POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Prova Finale di Ingegneria del Software

Anno Accademico 2022-2023

My Shelfie

Federico Saccani – Francesco Spangaro – Luca Pedersoli – Luca Sanvito

Cosa è stato Implementato

2.3 Funzionalità Avanzate

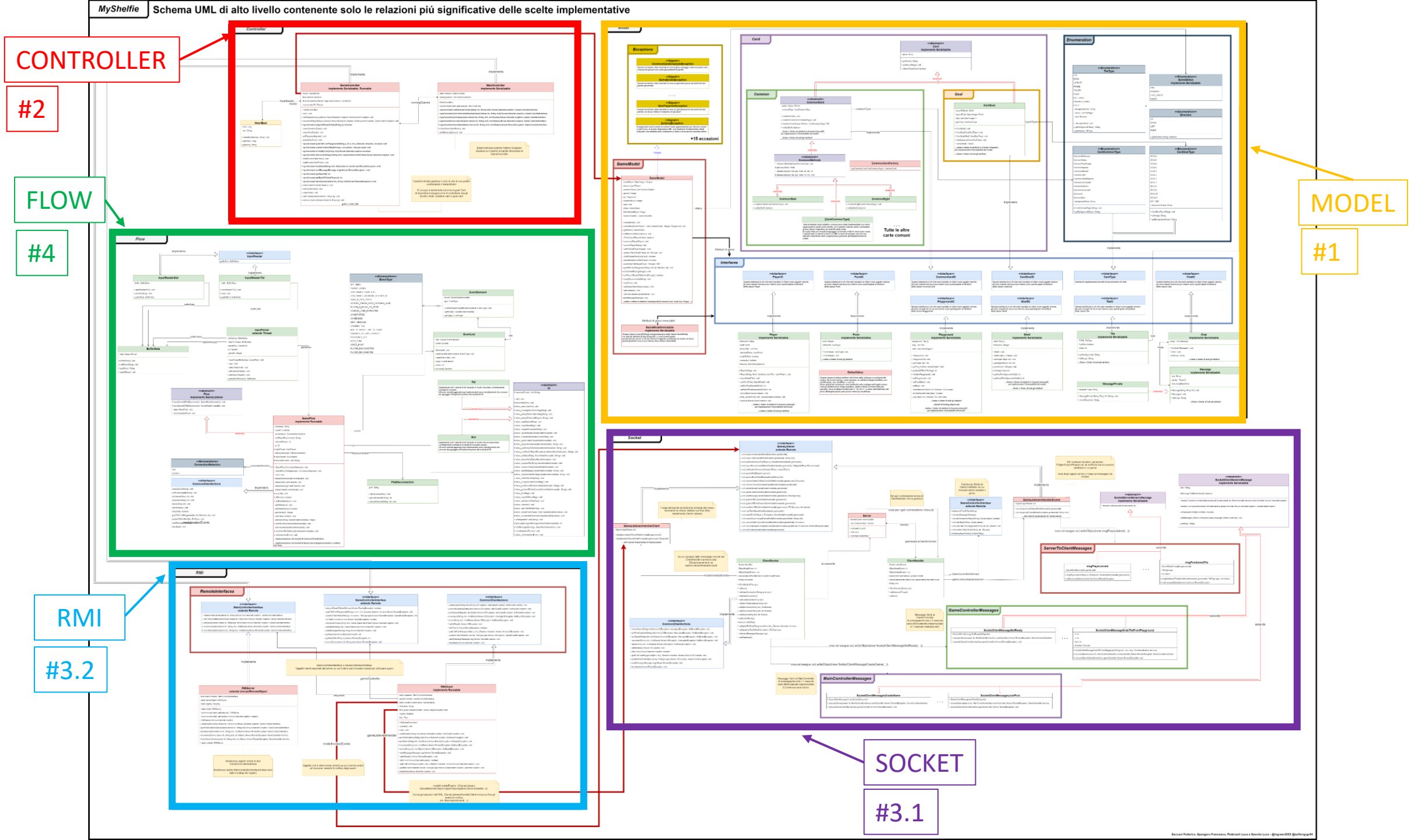
- **Partite multiple:** Realizzare il server in modo che possa gestire più partite contemporaneamente. Ai fini dell'implementazione di questa funzionalità aggiuntiva, le regole precedentemente specificate in merito alla creazione delle partite possono essere modificate in base alle esigenze implementative o di interfaccia utente.
- **Persistenza:** Fare in modo che il server salvi periodicamente lo stato della partita su disco, in modo che l'esecuzione possa riprendere da dove si è interrotta, anche a seguito del crash del server stesso. Per riprendere una partita, i giocatori si dovranno ricollegare al server utilizzando gli stessi nickname, una volta che questo sia tornato attivo. Si assume che il disco costituisca una memoria totalmente affidabile.
- **Resilienza alle disconnessioni:** I giocatori disconnessi a seguito della caduta della rete o del crash del client, possono ricollegarsi e continuare la partita. Mentre un giocatore non è collegato, il gioco continua saltando i turni di quel giocatore. Se rimane attivo un solo giocatore, il gioco viene sospeso a meno che non si ricolleghi almeno un altro giocatore, oppure scade un timer, che decreta la vittoria dell'unico giocatore rimasto connesso.
- **Chat:** Client e server devono offrire la possibilità ai giocatori coinvolti in una partita di chattare tra di loro, inviando messaggi (testuali) indirizzati a tutti i giocatori della partita o a un singolo giocatore.

