Name of Project: OOAD Texas Holdem

Team Members: Chirag Telang , Michael Truong

Final State of System Statement

1. **Features Implemented**

ID	Title
UR-01	Join Game
UR-02	Leave Game
UR-03	Sign Up
UR-04	Sign Out
UR-05	Fold
UR-06	Bet
UR-07	Check

2. **Features *Not* Implemented**

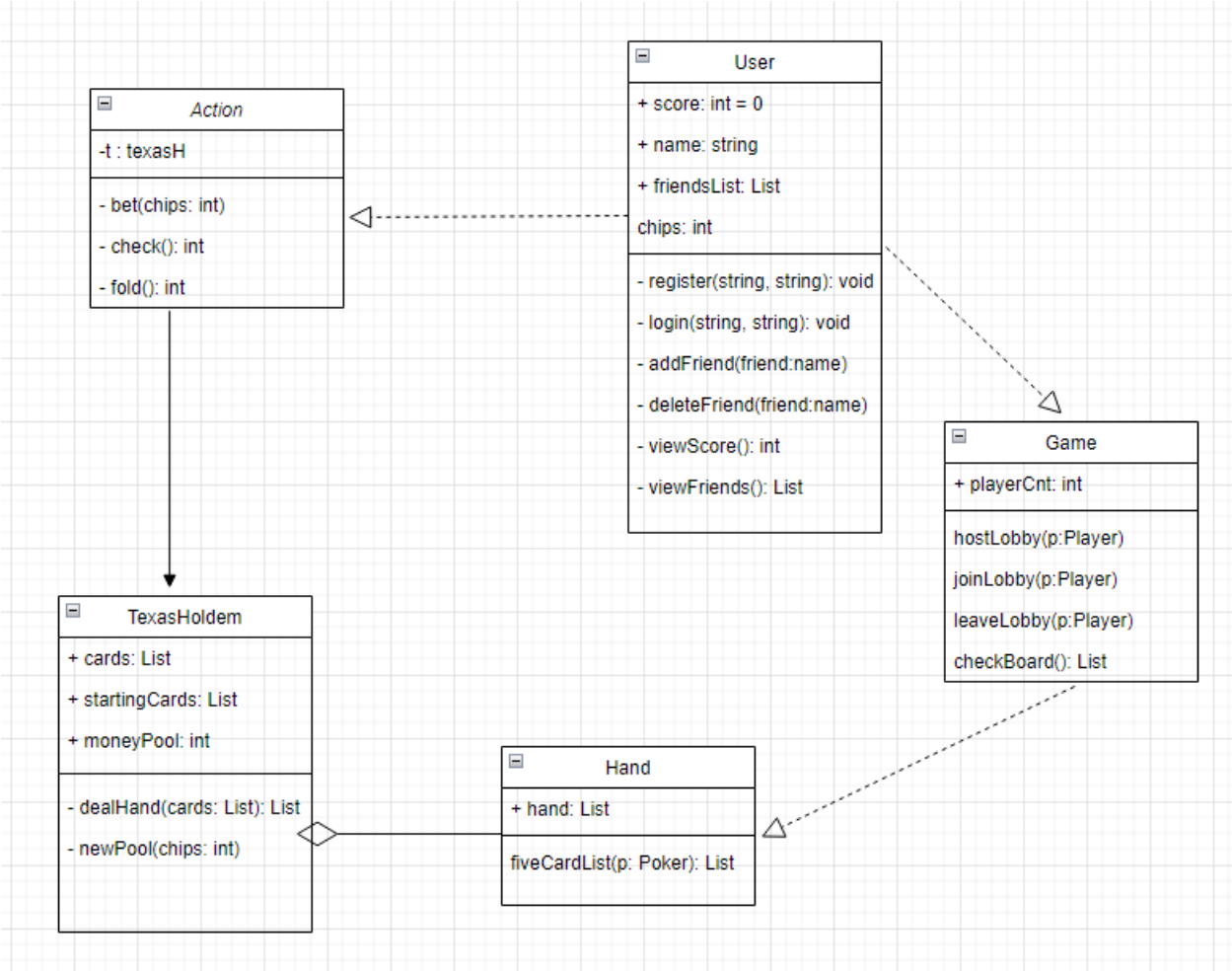
ID	Title
UR-08	Score Tracking
UR-09	Chip Distribution
UR-10	Joining a Friends' Game
UR-11	Inviting Friends
UR-12	Opponent Name Tracking

Implementing the game logic of Texas Hold'em proved to be more challenging than we had initially expected – which led us into not completing some functionalities that we had originally stated. Some of the main features we were not able to complete were in regards to the friend functionalities (adding friends, inviting friends, etc). As a group, we weren't able to think of a

