Softwaretechnikpraktikum WS 2018/2019

Interface Dokument

Version 1.1.0

Komitee-Mitglieder:

Alexander Meinhold
Hendrik Huecker
Jan Oberbeckmann
Linus Jungemann
Lukas Ostermann
Lukas Rink
Mengshi Ma
Viktor Weigandt
Wael Diab

Betreuer:

Mario Treiber

14. Dezember 2018

Inhaltsverzeichnis

1 Einleitung

Um die Kommunikation sicherzustellen, hat das Interface Komitee dieses Dokument für das Qwirkle-Projekt im Zuge des Software Praktikums im Wintersemester 2018/2019 entworfen. Es hat das Ziel, die Kommunikation zwischen Server und Clients zu beschreiben. So ermöglichen wir, dass diese Bestandteile unabhängig vom Entwickler korrekt miteinander kommunizieren können.

2 Glossar

- Spieler: Spieler ist ein Client, der einem Spiel als Spieler beigetreten ist.
- Beobachter: Beobachter ist ein Client, mit ClientType Beobachter oder ein ClientType Player bzw. EnginePlayer der dem Game als Beobachter beigetreten ist. Ein Player oder EnginePlayer bleibt bis zum Ende des Games ein Beobachter.
- Client: Mit Client sind alle Clients gemeint, die mit dem gleichen Game verbunden sind. Unabhängig davon, ob es ein Spieler oder Beobachter ist.
- Lobby: Alle Clients landen nach dem Verbinden in der Lobby. Jeder Server hat genau eine Lobby. Eine Lobby hat beliebig viele Spiele.
- Spiel: Eine Spielsitzung in der Lobby eines Servers. Die Anzahl der Spieler wird in der Spielkonfiguration bestimmt. Es können beliebig viele Beobachter in einem Spiel sein.
- aktiver Spieler: Der aktive Spieler ist der Spieler eines Spiels, der momentan am Zug ist.

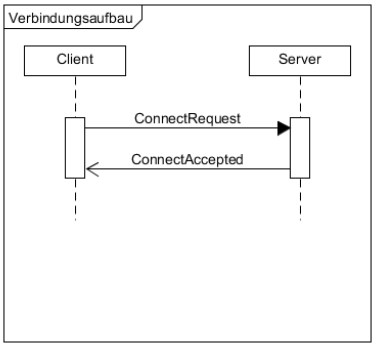
3 Verwendung

Das Interface-Komitee stellt eine Implementierung, des in diesem Dokument definierten Interfaces, bereit. Dieses ist wie folgt zu benutzen:
Jeder Entwickler-Gruppe wird dieses Dokument, sowie die Interface-Library (.jar) mit der dazu gehörenden README.md, bereit gestellt. Wir empfehlen vor der Nutzung der Library dieses Dokument und die README gründlich zu lesen. Bei Fragen wenden Sie sich an das Komitee-Mitglied Ihrer Gruppe.
Die im Interface bereitgestellten Interface-Objekte werden mittels der Parser-Klasse in einen JSON-String umgewandelt. Dieser muss nun mittels einer selbst zu implementierenden Netzwerkverbindung übertragen werden. Der Empfänger übergibt diesen String wieder der Parser-Klasse, welche dann wiederum ein Message-Objekt daraus generiert.
Die Netzwerkverbindung hat unverschlüsselt über TCP zu erfolgen, sodass sich der Client per IP und Port am Server anmelden kann. Jede Nachricht hat in einer Zeile zu erfolgen, sodass das End-Of-Line Symbol mit dem End-Of-Message Symbol gleichzusetzen ist. Die Codierung der Messages hat als UTF8 zu erfolgen.

4 Netzwerk Kommunikation

4.1 Verbindung zum Server aufbauen und beenden

ID	Pseudoname	Sender	Empfänger	Aktion
100	ConnectRequest	Client	Server	Teilt dem Server mit, dass ein Client dem Server beitreten möchte.
101	ConnectAccepted	Server	Client	Teilt dem Client mit, dass die *ConnectRequest* (**100**) erfolgreich war.



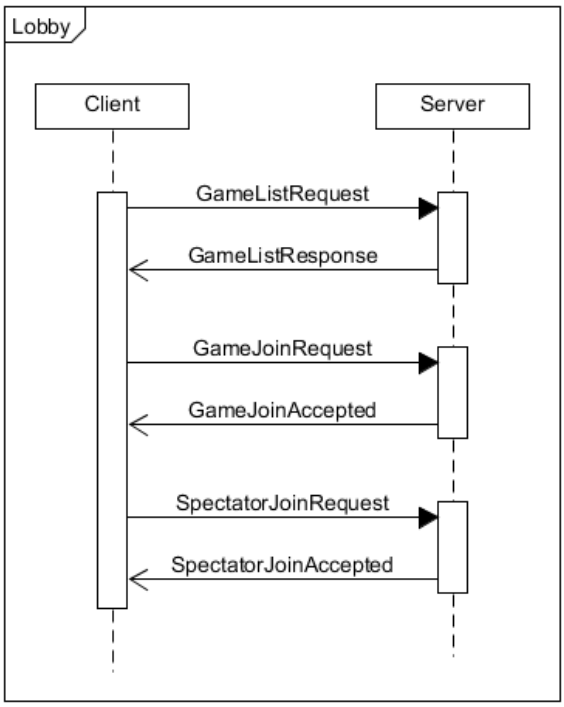
In dem Sequenz-Diagramm "Verbindungsaufbau" sieht man, wie der Verbindungsaufbau abläuft.

