Erweiterte Konzeption und Implementierung eines Lernspiels über Routing-Protokolle für den Informatik-Unterricht

Hendrik Geßner

22. Oktober 2013

Student:
Hendrik Geßner, Matrikelnummer: 751352
Fachsemester: 08
Hochschulsemester: 08

Adresse:
Stahnsdorfer Straße 148a
14482 Potsdam
HENGE01@gmail.com

Institution:
Universität Potsdam
Institut für Informatik

Inhaltsverzeichnis

1 Einleitung

Die Idee des Lernspieles als Unterrichtsmethode ist schon seit langem bekannt, Beispiele reichen von Vorschulspielen bis zu Planspielen für Manager. Hier wird gerne von *Edutainment* gesprochen, der Verknüpfung von *Education* und *Entertainment*. *Pervasive Computing* ist ebenfalls ein bekanntes Feld, nachdem Mark Weiser im Jahr 1991 seine Vision des *ubiquitous computing* vorgestellt hat. [Wei91]

Die Verknüpfung von pervasiven Elementen mit einem Lernspiel ermöglicht reales Erleben von virtuellen Konzepten. Das pervasive Lernspiel *RouteMe* setzt im Bereich der Mobilen Ad-hoc-Netzwerke (MANETs) an, um Studenten die Funktionsweise von Routing-Algorithmen näher zu bringen. Das Kernkonzept beleuchtet verschiedene Algorithmen in unterschiedlichen Schwierigkeitsgraden. Erste Ergebnisse deuten auf deutliche Vorteile gegenüber der klassischen Lehre hin. [ZML12]

Die erste Version des Spieles entstand im Rahmen einer Diplomarbeit von Tobias Moebert. [MLZ11] Nach verschiedenen intra-universitären Tests wurde eine von Julian Dehne und Hendrik Geßner überarbeitete Version veröffentlicht. [DG13] Auf Grund der Komplexität des Konzeptes ergaben sich weitere Anknüpfungspunkte, die eine erweiterte Bearbeitung des Spieles interessant machten.

Einerseits existiert eine Arbeit zur Evaluation pervasiver Lernspiele, die verschiedene Mängel im bisherigen Umfang beleuchtet: So fehlen unter anderem unterschiedliche Level. [Jul12] Andererseits sind in den Evaluationsrunden verschiedene technische Aspekte aufgefallen, die eine erweiterte Bearbeitung des Spieles erforderlich machen. Beispielhaft sei hier die unzuverlässige GPS-Ortung genannt.

Das Ziel dieser Bachelorarbeit ist also eine Verbesserung des existierenden Spieles auf verschiedenen Ebenen: Zum einen sollen die Lerninhalte und Spielelemente weiter ausgebaut und Defizite abgebaut werden, um das Lernziel besser erreichbar zu machen. Zum anderen sollen die Zuverlässigkeit und Pervasivität des Spieles erhöht werden, um dem Lerner eine Konzentration auf das Spiel zu vereinfachen. Außerdem soll eine Grundlage für eine Erweiterung der Inhalte des Spieles geschaffen werden, um zu einem späteren Zeitpunkt einen reduzierten Entwicklungsaufwand zu haben.

Im Folgenden wird der Aufbau der Arbeit beschrieben. In Abschnitt 2 wird das bestehende Spiel vorgestellt. Anschließend wird auf die Konzepte für eine Weiterentwicklung eingegangen, bevor in Abschnitt 4 die Umsetzung dieser vorgestellt wird. Anschließend folgt eine Evaluation der Ergebnisse. Die Arbeit endet mit einem Fazit und einem Ausblick auf offene Fragestellungen.

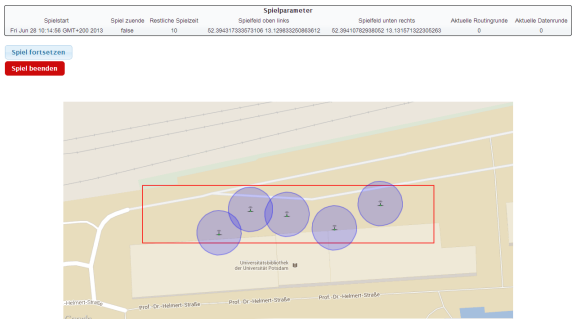

Abbildung 1: Admin-Sicht

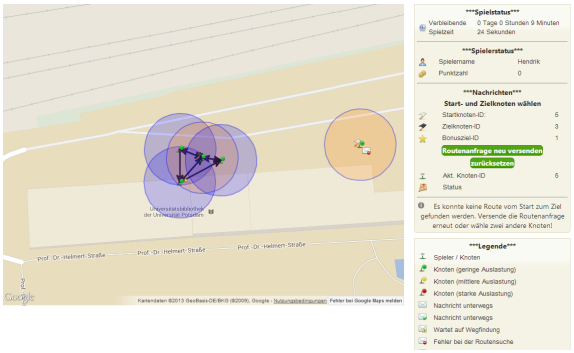

Abbildung 2: Indoor-Sicht

2 Ausgangszustand des Spieles

Wie bereits erwähnt besteht das Ziel des Spieles *RouteMe* im Erlernen von Routing-Algorithmen in MANETs. Eine ausführliche Einleitung ist bereits veröffentlicht. [ZML12] Vorgesehen ist eine Auswahl verschiedener Routing-Algorithmen mit jeweils drei Schwierigkeitsgraden. Derzeit ist eine angepasste Version von AODV [Int03a] mit nur einem Schwierigkeitsgrad implementiert.

Die Spieler werden in zwei Gruppen aufgeteilt, die gleichzeitig spielen und unterschiedliche Sichten auf das Spielgeschehen haben. Eine Gruppe von Spielern sitzt an klassischen Desktop-Rechnern, die andere Gruppe läuft mit Smartphones in einem festgelegten Gebiet herum. Während die Desktop-Spieler, im weiteren *Indoor-Spieler* genannt, Nachrichten zwischen Teilnehmern eines MANETs versenden, repräsentieren die Smartphone-Spieler, im weiteren *Knoten-Spieler*, die Teilnehmer dieses MANETs.

Die Aufgabe der Indoor-Spieler ist es also, in dem Netzwerk durch geeignete Auswahl von Start- und Zielknoten Nachrichten zu versenden und so Punkte zu sammeln. Diese Aufgabe wird dadurch erschwert, dass sich die Knoten im Netzwerk permanent bewegen, ausfallen und neu hinzukommen, sodass permanent Verbindungen entstehen und abbrechen. Indoor-Spieler haben eine Gesamtübersicht über das Spiel, wie in Abbildung 2 dargestellt ist.

Die Aufgabe der Knoten-Spieler ist es, zu überleben. Je länger Spieler überleben, je mehr Nachbarn sie haben und je mehr Nachrichten über sie gesendet werden, desto mehr Punkte sammeln sie. Jede dieser Möglichkeiten, Punkte zu sammeln, verbraucht aber Energie, wovon die Knoten nur einen begrenzten Vorrat haben. Energie und Sendereichweite können durch das Sammeln von virtuellen Gegenständen erhöht werden, die über das Spielfeld verteilt sind. Knoten-Spieler haben eine auf ihre eigene Sendereichweite beschränkte Sicht.

Als dritte Sicht auf das Spiel existiert das Admin-Interface. Es dient zum Erstellen, Manipulieren und Beenden eines Spieles. In der Admin-Übersicht werden erweiterte Details dargestellt, die für die Spieler nicht einsehbar sind. Ein Beispiel ist in Abbildung 1 dargestellt.

3 Konzeption

Im Folgenden werden die konzeptionalisierten Veränderungsvorschläge vorgestellt. Die Umsetzung dieser Vorschläge findet sich in Abschnitt 4.

3.1 Lerninhalte

Bei den hier beschriebenen Lerninhalten handelt es sich um Aspekte des Routing-Protokolls AODV [Int03a], die während des Spieles gelernt werden sollen. Es folgt eine kurze Einführung in die Grundkonzepte von AODV. Eine ausführliche Einführung gibt der RFC. [Int03a]

Ad hoc On-Demand Distance Vector (AODV) ist ein Routing-Algorithmus, der für 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. [Int13a]

Die Konzeption für das Spiel enthält drei Level, die aufeinander aufbauen. Ein Ziel dieser Arbeit ist die Schaffung des zweiten Schwierigkeitsgrades im Kontext des bereits implementierten Routing-Protokolles AODV. Im Folgenden sind die Level beschrieben:

Level 1 Spieler werden stark geführt. Knoten-Spieler müssen lediglich Gegenstände einsammeln und möglichst lange aktiv bleiben. Indoor-Spieler können Nachrichten versenden und dabei frei aus Start- und Zielknoten wählen. Die Details des Routing-Protokolls bleiben verborgen.

Level 2 Spielerführung ist reduziert. Knoten-Spieler müssen eine Nachbarsuche durchführen, um andere Knoten zu finden. Indoor-Spieler haben keine freie Start- und Zielwahl mehr, sondern müssen aus gegebenen Start- und Ziel-Paaren wählen. Soweit vom Routing-Protokoll unterstützt sind Paketpriorisierungen zu beachten.

Level 3 Spielerführung ist auf ein Minimum reduziert. Knoten-Spieler müssen komplexe Routing-Entscheidungen treffen. Indoor-Spieler sind mit einer Vielzahl von Prioritäten und Einschränkungen konfrontiert.

Level zwei ist eine Erweiterung von Level eins. Zu den aus Level eins bekannten Herausforderungen kommen die aktive Nachbarsuche und der Versand von Nachrichten aus einem vorgegebenen Pool hinzu. Als nächstes folgt eine Erklärung der Funktionsweise der aktiven Nachbarsuche im AODV-Protokoll. Anschließend wird beschrieben, wie für das Spiel eine Abbildung vom Protokoll-Verhalten auf das Spielgeschehen geschaffen wurde.

