Report for AI project 1

Group 7

Kwok Keith, Lim Hsien Hong Sean, Toh Hong Jing

1. **Description(s) of your state evaluation function(s)**
    - combined_evaluation

This is the primary evaluation function that aggregates the scores from various evaluation methods to compute a comprehensive evaluation of a game state. It considers the potential for captures, protection of pieces, the distance of pieces from their goal, and the maintenance of a defense line. The final score is adjusted based on whose turn it is, with the intention of maximizing the agent's position while minimizing the opponent's. Calculated moves score is subtracted from the total score as the smaller the number of possible moves, the closer the friendly pieces are to the enemy baseline and hence the goal state.

Evaluation Score = capture potential evaluation + protected evaluation - calculated moves to goal + defensive line evaluation

- capture_potential_evaluation

This function assesses the current game state to forecast the player's ability to capture opponent pieces in subsequent turns, based on the potential moves the opponent could make. It analyzes the current board layout to identify moves that could lead to capturing opponent pieces in the next turn after an opponent moves, prioritizing moves that place the agent's pieces in advantageous positions for future captures. The score increases with the number of potential captures.

- protected_evaluation

This function assesses the degree to which the agent's pieces protect each other. By identifying friendly pieces that are covered by other friendly pieces, especially in diagonal positions that are typically harder for the opponent to attack without being captured, it calculates a score that reflects the strength of the agent's defensive structure. Scores based on how many friendly pieces are protected (by another friendly piece) based on current state. Specifically, we check if the back-diagonals are friendly pieces – so in the case of a front-diagonal friendly piece being captured, we have a back-diagonal friendly piece to recapture. Higher score means more back-diagonal friendly pieces protecting a front-diagonal piece.

- calculate_moves_to_goal

This function estimates the number of moves required for the agent's pieces to reach the opponent's baseline, assuming a clear path. It provides insight into how close the agent is to achieving a winning condition or significantly advancing its pieces. The score is the total number of moves needed for all pieces to reach the goal, with a higher score indicating that nodes are generally far from the enemy baseline and thus the goal position.

- defense_line_evaluation

This function evaluates the importance of maintaining a defensive line at the friendly baseline, particularly focusing on preventing opponent pieces from reaching the friendly baseline. Opponents would have to move towards the baseline and capture a piece at the friendly baseline before winning if the defensive line still exists. If the defensive line exists, we are guaranteed that an opponent's piece would be captured before it is able to win from reaching the friendly baseline. The evaluation also considers the potential for capturing opponent pieces that threaten this line and gives a large score for this move, hence making the agent more rational by preventing an opponent's winning move. A high score from this function indicates a strong defensive position that impedes the opponent's winning chances.

2. A description of the experiments you ran

We ran experiments on game boards:

1. 3 x 5
2. 5 x 5
3. 7 x 7
4. 8 x 8
5. 10 x 10

We also experimented with the following play clocks:

1. 5 ms
2. 10 ms
3. 30 ms

The types of agent that we experimented were:

1. Remote vs Remote
2. Remote (White) vs Random (Black)
3. Random (White) vs Remote (Black)

We also performed tests on the number of nodes expanded; with and without pruning.

We primarily used our own agent, which we implemented together with our own environment. Initially, our testing concentrated exclusively on strategies for the white player, treating the black player's moves as random.

After finalizing the strategy implementation for the white player, we adapted and applied a similar strategy for the black player. This comprehensive approach enabled us to conduct experiments across all game boards.

3. Results of experiments

Note that for all the results of the experiments, you can think of the player (us) as remote white.

Please kindly see the excel file attached for the graph data. The file “Statistics After Pruning” contains data for Figure 6 and Figure 8.

The file “Statistics Before Pruning” contains data for the other figures.

