1. **Project:** Lucky Bet
    a. **Names:** Sammie Shellman and Zion Washington
    b. **Github repo**: https://github.com/samanthashellman/luckyBet

2. **Final state of system**

The final project implements the majority of features that we initially set out to include. It starts the user at a control center that allows them to either sign in to a previously created account (with their progress saved), create a new account (and save that progress), or play as a guest.

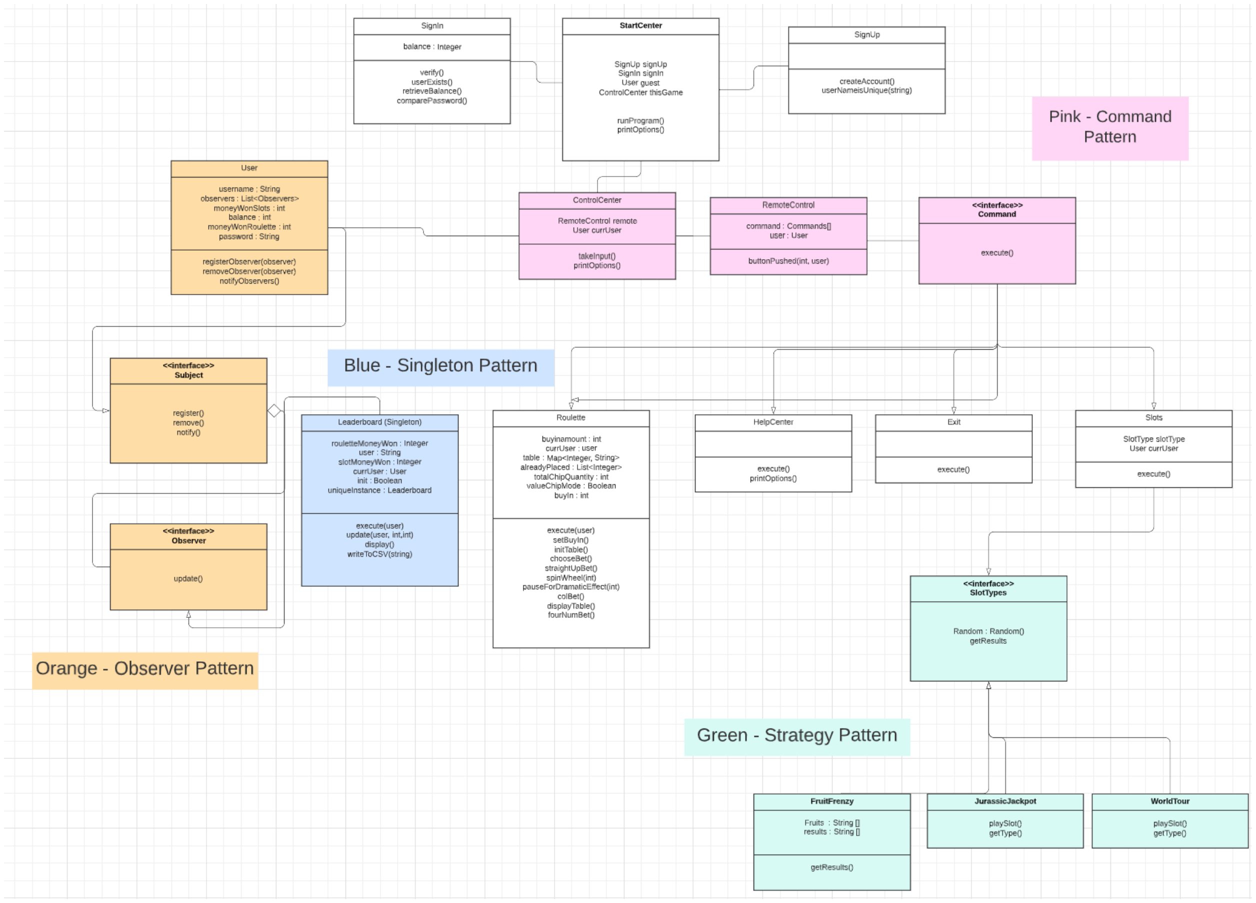
After that, they can play a variety of lottery-style games. The execution of the user's choice of game is done using a Command pattern. There are 3 different types of slot machines that use a Strategy pattern to output different categories of slot results (fruits, dinosaurs, or countries). They can also play Roulette and make a wide range of different bets here (color, number, range of numbers, etc). Through these games, they can win or lose money, which is updated in the database.

For the database, we opted to use a csv file rather than a sql database because it was clearer to implement and functioned just as well for our small amount of data.

