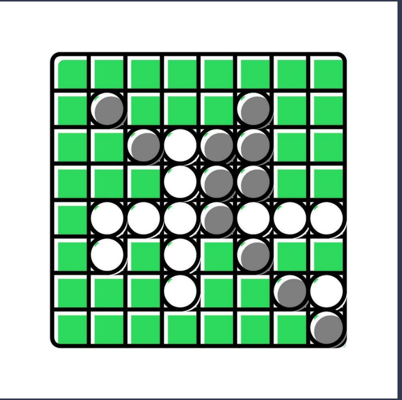
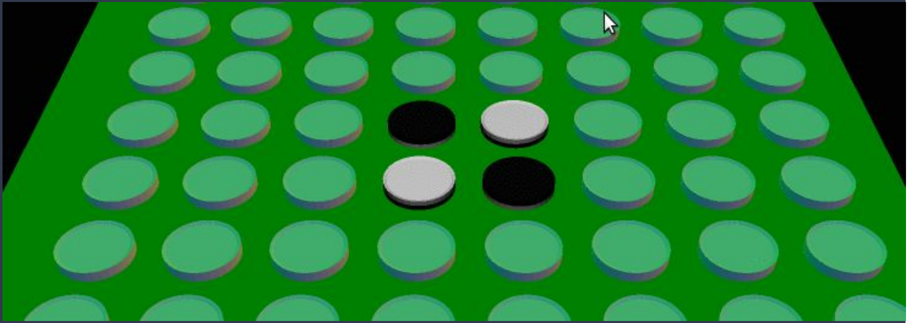
Comparative Analysis of AI Agents for Othello