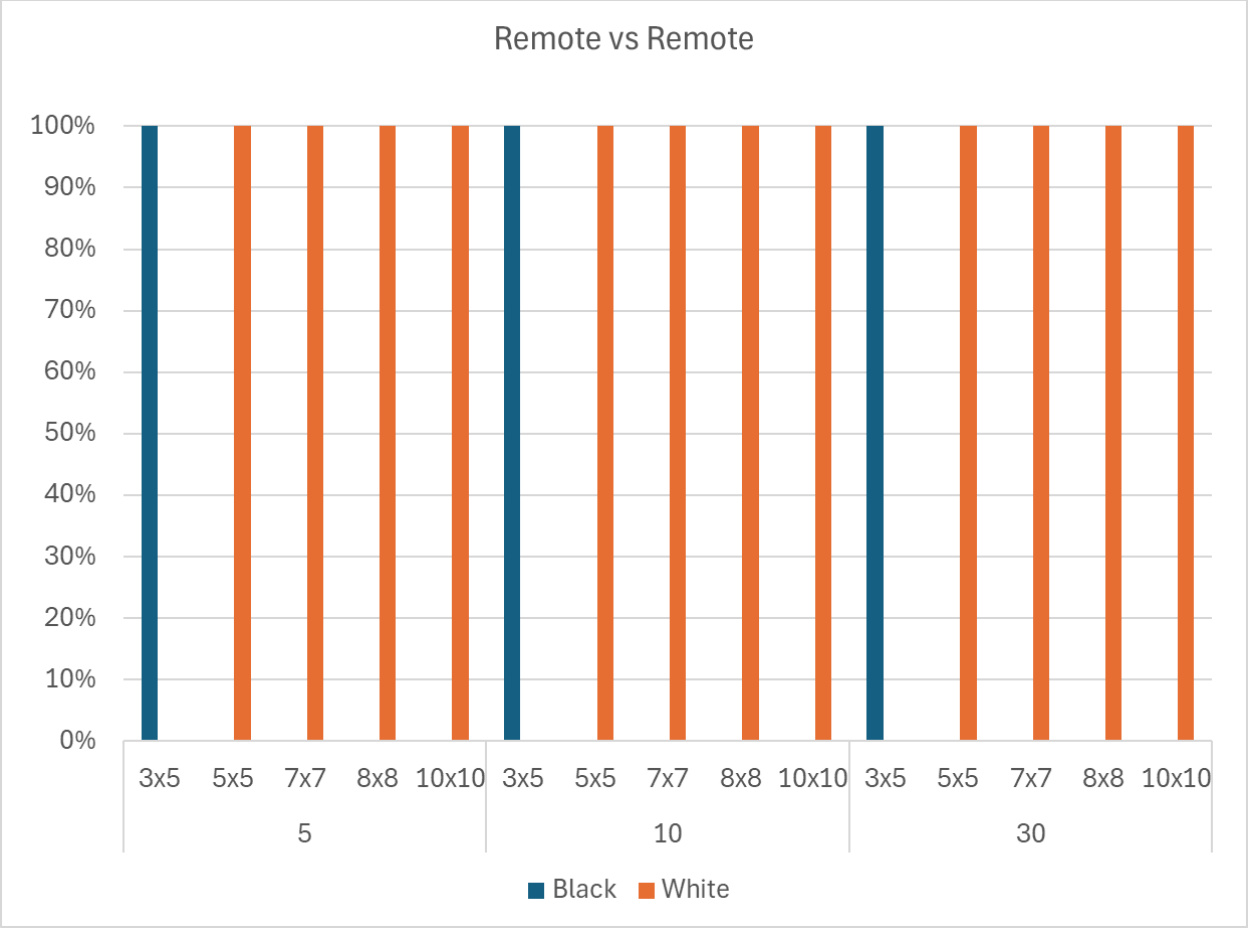


Figure 1: Winner ratios - for each board, playclocks 5s, 10s and 30s

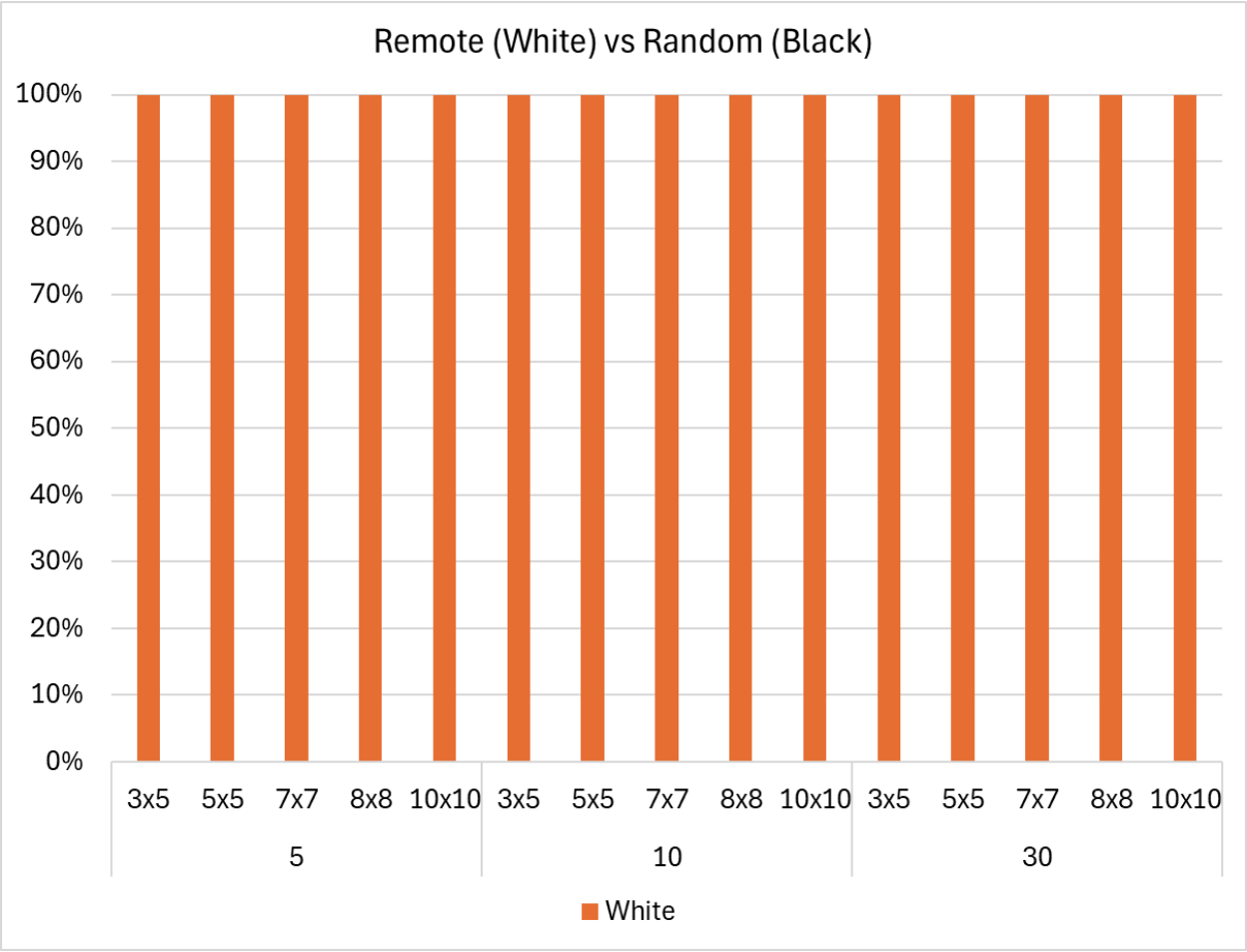


Figure 2: Winner ratios: for each board, playclocks 5s, 10s and 30s, remote White vs random Black

