Project 7
Submitted by Ankur Sharma & Lincoln Rychecky

Final System State

These were the features identified in Project 5:

Login/ Signup Screen:
Users can sign up for an account utilizing a new username and password.
Users can login using a pre-existing username and password.

> For the sake of our system we will be simply storing usernames and passwords in a mySQL database in order to keep track of them.

Mainscreen:
Users can select to view their overall win loss record/ number of games played.

> This data will be stored in a mySQL database.

Users can select “Begin Game” in order to deal cards and start a game.

Gameplay Screen:
The gameplay screen will have a card in the middle representing the current card that was played last.
A turn indicator will be displayed showing whose turn it is and what number turn the game is currently on.
Show a stack of cards to pick from.
Show the user’s cards and highlight the playable ones for the current turn or suggest picking from a stack of cards.

Computer Player:
The computer player will implement the logic necessary to play the game of uno against the player.
An easy and hard mode will be implemented.

> In the easy mode the computer player will take a naive approach to playing the game and only play cards of the same color until it runs out, then it will begin to play matching numbers.
>
> The hard mode computer player will utilize a more advanced strategy that will attempt to guide the game to allow for most of its cards to be played by utilizing numbers and colors to its advantage.

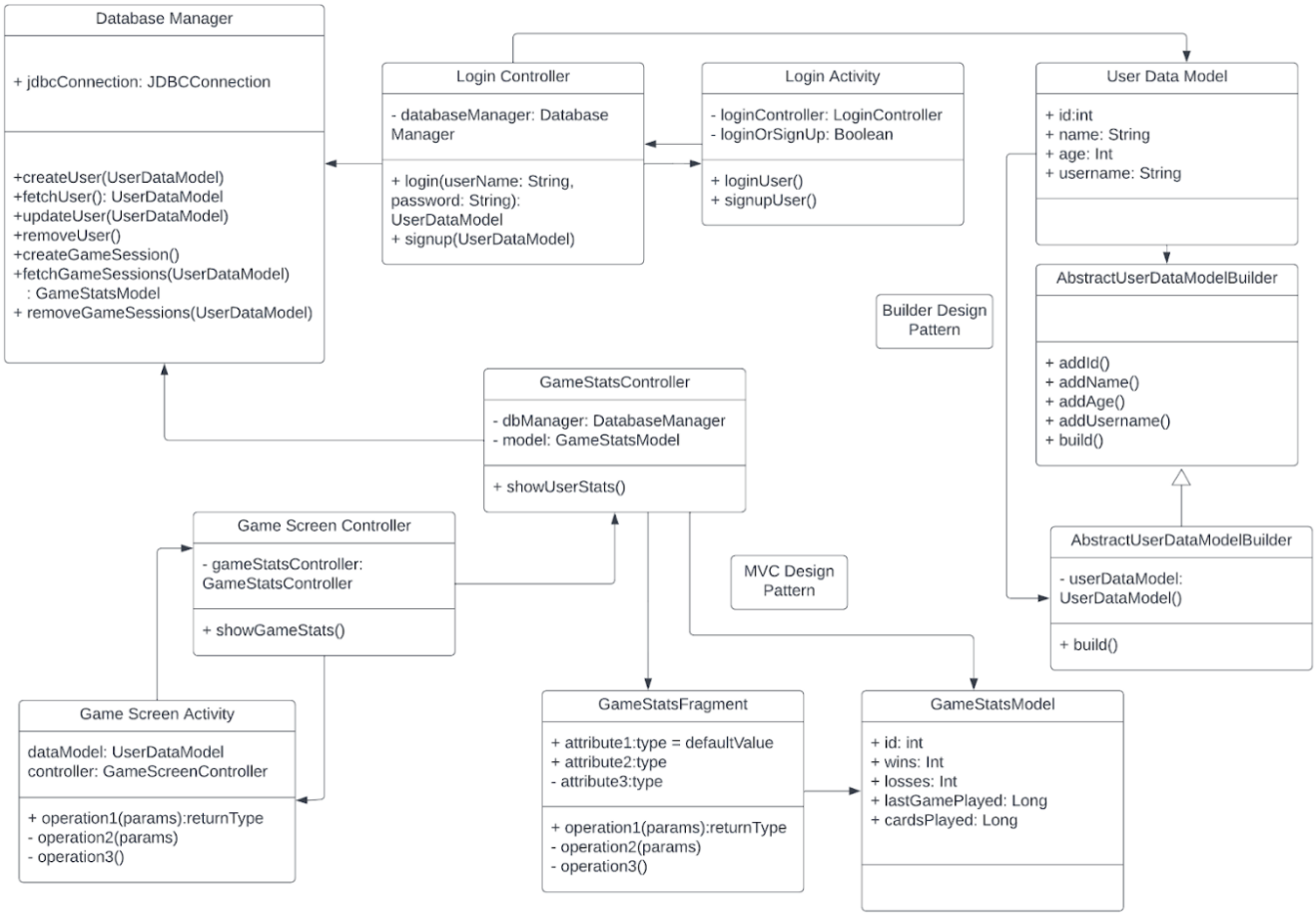
Based on project 5 and 6, everything has been implemented but the ability for a user to view their statistics. A login screen which utilizes an Elephant SQL cloud database in order to store usernames and passwords has been implemented. The Elephant SQL cloud database also required the creation of a google cloud web service which provided login and db query services. The mainscreen was implemented in its totality aside from the stats screen. The reason the stats screen was not implemented was because of the time crunched we faced when implementing the db connection. JBDC SQL connectors are no longer compatible with android applications due to a version update and all database ops are required to be handled by asynchronous http callouts to web services. Initially we implemented the database manager with the Postgresql JBDC connector, but ran into major dependency issues when building the entire project. Having to build out an entire web service and client program for connection to our cloud native Elephant SQL database turned out to be a laborious task that prevented us from implementing the stats screen. We believe the work done to build out the DB web service is a great feature to replace the stats screen though.

