Dokumentation Netzwerkprotokoll

Version	Projektname	Autor	Status	Datum	Kommentar
5	Rat um Rad	Emanuele Tirendi _____ _____ _____ _____ _____ ___	Zur Veröffentlichung freigegeben	21.05.2023	Von Emanuele Tirendi nachbearbeitete Version.

An wen richtet sich dieses Dokument?

In diesem Dokument wird das Netzwerkprotokoll beschrieben, welches für die Kommunikation zwischen Client und Server in der Software Rat-um-Rad verwendet wird.

Dieses Dokument richtet sich an die Mitglieder der Gruppe ProgRadler, welche die Software Rat-um-Rad implementieren, um eine wohldefinierte Schnittstelle für das Server-Implementierungsteam und das Client-Implementierungsteam zu definieren. Obwohl dieses Dokument eine umfangreiche Präzision und Formalität aufweist, hat es keinen verbindlichen Charakter, da das Protokoll dafür noch zu unpräzise definiert ist.

Struktur der versendeten Nachrichten

Auf Server- und Clientseite werden die Nachrichten in sogenannten Packets versendet. Diese werden dann von sogenannten Codern beim Senden zu Strings kodiert und beim Empfangen wieder dekodiert. Ein solches Packet besteht aus folgenden drei Inhalten.

- Der erste Inhalt ist der sogenannte **Command**. Dieser bestimmt die eigentliche Nachricht, die übermittelt werden soll. Wir unterscheiden Server-Commands (Commands, die vom Server zum Client geschickt werden) und Client-Commands (Commands, die vom Client zum Server geschickt werden).
- Der zweite Inhalt ist der sogenannte **Content-Type**. Oft müssen im Rahmen dieser Kommunikation Daten übermittelt werden. Da diese in unterschiedlichen Datentypen repräsentiert werden können, müssen sie auch auf unterschiedliche Weise zu Strings kodiert und dekodiert werden. Aus dem Content-Type liest der sogenannte PacketCoder, welchen anderen Coder (z.B. messageCoder, highscoreCoder etc.) er aufrufen soll, um das Packet zu kodieren und dekodieren.
- Der dritte Inhalt ist dann der zu überliefernde Content.

Commands

Client-Commands: Commands, die vom Client zum Server geschickt werden.

PONG	Implementierung der Ping-Pong-Verbindungsüberprüfung.
REGISTER_USER	Anfrage, sich im Server zu registrieren und den Benutzernamen zu wählen.
SET_USERNAME	Anfrage, den Benutzernamen zu ändern.
USER_SOCKET_DISCONNECTED	Dieser Command wird nicht von einem Client,

	sondern vom Server an sich selbst versendet, wenn der Socket, die ein bestimmter Client darstellen soll, eine SocketException wirft. Entsprechend ist er kein Client-Command im eigentlichen Sinne, wird aber wie alle Client-Commands im Server behandelt.
CREATE_GAME	Anfrage, ein neues Spiel zu erstellen.
REQUEST_GAMES	Anfrage, eine Liste von entweder allen offenen, allen begonnenen oder allen geschlossenen Spielen zu erhalten.
REQUEST_ALL_CONNECTED_PLAYERS	Anfrage, eine Liste aller mit dem Server verbundenen Spielern zu senden.
WANT_JOIN_GAME	Anfrage, einem bestimmten Spiel beizutreten.
SHORT_DESTINATION_CARDS_SELECTED	Informieren darüber, welche Destination-Cards im Spiel-Status PREPARATION gewählt worden sind.
REQUEST_SHORT_DESTINATION_CARDS	Anfrage, Destination-Cards aufzunehmen.
BUILD_ROAD	Anfrage, eine Radstrecke zu bauen.
REQUEST_WHEEL_CARDS	Anfrage, Radkarten aufzunehmen.
REQUEST_HIGHSCORES	Anfrage, Highscores zu erhalten.
SEND_BROADCAST_CHAT	Broadcast-Nachricht senden
SEND_GAME_INTERNAL_CHAT	Nachricht im Game-Chat senden
SEND_WHISPER_CHAT	Whisper-Nachricht senden

Server-Commands: Commands, die vom Server zum Client geschickt werden.

PING	Implementierung der Verbindungsüberprüfung.
USERNAME_CONFIRMED	Der Server bestätigt dem Spieler den Benutzernamen oder teilt mit, dass der Benutzername bereits vergeben war und deswegen abgeändert wurde zu einem neuen, vom Server selbst vorgeschlagenen Benutzernamen.
NEW_USER	Wird an andere Spieler geschickt, um mitzuteilen, dass sich ein neuer Spieler mit dem Server verbunden hat.
CHANGED_USERNAME	Der Server teilt allen Spielern mit, wenn ein Benutzername geändert worden ist.
USER_DISCONNECTED	Mitteilung an allen anderen Spielern, dass die Verbindung zu einem Spieler getrennt worden ist.
GAME_CREATED	Bestätigung, dass ein Spiel erstellt worden ist.
SEND_ALL_CONNECTED_PLAYERS	Sende alle verbundenen Spieler.
INVALID_ACTION_WARNING	Wird gebraucht, wenn ein Spieler eine invalide Aktion durchzuführen versucht, welche das grundsätzliche

