AI GAME

SOLVING CHECKERS WITH MINIMAX ALPHA BETA PRUNING

Muhammad Amin Hakim bin Sazali	17187994/2 | WID170028
Muhammad Aliff Danial bin Mohd Zahiruddin	17121897 | WID170030
Muhammad Akif Farhan	17080476/2 | WID170029
Fadelic Mordecai Makain	17123777/2 | WID170014

Problem Definition

How to win?

Game will end when any of the player cannot move the piece or the player no longer has the piece on the board. The player with no piece will lose.

User and AI bot can use multi leg move to capture more than 1 opponent piece

The AI Objective.

To implement tree algorithm on the game with pruning

To suggest the best move with the highest score to win the game.

To study different algorithm performance on the game

Related Work

Minimax Checkers Playing GUI: A Foundation for AI Applications

- Uses Minimax algorithm to choose the best possible move for the AI.
- Test the AI with Universidad de Ingeniería y Tecnología (UTEC) students
- Checkers AI win rate increases as the depth of the tree increases.
- As the depth of tree increases, the average run time also increases.

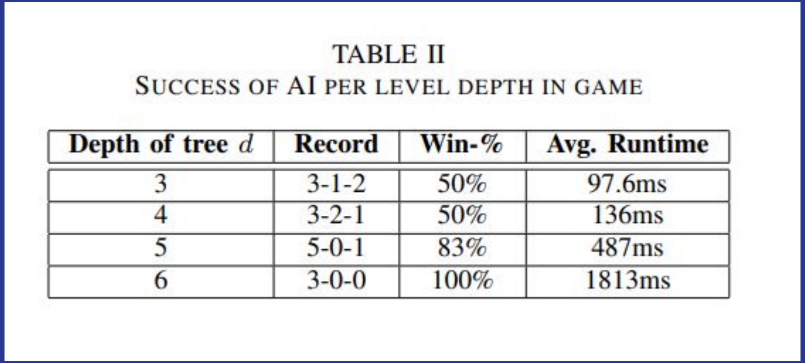


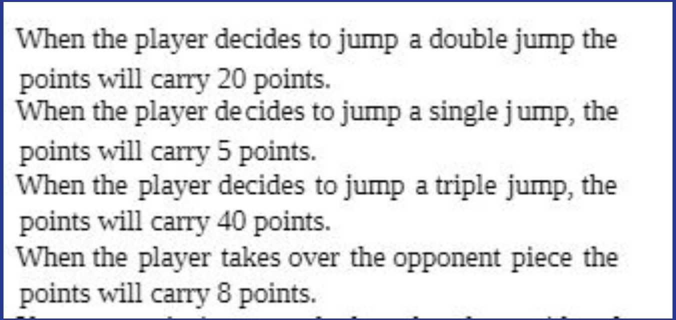
TABLE II
SUCCESS OF AI PER LEVEL DEPTH IN GAME

Depth of tree d	Record	Win-%	Avg. Runtime
3	3-1-2	50%	97.6ms
4	3-2-1	50%	136ms
5	5-0-1	83%	487ms
6	3-0-0	100%	1813ms

Related Work

Checkers: Implementation Of AI In Checkers

- Uses Minimax algorithm to choose the best possible move for the AI.
- 3 levels of difficulty.; easy, medium, hard.
- The AI is given 4 minutes to suggest the best move.
- The evaluation function used in this work is a lot.


When the player decides to jump a double jump the points will carry 20 points.
When the player de cides to jump a single j ump, the points will carry 5 points.
When the player decides to jump a triple jump, the points will carry 40 points.
When the player takes over the opponent piece the points will carry 8 points.

Example of evaluation function used.

Environment & Simulation

Python

Most of us familiar with Python Language

Easier to implement the logic of minimax algorithm with alpha beta pruning and Monte Carlo Tree Search