Aktive Nachbarsuche Die Nachbarsuche findet im AODV-Protokoll durch Route Reply (RREP)-Nachrichten statt. Eine RREP-Nachricht, die eine Reichweite von nur einem Knoten hat und dem Erhalt bestehender Routen dient, heißt HELLO-Nachricht. AODV definiert HELLO-Nachrichten als optional. Knoten senden regelmäßig HELLO-Nachrichten aus. Wenn die HELLO-Nachrichten eines Nachbarknotens zu lange ausbleiben, wird ein Abbruch der Verbindung angenommen, die eigene Routing-Tabelle aktualisiert und ein Route Error (RERR) gesendet. HELLO-Nachrichten sollen nur gesendet werden, wenn der sendende Knoten Teil einer aktiven Route ist. Außerdem tragen HELLO-Nachrichten eine Sequenz-Nummer und eine Lebensdauer in sich, die zur Bestimmung der Aktualität von Routen verwendet wird.

Abbildung auf das Spielelement Level zwei besagt, dass sich Benutzer untereinander nur noch durch aktive Suche finden. Im Folgenden geht es um die Frage, wie dieser Prozess im Spiel ausgestaltet werden soll.

Die Verbindung zwischen HELLO-Nachrichten und Nachbarschaftsverhältnissen ist mit dem Sonar von U-Booten vergleichbar: U-Boote besitzen ein passives und ein aktives Sonar. Bei Aktivsonaren wird ein Ping ausgesendet, der an Zielen reflektiert und vom aussendenden U-Boot empfangen wird. Passivsonare hingegen lauschen auf Geräusche und ermitteln so aussendende Ziele. Die Nachbarschaftssuche in AODV entspricht der Funktionsweise von Passivsonaren.

AODV besagt, dass HELLO-Nachrichten nur gesendet werden sollen, wenn eine aktive Route besteht und innerhalb des letzten HELLO_INTERVALs keine Broadcast-Nachricht gesendet wurde. [Int03a] Um standardkonform zu handeln, müsste der Spieler also eine Übersicht über die aktiven Routen und über die Broadcasts innerhalb des letzten HELLO_INTERVALs haben.

Alternativ besteht die Möglichkeit, diese Details zu abstrahieren und dem Spieler ein vereinfachtes Vorgehen anzubieten. Dabei könnten die AODV-Anforderungen bezüglich Intervall- und Routen-Gebundenheit durch Spielelemente nachgebildet werden. So lässt sich beispielsweise die Intervall-Limitierung über eine Limitierung der Häufigkeit von HELLO-Nachrichten darstellen.

Spieler sollten nicht durchgehend ohne Nachteil HELLO-Nachrichten versenden können, da sonst der Effekt von manuellen HELLO-Nachrichten verloren geht. Hierfür gibt es verschiedene Möglichkeiten. Einerseits kann der Vorrat an verfügbaren HELLO-Nachrichten begrenzt sein und zeitlich oder durch das Einsammeln von Gegenständen wieder aufgefüllt werden. Dieses Modell wird bereits bei der Knotenenergie und der Sendereichweite angewendet. Andererseits kann zwischen dem Aussenden von zwei HELLO-Nachrichten ein erzwungener Cooldown liegen, wie es aus diversen Computerspielen für Spezialfähigkeiten üblich ist. Als dritte Option kann eine HELLO-Nachricht mit einem Malus-Effekt verbunden sein, indem das Aussenden sich auf Batterie oder Punktzahl auswirkt.

Bei einem Blick auf das dargestellte Szenario, Routing in MANETs, fällt auf, dass nahezu alle realen Entscheidungen im Protokoll auf eine Minimierung des Energieverbrauches zurückzuführen sind. Da die Knotenenergie das beherrschende Element des Protokolles ist, liegt es nahe, dieses Element auch beim Aussenden von HELLO-Nachrichten einzubinden. Somit bietet sich das Malus-Modell für eine Umsetzung an.

Das Aufheben von Nachbarschaftsverhältnissen entspricht dem von AODV vorgeschriebenen Verhalten: Für HELLO-Nachrichten gilt ein HELLO_INTERVAL und ein ALLOWED_HELLO_LOSS. Daraus ergibt sich, wann Routen als ungültig erkannt werden. Es bietet sich an, den Spieler vor dem Ablauf des HELLO_INTERVAL*ALLOWED_HELLO_LOSS zu warnen, damit seine Routen und gültigen Nachbarn nicht verloren gehen. So lässt sich leichter ein Gefühl für das HELLO_INTERVAL entwickeln. Diese Warnung ist jedoch optional.

Routenauswahl durch den Spieler Auf Level zwei dürfen Indoor-Spieler Start- und Zielknoten nicht mehr selbst wählen, sondern müssen aus einer Menge von vorgegebenen Möglichkeiten wählen. Zudem sollen Paket-Prioritäten beachtet werden, soweit diese vom Protokoll unterstützt werden.

AODV bietet keine Paket-Priorisierung. Es existieren Vorschläge, den Standard zu erweitern, die bisher aber nicht übernommen wurden. [SP12]

Es stellt sich die Frage, wie dem Indoor-Spieler die verfügbaren Routen präsentiert werden, damit er eine davon auswählen kann. Es bietet sich an, eine einfache Liste mit Auswahlmöglichkeiten anzuzeigen. Um die einzelnen Listenelemente zu beschriften, sind verschiedene Möglichkeiten denkbar. So könnten die Listenelemente mit den Nicknamen der Knoten, zwischen denen geroutet werden soll, beschriftet werden. Es existieren keine Beschränkungen für Nicknamen, außerdem geben letztere keine Auskunft über Position oder Status der Knoten, die zu überbrückende Distanz oder die für die Route vergebenen Punkte. Aus Sicht der Indoor-Spieler wird das Verknüpfen von Knoten mit ihrer Position oder ihrem Verhalten derzeit nicht gefördert. Deshalb muss die Anzeige von Nicknamen kritisch betrachtet werden.

Alternativ könnten die Listenelemente mit der zu überbrückenden Distanz, einem Schwierigkeitsgrad, den zu erwartenden Punkten oder den zu erwartenden Hops beschriftet werden. Es muss allerdings die Frage gestellt werden, ob diese Informationen überhaupt einen Wert haben. Denkbar ist eine Vernachlässigung der Beschriftung der Listenelemente bei gleichzeitiger visueller Hervorhebung der Start- und Zielknoten auf der Karte. Eine begründete Entscheidung für ein Modell kann hier nur auf Basis intensiver Nutzertests erfolgen.

Der für die Indoor-Spieler entscheidende Faktor sind die erhaltenen Punkte pro Route. Deshalb erscheint es sinnvoll, mit Bonuspunkten vergebene Routen visuell hervorzuheben.

Bereitgestellte Routen Da die Indoor-Spieler nur aus vorgegebenen Routen wählen können, ist zu betrachten, wie die vorgegebenen Routen ausgewählt werden. Bei freier zufälliger Auswahl aus einer Menge von n Knoten sind somit $\frac{n*(n-1)}{2}$ Verbindungen möglich[1]. Da auf Verbindungen der Nachrichtenversand bidirektional erfolgen kann, ergibt sich also eine Anzahl von $n * (n-1)$ möglichen Routen. Bei einer in den Testläufen üblichen Größe von 6 Knoten ergibt dies 30 mögliche Routen.

[1] `http://dwb4.unl.edu/calculators/activities/middle/shake.html`

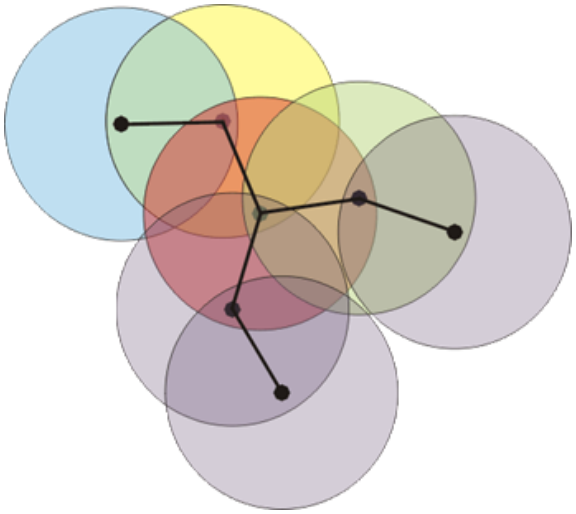

Abbildung 3: Einführungs-Folien (Ausschnitt)

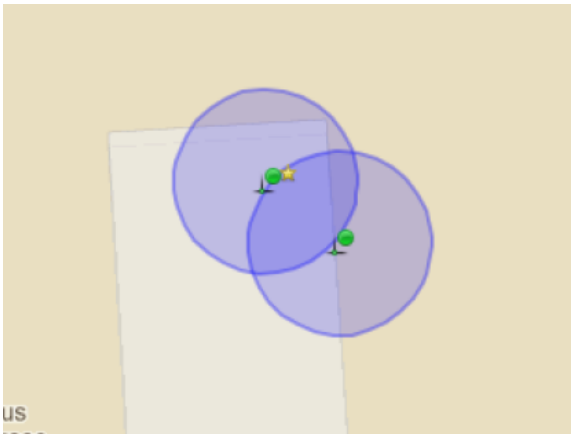

Abbildung 4: Indoor-Sicht (Ausschnitt)

Welche dieser Routen sollen also zur Verfügung gestellt werden? Um die Auswahl von der freien Wahl der Start- und Zielknoten zu variieren, darf nur eine eingeschränkten Untermenge der möglichen Routen verwendet werden. Zudem ist es möglich, lediglich Knoten zu verwenden, zwischen denen eine Verbindung besteht. Je nach Spielsituation könnte die Auswahl lange oder kurze, besonders anspruchsvolle oder stabile Routen präferieren. Die Auswahl von Routen ist eine gute Möglichkeit, das Spielgeschehen zu lenken.

