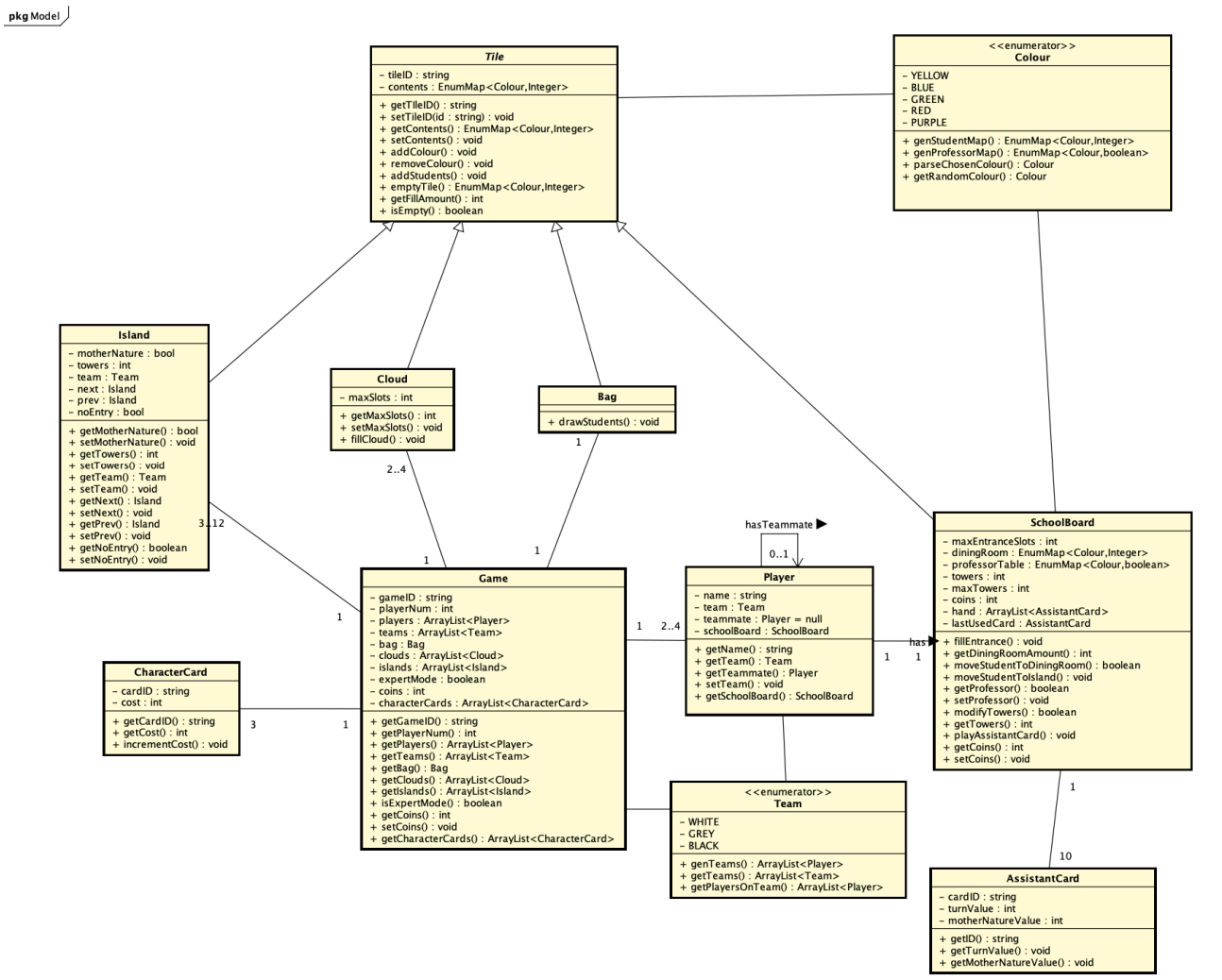
UML diagram Eriantys - Write-up 2 (Group GC33)

MODEL

We have moved object generation methods into controller classes and although not present in the model diagram we have also added Character Card subtypes for memory storage purposes (student disks, disabled colour, int: number of no entry tiles)

For client side we have opted for a reduced model (not pictured) which will be the one notifying the client's view



CONTROLLER

For the controller - based on your suggestion - we have considered a more centralised approach. The diagram shows the following classes:

-**LobbyController**: main server controller class server with access to all other controllers, its main role is to parse the **Message**s received from the lobby and call the appropriate controllers into action