Valutazione

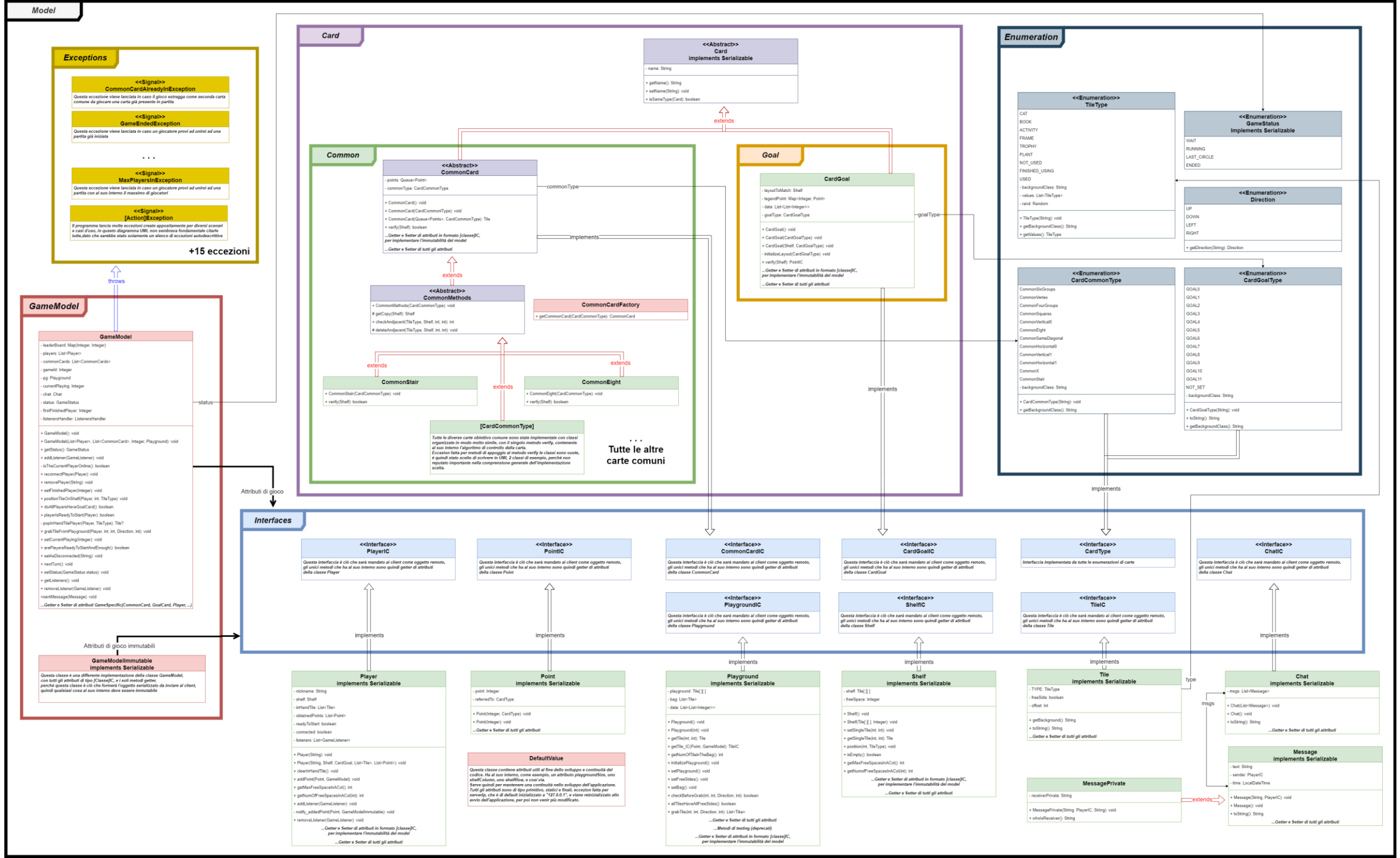
Requisiti Soddisfatti	Voto Massimo
Regole Semplificate + TUI + RMI o Socket	18
Regole Complete + TUI + RMI o Socket	20
Regole Complete + TUI + RMI o Socket + 1 FA	22
Regole Complete + TUI + GUI + RMI o Socket + 1 FA	24
Regole Complete + TUI + GUI + RMI + Socket + 1 FA	27
Regole Complete + TUI + GUI + RMI + Socket + 2 FA	30
Regole Complete + TUI + GUI + RMI + Socket + 3 FA	30L

Scelte e analisi implementative
UML

UML di Alto Livello completo

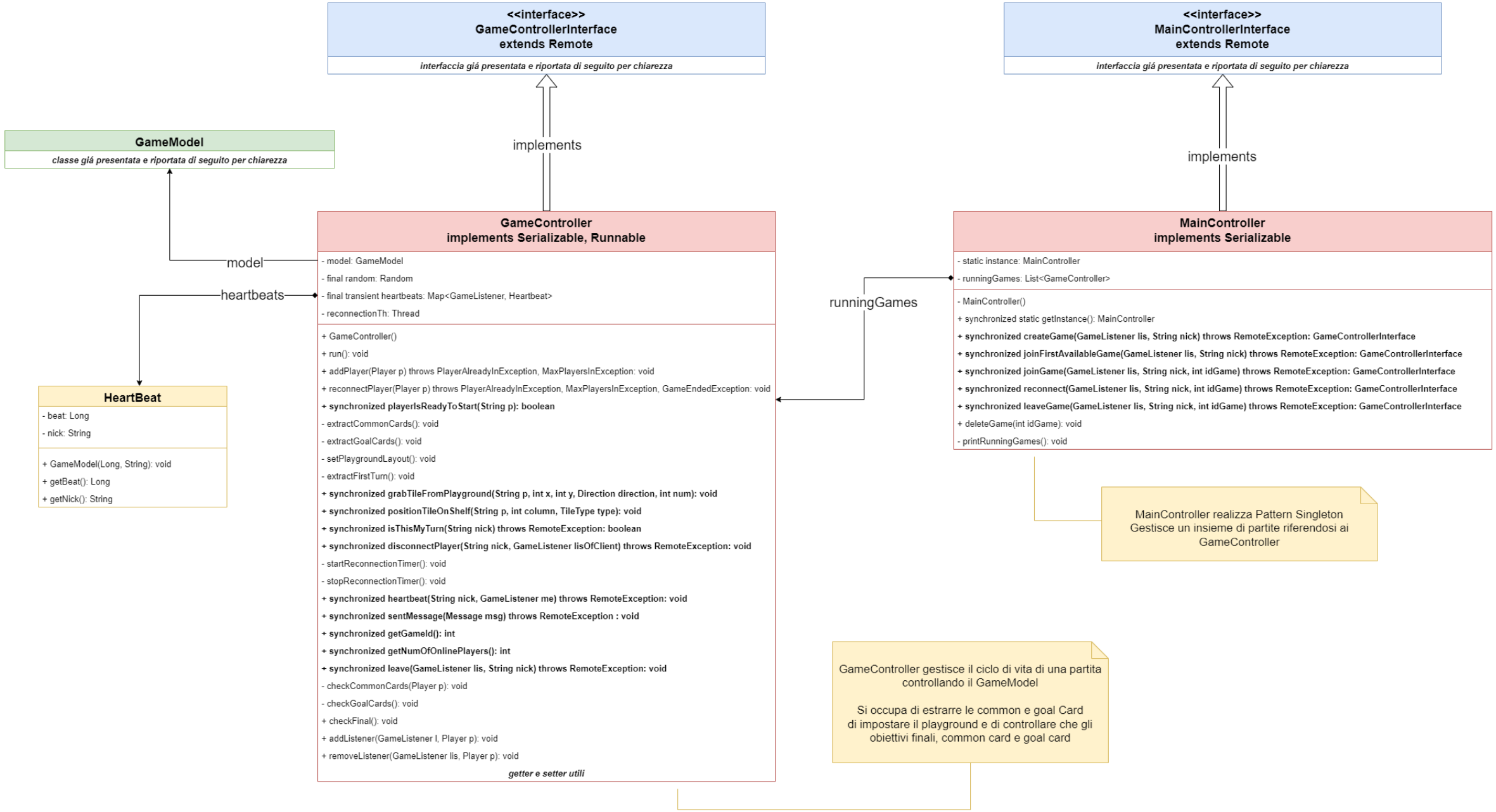

MyShelfie
Schema UML di alto livello contenente solo le relazioni piú significative delle scelte implementative
CONTROLLER
#2
FLOW
#4
RMI
#3.2
MODEL
#1
SOCKET
#3.1
Controller
Flow
RMI
RemoteInterfaces
Socket
Model
Exceptions
Card
Common
Goal
Enumeration
GameModel
Interfaces
ServerToClientMessages
GameControllerMessages
MainControlNerMessages

