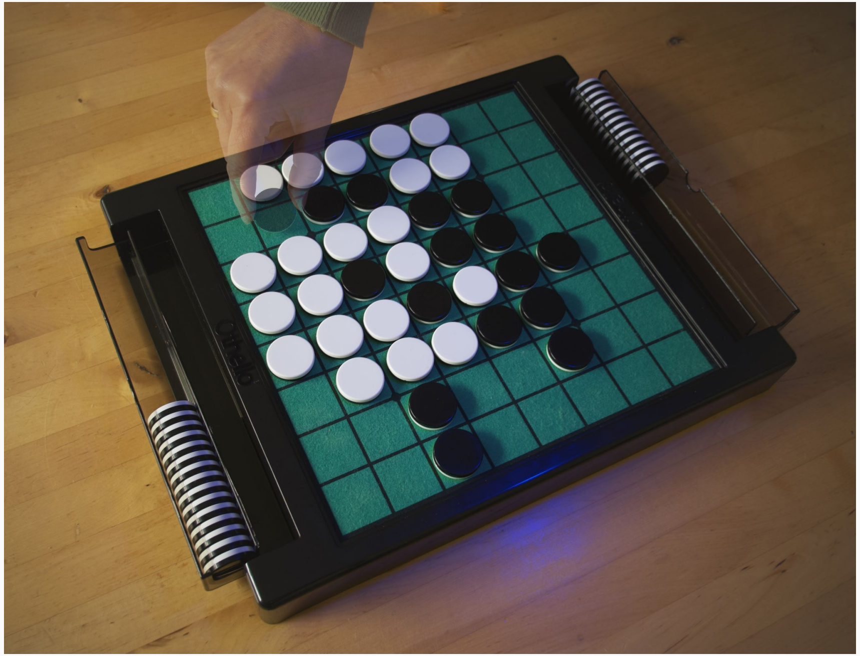
AI531 Final Project

By Mykyta Synytsia and Bach Xuan Phan

Table of Contents

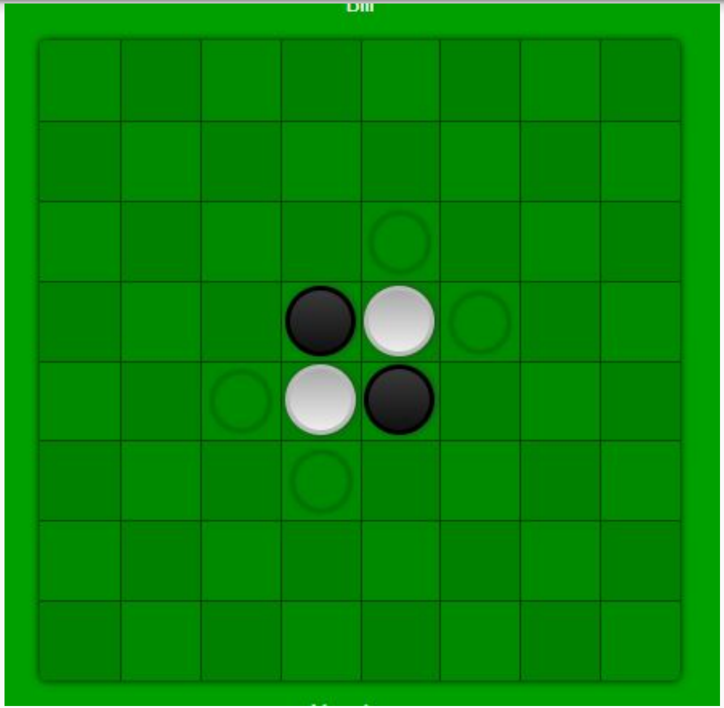
What is Reversi?

