Evaluation of the Artificial Intelligence Agents in the Mancala Game

Group 12, composed by:
Paulo Portela up202200871
Miguel Veloso up202202463

Artificial Intelligence, Master of Engineering and Data Science

Abstract

Adversarial search methods are used to find the best next move in games. However, as the depth of the search tree increases the performance of classic adversarial search methods suffers. This work aims to implement a game for two players and solve different versions of this game, using the Minimax search method with *αβ* cuts and its variants, for the Mancala Game, which is a popular two-player strategy board game that is played with small stones or seeds and a board with several pits or cups. To do this, the project team proceeded according to the methodologies and requirements defined for the adversarial search game. As our final output, it's expected to have several different skills (levels of difficulty) - refer to the level of artificial intelligence (AI) of the computer opponent – several different evaluation functions, different depth levels of Minimax, different successor generation ordering and/or variants of the Minimax algorithm. For those, the respective accuracy, performance, and execution time were computed and compared. In conclusion the minimax algorithm performs better as the depth increases, however the need for a high depth should be considered carefully. If not needed, alpha-beta pruning method should be preferred.

Keywords: Adversarial Search Methods. Minimax. Alfa-Beta Pruning. Two-Player Game. Mancala.

I. INTRODUCTION

Games were always an ideal testing ground for different artificial intelligence approaches. Two player games encapsulate adversary multiagent environments with simple and clear rules while also portraying decision problems of varying difficulty. Mancala is a traditional two-player strategy board game that could be used as an example to illustrate adversarial search methods in action. Mancala is played with a board consisting of two rows of six small pits and a larger pit, called the "Mancala", at each end. The objective of the game is to capture as many stones as possible from the opponent's side of the board and deposit them in one's own Mancala. Players take turns picking up all the stones from a pit and placing one in each consecutive pit in a counter-clockwise direction. If the last stone lands in the player's Mancala, they get an extra turn. If the last stone lands in an empty pit on the player's side, they capture that stone and all stones in the opposite pit and place them in their Mancala. The game ends when one player captures all the stones on their opponent's side of the board. In computing, it is often required to find an optimal solution to a given problem, which can be found through which can be found through.

Adversarial search methods are a common approach to evaluate possible solutions. This is done by defining different states of the problem and then transitioning through them. As search methods mimic human behaviour (e.g., decision making), they are part of cognitive simulation approach of artificial intelligence. [1]

Search methods transition thorough the search space to find the optimal solution and use an evaluation function to define their quality. Usually, the quality can be calculated without having to make assumptions about other external factors that might influence the quality. However, for example, in games or negotiations these changes, as the quality of the next problem state must take the actions of the opponent(s) into account. Adversarial search methods are used for problems where two or more parties influence the current problem state. This makes them suitable to evaluate the optimal next move in games.

The most common adversarial search methods are the minimax and alpha-beta pruning method. [1] However, as the number of possible next moves increases, these algorithms need more time, as there are more problem states to evaluate. In games, time is often a constraining factor. Additionally, as the depth of the search increases, both algorithms can explore more possibilities and potentially find better moves. Despite that deeper searches also require more computational resources and time. Therefore, it is important to carefully consider the trade-off between the depth of the search and the available resources when choosing between these algorithms. Alpha-Beta pruning can be used to improve the performance of Minimax by reducing the number of nodes that need to be explored, which can help to make deeper searches feasible within the available resources. As a result, if a high depth is not needed for the game or decision-making problem, Alpha-Beta pruning may be a better choice than Minimax.

This paper looks at adversarial search methods for two-player games to gain a better understanding of their strength and weaknesses. This was achieved by implementing these algorithms and conducting experiments for two two-player games.

II. PROBLEM FORMULATION

Before jumping on to finding the algorithm for evaluating the problem and searching for the solution, we first need to define and formulate the problem. Problem formulation involves deciding what actions and states to consider, given the goal. In general, it´s necessary to abstract the state details from the representation.