	Vorgehen nicht stört.
INVALID_ACTION_FATAL	Wird gebraucht, wenn der Spieler eine invalide Aktion durchführt und dadurch das Funktionieren des Programms negativ beeinflusst wird.
SEND_WAITING_GAMES SEND_STARTED_GAMES SEND_FINISHED_GAMES	Senden entweder aller offenen, aller begonnenen oder aller geschlossenen Spiele.
GAME_JOINED	Der Spieler, der dem Spiel beitreten will, wird darüber benachrichtigt, dass dies funktioniert hat.
NEW_PLAYER	Alle anderen Spieler in einem Spiel werden benachrichtigt, dass ein bestimmter Spieler diesem Spiel beigetreten ist.
GAME_STARTED_SELECT_DESTINATION_CARDS	Nachdem genug Spieler in einem Spiel sind, werden Destination-Cards ausgeteilt und die Spieler müssen sie auswählen.
DESTINATION_CARDS_SELECTED	Bestätigung, dass Destination-Cards im Spielstatus PREPARATION ausgewählt worden sind.
GAME_UPDATED	Wird an alle Spieler in einem Spiel geschickt, als Information, dass jemand Destination-Cards im Spielstatus STARTED aufgenommen hat.
REQUEST_SHORT_DESTINATION_CARDS_RESULT	Bestätigung, dass die Destination-Cards im Spielstatus Started gewählt worden sind.
ROAD_BUILT	Wird an alle Spieler im Spiel geschickt, als Information, dass jemand eine Radstrecke gebaut hat.
WHEEL_CARDS_CHOSEN	Benachrichtigung, dass eine Radkarte aufgenommen worden ist.
SEND_HIGHSCORES	Senden der Highscores.
BORADCAST_CHAT_SENT	Wird an alle Spieler mit der Broadcast-Nachricht geschickt, die ein Spieler gesendet hat.
GAME_INTERNAL_CHAT_SENT	Wird an alle Spieler in einem Spiel mit der Nachricht geschickt, die ein Spieler gesendet hat.
WHISPER_CHAT_SENT	Wird an den Spieler geschickt, dem ein anderer Spieler eine Whisper-Nachricht schicken will.
GAME_ENDED	Wird an Spieler geschickt, wenn das Spiel zu Ende ist.
GAME_ENDED_BY_PLAYER_DISCONNECTION	Wird an Spieler geschickt, wenn das Spiel dadurch zu Ende geht, wenn einer der Spieler seine Verbindung verliert.

Content-Types:

Content-Type	**Klasse, deren Instanz mit diesem Content-Type geschickt wird**
CHAT_MESSAGE	ChatMessage

STRING	String
INTEGER	Integer oder int
USERNAME_CHANGE	UsernameChange, eine Klasse, die einen alten und einen neuen Benutzernamen als Felder besitzt.
GAME	ClientGame, eine Game-Repräsentation, welche für den Client sichtbar sein darf, also keine Informationen wie verdeckte Karten oder Karten eines bestimmten Players enthält.
STRING_LIST	List<String>
GAME_INFO_LIST	List<GameBase>, Liste von Gamebase, eine Klasse, von der Game oder ClientGame erben.
GAME_INFO_LIST_STARTED	List<GameBase> mit Status STARTED und PREPARATION
GAME_INFO_LIST_WAITING	List<GameBase> mit Status WAITING_FOR_PLAYERS
GAME_INFO_LIST_FINISHED	List<GameBase> mit Status FINISHED
GAME_STATUS	GameStatus
NONE	null
WHEEL_CARD	WheelCard
HIGHSCORE_LIST	List<Highscore>

Commands Implementierungen & Beispiele

In den Beispielen werden wir die Packets folgendermaßen darstellen: **Command, Content, ContentType**. Wichtig zu erwähnen ist hier, dass diese Beispiele nur dem Verständnis des Protokolls dienen und für interne Entwickler dient, welche über mehr Wissen über das Protokoll verfügen.

