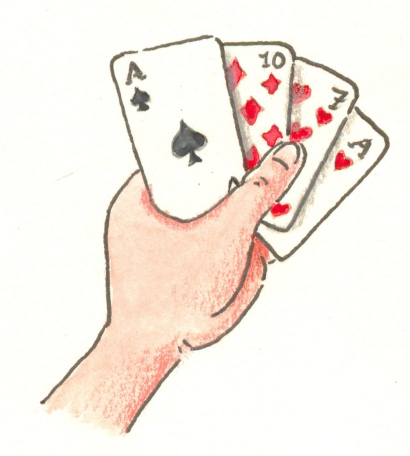
# SPEZIFIKATION

13. November 2013



# NET-WIZHEARTS

| Phase | Verantwortlicher | E-Mail |
|---|---|---|
| Pflichtenheft | Alina Meixl | alina@meixl.de |
| Entwurf | Viktoria Witka | witkaviktoria@freenet.de |
| Spezifikation | Daniel Riedl | dariedl14@yahoo.de |
| Implementation | Andreas Altenbuchner | a.andi007@gmail.com |
| Verifikation | Patrick Kubin | kubin@fim.uni-passau.de |
| Präsentation | w | w |

# Inhaltsverzeichnis

# 1 Einleitung

In diesem Dokument werden spezifische Angaben zu den bereits im Entwurf vorgestellten Klassen gemacht. Es werden im Folgenden Methoden und Attribute, sowie die Klassen genau beschrieben. Zunächst werden dazu die Änderungen vorgestellt, die seit dem Entwurf vorgenommen wurden. Des weiteren werden JUnit-Tests gezeigt, die zum Testen des späteren Programms essentiell sind. Zuletzt wird ein Implementierungsplan mit verschiedenen Milestones aufgezeigt.

# 2 Änderungen im Vergleich zum Entwurf

## 2.1 Server

### 2.1.1 Server

Die Klasse Server war zuvor ein Interface und ist jetzt eine abstrakte Klasse von der der LobbyServer und der GameServer erben. So kann Codeduplikation vermieden werden.

## 2.2 Client

### 2.2.1 Language

Das Enum Language wurde in das Packet ClientView verschoben, da es von der View benötigt wird um die richtige Sprache anzuzeigen.

### 2.2.2 MVMessage

Die MVMessages wurden gelöscht und durch ein Enum ViewNotification ersetzt. Der Observer bekommt Änderungen über die Notifications mitgeteilt. Dies macht eine weitere Einteilung in einzelnde Messages unnötig.

### 2.2.3 ClientState-Enum

Dieses Enum enthält jeden Status, den ein Client erreichen kann.

## 2.3 ClientView

### 2.3.1 MVMessages

Dieses Interface wurde gelöscht, da es unnötig ist.

### 2.3.2 ViewCard

ViewCard ist die View-seitige Repräsentation einer Karte. Sie wird verwendet um einzelne Karten auf das Spielfeld zu zeichnen.

## 2.4 Ruleset

#### 2.4.1 ServerRuleset

Das ServerRuleset ist statt einem Interface nun eine abstrakte Klasse, von der die Klassen ServerHearts (vorher Hearts) und ServerWizard (vorherWizard) erben. Codeduplikation kann so vermieden werden.

#### 2.4.2 ClientRuleset

Das ClientRuleset ist statt einem Interface nun eine abstrakte Klasse, von der die Klassen ClientWizard und ClientHearts erben. So kann Codeduplikation vermieden werden.

#### 2.4.3 Card

Die vorher abstrakte Klasse Card ist nun ein Interface, das von HeartsCard und WizardCard implementiert wird. So können die Methoden der Klasse Card von den anderen beiden so implementiert werden, wie es für das Spiel nötig ist.

#### 2.4.4 Colour

Ein Enum Colour wurde hinzugefügt. Dieses enthält alle Farben, die eine Karte haben kann unabhängig vom Regelwerk.

#### 2.4.5 RulesetType

Ein Enum RulesetType wurde hinzugefügt. Dieses enthält die Typen der Regelwerke, also deren Namen.

### 2.5 ComObjects

#### 2.5.1 ComObject

ComObject ist im Zuge des Einsetzens des VisitorPatterns von einer abstrakten Klassen zu einem Interface geändert worden.

#### 2.5.2 RulesetMessage

RulesetMessage ist im Zuge des Einsetzens des VisitorPatterns von einer abstrakten Klassen zu einem Interface geändert worden.

#### 2.5.3 ComLogin

Diese Klassen trägt den neuen Namen ComLoginRequest.

#### 2.5.4 ComKickPlayer

Diese Klasse trägt den neuen Namen ComKickPlayerRequest.

#### 2.5.5 GameClientUpdate

Diese Klasse wurde gelöscht, da sie unnötig ist.

#### 2.5.6 commands

Dieses Enum wurde gelöscht, da es unnötig ist. An Stelle von diesem wurden weitere ComObjects hinzugefügt.

#### 2.5.7 types

Dieses Enum wurde gelöscht, da es unnötig ist.

#### 2.5.8 ComBeenKicked

Diese Klasse wurde zusätzlich hinzugefügt. Die Nachricht wird an einen Spieler gesendet, wenn er aus einem Spiel erntfernt wurde.

#### 2.5.9 ComClientLeave

Diese Klasse wurde zusätzlich hinzugefügt. Sie wird zur Benachrichtigung gesendet, wenn ein Spieler ins nächste Fenster möchte und aus dem alten entfernt werden soll.

#### 2.5.10 ComClientQuit

Diese Klasse wurde zusätzlich hinzugefügt. Die Nachricht wird verschickt, wenn der Spieler ein Fenster schließt.

#### 2.5.11 ComCreateGameRequest

Diese Klasse wurde zusätzlich hinzugefügt. Diese Nachricht wird versendet, wenn ein neues Spiel erstellt werden soll.

#### 2.5.12 ComGameEnd

Diese Klasse wurde zusätzlich hinzugefügt. Diese Nachricht wird versendet, wenn ein Spiel oder eine Runde zu Ende ist.

#### 2.5.13 ComServerAcknowledgement

Diese Klasse wurde zusätzlich hinzugefügt. Diese Nachricht wird vom Server zur Bestätigung gesendet.

#### 2.5.14 ComStartGame

Diese Klasse wurde zusätzlich hinzugefügt. Diese Nachricht wird versendet, wenn ein Spiel gestartet werden soll.

#### 2.5.15 ComWarning

Diese Klasse wurde zusätzlich hinzugefügt. Diese Nachricht wird versendet, wenn ein Fehler passiert ist.

#### 2.5.16 MsgCardRequest

Diese Klasse wurde zusätzlich hinzugefügt. Diese Nachricht wird von Server gesendet, um einem Spieler mitzuteilen, dass er das Spielen einer Karte erwartet.

#### 2.5.17 MsgMultiCardsRequest

Diese Klasse wurde zusätzlich hinzugefügt. Diese Nachricht wird gesendet, wenn die Auswahl mehrerer Karten vom Spieler gefordert werden soll.

#### 2.5.18 MsgNumberRequest

Diese Klasse wurde zusätzlich hinzugefügt. Diese Nachricht wird gesendet, wenn die Eingabe einer Zahl gefordert werden soll.

#### 2.5.19 MsgSelectionRequest

Diese Klasse wurde zusätzlich hinzugefügt. Diese Nachricht sendet der Server an einen Spieler, wenn er eine Farbauswahl von diesem erwartet.

#### 2.5.20 ComObjects-Klassen allgemein

Da die ehemals abstrakten Klassen ComObject und RulesetMessage jetzt Interfaces sind, implementieren alle anderen Objekte diese statt von ihnen zu erben, des Weiteren implementieren alle Serializable.

# 3 Hierarchie-Verzeichnis

## 3.1 Klassenhierarchie

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

# 4 Klassen-Verzeichnis

## 4.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

# 5 Klassen-Dokumentation

## 5.1 ClientController Klassenreferenz

#### Private Attribute

- ClientModel **clientModel**
- ChooseCards **chooseCards**
- ScoreWindow **scoreWindow**
- InputNumber **inputNumber**
- ChooseItem **chooseColour**
- Lobby **lobby**
- Set< Login > **login**
- CreateGame **createGame**
- GameLobby **gameLobby**
- Game **game**
- Password **password**
- Language **language**
- Set< Warning > **warning**

### 5.1.1 Ausführliche Beschreibung

Der ClientController enthaelt alle ActionListener der View und leitet durch diese Benutzereingaben an das ClientModel weiter. Sobald der ClientController von der ClientMain-Klasse erzeugt wird, erzeugt er wiederum das ClientModel und die Fenster der ClientView, wobei zunaechst nur das Login Fenster sichtbar ist.

## 5.2 ClientMain Klassenreferenz

#### Öffentliche, statische Methoden

- static void main (final String[] args)

#### Private Attribute

- ClientController clientController

### 5.2.1 Ausführliche Beschreibung

Die ClientMain Klasse startet den Spielclient und initialisiert dessen Komponenten.

### 5.2.2 Dokumentation der Elementfunktionen

#### 5.2.2.1 static void main ( final String[] *args* ) [static]

**Parameter**

| | |
|---|---|
| *args* | |

## 5.3 ClientModel Klassenreferenz

Abgeleitet von Observable.

### Öffentliche Methoden

- ClientModel (MessageListenerThread netIO) throws IllegalArgumentException
- void leaveWindow ()
- void terminateProgram ()
- void receiveMessage (ComChatMessage msg)
- void receiveMessage (ComInitLobby msg)
- void receiveMessage (ComInitGameLobby msg)
- void receiveMessage (ComRuleset msg)
- void receiveMessage (ComServerAcknowledgement ack)
- void receiveMessage (ComBeenKicked msg)
- void receiveMessage (ComUpdatePlayerlist update)
- void receiveMessage (ComLobbyUpdateGamelist update)
- void receiveMessage (ComObject com)
- List< String > getPlayerlist ()
- List< GameServerRepresentation > getLobbyGamelist ()
- List< Card > getPlayedCards ()
- List< Card > getOwnHand ()
- List< String > getOtherPlayerData ()
- int getOwnScore ()
- void setLanguage (final Language language)
- Language getLanguage ()
- void kickPlayer (final String name)
- void hostGame (String gameName, boolean hasPassword, String password, RulesetType ruleset)
- void send (ComObject object)
- void send (RulesetMessage msg)
- int getPlayerCount ()
- String getWindowText ()
- List< Card > getChooseCards ()
- void giveChosenCards (List< Card > cards)
- void openChooseCards (List< Card > cards, String text)
- List< Object > getChooseItems ()
- void giveChosenItem (Object item)
- void openChooseItem (List< Object > items, String text)
- void giveInputNumber (int number)
- void openInputNumber (String text)
- void sendChatMessage (final String msg)
- void joinGame (final String name, final String password)
- void startGame ()
- void makeMove (Card card)
- void createConnection (final String username, final String serverAdress, final int port)
- String getWarningText ()
- RulesetType[] getRulesets ()

### Private Methoden

- void initGame ()
- void informView (ViewNotification note)

**Private Attribute**

- String playerName
- ClientRuleset ruleset
- Language language
- ClientState state
- List< String > **playerList**
- String **chatMessage**
- Set< GameServerRepresentation > **gameList**
- MessageListenerThread netIO

#### 5.3.1 Ausführliche Beschreibung

Das ClientModel ist die Schnittstelle zwischen dem MessageListenerThread, dem ClientRuleset und der View. Das Model prüft Nachrichten, welche es vom MessageListenerThread über die Methode receiveMessage() bekommt. RulesetMessages werden an das ClientRuleset weitergeleitet. Weiterhin informiert es seine Observer über Veraenderungen und stellt ihnen Methoden zu Verfügung um spielrelevante Daten zu lesen. Weiterhin kann das ClientModel ComMessages and den Server schicken, um Kommandos des ClientRulesets oder Eingaben des Controllers weiterzugeben.

#### 5.3.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.3.2.1 ClientModel ( MessageListenerThread *netIO* ) throws IllegalArgumentException

Erstellt ein ClientModel und erwartet als Argument einen MessageListenerThread fuer die Netzwerkanbindung.

**Parameter**

| | |
|---|---|
| *netIO* | MessageListenerThread fuer die Netzwerkverbindung. |

**Ausnahmebehandlung**

| | |
|---|---|
| *IllegalArgumentException* | Wird geworfen bei fehlerhaftem MessageListenerThread Argument. |

Benutzt ClientModel.netIO.

#### 5.3.3 Dokumentation der Elementfunktionen

##### 5.3.3.1 void leaveWindow ( )

Wird aufgerufen, wenn der User die GameLobby verlaesst.

Der Client gelangt zurueck in die Lobby.

##### 5.3.3.2 void receiveMessage ( ComChatMessage *msg* )

Sendet eine eingehende Chatnachricht direkt an alle Observer weiter.

**Parameter**

| | |
|---|---|
| *msg* | die ankommende ComChatMessage Nachricht |

##### 5.3.3.3 void receiveMessage ( ComInitLobby *msg* )

Diese Methode wird aufgerufen, falls der Server den Spieler erfolgreich in die Lobby hinzugefügt hat.

Empfaengt die ComInitGameLobby Nachricht, die eine Liste aller Spieler enthaelt, die sich in der Lobby befinden. Speichert diese Liste und benachrichtigt die Observer mit der loginSuccesful ViewNotification.

**Parameter**

| | |
|---|---|
| *msg* | die ankommende ComInitLobby Nachricht |

#### 5.3.3.4 void receiveMessage ( ComInitGameLobby *msg* )

Diese Methode wird aufgerufen, falls der Server den Spieler erfolgreich in die GameLobby hinzugefuegt hat.

Empfaengt die ComInitGameLobby Nachricht, die eine Liste aller Spieler enthaelt, die sich in der GameLobby befinden. Speichert diese Liste und benachrichtigt die Observer mit der joinGameSuccesful ViewNotification.

**Parameter**

| | |
|---|---|
| *msg* | die ankommende ComInitGameLobby Nachricht |

#### 5.3.3.5 void receiveMessage ( ComRuleset *msg* )

Diese Methode wird aufgerufen, falls eine Nachricht für das Regelwerk ankommt.

Die darin enthaltene RulesetMessage wird dem ClientRuleset zur Verarbeitung uebergeben.

**Parameter**

| | |
|---|---|
| *msg* | Die ankommende ComRuleset Nachricht |

#### 5.3.3.6 void receiveMessage ( ComServerAcknowledgement *ack* )

Diese Methode wird aufgerufen, falls ein Server Acknowledgement auftritt.

Dabei ist es von Bedeutung, in welchem Zustand sich der Client befindet.

**Parameter**

| | |
|---|---|
| *ack* | Eine Bestätigung durch den Server. |

#### 5.3.3.7 void receiveMessage ( ComBeenKicked *msg* )

Diese Methode wird aufgerufen, falls der Spieler aus der Spiellobby durch einen Spielleiter entfernt wurde.

Der Client gelangt zurueck in die Lobby, die Observer werden mit windowChangeForced benachrichtigt.

**Parameter**

| | |
|---|---|
| *msg* | die ankommende ComBeenKicked Nachricht |

#### 5.3.3.8 void receiveMessage ( ComUpdatePlayerlist *update* )

Diese Methode wird aufgerufen, falls auf dem Server ein neuer Spieler die Lobby/GameLobby betreten hat oder sie von einem Spieler verlassen wurde.

Empfaengt die ComUpdatePlayerlist Nachricht, die die Information enthaelt, ob und welcher Spieler hinzugefuegt oder entfernt werden muss. Die Spielerliste wird dementsprechend bearbeitet und die Observer mit playerList-Update informiert.

**Parameter**

| | |
|---|---|
| *update* | die ankommende ComLobbyUpdatePlayerlist Nachricht |

#### 5.3.3.9 void receiveMessage ( ComLobbyUpdateGamelist *update* )

Diese Methode wird aufgerufen, falls auf dem Server ein neues Spiel erstellt wurde oder ein Spiel geschlossen/beendet wurde.

Empfaengt die ComLobbyUpdateGamelist Nachricht, die die Information enthaelt, ob und welches Spiel hinzuge-

fuegt oder entfernt werden muss. Die Spielliste wird dementsprechend bearbeitet und die Observer mit gameList-Update informiert.

**Parameter**

| | |
|---|---|
| *update* | die ankommende ComLobbyUpdateGamelist Nachricht |

#### 5.3.3.10 void receiveMessage ( ComObject *com* )

Standard receiveMessage Methode, die ComObjekte zur Weiterverarbeitung identifiziert.

**Parameter**

| | |
|---|---|
| *com* | Das auflaufende ComObjekt. |

#### 5.3.3.11 List<String> getPlayerlist ( )

Liefert eine Liste der Namen der Spieler in der Lobby oder GameLobby.

**Rückgabe**

Liste von Spielernamen

#### 5.3.3.12 List<GameServerRepresentation> getLobbyGamelist ( )

Liefert eine Liste der Spiele, die aktuell auf dem Server offen sind oder gerade gespielt werden.

**Rückgabe**

Liste aller Spiele der Lobby.

#### 5.3.3.13 List<Card> getPlayedCards ( )

Gibt eine Liste aller bereits ausgespielten Karten zurueck.

**Rückgabe**

List<Card>. Eine Liste der gespielten Karten.

#### 5.3.3.14 List<Card> getOwnHand ( )

Gibt eine Liste der Handkarten des Spielers zurueck.

**Parameter**

| | |
|---|---|
| *Liste* | aller Handkarten des Spielers |

#### 5.3.3.15 List<String> getOtherPlayerData ( )

Gibt zusaetzliche Daten der anderen Spieler zurueck.

**Rückgabe**

Liste der Stringrepräsentationen der OtherData der Spieler

#### 5.3.3.16 int getOwnScore ( )

Gibt den Punktestand des Spielers zurueck.

**Rückgabe**

der eigene Punktestand.

#### 5.3.3.17 void setLanguage ( final Language *language* )

Setzt die Sprache der GUI.

**Parameter**

| | |
|---|---|
| *language* | Enumerator der die Spielsprache anzeigt. |

Benutzt ClientModel.language.

#### 5.3.3.18 Language getLanguage ( )

Liefert die Sprache der GUI.

**Rückgabe**

language Enumerator der die Spielsprache anzeigt.

Benutzt ClientModel.language.

#### 5.3.3.19 void kickPlayer ( final String *name* )

Entfernt einen Spieler aus der GameLobby.

**Parameter**

| | |
|---|---|
| *Name* | des Spielers, der enfernt werden soll |

#### 5.3.3.20 void hostGame ( String *gameName,* boolean *hasPassword,* String *password,* RulesetType *ruleset* )

Erstellt ein neues Spiel.

Sendet dazu eine ComCreateGameRequest Nachricht an den Server.

**Parameter**

| | |
|---|---|
| *gameName* | String Name des Spieles. |
| *hasPassword* | true, wenn das Spiel ein Passwort hat |
| *password* | String Passwort zum sichern des Spieles. |
| *ruleset* | das zu verwendende Regelwerk |

#### 5.3.3.21 void send ( ComObject *object* )

Sendet erstellte ComObjects an den Server.

**Parameter**

| | |
|---|---|
| *object* | ComObject, das verschickt wird |

Wird benutzt von ClientModel.send() und ClientModel.sendChatMessage().

#### 5.3.3.22 void send ( RulesetMessage *msg* )

Sendet eine RulesetMessage an den Server.

Erstellt dazu eine ComRuleset, die die RulesetMessage enthaelt.

**Parameter**

| | |
|---|---|
| *msg* | die RulesetMessage, die an den Server geschickt werden soll |

Benutzt ClientModel.send().

#### 5.3.3.23 int getPlayerCount ( )

Die die Anzahl der Spieler eines Spieles zurueck.

**Rückgabe**

int Die Spielerzahl eines Spieles.

#### 5.3.3.24 String getWindowText ( )

Gibt den Text zurueck, der in einem Sonderfenster (InputNumber, ChooseItem, ChooseCards) angezeigt werden soll.

**Rückgabe**

String

#### 5.3.3.25 List<Card> getChooseCards ( )

Gibt die Karten zurueck, aus denen gewaehlt werden soll.

**Rückgabe**

Karten, aus denen gewahlt werden kann

#### 5.3.3.26 void giveChosenCards ( List< Card > *cards* )

Uebergibt die Karten, die vom User gewahlt wurden.

Diese werden dann dem Regelwerk weitergegeben. Akzeptiert dieses die gewahlten Karten nicht, wird nochmal openChooseCards aufgerufen.

**Parameter**

| | |
|---|---|
| *cards* | Karten, die der User gewahlt hat |

#### 5.3.3.27 void openChooseCards ( List< Card > *cards*, String *text* )

Benachrichtigt die Observer mit der openChooseCards ViewNotification und speichert die Liste der Karten sowie den Anzeigetext des Regelwerks zwischen.

**Parameter**

| | |
|---|---|
| *cards* | Liste der Karten, von denen gewaehlt werden kann |
| *text* | Text, der dem User angezeigt werden soll |

#### 5.3.3.28 List<Object> getChooseItems ( )

Gibt die Items zurueck, aus denen eines gewaehlt werden soll.

**Rückgabe**

Items, aus denen gewahlt werden kann

#### 5.3.3.29 void giveChosenItem ( Object *item* )

Uebergibt das Item, das vom User gewahlt wurden.

Dieses wird dann dem Regelwerk weitergegeben. Akzeptiert dieses das gewahlte Item nicht, wird nochmal openChooseItem aufgerufen.

**Parameter**

| | |
|---|---|
| *item* | Item, das der User gewahlt hat |

#### 5.3.3.30 void openChooseItem ( List< Object > *items,* String *text* )

Benachrichtigt die Observer mit der openChooseItem ViewNotification und speichert die Liste der Items, von denen eines gewaehlt werden soll, sowie den Anzeigtetext des Regelwerks zwischen.

**Parameter**

| | |
|---|---|
| *items* | Liste der Items, von denen eines gewaehlt werden soll |
| *text* | Text, der dem User angezeigt werden soll |

#### 5.3.3.31 void giveInputNumber ( int *number* )

Uebergibt die Zahl, die vom User gewahlt wurde.

Diese wird dann dem Regelwerk weitergegeben. Akzeptiert dieses die gewaehlte Zahl nicht, wird nochmal openInputNumber aufgerufen.

**Parameter**

| | |
|---|---|
| *number* | Zahl, die vom User gewahlt wurde |

#### 5.3.3.32 void openInputNumber ( String *text* )

Benachrichtigt die Observer mit der openInputNumber ViewNotification und speichert den Anzeigetext des Regelwerks zwischen.

**Parameter**

| | |
|---|---|
| *text* | Text, der dem User angezeigt werden soll |

#### 5.3.3.33 void sendChatMessage ( final String *msg* )

Nimmt vom ClientController eine Chatnachricht entgegen und sendet diese an den Server.

**Parameter**

| | |
|---|---|
| *msg* | die Chatnachricht, die an den Server geschickt werden soll |

Benutzt ClientModel.send().

#### 5.3.3.34 void joinGame ( final String *name,* final String *password* )

Versucht einem Spiel beizutreten.

