Universität Basel
Programmierprojekt
FS21

Architecture

jDOGS

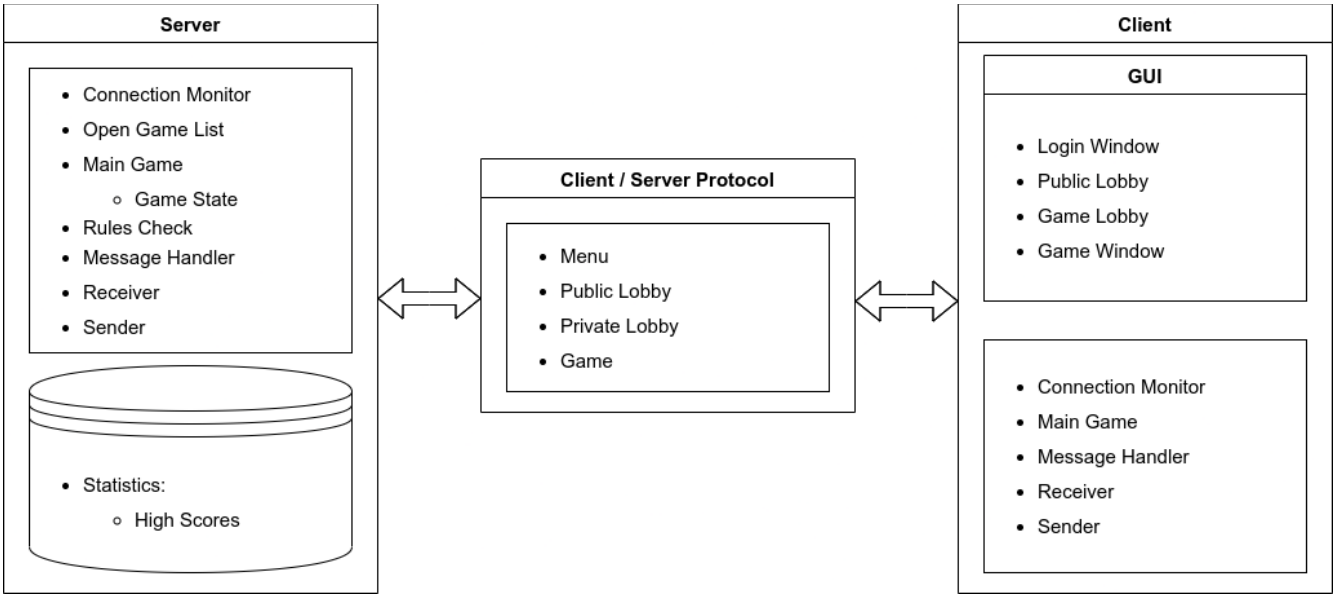
Gruppe 13

Ruben Hutter, Gregor Bachmann, Johanna Meyer

Einführung

jDogs ist eine digitale Version des Brettspiels Brändi Dog. Es handelt sich um ein rundenbasiertes Spiel mit einer Client/Server-Architektur. Der Server und die verschiedenen Clients kommunizieren dank eines selbst erstellten textbasierten Netzwerk-Protokolls.

Übersicht



Wie man aus diesem Diagramm erkennen kann, besteht das Programm hauptsächlich aus zwei Komponenten: Server und Client. Diese beiden Komponenten sind in der Lage, über das Protokollsystem zu kommunizieren. Für jede neue Socket-Verbindung, die unser Server erhält, wird ein neues ServerConnection-Objekt in einem eigenen Thread erstellt. Dieses ist das Zentrum der Verbindung und enthält ein Ping - Pong System zur Überwachung der Verbindung, 3 Queues zum Senden von Nachrichten an die verschiedenen Interessenten, eine zum Empfangen von Nachrichten und einen MessageHandler, der die empfangenen Nachrichten an die richtige Stelle verteilt. Sobald ein Client eine Verbindung mit dem Server herstellt, wird auf dem Gerät des Benutzers eine Client-Instanz erzeugt, die dasselbe Verbindungsüberwachungssystem, eine Queue zum Senden von Nachrichten, eine zum Empfangen von Nachrichten sowie einen MessageHandler enthält.

