System Design Document - Group 16

1 Introduction

The project is an implementation of the game UNO using JavaFX for the GUI, a server-client architecture for multiplayer functionality and an MVC design pattern for separation of concerns. The model is separated from the View and Controller with minimal coupling.

2 System architecture

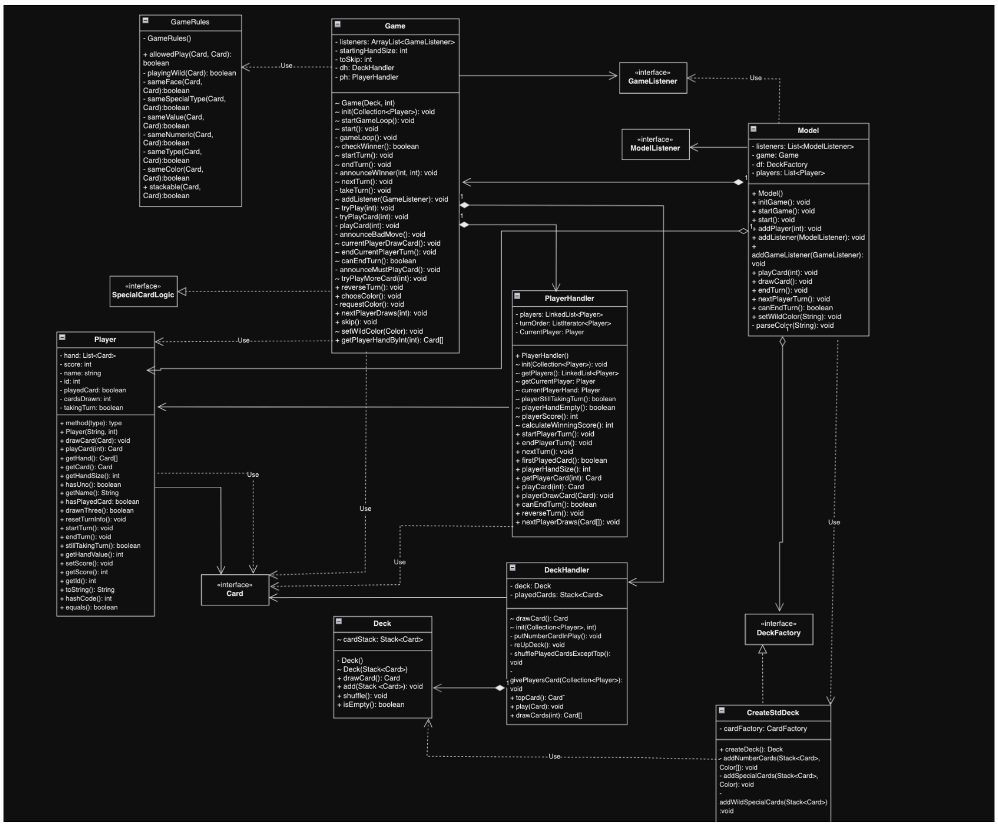
The system has a Client - Server architecture. The server manages game logic, client connections and game state, while the client is responsible for the UI and interacts with the server.

3 UML diagrams

The UML class diagrams are divided to show different aspects of the system.