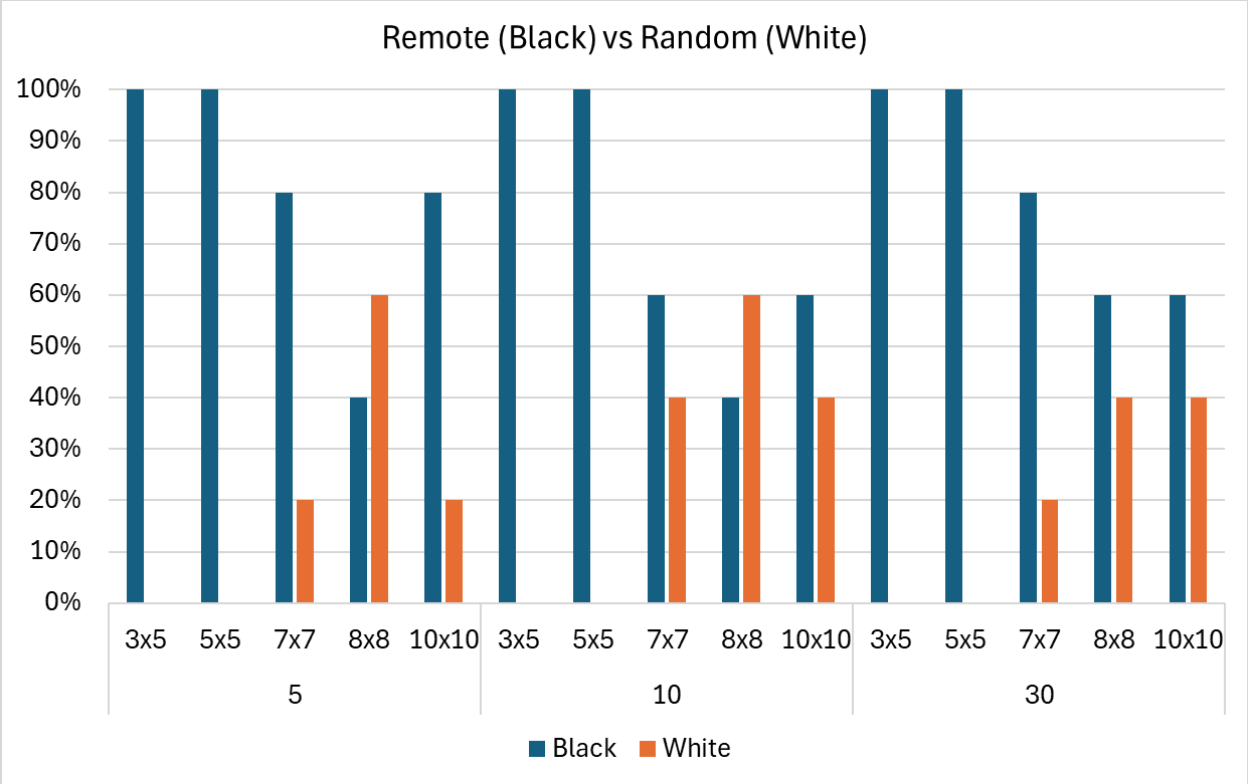


Figure 3: Winner ratios: for each board, playclocks 5s, 10s and 30s, remote Black vs random White

