Joe Shymanski 3/29/21

Project 3: Game Tree Search (Minimax and Alpha-Beta Pruning)

Objective

The objective of this project is to implement the Minimax algorithm with an Alpha-Beta Pruning option as well as an appropriate evaluation function for an adversarial soccer simulator with discrete moves.

Implementation

Minimax is fairly simple to implement. It accepts three arguments: the game state, the deciding player, and the current depth of recursion. It then enters a recursive loop between two functions: one which selects the next move with the maximum utility and one which selects the one with the minimum utility. Each of the functions accepts a game state, the deciding player, and the current depth of recursion. It then checks to see if the state it has been given is terminal or at the maximum depth. If either condition is met, it returns a value according to the evaluation function, which will be explained later. If not, it loops through every available action according to the state it has been given and generates a new state that results from taking that action. It then passes this new state, the deciding player, and the current depth plus one to the complementary function. That complementary function will eventually return an action and a value, and the current function should store and ultimately return the action that yields the highest or lowest utility.

Minimax with Alpha-Beta Pruning uses the same complementary functions as before, just with two extra inputs: alpha and beta. In the maximizing function, if a new action is stored since its utility is higher than the current action, then alpha is calculated to be the maximum of the current alpha value or the newly obtained action value. If the resulting alpha value is greater than

or equal to the beta value, no more actions need to be observed. In the minimizing function, if a new action is stored since its utility is lower than the current action, then beta is calculated to be the minimum of the current beta value or the newly obtained action value. If the resulting beta value is less than or equal to the alpha value, no more actions need to be observed.

Lastly, an appropriate evaluation function must be created to complete the Minimax algorithm. It accepts two inputs: a game state and the deciding player. First, it checks to see if the given game state is terminal. If so, it will save an incredibly large reward value which is positive if the player has won and negative if the player has lost. It will immediately return the large positive reward value if the player has won but will continue on if the player has lost. The reason for this is so the algorithm continues to choose good actions even if it is in a futile state in the hopes that a human opponent will choose a suboptimal action that will keep it in the game.

Next, the function checks to see if the player has the ball. If they do, then it returns a value calculated by subtracting the player's current distance to the goal from the diagonal distance of the field and then adding the large negative reward from earlier, if there is one. This value is always greater than 1 if there is no large negative reward and increases as the player gets closer to the goal.

If the player does not have the ball, the function checks to see if the opponent has the ball. If so, then it calculates the midpoint between the opponent and the center of the goal. Then the distance between the player and this midpoint is calculated. It is given a negative sign and is then added to the large negative reward from earlier, if there is one. This value is always negative and increases toward 0 as the player gets closer to the midpoint. I tried using the centroid of the triangle formed between the opponent and the two goalposts, but this caused the

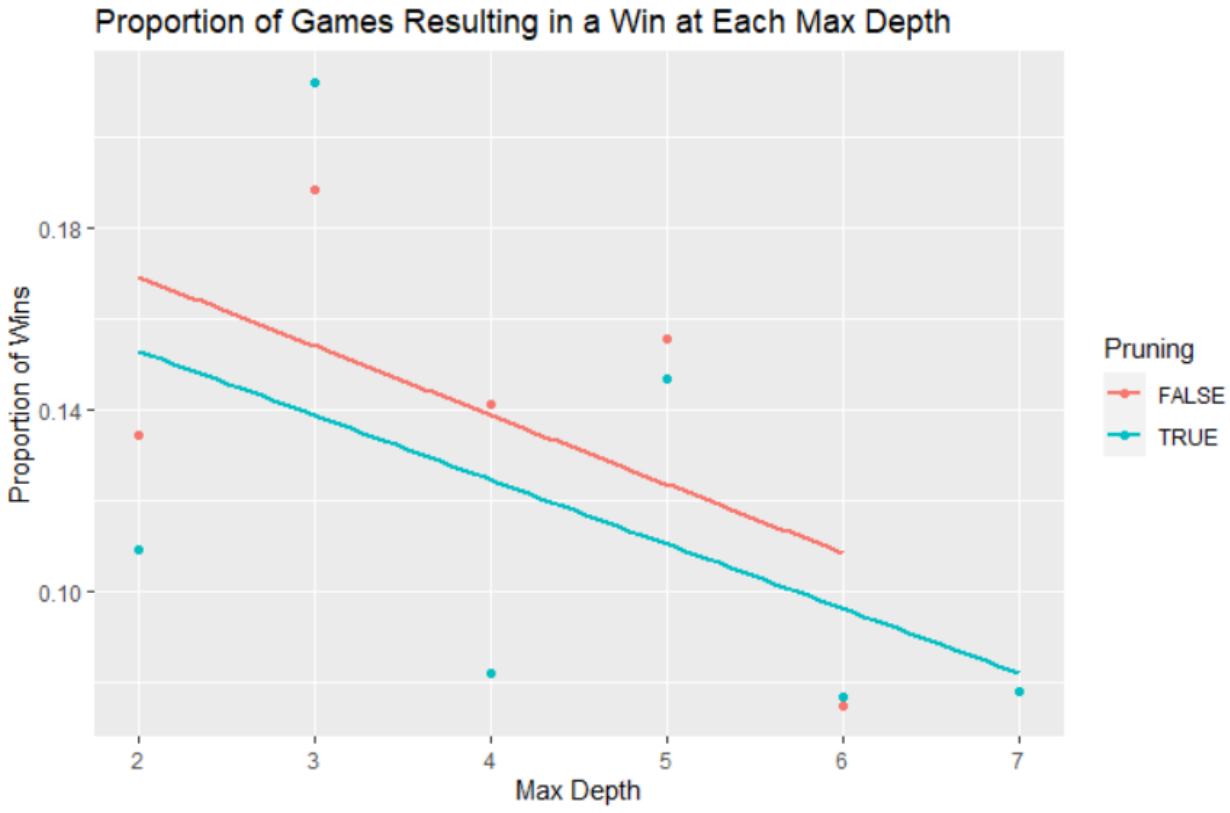
agent to favor moving closer to the end lines too often instead of jumping between the opponent and the goal to block a shot.