#1 MODEL

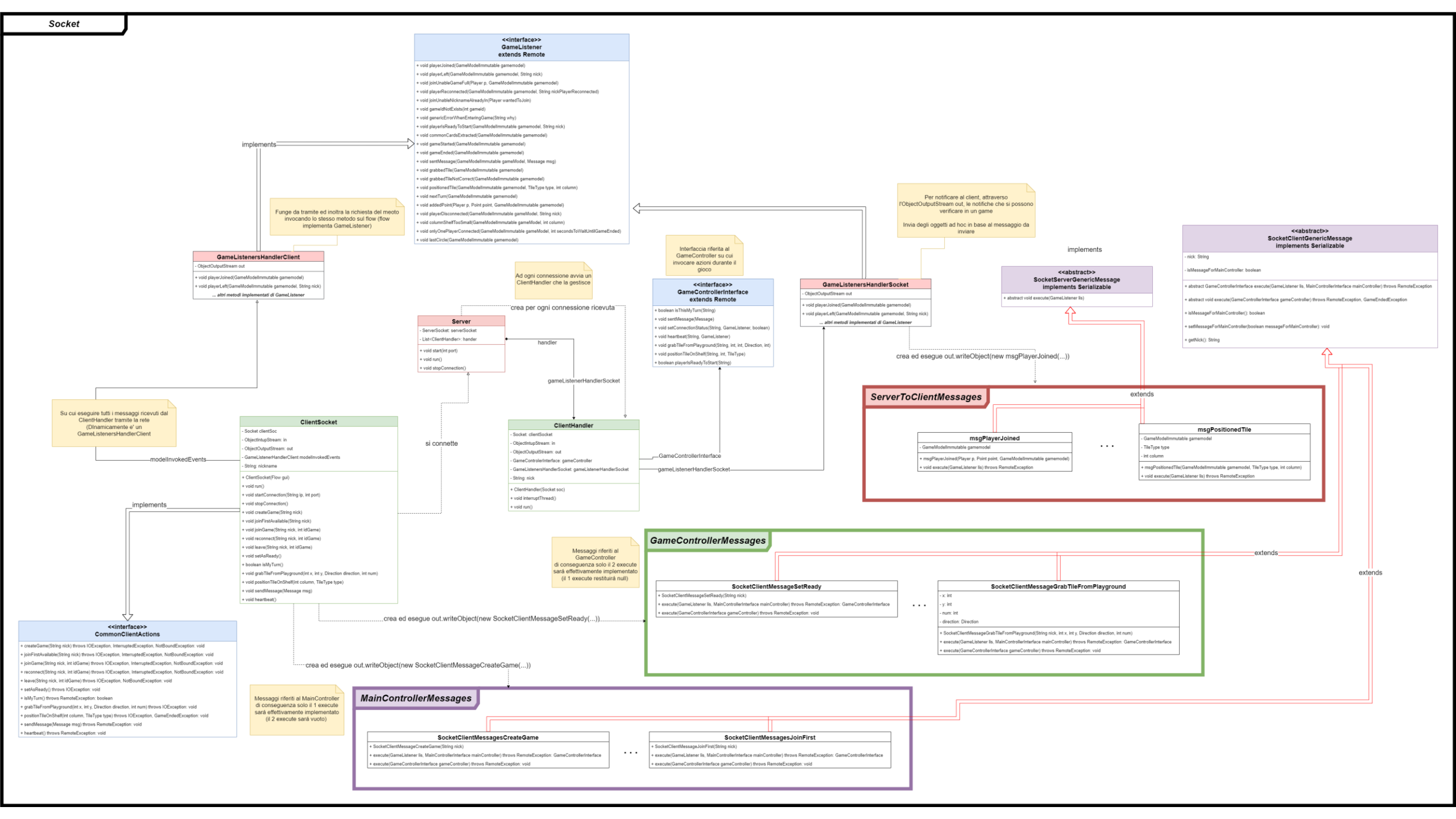

Model
Exceptions
<<Signal>>
CommonCardAlreadyInException
Questa eccezione viene lanciata in caso il gioco estragga come seconda carta comune da giocare una carta già presente in partita
<<Signal>>
GameEndedException
Questa eccezione viene lanciata in caso un giocatore provi ad unirsi ad una partita già iniziata
. . .
<<Signal>>
MaxPlayersInException
Questa eccezione viene lanciata in caso un giocatore provi ad unirsi ad una partita con al suo interno il massimo di giocatori
<<Signal>>
[Action]Exception
Il programma lancia molte eccezioni create appositamente per diversi scenari e casi d'uso, in questo diagramma UML non sembrava fondamentale citarle tutte,dato che sarebbe stato solamente un elenco di eccezioni autodescrittive
+15 eccezioni
throws
GameModel
GameModel
- leaderBoard: Map(Integer, Integer)
- players: List<Player>
- commonCards: List<CommonCards>
- gameId: Integer
- pg: Playground
- currentPlaying: Integer
- chat: Chat
- status: GameStatus
- firstFinishedPlayer: Integer
- listenersHandler: ListenersHandler
+ GameModel(): void
+ GameModel(List<Player>, List<CommonCard>, Integer, Playground): void
+ getStatus(): GameStatus
+ addListener(GameListener): void
- isTheCurrentPlayerOnline(): boolean
+ reconnectPlayer(Player): void
+ removePlayer(String): void
+ setFinishedPlayer(Integer): void
+ positionTileOnShelf(Player, int, TileType): void
+ doAllPlayersHaveGoalCard(): boolean
+ playerIsReadyToStart(Player): boolean
- popInHandTilePlayer(Player, TileType): Tile?
+ grabTileFromPlayground(Player, int, int, Direction, int): void
+ setCurrentPlaying(Integer): void
+ arePlayersReadyToStartAndEnough(): boolean
+ setAsDisconnected(String): void
+ nextTurn(): void
+ setStatus(GameStatus status): void
+ getListeners(): void
+ removeListener(GameListener): void
+sentMessage(Message): void
...Getter e Setter di attributi GameSpecific(CommonCard, GoalCard, Player, ...)
status
Attributi di gioco
Attributi di gioco immutabili
GameModelImmutable
implements Serializable
Questa classe è una differente implementazione della classe GameModel, con tutti gli attributi di tipo fClasse}fC, e i soli metodi getter, perchè questa classe è ciò che formerà l'oggetto serializzato da inviare al client, quindi qualsiasi cosa al suo interno deve essere immutabile
Card
<<Abstract>>
Card
implements Serializable
- name: String
+ getName(): String
+ setName(String): void
+ isSameType(Card): boolean
extends
Common
<<Abstract>>
CommonCard
- points: Queue<Point>
- commonType: CardCommonType
+ CommonCard(): void
+ CommonCard(CardCommonType): void
+ CommonCard(Queue<Points>, CardCommonType): Tile
+ verify(Shelf): boolean
...Getter e Setter di attributi in formato fclasse}fC, per implementare l'immutabilità del model
...Getter e Setter di tutti gli attributi
commonType
implements
extends
<<Abstract>>
CommonMethods
+ CommonMethods(CardCommonType): void
# getCopy(Shelf): Shelf
+ checkAndjacent(TileType, Shelf, int, int): int
# deleteAndjacent(TileType, Shelf, int, int): void
CommonCardFactory
+ getCommonCard(CardCommonType): CommonCard
extends
extends
extends
CommonStair
+ CommonStair(CardCommonType): void
+ verify(Shelf): boolean
CommonEight
+ CommonEight(CardCommonType): void
+ verify(Shelf): boolean
[CardCommonType]
Tutte le diverse carte obiettivo comune sono state implementate con classi organizzate in modo molto simile, con il singolo metodo verify, contenente al suo interno l'algoritmo di controllo della carta.
Eccezion fatta per metodi di appoggio al metodo verify le classi sono vuote, e quindi stato scelto di scrivere in UML 2 classi di esempio, perchè non reputato importante nella comprensione generale dell'implementazione scelta.
. . .
Tutte le altre carte comuni
Goal
CardGoal
- layoutToMatch: Shelf
- legendPoint: Map<Integer, Point>
- data: List<List<Integer>>
- goalType: CardGoalType
+ CardGoal(): void
+ CardGoal(CardGoalType): void
+ CardGoal(Shelf, CardGoalType): void
- initializeLayout(CardGoalType): void
+ verify(Shelf): PointIC
...Getter e Setter di attributi in formato fclasse}fC, per implementare l'immutabilità del model
...Getter e Setter di tutti gli attributi
goalType
implements
Enumeration
<<Enumeration>>
TileType
CAT
BOOK
ACTIVITY
FRAME
TROPHY
PLANT
NOT_USED
FINISHED_USING
USED
- backgroundClass: String
- values: List<TileType>
- rand: Random
+ TileType(String): void
+ getBackgroundClass(): String
+ getValues(): TileType
<<Enumeration>>
GameStatus
Implements Serializable
WAIT
RUNNING
LAST_CIRCLE
ENDED
<<Enumeration>>
Direction
UP
DOWN
LEFT
RIGHT
+ getDirection(String): Direction
<<Enumeration>>
CardCommonType
CommonSixGroups
CommonVertex
CommonFourGroups
CommonSquares
CommonVertical0
CommonEight
CommonSameDiagonal
CommonHorizontal0
CommonVertical1
CommonHorizontal1
CommonX
CommonStair
- backgroundClass: String
+ CardCommonType(String): void
+ getBackgroundClass(): String
<<Enumeration>>
CardGoalType