- A turn-based two-player game
- 8x8 grid
- Easy to learn but difficult to master
- Tree ~ 10^60 nodes



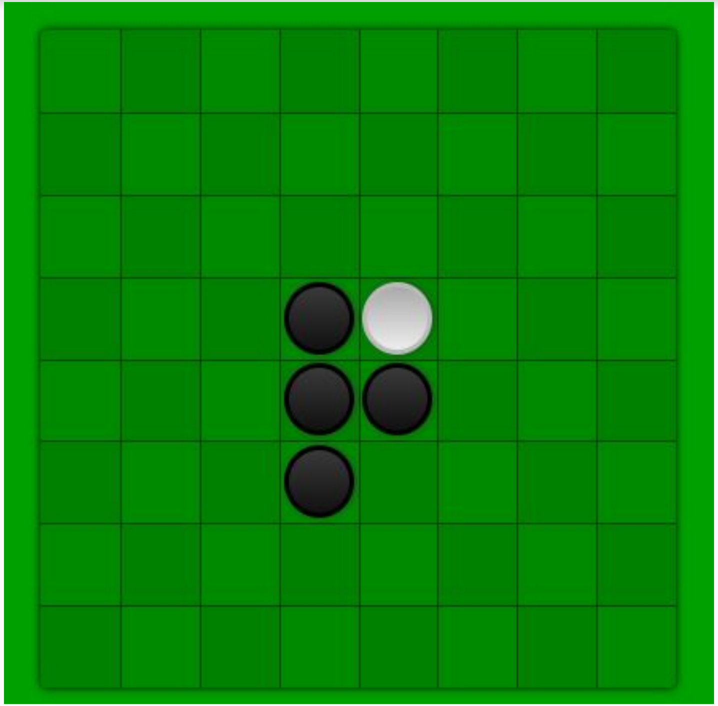
How to play Reversi?

- Initially, the four center cells are filled with four diagonally adjacent black and white discs
- Black goes first



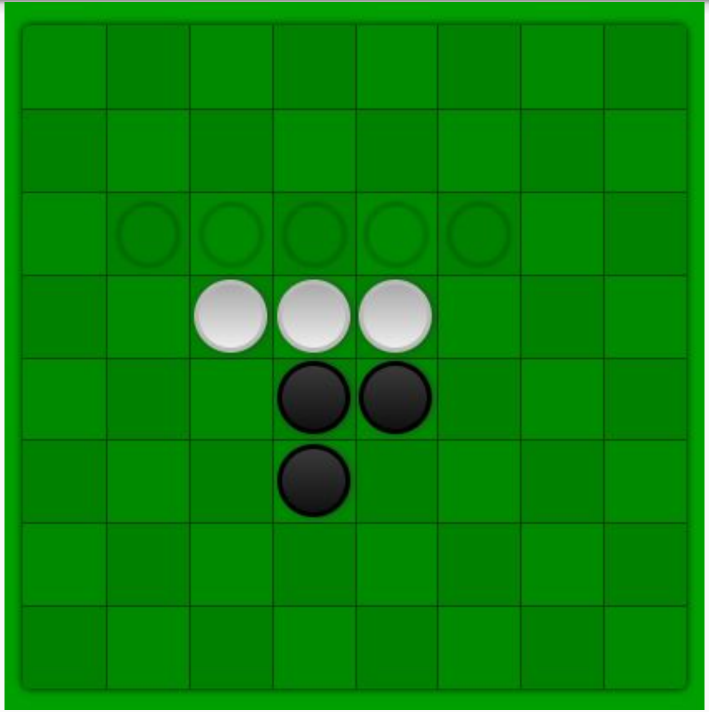
How to play Reversi?