Einmal eingeloggt, ist es möglich, Lobbys zu erstellen und/oder zu betreten und dann zu spielen. Diese initialisierten Lobbys werden in der "OpenGameList" gespeichert und den Benutzern angezeigt, damit diese sich zum Spielen anmelden können. Sobald eine Lobby voll ist, kann der Host das Spiel starten. Zu diesem Zeitpunkt wird ein neues MainGame für dieses Spiel initialisiert, das immer auf dem Server gespeichert wird und die Teilnehmer, den Modus und den aktuellen Spielstatus im GameState enthält.

Alle im Spiel durchgeführten Aktionen, d. h. jeder Zug einer Murmel, jede gespielte Karte, werden vom Server im RulesCheck überprüft. Auf diese Weise ist es für einen Benutzer unmöglich, zu

schummeln, da die Züge, auch wenn er seinen eigenen Client manipuliert, immer noch zentral vom Server gesteuert werden.

Wenn ein Spiel endet, wird das Ergebnis in der HighScoreList, .csv-Datei, gespeichert, in der die Namen der am Spiel beteiligten Benutzer, die Anzahl der gespielten Spiele und die Anzahl der gewonnenen Spiele festgehalten werden.

Monitoring

Für die Verbindungsüberwachung haben wir einen ScheduledExecutorService verwendet.

Dies sendet eine periodische Nachricht, die erst nach der angegebenen Anfangsverzögerung und dann mit der angegebenen Periode aktiviert wird; d. h. die Ausführungen beginnen nach initialDelay, dann initialDelay + Periode, dann initialDelay + 2 * Periode usw.

Die Abfolge der Task-Ausführungen wird so lange fortgesetzt, bis in einem der beiden Monitore, server- oder clientseitig, die Bedingung der maximalen Differenz von 10s zwischen einer Nachricht und der anderen nicht mehr eingehalten wird; in diesem Fall wird die Verbindung dieses Clients unterbrochen.

Blocking Queue

Auf der Serverseite gibt es 4 verschiedene Möglichkeiten, Nachrichten auszutauschen: an alle senden, an alle Teilnehmer der gleichen Lobby senden, an einen bestimmten Client senden und Nachrichten empfangen.

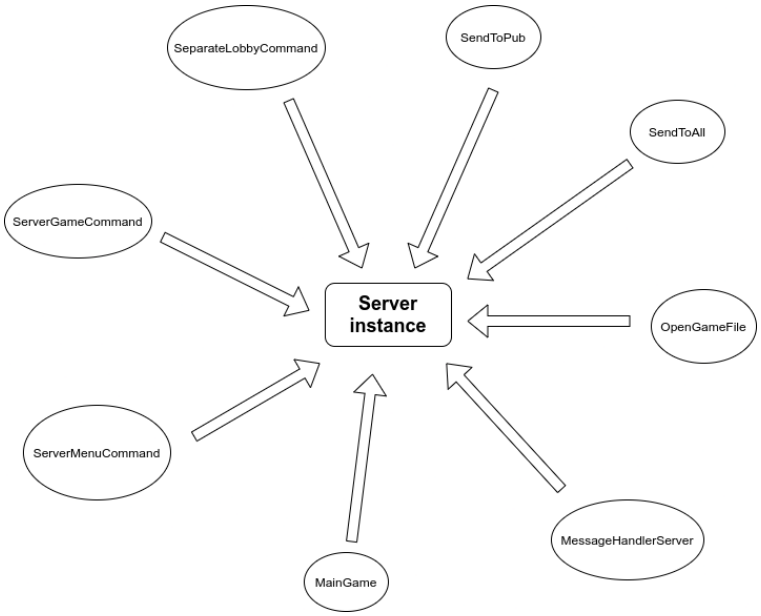
Um zu vermeiden, dass an 4 Stellen Ressourcen verschwendet werden, indem versucht wird, Elemente zu entfernen, wenn sich keine in der Queue befinden, oder im Gegenteil, diese überfüllt werden und nie eine Pause haben, haben wir uns für den Einsatz von BlockingQueue entschieden.