Ein Client, der sich mit dem Server verbinden möchte, schickt dem Server ein *ConnectRequest* (**100**). Erhält der Client daraufhin ein *ConnectAccepted* (**101**), hat sich der Client erfolgreich verbunden. Wird der *ConnectRequest* abgelehnt, erhält der Client ein *NotAllowed* (**920**). Das kann auftreten, wenn der Client bereits verbunden ist und schon eine *ClientId* besitzt. (siehe Abschnitt 5 Fehlerbehandlung)

Verlässt ein erfolgreich verbundener Client den Socket des Servers, ist das gleichbedeutend mit einem Disconnect. Ein Wiederherstellen dieser Sitzung ist nicht möglich. Er muss sich erneut mit *ConnectRequest* verbinden.

4.2 Einem Spiel beitreten & Ingame Chat

ID	Pseudoname	Sender	Empfänger	Aktion
300	GameListRequest	Client	Server	Der Client fordert alle Spiele an, die bald starten, schon gestartet wurden oder vor Kurzem beendet wurden.
301	GameListResponse	Server	Client	Der Server antwortet auf *GameListRequest* (**300**) und sendet alle Spiele, die bald starten, schon gestartet wurden oder vor Kurzem beendet wurden.
302	GameJoinRequest	Spieler	Server	Teilt dem Server mit, dass der Spieler einem Spiel als Spieler beitreten möchte.
303	GameJoinAccepted	Server	Spieler	Der Server antwortet auf *GameJoinRequest* (**302**) und teilt mit, dass der Spieler dem Spiel beigetreten ist.
304	SpectatorJoinRequest	Client	Server	Teilt dem Server mit, dass der Client einem Spiel als Beobachter beitreten möchte.
305	SpectatorJoinAccepted	Server	Client	Der Server antwortet auf *SpectatorJoinRequest* (**304**) und teilt mit, dass der Client dem Spiel beigetreten ist.
306	MessageSend	Client	Server	Sendet eine Chat-Nachricht an den Server.
307	MessageSignal	Server	berechtigte Clients	Sendet eine neue Chat-Nachricht an Clients, die diese Nachricht sehen dürfen.

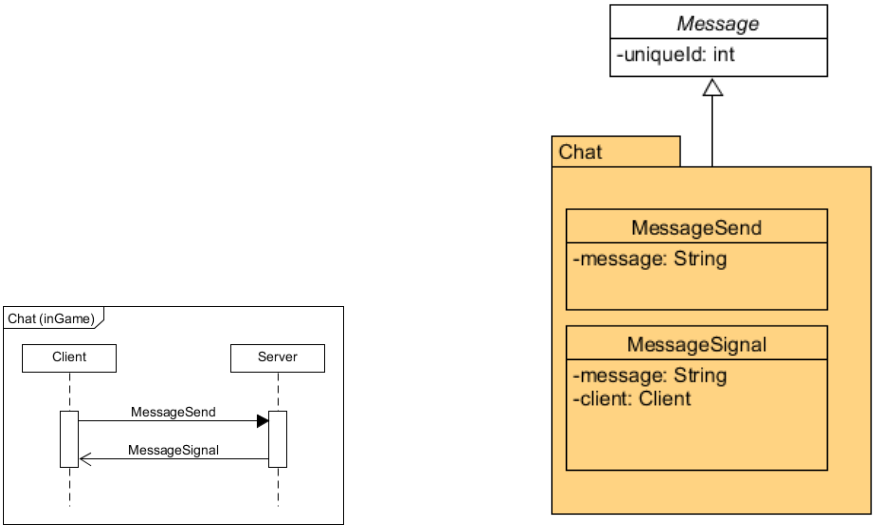


In dem Sequenz-Diagramm "Lobby" sieht man, welche Möglichkeiten ein Client nach dem Verbinden hat.

Ein Client kann mittels *GameListRequest* (**300**) alle existierende Spiele anfragen. Der Server antwortet darauf mit einer *GameListResponse* (**301**). Die *GameListResponse* umfasst eine ausführliche Liste aller Spiele. (siehe *Game* in Abschnitt 4.4 Erläuterung der Implementierung).

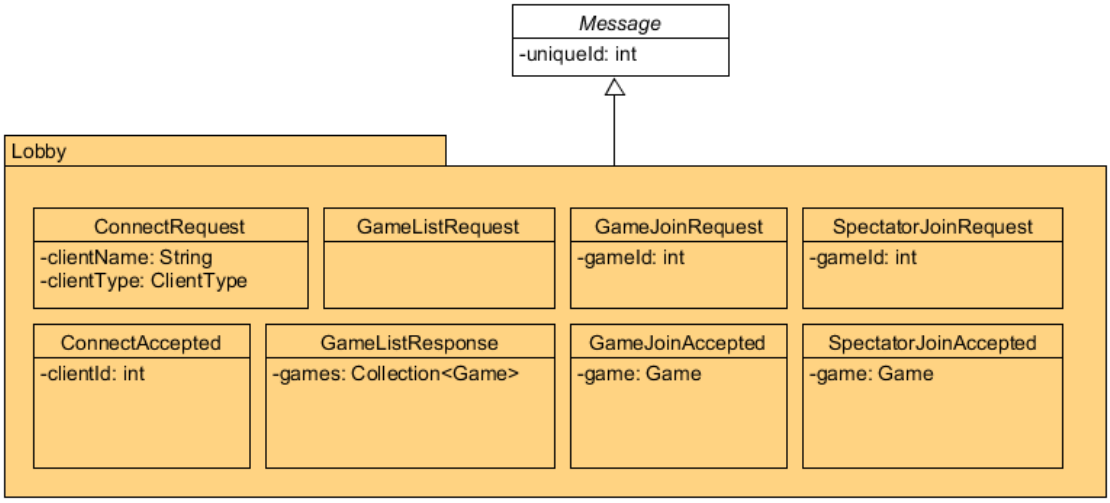
Mit *GameJoinRequest* (**302**) und *SpectatorJoinRequest* (**304**) kann ein Client einem Spiel als Spieler, bzw. Beobachter beitreten. Erhält der Spieler danach ein *GameJoinAccepted* (**303**), bzw. *SpectatorJoinAccepted* (**305**) ist er erfolgreich dem Spiel beigetreten. Wird ein *JoinRequest* vom Server abgelehnt, schickt er dem Client ein *NotAllowed* (**920**) (siehe Abschnitt 5 Fehlerbehandlung)