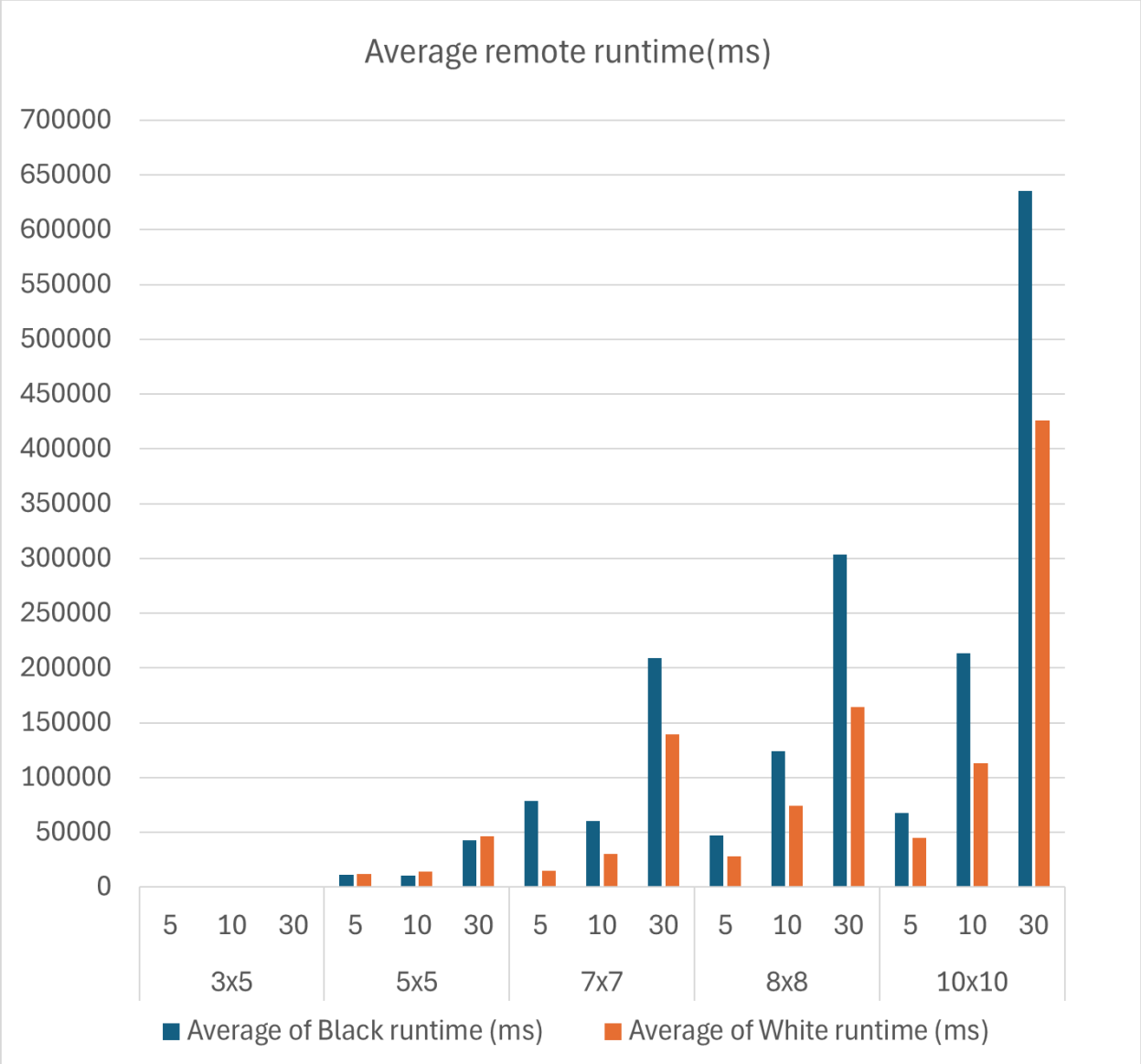


Figure 4: Runtime for playclocks 5s, 10s and 30s

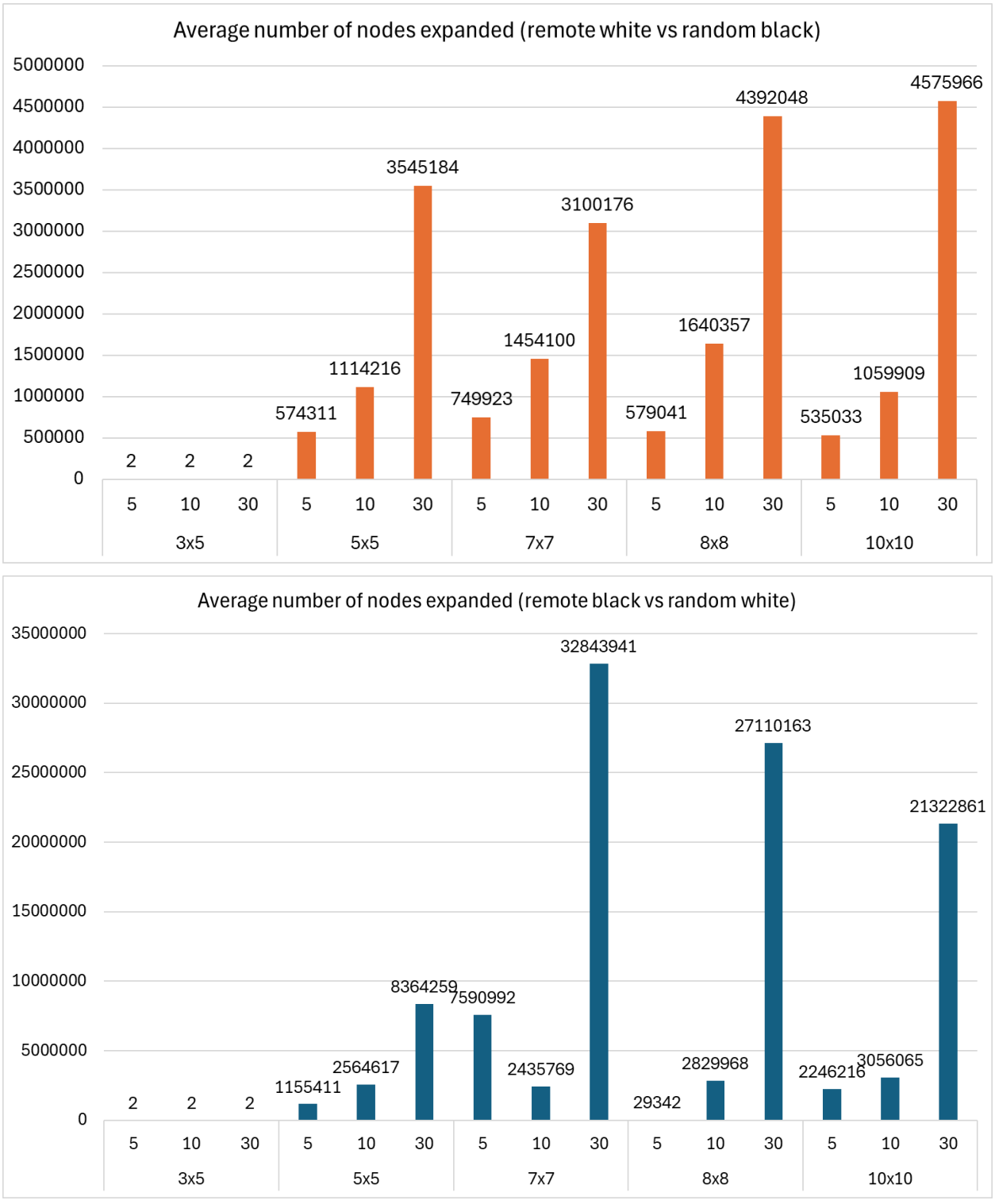


Figure 5: Average number of nodes expanded for playclocks 5s, 10s and 30s (with pruning) for the entire game