Sendet dazu eine ComJoinRequest Nachricht an den Server. Wird diese bestaetigt, gelangt der Client in die GameLobby. Wird die Nachricht nicht bestaetigt, wird eine Fehlermeldung ausgegeben und die Observer mit openWarning informiert.

**Parameter**

| | |
|---|---|
| *name* | String Der Name des Spiels. |
| *password* | String Passwort eines Spieles. |

#### 5.3.3.35 void startGame ( )

Versucht das erstellte Spiel zu starten.

Sendet dazu eine ComStartGame an den Server. Wenn der Client der Spielleiter des Spiels ist, gelangt er ins Spiel. Wenn der Client nicht der Spielleiter des Spiels ist, wird eine Fehlermeldung ausgegeben.

#### 5.3.3.36 void initGame ( ) [private]

Diese Methode wird innerhalb des ClientModels aufgerufen wenn ein Spiel vom Spielleiter gestartet wurde.

Der Client gelangt ins Spiel Die Observer werden über die gameStarted ViewNotification benachrichtigt.

#### 5.3.3.37 void makeMove ( Card *card* )

Versucht eine Karte auszuspielen.

Laesst dazu vom ClientRuleset ueberpruefen ob, die ausgewaehlte Karte gespielt werden darf. Wenn ja, wird sie im ClientRuleset weiterbehandelt. Wenn nein, wird eine Fehlermeldung ausgegeben und dazu die Observer mit openWarning informiert.

**Parameter**

| | |
|---|---|
| *die* | ID der gespielten Karte |

#### 5.3.3.38 void informView ( ViewNotification *note* ) [private]

Hilfsmethode die alle verbundenen Observer der GUI kontaktiert.

**Parameter**

| | |
|---|---|
| *note* | Enum der die Art des Aufrufes bestimmt. |

#### 5.3.3.39 void createConnection ( final String *username*, final String *serverAdress*, final int *port* )

Erstellt den MessageListenerThread und fuehrt den Benutzerlogin durch.

**Parameter**

| | |
|---|---|
| *username* | String der eindeutige Benutzername der für den Login verwendet wird. |
| *serverAdress* | String die Adresse des spielservers. |
| *port* | Integer der Port des Spielservers. |

#### 5.3.3.40 String getWarningText ( )

Gibt den Text aus der bei einer Spielwarnung angezeigt wird.

**Rückgabe**

String Text der Warnung.

#### 5.3.3.41 RulesetType [] getRulesets ( )

Liefert ein Array mit allen implementierten Regelwerken.

**Parameter**

| | |
|---|---|
| *RulesetType[]* | Array von unterstützten Regelwerken. |

## 5.4 ClientState Enum-Referenz

**Öffentliche Attribute**

- LOGIN
- SERVERLOBBY
- GAMECREATION
- PASSWORDREQUEST
- GAMELOBBY
- GAME
- USERREQUEST
- ENDING

### 5.4.1 Ausführliche Beschreibung

Dieser Enumerator enthält alle Zustaende in denen sich der Client befinden kann.

## 5.5 MessageListenerThread Klassenreferenz

Abgeleitet von Thread.

**Öffentliche Methoden**

- MessageListenerThread ()
- void startConnection (ClientModel model, Socket connection) throws IllegalArgumentException, IOException
- void closeConnection ()
- void send (ComObject object)
- void run ()

**Private Attribute**

- Socket **socket**
- ObjectInput **in**
- ObjectOutput **out**
- boolean **run** = false
- ClientModel **model**

### 5.5.1 Ausführliche Beschreibung

Diese Klasse implementiert die Netzwerkanbindung des Clients an den Server. Sie enthaelt den dazu noetigen Socket und ObjektStream Reader und Writer.

### 5.5.2 Dokumentation der Elementfunktionen

#### 5.5.2.1 void startConnection ( ClientModel *model,* Socket *connection* ) throws IllegalArgumentException, IOException

Initialisiert die ObjectStreams und speichert den TCP Socket im Thread.

**Parameter**

| | |
|---|---|
| *model* | ClientModel, Das Model das den Spielablauf und Serverkommunikation steuert. |
| *connection* | Socket, der Socket über den die TCP Verbindung laeuft. |

**Ausnahmebehandlung**

| | |
|---|---|
| *IllegalArgumentException* | Wird geworfen bei falschen ClientModel oder Socket Argumenten. |
| *IOException* | Wird geworfen beim fehlerbehafteten Erstellen der ObjectStreams. |

## 5.6 ChooseCards Klassenreferenz

Abgeleitet von Observer.

**Öffentliche Methoden**

- void update (Observable o, Object arg)

**Private Attribute**

- OwnHand **playerHandPanel**

### 5.6.1 Ausführliche Beschreibung

In diesem Fenster muss der Benutzer eine vorbestimmte Menge Karten auswaehlen.

### 5.6.2 Dokumentation der Elementfunktionen

#### 5.6.2.1 void update ( Observable *o*, Object *arg* )

Wird durch notify() im ClientModel aufgerufen.

Je nach dem in arg uebergebenen Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet: openChooseCards, chooseCardsSuccessful |

## 5.7 ChooseItem Klassenreferenz

Abgeleitet von Observer.

**Öffentliche Methoden**

- void update (Observable arg0, Object arg1)

**Private Attribute**

- Object **itemComboBox**

#### 5.7.1 Ausführliche Beschreibung

Dieses Fenster ermoeglicht es dem Spieler aus einer Liste von Items eines auszuwaehlen.

#### 5.7.2 Dokumentation der Elementfunktionen

##### 5.7.2.1 void update ( Observable *arg0,* Object *arg1* )

Wird durch notify() im ClientModel aufgerufen.

Je nach dem in arg uebergebenen Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet: openChooseItem, chooseItemSuccessful |

## 5.8 CreateGame Klassenreferenz

Abgeleitet von JFrame.

#### Öffentliche Methoden

- CreateGame ()
- void addPanelMouseListener (MouseListener m)
- void addRulesetSelectionListener (ItemListener i)
- void addCreateButtonListener (ActionListener a)
- void addLeaveButtonListener (ActionListener a)
- void setLanguage (Language l)

#### Öffentliche, statische Methoden

- static void **main** (String[] args)

#### Private Methoden

- void **updateLanguage** ()

#### Private Attribute

- Language **lang**
- JTextField **nameField**
- BufferedImage **image**
- JTextField **passwordField**
- JPanel **imagePanel**
- JLabel **lblSelect**
- JComboBox< String > **rulesetBox**
- JCheckBox **chckbxPassword**
- JButton **btnLeave**
- JButton **btnCreate**
- JLabel **lblGameName**

**Statische, private Attribute**

- static final long **serialVersionUID** = -2893031560688870723L

### 5.8.1 Ausführliche Beschreibung

Das Fenster CreateGame dient dem Benutzer zur Erstellung eines neuen Spieles. Es bietet alle Komponenten, um ein Regelwerk zu waehlen, einen Spielnamen festzulegen und das Spiel durch ein Passwort zu schuetzen. In der Spielerstellung wird ein Titelbild des ausgewaehlten Spiels und eine kurze Beschreibung angezeigt. ueber 'Leave' kehrt der Spieler in die Lobby zurueck und mit 'Create' wird das Spiel erstellt.

### 5.8.2 Dokumentation der Elementfunktionen

#### 5.8.2.1 void addPanelMouseListener ( MouseListener *m* )

Fuegt einen MouseListener zum ImagePanel des CreateGame Fensters hinzu, der zur Anzeige des MouseOver--Texts verwendet wird.

**Parameter**

| | |
|---|---|
| *m* | ein MouseListener |

#### 5.8.2.2 void addRulesetSelectionListener ( ItemListener *i* )

Fuegt einen Listener fuer die Regelwerk-Auswahl des CreateGame Fensters hinzu.

**Parameter**

| | |
|---|---|
| *i* | ein ItemListener |

#### 5.8.2.3 void addCreateButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Create' Button hinzu.

**Parameter**

| | |
|---|---|
| *a* | ein ActionListener |

#### 5.8.2.4 void addLeaveButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Leave' Button hinzu.

**Parameter**

| | |
|---|---|
| *a* | ein ActionListener |

#### 5.8.2.5 void setLanguage ( Language *l* )

Aendert die Sprache des Fensters.

**Parameter**

| | |
|---|---|
| *l* | Sprache in Form des Language-Enums |

## 5.9 DiscardPile Klassenreferenz

**Private Attribute**

- Set< ViewCard > **card**

#### 5.9.1 Ausführliche Beschreibung

Stellt einen Ablagestapel dar, dieser kann sowohl für jeden Spieler einzeln oder für alle Spieler gemeinsam in der Mitte des Spielfeldes angezeigt werden.

## 5.10 DrawDeck Klassenreferenz

#### 5.10.1 Ausführliche Beschreibung

Stellt einen Aufnahmestapel dar.

## 5.11 Game Klassenreferenz

Abgeleitet von JFrame und Observer.

#### Öffentliche Methoden

- Game () throws IOException
- void makeTrickGameBoard (int playercount)
- void update (Observable o, Object arg)
- void update (Observable o, String arg)

#### Öffentliche, statische Methoden

- static void **main** (String[] args) throws IOException

#### Private Attribute

- JPanel **contentPane**
- JTextField **textField**
- GamePanel **panel**

#### Statische, private Attribute

- static final long **serialVersionUID** = -2655520138213745249L

#### 5.11.1 Ausführliche Beschreibung

Im Game Fenster laeuft das Spiel ab.Es enthaelt den Spielchat und ein GamePanel. Ausserdem koennen ueber ein Dropdown-Menue Aenderungen an Hintergrundbild und Kartenhintergruenden vorgenommen werden. Schliesseen beendet das Spiel und der Spieler wird in die Lobby zurueckgeleitet.

#### 5.11.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.11.2.1 Game ( ) throws IOException

Erstellt das Game Fenster.

**Ausnahmebehandlung**

| | |
|---|---|
| *IOException* | |

#### 5.11.3 Dokumentation der Elementfunktionen

##### 5.11.3.1 void makeTrickGameBoard ( int *playercount* )

Arrangiert die Elemente der Spielfeld-Oberflaeche für ein Kartenspiel, bei dem Stiche gemacht werden.

Hierfuer hat jeder Spieler einen eigenen Ablagestapel vor sich. Es koennen 3, 4, 5, oder 6 Spieler gewaehlt werden.

**Parameter**

| | |
|---|---|
| *playercount* | Anzahl der Spieler, wobei 3 <= playercount <=6 einzuhalten ist |

##### 5.11.3.2 void update ( Observable *o,* Object *arg* )

Wird durch notify() im ClientModel aufgerufen.

Je nach dem in arg uebergebenen ViewNotification-Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet: playedCardsUpdate, otherDataUpdate, moveAcknowledged, gameStarted |

##### 5.11.3.3 void update ( Observable *o,* String *arg* )

Wird durch notify() im ClientModel aufgerufen, wenn eine Chatnachricht uebergeben wird.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet eine Chatnachricht in String-Form |

### 5.12 GameLobby Klassenreferenz

Abgeleitet von JFrame und Observer.

**Öffentliche Methoden**

- GameLobby ()
- void addStartButtonListener (ActionListener a)
- void addRemoveButtonListener (ActionListener a)
- void addLeaveButtonListener (ActionListener a)
- void addChatMessageListener (KeyListener k)
- void setLanguage (Language l)
- void update (Observable o, Object arg)
- void update (Observable o, String arg)

**Öffentliche, statische Methoden**

- static void **main** (String[] args)

#### Private Methoden

- void **updateLanguage** ()

#### Private Attribute

- JPanel **contentPane**
- JTextField **messageField**
- Language **lang**
- JButton **btnRemovePlayer**
- JButton **btnLeave**
- JTextArea **chatlog**
- JButton **btnStartGame**

#### Statische, private Attribute

- static final long **serialVersionUID** = -1899311213351027436L

### 5.12.1 Ausführliche Beschreibung

Die GameLobby modelliert das Wartefenster, in dem beigetretene Spieler auf den Start des Spieles durch den Spielleiter warten. Der Spielleiter kann Spieler mit dem Remove Player Button entfernen. ueber Leave kehren die Spieler in die Lobby zurueck. Der spielinterne Chat ist ab hier verfuegbar.

### 5.12.2 Dokumentation der Elementfunktionen

#### 5.12.2.1 void addStartButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Start Game' Button hinzu.

**Parameter**

| | |
|---|---|
| *a* | ein ActionListener |

#### 5.12.2.2 void addRemoveButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Remove Player' Button hinzu.

**Parameter**

| | |
|---|---|
| *a* | ein ActionListener |

#### 5.12.2.3 void addLeaveButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Leave' Button hinzu.

**Parameter**

| | |
|---|---|
| *a* | ein ActionListener |

#### 5.12.2.4 void addChatMessageListener ( KeyListener *k* )

Fuet einen KeyListener fuer das Nachricht-Senden-Feld der Lobby hinzu.

**Parameter**

| | |
|---|---|
| *k* | |

#### 5.12.2.5 void setLanguage ( Language *l* )

Aendert die Sprache des Fensters.

**Parameter**

| | |
|---|---|
| *l* | Sprache in Form des Language-Enums |

#### 5.12.2.6 void update ( Observable *o*, Object *arg* )

Wird durch notify() im ClientModel aufgerufen.

Je nach dem in arg uebergebenen ViewNotification-Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet: joinGameSuccessful, playerListUpdate, windowChangeForced, gameStarted |

#### 5.12.2.7 void update ( Observable *o*, String *arg* )

Wird aufgerufen, wenn eine String-Nachricht im notify() uebergeben wird.

Dieser wird als Chatnachricht interpretiert und dem Chatlog angefuegt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet einen String, der eine Chatnachricht darstellt |

## 5.13 GamePanel Klassenreferenz

Abgeleitet von JPanel.

### Öffentliche Methoden

- GamePanel ()
- void makeTrickGameBoardThreePlayers ()
- void makeTrickGameBoardFourPlayers ()
- void makeTrickGameBoardFivePlayers ()
- void makeTrickGameBoardSixPlayers ()
- void **paintComponent** (Graphics g)

### Private Attribute

- OwnHand **ownHand**
- Object **ownScoreLabel**
- Set< OtherPlayer > **otherPlayer**
- DrawDeck **drawDeck**
- Set< DiscardPile > **discardPiles**
- BufferedImage **background**

#### Statische, private Attribute

- static final long **serialVersionUID** = -1041218552426155968L

### 5.13.1 Ausführliche Beschreibung

Das GamePanel ist die Komponente des Game-Fensters, welche das eigentliche Spiel darstellt. Es besteht aus veschiedenen Panelobjekten, welche je nach Regelwerk auf das Spielfeld gezeichnet werden. Dazu gehoeren die eigenen Karten, eventuell ausgewaehlte Karten, ein Textfeld z.B. zur Anzeige der Anzahl der restlichen Karten der Mitspieler und den Ablagestapel. Nach jeder Runde wird der Punktestand aktualisiert.

## 5.14 InputNumber Klassenreferenz

Abgeleitet von Observer.

#### Öffentliche Methoden

- void update (Observable o, Object arg)

#### Private Attribute

- Object **numberTextfield**

### 5.14.1 Ausführliche Beschreibung

In diesem Fenster, kann der Benutzer eine Zahl eingeben.

### 5.14.2 Dokumentation der Elementfunktionen

#### 5.14.2.1 void update ( Observable *o,* Object *arg* )

Wird durch notify() im ClientModel aufgerufen.

Je nach dem in arg uebergebenen Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet: openInputNumber, inputNumberSuccessful |

## 5.15 Language Enum-Referenz

#### Öffentliche Attribute

- **German**
- **English**
- **Bavarian**

### 5.15.1 Ausführliche Beschreibung

Language stellt Repraesentationen verschiedener Sprachen dar, die von der GUI verwendet werden, um festzustellen welche Anzeigesprache verwendet werden soll.

## 5.16 Lobby Klassenreferenz

Abgeleitet von JFrame und Observer.

#### Öffentliche Methoden

- Lobby ()
- void addJoinButtonListener (ActionListener a)
- void addHostButtonListener (ActionListener a)
- void addLeaveButtonListener (ActionListener a)
- void addChatMessageListener (KeyListener k)
- void setLanguage (Language l)
- void update (Observable o, Object arg)
- void update (Observable o, String arg)

#### Öffentliche, statische Methoden

- static void **main** (String[] args)

#### Private Methoden

- void **updateLanguage** ()

#### Private Attribute

- JPanel **contentPane**
- JTextField **messageField**
- JList **playerList**
- JList **gameList**
- JScrollPane **scrollPane**
- JButton **btnHostGame**
- JButton **btnJoinGame**
- JButton **btnLeave**
- JTextArea **chatlog**
- Language **lang**

#### Statische, private Attribute

- static final long **serialVersionUID** = 1L

### 5.16.1 Ausführliche Beschreibung

Diese Klasse erzeugt die Ansicht der ServerLobby auf der Client Seite, in der die Spieler neue Spiele erstellen oder offenen beitreten koennen. In der Lobby werden die Benutzernamen der sich in der Lobby befindenden Spieler, sowie offene Spiele angezeigt. In der Lobby koennen Chatnachrichten gesendet und empfangen werden. ueber 'Leave' verlaesst der Spieler das Spiel. ueber 'Host Game' wird der Spieler zum CreateGame-Fenster weiter geleitet und mit 'Join Game' kann einem bereits erstellten Spiel beigetreten werden.

### 5.16.2 Dokumentation der Elementfunktionen

#### 5.16.2.1 void addJoinButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Join' Button hinzu.

**Parameter**

| | |
|---|---|
| *a* | ein ActionListener |

#### 5.16.2.2 void addHostButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Host' Button hinzu.

**Parameter**

| | |
|---|---|
| *a* | ein ActionListener |

#### 5.16.2.3 void addLeaveButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Leave' Button hinzu.

**Parameter**

| | |
|---|---|
| *a* | ein ActionListener |

#### 5.16.2.4 void addChatMessageListener ( KeyListener *k* )

Fuegt einen KeyListener fuer das Nachricht-Senden-Feld der Lobby hinzu.

**Parameter**

| | |
|---|---|
| *k* | |

#### 5.16.2.5 void setLanguage ( Language *l* )

Aendert die Sprache des Fensters.

**Parameter**

| | |
|---|---|
| *l* | Sprache in Form des Language-Enums |

#### 5.16.2.6 void update ( Observable *o*, Object *arg* )

Wird durch notify() im ClientModel aufgerufen.

Je nach dem in arg uebergebenen ViewNotification-Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet: joinGameSuccessful, windowChangeForced, playerListUpdate, gameListUpdate, chatMessage |

#### 5.16.2.7 void update ( Observable *o*, String *arg* )

Wird aufgerufen, wenn eine String-Nachricht im notify() uebergeben wird.

Dieser wird als Chatnachricht interpretiert und dem Chatlog angefuegt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |

| | |
|---|---|
| *arg* | erwartet einen String, der eine Chatnachricht darstellt |

## 5.17 Login Klassenreferenz

Abgeleitet von JFrame und Observer.

#### Öffentliche Methoden

- Login ()
- void addConnectButtonListener (ActionListener a)
- void addLanguageSelectionListener (ItemListener i)
- void setLanguage (Language l)
- void update (Observable o, Object arg)

#### Öffentliche, statische Methoden

- static void **main** (String[] args) throws IOException

#### Private Methoden

- void **updateLanguage** ()

#### Private Attribute

- JPanel **contentPane**
- JTextField **nameField**
- JTextField **serverField**
- JComboBox< Language > **languageComboBox**
- JButton **btnConnect**
- Language **lang**
- JLabel **lblNickname**
- JLabel **lblHostIp**
- JLabel **lblLanguage**

#### Statische, private Attribute

- static final long **serialVersionUID** = -2516577977746181978L

### 5.17.1 Ausführliche Beschreibung

Das Login-Fenster repraesentiert den initialen Dialog zwischen Benutzer und Client. In diesem Fenster kann der Benutzer seinen Namen und die Adresse des Servers eingeben. Ausserdem ist ueber den Login die Auswahl der Sprache moeglich. ueber den Login-Button wird die Verbindung zum Server hergestellt.

### 5.17.2 Dokumentation der Elementfunktionen

#### 5.17.2.1 void addConnectButtonListener ( ActionListener *a* )

Fuegt einen Listener fuer den 'Connect' Button des Login Fensters hinzu.

**Parameter**

| | |
|---|---|
| *a* | ein ActionListener |

#### 5.17.2.2 void addLanguageSelectionListener ( ItemListener *i* )

Fuegt einen Listener fuer die Sprachauswahl des Login Fensters hinzu.

**Parameter**

| | |
|---|---|
| *i* | ein ItemListener |

#### 5.17.2.3 void setLanguage ( Language *l* )

Aendert die Sprache des Fensters.

**Parameter**

| | |
|---|---|
| *l* | Sprache in Form des Language-Enums |

#### 5.17.2.4 void update ( Observable *o*, Object *arg* )

Wird durch notify() im ClientModel aufgerufen.

Je nach dem in arg uebergebenen ViewNotification-Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet: loginSuccessful |

## 5.18 OtherPlayer Klassenreferenz

**Private Attribute**

- Object **name**
- Object **info**

### 5.18.1 Ausführliche Beschreibung

Zeigt die Informationen über die anderen Spieler an, also den Namen, ein Symbol für die verdeckte Hand und das Label für zusaetzliche Angaben.

## 5.19 OwnHand Klassenreferenz

**Private Attribute**

- Object **cards**
- Set< ViewCard > **card**

### 5.19.1 Ausführliche Beschreibung

Stellt die Karten dar, die der Spieler auf der Hand hat. Der Spieler kann eine Karte durch Anklicken auswaehlen und durch einen zweiten Klick ausspielen.

## 5.20 Password Klassenreferenz

Abgeleitet von JFrame und Observer.

### Öffentliche Methoden

- Password ()
- void addJoinButtonListener (ActionListener a)
- void setLanguage (Language l)
- void update (Observable o, Object arg)

### Öffentliche, statische Methoden

- static void **main** (String[] args)

### Private Methoden

- void **updateLanguage** ()

### Private Attribute

- JPanel **contentPane**
- JTextField **textField**
- JButton **btnJoin**
- JLabel **lblEnterPasswordPlease**
- JButton **btnLeave**
- Language **lang**

### Statische, private Attribute

- static final long **serialVersionUID** = 7994797823893327272L

### 5.20.1 Ausführliche Beschreibung

Dieses Fenster ermoeglicht die Eingabe eines Passwortes um einem Passwortgeschuetztem Spiel beizutreten oder per 'Leave' wieder in die Lobby zurueckzukehren.

### 5.20.2 Dokumentation der Elementfunktionen

#### 5.20.2.1 void addJoinButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Join' Button hinzu.

**Parameter**

| | |
|---|---|
| *a* | ein ActionListener |

#### 5.20.2.2 void setLanguage ( Language *l* )

Aendert die Sprache des Fensters.

**Parameter**

| | |
|---|---|
| *l* | Sprache in Form des Language-Enums |

#### 5.20.2.3 void update ( Observable *o,* Object *arg* )

Wird durch notify() im ClientModel aufgerufen.

Je nach dem in arg uebergebenen ViewNotification-Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet: openWarning, passwordAccepted |

## 5.21 ScoreWindow Klassenreferenz

Abgeleitet von Observer.

