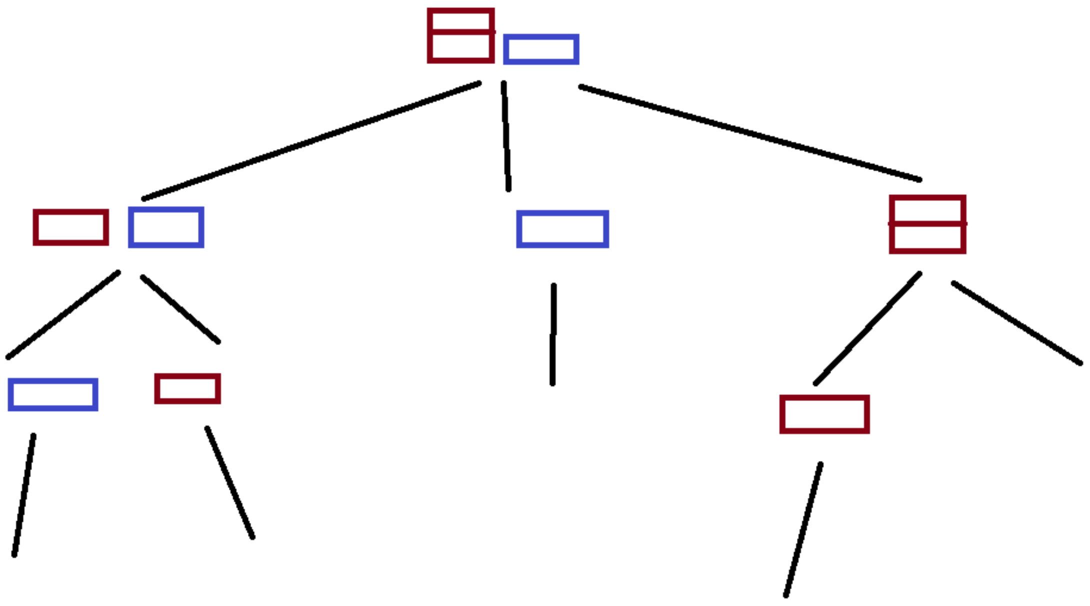
Joseph Pepe
Homework 4
November 6, 2025

<u>Problem 1</u>

Game Description

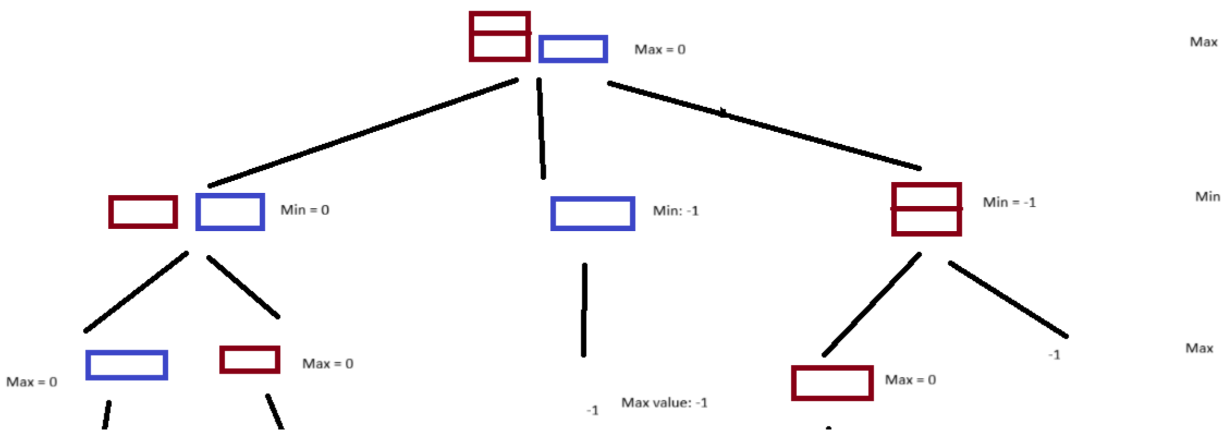
The game of Nim is an adversarial two person game in which there are a number of piles of a certain object. In each pile, there can be a varying amount of objects. A player can choose a pile to draw from, and they can select as many objects from that pile as they want. That will count as that player's turn. Note that the players must remove at least one object from a pile on their turn. The players take turns doing this, and the player to remove the last object wins the game. For this example, we will imagine that there are two distinct piles - one pile will have two objects and one pile will have one object. There will be two players and they will take turns removing from one of the piles.