The problem formulation, consisting of 5 components, is presented below:

- State Representation - Specifies the position of the seeds in the slots (several representations are possible), presented in the Mancala board (Figure 1);
- Initial State - [4, 4, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 0];
  In this representation, the first player's slots are represented by the first six elements (4, 4, 4, 4, 4, 4), and the second player's slots are represented by the next six elements (4, 4, 4, 4, 4, 4). The final two elements (0, 0) represent each player's store, which is initially empty;
- Objective Test - in a Mancala game is to determine when the game has ended, and one of the players has won. In this case. There are a set of rules that can lead to a game victory, a player turn ends when the stones are moved or when or there are no more stones left in his board side. There is also a possibility to capture the opposition stones, it happens if the last stone lands in an empty slot on the player's side of the board, and there are stones in the slot directly opposite, the player captures all the stones in the opposite pit and adds them to their Mancala (bucket). The game ends when one player has no more stones in their slots, and the player with the most stones in their Mancala wins;
- Actions/Operators – each player can move stones from one of their slots to another slot (own slot) in a counter-clockwise direction. If the last stone lands in his bucket (the last slot that collects the stones – the score) the player gets an extra play;
- Solution Cost: The solution cost in a Mancala game is the number of moves required to reach the goal state from the initial state.

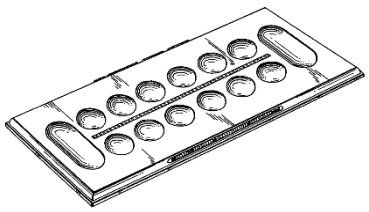

Figure 1. Mancala Board

The state space of a problem is defined by its initial state, possible actions, and transition model, encompassing all reachable states through action sequences. A path within the state space is a sequence of states linked by actions. To solve the problem, the sequence of actions required to reach the goal state from the initial state is considered. The path's cost is evaluated by a cost function, with the optimal solution being the path with the lowest cost among all available solutions.

III. IMPLEMENTATION DETAILS

In the Implementation Details section, the focus was on the technical aspects of the project. This section describes the requirements that the software should meet to better understand its strengths and weaknesses. The implementation details also cover the game modes that were chosen to be implemented and the technical decisions made during the development process.

A. Requirements

To gain a better understanding of the strengths and weaknesses, the following requirements for the software were defined:

1) In addition to the *Minimax and Alpha-Beta Pruning*, a random selection) algorithm was implemented, that every other algorithm should beat comfortably;
2) The depth of the search tree and the number of moves to take into account, has to be variable;
3) The software needs to:
   a) Display the AI best move - in the AI players;
   b) Display the utility value - used to determine the best move to make in the game (a higher utility value means a better state or outcome for the player)
   c) Elapsed time;
4) To gain a more transparent understanding of the algorithms, it must be possible to simulate a single game and view each move per player.

B. Game Modes

Regarding the game modes, it was decided to implement the following four:

1) Human *vs* Computer - In this mode, a human player plays against a computer agent. The human player will make his moves manually, while the computer agent will use an algorithm to select his moves;

2) Human *vs* Human - In this mode, two human players play against each other. Each player will make his moves manually and the goal is to beat his opponent;

3) Computer *vs* Computer - In this mode, two computer agents play against each other. Each agent will use an algorithm to select their moves and the goal is to beat the opponent;

4) Random Agent vs Computer - In this mode, a random agent plays against a computer agent. The random agent will select its moves randomly, while the computer agent will use an algorithm to select its moves. The goal is to evaluate the performance of the computer agent against an opponent who makes random decisions.

Based on the requirements and the way the game development was thought out and carried out, the team chose to implement the challenge as an interface for the game that

could be played in the terminal / console or through a graphical interface, developed through tkinder which was compiled using Python. Additionally, the implementation details explain the algorithms used in the game, including the minimax, alpha-beta, and random selection algorithms, as well as the variables that affect the depth of the search tree and the number of moves considered. The software must display the AI's best move, the utility value used to determine the best move, and the elapsed time. Finally, the implementation details provide insight into how the algorithms were simulated and tested to gain a better understanding of their performance.

IV. APPROACH

Evaluation functions are used by game-playing computer programs to estimate the value or goodness of a position in a game tree, usually at a leaf or terminal node. There are different types of evaluation functions, including the Evaluation Function, Evaluation Moves, and Evaluation Board. The Evaluation Function combines different rules and strategies of the game to give a score for each game state, while the Evaluation Moves computes the number of stones in each player's pits to assign a score based on the difference between the two. The Evaluation Board estimates the advantage of a player based on the number of moves available compared with the opponent. The difficulty level is set by the user and determines the maximum depth of the game tree that can be explored by the AI algorithm in each move. The higher the difficulty level, the deeper the game tree that needs to be explored, and the longer the algorithm will take to compute the optimal move but will make better decisions.

