Michael Trinh

CS 4200: Artificial Intelligence

Professor Atanasio

Project 3: Game(Isolation)

TEAM ++

In the program, I only implemented one evaluation function. The evaluation function I implemented was most legal moves. This means that higher evaluations would be based on the number of legal moves possible at a position. Being isolated means that you have zero legal moves, which would mean that it would never be considered as the next move unless the computer has lost. The drawback of this evaluation function is that it results into long games and that the computer plays defensively and not get trapped instead of trying to trap the opponent. The reason why I didn't implement other evaluation functions is that it would take too much extra time in testing which evaluation function is better and implementing them as well.

The evaluation function worked with the alpha-beta pruning algorithm such that max would always try to maximize the amount of moves possible with the evaluation function and updates alpha whenever the min subtrees result in a higher value than the previous alpha. Min is focused on minimizing the amount of moves possible with the evaluation function and updates beta whenever the max subtrees result in a lower value than the previous beta. Alpha-Beta pruning is designed to speed up the min-max decision by pruning away the rest of the subtrees whenever alpha is >= beta.

The initial application for the algorithm in Isolation is to perform alpha-beta pruning on all possible moves for the computer on their turn, and replace the best move that results in the highest value from that algorithm. If the value between new and best moves are the same, roll a 50% chance to switch to the new move. This would result in the computer going to a position where it would have the most amount of legal moves based on the opponent's optimal move for a certain tree depth.

While the alpha-beta pruning can continue until a subtree guarantees the opponent losing, done through an iterative deepening approach, the amount of time it will take is very long and that the depth of the tree will be very high. Therefore I used a depth-limited approach to limit the solution depth and decrease time to generate an optimal move. I tested several solution depths to see which depth would accommodate for the given time limit. At depth 5, it takes around 1 min to make a move. For 4, it takes 5-15 seconds, and for depth 3, 0-10 seconds. Therefore a depth of 3 or 4 would accommodate the 20 second time limit. Since I want to win the competition a depth of 4 would be better as it leads to a stronger computer.

I think the best way I could've implemented the algorithm was to slowly increase the solution depth as the game goes on, combining both depth-limited and iterative deepening approaches. Since the amount of time the computer takes to make a move later on is drastically faster than the time it took for the computer to make a move in the beginning. This would result in the computer guaranteed to not lose in the later stages of the game. But testing the rate at which the solution depth increases would take too much time and data for me. So I stayed with a depth-limited approach.

I had 2 problems while implementing the program. One being that I implemented the program in C++, which I haven't used in a year. This made the coding process slower as I was rusty with the language and had to read several C++ references multiple times. In fact, the code I used to randomize a number well was taken from a C++ reference website. While coding the program, I slowly relearned C++ bits at a time, gradually making it easier for me to code the program. The other problem I had was that I first tried to implement the alpha-beta pruning algorithm iteratively. This didn't work well when I tried to do it and gave me a headache. So I resolved this by implementing the algorithm recursively, which made it easier for me to understand what I was writing, and easier for me to implement the algorithm.