An alle: dringend Quellen suchen, die das Literaturverzeichnis füllen

Zusammenfassung

In der Vergangenheit konnte zwischen virtuellen Spielen und Spielen in der physischen Welt klar unterschieden werden. Die große Verbreitung von Mobiltelefonen mit hoher Rechenleistung und ständiger Internetverbindung macht es möglich, Spiele zu entwickeln, die virtuelle und physische Umgebung vereinen. In diesem Projekt soll untersucht werden, inwieweit es möglich ist, so traditionelle Bewegungsspiele mit einer App zu unterstützen oder klassische Videospiele ins Freie zu verlegen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

ein Satz, der das Thema der Arbeit paraphrasiert

Was bedeutet Interaktivität für uns in diesem Zusammenhang? Wir wollen es schaffen die physische Umgebung in eine virtuelle Welt einfließen zu lassen. Umgekehrt sollen Entscheidungen in der Realität von virtuellen Ereignissen beeinflusst werden. Smartphones bieten durch Sensorik und interaktiver Kommunikation die Grundlage hierzu. So ist es möglich ortsübergreifend mit dem Spielpartner zu interagieren. Dabei ist man auch ortsunabhängig im Austausch von Daten durch das Mobilfunknetz. Def. virtuelle Realität Eine weitere Ausprägung wäre die sogenannte “Augmented Reality. zuerst ein Satz, der AR definiert, Unterschied zur VR, dann konkrete Umsetzung auf dem Handy (folgenden Satz umformulieren) Hierbei werden meist in dem dargestellten Bild einer Smartphone-Kamera zusätzliche optische Informationen eingeblendet.

Welche zusätzlichen (auch nicht umgesetzte) Features bieten Smartphones vs Computerspiele vs ”Kinderspiele”/Bewegungsspiele ? wie kann man das Smartphone in einem solchen Spiel einsetzen? was ist der Mehrwert Smartphone einzusetzen?

Was gibts schon (Geocashing, Ingress) ? haben wir was neu gemacht ?

Warum haben wir uns für Android als Plattform entschieden? Die anderen Alternativen wären iOS und Windows Phone gewesen. Letzteres hat einen zu geringen Marktanteil Quelle und die Entwicklung für iOS kommt mit hohen Lizenzkosten daher.

Überblick folgende Kapitel (einfach nennen)

Kapitel 2

Spielideen

2.1 Umsetzung virtueller Spiele in der physischen Welt

Unter einem virtuellen Spiel verstehen wir jene, die sich ausschließlich mit bzw. auf einem Computers spielen lassen.
Es folgen eine Reihe von Spielideen, die in der virtuellen Welt sehr populär sind und sich für die Integration in die physische Welt eignen.

Snake

Ein absoluter Klassiker, der früher auf keinem Handy fehlen durfte. Gerade wegen der Popularität und Einfachheit recht interessant es in die physische Welt zu integrieren. In einem begrenzten Areal gilt es eine Schlange geschickt zu steuern. Es müssen kleine Items eingesammelt werden, wodurch die Schlange wächst. Außerdem muss darauf geachtet werden, dass weder der Rand des Areals noch die eigene Schlange berührt wird.

Capture the Flag

Nicht nur in der virtuellen Welt spielbar, aber wohl dort erst richtig bekannt geworden. Auch hier ist ein recht einfacher Spielmechanismus ausschlaggebend für einen leichten Einsatz in der physischen Welt. Zwei Teams treten gegeneinander an. Jedes Team besitzt eine Flagge, dessen Standort für das gegnerische Team bekannt ist. Es wird versucht die Flagge des jeweils anderen Teams zum Standort der eigenen Flagge zu bringen.

Domination

Meist in Kriegsszenarien integrierter virtueller Spielmodi, der sich mit wenig Aufwand in die physische Welt übertragen lässt. Zwei Teams treten gegeneinander an. Jedes Team hat einen Punktestand, der zu Anfang gleich ist.

