Tic Tac Toe - Post-Mortem

John (Zheng Yu) Bu

When the assignment was first given, I had the idea to challenge myself and try and do it using OpenGL rather than just with the console. This was due to wanting to get better at OpenGL since that was what we are going to be doing.
This was not the hardest part of the assignment, as it was mostly based on what he had done already for the graphics class. After I had gotten a usable framework for a game in OpenGL, only then did I start implementing a basic game structure for Tic Tac Toe.

The basic human-human, and easy AI gameplay was simple enough to implement. Using the mouse input for playing the game actually helped a lot to speed up the testing and debugging process, especially compared to if you had to input each coordinate using the keyboard.
The actual Minimax algorithm was definitely the hardest part of the project, as it took a long time to get my head around how it works. The first attempt I had used a single FOR loop in a recursive function to try and generate the children and check through them at the same time, but this did not work well at all. Because I chose to use a node-based system for the algorithm, it was easier to debug it, so eventually I ended up with different FOR loops for the generation and checking respectively.

After doing that, I had a basic AI that could play against the player, and in most situations will be able to win or draw. However, when I played a mark in one corner, and then played one in the opposite corner, the AI would give up and allow the player to win very easily.
This was a fairly big issue that took a good amount of time to debug and solve. Eventually through using many breakpoints and other debugging measures, I found that the reason it was doing that. It was because when the algorithm checked through a node's children, if the parent node was a max node for example, then it would only pass up a "10" or a "0", and if it was a min node, it would only pass up a "-10" or a "0". This was very wrong, because even if it was a max node, then it should be able to pass up a "-10" as well as "0" and "10", because when all of a node's children are "-10", then it must pass that up regardless of its min or max.

After fixing that, the AI essentially became unbeatable, even when I played in the corners. The next part was to optimize it using alpha-beta pruning. This also took longer than it should have, mostly due to trying to get it to work with my node-based structure. The main problem was getting the AI loop breaking too early, and not at the right time like the pruning should work. Comparing my own implementation and that from various examples from the lecture slides, I was able to eventually come to a good work-around. This meant that if a child in the loop gets a "10" and the parent is a max node, then the loop would break right there, since it won't matter if another gets 10 as well (since 10 is the highest value for a node anyways), the opposite was done for the min node. After implementing this and testing, the number of operations finally did go down. Despite all these challenges, I did end up learning a lot during this project.