Ping/Pong:
Server → Client1: PING, null, NONE
Client1 → Server: PONG, null, NONE
Server → Client1: PING, null, NONE

Fall, dass Client1 kein PONG zurückschickt:
Server → Client2: USER_DISCONNECTED, username, STRING

Fall, dass Socket-Connection nicht mehr funktioniert: Nicht Teil des Protokolls, sondern intern im Server
ClientInputListener → CommandHandler: USER_SOCKET_DISCONNECTED

Broadcast-Chat:
Client1 → Server: SEND_BROADCAST_CHAT, “Hallo”, STRING
Server → Client2: BROADCAST_SENT, “Hallo”, STRING

Game-Chat:
Client1 → Server: SEND_GAME_INTERNAL_CHAT, “Hallo”, STRING
Server → Client2: GAME_INTERNAL_CHAT_SENT, “Hallo”, STRING

Whisper-Chat:
Client1 → Server: SEND_WHISPER_CHAT, new ChatMessage(“Player2”, “Hallo”), CHAT_MESSAGE
Server → Client2: WHISPER_CHAT_SENT, new ChatMessage(“Player2”, “Hallo”), CHAT_MESSAGE

Neuer Username und Nutzerregistrierung:
Client1 → Server: REGISTER_USER, “newUsername”, STRING
Fall1:
Server → Client1: INVALID_ACTION_FATAL, “Username invalid, please try it again.”, STRING
Fall2:
Server → Client1: USERNAME_CONFIRMED, new Username(“oldUsername”, “newUsername”), USERNAME_CHANGE
Server → Client2: NEW_USER, “newUsername”, STRING

Username ändern:
Client1 → Server: SET_USERNAME, “newUsername”, STRING
Fall1:
Server → Client1: INVALID_ACTION_WARNING, “Username invalid, please try it again.”, STRING
Fall2:
Server → Client1: USERNAME_CONFIRMED, new UsernameChange(“oldUsername”, “newUsername”), USERNAME_CHANGE
Server → Client2: CHANGED_USERNAME, new UsernameChange(“oldUsername”, “newUsername”), USERNAME_CHANGE

Neues Game kreieren:
Client1 → Server: CREATE_GAME, 3, INTEGER (hier heisst 3, dass dieses Spiel für drei Spieler ist)
Server → Client1: GAME_CREATED, clientGame, GAME
Server → Client1: SEND_WAITING_GAMES, gameRepository.getWaitingGames(), GAME_INFO_LIST
Server → Client2: SEND_WAITING_GAMES, gameRepository.getWaitingGames(), GAME_INFO_LIST

Game-Liste anfragen:
Fall 1:
Client → Server: REQUEST_GAMES, WAITING_FOR_PLAYERS, GAME_STATUS
Server → Client: SEND_GAMES, gameRepository.getWaitingGames(), GAME_INFO_LIST_WAITING
Fall 2:
Client → Server: REQUEST_GAMES, STARTED, GAME_STATUS
Server → Client: SEND_GAMES, gameRepository.geStartedGames(), GAME_INFO_LIST_STARTED
Fall 3:
Client → Server: REQUEST_GAMES, FINISHED, GAME_STATUS
Server → Client: SEND_GAMES, gameRepository.getFinishedGames(), GAME_INFO_LIST_FINISHED

Player-Liste anfragen:
Client → Server: REQUEST_ALL_CONNECTED_PLAYERS, null, NONE
Server → Client: SEND_ALL_CONNECTED_PLAYERS, listOfUsernames, STRING_LIST

Spiel beitreten:
Client1 → Server: WANT_TO_JOIN_GAME, “gameId”, STRING
Server → Client1: GAME_JOINED, clientGame, GAME
Server → Client2: NEW_PLAYER, clientGame, GAME
Fall, dass genug Spieler sind in einem Game, sodass es anfangen kann: → siehe **Destination-Cards ausgewählt im Spielstatus PREPARATION**:

Destination-Cards ausgewählt im Spielstatus PREPARATION:
Server → Client1: GAME_STARTED_SELECT_DESTINATION_CARDS, clientGame, GAME
Client1 → Server: SHORT_DESTINATION_CARDS_SELECTED, listOfIDs, STRING_LIST
Server → Client1: DESTINATION_CARDS_SELECTED
Server → Client2: GAME_UPDATED, clientGame, GAME

Destination-Cards auswählen im Spielstatus STARTED
Client1 → Server: REQUEST_SHORT_DESTINATION_CARDS, null, NONE

Server → Client: REQUEST_SHORT_DESTINATION_CARDS_RESULT, clientGame, GAME

Road bauen:
Client1 → Server: BUILD_ROAD, roadId, STRING
Server → Client1: ROAD_BUILT, clientGame, GAME
Server → Client2: ROAD_BUILT, clientGame, GAME
Fall, dass jemand genug wenig Radkarten hat, sodass das Spiel enden kann: → Siehe **Spiel endet:**

Radkarte aufnehmen:
Client1 → Server: REQUEST_WHEEL_CARDS, null, NONE
Server → Client1: WHEEL_CARDS_CHOSEN, clientGame, GAME
Server → Client2: WHEEL_CARDS_CHOSEN, clientGame, GAME

Spiel endet:
Server → Client: GAME_ENDED, clientGame, GAME

Verbindung von Spieler wird abgebrochen:
Server → Client: GAME_ENDED_BY_PLAYER_DISCONNECTION, clientGame, GAME

Highscore anfragen:
Client → Server: REQUEST_HIGHSCORES, null, NONE
Server → Client: SEND_HIGHSCORES, highscores, HIGHSCORE_LIST