Hier sollte etwas sehr abstraktes stehen, fast schon unsichtbar

Inhaltsverzeichnis

Kapitel 1

Einleitung

Hier steht das Vorwort

Kapitel 2

Spielideen

2.1 Umsetzung virtueller Spiele in der physischen Welt

bla bla bla bal

2.2 Integration einer virtuellen Welt in Bewegungsspiele

blub blub blub blub

Kapitel 3

Spielkonzepte

3.1 Integration virtueller Objekte in die physische Umgebung

blog blog blog blog

3.2 Darstellung der physischen und virtuellen Umgebung

Bitte hier einen sinnvollen Text einfügen

3.3 Kompass

OOOOOOOOOOOOOORIENTIEEEERUNG

3.4 Arkustische und haptische Orientierungshilfen

Und hier wieder einfügen.

3.5 Synchronisation zwischen mobilen Endgeräten

Und hier wieder einfügen.

3.6 Kollision virtueller Objekte

teeeeeeeeeeeeeeeeeeeeeeeeeeest

3.7 Einsammeln von Objekten

teeeeeeeeeeeeeeeeeeeeeeeeeeeeeest

3.8 Geschwindigkeitsmessung

aaaaaaaaaaaaaaaaaaaaaaaargh

3.9 Mensch-Maschine-Kommunikation

aaaaaaaaaaaaaaaaaaaaaaaargh

3.10 Chat

aaaaaaaaaaaaaaaaaaaaaaaargh

Kapitel 4

Technische Loesungen

4.1 Positionsermittlung

4.1.1 LocationManager vs LocationClient

Location, Location, Location

4.1.2 Genauigkeit

Ich kann zielen, aber mit der Schüssel brauch ich das nicht

4.2 Kollisionsabfrage

4.2.1 Kollisionsabfrage über Abstandsmessung

Die Android API stellt eine Methode zur Verfügung, die die Entfernung zwischen zwei Punkten, die über geographische Koordinaten bestimmt sind, berechnet [1]. Bei der Kollisionsabfrage unterscheiden wir vier Fälle: sowohl das aktive, wie auch das passive beteiligte Objekt können jeweils durch einen Kreis oder einen Polygonzug (eine Schlange) realisiert sein. Das aktive Objekt symbolisiert meist den die Kollision mit dem passiven Objekt auslösenden Spieler. Das passive Objekt kann ein aufsammelbares Bonus-Objekt sein oder ein anderer Spieler. Falls beide Objekte Kreise sind wird eine Kollision ausgelöst, falls die Entfernung zwischen beiden Kreismittelpunkten kleiner ist, als die Summe der Radien.

$(M_{aktiv} - M_{passiv})^2 < r_{aktiv} + r_{passiv}$

Bei einer Kollision mit einer Schlange, wird dieses Kriterium auf alle Glieder der Schlange angewendet. Die Kollision zwischen dem aktiven Objekt und einem der Eckpunkt des Polygonzugs führt zur Kollision mit der Schlange. Falls das aktive Objekt ein Polygonzug ist, müssen nur Kollisionen mit dessen erstem Element (dem Kopf der Schlange) berücksichtigt werden, da Kollisionen mit Schwanz immer von anderen Objekten ausgelöst werden. Die weiteren Fälle sind analog zu den abgehandelten. Während Abstandsmessung als Kriterium für Kollision von Kreisen gut funktioniert, muss bei der Kollision von Polygonzügen darauf geachtet werden, dass die Radien der Objekte genügend groß gewählt werden, damit sich auch bei schwankender Genauigkeit der GPS-Werte und damit sehr unregelmäßigen Abständen zwischen den Gliedern der Schlange, die Radien aufeinanderfolgender Glieder überschneiden.

4.2.2 Kollisionsabfrage über Linienintersektion

uhhhhh das war knapp

[1] http://developer.android.com/reference/android/location/Location.html

4.3 Kartendarstellung mit Android

4.3.1 GoogleMaps vs OpenStreetMaps

Reaaaaaaaaaaaaaady?? FIGHT!!!!!!

