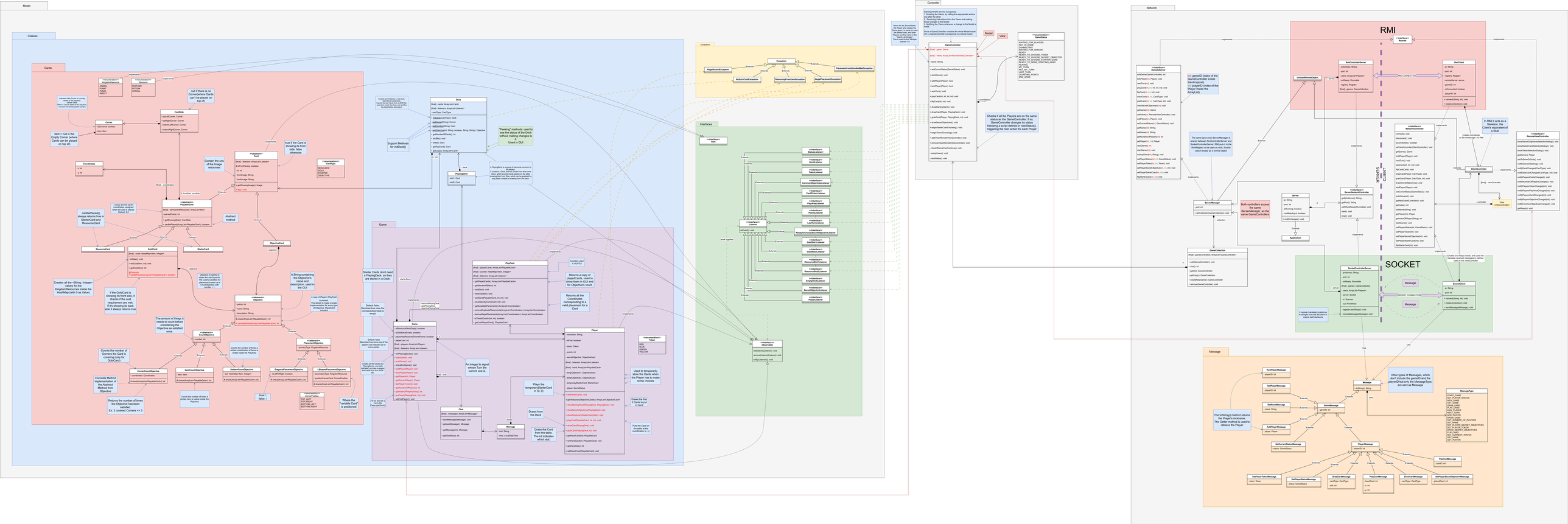
Model
Classes
Cards
Corner
Coordinates
Item
Abstract Method
ResourceCard
GoldCard
StarterCard
ObjectiveCard
Objective
CountObjective
CornerCountObjective
ItemCountObjective
BottomCountObjective
DiagonalPlacementObjective
LShapePlacementObjective
PlacementObjective
Game
Deck
PlayingDeck
PlayField
Player
Chat
Message
Controller
GameController
Exceptions
Exception
IllegalActionException
NoSuchCardException
RemovingFrameException
IllegalPlacementException
PlacementConditionNotMetException
Interfaces
Item
Listener
Network
RMI
SOCKET
RemoteServer
RmiController
RmiClient
SocketController
SocketClient
NetworkController
NetworkManager
Server
Application
GameCollection
Message
KickPlayerMessage
GetPlayerMessage
SetNameMessage
AddPlayerMessage
SetCurrentStatusMessage
GameMessage
PlayerMessage
SetPlayerTokenMessage
SetPlayerStatusMessage
GoldCardMessage
PlayCardMessage
DrawCardMessage
SetPlayerSecretObjectiveMessage
PgCardMessage
MessageType

Model

Classes

Cards

<<enumeration>>
KingdomResource
ANIMAL
PLANT
FUNGI
INSECT

<<enumeration>>
Resource
FEATHER
POTION
SCROLL

implements

implements

null if there is no Corner(where Cards can't be placed on top of)

Indicates if this Corner is currently shown in the playArea.
Default: false.
When a Card is added to the playArea it covers the nearby Cards' Corners