### Öffentliche Methoden

- void update (Observable o, Object arg)

### 5.21.1 Ausführliche Beschreibung

Dieses Fenster zeigt den momentanen Punktestand nach jeder Runde und den Gesamtpunktestand am Ende des Spieles an.

### 5.21.2 Dokumentation der Elementfunktionen

#### 5.21.2.1 void update ( Observable *o,* Object *arg* )

Wird durch notify() im ClientModel aufgerufen.

Je nach dem in arg uebergebenen Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet: showScore |

## 5.22 ViewCard Klassenreferenz

Abgeleitet von JPanel.

### Öffentliche Methoden

- ViewCard (String s, int n)
- int getID ()
- void **paintComponent** (Graphics g)

#### Private Attribute

- String **path**
- int **id**
- BufferedImage **face**

#### Statische, private Attribute

- static final long **serialVersionUID** = 8733682958484899430L

### 5.22.1 Ausführliche Beschreibung

ViewCard ist die View-seitige Repraesentation einer Karte. Sie wird verwendet um einzelne Karten auf das Spielfeld zu zeichnen. Dazu enthaelt sie die Pfadangabe zu dem Ordner, in dem die Bilder der Karten gespeichert sind, und eine ID, um das genaue Bild zu spezifizieren.

### 5.22.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.22.2.1 ViewCard ( String *s*, int *n* )

Erstellt eine neue Karte fuer die Anzeige und zeichnet dafuer das Bild, das durch die Pfadangabe s und seine Kardinaliaet n im Ordner angegeben ist.

Die Pfadangabe wird durch das Regelwerk bestimmt.

#### Parameter

| | |
|---|---|
| *s* | Pfadangabe zum zu zeichnenden Bild |
| *n* | ID der Karte |

### 5.22.3 Dokumentation der Elementfunktionen

#### 5.22.3.1 int getID ( )

Gibt die ID der Karte zurueck.

#### Rückgabe

ID der Karte

## 5.23 Warning Klassenreferenz

Abgeleitet von Observer.

#### Öffentliche Methoden

- void setText (String text)
- void update (Observable o, Object arg)

#### Private Attribute

- String **warningText**

#### 5.23.1 Ausführliche Beschreibung

Das Warning-Fenster zeigt dem Benutzer Fehlermeldungen bzw. Hinweise an, welche vom ClientModel uebergeben wurden. Es wird nur im Fehlerfall angezeigt.

#### 5.23.2 Dokumentation der Elementfunktionen

##### 5.23.2.1 void setText ( String *text* )

Setzt den Warnhinweis des Fensters.

**Parameter**

| | |
|---|---|
| *text* | Warnhinweis, der angezeigt werden soll |

##### 5.23.2.2 void update ( Observable *o,* Object *arg* )

Wird durch notify() im ClientModel aufgerufen.

Je nach dem in arg übergebenen Befehl wird ein Update des Fensters ausgeführt oder eine Fehlermeldung angezeigt.

**Parameter**

| | |
|---|---|
| *o* | erwartet ein Objekt von der Klasse ClientModel |
| *arg* | erwartet: openWarning |

## 5.24 ViewNotification Enum-Referenz

**Öffentliche Attribute**

- **moveAcknowledged**
- **chooseCardsSuccessful**
- **InputNumberSuccessful**
- **chooseItemSuccessful**
- **playerListUpdate**
- **gameListUpdate**
- **chatMessage**
- **loginSuccesful**
- **joinGameSuccesful**
- **gameStarted**
- **passwordAccepted**
- **playedCardsUpdate**
- **otherDataUpdate**
- **windowChangeForced**
- **openChooseCards**
- **openChooseItem**
- **openInputNumber**
- **openWarning**
- **showScore**

#### 5.24.1 Ausführliche Beschreibung

Enum, das vom ClientModel ueber notify an seine Observer geschickt wird, um mitzuteilen, welche Veraenderung stattgefunden hat.

## 5.25 ComBeenKicked Klassenreferenz

Abgeleitet von ComObject und Serializable.

### Öffentliche Methoden

- ComBeenKicked (String message)
- String getMessage ()
- void process (ClientModel model)
- void process (Player player, Server server)

### Private Attribute

- String **message**

### 5.25.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Die Nachricht wird an einen Spieler gesendet, wenn er aus einem Spiel erntfernt wurde. Dies geschieht, wenn ein Spieler ein Spiel verlaesst oder wenn der Spielleiter das Wartefenster verlaesst.

### 5.25.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.25.2.1 ComBeenKicked ( String *message* )

Dies ist der Kontruktor fuer eine neue ComBeenKicked-Nachricht.

**Parameter**

| | |
|---|---|
| *message* | ist die Nachricht. |

### 5.25.3 Dokumentation der Elementfunktionen

#### 5.25.3.1 String getMessage ( )

Diese Methode liefert die Nachricht, die an den Spieler gesendet wird, wenn er entfernt wird.

**Rückgabe**

die Nachricht.

#### 5.25.3.2 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.25.3.3 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.26 ComChatMessage Klassenreferenz

Abgeleitet von ComObject und Serializable.

**Öffentliche Methoden**

- ComChatMessage (String message)
- String getChatMessage ()
- void process (ClientModel model)
- void process (Player player, Server server)

**Private Attribute**

- String **chatMessage**

### 5.26.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie enthaelt eine Chatnachricht in Form eines Strings.

### 5.26.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.26.2.1 ComChatMessage ( String *message* )

Dies ist der Kontruktor fuer eine neue ComChatMessage-Nachricht.

**Parameter**

| | |
|---|---|
| *message* | ist die Chatnachricht, die versendet wird. |

### 5.26.3 Dokumentation der Elementfunktionen

#### 5.26.3.1 String getChatMessage ( )

Hier kann die versendete Nachricht von anderen Klassen ausgelesen werden.

**Rückgabe**

die Chatnachricht, die versendet wurde.

#### 5.26.3.2 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.26.3.3 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.27 ComClientLeave Klassenreferenz

Abgeleitet von ComObject und Serializable.

**Öffentliche Methoden**

- ComClientLeave ()
- void process (ClientModel model)
- void process (Player player, Server server)

### 5.27.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie wird zur Benachrichtigung gesendet, wenn ein Spieler ins naechste Fenster moechte und aus dem alten entfernt werden soll.

### 5.27.2 Dokumentation der Elementfunktionen

#### 5.27.2.1 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.27.2.2 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |

| | |
|---|---|
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.28 ComClientQuit Klassenreferenz

Abgeleitet von ComObject und Serializable.

### Öffentliche Methoden

- ComClientQuit ()
- void process (ClientModel model)
- void process (Player player, Server server)

### 5.28.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Die Nachricht wird verschickt, wenn der Spieler ein Fenster schlieÃŸt.

### 5.28.2 Dokumentation der Elementfunktionen

#### 5.28.2.1 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.28.2.2 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.29 ComCreateGameRequest Klassenreferenz

Abgeleitet von ComObject und Serializable.

### Öffentliche Methoden

- ComCreateGameRequest (String name, Enum ruleset, boolean hasPassword, String password)
- String getGameName ()
- Enum getRuleset ()
- boolean hasPassword ()
- String getPassword ()

- void process (ClientModel model)
- void process (Player player, Server server)

#### Private Attribute

- String **gameName**
- Enum **ruleset**
- boolean **hasPassword**
- String **password**

### 5.29.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Diese Nachricht wird versendet, wenn ein neues Spiel erstellt werden soll.

### 5.29.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.29.2.1 ComCreateGameRequest ( String *name,* Enum *ruleset,* boolean *hasPassword,* String *password* )

Dies ist der Kontruktor fuer eine neue ComCreateGameRequest-Nachricht.

Wurde kein Passwort gesetzt, bleibt dieses leer.

**Parameter**

| | |
|---|---|
| *name* | ist der Name des Spiels. |
| *ruleset* | ist die der Spieltyp, der erstellt werden soll. |
| *hasPassword* | sagt, ob ein Passwort gesetzt wurde. |
| *password* | ist das Passwort, das gesetzt wurde. |

Benutzt ComCreateGameRequest.hasPassword().

### 5.29.3 Dokumentation der Elementfunktionen

#### 5.29.3.1 String getGameName ( )

Diese Methode gibt den Namen des Spiels zurueck.

**Rückgabe**

den Spielnamen.

#### 5.29.3.2 Enum getRuleset ( )

Diese Methode gibt das Regelwerk zurueck, das benutzt werden soll.

**Rückgabe**

das Regelwerk, welches benutzt wird.

#### 5.29.3.3 boolean hasPassword ( )

Diese Methode gibt an, ob eine Passwort fuer ein Spiel gesetzt wurde.

**Rückgabe**

ob es ein Passwort gibt.

Wird benutzt von ComCreateGameRequest.ComCreateGameRequest().

#### 5.29.3.4 String getPassword ( )

Gibt das Passwort zurueck.

Sollte keines gesetzt sein, wird null zurueck gegeben.

**Rückgabe**

das Passwort.

#### 5.29.3.5 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.29.3.6 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.30 ComGameEnd Klassenreferenz

Abgeleitet von ComObject und Serializable.

**Öffentliche Methoden**

- ComGameEnd (String winner, Map< String, Integer > points)
- String getWinner ()
- Map< String, Integer > getPoints ()
- void process (ClientModel model)
- void process (Player player, Server server)

**Private Attribute**

- String **winner**
- Map< String, Integer > **points**

### 5.30.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie liefert den Gewinner eines Spiels und eine Auflistung der Spieler mit ihren erspielten Punkten und wird versendet, wenn ein Spiel zu Ende ist oder eine Runde. In diesem Fall wird der Gewinner einfach leer gelassen, da es noch keinen gibt.

#### 5.30.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.30.2.1 ComGameEnd ( String *winner,* Map< String, Integer > *points* )

Dies ist der Kontruktor fuer eine neue ComGameEnd-Nachricht.

**Parameter**

| | |
|---|---|
| *winner* | ist der Gewinner des Spiels. |
| *points* | ist eine Auflistung der Spieler mit ihren Punkten. |

#### 5.30.3 Dokumentation der Elementfunktionen

##### 5.30.3.1 String getWinner ( )

Diese Methode gibt den Gewinner zurueck.

**Rückgabe**

den Gewinner.

##### 5.30.3.2 Map<String, Integer> getPoints ( )

Diese Methode gibt die Auflistung der Spieler mit ihren zugehoerigen Punkten zurueck.

**Rückgabe**

die Liste der Spieler mit ihren Punkten.

##### 5.30.3.3 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

##### 5.30.3.4 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.31 ComInitGameLobby Klassenreferenz

Abgeleitet von ComObject und Serializable.

**Öffentliche Methoden**

- ComInitGameLobby (List< String > playerList)
- List< String > getPlayerList ()
- void process (ClientModel model)
- void process (Player player, Server server)

**Private Attribute**

- List< String > **playerList**

#### 5.31.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie liefert die Liste der Spieler, die sich bereits beim Betreten des Wartefensters darin befinden.

#### 5.31.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.31.2.1 ComInitGameLobby ( List< String > *playerList* )

Dies ist der Kontruktor fuer eine neue ComInitGameLobby-Nachricht.

**Parameter**

| | |
|---|---|
| *playerList* | ist die Liste aller Player, die sich im Wartefenster befinden. |

#### 5.31.3 Dokumentation der Elementfunktionen

##### 5.31.3.1 List<String> getPlayerList ( )

Diese Methode liefert die Liste der Player, die sich beim Hinzufuegen eines weiteren Spielers bereits im Wartefenster befinden.

**Rückgabe**

die Liste der Spieler, die im Wartefenster sind.

##### 5.31.3.2 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

##### 5.31.3.3 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

### 5.32 ComInitLobby Klassenreferenz

Abgeleitet von ComObject und Serializable.

#### Öffentliche Methoden

- ComInitLobby (List< String > playerList, Set gameList)
- List< String > getPlayerList ()
- Set< GameServerRepresentation > getGameList ()
- void process (ClientModel model)
- void process (Player player, Server server)

#### Private Attribute

- List< String > **playerList**
- Set< GameServerRepresentation > **gameList**

### 5.32.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie synchronisiert den Client mit der Lobby, wenn er sich mit dem Server verbindet oder nach einem Spiel in die Lobby zurueckkehrt. Dazu enthaelt sie sowohl die playerList, als auch die gameList.

### 5.32.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.32.2.1 ComInitLobby ( List< String > *playerList,* Set *gameList* )

Dies ist der Kontruktor fuer eine neue ComInitLobby-Nachricht.

**Parameter**

| | |
|---|---|
| *playerList* | ist die Liste der Spieler, die sich in der Lobby befinden. |
| *gameList* | ist die Liste der Spiele, die existieren und in der Lobby angezeigt werden. |

### 5.32.3 Dokumentation der Elementfunktionen

#### 5.32.3.1 List<String> getPlayerList ( )

Die Methode liefert die Liste aller Spieler, die in der Lobby sind.

**Rückgabe**

die Liste der Spieler.

#### 5.32.3.2 Set<GameServerRepresentation> getGameList ( )

Diese Methode liefert eine Liste aller Spiele, die erstellt wurden, damit sie in der Lobby angezeigt werden koennen.

**Rückgabe**

die Liste der Spiele.

#### 5.32.3.3 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.32.3.4 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.33 ComJoinRequest Klassenreferenz

Abgeleitet von ComObject und Serializable.

**Öffentliche Methoden**

- ComJoinRequest (String gameMasterName, String password)
- String getGameMasterName ()
- void process (ClientModel model)
- void process (Player player, Server server)

**Private Attribute**

- String gameMasterName
- String **password**

### 5.33.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie ist eine Nachricht, die an den Server gesendet wird, wenn der Spieler einem bestimmten Spiel beitreten will. Dazu enthaelt sie den Namen des Spielleiters als String und ein Passwort, falls dieses von Spielleiter gesetzt wurde.

### 5.33.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.33.2.1 ComJoinRequest ( String *gameMasterName,* String *password* )

Dies ist der Kontruktor fuer eine neue ConJoinRequest-Nachricht.

Ein Spiel kann durch den eindeutigen Namen der Spielleiters identifiziert werden.

**Parameter**

| | |
|---|---|
| *gameMaster-Name* | ist der Name der Spielleiters. |

| | |
|---|---|
| *password* | fuer das Spiel. |

Benutzt ComJoinRequest.gameMasterName.

#### 5.33.3 Dokumentation der Elementfunktionen

##### 5.33.3.1 String getGameMasterName ( )

Diese Methode gibt den Namen des Spielleiters zurueck.

Dieser ist eindeutig, so kann ein bestimmtes Spiel identifiziert werden.

**Rückgabe**

der Name des Spielleiters.

Benutzt ComJoinRequest.gameMasterName.

##### 5.33.3.2 void process ( **ClientModel** *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

##### 5.33.3.3 void process ( **Player** *player,* **Server** *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

#### 5.33.4 Dokumentation der Datenelemente

##### 5.33.4.1 String gameMasterName [private]

Der Name der Spielleiters muss enthalten sein um ein Spiel zuzuornen.

Der Spielname ist nicht eindeutig, aber der Spielleiter schon. Somit kann jedes Spiel mit Hilfe des Spielleiters identifiziert werden.

Wird benutzt von ComJoinRequest.ComJoinRequest() und ComJoinRequest.getGameMasterName().

### 5.34 ComKickPlayerRequest Klassenreferenz

Abgeleitet von ComObject und Serializable.

**Öffentliche Methoden**

- ComKickPlayerRequest (String playerName)

- String getPlayerName ()
- void process (ClientModel model)
- void process (Player player, Server server)

**Private Attribute**

- String playerName

#### 5.34.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie ist eine Nachricht an den Server, die angibt einen Spieler vom Spiel zu entfernen. Dazu enthaelt sie einen String, der den Namen des Spielers enthaelt.

#### 5.34.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.34.2.1 ComKickPlayerRequest ( String *playerName* )

Dies ist der Kontruktor fuer eine neue ComKickPlayerRequest-Nachricht.

Diese enthaelt den Namen des Spielers, der aus den Spiel geloescht werden soll.

**Parameter**

| | |
|---|---|
| *playerName* | ist der Name des Spielers. |

Benutzt ComKickPlayerRequest.playerName.

#### 5.34.3 Dokumentation der Elementfunktionen

##### 5.34.3.1 String getPlayerName ( )

Diese Methode liefert den Namen des Spielers, der aus dem Spiel entfernt werden soll.

**Rückgabe**

den Spielernamen.

Benutzt ComKickPlayerRequest.playerName.

##### 5.34.3.2 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

##### 5.34.3.3 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.35 ComLobbyUpdateGamelist Klassenreferenz

Abgeleitet von ComObject und Serializable.

### Öffentliche Methoden

- ComLobbyUpdateGamelist (boolean removeFlag, GameServerRepresentation gameServer)
- boolean isRemoveFlag ()
- GameServerRepresentation getGameServer ()
- void process (ClientModel model)
- void process (Player player, Server server)

### Private Attribute

- boolean **removeFlag**
- GameServerRepresentation **gameServer**

### 5.35.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie aktualisiert die Gameliste in der Lobby. Dazu enthaelt sie den GameServer und ein RemoveFlag.

### 5.35.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.35.2.1 ComLobbyUpdateGamelist ( boolean *removeFlag,* GameServerRepresentation *gameServer* )

Dies ist der Kontruktor fuer eine neue ComLobbyUpdateGamelist-Nachricht.

**Parameter**

| | |
|---|---|
| *removeFlag* | zeigt an, ob das Spiel geloescht werden soll. |
| *gameServer* | ist das Spiel. |

### 5.35.3 Dokumentation der Elementfunktionen

#### 5.35.3.1 boolean isRemoveFlag ( )

Diese Methode liefert, ob ein Spiel geloescht werden soll oder nicht.

**Rückgabe**

ob das Spiel gelÃ¶scht wird.

#### 5.35.3.2 GameServerRepresentation getGameServer ( )

Diese Methode liefert das Spiel, das geupdated werden soll.

**Rückgabe**

das Spiel.

#### 5.35.3.3 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.35.3.4 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.36 ComLoginRequest Klassenreferenz

Abgeleitet von ComObject und Serializable.

**Öffentliche Methoden**

- ComLoginRequest (String name)
- String getPlayerName ()
- void process (ClientModel model)
- void process (Player player, Server server)

**Private Attribute**

- String **playerName**

### 5.36.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie ist eine Nachricht, die beim Login an den Server gesendet wird. Dazu enthaelt sie den Namen des Spielers, der sich einloggen moechte.

### 5.36.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.36.2.1 ComLoginRequest ( String *name* )

Dies ist der Kontruktor fuer eine neue ComLoginRequest-Nachricht.

**Parameter**

| | |
|---|---|
| *name* | ist der Name des Spielers, des sich einloggen moechte. |

### 5.36.3 Dokumentation der Elementfunktionen

#### 5.36.3.1 String getPlayerName ( )

Diese Methode liefert den Namen des Spielers, des sich einloggen moechte.

Dieser muss auf Eindeutigkeit geprueft werden.

**Rückgabe**

den Spielernamen.

#### 5.36.3.2 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.36.3.3 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.37 ComObject Schnittstellenreferenz

Basisklasse für ComBeenKicked, ComChatMessage, ComClientLeave, ComClientQuit, ComCreateGameRequest, ComGameEnd, ComInitGameLobby, ComInitLobby, ComJoinRequest, ComKickPlayerRequest, ComLobbyUpdateGamelist, ComLoginRequest, ComRuleset, ComServerAcknowledgement, ComStartGame, ComUpdatePlayerlist und ComWarning.

**Öffentliche Methoden**

- void process (ClientModel model)
- void process (Player player, Server server)

### 5.37.1 Ausführliche Beschreibung

Die Klasse ComObject ist ein Interface, welches ein Objekt darstellt, das zur Kommunikation genutzt werden kann. Spezielle ComObject Klassen implementieren diese grundlegenden Klasse.

#### 5.37.2 Dokumentation der Elementfunktionen

##### 5.37.2.1 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert in ComCreateGameRequest, ComInitLobby, ComGameEnd, ComJoinRequest, ComUpdatePlayerlist, ComLobbyUpdateGamelist, ComKickPlayerRequest, ComInitGameLobby, ComBeenKicked, ComRuleset, ComWarning, ComLoginRequest, ComChatMessage, ComClientLeave, ComClientQuit, ComStartGame und ComServerAcknowledgement.

#### 5.37.2.2 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert in ComCreateGameRequest, ComInitLobby, ComGameEnd, ComJoinRequest, ComUpdatePlayerlist, ComLobbyUpdateGamelist, ComKickPlayerRequest, ComInitGameLobby, ComBeenKicked, ComRuleset, ComWarning, ComLoginRequest, ComChatMessage, ComClientLeave, ComClientQuit, ComStartGame und ComServerAcknowledgement.

## 5.38 ComRuleset Klassenreferenz

Abgeleitet von ComObject und Serializable.

**Öffentliche Methoden**

- ComRuleset (RulesetMessage rulesetMessage)
- RulesetMessage getRulesetMessage ()
- void process (ClientModel model)
- void process (Player player, Server server)

**Private Attribute**

- RulesetMessage **rulesetMessage**

### 5.38.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie ist die grundlegende Nachricht eines Regelwerkaufrufes und enthaelt eine verfeinerte Nachricht mit weiteren Informationen, die RulesetMessage.

### 5.38.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.38.2.1 ComRuleset ( RulesetMessage *rulesetMessage* )

Dies ist der Kontruktor fuer eine neue ComRuleset-Nachricht.

**Parameter**

| | |
|---|---|
| *rulesetMessage* | ist eine Nachricht, die ans Ruleset gesendet werden soll. |

#### 5.38.3 Dokumentation der Elementfunktionen

##### 5.38.3.1 RulesetMessage getRulesetMessage ( )

Diese Methode gibt die Nachricht zurueck, die ans Ruleset gesendet werden soll.

**Rückgabe**

die Nachricht.

##### 5.38.3.2 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

##### 5.38.3.3 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

### 5.39 ComServerAcknowledgement Klassenreferenz

Abgeleitet von ComObject und Serializable.

**Öffentliche Methoden**

- void process (ClientModel model)
- void process (Player player, Server server)

#### 5.39.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Diese Nachricht wird vom Server als Bestaetigung gesendet.

#### 5.39.2 Dokumentation der Elementfunktionen

##### 5.39.2.1 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.39.2.2 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.40 ComStartGame Klassenreferenz

Abgeleitet von ComObject und Serializable.

### Öffentliche Methoden

- ComStartGame ()
- void process (ClientModel model)
- void process (Player player, Server server)

### 5.40.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie wird versendet, wenn ein Spiel gestartet werden soll.

### 5.40.2 Dokumentation der Elementfunktionen

#### 5.40.2.1 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.40.2.2 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.41 ComUpdatePlayerlist Klassenreferenz

Abgeleitet von ComObject und Serializable.

### Öffentliche Methoden

- ComUpdatePlayerlist (String playerName, boolean removeFlag)
- String getPlayerName ()
- boolean isRemoveFlag ()
- void process (ClientModel model)
- void process (Player player, Server server)

### Private Attribute

- String **playerName**
- boolean **removeFlag**

### 5.41.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie sendet eine Nachricht zum Update der Playerliste in der Lobby und Spiellobby. Dazu enthaelt sie den Player und ein removeFlag.

