UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Ortsbezogenes Multiplayer Onlinespiel für mobile Endgeräte

Studienarbeit

im Studiengang Informatik

vorgelegt von

Rufus Linke 202120829
Martin Pätzold 202110191

Betreuer: Prof. Dr. Felix Hampe
Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich Informatik,

Dipl.-Inform. Stefan Stein
Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich Informatik

Koblenz, im August 2007

Erklärung

Wir versichern, dass wir die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek sind wir einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimmen wir zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

.....
(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Kapitel 1

Überblick

Die Verbindung zwischen mobilen Endgeräten und Positionsinformationen bietet interessante Ansätze zur Anwendung von Geoinformationssystemen. Sie ermöglicht es ortsbezogene Daten bereitzustellen und somit einen Mehrwert für den mobilen Anwender zu schaffen. Die Entwicklung einer Applikation basierend auf einer bestehenden Infrastruktur aus Geoinformationsdiensten ist Ziel der vorliegenden Arbeit. Diese Infrastruktur stellt positionsbezogene Raster- und Vektordaten über Webservices bzw. räumliche Datenbanken zur Verfügung. Um stabilen Zugriff auf diese Daten zu gewährleisten, erfordert es Probleme im Bereich der mobilen Netzwerkverbindungen zu behandeln. Dazu gehören niedrige Bandbreiten, hohe Latenzen und Verbindungsausfälle. Der entstandene Prototyp implementiert Lösungsansätze, um dem entgegenzuwirken.
Eine interessante Möglichkeit diese Ansätze zu demonstrieren besteht in der Entwicklung eines mobilen Multiplayerspiels, da hier weitere Aspekte der Nutzerkommunikation und -interaktion hinzukommen. Weiterhin wird durch Spiele eine große Nutzerzahl angesprochen. Bei dem entwickelten Prototyp handelt es sich um eine Aufbausimulation mit Rollenspielelementen. Die Spielwelt wird durch orthographische Luftbilder des Spielerstandortes dargestellt, die vom erwähnten Geoinformationssystem bezogen werden.

1.1 Motivation

Die Integration von GPS-Empfängern in massenmarkttaugliche Endgeräte führt zu einer Vielfalt neuer Anwendungsmöglichkeiten. Gerade in der Spieleentwick-

lung eröffnen sich durch diese Erweiterung um ortsbezogene Informationen bisher wenig genutzte Ansätze, da die übliche statische Spielwelt um die reale Umgebung ergänzt oder durch sie ersetzt wird. Eine damit einhergehende Steigerung des Spielerlebnisses bietet einen Vorteil gegenüber gängigen Spielen und somit Marktchancen.
Die prototypische Umsetzung soll weiterhin dazu dienen, die technischen Grundlagen für weitere Anwendungen zu schaffen. Beispiele hierfür sind Informationssysteme auf Veranstaltungen oder Navigationshilfen. Die zugrundeliegende, auf ein Geoinformationssystem aufbauende, Infrastruktur dient auch in diesen Fällen der sinnvollen Erweiterung der Anwendung.
Weiterhin erfordert die Nutzung mobiler netzwerkbasierender Anwendungen die bei mobiler Datenkommunikation auftretenden Verbindungsschwierigkeiten zu berücksichtigen. Aus diesem Grund beschäftigt sich diese Arbeit mit der Behandlung dieser Probleme.

1.2 Fragestellung

Das vorhandene Geoinformationssystem soll zur Darstellung der Spielwelt genutzt werden. Dazu ist es erforderlich den Zugriff auf die über den Webservice WMS angebotenen Orthofotos zu implementieren. Zusätzlich soll dieses Bildmaterial um virtuelle Spielgebäude ergänzt werden. Dies macht die Nutzung einer räumlichen PostGIS-Datenbank nötig.
Um eine möglichst große Nutzerzahl ansprechen zu können, sollte auf weit verbreitete Techniken gesetzt werden. Derzeit bietet GPRS die größte Abdeckung. Hieraus ergeben sich Anforderungen an den zu implementierenden Prototypen, da GPRS über eine geringe Bandbreite verfügt. Daraus ergibt sich die Notwendigkeit Datenvolumina einzusparen. Die Erreichung dieses Ziels erfordert die Entwicklung und Nutzung von Caching, Precaching und Datenkomprimierung. Außerdem ist die Sicherung der Stabilität der Verbindung zwischen Client und Server zu gewährleisten. Hierdurch wird die Nutzung eines Protokolls erforderlich, das die genannten Voraussetzungen erfüllt.
Weil es sich bei dem zu entwickelnden Prototypen um ein Multiplayerspiel handelt, müssen mehrspielertypische Aspekte berücksichtigt werden. Dies beinhaltet die Punkte Kommunikation und Interaktion, welche durch das Netzwerkprotokoll unterstützt werden müssen.

1.3 Aufbau der Arbeit

Die folgende Arbeit besteht aus vier Kapiteln. Nach dem Einleitungskapitel folgen in Kapitel 2 eine Übersicht über das Umfeld mobiler Spiele und die Beschreibung der Idee, sowie des Ablaufs des Spiels. Die technische Entwicklung und Implementierung thematisiert Kapitel 3. Wie in der Fragestellung beschrieben wird nach einem kurzen Überblick über Entwicklung auf mobilen Geräten auf die Nutzung des Geoinformationssystems eingegangen. Im Anschluss daran wird die Netzwerkkommunikation zwischen Server und Client behandelt. Die weiteren Abschnitte in Kapitel 3 erläutern die Nutzung der Datenbank, die Entwicklung einer Spielelogik und den Entwurf der Benutzeroberflächen. Kapitel 4 gibt einen Ausblick und fasst die Ergebnisse der Arbeit zusammen.

Kapitel 2

Prototyp Multiplayerspiel

Im folgenden Kapitel wird die momentane Problemsituation von Multiplayerspielen allgemein beschrieben. Danach wird speziell auf Probleme mobiler Spiele eingegangen und auf resultierende Möglichkeiten und Trends geschlossen. Die Spielidee des Prototypen wird daraufhin näher erläutert. Im Anschluß daran erfolgt die Aufschlüsselung der Spielelemente, die im Spiel zur Verfügung stehen.

2.1 Multiplayerspiele

Seit es das Internet gibt haben sich viele Formen der Interaktion herausgebildet. Eine davon ist das Multiplayerspiel. Hierbei verbinden sich mehrere Computerspieler mit einem Server oder untereinander und spielen gemeinsam in einer virtuellen Welt. Die ersten Probleme treten auf, wenn man die unterschiedlichen Verbindungsgeschwindigkeiten und Bandbreiten einzelner Spieler vergleicht und unter dem Aspekt betrachtet, dass alle Spieler in Echtzeit dieselbe Welt sehen. Hieraus wird klar, dass Wartezeiten auftreten und so unter Umständen das Spiel abgebrochen werden muss, da die Synchronität zwischen den Spielclients verloren geht. Dieses Problem kann damit gelöst werden, dass ein langsamer Spieler über Spieloptionen einstellen kann, wieviel Daten er senden will, um somit auch mit einer langsamen Verbindung am Spiel teilnehmen zu können. Ein kompletter Spielabbruch kommt dann zustande, wenn ein Spielserver ausfällt und somit kein Datenaustausch zwischen den Spielern mehr stattfinden kann. Ähnliche Störungen sind ebenfalls im mobilen Umfeld präsent. Diese Ausfälle treten hier vor allem auf, da die benötigte Netzabdeckung nicht an jedem Ort vorhanden ist.

Der Faktor Mensch spielt eine weitere große Rolle bei den Problemen eines Multiplayerspiels. Spieleserver sind häufig Opfer von Angriffen, nicht zuletzt weil die Angreifer beispielsweise so die Spieldatenbank manipulieren, dass sie sich einen Vorteil im Spiel schaffen. Der entstehende Schaden beeinflusst das Spielgeschehen allerdings nicht so weitreichend als das ein Spiel unterbrochen werden müsste. Um diese Problemsituation zu umgehen treffen die Administratoren des Servers verschiedene Sicherheitsmaßnahmen, wie den Einsatz von Anti-Cheat-Software[1][2] oder Blacklists. Aufgrund der mangelnden Verbreitung mobiler Multiplayerspiele ist dies noch kein Problem im mobilen Umfeld. Sollte es allerdings zu mehr Popularität von mobilen Spielen kommen, könnte dieses Problem ebenfalls auf diesem Gebiet entstehen.

2.2 Mobile Spiele

Der Unterabschnitt Mobile Spiele wird durch Möglichkeiten der Entwicklung auf mobilen Plattformen eingeleitet. Es folgt eine Betrachtung der potentiellen Kundschaft und eine Übersicht über vorhandene Abrechnungssysteme im kommerziellen Umfeld.

2.2.1 Möglichkeiten

Anhand der Tatsache, dass große Publisher wie Electronic Arts oder Sony BMG sich seit ein paar Jahren in dem Bereich des mobilen Entertainments engagieren, kann man erkennen wie hoch der Stellenwert von mobilen Spielen mittlerweile ist. Einige aktuelle Studien [Gar07, Pri07] zum Thema bescheinigen der Branche zukünftig ein großes Wachstum, was nicht zuletzt mit immer performanter werdenden Endgeräten als auch sinkenden Kosten für die Datenübertragung einhergeht.
Betrachtet man die Vielzahl an mobilen Endgeräten von Handys über PDAs bis hin zu Handheld-Konsolen, wie beispielsweise Nintendo DS oder Sony PSP, wird schnell das derzeitige Hauptproblem klar. Es bestehen gravierende Unterschiede zwischen den Möglichkeiten der Hardware. Diese äußern sich hauptsächlich in der Speicherausstattung, der Rechenleistung und der Displaygröße. Somit kann

[1] http://www.evenbalance.com/
[2] http://www.valvesoftware.com/

zwar ein Konzept für ein mobiles Spiel erarbeitet werden, aber die Umsetzung ist je nach Zielplattform verschieden.
Würde man sich beispielsweise auf die Plattform Handy konzentrieren so bestünde weiterhin das Problem, dass selbst die Endgeräte innerhalb der eigenen Produktgruppe einer Firma verschiedene Spezifikationen besitzen. Der Handyhersteller Nokia veröffentlicht zum Beispiel auf der eigenen Internetpräsenz unterschiedliche SDKs [Nok07]. Man hat sich mittlerweile zwar mit Java ME[3] auf einen Standard geeinigt, allerdings verzichtet man hiermit meist auf einen Teil an Funktionen und kann nicht die gesamte Leistung des Endgerätes nutzen. Dies zwingt die Entwickler dazu Änderungen entsprechend den Hardwareanforderungen zu implementieren [Tec07].

Da sich jedoch die Ausstattung der mobilen Endgeräte kontinuierlich verbessert, werden zusehends auch 3D-Spiele Einzug im mobilen Umfeld finden. Außerdem werden immer mehr Geräte mit integrierter Kamera und GPS-Unterstützung ausgeliefert, so dass auch hier eine spielerische Nutzung dieser Techniken stattfinden wird. Speziell die GPS-Unterstützung, die einen entscheidenden Teil des Prototyps ausmacht, findet momentan nur in einem größeren kommerziellen Projekt Verwendung. Dieses Rollenspiel ist in zwei Versionen mit und ohne GPS-Unterstützung erhältlich. In der GPS-Version ist es das Ziel des Spielers, ein virtuelles Dorf aufzubauen und zu beschützen. Während des Spielverlaufs wird das Dorf des Spielers angegriffen. Dieser Angriff erfordert vom Spieler, dass er sich zu einem definierten Standort bewegt, um dort sein virtuelles Dorf zu verteidigen [You07].

Eine Einschränkung für mobile Multiplayerspiele stellt die bisherige Technik zur Datenübertragung dar. Die geringe Bandbreite der existierenden Techniken führt bei größeren Datenmengen zu langen Wartezeiten, so dass Entwickler eines solchen Spiels darauf achten müssen, wie viele Daten wirklich benötigt werden bzw. ob man das Datenaufkommen noch verringern könnte. Außerdem ist zu beachten, dass häufig die Verbindung zum Spielserver abbrechen kann und deshalb entsprechende Algorithmen implementiert werden sollten, die Ausfälle handhaben und sicherstellen, dass der Spieler ohne möglichst hohen Aufwand zurück in das Spiel findet.