3.2 Spielelemente

Da es sich bei *RouteMe* um ein pervasives Lern*spiel* handelt, verdient die Erweiterung von Spielelementen eine gesonderte Betrachtung. Es handelt sich hierbei nicht um eine detaillierte Aufstellung möglicher Spielmuster, die bereits aus einschlägigen Veröffentlichungen bekannt sind. [DPB04] Vielmehr ist es das Ziel, das Spielerlebnis zu stärken.

Ein Problem, das bei den verschiedenen Testrunden auffiel, war die mangelnde Visualisierung des Netzwerkes. Die Einführungsfolien zum Spiel zeigen Knoten, ihren jeweiligen Sendebereich und die logischen Kommunikationspfade, wie in Abbildung 3 dargestellt ist. Das Indoor-Interface zeigt ebenfalls Knoten und ihre jeweiligen Sendebereiche, bietet jedoch keine Darstellung der logischen Kommunikationspfade, wie in Abbildung 4 zu sehen ist. Damit ist eine zusätzliche kognitive Hürde geschaffen, deren Wert für das Verständnis des unterliegenden Routing-Algorithmus kritisch betrachtet werden muss.

Es bietet sich an, die Verbindungslinien im Spiel nach dem Vorbild der Einführungsfolien anzuzeigen.

3.3 Pervasivität

Das Ziel von Pervasivität ist das Verweben von realen und virtuellen Eindrücken, sodass der Spieler an einem bestimmten Punkt keinen Unterschied mehr zwischen realen und virtuellen Elementen macht. Durch Handlungen, die an der Grenze zwischen virtueller und realer Welt verlaufen, soll der Spieler stärker in das Spiel gezogen werden. So können beispielsweise Hebe-Gesten für das Einsammeln von Gegenständen oder Wurfgesten für die Übertragung von Datenpaketen verwendet werden.

Ein erster Ansatzpunkt sind die Ereignisse, die Knoten-Spieler auf den Smartphones durch Druck auf eine Schaltfläche auslösen. Aber auch haptisches Feedback und angepasste Anzeigen gehören dazu.

3.3.1 Pervasive Gesten

Im Ausgangszustand des Spieles gibt es nur eine Aktion, die Knoten-Spieler durchführen können: Das Einsammeln von Gegenständen. Mit der Arbeit an Level zwei (siehe Abschnitt 3.1) kommt die Aktion *HELLO-Nachricht versenden* hinzu.

Einsammeln von Gegenständen Das Einsammeln von Gegenständen lässt sich durch Aufhebe- oder Schwenk-Gesten realisieren. Hierbei wird das Smartphone im Raum bewegt. Die einfachste Bewegung ist ein schnelles Schütteln des Smartphones. Komplexe, mehrstufige Bewegungs-Gesten würden den Rahmen dieser Arbeit überschreiten und werden deshalb nicht anvisiert.

Um dem Spieler eine Rückmeldung zu geben, eignen sich auditive oder haptische Feedbacks. Schnelle Bewegungen erschweren die Verfolgung des Smartphones mit den Augen. Deshalb sind visuelle Feedbacks ungeeignet. Da eine Bewegungs-Geste dadurch ausgelöst wird, dass das Smartphone in der Hand gehalten und die Hand bewegt wird, erscheint haptisches Feedback besonders geeignet.

Mit der Implementierung einer Geste ist die bisher eingesetzte Schaltfläche nicht mehr notwendig. Ein Nebeneffekt der Abkehr von der Einsammeln-Schaltfläche ist die Platzeinsparung: Da die Schaltfläche nicht mehr angezeigt werden muss, können andere Elemente wie die Karte entsprechend größer dargestellt werden.

HELLO-Nachrichten versenden HELLO-Nachrichten breiten sich kreisförmig um einen Knoten herum aus. Dieses Verhalten entspricht einem Stein, der in einen Teich geworfen wird und dessen Wellen sich ausbreiten, bis sie auf ein Hinderniss treffen. Der Stein entspricht hierbei dem sendenden Knoten, der Teich ist das Überträgermedium, die Wellen stehen für die HELLO-Nachrichten und Hindernisse sind andere Knoten, die von den Wellen erreicht werden. Die Metapher ist dem Lagoon-Bildschirmschoner[2] des Microsoft PixelSense[3], einem MultiTouch-fähigen Tisch, entliehen.

Die Übertragung dieser Metapher auf die Anwendung erfolgt, indem die auf dem Smartphone dargestellte Karte, insbesondere der den Spieler repräsentierende Knoten im Mittelpunkt, auf Touch-Gesten reagiert. Ein Tipp auf die Karte löst den Versand einer HELLO-Nachricht aus. Dies wird durch eine Welle dargestellt, die sich ausgehend vom Standort des Spielers ausbreitet.

[2] `http://www.youtube.com/watch?v=aaDeWhqLalo`
[3] `http://www.microsoft.com/en-us/pixelsense/default.aspx`

3.3.2 Selbstkonfiguration

Die manuelle Konfiguration von Smartphones und Desktop-PCs ist zeitaufwendig und fehleranfällig. Je größer der Anteil an automatisierbaren Konfigurationsaufgaben ausfällt, desto leichter ist die Inbetriebnahme und Wartung des Systems. Außerdem ist festzuhalten, dass einmalig zentralisiert abgelegte Konfigurationen weniger aufwändig zu verteilen sind als dezentralisierte, auf jedes Systembestandteil einzeln zu übertragende Konfigurationen. Im Folgenden wird eine Kurzübersicht über Konfigurationsschwierigkeiten gegeben, die in den Testläufen aufgefallen sind. Anschließend wird gezeigt, wie diese Konfigurationsaufgaben automatisiert oder zentralisiert werden können.

Schwierigkeiten in der Vorbereitung In verschiedenen Testläufen mit dem existierenden Spiel sind wiederholt Schwierigkeiten aufgefallen, die das Spielerlebnis beeinträchtigt oder die Vorbereitungsphase verlängert haben.

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

- 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.
- WLAN eingerichtet: Jedes Smartphone musste mit dem eingesetzten Router verbunden sein, um mit dem Server kommunizieren zu können. Hier half der Einsatz von Wi-Fi Protected Setup (WPS).
- Browser installiert: Durch die Browserabhängigkeit des Spieles (siehe Abschnitt 3.4) mussten die Smartphones auf korrekte Softwareausstattung überprüft werden.
- AccuracyLogger installiert: Um Browser-Einschränkungen auszugleichen, musste während des Spieles eine native Android-App laufen. Mehr dazu in Abschnitt 3.4.
- URL-Lesezeichen eingerichtet: Die eingesetzte URL basierte auf einer IP-Adresse und case-sensitiven URL-Bestandteilen. Einigen Spielern mangelte es an Erfahrung in der Bedienung von 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 die Spieler bei einem Bedienungs- oder Softwarefehler die URL vollständig eingeben müssen, wurden Lesezeichen auf jedem Startbildschirm eingerichtet.

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

Automatisierung und Zentralisierung Die Einrichtung des WLAN lässt sich zentralisieren und automatisieren, indem die Verbindungsparameter maschinenlesbar kodiert und durch eine Applikation gelesen werden. Anschließend lässt sich eine WLAN-Verbindung durch eine Applikation aufbauen. Da die Parameterübergabe bei Android standardisiert ist, können das Lesen und das Verbinden durch unterschiedliche Applikationen erfolgen, falls dies erforderlich sein sollte. Android-Applikationen lassen sich über QR-Codes starten. Hierbei können auch Parameter übergeben werden.

Für die maschinenlesbare Kodierung der Startparameter gibt es verschiedene Möglichkeiten. Die Entscheidung fiel zu Gunsten von QR-Codes aus. QR-Codes sind fehlertolerant, weit verbreitet und können von vielen Applikationen gelesen werden. [KTC, Ash10] Mindestens ein QR-Code-Leser muss installiert sein.

Die Installation des Browsers wird durch die Installation der RouteMe-App ersetzt. Hier lassen sich praktisch keine Verbesserungen erzielen. Es ist denkbar, die Installations-URL als QR-Code zu veröffentlichen, sodass zumindest die Manuelle Suche in Google Play[4] verkürzt wird.

Durch Nutzung von dynamischem DNS können IP-basierte Lesezeichen auf Domain-basierte Lesezeichen umgestellt werden. Dabei wird im Vorfeld eine URL bei einem Dynamic-DNS-Anbieter reserviert. Die zu dieser URL gehörige IP wird auf eine vom Anbieter definierte Art gesetzt. Sollten Änderungen an der IP notwendig sein, so kann die IP neu gesetzt werden. Die Änderung wird innerhalb weniger Sekunden wirksam. Da das Spiel innerhalb der Universität gehostet wird, sind die gesetzten IPs nur innerhalb des internen Netzwerkes der Universität Potsdam gültig. Somit ist das Risiko eines Angriffes auf den hostenden Server gering, obwohl die IP global bekanntgemacht wird.

3.3.3 Interaktive Karte

Mentale Hürden erhöhen die Hemmschwelle, um in einen pervasiven Kontext einzutauchen. Deshalb ist es von besonderem Interesse, mentale Hürden zu identifizieren und abzubauen. So ist in den Testläufen immer wieder aufgefallen, dass es Personen gab, die Schwierigkeiten mit der Koordination von virtueller Karte und realer Laufrichtung hatten. Das führte dazu, dass diese Personen immer wieder stehenblieben, das Smartphone mit der Karte nach Norden ausrichteten und anschließend überlegten, in welche Richtung sie weiterlaufen mussten, um zu einem auf der Karte angezeigten Punkt zu gelangen.