3.1 Overview of the Model

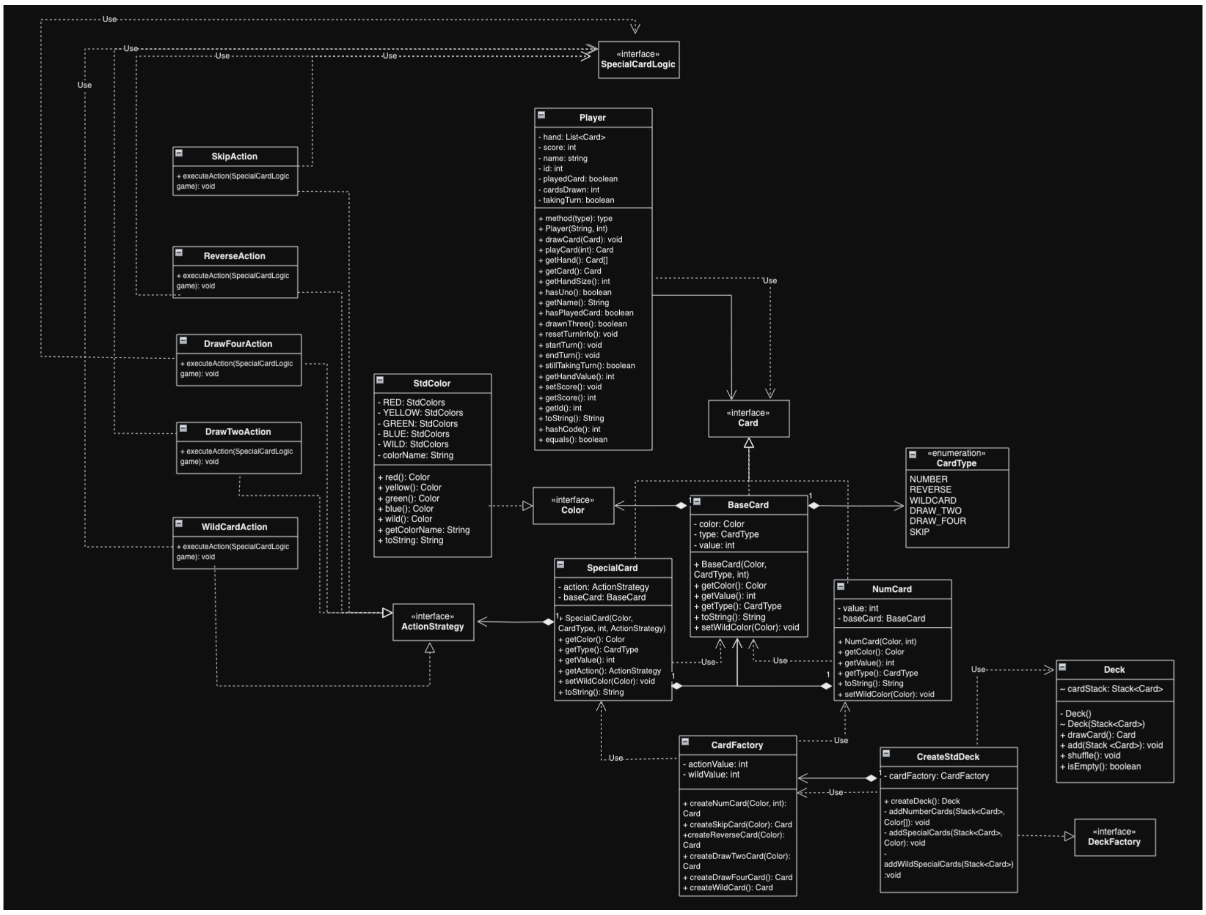
Here we have the main section of the Model. The Model class should be acting as a kind of facade with which the View and Controller interact. Beyond this facade, the Game class contains the virtual game board and the game loop that handles all the game logic of taking turns, playing cards et.c. To do this it has a linked list containing all the players whose Iterator handles the turn order of players (or descending order Iterator if the turn order has been reversed!), as well as a Deck, which is a decorated Card Stack, and a normal Card Stack of all the already played cards.
We are using an observer pattern to inform the other parts of the system of updates to the game state and when player actions are needed. The model also handles some of the setup with the functionality for receiving how many players will be playing, the user ID of those players and providing this as well as what deck to use for the game (only standard deck is implemented).



