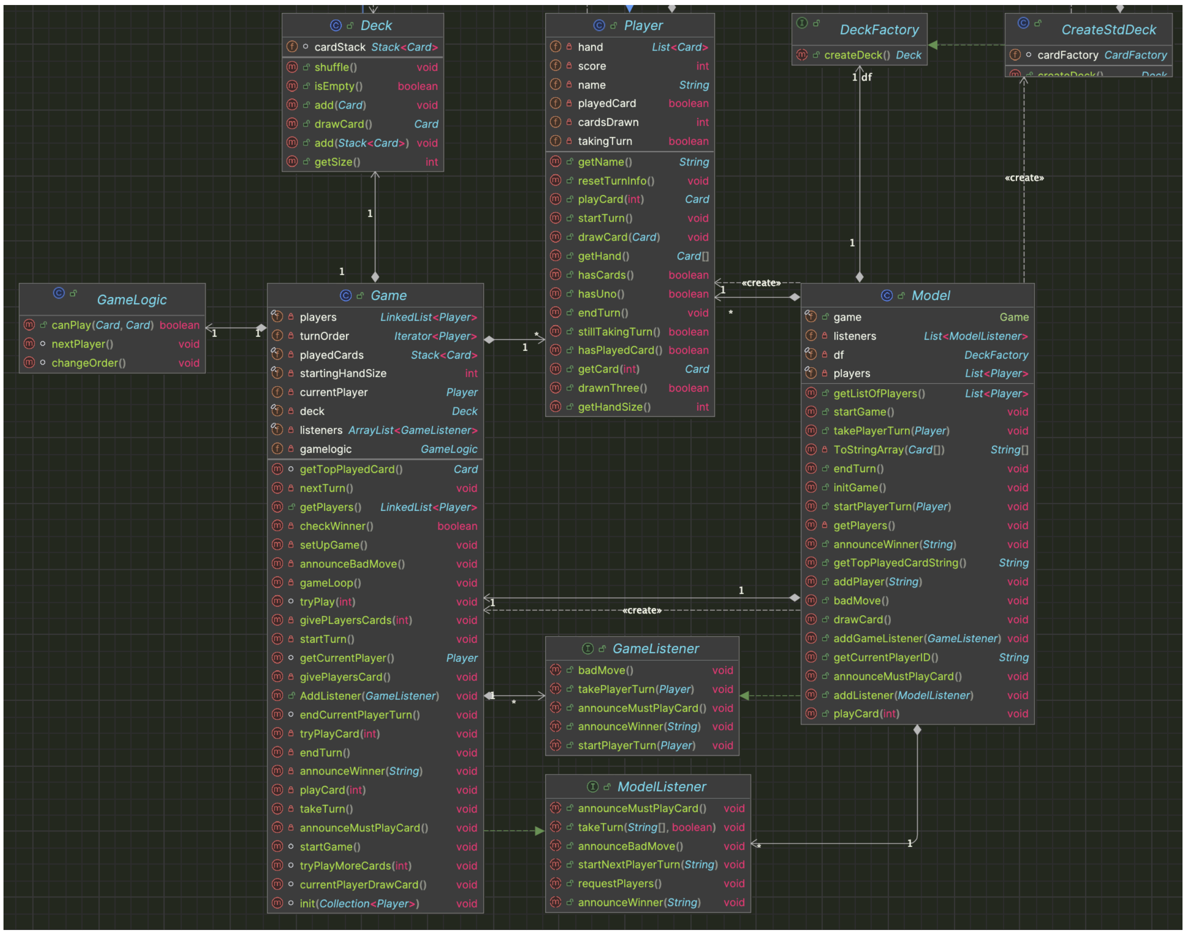
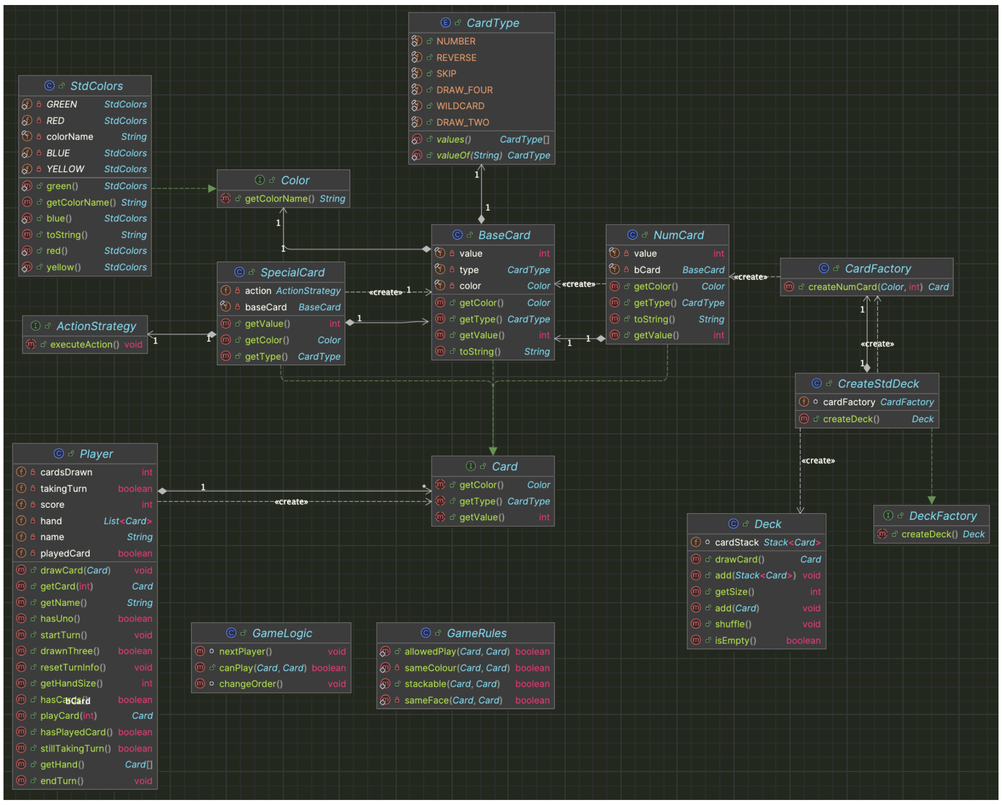
System Design Document - Group 16