Die BlockingQueue ist eine Queue, die es erlaubt, ähnlich wie ein Thread, zu "schlafen", wenn sie nicht gebraucht wird, und bei Bedarf "aufgeweckt" zu werden. Also eine sehr effektive Lösung, um eine Überlastung des Servers zu vermeiden und nicht 4 weitere Threads nur zur Überwachung dieser Queues erstellen zu müssen.

Singleton

Wir haben uns entschieden, den Server als Singleton zu implementieren, damit wir sicher sein können, immer die gleiche Instanz im Spiel zu verwenden und den Server als Parameter übergeben zu können, obwohl er kein Objekt ist.

Alle Komponenten, die Informationen vom Server benötigen, beziehen sich immer auf die gleiche zentrale Instanz.



Message Handler Server

Die Verbindung zum Server kann sich während ihrer Ausführung in drei Zuständen befinden. In der ersten Phase befindet sie sich nach dem Einloggen im Ausgangszustand "publicLobby". Wenn der Benutzer eine Lobby betritt, ändert er seinen Status auf "openGame" und wenn er spielt, auf "playing".

Diese Zustände werden vom MessageHandlerServer verwendet, um zu entscheiden, wie Informationen verarbeitet werden sollen. Wenn sich die Verbindung beispielsweise im Ausgangszustand befindet, wird eine hypothetische Chat-Nachricht an alle anderen mit dem Server verbundenen Benutzer gesendet, aber wenn der Zustand "playing" ist, wird die gleiche Nachricht nur an die Teilnehmer dieses Spiels gesendet.