4.4 Server-Client-Kommunikation

4.4.1 Serverseitige vs Clientseitige Logik

XMPP [2]

Um den Zustand der einzelnen mobilen Geräte zu synchronisieren und Chat-Nachrichten zu übertragen, ist ein Server nötig. Hierzu bieten sich zwei unterschiedliche Modelle an. Bei der klassischen Server-Client-Architektur übernimmt der Server einen Großteil der Berechnungen. Der Server hält einen zentralen, im Zweifelsfall gültigen Status Die (Thin-)Clients übertragen die Nutzereingaben und Sensordaten an den Server, der daraus einen neuen Zustand ermittelt und diesen den Clients mitteilt. Alternativ bietet sich das Publish-Subscribe-Pattern an. Dabei hält der Server keinen Zustand, sondern leitet bloß Nachrichten von einem Client (Publisher) an einen anderen Client (Subscriber) weiter. Im konkreten Fall eines interaktiven Spiels wird beispielsweise die Position eines Spielers an alle Spieler in der selben Spielinstanz weitergeleitet. Die (Fat-)Clients müssen dabei alle Berechnungen übernehmen. Dies fordert leistungsfähigere Endgeräte. Diese haben sich als genügend leistungsfähig erwiesen. Bei fast jeder Änderung müssen alle Clients aktualisiert werden, da die Programmlogik redundant auf jedem Client vorliegt. Dies verringert Skalierbarkeit, Wartbarkeit und Erweiterbarkeit. In Echtzeitsystemen kommt zusätzlich hinzu, dass, da es keinen definitiven, zentralen Zustand gibt, auf verschiedenen Clients zum gleichen Zeitpunkt unterschiedliche Zustände vorliegen. Da diese Zustände für Berechnungen genutzt werden, kann dies zu Unklarheiten und Fehlern führen, die abgefangen werden müssen. Das Publish-Subscribe-Pattern bietet jedoch den Vorteil, dass es in diesem konkreten Fall Internet-Bandbreite spart, da kleinere Daten übertragen werden müssen. Beispielsweise muss bloß der aktuelle Standpunkt eines Spielers übertragen werden, nicht die daraus resultierenden Daten, die der Fat-Client selber berechnet. Dieser Punkt war ausschlaggebend, da bei mobilen Endgeräten die Internet-Bandbreite eine stark limitierte Ressource ist. Zusätzlich verringert sich beim Publish-Subscribe-Pattern die Komplexität der Anwendung, da der Zustand nicht doppelt modelliert werden muss. Dies macht die Anwendung weniger fehleranfällig

Thin-Client	Fat-Client
- Datenübertragung	+ Datenübertragung
- Zustand doppelt	+ Zustand nur auf Client
+ zentraler, definitiver Zustand	- verteilter Zustand
+ Rechenleistung	- Rechenleistung
+ Skalierbarkeit	- Skalierbarkeit
+ Wartbarkeit	- Wartbarkeit
+ Erweiterbarkeit	- Erweiterbarkeit

[2] http://xmpp.org/extensions/xep-0060.html

4.4.2 Übersicht Server-Technologien

Die Kommunikation zwischen Client und Server kann als Push- oder Pull-Kommunikation realisiert werden. Bei der Pull-Kommunikation fordert der Client die benötigten Informationen vom Server an. Im Falle einer Echtzeitanwendung muss dies in regelmäßigen Abständen geschehen. Bei der Push-Kommunikation sendet der Server, wenn sich der Zustand ändert, unaufgefordert Informationen an die betroffenen Clients. Im Falle mobiler Endgeräte muss Push-Kommunikation über Sockets laufen, da diese Geräte keine IP-Adresse haben. Pull-Kommunikation über HTTP-Requests umgesetzt hat eine nicht hinnehmbare Verzögerung bewirkt. Die finale Implementation verwendet daher Push-Kommunikation. Eine spezialisierte Server-Architektur, die das Publish-Subscribe-Pattern und Push-Kommunikation verwendet ist XMPP [**?**]. Es wurde eine allgemeine Server-Lösung bevorzugt, um die Flexibilität zu erhöhen. Wir haben Node.js[3] und JeroMQ[4] verglichen. Wir haben uns für JeroMQ entschieden, um Server und Client in der selben Programmiersprache umsetzen zu können.

4.4.3 Umsetzung mit ZeroMQ

Ist es ein Vogel? Ein Flugzeug? Nein!! ES IST ZeroMQ !!!!!!!!!!!!!!!!!!!!!!einself

[3] http://nodejs.org/

[4] https://github.com/zeromq/jeromq

4.5 GUI

Die GUI (Grafic User Interface) wurde nur mit den Android bzw. GooglePlay Services (inkl. Googlemaps) um gesetzt. Die mitgelieferten Möglichkeiten des Android SDK haben in dieser Hinsicht für die GUI-Umsetzung dieses Projektes vollkommen zufriedenstellend. Die komplette GUI setzt sich aus 2 Activities zu sammen, die im folgenden ein wenig genauer beleuchtet werden.