Dieses Verhalten zeigt deutlich, dass die statische Karte eine mentale Hürde darstellt. Besser ist es, wenn sich die Karte nach der Position des Smartphones ausrichtet. Dabei ist die korrekte Ausrichtung der Himmelsrichtung nur ein Teilaspekt. Wenn auch die Neigung des Smartphones einbezogen wird, dann ist die mentale Hürde so stark reduziert, dass das Smartphone nicht mehr nur als Anzeigegerät für, sondern als Fenster in das Spiel wahrgenommen werden kann.

[4] `https://play.google.com/store`

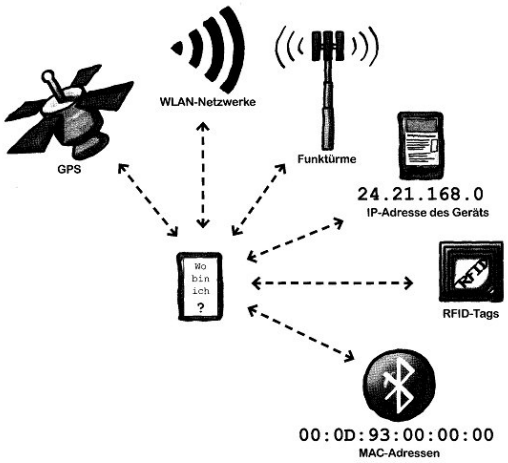

Abbildung 5: Mögliche Geolocation-Quellen [GGS12]

3.4 Zuverlässigkeit der Ausführungsumgebung

Für ein gutes Spielerlebnis zählt nicht nur die Pervasivität. Wenn die Spieler eingetaucht sind, muss verhindert werden, dass sie durch technische Probleme aus dem Spiel gerissen werden. Dieser Abschnitt befasst sich mit der Verbesserung der technischen Zuverlässigkeit von RouteMe.

Das bisherige Spiel ist auf die Ausführung im Browser ausgelegt. Das heißt, dass sowohl Indoor- als auch Knoten-Spieler das Spiel in einem Browser laufen lassen. Dabei hat sich gezeigt, dass die Smartphone-Sicht besonders fehler- und problemanfällig ist. Sowohl die Sensor-Zugriffe als auch die Seiten-Darstellung sind sehr Browser-spezifisch. Die Auswahl der in den gängigen Browsern implementierten HTML5-Features ist noch sehr unterschiedlich[5]. Hinzu kommt, dass nicht alle Browser auf allen Betriebssystemen verfügbar sind. So kann ist Google Chrome nur ab Android 4 installiert werden.

Im Folgenden werden die zwei größten Problembereiche vorgestellt. Zum einen wird auf die Sensoren mit Fokus auf GPS eingegangen. Zum anderen wird die Darstellung der Knoten-Sicht genauer beleuchtet.

GPS Geolocation-Daten können aus unterschiedlichen Quellen stammen. Eine Übersicht von Quellen stellt Abbildung 5 dar. Die häufigsten Geolocation-Quellen auf den getesteten Smartphones waren WLAN und GPS, wobei WLAN eine Genauigkeit von $20m$ bis $50m$ und GPS eine Genauigkeit von $5m$ bis $15m$ im vorgesehenen Spielbereich erreichte. Da der vorgesehene Spielbereich etwa $20m \times 50m$ groß ist, folgt, dass eine Genauigkeit von $5m$ und weniger notwendig ist, um ein gutes Spielerlebnis zu sichern.

Die getesteten Browser lieferten keine zuverlässigen Lokations-Daten. Um kontinuierlich zuverlässige Daten zu bekommen, wurde im Hintergrund eine Android-App gestartet, die direkt über das Betriebssystem auf die Lokations-Daten zugreift. Diese Daten werden an den Browser weitergegeben.

[5]Es gibt Seiten, die die unterschiedlichen Features der auf dem Markt verfügbaren Browser testen und gegenüberstellen. Eine solche Seite ist `http://html5test.com/`, von der die folgende Übersicht stammt: `http://html5test.com/compare/browser/android23/android40/operamobile1400.html`, `http://html5test.com/compare/browser/chromemobile25/ffmobile19/ios70.html`

Die Design-Philosophie für die Geolocation-Bestimmung ist bei W3C-kompatiblen Browsern und Android-Apps sehr ähnlich. Anstatt die zu verwendende Geolocation-Hardware anzugeben, werden stattdessen die erforderliche Genauigkeit sowie ein Aktualisierungsintervall angegeben. Der ausführende Container (also der Browser oder das Android-OS) wählt eine geeignete Datenquelle. Dabei kann der Container eigene Kriterien bei der Auswahl der Geolocation-Quelle einfließen lassen. So präferiert der Safari-Browser des iOS4 aus Energiegründen WLAN gegenüber GPS, obwohl letzteres eine höhere Genauigkeit liefern würde[6].

Auf den für diese Arbeit verfügbaren Smartphones benötigt der GPS-Sensor in den meisten Fällen zwischen 30 Sekunden und fünf Minuten, um eine initiale Geolocation zu erhalten. Anschließend wird die Geolocation einmal pro Sekunde aktualisiert. Bei der Nutzung von WLAN-Geolocations war keine initiale Verzögerung zu bemerken. Auch in diesem Fall erfolgt die Aktualisierung einmal pro Sekunde.

Wie die Android-App bereits zeigte, ist eine native App den getesteten Browser bei den Zugriffen auf Smartphone-Sensoren überlegen. Für das Spiel wird eine Genauigkeit benötigt, die Browser-basiert und ohne native Unterstützung nicht geliefert werden kann.

Darstellung Die Darstellung der Knoten-Sicht erfolgt auf Basis des Frameworks *Sencha Touch 1.1.0*[7]. Sencha Touch arbeitet auf HTML5-Basis und ist für WebKit-Browser entwickelt. Entgegen den Erwartungen ist die Darstellung der Oberfläche in verschiedenen Browsern sehr unterschiedlich.

Die Vorgängerarbeit hat sich bereits mit dem Testen des Spieles auf unterschiedlichen Browsern und Systemen beschäftigt. [DG13] Am intensivsten wurden Browser auf Android-4-Basis getestet. Durch die Beschränkung auf WebKit-Browser ist Firefox für Android nicht unterstützt[8]. Die Darstellung mit Firefox ist so fehlerhaft, dass die Nutzung nicht einmal eingeschränkt möglich ist. Der native Android-Browser hat Probleme mit der Kartendarstellung, da er Elemente doppelt und mit leichtem Versatz zeichnet. Chrome hat die besten Ergebnisse geliefert. Allerdings ist Chrome nur für Android 4 verfügbar.

Es lässt sich also zusammenfassen, dass aktuelle Browser als Ausführungsumgebung für *RouteMe* nicht geeignet sind. Der unzureichende Zugriff auf Sensoren sowie die unzuverlässige Darstellung legen den Schluss nahe, dass ein Wechsel der Ausführungsumgebung notwendig ist, um bessere Ergebnisse zu erzielen.

Aus dem Vorgängerprojekt gesammelte Erfahrungen mit Android zeigen, dass der Sensor-Zugriff auf Android-Basis zuverlässig und genau ist. [DG13] Neben der GPS-Genauigkeit bietet Android weitere Vorteile, die es also Ziel-Plattform für eine Portierung attraktiv machen. Zum einen hat Android einen sehr großen Marktanteil bei Smartphones und Tablets. Zum anderen bietet Android mit Java eine Programmiersprache, die im universitären Bereich gelehrt wird und stark verbreitet ist, womit die Langzeitunterstützung und Weiterentwicklung gesichert sind. Verbunden damit bietet eine Android eine breite und starke Community.

[6] `http://stackoverflow.com/questions/6790368/`
[7] `http://www.sencha.com/products/touch`
[8] `http://www.sencha.com/forum/showthread.php?120949-Sencha-Touch-WebKit-and-Firefox`

Eine Festlegung auf Android bringt aber auch verschiedene Einschränkungen mit sich. So geht beispielsweise die Plattformunabhängigkeit verloren, die mit der Browser-App vorgesehen war. Dies ist aber kein großer Verlust, da nicht einmal eine Browserunabhängigkeit erreicht wurde.

4 Umsetzung

Nachdem die Konzeption abschlossen ist, folgt nun eine Beschreibung der Konzeptumsetzung. Der Inhalt folgt dabei chronologisch dem Zeitpunkt der Umsetzung. Die Android-Portierung war die Basis aller Erweiterungen, um einen entstehenden Mehraufwand in der Implementierung zu verhindern.

4.1 Android-Portierung

Der erste Schritt besteht daraus, die im Knoten-Interface eingesetzten Methoden und Bibliotheken auf Android-Ersatz zu prüfen. Soweit möglich wurden verwendete Funktionalitäten mit Android-Mitteln abgedeckt. Falls nötig wurden Android-kompatible Bibliotheken gesucht.

Das Interface für Knoten-Spieler besteht aus

- einer HTML-Seite (MobileGWT.html)
- zwei Spiel-Skripten (functions_mobile.js, consts.js)
- zwei jQuery-Skripten (jquery-1.6.min.js, jquery-ui-1.8.10.custom.min.js)
- einem Maps-Skript (`http://maps.google.com/maps/api/js?sensor=true`)
- einem GWT-Skript (de.unipotsdam.nexplorer.MobileGWT.nocache.js)
- zwei Sencha-Touch-Skripten (sencha-touch.js, GmapTracker.js)

Das jQuery- und das Maps-Skript sind Drittanbieter-Bibliotheken.