GOAL0
GOAL1
GOAL2
GOAL3
GOAL4
GOAL5
GOAL6
GOAL7
GOAL8
GOAL9
GOAL10
GOAL11
NOT_SET
- backgroundClass: String
+ CardGoalType(String): void
+ toString(): String
+ getBackgroundClass(): String
implements
Interfaces
<<Interface>>
PlayerIC
Questa interfaccia è ciò che sarà mandato al client come oggetto remoto, gli unici metodi che ha al suo interno sono quindi getter di attributi della classe Player
<<Interface>>
PointIC
Questa interfaccia è ciò che sarà mandato al client come oggetto remoto, gli unici metodi che ha al suo interno sono quindi getter di attributi della classe Point
<<Interface>>
CommonCardIC
Questa interfaccia è ciò che sarà mandato al client come oggetto remoto, gli unici metodi che ha al suo interno sono quindi getter di attributi della classe CommonCard
<<Interface>>
CardGoalIC
Questa interfaccia è ciò che sarà mandato al client come oggetto remoto, gli unici metodi che ha al suo interno sono quindi getter di attributi della classe CardGoal
<<Interface>>
CardType
Interfaccia implementata da tutte le enumerazioni di carte
<<Interface>>
ChatIC
Questa interfaccia è ciò che sarà mandato al client come oggetto remoto, gli unici metodi che ha al suo interno sono quindi getter di attributi della classe Chat
<<Interface>>
PlaygroundIC
Questa interfaccia è ciò che sarà mandato al client come oggetto remoto, gli unici metodi che ha al suo interno sono quindi getter di attributi della classe Playground
<<Interface>>
ShelfIC
Questa interfaccia è ciò che sarà mandato al client come oggetto remoto, gli unici metodi che ha al suo interno sono quindi getter di attributi della classe Shelf
<<Interface>>
TileIC
Questa interfaccia è ciò che sarà mandato al client come oggetto remoto, gli unici metodi che ha al suo interno sono quindi getter di attributi della classe Tile
implements
Player
implements Serializable
- nickname: String
- shelf: Shelf
- inHandTile: List<Tile>
- obtainedPoints: List<Point>
- readyToStart: boolean
- connected: boolean
- listeners: List<GameListener>
+ Player(String): void
+ Player(String, Shelf, CardGoal, List<Tile>, List<Point>): void
+ clearInHandTile(): void
+ addPoint(Point, GameModel): void
+ getMaxFreeSpaceInACol(): int
+ getNumOfFreeSpacesInACol(int): int
+ addListener(GameListener): void
- notify_addedPoint(Point, GameModelImmutable): void
+ removeListener(GameListener): void
...Getter e Setter di attributi in formato fclasse}fC, per implementare l'immutabilità del model
...Getter e Setter di tutti gli attributi
Point
implements Serializable
- point: Integer
- referredTo: CardType
+ Point(Integer, CardType): void
+ Point(Integer): void
...Getter e Setter di tutti gli attributi
DefaultValue
Questa classe contiene attributi utili al fine dello sviluppo e continuità del codice. Ha al suo interno, come esempio, un attributo playgroundSize, uno shelfColumn, uno shelfRow, e così via.
Serve quindi per mantenere una continuità nello sviluppo dell'applicazione. Tutti gli attributi sono di tipo primitivo, statici e finali, eccezion fatta per serverIp, che è di default inizializzato a "127.0.0.1", e viene reinizializzato allo avvio dall'applicazione, per poi non venir più modificato.
Playground
implements Serializable
- playground: Tile[ ][ ]
- bag: List<Tile>
- data: List<List<Integer>>
+ Playground(): void
+ Playground(int): void
+ getTile(int, int): Tile
+ getTile_IC(Point, GameModel): TileIC
+ getNumOfTileInTheBag(): int
+ initializePlayground(): void
+ setPlayground(): void
+ setFreeSides(): void
+ setBag(): void
+ checkBeforeGrab(int, int, Direction, int): boolean
+ allTilesHaveAllFreeSides(): boolean
+ grabTile(int, int, Direction, int): List<Tile>
...Getter e Setter di tutti gli attributi
...Metodi di testing (deprecati)
...Getter e Setter di attributi in formato fclasse}fC, per implementare l'immutabilità del model
Shelf
implements Serializable
- shelf: Tile[ ][ ]
- freeSpace: Integer
+ Shelf(): void
+ Shelf(Tile[ ][ ], Integer): void
+ setSingleTile(int, int): void
+ getSingleTile(int, int): Tile
+ position(int, TileType): void
+ isEmpty(): boolean
+ getMaxFreeSpacesInACol(): int
+ getNumofFreeSpacesInACol(int): int
...Getter e Setter di attributi in formato fclasse}fC, per implementare l'immutabilità del model
...Getter e Setter di tutti gli attributi
Tile
implements Serializable
- TYPE: TileType
- freeSide: boolean
- offset: int
+ getBackground(): String
+ toString(): String
...Getter e Setter di tutti gli attributi
type
Chat
implements Serializable
- msgs: List<Message>
+ Chat(List<Message>): void
+ Chat(): void
+ toString(): String
...Getter e Setter di tutti gli attributi
msgs
Message
implements Serializable
- text: String
- sender: PlayerIC
- time: LocalDateTime
+ Message(String, PlayerIC): void
+ Message(): void
+ toString(): String
...Getter e Setter di tutti gli attributi
MessagePrivate
- receiverPrivate: String
+ MessagePrivate(String, PlayerIC, String): void
+ whoIsReceiver(): String
extends

