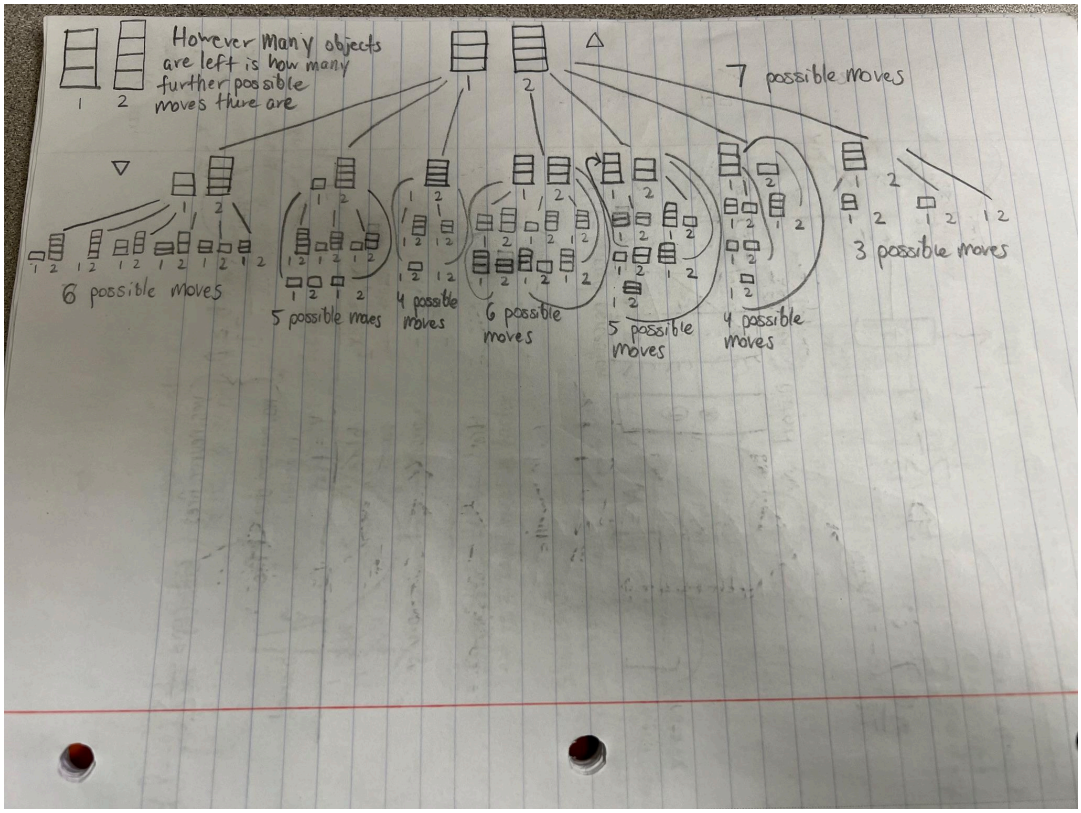
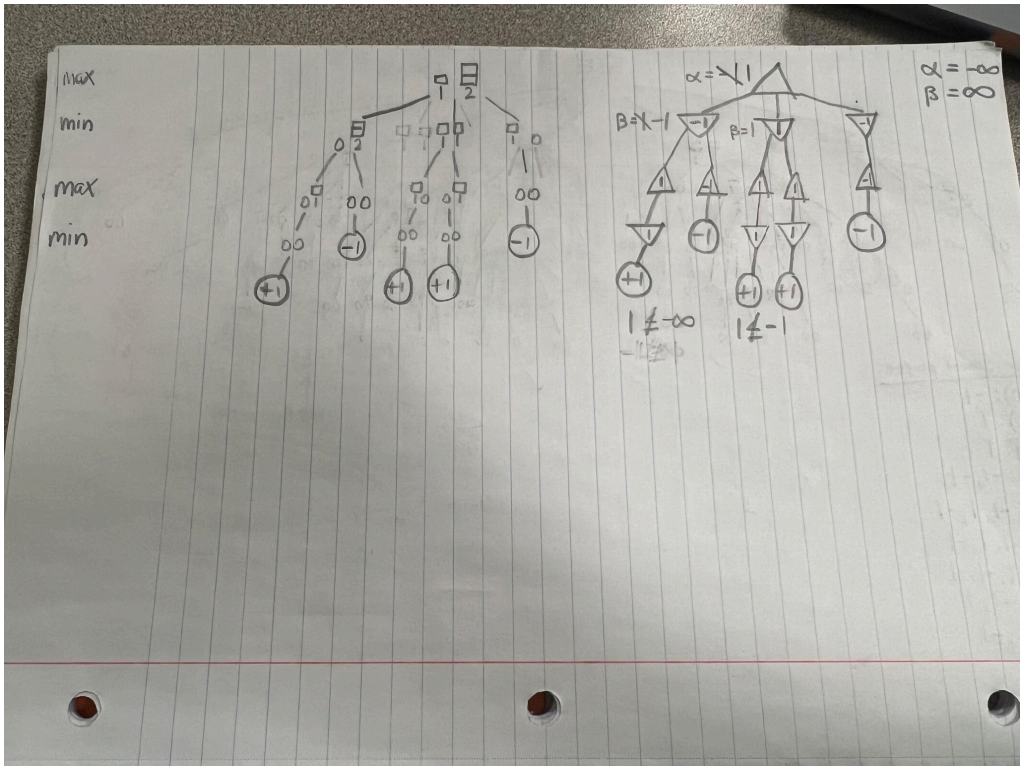
#1