Google Maps wird zur Darstellung von Karte und Karten-Symbolen verwendet. Maps ist als Bibliothek auf Android verfügbar, die API ist hier sehr ähnlich der Javascript-Version.

jQuery wird vom Spiel-Skript für unterschiedliche Aspekte genutzt. Einerseits wird die Bibliothek eingesetzt, um UI-Elemente zu manipulieren. Android hat hier eine andere Vorgehensweise, weshalb für UI-Manipulationen Android-Äquivalenzen verwendet wurden. Andererseits werden über jQuery AJAX-Anfragen abgesetzt. Hierfür bietet Android keinen gleichwertigen Ersatz, weshalb die Bibliothek *Spring for Android*[9] verwendet wurde. Die Kommunikation zwischen Client und Server erfolgt im JSON-Format. Da JSON eine Javascript-Objektnotation ist, kann es direkt aus Javascript heraus verwendet werden. Auf Android-Seite muss zwischen JSON-Objekten und Java-Objekten eine Umwandlung erfolgen. Hierfür wurde *Google Gson*[10] genutzt.

Außerdem bietet jQuery verschiedene Helfer-Funktionen an, von denen die Funktion *each* im Spiel-Skript verwendet wird. Diese Funktion lässt sich leicht mit Java-Mitteln nachbilden.

Das Spiel-Skript baut mit Hilfe von Sencha Touch die UI auf. Diese Funktionalität wird vollständig von Android-Bordmitteln bereitgestellt.

[9] `http://www.springsource.org/spring-android`
[10] `https://code.google.com/p/google-gson/`

4.1.1 Code-Portierung

Um den Code aus dem Hauptskript des Knoten-Interface für die Android-Plattform kompatibel zu machen, musste er von Javascript nach Java portiert werden. Im Vorgängerprojekt wurde bereits Erfahrung mit der Portierung von PHP-Servercode auf Java-Servercode gesammelt. [DG13] Javascript und PHP sind Skriptsprachen, Java ist eine statische Sprache. Javascript besitzt einige PHP-ähnliche Eigenschaften, weshalb bei der Portierung auf bereits bestehende Erfahrungswerte zurückgegriffen werden konnte.

So muss Code in Javascript und PHP - im Gegensatz zu Java - nicht in Strukturen wie Klassen existieren, sondern lediglich in einer Datei notiert sein. In PHP und Javascript sind keine Namespace-Konzepte erforderlich, für Java-Code werden Namespace-Deklarationen nachdrücklich empfohlen. In der Praxis findet sich keine Java-Bibliothek ohne Namespaces. Dem gegenüber sind fehlende Namespaces bei PHP und Javascript durchaus üblich.

Bei Java sind alle Funktionen mit einem Rückgabetyp und einem Funktionsnamen definiert und können um Optionen zur Sichtbarkeit erweitert werden. Javascript und PHP erfordern das Schlüsselwort *function* und einen Funktionsnamen, bei Javascript werden aber keine Sichtbarkeiten oder Rückgabetypen definiert. Die Sichtbarkeit einer Funktionen ergibt sich hier aus der Sichtbarkeit der Variable, in der die Funktion gehalten wird.

Zudem haben Javascript und PHP ein anderes Typkonzept als Java: Während in den Skriptsprachen keine Typen angegeben werden, sind Typdeklarationen in Java zwingend erforderlich.

Die von Javascript und die von Java bereitgestellten Funktionen unterscheiden sich erheblich. Abgesehen von den sprachlichen Unterschieden gibt es auch große Unterschiede in der Nutzung der einzelnen Frameworks.

Zusammenfassend müssen also bei der Portierung die folgenden Aspekte beachtet werden:

- Strukturanforderungen
- Syntaxunterschiede
- Typregeln
- Sprachlicher Umfang
- Frameworkumgang

Im Folgenden wird grob auf das Vorgehen bei der Übertragung des Codes eingegangen. Dabei wurden verschiedene Ideen aus Industrie-Standard-Werken als Orientierung verwendet. [Fea09, FB10]

Zuerst wurde eine Java-Klasse *FunctionsMobile* erstellt, die den gleichen Namen wie das Hauptskript trägt. Anschließend wurden die Funktionen korrigiert, indem aus *function* in Javascript *private void* in Java gemacht wurde. Dieser Schritt lies sich durch die Suchen-und-Ersetzen-Funktion von Eclipse automatisieren.

Um Javas Typenanforderungen gerecht zu werden, wurden alle Variablendeklarationen und das erste Auftreten einer Variable mit der Klasse *Object* versehen. Hierbei eignete sich *Object*, da es die Oberklasse aller Java-Objekte ist und somit den gemeinsamen Obertyp aller Java-Objekte darstellt. Wenn nötig wurden die Deklarationen später verfeinert.

Für Funktionen, die vom Browser oder von Frameworks bereitgestellt wurden, mussten Wrapperklassen und -funktionen eingeführt werden. Dies betraf vor allem Funktionen aus dem jQuery-Framework sowie UI-Elemente.

Anschließend konnten die vom Java-Compiler bemängelten Stellen Stück für Stück korrigiert werden. Hierbei wurden häufig Wrapperklassen und Code-Stubs erstellt, um Javascript-Objekte und deren Funktion nachzubilden.

Nachdem das Hauptskript vollständig syntaktisch portiert wurde, konnten die während des Portierungsprozesses erstellten Wrapper und Stubs implementiert werden. Hier wurden Adapter für die ausgetauschten Bibliotheken eingesetzt.

Um zu verhindern, dass bei der Portierung syntaktische Fehler in den bei der Kommunikation mit dem Server übertragenen Daten entstehen, wurden alle für die Android-App relevanten Server-Daten in einer Testschnittstelle mit vordefinierten Daten präpariert und deren Empfang auf Android durch Tests überprüft.

4.1.2 UI-Portierung

Die UI wird auf Android XML-basiert definiert, während im Javascript-Code das Sencha-Touch-Framework verwendet sind. Das aus dem Javascript-Code generierte HTML wurde mit Google Chrome untersucht und anschließend konzeptionell verwendet, um die UI mit Android nachzubilden. Das Ergebnis der Portierung ist in den Abbildung 6 und 7 dargestellt.

Eine Schwierigkeit in der Portierung ergab sich aus den unterschiedlichen UI-Architekturen der Plattformen. In Javascript ist das Konzept des UI-Thread unbekannt, UI-Elemente können somit beliebig verändert werden. Android hat einen UI-Thread, auf dem alle Manipulationen von UI-Elementen durchgeführt werden müssen.

Deshalb stellt Android verschiedene Funktionen zur Verfügung, um die Ausführung von Code auf dem UI-Thread zu erzwingen. Es ist allerdings nicht möglich, den gesamten Code auf dem UI-Thread auszuführen, da langanhaltende Operationen die Reaktionsfähigkeit der Oberfläche verhindern könnten[11,12]. So ist es nicht erlaubt, einen Netzwerkzugriff auf dem UI-Thread durchzuführen. Um die Manipulation der UI-Elemente zu ermöglichen, wurden die UI-Wrapperklassen angepasst. Sie ermöglichten damit transparenten Zugriff auf die UI-Elemente und abstrahierten das UI-Thread-Konzept, sodass der portierte Code keiner weiteren Anpassung bedurfte.

[11] `http://developer.android.com/guide/components/processes-and-threads.html`
[12] `http://developer.android.com/training/articles/perf-anr.html`