Sinkt der Punktestand auf Null, so ist das Spiel für dieses Team verloren. Es gibt verschiedene Areale, die eingenommen werden können. Hat ein Team mehr Areale eingenommen als das andere, läuft der Punktestand des Teams, das weniger Areale in besitzt hat gegen Null.

2.2 Integration einer virtuellen Welt in Bewegungsspiele

Unter Bewegungsspielen verstehen wir jene, die sich ausschließlich in der physischen Welt abspielen und außerdem mit örtlicher Bewegung der Spieler zu tun haben.

Natürlich können wir die Integration auch von der anderen Seite betrachten. Es kann in etablierte Bewegungsspiele der physischen Welt, eine virtuelle Umgebung eingefügt werden. Es folgen simple Spielevorschläge:

Verstecken

Es gibt eine Person, die der Suchende ist. Alle weiteren Mitspieler verstecken sich möglichst gut und versuchen vom Suchenden nicht entdeckt zu werden. Hat der suchende Spieler alle Mitspieler gefunden ist das Spiel vorbei.

Fangen

Ein Spieler versucht alle anderen Mitspieler durch berühren zu „fangen“. Hat er geschafft einen Mitspieler zu berühren ist nun dieser Mitspieler seinerseits dran einen anderen zu fangen.

Topfschlagen

Ein Spieler bekommt die Augen verbunden. Er versucht anhand von Tipps der Mitspieler („näher dran“ oder „weiter weg“) einen Gegenstand zu finden.

Kapitel 3

Spielkonzepte

Einige Hilfsmittel, die zur Umsetzung der oben genannten Beispiel-Spiele notwendig oder hilfreich sind.

3.1 Integration virtueller Objekte in die physische Umgebung

Um ein virtuelles Spiel in die physische Umgebung integrieren zu können, müssen auch alle Objekte des virtuelles Spieles in die physische Umgebung gebracht werden. Bei vielen Objekten kann dies durch einen eingeschränkten Zufall geschehen. Z.B. bei Snake die Items. Bei dem ein oder anderen Spiel soll es eine Basis für jedes Team geben. Dabei bietet sich an dieses auf eine relativ freie Fläche festzusetzen. Generell sollte darauf geachtet werden, dass die Objekte nicht innerhalb eines Gebäudes landen oder in nicht begehbares Terrain platziert werden.

Dies kann mit Hilfe der folgenden technischen Lösungen umgesetzt werden: Kartendarstellung (s. 4.3), Kollisionsabfrage (s. 4.2), Positionsabfrage (um sicherzustellen, dass Objekt in der Nähe des Spieler erstellt wird, s. 4.1).

3.2 Darstellung der physischen und virtuellen Umgebung

Die physische Welt in eine virtuelle Umgebung zu übertragen kann oft sehr hilfreich sein. Sei es in einer Karte als Übersicht oder lediglich eine Anzeige ob man sich innerhalb bzw. außerhalb des Spielfelds befindet. Eine Karte hat insoweit den Vorteil, dass man dort auch noch virtuelle Gegenstände einfügen kann, die in der physischen Welt nicht vorhanden sind. Z.B. die nächsten Items bei Snake. Dazu sind Kartendarstellung (s. 4.3) und Positionsermittlung (s. 4.1) nötig.

3.3 Kompass

Eine Richtungsangabe kann eine Karte ersetzen. Sei es um bei „Capture the Flag“ die Richtung der Fahne des Gegners anzuzeigen oder für die „Snake“ das nächste einzusammelnde Item. In der Regel hat es hier weniger Sinn, wenn die Kompassnadel nach Norden zeigt.
Technische Lösungen: Positionsermittlung (s. 4.1), GUI (s. 4.5), Andere Sensorik (s. 4.6)