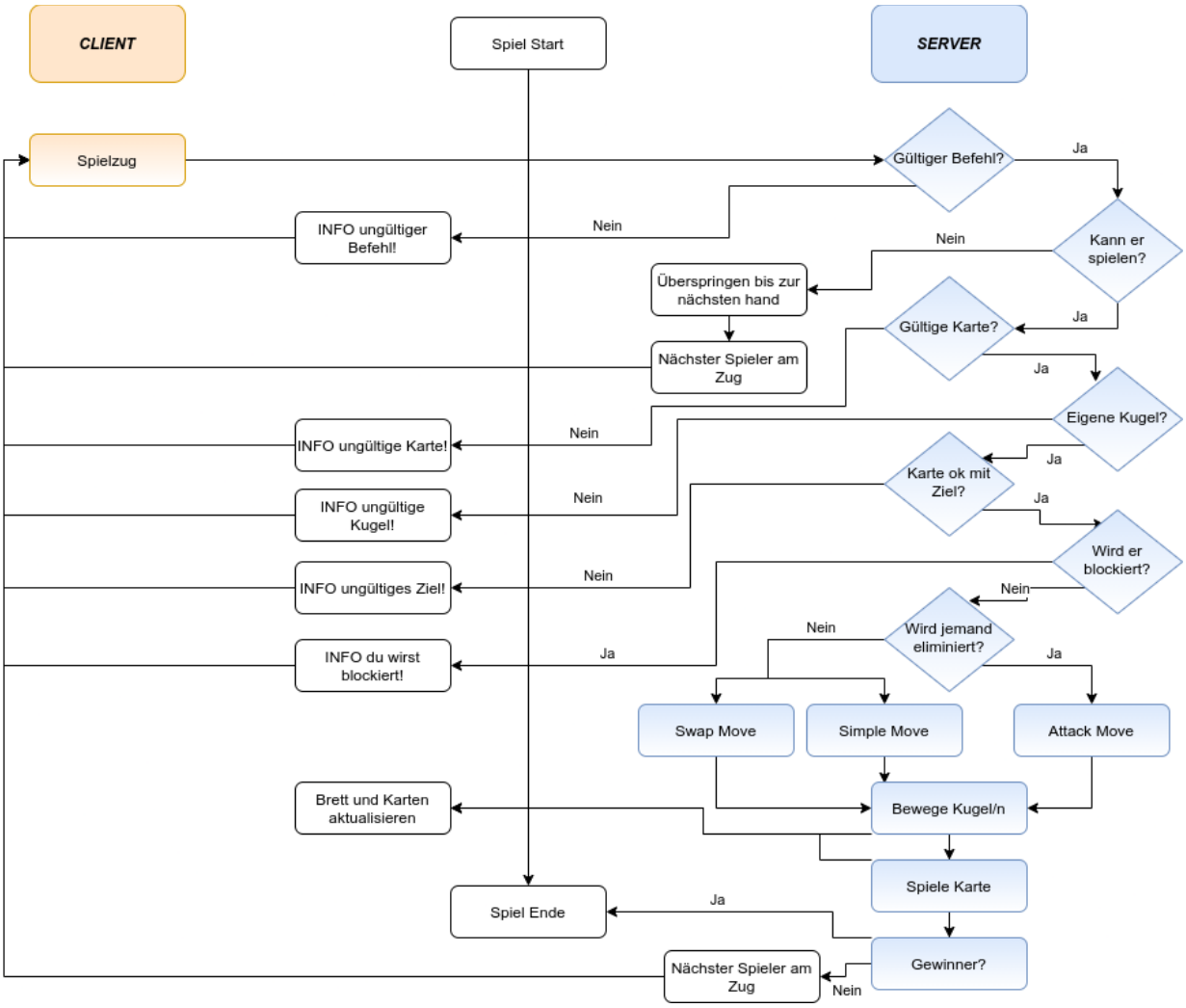
Game State & Game Logic

Unser Spiel benötigt ein Spielbrett, 4 Murmeln für jeden Spieler und ein Kartendeck.

Wenn ein MainGame erstellt wird, werden für jeden Spieler Murmeln generiert und die ersten 6 Karten werden zufällig verteilt.

Die Position jeder Murmel sowie die Hand jedes Spielers werden auf dem Server im GameState gespeichert und verändert und dann an die jeweiligen Clients übermittelt.

Die Spiellogik wird dann wie folgt geregelt:



Es ist gut zu sehen, dass, wie bereits in der Übersicht erwähnt, jeder Zug vom Server geprüft und, falls gültig, an die Clients kommuniziert wird, andernfalls wird der Benutzer über den Fehler informiert und kann es erneut versuchen.

High Score

Um die Spannung zu erhöhen und das Spiel kompetitiver zu machen, halten wir fest, was die Spieler in den Spielen erreicht haben.

Wenn der Server zum ersten Mal gestartet wird, prüft er, ob eine HighScoreList vorhanden ist, und erzeugt sie, falls sie noch nicht existiert, in einem bestimmten Ordner.

Nach jedem Spiel, das mit einer Siegermannschaft endete, wird die Liste aktualisiert geschrieben, wobei neue Spieler hinzugefügt und die Daten für vorhandene Spieler geändert werden. Von der Hauptlobby aus kann dann die Tabelle mit den Daten angezeigt werden.

Logging

Um den Überblick zu behalten, ist es notwendig, ein Protokoll zu führen, in dem man die erstellten Partien, die beendeten Partien, die ausgeführten Züge und nicht zuletzt die Fehler notiert. Eine Möglichkeit, die Fehler zu verfolgen, ist unerlässlich, um eventuelle Störungen korrigieren zu können.

Für jeden Tag, an dem das Programm verwendet wird, wird eine .log-Datei in einen speziellen Ordner geschrieben. Jede Zeile dieser Datei stellt einen ausgeführten Befehl dar und wird durch das Datum und die genaue Uhrzeit des Ereignisses eingeleitet.