Dan Smith and Noah Celuch

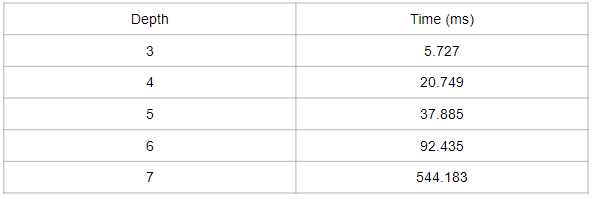
5/6/19

Checkers

Checkers is a common and easy to learn strategy game that dates back to the year 3000 B.C. The problem that we addressed is playing this game using an AI. The game is a zero-sum two player game that the minimax algorithm can be used on. Using just minimax would be way to slow because there are just far too many possibilities for the game state to have. The way that fewer possible game states can be checked is by using the alpha-beta pruning strategy. This will eliminate/prune many branches from the game state tree, but it is still not enough. The depth of the tree will have to be limited so that the time between moves is reasonable. We decided on a depth of five for our tree to limit the time it took to compute.

Despite putting depth limits on the tree, there still needs to be heuristic functions to be able to search to the final states. There are many heuristic functions out there that we could have implemented. Some examples of these heuristics are deciding on who has more pieces, who has more kings, how many pieces are left on the board, which pieces have better positioning, and adding a randomization component. For example, pieces that are positioned on the edge might hold more weight because they cannot be attacked from as many spots as pieces in the middle. Another common way to make the time reasonable was to set a time limit on how long the search could run for. This does not restrict the tree search to a certain level, but just runs as long as the time allows and returns the best possible game state based on the heuristics. If the search is cut off in the middle of the depth, then it returns the previous depth’s best board. A further improvement of the time limit is to test if any layer takes more than half of the allotted time, and if it does then the search is terminated because another level cannot be completed [1]. Our implemented version of the game did not have kings included so that the complexity of the problem was reduced. The heuristics that we have chosen focused on who has more pieces and how many pieces are left on the board.

We timed our AI at different levels of tree depth. We saw sharp increases in the timing as we increased our branching factor. Shown below is a table of the times corresponding to their depths. There was also an increase of memory as we increased depth, causing it to slow down. This is because the tree expands very rapidly because each move has every next move as a child node. This is where the alpha-beta pruning helps save time, because it cuts down on the number of branches that the search algorithm must search. Also, our heuristics were necessary because we could not search down to the final states of the game.



[1] D. Evans and C. Sable, “Alpha-Beta Pruning and Checkers,” *Alpha-Beta pruning and Checkers*. [Online]. Available: http://www.cs.columbia.edu/~devans/TIC/AB.html. [Accessed: 06-May-2019].