3.4 Arkustische und haptische Orientierungshilfen

Akustische oder haptische Signale können ebenso Hinweise geben auf in der Nähe befindliche Interessengebiete (z.B. ertönen eines Signals oder Vibration bei erreichen eines bestimmtem Umkreises von einem Item). Können aber auch als Bestätigung eingesetzt werden, wenn z.B. etwas eingesammelt wurde.
Technische Lösungen: Kollisionsabfrage (s. 4.2), Andere Sensorik (s. 4.6)

3.5 Synchronisation zwischen mobilen Endgeräten

Damit alle Mitspieler überhaupt miteinander Spielen können, ist es wichtig, dass die einzelnen, sich im Spiel befindlichen mobilen Geräte, synchronisiert werden. Das heißt, das jedes Gerät auf irgend eine Art und Weise mit den anderen Geräten kommuniziert. Es ist wichtig, dass Daten, die jeder sehen kann, bei jedem identisch und zur selben Zeit angezeigt werden.
Technische Lösungen: Server-Client-Kommunikation (s. 4.4)

3.6 Kollision virtueller Objekte

Treffen zwei virtuelle Objekte aufeinander muss in der Regel ein Event ausgelöst werden. Wird bei Snake z.B. der eigene Schwanz berührt, was laut der Regeln nicht erlaubt ist, muss dies dem Spieler mitgeteilt werden und evtl. weitere Ereignisse ausgeführt werden.
Technische Lösungen: Kollisionsabfrage (s. 4.2), Positionsermittlung (s. 4.1)

3.7 Einsammeln von Objekten

Eine Variante der Kollision mit virtuellen Objekten ist das Einsammeln. Wenn ein Spieler in Reichweite eines Items ist, das es einzusammeln gilt, kann dies entweder automatisch passieren oder über eine Aufforderung auf dem mobilen Gerät. Zur Bestätigung, dass etwas eingesammelt wurde, kann nun wiederum ein akustisches oder haptisches Signal gegeben werden.

Technische Lösungen: Positionsermittlung (s. 4.1), Kollisionsabfrage (s. 4.2)

3.8 Geschwindigkeitsmessung

Um das Spielgeschehen besser zu kontrollieren zu können, kann eine Messung der Geschwindigkeit von Vorteil sein. Möchten wir z.B. bei Capute the Flag dem Fahnenträger nicht erlauben eine gewissen Geschwindigkeit zu überschreiten, ist eine Geschwindigkeitsmessung unabdingbar.
Technische Lösungen: Positionsermittlung (s. 4.1), Server-Client-Kommunikation (s. 4.4), Andere Sensorik (s. 4.6)

3.9 Mensch-Maschine-Kommunikation

Das komplette Spielgeschehen lebt nach der Integrierung von mobilen Endgeräten von der Kommunikation zwischen Mensch und Gerät. Wird auf dem Endgerät ein Kompass angezeigt, muss der Spieler insofern reagieren, dass er sich in die richtige Richtung dreht. Wenn er etwas einsammeln möchte, kann es erforderlich sein, dass ein Button gedrückt wird.
Technische Lösungen: GUI (s. 4.5), Kartendarstellung (s. 4.3), Andere Sensorik (s. 4.6)

3.10 Chat

Um mit seinen Teammitgliedern oder dem gegnerischen Team zu kommunizieren gibt es eine Reihe von Möglichkeiten. Wenn man sich außer Sicht- und Hörweite befindet kann dies über ein mobiles Gerät stattfinden. Beim Chat wird eine Nachricht verschickt, die der Empfänger auf seinem Gerät einsehen kann und dann seinerseits eine Antwort verfassen kann.
Technische Lösungen: Server-Client-Kommunikation (s. 4.4), GUI (s. 4.5)

Kapitel 4

Technische Loesungen

4.1 Positionsermittlung

4.1.1 LocationManager vs LocationClient

Im folgenden werden zwei mögliche Dienste zur Positionsermittlung verwendet. Der LocationManager[1] und der LocationClient[2].

