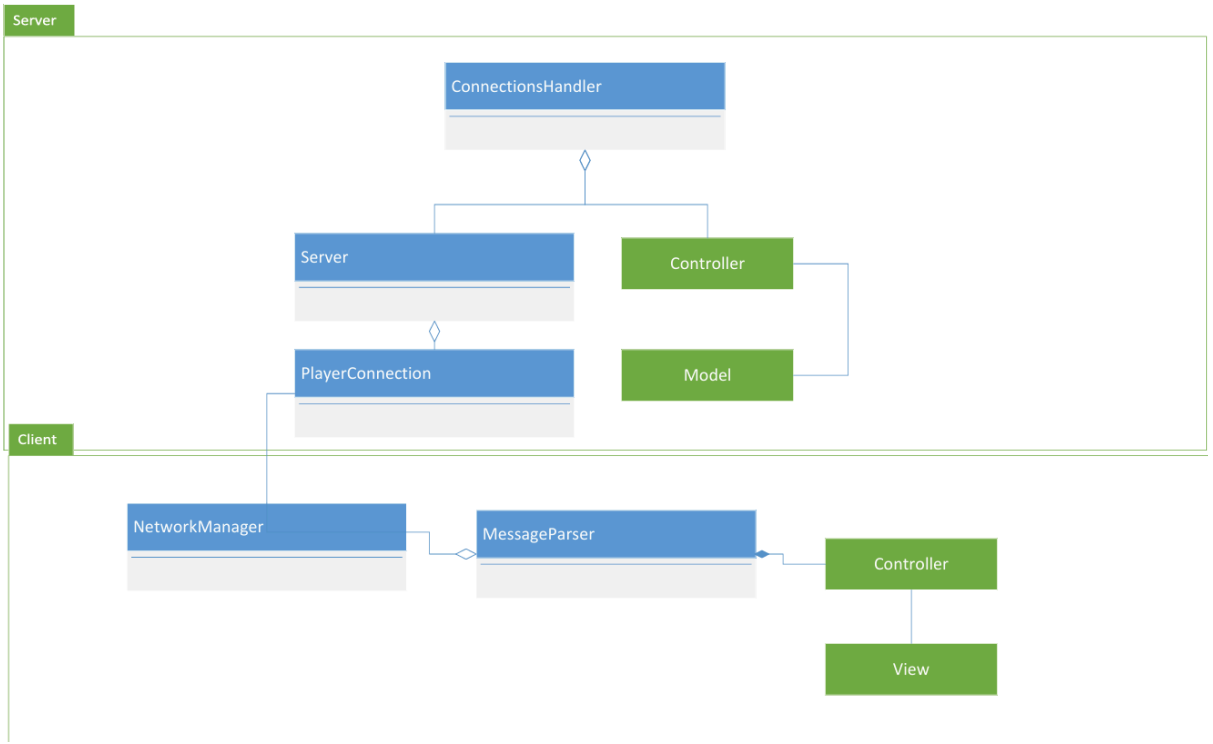
NETWORK

CLASS DIAGRAM(simplified)



Server

This class represents ServerSocket and its connection establishment logic. Also, it stores all the current connections

PlayerConnection

This class represents the player's connection. It contains logic to send and receive messages.

ConnectionsHandler

This class is used to handle the connections. Contains all the logic to check the connections status and server inputs.

NetworkManager

This class is used to handle the client Socket and contains logic to send and receive messages

MessageParser

This class is used to parse the current message and to call an update to the view controller

MESSAGES

PERSISTENCE

PERSISTENCE(Server)

This message is sent when the PersistenceHandler identifies a persistence instance

```
{
 "type":"PERSISTENCE",
 "params":{}
}
```

PERSISTENCE(Client)

This message is sent as a choice of the first player, it represent if he wants to continue a previous game or not

```
{
 "type":"PERSISTENCE",
 "params":{
   "persistence": Boolean
 }
}
```