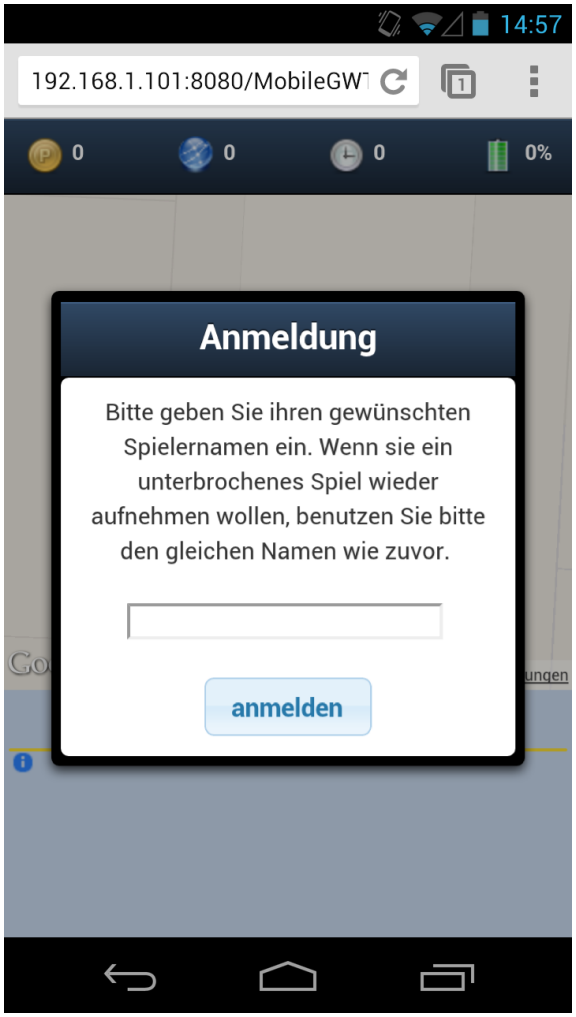

Abbildung 6: Login in der Browser-Version

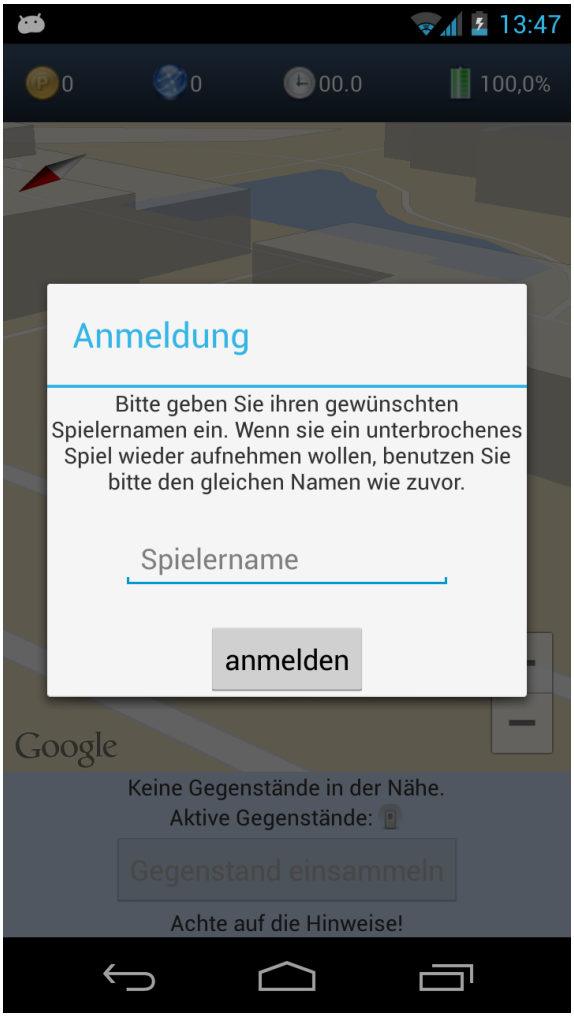

Abbildung 7: Login in der Android-Version

4.2 Selbstkonfiguration

4.2.1 DNS

Für die Umsetzung von Dynamic DNS wurde basierend auf zwei obligatorischen und einem optionalen Kriterium ein Anbieter gesucht. Zum einen musste die Nutzung des Dienstes kostenlos sein. Zum anderen sollte die Oberfläche zum Einstellen der IP dynamisch sein. Optional war, dass der generierte Domain-Name gut lesbar ist. Ein Anbieter, der alle drei Kriterien erfüllt, ist DNSdynamic.org[13]. Die reservierte URL lautet `routeme.dnsdynamic.com`.

[13] `https://www.dnsdynamic.org/`

Abbildung 8: QR-Code zu `routeme://android?ssid=WLANSSID&password=WLANPASSWORD`

4.2.2 WiFi-Modul

Android-Apps können WLAN-Verbindungen suchen und aufbauen. Im Rahmen dieser Arbeit entstand ein Modul, das zu einem Tupel aus SSID und Passwort eine Verbindung aufbaut. Um die Wiederverwendbarkeit in anderen Projekten zu fördern wurde dabei auf eine hohe Entkopplung geachtet.

4.2.3 QR-Code

Da für QR-Codes bereits viele Lese-Apps existieren, wurde auf die Entwicklung einer eigenen Lese-Komponente verzichtet. Stattdessen wurde die Android-App so aufgebaut, dass sie durch den Aufruf einer URL gestartet wird. Nach der Installation ist die App auf die URL `routeme://android?ssid=WLANSSID&password=WLANPASSWORD` registriert. `WLANSSID` muss hierbei durch die SSID des WLAN ersetzt werden, `WLANPASSWORD` durch das WLAN-Passwort. Der der URL zugehörige QR-Code ist in Abbildung 8 dargestellt.

Die Erstellung kompatibler QR-Codes ist nicht in das Spiel integriert. Stattdessen wird an dieser Stelle auf geeignete Dienste wie QRCode-Generator.de[14] verwiesen, mit dem auch der abgebildete QR-Code erstellt wurde.

4.3 Pervasive Gesten

Es existiert bereits eine Bachelorarbeit, in der ein Framework für eine allgemeine Gestenerkennung auf Smartphones implementiert und evaluiert wurde. Allerdings bietet das Ergebnis nur eine unbefriedigende Erkennungsrate von etwa 60% und viele *false positives*. [Pfe11]

[14] `http://www.qrcode-generator.de/`

Deshalb wurde ein Modul entwickelt, das Schüttel-Bewegungen erkennt. Das Modul bedient sich dazu des Beschleunigungs-Sensors, das die Beschleunigung in $\frac{m}{s^2}$ angibt. Um fälschlich erkannte Schüttelbewegungen zu verhindern, wird das Modul mit der Basisbeschleunigung der Erde ($9.80665\frac{m}{s^2}$) initialisiert. Da die Beschleunigungsbewegung auch bei bereits erkannter Schüttelbewegung nicht plötzlich abreißt, ist ein Timeout eingerichtet, der verhindert, dass innerhalb einer bestimmten Zeitspanne mehr als eine Schüttelbewegung erkannt wird.

Die im Spiel verwendeten Werte sind

- Beschleunigung: $1\frac{m}{s^2}$
- Timeout: 750ms

Die Erkennung der Schüttel-Geste wird dem Spieler durch haptisches Feedback in Form einer kurzen Vibration vermittelt.

4.4 Interaktive Karte

Das in der Android-App eingesetzt Karten-Fragment hat Methoden, um die Geo-Position, einen Dreh- und einen Neigungswinkel einzustellen. Der Drehwinkel liegt hierbei zwischen 0° und 360°, wobei 0° eine Ausrichtung nach Norden bedeutet und der Winkel im Uhrzeigersinn vergrößert wird. Der Neigungswinkel liegt zwischen 0° und 67.5°, wobei 0° für Vogelperspektive steht.

Android abstrahiert die für diese Werte notwendigen Sensoren. Das Ergebnis ist eine gesammelte Übersicht von Sensordaten, die unter anderem den Dreh- und Neigungswinkel des Smartphones angeben und ohne erweiterte Umformung für die Manipulation des Karten-Fragmentes geeignet sind.

Auf Basis dieser Sensoren wurde ein Modul entwickelt, das die Karten-Darstellung mit der Ausrichtung und der Position des Smartphones synchronisiert. Um die Wiederverwendbarkeit in anderen Projekten zu fördern wurde bei der Entwicklung des Modules auf eine hohe Entkopplung geachtet. Eine Darstellung der Implementierung ist in Abbildung 9 zu sehen.

4.5 Netzwerk-Visualisierung

Die Javascript-Google-Maps-API bietet Pfeile als Kartenelemente an. In den vom Server übertragenen Daten waren die Nachbarbeziehungen bereits enthalten. Somit musste nur die Anzeige um Nachbarverbindungen erweitert werden.

Abbildung 9: Implementierung der Interaktiven Karte

Wahl der Programmiersprache Die Indoor-Sicht bestand aus einer Mischung von Java-Code und Javascript-Code. In der Vorgängerarbeit wurde empfohlen, neuen Code in Java zu schreiben, um so immer weiter auf eine reine Java-Codebasis zuzusteuern. [DG13] Allerdings kann die Maps-API nur aus Javascript heraus angesprochen werden. Zudem geschieht der Empfang der Nachbarsverbindungen im in Javascript geschriebenen Teil des Codes. Bei einer auf Java basierenden Lösung wäre es also notwendig, die empfangenen Daten aus ihrer Javascript-Repräsentation nach Java zu transformieren, zu analysieren und anschließend über Java-nach-Javascript-Wrapper die Analyseergebnisse zur Kartenmanipulation zu nutzen.

Dieses Vorgehen würde sich lohnen, wenn die Analyse der Daten ausreichend komplex wäre, um eine doppelte Konversion zu rechtfertigen. Da die Analyse aber lediglich in einem Vergleich von bereits bekannten und unbekannten Nachbarverbindungen besteht, erscheint der Weg über Java unnötig komplex. Daher field die Entscheidung für eine Javascript-basierte Erweiterung.

Sicherung der Qualität Bevor eine Veränderung von bestehendem Code durchgeführt wird, sollten alle beteiligten Komponenten durch Tests abgesichert sein. [Fea09] Im Gegensatz zu in Java geschriebenem Code war für den Javascript-basierten Teil des Projektes noch kein automatisierter Qualitätssicherungs-Mechanismus etabliert. Bevor die Arbeiten am Code begannen, wurden deshalb der Javascript-Testrunner Karma (vormals Testacular)[15] und das Javascript-Test-Framework Jasmine[16] eingerichtet.

Karma überwacht das Dateisystem und lässt alle im Projekt verfügbaren Tests durchlaufen, sobald eine Datei geändert wird. Jasmine wird zur Definition der zu durchlaufenden Tests verwendet. Diese Kombination vereinfacht testgetriebenes Vorgehen massiv. Durch das testgetriebene Vorgehen können Erweiterungen schnell und sicher implementiert werden. [Fea09]

4.6 AODV Level 2

4.6.1 Nachrichtenversand

Technische Anpassungen Aus programmatischer Sicht handelt es sich beim veränderten Nachrichtenversand von Level zwei um den Nachrichtenversand von Level eins mit ein paar Veränderungen. So werden Start und Ziel nicht mehr vom Spieler über die Karte, sondern vom Spieler aus einer Liste ausgewählt. Außerdem werden in der Knotenauswahl nicht mehr der gewählte Start- und Zielknoten angezeigt, sondern eine Liste von Paaren von Start- und Zielknoten, aus denen gewählt werden kann.

Das Ziel war es also, möglichst viel bereits vorhandenen Code zu behalten, um den Aufwand einer Neuimplementierung und die Wahrscheinlichkeit für Fehler gering zu halten. In der Kommunikation mit dem Server musste nichts geändert werden, alle erforderlichen Änderungen konnten vollständig auf Clientseite durchgeführt werden. Deshalb bot sich ein Mix aus Code in Java und Code in Javascript an. Die kartennahen Bestandteile wurden weiter durch Javascript-basierten Code manipuliert, die Legende durch Java-basierten. Der zu schreibende Code wurde nahezu vollständig in Java gehalten, lediglich die für die Javascript-zu-Java erforderlichen Wrapper entstanden in Javascript.