A. Evaluation Functions

An evaluation function, also known as a heuristic evaluation function or static evaluation function, is a function used by game-playing computer programs to estimate the value or goodness of a position (usually at a leaf or terminal node) in a game tree. The following 3 evaluations functions were developed:

1) Evaluation Extra-Turn and Capture

This is the most complete evaluation function of all three as it combines different rules/strategies of the game such as checking for a steal, or an extra turn and it gives a score for each game state that represents how desirable is that state for the player/ai. Of all three this one has the more aggressive game strategy.
In the Mancala game it is possible to steal pieces when the player last stone falls in an empty own slot and the opposition player (parallel) slot has a least one piece, then it takes/ captures all the pieces. The extra turn is achieved when the last piece played falls into the player bucket.

2) Evaluation Moves

It aims to give the player more options and control over the game, the way it works is to compute the number of stones in each player's pits and to assign a score based on the difference between the two. For example, the function could give a positive score if the current player has more stones in their pits than the opponent, and a negative score if the opposite is true. Then the value is calculated by the following formula:

$$0.5 \times score\ value + 0.5\ \times piece\ value$$

3) Evaluation Board

In this approach, this function tries to estimate the advantage of a player/AI based on the number of the moves available compared with the opponent.
It calculates the number of moves available to the player by checking if there are any stones in the slots, if there are then it counts one move. If the player has more moves available than the opponent, the utility score will be positive, indicating an advantage for the player.

To summarize, the implementation of the first evaluation function considers various factors such as the current game state, the scores of each player, and the desirability of certain conditions such as extra turns and stealing stones. The Board evaluation function considers factors such as the number of stones in each player's pits, the number of empty pits, and the number of stones in transit. And the Moves evaluation function will consider the pieces in the slots of the player to evaluate the best move possible.
All of them have provide different decision, different strategies and a battle between AI with different evaluation functions will be very interesting.

B. Difficulty Level

The difficulty level is selected by the user's choice, which is used set the recursion limit for the Minimax and Alpha-Beta Pruning algorithms. The following difficulty levels were defined:

- Easy
- Medium
- Hard
- World Champion

The recursion limit determines the maximum depth of the game tree that can be explored by the AI algorithm in each move. This value is calculated by the following formula:

$$2 + User\ Choice\ (as\ an\ integer) * 2$$

The higher the difficulty level chosen by the user, the deeper the game tree that needs to be explored, and hence the higher the value of recursion limit. On the other hand, a higher value of this, implies that the AI algorithm will take longer to compute the optimal move, but will make better decisions. This value determines the maximum depth of the game tree that can be explored by the algorithm. If the depth of the game tree exceeds it, the algorithm will stop exploring further and return an estimate of the game value based on the evaluation function.

V. ALGORITHM IMPLEMENTED

The fields of decision-making and game theory have been revolutionized by two algorithms - Minimax and Alpha-Beta Pruning. The Minimax algorithm is a backtracking algorithm that helps in decision-making and is widely used in game-playing Artificial Intelligence. It determines the optimal move for a player while assuming that the opponent is also playing optimally. On the other hand, Alpha-Beta pruning is an optimization technique for the Minimax algorithm, which reduces the number of game states that need to be examined, making the algorithm faster and more practical for real-time games. In this way, both Minimax and Alpha-Beta pruning play a vital role in game-playing AI and decision-making, described in more detail below.

A. Minimax

The Minimax algorithm is a widely used backtracking algorithm in decision-making and game theory. Its primary application is in game-playing Artificial Intelligence (AI), such as Chess, Checkers, Tic-Tac-Toe, Go, and other two-player games. The game theory algorithm provides an optimal move for the player while assuming that the opponent is also playing optimally (Figure 2).

