Caleb Bettcher and Kenneth Flowe | Final Project Report | 11/4/2025

Data:

*Modified Heuristic shows data for our extra credit modified utility function AB algorithm (part 9)

	Random VS Random	**MiniMax (depth = 5) VS Random**	**ModifiedHueristic (depth = 10) VS Random**
Avg Runtime	<1ms	1.042s	47.2s
Player 1 Wins %	48%	96%	99%
Player 2 Wins %	45%	4%	0%
Ties %	7%	0%	1%
Avg # Turns	31	27.43	25.76
Avg p1 Score	26	35.07	36.55
Avg p2 Score	24	12.93	11.45
	ABPruning (depth = 2) VS Random	**ABPruning (depth = 5) VS Random**	**ABPruning (depth = 10) VS Random**
Avg Runtime	0.011s	0.0264s	54.912s
Player 1 Wins %	94%	96%	99%
Player 2 Wins %	5%	3%	1%
Ties %	1%	1%	0%
Avg # Turns	33.42	28.28	25.43
Avg p1 Score	32.34	34.85	36.16
Avg p2 Score	15.66	13.15	11.84

Questions:

5a) Is your AI player better than random chance? Write a paragraph or two describing or why not

Our AI proved to be way better than random chance when we ran 100 games. During the trials our min max won 96 percent of the time showing our simple algorithm is far superior to simple guessing. Our AI implements a Min Max algorithm that is able to look into possible moves that future opponents can make and choose the most favorable move accordingly. When we have our AI look ahead to future moves it makes the assumption that the opponent will choose the most optimal move by recursively calling itself. In this we made our AI look ahead by 5 moves but If we wanted to we could make this any number.

7a) Are your results for this part different from those for your minimax AI player?

Write a paragraph or two describing why or why not

The results were almost identical in the AB pruning version of the minMax function. The biggest difference is the runtime of the AB pruning algorithm; it's much lower than the minimax algorithm. This makes sense because the algorithm for our AI is pretty much the exact same except we use Alpha Beta pruning to optimize the number of branches searched.

With Alpha Beta Pruning not only is our AI checking for the most favorable, but it also checks if we need to explore every branch of decisions to find a more optimal move and if not we cut the branches off the decision tree with a break in the loop. Due to this we can significantly cut down our runtime, as we are not exploring every possible gamestate.

8a) How much does the Alpha Beta algorithm speed up the game? Compare your run time for 5 ply minimax against 5 ply Alpha Beta. Project how long Minimax would take to run 10 plies.