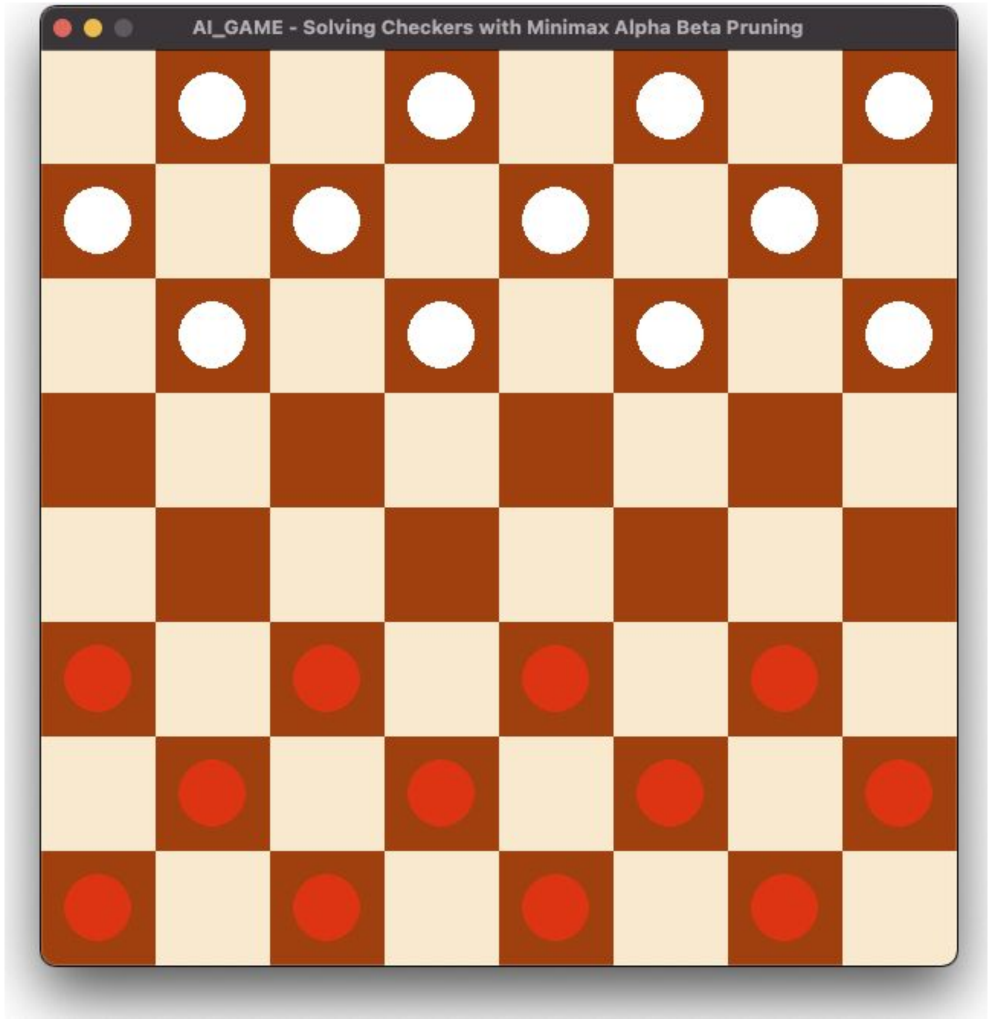
PyGame

Python wrapper for Simple DirectMedia Library (SDL)

SDL offers cross-platform connectivity, such as sound, video, mouse, and keyboard to the underlying multimedia hardware components of a device.

Highly portable and runs on nearly every platform and operating system

Use to control GUI


AI_GAME - Solving Checkers with Minimax Alpha Beta Pruning

Project structure

Classes

- Board
  - Draw and update the GUI for every move user and AI made
- Piece
  - Represent each pieces on the board
  - Able to control color and king state
- Game
  - Control the rules and state of the game
- Constants
  - Store configuration of the game
- AI_Bot
  - Implemented minimax and alpha beta pruning algorithm
- Main
  - Instantiated Game and AI_Bot class and run the program

Proposed Approach

Random

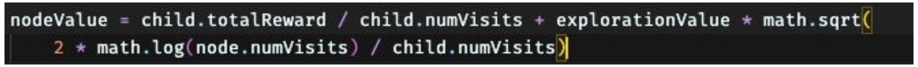
A move will be chosen randomly from all possible moves

Monte Carlo Tree Search

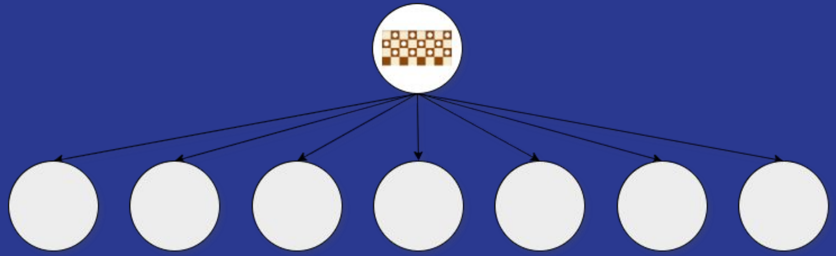
Edge of a node tells what possible states can be reached from the current state and its reward(score).