#2 CONTROLLER

Controller


<<interface>>
GameControllerInterface
extends Remote
interfaccia giá presentata e riportata di seguito per chiarezza
<<interface>>
MainControllerInterface
extends Remote
interfaccia giá presentata e riportata di seguito per chiarezza
GameModel
classe giá presentata e riportata di seguito per chiarezza
implements
implements
model
heartbeats
runningGames
GameController
implements Serializable, Runnable
- model: GameModel
- final random: Random
- final transient heartbeats: Map<GameListener, Heartbeat>
- reconnectionTh: Thread
+ GameController()
+ run(): void
+ addPlayer(Player p) throws PlayerAlreadyInException, MaxPlayersInException: void
+ reconnectPlayer(Player p) throws PlayerAlreadyInException, MaxPlayersInException, GameEndedException: void
+ synchronized playerIsReadyToStart(String p): boolean
- extractCommonCards(): void
- extractGoalCards(): void
- setPlaygroundLayout(): void
- extractFirstTurn(): void
+ synchronized grabTileFromPlayground(String p, int x, int y, Direction direction, int num): void
+ synchronized positionTileOnShelf(String p, int column, TileType type): void
+ synchronized isThisMyTurn(String nick) throws RemoteException: boolean
+ synchronized disconnectPlayer(String nick, GameListener lisOfClient) throws RemoteException: void
- startReconnectionTimer(): void
- stopReconnectionTimer(): void
+ synchronized heartbeat(String nick, GameListener me) throws RemoteException: void
+ synchronized sentMessage(Message msg) throws RemoteException : void
+ synchronized getGameId(): int
+ synchronized getNumOfOnlinePlayers(): int
+ synchronized leave(GameListener lis, String nick) throws RemoteException: void
- checkCommonCards(Player p): void
- checkGoalCards(): void
+ checkFinal(): void
+ addListener(GameListener l, Player p): void
+ removeListener(GameListener lis, Player p): void
getter e setter utili
MainController
implements Serializable
- static instance: MainController
- runningGames: List<GameController>
- MainController()
+ synchronized static getInstance(): MainController
+ synchronized createGame(GameListener lis, String nick) throws RemoteException: GameControllerInterface
+ synchronized joinFirstAvailableGame(GameListener lis, String nick) throws RemoteException: GameControllerInterface
+ synchronized joinGame(GameListener lis, String nick, int idGame) throws RemoteException: GameControllerInterface
+ synchronized reconnect(GameListener lis, String nick, int idGame) throws RemoteException: GameControllerInterface
+ synchronized leaveGame(GameListener lis, String nick, int idGame) throws RemoteException: GameControllerInterface
+ deleteGame(int idGame): void
- printRunningGames(): void
HeartBeat
- beat: Long
- nick: String
+ GameModel(Long, String): void
+ getBeat(): Long
+ getNick(): String
MainController realizza Pattern Singleton
Gestisce un insieme di partite riferendosi ai GameController
GameController gestisce il ciclo di vita di una partita controllando il GameModel
Si occupa di estrarre le common e goal Card di impostare il playground e di controllare che gli obiettivi finali, common card e goal card