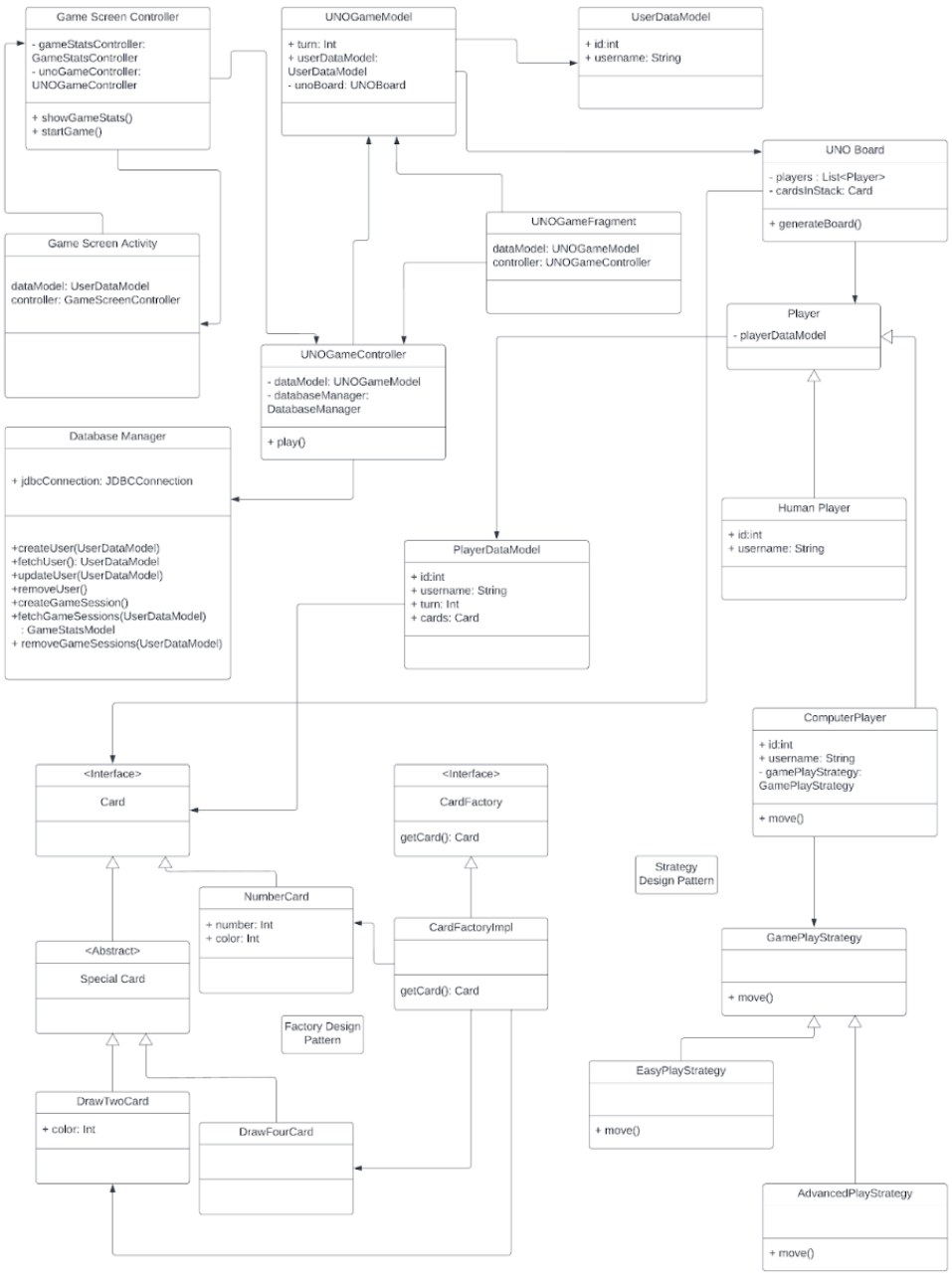
The gameplay screen is implemented in its entirety along with the computer player. We did opt to not implement a ‘hard’ strategy for the computer player due to the massive complexity in logic that we uncovered when it comes to implementing an intelligent uno computer player. We figured out time was better spent working on the rest of the codebase rather than laboring over uno logic details for a harder computer player.