[3] http://java.sun.com/javame/index.jsp

Weiterhin haben mobile Spiele momentan noch das Problem, dass die Datenübertragungskosten für den Kunden relativ hoch sind, wenn dieser sich per GPRS oder UMTS zu einem Server verbindet und ständig Daten übertragen werden. Diese Beschränkung führte in der Vergangenheit dazu, dass hauptsächlich alte Spieleklassiker für den PC im Einzelspielermodus auf das mobile Umfeld übertragen wurden. Da sich aber gezeigt hat, dass Übertragungskosten schon teilweise pauschal abgerechnet werden können, verbessert dies die Konditionen, so dass mobile Multiplayerspiele durchaus massenmarktfähig werden können. So bietet beispielsweise E-Plus eine mobile Datenflatrate an [EP07]. Es werden Webtechniken, wie Flash [Sys07] oder Java Applets, zur Spielerstellung ebenso Verwendung finden, wie auch mobile Spieler mit Internetspielern in derselben virtuellen Welt interagieren werden [Dre07].

2.2.2 Potentielle Kundschaft

Ein anderer Trend, der sich abzeichnet ist die Entwicklung von sogenannten Casual Games oder auch Gelegenheitsspielen. Man hat festgestellt, dass hierfür die potentielle Kundschaft nicht nur die übliche Zielgruppe der Jungen unter 18 Jahren umfasst, sondern quer durch alle Bevölkerungsschichten akzeptiert wird [Rea07].

Dass sich mittlerweile besonders auch Frauen jeglichen Alters und Senioren für mobiles Gaming interessieren, beweist die Firma Nintendo mit den Spielen Nintendogs und Dr. Kawashimas Gehirn Jogging für die NDS [Nin07]. Dies zeigt, dass potentiell alle Menschen mit einem geeigneten mobilen Endgerät dazu neigen ein kleines Spiel zu spielen, wenn es die jeweiligen Interessen bedient. Die potentielle Kundschaft des im weiteren Verlauf vorgestellten Prototypen beschränkt sich allerdings auf eine männliche Zielgruppe.

2.2.3 Zahlungsmodalitäten

Bei der Bezahlung für mobile Spiele ist zwischen Handheld-Konsolen und anderen mobilen Endgeräten zu differenzieren. Während die Handheld-Konsolen direkt von Firmen vertrieben werden und diese auch Spiele hierfür bereitstellen, so wird bei anderen mobilen Geräten meist vom zuständigen Netzbetreiber ein

Portal betrieben auf welchem man Spiele erwerben kann. Hier zeichnet sich allerdings ein Trend dahingehend ab, dass immer mehr Entwickler den direkten Kontakt zum Kunden suchen und beispielsweise auf den eigenen Internetseiten ihre Produkte vertreiben.

Die Zahlungsmodalitäten für mobile Spiele sind sehr vielseitig. Am gebräuchlichsten sind mit mehr als 75 Prozent die Einmalzahlung und das monatliche Gebührenmodell. Es existieren weiterhin Abrechnungsmodelle mit pauschalen Gebühren, die die Nutzung von beliebig vielen Spielen ermöglicht oder auch die Bezahlung für einen einzelnen Spielablauf der vergleichbar mit einem Spielautomaten wäre [IDC07].
Ein weiterer Trend ist die Erschließung der mobilen Spiele durch die Werbewirtschaft. Spiele werden kostenlos angeboten und der Nutzer bekommt während des Spielens Einblendungen der Sponsoren zu sehen. Die Erschließung des Massenmarktes wird hierdurch weiter vorangetrieben [Dre07].

Der in dieser Arbeit behandelte Prototyp basiert auf einem monatlichen Gebührenmodell. Der Grund hierfür liegt in der gewollten Kundenbindung und dem Konzept der ständigen Erweiterbarkeit. Der Nutzer kann das Spiel kostenlos auf sein mobiles Gerät laden und befindet sich ab diesem Zeitpunkt in einem Demomodus. Der Spieler kann sich mit den Funktionen vertraut machen, gefällt ihm das Spiel kann er sich registrieren lassen und auf eine Aktivierung vom Server warten. Ist der Spieler dann freigeschaltet, stehen ihm sämtliche Spielfunktionen zur Verfügung.

2.3 Spielidee

Die Grundlage des Spiels basiert auf der Überlegung ein GPS-Signal so einzubinden, dass ein Mehrwert für das Spiel entsteht. Hieraus entstand die Idee die GPS-Koordinaten zu nutzen um, abhängig vom Standort, mehrere Werte errechnen zu lassen, die dem Spieler in Form von vier in Abbildung 2.1 zu sehenden virtuellen Ressourcen dargestellt werden. Der Spieler hat die Aufgabe mit seinem mobilen Gerät einen geeigneten Standort in der realen Welt zu finden, an welchem die Verteilung der durch die geographischen Koordinaten berechneten Ressourcen für den Spieler vorteilhaft ist. Die Positionsinformation stellt damit den wichtigsten Aspekt im Ablauf des Spiels dar.

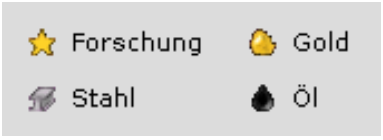

Abbildung 2.1: Die vier zu fördernden Ressourcen

Nun kann er, vorausgesetzt seine bestehenden Ressourcen erlauben es ihm, eine virtuelle Fabrik bauen. Ab diesem Zeitpunkt wird der entsprechende Wert jeweils pro Spieltag zum Ressourcenkonto des Nutzers addiert. Die geförderten Rohstoffe dienen dem Spieler nun dazu mehrere Aktionen auszuführen und sind von unterschiedlicher Wichtigkeit. Der Rohstoff Öl wird beispielsweise benötigt, um die Produktion der Fabriken zu gewährleisten und ist damit sehr entscheidend, während die Ressource Stahl für das Aufwerten der Gebäude benutzt wird und somit möglicherweise vernachlässigt werden kann.

Anhand der Aufgabenstellung war gegeben, dass Rollenspielelemente im Spiel vorhanden sein sollen. Hierfür entstanden typische Elemente in Form von Charaktereigenschaften. Diese dienen dem Spieler dazu, sich zu verbessern und somit Vorteile gegenüber anderen Spielern zu erlangen.

Man kann also festhalten, dass eine Mischung aus Aufbau- und Rollenspiel die Idee des Spiels darstellt. Aufgrund des prototypischen Charakters des Spiels gibt es kein fest definiertes Spielziel. Möglichkeiten hierfür in Form von Highscorelisten oder dem Erreichen eines vorher definierten Zieles sind allerdings gegeben.

2.4 Spielelemente

Die Elemente des Spiels lassen sich in drei Aspekte unterteilen. Der Bau von virtuellen Gebäuden, die Kommunikation zwischen den Spielern und die Rollenspielelemente in Form von Charaktereigenschaften werden in den drei folgenden Unterabschnitten betrachtet.

2.4.1 Bauen von Gebäuden

Ein Hauptbestandteil des Spiels besteht darin, mit dem mobilen Gerät anhand der Ressourcenbestimmung Fabriken zu bauen und Ressourcen fördern zu lassen. Nachdem ein Gebäude errichtet ist, ergeben sich mehrere Möglichkeiten der Interaktion mit diesem.

Der Besitzer hat die Möglichkeit seine Fabrik entsprechend 3 Kriterien aufzuwerten, um damit eine Leistungssteigerung zu erzielen. Diese stellt sich in Form einer Produktionssteigerung, eines Verteidigungswertes und der geförderten Ressourcenanzahl dar. Die Ressourcensteigerung produziert abhängig vom Standort mehr Ressourcen. Die Ressourcenanzahl ermöglicht einer Fabrik die Förderung von zwei, drei oder allen Ressourcenarten. Der Verteidigungswert stellt einen Wert für die Erfolgswahrscheinlichkeit einer feindlichen Übernahme oder eines Diebstahls dar.

Die Interaktionen der gegnerischen Spieler mit der Fabrik umfassen das Stehlen von Ressourcen und das Übernehmen des Gebäudes, so dass dieses bei Erfolg vom Gegner betrieben wird und somit Ressourcen für das gegnerische Konto produziert werden. Eine weitere Möglichkeit ist zu sehen, wo sich ein Spieler gerade befindet. Entsteht beispielsweise während des Spielens auf der Karte ein Gebäude, gibt der bauende Spieler damit an, wo er sich befindet.

2.4.2 Kommunikation zwischen den Spielern

Zur Interaktion innerhalb der Spielergemeinschaft stehen verschiedene Wege der Kommunikation zur Verfügung. Zum einen gibt es einen Chat an dem alle Spieler teilhaben können, wenn sie sich im Spiel befinden und zum anderen besteht ein Mailsystem, so dass sich Spieler auch dann gegenseitig kontaktieren können, wenn sich einer der beiden Spieler nicht im Spiel befindet.
Eine weitere Möglichkeit bietet das Handelssystem. Hiermit ist es den Spielern untereinander möglich Ressourcen zu tauschen oder auch zu verschenken. Damit ist für den Notfall des Ressourcenengpasses und der damit verbundenen Handlungsunfähigkeit innerhalb des Spiels auch eine Möglichkeit geschaffen, um den Spieler wieder zu motivieren, ohne das er sich aufgrund seiner misslichen Lage vom Spiel abwendet.

Die spannendste Interaktion liegt zwischen dem Spieler und den gegnerischen Gebäuden. Hier hat der Spieler zwei Optionen dem Gegner Schaden zuzufügen. Der Nutzer ist in der Lage jedes Gebäude anzugreifen, was sich in seiner näheren Umgebung befindet. Dieser Angriff kann entweder ein versuchter Diebstahl oder die Übernahme der kompletten gegnerischen Fabrik sein.
Dem Angreifer wird bei einem fehlgeschlagenen Diebstahl ein fester Betrag Ressourcen abgezogen. Gelingt der Diebstahl allerdings, so werden dem Fabrikbesitzer anteilig Rohstoffe abgezogen und dem Angreifer gutgeschrieben. Die Übernahme eines Gebäudes zieht dem Angreifer bei Misserfolg mehr Ressourcen ab als beim Diebstahl und die Chancen der erfolgreichen Ausführung sind halbiert. Gelingt dem Angreifer allerdings die Übernahme einer Fabrik, wird ihm diese übertragen. Die Produktion des eingenommenen Gebäudes wird somit ab diesem Zeitpunkt auf das Konto des Angreifers geleitet. Natürlich steht dem neuen Besitzer nun auch frei, das Gebäude abzureißen.

2.4.3 Charaktereigenschaften

Ein weiteres Hauptelement des Spiels stellt das integrierte Charaktersystem dar. Jeder Spieler besitzt die Attribute Angriff und Spionage. Mit dem Angriffswert kann der Spieler seine Erfolgschancen eines Diebstahls oder einer Gebäudeübernahme erhöhen. Der Spionagewert erhöht den Sichtradius bzw. die Entfernung wie nah der Spieler an gegnerischen Gebäuden stehen muss, um diese in der Angriffsliste angezeigt zu bekommen.
Weiterhin besitzt der Charakter vier Faktoren, die angeben, wie viel eine Ressource anteilig von den ursprünglich anfallenden Kosten für die entsprechenden Aktionen noch kostet. Ein hoher Attributwert an Gold würde beispielsweise dazu führen, dass beim Bau einer Fabrik weniger von der Ressource Gold abgezogen würde, als eigentlich für den Bau notwendig wäre.
Die Punkte für die Eigenschaften werden anhand verschiedener Levelstufen vergeben. Sobald der Spieler von der Ressource Forschung einen bestimmten Betrag erreicht hat, kann er diesen dazu benutzen eine Stufe aufzusteigen und drei Punkte an die bereits erwähnten Attribute zu verteilen. Im Verlauf der Aufwertung steigen die Levelkosten, so dass weiterhin ein Anreiz für einen fortgeschrittenen Spieler besteht, seine Charakterwerte aufzubessern.

Kapitel 3

Entwicklung des Prototypen

Dieses Kapitel behandelt die technische Umsetzung des im vorherigen Kapitel vorgestellten Prototypen in Form eines Spiels. Zunächst wird dazu die Entwicklungsplattform vorgestellt, um dann einen Überblick über die Architektur der Anwendung und die genutzte Infrastruktur zu geben. Anschließend wird der Prototyp in den darauf folgenden Abschnitten in seiner Implementierung näher betrachtet. Schließlich folgen zwei weitere Unterpunkte, die auf die Spielelogik und die Umsetzung einer Benutzerschnittstelle eingehen.