Zusätzlich kann der Server jederzeit ein *GameJoinAccepted* (**303**) oder *SpectatorJoinAccepted* (**305**) an Clients in der Lobby schicken. Wenn ein Client ein *JoinAccepted* erhält, ohne ein *JoinRequest* verschickt zu haben, wurde der Client vom Server zu einem Spiel hinzugefügt.

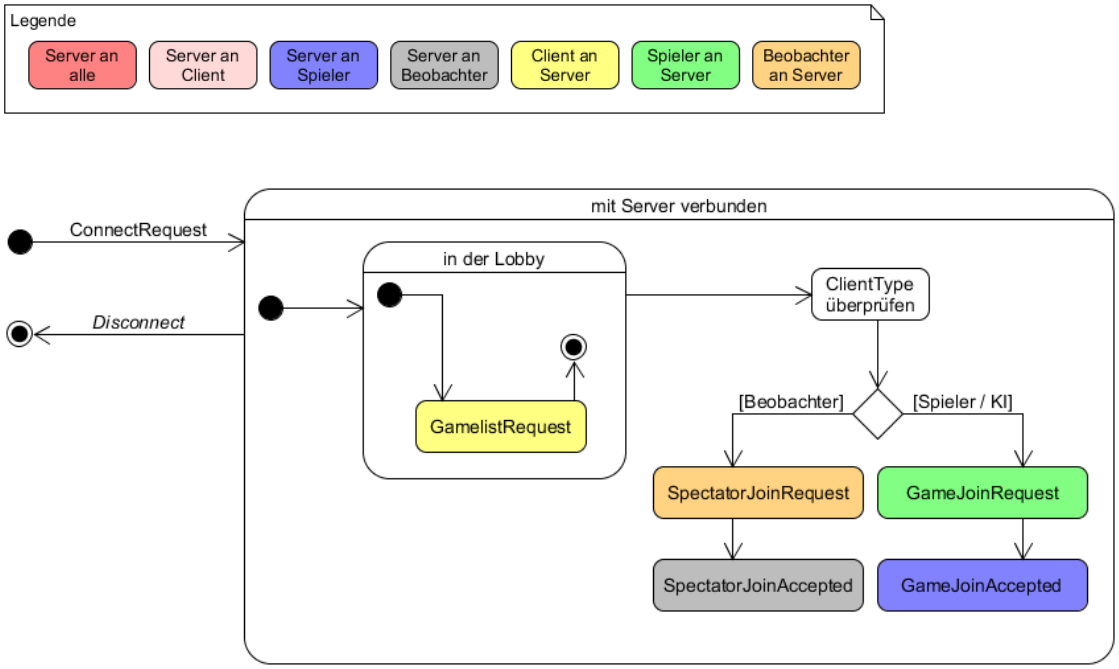


Zusätzlich unterstützt jeder Server und Client eine Chatfunktion in einem Spiel. Ein Client kann mittels *MessageSend* (**306**) eine Chat-Nachricht an den Server schicken. Erhält der Server ein *MessageSend* prüft er den Status des Clients und wählt eine der folgenden Reaktionen:

- Client ist Spieler in einem Spiel. Server sendet *MessageSignal* (**307**) an alle Spieler und Beobachter in diesem Spiel.
- Client ist Beobachter in einem Spiel. Server sendet *MessageSignal* (**307**) an alle Beobachter in diesem Spiel.



Das hier zu sehende Klassendiagramm "Lobby" zeigt alle Nachrichten im Detail. Zusätzlich zu den IDs und den Beschreibungen weiter oben, sieht man hier die einzelnen Klassen und ihre Attribute.