Lastly, if neither agent has the ball, then a small positive value is calculated by taking the inverse of the distance between the player and the ball. It is then added to the large negative reward if one exists. This resulting value is returned by the function.

Thus, the evaluation function creates a distinct hierarchy between states in the game. Winning the game is most preferable. Then possessing the ball as close to the goal is most preferable. Then being as close to the unpossessed ball is most preferable. Then being as close to the midpoint between the opponent, who has the ball, and the center of the goal is most preferable. The least preferable action is obviously losing.

Results & Analysis

Figure 1



As you can see from Figure 1, as maximum depth increases, the proportion of games resulting in a win generally decreases. However, the actual data points seem to vary sporadically, almost in a sinusoidal pattern. Unfortunately, the higher the maximum depth, the longer each data point takes to collect, so the later depths have much fewer data points than the earlier ones. This is why there is no data collected without Alpha-Beta Pruning at a depth of 7.

This inverse relationship between maximum depth and proportion of wins seems reasonable. As the agents see further into the future before taking each action, they can realize if the other agent will reach the ball first and defend more quickly. We would expect nearly all games to end in a draw at high levels of maximum depth.

Figure 2

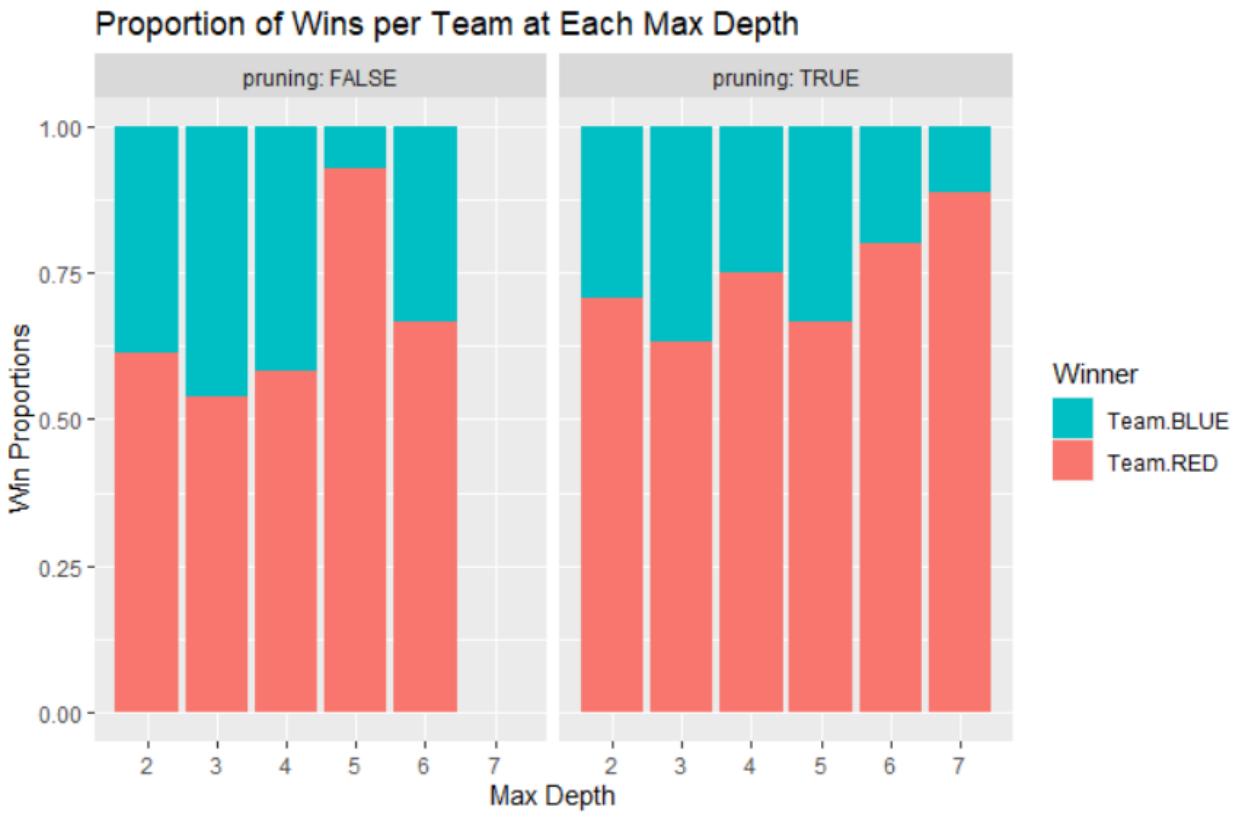
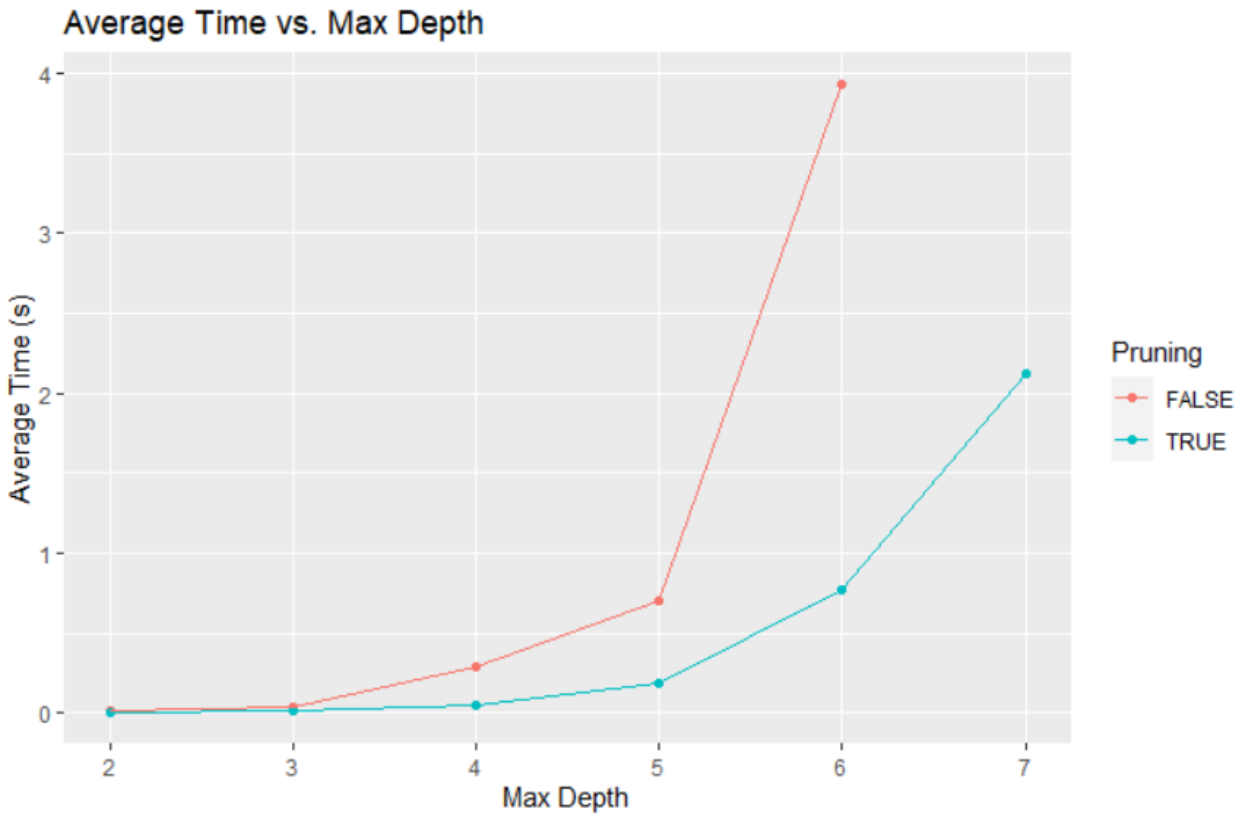


