Bachelorarbeit

Hendrik Geßner, 751352

Inhaltsverzeichnis

1 Grundlagen

1.1 Spielidee

1.2 Existierendes Spiel

- hält sich nicht ganz an AODV
- RREQ / RREP / Routenaufbau nicht nach Standard -> kein RREP zurückgesendet
- Automatisches Neighbour Find in einfachstem Schwierigkeitsgrad stellt Verknüpfungen zu Nachbarn her (RERR bei verlust des Nachbarn) -> nicht standardkonform

1.3 Schwierigkeiten beim Spielen

In verschiedenen Testläufen mit dem existierenden Spiel sind wiederholt Schwierigkeiten aufgefallen, die das Spielerlebnis beeinträchtigt haben oder die Vorbereitungsphase sehr langwierig machten.

Da für das Spiel immer wieder Smartphones verwendet wurden, die nicht für das RouteMe-System konfiguriert wurden, mussten vor jedem Spiel mit jedem Smartphone folgende Punkte überprüft werden:

- WLAN eingerichtet: Jedes Smartphone musste mit dem eingesetzten Router verbunden sein, um mit dem Server kommunizieren zu können. Hilf half der Einsatz von Wi-Fi Protected Setup (WPS).
- Akku geladen: Jedes Smartphone musste vollständig geladen sein, um während des Spieles nicht auszugehen. Dabei ist zu beachten, dass die im Spiel genutzten Hardware-Ressourcen Display, GPS-Empfänger und WLAN-Modul besonders energieverbrauchend sind.
- URL-Lesezeichen einrichten: Die eingesetzte URL basierte auf einer IP-Adresse und case-sensitiven URL-Bestandteilen. Nicht alle Spieler hatten Vorerfahrung in der Benutzung von Android-Smartphones. Somit stellte sich bereits das Starten eines Browsers oder das Eingeben einer URL auf der Touch-Tastatur als Herausforderung dar. Um beim Spielen zu vermeiden, dass dass die Spieler bei einem Bedienungs-Fehler oder Software-Fehler die URL vollständig eingeben müssen, wurden Lesezeichen auf dem Startbildschirm der Smartphones eingerichtet.

Zudem sind verschiedene menschliche Aufgaben in Vorbereitung auf das Spiel notwendig. So müssen alle Spieler in die Funktionsweise und Ziele des Spieles eingeführt werden. Da das Spiel mit wechselnden Gruppen gespielt wurde, wurde jede Gruppe in die Funktionsweise eines Spiel-Interfaces eingeführt, direkt bevor sie dieses Interface verwenden sollte. Da die Instruktoren keine ausgebildeten Trainer sind, ist es vorgekommen, dass die Erklärung zwischen den Gruppen unterschiedlich war. Außerdem konnte es passieren, dass einzelne Punkte besonders ausführlich erklärt oder vergessen wurden.

Zusammenfassend ist festzuhalten, dass die Einrichtung von acht Smartphones etwa zwei Stunden benötigt hat. Zusammen mit der Einrichtung des Servers und des erforderlichen Netzwerkes wurden etwa vier Stunden benötigt.

- Übersichtlichkeit in Indoor-UI mangelhaft
- Verbindungen zwischen Knoten / aktuelles Netzwerk nur mit Erfahrung oder Zeit zu erkennen
- Spiel-Geschwindigkeit und Spieler-Geschwindigkeit passen nicht zusammen -> Routing-Vorgänge fühlen sich langsam an, Spieler bewegen zu schnell, um Routen, die mehr als drei Knoten beinhalten, zu nutzen
- server ausgelastet
- lange, ständig wechselnde ip-basierte addressen
- IP muss immer auf jedem einzelnen Client geändert werden
- Sehr ähnliche IPs -> automatische vervollständigung des Browsers unnütz, da nicht alle passenden IPs angezeigt
- Spielfeld rechteckig entlang längen- und breitengrade, aber nicht drehbar oder in Form änderbar