3.1 Entwicklungsumgebung

Der nachfolgende Abschnitt thematisiert zunächst allgemein die Entwicklung auf mobilen Endgeräten. Der Schwerpunkt liegt dabei in der Entwicklung für PDAs und Smartphones. Anschließend wird die für die Realisierung des Prototypen gewählte Plattform, das .NET Compact Framework 2.0, vorgestellt und ein kurzer Überblick über zusätzlich eingebundene Klassenbibliotheken gegeben.

3.1.1 Entwicklung auf mobilen Geräten

Obwohl die Entwicklung auf mobilen Endgeräten in den letzten Jahren weit voran geschritten ist, gibt es immer noch viele Einschränkungen, auf die bei der Entwicklung Acht gegeben werden muss. Diese beruhen zum Großteil auf der im Vergleich zu herkömmlichen Systemen viel schwächeren Hardware. So müssen die meisten PDAs mit 64 MB Arbeitsspeicher auskommen, wovon oft bereits ein

Großteil vom Betriebssystem in Anspruch genommen wird. Auf Mobiltelefonen ist der verfügbare Arbeitsspeicher dagegen noch viel geringer. Bei der Anwendungsentwicklung muss also darauf geachtet werden, den Speicherbedarf möglichst klein zu halten.
Ein weiterer Nachteil, der vor allem in der Spieleentwicklung deutlich wird, ist die für heutige Verhältnisse schwache Rechenleistung dieser Geräte. Die CPUs unserer Testgeräte hatten eine Rechenleistung von ca. 600 MHz (Dell Axim X51) bzw. ca. 200 MHz (O2 Xda Orbit). Auch gibt es im Gegensatz zu Desktopcomputern kaum beschleunigte Grafikprozessoren, da mobile Geräte eher für Officeanwendungen, als für Spiele ausgelegt sind. Für die für den Prototypen entwickelte 2D-Grafik reicht diese Leistung jedoch vollkommen aus.
Durch die Verkleinerung sämtlicher Komponenten ergeben sich auch softwareergonomische Probleme. Auf einen Bildschirm der Auflösung 240x320 können bei weitem nicht so viele Bedien- oder Anzeigeelemente untergebracht werden, wie man es normalerweise von Desktopanwendungen gewöhnt ist. Hier ist vor allem zu beachten, dass die Übersichtlichkeit erhalten bleibt und die Bedienelemente nicht zu klein werden. Weiterhin muss berücksichtigt werden, dass die Eingabe nicht über Tastatur/Maus erfolgt, sondern meist über Zeigestift und On-Screen-Tastatur. Daher sollte man viele Texteingaben vermeiden und die Oberfläche so einfach wie möglich gestalten.
Die Vielfalt an Geräten auf dem Markt erschwert nochmals die Entwicklung auf mobilen Geräten, da hier auf viele verschiedene Konfigurationen Rücksicht genommen werden muss, um möglichst jedem Anwender die Benutzung der Software zu ermöglichen.
Bei der Implementierung der Speicherung des Client-Cache beim Beenden des Spiels fiel auf, dass auch in diesem Bereich stark optimiert werden muss, da der Zugriff auf das Speichermedium sehr lange dauerte. Der Ansatz, die anfallenden Bilddaten vor der Speicherung stärker zu komprimieren, stellte sich ohne zusätzliche Klassenbibliotheken als nicht umsetzbar heraus. Durch die genannten Hardwareeinschränkungen ergibt sich, dass die zur Programmiersprache gehörenden Systembibliotheken ebenfalls auf ein Minimum reduziert sind. Mit den Unterschieden zur Standard-Systembibliothek befasst sich der nächste Abschnitt am Beispiel des .NET (Compact) Framework.

3.1.2 .NET Compact Framework 2.0

Microsofts .NET Framework besitzt ähnlich wie Java eine Virtual Machine, die Common-Language-Runtime (CLR), die den für .NET übersetzten Bytecode interpretiert und sich auch um die Speicherverwaltung kümmert. Der geschriebene Programmcode wird also zunächst in einen Zwischencode übersetzt, welcher dann von der CLR in Maschinenbefehle umgesetzt wird. Durch dieses Konzept wird die Programmierung erheblich vereinfacht. Zwar wird es in ähnlicher Weise von Java unterstützt, die Entscheidung fiel jedoch auf .NET, da die Entwicklung (vor allem grafischer Oberflächen) dort etwas einfacher ist und dadurch schneller Ergebnisse hervorbringt.
Es ist in .NET auch möglich, Programmteile, die in verschiedenen Programmiersprachen verfasst sind, gemeinsam zu nutzen. Von .NET unterstütze Sprachen sind beispielsweise C#, J# und Visual Basic. Wir haben uns für C# entschieden, da dessen Syntax der von Java ähnlich ist und dadurch der Einstieg in die Programmierung mit .NET erleichtert wurde.
Auf Geräten, die Windows Mobile als Betriebssystem nutzen, kommt jedoch nicht das Standard .NET Framework zum Einsatz, sondern die für mobile Endgeräte angepasste Version .NET Compact Framework. Es beinhaltet eine Teilmenge der Klassen des normalen .NET Framework, sowie einige weitere, für mobile Geräte spezifische, Klassen. Die größten, für die Entwicklung des Protypen relevanten, Unterschiede machten sich in der Grafikprogrammierung sowie in der Gestaltung der Benutzeroberfläche bemerkbar. In der Klassenbibliothek des .NET Compact Framework 2.0 fehlt beispielsweise die Unterstützung zum Drehen von Grafiken, so dass eine Drehung der Karte in Sichtrichtung nicht möglich ist. Außerdem lässt sich der Komprimierungsgrad von Grafiken im PNG oder JPEG-Format nicht einstellen, wodurch sich eine weitere Optimierung des Speicherbedarfs nicht ohne zusätzliche Bibliotheken verwirklichen lässt.
Beim Design der Oberfläche ist zu beachten, dass Windows Mobile jeweils nur eine Form im Vollbildmodus (abgesehen von der Windows-Menüleiste am oberen Bildschirmrand) darstellen kann. Dadurch ergaben sich weitere Probleme, da das Spiel viele weitere Menüs zur Anzeige von Informationen bietet. Gelöst wurden diese durch die Darstellung der Menüs in Panels, die je nach Bedarf eingeblendet werden. Weitere Informationen finden sich in Abschnitt 3.8.2. Durch den in Visual Studio integrierten Form Designer wurde die Arbeit an der Benutzero-

berfläche jedoch im Allgemeinen sehr vereinfacht, auch wenn einige gewünschte Optionen, wie beispielsweise das Ausblenden der nicht sichtbaren Menüpanels, nicht zur Verfügung standen. Auch der Debugger von Visual Studio war bei der Entwicklung durch On-Device-Debugging sehr hilfreich. Lediglich der Emulator ist wenig brauchbar, da er sehr ressourcenbelastend ist und die Geschwindigkeit der Ausführung kaum vernünftiges Testen zulässt.

3.1.3 Weitere Klassenbibliotheken

Auf Seite des Clients wurden keine weiteren Klassenbibliotheken eingebunden (hauptsächlich, da nur wenige für das .NET Compact Framework frei verfügbare Klassenbibliotheken zur Verfügung stehen).
Die Serveranwendung bedient sich zweier unter der GNU Lesser General Public License (LGPL)[1] stehenden Bibliotheken:

Npgsql [pgF07]
Npgsql ermöglicht den Zugriff auf PostgreSQL-Datenbanken. Diese Funktionalität war notwendig, da sämtliche Spieldaten, sowie die geographischen Objekte in einer PostgreSQL-Datenbank gehalten werden. Da Npgsql die `Mono.Security`-Bibliothek nutzt, wird diese ebenfalls benötigt.
SharpMap [Sha09]
SharpMap dient zur einfachen Abfrage eines Web Map Service (WMS). WMS stellt geographische Rasterdaten über eine Webschnittstelle bereit. Weiterführende Erklärungen zu WMS beinhaltet Abschnitt 3.4.1. Ein WMS-Request ist prinzipiell ein HTTP-Request. Daher wäre es auch ohne zusätzliche Klassenbibliotheken möglich, Kartenmaterial „von Hand“ abzufragen, indem man die `HttpWebRequest`-Klasse des .NET Frameworks nutzt. Jedoch ist der Zugriff auf WMS mit Hilfe von SharpMap bedeutend komfortabler und bietet auch mehr Kompatibilität mit verschiedenen WMS-Servern. Außerdem kann SharpMap bereits Vektoren aus PostGIS-Datenbanken als Layer über eine Karte legen und bietet viele weitere Funktionen zur Abfrage von WMS-Kartenmaterial.

[1] http://www.gnu.org/licenses/lgpl.html

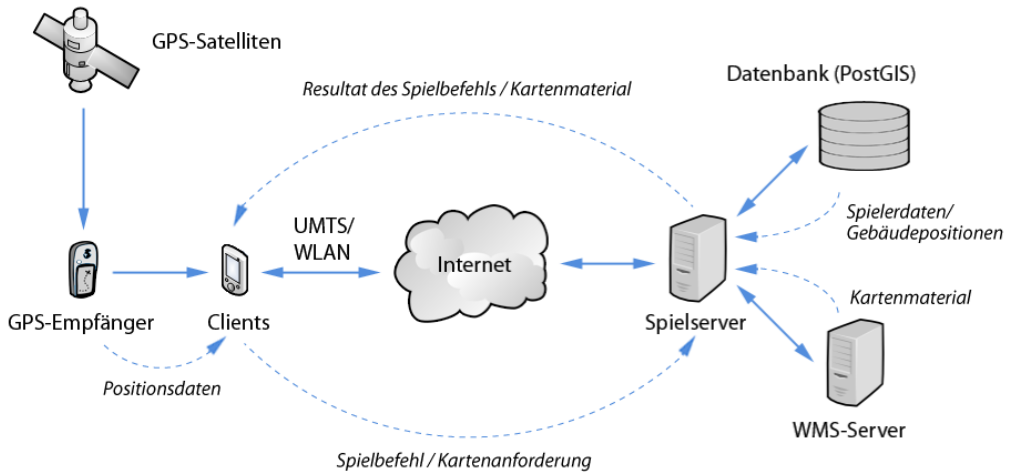

Abbildung 3.1: Architektur des Prototypen

3.2 Architektur

Beim Entwurf der Architektur des Prototypen wurde darauf geachtet, den Client möglichst über einen zentralen Server mit Daten zu versorgen, um aufgrund instabiler und langsamer Netzwerkverbindungen bessere Kontrolle über die gesendeten Daten zu haben. Würden Clients beispielsweise das benötigte Kartenmaterial selbst über einen WMS-Server beziehen, so wäre es schwierig Parameter wie Komprimierungsgrad der Bilder oder den zu verwendenden WMS-Server zu setzen. Indem sämtliche Daten über einen zentralen Spielserver ausgetauscht werden, lassen sich solche Optionen einfach und global für alle Clients festlegen. Nachdem ein Client seine Position mit Hilfe eines GPS-Empfängers ermittelt hat, kann er die für die Position benötigten Kartenabschnitte beim Server anfordern. Der Server prüft daraufhin, ob sich das angeforderte Kartenstück bereits in seinem internen Cache befindet, oder vom WMS-Server bezogen werden muss. Hat er das gesuchte Bild beschafft und aufbereitet, kann es dann an den Client versendet werden, welcher es dann direkt zur Darstellung der Umgebung nutzen kann. Zusätzliche Spielinformationen auf den Kartenstücken, wie die Gebäude, werden bereits vom Server vorm Versenden des Bildes eingezeichnet.

Alle weiteren Spielaktionen funktionieren ähnlich, indem Clients Spielbefehle an den Server senden und deren Ergebnisse als Antwort erhalten. Es besteht während des gesamten Spiels eine direkte Verbindung zwischen den Clients und dem Server. Dadurch hat der Server von sich aus die Möglichkeit, jederzeit die verbundenen Clients zu kontaktieren, um globale Nachrichten wie beispielsweise wichtige Mitteilungen zu versenden.
Die zu speichernden Daten der Spieler und Gebäude werden in einer externen SQL-Datenbank mit PostGIS-Erweiterung gehalten, auf die nur der Spielserver Zugriff hat.

