# Journal

## Milestone 1

For milestone 1, the goal was to create Tower of Hanoi and Tic Tac Toe and then build Monte Carlo Tree Search (MCTS) AIs to play them. I was able to build playable versions of both games but I only finished the AI for Tower of Hanoi.

As with DFS, I learned there’s an annoying amount of setup involved even if the projects are simple. I started with Tower of Hanoi first and found the biggest hurdle was writing all of the helper methods needed for MCTS to function. I also wrote this iteration poorly. I first wrote Monte Carlo without the tree search part which surprisingly worked. The AI found winning moves. When I changed to MCTS I made the decision of building my tree structure with all of the methods bake in. This made all of the code tangled together. It became difficult deciding if a method should be in Game (the manages the game), MonteCarlo (manages the AI), and mcts tree node (is and also manages the data structure???). Once implemented, the AI had tons of bugs that were hard to debug because many would happen only hundreds to thousands of simulations in. I even hit a stack overflow from recursively looping over too many nodes. Also, Tower of Hanoi has specific problems to it like finding the quickest solution is the goal instead of just a solution. In the end, the AI performed significantly worse the more rings that were added. This may seem obvious, but it was hard to tell when it was a bug versus when it was a limitation of the implementation.

I also didn’t realize I was missing one of the most important rules of Tower of Hanoi until the last weekend. You can’t put large rings on top of smaller rings. This was really dumb on my part, and when I changed over to the correct implementation, more bugs were revealed.

When starting Tic Tac Toe, I decided to have more structure to what does what. Game will have all methods that deal with the game. For instance, you can ask it what are valid moves, is a move valid, and given a game state + move return the updated game state. It is important that the AI doesn’t have any ways that can break the game that player wouldn’t be able to as well. TreeNode is a data structure only and has no methods on it at all. This may be an overcorrection from Tower of Hanoi but doing it barebones can help with determining what’s actually useful. Currently it contains pointers to its parent, children, and a data struct. The data struct then contains the meta data, game state, and input move to reach this node. MonteCarlo then has the 6 main methods of RunSimulations, GetBestNodeToSelect, ExpandNode, RunSimulationOnNode, BackPropagateResults, and GetUCBValueAtNode. There’s a bunch of helper methods to go along with this, but the main idea is Game will call RunSimulations passing an amount. MonteCarlo then calls the next 4 in a loop with the UCB value being used in GetBestNodeToSelect. The really nice thing about this implementation is there’s very little game knowledge that MonteCarlo holds. This will be extremely useful when I start on Dominion. The AI is not done yet, but it is close. Unfortunately its an all or nothing situation.

I did a little more research on handling hidden information. I’m still not set on my solution, but there’s some interesting ideas to try. One would be choosing between simple state machine AIs to solve parts of a turn like in Hierarchical Portfolio Search. The benefit is a MUCH smaller tree with the drawback being the simple AIs may not contain the exact best move.