LocationManager

Der LocationManager ist ein von Android bereitgestellter Dienst, der über den GPS Sensor die Position des Smartphones ermittelt. Ein Vorteil bei verwendenden dieses Dienstes ist, das man nicht an die Dienste von Google gebunden ist, und somit theoretisch auch für nicht Google-Konforme Androidbetriebssysteme entwickeln kann. Da nur der GPS Sensor verwendet wird, wäre eine entsprechende Anwendung sogar unabhängig von einer Internetverbindung. Auch wird dieser Dienst auch noch von älteren Android Versionen unterstützt. Nachteilig wird in einer Forendiskussion [3] der hohe Akku verbrauch genannt.

LocationClient

Der LocationClient ist ein von Google bereitgestellter Dienst, der über den GPS Sensor und andere Quellen, die Google zur Verfügung stellt, die Position des Smartphones zu ermitteln. Als Vorteile dieses Dienstes werden in der oben schon erwähnten Forendiskussion wird erwähnt, dass dieser Dienst

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

[2] `http://www.doc.ic.ac.uk/project/2013/271/g1327125/android-studio/sdk/extras/google/google_play_services/docs/reference/com/google/android/gms/location/LocationClient.html`

[3] http://stackoverflow.com/questions/20908822/android-difference-between-locationmanager-addproximityalert-locationclien bessere Quelle

Akku-schonender sei. Auch sei er zuverlässiger. Die anderen Quellen zur Positionsermittlung könnten auch zu einer genauen Positionsermittlung beitragen, wenn es eventuell Übertragungsprobleme zum GPS-Satelliten gibt. Nachteilig an diesem Dienst ist, dass man eine Internetverbindung benötigt. Außerdem werden die Google Play Services vorausgesetzt, welche erst ab Android Version 2.2 zur Verfügung stehen. Zudem ist man von Google abhängig.

Messdaten

Um LocationManager und LocationClient für unsere Zwecke vergleichen zu können, wurde ein Testprogramm entwickelt, welches jeweils in zwei separaten Activities GPS Messdaten in einer Datei sammelt. Es werden jeweils Koordinaten und Genauigkeit gesammelt. Zur optischen Darstellung der Ergebnisse wird aus den aufgezeichneten Koordinate eine Linie (PolyLine[4]) gezeichnet. Als Vergleich wird eine Luftlinie zwischen Anfangs- und Endpunkt gezogen.
Im Folgenden werden nun die ausgewerteten Sensordaten dargestellt.

Tabelle 4.1: Location Versuch 1

	locationClient	locationManager
Datum	10.12.2014	
Ort	Universität Koblenz, Campus	
Wetter	Bewölk, leichter Niesel	
Gerät	Samsung Galaxy S3	
Genauigkeit	7,049	8,698
Updates/Sekunde	0,936	1,016

Tabelle 4.2: Location Versuch 2

	locationClient	locationManager
Datum	14.12.2014	
Ort	Ransbach-Baumbach, L307richtung Mogendorf	
Wetter	Wetter:,leicht bewölk leicht windig	
Gerät	Samsung Galaxy S3	
Genauigkeit	5,482	6,216
Updates/Sekunde	1,003	0,969

Hierbei muss noch ergänzt werden, dass der Genauigkeitswert folgendermaßen beschrieben wird. An Längen- und Breitengrad der Position wird ein Kreis mit Radius der Genauigkeit gezeichnet. Es besteht eine 68-prozentige

[4] `https://developer.android.com/reference/com/google/android/gms/maps/model/Polyline.html`

Wahrscheinlichkeit, dass sich die Position innerhalb des Kreises Befindet[5]. Es gilt, je kleiner der Wert desto besser. In den Tabellen wurde jeweils das arithmetische Mittel der aufgezeichneten Genauigkeiten eingetragen.