Game Tree



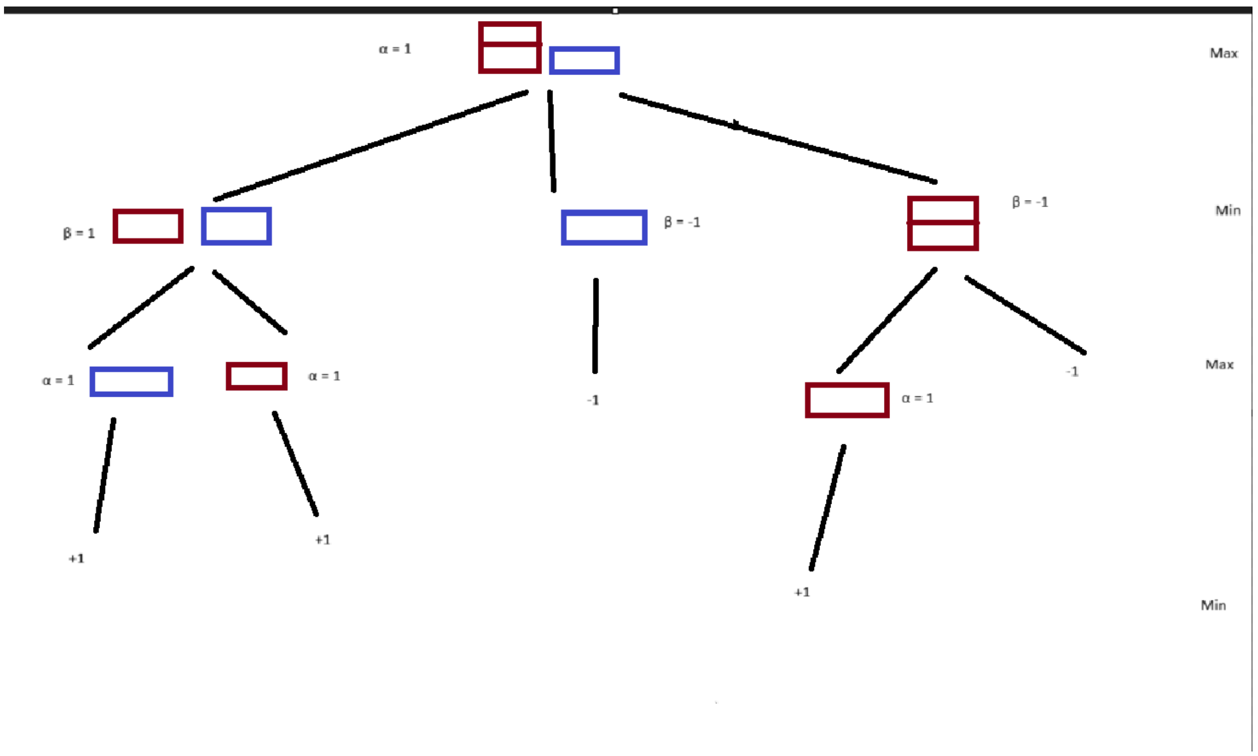
Here is a game tree for a simple game of Nim. Let's go through all the branches. We start with two objects and one object. Let's assume it is Player One's turn. In the left branch, Player One selects one red token. This can result in Player Two selecting either the red token, or the blue token, and Player One will get to select the last token and win in either scenario. In the center branch, Player One selects the two red tokens. This results in Player Two getting to select the last blue token then winning. In the right branch, Player One selects the blue token. From here, Player Two can either select one token, which will allow Player One to select the last token and win, or they can select both tokens in which they themselves will win.

Apply Min-Max



Here I apply the Minimax algorithm. The instructions call for it to be applied to a two-ply game, so that is what we have. Since it is only two ply, there are not that many terminal nodes, but all the nodes still have values that are being pushed up by terminal nodes. This is how the values will be determined: a value of 1 indicates a win while -1 indicates a loss. There is a third option: 0, which means it is a node that has not hit a terminal node yet. Let's start with the bottom left. There is a value of 0 in that node. In the adjacent node, that value also has a value of 0. We can't see what is below those nodes, but since we are in a max row (shown in the far right), so for now we are going to have to stick with 0 as those values. Now let's look at the node above those two. We are now in a min row. However, the only value that can be pushed up is 0, since both children have a value of01, so min will assume a value of 0. Now let's look at the center column at the lowest terminal node. The utility is a -1, meaning that it resulted in a loss. It is a terminal value, so we don't need to pay too much attention to the max column: -1 is as high as it can go, since it is terminal. In the node above that one, we have a min value of -1 because it is the lowest utility value of all that node's children (in this case there is only one child, but the point stands). Now let's look at the right branch. There are two children, let's look at the left child first. It is not a terminal node, so it has a value of 0 since we cannot look below it. In the adjacent node, we have a terminal node of -1. Above that, we are now in a min row. The min row is selected from its children - one child has a value of 0 and one child has a value of -1. Since it is a min function, it will pick the -1. We have found all of the values for the three child nodes to the root, now we can evaluate the root itself. The root is in a max row. It is selected from children who have values of 0, -1 and -1 respectively. Since it is a max row, it will go with the largest value, which is 0, or the leftmost column.

Alpha-Beta Pruning



This diagram shows alpha-beta pruning for this game of Nim that we have been working with. All of the finalized alpha and beta values are shown. If anything is surrounded by a box, that means that it was pruned… but nothing is surrounded, so nothing was pruned. Let's go over why. Nothing can really be pruned in the first tree, since they both have terminal values of +1 and the algorithm has to explore both already. They both push up alpha values, which becomes a beta value at the min node and an alpha value at the root. For the center node, the alpha value of the ancestor (root) is greater than the beta value, so we would prune below min in theory. However, below min is a terminal node which already had to be explored in order to even be able to push that beta value up, so it doesn't happen. In the rightmost tree, the algorithm explores left first and pushes an alpha of 1 up. This becomes beta in the parent node, then the right branch is explored. Again, there is not pruning to do in this tree because it just reaches another terminal node. Alpha beta pruning might be more applicable with a large game size, but as discussed before, it does not work very well with a heuristic such as the one we are using.