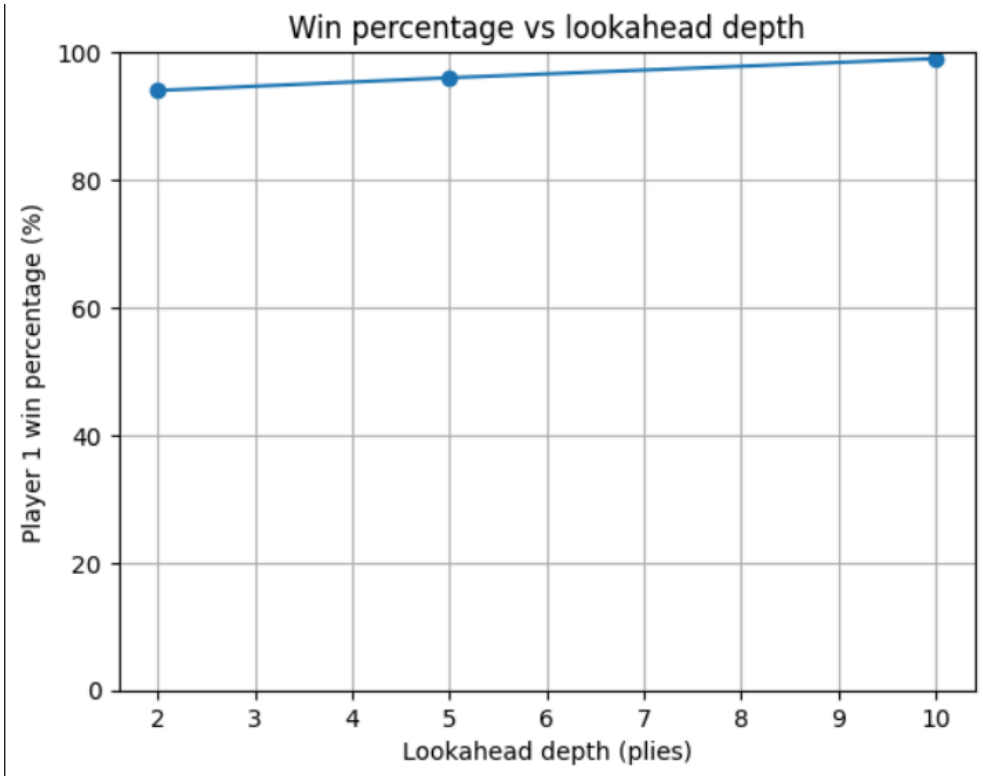
AB Minimax at a depth of 5 took an average of 0.0264 seconds per game over 100 games, while Minimax took an average of 1.042 seconds per game over the same number of games. This means regular Minimax took about 40 times longer than with AB pruning. To estimate how long a game played with Minimax at a depth of 10 takes, we can take 2 different approaches:

1. Knowing regular minimax takes about 40x longer than with AB pruning, and knowing that AB at a depth of 10 took 54.912 seconds each, we can estimate minimax at a depth of 10 will take almost 40 minutes!

2. Minimax expands 6 children nodes for each parent. Therefore, around 6^5 expansions are made for minimax at a depth of 5. For a depth of 10, this would be 6^10. 6^10 is 7776 times larger than 6^5 by basic estimation, so we can multiply time runtime of depth 5 minimax by 7776 (1.042*7776) to get an estimated runtime of 2 hours and 15 minutes!

These estimates are very different, but they're within the same timeframe (in terms of exponential/logarithmic differences), so we can assume that 100 minimax depth 10 games would be counted in hours, rather than seconds or minutes.

8b) Plot a curve showing the win percentage for a player looking ahead 2 plies, 5 plies and 10 plies



8c) As you increase the number of plies, does the AI player win more games? Explain why or why not.

As we increase the AI's lookahead depth, we see more games are won. This is due to the AI being able to choose more optimal moves which consider the state of the game further down the line and maximize the AI's score.

Extra Credit (Section 9)

9a) In your writeup, explain how your new utility function improves on the utility function described above?

Our new utility function combines the previous utility function with a new scoring technique which maximizes the number of stones on the AI's side of the board. The idea behind this improvement is that having more stones on your side of the board not only provides more opportunity to drop stones into your own mancala, but also provides more opportunities to circle around and place your last stone in an empty pit to capture the stones adjacent.

9b) Explain how increasing the number of plies improve the play for the AI Player?

Increasing the number of plies means the AI is looking further ahead into the future game states before choosing a move. At a shallow depth (like 1–2 plies), the AI can only see the immediate consequences of its move. It may make choices that look good right now but lead to bad positions later. When our AI looks farther into the future of moves, it can make decisions from a much larger pool of possibilities and choose the most optimal move.

9c) Is this new utility function a better way to evaluate the strength of a
A particular match? Explain how?

This utility function does provide a better evaluation of the strength of a current board position since it takes into consideration more than just the score. We can see this reflected in the results where the AB run with a depth of 10 and the original heuristic had an average score of 36.16 to 11.84 whereas the modified heuristic had an average score of 36.55 to 11.45. While this 0.4 avg score difference may not seem huge at first, considering there are only 48 stones on the board in total and the regular AB algorithm with a depth of 10 is already fairly optimized, this is a huge difference and proves our new heuristic is a stronger representation of the game state.