Rules

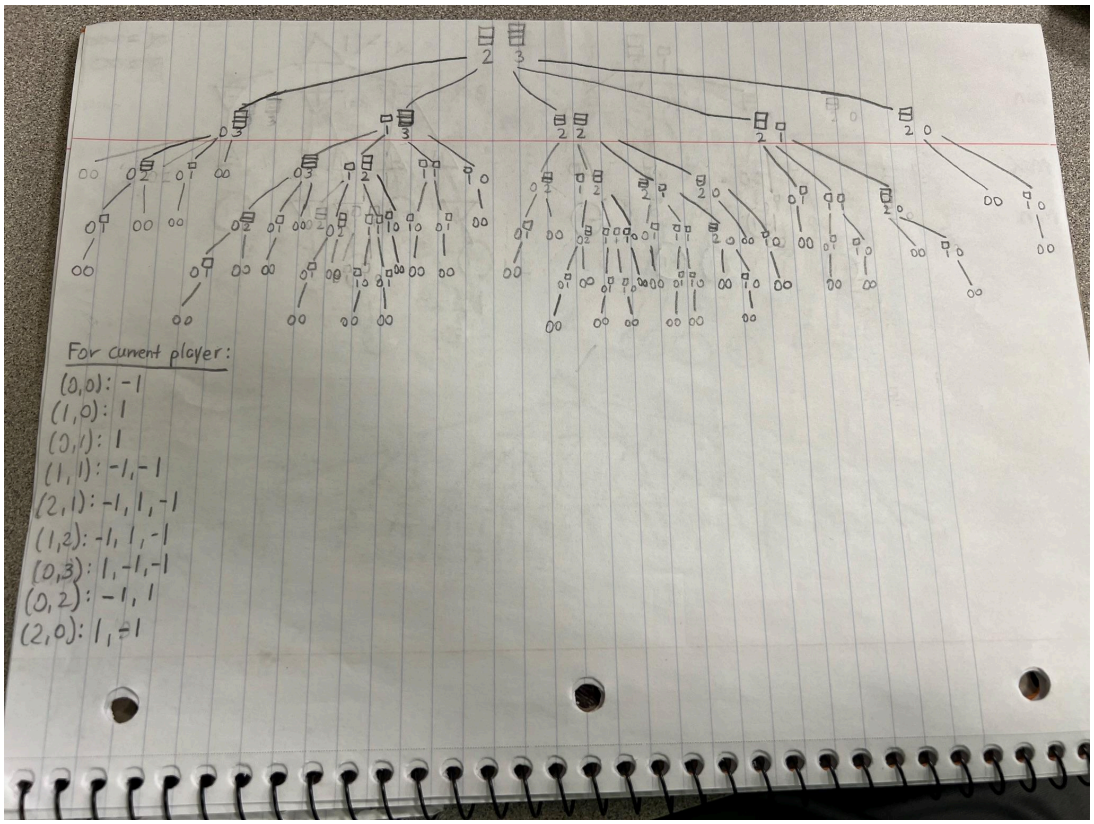
- It is a two-player game that involves players taking turns removing objects from distinct piles.
- The game starts with a specified number of heaps, each containing a variable number of objects. In this assignment, I've used two heaps, each containing 2 and 3 objects, respectively.
- Players take turns choosing a heap and removing any number of objects from that heap, but they must remove at least one object on their turn.
- Players may not remove objects from more than one heap per turn.
- The player who removes the last object wins the game.



