1 N Game

Let define a new game, called the N Game. It works as follows. There are 2 players, and N marbles in the middle. Each turn player 1 takes 1, 2, or 3 marbles from the center. If they take the last marble, they win. Otherwise, player 2 takes 1, 2, or 3 marbles from the center. If they take the last marble, they win. Repeat until there are no more marbles.

For example, for a 5 Game, there are 5 marbles in the middle. For a 200 game, there are 200 in the middle, and so on.

1.1 Whats the branching factor for each ply (move of one player)?

3, since each player can take 1, 2, or 3 marbles out from the center.

1.2 Whats the maximum depth that a solution can exist for the N game?

N, since each player could take 1 marble.

For the 200 game, more nodes exist at a depth of 200 than there are atoms in the universe. Let's take a closer look and see if there is another way to approach the problem.

1.3 4 Game

Let's say you were going to play a 4 game, would you rather go first, or second? Why?

Answer: Would rather go second, since no matter if player 1 picks 1, 2, or 3, I can always take the remaining marbles and win.

Lets look at the game tree to make more sense of this:

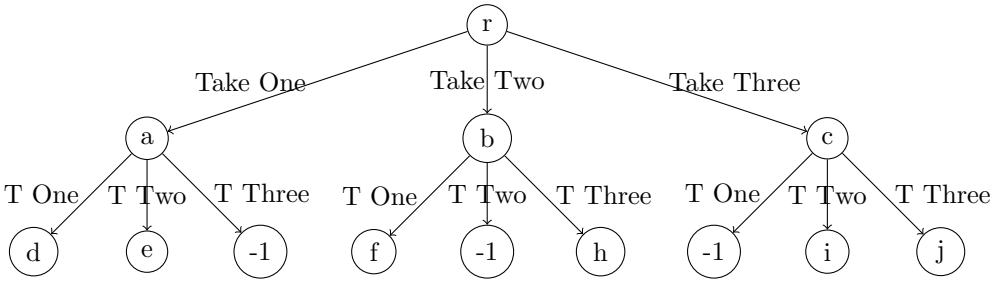

Figure 1: 4 Game

Here a -1 indicates that the node is a terminal state, and that player 2 wins. We know that player 2 will therefore get at least -1 no matter what player 1 does. In other words, player 2 can win deterministically every time. Feel free to draw the mini-max tree to show this.

1.4 8 Game

What about an 8 game, would we want to go first or second?

Answer: Second again. Just like the 4 game, no matter what the first player does for the first turn, we always pick up such that the total missing number of marbles is 4. For example if they pick up 1, then we pick up 3, and so on. Then, after the first turn, it is exactly like the 4 problem which we already proved we can deterministically win.

1.5 10 Game

What about the 10 game, do we want to go first or second?

Answer: First. We take 2 the first turn, then the game is equivalent to the 8 game where we go second. We have already claimed that we can deterministically win this.

1.6 Generalizing

This seems to be following a pattern. See if the students can formally define which games we want to go first/second, and what the winning strategy is in each.

Answer: For all N games where N mod 4 is 0, we want to go second. We follow the strategy observed in the 4 and 8 games, and induct to N. For all other games, we want to go first. We start by picking up N mod 4, then the game turns into an N game where N mod $4 = 0$, and we are again going second. As we showed previously, we can always win these games. Feel free to write a formal inductive proof of correctness on the board or ask students to.

1.7 Would Using Mini-max still Work?

Mini-max would still work to calculate the same optimal moves we have derived. In this case, it is unnecessary to derive the whole tree because the properties of the game make it generalize. This will not always be the case. For example, you are not able to generalize chess to a repeatable sub-problem after 2 moves, or 4, etc. The search space continue to expand.