Figure 2 demonstrates that the red player is winning the majority of games at all depths and regardless of Alpha-Beta Pruning. This conclusion also seems reasonable, since red takes the

first move in the game, causing it to reach the ball more quickly than blue on average. It is not entirely clear if the red player's share of won games increases or decreases as maximum depth increases.

Figure 3



The relationship between average time per move and maximum depth appears to be exponential, as shown in Figure 3. Unsurprisingly, moves using Alpha-Beta Pruning take much less time than those without Pruning at the same maximum depth.

Figure 4

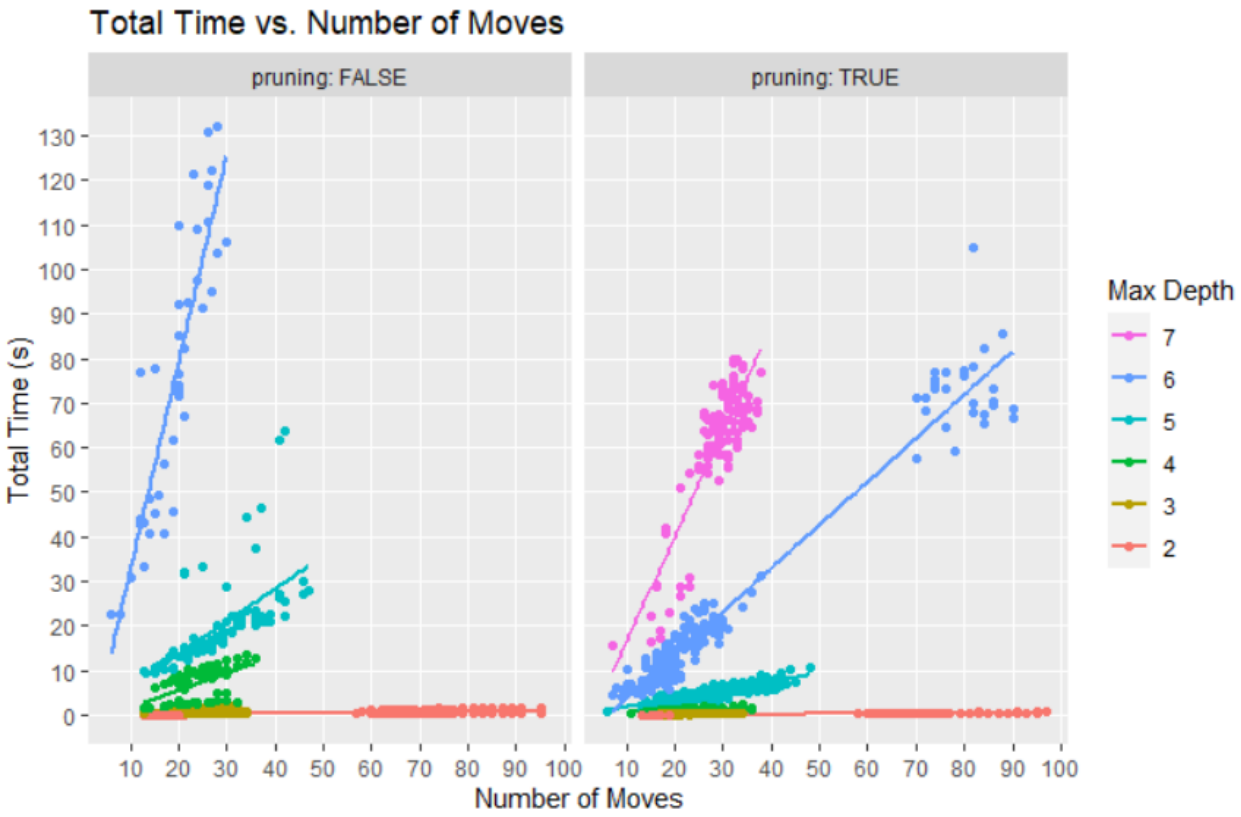
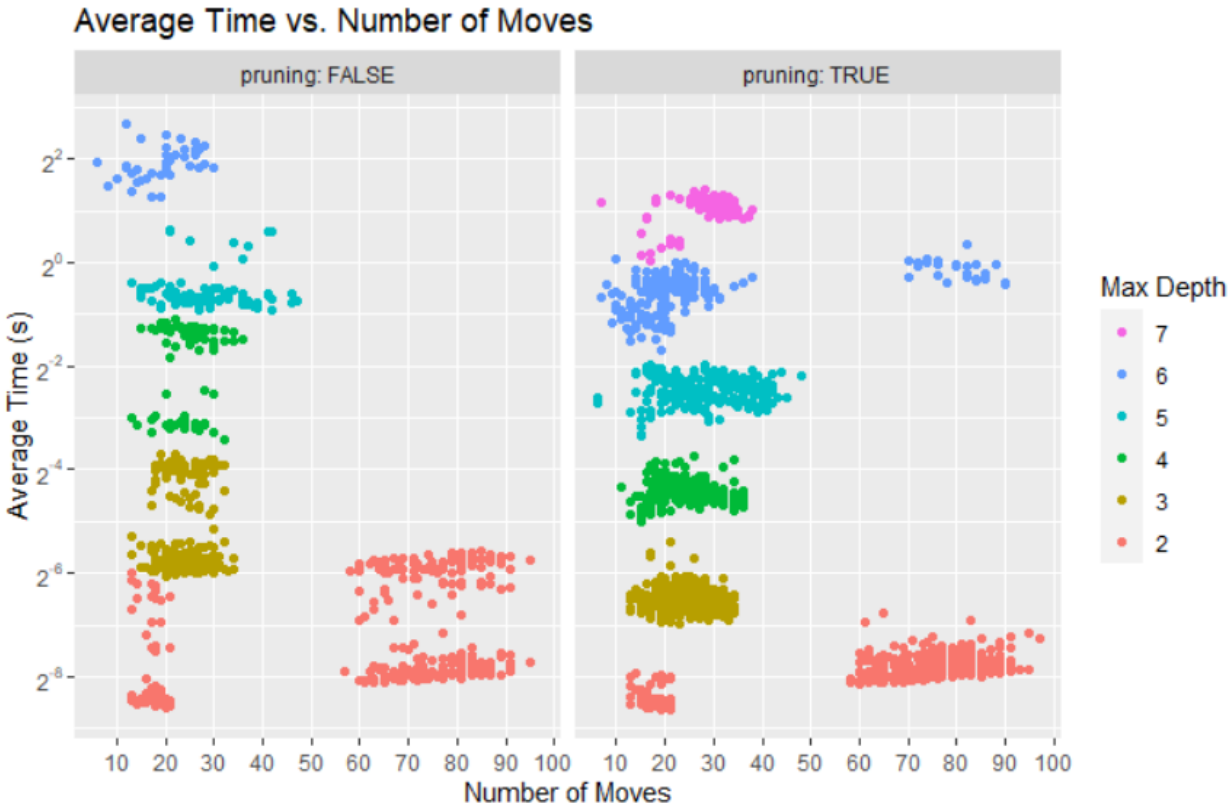


Figure 4 shows the distribution of points, each of which represents a single game played, according to their maximum depth. As you can see, the total time needed to complete each game is linearly related to the number of moves in the game. The slopes of these linear relationships increase as maximum depth increases. In fact, they increase exponentially, since these slopes represent the average time per move at each depth, and Figure 3 shows that their relationship is exponential. The slopes are much steeper without Alpha-Beta Pruning, which is also consistent with Figure 3.

Figure 5



Since average time per move is exponentially related to maximum depth, we can visually separate the data at each depth using a logarithmic scale. Figure 5 demonstrates this clear separation when plotting average time per move against the number of moves. For some reason, games played without Alpha-Beta Pruning seem to demonstrate a distinct separation within each maximum depth at low depth levels. Another interesting observation is that the more moves in the game, the higher the average time per move. This is likely due to the fact moves near the end of the game take less time to compute, since they are nearing a terminal state. Thus, the shorter the game, the closer to a terminal state each move is, and the longer the game, the farther from a terminal state each move is.