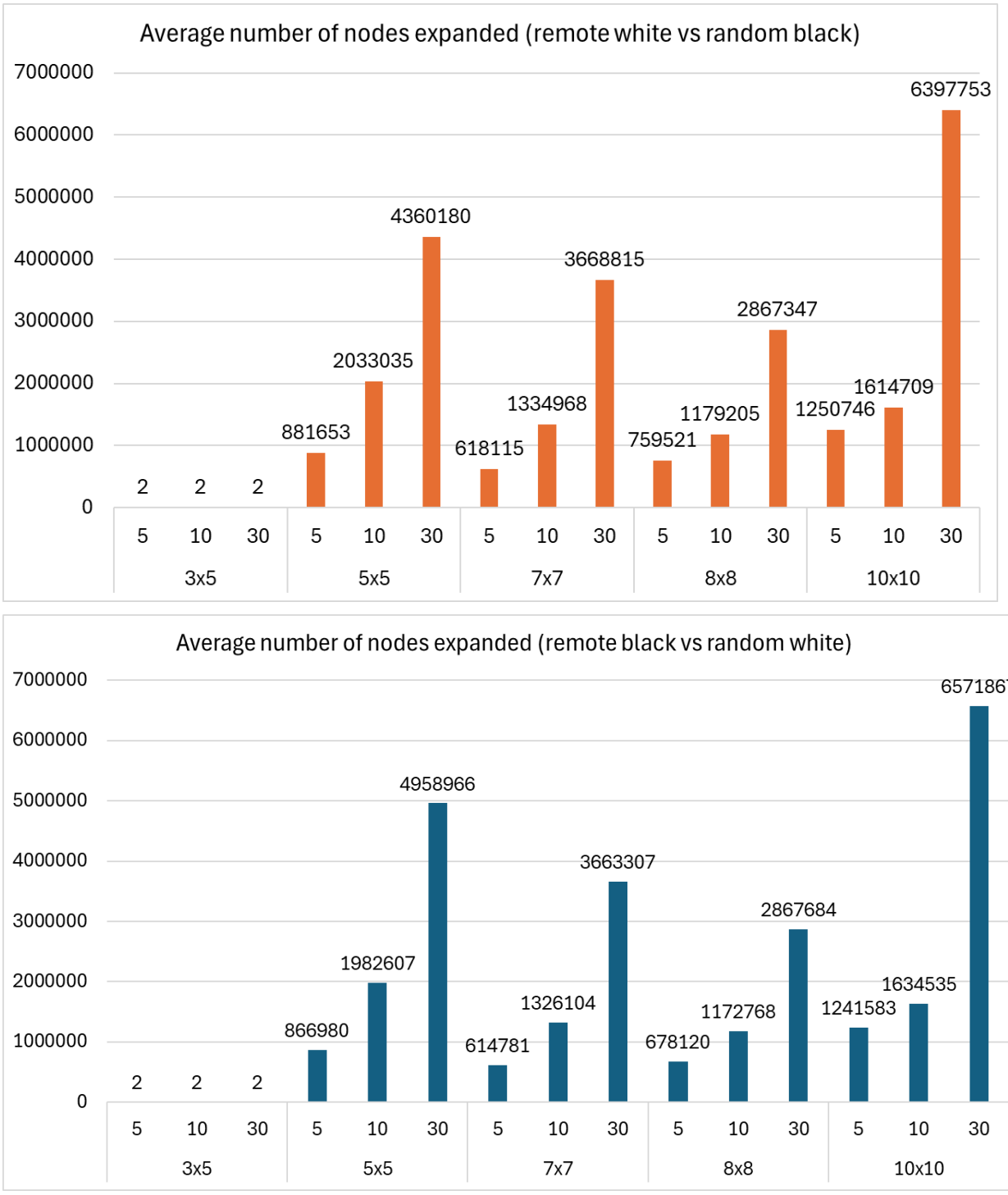
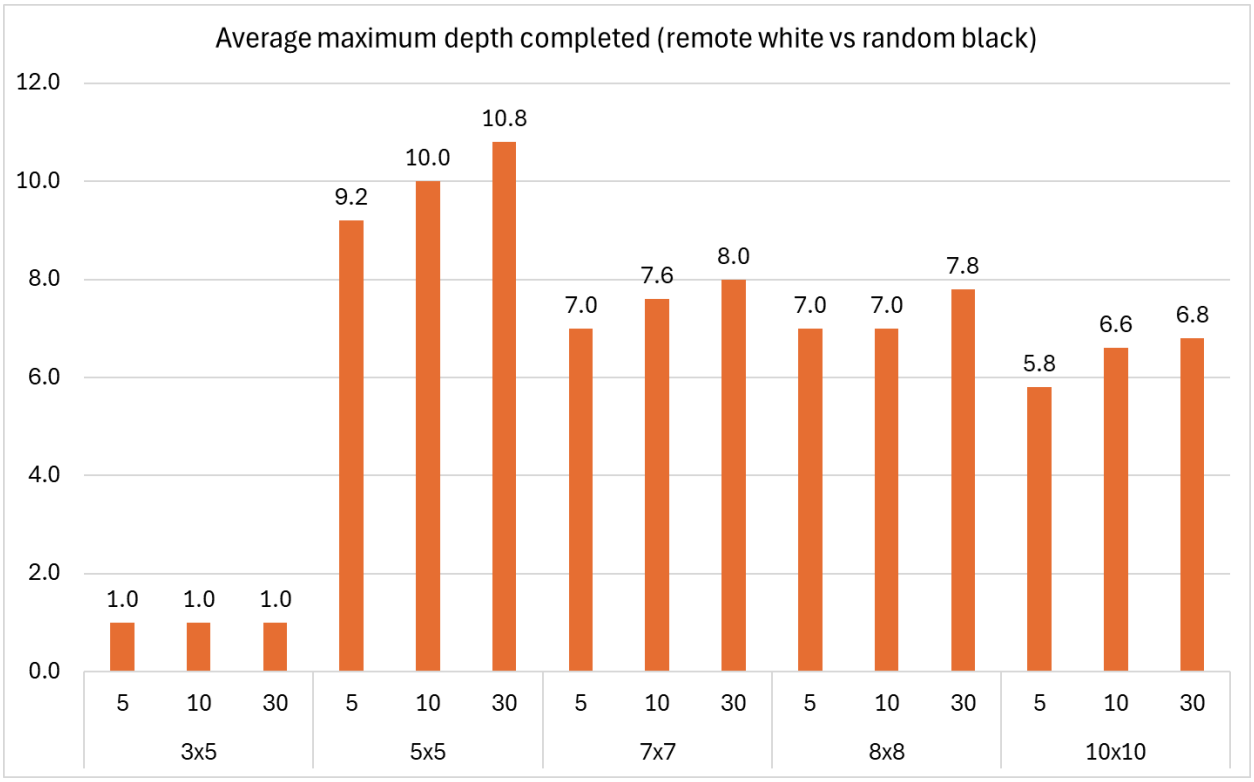