4.1.2 Genauigkeit

Nun folgen eine grafische Darstellung der Genauigkeitsaufzeinungen die mit dem Testprogramm gemacht wurden

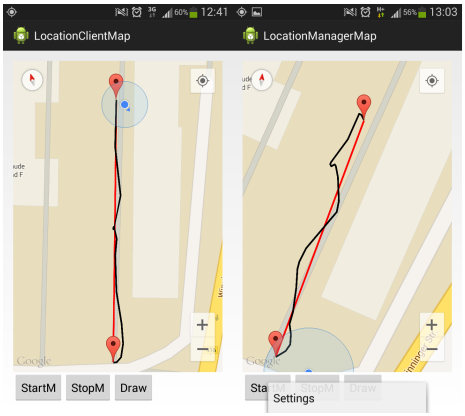

Abbildung 4.1: Location Versuch 1

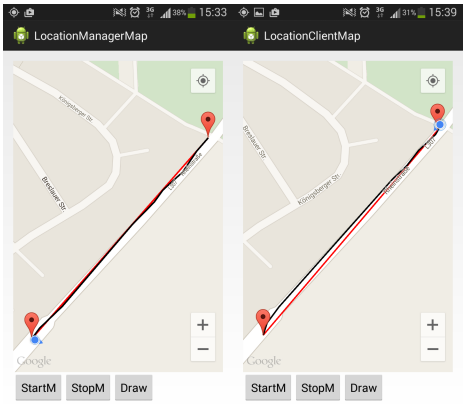

Abbildung 4.2: Location Versuch 2

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

4.1.3 Fazit

Die Messdaten belegen, dass der LocationClient die minimal die besseren Ergebnisse liefert. Dies ist der Fall bei der Genauigkeit der Sensordaten, als auch wei der grafischen Darstellung. Zweiteres ist ein wichtiger Faktor beim Umsetzten des Snakespiels. Die Unterschiede bei Updates pro Sekunde sind verschwindend gering und nicht eindeutig. Ein weiteres Faktor ist bereits, dass sich schon für die Kartenanzeige für GoogleMaps entschieden wurde. Da der LocationClient minimal die besseren Ergebnisse liefert und schon ein dienst von Google verwendet wird, wird zur Positionsermittlung in diesem Projekt der LocationClient verwendet.

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 [6]. Bei der Kollisionsabfrage müssen wir für das Snake-Spiel vier Fälle unterscheiden: 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.

$|M_{aktiv} - M_{passiv}| < r_{aktiv} + r_{passiv}$

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

fehlt

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

4.3 Kartendarstellung mit Android

4.3.1 GoogleMaps vs OpenStreetMaps

Die Basis der Spielkonzepte ist eine Karte der reellen Umgebung, die mit Android dargestellt wird. Realisiert wurde dies mit Google Maps[7], obwohl Open Street Maps[8] auch eine Alternative gewesen wäre. Im folgenden möchten wir beide Optionen durchleuchten und die Vor- und Nachteile sowie die Unterschiede betrachten. Google Maps ist Eigentümer aller Karten und Informationen die diese beinhalten. OSM dagegen kommt mit einer offenen Lizenz daher. Die zugrundeliegenden Geodaten der Karten stellt Google Maps nicht zur Verfügung, OSM hingegen schon. Beide erhalten tägliche Updates, wobei jedoch die Satellitenkarte von Google Maps nur alle 1 bis 3 Jahre aktuell gehalten wird. Diese wird von OSM allerdings nicht angeboten. OSM bietet volle Funktionalität in allen Ländern, wogegen diese bei Google Maps auf spezielle Länder beschränkt ist. Als Datentyp im Backend nutzt Google Maps JSON, OSM setzt hingegen auf XML. Beide bieten Bibliotheken für Java und Javascript. Wobei die letzteren für OSM von Drittanbietern bereitgestellt werden. Beide nutzen GPS und Wi-Fi des Mobilen Endgerätes für die Positionsbestimmung und bieten die Darstellung der Bewegungsrichtung. Google Maps unterstützt zusätzlich den Zugriff auf Mobilfunkzellen für die Ortsbestimmung. Wir haben uns für Google Maps entschieden, weil die API sehr gut und zentral dokumentiert ist. Die Integration in die Android App war unkompliziert und die Karten-Funktionalitäten waren leicht zu nutzen. Google bietet hier für alle Zwecke ausführliche Tutorials an. Auch die Geodaten benötigten wir nicht. Einzig die Anzeige der Karte und die Interaktion der Benutzer untereinander war uns wichtig.