### 5.41.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.41.2.1 ComUpdatePlayerlist ( String *playerName,* boolean *removeFlag* )

Dies ist der Kontruktor fuer eine neue ComUpdatePlayerlist-Nachricht.

Diese beinhaltet den Namen des Spielers und die Angabe ob er geloescht werden soll.

**Parameter**

| | |
|---|---|
| *playerName* | ist der Name der Spielers. |
| *removeFlag* | zeigt, ob der Spieler geloescht werden soll. |

### 5.41.3 Dokumentation der Elementfunktionen

#### 5.41.3.1 String getPlayerName ( )

Diese Methode gibt den Namen des Spielers zurueck.

**Rückgabe**

den Spielernamen.

#### 5.41.3.2 boolean isRemoveFlag ( )

Diese Methode gibt zurueck, ob der Spieler aus der Liste geloescht werden soll oder nicht.

**Rückgabe**

ob der Spieler geloescht werden soll.

#### 5.41.3.3 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.41.3.4 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.42 ComWarning Klassenreferenz

Abgeleitet von ComObject und Serializable.

**Öffentliche Methoden**

- ComWarning (String warning)
- String getWarning ()
- void process (ClientModel model)
- void process (Player player, Server server)

**Private Attribute**

- String **warning**

### 5.42.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie soll dem Spieler eine Mitteilung senden und so ueber ein Fehlerevent informieren.

### 5.42.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.42.2.1 ComWarning ( String *warning* )

Dies ist der Konstruktor einer neuen ComWarning-Nachricht.

Er enthaelt eine Warnung an den Spieler, wenn ein Fehler passiert.

**Parameter**

| | |
|---|---|
| *warning* | ist die Warnung, die der Spieler erhaelt. |

### 5.42.3 Dokumentation der Elementfunktionen

#### 5.42.3.1 String getWarning ( )

Diese Methode gibt die Nachricht zurueck, die dem Spieler den Fehler mitteilt.

**Rückgabe**

die Warnnachricht.

#### 5.42.3.2 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *model* | ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert ComObject.

#### 5.42.3.3 void process ( Player *player,* Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *player* | Der Client welcher den Aufruf startet. |
| *server* | Der Server an den sich das ComObjekt weitergibt. |

Implementiert ComObject.

## 5.43 MsgCard Klassenreferenz

Abgeleitet von RulesetMessage und Serializable.

**Öffentliche Methoden**

- MsgCard (Card card)
- Card getCard ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

**Private Attribute**

- Card **card**

### 5.43.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie beinhaltet die ausgespielte Karte eines Spielers.

### 5.43.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.43.2.1 MsgCard ( Card *card* )

Dies ist der Kontruktor fuer eine neue MsgCard-Nachricht.

Diese enthaelt die Information, welche Karte von einem Spieler gespielt wurde.

**Parameter**

| | |
|---|---|
| *card* | ist die Karte. |

### 5.43.3 Dokumentation der Elementfunktionen

#### 5.43.3.1 Card getCard ( )

Diese Methode gibt die ausgespielte Karte des Spielers zurueck.

**Rückgabe**

die Karte.

#### 5.43.3.2 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert RulesetMessage.

#### 5.43.3.3 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert RulesetMessage.

## 5.44 MsgCardRequest Klassenreferenz

Abgeleitet von RulesetMessage und Serializable.

**Öffentliche Methoden**

- MsgCardRequest ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

### 5.44.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Diese Nachricht wird von Server gesendet, um einem Spieler mitzuteilen, dass er das Spielen einer Karte erwartet.

### 5.44.2 Dokumentation der Elementfunktionen

#### 5.44.2.1 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert RulesetMessage.

#### 5.44.2.2 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert RulesetMessage.

## 5.45 MsgGameEnd Klassenreferenz

Abgeleitet von RulesetMessage und Serializable.

**Öffentliche Methoden**

- MsgGameEnd (String name)
- String getWinnerName ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

**Private Attribute**

- String **winnerName**

### 5.45.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie signalisiert dem ClientRuleset, dass das Spiel zu Ende ist.

### 5.45.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.45.2.1 MsgGameEnd ( String *name* )

Dies ist der Kontruktor fuer eine neue MsgGameEnd-Nachricht.

**Parameter**

| | |
|---|---|
| *name* | ist der Name des Gewinners. |

### 5.45.3 Dokumentation der Elementfunktionen

#### 5.45.3.1 String getWinnerName ( )

Diese Methode liefert den Namen des Gewinners eines Spiels.

**Rückgabe**

den Gewinnernamen.

#### 5.45.3.2 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert RulesetMessage.

#### 5.45.3.3 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert RulesetMessage.

## 5.46 MsgMultiCards Klassenreferenz

Abgeleitet von RulesetMessage und Serializable.

**Öffentliche Methoden**

- MsgMultiCards (Set cardList)
- Set< Card > getCardList ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

**Private Attribute**

- Set< Card > **cardList**

### 5.46.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie liefert mehrere Karten zum Tausch fuer das Regelwerk Hearts.

### 5.46.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.46.2.1 MsgMultiCards ( Set *cardList* )

Dies ist der Kontruktor fuer eine neue MsgMultiCards-Nachricht.

**Parameter**

| | |
|---|---|
| *cardList* | ist die Liste der ausgewaehlten Karten. |

### 5.46.3 Dokumentation der Elementfunktionen

#### 5.46.3.1 Set<Card> getCardList ( )

Gibt die Liste der gewaehlten Karten zurÃ¼ck.

**Rückgabe**

die Liste der Karten.

#### 5.46.3.2 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert RulesetMessage.

#### 5.46.3.3 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert RulesetMessage.

## 5.47 MsgMultiCardsRequest Klassenreferenz

Abgeleitet von RulesetMessage und Serializable.

**Öffentliche Methoden**

- MsgMultiCardsRequest (int count)
- int getCount ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

**Private Attribute**

- int count

### 5.47.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Diese Nachricht wird gesendet, wenn die Auswahl mehrerer Karten vom Spieler gefordert werden soll.

#### 5.47.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.47.2.1 MsgMultiCardsRequest ( int *count* )

Dies ist der Kontruktor fuer eine neue MsgMultipleCardsRequest-Nachricht.

**Parameter**

| | |
|---|---|
| *count* | ist die erwartete Anzahl an Karten. |

Benutzt MsgMultiCardsRequest.count.

#### 5.47.3 Dokumentation der Elementfunktionen

##### 5.47.3.1 int getCount ( )

Diese Methode gibt die Anzahl der Karten zurueck, die der Server vom Spieler erwartet.

**Rückgabe**

die Anzahl der Karten.

Benutzt MsgMultiCardsRequest.count.

##### 5.47.3.2 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert RulesetMessage.

##### 5.47.3.3 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert RulesetMessage.

### 5.48 MsgNumber Klassenreferenz

Abgeleitet von RulesetMessage und Serializable.

**Öffentliche Methoden**

- MsgNumber (int number)
- int getNumber ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

#### Private Attribute

- int **number**

### 5.48.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie enthaelt eine Zahl, die ein Spieler zuvor ausgewaehlt hat.

### 5.48.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.48.2.1 MsgNumber ( int *number* )

Dies ist der Kontruktor fuer eine neue MsgNumber-Nachricht.

**Parameter**

| | |
|---|---|
| *number* | ist eine Eingabe eines Spielers. |

### 5.48.3 Dokumentation der Elementfunktionen

#### 5.48.3.1 int getNumber ( )

Diese Methode liefert die Eingabe eines Spielers.

**Rückgabe**

eine Zahl, die Eingabe des Spielers.

#### 5.48.3.2 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert RulesetMessage.

#### 5.48.3.3 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert RulesetMessage.

## 5.49 MsgNumberRequest Klassenreferenz

Abgeleitet von RulesetMessage und Serializable.

**Öffentliche Methoden**

- MsgNumberRequest ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

#### 5.49.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie Wird gesendet, wenn die Eingabe einer Zahl gefordert werden soll.

#### 5.49.2 Dokumentation der Elementfunktionen

##### 5.49.2.1 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert RulesetMessage.

##### 5.49.2.2 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert RulesetMessage.

### 5.50 MsgSelection Klassenreferenz

Abgeleitet von RulesetMessage und Serializable.

**Öffentliche Methoden**

- MsgSelection (Colour selection)
- Colour getSelection ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

**Private Attribute**

- Colour **selection**

#### 5.50.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Diese Nachricht enthaelt eine Kartenfarbe, die der Spieler zuvor ausgewaehlt hat.

#### 5.50.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.50.2.1 MsgSelection ( Colour *selection* )

Dies ist der Kontruktor fuer eine neue MsgSelection-Nachricht.

**Parameter**

| | |
|---|---|
| *selection* | ist die Farbe der Karte, die der Spieler gewaehlt hat. |

#### 5.50.3 Dokumentation der Elementfunktionen

##### 5.50.3.1 Colour getSelection ( )

Diese Methode gibt die Farbe zurueck, die der Spieler gewaehlt hat.

**Rückgabe**

die gewaehlte Farbe.

##### 5.50.3.2 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert RulesetMessage.

##### 5.50.3.3 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert RulesetMessage.

## 5.51 MsgSelectionRequest Klassenreferenz

Abgeleitet von RulesetMessage und Serializable.

**Öffentliche Methoden**

- MsgSelectionRequest ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

#### 5.51.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Diese Nachricht sendet der Server an einen Spieler, wenn er eine Farbauswahl von diesem erwartet.

#### 5.51.2 Dokumentation der Elementfunktionen

##### 5.51.2.1 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert RulesetMessage.

##### 5.51.2.2 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert RulesetMessage.

### 5.52 MsgUser Klassenreferenz

Abgeleitet von RulesetMessage und Serializable.

#### Öffentliche Methoden

- MsgUser (GameClientUpdate gameClientUpdate)
- GameClientUpdate getGameClientUpdate ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

#### Private Attribute

- GameClientUpdate **gameClientUpdate**

#### 5.52.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie wird dem Client gesendet, um dem Client-Ruleset den aktuellen Spielzustand in Form eines GameClientUpdate zu uebermitteln.

#### 5.52.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.52.2.1 MsgUser ( GameClientUpdate *gameClientUpdate* )

Dies ist der Konstruktor einer neuen MsgUser-Nachricht.

**Parameter**

| | |
|---|---|
| *gameClient-Update* | ist der aktuelle Spielstand. |

#### 5.52.3 Dokumentation der Elementfunktionen

##### 5.52.3.1 GameClientUpdate getGameClientUpdate ( )

Diese Methode liefert den den aktuellen Spielzustand, der fuer ein Update benoetigt wird.

**Rückgabe**

den aktuellen Spielzustand.

##### 5.52.3.2 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert RulesetMessage.

##### 5.52.3.3 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert RulesetMessage.

## 5.53 RulesetMessage Schnittstellenreferenz

Basisklasse für MsgCard, MsgCardRequest, MsgGameEnd, MsgMultiCards, MsgMultiCardsRequest, MsgNumber, MsgNumberRequest, MsgSelection, MsgSelectionRequest und MsgUser.

**Öffentliche Methoden**

- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

#### 5.53.1 Ausführliche Beschreibung

Dieses Interface ist eine Verfeinerung der ComRuleset-Klasse. Es enthaelt Methoden, die von speziellen Ruleset-Messages implementiert werden mÃ¼ssen.

#### 5.53.2 Dokumentation der Elementfunktionen

#### 5.53.2.1 void visit ( ServerRuleset *serverRuleset,* String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *serverRuleset* | ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |
| *name* | ist der Name des Spielers. |

Implementiert in MsgMultiCardsRequest, MsgSelection, MsgCard, MsgUser, MsgGameEnd, MsgMultiCards, MsgNumber, MsgCardRequest, MsgNumberRequest und MsgSelectionRequest.

#### 5.53.2.2 void visit ( ClientRuleset *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

| | |
|---|---|
| *clientRuleset* | ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird. |

Implementiert in MsgMultiCardsRequest, MsgCard, MsgSelection, MsgUser, MsgGameEnd, MsgMultiCards, MsgNumber, MsgNumberRequest, MsgSelectionRequest und MsgCardRequest.

## 5.54 Card Schnittstellenreferenz

Basisklasse für HeartsCard und WizardCard.

### Öffentliche Methoden

- int getValue ()
- Colour getColour ()

### 5.54.1 Ausführliche Beschreibung

Dieses Interface modelliert eine Spielkarte

### 5.54.2 Dokumentation der Elementfunktionen

#### 5.54.2.1 int getValue ( )

Gibt den Wert der Karte zurück.

**Rückgabe**

Der Wert der Karte

Implementiert in WizardCard und HeartsCard.

#### 5.54.2.2 Colour getColour ( )

Gibt die Farbe der Karte zurück.

**Rückgabe**

Die Farbe der Karte

Implementiert in WizardCard und HeartsCard.

## 5.55 ClientHearts Klassenreferenz

Abgeleitet von ClientRuleset.

### Öffentliche Methoden

- ClientHearts (ClientModel client)
- boolean isValidMove (Card card)
- void resolveMessage (MsgMultiCardsRequest msgMultiCardsRequest)
- boolean areValidChoosenCards (Set< Card > cards)

### Statische, private Attribute

- static final int MIN_PLAYERS = 4
- static final int MAX_PLAYERS = 4
- static final RulesetType RULESET = RulesetType.Hearts

### Weitere Geerbte Elemente

### 5.55.1 Ausführliche Beschreibung

Diese Klasse bildet das Regelwerk für den Clientmodel bei einer Partie Hearts

### 5.55.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.55.2.1 ClientHearts ( ClientModel *client* )

Erzeugt ein ClientHearts.

**Parameter**

| | |
|---|---|
| *client* | Das Model auf dem gespielt wird |

Benutzt RulesetType.Hearts, ClientHearts.MAX_PLAYERS und ClientHearts.MIN_PLAYERS.

### 5.55.3 Dokumentation der Elementfunktionen

#### 5.55.3.1 boolean isValidMove ( Card *card* ) [virtual]

Prueft ob ein gemachter Zug in einem Spiel gueltig war.

**Parameter**

| | |
|---|---|
| *card* | Die Karte |

**Rückgabe**

true falls die Karte gueltig ist, false wenn nicht

Implementiert ClientRuleset.

#### 5.55.3.2 void resolveMessage ( MsgMultiCardsRequest *msgMultiCardsRequest* )

Verarbeitet die RulesetMessage dass der Server von dem Spieler verlangt mehrere Karten anzugeben.

**Parameter**

| | |
|---|---|
| *msgMultiCards-Request* | Die Nachricht vom Server |

#### 5.55.3.3 boolean areValidChoosenCards ( Set< Card > *cards* )

Gibt zuueck ob die Karten die der Client tauschen will, gueltig sind.

**Parameter**

| | |
|---|---|
| *cards* | Die zu tauschenden Karten |

**Rückgabe**

true wenn Karten valide sind, false wenn nicht

## 5.56 ClientRuleset Klassenreferenz

Basisklasse für ClientHearts und ClientWizard.

**Öffentliche Methoden**

- ClientRuleset (RulesetType ruleset, int minPlayers, int maxPlayers, ClientModel client)
- RulesetType getRulesetType ()
- int getMinPlayers ()
- int getMaxPlayers ()
- GamePhase getGamePhase ()
- List< Card > getOwnHand ()
- OtherData getOwnData ()
- OtherData getOtherPlayerData (String player)
- PlayerState getCurrentPlayer ()
- Card getTrumpCard ()
- void resolveMessage (RulesetMessage message)
- void resolveMessage (MsgUser clientUpdate)
- void resolveMessage (MsgCardRequest msgCardRequest)
- abstract boolean isValidMove (Card card)

**Geschützte Methoden**

- void send (RulesetMessage message)

**Private Attribute**

- ClientModel client
- GameClientUpdate gameState
- final RulesetType RULESET
- final int MIN_PLAYERS
- final int MAX_PLAYERS
- GamePhase gamePhase

### 5.56.1 Ausführliche Beschreibung

ClientRuleset ist eine abstrakte Klasse und wird zur Regelvorauswertung im Client verwendet. Dazu benutzt es die isValidMove() Methode. Des Weiteren kann es vom ClientModel erhaltene RulesetMessages mit der resolveMessage() Methode behandeln und neue RulesetMessages senden.

#### 5.56.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.56.2.1 ClientRuleset ( RulesetType *ruleset,* int *minPlayers,* int *maxPlayers,* ClientModel *client* )

Erstellt eine ClientRuleset Klasse.

**Parameter**

| | |
|---|---|
| *ruleset* | Das Ruleset zum Spiel |
| *minPlayers* | Die minimale Spieleranzahl |
| *maxPlayers* | Die maximale Spieleranzahl |
| *client* | Das ClientModel auf dem gespielt wird |

Benutzt ClientRuleset.client, ClientRuleset.gamePhase, ClientRuleset.MAX_PLAYERS, ClientRuleset.MIN_PLAYERS, ClientRuleset.RULESET und GamePhase.Start.

#### 5.56.3 Dokumentation der Elementfunktionen

##### 5.56.3.1 RulesetType getRulesetType ( )

Gibt den Typ des Regelwerks zurueck.

**Rückgabe**

Der Typ vom Regelwerk

Benutzt ClientRuleset.RULESET.

##### 5.56.3.2 int getMinPlayers ( )

Gibt die Mindestanzahl an Spielern zurueck für dieses Spiel.

**Rückgabe**

Die Mindestanzahl an Spielern

Benutzt ClientRuleset.MIN_PLAYERS.

##### 5.56.3.3 int getMaxPlayers ( )

Gibt die Maximale Anzahl an Spielern zurueck.

**Rückgabe**

Die maximale Anzahl an Spielern

Benutzt ClientRuleset.MAX_PLAYERS.

##### 5.56.3.4 GamePhase getGamePhase ( )

Gibt die momentane Spielphase zurück.

**Rückgabe**

gamePhase Die Spielphase

Benutzt ClientRuleset.gamePhase.

#### 5.56.3.5 List<Card> getOwnHand ( )

Gibt die eigenen Handkarten zurueck.

**Rückgabe**

Liste von Karten

#### 5.56.3.6 OtherData getOwnData ( )

Gibt die OtherData des Models zurueck.

**Rückgabe**

Die Otherdata des Models

#### 5.56.3.7 OtherData getOtherPlayerData ( String *player* )

Holt die OtherData eines anderen Spielers.

**Parameter**

| | |
|---|---|
| *Der* | Spielername |

**Rückgabe**

otherPlayerData Die OtherData

#### 5.56.3.8 PlayerState getCurrentPlayer ( )

Gibt den Spieler der momentan am Zug ist zurueck.

**Rückgabe**

Der momentane Spieler

#### 5.56.3.9 Card getTrumpCard ( )

Holt die aufgedeckte Trumpfkarte.

**Rückgabe**

Eine Karte

#### 5.56.3.10 void resolveMessage ( RulesetMessage *message* )

Verarbeitet eine RulesetMessage vom Server.

**Parameter**

| | |
|---|---|
| *clientUpdate* | Die Nachricht vom Server |

#### 5.56.3.11 void resolveMessage ( MsgUser *clientUpdate* )

Verarbeitet die RulesetMessage dass der Server ein Spielupdate an den Client schickt.

**Parameter**

| | |
|---|---|
| *clientUpdate* | Die Nachricht vom Server |

#### 5.56.3.12 void resolveMessage ( MsgCardRequest *msgCardRequest* )

Verarbeitet die RulesetMessage dass der Server von dem Spieler verlangt eine Karte zu spielen.

**Parameter**

| | |
|---|---|
| *msgCard-Request* | Die Nachricht vom Server |

#### 5.56.3.13 void send ( RulesetMessage *message* ) [protected]

Ruft beim Model die send Methode auf und verschickt eine Rulesetmessage.

**Parameter**

| | |
|---|---|
| *message* | Die Nachricht |

#### 5.56.3.14 abstract boolean isValidMove ( Card *card* ) [pure virtual]

Prueft ob ein gemachter Zug in einem Spiel gueltig war.

**Parameter**

| | |
|---|---|
| *card* | Die Karte |

**Rückgabe**

true falls die Karte gueltig ist, false wenn nicht

Implementiert in ClientHearts und ClientWizard.

## 5.57 ClientWizard Klassenreferenz

Abgeleitet von ClientRuleset.

**Öffentliche Methoden**

- ClientWizard (ClientModel client)
- boolean isValidMove (Card card)
- void resolveMessage (MsgNumberRequest msgNumber)
- void resolveMessage (MsgSelectionRequest msgSelection)
- boolean isValidTrickNumber (int number)
- boolean isValidColour (Colour colour)

**Statische, private Attribute**

- static final int MIN_PLAYERS = 3
- static final int MAX_PLAYERS = 6
- static final RulesetType RULESET = RulesetType.Wizard

#### Weitere Geerbte Elemente

### 5.57.1 Ausführliche Beschreibung

Diese Klasse bildet das Regelwerk fuer den Client bei einer Partie Wizard

### 5.57.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.57.2.1 ClientWizard ( ClientModel *client* )

Erzeugt ein ClientWizard.

**Parameter**

| | |
|---|---|
| *client* | Das Model auf dem gespielt wird |

Benutzt ClientWizard.MAX_PLAYERS, ClientWizard.MIN_PLAYERS und ClientWizard.RULESET.

### 5.57.3 Dokumentation der Elementfunktionen

#### 5.57.3.1 boolean isValidMove ( Card *card* ) [virtual]

Prueft ob ein gemachter Zug in einem Spiel gueltig war.

**Parameter**

| | |
|---|---|
| *card* | Die Karte |

**Rückgabe**

true falls die Karte gueltig ist, false wenn nicht

Implementiert ClientRuleset.

#### 5.57.3.2 void resolveMessage ( MsgNumberRequest *msgNumber* )

Verarbeitet die RulesetMessage dass der Server von dem Spieler verlangt eine Stichanzahl anzugeben.

**Parameter**

| | |
|---|---|
| *msgNumber* | Die Nachricht vom Server |

#### 5.57.3.3 void resolveMessage ( MsgSelectionRequest *msgSelection* )

Verarbeitet die RulesetMessage dass der Server von dem Spieler verlangt eine Farbe auszuwählen.

**Parameter**

| | |
|---|---|
| *msgSelection* | Die Nachricht vom Server |

#### 5.57.3.4 boolean isValidTrickNumber ( int *number* )

Prüft ob die Anzahl der angesagten Stiche vom Spieler gültig sind.

**Parameter**

| | |
|---|---|
| *number* | Die Anzahl der angesagten Sticht |

**Rückgabe**

true falls die Anzahl der Stiche passen, false wenn nicht

#### 5.57.3.5 boolean isValidColour ( Colour *colour* )

Prüft ob die angesagte Trumpffarbe richtig.

