Jasper Shen and Ethan Cuthrell
4/18/2025

Mancala Project Writeup

Introduction

For our project, we are making Mancala according to the rules discussed in class. We have created players with various decision-making techniques such as random, minimax, and alpha-beta. See the Project_Writeup.ipynb file for the implementation of the game along with the various helper functions and code for running the simulations. We use modified versions of the minimax and alpha-beta search adapted from the AIMA Python library, which were from the games4e.py file.

Random Player vs Random Player

To gather data on our program, as well as ensure functionality of our random player, we ran 100 games with two random players where Player 1 always moved first. According to our results, Player 1 won 49% of the time, Player 2 won 48% of the time, and they tied with each other 3% of the time. We also measured the average number of turns each player took and found that Player 1 takes about 21.06 turns per game while Player 2 takes 20.52. The number of turns each player took per game on average is similar for both players, which is expected since they are both making random choices. After comparing the win percentages of Player 1, who moved first, and Player 2, who moved second, we realized that there is a small first-turn advantage because the win ratio between random player 1 and 2 is slightly greater than 50%. This makes sense because the player who moves first in the game of Mancala gets to start building their board first. However, since both players in our tests are making random choices, the first-turn advantage isn't very significant because they don't strategically take advantage of moving first. We also observed that Player 1 requires slightly more average turns per game than Player 2. This supports the idea of the first-turn advantage as Player 1 starts off the game every time, and ends off the game every time they play the winning move, while Player 2 can only match the number of turns as Player 1, even if they play the winning move.

Minimax Player vs Random Player (Variable Number of Plies)

After implementing our AI minimax player, we wanted to test it at a variable number of plies from 1 to 7 to check performance, so we ran another 100 games with a random player and a minimax player. The number of plies was chosen by a random number generator, where the number it generates dictates the maximum number of plies the search algorithm will go for the entirety of a single game. It took about 3 minutes and 38 seconds to simulate these games. According to our results, the minimax player won 91% of the time, the random player won 7% of the time, and they tied 2% of the time. We found that the minimax player took 14.67 turns per game on average, while the random player took 14.12 turns per game on average.

Minimax Player vs Random Player (5 Plies Depth)

We then wanted to test our AI minimax player at a constant number of plies of 5, so we ran another 100 games with a random player and a minimax player. It took about 1 minute to simulate these games. According to our results, the minimax player won 89% of the time, the random player won 10% of the time, and they tied with each other 1% of the time. We also found that the minimax player takes 14.39 turns per game on average, while the random player takes 13.85. Playing two random players against each other resulted in around a 50-50 win rate; however, when we added our minimax player, the win ratio changed drastically, causing the minimax player to win a significant margin of the time. To rigorously demonstrate whether the AI player is better than the random player, we can conduct a binomial test. The AI player won 89 out of the 100 games, so the probability of this happening, or a more extreme event (winning 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, or all 100 out of the 100 games) if the AI player were the same as a random player is:

$$p = \sum_{i=89}^{100} \binom{100}{i} (0.5)^{100} \approx 1.11726 \times 10^{-16} \leq 0.05$$

Since the p-value is significantly lower than the 0.05 threshold, we conclude that our AI player performs significantly better than random chance.

Alpha-Beta Player vs Random Player (5 Plies Depth)

Finally, we wanted to analyze how our alpha-beta AI player would perform, so we ran 100 games with a random player and an alpha-beta player. It took about 11 seconds to simulate these games. According to our results, the alpha-beta player won about 91% of the time, the random player won about 8% of the time, and they tied with each other 1% of the time. We also found that the alpha-beta player takes 14.46 turns per game on average, while the random player takes 13.84. Our average win rate over the course of 100 games against a random player 5 plies deep for our minimax and alpha-beta player is very similar, within 2%, which is to be expected with repeated trials against a random player. Additionally, our average number of moves for the alpha-beta and minimax players is nearly identical. A reason for the small variability in the performances is that the random player is not playing identical moves. Therefore, it appears that our results for our alpha-beta implementation are not different from the minimax player. This makes sense because alpha-beta pruning should not impact the optimal move output from the function, it should only change how the tree is pruned to reduce runtime. This reduction of time taken to run the 100 matches between a random player and an alpha-beta player is seen in the almost minute decrease compared to the runtime of the 100 matches between a random player and a minimax player.

Extra Credit - Alpha Beta Player vs Random Player (10 Plies Depth with Modified Utility Function)

We wanted to try to improve the performance of our Mancala AI even further, so we devised a new utility function and ran 100 games with a random player and an alpha-beta player at a depth of 10 plies. To run a single game at 10 plies, it took about 12 seconds to run. This new utility function only measures the number of stones in the player's Mancala. According to our results, the alpha-beta player won about 96% of the time, the random player won about 3% of the time, and they tied with each other 1% of the time. We also found that the alpha-beta player takes 15.82 turns per game on average, while the random player takes 15.19.

We believe this new utility function improves over the previous utility function as the alpha-beta player should focus on getting a lead over the opponent, irrespective of how many stones they have in their Mancala, since there is a limited number of stones on the board. While the difference between the win rate of the Alpha-Beta player using the provided utility function and the win rate of the Alpha-Beta player using the new utility function is not very different, we see that there is still an increase in the win percentage. This may be an effect of either the increased number of plies or the changed utility function. However, since the objective of Mancala is to have more stones in your Mancala than you opponents at the end of the game, then focusing on getting the most stones in your Mancala is a more direct way to achieve the objective and provides a better way to evaluate the strength of a match than the difference in stones.

Increasing the number of plies to 10 benefits the AI player since it allows them to plan further into all the possible moves to determine the best move to make. It may be the case that the best move was available after the 5th ply, hidden behind a less favorable move, and the AI would have been unable to see it if we still limited it to 5 plies.