In this algorithm, there will be a selection phase which action will be lead to the node which the following equation maximized.

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$$



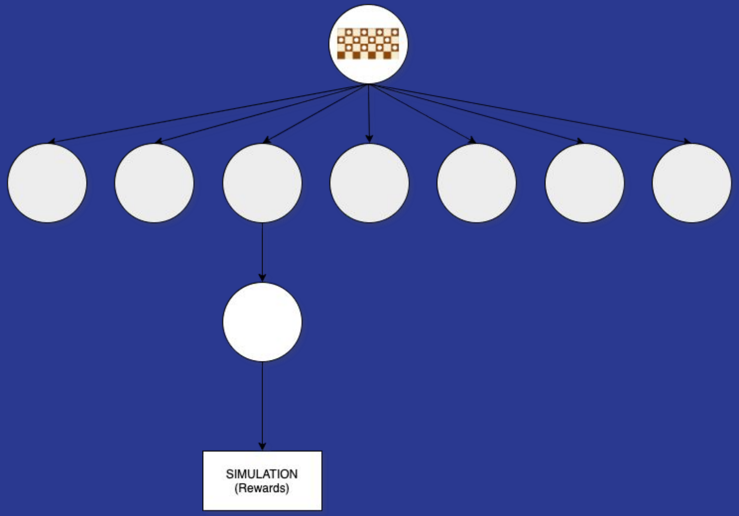
```
nodeValue = child.totalReward / child.numVisits + explorationValue * math.sqrt(
    2 * math.log(node.numVisits) / child.numVisits)
```



Monte Carlo Tree Search

Expansion phase, an action will be chosen randomly and a simulation of the game will be done in the current node.

Actions in the simulation phase are chosen randomly. This purpose it to help in evaluation of the current node position.

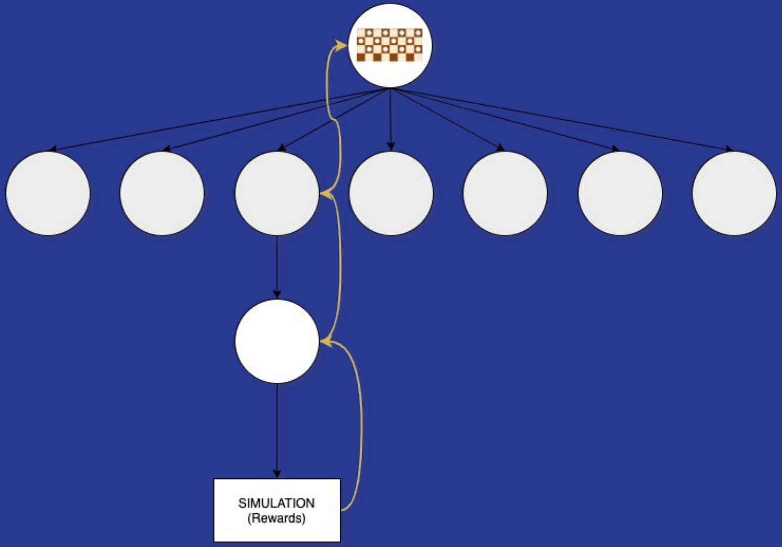


Monte Carlo Tree Search

During back propagation phase, the result from the simulation will be updated to the selected nodes.

After many iterations through those steps, the possible move will be chosen for the node that have highest score.

Then the process will be repeated over and over until the end of the game.

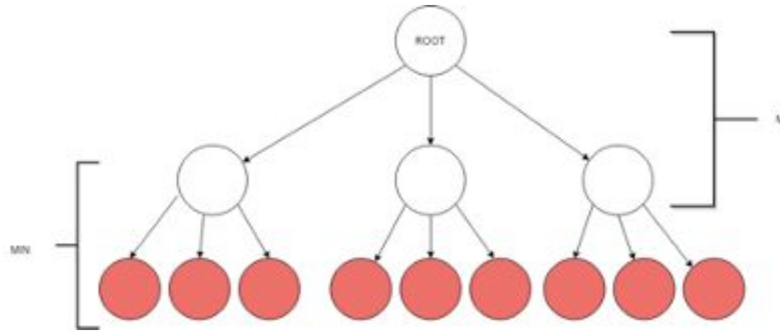


Successor function

To generate nodes, **recursive function** is used to get min and max values for each node

At each traverse of node, the alpha and beta value will be compared and updated

Minimax evaluation



Simple 2 tree depth visualization of minimax algorithm

The algorithm is set up to **5 tree depth**.

