Note: I did not work with a partner for this project. My program is my own (Eli Tolentino).

In my time writing this program to implement alpha-beta pruning, I referred to the minimax skeleton in “tictactoe.doc” as well as the chapter 5 slides that introduced the algorithm. I made some changes to this example code that I figured would improve the efficiency of the program (i.e. I stored check_for_win() as a variable to minimize the number of times the function was ran). Furthermore, I implemented an evaluation function to terminate the search if the depth limit has been reached (I set this to 6) or if the maximum amount of time has passed (5 seconds per the project requirements).

To elaborate on my evaluation function, the points of each move were scaled based on how many X’s or O’s were in each range (it checks 4 spaces at a time). I made high-threat ranges (i.e. 3 X’s and 1 empty space = +4000) scale much higher than ranges with less threats (i.e. 2 X’s and 2 empty spaces = +1000). Additionally, I coded my evaluation function to make blocking potential wins from the user the highest priority (i.e. 3 O’s and 1 empty space = -15000). This made my program very defensive against my moves, which I believe is why I could not manage to win against it. On a final note, my evaluation did not check for 4-space ranges where there were both X’s and O’s, as neither player would be able to win in that range.

I reached this evaluation after a lot of trial-and-error playing the game against my program. In the beginning, it would let me win the game easily, and each revision I made was to make the program a stronger competitor against me. This project was fun, as it felt like I was “training” my program in a way by revising the evaluation function. I am glad I was able to implement this algorithm, as it made me better understand how game-playing robots work.