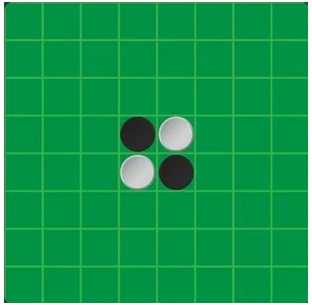
Arman Ashkari

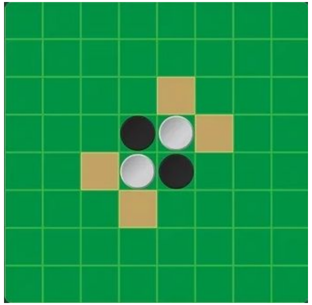
Mir Mahathir Mohammad

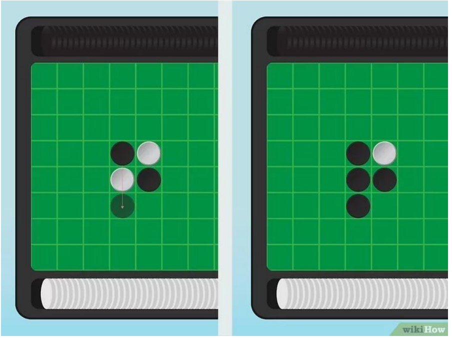


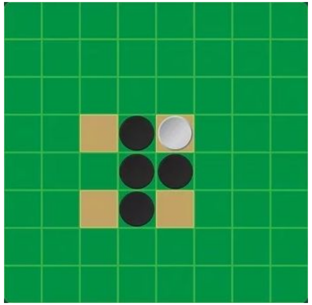
What is Othello





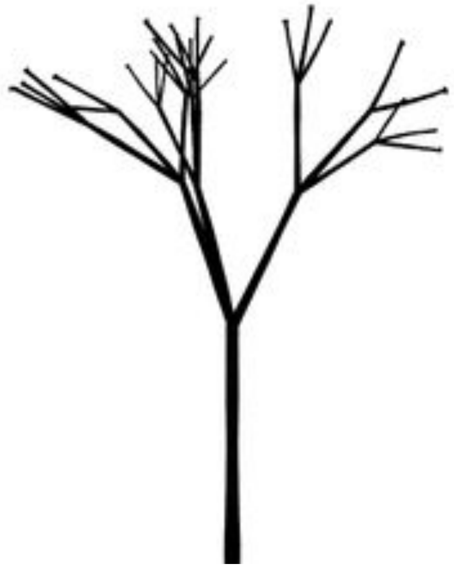




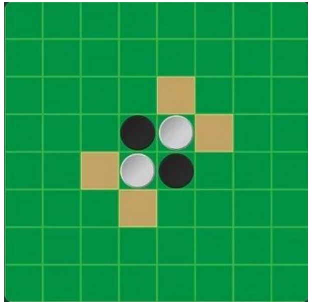


The player with the most disks wins

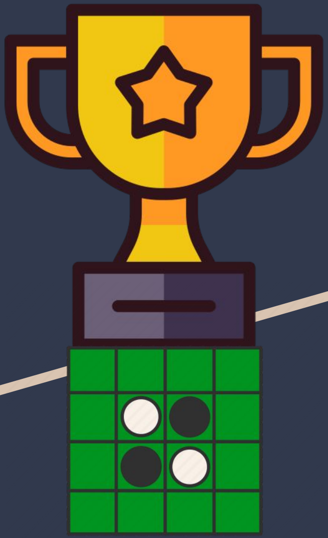
Why it is interesting?



Branching Factor: 5 to 15



Behold! The Tournament of Agents!



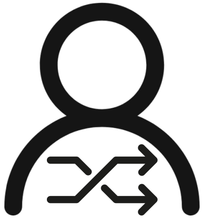
Our Contestant Types

- Traditional Agents
- Reinforcement Learning Agents
- Hybrid Agents

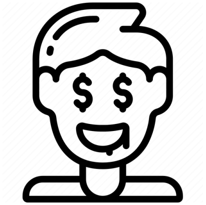
Traditional Agents

Moves randomly from the set of valid options

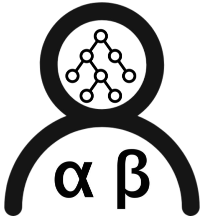
Selects the move which results in the maximum immediate gain in flipped opponent discs



Random

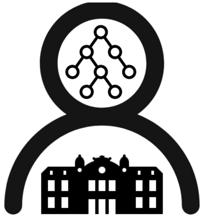


Greedy



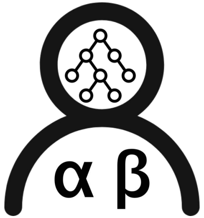
Alpha Beta Pruning

3 Variants
- Depth 1
- Depth 3
- Depth 4



Monte-Carlo Tree Search

Alpha-Beta Pruning (Details)

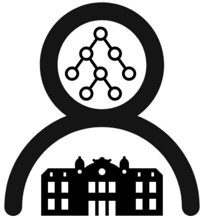


Alpha Beta Pruning

3 Variants
- Depth 1
- Depth 3
- Depth 4

- Uses the Minimax algorithm with alpha-beta pruning to make optimal decisions up to a specified search depth.
- This agent simulates future moves recursively, alternating between maximizing and minimizing the evaluation score depending on the turn.
- It leverages alpha-beta pruning to eliminate branches in the game tree that cannot influence the final decision, improving efficiency over standard Minimax.

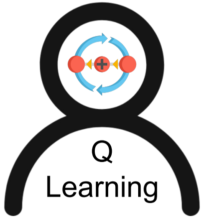
Monte-Carlo Tree Search (Details)



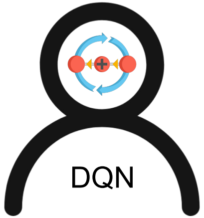
Monte-Carlo Tree Search

- Makes decisions in Othello by performing a series of randomized simulations to estimate the most promising move.
- For each valid move from the current game state, the agent runs a fixed number of playouts where it plays out the rest of the game using random moves until the end.
- It tracks the number of wins for each candidate move and selects the one with the highest success rate