3.3 GPS

Im Folgenden geht es um die Vorgehensweise zur Verarbeitung der vom GPS-Empfänger gelieferten Daten.
GPS-Empfänger liefern ihre Daten meist im NMEA-0183-Datenformat. Dies ist ein Format, welches verschiedene Datensätze zur Angabe des Geräts, der Position, der Richtung, der Geschwindigkeit usw. liefert. Wichtig für den Spielprototypen sind dabei hauptsächlich die folgenden beiden Datensätze:

GPRMC
Der GPRMC-Datensatz beinhaltet unter Anderem die Positionsinformation als nördliche Breite und östliche Länge in Grad.
HCHDG
Besitzt das GPS-Gerät einen Magnetkompass, so gibt es über diesen Datensatz die aktuelle Richtung aus.

Hier zwei Beispiele für die genannten Datensätze:

```
$GPRMC,191410,A,4735.5634,N,00739.3538,E,0.0,0.0,181102,0.4,E,A*19
$HCHDG,170.4,,,0.4,E*03
```

Im oberen Beispiel kann man im vierten und sechsten Datenfeld ablesen, dass sich das Gerät an der Position 47° 35,5634’ nördlicher Breite, 7° 39,3538’ östlicher Länge befindet. Das untere Beispiel zeigt die Richtung 170,4° im zweiten Feld an. Alle weiteren Datensätze und deren Erklärungen finden sich unter [NME07].

Die NMEA-Datensätze werden über die serielle Schnittstelle kontinuierlich vom GPS-Empfänger an die Clientsoftware gesendet. Sie muss dann nur noch aus den empfangenen Daten die benötigten Informationen herausfiltern, prüfen und dem Hauptprogramm eine Benachrichtigung über mögliche Änderungen liefern.
Diese Aufgaben übernimmt die Klasse `GPSAccess`. Bei ihrer Initialisierung müssen der COM-Port, mit dem das GPS-Gerät verbunden ist, und ob das Gerät einen Magnetkompass besitzt, angegeben werden. Dann kann der COM-Port geöffnet werden. Jedes mal, wenn ein neuer Datensatz empfangen wird, wird nun die Methode `com_DataReceived` aufgerufen.

```
1 if (dataReceived.StartsWith("$GPRMC"))
2 {
3    dataSet = dataReceived.Split(new Char[] { ',' });
4    newLat = Double.Parse(dataSet[3], new System.
        Globalization.CultureInfo("en-US"));
5    newLat = toDegrees(newLat);
6    newLon = Double.Parse(dataSet[5], new System.
        Globalization.CultureInfo("en-US"));
7    newLon = toDegrees(newLon);
8
9    if (Math.Abs(newLat - oldLat) > 0.00002)
10   {
11     if (!compassAvailable)
12     updateAngle();
13     oldLat = newLat;
14     update = true;
15    }
```

Listing 3.1: Ausschnitt aus com_DataReceived in GPSAccess

Sie liest zunächst den Datensatz ein und prüft dann, ob der eingegangene Datensatz einer der gesuchten (GPRMC oder HCHDG) ist. Auf den Datensatz HCHDG wird dabei nur geprüft, wenn `compassAvailable` auf `true` gesetzt ist. Anschließend werden die gesuchten Datenfelder (geographische Länge und Breite bzw. Richtung) aus dem Datensatz getrennt und in das gewünschte Format umgerechnet. Nun wird geprüft, ob seit dem letzten erhaltenen Wert eine Veränderung stattgefunden hat. Diese Prüfung ist notwendig, da das Gerät seine Daten kontinuierlich sendet, also auch, wenn sich die Position nicht geändert hat, und somit nicht jeder Datensatz notwendigerweise eine neue Position darstellt. Hier

ist außerdem noch eine Toleranzschwelle von 0,00002° eingebaut, damit sich das Bild bei geringen Schwankungen nicht bewegt.
Wurden neue Daten empfangen, setzt die Methode `update=true`. Dadurch wird nach dem Verarbeiten des Datensatzes ein `PlayerMoved`-Event gestartet, welches `MainForm` dazu veranlasst, die Karte zur entsprechenden Position zu bewegen, bzw. den Richtungszeiger zu drehen.
Besitzt das Gerät einen Magnetkompass, so wird zur Anzeige der Bewegungsrichtung einfach dessen Wert benutzt, der genau wie die Positionsdaten aus einem NMEA-Datensatz bezogen werden kann. Andernfalls wird mit Hilfe der vorherigen Position die ungefähre Richtung des letzten Schrittes bestimmt (Kurswinkel).
Die Klasse `GPSAccess` enthält außerdem noch einen Timer, der dazu dient, bei Initialisierung des Spiels festzustellen, ob das Gerät Positionen liefert. Er wird dazu bei Initialisierung des `GPSAccess`-Objekts gestartet und benachrichtigt `MainForm` nach spätestens 20 Sekunden, dass fortgefahren werden kann. Sind dann gültige Koordinaten vorhanden, wird der Ladevorgang fortgesetzt. Ist keine Position erhalten worden, wird dem Anwender die Möglichkeit gegeben nochmals 20 Sekunden zu warten oder den Vorgang abzubrechen. So wird verhindert, dass das Spiel zu Beginn „hängt“, falls keine gültige Position ermittelt werden kann.

3.4 Tile-Engine

Das Spielfeld des Clients muss aus dem Luftbildmaterial des jeweiligen Spielerstandortes und den nutzergenerierten Gebäuden bestehen. Um dieser Anforderung gerecht zu werden, wurde hierfür eine Tile-Engine entworfen, die das gegebene Kartenmaterial in Kacheln unterteilt und bei Bedarf zusammenfügt. Sie basiert hauptsächlich auf dem Web Map Service. Zu Beginn wird in diesem Abschnitt ein Überblick über WMS gegeben, gefolgt von der Erklärung, wie der Zugriff auf diese Schnittstelle ermöglicht wird. Danach folgen die Erklärungen für die einzelnen Mechanismen, um den Anforderungen an das mobile Umfeld gerecht zu werden. Hierbei handelt es sich um die geeignete Aufteilung des Kartenmaterials in Kacheln, sowie die Caching- und Precaching-Techniken, um möglichst wenig Bildmaterial übertragen zu müssen. Außerdem werden die durch Spielerbewegung und Spielerhandlung verursachten Veränderungen auf den Kacheln erläutert.

3.4.1 Übersicht WMS und WFS

Um auf das Kartenmaterial zugreifen zu können, werden Schnittstellen zu dem bestehenden Geoinformationssystem (GIS) benötigt. Diese Schnittstellen stellen der Web Map Service (WMS) und der Web Feature Service (WFS) dar. WFS ermöglicht es, per HTTP-Anfrage Karteninformationen als Vektordaten vom GIS zum Nutzer zu übertragen. Dies ist zum Beispiel sehr nützlich, wenn man ein Luftbild um bestimmte Elemente erweitern möchte. In der hier vorgestellten Umgebung könnte mithilfe von WFS die Verwaltung der Spielgebäude durchgeführt werden. Es wird allerdings kein WFS verwendet, da bereits eine direkte Datenbankverbindung besteht und somit durch SQL-Befehle auf die Vektorinformationen der PostgreSQL-Erweiterung PostGIS (zur Erklärung von PostGIS siehe 3.6.2) zugegriffen werden kann. Hiermit wird ein einfacher und schneller Zugriff auf die geographischen Objekte ermöglicht, ohne dass ein Mehraufwand durch HTTP-Anfragen entsteht.

WMS liegt als Spezifikation vom Open Geospatial Consortium[2] in der aktuellen Version 1.3 vor. Es dient im Gegensatz zu WFS der Übertragung von gerasterten und vektorisierten Kartenbildern. Hierfür werden HTTP-Anfragen an den Server gestellt, woraufhin das Bildmaterial gesendet wird. Aus diesem Grund kann ein beliebiger Browser zur Ansicht der Karten genutzt werden.

Damit ein Kartenausschnitt gesendet werden kann, muss zunächst einmalig eine `GetCapabilities`-Abfrage gestellt werden, um die unterstützten Funktionen und die verfügbaren Informationen des jeweiligen Mapservers abzufragen. Welche Koordinatensysteme zur Einordnung des Kartenmaterials unterstützt werden oder welche Layer angezeigt werden können, kann mithilfe folgender Anfrage herausgefunden werden.

```
http://lbs.uni-koblenz.de/cgi-bin/mapserv.exe?map=
/ms4w/mobico.map&REQUEST=GetCapabilities&SERVICE=WMS
```

Es wird nun eine Antwort in Form einer XML-Datei vom Server zurückgeschickt.

```
1     <Layer>
2     <Name>server</Name>
```

[2] http://www.opengeospatial.org/

```
3     <Title>WMS Mobico Server</Title>
4     <SRS>EPSG:31467</SRS>
5     <SRS>EPSG:4326</SRS>
6     <LatLonBoundingBox
7     minx="7.48162" miny="50.3292"
8     maxx="7.51196" maxy="50.3476" />
9     <BoundingBox SRS="EPSG:31467"
10    minx="3.39194e+006" miny="5.578e+006"
11    maxx="3.39406e+006" maxy="5.58e+006" />
12        <Layer queryable="0" opaque="0" cascaded="0">
13        <Name>mittelrhein</Name>
14        <Title>orthopics</Title>
```

Listing 3.2: Ausschnitt der Serverantwort

Nachdem entsprechende Optionen ausgewertet wurden, kann eine Anfrage nach einem Kartenausschnitt gestellt werden. Dieser enthält die Angaben über die Position und Größe des Ausschnittes, sowie die Größe und das Format des zu sendenden Bildes. Weiterhin wird definiert, welche der zur Verfügung stehenden Layer in den Rendervorgang miteinbezogen werden sollen, welches Koordinatenreferenzsystem für die Angaben der Position benutzt wurden und welche Version von WMS benutzt werden soll.

```
http://lbs.uni-koblenz.de/cgi-bin/mapserv.exe?
map=/ms4w/mobico.map&REQUEST=GetMap
&BBOX=7.560000,50.362500,7.561500,50.364000
&WIDTH=300&HEIGHT=300&LAYERS=mittelrhein
&FORMAT=image/jpeg&SRS=EPSG:4326&VERSION=1.1.1
```

Als Antwort wird hierauf der gewünschte Kartenausschnitt an den Client gesendet.

3.4.2 Zugriff auf WMS

Die Open Source Bibliothek SharpMap wird eingesetzt um die HTTP-Anfragen nicht von Hand zusammensetzen zu müssen und somit in C# bequemere Anpassungsmöglichkeiten zu haben, falls die Einstellungen des zugrundeliegenden Mapservers verändert werden sollen. Die Benutzung von SharpMap erfolgt über

die Einbindung einer DLL-Programmbibliothek [Sha09]. Es stehen nun verschiedene Klassen zur Verfügung. Die hier verwendeten beschränken sich auf die drei Klassen `SharpMap.Map`, `SharpMap.Layers.WmsLayer` und `SharpMap.Geometries.BoundingBox`.

Eine Instanz von `SharpMap.Map` dient als Gerüst, um darin die Layer-Instanz von `SharpMap.Layers.WmsLayer` hinzuzufügen. Für das Spiel wird nur ein Layer mit den orthographischen Luftbildern benötigt.

```
1 layer = new SharpMap.Layers.WmsLayer("MainLayer", url);
2 layer.ContinueOnError = false;
3 layer.SpatialReferenceSystem = srs;
4 layer.SetImageFormat(imageFormat);
5 layer.TimeOut = 10000;
6 layer.AddLayer(s);
7 mapSize = new System.Drawing.Size(tileSize, tileSize);
8 map = new SharpMap.Map(size);
9 map.Layers.Add(layer);
10 map.Zoom = zoom;
```

Listing 3.3: Konfiguration für den Mapserverzugriff mit Sharpmap

Um den benötigten Kartenausschnitt in Form eines Rechtecks zu definieren wird `SharpMap.Geometries.BoundingBox` benutzt. Sind alle Parameter gesetzt, genügt der Aufruf von `GetMap`, um die HTTP-Anfrage zusammenstellen zu lassen, diese zu versenden und ein Bild zurück zu geben.

```
1 bbox = new SharpMap.Geometries.BoundingBox(lon,lat,lon +
      zoom,lat + zoom);
2 map.ZoomToBox(bbox);
3 Image newImage = map.GetMap();
```

Listing 3.4: Mapserverzugriff mit Sharpmap

3.4.3 Aufteilung des Kartenmaterials in Kacheln

Damit möglichst wenig Datenaufkommen entsteht, wurde ein Algorithmus implementiert, der die gegebene Luftbildkarte angemessen unterteilt. Basierend auf dem Koordinatenreferenzsystem EPSG:4326 bzw. WGS84[3] wird die Position des

[3]http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf

Senders verarbeitet. Der Längen- bzw. Breitengrad wird jeweils durch einen konstant definierten Wert von 0,0015 dividiert. Durch das Abschneiden der Nachkommastellen entstehen hierbei ganze Zahlen. Diese werden dazu benutzt ein Kartenteil eindeutig anhand von zwei Zahlenwerten zu kennzeichnen. Somit wird ein eigenes Koordinatensystem über die Karte gelegt, anhand welchem man festlegen kann, welche Teile jeweils für den Nutzer notwendig sind. Jedes Teil hat im Fall das Prototyps eine Größe von 300x300 Pixeln und wird im JPEG-Format auf rund 20 KB komprimiert. 300 Pixel entsprechen auf dem Luftbild in etwa 100 Metern.