[15] `http://karma-runner.github.io/`
[16] `http://pivotal.github.io/jasmine/`

Spielerische Entscheidungen Es hat sich als günstig erwiesen, dem Spieler etwa zehn Routen gleichzeitig zur Auswahl zu stellen. Die angezeigten Routen werden dabei zufällig aus dem Pool von möglichen Routen ausgewählt. Wenn eine angezeigte Route nicht mehr verfügbar ist, beispielsweise weil einer der beteiligten Knoten ausgefallen ist, wird die ausgefallene Route, soweit möglich, durch eine andere mögliche Route ersetzt.

4.6.2 HELLO-Nachrichten

Technische Anpassungen Um das Konzept der HELLO-Nachrichten umzusetzen, mussten einige technische Veränderungen an Server und Knoten-Interface durchgeführt werden. Die Indoor- und Admin-Sicht blieben dabei unverändert.

Um einem Knoten A zu kommunizieren, dass sein Nachbar B eine HELLO-Nachricht abgesetzt hat, die Knoten A empfängt, musste die Kommunikation zwischen Server und Client verändert werden. Die Darstellung sollte dabei so erfolgen, dass bei Empfang einer HELLO-Nachricht der sendende Knoten auf der Karte des empfangenden Knotens mit Wellensymbolen markiert wird. Die Dauer einer Hervorhebung ist für alle Knoten gleich. Im Folgenden werden die architekturellen Möglichkeiten für eine solche Kommunikation vorgestellt und anschließend gegeneinander abgewogen.

Als erste Option war denkbar, dass der sendende Knoten ein Ping-Flag setzt, dass er nach einiger Zeit selbst entfernt. Der empfangende Knoten würde dann bei jeder Statusabfrage an den Server überprüfen, ob das Ping-Flag gesetzt ist. Falls es gesetzt ist, so würde der sendende Knoten mit Wellen markiert.

Die zweite Option war ein mit dem Server synchronisierter Zeitstempel, der aussagt, wann ein sendender Knoten nicht mehr als sendend markiert wäre. Der empfangende Knoten würde dann den empfangenen Zeitstempel mit der eigenen Uhr vergleichen, um zu ermitteln, ob ein Knoten noch als sendend gilt.

Zuletzt war die dritte Option, dass die Zeitstempel aus Option zwei durch ein Feld für verbleibende Dauer ersetzt werden. Dem empfangenden Knoten würde also bei einer Serveranfrage mitgeteilt, wie lange ein sendender Knoten noch als sendend gilt.

Von den vorgestellten Optionen eignete sich die dritte für eine Implementierung. Die erste Option hätte zwei Kommunikationen zwischen sendendem Knoten und Server benötigt, die zweite wäre mit den hohen Hürden der Zeitstempelsynchronisation unnötig komplex geworden. Mit der dritten Option kann der Zeitstempel innerhalb des Servers bleiben, sodass keine Synchronisation mit den Knoten erforderlich ist.

Neben der Anpassung der Android-App wurden auch die künstlichen Mitspieler angepasst, sodass sie in regelmäßigen Abständen HELLO-Nachrichten aussenden.

Spielerische Entscheidungen Für das Senden einer HELLO-Nachricht hat sich ein konstanter Malus von fünf Batterie-Punkten als günstig erwiesen. Die Punkte für Nachbarverbindungen bleiben gleich, werden allerdings nur vergeben, wenn ein Nachbar durch den Empfang einer HELLO-Nachricht ermittelt wurde.

Unter kompetitiven Gesichtspunkten mag erst im ersten Moment günstig erscheinen, selbst keine HELLO-Nachrichten zu senden, da der Empfang einer gesendeten HELLO-Nachricht dem Empfänger Punkte für gefundene Nachbarverbindungen gibt. Allerdings Erlangen Knoten, die keine HELLO-Nachrichten aussenden, auch keine Pakete, mit deren Routing sie sich Punkte verdienen.

5 Evaluation

Für RouteMe existiert bereits ein Evaluationskonzept, das sehr umfangreich ist. [Jul12] Aus Zeitgründen war eine Evaluation des Lernerfolges allerdings nicht möglich. Deshalb folgt eine funktionale Evaluation, in die einzelnen Bestandteile geprüft werden.

5.1 Android-Portierung

Da es sich bei der Android-App um eine portierte Version der Browser-App handelt, soll zunächst sichergestellt sein, dass alle Funktionen aus der Browser-App auch in der Android-App funktionieren. Die zu testenden Funktionen sind hierbei der Login, das Übermitteln der aktuellen Position, das Einsammeln von Gegenständen und das Sehen von Nachbarn.

Login (ok)

1. (ok) Setzen der Server-IP-Adresse
2. (ok) Starten des Servers
3. (ok) Erstellen eines Level-eins-Spieles
4. (ok) Starten der Android-App
5. (ok) Starten der KI
6. (ok) Login auf Android-App

Positionsübermittlung (ok)

1. (ok) Setzen der Server-IP-Adresse
2. (ok) Starten des Servers
3. (ok) Erstellen eines Level-eins-Spieles
4. (ok) Starten der Android-App
5. (ok) Starten der KI
6. (ok) Login auf Android-App
7. (ok) Starten des Spieles
8. (ok) Automatische Übertragung der Position

Aufsammeln eines Gegenstandes (ok)

1. (ok) Setzen der Server-IP-Adresse
2. (ok) Starten des Servers
3. (ok) Erstellen eines Level-eins-Spieles
4. (ok) Starten der Android-App
5. (ok) Starten der KI
6. (ok) Login auf Android-App
7. (ok) Starten des Spieles
8. (ok) Aufsammeln eines Gegenstandes

Nachbarsichtung (ok)

1. (ok) Setzen der Server-IP-Adresse
2. (ok) Starten des Servers
3. (ok) Erstellen eines Level-eins-Spieles
4. (ok) Starten der Android-App
5. (ok) Starten der KI
6. (ok) Login auf Android-App
7. (ok) Starten des Spieles
8. (ok) Automatisches Sichten eines Nachbarn

5.2 Selbstkonfiguration

Die Selbstkonfiguration soll den Spielaufbau erleichtern. Zu testende Funktionen sind das Setzen der Server-IP-Adresse, das Starten der Android-App durch Lesen eines QR-Codes und die automatische Verbindung mit einem WLAN.

Setzen der Server-IP-Adresse (ok)

1. (ok) Setzen der Server-IP-Adresse
2. (ok) Überprüfen der IP mittels Ping

Lesen eines QR-Codes (fehlgeschlagen)

1. (ok) Erstellen eines QR-Codes
2. (ok) Starten der QR-Reader-App
3. (ok) Lesen des QR-Codes
4. (fehlgeschlagen) Automatisches Starten der Android-App

Die QR-Reader-App zeigt die Fehlermeldung „Die angeforderte Aktion kann nicht ausgeführt werden, weil sie eine Internetverbindung erfordert“ an. Der Grund für den Fehler ist eine fehlende Netzwerkverbindung. In den folgenden Testfällen wird das Smartphone mit bestehender GPRS-Verbindung verwendet.

WLAN-Verbindung (eingeschränkt)

1. (ok) Erstellen eines QR-Codes
2. (ok) Starten der QR-Reader-App
3. (ok) Lesen des QR-Codes
4. (eingeschränkt) Automatisches Starten der Android-App
5. (ok) Automatisches Verbinden mit WLAN

5.3 Pervasive Gesten

Zu den pervasiven Gesten gehören das Schütteln des Smartphones zum Aufsammeln von Gegenständen sowie das Berühren der Karte zum Aussenden einer HELLO-Nachricht, was durch eine Welle visualisiert wird. Dabei ist zu beachten, dass der HELLO-Versand nur in Level-zwei-Spielen möglich ist.

Schüttel-Erkennung (ok) Diese Funktion wurde bereits in Abschnitt 5.1 getestet.

HELLO-Versand (ok)

1. (ok) Setzen der Server-IP-Adresse
2. (ok) Starten des Servers
3. (ok) Erstellen eines Level-zwei-Spieles
4. (ok) Starten der Android-App
5. (ok) Starten der KI
6. (ok) Login auf Android-App

7. (ok) Starten des Spieles
8. (ok) Tippen auf Karte
9. (ok) Automatisches Ausbreiten der Welle

5.4 Interaktive Karte

Die interaktive Karte soll sich entsprechend der Lage und der Bewegung des Smartphones anpassen. Dabei ist zu testen, ob die Drehung, die Neigung und Positionsaktualisierung funktionieren.

Drehende Karte (ok)

1. (ok) Starten der Android-App
2. (ok) Drehen des Smartphones
3. (ok) Automatisches Drehen der Karte

Neigende Karte (ok)

1. (ok) Starten der Android-App
2. (ok) Neigen des Smartphones
3. (ok) Automatisches Neigen der Karte

Positionsaktualisierende Karte (ok)

1. (ok) Starten der Android-App
2. (ok) Automatisches Aktualisieren der Position auf der Karte

5.5 Netzwerk-Visualisierung

Die Verbindungen zwischen Knoten sollen in der Indoor-Sicht sichtbar sein. Dazu müssen Verbindungslinien angezeigt werden.

Anzeige von Verbindungslinien (ok)

1. (ok) Setzen der Server-IP-Adresse
2. (ok) Starten des Servers
3. (ok) Erstellen eines Level-eins-Spieles
4. (ok) Starten der Indoor-Sicht

5. (ok) Starten der KI
6. (ok) Login auf Indoor-Sicht
7. (ok) Starten des Spieles
8. (ok) Automatisches Anzeigen von Verbindungslinien

5.6 AODV Level 2

Die Implementierung eines zweiten Level machen sich sowohl in der Indoor- als auch in der Knoten-Sicht bemerkbar. Deshalb müssen die spezifischen Eigenschaften beider Sichten getrennt getestet werden. In der Indoor-Sicht ist der Versand von Nachrichten zu testen. In der Knoten-Sicht sind der Versand und Empfang von HELLO-Nachrichten zu prüfen.