We try to follow the MVC design pattern, as of now it is not super well implemented, the Model is separate from the View and Controller with almost minimal coupling but currently the View and Controller parts are intermingled and will be separated properly.

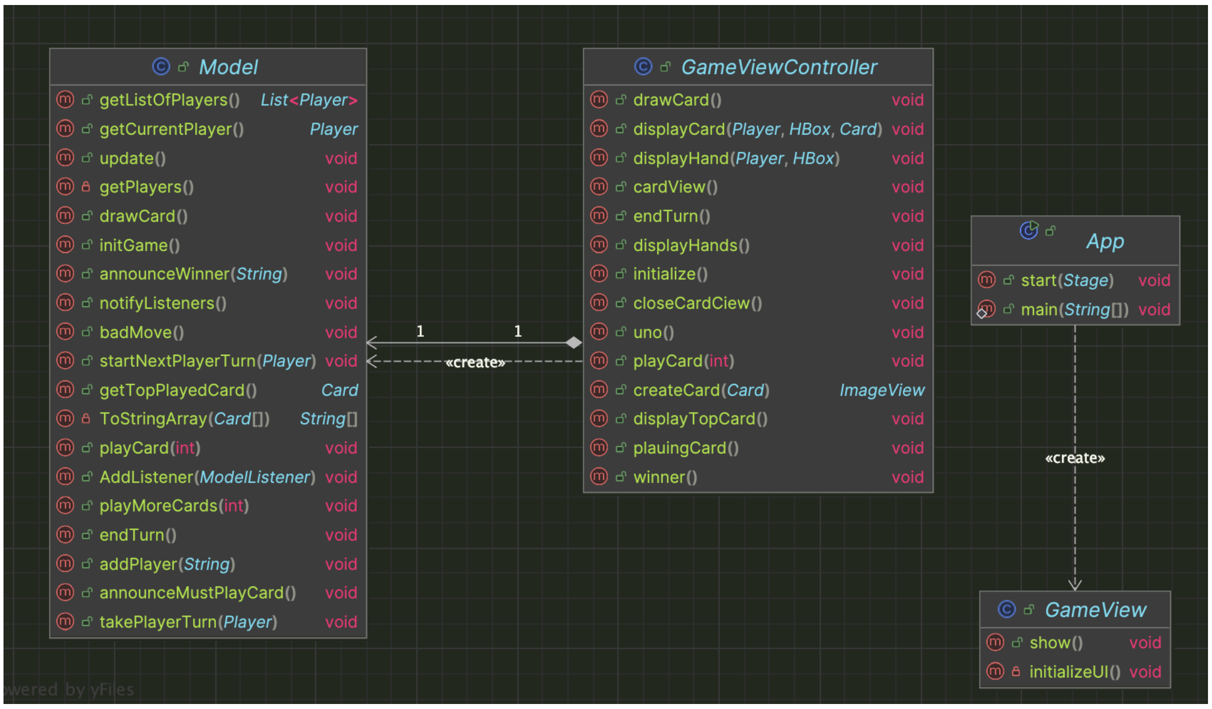
We have separated the UML diagrams into a few separate parts.
First we have the main section of the Model, with the plan that the Model class should be acting as a kind of facade with which the View and Controller interact. Beyond this facade, the Game class contains the virtual game board and the game 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. Currently the setup of the model and game listeners are redundant and needlessly complicated. Model is the only game listener and all signals it receives are sent on to the model listeners, however this step is completely unneeded and the parts that need the signals from the game should simply receive them directly. Right now the model also acts as a kind of setup lobby 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).



