Comp 424: Final Project Write-up

Abdullahi Elmi

Explanation:

My program utilizes a regular Monte Carlo Tree Search (MCTS) algorithm, that is to say, I have not bolstered it with a heuristic, starting strategy or anything of the sort, simply following the upper confidence trees formula. I created 3 separate objects within the java program to be utilized by the MCTS algorithm; a tree object, a node object, and a state object. The first two were simply requirements in order to run a tree-searching algorithm while the state object was to represent information that each node contains, besides the information in the PentagoBoardState, the state object kept count of the number of times a node was visited, the current scoring value of the node, and the last move that was made to get to the current node.

I also had a separate class for evaluating the UCT value of a node, summing up exploitation and exploration, with a scaling constant equal to ($\sqrt{2}$). I created multiple versions of my program with slightly different scaling constants, but the only definitive information I yielded was that a scaling constant higher than that was less efficient. Lower scaling constants (from $\sqrt{2}$ to 1.2) yielded mixed results, and so I decided to stick with the theoretically optimal value of ($\sqrt{2}$). After all of that was defined, then there was the main MCTS class for the algorithm, divvied up into five methods. Four of which each represented one of the four phases of MCTS; selection, expansion, simulation and back propagation, the fifth being the central method of the class creating the tree, and running it through all four phases, and eventually after deciding on the node it has the most confidence in, extracts the PentagoMove for that node, and returns it. The four phases run in a loop for 600 milliseconds at least (a limit I decided on after multiple trials with different time-limits) before the algorithm decides on an optimal move.

Motivation:

The motivation for this approach came from the fact that I did not fully understand strategies for the game to create an optimal or even comprehensive evaluation function. I tried researching strategies for the general pentago game, and while I did see a few strategies for the original game, I honestly did not have the confidence that I could apply them within minimax for the pentagon-swap game. I also avoided a reinforcement learning algorithm, because I felt much more confident in my ability to comprehensively apply

minimax and MCTS than reinforcement learning. Lastly, there were also the drawbacks of minimax not present in MCTS such as assuming my opponent is playing optimally according to the same evaluation function, and the expensiveness of the minimax algorithm, whereas MCTS could easily be configured to stop within a time limit.

Theoretical Basis:

Theoretically, the basis of the approach is to combine the Monte Carlo Method with tree expansion. Essentially, starting from our current tree configuration, we'd like to first select what we currently consider the most promising leaf node (according to our UCT formula). Then we expand from this selected node to include its children, randomly select one of the said children, and then play out a match from there where the moves for both players are decided randomly. Once the simulation reaches its conclusion, we update information for all the nodes along this simulation, based on the result of the match. This tree search draws on the understanding that repeatedly simulating the environment at random points provides data to inform our confidence in decisions, allowing us to decide on the decision that we feel most confidently is optimal after hundreds or sometimes even thousands of iterations of said simulations.

Reference: https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/

Advantages & Disadvantages:

One of the main advantages is that my approach does not require an evaluation function that demands sufficient knowledge regarding the game of pentago-swap. Furthermore, MCTS is often less computationally expensive than minimax, and it is much easier to confidently bound it to a time limit, and it will still provide what it believes to be the most promising move based on the simulations it has run in that time limit. Another advantage is that MCTS is a widely understood algorithm and not too complicated to implement the general version. Pentago-swap involves a significantly high branching factor, and MCTS' ability to asymmetrically expand. This often means that it can make better decisions in situations that involve high branching factors.

MCTS also comes with its own disadvantages, for one, it does have a significant memory requirement, considering the high branching factor in the game, the tree can begin to grow rapidly after multiple iterations of the four phases. Furthermore, the randomness of the algorithm comes with its own drawbacks, there's a very large number of combinations of states and moves, it's easily plausible that MCTS misses branches with vital

information. An example is that a certain branch could lead to a definitive loss (if our opponent is also playing optimally), but the randomness in the selection and playout phases of MCTS could cause us to avoid taking it into account completely. I've noticed that when playing hundreds of games against the random bot, my algorithm can still lose 1-2% of the time, and I think that situations like the one I just laid out could be the cause (aside from the random bot, randomly making all of the perfect decisions). Lastly, since MCTS draws from a multitude of random simulations to inform its decision, it requires a significant number of these simulations to make a good decision, and as such time becomes even more of a limiting factor. After many matches, it's also become clear that my program is at a disadvantage when playing second, but I've also read online that the original pentago game at least is a first player win, and some other programs I've faced also lose to me more often when my program goes first, so I'm not entirely sure if it's the nature of the game itself, or my program was simply coded improperly to deal with playing second, but I believe it is a weakness nonetheless

Other Approaches:

My first and only other approach I attempted to implement was a minimax algorithm utilizing alpha-beta pruning, but even then, I did not finish that implementation to completion, deciding to switch to MCTS when I hit a significant roadblock with regards to the evaluation function. I found the MCTS algorithm much more intuitive to implement with regards to the pentago-swap game but cannot speak as to how the minimax would have performed against my final MCTS. That being said, I've run my MCTS against a couple of minimax algorithms and found it happened to win more often than lose, but that could easily depend on the heuristic, evaluation function and starting strategy the opponent was utilizing, so I'm not comfortable ascertaining that MCTS is definitively a stronger approach, although it does seem to be on paper considering the category of board games that pentago-swap falls under.

Improvements:

I believe one of the main improvements I could have made to my program would have been to utilize another strategy for choosing the very first few moves before then using my MCTS. It's possible I could have pre-coded aggressive offensive moves to start the game with, and then if that strategy was defended against by the opponent, utilize MCTS to play the rest of the game. Another improvement that could have likely given my program a significant edge would have been to combine MCTS with a heuristic. The early phase of

MCTS is quite random in its decision making, so involving a heuristic based on deeper knowledge of the pentago-swap game may have allowed it to significantly reduce these random moves that would naturally be less optimal at times. Even if I couldn't develop a heuristic by learning the game more comprehensively, it may have been possible to build up a database after many games played by my MCTS (against other programs not random) and prune early decisions that lead to losses immediately or be biased for paths that often lead to victory. I can't be sure how this improvement would fare regarding the memory and time limitations enforced by the tournament, but if it could be implemented while following these limitations, it would almost definitely lead to a significantly better version of my program.

Word Count: 1351