#3.1 SOCKET


Socket
<<interface>>
GameListener
extends Remote
+ void playerJoined(GameModelImmutable gamemodel)
+ void playerLeft(GameModelImmutable gamemodel, String nick)
+ void joinUnableGameFull(Player p, GameModelImmutable gamemodel)
+ void playerReconnected(GameModelImmutable gamemodel, String nickPlayerReconnected)
+ void joinUnableNicknameAlreadyIn(Player wantedToJoin)
+ void gameIdNotExists(int gameId)
+ void genericErrorWhenEnteringGame(String why)
+ void playerIsReadyToStart(GameModelImmutable gamemodel, String nick)
+ void commonCardsExtracted(GameModelImmutable gamemodel)
+ void gameStarted(GameModelImmutable gamemodel)
+ void gameEnded(GameModelImmutable gamemodel)
+ void sentMessage(GameModelImmutable gameModel, Message msg)
+ void grabbedTile(GameModelImmutable gamemodel)
+ void grabbedTileNotCorrect(GameModelImmutable gamemodel)
+ void positionedTile(GameModelImmutable gamemodel, TileType type, int column)
+ void nextTurn(GameModelImmutable gamemodel)
+ void addedPoint(Player p, Point point, GameModelImmutable gamemodel)
+ void playerDisconnected(GameModelImmutable gameModel, String nick)
+ void columnShelfTooSmall(GameModelImmutable gameModel, int column)
+ void onlyOnePlayerConnected(GameModelImmutable gameModel, int secondsToWaitUntilGameEnded)
+ void lastCircle(GameModelImmutable gamemodel)
implements
Funge da tramite ed inoltra la richiesta del meoto invocando lo stesso metodo sul flow (flow implementa GameListener)
GameListenersHandlerClient
- ObjectOutputStream out
+ void playerJoined(GameModelImmutable gamemodel)
+ void playerLeft(GameModelImmutable gamemodel, String nick)
... altri metodi implementati di GameListener
Su cui eseguire tutti i messaggi ricevuti dal ClientHandler tramite la rete (DInamicamente e' un GameListenersHandlerClient
modelInvokedEvents
Ad ogni connessione avvia un ClientHandler che la gestisce
crea per ogni connessione ricevuta
Server
- ServerSocket: serverSocket
- List<ClientHandler>: hander
+ void start(int port)
+ void run()
+ void stopConnection()
handler
gameListenerHandlerSocket
si connette
ClientSocket
- Socket clientSoc
- ObjectIntupStream: in
- ObjectOutputStream: out
- GameListenerHandlerClient modelInvokedEvents
- String: nickname
+ ClientSocket(Flow gui)
+ void run()
+ void startConnection(String ip, int port)
+ void stopConnection()
+ void createGame(String nick)
+ void joinFirstAvailable(String nick)
+ void joinGame(String nick, int idGame)
+ void reconnect(String nick, int idGame)
+ void leave(String nick, int idGame)
+ void setAsReady()
+ boolean isMyTurn()
+ void grabTileFromPlayground(int x, int y, Direction direction, int num)
+ void positionTileOnShelf(int column, TileType type)
+ void sendMessage(Message msg)
+ void heartbeat()
implements
<<interface>>
CommonClientActions
+ createGame(String nick) throws IOException, InterruptedException, NotBoundException: void
+ joinFirstAvailable(String nick) throws IOException, InterruptedException, NotBoundException: void
+ joinGame(String nick, int idGame) throws IOException, InterruptedException, NotBoundException: void
+ reconnect(String nick, int idGame) throws IOException, InterruptedException, NotBoundException: void
+ leave(String nick, int idGame) throws IOException, NotBoundException: void
+ setAsReady() throws IOException: void
+ isMyTurn() throws RemoteException: boolean
+ grabTileFromPlayground(int x, int y, Direction direction, int num) throws IOException: void
+ positionTileOnShelf(int column, TileType type) throws IOException, GameEndedException: void
+ sendMessage(Message msg) throws RemoteException: void
+ heartbeat() throws RemoteException: void
Interfaccia riferita al GameController su cui invocare azioni durante il gioco
<<interface>>
GameControllerInterface
extends Remote
+ boolean isThisMyTurn(String)
+ void sentMessage(Message)
+ void setConnectionStatus(String, GameListener, boolean)
+ void heartbeat(String, GameListener)
+ void grabTileFromPlayground(String, int, int, Direction, int)
+ void positionTileOnShelf(String, int, TileType)
+ boolean playerIsReadyToStart(String)
GameControllerInterface
gameListenerHandlerSocket
ClientHandler
- Socket: clientSocket
- ObjectIntupStream: in
- ObjectOutputStream: out
- GameControlerInterface: gameController
- GameListenersHandlerSocket: gameListenerHandlerSocket
- String: nick
+ ClientHandler(Socket soc)
+ void interruptThread()
+ void run()
Per notificare al client, attraverso l'ObjectOutputStream out, le notifiche che si possono verificare in un game
Invia degli oggetti ad hoc in base al messaggio da inviare
GameListenersHandlerSocket
- ObjectOutputStream out
+ void playerJoined(GameModelImmutable gamemodel)
+ void playerLeft(GameModelImmutable gamemodel, String nick)
... altri metodi implementati di GameListener
crea ed esegue out.writeObject(new msgPlayerJoined(...))
implements
<<abstract>>
SocketServerGenericMessage
implements Serializable
+ abstract void execute(GameListener lis)
<<abstract>>
SocketClientGenericMessage
implements Serializable
- nick: String
- isMessageForMainController: boolean
+ abstract GameControllerInterface execute(GameListener lis, MainControllerInterface mainController) throws RemoteException
+ abstract void execute(GameControllerInterface gameController) throws RemoteException, GameEndedException
+ isMessageForMainController(): boolean
+ setMessageForMainController(boolean messageForMainController): void
+ getNick(): String
ServerToClientMessages
extends
msgPlayerJoined
- GameModelImmutable gamemodel
+ msgPlayerJoined(Player p, Point point, GameModelImmutable gamemodel)
+ void execute(GameListener lis) throws RemoteException
msgPositionedTile
- GameModelImmutable gamemodel
- TileType type
- int column
+ msgPositionedTile(GameModelImmutable gamemodel, TileType type, int column)
+ void execute(GameListener lis) throws RemoteException
Messaggi riferiti al GameController di conseguenza solo il 2 execute sarà effettivamente implementato (il 1 execute restituirà null)
GameControllerMessages
extends
SocketClientMessageSetReady
+ SocketClientMessageSetReady(String nick)
+ execute(GameListener lis, MainControllerInterface mainController) throws RemoteException: GameControllerInterface
+ execute(GameControllerInterface gameController) throws RemoteException: void
SocketClientMessageGrabTileFromPlayground
- x: int
- y: int
- num: int
- direction: Direction
+ SocketClientMessageGrabTileFromPlayground(String nick, int x, int y, Direction direction, int num)
+ execute(GameListener lis, MainControllerInterface mainController) throws RemoteException: GameControllerInterface
+ execute(GameControllerInterface gameController) throws RemoteException: void
crea ed esegue out.writeObject(new SocketClientMessageSetReady(...))
crea ed esegue out.writeObject(new SocketClientMessageCreateGame(...))
Messaggi riferiti al MainController di conseguenza solo il 1 execute sará effettivamente implementato (il 2 execute sará vuoto)
MainControllerMessages
extends
SocketClientMessagesCreateGame
+ SocketClientMessageCreateGame(String nick)
+ execute(GameListener lis, MainControllerInterface mainController) throws RemoteException: GameControllerInterface
+ execute(GameControllerInterface gameController) throws RemoteException: void
SocketClientMessagesJoinFirst
+ SocketClientMessageJoinFirst(String nick)
+ execute(GameListener lis, MainControllerInterface mainController) throws RemoteException: GameControllerInterface
+ execute(GameControllerInterface gameController) throws RemoteException: void

#3.2 RMI

RMI

RemoteInterfaces

<<interface>>
MainControllerInterface
extends Remote

+ createGame(GameListener lis, String nick) throws RemoteException: GameControllerInterface
+ joinFirstAvailableGame(GameListener lis, String nick) throws RemoteException: GameControllerInterface
+ joinGame(GameListener lis, String nick, int idGame) throws RemoteException: GameControllerInterface
+ reconnect(GameListener lis, String nick, int idGame) throws RemoteException: GameControllerInterface
+ leaveGame(GameListener lis, String nick, int idGame) throws RemoteException: GameControllerInterface

<<interface>>
GameControllerInterface
extends Remote

+ playerIsReadyToStart(String p) throws RemoteException: boolean
+ grabTileFromPlayground(String p, int x, int y, Direction direction, int num) throws RemoteException: void
+ positionTileOnShelf(String p, int column, TileType type) throws RemoteException, GameEndedException: voic
+ isThisMyTurn(String nick) throws RemoteException: boolean
+ disconnectPlayer(String nick, GameListener lisOfClient) throws RemoteException: void
+ heartbeat(String nick, GameListener me) throws RemoteException: void
+ sentMessage(Message msg) throws RemoteException: void
+ getGameId() throws RemoteException: int
+ getNumOnlinePlayers() throws RemoteException: int
+ leave(GameListener lis, String nick) throws RemoteException: void