In the end, our app does everything that we intended for it to do in the first place. Allow a user to login and persist user login data and allow a user to play an UNO game.

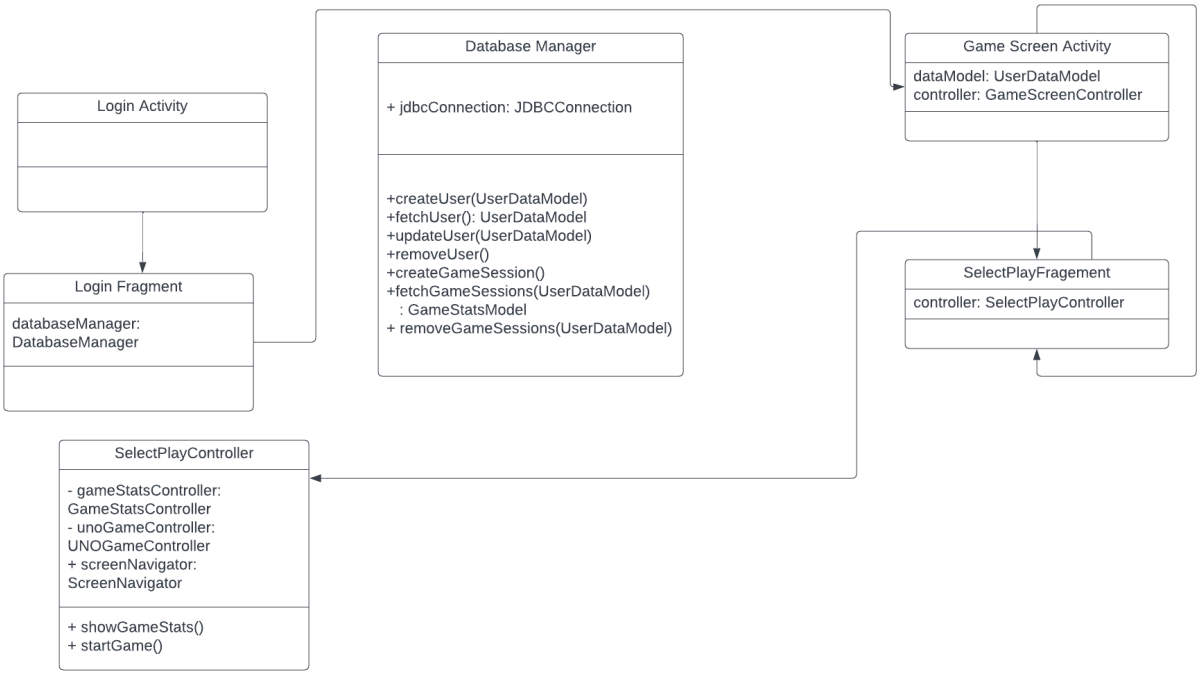
Final Class Diagram and Comparison Statement