Analysis

Alpha beta pruning and the minimax algorithm are relatively successful at evaluating the game of nim. Nim is a very simplistic game, so even though minimax can evaluate it, some of the complexity of the algorithm is being lost on such a simple game. This is because it is difficult to come up with utility function evaluations outside of wins and losses (which would be 1s and -1s). In reality, minimax can handle a variety of utility values which actually makes the algorithm much more effective. This is similar for alpha beta pruning - alpha beta pruning is a brilliant design which can handle much more than the game of nim can offer. So let's talk about how something like branching factor affects minimax and alpha beta pruning in a game like nim. Alpha beta pruning is meant to shine in games with large branching factors (for the purposes of this, I will say nim has a large branching factor, but it really has a very tiny one compared to games like chess and go), but it is not exactly the case in a game like nim. That is because it is difficult to evaluate utility unless you reach the terminal node, in which a lot of the tree is already being explored anyways. This differs from a game like chess, where utility can be evaluated based on the value of the pieces and it can be used to eliminate huge swathes of the game tree early. While alpha beta pruning might prune some nodes in nim, it is still not the most effective for these reasons. Minimax is also not effective because of the large branching factor. Minimax will help you arrive at the correct solution, but nim is so simple that it is a lot of computation and node exploration in order to solve a game that is solved. Now lets go over the number of heaps and the number of objects in each heap. The game gets increasingly more complex the more objects are added, whether it be a token in a different heap or a token piled on top of an existing heap. Each token adds an extra branch to the search space. As mentioned before, this is not great for minimax or alpha beta pruning, because it adds a lot of space to search. Additionally, many of the states throughout the course of the game will be the same, even if they exist in different branches. Starting with a heap of (7,7), there are many ways to get to (3,2), for example. But again, it is inefficient because minimax will evaluate everything, including repeat nodes in different branches. Thus, the more heaps or tokens added, the worse it is for the algorithm. Finally, let's discuss the complexity of the game. The game is solved and not very complex, as stated. There is a term called a nim-sum which can determine who can guarantee a win if the game is played optimally, and an optimal move can be computed from there. Since an optimal move can be found with a simple algorithm, it makes minimax not optimal because minimax would be exploring a massive amount of nodes to come up with the same result a simple algorithm would.

<u>Problem 2</u>

Optimization Techniques

There are a number of ways we can try to optimize minimax for the game of nim. Alpha beta pruning is one such way, but we have already discussed that. Another such way would be iterative deepening. There are some advantages to iterative deepening - the main one is that the algorithm always has a "best move", even if it is only so far along in its search. Another advantage is that it is more memory efficient but still gives optimal results. There are some major problems, however. The foremost problem is that we would need a reliable heuristic to evaluate nodes, and as discussed, this is a huge issue with nim. That is because we don't really have a heuristic outside of wins and losses, so our iterative deepening algorithm would not be able to evaluate nodes that it hasn't seen the terminal value of yet. Additionally, nim has a relatively small search space (at least compared to other games like chess and go). This means that while it might actually be better to use a search like DFS where the terminal nodes are reached because of the aforementioned heuristic problem and the search space could be small enough to allow for that within a computationally reasonable time.

Heuristic Evaluation

There are many different heuristics that could be used to evaluate the game of nim. Here is one that might work for the game of nim: the game will compute if the player is currently in a nim sum position or not. Nim is a solved game, and it can be solved using something called a nim sum. This computation takes the exclusive or of the binary of the amount of coins in each heap (ex. 3 XOR 5 for two heaps of 3 and 5). If the nim sum is not zero, it means the player can currently force a win if they play optimally. They try to find a move that makes the nim sum zero for the other player. Lets apply this to a 2 ply game of nim. We can use the same example from the last problem. The agent will be able to favor paths that keep it in nim sum even if it is not able to determine exactly how it wins yet. For example, it will favor the leftmost branch, because the binary would be 001 and 001, which would XOR to zero. This means it can force a win with the left branch. It will avoid branches like the center and the rightmost branch, where a win is not guaranteed because of the nim sum. In the context of this 2 ply game, this nim sum heuristic does not seem very important because there are a very limited number of outcomes. However, if there were a total of 15 coins split among a number of heaps, this heuristic would become very useful. This is because it would allow the algorithm to somewhat reliably determine if it is in a good state to win or not, even if it cannot explore to the terminal nodes. This would make algorithms like iterative deepening much more effective. The nim sum heuristic would also benefit from better alpha beta pruning, which would save a lot of computational power. Unlike the previous heuristic that was discussed in which only wins and losses were accounted for, a nim sum heuristic would allow the algorithm to prune without having to explore almost every terminal node.