Kurzbericht   
Snake-Core, SnakeFX, SnakeServer

Ostfalia Fachhochschule für angewandte Wissenschaften

Benjamin Wulfert

Leonard Reidel

Semester: Wintersemester 2020

20. Januar 2021

Inhaltsverzeichnis

[Projektdokumentation 3](#_Toc61112001)

[Umsetzung der Anforderungen 4](#_Toc61112002)

[Erweiterungen und Extras 5](#_Toc61112003)

[Bedienungsanleitung und Spielregeln 6](#_Toc61112004)

[Projektplan 7](#_Toc61112005)

[STOMP-Server 8](#_Toc61112006)

[StompClient 8](#_Toc61112007)

[StompSessionHandler 8](#_Toc61112008)

[ChatController 8](#_Toc61112009)

[WebSocketConfig 8](#_Toc61112010)

[STOMP / Websockets – Use Cases 9](#_Toc61112011)

[Neues Spiel wird erstellt 9](#_Toc61112012)

[Spieler tritt Spiel bei 9](#_Toc61112013)

[Admin startet Spiel 10](#_Toc61112014)

# Kurzbericht

Dieses Dokument stellt den Kurzbericht für das Projekt des Modul Patterns und Frameworks dar. Im Folgenden werden die verschiedenen Aspekte des Projekts beschrieben. Des Weiteren wird in dieser Kurzbericht dargelegt wie die Anforderungen des Projekts umgesetzt wurden sowie die Erweiterungen vorgestellt, Anleitung zur Bedienung gegeben und ein aktualisierter Projektplan dargestellt.

Der Quelltext des Projekts sowie alle während der Umsetzung angefallenen Dokumente (Text-Dokumente, UML-Diagramme, Gantt-Diagramme, etc. ) kann den folgenden GitHub-Repositories entnommen werden:

<https://github.com/benjaminfoo/SnakeFX> - Benjamin Wulfert

<https://github.com/Bummelnderboris/Patterns_and_Frameworks> - Leonard Reidel

## Umsetzung der Anforderungen

### Accountverwaltung

Registrierung, Login, Logout

### Spielhistorie

Persistenz einer Spielrunde, wann wird persistiert? Wie kann ein Spieler diese betrachten?

### Spiel-Implementierung

Pro Spieler einen Wurm, Spieler-Größe, Spieler-Bewegung, Teil eines Spielers abbeißen, abspielen von Sounds, Food, ~~PowerUps (Länge verändern, beweglichkeit modifizieren, schutzschild~~), berührt sich ein wurm selbst ist das Spiel vorbei (sound ertönt)

Spiel-Ende: Spieler zerstört sich selbst (Spieler verliert), ein Wurm die X-fache Länge eines anderen Wurms besitzt.

Nach Ende des Spiels kann es in einem höheren Level erneut begonnen werden. Wie viele Elemente es gibt, wie die Berührungen stattfinden, wie eine erreichte Punktzahl ermittelt wird, wodurch sich der Schwierigkeitsgrad erhöht etc. entscheiden Sie selbst <- Das machen wir anders.

Im Folgenden wird die Logik der Kernelemente für die Spielimplementierung prägnant dargelegt:

Das Spiel baut auf einer JavaFX Stage auf, welche eine Timeline besitzt um Schlangenbewegungen und Foodspawn zu koordinieren, folgend wird das Wort *Tick* für einen Spiel Moment verwendet, in einem Spielmoment rückt eine Schlange ein Feld nach vorne.

* Spielfeld: Das Spielfeld ist ein x-y Koordinatensystem
* Schlangenbeschaffenheit: Eine Liste aus x,y Koordinaten mit Kopf = Listenpoisition 0
* Schlangenbewegung: Für jeden Tick wird ein Richtungsvektor aus der Eingabe ermittelt und auf den Kopf der Schlange angewandt, um die nächste Position zu bestimmen, dazu wird jeder Listeneintrag durch den Vorgänger ersetzt und der letzte Eintrag der Liste gelöscht. (Bei einer nicht Eingabe wird der vorherige Richtungsvektor gemerkt)
* Schlangenspawn: Es wird eine Initiallänge i und eine Startkkordinate für Schlangen gewählt. Die Listenlänge der Schlange ist gleich i und wegen der Schlangenbewegungslogik genügt dies zu Initialisierung.
* Schlangenteleport: Wenn eine Schlange die x/y Koordinate Spielfeldlänge/Spielfeldtiefe + 1 erreicht, wird der Kopf der Schlange auf x/y Koordinate = 0 gesetzt. Wegen der Schlangenbewegungslogik genügt dies um es zu realisieren. (Selbes umgekehrt)
* Power UP- Schlange beißt eine andere ab: Wenn der Kopf einer Schlange den Körper einer anderen trifft, wird der Index des Eintrags der anderen Schlange ermittelt, dann die Gesamte Listenlänge Minus dem Index gerechnet und die daraus entstehende Zahl als Elemente der beißenden Schlange angehängt, indem einfach die Koordinate der beißenden Schlange x mal in die Liste eingefügt wird (ähnlich der Initialisierungsmethodik). Dass dies durch ein Power-Up realisiert werden kann, haben Schlangen das Attribut isPredator, was default false ist und durch ein solches PowerUp auf True gesetzt werden kann. Ist es auf True wird diese Logik aktiviert und wird mit dem Essen eines anderen Power Ups wieder auf false gesetzt.
* Für das Essen ist eine Liste angelegt um mehrere Essen gleichzeitig haben zu können, dabei werden Koordinaten über ein Zufallsgenerator von 0 bis Spielfeldlänge/breite Erzeugt. Auf den in der Essensliste liegenden Koordinaten wird Essen gezeichnet sowie eine Essensaktion ausgelöst, sofern ein Kopf einer Schlange auf diese Koordinate kommt.