Figure 6

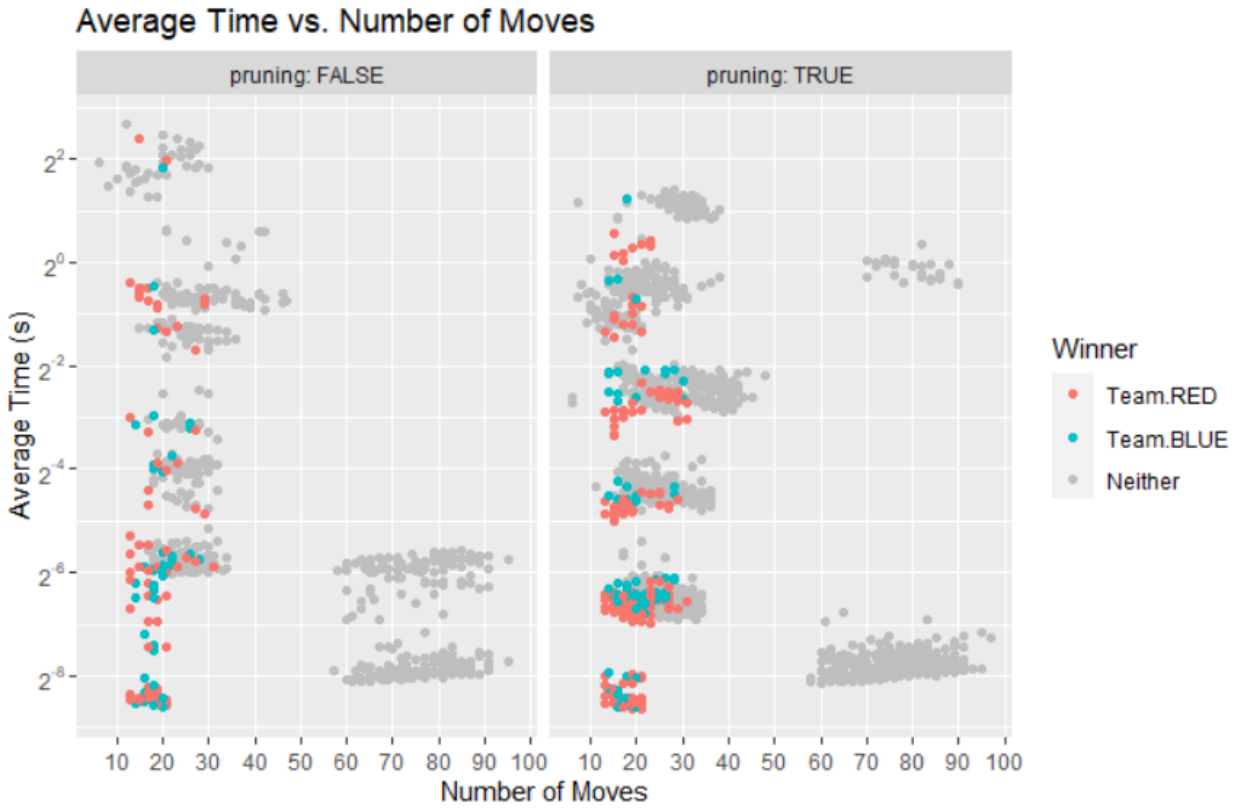
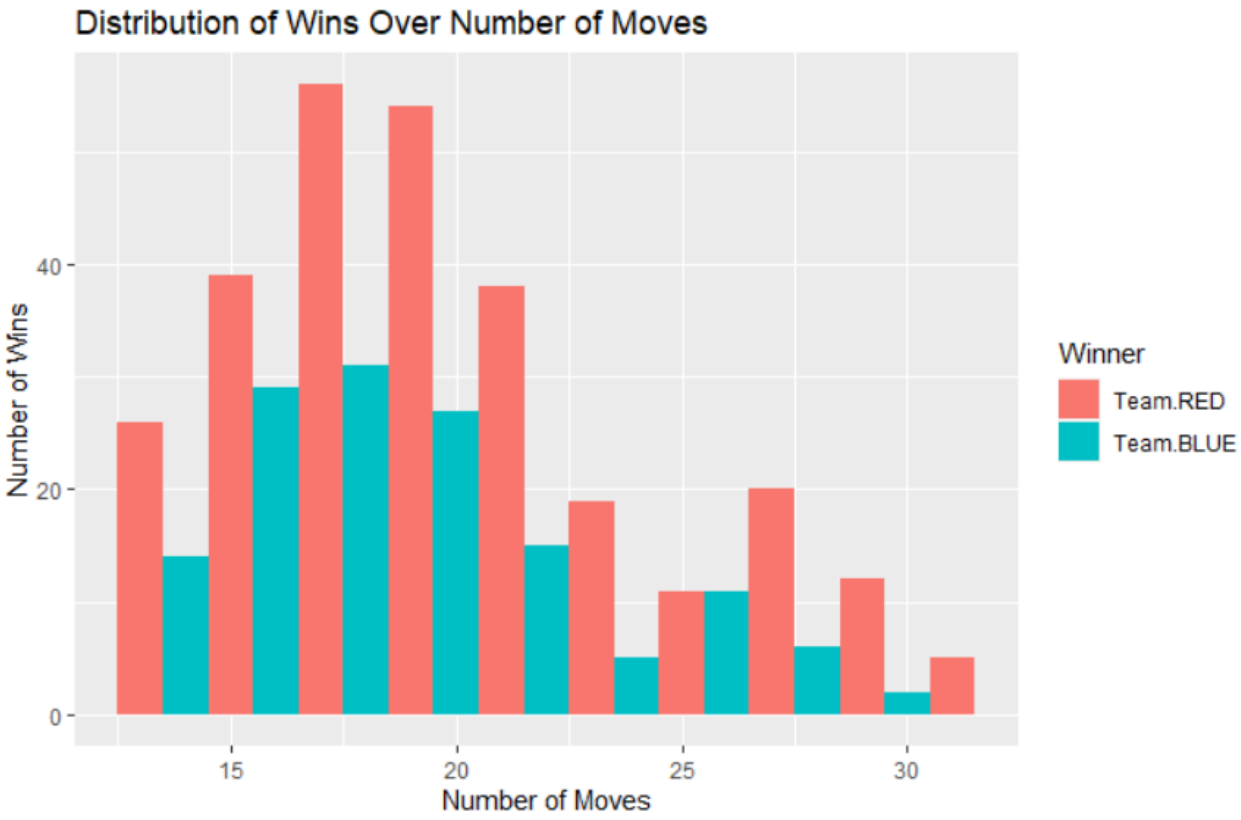


