account. Mobility is calculated by how many potential moves you will have minus the potential moves by your opponent.

```
var mobility = successor(node, player).length;
var oppMobility = successor(node, !player).length;
```

### Stability

In Othello, stable pieces are pieces that can not be flipped. These pieces are very important. This program takes into account corner pieces, full columns, full rows and full diagonals. This AI weighs stability fairly heavily.

### Evaluation Function

The values from the above heuristics are then summed and weighted as below:

```
return 10* (score - oppScore) + 5*(mobility - oppMobility) + 40*(stable -
oppStable);
```

The values for the weigths were tweaked by running simulations against a random AI. It has an approximately 62% win rate against the random AI.

## Look Ahead Steps

This program uses 5 lookahead steps if there are more than 7 blank spaces on the board. Otherwise it will explore to the end of the game tree (infinity).

## Databases

This program does not use any beginning or end game databases.