1.8 Repeated Tree States

Dynamic programming is the concept of solving a complex problem by breaking it down into simpler subproblems and storing their solution. This seems like a perfect candidate for such a paradigm. Lets assume we have a boolean array of length N, $K[N]$. Each index i in $K[i]$ stores whether or not we can win an i game if we have to go first. We know that K[1] = true, K[2] = true, and K[3] = true. These are considered the base cases. Then, we establish a recurrence. For all $i > 3$:

$$K[i] = \quad NOT(K[i-1]) \quad OR \quad NOT(K[i-2]) \quad OR \quad NOT(K[i-3])$$

In other words, we can win an i game if and only if there is no way to win an $i-1$, $i-2$, or $i-3$ game. If this is the case, then we can force our opponent to take the losing turn. This array can be filled bottom up in linear time. Using this technique, we get $K[i]$ is false for all i where i mod 4 is 0, just like we calculated previously.

Additionally, many game tree states will be repeated. For example, taking 2 then 3 marbles is no different than taking 3 than 2 marbles, they still end up in the same place. Accounting for these repeated states and storing their solutions can allow for drastic improvements in space, also in runtime.

2 Search And Rescue

Consider the following scenario. You are sent into a collapsing building to find someone. You want to minimize the worst case time to find and rescue them. The floor plan can be discretized to an N by N grid. The person is disoriented and at each timestep moves one step in a random free (unobstructed) x or y direction with equal probability. You also can only move one square at a time, but you can move diagonal (the other player cannot). Assume that you get the first move, then each person alternates making successive movements. The "game" is over once you are both standing on the same square.

2.1 Example

Consider this is the start state. We are the black king, trying to rescue the white king.
The best move in this case would be to go diagonally
Now, no matter where they move, we are one step away, and are guaranteed to rescue them next turn. **Note** The white king would have to move to d3 or c4 next turn, each with probability 0.5. They cannot stay in place. Also, a square with the rescuer on it is not obstructed, so the person being rescued can technically walk on top of the rescuer, ending the game. This is not possible in

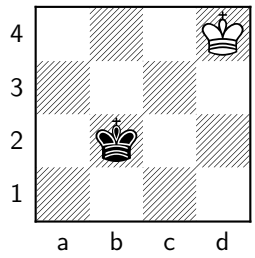

Figure 2: Example Board Start State

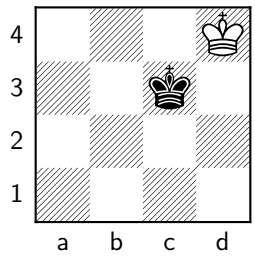

Figure 3: Example Board After First Move

the above game, since they cannot move diagonal, but there are states where it could be relevant.

2.2 Formalizing this in the context of Game Playing

How could you formalize this as a game tree? Note, this does not need to be a zero sum game.

Answer: The rescuer gets the first move, then each move of the person being rescued is modeled as a **nature** move (please see slides for explanation of what this is), where each possible walk location has equal probability. The game terminates when they are standing on the same square. We want to find the strategy that minimizes the max time to rescue them.

Note, in this case the probability of each move does not have to be considered. We can still just run mini-max. Why? We are still trying to optimize for the worst case, not an expected value of moves, which would require the probability of each move.

2.2.1 What if we wanted to minimize the expected number of moves?

This games are sometimes referred to as expectimax, rather than minimax, because we take a sum over each movement of the person being rescued weighted by the probability that they will make that move, rather than only considering their "best" move. Rather than taking the minimum from the max of all rescued person moves, we take the min over the sum of all rescued person moves multiplied by the probability of that move. The idea of weighting based on probability will come up in more detail when we learn about probabilistic reasoning.

2.3 What if we wanted to limit search depth

If we wanted to limit the search depth, we need some sort of evaluation function to estimate the non terminal states. What are some possibilities?

Answer: Manhattan distance seems like the most obvious. Anything related to distance makes sense in this case. Also, the closer they are to a corner, the better. This prevents them from evading as we chase after them. Our objective would be to prioritize states that minimize this value.

How is our minimax algorithm altered with an evaluation function? Who is minimizing and who is maximizing?

Answer: Once we reach the max depth, we use the evaluation algorithm to estimate the value of the state. We assume the person being rescued will want to maximize this, and we will want to minimize this.