For the most part my design from milestones A and B did not change as much as I had anticipated, but there was one big implementation error that I did not manage to catch the first time I did unit testing's, which I had to account for in my new Implementation. This error was my draw function for my tile deck class, when I was creating my list of tiles, I was accidentally aliasing all of the same lettered tiles, meaning when I went to place a tile of the same type/letter if I had placed a meeple on it or rotated it, it will alias to that one and not a new tile as anticipated. The only way I found to get around this was to hold a new instance of the tiles every time I went to draw a tile and keep a separate list that just contains the tile identifiers and not the tiles themselves. After I did this all of the aliasing errors I was having went away.

Another small change I made to my Carcassonne game implementation is that I removed my initial constructor for a game from taking in the number of players, to simply starting a game with 0 players and adding players later. I found this easier to control because I could create a game and then update the players playing in the action performed function for the start game button.

Lastly I think I misunderstood how java swing worked at first, and initially left my main Carcassonne class very barebones because I thought the game GUI would be maintained by a continuous while loop. I learned the hard way, that this is indeed not how swing is maintained, and I turn I had to add a lot more getter functions and made the game more modular instead of having this one big continuous loop always running. Overall the backbone of my Carcassonne implementation remained the same, along with my scoring and placing functions which follows the design model we discussed in class. That being said I felt that with java swing I did not have very main choices for actually making my GUI aesthetically pleasing and found it very difficult to format anything in a way that made any sense, but swing uses the observer pattern which made a lot of my functionality easy to implement.