Sean Gor

11/5/25

Assignment 4

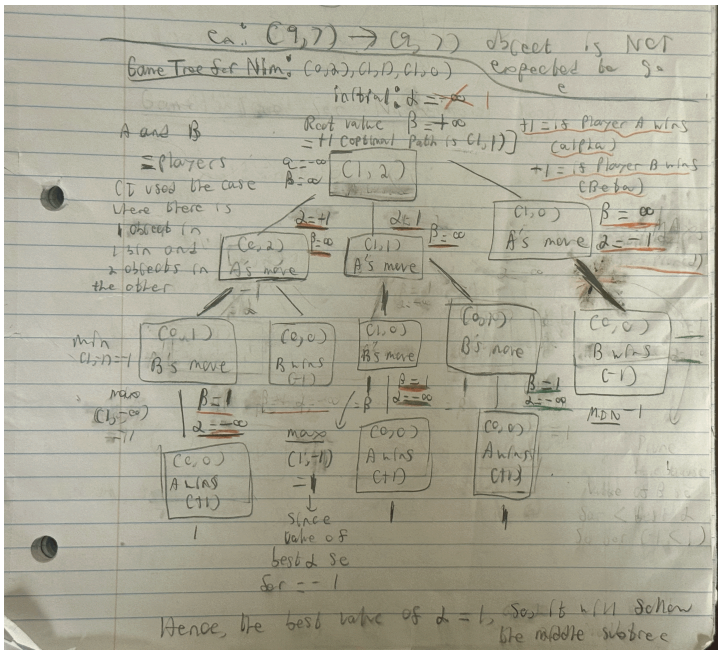
Purpose: To describe and implement a Game Tree and a minimax/alpha beta pruning situation for the 2-ply game called Nim. In addition, this assignment explores additional optimization techniques as well as a heuristic that can be used to simulate this game.

Part 1:

Intro/brief description: Nim is a two-player, zero-sum strategy game played with several heaps (or piles) of objects. On each turn, a player chooses **one heap** and removes **at least one object** from it, but may remove any number of objects from that heap. The players alternate turns until **all objects are removed**, and the player who takes the **last object** wins the game.

For example, a simple configuration might begin with **two heaps containing 3 and 4 objects**, written as **[3,4]**. From this state, each player can choose one of the heaps and remove any number of objects from it.

Below is a Game tree (with the minimax and alpha beta pruning):



Analysis:

The minimax algorithm is effective for finding the optimal move in Nim, but its performance decreases as the number of heaps and stones increases. Each state generates many possible moves, so the game tree grows exponentially with depth. For a small case like [1,2], minimax explores only a few nodes and correctly finds the winning move, but for larger heaps such as [3,4], the number of possible states can increase by a big factor. Despite being slow, the minimax guarantees the best possible play and helps visualize how the game's logic works.

Alpha–beta pruning improves efficiency by cutting off branches that cannot affect the final outcome. When the nodes are ordered well (like checking the nim-sum-0 moves first), many unnecessary evaluations are skipped. In my game tree example for two heaps with one object in the first heap and 2 in the second, one branch was pruned, and for bigger trees the savings would be much higher. Both algorithms reach the same result, but alpha–beta pruning does it faster by avoiding redundant checks. Overall, the minimax ensures correctness, while alpha–beta pruning provides the same accuracy with fewer computations, especially as the game complexity grows.

Part 2:

Optimization Techniques: Explore additional optimization techniques for the minimax algorithm, such as the use of iterative deepening, to improve its performance in analyzing larger game trees. Use a 2-ply game to analyze and explain your strategy.

To optimize the minimax algorithm for the 2-ply Nim game, I used *iterative deepening* and *move ordering*. Iterative deepening means the algorithm first searches to a shallow depth (1-ply) to find a promising move, and then gradually deepens the search while reusing information from earlier layers. This approach improves time management and ensures that the best move is always available even if the search is interrupted early. For the 2-ply Nim game, this meant the algorithm first explored each possible heap reduction quickly, then used those results to order the next level of moves more efficiently. Combined with alpha–beta pruning, evaluating the most promising (nim-sum-zero) moves first drastically reduced the number of nodes explored while keeping the same optimal result.

Heuristic Evaluation: Develop a heuristic evaluation function to estimate the value of game states in Nim, and compare the expected performance of heuristic-based minimax with the standard minimax algorithm. Discuss your approach using a 2-ply game.

For the 2-ply Nim game, a simple heuristic I can come up with is one that minimizes the total number of stones left after the player's move.

h(s) = -(sum of heaps after move)

The logic is that a player should try to "clear the board" as quickly as possible, assuming fewer stones means being closer to winning. In practice, this heuristic assigns higher scores to moves

that leave fewer stones, without considering the deeper strategic value of the position (like the nim-sum).

When tested on a 2-ply game starting at **[1,2]**, this heuristic chooses the move **[1,0]** (leaving only one stone) because it produces the lowest total. However, that move immediately lets the opponent take the last stone and win. The standard minimax algorithm, which evaluates all outcomes correctly, instead chooses **[1,1]**, which leads to a guaranteed win. This example shows how a simple but shortsighted heuristic can change the result of the game, favoring moves that *look good locally* but can fail strategically (or in the long run).

***Part 3 (and the extra leetcode problem) are included on a Google Collab file.**