### Server

Für die Server-Komponente gilt:

- Zu verwendende Programmiersprache: Java.

- Registrierte Anwender und deren Spielhistorie sind zentral in einer relationalen Datenbank gespeichert.

- Es wird ein Framework für das Object-Rational Mapping eingesetzt.

- Es wird eine API für den Datenaustausch im JSON- oder XML-Format mit dem Client angeboten.

- JSON Web Tokens werden zur Authentifizierung von Anwendern unterstützt. (machen wir nicht weil Zeit und nur zwei Leute)

### Client

Für die Client-Komponente gilt:

- Hier ist die grafische UI umgesetzt.

- Es werden alle funktionalen Anforderungen abgebildet.

- Es muss mindestens ein Client mit JavaFX erstellt werden.

~~- Ein zweiter Client soll mit Angular erstellt werden, wenn die Gruppengröße 4 Personen ist. Beide Clients teilen sich die serverseitige API.~~ – wir sind nur zwei Leute

### Entwurfsmuster

Composite, Factory und Observer

### Kommunikation

Synchron / Asynchron – http / REST vs WebSocket / STOMP-Protocoll

### Sounds

Erstellt mit OpenSource Software sfxr.

AppSnakeFX-Klasse besitzt referenz auf BaseApplication – diese wiederrum besitzt eine Referenz auf den Singleton SoundManager – der Soundmanager kann genutzt werden um verschiedene Sounds im JavaFX-Client wiederzugeben.

## Erweiterungen und Extras

## Bedienungsanleitung und Spielregeln

## Projektplan

# STOMP-Server

Der folgende Abschnitt beschreibt die Integration des STOMP-Services sowie dessen Bereitstellung. Des Weiteren wird erläutert wie mithilfe eines STOMP-fähigen Clients die Kommunikation zwischen dem Front- und dem Backend hergestellt und verwendet werden kann.

ws://localhost:8080/chat

## StompClient

Der StompClient …

*CONNECTION\_URL* -> Die URL zum STOMP-Service.

## StompSessionHandler

Der StompClient …

## ChatController

Der StompClient …

## WebSocketConfig

Der StompClient …

Was ist ein Broker?

Ein Broker

Was ist ein Endpoint?

Neues Spiel in der Lobby

Authentisierter Client erzeugt neue Spiel definition mittels „Neues Spiel -> Erstellen“ schaltflächen. Im Anschluss dessen wird

/snakeserver  
Endpoint für http-Handshake und STOMP-Upgrade

/topic/  
Broker-Topic für Broadcast-Messages

/topic/messages  
Dient dem Austausch von Chat-Nachrichten

/topic/games  
Informationen über alle (laufenden) Spiele in der Lobby

/app/games/{gameId}

Ein Aufruf an /app/games/1 mit einer GameInputMessage hat zufolge, dass alle Clients welche bspw. die URL „ws://localhost:8080/app/game/1“ subscribed sind alle folgenachrichten der Clients erhalten – somit ist die Eingabe für Multiplayer gesichert =)

# STOMP / Websockets – Use Cases

Der folgende Abschnitt beschreibt die verschiedenen Anwendungsfälle in denen STOMP / WebSocket Nachrichten ausgetauscht werden.

## Neues Spiel wird erstellt

Spieler -> NewGameScreen -> Sendet LobbyMessage an Backend -> /app/games

LobbyMessage enthält(erstellender Spieler („admin“), spielDefinition)

Backend -> empfängt nachricht auf /app/games -> ruft broadcastGames() auf

Backend überprüft ob Spiel in LobbyController vorhanden

Wenn nicht -> trage Spiel als runningGame ein, setze den erstellenden Spieler als Admin

Backend -> Broadcasted LobbyMessage an subscribte Clients

Client -> onLobbyMessageReceived() wird aufgerufen -> update der UI

## Spieler tritt Spiel bei

1. Spieler wählt in ListView ein RunningGame aus
2. Spieler klickt auf „Trete Spiel bei“ / joinGame-Button
3. Client -> Sendet PlayerJoinsGameMessage
   1. PlayerJoinsGameMessage enthält (spieler, runningGame)
4. Backend empfängt nachricht in StompLobbyController
5. Backend ruft broadcastPlayerJoinedGameToClients auf
6. Backend trägt Client als Spieler des RunningGames ein, teilnehmer ++
7. Broadcast an subscribte Clients
8. Client erhält Callback auf onPlayerJoinedGameMessageReceived
9. Client aktualisiert UserInterface mit LobbySpielen
10. Client subscribed sich auf Spiel mit ausgewählten GameTopic
    1. = Subscribtion auf Stomp-Pfad – z.b. /app/games/1/<SpielerName>

## Admin startet Spiel

1. Admin-Client -> Sendet GameSessionMessage an Backend
2. GameSessionMessage enthält STARTING, userName, runningGame