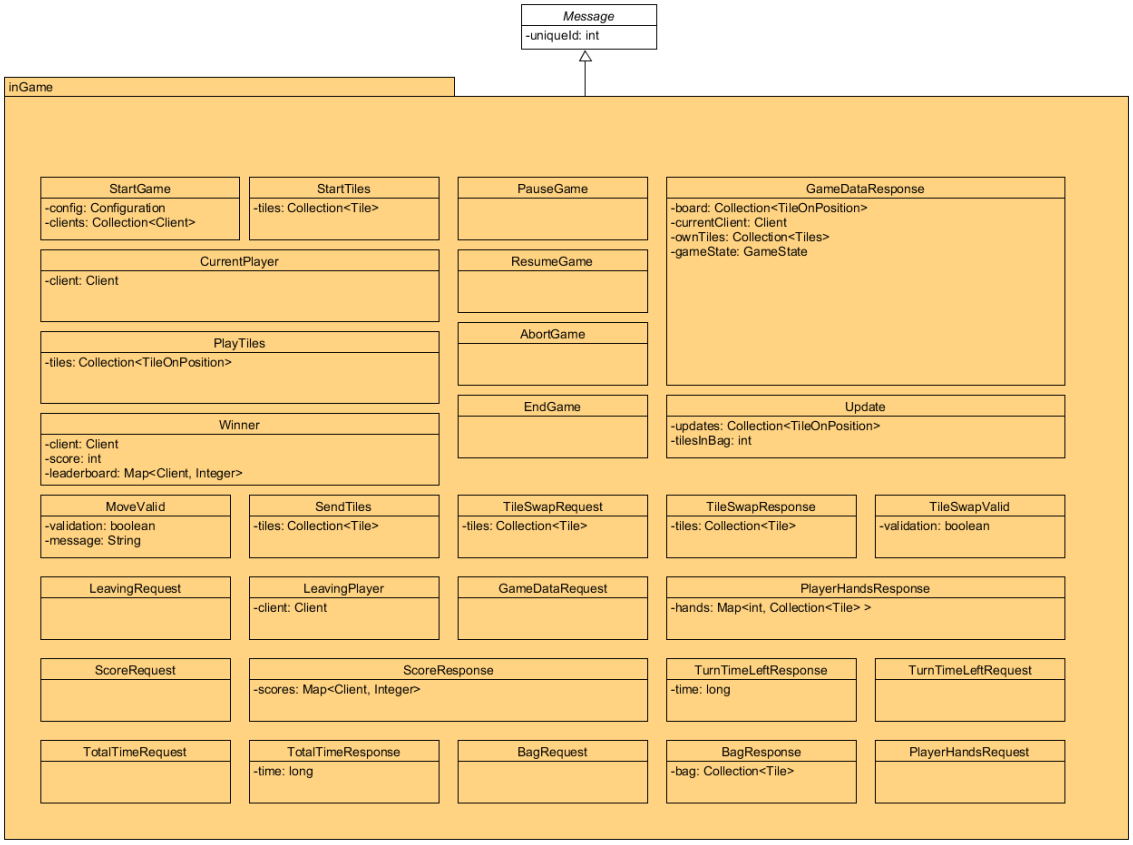


In diesem State-Chart ist der Ablauf in der Lobby noch einmal formal festgehalten.