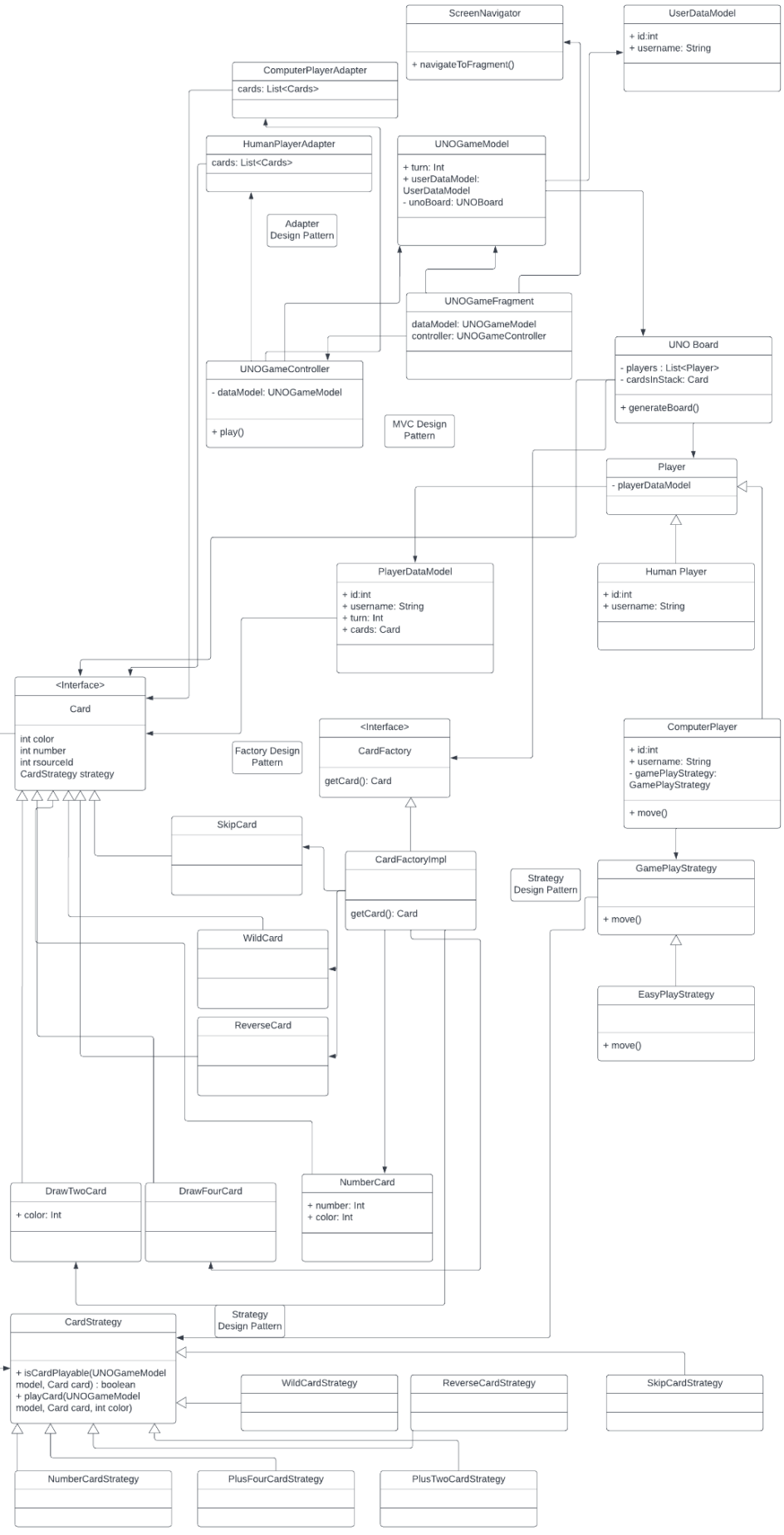
Original Class Diagram





Updated Class Diagram


Login Activity
Database Manager
+ jdbcConnection: JDBCConnection
+createUser(UserDataModel)
+fetchUser(): UserDataModel
+updateUser(UserDataModel)
+removeUser()
+createGameSession()
+fetchGameSessions(UserDataModel)
: GameStatsModel
+ removeGameSessions(UserDataModel)
Game Screen Activity
dataModel: UserDataModel
controller: GameScreenController
Login Fragment
databaseManager:
DatabaseManager
SelectPlayFragement
controller: SelectPlayController
SelectPlayController
- gameStatsController:
GameStatsController
- unoGameController:
UNOGameController
+ screenNavigator:
ScreenNavigator
+ showGameStats()
+ startGame()


ScreenNavigator
+ navigateToFragment()
UserDataModel
+ id:int
+ username: String
ComputerPlayerAdapter
cards: List<Cards>
HumanPlayerAdapter
cards: List<Cards>
Adapter Design Pattern
UNOGameModel
+ turn: Int
+ userDataModel: UserDataModel
- unoBoard: UNOBoard
UNOGameFragment
dataModel: UNOGameModel
controller: UNOGameController
UNOGameController
- dataModel: UNOGameModel
+ play()
MVC Design Pattern
UNO Board
- players : List<Player>
- cardsInStack: Card
+ generateBoard()
Player
- playerDataModel
PlayerDataModel
+ id:int
+ username: String
+ turn: Int
+ cards: Card
Human Player
+ id:int
+ username: String
<Interface>
Card
int color
int number
int rsourceId
CardStrategy strategy
Factory Design Pattern
<Interface>
CardFactory
getCard(): Card
ComputerPlayer
+ id:int
+ username: String
- gamePlayStrategy: GamePlayStrategy
+ move()
SkipCard
CardFactoryImpl
getCard(): Card
Strategy Design Pattern
GamePlayStrategy
+ move()
WildCard
ReverseCard
EasyPlayStrategy
+ move()
DrawTwoCard
+ color: Int
DrawFourCard
NumberCard
+ number: Int
+ color: Int
CardStrategy
Strategy Design Pattern
+ isCardPlayable(UNOGameModel model, Card card) : boolean
+ playCard(UNOGameModel model, Card card, int color)
WildCardStrategy
ReverseCardStrategy
SkipCardStrategy
NumberCardStrategy
PlusFourCardStrategy
PlusTwoCardStrategy

Key Changes

- Introduced Card Strategy to delegate card functionality
- Simplified login structure

Third-Party code vs. Original Code Statement

Our code is all original, but does make use of the Dagger framework in order to implement dependency injection.

No large existing code samples (outside of reference to standard documentation) were used in the creation of this project.