1.4 AODV

Ad hoc On-Demand Distance Vector (AODV) ist Routing-Algorithmus, der für Mobile Ad-hoc-Netzwerke (MANETs) ausgelegt ist. Jeder Knoten ist gleichberechtigt. Durch das Versenden und Weiterleiten von Nachrichten an Nachbarn entsteht ein Netzwerk. Dieses Netzwerk ist ständigen Veränderungen unterworfen, da sich Nachbarschaftsverhältnisse ändern und Knoten ausfallen können. Diese Veränderungen erfordern eine permanente Anpassung der Routen.

AODV gehört zu den reaktiven Routing-Algorithmen. Routen werden erst bei Bedarf ermittelt. Alle an der Route beteiligten Knoten überwachen die Aktualität und senden Fehlernachrichten, falls eine Route ausfällt.

Es wird an einem Nachfolger namens AODVv2 gearbeitet.

1.4.1 RREQ und RREP

Routen werden über eine Kombination von Route Request (RREQ) und Route Reply (RREP) gefunden. Benötigt ein Quellknoten eine Route zu einem Zielknoten, so sendet er einen RREQ an alle Nachbarknoten. Jeder Nachbarknoten überprüft, ob er bereits eine Route zum Zielknoten kennen. Falls eine Route bekannt zum Zielknoten bekannt ist oder falls der Knoten selbst der Zielknoten ist, so sendet er einen RREP an den Quellknoten zurück. Ist keine Route bekannt, so leitet er den RREQ an seine eigenen Nachbarknoten weiter. Nachbarknoten speichern eine Route zum Quellknoten des RREQ.

1.4.2 Hello-Nachrichten

Knoten können Hello-Nachrichten versenden, um ihren Nachbarn Verbindungs-Informationen zu Verfügung zu stellen. Dabei sollen Hello-Nachrichten nur verwendet werden, wenn der aussendende Knoten teil einer aktiven Route ist. Bleiben Hello-Nachrichten für einen bestimmten Zeitraum aus, so kann der empfangende Knoten davon ausgehen, dass die Route ausgefallen ist.

1.4.3 RERR

Fällt eine Route aus, so wird ein Route Error (RERR) versendet. Empfangende Knoten können anhand des RERR ihre Routen-Einträge korrigieren.

1.5 Pervasive Games

1.6 Android

1.6.1 Programmiermodell

- Subject-Observer-Pattern
- Besteht aus `Activitys` und `Services`

Exceptions, die nicht abgefangen werden, lassen die Applikation abstürzen.

1.6.2 Lebenszyklus

HIER BILD AUS ANDROID-DOKU!

1.6.3 Rechtemanagement

Um Zugriff auf Hardware-Ressourcen oder sensible Informationen zu erlangen muss die Applikation vorher diese Rechte am System anmelden. Dies geschieht über das `ApplicationManifest`, in dem die benötigten Rechte eingetragen werden. Vor der Installation der Applikation bekommt der Benutzer eine Übersicht über die für die Applikation erforderlichen Rechte. Mit der Installation werden der Applikation die eingetragenen Rechte gewährt. Rechte werden immer vollständig gewährt. Es ist nicht möglich, Rechte im Nachhinein zu entziehen oder einzuschränken.

Typische Rechte sind der Zugriff auf das Internet, Zugriff auf den GPS-Empfänger oder Zugriff auf die auf dem Smartphone gespeicherten Kontaktdaten.

1.6.4 UI-Thread

Wie viele andere Desktop-UIs setzt auch Android auf ein UI-Thread-Modell. Änderungen an GUI-Elementen dürfen nur auf dem UI-Thread durchgeführt werden. Versuche, Änderungen außerhalb des UI-Threads durchzuführen, werden mit Exceptions zurückgewiesen.

Um Änderungen auf dem UI-Thread durchzuführen müssen Aufgabenblöcke, in Java `Runnables`, explizit auf den UI-Thread geschoben werden.