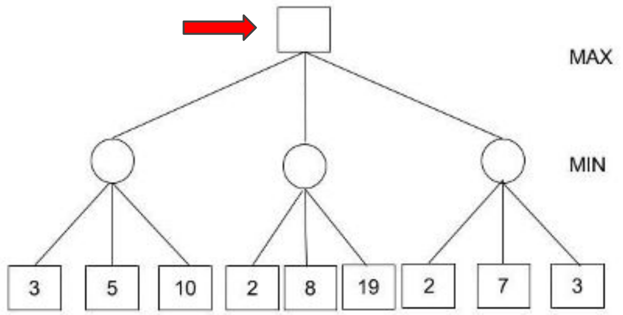
1st Layer (root) represent a white piece. There are **3 possible moves**.

It created a **2nd layer** of **3 white nodes**.

For each white move, the opponent has 3 possible moves as well.

Hence, on **3rd layer**, each white node has **3 red nodes** (possible moves from the opponent)

Alpha-Beta pruning



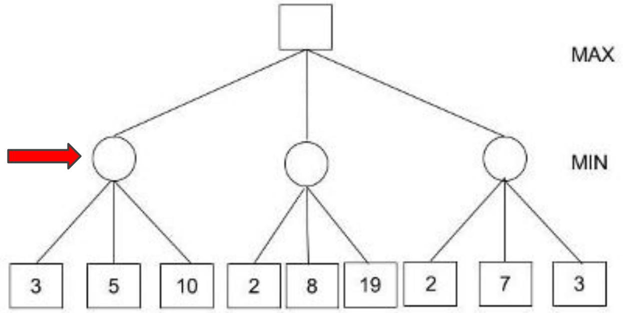
Position: Root

Initialize α=-∞ and β=∞ as worst possible cases.

The condition to prune a node is when α >= β

Start with assigning initial values of α and β to root. Since α < β, **don't prune**.

Alpha-Beta pruning



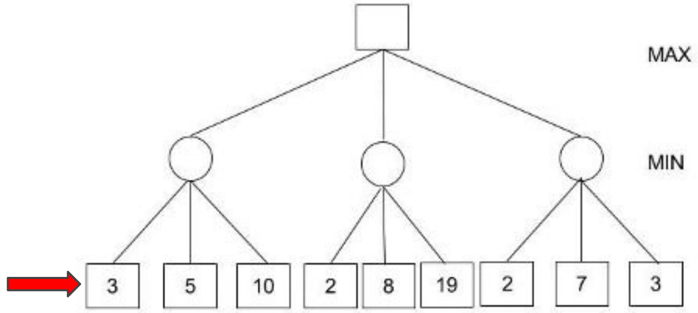
Position: 2nd child node

α and β values carried to the child node on the left.

From the utility value of the terminal state, the values **α and β updated**.

Don't prune, as the condition remains the same.

Alpha-Beta pruning

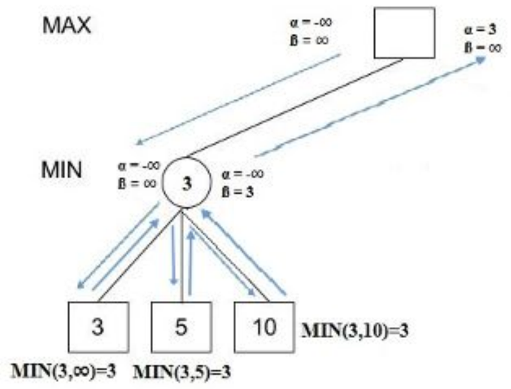


Position: 3rd child node

Don’t prune, as the condition remains the same.

Then backtracking to the root, we set α=3 as it is the min value that α can have.

Alpha-Beta pruning

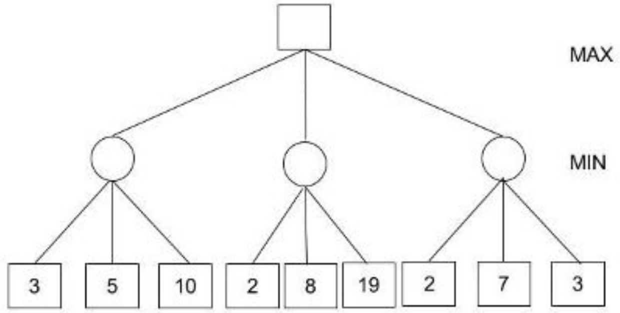


Since the value remains, **don't prune** and the values carried to center node.