The adapter and observable design pattern implementations are part of the native android framework.

Statement on the OOAD process for this Project

#1) The process of diagramming our system utilizing UML was a very positive experience for the team. Our UML stayed relatively consistent throughout the project, and did in projects throughout the semester, which is a testament to the fact that serious thought and consideration put in at the beginning of a project is highly beneficial and can give a relatively accurate look at a project before implementation begins. Having the UML diagram to refer to, especially when a codebase starts to expand beyond a trivial size, is invaluable during implementation.

#2) A negative experience that we had related to analysis and design of the project ironically is related to our UML diagramming beforehand. We had diagrammed our initial database manager to make use of the JDBC connector and had become somewhat committed to the idea of using this particular technology in the project. This ended up leading to quite a significant sunk cost in our project as we burnt more and more time trying to troubleshoot the various poorly documented errors we were getting. We could have avoided this time burn if we had just initially researched best practices for interacting with SQL databases in an Android app initially instead of trying to force a square peg into a round hole. I think the key takeaway from this is to not commit so strongly to a specific solution in the beginning without having researched best practices in order to identify roadblocks that may appear later with the less optimal solution.

#3) Another very positive design and analysis experience was activity diagramming. At the beginning, the activity diagram was rather tedious to design, but understanding the flow of system activities proved very useful. A good example would be in the section relating directly to the card game itself. Being able to think through the various logical parts of putting together uno game logic helped immensely when implementing the game controller and the various classes that it made use of. For example, being able to understand what a controller needs access to in terms of game and deck states for each player was instrumental in informing our design.