Key: 3D = 3DTTT folder on github
FKG = For the Glory of Kiang folder on github

In the beginning I thought my 3-D tic tac toe game would be straight forward because it's very easy to grasp, but after thinking about all of the methods that I would need to integrate into my game I got kind of nervous. So, the first thing I did was outline the different methods that I thought I would initially need. At first in class I just focused on making the board in the command line. After asking around I realized that a recursion would probably do the trick and so I spent the class just making the spaces. (FKG 2)

I had the problem of figuring out how to give different values to the elements in the arraylist after that, and after having a long talk with Malcolm I started to research a 3d array. I couldn't really grasp it, and since the board was going to be displayed on a 2d plane I decided to just keep it a 2d array. I also had a problem with the main method erasing all the work that happened below it, but after talking to Mr. Kiang I decided to make a driver instead along with a separate piece class to hold all the different variables of the spaces. (FKG 5) I used a special mathematical square to add a separate value for the elements of the array list so that most of the combinations to win would add up to 34. However, further along I realized that I couldn't figure out a way to go through every combination of moves a player had made. Sadly I forgot to press the sync branch button so the separate days that I coded things aren't really clear, but a few days before the project was due I decided to brute force the code by checking which squares were filled by a player, and if the right ones were filled to win. (3D 3)

This was the first time that I made a lot of methods that relied on each other by myself, but I was surprised when it worked perfectly (with a few colons missing) on the first try after I brute forced all the ways to win. I expected this to be a challenge because I haven't done it before to this extent in previous coding homework, but because I was prepared for this challenge I spent extra time trying to make sure that they would work together without an error. However, I started to make more and more methods as I realized that there were many small methods needed to make the game work. (FGK 7) At first I started to make methods in anticipation of needing to use them, but I'm not sure whether or not it helped. On one hand it helped me to remember the different steps that I needed to do, but it also didn't help when I tried to figure out different methods because sometimes I would spend time thinking of how to use the methods I had named even if I didn't need them.

For some methods I had trouble putting the brackets in the right places because I had a lot of if statements that depended on the other methods. Methods such as checkBattle (3D 2) had recursion and if statements that made it difficult for me to fix. I started to have errors because I put else statements in the wrong places, but after having Mr. Kiang look at it, they were put in the right places.

I didn't collaborate with anyone because most of my project seemed specific to myself, but I talked with Rafe about printing out my board in the command box. (We eventually figured out that we had similar code that did the same thing basically.) I got input Malcolm and Zach when coding a way to check if a person won or not. Zach recommended a method that checked the surrounding elements of the array to see if they were used or not, but after realizing the amount of possibilities that a single square had, I decided to not take that path. I also went over the idea of using an equation to check specific ways of winning, but every few different ways had different equations, which also made the coding too tedious. (3D 2) Malcolm helped me with various problems such as correctly printing out my strings into the command box after I added in the piece class. We actually took a long time to figure it out and in the end he ended up knowing a lot of my code to help me. (FGK 6)

During the midterm I didn't know anything about arrays or arraylists (I'm still kind of not too clear about arraylists) but the use of arrays in my code allowed me to practice using them and coding them. I hoped to learn how to extract different values from elements of an array since get methods from the midterm also confused me. In the end I learned how to do all the above, and I also learned how to use recursion with more confidence and how to use them to search elements of arrays. In the beginning I wanted to make the game along with an AI (which would have probably been harder than the actual game) but that obviously didn't happen. I didn't realize how hard it would be to simply do the game, but after making each plane 4x4 and adding an extra plane to make it 4x4x4, the coding was much harder. (FGK 6) Originally this was because of the special sum of 34, but I ended up not using this code. However, I kept it 4x4x4 because Mr. Kiang pointed out that having a 3x3x3 game made it very easy for the first player to win because the first person who occupied the middle space in the middle plane had many more opportunities to win based on that one move. Although this made coding the game much more difficult to code, it also made it

I was really proud of the minimal amount of errors that I had after brute forcing all the different ways to win. The methods all linked together correctly and the only errors that I had were usually connected renamed methods or variables that I didn't keep constant. However, I understand the errors I made and I think that making them initially helped me remember not to make them again. (Just kidding, I probably will, but at least now I know that I need to look out for them.) So far after playing 8 full games with various people the code didn't break. I tried to make a try catch method, but no one in the class that I talked to (Jaz, Nolan, Kapri, etc.) knew how to do it well enough to help me. I'm really proud of the ways to win not breaking. I expected it to either not initialize a few ways to win or accidentally say someone won because I coded 74 different ways to win.

If I did this again I would try to see Mr. Kiang more often and make sure that my methods all use the same words so that I could skip a lot of the errors that I initially made.