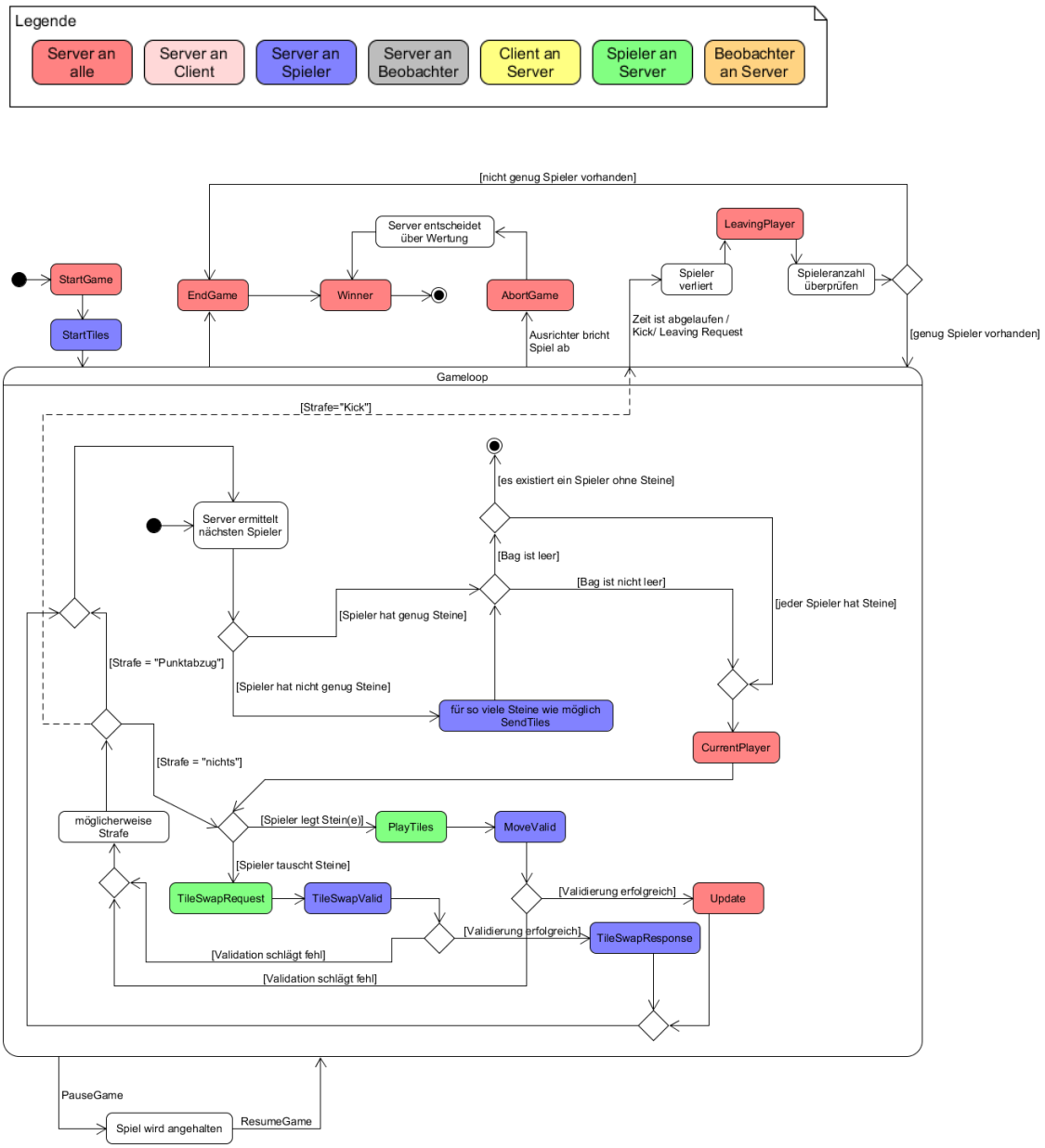
4.3 Spiellogik

ID	Pseudoname	Sender	Empfänger	Aktion
400	StartGame	Server	Alle Clients	Benachrichtigt über den Spielstart und sendet die Konfiguration sowie eine Liste aller Spieler an alle Clients.
401	EndGame	Server	Alle Clients	Benachrichtigt über das reguläre Ende des Spiels, welches herkömmlich durch den Sieg eines Spielers hervorgerufen wird oder dadurch eintritt, dass nur noch ein Spieler am Spiel beteiligt ist.
402	AbortGame	Server	Alle Clients	Benachrichtigt über den Abbruch des Spiels, der durch den Ausrichter herbeigeführt wurde.
403	PauseGame	Server	Alle Clients	Benachrichtigt darüber, dass der Ausrichter das Spiel pausiert hat.
404	ResumeGame	Server	Alle Clients	Benachrichtigt darüber, dass der Ausrichter das Spiel fortsetzt.
405	LeavingRequest	Client	Server	Teilt dem Server mit, dass der Client das Spiel verlässt.
406	LeavingPlayer	Server	Alle Clients	Teilt mit, dass ein Spieler das Spiel verlassen hat.
407	Winner	Server	Alle Clients	Teilt allen Clients dieses Spiels den Gewinner des Spiels und die Punkte aller Spieler mit.
408	StartTiles	Server	Spieler	Sendet dem Spieler die erste Spielhand mit der konfigurierten Anzahl an Spielsteinen.
409	CurrentPlayer	Server	Alle Clients	Teilt mit, welcher Spieler am Zug ist.
410	SendTiles	Server	Spieler	Füllt die Hand des Spielers mit Steinen aus dem Bag.
411	TileSwapRequest	Spieler	Server	Der Spieler sendet dem Server eigene Steine, die er tauschen möchte.
412	TileSwapValid	Server	Spieler	Der Server teilt dem Spieler mit, ob ein *TileSwapRequest* (**411**) einem gültigen Zug entspricht.
413	TileSwapResponse	Server	Spieler	Der Server tauscht die Steine eines *TileSwapRequest* (**411**) ein und sendet dem Spieler neue Steine.

ID	Pseudoname	Sender	Empfänger	Aktion
414	PlayTiles	Spieler	Server	Der Spieler teilt dem Server mit, welche Steine er wohin legen möchte.
415	MoveValid	Server	Spieler	Der Server teilt dem Spieler mit, ob ein *PlayTiles* (**414**) einem gültigen Zug entspricht.
416	Update	Server	Alle Clients	Der Server teilt mit, welche Steine an welcher Position neu auf das Spielbrett gekommen sind.
417	ScoreRequest	Client	Server	Sendet eine Anfrage nach den Scores an den Server.
418	ScoreResponse	Server	Client	Antwortet auf *ScoreRequest* (**417**) und sendet die Scores.
419	TurnTimeLeftRequest	Client	Server	Sendet eine Anfrage nach der verbleibenden Zeit für den aktuellen Zug an den Server.
420	TurnTimeLeftResponse	Server	Client	Antwortet auf *TurnTimeLeftRequest* (**419**) und sendet die verbleibende Zeit in Millisekunden für den aktuellen Spielzug.
421	TotalTimeRequest	Client	Server	Sendet eine Anfrage nach der Gesamtlaufzeit der Partie.
422	TotalTimeResponse	Server	Client	Antwortet auf *TotalTimeRequest* (**421**) und sendet die Gesamtlaufzeit der Partie in Millisekunden.
423	BagRequest	Beobachter	Server	Sendet eine Anfrage nach dem Beutelinhalt.
424	BagResponse	Server	Beobachter	Antwortet auf *BagRequest* (**423**) und sendet den Inhalt des Beutels.
425	PlayerHandsRequest	Beobachter	Server	Sendet eine Anfrage nach den Steinen, die alle Spieler auf der Hand haben.
426	PlayerHandsResponse	Server	Beobachter	Antwortet auf *PlayerHandsRequest* (**425**) und sendet die Steine aller Spieler.
498	GameDataRequest	Client	Server	Der Client fordert den gesamten Spielzustand an.
499	GameDataResponse	Server	Client	Der Server sendet den gesamten Spielzustand.



Das hier zusehende Klassendiagramm "InGame" zeigt alle Nachrichten im Detail. Zusätzlich zu den IDs und den Beschreibungen weiter oben, sieht man hier die einzelnen Klassen und ihre Attribute.



In diesem State-Chart wird das Verhalten in einer Spielsitzung beschrieben. Wir beschreiben hier nur Eigenschaften, die nicht eindeutig aus dem State-Chart erkennbar sind.

Wird ein Spiel vom Ausrichter gestartet, wird vom Ausrichter eine Reihenfolge der Spieler gewählt, nach dieser sind die Spieler pro Runde am Zug.

In jedem Spiel muss beim ersten *PlayTiles* (**414**) genau einen Stein die Koordinaten $(x, y) = (0, 0)$ haben. Ansonsten ist der Zug nicht valide und der Server sendet dem Client *MoveValid* (**415**) mit *validation=false*.

Die Zugzeit eines Spielers beginnt, nachdem vom Server *CurrentPlayer* (**409**) an alle Clients in diesem Spiel verschickt wurde. Der Spieler muss innerhalb seiner Zugzeit einen validen Zug mit *PlayTiles* (**414**) oder *TileSwapRequest* (**411**) an den Server verschicken.