1.6.5 Hintergrundoperationen

Es gibt verschiedene Möglichkeiten, Operationen im Hintergrund laufen zu lassen:

- Android-AsyncTasks
- Android-Services
- Java-Threads

AsyncTasks Hintergrundoperationen, die nur für eine sehr begrenzte Zeit laufen, können in einem `AsyncTask` durchgeführt werden. Aus Kompatibilitätsgründen wird pro Applikation und zu jedem Zeitpunkt standardmäßig immer nur ein `AsyncTask` ausgeführt. Ein `AsyncTask` muss vom UI-Thread gestartet werden. Android übernimmt die Bearbeitung im Hinter-

grund. Bei Beendigung der Bearbeitung wird eine Methode mit dem Ergebniss der Hintergrund-Aufgabe aufgerufen. Android stellt sicher, dass diese Methode im Kontext des UI-Threads läuft. Somit können UI-Manipulationen, die auf dem Ergebnis der Hintergrundoperation beruhen, ohne weiteren Aufwand durchgeführt werden.

Services Lange Hintergrundoperationen, die unabhängig von der GUI laufen sollen, können in Services durchgeführt werden. Wie eine `Activity` hat auch ein `Service` einen eigenen Lebenszyklus. Die Bindung zwischen `Activity` und `Service` wird beim Erstellen des `Service` festgelegt. `Service` und `Activity` können unabhängig voneinander arbeiten und undabhängig voneinander Lebenszyklus-Phasen durchlaufen.

Threads Die aus der Java-Welt bekannten Threads sind auch in Android verfügbar. Sie bieten feingranulare Zugriffs- und Einstellungsmöglichkeiten für parallele Verarbeitung, bieten aber keine Unterstützung für die Manipulation des UI-Thread. Sie sind am besten als Zwischenstufe für die Portierung von Java-Code auf Android-Systeme geeignet.

1.6.6 Fragments

Aus Einzelblöcken zusammengesetzte UI-Elemente, die wiederverwendet werden können, heißen *Fragments*. Diese haben ein eigenes Layout und eigenen Java-Code. Kommunikation mit *Fragments* erfolgt über Interfaces. Um die Übersichtlichkeit in Applikationen zu erhalten, wird empfohlen, logisch zusammenhängende Einheiten in *Fragments* zu kapseln.

Eines der bekannten *Fragments* ist das `SupportMapFragment` für Googles Maps-Service.

1.6.7 Sensoren

Android-Smartphones besitzen eine Vielzahl von verfügbaren Sensoren. Dazu gehören unter anderem GPS-Empfänger, Beschleunigungs- und Lage-Sensoren. Die Vielzahl an automatisch auslesbaren Sensoren machen Smartphones besonders interessant für pervasive Anwendungen. Die große Verbreitung von Android sorgt dabei für eine Vielzahl an unterschiedlichen Smartphones mit sehr verschiedenen Konfigurationen und Ausstattungen.

Android bietet einfachen Zugriff auf die verfügbaren Sensoren. Die Applikation ist dabei selbst verantwortlich, sich an das verfügbare System anzupassen und nicht vorhandene Sensoren zu kompensieren. Sensoren werden über den `SensorManager` angesprochen. Android verwendet auf für Sensoren

das *observer pattern*. Im Folgenden werden drei Sensoren vorgestellt, die in der Arbeit verwendet wurden.

- Motion-Sensor
- Tilt-Sensor
- GPS-Sensor

1.6.8 APIs

- Google Maps -> Play Services
- Einbindung von Google Maps
- Deployment und API-Keys
- Aufbau und Benutzung sehr ähnlich zu JS-API Version 3

1.6.9 Einbindung von Drittbibliotheken

Wie von Java bekannt können externe Bibliotheken über JAR-Archive eingebunden werden. Dabei ist allerdings ganz besonders auf die Kompatibilität mit der Android-VM zu achten. Android bietet nur eine Untermenge der aus der Java-API bekannten Klassen und Interfaces an. Somit lassen sich viele Bibliotheken nur mit Anpassung des Quellcodes nutzen.