CONTINUE

One message for each player is sent in broadcast when all users have joined a persistence affected game. It contains all the information necessary to continue a previous game.

```
{
 "type": "CONTINUE",
 "params": {
   "affectedPlayer": String,
   "players": [ //Contains a list of players ordered by the turn order
     String,
     String,
     String,
     String
   ],

   "score": Integer, //Contains the player's score
   "hand": [ //Contains the player's hand
     Card,
     Card,
     Card
   ],
   "placedCards":  //Contains an array composed by a serialization of the placed
cards in the affected player's board,
 [
 PlacedCard,
 ...,
 PlacedCard
 ],
   "publicObjectives":
     [Objective, Objective], //Contains the serialization of the public objectives
```

```
 "chosenPrivateObjective": {
   Objective //Contains the serialization of the objective of the affected player }
 },
   "resourceDrawableArea": [Card, Card, Card],//Contains a serialization of all the
cards in the resource drawable area
   "goldDrawableArea": [Card, Card, Card],//Contains a serialization of all the
cards in the gold drawable area
}
}
```

CONNECT

This message is sent when the ServerSocket accepts a player.

```
{
 "type":"CONNECT",
 "params":{
   "masterStatus": Boolean
 }
}
```

CREATE

CREATE(Client)

This message is sent when the master creates the game

```
{
 "type":"CREATE",
 "params":{
   "username": String,
   "numberOfPlayers": Integer
 }
}
```

JOIN

JOIN(Client)

This message is sent when a player joins a created game

```
{
 "type":"JOIN",
 "params":{
   "username": String,
 }
```

JOIN(Server)

This message is sent as response of the player's join

```
{
 "type":"JOIN",
 "params":{
```

```
   "currentPlayer": String,
   "unavailableUsername": Boolean
 }
}
```

CHAT

CHAT(Client)

This message is sent to the server when a player want to send a message to a player or to all players.

```
{
 "type":"CHAT",
 "params":{
   "username": String,//the sender
   "affectedPlayer": String,//the receiver, can be a player or -all
   "chat": String,//the message
 }
```

CHAT(Server)

This message is sent in broadcast when a player want to send a message to a player or to all players.

```
{
 "type":"CHAT",
 "params":{
   "currentPlayer": String,//the sender
   "affectedPlayer": String,//the receiver, can be a player or -all
   "chat": String,//the message
 }
```

FIRSTROUND

START_FIRSTROUND(Server)

This message is sent in broadcast when all users have joined the game

```
{
 "type": "START_FIRSTROUND",
 "params": {
   "players": [ //Contains a list of players ordered by the turn order
     String,
     String,
     String,
     String
   ],
   "card": {
     Card //Contains a serialization of the card in the model
   },
   "resourceDrawableArea": [Card, Card, Card],//Contains a serialization of all the
cards in the resource drawable area
   "goldDrawableArea": [Card, Card, Card],//Contains a serialization of all the
cards in the gold drawable area
   "availableColors":["RED", "GREEN", "BLUE", "YELLOW"]//Contains all the actual
available colors
 }
```

```
}
```

FIRSTROUND(Client)

This message represents the input of the user during the first round

```
{
 "type":"FIRSTROUND",
 "params":{
   "username": String,
   "isFaceUp":Boolean, //Contains the choice of the user to flip the card
   "color": String //Contains the color chosen by the user
 }
}
```

FIRSTROUND(Server)

This message is sent in broadcast after a first round is played by the affectedPlayer

```
{
 "type": "FIRSTROUND",
 "params": {
   "card": {
     Card //Contains a serialization of the current player starting card
   },
   "currentPlayer": String, //Contains the current player
   "availableColors": [String, String, ...], // Contains an array of colors
   "affectedPlayer": String, //Contains the player to update in the view
   "placedCards": {
       [PlacedCard,...,PlacedCard] //Contains an array composed by a serialization
of the placed cards in the affected player's board
   }
   "color": String //Contains the color the player chose
 }
}
```

SECONDROUND

START_SECONDROUND(Server)