- Players take turns placing discs



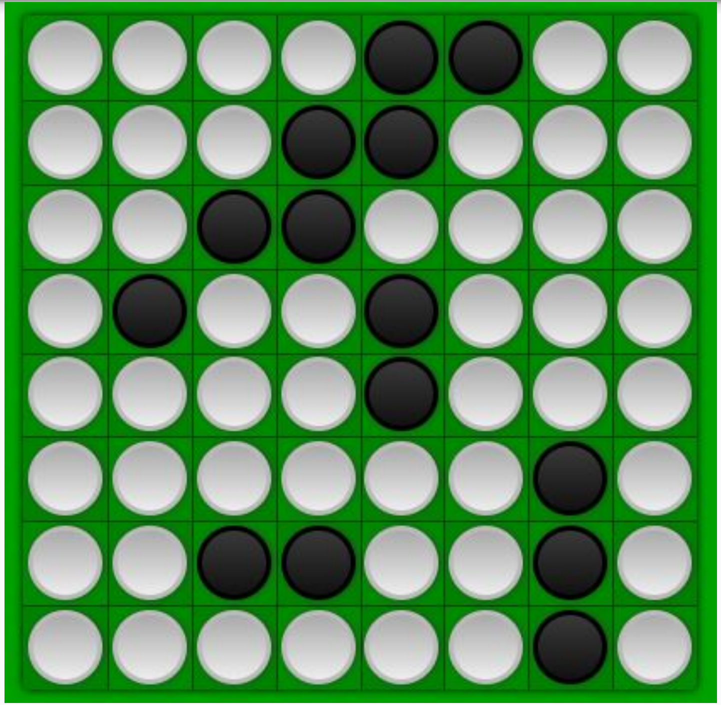
How to play Reversi?

- For a disc to be placed legally, it must trap the opponent's discs vertically, horizontally or diagonally which results in those discs being flipped



How to play Reversi?

- Once the grid is filled, or neither player can move, the player with the most discs wins.



Fair comparison of MCTS and Minimax

To choose appropriate depths for minimax and iterations for MCTS we approximate the number of nodes explored for an 8x8 board per move

- Since the average branching factor is 10, a good approximation of the number of nodes explored for depth limited minimax would be 10^L, where L is the depth
- In the case of MCTS, 60*I, where I is the number of iterations

Fair comparison of MCTS and Minimax

- From these approximations we chose depths and iterations that result in fairly equivalent node exploration numbers:
    - Depths of 3, 4, 5 and 6 => 10^3, 10^4, 10^5, and 10^6
    - Iterations of 10, 100, 1000, 10000 => 600, 6000, 60000, 600000
- After every game, swap players

Minimax evaluation function

$$score(n) = w_{pos} * \left(\sum n.discs(p) * w_b - \sum n.discs(o) * w_b\right) + w_{mob} * (n.moves(p) - n.moves(o))$$

- Positional Disc Score: Difference between the sum of the current player's discs and the opponent's discs
- Mobility Score: Difference in the number of possible moves between the current player and the opponent

$$score(n) = w_{pos} * \left(\sum n.discs(p) * w_b - \sum n.discs(o) * w_b\right) + w_{mob} * (n.moves(p) - n.moves(o))$$

- Where n is the current node and p and o correspond to player and opponent, respectively.
- w_{pos} and w_{mob} correspond to a positional weight and mobility weight, we used a constant weight of 10 and 5.
- n.discs(x) returns a matrix where indices are 1 for locations where player x’s disks are present, 0 otherwise.
- W_b is a matrix with weights for each disc position.

$$score(n) = w_{pos} * \left(\sum n.discs(p) * w_b - \sum n.discs(o) * w_b\right) + w_{mob} * (n.moves(p) - n.moves(o))$$

- The number of valid moves in the current state for player x is designated as n.moves(x).
- With this heuristic evaluation function, scores are greater for the player when they have more discs on the board than the opponent and/or when the number of possible moves is greater.

MCTS selection policy

$$UCB1(n) = \frac{wins(n)}{total\_games(n)} + C * \sqrt{\frac{\log(\mathrm{visits}(\mathrm{n.parent}))}{visits(n)}}$$

Exploitation

Exploration

$$UCB1(n) = \frac{wins(n)}{total\_games(n)} + C * \sqrt{\frac{\log(\text{visits}(\text{n.parent}))}{visits(n)}}$$

Exploitation Exploration

- Upper confidence bound
- UCB1 selection policy for node n, balances exploration (exploring states which were visited less) and exploitation (exploring states which have performed well before).
- Helps to evaluate the utility of node n based on exploitation, win rate of node n, and exploration

$$UCB1(n) = \frac{wins(n)}{total\_games(n)} + C * \sqrt{\frac{\log(\text{visits}(\text{n.parent}))}{visits(n)}}$$

Exploitation Exploration

- Along comes a balancing constant, C, a popular value for which is the square root of 2.
- The exploitation factor rises as the win rate increases, and when a node was not explored very much in relation to the parent, the exploration factor increases.
- This selection policy prioritizes resources to actions which statistically lead to better outcomes.

