**CS 4346- Project #2 Report**

* **The Problem:** This system has been designed to simulate the playing of a Tic Tac Toe game by two AI players. These AI players use minimax alpha-beta pruning and various evaluator functions to play against their opponent.
* **The Domain:** The domain of this problem is a virtual tic tac toe game. The game features nine spaces arranged in a 3x3 pattern and two players who take turns claiming spaces until one player has claimed three in a line, or neither player can do so. Both players in this system are controlled by a simple AI decision-making algorithm
* **Methodologies:** The primary methodology this solution uses is a minimax alpha-beta pruning algorithm. Minimax is a search algorithm that seeks to evaluate the optimal next move in a game based on an evaluated score of possible future states that could follow that move. This algorithm uses a tree structure to organize the possible game states, and a heuristic evaluation function to assign a value to each of these game states. Alpha-beta pruning is a strategy that allows the algorithm to save time by pruning and ignoring entire branches of the tree because a different branch meets a certain threshold. In the case of Tic tac toe, the game states are the layout of the game board after a new move has been made, and the heuristic function evaluates how valuable a state is by how effectively it leads to a winning state over a losing state (player with three in a row vs opponent with three in a row) .
* **Team Contributions:** This project was completed by one person, Connor Steed. Connor wrote all of the source code for the project, did all of the testing, and wrote the whole project report.
* **Program Implementation:** In this program, I utilized a move choice structure that stored the heuristic value, path length, and child node count of a move as well as its coordinates for choosing where it goes on the board, which is just a 2-dimentional char array.   
  The program alternates between two AI and tries to find the best move for the AI by using a bestMove function to check the value of each space on the board. The bestMove function calls the minimaxAB algorithm, which in turn recursively calls itself with new moves added until it reaches the end of the tree. At the end of the tree, the function calls a heuristic evaluator function which uses a set of criteria to determine the value of the board at that point in time, before returning up the tree with the new value. After each recursion finishes, alpha-beta pruning is applied to decide whether to continue that path or to return up the tree because the condition has been satisfied.   
  After all the recursion has finished, the minimax function returns the value of making that move, and the bestMove function updates its current best move option if needed. After checking each space on the board, the function returns the details of the best move to the main process, who makes the move, updates the board, and switches players. Each turn, the game checks to see if a player has one after their move, and if they have, ends the game, declaring them the winner.
* **Description of Heuristic Functions:** 
  + **EV1:** Normal heuristic function that values any winning state as +/- 10
  + **EV2:** Adds +/- 10 for every square owned in a win line
  + **EV3:** Values blocking an opponent win state as +/-10
  + **EV1:** Values diagonal wins only as +/-10
* **Analysis of the program:** This program creates a simple tic tac toe game using text characters and creates two ai players that will play against each other. I had to apply minimax with Alpha-Beta pruning to the game to run the AI’s decision making and created four heuristic evaluator functions to drive its evaluation process. I created data structures for storing Moves and their values in a way that could be passed up and down a recursive tree, and I adjusted the minimax logic to perform based on the tic tac toe board structure, two player tags, and a specified evaluator tag that tells the evaluator function which heuristic strategy it should apply for that player.
* **Analysis of the results:** I tried many different types of evaluators, and most had little affect on the choices of the AI. The AI often chose to make the same choices regardless of how different the scoring was. Also, with TTT being a simple game with an optimal solution, the player that plays first is usually the one that wins, regardless of the strategy, especially if the AI is not very smart. This caused my statistics and ratios to be very average and not very diverse.
* **Conclusion:** This project taught me that even with simple games with optimal solutions, programming an AI to play the game against another AI is rather difficult. My AI struggled to strategize against the other player and tended to focus on its personal victory goals over true strategy. However, the challenge of applying the minimax AB algorithm to a game taught me a lot about what kind of data structures I need to be aware of and how I want to pass data between levels of recursion. I had to think differently to complete the algorithm and to create a working game that manipulated the right data at the right time.
* **References:** No references