**Parameter**

| | |
|---|---|
| *colour* | Die angesagte Trumpffarbe |

**Rückgabe**

true falls die Farbe in Ordnung ist, false wenn nicht

## 5.58 Colour Enum-Referenz

**Öffentliche Attribute**

- **NONE**
- **HEART**
- **CLUB**
- **SPADE**
- **DIAMOND**
- **BLUE**
- **RED**
- **YELLOW**

### 5.58.1 Ausführliche Beschreibung

Repräsentiert die Farbe einer Karte

## 5.59 GameClientUpdate Klassenreferenz

**Geschützte Methoden**

- GameClientUpdate (PlayerState playerState, Map< String, Card > discardPile, Map< String, OtherData > otherPlayerData, PlayerState currentPlayer, Card trumpCard)
- List< Card > getOwnHand ()
- Map< String, Card > getPlayedCards ()
- OtherData getOwnData ()
- OtherData getOtherPlayerData (String player)
- PlayerState getCurrentPlayer ()
- Card getTrumpCard ()

**Private Attribute**

- PlayerState playerState
- Map< String, Card > discardPile
- Map< String, OtherData > otherPlayerData
- PlayerState currentPlayer
- Card trumpCard

#### 5.59.1 Ausführliche Beschreibung

Das GameClientUpdate wird vom RuleSet ueber den GameServer an den Client geschickt und enthaelt alle Aenderungen des GameState, die für den Client relevant sind. Das waeren seine Spielhand, der Ablagestapel sowie die Otherdata von allen Spielern und die Trumpfkarte.

#### 5.59.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.59.2.1 GameClientUpdate ( PlayerState *playerState,* Map< String, Card > *discardPile,* Map< String, OtherData > *otherPlayerData,* PlayerState *currentPlayer,* Card *trumpCard* ) [protected]

Erstellt ein GameClientUpdate.

**Parameter**

| | |
|---|---|
| *playerState* | Der Spielerzustand des Client |
| *discardPile* | Der Ablagestapel |
| *otherPlayerData* | Die Daten der anderen Spieler |
| *currentPlayer* | Der momentan aktive Spieler |
| *trumpCard* | Die Trumpffarbe |

Benutzt GameClientUpdate.currentPlayer, GameClientUpdate.discardPile, GameClientUpdate.otherPlayerData, GameClientUpdate.playerState und GameClientUpdate.trumpCard.

#### 5.59.3 Dokumentation der Elementfunktionen

##### 5.59.3.1 List<Card> getOwnHand ( ) [protected]

Holt die Karten die der Client auf der Hand hat.

**Rückgabe**

ownHand Die Hand des Clients

##### 5.59.3.2 Map<String, Card> getPlayedCards ( ) [protected]

Holt die gespielten Karten auf dem Ablagestapel.

**Rückgabe**

discardPile Die gespielten Karten

Benutzt GameClientUpdate.discardPile.

##### 5.59.3.3 OtherData getOwnData ( ) [protected]

Holt die Otherdata des Client als String als Stringrepräsentation.

**Rückgabe**

ownData Die Otherdata des Clients

##### 5.59.3.4 OtherData getOtherPlayerData ( String *player* ) [protected]

Holt die OtherData eines anderen Spielers als Stringrepräsentation.

**Parameter**

| | |
|---|---|
| *player* | Der Name des Spielers |

**Rückgabe**

otherPlayerData Die OtherData der anderen Spieler

#### 5.59.3.5 PlayerState getCurrentPlayer ( ) [protected]

Gibt den Spieler der momentan am Zug ist zurück.

**Rückgabe**

Der momentane Spieler

Benutzt GameClientUpdate.currentPlayer.

#### 5.59.3.6 Card getTrumpCard ( ) [protected]

Holt die aufgedeckte Trumpfkarte.

**Rückgabe**

trumpCard Die Trumpfkarte

Benutzt GameClientUpdate.trumpCard.

## 5.60 GamePhase Enum-Referenz

**Öffentliche Attribute**

- Start
- Playing
- CardRequest
- MultipleCardRequest
- TrickRequest
- SelectionRequest
- RoundEnd
- Ending

### 5.60.1 Ausführliche Beschreibung

Die GamePhase modelliert die verschiedenen Zustände des Spiels im GameState

## 5.61 GameState Klassenreferenz

**Geschützte Methoden**

- GameState (RulesetType ruleset, List< Card > deck)
- boolean addPlayerToGame (String name)
- void setFirstPlayer (PlayerState player)
- PlayerState getFirstPlayer ()
- boolean setCurrentPlayer (PlayerState player)
- PlayerState getCurrentPlayer ()

- List< Card > getCardsLeftInDeck ()
- Map< String, Card > getPlayedCards ()
- PlayerState getPlayerState (String name)
- void setTrumpCard (Card trumpCard)
- Card getTrumpCard ()
- int getRoundNumber ()
- int getNumberOfPlayedCards ()
- List< Card > getPlayerCards (PlayerState player)
- void shuffleDeck ()
- boolean dealCards (int number)
- void nextPlayer ()
- boolean giveACard (PlayerState player, Card card)
- boolean playCard (Card card)

**Private Attribute**

- List< PlayerState > players
- RulesetType ruleset
- PlayerState firstPlayer
- PlayerState currentPlayer
- int roundNumber
- Map< String, Card > discardPile
- List< Card > deck
- Card trumpCard

### 5.61.1 Ausführliche Beschreibung

Das GameState modelliert einen aktuellen Spielzustand, es wird vom GameServer instanziert und vom RuleSet bearbeitet. Es enthält die einzelnen PlayerStates, sowie Informationen zum Ablage-, Aufnahmestapel, Rundenanzahl, den momentan aktiven Spieler sowie GamePhase.

### 5.61.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.61.2.1 GameState ( RulesetType *ruleset,* List< Card > *deck* ) [protected]

Erstellt eine GameStateklasse und trumpCard wird als WizardCard.Empty instanziert.

**Parameter**

| | |
|---|---|
| *ruleset* | Der Regelwerktyp des Spiels |
| *deck* | Das Kartendeck im Spiel |

Benutzt GameState.deck, GameState.discardPile, GameState.players, GameState.ruleset und GameState.trumpCard.

### 5.61.3 Dokumentation der Elementfunktionen

#### 5.61.3.1 boolean addPlayerToGame ( String *name* ) [protected]

Fügt den Spieler ins Spiel hinein, falls er nicht schon im Spiel ist.

**Parameter**

| | |
|---|---|
| *name* | Der Name eines Spielers |

**Rückgabe**

true falls der Spieler noch nicht im Spiel ist, und false sonst

Benutzt GameState.players und GameState.ruleset.

#### 5.61.3.2 void setFirstPlayer ( PlayerState *player* ) [protected]

Setzt einen neuen Spieler als firstPlayer.

**Parameter**

| | |
|---|---|
| *player* | Der neue firstPlayer |

Benutzt GameState.firstPlayer.

#### 5.61.3.3 PlayerState getFirstPlayer ( ) [protected]

Holt den Spieler der als erster am Zug war.

**Rückgabe**

firstPlayer Der Spielzustand des Spielers der als erster am Zug war

Benutzt GameState.firstPlayer.

#### 5.61.3.4 boolean setCurrentPlayer ( PlayerState *player* ) [protected]

Setzt einen neuen Spieler als currentPlayer.

**Parameter**

| | |
|---|---|
| *player* | Der neue currentPlayer |

Benutzt GameState.currentPlayer.

#### 5.61.3.5 PlayerState getCurrentPlayer ( ) [protected]

Holt den Spieler der momentan am Zug ist.

**Rückgabe**

currentPlayer Der Spielzustand des Spielers der grad am Zug ist

Benutzt GameState.currentPlayer.

#### 5.61.3.6 List<Card> getCardsLeftInDeck ( ) [protected]

Holt die Karten die noch im Aufnahmestapel sind.

**Rückgabe**

deck Holt die Karten die noch im Aufnahmestapel sind

#### 5.61.3.7 Map<String,Card> getPlayedCards ( ) [protected]

Holt die gespielten Karten im Ablagestapel.

**Rückgabe**

discardPile Die gespielten Karten

Benutzt GameState.discardPile.

#### 5.61.3.8 PlayerState getPlayerState ( String *name* ) [protected]

Holt einen bestimmten Spieler.

**Parameter**

| | |
|---|---|
| *name* | Der Name des Spielers |

**Rückgabe**

player Der Spielzustand des Spielers

Benutzt GameState.players.

#### 5.61.3.9 void setTrumpCard ( Card *trumpCard* ) [protected]

Setzt die Trumpfkarte.

**Parameter**

| | |
|---|---|
| *trumpCard* | Die Trumpfkarte |

Benutzt GameState.trumpCard.

#### 5.61.3.10 Card getTrumpCard ( ) [protected]

Holt die momentane Trumpfkarte im Spiel.

**Rückgabe**

trumpCard Die momentane Trumpfkarte

Benutzt GameState.trumpCard.

#### 5.61.3.11 int getRoundNumber ( ) [protected]

Holt die Anzahl an Runden.

**Rückgabe**

Die Anzahl der Runden

Benutzt GameState.roundNumber.

#### 5.61.3.12 int getNumberOfPlayedCards ( ) [protected]

Holt die Anzahl der gespielten Karten.

**Rückgabe**

Die Anzahl der gespielten Karten

#### 5.61.3.13 List<Card> getPlayerCards ( PlayerState *player* ) [protected]

Holt die Karten eines Spielers.

**Parameter**

| | |
|---|---|
| *name* | Der Name vom Spieler |

**Rückgabe**

Die Karten eines Spielers zurueck, wenn der Spieler nicht gefunden wird, wird eine leere Liste zurückgegeben

Benutzt GameState.players.

#### 5.61.3.14 boolean dealCards ( int *number* ) [protected]

Verteilt eine bestimmte Anzahl an Karten an die Spieler.

**Parameter**

| | |
|---|---|
| *number* | Die Anzahl an Karten |

**Rückgabe**

True falls ein Spieler keine Karten hat, false sonst

Benutzt GameState.deck und GameState.players.

#### 5.61.3.15 boolean giveACard ( PlayerState *player,* Card *card* ) [protected]

Gibt eine bestimmte Karte einem Spieler.

**Parameter**

| | |
|---|---|
| *name* | Der Name des Spielers |
| *card* | Die Karte |

**Rückgabe**

true falls die Karte im Stapel ist, false wenn nicht

#### 5.61.3.16 boolean playCard ( Card *card* ) [protected]

Entfernt eine Karte aus der Hand des currentPlayer und legt sie auf dem Ablagestapel.

**Parameter**

| | |
|---|---|
| *card* | Die gespielte Karte |

**Rückgabe**

isInHand Gibt true zurück wenn die gespielte Karte auf der Hand vom Spieler liegt und false sonst

## 5.62 HeartsCard Enum-Referenz

Abgeleitet von Card.

**Öffentliche Methoden**

- int getValue ()
- Colour getColour ()

#### Öffentliche Attribute

- **Empty** =(0,Colour.NONE)
- **Herz2** =(0,Colour.HEART)
- **Caro3** =(3,Colour.DIAMOND)

#### Private Methoden

- HeartsCard (int value, Colour colour)

#### Private Attribute

- final int value
- final Colour colour

### 5.62.1 Ausführliche Beschreibung

Modelliert eine Heartskarte.

### 5.62.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.62.2.1 HeartsCard ( int *value*, Colour *colour* ) [private]

Erzeugt eine Heartskarte mit einem Wert und einer Farbe.

**Parameter**

| | |
|---|---|
| *value* | Der Wert der Karte |
| *colour* | Die Farbe der Karte |

### 5.62.3 Dokumentation der Elementfunktionen

#### 5.62.3.1 int getValue ( )

Gibt den Wert der Karte zurück.

**Rückgabe**

Der Wert der Karte

Implementiert Card.

#### 5.62.3.2 Colour getColour ( )

Gibt die Farbe der Karte zurück.

**Rückgabe**

Die Farbe der Karte

Implementiert Card.

## 5.63 HeartsData Klassenreferenz

Abgeleitet von OtherData.

#### Öffentliche Methoden

- String toString ()

#### Geschützte Methoden

- HeartsData ()

### 5.63.1 Ausführliche Beschreibung

Die Otherdata eines Spielers zum Spiel Hearts

## 5.64 OtherData Klassenreferenz

Basisklasse für HeartsData und WizData.

#### Öffentliche Methoden

- abstract String toString ()

#### Geschützte Methoden

- OtherData ()
- void madeTrick (Set< Card > tricks)
- List< Card > removeTricks ()
- int getNumberOfTricks ()
- void setPoints (int points)
- int getPoints ()

#### Private Attribute

- Set< Card > madeTricks
- int points

### 5.64.1 Ausführliche Beschreibung

OtherData speichert alle Spielinformationen eines Spielers, außer seiner eigenen Spielhand.

### 5.64.2 Dokumentation der Elementfunktionen

#### 5.64.2.1 void madeTrick ( Set< Card > *tricks* ) [protected]

Gibt dem Spieler die Stichkarten die er gemacht.

**Parameter**

| | |
|---|---|
| *tricks* | Die Stiche |

#### 5.64.2.2 List<Card> removeTricks ( ) [protected]

Entfernt die gemachten Stichkarten eines Spielers und fuegt sie wieder in den Kartenstapel.

**Rückgabe**

Die Kartenstiche

Benutzt OtherData.madeTricks.

#### 5.64.2.3 int getNumberOfTricks ( ) [protected]

Gibt die Anzahl der gemachten Stiche des Spielers zurueck.

**Rückgabe**

Die Anzahl der gemachten Stiche

#### 5.64.2.4 void setPoints ( int *points* ) [protected]

Setzt den Punktestand eines Spielers.

**Parameter**

| | |
|---|---|
| *points* | Der neue Punktestand |

Benutzt OtherData.points.

#### 5.64.2.5 int getPoints ( ) [protected]

Gibt den Punktestand eines Spielers zurück.

**Rückgabe**

Der Punktestand

Benutzt OtherData.points.

## 5.65 PlayerState Klassenreferenz

**Öffentliche Methoden**

- PlayerState (String name, RulesetType ruleset)

**Geschützte Methoden**

- String getName ()
- List< Card > getHand ()
- OtherData getOtherData ()
- void addCard (Card card)
- boolean removeCard (Card card)

**Private Attribute**

- String name
- List< Card > ownHand
- OtherData otherData

#### 5.65.1 Ausführliche Beschreibung

Repraesentiert den Spielzustand eines Spielers, und wird unter anderem im GameState gespeichert. Sie enthält den Namen des Spielers, seine Handkarten und OtherData.

#### 5.65.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.65.2.1 PlayerState ( String *name,* RulesetType *ruleset* )

Erstellt einen PlayerState.

**Parameter**

| | |
|---|---|
| *name* | Der Name des Spielers |
| *ruleset* | Der Typ des Spiels |

Benutzt RulesetType.Hearts, PlayerState.name und PlayerState.otherData.

#### 5.65.3 Dokumentation der Elementfunktionen

##### 5.65.3.1 String getName ( ) [protected]

Holt den Namen eines Spielers.

**Rückgabe**

name Der Name des Spielers

##### 5.65.3.2 List<Card> getHand ( ) [protected]

Holt die Kartenhand des Spielers.

**Rückgabe**

ownHand Die Kartenhand des Spielers

##### 5.65.3.3 OtherData getOtherData ( ) [protected]

Gibt die OtherData des Spielers zurück.

**Rückgabe**

otherData Die OtherData eines Spielers

Benutzt PlayerState.otherData.

Wird benutzt von ServerRuleset.getPoints().

##### 5.65.3.4 void addCard ( Card *card* ) [protected]

Gibt dem Spieler eine Karte.

**Parameter**

| | |
|---|---|
| *card* | Die Karte die dem Spieler gegeben wird |

##### 5.65.3.5 boolean removeCard ( Card *card* ) [protected]

Entfernt eine Karte aus der Hand des Spielers.

#### Parameter

| | |
|---|---|
| *card* | |

#### Rückgabe

Gibt true zurueck wenn die Karte in der Hand ist und false sonst

## 5.66 RulesetType Enum-Referenz

#### Öffentliche Attribute

- Wizard
- Hearts

### 5.66.1 Ausführliche Beschreibung

Den Regelwerk Typ den ein Spiel hat.

## 5.67 ServerHearts Klassenreferenz

Abgeleitet von ServerRuleset.

#### Öffentliche Methoden

- ServerHearts (GameServer s)
- void resolveMessage (MsgMultiCards msgMultiCard, String name)

#### Geschützte Methoden

- boolean isValidMove (Card card)
- int getEndingPoints ()
- void calculateRoundOutcome ()
- void calculateTricks ()
- String getWinner ()
- GameClientUpdate generateGameClientUpdate (String player)

#### Private Methoden

- boolean areValidChoosenCards (Set< Card > cards, String name)

#### Statische, private Attribute

- static final int MIN_PLAYERS = 4
- static final int MAX_PLAYERS = 4
- static final RulesetType RULESET = RulesetType.Hearts
- static final int ENDING_POINTS = 100

### 5.67.1 Ausführliche Beschreibung

Diese Klasse erstellt das Regelwerk zum Spiel Hearts. Sie enthaelt zudem weitere Methoden, welche für das Spiel Hearts spezifisch benoetigt werden, wie die Regelung zum Tausch von Karten und die Berechnung der Stichpunkten.

#### 5.67.2 Dokumentation der Elementfunktionen

##### 5.67.2.1 boolean isValidMove ( Card *card* ) [protected],[virtual]

Prueft ob ein gemachter Zug vom currentPlayer in einem Spiel gueltig war, wenn nicht wird an den Spieler erneut eine MsgCardRequest gesendet.

**Parameter**

| | |
|---|---|
| *card* | Die Karte die gespielt wurde |

**Rückgabe**

true falls Zug gueltig und false wenn nicht

Implementiert ServerRuleset.

##### 5.67.2.2 int getEndingPoints ( ) [protected]

Holt die Anzahl der Punkte die ein Spieler haben kann ab der, das Spiel vorbei ist.

**Rückgabe**

Anzahl der maximalen Punkte

Benutzt ServerHearts.ENDING_POINTS.

##### 5.67.2.3 void resolveMessage ( MsgMultiCards *msgMultiCard,* String *name* )

Verarbeitet die RulesetMessage dass mehrerer Karten vom Spieler uebergeben werden.

Die wird dann in areValidChoosenCards ueberprueft, bei falsche Eingabe wird´ eine MsgMultiCardsRequest an den selben Spieler gesendet. Bei richtiger Eingabe geht das Spiel weiter.

**Parameter**

| | |
|---|---|
| *msgMultiCard* | Die Nachricht vom Client |
| *name* | Der Name des Spielers |

##### 5.67.2.4 boolean areValidChoosenCards ( Set< Card > *cards,* String *name* ) [private]

Ueberprueft ob eine uebergebenes Kartenset von einem Spieler gültig ist.

**Parameter**

| | |
|---|---|
| *cards* | Ein Kartenset |

**Rückgabe**

true falls das Kartenset gueltig ist, false wenn nicht

##### 5.67.2.5 GameClientUpdate generateGameClientUpdate ( String *player* ) [protected],[virtual]

Erzeugt ein GameClientUpdate welches individuell für jeden Benutzer ist.

**Parameter**

| | |
|---|---|
| *player* | Dem Spieler |

Implementiert ServerRuleset.

## 5.68 ServerRuleset Klassenreferenz

Basisklasse für ServerHearts und ServerWizard.

### Öffentliche Methoden

- ServerRuleset (RulesetType ruleset, int min, int max, GameServer server)
- RulesetType getRulesetType ()
- int getMinPlayers ()
- int getMaxPlayers ()
- GamePhase getGamePhase ()
- void runGame ()
- void resolveMessage (RulesetMessage message, String name)
- void resolveMessage (MsgCard msgCard, String name)

### Geschützte Methoden

- int getRoundNumber ()
- void setGamePhase (GamePhase phase)
- void setFirstPlayer (PlayerState player)
- PlayerState getFirstPlayer ()
- boolean nextPlayer ()
- boolean setCurrentPlayer (PlayerState player)
- OtherData getOtherData (PlayerState player)
- PlayerState getCurrentPlayer ()
- void addPlayerToGame (String name)
- PlayerState getPlayerState (String name)
- List< Card > getPlayerCards (PlayerState player)
- void send (RulesetMessage message, String name)
- void broadcast (RulesetMessage message)
- void setPoints (PlayerState player, int i)
- int getPoints (PlayerState player)
- boolean dealCards (int number)
- boolean giveACard (PlayerState player, Card card)
- boolean playCard (Card card)
- void setTrumpCard (Card card)
- abstract boolean isValidMove (Card card)
- abstract void calculateTricks ()
- abstract void calculateRoundOutcome ()
- abstract String getWinner ()
- abstract GameClientUpdate generateGameClientUpdate (String player)

### Private Methoden

- List< Card > createDeck ()

**Private Attribute**

- GameServer server
- GameState gameState
- GamePhase gamePhase
- final RulesetType RULESET
- final int MIN_PLAYERS
- final int MAX_PLAYERS

### 5.68.1 Ausführliche Beschreibung

Das ServerRuleset ist eine akstrakte Klasse und fuer den Ablauf und die Einhaltung der Regeln eines Spiels zustaendig (/L280/). Das ServerRuleset wird im GameServer instanziert und verwaltet die Zustaende des GameStates im Server. Mit der Methode isValidMove() wird eine Eingabe eines Clients auf Regelkonformität ueberprueft und dann das GameState veraendert. Ueber resolveMessage() kann eine GameServerinstanz eine RulesetMessage vom Player an das Ruleset weiterleiten.

### 5.68.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.68.2.1 ServerRuleset ( RulesetType *ruleset,* int *min,* int *max,* GameServer *server* )

Erstellt ein ServerRuleset und erzeugt ein GameState.

**Parameter**

| | |
|---|---|
| *ruleset* | Der Rulesettyp vom Server |
| *min* | Die minimale Anzahl an Spielern |
| *max* | Die maximale Anzahl an Spielern |
| *server* | Der Server auf dem gespielt wird |

Benutzt ServerRuleset.createDeck(), ServerRuleset.gamePhase, ServerRuleset.gameState, ServerRuleset.MAX_PLAYERS, ServerRuleset.MIN_PLAYERS, ServerRuleset.RULESET, ServerRuleset.server und GamePhase.Start.

### 5.68.3 Dokumentation der Elementfunktionen

#### 5.68.3.1 int getRoundNumber ( ) [protected]

Holt die aktuelle Rundenanzahl zurueck.

**Rückgabe**

Die aktuelle Rundenanzahl

#### 5.68.3.2 RulesetType getRulesetType ( )

Gibt den Typ des Regelwerks zurueck.

**Rückgabe**

Der Typ vom Regelwerk

Benutzt ServerRuleset.RULESET.

#### 5.68.3.3 int getMinPlayers ( )

Gibt die Mindestanzahl an Spielern zurueck für dieses Spiel.

**Rückgabe**

Die Mindestanzahl an Spielern

Benutzt ServerRuleset.MIN_PLAYERS.

#### 5.68.3.4 int getMaxPlayers ( )

Gibt die Maximale Anzahl an Spielern zurueck.

**Rückgabe**

Die maximale Anzahl an Spielern

Benutzt ServerRuleset.MAX_PLAYERS.

#### 5.68.3.5 void setGamePhase ( GamePhase *phase* ) [protected]

Aendert den momentanen Spielphase.