3.2 Game Logic (Model)

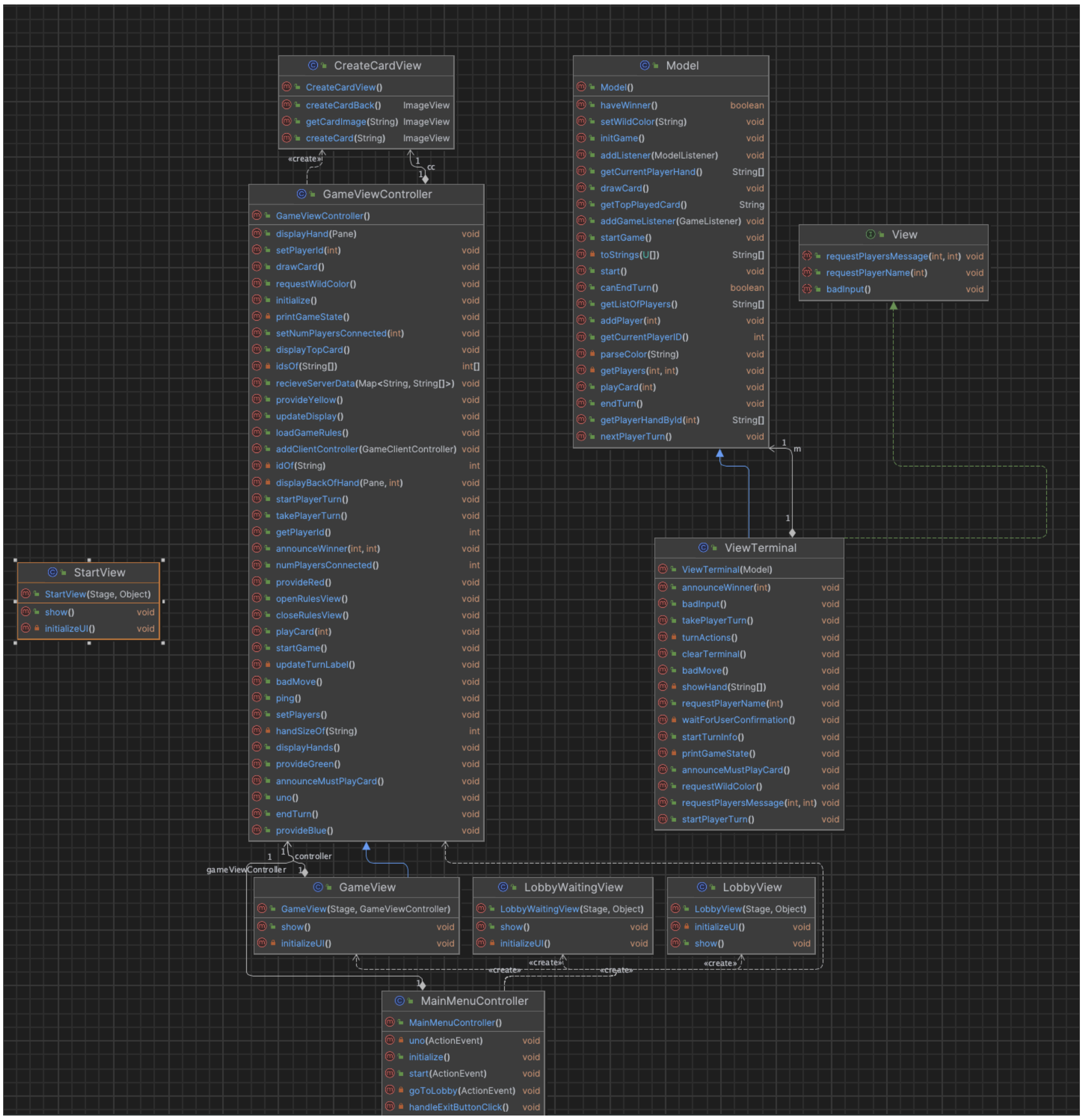
The cards are separated into two types, number cards and special cards. Both have a ‘BaseCard’ that contains the basic functionality and both are implementing the Card interface. This ensures the use of polymorphism and maintains the Dependency Inversion Principle.

Card and deck creation is handled through Factories, allowing for easy extension to support new types of cards and decks. The special cards actions are implemented using the strategy pattern, enabling easy addition of new actions.