This algorithm uses recursion to search through the game-tree, which is a tree representation of all possible moves and outcomes of the game. The computation for the current state is made, where two players play the game, one is called MAX, and the other is called MIN. In this game, both players are opponents of each other. The MAX player will select the maximized value, while the MIN player will select the minimized value. Assumes that both players are rational and always choose the optimal move to maximize their score. Therefore, the algorithm tries to predict the opponent's move and selects the move that provides the maximum benefit to the player, and it continues to evaluate each possible move and its outcome until it reaches the terminal node of the game tree. Additionally, the Minimax algorithm incorporates noise in its evaluation function by adding a random value to the utility score of each node, which helps to avoid getting stuck in local optimum.

In conclusion, the Minimax algorithm is a powerful tool in game-playing AI that enables computers to make optimal decisions while playing two-player games. It uses a depth-first search algorithm and recursion to evaluate all possible moves and their outcomes, providing the player with an optimal strategy while assuming that the opponent is also playing optimally.

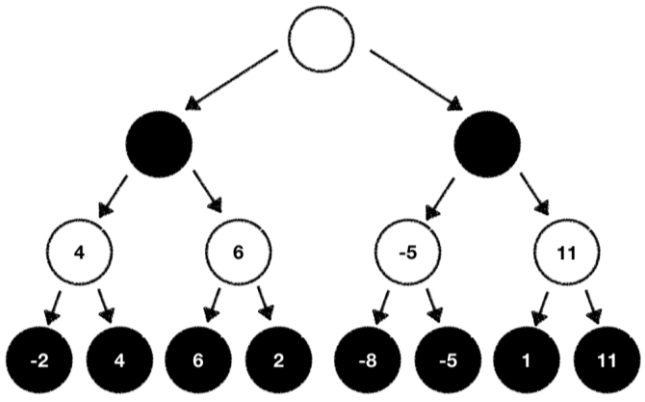

Figure 2. Representative image of Minimax Algorithm

B. Alpha-Beta Pruning

Alpha-beta pruning is an optimization technique for the minimax algorithm used in game theory. The minimax algorithm examines all possible game states in a game tree to determine the best move for a player, but the number of states it needs to examine can be exponential in the depth of the tree. Alpha-beta pruning reduces the number of states that need to be examined by cutting the search tree in half.

The technique involves using two threshold parameters called Alpha and Beta to determine which nodes in the game tree can be pruned. Alpha represents the best choice found so far along the path of the maximize, while Beta represents the best choice found so far along the path of the minimizer. Alpha is initially set to negative infinity, and Beta is initially set to positive infinity. Alpha-beta pruning can be applied at any depth of a tree and can prune not only the leaves of the tree but also entire subtrees. By pruning nodes that do not affect the final decision, the algorithm can be made faster without affecting the accuracy of the result.

Overall, Alpha-Beta Pruning is a powerful optimization technique that can significantly speed up the minimax algorithm and make it more practical for use in real-time games. The addition of random noise to the algorithm evaluation can further enhance the algorithm's performance and provide more accurate results when exploring a wide range of possible moves.

VI. EXPERIMENTAL RESULTS

In this section, the experimental results - which refer to the outcome of an experiment or study that has been conducted to test a hypothesis or research question - obtained from testing different strategies for playing Mancala will be presented. The game was implemented and tested using two different algorithms, Minimax and Alpha-Beta Pruning.
Regarding the obtained results, here are some aspects to take into consideration:

- Firstly, it is verified that the classic adversarial search methods dominate against the random selection algorithm;
- Secondly, it is verified that Alpha-Beta Pruning finds the optimal solution significantly faster.

The several tests performed not only considered the order of players and the number of simulations required for reliable results but also evaluated the effectiveness of different evaluation functions. As mentioned earlier, there are three evaluation functions that determine the quality of a given state of the game and result in different game strategies. These evaluation functions were tested in simulations of 30 times against the random agent, and the results were slightly different than what was expected.
To ensure the accuracy of the results, it is crucial to consider all these factors when comparing different algorithms and evaluation functions. The simulations were performed over 30 to 50 times to achieve statistically significant results, and the games were simulated in both directions to avoid skewing the advantage towards any algorithm or evaluation function. By taking all these factors into account, research can obtain

reliable and accurate results to compare different strategies for playing Mancala.