**Parameter**

| | |
|---|---|
| *phase* | Die neue Spielphase |

#### 5.68.3.6 GamePhase getGamePhase ( )

Gibt den momentanen Spielzustand zurueck.

**Rückgabe**

Gibt die momentan Spielphase zurueck

Benutzt ServerRuleset.gamePhase.

#### 5.68.3.7 List<Card> createDeck ( ) [private]

Erzeugt ein Kartendeck, abhängig von dem RulesetType.

**Rückgabe**

Gibt ein Kartendeck zurueck

Benutzt ServerRuleset.RULESET.

Wird benutzt von ServerRuleset.ServerRuleset().

#### 5.68.3.8 void setFirstPlayer ( PlayerState *player* ) [protected]

Setzt den Spieler der als Erster am Zug ist, im Gamestate.

**Parameter**

| | |
|---|---|
| *Der* | Spielerzustand des Spielers |

#### 5.68.3.9 PlayerState getFirstPlayer ( ) [protected]

Holt den Spieler der als erster am Zug war.

**Rückgabe**

firstPlayer Der Spielzustand des Spielers der als erster am Zug war

#### 5.68.3.10 boolean nextPlayer ( ) [protected]

Setzt den naechsten Spieler in der List als currentPlayer.

**Rückgabe**

true falls es ein anderer Spieler ist und false wenn es derselbe ist.

#### 5.68.3.11 boolean setCurrentPlayer ( PlayerState *player* ) [protected]

Setzt den Spieler der am Naechsten am Zug ist, im Gamestate.

**Parameter**

| | |
|---|---|
| *player* | Der Playerstate eines Spielers |

**Rückgabe**

false wenn der selbe Spieler nochmal als currentPlayer gesetzt wird

#### 5.68.3.12 OtherData getOtherData ( PlayerState *player* ) [protected]

Die OtherData eines Spielers.

**Parameter**

| | |
|---|---|
| *player* | Der Spielerzustand |

**Rückgabe**

Gibt OtherData zurück

#### 5.68.3.13 PlayerState getCurrentPlayer ( ) [protected]

Holt den Spieler der gerade am Zug ist.

**Rückgabe**

currentPlayer Der Spielzustand des Spielers der grad am Zug ist

#### 5.68.3.14 void addPlayerToGame ( String *name* ) [protected]

Fuegt einen Spieler ins Spiel ein.

**Parameter**

| | |
|---|---|
| *name* | Der name vom Spieler |

#### 5.68.3.15 PlayerState getPlayerState ( String *name* ) [protected]

Holt den Spielerzustand eines Spielers.

**Parameter**

| | |
|---|---|
| *name* | Der Name des Spielers |

**Rückgabe**

playerState Spielzustand eines Spielers

#### 5.68.3.16 List<Card> getPlayerCards ( PlayerState *player* ) [protected]

Holt die Spielkarten eines Spielers.

**Parameter**

| | |
|---|---|
| *name* | Der Name eines Spielers |

**Rückgabe**

Die Spielkarten des Spielers

#### 5.68.3.17 void send ( RulesetMessage *message,* String *name* ) [protected]

Schickt eine Nachricht an einen Spieler, über den Gameserver.

**Parameter**

| | |
|---|---|
| *message* | Die Nachricht vom Typ RulesetMessage |
| *name* | Der Name vom Spieler |

#### 5.68.3.18 void broadcast ( RulesetMessage *message* ) [protected]

Schickt eine Nachricht an alle Spieler.

**Parameter**

| | |
|---|---|
| *message* | Die Nachricht |

#### 5.68.3.19 void resolveMessage ( RulesetMessage *message,* String *name* )

Verarbeitet eine allgemeine RulesetMessage.

**Parameter**

| | |
|---|---|
| *msgCard* | Die Nachricht vom Client welche Karte gespielt wurde |
| *name* | Der Name des Spielers |

#### 5.68.3.20 void resolveMessage ( MsgCard *msgCard,* String *name* )

Verarbeitet die RulesetMessage dass eine Karte vom Spieler gespielt.

Die wird dann in isValidMove überprüft, bei falsche Eingabe wird´ eine MsgCardRequest an den selben Spieler geschickt. Bei richtiger Eingabe geht das Spiel weiter.

**Parameter**

| | |
|---|---|
| *msgCard* | Die Nachricht vom Client welche Karte gespielt wurde |
| *name* | Der Name des Spielers |

#### 5.68.3.21 void setPoints ( PlayerState *player,* int *i* ) [protected]

Setzt den Punktestand eines Spielers.

**Parameter**

| | |
|---|---|
| *player* | Der Spieler |
| *i* | Punktestand |

#### 5.68.3.22 int getPoints ( PlayerState *player* ) [protected]

Holt den Punktestand eines Spielers.

**Parameter**

| | |
|---|---|
| *player* | Der Spieler |

**Rückgabe**

Die Punkte des Spielers

Benutzt PlayerState.getOtherData().

#### 5.68.3.23 boolean dealCards ( int *number* ) [protected]

Verteilt eine bestimmte Anzahl an Karten an die Spieler.

**Parameter**

| | |
|---|---|
| *number* | Die Anzahl an Karten |

**Rückgabe**

Gibt true zurueck wenn ein Spieler keine Karten hat, false sonst

#### 5.68.3.24 boolean giveACard ( PlayerState *player,* Card *card* ) [protected]

Gibt einem Spieler eine bestimmte Karte.

**Parameter**

| | |
|---|---|
| *player* | Der Name eines Spielers |
| *card* | Eine Karte |

**Rückgabe**

Gibt true zurück wenn die Karte im Deck ist, false sonst

#### 5.68.3.25 boolean playCard ( Card *card* ) [protected]

Der momentane Spieler spielt eine Karte.

**Parameter**

| | |
|---|---|
| *card* | Die gespielte Karte |

**Rückgabe**

true falls der Spieler die Karte hat, false wenn nicht

#### 5.68.3.26 void setTrumpCard ( Card *card* ) [protected]

Setzt eine Karte als Trumpf.

**Parameter**

| | |
|---|---|
| *card* | Eine Karte |

#### 5.68.3.27 abstract boolean isValidMove ( Card *card* ) [protected], [pure virtual]

Prueft ob ein gemachter Zug vom currentPlayer in einem Spiel gueltig war, wenn nicht wird an den Spieler erneut eine MsgCardRequest gesendet.

#### Parameter

| | |
|---|---|
| *card* | Die Karte die gespielt wurde |

#### Rückgabe

true falls Zug gueltig und false wenn nicht

Implementiert in ServerWizard und ServerHearts.

#### 5.68.3.28 abstract GameClientUpdate generateGameClientUpdate ( String *player* ) [protected], [pure virtual]

Erzeugt ein GameClientUpdate welches individuell für jeden Benutzer ist.

#### Parameter

| | |
|---|---|
| *player* | Dem Spieler |

Implementiert in ServerWizard und ServerHearts.

## 5.69 ServerWizard Klassenreferenz

Abgeleitet von ServerRuleset.

#### Öffentliche Methoden

- ServerWizard (GameServer s)
- void resolveMessage (MsgNumber msgNumber, String name)
- void resolveMessage (MsgSelection msgSelection, String name)

#### Geschützte Methoden

- boolean isValidMove (Card card)
- void calculateRoundOutcome ()
- int getplayingRounds ()
- void calculateTricks ()
- String getWinner ()
- GameClientUpdate generateGameClientUpdate (String player)

#### Private Methoden

- void setPlayingRounds (int rounds)
- boolean isValidNumber (int number, String name)
- boolean isValidColour (Colour colour, String name)

#### Private Attribute

- int playingRounds

#### Statische, private Attribute

- static final int MIN_PLAYERS = 3
- static final int MAX_PLAYERS = 6
- static final RulesetType RULESET = RulesetType.Wizard

#### 5.69.1 Ausführliche Beschreibung

Diese Klasse erstellt das Regelwerk zum Spiel Wizard. Sie enthaelt zudem weitere Methoden, welche für das Spiel Wizard spezifisch benoetigt werden, wie das Ansage von Stichen, der Bestimmung von Trumpffarben und die Bestimmung der Rundenanzahl.

#### 5.69.2 Dokumentation der Elementfunktionen

##### 5.69.2.1 boolean isValidMove ( Card *card* ) [protected], [virtual]

Prueft ob ein gemachter Zug vom currentPlayer in einem Spiel gueltig war, wenn nicht wird an den Spieler erneut eine MsgCardRequest gesendet.

**Parameter**

| | |
|---|---|
| *card* | Die Karte die gespielt wurde |

**Rückgabe**

true falls Zug gueltig und false wenn nicht

Implementiert ServerRuleset.

##### 5.69.2.2 void setPlayingRounds ( int *rounds* ) [private]

Setzt die Anzahl an Runden die es in diesem Spiel gibt.

**Parameter**

| | |
|---|---|
| *rounds* | Die Anzahl an Runden |

Benutzt ServerWizard.playingRounds.

##### 5.69.2.3 int getplayingRounds ( ) [protected]

Holt die Anzahl der Runden die gespielt werden.

**Rückgabe**

playingRounds Die Anzahl an Runden

Benutzt ServerWizard.playingRounds.

##### 5.69.2.4 void resolveMessage ( MsgNumber *msgNumber,* String *name* )

Verarbeitet die RulesetMessage dass der Spieler eine Stichansage macht.

Die wird dann in isValidNumber überprüft, bei falsche Eingabe wird´ eine MsgCardRequest an den selben Spieler geschickt. Bei richtiger Eingabe geht das Spiel weiter.

**Parameter**

| | |
|---|---|
| *msgNumber* | Die Nachricht vom Client |
| *name* | Der Name des Spielers |

##### 5.69.2.5 boolean isValidNumber ( int *number,* String *name* ) [private]

Ueberprueft ob eine eingegebene Stichangabe eines Spielers gueltig ist.

**Parameter**

| | |
|---|---|
| *number* | Die Stichangabe |
| *name* | Der Name vom Spieler |

**Rückgabe**

true falls die Stichangabe gültig ist, false wenn nicht

#### 5.69.2.6 void resolveMessage ( MsgSelection *msgSelection,* String *name* )

Verarbeitet die RulesetMessage dass mehrerer Karten vom Spieler uebergeben werden.

Die wird dann in isValidColour überprüft, bei falsche Eingabe wird´ MsgMultiCardRequest an den selben Spieler geschickt. Bei richtiger Eingabe geht das Spiel weiter.

**Parameter**

| | |
|---|---|
| *msgSelection* | Die Nachricht vom Client |
| *name* | Der Name des Spielers |

#### 5.69.2.7 boolean isValidColour ( Colour *colour,* String *name* ) [private]

Ueberprueft ob eine eingebene Trumpffarbe eines Spielers gueltig ist.

**Parameter**

| | |
|---|---|
| *colour* | Die Trumpffarbe |
| *name* | Der Name des Spielers |

**Rückgabe**

true falls die Farbe gueltig ist, false wenn nicht

#### 5.69.2.8 GameClientUpdate generateGameClientUpdate ( String *player* ) [protected], [virtual]

Erzeugt ein GameClientUpdate welches individuell für jeden Benutzer ist.

**Parameter**

| | |
|---|---|
| *player* | Dem Spieler |

Implementiert ServerRuleset.

### 5.69.3 Dokumentation der Datenelemente

#### 5.69.3.1 int playingRounds [private]

Die Anzahl an Runden die gespielt wird.

Ist abhaengig von der Spieleranzahl.

Wird benutzt von ServerWizard.getplayingRounds() und ServerWizard.setPlayingRounds().

## 5.70 WizardCard Enum-Referenz

Abgeleitet von Card.

**Öffentliche Methoden**

- int getValue ()

- Colour getColour ()

#### Öffentliche Attribute

- **Empty** =(0,Colour.NONE)
- **NarrBlau** =(0,Colour.BLUE)
- **ZaubererRot** =(14,Colour.RED)
- **EinsGruen** =(1,Colour.GREEN)
- **ZweiGruen** =(2,Colour.GREEN)
- **DreiGruen** =(3,Colour.GREEN)
- **ZweiRot** =(2,Colour.RED)
- **DreiRot** =(3,Colour.RED)
- **VierRot** =(4,Colour.RED)

#### Private Methoden

- WizardCard (int value, Colour colour)

#### Private Attribute

- final int value
- final Colour colour

### 5.70.1 Ausführliche Beschreibung

Modelliert eine Wizardkarte

### 5.70.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.70.2.1 WizardCard ( int *value*, Colour *colour* ) [private]

Erzeugt eine Wizardkarte.

**Parameter**

| | |
|---|---|
| *value* | Der Wert der Karte |
| *colour* | Die Farbe der Karte |

### 5.70.3 Dokumentation der Elementfunktionen

#### 5.70.3.1 int getValue ( )

Gibt den Wert der Karte zurück.

**Rückgabe**

Der Wert der Karte

Implementiert Card.

#### 5.70.3.2 Colour getColour ( )

Gibt die Farbe der Karte zurück.

**Rückgabe**

Die Farbe der Karte

Implementiert Card.

## 5.71 WizData Klassenreferenz

Abgeleitet von OtherData.

**Öffentliche Methoden**

- String toString ()

**Geschützte Methoden**

- WizData ()
- int getAnnouncedTricks ()
- void setAnnouncedTricks (int annouceTricks)

**Private Attribute**

- int announcedTricks

### 5.71.1 Ausführliche Beschreibung

Die Otherdata eines Spielers zum Spiel Wizard

### 5.71.2 Dokumentation der Elementfunktionen

#### 5.71.2.1 int getAnnouncedTricks ( ) [protected]

Holt die angesagten Stiche des Spielers.

**Rückgabe**

announcedTricks Die angesagten Stiche

#### 5.71.2.2 void setAnnouncedTricks ( int *annouceTricks* ) [protected]

Beim Spielstart werden die vorausgesagten Stiche des Spieler gespeichert.

**Parameter**

| | |
|---|---|
| *annouceTricks* | Die vorausgesagten Stiche des Spielers |

## 5.72 GameServer Klassenreferenz

Abgeleitet von Server.

#### Öffentliche Methoden

- GameServer (LobbyServer server, Player gameMaster, String GameName, RulesetType ruleset, String password, boolean hasPassword)
- GameServerRepresentation getRepresentation ()
- synchronized void addPlayer (Player player)
- synchronized void removePlayer (Player player)
- void sendRulesetMessage (String player, RulesetMessage message)
- void broadcastRulesetMessage (RulesetMessage message)
- void receiveMessage (Player player, ComKickPlayerRequest kickPlayer)
- void receiveMessage (Player player, ComChatMessage chat)
- void receiveMessage (Player player, ComClientLeave leave)
- void receiveMessage (Player player, ComClientQuit quit)
- void receiveMessage (Player player, ComStartGame start)
- void receiveMessage (Player player, ComRuleset ruleset)
- ComInitGameLobby initLobby ()
- void handleIOException (Player player)

#### Private Attribute

- LobbyServer lobbyServer
- String gameMasterName
- String name
- String password
- int maxPlayers
- int currentPlayers
- boolean hasPassword
- RulesetType rulesetType
- ServerRuleset ruleset

#### Weitere Geerbte Elemente

### 5.72.1 Ausführliche Beschreibung

Diese Klasse ist fuer die Verwaltung eines Spieles zustaendig. Sie verwaltet die Kommunikation zwischen den Clients waehrend eines Spieles. Die GameServer-Klasse erbt Methoden zur Kommunikation vom Server. Der GameServer tauscht Nachrichten zwischen Ruleset und Player aus, um so den Spielablauf zu koordinieren.

### 5.72.2 Beschreibung der Konstruktoren und Destruktoren

#### 5.72.2.1 GameServer ( LobbyServer *server,* Player *gameMaster,* String *GameName,* RulesetType *ruleset,* String *password,* boolean *hasPassword* )

Konstruktor des GameServers.

Setzt die Attribute lobbyServer, name, password, hasPasword und rulesetType auf die uebergebenen Werte. Setzt den gameMasterName auf den Namen des gameMaster und fuegt den gameMaster dem Set an Spielern hinzu. Bestimmt mithilfe des Enums RulesetType das Ruleset und erstellt es. Setzt currentPlayers auf eins und maxPlayers je nach Ruleset.

**Parameter**

| | |
|---|---|
| *server* | ist der LobbyServer der den GameServer erstellt hat. |
| *gameMaster* | ist der Name des Spielleiters |
| *GameName* | ist der Name des Spiels |
| *ruleset* | gibt an, welches Ruleset verwendet wird |
| *password* | speichert das Passwort des Spiels |
| *hasPassword* | gibt an, ob das Spiel ein Passwort hat |

### 5.72.3 Dokumentation der Elementfunktionen

#### 5.72.3.1 GameServerRepresentation getRepresentation ( )

Erstellt eine neue GameServerRepresentation und gibt sie zurueck.

**Rückgabe**

Gibt die neue GameServerRepresentation zurueck

Benutzt GameServer.currentPlayers, GameServer.gameMasterName, GameServer.hasPassword, GameServer.maxPlayers, GameServer.name und GameServer.rulesetType.

#### 5.72.3.2 synchronized void addPlayer ( Player *player* )

Diese Methode fuegt einen Player dem Set an Playern hinzu, welche der Server verwaltet.

Es wird vorrausgesetzt, dass der Player gueltig und noch nicht im Set vorhanden ist. Zusaetzlich wird die Zahl der currentPlayers um eins Erhoeht.

**Parameter**

| | |
|---|---|
| *player* | ist der Player, der hinzugefuegt wird |

Benutzt GameServer.currentPlayers.

#### 5.72.3.3 synchronized void removePlayer ( Player *player* )

Diese Methode entfernt einen Player aus dem Set an Playern, welche der Server verwaltet.

Es wird vorrausgesetzt, dass der Player gueltig und im Set vorhanden ist. Zusaetzlich wird die Zahl der currentPlayers um eins Verringert.

**Parameter**

| | |
|---|---|
| *player* | ist der Player, der entfernt wird |

Benutzt GameServer.currentPlayers.

#### 5.72.3.4 void sendRulesetMessage ( String *player,* RulesetMessage *message* )

Diese Methode verpackt eine RulesetMessage in ein ComObject und verschickt es mit sendToPlayer() an einen bestimmten Spieler.

**Parameter**

| | |
|---|---|
| *player* | ist der Name des Spielers an den die Nachricht verschickt wird |
| *message* | ist die Ruleset Nachricht, die in ein ComObject verpackt wird |

Benutzt Server.sendToPlayer().

#### 5.72.3.5 void broadcastRulesetMessage ( RulesetMessage *message* )

Diese Methode verpackt eine RulesetMessage in ein ComObject und verschickt es mit broadcast() an alle Spieler.

**Parameter**

| | |
|---|---|
| *message* | ist die Ruleset Nachricht, die in ein ComObject verpackt wird |

Benutzt Server.broadcast().

#### 5.72.3.6 void receiveMessage ( Player *player,* ComKickPlayerRequest *kickPlayer* )

Diese Methode ist dafuer zustaendig zu ermitteln, was passiert wenn ein Spieler aus der GameLobby geworfen wird.

Der Player wird durch Aufruf von changeServer an die Lobby zurueckgegeben. An diesen Spieler wird ein ComInitLobby und ein ComWarning geschickt. Danach wird ein ComUpdatePlayerlist Objekt mit broadcast an alle Clients im Spiel verschickt.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *kicked* | ist das ComObject, das verarbeitet wird |

#### 5.72.3.7 void receiveMessage ( Player *player,* ComChatMessage *chat* )

Diese Methode ist dafur zustaendig eine Chatnachricht an alle Clients im Spiel zu verschicken.

Dafuer wird die ComChatMessage mit broadcast an alle Spieler im playerSet verteilt.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *chat* | ist das ComObject, das die Chatnachricht enthaelt |

Benutzt Server.broadcast().

#### 5.72.3.8 void receiveMessage ( Player *player,* ComClientLeave *leave* )

Diese Methode gibt einen Player, der die GameLobby verlassen will, durch Aufruf von changeServer an die ServerLobby zurueck und schickt ihm ein ComInitLobby.

Danach wird ein ComUpdatePlayerlist Objekt mit broadcast an alle Clients im Spiel verschickt.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *leave* | ist das ComObject, welches angibt, dass der Spieler in die Lobby zurueckkehrt |

#### 5.72.3.9 void receiveMessage ( Player *player,* ComClientQuit *quit* )

Diese Methode behandelt den Fall, dass ein Spieler das laufende Spiel verlaesst.

Alle Spieler, die sich im Spiel befinden werden durch Aufruf von changeServer an die Lobby zurueckgegeben und bekommen ein ComInitLobby und ein ComWarning. Das Spiel wird aus dem gameServerSet des LobbyServers entfernt.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *quit* | ist das ComObject, welches angibt, dass der Spieler das Spiel verlaesst |

#### 5.72.3.10 void receiveMessage ( Player *player,* ComStartGame *start* )

Diese Methode sagt dem Ruleset, dass ein neues Spiel gestartet werden soll indem er dessen runGame Methode aufruft.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *start* | ist das ComObject, dass angibt, dass das Spiel gestartet werden soll |

#### 5.72.3.11 void receiveMessage ( Player *player,* ComRuleset *ruleset* )

Diese Methode gibt das erhaltene ComRuleset durch einen Aufruf von resolveMessage an das Ruleset weiter.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *ruleset* | ist das ComObject, das zeigt, dass das Object vom Ruleset bearbeitet werden muss |

#### 5.72.3.12 ComInitGameLobby initLobby ( )

Baut ein neues ComInitGameLobby Objekt und gibt es zurueck.

**Rückgabe**

Gibt das ComInitGameLobby Objekt zurueck

#### 5.72.3.13 void handleIOException ( Player *player* )

Diese Methode legt den Ablauf fest, was passiert, falls die Verbindung zu einem Client verloren gegangen ist.

Der uebergebene Player wird aus dem playerSet im GameServer, sowie dem names Set im LobbyServer entfernt. Alle Spieler, die sich im Spiel befinden werden durch Aufruf von changeServer an die Lobby zurueckgegeben und bekommen ein ComInitLobby und ein ComWarning. Das Spiel wird aus dem gameServerSet des LobbyServers entfernt.

**Parameter**

| | |
|---|---|
| *player* | ist der Tread von dem die IOException kommt |

### 5.72.4 Dokumentation der Datenelemente

#### 5.72.4.1 String password [private]

Das Passwort, das der Spielleiter beim erstellen gesetzt hat.

Ist NULL, falls es kein Passwort gibt.

## 5.73 GameServerRepresentation Klassenreferenz

**Öffentliche Methoden**

- GameServerRepresentation (String gameMaster, String gameName, int max, int current, RulesetType type, boolean password)
- String getGameMasterName ()
- String getName ()
- int getMaxPlayers ()
- int getCurrentPlayers ()
- RulesetType getRuleset ()
- boolean isHasPassword ()

**Private Attribute**

- String gameMasterName
- String name
- int maxPlayers
- int currentPlayers
- RulesetType ruleset
- boolean hasPassword

#### 5.73.1 Ausführliche Beschreibung

Dies ist eine Klasse, die Informationen ueber den Zustand eines Spielservers bereithaelt. Sie wird dem ComObjekt ComLobbyUpdateGameList angehaengt, um die Spielliste in der GameLobby aktualisieren zu koennen.