Challenges: Time

- Single thread is impossible because it took too long.
- Python “threading” has Global Interpreter Lock, preventing multithread aside from I/O bound tasks => “multiprocessing”
- Simulations took a very long time. 1 preliminary test with just 8x8 took 48 hours. => Implement stop_early

Challenges: Multiprocessing Bug

- AttributeError: Python could not find WEIGHTS in MinimaxPlayer
- The child processes (used by starmap) won’t inherit that WEIGHTS value
- But WEIGHTS is a static variable in MinimaxPlayer, that should mean it sticks through new processes

Challenges: Multiprocessing Bug

Behavior	**Linux/Unix (fork)**	**Windows (spawn)**
Memory sharing	Child process inherits the **parent's memory** (Copy-on-Write).	Child starts **fresh**, no memory inheritance.

Results

To evaluate the algorithms we focused on:

- Average total duration of searches
- Average number of nodes explored
- Win rate
- Searches that failed due to time limit

Average duration of searches

Minimax

Minimax average of total search time (s) vs board size and search depth				
Board Size	**Depth**			
	3	**4**	**5**	**6**
4	0.022	0.04	0.077	0.121
8	6.808	30.616	167.335	632.757
12	87.183	390.696	856.841	1376.846

MCTS

MCTS average of total search time (s) vs board size and iterations				
Board Size	**Iterations**			
	10	**100**	**1000**	**10000**
4	0.044	0.299	1.834	12.024
8	4.062	39.075	370.572	1888.147
12	34.755	373.311	1772.112	2851.605

Average of total nodes explored

Minimax

Minimax average nodes explored vs board size and search depth				
Board Size	Depth			
	3	4	5	6
4	104.04	196	381.5	658.42
8	6924.04	32816.58	180567.2	745303
12	41911.46	189724.4	319085.7	666029

MCTS

MCTS average nodes explored vs board size and iterations				
Board Size	Iterations			
	10	100	1000	10000
4	321.67	2237.54	12231.88	59256.71
8	8959.71	87142.33	836543.3	4195134
12	32708.46	361819.5	1598819	2729862

Average minimax win rate

Minimax win rate vs board size						
Board Size	Depth	Iterations				Average
		10	100	1000	10000	
4	3	33.33%	33.33%	50.00%	50.00%	41.67%
	4	16.67%	0.00%	0.00%	0.00%	4.17%
	5	16.67%	0.00%	0.00%	0.00%	4.17%
	6	16.67%	0.00%	0.00%	0.00%	4.17%
8	3	100.00%	0.00%	16.67%	50.00%	41.67%
	4	100.00%	66.67%	16.67%	66.67%	62.50%
	5	100.00%	50.00%	16.67%	50.00%	54.17%
	6	100.00%	33.33%	66.67%	83.33%	70.83%
12	3	83.33%	66.67%	100.00%	100.00%	87.50%
	4	83.33%	33.33%	100.00%	83.33%	75.00%
	5	100.00%	100.00%	100.00%	83.33%	95.83%
	6	100.00%	83.33%	83.33%	66.67%	83.33%

Failed searches (due to time limit)

Games stopped early vs board size, depth and MCTS iterations					
Board Size	Depth	Iterations			
		10	100	1000	10000
4	3	0	0	0	0
	4	0	0	0	0
	5	0	0	0	0
	6	0	0	0	0
8	3	0	0	0	6
	4	0	0	0	6
	5	0	0	0	6
	6	0	0	0	6
12	3	0	0	6	6
	4	0	0	5	6
	5	4	5	5	6
	6	6	5	6	6

Other potential metrics for results

- Varying position/mobility weights for minimax evaluation function
- Different heuristic functions on minimax
- Varying balancing constant for UCB1

Thank you for your time!