This way, and considering the formula used to calculate the maximum depth of the game tree that can be exploited by the AI algorithm at each turn, the number of possible game states for a Mancala game with a depth of 1 to 9 was analysed and verified (Figure 3). The number of possible game states increases exponentially with depth, making it difficult to compute the number of states for larger depths. These numbers assume that each pit can hold up to six stones and there are six pits per player.
While the first four levels of depth show a relatively small increase in the number of possible states, as the depth increases, the number of possible states grows exponentially. This exponential growth makes it difficult to compute the number of states for larger depths, and at the eighth and ninth level of depth, the number of possible game states reaches a staggering of 4,503,168 and 67,386,240, respectively.
In the end the analysis of the number of possible game states for a Mancala game with depths ranging from 1 to 9 reveals that the number of states increases exponentially with depth which highlight the complexity of the Mancala game and the challenge faced by AI algorithms attempting to explore the game tree.

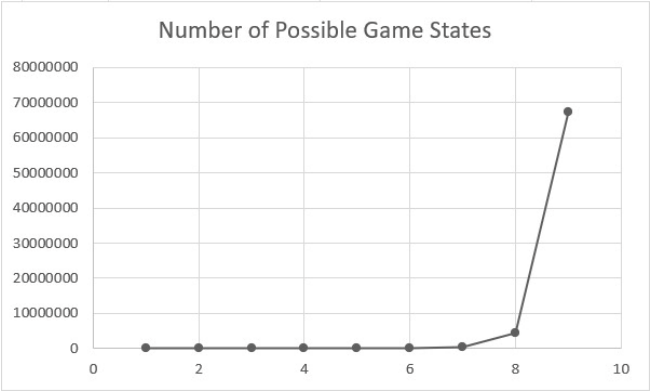

Figure 3. Number of possible game states, considering depth

Then it was decided to evaluate the performance of the Random Agent, which can be verified through the number of victories obtained against the search algorithms (Table 1). It was possible to conclude that against Minimax it obtained a winning percentage of 2% and a tie percentage of 8% against the 90% of Minimax wins, for a difficulty level of 1, Easy. For this same level of difficulty, when compared against the Alpha-Beta Pruning, it was possible to observe that it didn't exist any type of wins, being the best results obtained the 3 draws, what consists in a percentage of 6, against the 94% of wins achieved by the Alpha-Beta Pruning. As for difficulty level 2, Medium, it was not possible to visualize any win by the Random Agent. Against Minimax, the best results obtained were 2 ties against the 96% of victory of the algorithm and against Alpha-Beta Pruning not even a tie was possible to observe, having the algorithm won in all 50 tests, constituting a victory percentage of 100%. For the difficulty level 3, Hard, and as it was intended to verify, it was observed that the adversarial search methods Minimax and Alpha-Beta Pruning end up winning in the 50 tests performed for both algorithms, presenting a winning percentage of 100% against the Random Agent.
Based on the evaluation of the Random Agent's performance against Minimax and Alpha-Beta Pruning algorithms, it can be concluded that the Random Agent's performance is poor compared to these search algorithms. The Random Agent obtained only a 2% winning percentage against Minimax and no wins against Alpha-Beta Pruning at the easy level. At the medium level, the Random Agent did not win against either algorithm, and at the hard level, both algorithms won all 50 tests against the Random Agent, indicating a 100% winning percentage. Due to the high computational cost of the World champion level, it was not compared, but it is expected to yield similar results as the hard level. Therefore, Random Agent is not an effective strategy for playing against these search algorithms in games.

Random Agent vs Minimax (Easy)	
Wins by Minimax	45
Wins by Random Agent	1
Number of Ties	4
Random Agent vs Alpha-Beta Pruning (Easy)	
Wins by Alpha-Beta Pruning	47
Wins by Random Agent	0
Number of Ties	3
Random Agent vs Minimax (Medium)	
Wins by Minimax	48
Wins by Random Agent	0
Number of Ties	2
Random Agent vs Alpha-Beta Pruning (Medium)	
Wins by Alpha-Beta Pruning	50
Wins by Random Agent	0
Number of Ties	0
Random Agent vs Minimax (Hard)	
Wins by Minimax	50
Wins by Random Agent	0
Number of Ties	0
Random Agent vs Alpha-Beta Pruning (Hard)	
Wins by Alpha-Beta Pruning	50
Wins by Random Agent	0
Number of Ties	0

