Post Mortem

For This project we were instructed to create an artificial intelligence using one of the algorithms we learned about in class and it was to play a game of tic-tac-toe against the user.

# The Project

For this project I specifically chose Alpha-Beta pruning based on the Minimax algorithm as these two algorithms are especially good for zero-sum games such as tic-tac-toe or chess. These are also ideal because the game is deterministic and turn-based so the A.I. player can view the game state every time it takes a turn which is ideal for these algorithms to make “good” decisions. Good in this sense is viewed as best possible move for the A.I. to either win or tie.

## Algorithm Design

The way minimax and subsequently alpha-beta pruning work is by checking each possible move to a certain depth, since tic-tac-toe has a relative low state space, with a branching factor of 9 factorial, the program is able to save the whole game tree and retrieve information quick enough to not be noticed by the user. The idea of the algorithm is the deeper it can look, the more likely it is to win. This happens because the agent assumes the opponent will always choose the best choice of move and then chooses its best move based on that assumption. Specifically, in tic-tac-toe suboptimal choices in most cases lead to very bad outcomes so the assumption the opponent will make the best choice is a safe play.

# Troubles and Choices

Alpha-Beta pruning was a good choice for this project as it behaves like minimax but with the added benefits of memory saving by not storing everything about the game and only the most important states. Some issues I ran into at first were trying to realize the creation of the algorithm and applying it to the game, however I found a lot of documentation about game theory and alpha-beta pruning which were particularly helpful in understanding how things had to be done. I also learned a great deal about why agents designed for games like this are so difficult to beat. The deeper the algorithm can see the more likely it will be to make the best choice to win. In this algorithm I decided to make it like most games where a win for the agent yields the highest possible score so that is the goal and the opponent winning yields the lowest and therefore is bad, and a tie is in the middle. This makes it so the agent prioritizes winning but will also accept a tie as long as the opponent does not win. During my research about the topic I also found a few examples of minimax where the creator limited the depth that was able to be searched and could increment the search by one to show the differences in difficulty. For example, if the depth is limited to one then it is just as likely you will win over the agent because it only chooses based on one move ahead. However, the difficulty increases as you move the depth further to around three or four where the agent can see enough into the future possible moves that it will win or tie most if not every time. Also, from researching I found that for the original, and most of the current ones, Microsoft games that came with Windows like chess and checkers, to increase the difficulty incrementally they took an approach like this. The idea is that if you play on the easiest difficulty for chess it only looks at a depth of two or three, this returns a good move but possibly not the best, so the player has a good chance of winning. Alternatively, as the player increases the difficulty the depth gets larger and larger which allows the computer difficulty to have the appearance that it is more difficult when it is just selecting better moves based on future board states where it is likely to win, this makes it much more difficult for the player to win but not impossible as everyday computers especially when these were first made around the early 2000s a computer was not computationally capable of looking that far ahead which is why even the hardest difficulty would still be easily beaten by a professional chess player.