COMS3005A Report: Congo

Yaseen Haffejee
1827555

November 2021

1 Methods

For each algorithm being tested, describe exactly what was done and what changes or modifications you made. If you are working individually, you may have only two algorithms: minimax with the simple evaluation function, and alpha-beta pruning with the more complex one. In that case, describe how these worked briefly, but if you made any additional changes and are excited to share them here, please do.

For those working in groups, please describe the further two modifications that were made on top of alpha-beta pruning and the advanced evaluation function. Describe all four cases. If you implemented more and would like to share, please do. For example:

1.1 Alpha-Beta Pruning

The fundamental goal of Alpha-Beta pruning is to reduce the number of nodes that are expanded within the search tree. This is achieved by pruning(removing) nodes and possibly sub-trees that will have no influence on the final decision. Two important values for the function are Alpha and Beta. Alpha will store the value of the highest score we have managed to achieve thus far as we are expanding and trying to find the optimal move for the maximizing agent. Beta will store the value of the lowest score we have managed to find while trying to find the optimal move for the minimizing agent. Initially, we initialize Alpha to negative infinity and Beta to positive infinity. We then explore each possible child state of the incoming state and recursively explore their children states either until our depth is zero, or we reach a terminal state, i.e one of the lions have been captured. During the recursive calls , the values of alpha and beta are updated respectively , depending on the values returned. Upon completion of all the recursive calls , we evaluate two possibilities , if the value returned is greater than the value of alpha, we update alpha to equal this new value and move onto the next possible move and repeat the procedure. However, if the value returned is greater than or equal to beta, it implies the tree has been pruned and that beta is the best score we are going to achieve and thus return it. If we never reach the second scenario, where the value returned is greater than or equal to beta, we simply move through all the children states and update alpha and eventually return it.

1.2 Advanced evaluation function

The advanced evaluation function is a function that will take in a given state and determine how good is the position for the current agent. Our evaluation function is calculated as follows :
$rawScore = \Delta material + \Delta mobility + \Delta attack.$

Each component is broken down as follows(Δ is taken into account in the explanations):

1) The Material score is calculated by assigning each piece a score, and multiplying each score by the amount of corresponding pieces that are on the board. This is done for both sides and the result is the difference between the score of the current player and the score of the opposing player. This score is a good indication of which side has more pieces in total or which side has more pieces that are "valuable",i.e more pieces that have a higher score assigned to them which imply a grater strength. Hence, the side with a greater score, should theoretically have some advantage.

2) The mobility score is can be determined by evaluating the number of moves each side has and subtracting the number of moves available to the current player from the number of moves available to the opposing player. The greater the score, the more moves the current player has available to him and can be considered to have an advantage.

3) The attack score is the number of opposing pieces that the current player can capture. We calculate this for each side and the the difference between the current players' score and the opposing players' score. A higher attack score implies that the current player has the ability to capture more pieces than their opponent.

If we combine all of these scores , we are able to get a good indication of which side has the advantage. A higher $rawScore$ implies stronger pieces are available to the current player, the current player has a larger variety of possible moves to consider and that the current player has more pieces that are in positions to capture opposing pieces. Consequently, the higher the $rawScore$, the more likely you are to win the

2 Results

I implemented the project individually and thus only have two agents.
Upon implementation of the game , i found that the two agents consistently played a move in one round and thereafter played the reverse move in the next round. Consequently, the game always ended in a draw, irrespective of which agent was white or black. The results are captured below in Table 1.
In order to combat this, i implemented the threefold repetition rule, which states that a move can only be played twice by an agent , thereafter if the move is played a third time, the game is automatically a draw. I tweaked this rule in the following way, if a move was played twice by an agent, in the next round when the agent calls the Minimax or Alpha-Beta search methods, i remove any move that has already been played twice from the list of possible moves. This forced the agents to play the game and not cycle between two moves. The results of this are captured below in Table 2.

		Black	
		Minimax	**Alpha-Beta**
White	**Minimax**	–	$1/2 - 1/2$
	Alpha-Beta	$1/2 - 1/2$	–

Table 1: Results for tournament without threefold repetition

		Black	
		Minimax	**Alpha-Beta**
White	**Minimax**	–	0 - 1
	Alpha-Beta	1 - 0	–

Table 2: Results for tournament with threefold repetition

We can deduce from the above tables that when both agents are forced to play the game , the Alpha-Beta agent performs much better and will always win , since it can search deeper into the tree in the same amount of time because of pruning. Consequently, the deeper search enables the agent to look further into the future and make a better choice of move which results in it winning the game.