Hansen Tamara Dharmawan | A91413023
Mark Darmadi | A11410141
Vania Christie Chandra | A12496582

CSE 150 Assignment 2 write up

Briefly describe of your design to solve problems 1 - 3.

For problem 1, we are required to implement the minimax algorithm. Referring to the slides on lecture 8b, 9 and 10, I followed the given algorithm. On top of the algorithm, we need to include extra checks so that when the given player has no valid moves, it will just skip him and let the other moves until both players have no valid moves left. Analyzing this algorithm, it has a runtime complexity of $O(b^m)$, where b is the branching factor of this algorithm and m is the maximum depth of the tree. This means that it will take a long time for a big board because of the inefficiency of how it is implemented. This algorithm does not require any complex data structure because it is a recursive function which returns a utility value in which we will take the maximum and return that move with the highest utility.

For problem 3, we are required to implement a simple heuristic function. In this problem, we are required to store the necessary variables to calculate the function. We will need M, the number of pits and N, the number of stones in each pit which can be accessed easily in the state class. We then need the stones in the evaluating player's goal and also the stones in the opponent's goal which can be accessed using the board index. Stones on the evaluating player's side and the stones on the opponent's side can be calculated by finding the valid index pit that each player has. The evaluating player's row was passed in so that we can determine the index for the evaluating player's pits. No complex data structure was required for this because we only need to calculate the utility value for the next move. With that, we are able to calculate the best move for the evaluating player.

Describe the approach in details you used in Problem 4 and other approaches you tried considered, if any. Which techniques were the most effective?

As suggested in the write up, we apply iterative deepening minimax search with alpha-beta pruning, transposition table, and move ordering based on the evaluation function that was applied in the previous problems.

The general algorithm idea that we do for problem 4 is, we are doing minimax search with alpha-beta pruning as we have done in problem 2. However, we apply depth limit such that when we hit the limit, the search will not go any deeper (even though we haven't found a leaf node yet) and will go straight to the evaluation function. Another variable that we measured besides the depth limit is the time limit, such that when the time

is up, the search will also automatically go straight to the evaluation function to calculate the utility.

Transposition table was used in this problem so that we can save some time in expense of memory. Terminal states and its utility value are stored inside so that we can look up utility value for visited states. Another table that was added is action transposition table that is similar to the normal transposition table but instead of storing the state and the utility value, it stores the action that causes the state and its utility value. Action transposition table were not used until the end of move function when we are about to choose which action we should return. We will compare all the actions that are in the table, compare the key values, then choose the action with the highest utility value.

Most of the functions that we used are from previous problems with a bit of tune up so we can check if we have hit the time or depth limit. The check for both time and depth are located at the beginning of minVal and maxVal functions which are used to search for leaf node. This is to ensure that once time is up, we can go back as fast as possible to the root and return the value by the evaluation function instead.

For the evaluation function, we choose to stick with what we applied on problem 3 since we think that it was a good function.

Evaluate qualitatively how your custom agent plays. Did you notice any situations where they make seemingly irrational decisions? What could you do/what did you do to improve the performance in these situations?

The custom agent always take the pit nearest to his/her goal pit. This is disadvantageous as it gives some stones to the opponent's side. We could improve the decision making by always choosing in such a way that ends at the goal pit.

What is the maximum number of empty pits and stones per pit (i.e. M,N) on the board for which the minimax agent can play in a reasonable amount of time? What about the alpha-beta agent and your custom agent, in the same amount of time?

For minimax agent, the maximum number of empty pits (M) and stones per pit (N) are 3 and 2. The maximum time taken when on this size is 3.34 second for time interval around 3.5 second. It's worth noting that the move that takes the longest time are the very first move by minimax agent. This makes sense because on the very first move, the agent looks through the whole game tree without exception. For M and N that exceed these numbers, the execution time by minimax agent gets exponentially higher to 242 seconds if the number of M and N are 3 and 3.

If the time interval is increased to around 5 seconds, the maximum M and N for minimax agent are 2 and 8.

The alpha-beta agent can have bigger M and N value than minimax agent. The maximum value of M and N for alpha-beta agent are 3 and 3 for time interval around 3.5 seconds. The maximum time for this value of M and N are 3.16 second. Changing M and N to 3 and 4 increases the time exponentially to 31 seconds.

If the time interval is increased to around 5 seconds, the maximum M and N for alpha-beta agent are 2 and 15.

The custom agent can have a very big M and N in similar time interval. The maximum M and N for custom agent are 6 and 7 for time interval around 3.5 second. If the time interval is increased to around 5 seconds, the maximum value of M and N for custom agent are 6 and 10

Create multiple copies of your custom agent with different depth limits. (You can do this by copying the p4_custom_player.py file to other _player.py and changing the class names inside.) Make them play against each other in at least 10 games on a game. Report the number of wins, losses and ties in a table. Discuss your findings.
Values of M and N used: 4,1; 2,3 ; 3,3 ; 3,5; 2,2; 6,4; 7,2; 3,2; 5,3; 51

Custom Agent	Depth = 3			Depth = 7			Depth = 10		
Player 0	Win	Lose	Draw	Win	Lose	Draw	Win	Lose	Draw
Depth = 3	4	2	4	4	2	4	4	2	4
Depth = 7	4	2	4	4	2	4	4	2	4
Depth = 10	4	2	4	4	2	4	4	2	4

The result of the games are generally the same regardless of the depth of search by the custom agent. This is because the custom agents are evenly matched such that whoever goes first generally have the advantage.

A paragraph from each author stating what their contribution was and what they learned.

Hansen Dharmawan -
I was responsible for testing the program and writing the report for this assignment. I learned the different characteristics of each searching algorithm and their advantages. For example, I noticed that for small mancala, minimax and alpha-beta agent are equally strong in finding the best move. However, as the mancala grows bigger, the time it takes for one minimax agent move is significantly longer than alpha-beta agent.

Mark Darmadi -
I was responsible for handling problem 1, the minimax algorithm and problem 3, the heuristic evaluation. From this two problems, I learnt how properly utilize assignment2.py's API. While most of the methods that I need are already implemented, I initially thought that some of these problems were unsolvable given the methods, but after reading the API time and time again, the given methods were sufficient in solving the problems. I also learn that the minimax algorithm performs VERY very poorly when given a board size of bigger than 4, this is because the branching factor gets really big which is made exponentially bigger by the maximum depth to find the leaf.

Vania Chandra -
I was in charge of handling problem 2, the Alpha-beta problem and problem 4, which is the custom agent player. From these two problems, I learnt the algorithm for alpha-beta pruning better and also how to apply every algorithms in order to make my own agent. It is fun to play around with which function works best even though in the end we decide to go with the function given by the write-up. It also helps me to understand the material for the class better and know how computer player works. We would like to make efficient and fast player and through some pruning methods, depth and time limit, so by playing with them, we can know which one works best.

All -
Testing and debugging the program together.