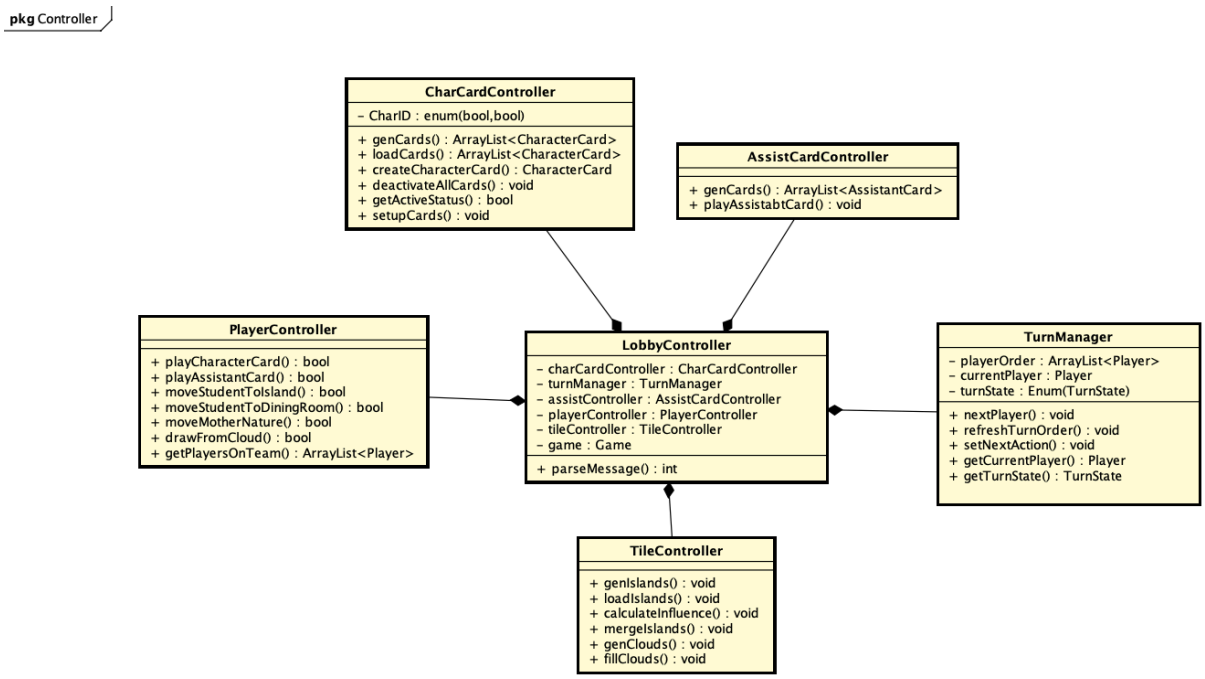
-**TurnManager**: manages game phases and player turns, it contains an ordered player list based on the last **Assistant Card** that was played (during the first planning phase this order is the same as the randomised one on the **LobbyController**)

-**CharCardController**: contains the factory method to generate **Character Card**s as well as methods to manage their active state or their effect

-**AssistantCardController**: contains method to generate **Assistant Card**s and to play them

-**PlayerController**: manages all actions that modify **Player**s

-**TileController**: contains methods to generate and setup **Tile**s, as well as calculating island influence and merging islands



NETWORK

The network diagram shows how our setup is mostly server side, with the client relying on a simple controller to send messages to the server based on view actions and parse received messages to update its model accordingly. Included in our diagram are the classes:

-**Client**: main class running client-side that establishes the connection between local MVC and server, it saves the current user's username so that the controller can generate messages with the appropriate sender, it also contains default values for socket connection and parsers for correct ip addresses

-**Server**: the main class running server-side, its role is to satisfy initial client requests (create lobby, join lobby and disconnect) before handing connection over to a Lobby thread, below you will also find a sequence diagram showing how this process works

-**Lobby**: implements the Runnable class, it sets up the game settings based on lobby master messages (player num and expertMode), then once all players have joined it creates a controller that in turn generates the game model. It contains a synchronised list of current players and **Message**s