[7] Link zur Doku
[8] Link zur Doku

4.4 Server-Client-Kommunikation

4.4.1 Serverseitige vs Clientseitige Logik

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 [Schäffer(2003), Millard et al.(2010)Millard, Saint-Andre, and Meijer].

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

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 [Millard et al.(2010)Millard, Saint-Andre, and Meijer]. Es wurde eine allgemeine Server-Lösung bevorzugt, um die Flexibilität zu erhöhen. Wir haben Node.js[9] und ZeroMQ[10] verglichen. Beides sind asynchrone Server-Technologien. JeroMQ[11] ist eine Implementation von ZeroMQ in Java. Wir haben uns für JeroMQ entschieden, um Server und Client in der selben Programmiersprache umsetzen zu können.

4.4.3 Umsetzung mit ZeroMQ

Für Echtzeit-Spiele benötigt man eine mehr oder weniger 'mehr oder weniger' streichen permanente Datenübertragung. Nicht nur weil langsame Updates irreführend sein können, sondern insbesondere wenn man mit mehreren Spielern gegeneinander spielt und die Versionen nicht synchron sind. Bei Verstecken etwa könnte ein um sechs Sekunden verspätetes Update dazu führen, dass man nach einer Richtungsänderung glaubt, in die richtige Richtung zu gehen, obwohl das Update vorher hätte eintreffen sollen. Bei Snake kann man z.B. durch eine Schlange laufen, ohne eine Kollision zu bekommen, wenn die eigene Version überzeugt ist, dass da noch keine Schlange ist, weil der Zustand des Spiels auf dem eigenen Smartphone noch nicht aktuell ist.

Generell kann man davon ausgehen, dass je schneller das Spiel ist und je mehr virtuelle Objekte einbezogen werden, desto häufiger und schneller müssen Updates kommen. Daher wollten wir eine Übertragung, die performant ist, schnell viele Nachrichten senden kann und außerdem asynchron Nachrichten verarbeitet, also nicht jedes mal auf eine Antwort des Empfängers wartet, bevor eine weitere Nachricht abgeschickt werden kann.

ZeroMQ kann all das. In unseren Probeläufen war ZMQ (ZeroMQ) in der Lage, mehrere Tausend Nachrichten innerhalb einer Sekunde abzuschicken.

[9] `http://nodejs.org/`
[10] `http://zeromq.org/`
[11] `https://github.com/zeromq/jeromq`

ZMQ arbeitet standardmäßig asynchron. Außerdem sendet ZMQ Nachrichten erneut, falls die Nachricht beim ersten Senden nicht an kam, sodass es relativ unwahrscheinlich wird, dass eine Nachricht komplett ausfällt und Spieler mit zwei unterschiedlichen Versionen spielen. Das ermöglicht eine relativ kleine Serverarchitektur, da der Server nicht eine Version des momentanen Spiels speichern muss. Außerdem ist es recht einsteigerfreundlich. Das alles machte es für unsere Test-Apps zum idealen Kandidaten.

Hier kommen Vergleiche zwischen ZMQ und seinen Konkurrenten (andere messaging Protokolle) hin.

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 sind 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.

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.