Corner
- isCovered: boolean
- item: Item

4

CardSide
- topLeftCorner: Corner
- topRightCorner: Corner
- bottomLeftCorner: Corner
- bottomRightCorner: Corner

item = null is the Empty Corner (where Cards can be placed on top of)

implements

true if the Card is showing its front side, false otherwise

Contain the urls of the Image resources

<<abstract>>
Card
[final] - listeners: ArrayList<Listener>
isFrontFacing: boolean
- id: int
- frontImage: String
- backImage: String
+ getShowingImage(): Image
+ flip(): void

<<enumeration>>
CardType
RESOURCE
GOLD
STARTER
OBJECTIVE

Coordinates
- x: int
- y: int

[final] - coordinates

2: frontSide, backSide

Extends

x and y are the card's coordinates, assigned when the card is placed. Default: 0,0

canBePlaced() always returns true in StarterCard and ResourceCard

Abstract method

<<abstract>>
PlayableCard
[final] - permanentResources: ArrayList<Item>
- earnedPoints: int
+ getShowingSide(): CardSide
+ canBePlaced(ArrayList<PlayableCard>): boolean

Extends

Extends

Extends

ResourceCard

GoldCard
[final] - costs: HaspMap<Item, Integer>
- initMap(): void
+ setCost(Item, int): void
+ getCost(Item): int
@Override
+ canBePlaced(ArrayList<PlayableCard>): boolean

StarterCard

Creates all the <String, Integer> values for the KingdomResources inside the HashMap (with 0 as Value)

if the GoldCard is showing its front side, it checks if the cost requirement are met.
If it's showing its back side it always returns true

- objective

Objective to satisfy to obtain the card's points.
The Card's condition for placement is seen as a CountObjective with number = 1.

Extends

ObjectiveCard

- objective

A String containing the Objective's name and description, used in the GUI

<<abstract>>
Objective
- points: int
- name: String
- description: String
check(ArrayList<PlayableCard>): int
+ calculatePoints(ArrayList<PlayableCard>): int

A copy of Player's PlayField is passed.
This allows to make a single implementation for every type of Objective. Placement included

The amount of things it needs to count before considering the Objective as satisfied once.

Extends

Extends

<<abstract>>
CountObjective
- number: int

<<abstract>>
PlacementObjective
- primaryType: KingdomResource

Extends

Extends

Extends

Extends

Extends

Counts the number of Corners the Card is covering (only for GoldCard)

Counts the number of times a certain combination of Items is visible inside the PlayArea

CornerCountObjective
- coordinates: Coordinates
check(ArrayList<PlayableCard>): int

ItemCountObjective
- item: Item
check(ArrayList<PlayableCard>): int

SetItemCountObjective
- set: HashMap<Item, Integer>
check(ArrayList<PlayableCard>): int

DiagonalPlacementObjective
- isLeftToRight: boolean
check(ArrayList<PlayableCard>): int

LShapedPlacementObjective
- secondaryType: KingdomResource
- positionCornerCard: CornerPosition
check(ArrayList<PlayableCard>): int

Concrete Method implementation of the Abstract Method from Objective

Returns the number of times the Objective has been satisfied.
Ex: 3 covered Corners => 3

Counts the number of times a certain Item is visible inside the PlayArea

true: \
false: /

<<enumeration>>
CornerPosition
TOP_LEFT
TOP_RIGHT
BOTTOM_LEFT
BOTTOM_RIGHT

Where the "variable Card" is positioned

Creates and initializes a new Deck. Used like a Constructor.
Imports the real Cards from a JSON file, puts them inside the Deck, and shuffles the Deck before returning it.

Deck
[final] - cards: ArrayList<Card>
[final] - listeners: ArrayList<Listener>
- cardType: CardType
+ initDeck(CardType): Deck
- getCorner(String): Corner
- getObjective(String): Item
- initObjectives(int, String, boolean, String, String): Objective
+ getNumberOfCards(): int
+ shuffle(): void
+ draw(): Card
+ getTopCard(): Card
+ getCopy(): ArrayList<Card>

Support Methods for initDeck()

"Peeking" methods: used to see the status of the Deck without making changes to it.
Used in GUI

0..102

- deck

PlayingDeck
- slot1: Card
- slot2: Card

A PlayingDeck is a group of elements common to all players.
It contains a Deck and two Cards from that same Deck, which are the Cards placed on the table showing their front Side, which can be grabbed by any player instead of drawing from the Deck.