Table 1. Random Agent performance against Minimax and Alpha-Beta Pruning, considering several difficulties

Afterwards, was intended to find out the average number of moves and the average time between the Random Agent and the implemented algorithms for the Easy, Medium and Hard difficulties, respectively (Table 2).
On the first level - Easy - the average number of moves per player and the average time of the moves between the random agent and the minimax were 15 moves and 0.061 seconds. On the other hand, and as expected, for Alpha-Beta Pruning slightly lower values were presented, with the average number of moves per player becoming 13 and the average time 0.032 seconds. For the second level of difficulty - Medium - it is possible to observe that there was an increase in both the number of moves and time per move averages, passing these to present values of 22 moves and 1.02 seconds and 19 moves and 0.09 seconds, for the Minimax and Alpha-Beta Pruning, respectively.
Finally, and as expected, the level of difficulty - Hard - constitutes the level with the highest values obtained, in both study variables. It was obtained an average value of 25 moves per player and 15 seconds for the Minimax algorithm and

average values of 21 moves per player and 1.12 seconds for the Alpha-Beta algorithm.

Based on the results presented in Table 2, it can be concluded that both the number of moves and time per move averages increase as the level of difficulty increases. The Minimax and Alpha-Beta Pruning algorithms showed significant differences in their performance, with Alpha-Beta Pruning presenting lower values for both study variables in all difficulty levels. Therefore, the results suggest that Alpha-Beta Pruning algorithm is a more efficient and effective method for playing Mancala than the Minimax algorithm, especially in more challenging levels, because this one does not explore all paths and nodes, like Minimax does, but prunes those that are guaranteed not to be an optimal state for the current player, that is max or min.

Random Agent vs Minimax (Easy)	
Avg. number of moves per Player	15
Avg. time per move of Minimax	0.061 seconds
Random Agent vs Alpha-Beta Pruning (Easy)	
Avg. number of moves per Player	13
Avg. time per move of Alpha-Beta Pruning	0.032 seconds
Random Agent vs Minimax (Medium)	
Avg. number of moves per Player	22
Avg. time per move of Minimax	1.02 seconds
Random Agent vs Alpha-Beta Pruning (Medium)	
Avg. number of moves per Player	19
Avg. time per move of Alpha-Beta Pruning	0.09 seconds
Random Agent vs Minimax (Hard)	
Avg. number of moves per Player	25
Avg. time per move of Minimax	15 seconds
Random Agent vs Alpha-Beta Pruning (Hard)	
Avg. number of moves per Player	21
Avg. time per move of Alpha-Beta Pruning	1.12 seconds

Table 2. Average number of moves and time per each player between the Random Agents and Search Algorithms

Knowing the impact of depth in the quality of game and how it can make the algorithm more competitive, a new set of tests were conducted to evaluate if a lower depth would lose against a higher depth. A higher depth means that the algorithm should be able to look further into the future and check more possible moves which leads to a better decision-making stronger gameplay. As stated above, it will also increase the computational costs and time (Table 2). The results were accordingly what was expected with the "Medium" level with 100% rate winning against the "Easy" level but with 100% loss rate against "Hard" mode. If the results were much different from the outcome would probably mean that the algorithm code may not be correct, but fortunately it did not happen. For these tests the algorithm used was the Alpha Beta Pruning (to save some time - not that the level hard can take more than 1 minute per game) with each evaluation function.

Finally, after having compared the different search algorithms and the various levels of difficulty we wanted to evaluate and compare the different evaluation functions (Figure 4). To achieve this, we defined three distinct evaluation functions and evaluated their effectiveness in finding optimal solutions:

1) *Evaluation Extra-Turn and Capture*
2) *Evaluation Moves*
3) *Evaluation Board*

For these, the number of wins and the average total time spent by them for the same algorithm (Alpha-Beta Pruning) and the same difficulty level (2) were compared, to make the results as reliable as possible, against the random agent. For the 30 tests performed for each of the evaluation functions, it was possible to observe that the number of victories is proportional to the average total time spent by the evaluation function to finish the game, that is, the one that presents a higher number of wins presents a higher average total time - Extra-Turn and Capture - and, in the other hand, the one that presents a lower number of victories is the one that presents a lower total average game time - Moves . The most effective evaluation function - Extra-Turn and Capture - obtained 29 wins out of 30 tests, which corresponds to a winning percentage of 97% and a total average time of 2.5 seconds. As for the one in the middle - Board - this one had a winning percentage of 87%, 26 wins out of 30, and a total average time of 2.36 seconds. For the one that presented the lowest results - Moves - in both variables and therefore a lower performance than when compared to the others, it obtained 22 wins out of 30, which corresponds to a winning percentage of 73%, and an average total time of 1.86 seconds.