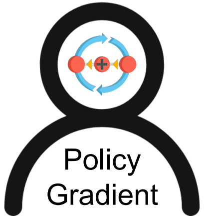
Reinforcement Learning Agents



Q Learning

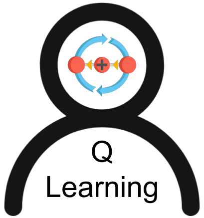


DQN



Policy Gradient

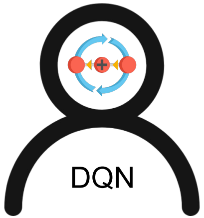
Q-Learning (Details)



Q Learning

- It maintains a dictionary of Q-values, each representing the expected long-term reward for taking a specific action from a particular board state.
- During training, it uses an epsilon-greedy policy to balance exploration and exploitation, gradually improving its strategy based on rewards received at the end of games or intermediate states.
- The agent updates its Q-values using the Bellman equation, factoring in learning rate (alpha), discount factor (gamma), and received reward. Q-values are periodically saved and reloaded to enable persistent learning across sessions

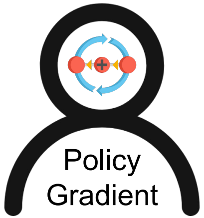
DQN (Details)



DQN

- It uses a convolutional neural network to map board states to Q-values for all possible actions, enabling the agent to estimate the long-term value of each move.
- The agent maintains a replay buffer to store past experiences and performs mini-batch training to stabilize learning.
- A separate target network is used to improve training stability, and epsilon-greedy exploration is employed to balance exploration with exploitation during training

Policy Gradients (Details)

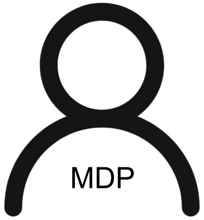


Policy
Gradient

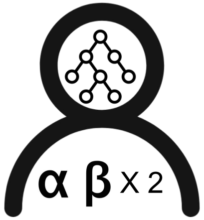
- It employs a neural network to map board states to a probability distribution over all possible actions.
- During gameplay, the agent samples actions based on this distribution, accumulating log-probabilities and rewards over the course of each episode.
- At the end of each game, it updates its policy using the REINFORCE algorithm, computing discounted returns and optimizing the network to increase the likelihood of successful actions.

Hybrid Agents

Monte Carlo Tree Search with Multi-Armed Bandit



Markov Decision Process with Value Iterations



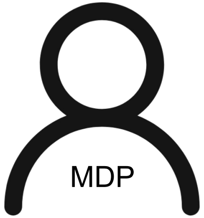
Two Alpha-Beta Agent

Monte-Carlo Tree Search with Multi-Armed Bandit (Details)

Monte Carlo Tree Search with Multi-Armed Bandit

- Monte Carlo Tree Search-based Othello agent that models each valid move as a separate arm in a multi-armed bandit problem.
- It performs a fixed number of simulations to evaluate move quality, using the Upper Confidence Bound (UCB1) formula to balance exploration and exploitation.
- For each iteration, the agent selects a move based on its current win/play statistics and simulates a complete random playout from that move.
- The outcomes are recorded to update win rates. After all iterations, it chooses the move with the highest empirical win rate

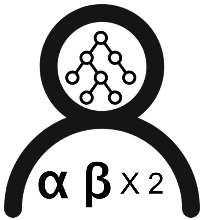
Markov Decision Process with Value Iteration (Details)



Markov Decision Process with Value Iterations

- It models the game as a Markov Decision Process (MDP), where each state is defined by the board configuration and current player.
- The agent explores the state space via simulation and uses depth-limited search to evaluate the consequences of actions.
- At each iteration, it updates its value estimates based on the Bellman equation and refines its policy to select actions that maximize expected future rewards.
- Rewards are based on the final game outcome or piece advantage in intermediate states.