<<interface>>
CommonClientActions

+ createGame(String nick) throws IOException, InterruptedException, NotBoundException: void
+ joinFirstAvailable(String nick) throws IOException, InterruptedException, NotBoundException: void
+ joinGame(String nick, int idGame) throws IOException, InterruptedException, NotBoundException: void
+ reconnect(String nick, int idGame) throws IOException, InterruptedException, NotBoundException: void
+ leave(String nick, int idGame) throws IOException, NotBoundException: void
+ setAsReady() throws IOException: void
+ isMyTurn() throws RemoteException: boolean
+ grabTileFromPlayground(int x, int y, Direction direction, int num) throws IOException: void
+ positionTileOnShelf(int column, TileType type) throws IOException, GameEndedException: void
+ sendMessage(Message msg) throws RemoteException: void
+ heartbeat() throws RemoteException: void

implements

MainControllerInterface e GameControllerInterface
Oggetti remoti esportati dal server su cui il client puó invocare metodi per effettuare azioni

gameController

implements

requests

RMIServer
extends UnicastRemoteObject

- final mainController: MainControllerInterface
- static serverObject: RMIServer
- static registry: Registry

+ static bind(): RMIServer
+ synchronized static getInstance(): RMIServer
+ synchronized static getRegistry() throws RemoteException: Registry
+ RMIServer() throws RemoteException
+ createGame(GameListener lis, String nick) throws RemoteException: GameControllerInterface
+ joinFirstAvailableGame(GameListener lis, String nick) throws RemoteException: GameControllerInterface
+ joinGame(GameListener lis, String nick, int idGame) throws RemoteException: GameControllerInterface
+ reconnect(GameListener lis, String nick, int idGame) throws RemoteException: GameControllerInterface
+ leaveGame(GameListener lis, String nick, int idGame) throws RemoteException: GameControllerInterface
+ static unbind(): RMIServer

RMIClient
implements Runnable

- static requests: MainControllerInterface
- gameController: GameControllerInterface
- static modelInvokedEvents: GameListener
- nickname: String
- final gameListenersHandler: GameListenersHandlerClient
- registry: Registry
- flow: Flow

+ RMIClient(Flow flow)
+ connect(): void
+ run(): void
+ createGame(String nick) throws RemoteException, NotBoundException: void
+ joinFirstAvailable(String nick) throws RemoteException, NotBoundException: void
+ joinGame(String nick, int idGame) throws RemoteException, NotBoundException: void
+ reconnect(String nick, int idGame) throws RemoteException, NotBoundException: void
+ leave(String nick, int idGame) throws IOException, NotBoundException: void
+ sendMessage(Message msg) throws RemoteException: void
+ setAsReady() throws RemoteException: void
+ isMyTurn() throws RemoteException: boolean
+ grabTileFromPlayground(int x, int y, Direction direction, int num) throws RemoteException: void
+ positionTileOnShelf(int column, TileType type) throws GameEndedException, RemoteException: void
+ heartbeat() throws RemoteException: void

modelInvokedEvents

Restituisce oggetti remoti di tipo
GameControllerInterface

Restituisce anche MainControllerInterfaced dopo aver fatto il lookup del registry

Oggetto che il client rende remoto su cui il server andrá ad invocare i metodi di notifica degli eventi

<<interface>>
GameListener
extends Remote

interfaccia giá presentata e riportata di seguito per chiarezza

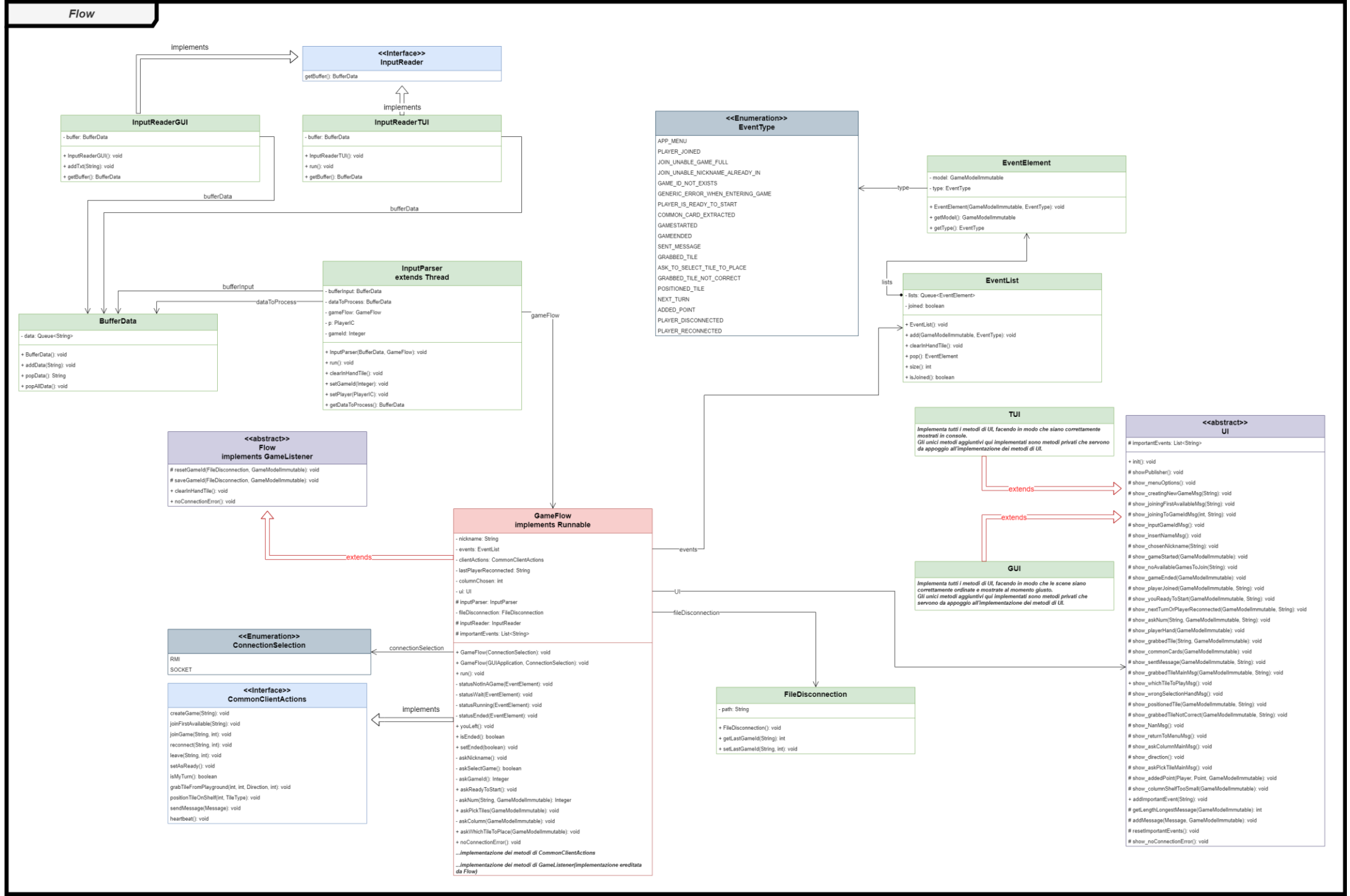
GameListenersHandlerClient
implements GameListener

classe giá presentata e riportata di seguito per chiarezza

modelInvokedEvents =(GameListener)
UnicastRemoteObject.exportObject(gameListenersHandler, 0);

Come giá esposto nell'UML, GameListenersHandlerClient invoca su flow gli eventi di notifica
(es. flow.playerJoined(...))