#### 5.73.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.73.2.1 GameServerRepresentation ( String *gameMaster,* String *gameName,* int *max,* int *current,* RulesetType *type,* boolean *password* )

Der Konstruktor der Klasse GameServerRepresentation initialisiert die Attribute mit den vom GameServer übergebenen Werten.

**Parameter**

| | |
|---|---|
| *gameMaster* | der Name des Spielleiters |
| *gameName* | der Name des Spiels |
| *max* | Maximal moegliche Anzahl teilnehmender Spieler |
| *current* | Anzahl momentaner Spieler |
| *type* | Welches Ruleset verwendet wird |
| *password* | ob das Spiel ein Passwort hat |

### 5.74 LobbyServer Klassenreferenz

Abgeleitet von Server.

**Klassen**

- class ClientListenerThread

**Öffentliche Methoden**

- LobbyServer ()
- void addName (String name)
- void removeName (String name)
- void addGameServer (GameServer game)
- void removeGameServer (GameServer game)
- void receiveMessage (Player player, ComChatMessage chat)
- void receiveMessage (Player player, ComClientQuit quit)
- void receiveMessage (Player player, ComCreateGameRequest create)
- void receiveMessage (Player player, ComJoinRequest join)
- void receiveMessage (Player player, ComLoginRequest login)
- ComInitLobby initLobby ()
- void handleIOException (Player player)

#### Private Attribute

- Set< String > names
- Set< Player > noNames
- Set< GameServer > gameServerSet
- ClientListenerThread clientListenerThread
- ServerSocket socket

#### Weitere Geerbte Elemente

### 5.74.1 Ausführliche Beschreibung

Diese Klasse ist fuer die Verwaltung der Spiellobby auf dem Server verantwortlich. Sie erstellt neue Spiele und verwaltet laufende Spiele. Auch wird der Chatverkehr ueber sie an die verbundenen Spieler weitergeleitet. Die LobbyServer-Klasse erbt Methoden zur Kommunikation vom Server.

### 5.74.2 Dokumentation der Elementfunktionen

#### 5.74.2.1 void addName ( String *name* )

Fuegt einen neuen Benutzennamen in das Namensset ein.

Es wird vorrausgesetzt, dass der Name noch nicht im Set vorhanden ist.

**Parameter**

| | |
|---|---|
| *name* | ist der Name der eingefuegt wird |

#### 5.74.2.2 void removeName ( String *name* )

Loescht einen Benutzennamen aus dem Namensset.

Es wird vorrausgesetzt, dass der Name im Set vorhanden ist.

**Parameter**

| | |
|---|---|
| *name* | ist der Name der geloescht wird |

#### 5.74.2.3 void addGameServer ( GameServer *game* )

Fuegt einen neuen GameServer in das gameServerSet ein.

**Parameter**

| | |
|---|---|
| *game* | ist der GameServer der eingefuegt wird |

#### 5.74.2.4 void removeGameServer ( GameServer *game* )

Loescht einen GameServer aus dem Gameserverset.

**Parameter**

| | |
|---|---|
| *game* | ist der GameServer der geloescht wird |

#### 5.74.2.5 void receiveMessage ( Player *player*, ComChatMessage *chat* )

Diese ueberladene Methode ist dafuer zustaendig eine Chatnachricht an alle Clients im Spiel zu verschicken.

Dafuer wird die ComChatMessage mit broadcast an alle Spieler im playerSet verteilt.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *chat* | ist das ComObject, das die Chatnachricht enthaelt |

Benutzt Server.broadcast().

#### 5.74.2.6 void receiveMessage ( Player *player,* ComClientQuit *quit* )

Diese ueberladene Methode schliesst die Verbindung, der Player wird aus dem playerSet (bzw.

noNames Set) entfernt, der Name des Players wird aus dem Set names entfernt. War der Spieler im playerSet, wird ein ComUpdatePlayerlist mit broadcast an alle Clients verschickt.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *quit* | ist das ComObject, welches angibt, dass der Spieler das Spiel vollstaendig verlaesst |

#### 5.74.2.7 void receiveMessage ( Player *player,* ComCreateGameRequest *create* )

Diese ueberladene Methode erstellt einen neuen GameServer fuegt ihm den Player durch Aufruf von dessen changeServer Methode hinzu.

Der neue GameServer wird in das gameServerSet eingefuegt. Durch broadcast wird im LobbyServer sowohl ComUpdatePlayerlist als auch ein ComLobbyUpdateGamelist verschickt. Zusaetzlich wird dem Client mit sendToPlayer ein ComInitGameLobby geschickt.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *create* | ist das ComObject, welches angibt, dass der Player ein neues Spiel erstellt hat |

#### 5.74.2.8 void receiveMessage ( Player *player,* ComJoinRequest *join* )

Diese ueberladene Methode fuegt einen Player dem entsprechenden GameServer hinzu.

Falls das Passwort nicht leer ist wird geprueft, ob es mit dem Passwort des Spieles uebereinstimmt, wenn nicht, wird ein ComWarning an den Client geschickt. Ansonsten wird und der Player dem, durch Namen des Spielleiters identifizierten, Gameserver, durch Aufruf von changeServer uebergeben. Dem joinendenClient wird mit sendToPlayer ein ComInitGameLobby geschickt. Durch broadcast wird sowohl im LobbyServer ein ComUpdatePlayerlist verschickt.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *join* | ist das ComObject, welches angibt, dass der Player einem Spiel beitreten will |

#### 5.74.2.9 void receiveMessage ( Player *player,* ComLoginRequest *login* )

Diese ueberladene Methode ueberprueft, ob der Name im Set names vorhanden ist, falls ja, wird ein ComWarning an den Client geschickt, dass der Name bereits vergeben ist, falls nein, wird im Player setName aufgerufen.

Der Player wird aus dem noNames Set entfernt und in das playerSet eingefuegt. Der Name wird in das Set names eingefuegt. Dem Client wird ein ComServerAcknowledgement geschickt.

**Parameter**

| | |
|---|---|
| *player* | ist der Thread der die Nachricht erhalten hat |
| *login* | ist das ComObject, dass den Benutzernamen des Clients enthält |

#### 5.74.2.10 ComInitLobby initLobby ( )

Diese Methode baut ein neues ComInitLobby Objekt und gibt es zurueck.

**Rückgabe**

Gibt das ComInitLobby Objekt zurueck

#### 5.74.2.11 void handleIOException ( Player *player* )

Diese Methode legt den Ablauf fest, was passiert, falls die Verbindung zu einem Client verloren gegangen ist.

Der uebergebene Player wird aus dem playerSet sowie dem names Set im LobbyServer entfernt.

**Parameter**

| | |
|---|---|
| *player* | ist der Tread von dem die IOException kommt |

## 5.75 LobbyServer.ClientListenerThread Klassenreferenz

Abgeleitet von Runnable.

**Öffentliche Methoden**

- ClientListenerThread ()
- void run ()

### 5.75.1 Ausführliche Beschreibung

Diese innere Klasse ist fuer das Zustandekommen von Clientverbindungen zustaendig. Der Thread wartet auf eingehende Clientverbindungen, stellt diese her und instanziiert fuer jede Verbindung eine Klasse Player. Dieser wird dann dem LobbyServer uebergeben.

## 5.76 Player Klassenreferenz

Abgeleitet von Runnable.

**Öffentliche Methoden**

- Player (Server lobbyServer, ObjectOutputStream output, ObjectInputStream input)
- void run ()
- void send (ComObject com)
- void changeServer (Server newServer)
- String getName ()
- void setName (String newName)

**Private Attribute**

- String name
- Server server
- ObjectOutputStream comOut
- ObjectInputStream comIn

#### 5.76.1 Ausführliche Beschreibung

Die Player-Klasse wird zum Versenden von Java Serializable Objects, sowie zum Annehmen solcher verwendet. Sie verwaltet fuer die Dauer einer Serververbindung die Verbindung zu einem Client.

#### 5.76.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.76.2.1 Player ( Server *lobbyServer,* ObjectOutputStream *output,* ObjectInputStream *input* )

Konstruktor des Players, in ihm werden die Attribute server, comOut und comIn mit vom ClientListererThread uebergebenen Werten Instanziiert.

**Parameter**

| | |
|---|---|
| *lobbyServer* | ist der LobbyServer, der zu Beginn den Player verwaltet. |
| *output* | ist der ObjectOutputStream an den entsprechenden Client |
| *input* | ist der ObjectInputStream vom entsprechenden Client |

Benutzt Player.comIn, Player.comOut und Player.server.

#### 5.76.3 Dokumentation der Elementfunktionen

##### 5.76.3.1 void run ( )

Die run-Methode des Thread nimmt eingehende Nachrichten des Client entgegen und uebergibt diese an den Server durch Aufruf der Methode resolveMessage() Faengt eine ClassNotFoundException ab, falls die Klasse nicht gefunden werden kann und gibt einen Fehler aus.

Faengt eine IOException ab und ruft im jeweiligen Server, dem er zugeteilt ist die handleIOException Methode auf.

Benutzt Player.comIn und Player.server.

##### 5.76.3.2 void send ( ComObject *com* )

Diese Methode schickt ein ComObjekt an den Client.

Sie faengt eine IOException ab und ruft im jeweiligen Server, dem er zugeteilt ist die handleIOException Methode auf.

**Parameter**

| | |
|---|---|
| *com* | ist das ComObject das verschickt wird |

##### 5.76.3.3 void changeServer ( Server *newServer* )

Diese Methode wechselt beim Player den Server an den er comObjects weiterleiten soll.

Dabei wird er aus dem playerSet des alten Servers entfernt und in das playerSet des neuen Players eingefuegt. Danach wird vom neuen Server ein ComUpdatePlayerlist Objekt mit broadcast an alle Clients, die vom Server verwaltet werden, verschickt.

**Parameter**

| | |
|---|---|
| *newServer* | ist der neue Server |

Benutzt Player.getName() und Player.server.

##### 5.76.3.4 String getName ( )

Getter-Methode fuer den Benutzernamen.

#### Rückgabe

gibt den Benutzernamen des Spielers zurueck

Benutzt Player.name.

Wird benutzt von Player.changeServer().

#### 5.76.3.5 void setName ( String *newName* )

Setter-Methode fuer den Benutzernamen.

#### Parameter

| | |
|---|---|
| *newName* | ist der neue Name |

Benutzt Player.name.

## 5.77 Server Klassenreferenz

Basisklasse für GameServer und LobbyServer.

#### Öffentliche Methoden

- void receiveMessage (Player player, ComObject com)
- synchronized void sendToPlayer (String name, ComObject com)
- synchronized void addPlayer (Player player)
- synchronized void removePlayer (Player player)
- synchronized void broadcast (ComObject com)
- void handleIOException (Player player)

#### Geschützte Attribute

- Set< Player > playerSet

### 5.77.1 Ausführliche Beschreibung

Ist eine abstrakte Klasse, von der die Klassen LobbyServer und GameServer erben. Es stellt Methoden zur Nachrichtenversendung und -verarbeitung bereit, sowie zur Verwaltung von Playern

### 5.77.2 Dokumentation der Elementfunktionen

#### 5.77.2.1 void receiveMessage ( Player *player,* ComObject *com* )

Diese Methode dient zur Verarbeitung von eingehenden ComObjects.

#### Parameter

| | |
|---|---|
| *player* | ist der Player von dem die Nachricht kommt |
| *com* | ist das ComObjekt vom Client verschickt wurde |

#### 5.77.2.2 synchronized void sendToPlayer ( String *name,* ComObject *com* )

Diese Methode wird genutzt, um ein ComObject an einen einzigen Client zu verschicken.

Der Player der die Nachricht verschicken soll wird Anhand des uebergebenen Benutzernamens identifiziert. Es wird vorrausgesetzt, dass der Name und das ComObject gueltig sind.

**Parameter**

| | |
|---|---|
| *name* | ist der Name des Clients, an den der Player die Nachricht verschicken soll |
| *c* | ist das ComObject, dass verschickt werden soll |

Benutzt Server.playerSet.

Wird benutzt von GameServer.sendRulesetMessage().

#### 5.77.2.3 synchronized void addPlayer ( Player *player* )

Diese Methode fuegt einen Player dem Set an Playern hinzu, welche der Server verwaltet. Es wird vorrausgesetzt, dass der Player gueltig und noch nicht im Set vorhanden ist.

**Parameter**

| | |
|---|---|
| *player* | ist der Player, der hinzugefuegt wird |

#### 5.77.2.4 synchronized void removePlayer ( Player *player* )

Diese Methode entfernt einen Player aus dem Set an Playern, welche der Server verwaltet.

Es wird vorrausgesetzt, dass der Player gueltig und im Set vorhanden ist.

**Parameter**

| | |
|---|---|
| *player* | ist der Player, der entfernt wird |

#### 5.77.2.5 synchronized void broadcast ( ComObject *com* )

Diese Methode wird genutzt, um ein ComObject an alle Clients, die vom Server verwaltet werden, zu schicken.

Es wird vorrausgesetzt, dass das ComObject gueltig ist.

**Parameter**

| | |
|---|---|
| *com* | ist das ComObject, dass verschickt werden soll |

Benutzt Server.playerSet.

Wird benutzt von GameServer.broadcastRulesetMessage(), LobbyServer.receiveMessage() und GameServer.-receiveMessage().

#### 5.77.2.6 void handleIOException ( Player *player* )

Diese Methode legt den Ablauf fest, was passiert, falls die Verbindung zu einem Client verloren gegangen ist.

**Parameter**

| | |
|---|---|
| *player* | ist der Tread von dem die IOException kommt |

## 5.78 ServerMain Klassenreferenz

**Öffentliche, statische Methoden**

- static void main (String[] args)

**Private Attribute**

- LobbyServer lobbyServer

#### 5.78.1 Ausführliche Beschreibung

Diese Klasse startet den Server und ist fuer die Konfiguration des Servers verantwortlich.

#### 5.78.2 Dokumentation der Elementfunktionen

##### 5.78.2.1 static void main ( String[] *args* ) [static]

Die main-Methode erstellt einen neuen LobbyServer.

**Parameter**

| | |
|---|---|
| *args* | |

# 6 JUnit-Tests

JUnit-Tests werden für die folgenden Klassen geschrieben: ClientModel, LobbyServer, GameServer, ClientHearts, ClientWizard, ServerHearts und ServerWizard. Für die folgenden Fälle wurden bereits Tests implementiert.

## 6.1 isValidWizardMove

```
package Ruleset;


import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import test.TestGameServer;
import test.TestLobbyServer;
import test.TestPlayer;

import Server.GameServer;
import Server.LobbyServer;
import Server.Player;

public class TestisValidMoveWizard {

	ServerRuleset ruleset;

	GameServer gameServer;

	LobbyServer lobbyServer;

	Player player;
	String player1;
	String player2;
	String player3;
	PlayerState playerState1;
	PlayerState playerState2;
	PlayerState playerState3;

	@Before
	public void setUp() throws Exception {
		player1 = "Tick";
```

```
        player2 = "Trick";
        player3 = "Track";
        lobbyServer = new TestLobbyServer();
        player = new TestPlayer(lobbyServer,null,null);
        gameServer = new TestGameServer(lobbyServer,player,"Mein
            Spiel",RulesetType.Wizard,
                    "",false);
        ruleset = new ServerWizard(gameServer);

        ruleset.addPlayerToGame(player1);
        ruleset.addPlayerToGame(player2);
        ruleset.addPlayerToGame(player3);

        playerState1 = ruleset.getPlayerState(player1);
        playerState2 = ruleset.getPlayerState(player2);
        playerState3 = ruleset.getPlayerState(player3);

        ruleset.setFirstPlayer(ruleset.getPlayerState(player1));
        ruleset.setTrumpCard(WizardCard.VierRot);

        ruleset.giveACard(playerState1, WizardCard.DreiGruen);
        ruleset.giveACard(playerState1, WizardCard.ZaubererRot);
        ruleset.givaACard(playerState1, WizardCard.ZweiBlau);

        ruleset.giveACard(playerState2, WizardCard.ZweiGruen);
        ruleset.giveACard(playerState2, WizardCard.DreiRot);
        ruleset.givaACard(playerState2, WizardCard.ZweiGelb);

        ruleset.giveACard(playerState3, WizardCard.NarrBlau);
        ruleset.giveACard(playerState3, WizardCard.EinsGruen);
        ruleset.giveACard(playerState3, WizardCard.ZweiRot);
    }

    @Test
    public void testSorcerer() {
        ruleset.playCard(WizardCard.ZaubererRot);
        ruleset.setCurrentPlayer(playerState2);

        boolean isValidMove = ruleset.isValidMove(WizardCard.DreiRot);

        assertTrue(isValidMove);
    }

    @Test
    public void testRed3OnGreen3() {
        ruleset.playCard(WizardCard.DreiGruen);
        ruleset.setCurrentPlayer(playerState2);
        boolean isValidMove = ruleset.isValidMove(WizardCard.DreiRot);

        assertFalse(isValidMove);
    }

    @Test
    public void testGreen2OnGreen3() {
        ruleset.playCard(WizardCard.DreiGruen);
        ruleset.setCurrentPlayer(playerState2);

        boolean isValidMove = ruleset.isValidMove(WizardCard.ZweiGruen);

        assertTrue(isValidMove);
    }

    @Test
```

```
	public void testFoolBlueOnGreen2OnGreen3() {
		ruleset.playCard(WizardCard.DreiGruen);
		ruleset.setCurrentPlayer(playerState2);

		ruleset.playCard(WizardCard.ZweiGruen);
		ruleset.setCurrentPlayer(playerState3);

		boolean isValidMove = ruleset.isValidMove(WizardCard.NarrBlau);

		assertTrue(isValidMove);
	}

}
```

## 6.2 isValidHeartsMove

```
package Ruleset;


import static org.junit.Assert.*;
import static org.junit.Assert.assertEquals;

import org.junit.Before;
import org.junit.Test;

import test.TestGameServer;
import test.TestLobbyServer;
import test.TestPlayer;

import Server.GameServer;
import Server.LobbyServer;
import Server.Player;

public class TestisValidMoveHearts {
	ServerRuleset ruleset;

	GameServer gameServer;

	LobbyServer lobbyServer;

	Player player;
	String player1;
	String player2;
	String player3;
	String player4;

	PlayerState playerState1;
	PlayerState playerState2;
	PlayerState playerState3;
	PlayerState playerState4;

	@Before
	public void setUp() throws Exception {
		player1 = "Tick";
		player2 = "Trick";
		player3 = "Track";
		player3 = "Duck";
		lobbyServer = new TestLobbyServer();
		player = new TestPlayer(lobbyServer,null,null);
		gameServer = new TestGameServer(lobbyServer,player,"Mein Spiel",
				RulesetType.Hearts, "",false);
```

```
            ruleset = new ServerHearts(gameServer);

            ruleset.addPlayerToGame(player1);
        ruleset.addPlayerToGame(player2);
        ruleset.addPlayerToGame(player3);
        ruleset.addPlayerToGame(player4);

        playerState1 = ruleset.getPlayerState(player1);
        playerState2 = ruleset.getPlayerState(player2);
        playerState3 = ruleset.getPlayerState(player3);

        ruleset.setFirstPlayer(playerState1);
        }

        @Test
        public void testIsValidMove() {
               ruleset.giveACard(playerState1, HeartsCard.Herz2);
            ruleset.giveACard(playerState1, HeartsCard.Kreuz9);
            ruleset.giveACard(playerState2, HeartsCard.Caro3);
            ruleset.giveACard(playerState2, HeartsCard.Caro6);
            ruleset.giveACard(playerState3, HeartsCard.Pik4);
            ruleset.giveACard(playerState3, HeartsCard.Pik5);
         ruleset.giveACard(playerState4, HeartsCard.Pik1);
            ruleset.giveACard(playerState4, HeartsCard.Herz7);

            boolean isValidMove = ruleset.isValidMove(HeartsCard.Herz2);

            assertFalse(isValidMove);

            boolean isValidMove2 = ruleset.isValidMove(HeartsCard.Caro3);

            assertTrue(isValidMove2);
        }

        @Test
        public void testIsValidMoveOnlyHearts() {
               ruleset.giveACard(playerState1, HeartsCard.Herz2);
            ruleset.giveACard(playerState1, HeartsCard.Herz5);
            ruleset.giveACard(playerState2, HeartsCard.Caro3);
            ruleset.giveACard(playerState2, HeartsCard.Caro6);
            ruleset.giveACard(playerState3, HeartsCard.Pik4);
            ruleset.giveACard(playerState3, HeartsCard.Pik5);
         ruleset.giveACard(playerState4, HeartsCard.Pik1);
            ruleset.giveACard(playerState4, HeartsCard.Herz7);

            boolean isValidMove = ruleset.isValidMove(HeartsCard.Herz2);

            assertTrue(isValidMove);

            boolean isValidMove2 = ruleset.isValidMove(HeartsCard.Herz5);

            assertTrue(isValidMove2);
        }

}
```