Two Alpha-Beta Agent (Details)



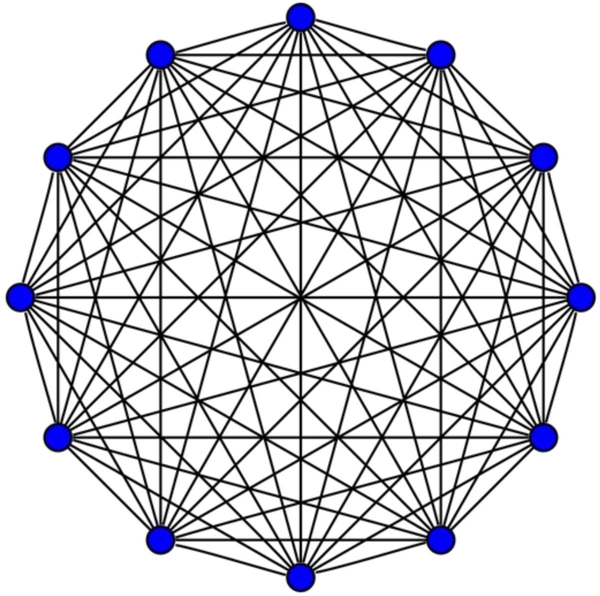
Two Alpha-Beta Agent

- During the early game (when the number of empty squares exceeds a predefined threshold), it uses a shallower search for speed and efficiency.
- In the late game, it switches to a deeper search to make more precise decisions.