Ob ein *TileSwapRequest* (**411**) valide ist, wird vom Server geprüft. Der Client erhält die Antwort mittels *TileSwapValid* (**412**). Ein *TileSwapRequest* ist nicht valide, wenn der Client mehr Steine anfordert, als er auf der Hand hat oder noch im Bag sind.

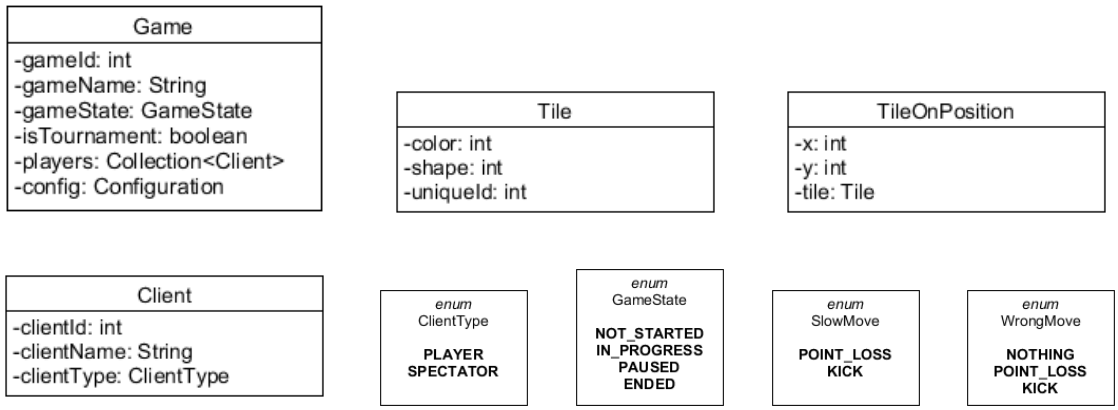
Alle Antworten auf *Request*-Anfragen basieren auf dem Stand der bereits per *CurrentPlayer* (**409**)

oder *Update* (**416**) an alle Clients verschickt wurde. Auch wenn der Server schon einen aktuelleren Stand hat.
Beispiel: Spieler A schickt validen Zug an Server und hat ein *MoveValid* erhalten. Nun fragt Beobachter B mit *GameDataRequest* die Spieldaten ab. Wird die *GameDataResponse* vor dem *Update* verschickt, ist der Zug von Spieler A noch nicht darin enthalten.

Fragt ein Beobachter mit *GameDataRequest* (**498**) die Spieldaten ab, ist *OwnTiles* eine leere Liste.

Wird ein Spieler aus einem Spiel gekickt oder disconnected, entfällt er aus der Wertung, seine Steine auf der Hand werden in den Bag zurückgegeben und die Steine auf dem Spielfeld bleiben liegen.

4.4 Erläuterung der Implementierung



Diese Klassen werden vom Interface-Komitee bereitgestellt. Sie werden in einigen Message-Klassen benutzt, um Informationen strukturiert zu übermitteln.

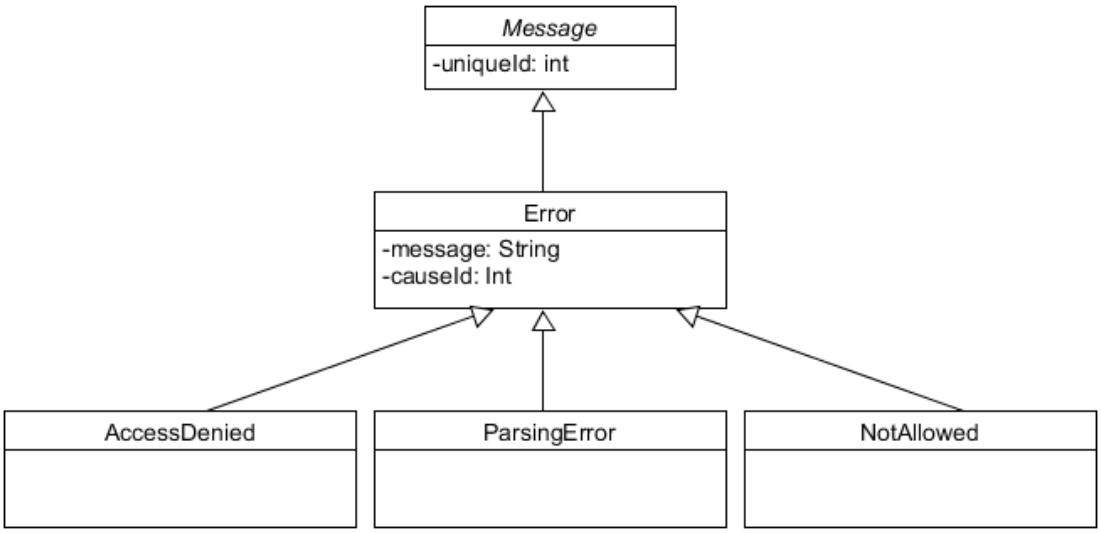
ID	Pseudoname	Attribute	Erklärung
100	ConnectRequest	String *clientName*	Name des Spielers
		ClientType *clientType*	Typ des Clients
101	ConnectAccepted	int *clientId*	interne ID des Spielers
300	GameListRequest		
301	GameListResponse	Collection<Game> *games*	Liste aller Spiele
302	GameJoinRequest	int *gameId*	ID des Spiels
303	GameJoinAccepted	Game *game*	ID des Spiels
304	SpectatorJoinRequest	int *gameId*	ID des Spiels
305	SpectatorJoinAccepted	Game *game*	ID des Spiels
306	MessageSend	String *message*	Die zu übertragene Chat-Nachricht
307	MessageSignal	String *message*	Die neue Chat-Nachricht
		Client *client*	Klasse des Clients, der die Nachricht verschickt hat

