CSC 215 Project 4 Report

Deadline: 3:00 PM, April 22, 2023

Contents

Members: Aayush Shukla (ID: 302777098), Shubham Ghule (ID: 302784482), Rajvee Modi (ID: 219712052)

- **Problem Statement:** The problem statement for this project is to complete the implementation of the minimax function for two versions of the Tic-Tac-Toe game (standard and wild), and then verify certain characteristics of each game through simulations. The simulations will be run multiple times under different settings, including optimal vs optimal and optimal vs random. For Tic-Tac-Toe, the characteristics to be verified include whether the game will always be a draw if both players are optimal, and whether the optimal player will always win if one player is optimal and the other is random. For Wild Tic-Tac-Toe, the characteristics to be verified include whether the first player can always win if both players are optimal and the middle cell is chosen as the first move, whether the game will always be a draw if both players are optimal and the first player does NOT choose the middle cell as the first move, and whether the optimal player will win all games if she moves first and most games if she moves second against a random player. The project provides three verification functions for each game to run the simulations: optimal_vs_optimal, random_vs_optimal, and you_vs_optimal. The findings from the simulations will be included in the final report.
- **Methodology:** The code provided implements two variations of the minimax algorithm: a basic minimax function and an alpha-beta pruning minimax function. Both functions take as input the current state of a tic-tac-toe board and the current player, and return the optimal score for that player.
  The basic minimax function recursively searches through all possible moves and calculates the score for each possible move using the check_game_over() function, which returns a score of 10 if the current player wins, a score of -10 if the opponent wins, and a score of 0 if the game is a tie or not over yet. The function also keeps track of the optimal move and adds it to a list called "current", which can be used to visualize the computer's moves.

  If the current player is "x", the function selects the move with the highest score, while if the current player is "o", the function selects the move with the lowest score.

  The alpha-beta pruning minimax function is a more optimized version of the basic minimax function, which reduces the number of nodes that need to be searched by pruning branches of the game tree that are guaranteed to not produce the optimal move. The function uses alpha-beta pruning to eliminate nodes that cannot produce a better outcome than previously explored nodes.

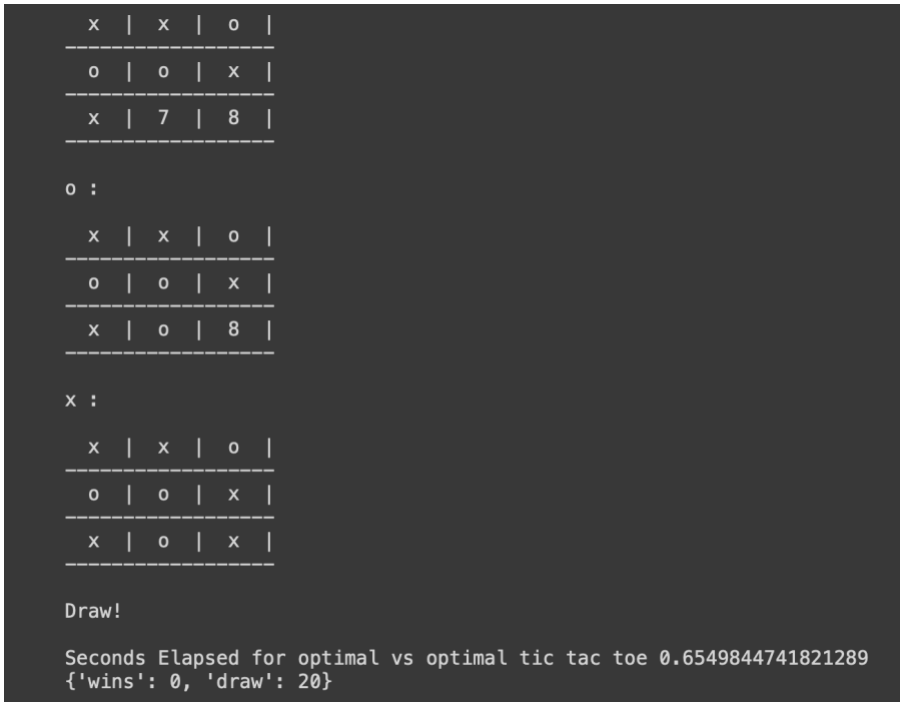
  The alpha-beta pruning minimax function starts with initial values of alpha=-inf and beta=inf for player "x" and alpha=inf and beta=-inf for player "o". The function then recursively searches through all possible moves, updating the alpha and beta values as it goes, and pruning any branches of the tree that cannot produce a better outcome than what has already been found. If the current player is "x", the function selects the move with the highest score that satisfies the alpha-beta condition, while if the current player is "o", the function selects the move with the lowest score that satisfies the alpha-beta condition.