Game

Starter Cards don't need a PlayingDeck, so they are stored in a Deck.

- starterDeck

Default: false.
Becomes true once the corresponding Deck is empty

implements

- resourcePlayingDeck
- goldPlayingDeck
- objectivePlayingDeck

Default: false.
Becomes true once one of the players has reached 20 or more points.

Creates all the Decks and PlayingDecks, and calls initDeck() on them to import the Cards from the JSON file.

Fill the slot with a Card after Player.grabCard()

Game
- isResourceDeckEmpty: boolean
- isGoldDeckEmpty: boolean
- playerHasReachedTwentyPoints: boolean
- playerTurn: int
[final] - players: ArrayList<Player>
[final] - listeners: ArrayList<Listener>
+ initPlayingDecks(): void
+ startGame(): void
+ endGame(): void
- checkEndGame(): void
+ addPlayer(Player): void
+ kickPlayer(Player): void
+ getPlayer(int): Player
+ getCurrentPlayer(): Player
+ setPlayerTurn(int): void
+ getNumberOfPlayers(): int
+ getIndexOfPlayer(String): int
+ putDown(PlayingDeck, int): void
- setFirstPlayer(): void

2..4

An integer to signal whose Turn the current one is.

[final] - chat

PlayField
[final] - playedCards: ArrayList<PlayableCard>
[final] - counter: HashMap<Item, Integer>
[final] - listeners: ArrayList<Listener>
+ getPlayedCards(): ArrayList<PlayableCards>
+ getNumberOf(Item): int
+ add(Item): void
+ remove(Item): void
+ addCard(PlayableCard, int, int): void
- coverNearbyCorners(int, int): void
+ getAvailablePlacements(): ArrayList<Coordinates>
- removeDuplicatePlacements(ArrayList<Coordinates>) ArrayList<Coordinates>
- removeIllegalPlacements(ArrayList<Coordinates>): ArrayList<Coordinates>
- isThereACardIn(int, int): boolean
+ getLastPlayedCard(): PlayableCard

implements

Counters used in GUITUI

Returns a copy of playedCards, used to show them in GUI and for Objective's count

Returns all the Coordinates corresponding to a valid placement for a Card

Player
- nickname: String
- isFirst: boolean
- token: Token
- points: int
- secretObjective: ObjectiveCard
[final] - listeners: ArrayList<Listener>
[final] - hand: ArrayList<PlayableCard>
- tempObjective1: ObjectiveCard
- tempObjective2: ObjectiveCard
- temporaryStarterCard: StarterCard
- status: GameStatus
+ setStarterCard(): void
+ getTemporaryObjectiveCards(): ArrayList<ObjectiveCard>
+ drawStartingHand(PlayingDeck, PlayingDeck): void
+ drawSecretObjectives(PlayingDeck): void
+ drawTemporaryStarterCard(Deck): void
+ playCard(PlayableCard, int, int): void
+ drawCard(PlayingDeck): void
+ grabCard(PlayingDeck,int): void
+ getHandCard(int): PlayableCard
+ setHandCard(int, PlayableCard): void
+ getHandSize(): int
+ addHandCard(PlayableCard): void

implements

<<enumeration>>
Token
RED
BLUE
GREEN
YELLOW

Used to temporarily store the Cards when the Player has to make some choices

Plays the temporaryStarterCard in (0, 0)

Draws from the Deck

Grabs the Card from the table. The int indicates which slot.

Draws the first 3 Cards to put in hand

Puts the Card on the table at the coordinates (x, y)

Chat
[final] - messages: ArrayList<Message>
+ sendMessage(Message): void
+ getLastMessage(): Message
+ getMessage(int): Message
+ getChatSize(): int