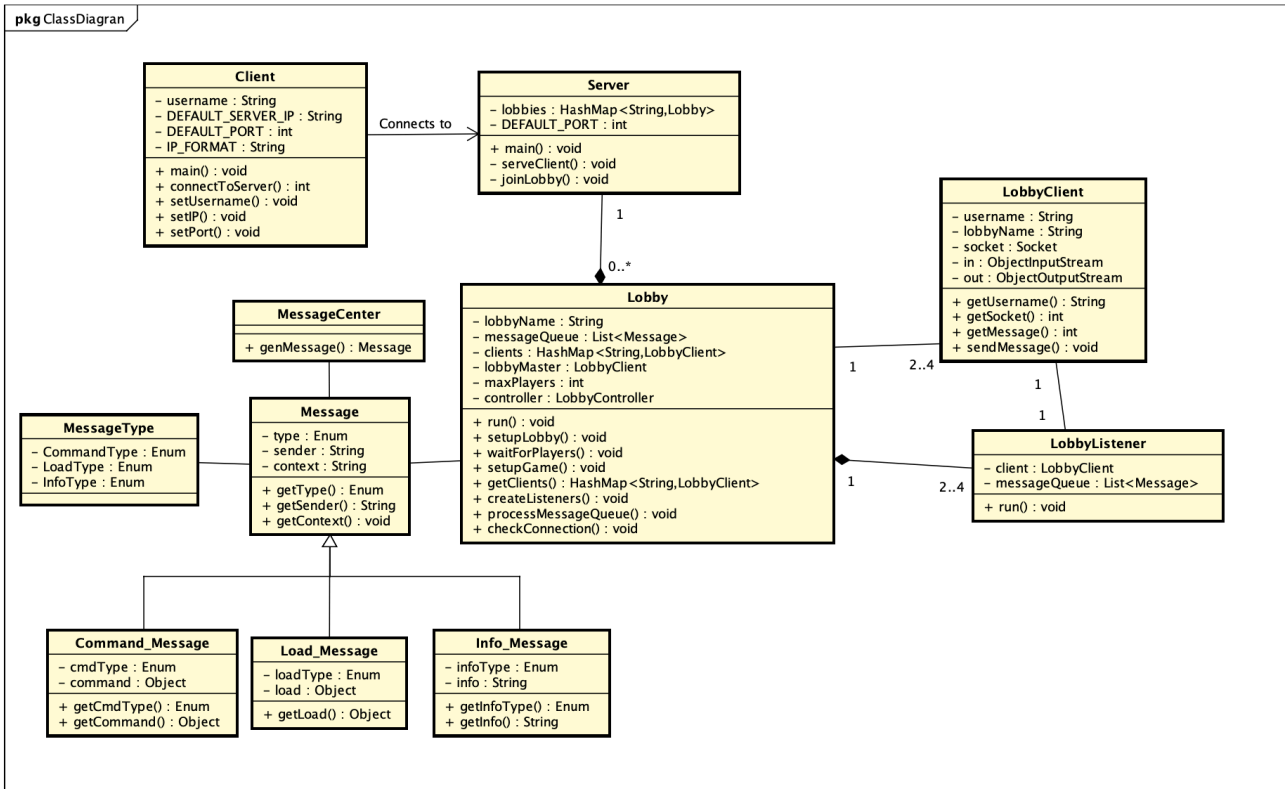
-**LobbyClient**: class identifying each client connected to a lobby, it also contains the methods to interact with the client (send and get message)

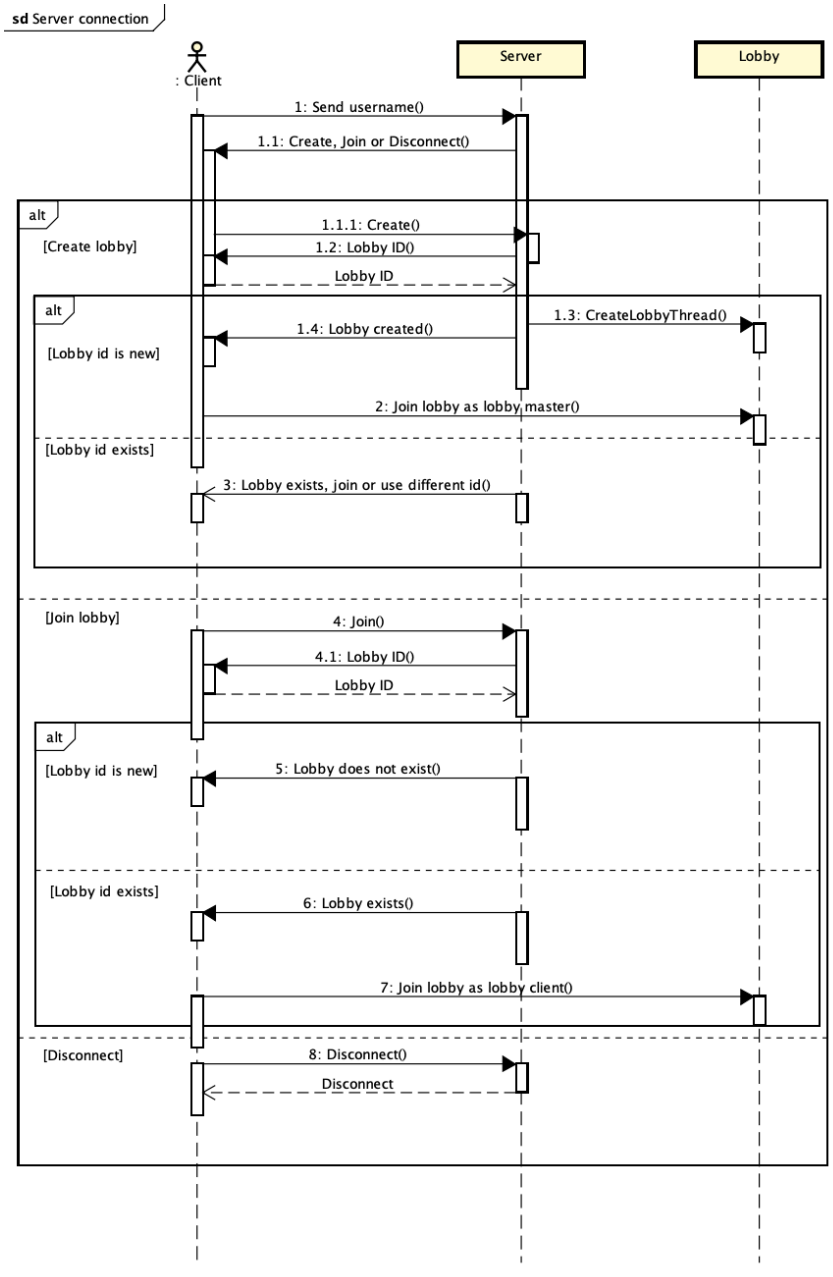
-**LobbyListener**: implements the Runnable class, it's main purpose is to be a daemon thread for each client so that it can stack **Message**s onto the **Lobby Message** queue