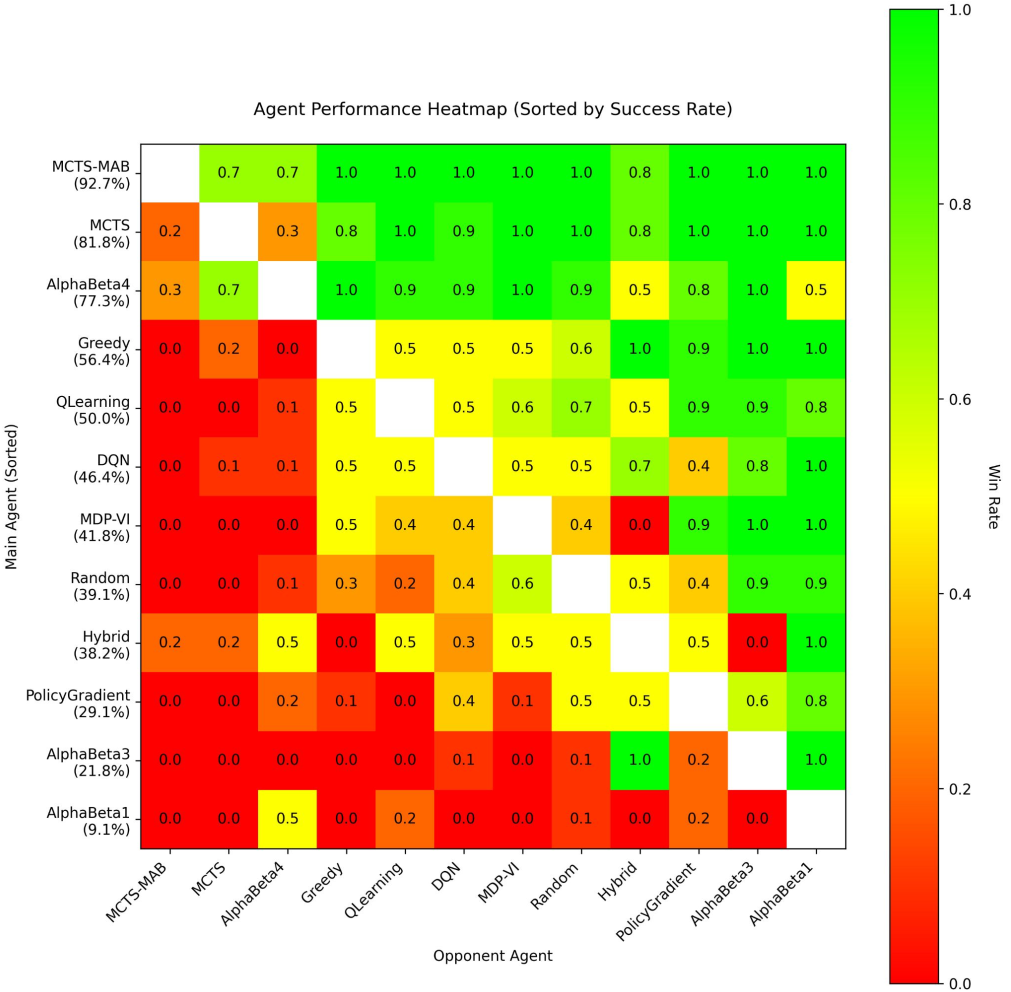
Tournament Design

Everyone will play 10 matches against everyone

Agent with the most wins gets the cup



Our Results

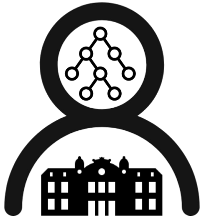

Agent Performance Heatmap (Sorted by Success Rate)
Main Agent (Sorted)
Opponent Agent
Win Rate
MCTS-MAB (92.7%)
MCTS (81.8%)
AlphaBeta4 (77.3%)
Greedy (56.4%)
QLearning (50.0%)
DQN (46.4%)
MDP-VI (41.8%)
Random (39.1%)
Hybrid (38.2%)
PolicyGradient (29.1%)
AlphaBeta3 (21.8%)
AlphaBeta1 (9.1%)
MCTS-MAB
MCTS
AlphaBeta4
Greedy
QLearning
DQN
MDP-VI
Random
Hybrid
PolicyGradient
AlphaBeta3
AlphaBeta1
0.7 0.7 1.0 1.0 1.0 1.0 1.0 0.8 1.0 1.0 1.0
0.2 0.3 0.8 1.0 0.9 1.0 1.0 0.8 1.0 1.0 1.0
0.3 0.7 1.0 0.9 0.9 1.0 0.9 0.5 0.8 1.0 0.5
0.0 0.2 0.0 0.5 0.5 0.5 0.6 1.0 0.9 1.0 1.0
0.0 0.0 0.1 0.5 0.5 0.6 0.7 0.5 0.9 0.9 0.8
0.0 0.1 0.1 0.5 0.5 0.5 0.5 0.7 0.4 0.8 1.0
0.0 0.0 0.0 0.5 0.4 0.4 0.4 0.0 0.9 1.0 1.0
0.0 0.0 0.1 0.3 0.2 0.4 0.6 0.5 0.4 0.9 0.9
0.2 0.2 0.5 0.0 0.5 0.3 0.5 0.5 0.5 0.0 1.0
0.0 0.0 0.2 0.1 0.0 0.4 0.1 0.5 0.5 0.6 0.8
0.0 0.0 0.0 0.0 0.0 0.1 0.0 0.1 1.0 0.2 1.0
0.0 0.0 0.5 0.0 0.2 0.0 0.0 0.1 0.0 0.2 0.0
1.0
0.8
0.6
0.4
0.2
0.0

Champions

#1

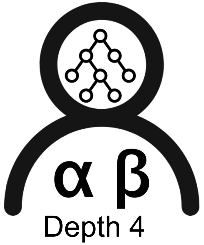
Monte Carlo Tree Search with Multi-Armed Bandit

#2



Monte-Carlo Tree Search

#3



Alpha Beta Pruning

Result Analysis

No Time Constraints

Tree algorithms took the most time

Limited State Representations and Reward Shaping

Q-Learning and DQN

Low Episode Count

Reinforcement Learning Agent

Project Completion Steps

Train neural network with MCTS-MAB

More episodes of RL agents

Better design of RL agents