Abbildung 3.2: Beispiel für die Kachelbezeichnung

In der vorhergehenden Abbildung 3.2 ist ein Beispiel der Kacheluntergliederung zu sehen. Die Kachel mit den Kachelkoordinaten 5040/33576 deckt den Bereich der GPS-Koordinaten von 7,56° bis 7,5615° östlicher Länge und 50,364° bis 50,3655° nördlicher Breite ab. Befindet sich der Spieler an der GPS-Position 7,560389° Ost, 50,364554° Nord und somit innerhalb dieser beiden Intervalle, so ergeben sich anhand der Division mit dem vorher festgelegten Wert von 0,0015 die Werte

5040,2593 und 33576,3693. Die Stellen hinter dem Komma werden ignoriert und damit steht der Spieler auf der besagten Kachel. Der konstante Wert 0,0015 stellt die Grad dar, die auf einem Kartenteil abgebildet werden. Durch die Veränderung des Verhältnisses zwischen den Grad pro Kachel und den Ausmaßen einer Kachel lässt sich ein Vergrößern oder Verkleinern des Kartenausschnittes erzielen.

3.4.4 Caching

Die Zwischenspeicherung von Kartenteilen erfolgt sowohl auf dem Spielserver als auch auf jedem teilnehmenden Client. Der Server speichert bis zu 500 angefragte Teile und behält sie im Cache, bis diese Anzahl überschritten wird. Während dieser Überschreitung werden nach dem First In - First Out - Prinzip die Kacheln gelöscht, bis die Anzahl der Kacheln 400 beträgt.

Der Spielclient verfährt nach einem anderen Prinzip mit seinen angefragten Teilen. Die Beschränkung der Kartenteile liegt bei einer Anzahl von 30. Sobald diese Zahl erreicht ist, wird mithilfe der Funktion `flushCache` in der Klasse `HashCache` ermittelt, welche Kacheln nicht gelöscht werden dürfen. Der Client muss im Gegensatz zum Server bestimmte Kacheln immer verfügbar haben, da an der Position des Spielers stets Kartenkacheln zur Anzeige benötigt werden. Das beim Server angewendete FIFO-Prinzip könnte dazu führen, dass Kacheln gelöscht werden, die dem Spieler momentan an seiner Position angezeigt werden müssen. Aus diesem Grund wird die Fläche von 5x5 Kacheln mit der Kachel der Spielerposition als Mittelpunkt bestimmt, so dass der gesamte Cache gelöscht wird, es sei denn eine zu löschende Kachel befindet sich in dem erwähnten Umkreis.

```
1 for (int i = 0; i < cacheAmount; i++)
2     {
3     if (i % cacheSquareSize == 0)
4         {
5         deltaX = playerTilePos.X - startX;
6         deltaY += 1;
7         }
8     if (ContainsKey(deltaX.ToString() + "," + deltaY.
      ToString()))
9         notToDelete[deltaX.ToString() + "," + deltaY.
```

```
                ToString()] = true;
10      deltaX += 1;
11      }
```

Listing 3.5: Berechnung der nicht zu löschenden Kacheln

Es wird von links nach rechts und von oben nach unten innerhalb des nicht zu löschenden Kachelbereichs kontrolliert, ob Kartenmaterial an den Teilkoordinaten vorhanden ist. Ist dies der Fall, wird in der Hashtable `notToDelete` vermerkt, dass ein gefundenes Teil im Cache nicht gelöscht werden darf. Nach Durchlaufen der Schleife werden vom gesamten Zwischenspeicher alle in `notToDelete` gemerkten Kacheln abgezogen, so dass alle im Cache verbleibenden Kacheln gelöscht werden können.

Beendet der Client sein Spiel, so werden alle Karten, die im Cache vorhanden sind, auf dem Endgerät gespeichert, so dass beim Neustart nur die geänderten oder nicht vorhandene Teile neu geladen werden müssen. Diese Maßnahme reduziert das zu übertragende Datenvolumen beim Start drastisch.

3.4.5 Änderungen an Kacheln und deren Auswirkungen

Da statische Kartenausschnitte für das Spiel nicht ausreichen und virtuelle Spielgebäude das Kartenbild verändern, erfordert es eine Verwaltung der Kacheln. Hierfür wurde ein Versionssystem entwickelt. Zu Beginn des Spiels bekommt jede vom Server geladene Kachel außer ihren beiden eindeutigen Teilkoordinaten die Versionsnummer 1 zugeordnet. Verändert ein Nutzer nun im Verlauf des Spiels eine Kachel, indem er ein Gebäude baut, aufwertet oder abreißt, so ändert sich das Aussehen der Kachel. Die Versionsnummer der Kachel wird nun um eins erhöht. Da eine Aktualisierung des Kartenteils stattgefunden hat, schickt der Server eine Benachrichtigung an alle verbundenen Clients. Diese überprüfen daraufhin, ob die aktualisierte Kachel im eigenen Cache vorhanden ist und ob die Version vom Server aktueller ist. Wurde hierdurch eine veraltete Kachel festgestellt sendet der Client eine Anfrage an den Server. Als Antwort bekommt der Client das aktualisierte Kartenteil zugesendet.

3.4.6 Precaching der Kacheln

Zur Vorsorge von Engpässen bei der Datenübertragung erfolgt ein Vorladen der Kartenteile nach einem bestimmten Schema. Beim Loginvorgang des Nutzers wird anhand der GPS-Position bestimmt, auf welchem Teil sich der Spieler befindet. Nun werden anhand dieses Teils die 8 umliegenden Kartenteile ermittelt. Der Server lädt nun insgesamt 9 Kartenausschnitte vom Mapserver bzw. aus seinem Cache. Nun erfolgt die Abfrage, ob der Client nicht schon im eigenen Cache die entsprechenden Kartenteile gespeichert hat, so dass eine erneute Übertragung der Karten nicht erforderlich wäre. Abhängig von dieser Überprüfung sendet der Server daraufhin 0 bis 9 Kartenausschnitte an den Client. Die 9 Kacheln werden nun zu einem großen Bild zusammengefügt, welches im weiteren Verlauf immer dazu dient, den passenden Ausschnitt entsprechend der Spielerposition bereitzustellen.

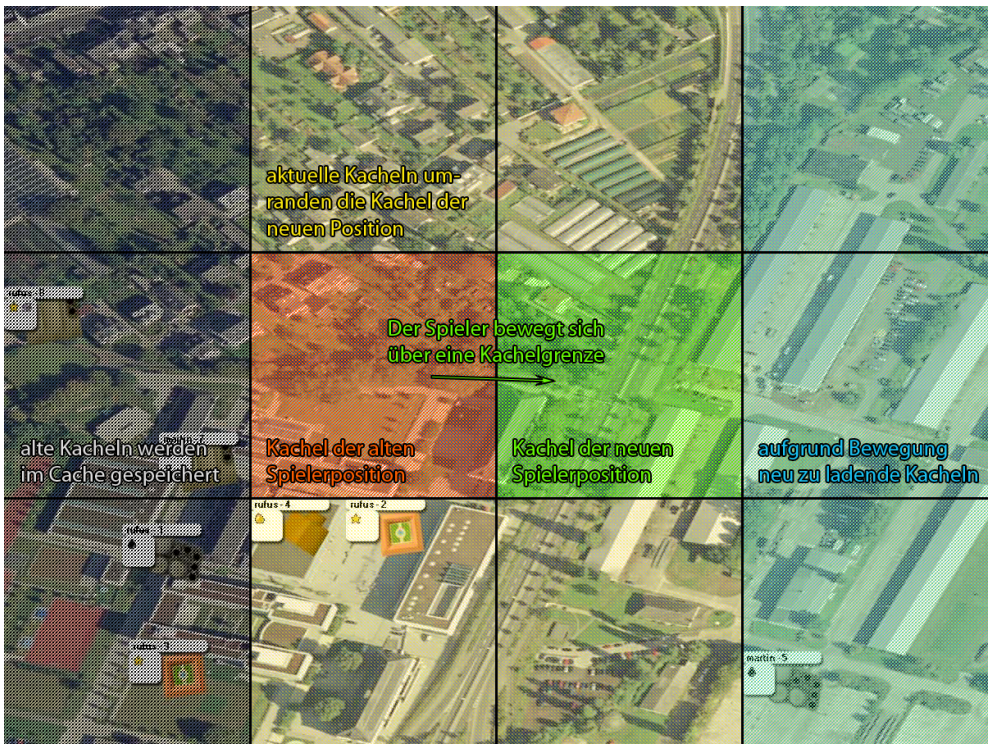

Abbildung 3.3: Die Überschreitung eines Kachelrandes

Die Idee ist demnach, dass der Spieler sich immer auf der mittleren Kachel von einem 3x3 Gitter befindet. Um dies auch im Verlauf des Spiels zu gewährleisten, wird erkannt wann sich der Client über eine Kachelgrenze bewegt. Bewegt sich der Client nun soweit, dass dieser sich nicht mehr auf dem mittleren Teil befindet, so wird es notwendig in der entsprechenden Richtung weitere Kartenausschnitte zu laden. Die Anzahl der nachzuladenden Teile beträgt hierbei stets 3 Kacheln. Außerdem wird das bisher bestehende Bild dahingehend verändert, dass diese 3 neuen Kacheln in das Bild eingefügt werden und an der entgegengesetzten Seite drei Kacheln gelöscht werden. Die drei alten Kacheln bleiben im Cache erhalten, solange dieser nicht voll ist, um weiter Zugriff darauf zu haben, falls sich der Spieler wieder zurückbewegt. In Kombination mit dem Versionssystem für die Kartenteile wird bei jeder Kachelüberschreitung zusätzlich kontrolliert, ob die 3 benötigten und eventuell im Cache vorhandenen Kacheln noch aktuell sind oder ob eine Anfrage nach aktualisierten Kacheln stattfinden muss. Ein Beispiel für diesen Ablauf visualisiert die Abbildung 3.3
Analog zum Loginprozess wird für das Nachladen beim Server angefragt, ob dieser die Kacheln zwischengespeichert hat oder ob er sie neu vom Mapserver laden muss. Vor der Übertragung der Daten wird ebenfalls wieder ermittelt, ob der Client im Cache nicht schon die aktuellste Version der einzelnen Kachel gespeichert hat, so dass wieder darauf geachtet wird, nicht unnötig Daten zu übertragen.