-**Message**: abstract object class to be serialised and sent across the network, it has several subtypes defined in **MessageType** enums to identify what is being transmitted and as such has implementation for these types (**Command_Message**, **Load_Message**, **Info_Message**)

-**MessageCenter** : contains the factory method to generate **Message**s

Network


pkg ClassDiagran
Client
- username : String
- DEFAULT_SERVER_IP : String
- DEFAULT_PORT : int
- IP_FORMAT : String
+ main() : void
+ connectToServer() : int
+ setUsername() : void
+ setIP() : void
+ setPort() : void
Connects to
Server
- lobbies : HashMap<String,Lobby>
- DEFAULT_PORT : int
+ main() : void
- serveClient() : void
- joinLobby() : void
1
0..*
LobbyClient
- username : String
- lobbyName : String
- socket : Socket
- in : ObjectInputStream
- out : ObjectOutputStream
+ getUsername() : String
+ getSocket() : int
+ getMessage() : int
+ sendMessage() : void
MessageCenter
+ genMessage() : Message
Lobby
- lobbyName : String
- messageQueue : List<Message>
- clients : HashMap<String,LobbyClient>
- lobbyMaster : LobbyClient
- maxPlayers : int
- controller : LobbyController
+ run() : void
+ setupLobby() : void
+ waitForPlayers() : void
+ setupGame() : void
+ getClients() : HashMap<String,LobbyClient>
+ createListeners() : void
+ processMessageQueue() : void
+ checkConnection() : void
1
2..4
1
1
MessageType
- CommandType : Enum
- LoadType : Enum
- InfoType : Enum
Message
- type : Enum
- sender : String
- context : String
+ getType() : Enum
+ getSender() : String
+ getContext() : void
LobbyListener
- client : LobbyClient
- messageQueue : List<Message>
+ run() : void
1
2..4
Command_Message
- cmdType : Enum
- command : Object
+ getCmdType() : Enum
+ getCommand() : Object
Load_Message
- loadType : Enum
- load : Object
+ getLoad() : Object
Info_Message
- infoType : Enum
- info : String
+ getInfoType() : Enum
+ getInfo() : String


sd Server connection
: Client
Server
Lobby
1: Send username()
1.1: Create, Join or Disconnect()
alt
[Create lobby]
1.1.1: Create()
1.2: Lobby ID()
Lobby ID
alt
[Lobby id is new]
1.3: CreateLobbyThread()
1.4: Lobby created()
2: Join lobby as lobby master()
[Lobby id exists]
3: Lobby exists, join or use different id()
[Join lobby]
4: Join()
4.1: Lobby ID()
Lobby ID
alt
[Lobby id is new]
5: Lobby does not exist()
[Lobby id exists]
6: Lobby exists()
7: Join lobby as lobby client()
[Disconnect]
8: Disconnect()
Disconnect