Das Android-Eclipse-Plugin ermöglicht die Erstellung von Android-Bibliotheken. Eine Android-Bibliothek ist ein Android-Projekt, bei dem das `android.library`-Flag gesetzt ist. Android-Bibliotheken sind nicht ausführbar und können deshalb auch nicht separat getestet werden. Android-Bibliotheken können sowohl von Android-Projekten als auch von Java-Projekten verwendet werden. Im Gegensatz dazu können Java-Projekte nicht als Bibliotheken in Android-Projekten eingebunden werden.

1.6.10 Audio

- Audio muss beantragt werden
- Für kurze Audio-Schnippsel gibt es ein besonderes Flag, das Hintergrundgeräusche / Hintergrundmusik nicht stoppt, sondern nur herunterregelt

1.7 GPS

- Anzahl GPS-Satelliten
- Verfahren für Genauigkeit
- Hindernisse
- Fixierung auf Hardware
- Alternativen zu GPS
- Erweiterungen von GPS (AGPS, DGPS)
- Für DGPS -> Anbieter oder Brandeburg Viewer + Eigene Implementierung

1.8 Abstandsbestimmung auf der Erde

- Erde ist keine Kugel
- Abstandsbestimmung zweier Punkte auf Oberfläche über direkten Weg gehen meist durch Oberfläche hindurch
- Bei geringem Abstand und Fehlertoleranz kann Fehler vernachlässigt werden
- Für unser Szenario ausreichen
- GRAFIK FÜR ABSTANDSBERECHNUNG EINFÜGEN

1.9 QR-Codes

- QR-Tags sind Schwarz-Weiß-Muster
- Pervasive-Computing-Vorlesung
- Benötigte Hardware: Kamera
- Rest: Softwarelösung
- Einfach in Herstellung, günstig, viele Leser

1.10 WiFi

- Kommunikation mit Server
- Alternative: GPRS / UMTS
- Smartphones nicht mit SIM-Karte ausgestattet -> kein Zugriff auf diese Netze

1.11 GWT / JSNI

1.12 Rest / RESTful

1.12.1 Ein- und Ausgabeformate

- Nicht festgelegt
- XML und JSON am häufigsten

1.12.2 Hyperlinks

- XML hat XLink
- JSON fehler Hyperlinks
- Verschiedene Ansätze, JSON um Links zu erweitern, bisher kein Standard

2 Konzept

2.1 Portierung

- Portierung von JS nach Android-Java
- Struktur des Codes übernehmen
- bereits Erfahrung in der Portierung von PHP nach Java
- kopieren / einfügen, anschließend an Java-Syntax anpassen
- Wrapper für UI-Elemente und Browser-Funktionen
- UI vollständig anders umsetzen
- Einstieg an Android-Gegebenheiten anpassen

2.2 Umsetzung von Pervasivität

- Schüttel-Gesten für Einsammeln
- Audio-Feedback für Ping aussenden / empfangen
- Vibrations-Feedback für Ping empfangen
- Pervasive Karte
- Items an Horizont

2.3 Hello-Nachrichten

Benutzer finden sich untereinander nur noch, wenn sie einen Ping aussenden. Wie soll der Prozess des Findens und Haltens von Nachbarn umgesetzt werden?

Blick auf das AODV-Protokoll: Knoten senden regelmäßig HELLO-Nachrichten aus. Wenn HELLO-Nachrichten zu lange ausbleiben wird Abbruch der Verbindung angenommen, Routing-Tabelle aktualisiert und RERR gesendet.

HELLO-Nachrichten werden nur gesendet, wenn Knoten Teil einer aktiven Route ist. Außerdem tragen HELLO-Nachrichten eine Sequence-Number in sich, die zur Bestimmung der Aktualität von Routen verwendet wird.