3.4.7 Spielerbewegung auf der Karte

Damit ein Spieler stets seine genaue Kartenposition in der Mitte des Bildschirms angezeigt bekommt, muss anhand der Kachelkoordinaten und der GPS-Position bestimmt werden, wo sich der Spieler befindet. Die Teilkoordinate der betreffenden Kachel wird mit der bereits eingeführten Konstante für die Berechnung des Gitters multipliziert, so dass sich die Differenz zwischen GPS-Position und Kachelrand berechnen lässt. Diese Differenz stellt den Abstand des Nutzers zum Rand der mittleren Kachel in Grad dar. Um den Abstand in Pixel umzurechnen, wird der Abstand durch ein Pixelverhältnis dividiert. Der Wert für dieses Verhältnis entsteht aus der Division einer Kachelbreite in Pixel durch eine Kachelbreite in Grad.

```
1 private Position getPositionOnTileByWorldCoordinates()
2     {
3     int pixelPosX = (int)((lon - playerTilePos.X *
          tileSizeInWorldCoordinates) / pixelRatio);
4     int pixelPosY = (int)(((playerTilePos.Y *
          tileSizeInWorldCoordinates +
          tileSizeInWorldCoordinates) - lat) / pixelRatio);
5     return new Position(pixelPosX,pixelPosY);
6     }
```

Listing 3.6: Umrechnung von GPS-Koordinaten in Pixelkoordinaten

Wie man dem Listing 3.6 entnehmen kann, werden der x-Abstand und y-Abstand auf verschiedene Weisen berechnet. Das Problem liegt in der Transformation vom Weltkoordinatensystem in das Pixelkoordinatensystem begründet. Der Ursprung des Weltkoordinatensystems befindet sich von Deutschland aus betrachtet in der unteren linken Ecke, wohingegen das Pixelkoordinatensystem im .NET Compact Framework mit dem Ursprung in der oberen linken Ecke rechnet. Hierdurch wird es notwendig, die Berechnung der y-Position abzuwandeln, während die x-Position den linken Rand als Referenz nutzen kann. Zur y-Kachelposition wird eine Kachelbreite in Grad hinzuaddiert, so dass mit dem oberen Rand der Spielerkachel gerechnet werden kann. Dies ermöglicht die Subtraktion der eigentlichen Spielerposition und eine geeignete Interpretation der Pixelposition.

Um nun ausgehend von der richtigen Spielerposition den entsprechenden Kartenausschnitt zu berechnen, wird die Auflösung des mobilen Endgerätes in Pixeln dazu benutzt ein Rechteck zu definieren, welches das anzuzeigende Bild aus den jeweils aktuellen 9 Kacheln erzeugt (siehe Abbildung 3.4). Die errechnete Position stellt den Mittelpunkt dieses Rechtecks dar.

Für das Zuschneiden werden je nach Position, Größe der Kacheln und Auflösung des mobilen Endgerätes eine, zwei, vier, sechs oder alle neun Kacheln benutzt, um die richtige Darstellung des Ausgabebildes zu erzielen. Da das GPS-Gerät konstant Werte übermittelt und diese sich selbst ohne Bewegung des Nutzers leicht verändern, wurde ein Grenzwert eingeführt, um das ständige Neuzeichnen der Ausgabe und ein damit verbundenes Flackern zu unterbinden. Weiterhin wird je nach Hardwarevoraussetzung mit Magnetkompass oder durch Be-

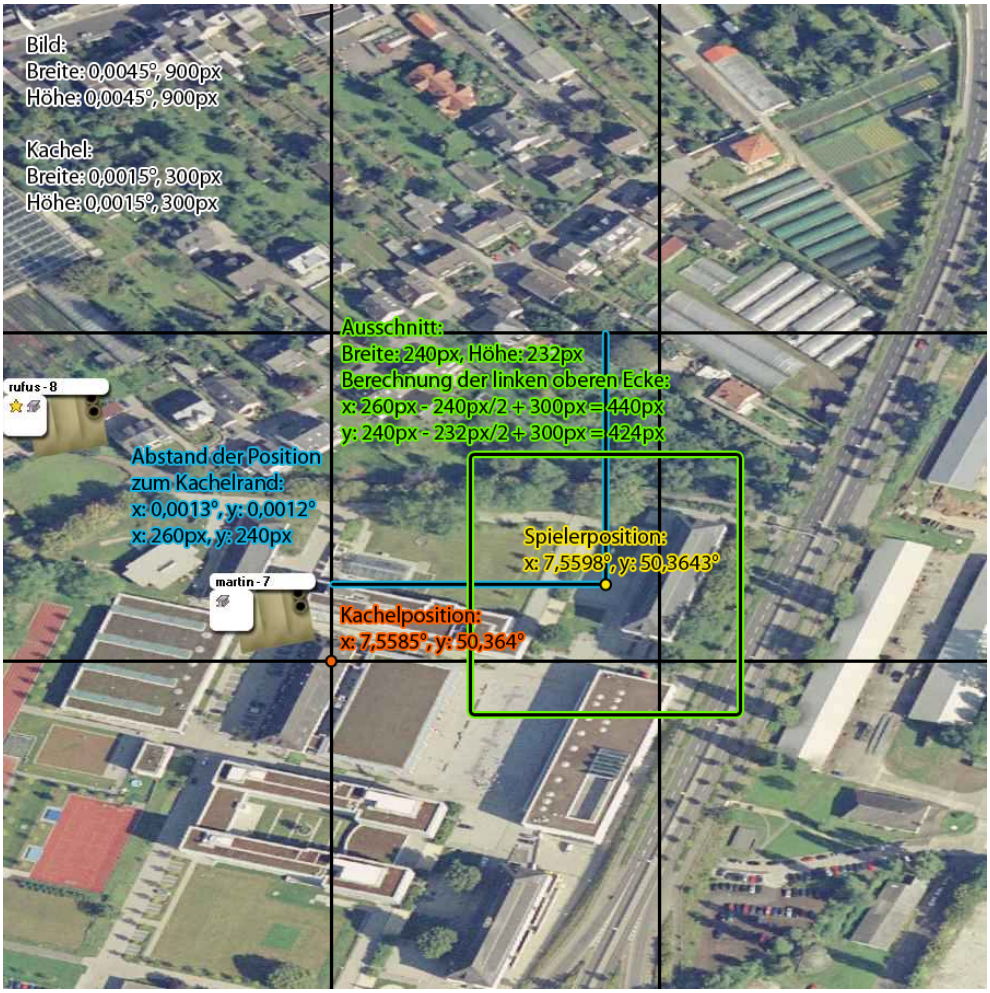

Abbildung 3.4: Das Spielfenster wird aus den 9 aktuellen Kacheln ausgeschnitten

rechnung mit einem Pfeil die Blickrichtung des Nutzers angezeigt. Auch dieser Wert besitzt einen entsprechenden Grenzwert, um so eine ruhigere Bewegung des Pfeils zu ermöglichen.

Falls sich der Spieler bewegt und eine neue Koordinate gesendet wird, so würde an dieser Stelle der Kartenausschnitt zu der neuen Position springen. Um diesen Sprung zu vermeiden, wurde die Methode `ScrollMapTo` implementiert, welche eine gleichmäßig scrollende Bewegung des Spielfelds vom alten zum neuen Standort durchführt. Hierbei wird zu Beginn die Differenz in Form von Blickrichtung, x-Wert und y-Wert zwischen zwischen altem Standort und neuem Standort

berechnet, so dass im Anschluss daran mit diesen Werten ermittelt wird, wieviel Zwischenstufen benötigt werden, um das Bild flüssig zu bewegen. Ist diese Bestimmung erfolgt, wird zwischen zwei Möglichkeiten der Verschiebung unterschieden. Je nachdem auf welcher der beiden Koordinatenachsen eine größere Bewegungsdifferenz vorhanden ist, wird diese als Referenz dazu verwendet an dieser das Bild entlang zu schieben. Abhängig von der Veränderung der jeweils anderen Koordinate wird nun in dieser Richtung das Bild ebenfalls verschoben. Diese Vorgehensweise ist dem Algorithmus von Jack Bresenham entlehnt [Bre07].

```
1 for (int i = 1; i <= xSteps; i++)
2 {
3     // In X-Richtung wird immer verschoben.
4     oldPixelPosition.X += xDiff;
5     stepsCounter++;
6     // Winkel fuer Richtungszeiger einen Schritt weiter
          bewegen.
7     oldAngle += angleStep;
8     // In Y-Richtung dagegen nur, wenn genuegend X-Schritte
          gemacht wurden.
9     if (yStepRatio != 0 && stepsCounter >= yStepRatio)
10    {
11        stepsCounter = 0;
12        oldPixelPosition.Y += yDiff;
13    }
```

Listing 3.7: Bewegung entlang der x-Achse - Auszug aus ScrollMapTo

3.5 Client/Server-Kommunikation

Zunächst befasst sich der Abschnitt Client/Server-Kommunikation mit den notwendigen Voraussetzungen für eine mobile Netzwerkkommunikation. Es folgen die Beschreibung und die Implementierung des für den Prototypen entworfenen Protokolls, um den genannten Anforderungen gerecht zu werden. Letztlich behandelt der Abschnitt, wie sich die Anwendung bei Verbindungsausfällen verhält und wie dies umgesetzt wurde.

3.5.1 Anforderungen

Um Datenkommunikation im mobilen Umfeld zu ermöglichen, gibt es derzeit drei weit verbreitete Technologien: WLAN, UMTS und GPRS. GPRS bietet dabei die geringste Bandbreite, hat jedoch die größte Verfügbarkeit und ist heutzutage mit fast jedem Mobilfunkgerät nutzbar. Um eine große Anwenderschicht ansprechen zu können, sollte daher die Datenmenge möglichst gering gehalten werden. Das größte Problem zur Einsparung von Datenvolumina stellt dabei das vom Geoinformationssystem zur Verfügung gestellte Kartenmaterial. Dazu implementiert der Prototyp, wie bereits oben beschrieben, verschiedene Techniken wie Caching/Precaching und Aufteilung des Kartenmaterials in Kacheln, um die übertragene Datenmenge möglichst gering zu halten. Außerdem werden die Bilder vor der Übertragung zum Client stark komprimiert, da der Unterschied der schlechteren Darstellung auf kleinen Displays nicht so sehr ins Gewicht fällt.
Auch die sich ständig ändernde Umgebung birgt bei mobiler Datenübertragung einige Probleme. Bandbreiten können nicht garantiert werden und schwanken stark je nach Empfangsstärke. Ebenso treten häufig komplette Verbindungsausfälle auf. Daher muss in einem solchen Fall die Verbindung zwischen Client und Server möglichst schnell und automatisch wieder hergestellt werden.

3.5.2 Entworfenes Protokoll

Um den Overhead möglichst gering zu halten, fiel die Entscheidung auf die Entwicklung eines eigenen Protokolls, welches nur die wirklich notwendigen Daten über TCP-Sockets überträgt. Die übertragenen Daten werden dabei in XML dargestellt.
Die Erzeugung dieser XML-Daten erfolgt durch die statischen Methoden der Klasse `CommandCreator`. Durch die gute XML-Unterstützung im .NET (Compact) Framework lassen sich die XML-Nachrichten unkompliziert mit der Klasse `XmlDocument` erzeugen. Anschließend wird das XML-Dokument in einen String umgewandelt, der dann wiederum in ein Byte-Array konvertiert werden kann. Dieses Byte-Array lässt sich dann über den bei Verbindungsbeginn geöffneten Socket übertragen. Dazu ein Beispiel:

```
1 <command>
2         <upgradebuilding>
3                 <buildingid>4</buildingid>
```

```
4                   <type>production</type>
5           </upgradebuilding>
6   </command>
```

Listing 3.8: Befehl zum Aufwerten eines Gebäudes.

Der Befehl im obigen Beispiel fordert beim Server die Aufwertung der Produktion des Gebäudes mit der ID 4 an. Nach Empfang des Befehls führt der Server die angeforderte Aktion aus und schickt seine Antwort an den Client. Diese könnte wie folgt aussehen:

```
1   <command>
2           <buildingupgradestatus>
3                   <status>ok</status>
4                   <buildingid>4</buildingid>
5           </buildingupgradestatus>
6   </command>
```

Listing 3.9: Antwort auf Aufwertungsanfrage

Damit teilt der Server dem Client mit, dass das Gebäude mit der ID 4 die gewünschte Aufwertung erhalten hat.
Auf die genauere Funktionsweise der Datenübertragung wird im Abschnitt 3.5.3 näher eingegangen. Im Folgenden werden kurz die verschiedenen Nachrichten, welche Client und Server austauschen, vorgestellt.