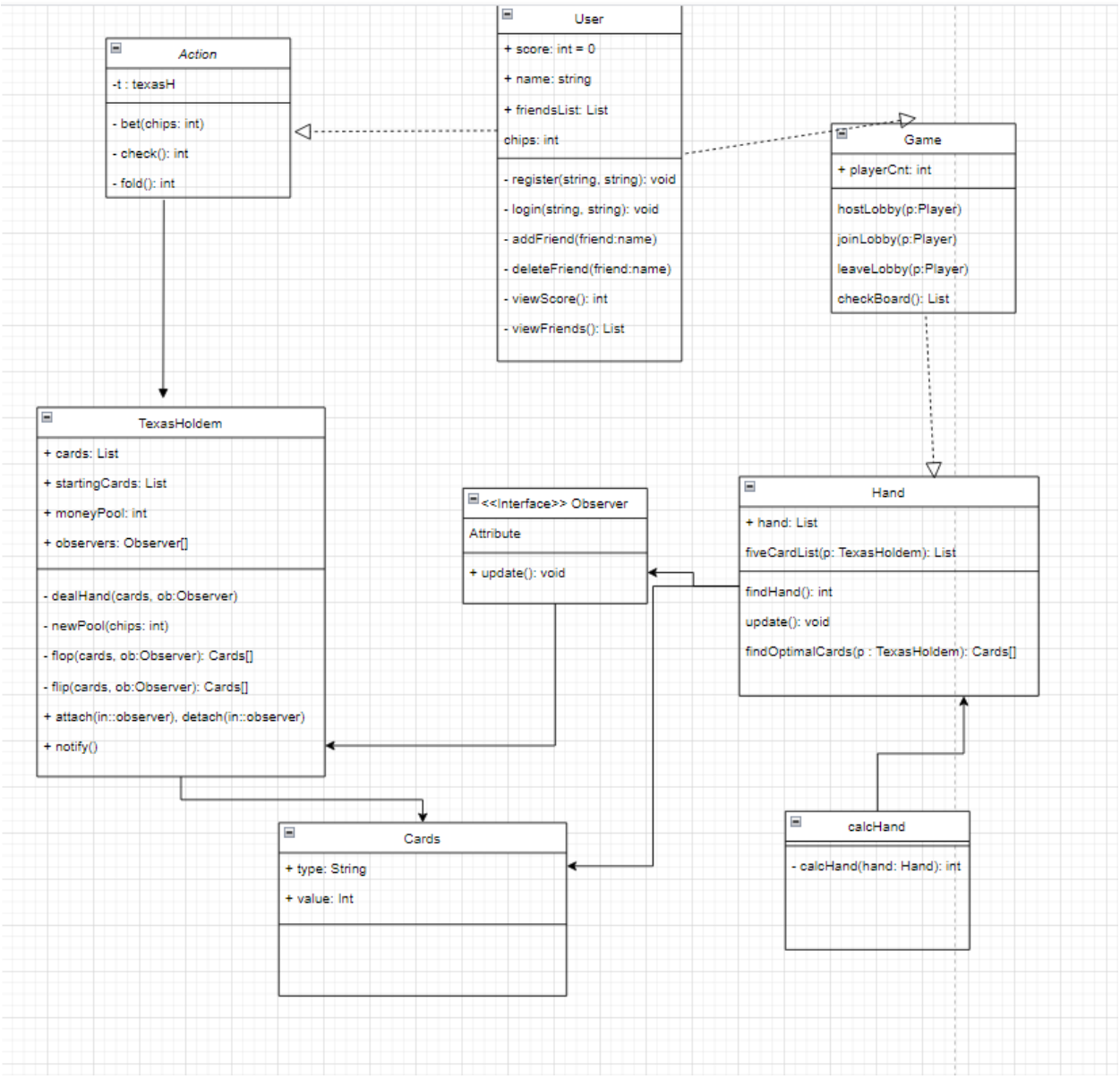
good way of keeping track of friends and in order to add other people in the game we needed to host it server-side which we did not get to either. Some of our major pitfalls can be attributed to elements which were in our control – like project management, communication, planning – but also to elements that were not in our control – like sickness (COVID) and family issues. This is certainly not an excuse for not meeting our deliverables since external problems are something that all developers have to deal with, though it is a valuable lesson as to the importance of planning and leaving extra time for situations that are unexpected. Ultimately, the state of our final system was passable, yet unsatisfactory in our eyes – but what we've learned and took away from the process was invaluable and we are both extremely appreciative of our failures and successes throughout the development as, through them, we've grown into much more capable and aware programmers.

Final Class Diagram and Comparison Statement

Project 5 Class Diagram:



Final Class Diagram:



DISCLAIMER: Descriptions of the design patterns used in our project are located below in order to reduce disorder and increase readability of the UML diagram above. We hope this is an acceptable alteration of the project assignment!

