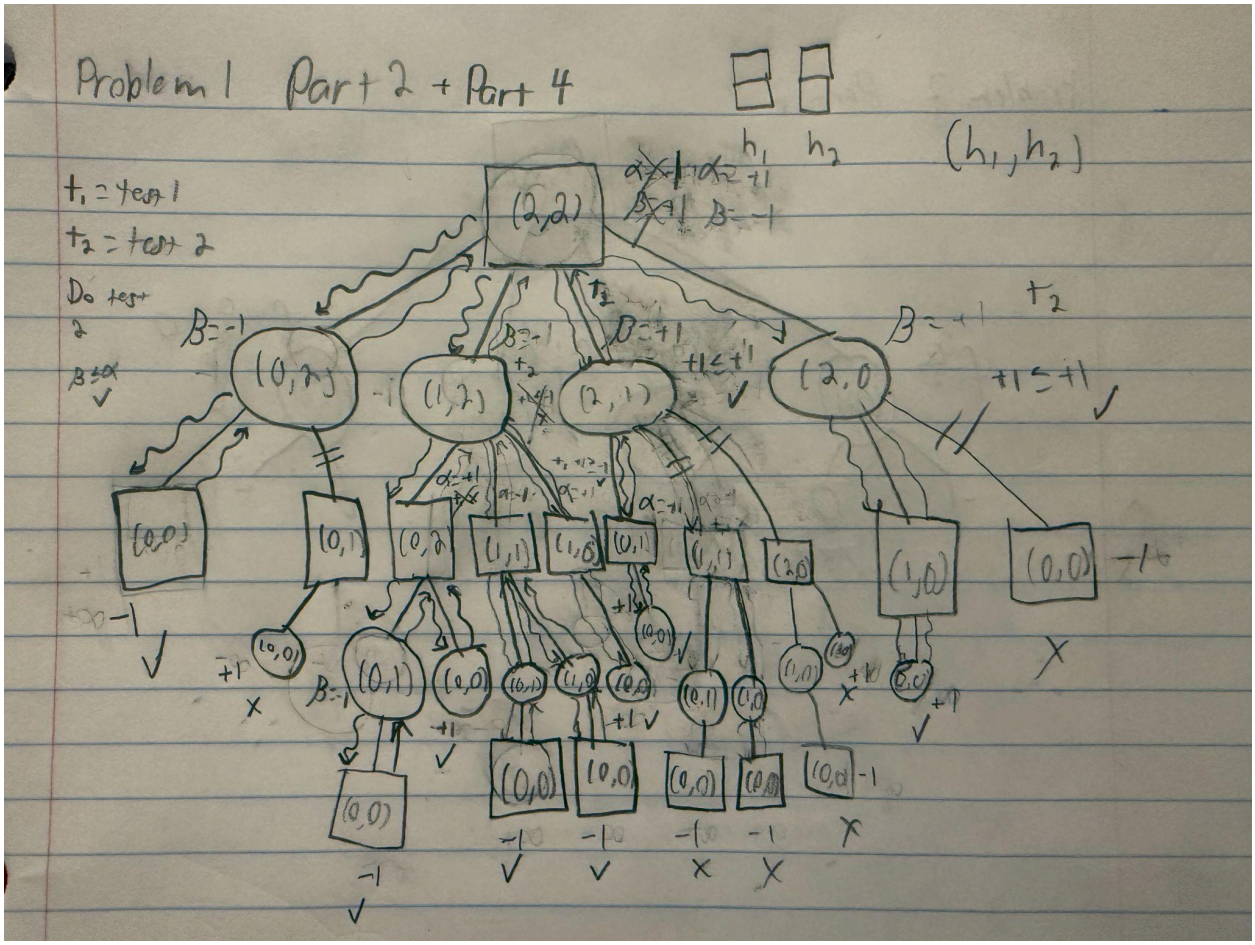
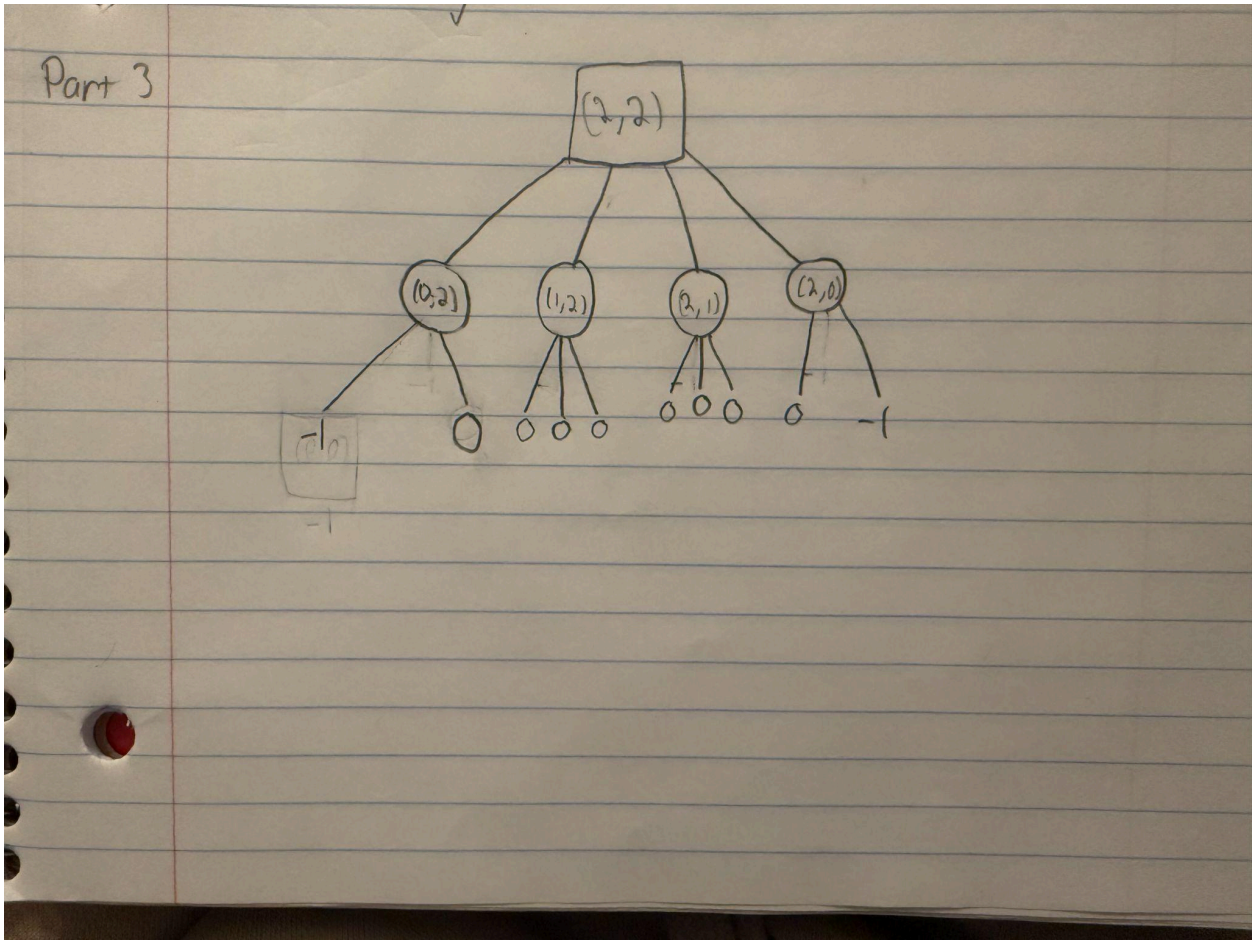
Kevin Scott
Dr. Forouraghi
CSC 362
11/6/25