4.5.1 Login-Screen

Diese Activity wird zuerst aufgerufen und zeigt einen Bildschirm auf dem sich der Spieler einen Benutzernamen aussucht und danach mit dem „Start“ Button zu nächsten Activity wechselt, welche das eigentliche Spiel zeigt.

4.5.2 Swipe-Screen

Schon in der sehr frühen Entwicklungsphase war festzustellen, das die verschiedenen Elemente der GUI - wie im folgenden weiter erläutert- zu zahlreich sind, um sie auf einen Bildschirm umzusetzen. Die eigentliche Kartendarstellung wäre sonst zu klein gewesen. Also wurde entschieden die verschiedenen Elemente auf weitere Bildschirme zu verteilen. In den ersten entwürfen geschah dies über einzelne Activities. Um gewisse Android Komfort-Funktionen und Gesten dem Nutzer zu verfügung zu stellen wurden die zunächst eigenständigen Activites zu Fragments umgebeut, die dann in einem so genannten Swipe-Screen zusammengefasst werden. In diesem werden die Fragments als Tabs organisiert und der User kann entweder durch „wischen“ oder durch klicken auf die Tabs durch die GUI Navigieren.
Ein weiterer Vorteil ist ebenfalls, dass benachbarte Tabs jeweils ein wenig vorgeladen(laden der Widgets) bzw. noch im Speicher behal-

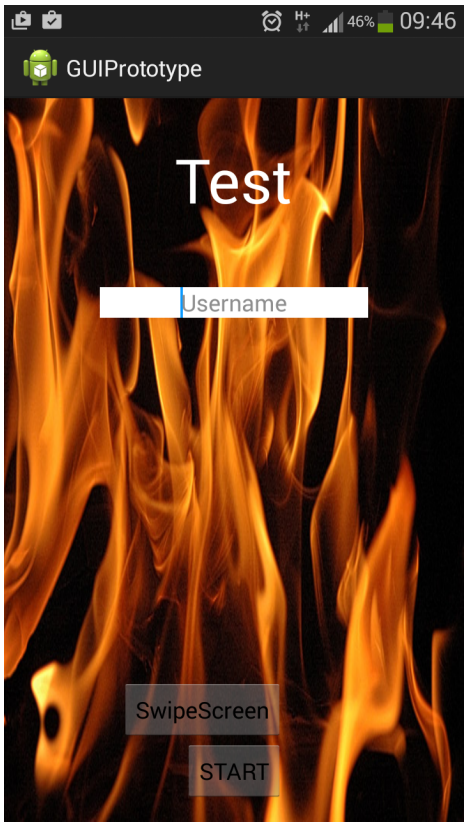

Abbildung 4.1: Login Screen

ten werden. Zum einen wird so sichergestellt, dass der Tab-Wechsel per wischen „geschmeidig“ abläuft, aber auch gibt es so nur sehr geringe Ladezeiten zwischen den einzelenen Bildschirmen. Zudem ist die von Google angepriesene „Wiederverwendbarkeit“ von Fragments als UI (User Interface) ebenfalls nützlich, wenn weitere ähnliche Spiele umgesetzt werden sollen.

Chat-Screen

In desem Fragment wird die möglichkeit des Chattens zwischen mehreren Spielern umgesetzt. Die grafische Umsetzung des Chats wurde der von IRC (Instant Relay Chat) Clients nachempfunden und ist entsprechend simpel gelöst. Es wird der jeweilige Benutzername, Uhrzeit und die eigentliche Nachricht angezeigt. Die Eingabe der Chat-Nachricht erfolgt in einem Text-Eingabe-Feld. Die Anzeige der Chat-Nachrichten erfolgt in einem einfachen Textanzeige-Feld (TextView) was wiederum in einem scrollbaren Feld (SrcollView) liegt. Hierdurch ist es möglich durch alle empfangenen Nachrichten „durchzuscrollen“. Wird eine Chat-Message (genaueres in Kapitel 4.4) empfangen, wird diese mittels Stringmanipulation an das Textfeld angehängt. Hierbei ist zu beachten, dass die selbst verschickten Nachrichten erst an den Server gesendet werden und dann jeweils an die entsprechenden Nutzer. Somit kann es aufgrunde von Übertragungsverzögerungen dazu kommen, das die eigene Nachricht verzögert angezeigt, jedoch ist die korrekte Reihenfolge der Nachrichten gewahrt.

Abbildung 4.2: Chat Screen

Map-Screen

In diesem Fragment wird die Karte in einem weiterem Fragment angezeigt. Je nachdem welches Spiel gespielt wird zu zusätzliche Widgets vorhanden, wie z.B. Interaktions Buttons, (Team-)Punkte anzeige usw.