Figure 6: Average number of nodes expanded for playclocks 5s, 10s and 30s (w/o pruning) for the entire game



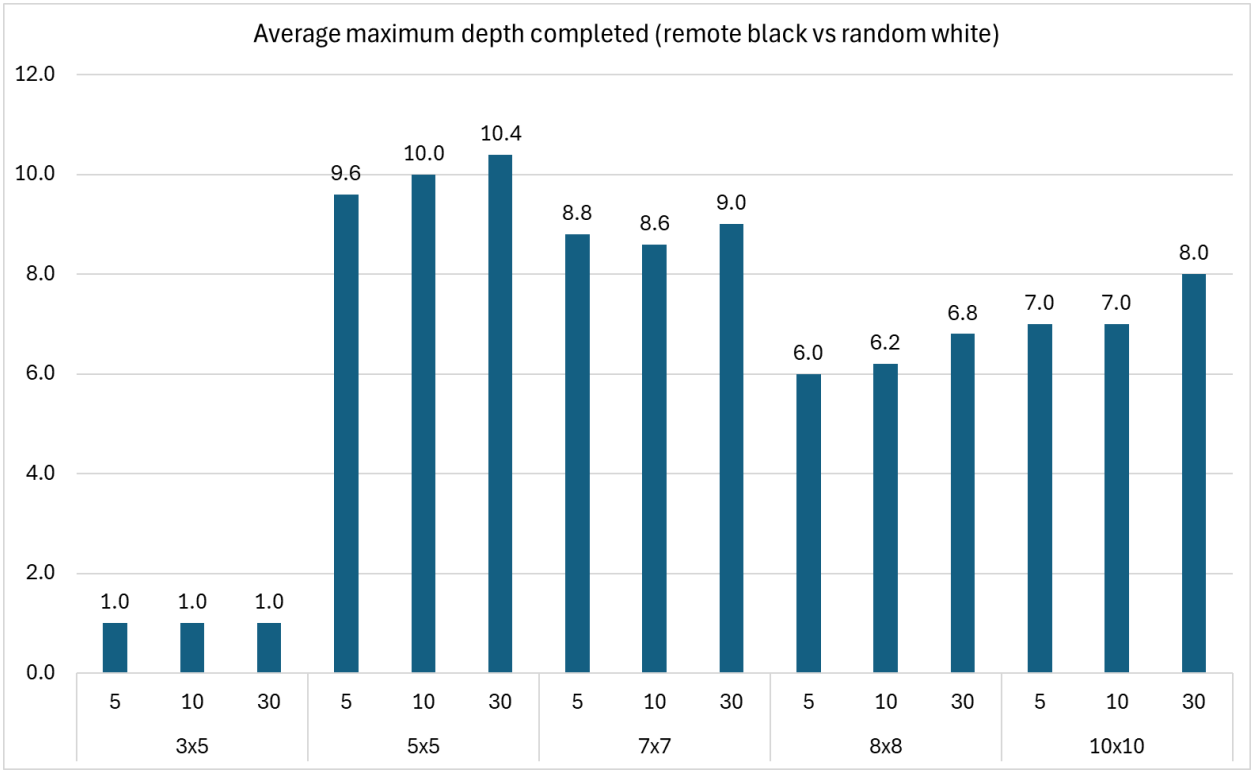
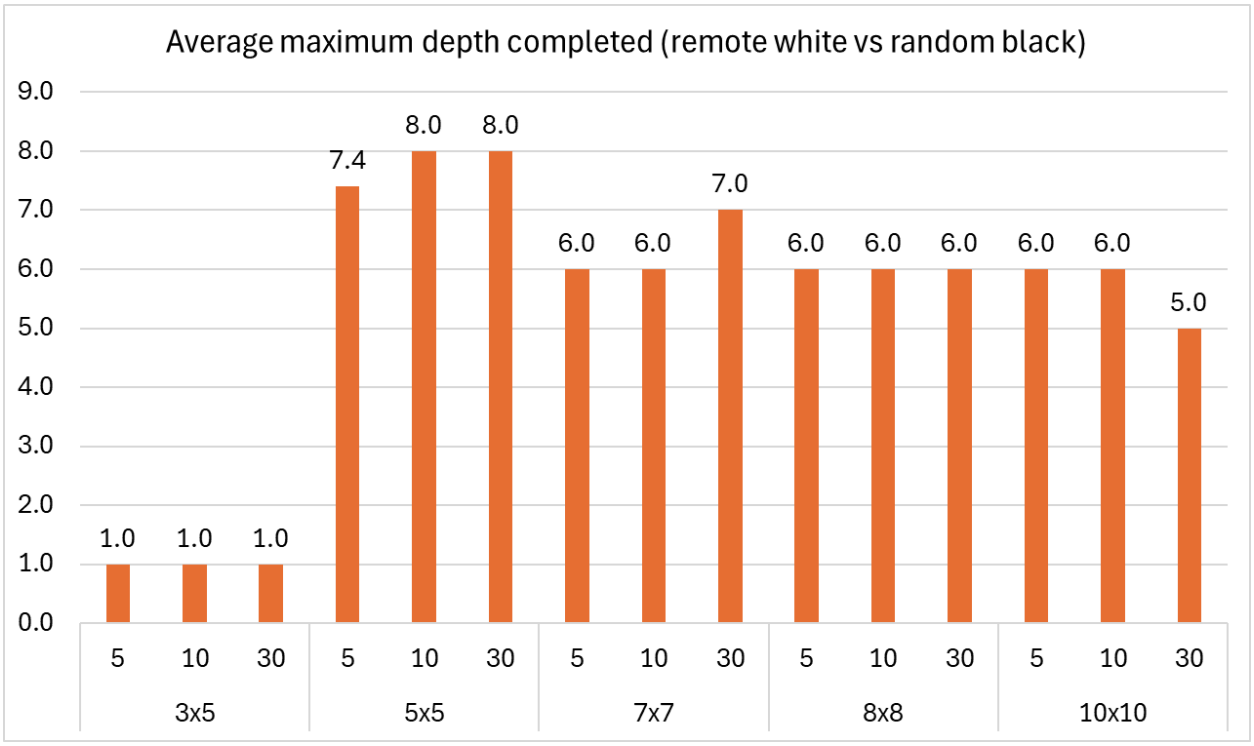