- sender

Message
- text: String
- time: LocalDateTime

Exceptions

Exception

Extends

IllegalActionException

NoSuchCardException

RemovingFromZeroException

IllegalPlacementException

PlacementConditionNotMetException

Interfaces

<<interface>>
Item

<<interface>>
Listener
onEvent(): void

Extends

<<interface>>
StatusListener

<<interface>>
HandListener

<<interface>>
TokenListener

<<interface>>
CommonObjectivesListener

<<interface>>
CardPickerListener

<<interface>>
PlayAreaListener

<<interface>>
PointsListener

<<interface>>
LastTurnListener

<<interface>>
ReadyToChooseSecretObjectiveListener

<<interface>>
GoldSlot1Listener

<<interface>>
GoldSlot2Listener

<<interface>>
ResourceSlot1Listener

<<interface>>
ResourceSlot2Listener

<<interface>>
EmptyDeckListener

<<interface>>
SecretObjectiveListener

<<interface>>
PlayerListener

work together

<<interface>>
Observable
setListener(Listener): void
removeListener(Listener): void
notifyListeners(): void

Controller

GameController serves 3 purposes:
1. Scripting the Game, by calling the appropriate actions one after the other.
2. Receiving instructions from the Views and making those changes in the Model.
3. Notifying the Views whenever a change to the Model is made.

Since a GameController contains the whole Model inside of it, a GameController corresponds to a whole match.

Name for the Game/Match: the Player who creates the Game gives it a name (or uses the default one), and other Players see that name in the "Game List Screen". This is used for the "Multiple Games" FA

Model

View

GameController

1

[final] - game: Game

[final] - views: ArrayList<RemoteViewController>

- name: String

+ setCurrentStatus(GameStatus): void

+ startGame(): void

+ addPlayer(Player): bool

+ kickPlayer(Player): bool

+ nextTurn(): void

+ playCard(int, int, int, int): void

+ flipCard(int, int): void

+ drawStartingHand(): void

+ drawCard(Player, PlayingDeck): void

+ grabCard(Player, PlayingDeck, int): void

+ drawSecretObjectives(): void

+ beginStarterCardChoosing(): void

+ beginTokenChoosing(): void

+ addView(RemoteViewController): void

+ removeView(RemoteViewController): void

- checkIfGameCanContinue(): void

- setupViews(): void

- nextStatus(): void

- currentStatus

<<enumeration>>
GameStatus

WAITING_FOR_PLAYERS
NOT_IN_GAME
CONNECTING
WAITING_FOR_SERVER
READY
READY_TO_CHOOSE_TOKEN
READY_TO_CHOOSE_SECRET_OBJECTIVE
READY_TO_CHOOSE_STARTER_CARD
READY_TO_DRAW_STARTING_HAND
PLAYING
MY_TURN
NOT_MY_TURN
LAST_TURN
COUNTING_POINTS
END_GAME

Checks if all the Players are on the same status as the GameController: if so, GameController changes its status following a script defined in nextStatus(), triggering the next action for each Player

Network

RMI

SERVER

CLIENT

SOCKET

<<interface>>
RemoteServer
addGame(GameController): int
kickPlayer(int, Player): void
nextTurn(int): void
playCard(int, int, int, int, int): void
flipCard(int, int, int): void
drawCard(int, int, CardType): void
grabCard(int, int, CardType, int): void
drawSecretObjectives(int): void
getGame(int): Game
addView(int, RemoteViewController): void
addPlayer(int, Player): void
setCurrentStatus(int, GameStatus): void
getName(int): String
setName(int): String
getNumberOfPlayers(int): int
getPlayer(int, int): Player
newGame(): int
startGame(int): void
lookupClient(int, String): void
setPlayerStatus(int, int, GameStatus): void
setPlayerToken(int, int, Token): void
setPlayerSecretObjective(int, int, int): void
setPlayerStarterCard(int, int): void
flipStarterCard(int, int): void

implements

int: gameID (index of the GameController inside the ArrayList)
int: playerID (index of the Player inside the ArrayList)

The same (and only) ServerManager is shared between RmiControllerServer and SocketControllerServer: RMI puts it in the RmiRegistry to be used as stub, Socket uses it locally as a normal object

Extends

implements

<<interface>>
Remote

implements

UnicastRemoteObject

- server

RmiControllerServer
- ipAddress: String
- port: int
- users: ArrayList<Players>
- onReady: Runnable
- registry: Registry
[final] - games: GameCollection