#4 FLOW


Flow
Implements
<<Interface>>
InputReader
getBuffer(): BufferData
implements
InputReaderGUI
- buffer: BufferData
+ InputReaderGUI(): void
+ addTxt(String): void
+ getBuffer(): BufferData
InputReaderTUI
- buffer: BufferData
+ InputReaderTUI(): void
+ run(): void
+ getBuffer(): BufferData
bufferData
bufferData
<<Enumeration>>
EventType
APP_MENU
PLAYER_JOINED
JOIN_UNABLE_GAME_FULL
JOIN_UNABLE_NICKNAME_ALREADY_IN
GAME_ID_NOT_EXISTS
GENERIC_ERROR_WHEN_ENTERING_GAME
PLAYER_IS_READY_TO_START
COMMON_CARD_EXTRACTED
GAMESTARTED
GAMEENDED
SENT_MESSAGE
GRABBED_TILE
ASK_TO_SELECT_TILE_TO_PLACE
GRABBED_TILE_NOT_CORRECT
POSITIONED_TILE
NEXT_TURN
ADDED_POINT
PLAYER_DISCONNECTED
PLAYER_RECONNECTED
type
EventElement
- model: GameModelImmutable
- type: EventType
+ EventElement(GameModelImmutable, EventType): void
+ getModel(): GameModelImmutable
+ getType(): EventType
lists
EventList
- lists: Queue<EventElement>
- joined: boolean
+ EventList(): void
+ add(GameModelImmutable, EventType): void
+ clearInHandTile(): void
+ pop(): EventElement
+ size(): int
+ isJoined(): boolean
InputParser
extends Thread
- bufferInput: BufferData
- dataToProcess: BufferData
- gameFlow: GameFlow
- p: PlayerIC
- gameId: Integer
+ InputParser(BufferData, GameFlow): void
+ run(): void
+ clearInHandTile(): void
+ setGameId(Integer): void
+ setPlayer(PlayerIC): void
+ getDataToProcess(): BufferData
bufferInput
dataToProcess
gameFlow
BufferData
- data: Queue<String>
+ BufferData(): void
+ addData(String): void
+ popData(): String
+ popAllData(): void
<<abstract>>
Flow
implements GameListener
resetGameId(FileDisconnection, GameModelImmutable): void
saveGameId(FileDisconnection, GameModelImmutable): void
+ clearInHandTile(): void
+ noConnectionError(): void
extends
GameFlow
implements Runnable
- nickname: String
- events: EventList
- clientActions: CommonClientActions
- lastPlayerReconnected: String
- columnChosen: int
- ui: UI
inputParser: InputParser
- fileDisconnection: FileDisconnection
inputReader: InputReader
importantEvents: List<String>
+ GameFlow(ConnectionSelection): void
+ GameFlow(GUIApplication, ConnectionSelection): void
+ run(): void
- statusNotInAGame(EventElement): void
- statusWait(EventElement): void
- statusRunning(EventElement): void
- statusEnded(EventElement): void
+ youLeft(): void
+ isEnded(): boolean
+ setEnded(boolean): void
- askNickname(): void
- askSelectGame(): boolean
- askGameId(): Integer
+ askReadyToStart(): void
- askNum(String, GameModelImmutable): Integer
+ askPickTiles(GameModelImmutable): void
- askColumn(GameModelImmutable): void
+ askWhichTileToPlace(GameModelImmutable): void
+ noConnectionError(): void
...implementazione dei metodi di CommonClientActions
...implementazione dei metodi di GameListener(implementazione ereditata da Flow)
events
UI
fileDisconnection
TUI
Implementa tutti i metodi di UI, facendo in modo che siano correttamente mostrati in console.
Gli unici metodi aggiuntivi qui implementati sono metodi privati che servono da appoggio all'implementazione dei metodi di UI.
extends
GUI
Implementa tutti i metodi di UI, facendo in modo che le scene siano correttamente ordinate e mostrate al momento giusto.
Gli unici metodi aggiuntivi qui implementati sono metodi privati che servono da appoggio all'implementazione dei metodi di UI.
extends
<<abstract>>
UI
importantEvents: List<String>
+ init(): void
# showPublisher(): void
# show_menuOptions(): void
# show_creatingNewGameMsg(String): void
# show_joiningFirstAvailableMsg(String): void
# show_joiningToGameIdMsg(int, String): void
# show_inputGameIdMsg(): void
# show_insertNameMsg(): void
# show_chosenNickname(String): void
# show_gameStarted(GameModelImmutable): void
# show_noAvailableGamesToJoin(String): void
# show_gameEnded(GameModelImmutable): void
# show_playerJoined(GameModelImmutable, String): void
# show_youReadyToStart(GameModelImmutable, String): void
# show_nextTurnOrPlayerReconnected(GameModelImmutable, String): void
# show_askNum(String, GameModelImmutable, String): void
# show_playerHand(GameModelImmutable): void
# show_grabbedTile(String, GameModelImmutable): void
# show_commonCards(GameModelImmutable): void
# show_sentMessage(GameModelImmutable, String): void
# show_grabbedTileMainMsg(GameModelImmutable, String): void
+ show_whichTileToPlayMsg(): void
# show_wrongSelectionHandMsg(): void
# show_positionedTile(GameModelImmutable, String): void
# show_grabbedTileNotCorrect(GameModelImmutable, String): void
# show_NanMsg(): void
# show_returnToMenuMsg(): void
# show_askColumnMainMsg(): void
# show_direction(): void
# show_askPickTileMainMsg(): void
# show_addedPoint(Player, Point, GameModelImmutable): void
# show_columnShelfTooSmall(GameModelImmutable): void
+ addImportantEvent(String): void
# getLengthLongestMessage(GameModelImmutable): int
# addMessage(Message, GameModelImmutable): void
# resetImportantEvents(): void
# show_noConnectionError(): void
<<Enumeration>>
ConnectionSelection
RMI
SOCKET
connectionSelection
<<Interface>>
CommonClientActions
createGame(String): void
joinFirstAvailable(String): void
joinGame(String, int): void
reconnect(String, int): void
leave(String, int): void
setAsReady(): void
isMyTurn(): boolean
grabTileFromPlayground(int, int, Direction, int): void
positionTileOnShelf(int, TileType): void
sendMessage(Message): void
heartbeat(): void
implements
FileDisconnection
- path: String
+ FileDisconnection(): void
+ getLastGameId(String): int
+ setLastGameId(String, int): void