- Spielidee: Knoten bekommen Punkte für den Aufbau einer Route. Vermutung: Dadurch wird das Aufbauen von Routen gestärkt, was auch der Sinn eines MANETs ist. Außerdem sollen HELLO-Nachrichten laut RFC nur gesendet werden, wenn Knoten Teil einer aktiven Route ist. Deshalb muss für die sinnvolle / standardkonforme Nutzung von HELLO-Nachrichten der Aufbau von Routen gefördert werden.
- Ziel: Routenbewusstsein stärken? Idee: Aktive Routen anzeigen. Einwand: Wenig Platz auf Smartphone-Screen, da Button für Gegenstand einsammeln und ein paar Gegenstands-Informationen sowie Hilfstexte verwendet wird. Lässt sich dieser Platz besser nutzen? Ja!

Umsetzung auf Mobile: Spieler senden Pings manuell aus. Sollen sie Nachbarn finden, indem sie selbst einen Ping aussenden (Aktiv-Sonar eines U-Bootes) oder sollen sich Nachbarn zeigen, indem der Spieler ihren Ping empfängt?

Häufigkeit der Pings: Spieler sollten nicht durchgehend ohne Nachteil pingen können, da sonst Effekt des manuellen Pings verloren geht. Verschiedene Möglichkeiten:

- Bestimmte Anzahl an Pings für Spieler verfügbar, neue werden zeitlich oder über Item verfügbar
- Pings sind nur mit Cooldown verfügbar
- Pings haben Malus-Effekt (z.B. auf Batterie oder Punkte)

Wie werden Nachbarschaftsverhältnisse wieder aufgebhoben? Sollen Nachbarschaftsverhältnisse automatisiert aufgehoben werden?

Blick auf AODV-Protokoll: Für HELLO-Nachrichten gilt ein HELLO_INTERVAL und ein ALLOWED_HELLO_LOSS. Daraus ergibt sich, wann Routen als ungültig erkannt werden sollen.

- Spielerische Idee: Vor Ablauf des HELLO_INTERVAL*ALLOWED_HELLO_LOSS den Knoten warnen, der das HELLO ausgesendet hat, damit seine Routen und gültigen Nachbarn nicht verloren gehen. So lässt sich leichter ein Gefühl für HELLO_INTERVAL entwickeln.

2.4 Nachrichtenversand

Auf Level zwei dürfen Routen nicht mehr selbst gewählt werden sondern müssen aus einer Liste von Vorgaben ausgewählt werden. Was soll auf dem Auswahlfeld angezeigt werden?

- Nickname der Knoten, zwischen denen geroutet werden soll. Nicknamen sind keine Beschränkungen auferlegt, außerdem geben Nicknamen keine Auskunft über Position / Status / Distanz der Route / Punkte der Route etc. Indoor-Spieler bauen bisher keine Verbindung zu einzelnen Knoten (also Verknüpfung von Nickname und Position / Verhalten), was bringen also Nicknames?
- Alternative: Distanz / Anforderung / Vermutete Punkte / Vermutete Hops
- Außerdem: Bonus-Eigenschaft
- Bei Maus-Hover: Start- und Zielknoten auf Karte hervorheben

Welche Routen sollen zur Verfügung gestellt werden?

- Nur derzeit verknüpfte Wege
- Beliebige Auswahl an Knoten. Bei 6 Knoten sind das 15 ($\frac{n*(n-1)}{2}$) mögliche Wege. Bei Bidirektionalität sogar 30.