3.5.2.1 Spielbefehle und Benachrichtigungen

Die meisten Spielbefehle funktionieren nach dem oben vorgestellten Prinzip. Der Client erstellt eine Anfrage an den Server, die er als XML dargestellt versendet. Auf dem Server wird nach Empfang ein der Anfrage entsprechendes `Command` erstellt und sobald wie möglich ausgeführt. Nach der Ausführung kann der Server dem Client das Ergebnis der Anfrage ebenfalls über eine XML-Nachricht mitteilen. Dabei werden sowohl einzelne Statusmitteilungen, als auch lange Listen von Informationen übertragen. Eine Übersicht über alle zwischen Server und Client ausgetauschten Spielbefehle findet sich im Anhang A.
Es ist auch möglich, dass Nachrichten vom Server verschickt wird, ohne dass der Client Anfragen gestellt hat. Dies ist zum Beispiel notwendig, um Spieler über neue Post/Chatnachrichten, Angriffe oder Handelsangebote zu informieren. Diese Benachrichtigungen werden vom Client in ähnlicher Weise wie vom Server

empfangen und weiter verarbeitet, indem sie meist im Benachrichtigungsfeld des Menüs angezeigt werden. Weiteres dazu findet sich im Abschnitt 3.8.2.

3.5.2.2 Loginvorgang

Durch Initialisieren eines `Client`-Objekts wird auf Clientseite zunächst versucht, eine Verbindung zum Server zu öffnen. Ist der Server gestartet, befindet er sich im Listen-Zustand, wodurch er auf eingehende Verbindungen wartet und die Verbindung mit dem anfragenden Client akzeptieren sollte. Nun erzeugen beide Seiten ein `NetworkStream`-Objekt, über welches sie miteinander kommunizieren können. Anschließend sendet der Client eine `<login>`-Nachricht, mit dem er seinen Benutzernamen und sein Passwort mitteilt. Auf Serverseite prüft der Server diese Daten und antwortet mit einer `<loginreponse>`-Nachricht. Für den Login-Status dieser Nachricht ergeben sich folgende Möglichkeiten:

- Der Login war erfolgreich
  Status ist `LOGIN_SUCCESSFUL`
- Das Passwort war falsch
  Status ist `WRONG_PASSWORD`
- Der Benutzer ist bereits eingeloggt
  Status ist `ALREADY_LOGGED_IN`

Bei einer der beiden Fehlermeldungen wird der Vorgang auf Clientseite abgebrochen und die Verbindung damit geschlossen. War der Login-Versuch erfolgreich, so antwortet der Client mit einer `<loginok />`-Nachricht und führt seinen Login-Vorgang weiter aus. Durch diese Nachricht kann der Server den Client auf `loggedIn` setzen und weiß damit, dass dieser Client von nun an Daten empfangen kann. Man könnte den Client auch bereits vorher auf `loggedIn` setzen, da der Server bereits eher weiß, dass der Login erfolgreich war. Jedoch verteilt der Server gelegentlich Daten an alle verbundenen Clients. Da der hier beschriebene Client aber seinen Loginvorgang noch nicht abgeschlossen hat, ist er auch noch nicht bereit Spieldaten zu empfangen. Zu diesem Zeitpunkt könnten andere Spielnachrichten möglicherweise zu einem Absturz führen, da der Client bisher nur Loginmeldungen erwartet. Abbildung 3.5 stellt den Loginvorgang zur Übersicht visuell dar.

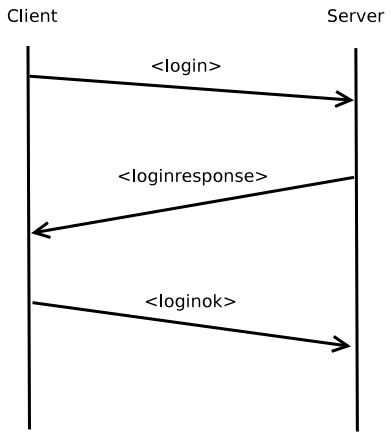

Abbildung 3.5: Loginvorgang des Prototypen

3.5.2.3 Übertragung des Kartenmaterials

Die Übertragung des Kartenmaterials unterscheidet sich erheblich von den anderen Spielnachrichten, da hier neben reinen Textinformationen auch Binärdaten übertragen werden. Bei der Übertragung eines Bildes ist diesem die Information über die x und y-Position der Kachel, sowie die Revision anzuhängen, damit es der Client richtig zuordnen kann.

```
1 <command>
2         <requestmap>
3                 <x>5038</x>
4                 <y>33575</y>
5                 <rev>5</rev>
6         </requestmap>
7 </command>
```

Listing 3.10: Anforderung eines Kartenstücks an den Server

Die Anforderung der Kacheln funktioniert wie jeder andere Befehl auch. Wie man im Listing 3.10 sieht, übermittelt der Client um ein Bild anzufragen lediglich dessen Position und Revision. Diese Anfrage wird vom Server wie in Abbildung 3.6 beschrieben abgearbeitet.

Um das Bild zu versenden erzeugt der Server zunächst auch eine übliche Nachricht als String im XML-Format. Sie beinhaltet die benötigten Daten, damit der Client das Bild richtig zuordnen kann.

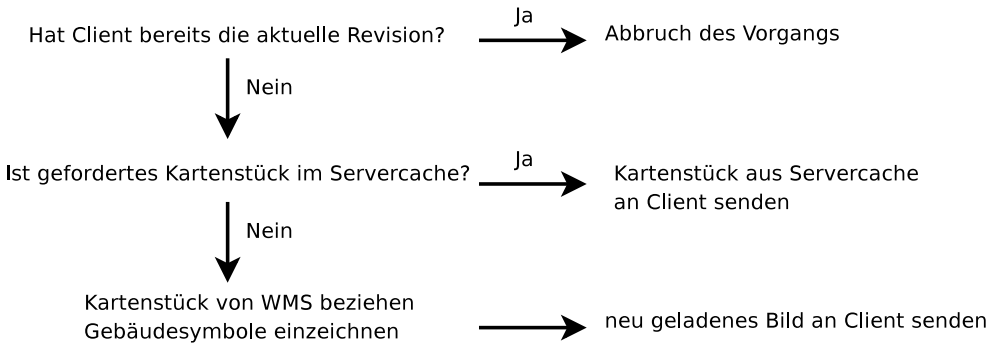

Abbildung 3.6: Bearbeitung einer Kartenanforderung im Server

Dann wird als Separator zwischen Text- und Binärdaten ein `<StartOfImage>`-Tag angehängt. Die Bilddaten werden nun mit Hilfe der Methode `Convert.ToBase64String` Base64 kodiert und können dann ebenfalls an den Nachrichtenstring angehängt werden, da sie nun nur noch als ASCII-Zeichen vorliegen. Schließlich wird die Nachricht mit einem `<EndOfFile>` abgeschlossen (diese abschließende Zeichenfolge erhält jede Nachricht, die Erklärung dazu findet sich unter 3.5.3).

```
1  <command>
2          <map>
3                  <coordinates>
4                          <x>5038</x>
5                          <y>33575</y>
6                          <rev>5</rev>
7                  </coordinates>
8          </map>
9  </command>
10 <StartOfImage>
11 //
12 // Bilddaten als Base64
13 //
14 <EndOfFile>
```

Listing 3.11: Antwort des Server auf eine Kachelanfrage

Empfängt der Client eine `map`-Nachricht, wird diese im Gegensatz zu allen anderen Nachrichten gesondert behandelt.

```
1 if (messageString.StartsWith("<command><map>"))
2 {
3   String[] commandAndMap;
4   commandAndMap = System.Text.RegularExpressions.Regex.Split
        (messageString, "<StartOfImage>");
5
6   messageString = commandAndMap[0];
7   if (commandAndMap.Length > 1 && commandAndMap[1]!="")
8       newImage = byteArrayToImage(Convert.FromBase64String(
            commandAndMap[1]));
9 }
```

Listing 3.12: Trennen der Bild- von den Textdaten beim Client

Der `<StartOfImage>`-Separator wird nun genutzt, um an dieser Stelle Text und Bild von einander zu trennen. Nach dieser Trennung liegen Position und Revision des Bildes wieder wie üblich als XML vor, während das Bild aus den Base64-Daten wieder in ein `Image`-Objekt konvertiert wird und anschließend als `newImage` an anderer Stelle weiter verarbeitet werden kann.

3.5.3 Behandlung von Verbindungsausfällen

Zunächst wird das Versenden von Daten auf Server- und Clientseite erklärt und was dabei zu beachten ist, dann widmet sich der folgende Abschnitt der Empfangsseite.
Für das Versenden der Nachrichten ist die Methode `sendData` zuständig, welche sowohl Client als auch Server in der Klasse `Client` besitzen. Auf Serverseite versucht sie, die Daten aus bytesToSend auf den NetworkStream zu schreiben und bricht bei Fehlern die Verbindung sofort ab. Für das Wiederherstellen der Verbindung ist der Client zuständig.
Tritt also auf Clientseite beim Versuch des Versendens von Daten ein Fehler auf, wird das Wiederaufbauen der Verbindung an die Methode `connectionErrorOccured` delegiert. Sie stoppt zunächst das Empfangen weiterer Daten vom Server und sorgt dafür, dass dem Benutzer der neue Verbindungsstatus angezeigt wird. Außerdem zählt sie, wie viele Fehler bereits aufgetreten sind. Dann versucht `connectionErrorOccured` mit Hilfe von `handleConnectionError` die Verbindung wieder herzustellen. Schlägt dies drei mal fehl, so erhält der Benutzer die Optionen weiter zu versuchen, den Ser-

ver zu kontaktieren, im Offline-Modus weiterzuspielen (hier sind alle Aktionen, die eine Serververbindung benötigen, deaktiviert) oder das Spiel zu beenden. War der Verbindungsaufbau mit `handleConnectionError` erfolgreich, wird das Spiel fortgeführt und der Status im Menü wieder auf „online“ gesetzt.

```
1 private bool handleConnectionError(Byte[] oldCommand)
2 {
3   if (login() && clientConnected)
4   {
5      try
6      {
7       if (oldCommand != null)
8          connection.Write(oldCommand, 0, oldCommand.Length);
9       connectionErrors = 0;
10      return true;
11     }
12     catch (Exception)
13     {
14      receiver.stopReceiver();
15      clientConnected = false;
16      return false;
17     }
18  }
19  else return false;
20 }
```

Listing 3.13: Die Methode handleConnectionError in der Klasse Client

In Zeile 3 von Listing 3.13 sieht man, dass hier der Login erneut gestartet wird. Die Methode `login` ohne Parameter baut im Gegensatz zur normalen Loginmethode, welche Benutzernamen und Passwort verlangt, vor dem Loginversuch eine neue Verbindung zum Server auf und setzt die für die Verbindung benötigten Objekte `tcpclient, connection` und `receiver` des Client-Objekts neu. Den Benutzernamen und das Passwort erhält sie aus den beim ersten Einloggen gespeicherten Membervariablen `name` und `password`. Waren Verbindungsaufbau und Neu-Login erfolgreich, ist ihr Rückgabewert `true`, bei jedem Fehler `false`. Falls die Verbindung nun wieder besteht, versucht `handleConnectionError`, die Daten der zuvor fehlgeschlagenen Übertragung erneut zu senden. Bei Erfolg

wird die Anzahl der Verbindungsfehler wieder zurückgesetzt und `true` zurückgegeben. Tritt dabei wieder ein Fehler auf, wird die neu hergestellte Verbindung komplett geschlossen und der Rückgabewert ist `false`.
Beim Login ist zu beachten, dass bei erfolgreichem Login je nach Logintyp (erster Login/Folgelogin aufgrund von Verbindungsausfall) unterschiedlich verfahren werden muss. Um anzuzeigen, um welchen Typ von Login es sich handelt dient der Parameter `initial` der Methode `doLogin`. Ist er `true`, handelt es sich um einen ersten Login, der vom Loginbildschirm gestartet wurde, ist er `false` handelt es sich um einen durch Netzwerkfehler verursachten, automatisch ausgeführten, Folgelogin.

```
1 if (initial)
2 {
3   this.name = name;
4   this.password = password;
5   notifyClientLoggedIn("OK");
6 }
7 else
8 {
9   ((MainForm)mainForm).initializeTiles(true);
10 }
11 return true;
```

Listing 3.14: Ausschnitt aus doLogin in der Klasse Client bei erfolgreichem Login

In beiden Fällen wird die Verbindung auf aktiv gesetzt. Jedoch werden beim ersten Login `name` und `password` gesichert, um Folgelogins zu ermöglichen und eine Benachrichtigung wird an den Login-Thread gesendet, damit dieser mit der Initialisierung des Spiels fortfahren kann. Im Falle eines Folgelogins muss nur sicher gestellt werden, dass die neun geladenen Umgebungskacheln noch aktuell sind, da sich diese während der Abwesenheit durch Aktivität anderer Spieler geändert haben können. Die Methode `initializeTiles` implementiert diese Funktionalität.