Calculate *min(2,∞)*. α=3, β=2.

Prune the 2nd and 3rd child nodes because α > β

Alpha-Beta pruning



Position: 2nd child node

α at root remain 3 because it greater than 2. The values carried to the rightmost child node, and evaluate *min(2, ∞).*

Update β=2, **prune** the 2nd and 3rd child nodes because α > β.

We get 3, 2, 2 at the left, center and right MIN nodes. Calculate the *max(3,2,2) = 3.*

Therefore, without even looking at 4 leaves we could find the minimax decision.

Appropriate use of heuristic

Reduce red pieces score and increase the white pieces score

MAX function used on the **white nodes**

MIN function applied on **red nodes**

Experiments

Experiments

Checkers AI Comparison

We conducted a simple experiment to see the performance between these a few algorithms.

Experiment 1

Base Bot vs Minimax with AB Pruning

Experiment 2

Base Bot vs Monte Carlo Tree Search

Experiment 3

Minimax with AB Pruning vs Monte Carlo Tree Search

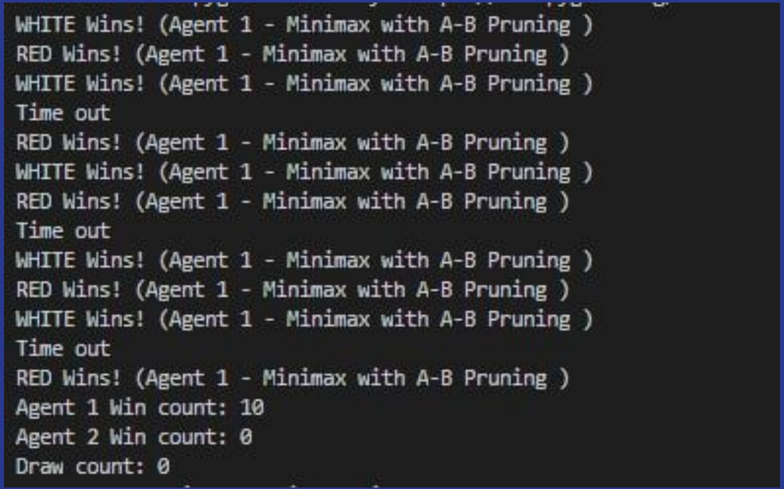
Initial Results

Experiment 1

BASE Bot VS AI Bot - 1

In this experiment, the algorithm of the Base Bot only moves randomly while the minimax agent uses a heuristic function to try to maximise its score during each movement.

Outcome



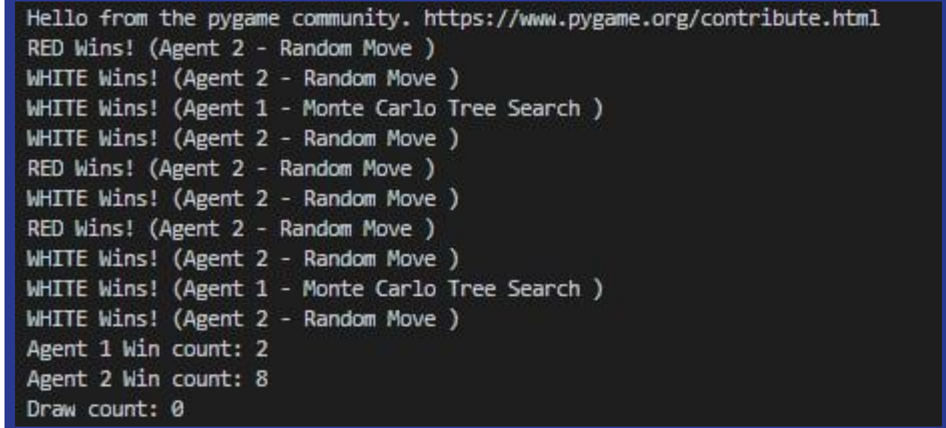
```
WHITE Wins! (Agent 1 - Minimax with A-B Pruning )
RED Wins! (Agent 1 - Minimax with A-B Pruning )
WHITE Wins! (Agent 1 - Minimax with A-B Pruning )
Time out
RED Wins! (Agent 1 - Minimax with A-B Pruning )
WHITE Wins! (Agent 1 - Minimax with A-B Pruning )
RED Wins! (Agent 1 - Minimax with A-B Pruning )
Time out
WHITE Wins! (Agent 1 - Minimax with A-B Pruning )
RED Wins! (Agent 1 - Minimax with A-B Pruning )
WHITE Wins! (Agent 1 - Minimax with A-B Pruning )
Time out
RED Wins! (Agent 1 - Minimax with A-B Pruning )
Agent 1 Win count: 10
Agent 2 Win count: 0
Draw count: 0
```