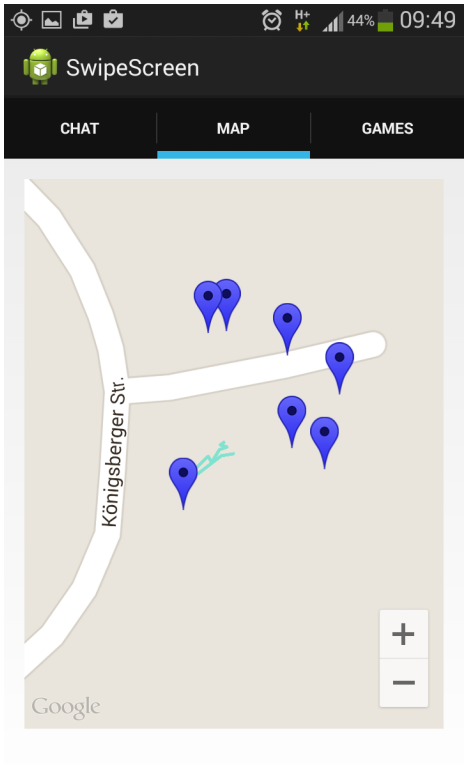

Abbildung 4.3: Map Screen

Game-Screen

In diesem Fragment werden momentan aktive Spiele angezeigt. Auch ist es möglich neue Spiele zu erstellen. Die Anzeige der Liste der Spiele erfolgt in einer ListView. Diese wird durch entsprechend angefordete Informationen über andere Spiele geupdatet. Diese Informationen werden durch Anfrage bei anderen Benutzer die sich eingeloggt haben über einen bestimmen Message-Typ abgerufen. Die Listen-Elemente sind interaktiv. Ein klick auf das entsprechende Spiel startet den beitritt. Um ein Spiel eines gewissen Typs zu Starten wählt man in einen Dropdown-Menü (bei Android Spinner) den entsprechenden Modus aus in klickt auf create. Man selbst betritt dieses Spiel und eintrag in der Spiele-Liste wird vorgenommen.

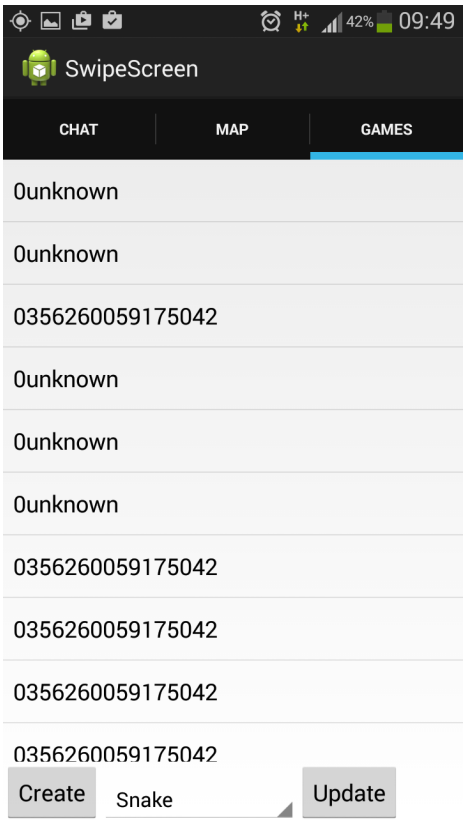

Abbildung 4.4: Game Screen

4.6 AndereSensorik

Seeeeeeeeeeeeeeeeeeeeeeeeeeeeeensation

Kapitel 5

Implementation von Beispielapps

5.1 Snake

5.1.1 Spielidee

aaaaaaaaarg gkls akj dasdlj ad ka kd askd

5.1.2 Spiellogik

aaaaaaaaarg gkls akj dasdlj ad ka kd askd

5.1.3 Umgesetzte Features

aaaaaaaaarg gkls akj dasdlj ad ka kd askd

5.2 Snake

5.2.1 Spielidee

aaaaaaaaarg gkls akj dasdlj ad ka kd askd

5.2.2 Spiellogik

aaaaaaaaarg gkls akj dasdlj ad ka kd askd

5.2.3 Umgesetzte Features

aaaaaaaaarg gkls akj dasdlj ad ka kd askd

Kapitel 6

Fazit

Hier sollte das Fazit Stehen

Kapitel 7

Ausblick

7.1 Weitere Features

Features featuring more Features

7.2 Weitere Spiele

Mögen die Spiele beginnen