Figure 7: Average maximum depth completed for playclocks 5s, 10s and 30s (with pruning)



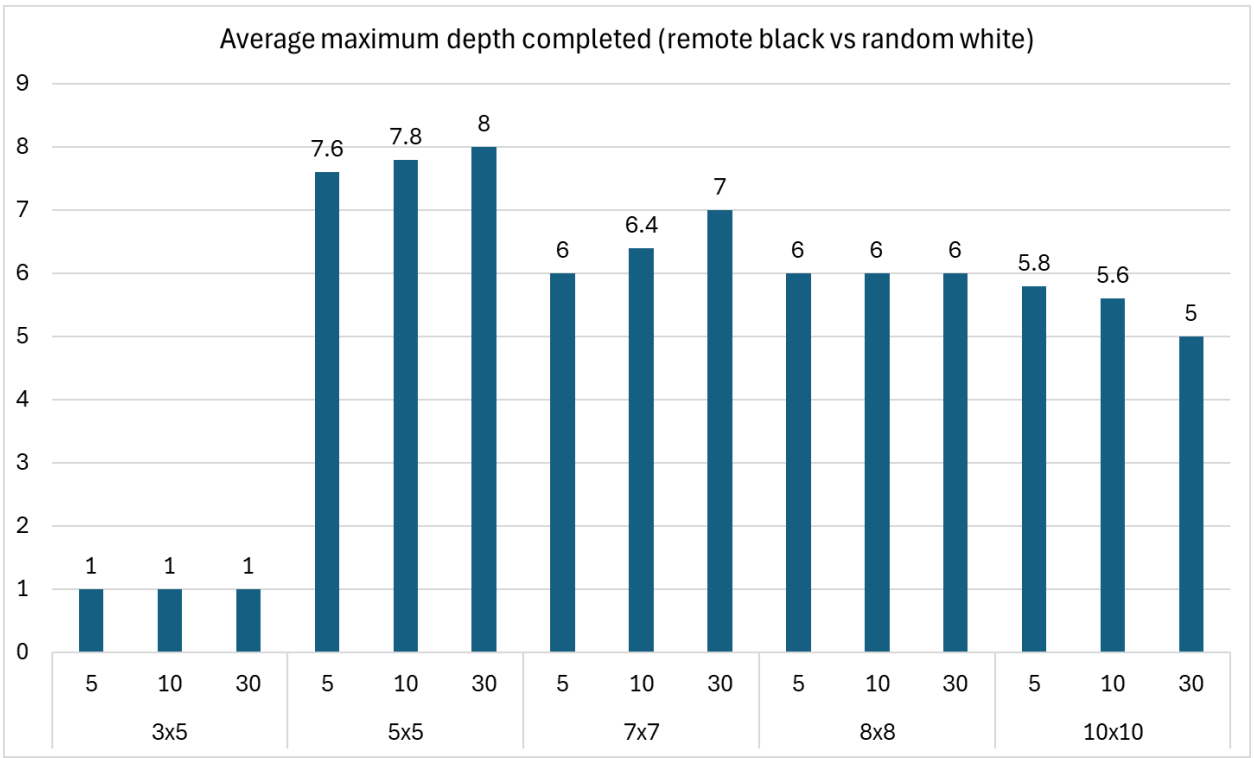


Figure 8: Average maximum depth completed for playclocks 5s, 10s and 30s (w/o pruning)

Game	Startclock	Playclock	Player Type	Player Role	Opponent Type	Opponent Role	Winner
1	10	5	Remote	White	Random	Black	Black
2	10	5	Remote	White	Random	Black	White
3	10	5	Remote	White	Random	Black	Black
4	10	5	Remote	White	Random	Black	Black
5	10	5	Remote	White	Random	Black	White
6	10	10	Remote	White	Random	Black	Black
7	10	10	Remote	White	Random	Black	White
8	10	10	Remote	White	Random	Black	Black
9	10	10	Remote	White	Random	Black	White
10	10	10	Remote	White	Random	Black	Black
11	10	30	Remote	White	Random	Black	Black
12	10	30	Remote	White	Random	Black	Black
13	10	30	Remote	White	Random	Black	Black
14	10	30	Remote	White	Random	Black	White
15	10	30	Remote	White	Random	Black	White
16	10	5	Remote	Black	Random	White	Black
17	10	5	Remote	Black	Random	White	Black
18	10	5	Remote	Black	Random	White	Black
19	10	5	Remote	Black	Random	White	White
20	10	5	Remote	Black	Random	White	Black
21	10	10	Remote	Black	Random	White	White
22	10	10	Remote	Black	Random	White	Black
23	10	10	Remote	Black	Random	White	Black
24	10	10	Remote	Black	Random	White	Black
25	10	10	Remote	Black	Random	White	Black
26	10	30	Remote	Black	Random	White	Black
27	10	30	Remote	Black	Random	White	Black
28	10	30	Remote	Black	Random	White	Black
29	10	30	Remote	Black	Random	White	Black
30	10	30	Remote	Black	Random	White	Black
						Black Win Ratio	**0.7333**
						White Win Ratio	**0.2667**