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, also mehrere Bildschirme. 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“ (swipe) oder durch klicken auf die Tabs durch die GUI Navigieren.
Ein weiterer Vorteil ist ebenfalls, dass benachbarte Tabs jeweils vorgeladen (Laden der Widgets) bzw. noch im

Abbildung 4.3: Login Screen

Speicher behalten 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[12]) was wiederum in einem scrollbaren Feld (SrcollView[13]) 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 4.4: Chat Screen

[12] http://developer.android.com/reference/android/widget/TextView.html
[13] http://developer.android.com/reference/android/widget/ScrollView.html

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.

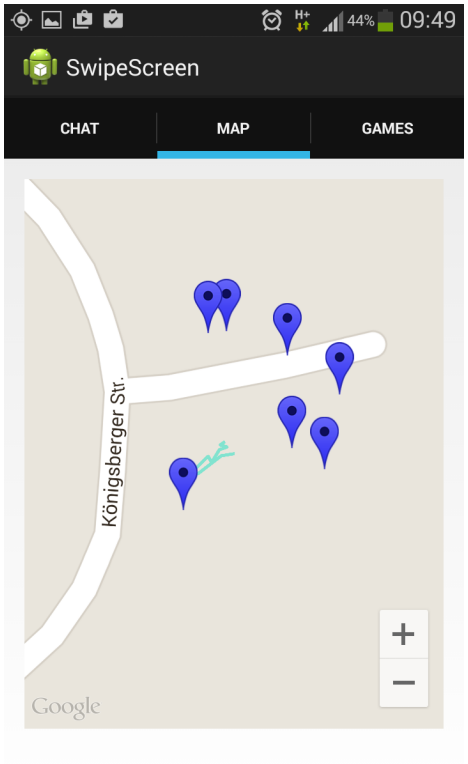

Abbildung 4.5: Map Screen

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 scrollfähigen Liste (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.

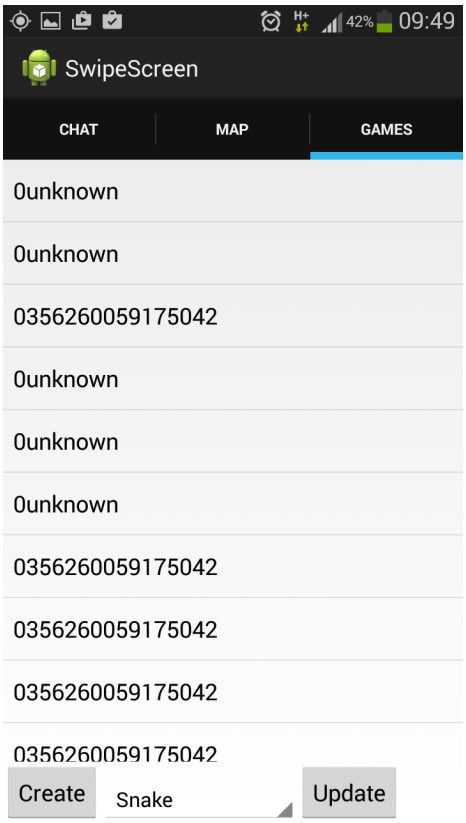

Abbildung 4.6: Game Screen

4.6 AndereSensorik

Recherche, welche weitere Sensoren (Bewegungssensor, Termometer? etc.) Android hat, Ã¼berlegen ob man die verwenden kann

Kapitel 5

Implementation von Beispielapps

5.1 Snake

5.1.1 Spielidee

Als erstes Spiel, das wir umsetzen, haben wir uns für Snake entschieden. Die Grundregeln von Snake sind nahezu jedem bekannt, da es auf allen Nokia Handys der 90er Jahre mitgeliefert wurde. Des weiteren lässt sich das Spiel relativ leicht von der virtuellen Welt in die physische übertragen, da die Steuerung nur auf der Bewegung des Kopfes der Schlange basiert, was man in der Realität durch die Bewegung des Spielers steuern kann. Wir haben uns für eine Mehrspieler Variante entschieden bei der in einem Spiel mehrere Zielpunkte gleichzeitig angezeigt werden, die bei Berührung den Punktestand des Spielers erhöhen und die Schlange etwas 'etwas' streichen vergrößern. Ziel des Spiels ist es schneller als die anderen eine einstellbare Anzahl von Punkten einzusammeln. Wenn man mit sich selbst oder einer anderen Schlange kollidiert wird der eigene Punktestand und die Länge der Schlange zurückgesetzt.