Therefore, the study concluded that the Extra-Turn and Capture evaluation function was the most effective in finding optimal solutions in the game, followed by the Board evaluation function, while the Moves evaluation function had the lowest performance.

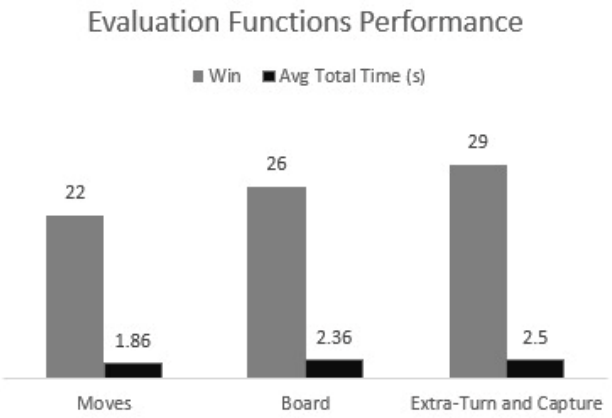

Figure 4. Evaluation Functions Performance

VII. CONCLUSION

Games were created to be played and it only happens if the game is attractive to the user. Given this, some points were analysed and considered in the process of creation, research and development of the Mancala Game.

The Minimax is a powerful algorithm for game-playing, namely, adversarial search, however, it needs a certain of depth in the tree to increase its difficulty level which makes the computing process long and time consuming. For example, for level Hard the first iteration can take up to one minute, that's a long time and will probably demotivate the user to keep playing the game. The player can choose the level Easy which turns out to be much faster, where the sum of all iterations can be less than 5 seconds but will probably be too easy for a medium player. To avoid all this long processing time the Alpha-Beta Pruning was implemented and improved considerable the waiting time even reducing it by 10 times.

The strategy used by the AI algorithms is another important part of the user experience and to make it more interesting, 3 different strategies - evaluation functions - were implemented and could be chosen in the main menu by the user. For the 30 tests performed, the "Extra-Turn and Capture" which can be considered as an aggressive/attacking strategy showed better results against the random agent (29/30 games) when compared against the others, for example, with the evaluation "Moves" (22/30 games). New players might have less skills to play the game and creating one single level of difficulty would make the game less engaging for them or for the pro players (it would depend on the difficulty level implemented). Additionally, 4 levels of difficulty were implemented where, the higher the difficulty level chosen by the user, the deeper the game tree that needs to be explored, and hence the higher the value of recursion limit. On the other hand, a higher value of this, implies that the AI algorithm will take longer to compute the optimal move, but will make better decisions (the Table 1 and Table 2 can confirm it).
Additionally, several tests were performed from which was concluded that an easier level will lose against a higher level, when playing against each other, with the same algorithm and evaluation function, which proves that higher the depth stronger the AI - it will consider more and take better decisions.

ACKNOWLEDGMENTS

"Acknowledgment(s)" is spelled without an "e" after the "g" in American English.

As you can see, the formatting ensures that the text ends in two equal-sized columns rather than only displaying one column on the last page.

This template was adapted from those provided by the IEEE on their own website.

REFERENCES

[1] M. Flasiński, "Introduction to Artificial Intelligence," in Introduction to Artificial Intelligence, Springer International Publishing Switzerland, 2016, pp. 31, 41-44.

[2] Goodman D.; Keen, R. (1997) Man versus Machine: Kasparov versus Deep Blue, H3 Publications

[3] Russel, S.; Norvig, P. (2010) Artificial Intelligence: A Modern Approach, Prentice Hall

[4] Zobrist, A. (1970) A New Hashing Method with Application for Game Playing. Technical Report #88, Computer Science Department, The University of Wisconsin, Madison, WI, USA. Reprinted (1990) in ICCA Journal, Vol. 13, No.