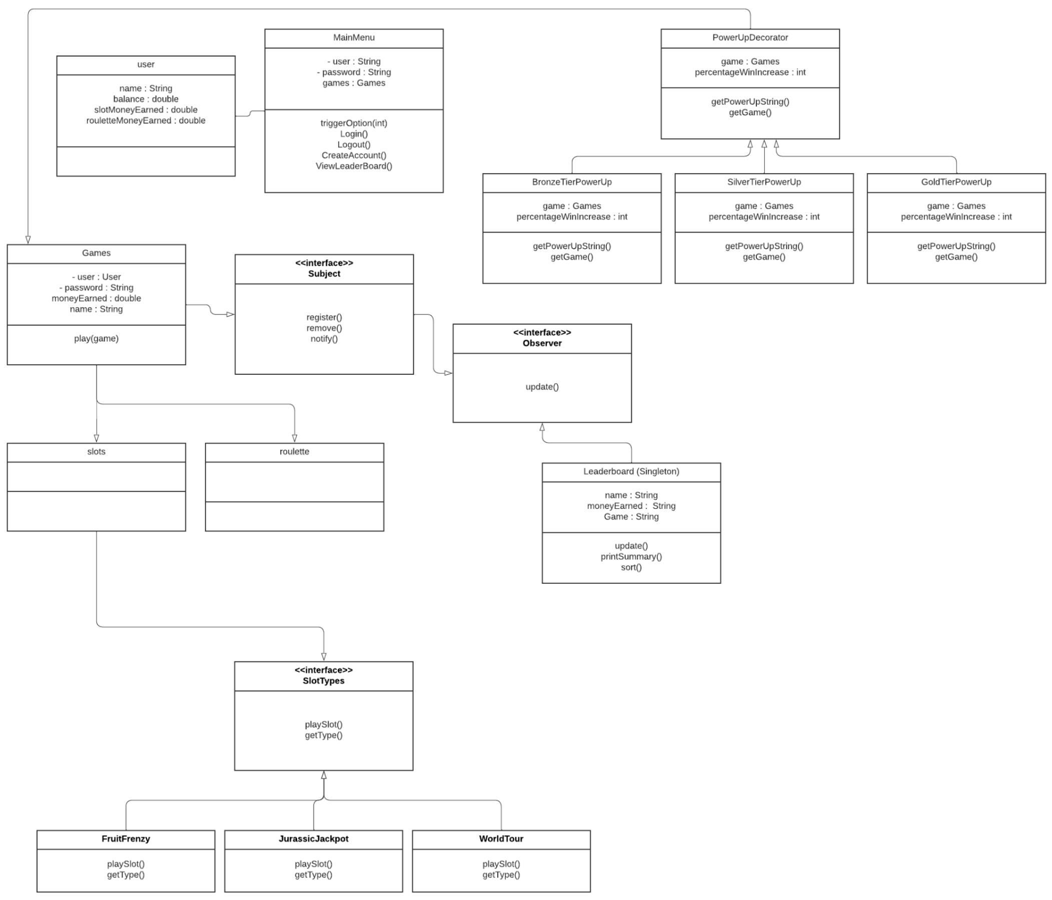
The user can also check the leaderboard to see which users have earned the most amount of money overall. The leaderboard uses the Singleton pattern (to make sure only one is ever created) and the Observer pattern (to stay updated on changes to the rankings). They can also enter a 'help' area that can give information on the different types of games, or let them check their current monetary balance. Although we originally wanted to implement power-ups for the games that users could buy, we realized this was a much more complex task than we originally thought and so we opted to finalize the rest of the features instead.

3. Final class diagram & comparison

Final class diagram:



Project 5 class diagram:



Comparison:

As discussed elsewhere, we opted to add the Command pattern which shifted around some class relationships. We also determined that sign up and sign in should each be their own class as the functionality is somewhat complex and we wanted to maintain cohesion. The rest of the class structure stayed mostly the same, with the exception of removing the power-up classes. Also, relevant variables and functions were added as necessary.

4. **Third-party code**
   - We used this post from Stack Overflow for the function pauseForDramaticEffect that causes the program to pause for a given number of seconds.
   - We also used code from a third party website to read and write to our csv files. Post is here.

5. **OOAD process:**

- Though we didn't originally plan to use the Command pattern, when we started coding, we realized that it could be really useful for structuring the different functions the user might choose to execute from the main control center. We rearranged the class structure to accommodate the pattern and found it to be more efficient that way.

- During the planning phase of the project, we didn’t specify exactly what the power-ups needed to do. We then implemented the majority of the functionality before we sat down to detail how to implement the power-ups. At this point, we realized that any sort of power-ups that would actually be enticing for users to ‘buy’ would be really complex to add because our existing code didn’t account for it. This was a lesson learned in the importance of fully fleshing out all needed specifications before beginning execution!
- The observer pattern was very useful in updating user balances and scores to the leaderboard. It was simpler to notify observers, rather than have to create completely new copies of information to pass the same data to different classes, and eventually the database. Though at first it was a bit difficult to understand its structure, it proved to be invaluable when designing our program.