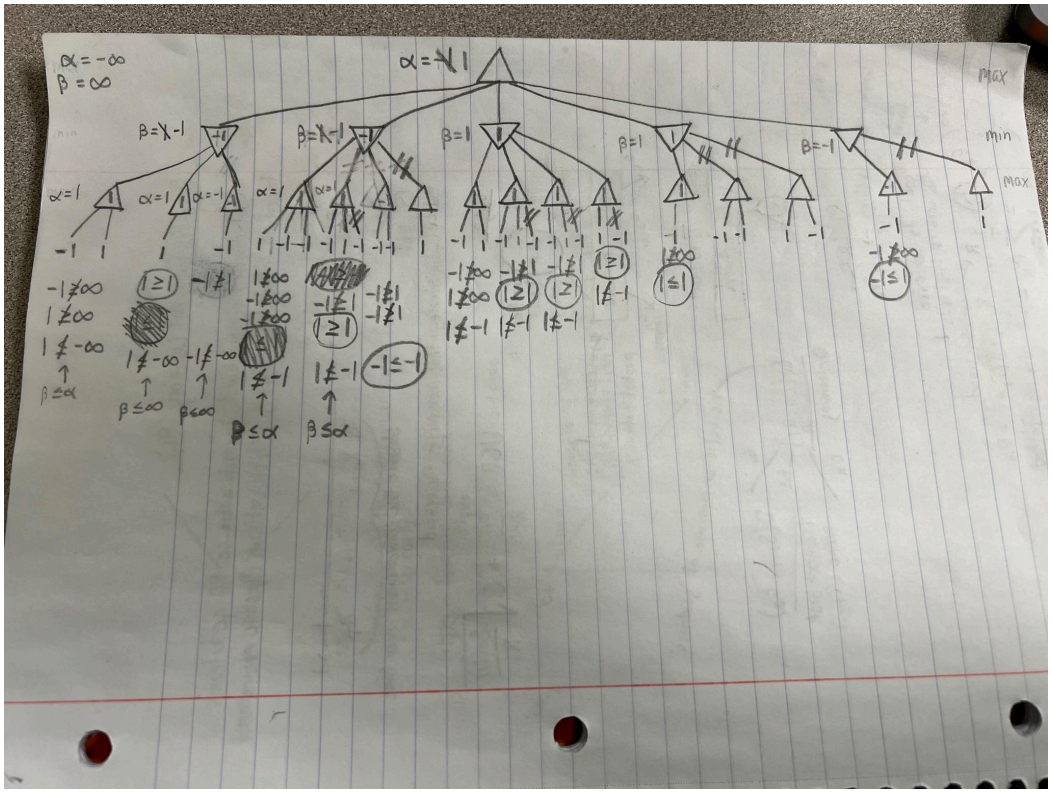
At first I started drawing the tree using two heaps of 3 and 4 objects, respectively. I soon realized that this requires too large of a tree to draw.