Problem 1

The game is played by 2 players, there are any number of heaps or piles and any number of pieces per heap. I did a little bit of research and it said that the most common version is played with 3 heaps with 3 in the first, 5 in the second, and 7 in the last heap. The game is played by the active player taking any number of pieces from one heap, the next player then does the same thing. There is no restriction on which heap you must take from, the only rule is that you must take at least one object per turn. The game is won by whichever player takes the last piece.

& 4.





See Part 2

The minimax algorithm was very effective with this specific game because the number of nodes to explore was very small. If we ended up using 100 heaps with 100 in each heap for example then the decision tree would possibly become too large and would make the minimax algorithm take too long and use up too much space. This would result in a correct answer technically but would be severely inefficient. Alpha-Beta pruning helped a large amount even though the number of explorable nodes was only 33. It helped eliminate 12 total nodes. The amount of terminal nodes that I had was 14 and I eliminated 6 through pruning which results in an undeveloped percentage of 42.9%. If the gamespace was larger then that would be a significant amount of time and space saved.

Problem 2

1. A potentially effective optimization technique would be to use an iterative deepening algorithm. This algorithm essentially works as a Depth First Search (DFS) algorithm but has the completeness and optimality guarantees of a Breadth First Search (BFS) algorithm which makes it very attractive. This would hypothetically be very good for a

game such as Nim that would not have such a large branching factor. Even if the branching factor was larger it would still be better than a regular DFS algorithm. Because Nim has the potential to be a very short game in terms of the number of turns required to reach a terminal node that makes it perfect for this algorithm.

Using a 2-ply game as an example we would first look at the child nodes from the starting point and practice DFS. After we explore all of the first level, the algorithm will then increase the depth by one and will begin looking at the children of the nodes. Through this we then find a terminal node immediately which would result in a win for the min.

2. A heuristic that I came up with is $f = \frac{\#\ of\ heaps\ with\ odd\ counts}{\#\ Total\ heaps}$. This creates a sort of gradient function, the higher the gradient score the better the chance of a win. When looking at the 2-ply game, you can see that every time a heap has an odd count that results in a higher chance of a win for the player whose turn it currently is. If you use this heuristic it performs the same as the minimax algorithm in this particular case but I feel like if the number of heaps increased and we went deeper into the game then it would be an even more effective heuristic.