  Both functions rely on the check_game_over() function to calculate the score for each possible move. The check_game_over() function examines the current state of the board and returns the score for that state. If the game is not over yet, the function returns False.

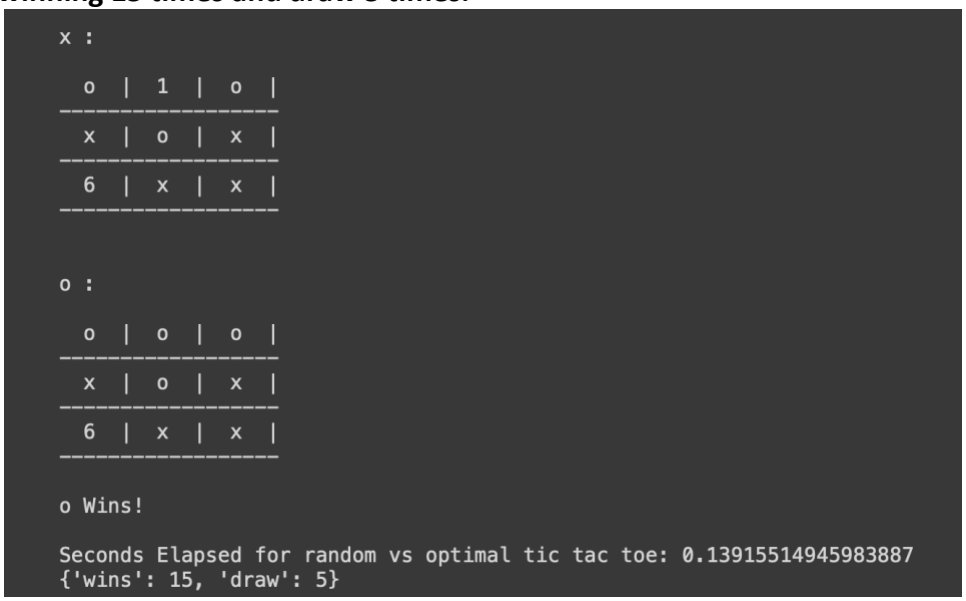
- **Experimental result and analysis:** The following table shows the result.

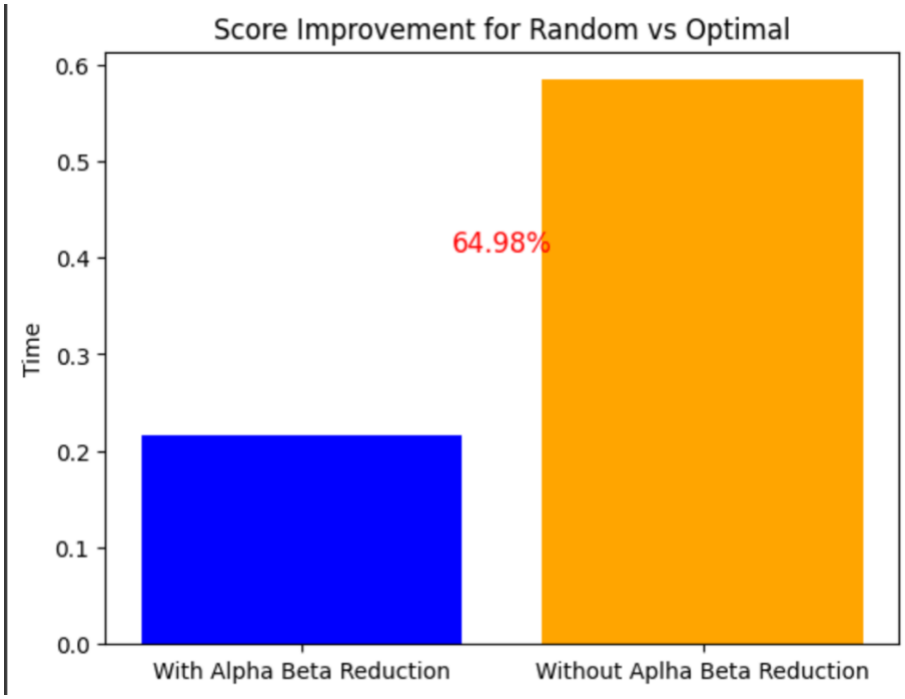
  Tic tac toe results:

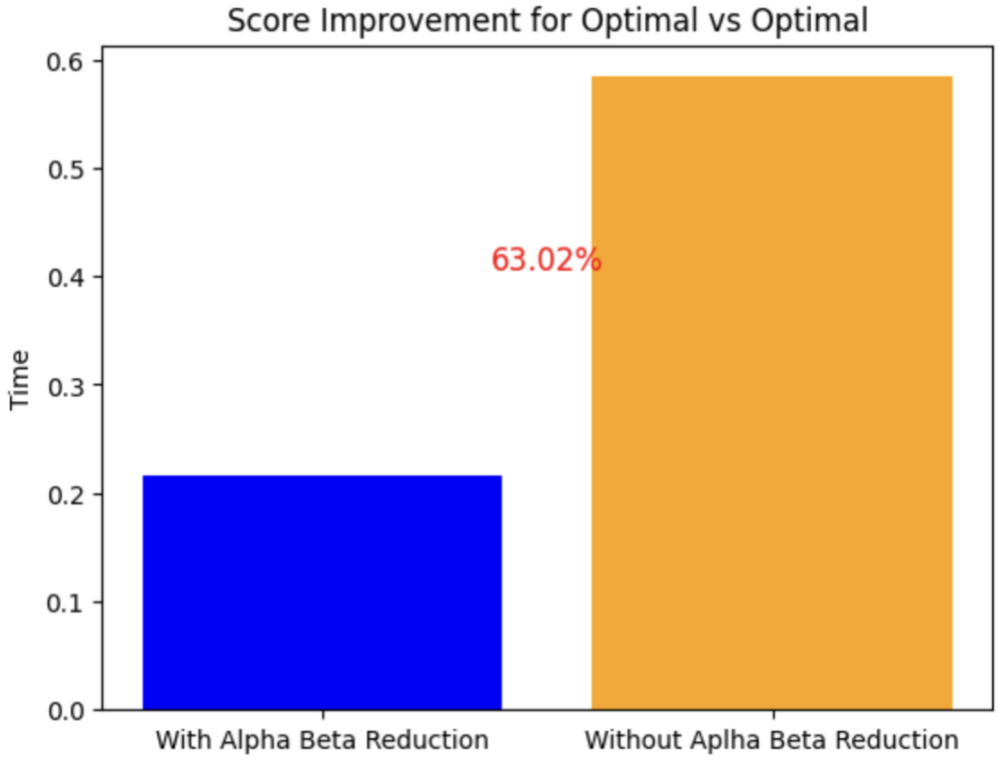
  This image shows that while playing tic tac toe for optimal vs optimal, it always results in a draw.



  This image shows that while playing tic tac toe for random vs optimal, it results in optimal winning 15 times and draw 5 times.