Figure 9: Win result for 7x7 board with ordering

4. Interpretation of results

The outcomes indicated a clear bias in the algorithm towards White (refer to Figure 1). Specifically, as the size of the board increases (5x5, 7x7, 8x8 and 10x10), the algorithm's accuracy improves for the white side. Conversely, on smaller boards (particularly 3x5), the algorithm tends to be more accurate for the black side. This pattern emerged from testing our bot against a remote black opponent, revealing a tendency for the algorithm to favor one side over the other. The side that benefits from this bias varies: on larger game boards, the white side is more likely to win, whereas on smaller boards, the advantage shifts to the black side. The outcome of the game is thus rather deterministic when the evaluation function is pitted against each other as the sequence of moves played by the evaluation function should be roughly the same (due to some noise, some runs might have more or less node expansions than others).

From Figure 3, it is clear that the evaluation function is weaker if the player is Black. This is especially so for larger boards, namely the 7x7, 8x8 and 10x10. This aligns with the observation from the experiments for remote vs remote where the evaluation functions tend to be weaker for Black for bigger boards.

However, the algorithm demonstrated significant strength on larger boards, overwhelmingly outperforming random bots. This effectiveness is highlighted by a win rate approaching 100% as the board size increases, as detailed in Figure 2 and Figure 3.

We also noted that there was a decrease in the number of nodes expanded (refer to Figure 5 and Figure 6) and a deeper depth was reached in the search tree (deeper by 1 depth) if pruning was used, as detailed in Figure 7 and Figure 8. However, we can see that without pruning the algorithm reaches around depth 6 for White and depth 5 for Black. It is also noted that for a 10x10 board (large enough board), the maximum depth is approximately the same as that of without pruning. The algorithm’s ability to reach deeper depth could be a result of efficient pruning and hence in a given time frame, the pruning algorithm would discard irrelevant and less promising paths allowing it to focus on exploring more nodes in more critical parts of the game tree. This focus could have resulted in more nodes expanded with pruning than without. We also noted that the number of nodes expanded was for the entire game which could have also resulted in some noise being produced due to the different length of the game caused by playing against a random player. This could have resulted in outliers or spikes in the number of nodes expanded for certain trials. Due to time limitations, we were unable to test further to gather more data to result in more accurate values.

Referring to Figure 9, we also noted that with the custom ordering implemented for White turn such that the legal moves of those pieces closer to the Black baseline were evaluated first, resulted in a tendency for White to lose. From observations during the experiments, we could see that with this move ordering, the White pieces at the friendly baseline tend to move forward first. This breaks our strategy of keeping the defensive line at the friendly baseline to minimize the number of possible actions that the enemy can take to reach the friendly baseline and win the game. Hence, this could be a possible reason for the tendency for White to lose the game

with this move ordering. As such, with the implemented evaluation function, using this custom move ordering of prioritizing pieces closer to the enemy baseline may not be an ideal strategy.

5. Short description and evaluation of any improvements made to the program

Initially, our algorithm was equipped with a limited set of evaluation functions. Over time, we expanded this, integrating additional evaluation functions that significantly enhanced the algorithm's accuracy in predicting optimal moves.

To more efficiently represent the game's various states, we opted to use a 2D matrix representation instead of a hash set. This change was motivated by the lower time complexity associated with 2D matrices, which proved particularly advantageous for managing our game's smaller states, such as those in a 10x10 grid. This adjustment not only streamlined state management but also improved the algorithm's overall performance by reducing computational overhead.

Furthermore, we adopted the alpha-beta pruning technique within our Minimax algorithm to significantly cut down on the number of game states that was suboptimal. This method effectively ignores branches of the game tree that won't impact the final decision, thereby reducing computation time without sacrificing the quality of the outcome. As such, this has led to an increase in the depth reached by the state tree, resulting in a more optimal move.

Additionally we hypothesized that a move to a piece closer to the enemy's baseline is more likely to lead to a winning state. Hence, we experimented with the legal move ordering such that the pieces closer to the enemy baseline are evaluated first. However, this strategy was found to have a negative effect on the outcomes of white's score (White win ratio was 0.2667 while Black win ratio was 0.7333) and hence was not used.

We also implemented an iterative deepening strategy, which enhances the algorithm's ability to perform a thorough yet efficient search by gradually increasing the search depth while keeping within the time restrictions. This approach balances the depth of search with computational feasibility, ensuring that deeper levels of the game tree are explored as needed without prematurely consuming computational resources.

The next improvement is to immediately play the move and to skip the evaluation function if there is a winning move such as an available move to the enemy baseline. The algorithm also immediately plays a defensive move to capture any enemy opponents that threaten the friendly baseline, that is the enemy is in the row one tile ahead of the friendly baseline.. These implementations ensure that the friendly AI is rational, attempting to deny any opponent's winning move or attempting to win the game if a winning move exists.

Lastly, the capability to undo moves after their evaluation was introduced, allowing our algorithm to revert to previous game states for further exploration. This feature is crucial for accurately assessing the potential of various moves by considering different outcomes without committing to irreversible changes.