This message is sent in broadcast when all users have chose their colors and placed their starting cards

```
{
 "type": "START_SECONDROUND",
 "params": {
   "currentPlayer": String, //Contains the current player username
   "hand": [Card, Card, Card],//Contains an ordered array of serialized cards based
on the model's hand
   "firstPrivateObjective": {//They both contains serialization of the current
player private objectives
     Objective
   },
   "secondPrivateObjective": {
     Objective
   },
   "firstPublicObjective": {//They both contains serialization of the public
objectives
```

```
    Objective
  },
  "secondPublicObjective": {
    Objective
  }
 }
}
```

SECONDROUND(Client)

This message represents the input of the user during the second round

```
{
 "type":"SECONDROUND",
 "username": String,
 "params":{
   "chosenObjectiveIndex":Integer, //Contains the index in the board of the chosen
objective
 }
}
```

SECONDROUND(Server)

This message is sent in broadcast to the clients to update the affected player and activate the current player

```
{
 "type":"SECONDROUND",
 "params":{
   "currentPlayer": String, //Contains the current player's username
   "firstPrivateObjective": {//They both contains serialization of the current
player private objectives
    Objective
  },
  "secondPrivateObjective": {
    Objective
  },
  "affectedPlayer": String, //Contains the player to update
  "chosenPrivateObjective":{
    Objective //Contains the serialization of the objective of the affected player
},}}
```

ACTION

STARTGAME(Server)

This message is sent in broadcast to the clients when all users have chosen their objectives.

```
{
 "type": "STARTGAME",
 "params": {
   "currentPlayer": String
 }
}
```

ACTION_PLACECARD(Client)

This message is sent when a user places a card

```
{
 "type": "ACTION_PLACECARD",
 "username": String, //The user who placed
 "params": {
   "cardHandIndex": Integer, //Contains he index of the card in the player's hand
in the model
   "flip": Boolean, //Contains if the card is flipped
   "Coordinates": { //Contains the coordinates to place the card
     "x": Integer,
     "y": Integer
   }
 }
}
```

ACTION_PLACECARD(Server)

This message is sent in broadcast when a ACTION_PLACECARD message is solved

```
{
 "type": "ACTION_PLACECARD",
 "params": {
   "currentPlayer": String,//Contains the current player username
   "affectedPlayer": String,//Contains the player who placed a card
   "score": Integer, //Contains the affected player's score
   "hand": [Card, Card, Card],//Contains an ordered array of serialized cards based
on the model's hand
   "placedCards": {
[PlacedCard,...,PlacedCard] //Contains an array composed by a serialization of the
placed cards in the affected player's board
   }
 }
}
```

ACTION_DRAWCARD(Client)

This message is sent when a client draws a card

```
{
 "type": "ACTION_DRAWCARD",
 "username": String //The player who is drawing
 "params": {
   "drawableSection": String,//Contains the drawableSection name
   "drawIndex": String,//Contains the drawn card index
 }
}
```

ACTION_DRAWCARD(Server)

This message is sent in broadcast when an ACTION_DRAWCARD is solved

```
{
```

```
 "type": "ACTION_DRAWCARD",
 "params": {
   "currentPlayer": String, //Contains the current player
   "affectedPlayer": String, //Contains the player who drawn a card
   "hand": [Card, Card, Card],//Contains an ordered array of serialized cards based
on the model's hand
   "placedCards": {
[PlacedCard,...,PlacedCard] //Contains an array composed by a serialization of the
placed cards in the affected player's board
   }
 }
   "score": Integer, //Contains the affected player's score
   "resourceDrawableArea": [Card, Card, Card],//Contains a serialization of all the
cards in the resource drawable area
   "goldDrawableArea": [Card, Card, Card],//Contains a serialization of all the
cards in the gold drawable area
 }
}
```

ENDGAME

This message is sent in broadcast when the game ends.

```
{
 "type":"ENDGAME",
 "params":{
   "scoreboard":[
     ScoreBoardPositionDTO //Contains each username associated with his score and
position in scoreboard
   ],
 }
}
```

ABORT

ABORT(Server)

This message is sent in broadcast for various reasons. It tells the clients to stop and exit the game in safe mode.

```
{
 "type": "ABORT",
 "params": {
   "cause": String //Contains the abort reason
 }
}
```