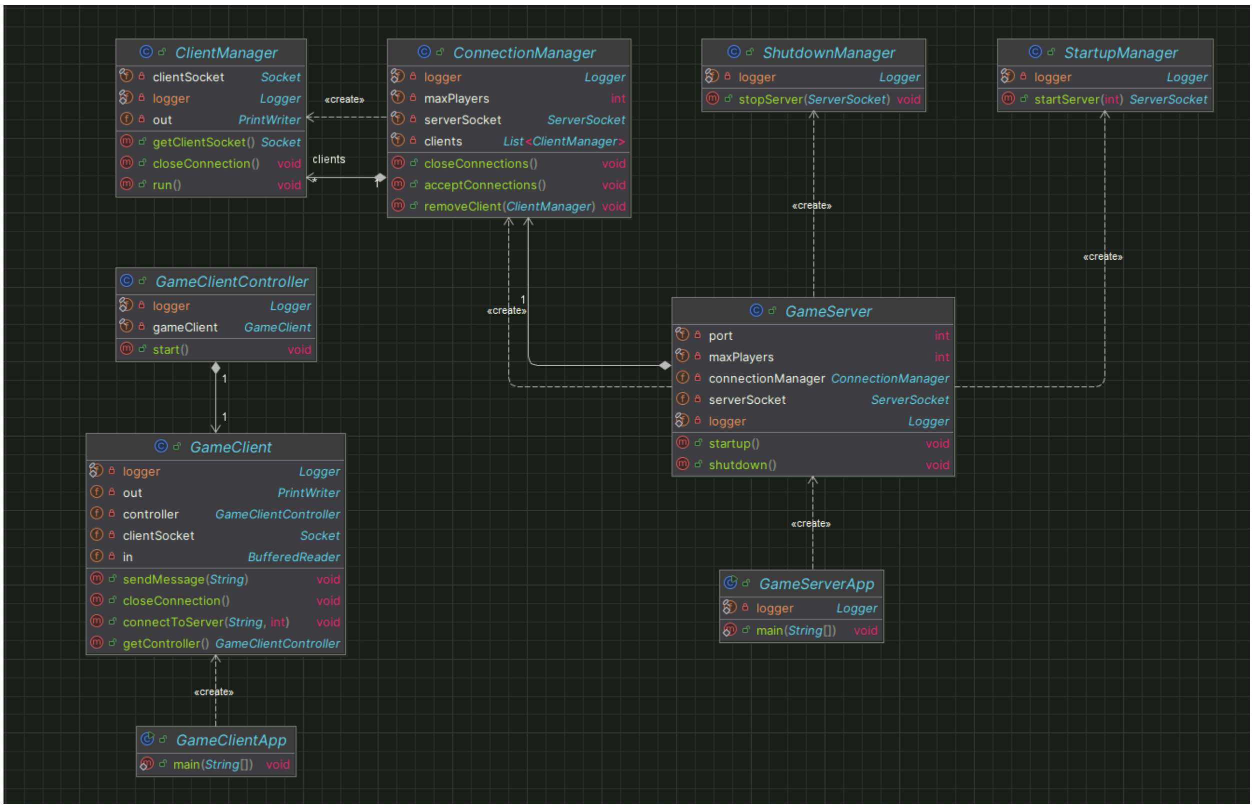
UML of the game logic. Currently there is a lot of overlap between the GameLogic and GameRules classes and will be differentiated and separated later on. The cards are separated into two types, number cards and special cards. Both have a ‘BaseCard’ with the basic functionality and both are implementing the Card interface to make use of polymorphism and maintain the Dependency Inversion Principle. Creation of cards and the deck is done using Factories allowing for different kinds of cards and decks to be implemented in the future.



UML of the view. As mentioned above, GameViewController does not as of yet follow propper MVC separation of View and Controller.
Also note that *App* has been renamed to *AppWindow* since this UML was created.



Multiplayer UML, game logic is not implemented yet.


ClientManager
clientSocket Socket
logger Logger
out PrintWriter
getClientSocket() Socket
closeConnection() void
run() void
«create»
clients
*
1
ConnectionManager
logger Logger
maxPlayers int
serverSocket ServerSocket
clients List<ClientManager>
closeConnections() void
acceptConnections() void
removeClient(ClientManager) void
ShutdownManager
logger Logger
stopServer(ServerSocket) void
StartupManager
logger Logger
startServer(int) ServerSocket
«create»
«create»
1
«create»
GameClientController
logger Logger
gameClient GameClient
start() void
1
1
GameServer
port int
maxPlayers int
connectionManager ConnectionManager
serverSocket ServerSocket
logger Logger
startup() void
shutdown() void
«create»
GameServerApp
logger Logger
main(String[]) void
GameClient
logger Logger
out PrintWriter
controller GameClientController
clientSocket Socket
in BufferedReader
sendMessage(String) void
closeConnection() void
connectToServer(String, int) void
getController() GameClientController
«create»
GameClientApp
main(String[]) void