Experiment 2

BASE Bot VS AI Bot - 2

This time we tested the Base Bot against the MCTS Bot. The base bot moves randomly while the MCTS agent uses the same heuristic function as the minimax agent to try to maximise its score during each movement.

Outcome



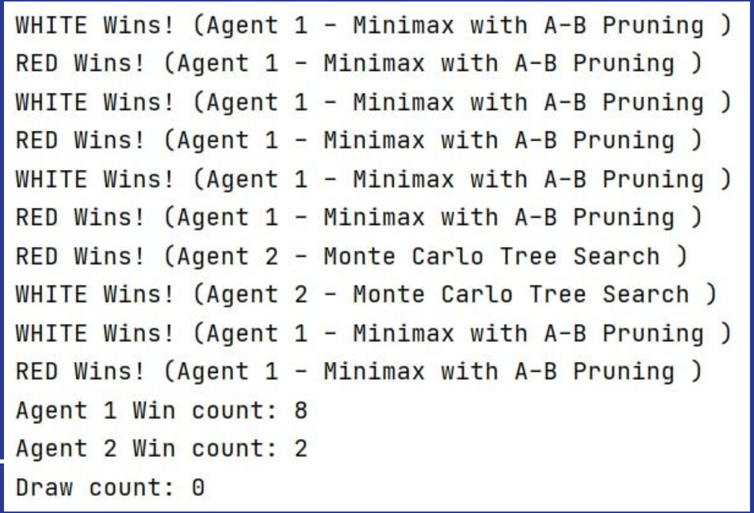
```
Hello from the pygame community. https://www.pygame.org/contribute.html
RED Wins! (Agent 2 - Random Move )
WHITE Wins! (Agent 2 - Random Move )
WHITE Wins! (Agent 1 - Monte Carlo Tree Search )
WHITE Wins! (Agent 2 - Random Move )
RED Wins! (Agent 2 - Random Move )
WHITE Wins! (Agent 2 - Random Move )
RED Wins! (Agent 2 - Random Move )
WHITE Wins! (Agent 2 - Random Move )
WHITE Wins! (Agent 1 - Monte Carlo Tree Search )
WHITE Wins! (Agent 2 - Random Move )
Agent 1 Win count: 2
Agent 2 Win count: 8
Draw count: 0
```

Experiment 3

AI Bot - 1 VS AI Bot - 2

In this experiment, we tested the minimax Bot against the MCTS Bot. Both the minimax agent and MCTS agent are using same heuristic function.

Outcome



```
WHITE Wins! (Agent 1 - Minimax with A-B Pruning )
RED Wins! (Agent 1 - Minimax with A-B Pruning )
WHITE Wins! (Agent 1 - Minimax with A-B Pruning )
RED Wins! (Agent 1 - Minimax with A-B Pruning )
WHITE Wins! (Agent 1 - Minimax with A-B Pruning )
RED Wins! (Agent 1 - Minimax with A-B Pruning )
RED Wins! (Agent 2 - Monte Carlo Tree Search )
WHITE Wins! (Agent 2 - Monte Carlo Tree Search )
WHITE Wins! (Agent 1 - Minimax with A-B Pruning )
RED Wins! (Agent 1 - Minimax with A-B Pruning )
Agent 1 Win count: 8
Agent 2 Win count: 2
Draw count: 0
```

References

- Escandon, E. R., & Campion, J. (2018). Minimax Checkers Playing GUI: A Foundation for AI Applications. 2018 IEEE XXV International Conference on Electronics, Electrical Engineering and Computing (INTERCON). doi:10.1109/intercon.2018.8526375
- Alkharusi, Suhaib. (2020). checkers research paper based on AI (2). 1. 7.
- Fincher, J. (n.d.). *Real Python*. Retrieved from PyGame: A Primer on Game Programming in Python: https://realpython.com/pygame-a-primer/#background-and-setup