## 6.3 getWinner

```
package Ruleset;

import static org.junit.Assert.*;
```

```
import java.util.List;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import testKlassen.TestPlayer;
import ComObjects.ComObject;
import ComObjects.ComRuleset;
import ComObjects.MsgGameEnd;
import Server.GameServer;
import Server.LobbyServer;
import Server.Player;

public class TestHeartsWinner {

LobbyServer lobbyServer;

	GameServer gameServer;

	ServerRuleset heartsServerRuleset;

	TestPlayer blue;

	TestPlayer white;

	TestPlayer orange;

	TestPlayer brown;

	List<ComObject> inputList;

	ComRuleset comObject;

	MsgGameEnd endMsg;

	String winner;

	@Before
	public void setUp() {
		lobbyServer = new LobbyServer();
		blue = new TestPlayer(lobbyServer, null, null);
		white = new TestPlayer(lobbyServer, null, null);
		orange = new TestPlayer(lobbyServer, null, null);
		brown = new TestPlayer(lobbyServer, null, null);
	}

	@After
	public void tearDown() {
		blue = null;
		white = null;
		orange = null;
		brown = null;
		lobbyServer = null;
		gameServer = null;
		inputList = null;
		comObject = null;
		endMsg = null;
		winner = null;
	}

	@Test
	public void testGetWinner() {
```

```
            gameServer = new GameServer(lobbyServer, blue, "Test Game",
                RulesetType.Wizard, "", false);
            gameServer.addPlayer(white);
            gameServer.addPlayer(orange);
            gameServer.addPlayer(brown);

            heartsServerRuleset = new ServerWizard(gameServer);
            heartsServerRuleset.addPlayerToGame("Mr. Blue");
            heartsServerRuleset.addPlayerToGame("Mr. White");
            heartsServerRuleset.addPlayerToGame("Mr. Orange");
            heartsServerRuleset.addPlayerToGame("Mr. Brown");
            heartsServerRuleset.setPoints(heartsServerRuleset.getPlayerState("Mr.
                Blue"),80);
            heartsServerRuleset.setPoints(heartsServerRuleset.getPlayerState("Mr.
                White"),20);
            heartsServerRuleset.setPoints(heartsServerRuleset.getPlayerState("Mr.
                Orange"),60);
            heartsServerRuleset.setPoints(heartsServerRuleset.getPlayerState("Mr.
                Brown"),110);
            heartsServerRuleset.setGamePhase(GamePhase.Ending);
            heartsServerRuleset.calculateRoundOutcome();

            assertTrue(heartsServerRuleset.getWinner().equals("Mr. Brown"));

            inputList = blue.getServerInput();
            comObject = (ComRuleset) inputList.get(1);
            endMsg = (MsgGameEnd) comObject.getRulesetMessage();
            winner = endMsg.getWinner();
            assertEquals("Nachricht an Blue", "Mr. Brown", winner);

            inputList = white.getServerInput();
            comObject = (ComRuleset) inputList.get(1);
            endMsg = (MsgGameEnd) comObject.getRulesetMessage();
            winner = endMsg.getWinner();
            assertEquals("Nachricht an White", "Mr. Brown", winner);

            inputList = orange.getServerInput();
            comObject = (ComRuleset) inputList.get(1);
            endMsg = (MsgGameEnd) comObject.getRulesetMessage();
            winner = endMsg.getWinner();
            assertEquals("Nachricht an Orange", "Mr. Brown", winner);

            inputList = brown.getServerInput();
            comObject = (ComRuleset) inputList.get(1);
            endMsg = (MsgGameEnd) comObject.getRulesetMessage();
            winner = endMsg.getWinner();
            assertEquals("Nachricht an Brown", "Mr. Brown", winner);
        }
}
```

```
package Ruleset;

import static org.junit.Assert.*;

import java.util.List;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import testKlassen.TestPlayer;
import ComObjects.ComChatMessage;
import ComObjects.ComObject;
```

```
import ComObjects.ComRuleset;
import ComObjects.MsgGameEnd;
import Server.GameServer;
import Server.LobbyServer;

/**
 * Testet ob der richtige Sieger ermittelt wird und ob jedem Mitspieler
 * der richtige Sieger mitgeteilt wird
 */
public class TestWizardWinner {

	LobbyServer lobbyServer;

	GameServer gameServer;

	ServerRuleset wizardServerRuleset;

	TestPlayer blue;

	TestPlayer white;

	TestPlayer orange;

	TestPlayer brown;

	List<ComObject> inputList;

	ComRuleset comObject;

	MsgGameEnd endMsg;

	String winner;

	@Before
	public void setUp() {
		lobbyServer = new LobbyServer();
		blue = new TestPlayer(lobbyServer, null, null);
		white = new TestPlayer(lobbyServer, null, null);
		orange = new TestPlayer(lobbyServer, null, null);
		brown = new TestPlayer(lobbyServer, null, null);
	}

	@After
	public void tearDown() {
		blue = null;
		white = null;
		orange = null;
		brown = null;
		lobbyServer = null;
		gameServer = null;
		inputList = null;
		inputList = null;
		comObject = null;
		endMsg = null;
		winner = null;
	}

	@Test
	public void testGetWinner() {

		gameServer = new GameServer(lobbyServer, blue, "Test Game",
		    RulesetType.Wizard, "", false);
		gameServer.addPlayer(white);
```

```
            gameServer.addPlayer(orange);
            gameServer.addPlayer(brown);

            wizardServerRuleset = new ServerWizard(gameServer);
            wizardServerRuleset.addPlayerToGame("Mr. Blue");
            wizardServerRuleset.addPlayerToGame("Mr. White");
            wizardServerRuleset.addPlayerToGame("Mr. Orange");
            wizardServerRuleset.addPlayerToGame("Mr. Brown");
            wizardServerRuleset.setPoints(wizardServerRuleset.getPlayerState("Mr.
                Blue"),80);
            wizardServerRuleset.setPoints(wizardServerRuleset.getPlayerState("Mr.
                White"),200);
            wizardServerRuleset.setPoints(wizardServerRuleset.getPlayerState("Mr.
                Orange"),130);
            wizardServerRuleset.setPoints(wizardServerRuleset.getPlayerState("Mr.
                Brown"),240);
            wizardServerRuleset.setGamePhase(GamePhase.Ending);
            wizardServerRuleset.calculateRoundOutcome();

            assertTrue(wizardServerRuleset.getWinner().equals("Mr. Brown"));

            inputList = blue.getServerInput();
            comObject = (ComRuleset) inputList.get(1);
            endMsg = (MsgGameEnd) comObject.getRulesetMessage();
            winner = endMsg.getWinner();
            assertEquals("Nachricht an Blue", "Mr. Brown", winner);

            inputList = white.getServerInput();
            comObject = (ComRuleset) inputList.get(1);
            endMsg = (MsgGameEnd) comObject.getRulesetMessage();
            winner = endMsg.getWinner();
            assertEquals("Nachricht an White", "Mr. Brown", winner);

            inputList = orange.getServerInput();
            comObject = (ComRuleset) inputList.get(1);
            endMsg = (MsgGameEnd) comObject.getRulesetMessage();
            winner = endMsg.getWinner();
            assertEquals("Nachricht an Orange", "Mr. Brown", winner);

            inputList = brown.getServerInput();
            comObject = (ComRuleset) inputList.get(1);
            endMsg = (MsgGameEnd) comObject.getRulesetMessage();
            winner = endMsg.getWinner();
            assertEquals("Nachricht an Brown", "Mr. Brown", winner);
      }
}
```

## 6.4 QuitPlayer

```
ackage Server;


import static org.junit.Assert.*;

import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;
```

```
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import test.TestGameServer;
import test.TestLobbyServer;
import test.TestPlayer;

import ComObjects.*;
import ComObjects.ComWarning;
import Ruleset.RulesetType;

public class QuitGameTest {

	TestLobbyServer lobby;

	TestPlayer player1;

	TestPlayer player2;
	TestPlayer player3;
	TestPlayer player4;

	TestGameServer game;

	ComClientQuit quit;

	@Before
	public void setUp() throws Exception {
		lobby = new TestLobbyServer();

		player1 = new TestPlayer(lobby, null, null);
		player1.setName("MrBlue");
		lobby.addPlayer(player1);

		player2 = new TestPlayer(lobby, null, null);
		player2.setName("MrWhite");
		player3 = new TestPlayer(lobby, null, null);
		player3.setName("MrPink");
		player4 = new TestPlayer(lobby, null, null);
		player4.setName("MrRed");

		game = new TestGameServer(lobby, player1, "MrBluesGame",
		    RulesetType.Hearts, null, false);

		game.addPlayer(player2);
		game.addPlayer(player3);
		game.addPlayer(player4);

		quit = new ComClientQuit();
	}

	@After
	public void tearDown() throws Exception {
		lobby = null;
		player1 = null;
		player2 = null;
		player3 = null;
		player4 = null;
		game = null;
	}

	@Test
	public void testPlayerQuitGame() throws IOException{
```

```
            player1.changeServer(game);
            assertTrue(game.initLobby().getPlayerList().contains(player1.getName()));

            ComInitLobby initLobby = lobby.initLobby();
            ComWarning warning = new ComWarning("Ein Spieler hat das Spiel
                verlassen");

            player1.injectComObject(quit);

            assertFalse(lobby.initLobby().getGameList().contains(game));
            assertTrue(lobby.initLobby().getPlayerList().contains(player1.getName()));
            assertTrue(lobby.initLobby().getPlayerList().contains(player1.getName()));
            assertTrue(lobby.initLobby().getPlayerList().contains(player1.getName()));
            assertTrue(lobby.initLobby().getPlayerList().contains(player1.getName()));

            assertTrue(player1.getServerInput().get(0).getClass() ==
                initLobby.getClass());
            assertTrue(player1.getServerInput().get(1).getClass() ==
                warning.getClass());
            assertTrue(player2.getServerInput().get(0).getClass() ==
                initLobby.getClass());
            assertTrue(player2.getServerInput().get(1).getClass() ==
                warning.getClass());
            assertTrue(player3.getServerInput().get(0).getClass() ==
                initLobby.getClass());
            assertTrue(player3.getServerInput().get(1).getClass() ==
                warning.getClass());
            assertTrue(player4.getServerInput().get(0).getClass() ==
                initLobby.getClass());
            assertTrue(player4.getServerInput().get(1).getClass() ==
                warning.getClass());
      }
}
```

## 6.5 Chat

```
package chat;

import static org.junit.Assert.*;

import org.junit.Test;
import org.junit.After;
import org.junit.Before;

import testKlassen.TestMessageListenerThread;
import testKlassen.TestObserver;
import Client.MessageListenerThread;
import Client.ClientModel;
import ComObjects.ComChatMessage;

public class ClientModelChatTest {

      ComChatMessage testMessage;

      ClientModel testModel;

      TestObserver testObserver;

      TestMessageListenerThread testNetIO;
```

```
      String testText;

      @Before
   public void setUp() {
            testNetIO = new TestMessageListenerThread();
            testObserver = new TestObserver();
            testMessage = new ComChatMessage("Hello Test!");
            testModel = new ClientModel((MessageListenerThread) testNetIO);
            testNetIO.setModel(testModel);
            testModel.addObserver(testObserver);
   }

   @After
   public void tearDown() {
      testNetIO = null;
      testMessage = null;
      testModel = null;
      testObserver = null;
   }

      @Test
      public void testSendChatMessage() {
            String inputText = "Hello Test!";
            testModel.sendChatMessage(inputText);
            testText = ((ComChatMessage)
                testNetIO.getModelInput().get(0)).getChatMessage();
            assertEquals("Vergleich der gesendeten Chatnachrichten", testText,
                inputText);
      }

      @Test
      public void testReceiveChatMessage() {
            testNetIO.injectComObject(testMessage);
            assertTrue("Vergleich der empfangenen Chatnachrichten",
                        testObserver.getChatMessage().equals(testMessage.getChatMessage()));
      }
}
```

```
package chat;

import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import testKlassen.TestPlayer;
import Server.LobbyServer;
import ComObjects.ComChatMessage;

public class LobbyServerChatTest {

      ComChatMessage testMessage;

      LobbyServer testServer;

      TestPlayer player1;

      TestPlayer player2;

      TestPlayer player3;

      String testText1;
```

```
	String testText2;

	String testText3;

	@Before
	public void setUp() {
		testMessage = new ComChatMessage("Hello Test!");
		testServer = new LobbyServer();
		player1 = new TestPlayer(testServer, null, null);
		player2 = new TestPlayer(testServer, null, null);
		player3 = new TestPlayer(testServer, null, null);
	}

	@After
	public void tearDown() {
		testMessage = null;
		testServer = null;
		player1 = null;
		player2 = null;
		player3 = null;
		testText1 = null;
		testText2 = null;
		testText3 = null;
	}

	@Test
	public void testReceiveMessagePlayerComChatMessage() {
		String messageToMatch = testMessage.getChatMessage();
		testServer.addPlayer(player1);
		testServer.addPlayer(player2);
		testServer.addPlayer(player3);
		player1.injectComObject(testMessage);
		testText1 = ((ComChatMessage)
		    player1.getServerInput()).getChatMessage();
		testText2 = ((ComChatMessage)
		    player2.getServerInput()).getChatMessage();
		testText3 = ((ComChatMessage)
		    player3.getServerInput()).getChatMessage();
		assertEquals("Nachricht an Spieler 1", messageToMatch, testText1);
		assertEquals("Nachricht an Spieler 2", messageToMatch, testText2);
		assertEquals("Nachricht an Spieler 3", messageToMatch, testText3);
	}
}
```

# 7 Implementierungsplan

Es werden für jeden Milestone die einzelnen Arbeitspakete angegeben. Die angegebenen Klassen werden nicht sofort vollständig implmentiert, sondern mit den vom Arbeitspaket und Milestone verlangten Funktionen ausgestattet.

## 7.1 Milestone 1

Für den ersten Milestone werden folgende Funktionen angestrebt:
Der Nutzer kann sich im Login-Fenster anmelden und die Lobby betreten. Er kann ein Spiel erstellen und offenen Spielen beitreten. Das wird in der Lobby angezeigt, man gelangt jedoch noch nicht ins Wartefenster. Nebenläufig dazu wird die Datenschicht der Regelwerke implementiert.

- View(Login+Lobby) Dauer: 8 Std.
  Klassen: Login, Lobby, Warning, ClientController
- Client(Login) Dauer 8 Std.
  Klassen: ClientMain, ClientModel, MessageListener Thread, ClientState, ViewNotification
- Server(Login) Dauer 16 Std.
  Klassen: Server, ServerMain, LobbyServer, Player, ClientListenerThread, ComObject, ComLoginRequest, ComClientQuit, ComServerAcknowledgement, ComWarning
- Ruleset(Daten) Dauer 20 Std.
  Klassen: Card, Colour, HeartsCard, WizCard, OtherData, WizData, HeartsData, GameClientUpdate, GameState, PlayerState, RulesetType
- Client(Lobby) Dauer 8 Std.
  Klassen: ClientModel
- Server(Lobby) Dauer 8 Std.
  Klassen: LobbyServer, ComChatMessage, ComLobbyUpdateGamelist, ComJoinRequest, ComInitLobby, ComUpdatePlayerlist
- View(Create+Join) Dauer 8 Std.
  Klassen: Password, CreateGame, ClientController
- Client(Create+Join) Dauer 8 Std.
  Klassen: ClientModel
- Server(Create+Join) Dauer 12 Std.
  Klassen: LobbyServer, ComJoinRequest, ComCreateGameRequest

## 7.2 Milestone 2

Für den zweiten Milestone werden folgende Funktionen angestrebt:
Beim Beitreten oder Erstellen eines Spiels gelangt man ins Wartefenster. Der Spielleiter kannn hier Spieler entfernen. Diese gelangen zurück in die Lobby. Das Spiel kann noch nicht gestartet werden, aber das Regelwerk wird bereits serverseitig implementiert.

- Ruleset(Wizard-Server) Dauer 30 Std.
  Klassen: ServerRuleset, ServerWizard, RulesetMessage, MsgCard, MsgCardRequest, MsgGameEnd, MsgNumber, Msg NumberRequest, MsgSelection, MsgSelectionRequest, MsgUser
- View(GameLobby) Dauer 8 Std.
  Klassen: GameLobby, ClientController
- Client(GameLobby) Dauer 8 Std.
  Klassen: ClientModel
- Server(GameLobby) Dauer 10 Std.
  KLassen: LobbyServer, GameServer, ComBeenKicked, ComClientLeave, ComInitGameLobby, ComKickPlayerRequest, ComStartGame
- View(Game) Dauer 20 Std.
  Klassen: Game, GamePanel, OtherPlayer, OwnHand, ViewCard, DrawDeck, DiscardPile, ScoreWindow, ClientController
- Client(Game) Dauer 14 Std.
  Klassen: ClientModel

### 7.3 Milestone 3

Für den dritten Milestone werden folgende Funktionen angestrebt:
Es kann schon eine vollständige Partie Wizard gespielt werden.

- Server(Game) Dauer 4 Std.
  Klassen: GameServer, ComStartGame, ComRuleset, ComGameEnd
- Ruleset(Wizard-Client) Dauer 12 Std.
  Klassen: ClientWizard
- View(WizardWindows) Dauer 4 Std.
  Klassen: ChooseItem, InputNumber, ClientController
- Client(Wizard) Dauer 6 Std.
  Klassen: ClientModel
- View(HeartsWindows) Dauer 2 Std.
  Klassen: ChooseCards, ClientController
- Client(Hearts) Dauer 4 Std.
  Klassen: ClientModel
- Ruleset(Hearts-Server) Dauer 16 Std.
  Klassen: ServerHearts

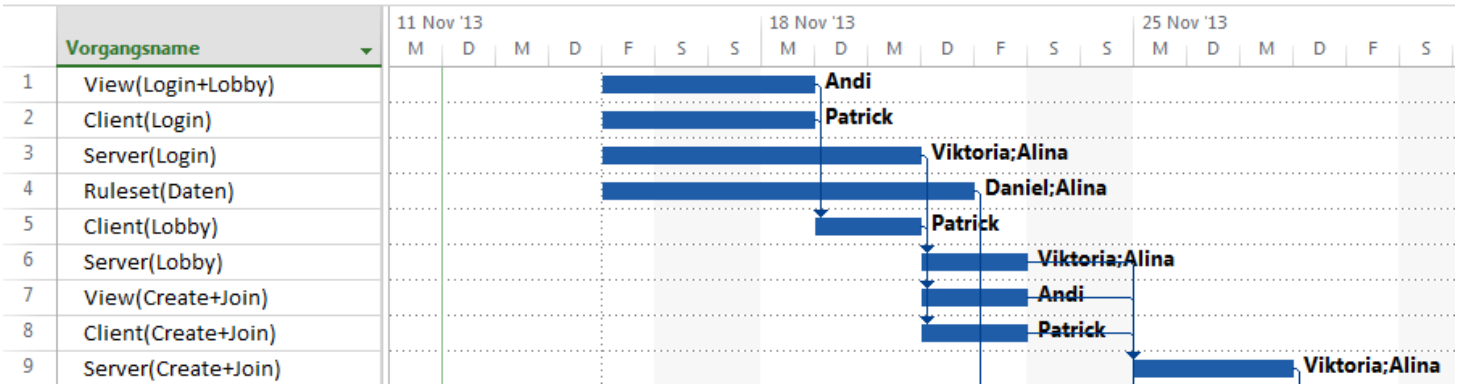
### 7.4 Finale Version

Die finale Version enthält die volle Funktionalität des Programs. Es können also sowohl Wizard als auch Hearts gespielt werden.

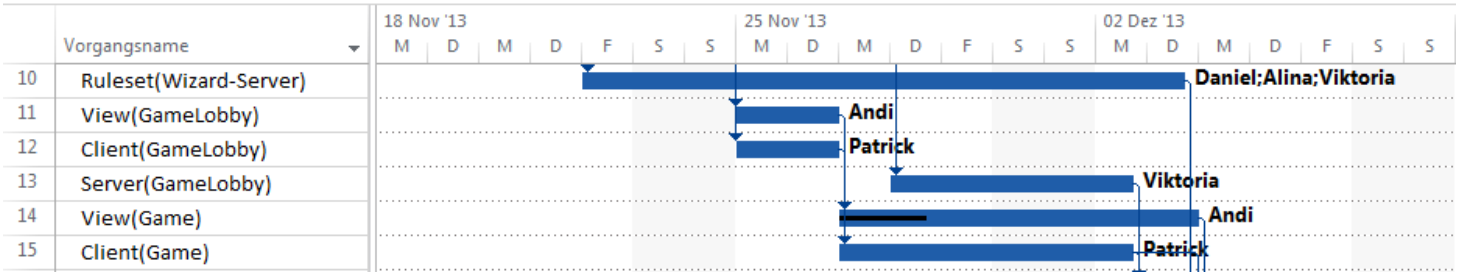
- Ruleset(Hearts-Client) Dauer 10Std.
  Klassen: ClientHearts
- ViewPolishing(evtl Tests) Dauer 10Std
  Verbesserungen an der bisherigen Implementierung. Gegebenfalls Schreiben von zusätzlichen Tests
- ClientPolishing(evtl Tests) Dauer 10Std
  Verbesserungen an der bisherigen Implementierung. Gegebenfalls Schreiben von zusätzlichen Tests
- ServerPolishing(evtl Tests) Dauer 10Std
  Verbesserungen an der bisherigen Implementierung. Gegebenfalls Schreiben von zusätzlichen Tests
- RulesetPolishing(evtl Tests) Dauer 10Std
  Verbesserungen an der bisherigen Implementierung. Gegebenfalls Schreiben von zusätzlichen Tests

## 7.5 Gantt-Diagramme

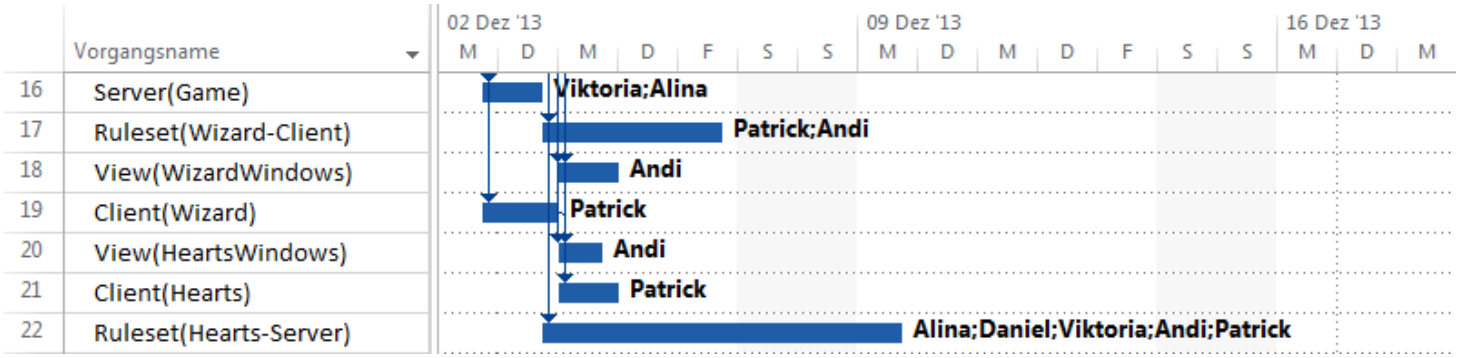
Milestone 1:



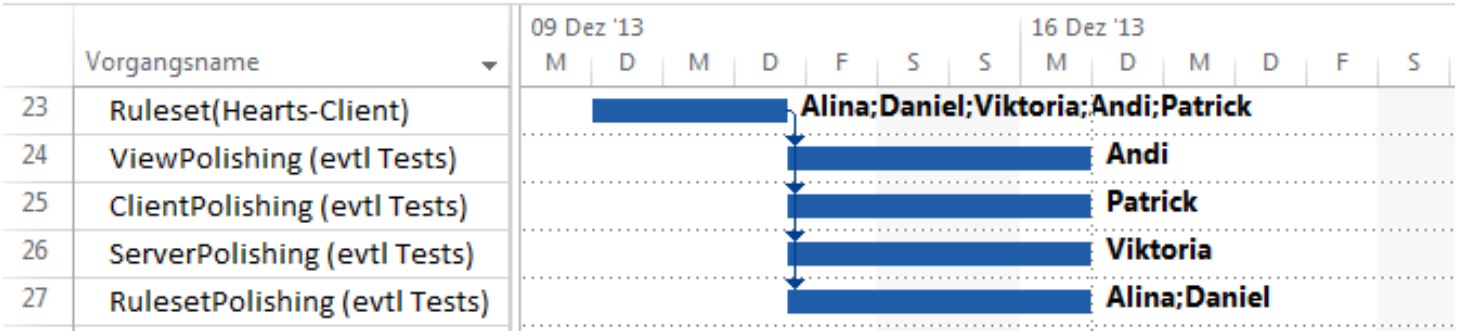

Milestone 2:




Milestone 3:




Finale Version:


Vorgangsname
09 Dez '13
16 Dez '13
M D M D F S S M D M D F S
23 Ruleset(Hearts-Client)
24 ViewPolishing (evtl Tests)
25 ClientPolishing (evtl Tests)
26 ServerPolishing (evtl Tests)
27 RulesetPolishing (evtl Tests)
Alina;Daniel;Viktoria;Andi;Patrick
Andi
Patrick
Viktoria
Alina;Daniel