Figure 6 is almost identical to Figure 5, but instead it colors the points according to their winner. All games which have a winner are won with a relatively small number of moves. This is because the only real way to win against another Minimax agent is to be randomly assigned a favorable starting position while your opponent is assigned an unfavorable one. Then you can immediately take the ball and streak down the field faster than your opponent can recover and defend your progress. If no one can score off the opening possession, the game is guaranteed to end in a draw. Most games end immediately after this first possession, as a state in the game is repeated or the game is won. However, at maximum depths of 2 and 6, secondary groupings form around 60 to 100 moves. For whatever reason, the agents at these depths turn the ball over and begin defending, after which the game finally ends as a draw. In fact, all games with a

maximum depth of 2 either end in a win after one possession or a draw after multiple possessions. This is most likely due to these agents acting most like greedy agents.

Figure 7



In Figure 7, the distribution of wins over the number of moves appears to be bimodal, with peaks around 17-18 moves and 26-27 moves. The reason for this is likely due to the two distinct outcomes of games ending in a win. Outcome 1 occurs when one agent is placed very close to the ball and the other is placed so unfavorably that it has no chance to defend at all. The attacking agent picks up the ball, sprints to the shooting area, and kicks a goal immediately. Outcome 1 only takes about 17-18 moves, on average.

Outcome 2 occurs when one agent is placed somewhat favorably near the ball and the other agent is placed somewhat unfavorably away from the ball. The attacking agent picks up the ball and streaks to the shooting area. The defending agent does its best to recover and stay

between the attacker and the goal. The attacker is deflected to the side a bit by the defender and both crash toward the net. However, by the time the attacker reaches the corner of the net, the defender is incapable of pushing the attacker out any further. The attacker walks the ball into the net, with the defender merely one block away. Outcome 2 takes about 26-27 moves, on average.

Figure 8

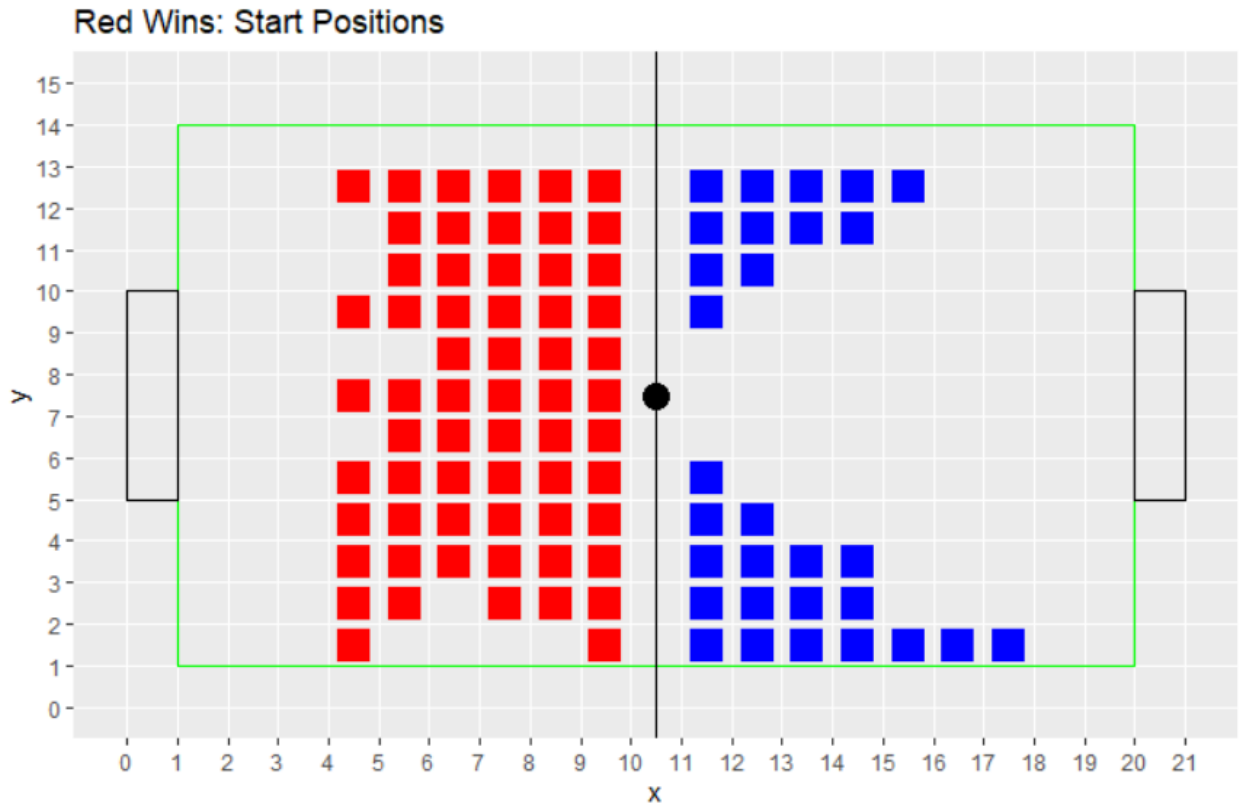


Figure 8, along with each subsequent figure, demonstrates the position of each agent on the field in the event of a win. One thing to notice in these graphs is that no player ever starts out in the topmost row of the field. This is due to a slight bug in the initial position code which was only revealed to me after creating these graphs. I did not have time to revise the code and recollect data. Nonetheless, the above graph demonstrates that red wins come in the form of the red agent spawning within 6 columns of the midline and the blue agent spawning near the sidelines by midfield.