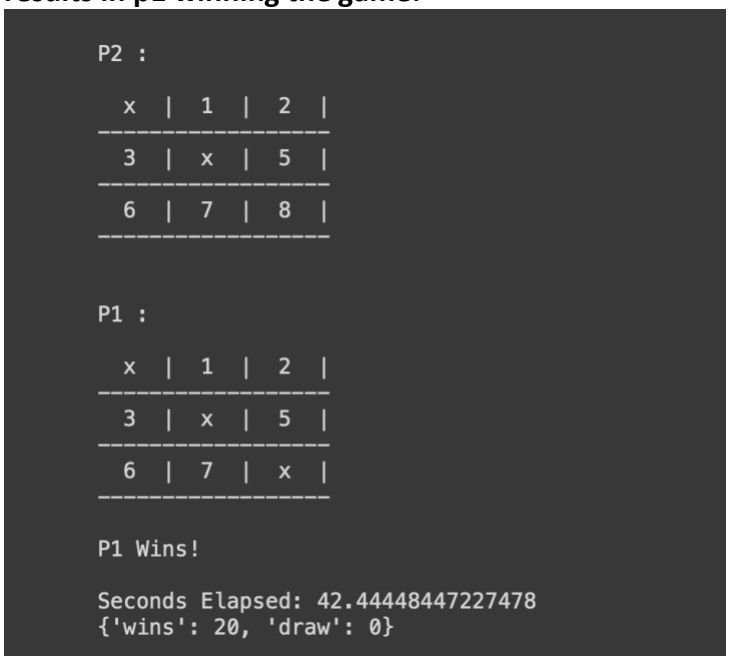



Score Improvement for Random vs Optimal
Time
0.0
0.1
0.2
0.3
0.4
0.5
0.6
64.98%
With Alpha Beta Reduction
Without Aplha Beta Reduction

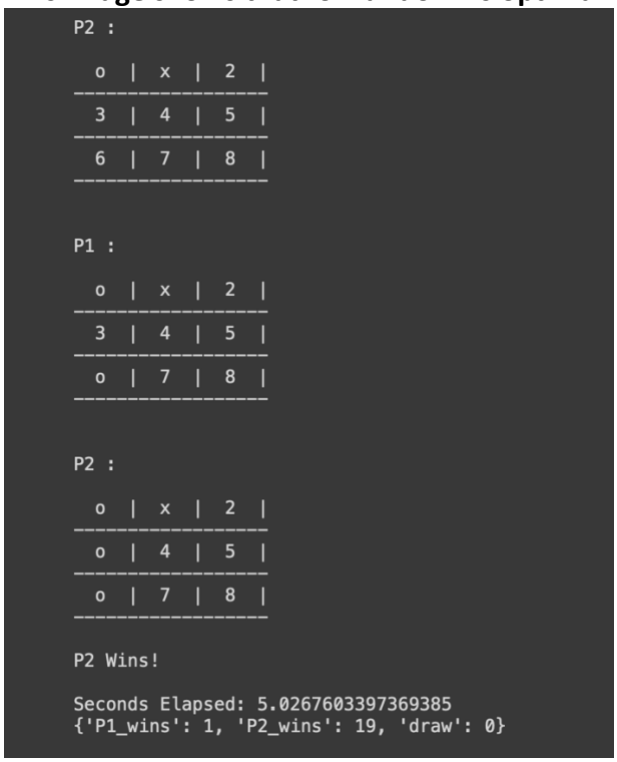

Score Improvement for Optimal vs Optimal
Time
0.0
0.1
0.2
0.3
0.4
0.5
0.6
63.02%
With Alpha Beta Reduction
Without Aplha Beta Reduction

Wild Tic tac toe results:

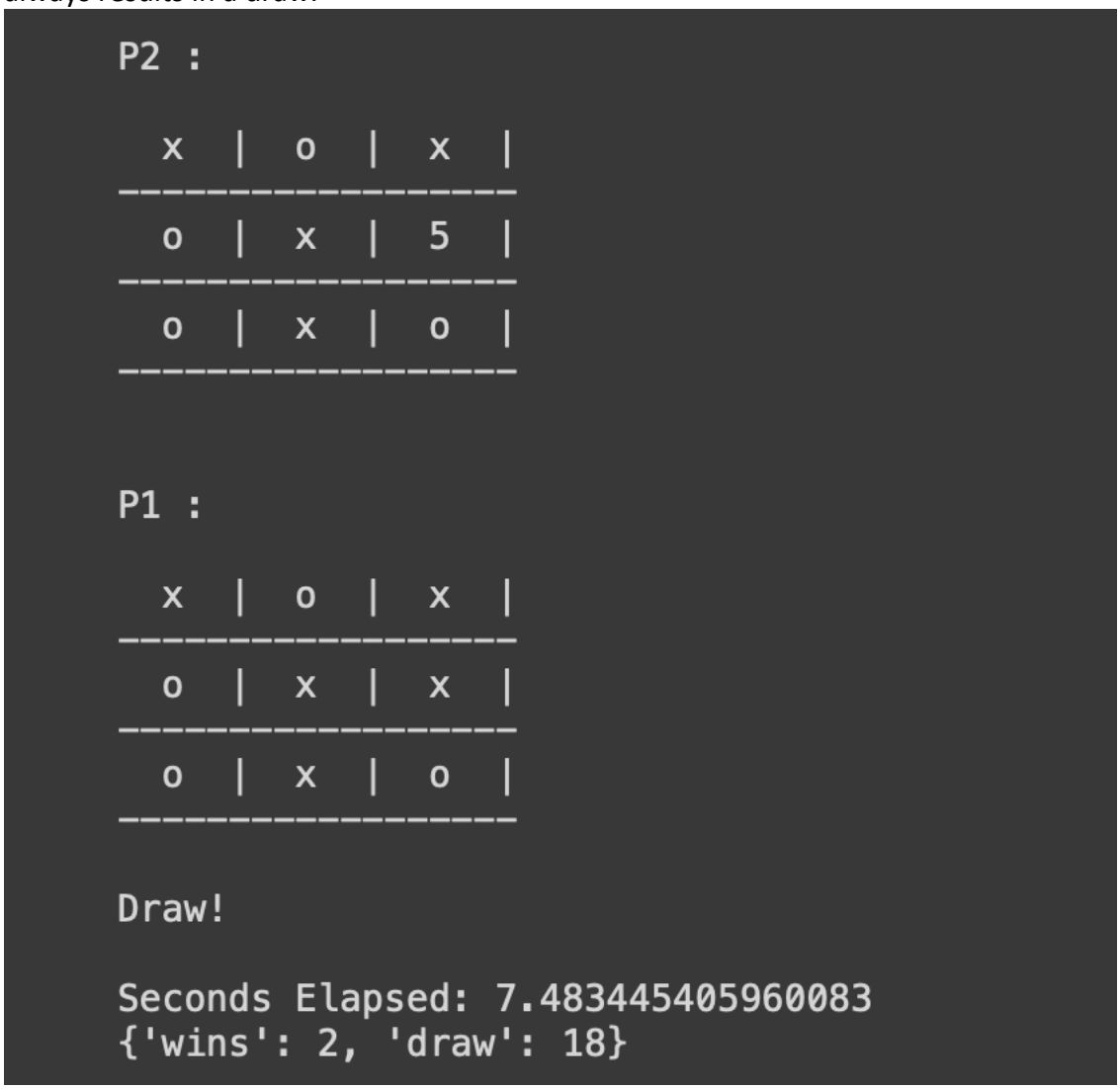
This image shows that for optimal vs optimal wild tic tac toe when P1 chooses middle cell, it always results in p1 winning the game.



This image shows that for random vs optimal wild tic tac toe it results in optimal winning 19 games



This image demonstrates that when choosing the first cell randomly in a game of Wild Tic Tac Toe with two optimal players, there were 18 draws and 2 wins. If any cell besides the middle one is chosen, it always results in a draw.



- Task division and Project Reflection
  - Task Division:

Sr no.	Task	Assigned to
1.	Tic tac toe	Aayush, Rajvee
2.	Aplha Beta Pruning	Shubham
3.	Wild Tic Tac Toe	Rajvee
4.	Report	Aayush

  - **Challenges:** Following are the challenges we faced while working on this project:
    - Implementing Alpha Beta pruning
    - Understanding wild tic tac toe

- **Learnings:** One of the primary learnings from this project is the importance and effectiveness of alpha-beta pruning in optimizing the minimax algorithm. Alpha-beta pruning is a technique that can help reduce the number of nodes evaluated by the minimax algorithm by eliminating sub-trees that do not need to be evaluated. This is achieved by keeping track of two values, alpha and beta, which represent the best scores found so far for the maximizing and minimizing players, respectively. As the algorithm progresses, if it finds a node that has a score worse than the best score found so far for the opposing player (i.e., if the current node is a maximizing node, and its score is worse than the best score found so far by the minimizing player), it can stop evaluating the rest of the nodes in that sub-tree, as they cannot possibly lead to a better outcome.

- **Additional Features:** We implemented Alpha Beta Pruning as additional feature for this project. Alpha-beta pruning is a technique that can help reduce the number of nodes evaluated by the minimax algorithm by eliminating sub-trees that do not need to be evaluated. This is achieved by keeping track of two values, alpha and beta, which represent the best scores found so far for the maximizing and minimizing players, respectively. As the algorithm progresses, if it finds a node that has a score worse than the best score found so far for the opposing player (i.e., if the current node is a maximizing node, and its score is worse than the best score found so far by the minimizing player), it can stop evaluating the rest of the nodes in that sub-tree, as they cannot possibly lead to a better outcome.