ID	Pseudoname	Attribute	Erklärung
400	StartGame	Configuration *config*	Spielkonfiguration
		Collection<Client> *clients*	Liste aller Spieler
401	EndGame		
402	AbortGame		
403	PauseGame		
404	ResumeGame		
405	LeavingRequest		
406	LeavingPlayer	Client *client*	Client, der das Spiel verlassen hat
407	Winner	Client *client*	Client, der gewonnen hat
		int *score*	Punkte des Gewinners
		Map<Client, Integer> *leaderboard*	Auflistung aller Spieler (als Key) und der jeweiligen Punktzahl (als Value)
408	StartTiles	Collection<Tile> *tiles*	Liste aller Steine, die der Spieler zu Beginn bekommt
409	CurrentPlayer	Client *client*	Client, der gerade am Zug ist
410	SendTiles	Collection<Tile> *tiles*	Liste der Steine, die ein Spieler aus dem Bag bekommt
411	TileSwapRequest	Collection<Tile> *tiles*	Liste der Steine, die der Spieler tauschen möchte
412	TileSwapValid	boolean *validation*	Dürfen die Steine getauscht werden?
413	TileSwapResponse	Collection<Tile> *tiles*	Liste der Steine, die der Spieler durch einen Tausch erhält
414	PlayTiles	Collection<TileOnPosition> *tiles*	Liste der Steine, die der Spieler legen will sowie deren Position auf dem Spielfeld
415	MoveValid	boolean *validation*	Dürfen die Steine so gelegt werden?
		String *message*	Begründung
416	Update	Collection<TileOnPosition> *updates*	Liste aller Steine, die im letzten Spielzug auf das Brett gelegt wurden sowie deren Position auf dem Brett. Außerdem wird die Anzahl der Steine im Beutel mitgesendet.
		int *numberTilesInBag*	Anzahl der Steine die noch im Bag verbleiben.
417	ScoreRequest		
418	ScoreResponse	Map<Client, Integer> *scores*	Auflistung aller Spieler (als Key) und des jeweiligen Punktestands (als Value)
419	TurnTimeLeftRequest		
420	TurnTimeLeftResponse	long *time*	Die restliche Zeit für den aktuellen Spielzug
421	TotalTimeRequest		
422	TotalTimeResponse	long *time*	Zeit, die bisher im gesamten Spiel verstrichen ist
423	BagRequest		
424	BagResponse	Collection<Tile> *bag*	Liste aller Steine im Beutel
425	PlayerHandsRequest		
426	PlayerHandsResponse	Map<Client, Collection<Tile> > *hands*	Auflistung aller Spieler (als Key) und jeweils einer Liste der Handsteine des Spielers (als Value)
498	GameDataRequest		
499	GameDataResponse	Collection<TileOnPosition> *board*	Liste aller auf dem Brett liegenden Steine sowie deren Position
		Client *currentClient*	Die ID des Spielers, der gerade am Zug ist
		Collection<Tile> *ownTiles*	Liste aller Steine, die der Spieler auf der Hand hat (leere Liste, wenn von Beobachter angefragt)
		GameState *gameState*	Der aktuelle Zustand des Spiels.

Klasse	Attribute / **Instanzen**	Erklärung
ClientType **enum**	**PLAYER**	Der Client ist ein aktiver Spieler.
	SPECTATOR	Der Client ist ein Beobachter.
WrongMove **enum**	**NOTHING**	Der Spieler ist erneut an der Reihe
	POINT_LOSS	Der Spieler erhält die konfigurierte Anzahl an Strafpunkten und der nächste Spieler ist an der Reihe
	KICK	Der Spieler wird vom Spiel ausgeschlossen und der nächste Spieler ist an der Reihe
SlowMove **enum**	**POINT_LOSS**	Der Spieler erhält die konfigurierte Anzahl an Strafpunkten und der nächste Spieler ist an der Reihe
	KICK	Der Spieler wird vom Spiel ausgeschlossen und der nächste Spieler ist an der Reihe
GameState **enum**	**NOT_STARTED**	Das Spiel ist noch nicht gestartet
	IN_PROGRESS	Das Spiel läuft
	PAUSED	Das Spiel ist pausiert
	ENDED	Das Spiel ist beendet
Configuration	int *colorShapeCount*	Anzahl der Farben bzw. der Symbole
	int *tileCount*	Häufigkeit eines Spielsteins
	int *maxHandTiles*	Anzahl der Steine, die man auf der Hand haben darf
	long *turnTime*	Zeit für jeden Zug
	long *timeVisualization*	Zeit für die Visualisierung des letzten Zugs
	WrongMove *wrongMove*	Strafe für fehlerhaften Zug
	int *wrongMovePenalty*	Strafpunkte für einen fehlerhaften Zug
	SlowMove *slowMove*	Strafe für Ablauf der Zugzeit
	int *slowMovePenalty*	Strafpunkte für Ablauf der Zugzeit
	int *maxPlayerNumber*	maximale Spieleranzahl für das Spiel
Client	int *clientId*	ID des Clients
	String *clientName*	Name des Client
	ClientType *clientType*	Typ des Clienten
Game	int *gameId*	ID des Spiels
	String *gameName*	Name des Spiels
	GameState *gameState*	Status des Spiels
	boolean *isTournament*	Gehört das Spiel zu einem Turnier?
	Collection<Client> *players*	Liste der Spieler des Spiels
	Configuration *config*	Spielkonfiguration
Tile	int *color*	Farbe des Spielsteins
	int *shape*	Symbol des Spielsteins
	int *uniqueId*	einzigartige ID zur eindeutigen Identifizierung jedes Steins
TileOnPosition	int *x*	x-Koordinate des Steins
	int *y*	y-Koordinate des Steins
	Tile *tile*	Der Spielstein zu den Koordinaten

5 Fehlerbehandlung

Bei der Kommunikation zwischen Client und Server können viele verschiedene Fehler auftreten. Ein Fehler enthält eine Nachricht, die Id der Nachricht die den Fehler hervorruft und ist genau einer der drei Typen: AccessDenied, ParsingError oder NotAllowed. Fehler werden ebenfalls als Message-Objekt vom Server an den jeweiligen Client gesendet.



