1. I found that the c values were pretty interesting when I changed them. I did a few runs with the c value being really low, from .1 to 1. The main thing I saw, was that it ran really similar to the original c value (sqrt 2), and that there were a lot of different nodes visited. Almost every move was tested the same number of times, which means that there was a very close choice to which move would in the end be taken. With the c value as 100 and higher, I found that it only ever visited all the nodes one time, and focused on only one move value over and over again. This one was not ideal against my alpha beta ai, and was really lacking the defensive and forward thinking that other c values gave. While it still seemed to do ok, it never won against the ai. Overall the original c value, the one where c = sqrt(2) was the best. It was the right amount to visit a good number of nodes, but also focus on the nodes that would provide an optimal result. As shown later, the Monty Carlo Tree Search player seemed to be able to stand up to the ai a lot of the time, and it would do this while going second which was really fascinating.
2. I found that over all, the alpha beta search and the monty carlo tree search were pretty evenly matched. One thing I did notice, however, is that the monty carlo tree search actually won one more time (out of the ten) than the alpha beta search. It seemed to do best when it was going second, which I found to be super interesting. Overall, the alpha beta search seemed to always win when it would go first, but against the mcts player, it would be super evenly matched. I found the mcts player would not do as optimal if it went first, and that the alpha beta search seemed to always run the game. This being said, it fascinates me that the mcts was able to defensively play and still make strategic moves to counter the alpha beta’s strategy.
3. Alpha beta search works by creating a min max tree, and picking the optimal value that it would have at a better score. It would pick the value that would optimize it’s board score, even when the other player would pick the value that would try and minimize it. This allowed the alpha beta search to choose the optimal move every time, and give it a greater option to win than other moves. This requires an evaluation function, and requires that each board state has a way to turn that into a number depending on the player who’s turn it is.

Monty Carlo Tree Search works a little differently. It does not require an evaluation function, and instead just makes sure that there is a value assigned to a win and a loss (and in this case a tie). MCTS will take a new node in its tree, and randomly run different moves until it results in a win or a loss. By doing this thousands of times, we end up with a good average of how many wins or loses we would get by going there. With different c values, we are able to alter how many times we run on a single node versus expanding it.

Both agents try and optimize the value it gets at every move. While they do it in different ways, the overall aspect is the same. Both try and estimate what moves the other player will take, one randomly, the other systematically. The alpha beta search works really well for defined rules, like connect four, but mcts works for a wide variety of places like those of which that have a lot of different moves that it could take, like chess and go.