Figure 9

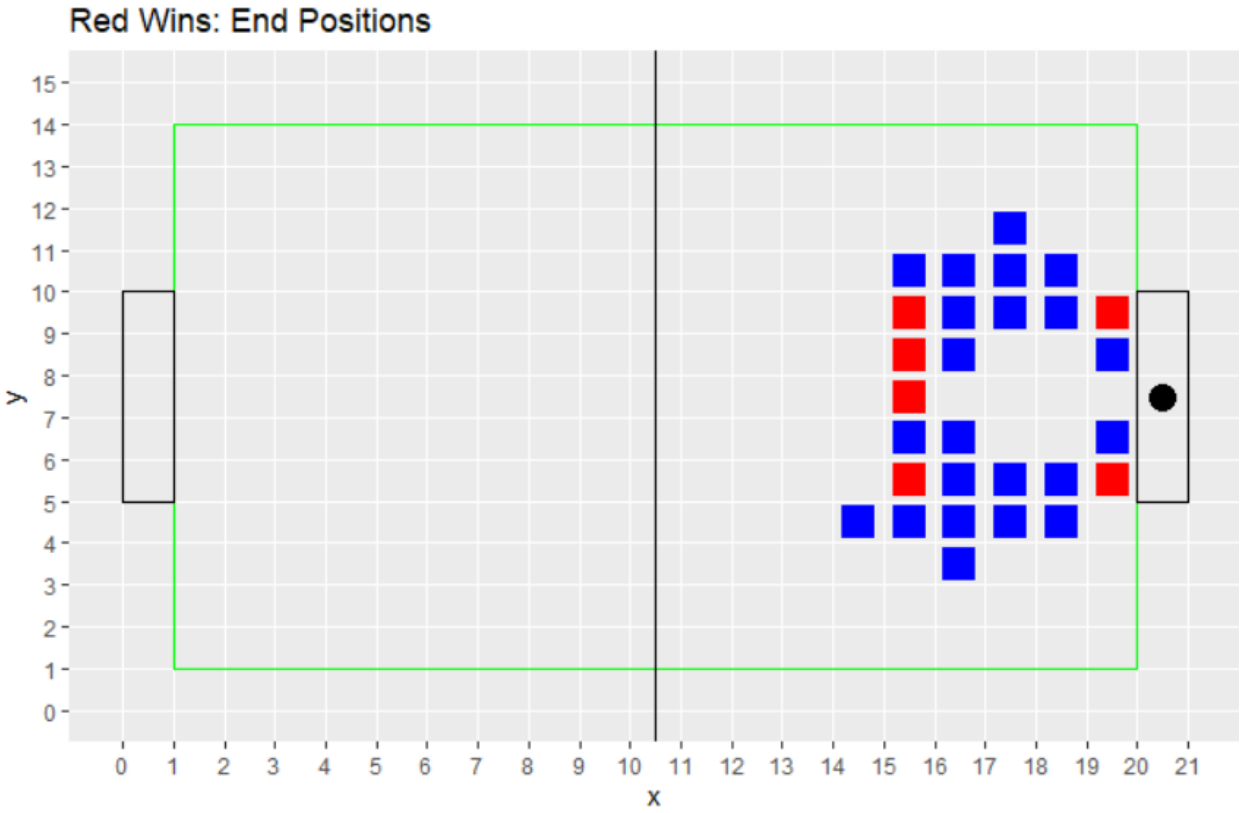


Figure 9 shows the end positions of both agents after red has won. The blue player is never in the middle of the shooting area, since this would almost certainly result in a blocked shot. The red player is either at the top of the shooting area, which results from Outcome 1, or at the corners of the goal, which results from Outcome 2.