Design Pattern #1: Observer

The observer design principle is a medium that allows objects to receive notifications whenever the state of another object is changed or altered. In our system, an observer is used to notify the Player class that the dealer has dealt a card to the player's hand after they have joined a table. The notify method in the Player class calls the calcHand method in the Cards class to initiate the hand analyzing and classification processes – which is the alteration from the dealing game state to the decision game state where a player can then choose to continue or fold depending on how the hand is classified.

Design Pattern #2: Iterator

The iterator design pattern creates a pathway to sequentially access elements of a collection. In our system, we utilized the iterator pattern to handle the initialization of a deck of cards whenever a new game of Texas Holdem is created. Our implementation of this involves an iterator class that contains a list of all the cards in a deck and a pointer to the current position in the list. This class also provides methods for accessing and manipulating the list of cards in regards to its use cases in the Texas Holdem environment. For example, the iterator class is invoked when dealing the next card from a deck, getting the size of the deck, and shuffling the deck. Additionally, the use of this pattern also allows us to check whether the deck has been exhausted, though the case is unneeded in the way our project is currently designed. This functionality would become useful, though, if we provided the user with several rule sets they can pick from – some of which would make use of all the cards in the deck.

Design Pattern #3: Decorator

We have implemented the Decorator design pattern within our codebase. We thought that a nice subtle feature to add was the addition of an expression after winning a hand in poker. Some people would rather keep to themselves whereas others like to flaunt themselves like a flamingo. Unlike the RotLa simulation, the celebration or in our case expression that players have after winning a hand will be up to the player. The player will have the option to stay silent, or cheer, or even stand up (virtually since it is an app). The classes that we used to implement the decorator design pattern are the classes postWin and expressionsDecorator.

Design Pattern #4: Singleton [factory]

During our OOAD project, we looked into design patterns to load or spawn our cards in. We did not want anyone to be able to make an instance of any card. This would break the competitive integrity of Texas Holdem. Everyone from a group should be playing from the same deck of

cards. The singleton design pattern features a private constructor with a static method to access that single instance. In the Cards class, the deck constructor is private and provides methods to return (singleton) a single instance of the deck and reset (shuffle) the deck if needed.

In regards to the evolution of our UML diagram from Project 5 to Project 7, the main differences are attributed to the addition of the design patterns. In Project 5, we had a base idea of what our classes and codebase would look like without the design patterns we were going to use because we were not sure which ones would be appropriate/applicable to our implementation of Texas Holdem. As we progressed through the weeks and started to actually write code, we began to notice places where we could replace what we currently had with a particular design pattern to improve efficiency and readability. For example, our original implementation of the card deck was just a for loop that assigned an empty instance with a suit and a value. However, the introduction of the iterator pattern here was quite intuitive since we were ultimately traversing a collection of elements and assigning certain attributes. In the final UML diagram, we also introduced two new classes (Cards and calcHand) which assisted in better modularity of the functionalities of the system. The new classes also increased the robustness of our application by accounting for certain elements in Texas Holdem (such as calculating the strength of a hand based on the cards dealt) that we had previously overlooked or neglected. Finally, the last major difference between the initial and final versions of the UML diagram was the refactoring of existing classes, in which we included new methods or functions to account for new scopes as the project went on.

Third-Party Code vs Original Code Statement

No Third-Party code was used in the development of the project, as popular frameworks or tools were not particularly applicable to creating a card game. However, if we were to do this project again, it would certainly be beneficial to do some additional research to find any libraries or frameworks that could help in the general logic of the game – whether that be calculating hands (royal flush, pair, etc.) or creating an AI/computer controlled player that could join games. In all honesty, we definitely overlooked the potential advantages of outside sources and unfortunately that cost us a lot of time where we could've implemented additional features.

We did, however, find a helpful video on YouTube that covered all the elements that a game of Texas Holdem entails pragmatically, which helped us refine our technical approach to the project. The link to the video is: https://www.youtube.com/watch?v=QvIKDImmEhQ

Statement on the OOAD process for your overall Semester Project

One of the key design process elements that our group experienced was debating the use of one design pattern over another. We needed to consider both the pros and cons between using a

factory to create the cards or to use the singleton design pattern. We would say that this is a positive issue to have. As aspiring software developers, we would need to weigh the pros and cons.

In terms of the positive, in order to mitigate problems with combining pieces of code together (since we both made sections of the project separately and collected them via Github) the utilization of the UML diagram was beyond helpful, With this diagram, we were able to break the code into parts that were closely related. Additionally, the use of design patterns helped us in this process by creating convenient “bundles” of code that could be distributed.

An issue that our team experienced in the OO semester project was the unfamiliarity of some of the design patterns. Although we had gone over the concepts together in class, it is hard when you try to implement it yourself. In class, you shake your head yes and it looks simple enough but it really isn't the case.

Last but not least, our team had some positive elements as well. The feeling when something that you have been working on finally clicks and runs. Sort of the opposite of the issue above but it is such a satisfying feeling. We hope to chase this feeling and never be contempt with our work.