unbedingt erwähnen was chicken sind

5.1.2 Spiellogik

Unser System besteht aus 2 Komponenten. Eine Android-App mit der gesamten Spiellogik und einem Server über den die Kommunikation zwischen den mobilen Endgeräten abläuft.

Server

Unser Server arbeitet mit Java und ZeroMQ. Er öffnet zwei Kommunikationskanäle mit dem Client. Der erste ist ein ZeroMQ Request Reply Socket

Paar über das die Endgeräte Nachrichten an den Server senden. Der zweite ist ein Publish Subscribe Socket Paar über das die Nachrichten möglichst schnell an Gruppen von Endgeräten weitergeleitet werden. Eine Nachricht besteht bei uns aus 3 Teilen: Der Empfänger der Nachricht, der Nachrichtentyp und der eigentlichen Nachricht. Wir nutzen ZeroMQs multipart Message Feature um diese Teile voneinander getrennt bei der Kommunikation zu übermitteln. Hauptsächlich sendet der Server Nachrichten einfach an den Empfänger, welcher entweder ein einzelner Client oder alle Clients die sich in eine Spielsession befinden ist, weiter. Des weiteren nutzen wir den Server um die Gamesessions zu verwalten. Wenn eine Nachricht vom Typ create_game empfangen wird schreibt der Server die ID des Spiels in eine Liste ein. Diese Liste wird einem Client gesendet wenn er sie über eine Nachricht vom Typ Get_Gamelist anfragt.

Client

Unsere Android Applikation besteht aus 3 Fragmenten zwischen denen man durch wischen wechseln kann. Das erste Fragment ist ein kleiner Chat über den Spieler Textnachrichten an ihre Mitspieler im gleichen Spiel senden können. Hier sollte ein Screenshot hin das verbraucht Platz und so Das zweite Fragment ist die Karte auf der wir unser Spiel darstellen. Hier sollte ein Screenshot hin Für die Darstellung der Karte verwenden wir GoogleMaps. Im dritten Fragment kann ein Spieler eine neue Spielsession erstellen oder einem bereits laufendem Spiel beitreten. Die Liste der momentan laufenden Spiele wird per Knopfdruck vom Server abgefragt. Noch ein dritter Screenshot oder so Im Hintergrund laufen zwei Threads für die Kommunikation. Der eine kümmert sich um den Empfang von Nachrichten und beinhaltet ein ZeroMQ Subscriber Socket welches vom Publisher auf dem Server alle Nachrichten, die direkt an die ID des Clients oder die ID des Spiels in dem er sich befindet adressiert sind, empfängt. Der zweite sendet Nachrichten an den Server. Des weiteren läuft ein LocationClient welcher immer wenn ein Positionsupdate reinkommt die neue Position als Nachricht an das ganze Spiel weiterschickt.

Beschreibung der eigentlichen Spiellogik (Schlangen Chickens etc.) fehlt

5.1.3 Umgesetzte Features

fehlt

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

Literaturverzeichnis

[Millard et al.(2010)Millard, Saint-Andre, and Meijer] Peter Millard, Peter Saint-Andre, and Ralph Meijer. Publish-subscribe. `http://xmpp.org/extensions/xep-0060.html#intro`, 2010. [abgerufen am 05. März 2015].

[Schäffer(2003)] Bruno Schäffer. Probleme bei thin-clients. `http://www.sigs.de/publications/js/2003/01/schaeffer_JS_01_03.pdf`, 2003. [abgerufen am 05. März 2015].