Then, I drew a tree using two heaps of 1 and 2 objects, respectively. I realized that this was too simple for the assignment.



Finally, this is the full game tree for two heaps with 2 and 3 objects, respectively. I decided not to do the minimax algorithm on this expanded version of the tree because I felt that it would be easier to do if I condensed the tree. So, using this tree, I noted the outcomes for each node on the third level of the tree and used these for the terminal values. This is described in the following picture.



In the above picture is both the minimax algorithm and alpha beta pruning. As you can see, I used the chart from the previous picture in this one for the terminal values.

Analysis

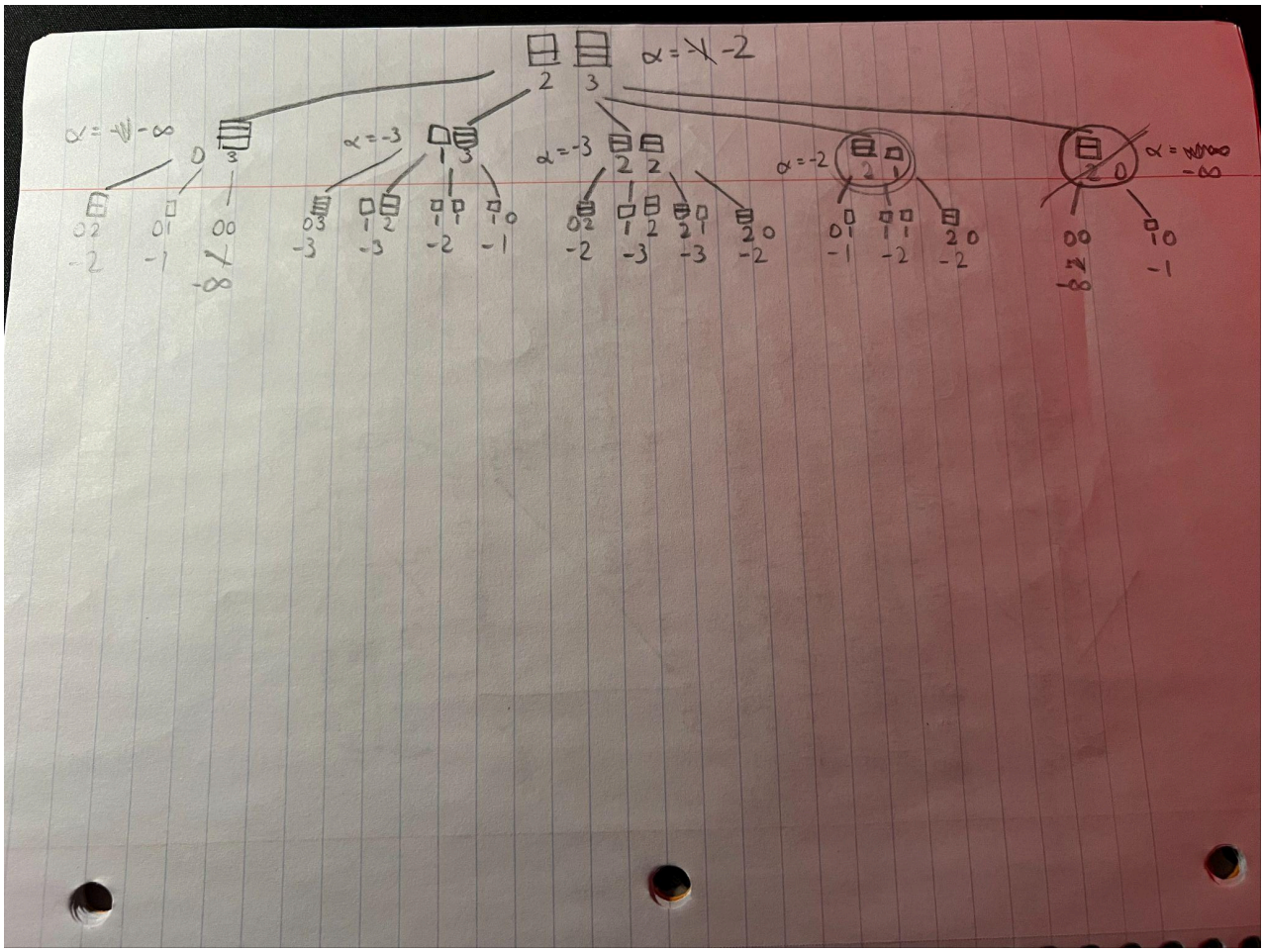
Since minimax is essentially a modified depth first search, it is sufficient for small trees such as the ones covered in this assignment. However, game trees can quickly grow for this game. A game tree with a much larger branching factor renders regular minimax much less effective because it can get lost deep inside the tree. Alpha beta pruning does help eliminate some nodes, though, as you can see from the above picture. The last two nodes on the second level are essentially not explored (the first child of each is, but this is merely a formality to execute the beta <= alpha check). After 2,2 returns a value of 1 to the root, the rest of the tree becomes irrelevant. With a larger tree though, there is no guarantee of this happening because the algorithm can get lost.