RMI CONNECTION

RmiClient
- ip: String
- port: Int
- registry: Registry
- remoteServer: server
- gameID: int
- isConnected: boolean
- playerID: int
+ connect(String, Int): void
+ closeConnection(): void

implements

<<interface>>
NetworkController
connect(): void
disconnect(): void
isConnected(): boolean
setViewController(ClientController): void
getGame(): Game
kickPlayer(Player): void
nextTurn(): void
playCard(int, int, int): void
flipCard(Card): void
drawCard(Player, CardType): void
grabCard(Player, CardType, int): void
drawSecretObjectives(): void
addPlayer(Player): void
setCurrentStatus(GameStatus): void
pickGame(int): void
getNewGameController(): void
getIndex(): int
setName(String): void
getPlayer(int): Player
getIndexOfPlayer(String): int
startGame(): void
setPlayersStatus(int, GameStatus): void
setPlayerToken(int): void
setPlayerSecretObjective(int): void
setPlayerStarterCard(int): void
flipStarterCard(int): void

Creates and sends
to ServerManager via RMI

In RMI it acts as a Skeleton, the Client's equivalent of a Stub

Extends

ClientController

implements

<<interface>>
RemoteViewController
showSecretObjectivesSelectionDialog(): void
showStarterCardSelectionDialog(): void
showTokenSelectionDialog(): void
getOwner(): Player
askToDrawOrGrab(): void
notifyGameIsStarting(): void
notifyDeckChanged(CardType): void
notifySlotCardChanged(CardType, int): void
notifyPlayersPointsChanged(): void
notifyNumberOfPlayersChanged(): void
notifyPlayersTokenChanged(int): void
notifyPlayersPlayAreaChanged(int): void
notifyPlayersHandChanged(int): void
notifyPlayersObjectiveChanged(int): void
notifyCommonObjectivesChanged(): void
getReady(): void

[final] - viewController

- controller

View
(ViewController)

ServerManager
- port: int
+ setCollection(GameCollection): void

Both controllers access the same ServerManager, so the same GameControllers

- collection

Server
- ip: String
- port: Int
- isRunning: boolean
- canReadInput: boolean
+ notifyChanges(): void

2

Extends

Application

<<interface>>
ServerNetworkController
getIpAddress(): String
getPort(): String
setWhenReady(Runnable): void
start(): void
stop(): void

implements

GameCollection
[final] - gameControllers: ArrayList<GameController>
+ add(GameController): void
+ size(): int
+ get(int): GameController
+ getCopy(): GameCollection
+ createNewGame(): GameController
+ remove(GameController): void

implements

Creates and keeps inside, and uses it to translate received messages in method calls on the ViewController

SocketControllerServer
- ipAddress: String
- port: int
- onReady: Runnable
[final] - games: GameCollection
- users: ArrayList<Players>
- server: Socket
- in: Scanner
- out: PrintWriter
- registerUser(Player): void
- receiveMessage(Message): void

- server

Message

SOCKET CONNECTION

Message

SocketClient
- ip: String
- port: Int
+ connect(String, Int): void
+ closeConnection(): void
+ sendMessage(Message): void

Il metodo translate() trasforma le stringhe ricevute dal client in metodi dell'interfaccia

Use

Use

Message

KickPlayerMessage
- playerID: int

GetPlayerMessage
- playerID: int

SetNameMessage
- name: String

AddPlayerMessage
- player: Player

The toString() method returns the Player's nickname.
The Getter method is used to retrieve the Player

Extends

GameMessage
- gameID: int

Extends

Message
+ toString(): String

Other types of Messages, which don't include the gameID and the playerID but only the MessageType, are sent as Message

- type

MessageType
START_GAME
SET_PLAYER_STATUS
NEW_GAME
GET_GAME
GRAB_CARD
PLAY_CARD
KICK_PLAYER
NEXT_TURN
ADD_PLAYER
DRAW_CARD
GET_NUMBER_OF_PLAYERS
SET_NAME
SET_PLAYER_SECRET_OBJECTIVES
SET_PLAYER_TOKEN
DRAW_SECRET_OBJECTIVES
FLIP_CARD
SET_CURRENT_STATUS
GET_NAME
GET_PLAYER

SetCurrentStatusMessage
- status: GameStatus

PlayerMessage
- playerID: int

Extends

SetPlayerTokenMessage
- token: Token

SetPlayerStatusMessage
- status: GameStatus

GrabCardMessage
- cardType: CardType
- slot: int

PlayCardMessage
- handCard: int
- x: int
- y: int

DrawCardMessage
- cardType: CardType

SetPlayerSecretObjectiveMessage
- pickedCard: int

FlipCardMessage
- cardID: int