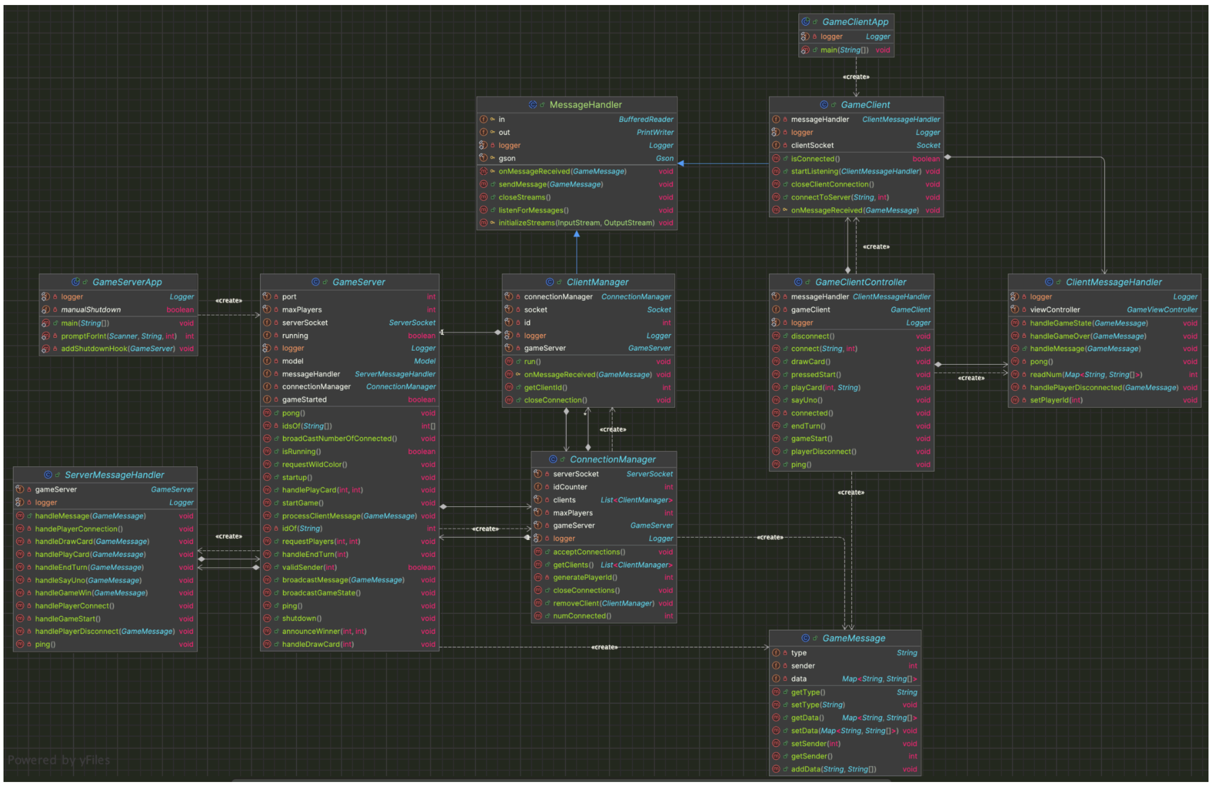
3.3 View

GameViewController does not as of yet follow proper MVC separation of View and Controller. Since JavaFX is used the separation of these ended up being much more complicated than what was thought initially.



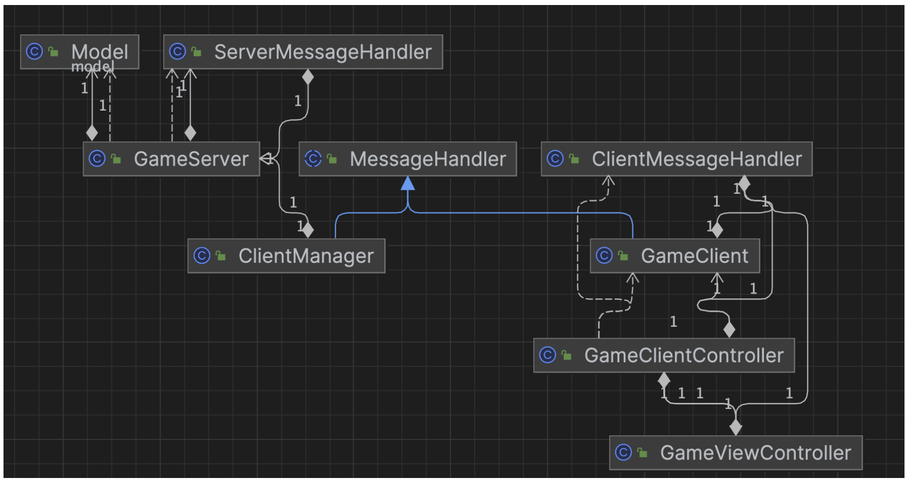
3.4 Server

The design follows object oriented principles like encapsulation and SoC. New game actions can be added without modifying existing code. However, it could be further improved by implementing additional interfaces to reduce coupling.



3.5 Overview of MVC

When a button is pressed, the controller sends a message containing relevant information to the server. The server calls the appropriate method in the model to update the game state. After that, the server sends a message to GameClients, providing information for updating the view. The GameControllerClient then calls methods in the view.


Model
model
ServerMessageHandler
GameServer
MessageHandler
ClientMessageHandler
ClientManager
GameClient
GameClientController
GameViewController
1