Figure 10

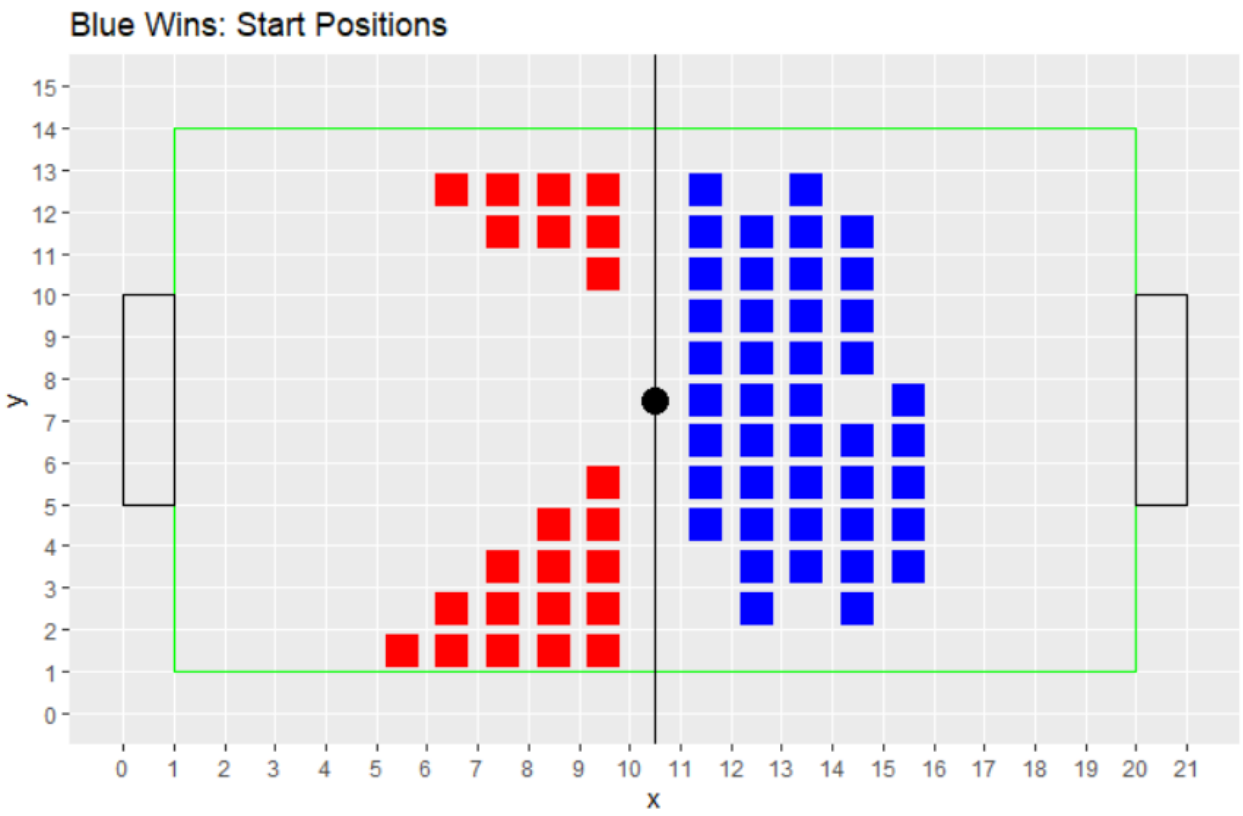


Figure 10 shows much of the same that Figure 8 showed. However, the blue agent is always within 5 columns of the midline, one less than the 6 allowed by the red agent. This is likely due to the fact that the blue player is always a single move behind the red player.

Figure 11

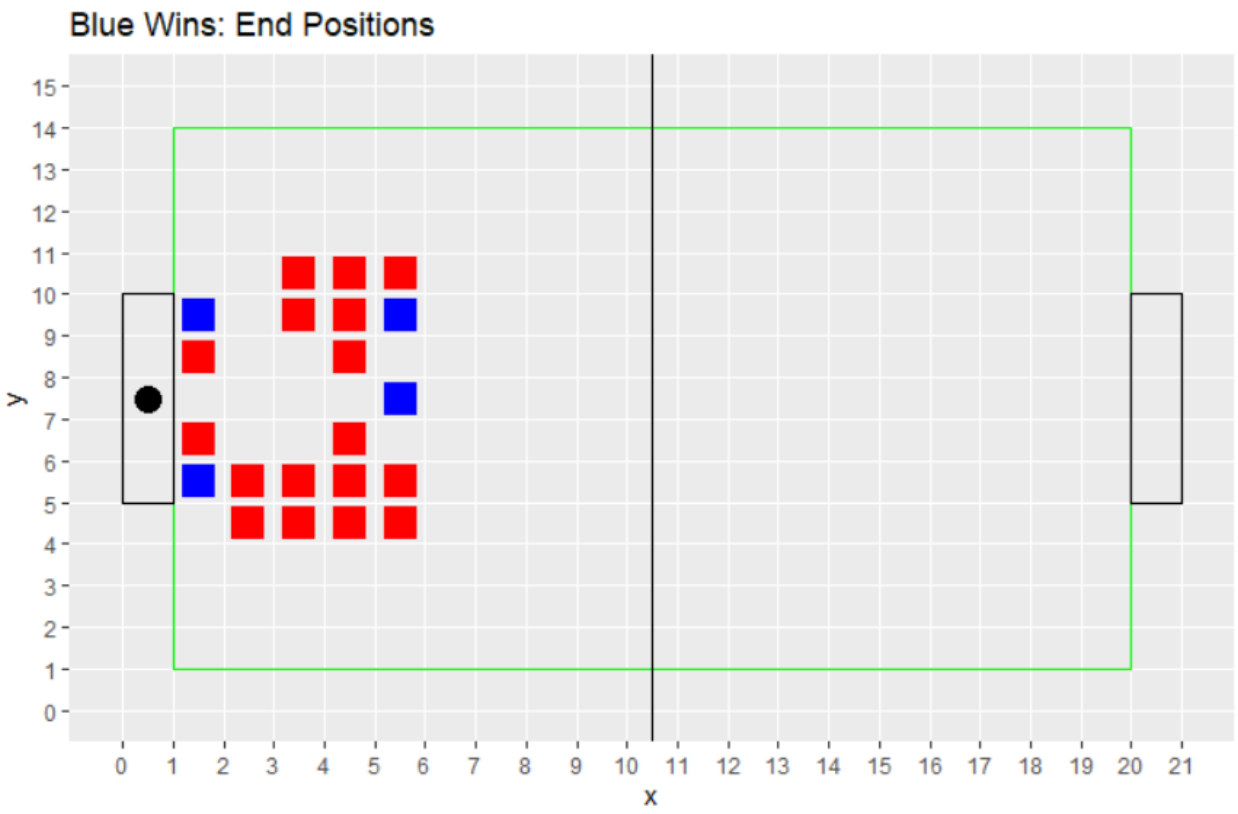


Figure 11 also shows the same characteristics that Figure 9 shows. The red agent is never in the center of the shooting lane after a blue win. The blue agent is always at the top of the shooting area from Outcome 1 or at the corners of the goal from Outcome 2.

Conclusion

In this domain of discrete soccer, between two Minimax agents with the specified evaluation function, most games end in a draw. More games generally end in a draw the higher the maximum depth of recursion. The average time an agent takes to make a move is exponentially related to the maximum recursion depth. This decision time also increases significantly if the Minimax algorithm does not utilize Alpha-Beta Pruning. Finally, all games ending in a win occur in the first possession of the game according to two distinct outcomes described as Outcome 1 and Outcome 2.