#2

- Iterative deepening is good if there is a time constraint because there is always a valid best move if time runs out
- As it keeps iterating through the tree, the best move gets more and more accurate

- In large game trees, you can't search the terminal states because the tree is too big. Estimate the value of a position whenever you stop searching
- Iterative deepening controls how deep the minimax goes, but minimax keeps working the entire time
- After the time is up -> evaluate the current state and propagate a value

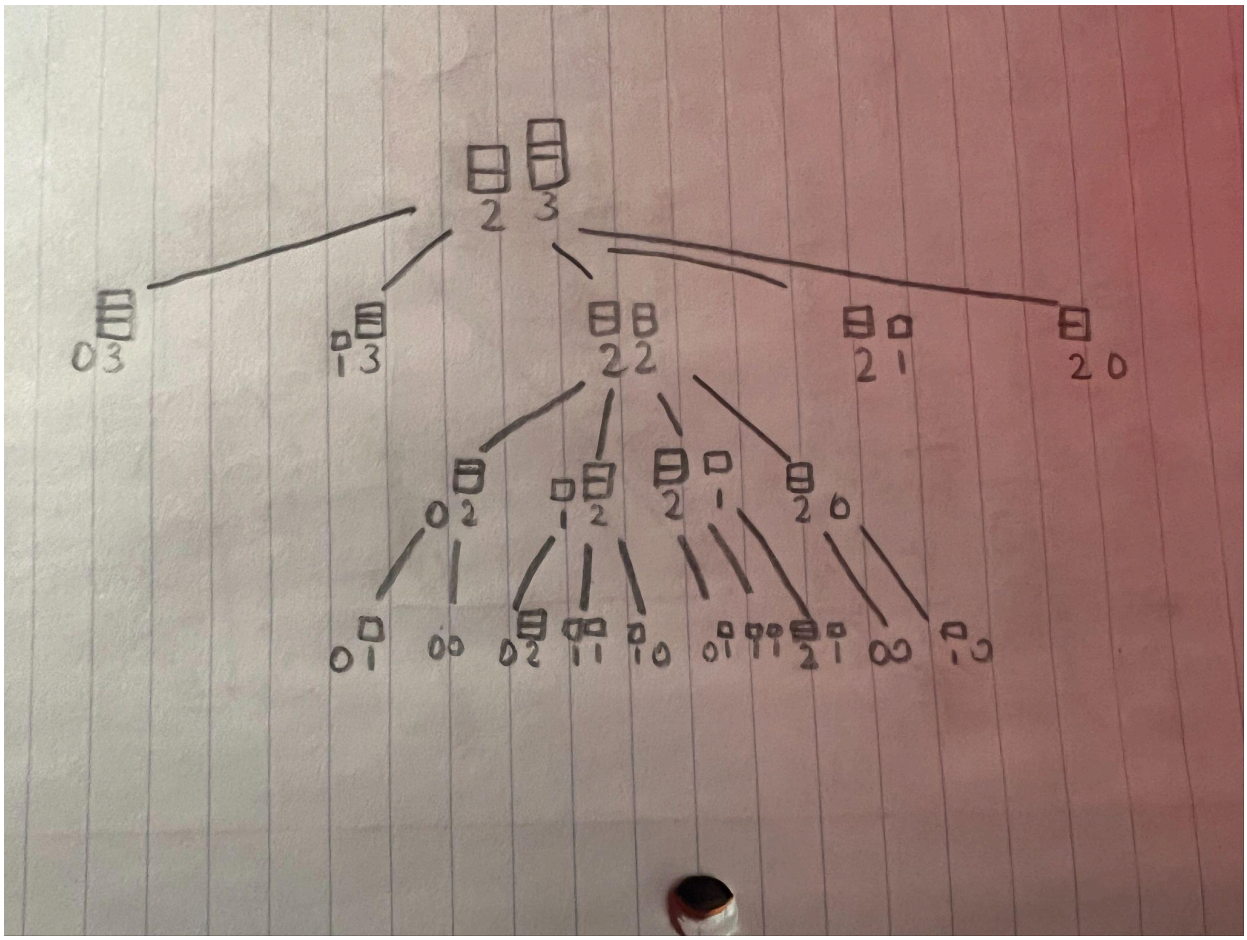
Iterative deepening involves repeated depth limited search, going to a deeper and deeper layer each iteration. This is advantageous for large game trees because you can't get all the way to the terminal nodes using regular minimax. Iterative deepening is good for when there is a time constraint. After each iteration, there is a best move stored which can be used if the time runs out. This best move is also used to start off the next iteration of iterative deepening. When there are no terminal nodes, a heuristic must be used to estimate the best move at that position. Something simple, like prioritizing having fewer stones, can work. With a Nim game starting at 2,3, a 2-ply game doesn't yield any terminal nodes, so it is necessary to use the heuristic. It would go something like this:



After the first iteration, the best option using the heuristic -(A+B) is 2,0. That means we start the second iteration going to 2,0 first. After the second iteration, the best option is 2,1. That means we start the third iteration going to 2,1 first. So far, the algorithm cannot see any wins for MAX, so it is misleading. From prior knowledge, we know 2,2 is the best option. This is not the best example of iterative deepening, but the idea is that if time ran out having only done two

iterations, there is an estimate of the move. Doing a third iteration would lead to a win for MAX, since it would know that 2,2 is the best move.

The heuristic I will show for the last problem is as follows: compare two piles. If A - B != 0, this is not a winning position for MAX. If A - B = 0, this is a winning position for MAX. Take 1,1 as an example. The only possible moves from here for MIN are 1,0 or 0,1, both of which lead to a MAX win.



Using the heuristic, MAX would choose 2,2 because it is the only winning position available. Evaluating all the children, it is evident that they are all losing positions for MIN. 0,2 and 2,0 lead to immediate wins for MAX, while 1,2 and 2,1 are both still losing positions because they lead to 1,1, which is again a winning position for MAX. This heuristic makes much more sense than standard minimax, because moves can confidently be made without guessing. I think I am technically using iterative deepening here, too. 2,2 is the only winning position, so that is marked as the node to be explored from the root for the next iteration. And so on and so forth.