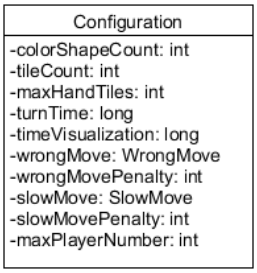
In der folgenden Tabelle wird kurz erklärt in welchen Situationen die jeweiligen Fehlertypen vorkommen.

ID	Pseudoname	Sender	Empfänger	Erklärung
900	AccessDenied	Server	Client	Der Client hat nicht die nötigen Rechte um diese Nachricht zu senden.
910	ParsingError	Server	Client	Die zuletzt gesendete Nachricht konnte nicht korrekt interpretiert werden.
920	NotAllowed	Server	Client	Die Nachricht ist im aktuellen Zustand nicht zulässsig.

Die folgende Tabelle stellt dar, zu welchen Nachrichten welche Fehler auftreten können und welche Nachrichtensender dies betrifft. Sollten mehrere Fehler auftreten wird immer der Fehler mit der höchsten Priorität zurückgesendet. Die Prioritäten sind: ParsingError > AccessDenied > NotAllowed.
Diese Tabelle ist nicht vollständig und nur ist nur als Beispiel anzusehen.

ID	Rolle	Beschreibung	Fehlermeldung
alle	-	Nachricht konnte nicht korrekt interpretiert werden.	ParsingError
100	Client	Client ist schon verbunden.	NotAllowed
300	Client	Client ist schon in einem Spiel.	NotAllowed
302	Spieler	Spieler ist schon in einem Spiel.	NotAllowed
302	Spieler	Das Spiel hat schon angefangen.	NotAllowed
302	Beobachter	Keine Berechtigung für Beobachter.	AccessDenied
304	Client	Client beobachtet oder spielt schon.	NotAllowed
306	Client	Client nicht im Spiel.	NotAllowed
405	Client	Client nicht im Spiel.	NotAllowed
411	Spieler	Spieler ist nicht am Zug.	NotAllowed
411	Beobachter	Keine Berechtigung als Beobachter.	AccessDenied
414	Spieler	Spieler ist nicht am Zug.	NotAllowed
414	Spieler	Der Stein ist nicht bekannt.	ParsingError
414	Beobachter	Keine Berechtigung als Beobachter.	AccessDenied
417	Client	Client nicht im Spiel.	NotAllowed
419	Client	Client nicht im Spiel.	NotAllowed
421	Client	Client nicht im Spiel.	NotAllowed
423	Client	Client nicht im Spiel.	NotAllowed
423	Spieler	Keine Berechtigung als Spieler.	AccessDenied
425	Beobachter	Beobachter nicht im Spiel.	NotAllowed
425	Spieler	Keine Berechtigung als Spieler.	AccessDenied
498	Client	Client nicht im Spiel.	NotAllowed

6 Konfiguration



Die Konfiguration wird gespeichert, indem die Configuration Klasse serialisiert in einer .json Datei gespeichert wird.

In der Konfiguration besteht die Option slowMove als KICK oder POINT_LOSS festzulegen. Diese Option definiert, wie sich der Server bei abgelaufender Zugzeit verhält.
Die Option wrongMove erlaubt: KICK, POINT_LOSS, NOTHING. Diese Option tritt bei regelwidrigen Zügen in Kraft.
Die Anzahl der abgezogenen Punkte lässt sich unter der Option penalty konfigurieren.
Die Optionen KICK, POINT_LOSS und NOTHING bewirken folgendes:

- KICK: Der Client wird wieder in die Lobby überwiesen.
- POINT_LOSS: Die in penalty definierte Anzahl an Punkten wird vom Punktekonto des Spielers subtrahiert.
- NOTHING: Der Spieler bekommt keine Strafe und darf in seiner verbleibenden Zugzeit einen neuen Zug machen.

Mit der Option maxPlayerNumber wird bestimmt, wie viele Spieler in diesem Spiel sein dürfen. Ist die maximale Spieleranzahl mit 0 in der Konfiguration angegeben, bedeutet dies das keine Spielerobergrenze definiert ist. Die minimale Spieleranzahl muss immer mindestens 2 sein. ColorShapeCount ist 0 basiert.

7 Changelog

Version 1.0.1	Rechtschreibfehler behoben, Erläuterungen im Dokument hinzugefügt, Implementierung optimiert (ohne Änderung der API).
Version 1.1.0	List<> und HashMap<> ersetzt mit Interface Collection<>, bzw. Map<>, Alle Klassen des haben eine überschriebene equals() Methode bekommen, DisconnectSignal entfernt (Message ohne nutzen), GameDataRequest: boolean paused zu GameState gameState, Update: int numberTilesInBag hinzugefügt, Alle "non-MessageKlassen in eigenes Package bewegt, nach AbortGame folgt Winner Message, Error-Messages: int causeId hinzugefügt (Id der Error-erzeugenden Nachricht)