Versand von Nachrichten (ok)

1. (ok) Setzen der Server-IP-Adresse
2. (ok) Starten des Servers
3. (ok) Erstellen eines Level-zwei-Spieles
4. (ok) Starten der Indoor-Sicht
5. (ok) Starten der KI
6. (ok) Login auf Indoor-Sicht
7. (ok) Starten des Spieles
8. (ok) Auswählen einer Route
9. (ok) Automatisches Transportieren der Nachricht

Versand von HELLO-Nachrichten (ok)

1. (ok) Setzen der Server-IP-Adresse
2. (ok) Starten des Servers
3. (ok) Erstellen eines Level-zwei-Spieles
4. (ok) Starten der Android-App
5. (ok) Starten der KI
6. (ok) Login auf Android-App

7. (ok) Starten des Spieles
8. (ok) Tippen auf Karte
9. (ok) Automatisches Reduzieren des Akku-Standes

Empfang von HELLO-Nachrichten (ok)

1. (ok) Setzen der Server-IP-Adresse
2. (ok) Starten des Servers
3. (ok) Erstellen eines Level-zwei-Spieles
4. (ok) Starten der Android-App
5. (ok) Starten der KI
6. (ok) Login auf Android-APP
7. (ok) Starten des Spieles
8. (ok) Automatisches Empfangen von HELLO-Nachrichten

6 Zusammenfassung und Ausblick

Diese Arbeit hat sich mit der Weiterentwicklung unterschiedlicher Aspekte von *RouteMe* befasst. Es ging sowohl um Lerninhalte, als auch um Spielelemente, Pervasivität und Zuverlässigkeit. Obwohl die Konzeption nahezu vollständig umgesetzt werden konnten, ergeben sich noch viele Anknüpfungspunkte.

So fehlen ein drittes Level für AODV sowie die Implementierung weiterer Routing-Algorithmen. Hier bieten sich insbesondere solche Algorithmen an, die dem vorhandenen reaktiven kontrastreich gegenübergestellt werden können. [Int03b, Int13b]

Außerdem war die Umsetzung nicht vollständig erfolgreich. Das Lesen von QR-Tags ist nur eingeschränkt brauchbar. Hier verspricht eine andere Herangehensweise mit in die App integrierter Erkennung, z.B. über das Framework ZXing[17], mehr Erfolg.

Die Erstellung von QR-Codes muss derzeit über Drittplattformen durchgeführt werden. Für Java existieren bereits verschiedene Bibliotheken zur Generierung, durch Integration in die Admin-Sicht lassen sich hier weitere manuelle Schritte einsparen. So könnten die erforderlichen Zugangsdaten bereits bei der Erstellung des Spieles abgefragt werden. Anschließend würden dem Spielleiter ein Link zum generierten QR-Code sowie die Möglichkeit, den Code in einem PDF auszudrucken, angeboten.

Die Methode, mit der derzeit die auswählbaren Routen in Level zwei ermittelt werden, basiert lediglich auf Zufall. Hier besteht für den Server eine sehr gute Möglichkeit, in das Spielgeschehen einzugreifen. So könnten mit steigender Punktzahl immer anspruchsvollere Routen ausgewählt werden. Ebenso könnten schwächere Spieler durch leichtere Routen unterstützt werden.

Es bietet sich an, die gesamte Kommunikation zwischen dem Server und seinen Clients nach REST zu portieren. An der Portierung der App war bereits zu erkennen, wie sehr eine gute Schnittstelle zur Plattformunabhängigkeit beiträgt.

Die künstlichen Spieler sollten ausgebaut und um Intelligenzen erweitert werden. Denkbar ist ebenfalls eine Manipulation von Objekten der realen Welt durch die künstlichen Spieler. So könnten künstliche Intelligenzen Spielzeug-Drohnen steuern, um mangelnde Knoten-Spieler-Beteiligung auszugleichen. Künstliche Spieler für die Indoor-Sicht sind ebenfalls denkbar.

Die spielerischen Aspekte sollten weiter gestärkt werden. So existiert bisher kein befriedigender Umgang mit Spielern, deren virtuelle Batterie vorzeitig entladen ist. Bisher können diese Spieler wiedereinsteigen, indem sie sich unter neuem Namen anmelden. In Computerspielen gibt es häufig ein Respawn-Modell, das Spielern einen geordneten Wiedereinstieg in das Spiel ermöglicht. AODV definiert sogar ein Reboot-Verfahren, über das zwischenzeitlich ausgefallene Knoten wieder in das Netzwerk integriert werden. [Int03a]

Die Fähigkeiten der Smartphones sollten stärker genutzt werden. So gibt es bisher keinerlei Audio-Nutzung. Denkbar ist ein akustisches Feedback beim Versenden und Empfangen von HELLO-Nachrichten: Beim Empfang einer HELLO-Nachricht ertönt auf dem Smartphone ein Sonar-ähnlicher Ping. Das würde nicht nur die Aufmerksamkeit des Empfängers erregen, sondern auch dem Sender, der sich nahe dem Empfänger befinden muss, die Gewissheit geben, dass sein Handeln unmittelbare Auswirkungen auf seine Umwelt hat.

[17] `https://code.google.com/p/zxing/`

Die Fähigkeiten der Karte könnten erweitert werden. So würde das Suchen nach Gegenständen vielleicht weniger frustrierend sein, wenn am Rand des Spielbereiches angezeigt würde, in welche Richtung noch etwas zu finden ist.

Die App sollte einer breiteren Öffentlichkeit zugänglich gemacht werden. Derzeit ist die App nur für Android 4 und höher ausgelegt. Mit vergleichsweise geringem Aufwand wären hier mehr Smartphones unterstützbar, sodass Studenten ihre eigenen Geräte mitbringen könnten und nicht auf fremde Smartphones angewiesen wären.

Die Knoten- und Indoor-Sicht können durch den Einsatz von Animationen ihren Spielcharakter stärker betonen. Visuelles Feedback wirkt belohnend und gibt Anreize auszuprobieren.

Zusammenfassend ist zu sagen, dass das Spiel RouteMe mit dieser Arbeit ein gutes Stück vorangekommen ist. Es bleiben aber noch viele Verbesserungsmöglichkeiten und offene Fragen, die in zukünftigen Projekten realisiert werden sollten.

7 Literaturverzeichnis

[Ash10] Robin Ashford. QR codes and academic libraries: Reaching mobile users. *College & Research Libraries News*, 71(10):526–530, 2010.

[DG13] Julian Dehne und Hendrik Geßner. RouteMe - Dokumentation des Refactoring, 26. April 2013.

[DPB04] Ola Davidsson, Johan Peitz und Staffan Björk. Game design patterns for mobile games. *Project report to Nokia Research Center, Finland*, 2004.

[FB10] Martin Fowler und Kent Beck. *Refactoring: Improving the design of existing code.* The Addison-Wesley object technology series. Addison-Wesley, Boston, 24. Auflage, 2010.

[Fea09] Michael C. Feathers. *Working effectively with legacy code.* Robert C. Martin series. Prentice Hall Professional Technical Reference, Upper Saddle River and NJ, 10. Auflage, 2009.

[GGS12] Lyza Danger Gardner, Jason Grigsby und Lars Schulten. *Mobiles Web von Kopf bis Fuß: Mobile Websites & Web-Apps entwickeln.* O'Reilly, Beijing, 1. Auflage, 2012.

[Int03a] Internet Engineering Task Force. Ad hoc On-Demand Distance Vector (AODV) Routing, 07.2003.

[Int03b] Internet Engineering Task Force. Optimized Link State Routing Protocol (OLSR), 10.2003.

[Int13a] Internet Engineering Task Force. Dynamic MANET On-demand (AODVv2) Routing, 12.03.2013.

[Int13b] Internet Engineering Task Force. The Optimized Link State Routing Protocol version 2, 23.03.2013.

[Jul12] Julian Dehne. Evaluation pervasiver Lernspiele, 2012.

[KTC] Tai-Wei Kan, Chin-Hung Teng und Wen-Shou Chou. Applying QR code in augmented reality applications. In *ACM 2009 – Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry*, Seiten 253–257.

[MLZ11] Tobias Moebert, Ulrike Lucke und Raphael Zender. A Pervasive Educational Game on Pervasive Computer Network. In *Int. Conf. on E-Learning; Honolulu*, 2011.

[Pfe11] Linda Pfeiffer. Untersuchung und exemplarische Evaluierung der Möglichkeiten zur Gestenerkennung durch mobile Nutzerendgeräte, 30.08.2011.

[SP12] Sanjay Dr. Sharma und Pushpinder Singh Patheja. Improving AODV Routing Protocol with Priority and Power Efficiency in Mobile Ad hoc WiMAX Network. *International Journal of Computer Technology and Electronics Engineering (IJCTEE)*, (2):87–93, 2012.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American (1991), September, S. 66-75*, Seiten 66–75, 1991.

[ZML12] Raphael Zender, Tobias Moebert und Ulrike Lucke. RouteMe - Routing in Ad-hoc-Netzen als pervasives Lernspiel. In Jörg Desel, Jörg M. Haake und Christian Spannagel, Hrsg., *DeLFI 2012 – Die 10. e-Learning Fachtagung Informatik*, Seiten 201–212, Hagen und Heidelberg, 2012. Gesellschaft für Informatik.

8 Abkürzungsverzeichnis

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

9 Erklärung der Redlichkeit

Hiermit versichere ich, dass ich die Hausarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Unterschrift der Verfasserin / des Verfassers