2.5 Spielerleichterungen

- Nutze Möglichkeit, Android-Apps per URL zu starten
- Parameterübergabe für WLAN-SSID und WLAN-Schlüssel über URL-Parameter
- Nachteil: Kein Aufruf von URLs ohne Netzwerkverbindung. Nicht klar, warum dem so ist
- Workarounds: QR-Reader aus App heraus starten. Nachteil: Abhängigkeit von bestimmter QR-App, die installiert sein muss
- Alternative: Bild selbst untersuchen. Hoher Eigenaufwand
- QR-Code wird bei Spielerstellung generiert. PDF mit QR-Code und kodierten Daten wird erstellt
- Dynamisches DNS verhindert IP-Umstellung auf Clients

2.6 Skalierbarkeit

- Zwei Standpunkte: Beserer Server und stärkere Clients
- Servlerlosigkeit mit PushFork
- RESTful-Philosophie umsetzen über dokumentbasiertes Vorgehen

3 Implementation

3.1 Umsetzung von Pervasivität

- Shake-Modul entwickelt
- Modul für selbstdrehende Karte entwickelt (Umsetzung über Lagesensor und Kompass)
- WLAN-Verbindungs-Modul entwickelt

3.2 Hello-Nachrichten

- Auf Serverseite: Verbindungen bekommen Last-Ping-Timestamp
- Anzeigeideen hinter Ping-Feedback
- Synchronisierter Timestamp
- Mit Server synchronisierter Timestamp
- Server verbirgt Timestamp, arbeitet intern mit Server-Timestamp
- Knoten senden ping und ping-invalidate
- In Verbindungstabelle ist last-ping angegeben.
- Statischer Malus für jeden Ping von fünf Batterie-Punkten
- Punkte für Nachbar haben bleibt gleich

3.3 Nachrichtenversand

- Ziel: Möglichst viel Code behalten, um Aufwand gering zu halten
- Mix aus GWT und JS
- Originaler JS-Code für Maps erhalten, Click wird in JS an Server gesendet
- Bei Update von Server wird Subset von Infos per JSNI nach GWT gegeben
- GWT-UI reagiert darauf
- Einfach und gut
- Routen werden per Zufall zwischen den verfügbaren Knoten in Client erstellt
- Verhindert Server-Eingreifen und Server-Unterstützung

JS-Testing Verwendung von Karma (vormals Testacular) als Testrunner, Jasmine als Framework.

-

3.4 Spielerleichterungen

- QR-Code-Starten gestestet
- Start über beliebige externe App

3.5 Codestruktur

- Einsatz von ReSharper 101
- ReSharper-Metrik „Fat“ und „Tangled“

4 Evaluierung

- Evaluation in großem Stil -> Julians Arbeit
- Aus Zeitgründen war eine Evaluation des Lernerfolges nicht möglich
- Deshalb: Funktionale Evaluation

Szenario: Ping versenden

1. Spiel starten
2.

Szenario: Ping empfangen
Szenario: Gegenstand einsammeln
Szenario:

5 Zusammenfassung / Ausblick

- Spielerische Aspekte stärken
- Ausscheiden von Knoten: AODV definiert Reboot-Verfahren
- Audio einbinden: Bei Ping senden / Ping empfangen
- Vibration einsetzen: Bei Ping empfangen
- Ablauf eines Pings verdeutlichen
- App benötigt noch viele Anpassungen

- Weitere Level
- Anzeige, wann Ping ausläuft
- Stärkere Annäherung an Android best practices
- Aufteilung in Async Tasks
- Stärkere Entkopplung von Komponenten, um Wiederverwendbarkeit in anderen Android-Applikationen zu ermöglichen
- QR-Code-Generierung in Admin-Frontend einbauen
- Abhängigkeit von externem QR-Reader lösen
- Abhängigkeit von Netzwerk lösen, WLAN-Funktionalität sonst witzlos
- Indoor-Routen ein- und ausanimieren
- UI-Anpassungen im Indoor

6 Abkürzungsverzeichnis

AODV Ad hoc On-Demand Distance Vector
MANETs Mobile Ad-hoc-Netzwerke
RREQ Route Request
RREP Route Reply
RERR Route Error
WPS Wi-Fi Protected Setup

Literatur