Auf Empfängerseite steht bei Server und Client die Methode `run`, welche als Thread im Hintergrund Daten vom NetworkStream liest. Sie befindet sich beim Client in der Klasse `Receiver`, beim Server dagegen in der Klasse `Client`, da

dieser für jeden verbundenen Client nur ein Objekt halten soll.

```
1  while (receiverRunning)
2  {
3    client.sendData(Encoding.Default.GetBytes("<Alive><
         EndOfFile>"));
4    try {
5      receiveCount = connection.Read(receiveBuffer, 0,
           receiveBuffer.Length);
6    }
7    catch (Exception)
8    {
9      if (receiverRunning)
10         client.sendData(Encoding.Default.GetBytes("<Alive><
               EndOfFile>"));
11   }
12
13   if (receiveCount != 0)
14   {
15    message += Encoding.Default.GetString(receiveBuffer, 0,
          receiveCount);
16    receivedCommands = System.Text.RegularExpressions.Regex.
          Split(message, "<EndOfFile>");
17    for (int i = 0; i < (receivedCommands.Length - 1); i++)
18       processMessage(receivedCommands[i]);
19    message = receivedCommands[receivedCommands.Length - 1];
20   }
21 }
```

Listing 3.15: Ausschnitt aus der Methode run in der Klasse Receiver

Listing 3.15 zeigt die Schleife zum Empfang der Daten. Bevor versucht wird vom Stream zu lesen, wird eine `<alive>`-Nachricht gesendet, um zu prüfen, ob die Verbindung überhaupt noch besteht. Diese Nachrichten werden von Client und Server ignoriert und dienen nur zur Prüfung des Status des Netzwerks.
Dann wird versucht, Daten zu lesen. Bei einem hier auftretenden Fehler wird einfach nochmals `sendData` aufgerufen, da diese Methode sich um den aufgetretenden Fehler kümmern wird.
Die empfangenen Daten werden zunächst wieder in einen String konvertiert. Durch das Setzen des `<EndOfFile>`-Tags am Ende jeder Nachricht können jetzt

hier die einzelnen Nachrichten eindeutig von einander getrennt und rekonstruiert werden. Die Empfangenen XML-Nachrichten werden dann durch `processMessage` verarbeitet.
Da möglicherweise noch eine unvollständig empfangene Nachricht im Empfangspuffer gewesen sein könnte, wird der noch nicht verarbeitete Rest behalten und vor die nächsten empfangenen Daten eingefügt. So ist sichergestellt, dass nur vollständige XML-Befehle ankommen können und bei der weiteren Verarbeitung keine Fehler durch fehlerhafte Nachrichten entstehen.

Es kann vorkommen, dass Netzwerkverbindungen zu Clients unterbrechen und der Server davon zunächst nichts merkt. Daher wird, um dies einzuschränken, auch die Technik `<alive>`-Nachrichten zu senden genutzt. Jedes `Client`-Objekt im Server besitzt einen `aliveTimer`. Dieser ruft alle 30 Sekunden die Methode `checkAlive` auf, welche versucht, ein `<alive>` an den Client zu senden. Tritt dort ein Fehler auf, weiß der Server, dass dieser Client nicht mehr verbunden ist, und kann ihn aus der Clientliste entfernen.

3.6 Datenbank

Dieser Abschnitt geht auf den Zugriff und die Speicherung der Spieldaten in einer PostgreSQL[4]-Datenbank ein, welche um die Erweiterung PostGIS ergänzt wurde, um die Speicherung geographischer Objekte zu ermöglichen. Diese Funktionalität wird vom Prototypen benötigt, um die auf reale Orthofotos gesetzten Spielergebäude der Datenbank hinzufügen zu können. Abschließend wird ein Überblick über die weiteren in der Datenbank abgelegten Spielerdaten gegeben. Sämtliche Datenbankzugriffe werden von der Klasse `SqlAccess` des Servers bereitgestellt.

3.6.1 Datenbankzugriff mit Npgsql

Die Klassenbibliothek Npgsql ermöglicht den Zugriff auf PostgreSQL-Datenbanken aus .NET Anwendungen heraus. Zunächst ist es erforderlich, eine Verbindung zur Datenbank herzustellen. Dazu wird ein Objekt der Klasse `NpgsqlConnection` erzeugt, welchem im Konstruktor ein Verbindungsstring bestehend aus Hostname, Port, Datenbankname, Benutzername und Passwort

[4]http://www.postgresql.org

zu übergeben ist. Anschließend kann mit der Methode `Open` die Verbindung geöffnet werden. Da es häufig vorkam, dass die Verbindung zur Datenbank nach einem Timeout geschlossen wurde, während keine Aktivitäten stattfanden, und die ständige Aufrechterhaltung der Verbindung durch Erhöhung der Timeouts wenig sinnvoll ist, falls keine Clients verbunden sind, implementiert die Klasse `SqlAccess` die Methode `reconnect`. Sie wird bei Ausnahmen, die während der Ausführung einer Methode mit Datenbankzugriff auftreten, aufgerufen und versucht, die Verbindung zur Datenbank wiederherzustellen. War dieser Wiederherstellungsversuch erfolgreich, gibt sie `true` zurück, wodurch die aufrufende Methode erneut versucht, ihre SQL-Abfrage durchzuführen. Ist der Rückgabewert von `reconnect false`, gibt die aufrufende Datenbankmethode an Stelle des Ergebnisses der SQL-Abfrage einen Fehlerwert zurück. In jedem Fall erfolgt der Eintrag einer Fehlermeldung in die Logdatei des Servers, falls die Verbindung zur Datenbank unterbricht.
Die SQL-Anfragen werden dann über die Klasse `NpgsqlCommand` realisiert. Sie erwartet im Konstruktor die Angabe der gewünschten SQL-Anfrage als String, sowie der zu verwendenden Datenbankverbindung als Instanz von `NpgsqlConnection`.

```
1  NpgsqlCommand command =
2           new NpgsqlCommand("SELECT playerid FROM "
3                             + playersTableName
4                             + " WHERE nickname='"
5                             + nick + "';", conn);
6
7  NpgsqlDataReader sqlReader = command.ExecuteReader();
8
9  int playerId = -1;
10 if (sqlReader.Read())
11         playerId = (int)sqlReader[0];
12 sqlReader.Close();
```

Listing 3.16: Beispiel Datenbankabfrage mit Npgsql

Das Beispiel führt eine Anfrage nach der ID des Spielers mit Nickname `nick` durch. Nachdem das `NpgsqlCommand` erzeugt wurde, kann es mit der Methode `ExecuteReader` ausgeführt werden. Die Ergebnisse der Abfrage finden sich dann im von `ExecuteReader` zurückgegebenen `NpgsqlDataReader`-Objekt,

welches durch die `Read`-Methode jeweils eine Zeile der Ergebnisdaten einliest. Auf die einzelnen Spalten kann dann wie auf ein Array über einen Index zugegriffen werden. Anschließend sollte der `NpgsqlDataReader` geschlossen werden.
Die Methode `ExecuteReader` eignet sich nur für Aufrufe, die Daten zurückliefern. Bei SQL-Statements wie `DELETE` oder `INSERT`, die keine Daten zurückgeben, sollte die Ausführung des `NpgsqlCommand` über die Methode `ExecuteNonQuery` erfolgen.

3.6.2 Übersicht PostGIS

PostGIS[5] ist eine Erweiterung für das weit verbreitete Open-Source Datenbanksystem PostgreSQL, die es ermöglicht, neben den üblichen Datentypen auch geographische Objekte einer Datenbank hinzuzufügen. Um in einer Tabelle geographische Objekte abzulegen, muss zunächst eine Geometriespalte angelegt werden. Es können in einer Tabelle, die geometrische Objekte beinhaltet, beliebig viele weitere Spalten verschiedener Datentypen enthalten sein, jedoch ist nur eine Geometriespalte erlaubt. Zur Erstellung dient die PostGIS-Funktion `AddGeometryColumn`. Sie hat folgende Syntax:

```
AddGeometryColumn(<Name der Tabelle>,
                  <Name der Spalte>,
                  <SRID>, <Geometrietyp>, <Dimension>)
```

Listing 3.17: Syntax der PostGIS-Funktion AddGeometryColumn

Dazu ein Beispiel:

```
SELECT AddGeometryColumn('buildings',
                         'factories',
                          4326, 'POINT', 2);
```

Listing 3.18: Beispiel: Erstellung einer Geometriespalte

Dieses Beispiel erzeugt in der Tabelle `buildings` eine neue Geometriespalte `factories`. Sie soll geographische Objekte im EPSG:4326-Koordinatensystem (geographische Länge/Breite) beinhalten, die in Form von Punkten (`POINT`) dargestellt sind. Der Parameter 2 gibt an, dass es sich um 2-dimensionale Punkte handelt.

[5] http://postgis.refractions.net/

Neben `POINT` gibt es die Objekte `LINESTRING`, `POLYGON`, `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON` und `GEOMETRYCOLLECTION`. Die Repräsentation dieser Objekte erfolgt über das Well Known Text-Format (WKT)[6], welches die einfache Eingabe in Textform ermöglicht.

```
INSERT INTO buildings
VALUES(GeometryFromText('POINT(7.6 50.5)', 4326));
```

Listing 3.19: Beispiel: Hinzufügen eines POINT in WKT

Im Beispiel wird der Tabelle `buildings`, welche nur eine Geometriespalte beinhaltet, ein Punkt an der Position 7,6° östlicher Länge und 50,5° nördlicher Breite hinzugefügt.
Zur Abfrage der Daten gibt es verschiedene Möglichkeiten. Beispielsweise lassen sich mit `X(Spaltenname)` oder `Y(Spaltenname)` einzelne Koordinaten abfragen. `AsText(Spaltename)` liefert wiederum die Darstellung der Geometrie im WKT-Format.

```
lbspiel=# SELECT AsText(factories) FROM buildings;
Antwort:
     astext
-----------------
 POINT(50.5 7.5)
```

Listing 3.20: Beispiel: Abfrage von Geometriedaten

Post-GIS bietet viele weitere Funktionen zur Arbeit mit geographischen Objekten [Res21]. Wichtig für den Prototypen sind die Funktionen `Within` bzw. `Contains`, die im nächsten Abschnitt näher betrachtet werden.

3.6.3 Gebäude

Die räumlichen Erweiterungen durch PostGIS werden vom Prototypen zur Speicherung der Lage der einzelnen Spielergebäude genutzt. Wählt ein Spieler die Funktion zum Bau eines Gebäudes, so werden dem Server die Position, die Anzahl der verfügbaren Ressourcen an dieser Position und der Typ der gewünschten zu fördernden Ressource übermittelt. Die verfügbaren Ressourcen müssen mitgeliefert werden, da z.B. durch Rundungsfehler die Berechnung der Ressourcen aus

[6] http://www.giswiki.org/index.php/WKT

den Koordinaten auf dem Server schon andere Ergebnisse hervor bringen könnte, als sie beim Client angezeigt wurden (siehe 3.7.4). Dann folgen verschiedene Prüfungen, die feststellen sollen, ob ein Bau möglich ist:

1. Festellen, ob ein freigeschalteter oder ein Testaccount vorliegt.
Ist der Account freigeschaltet, wird fortgefahren.
Andernfalls muss geprüft werden, ob die gewählte Aktion noch innerhalb seines erlaubten Aktionsrahmens stattgefunden hat. Dazu dient die Methode `checkActionRadius`. Sie liest zunächst den in der Datenbank gespeicherten Startpunkt des Spielers ein und vergleicht dann, ob dieser Punkt innerhalb des Aktionsrahmens liegt.

```
1 startX -= server.UnpaidRectX / 2.0;
2 startY -= server.UnpaidRectY / 2.0;
3
4 command = new NpgsqlCommand("SELECT Within(
5           GeometryFromText('Point(x y)',4326),
6           GeometryFromText('Polygon((startX startY,
7                 (startX + server.UnpaidRectX) startY,
8                 (startX + server.UnpaidRectX) (startY +
                     server.UnpaidRectY),
9                  startX (startY + server.UnpaidRectY),
10                 startX startY))', 4326));", conn);
11 sqlReader = command.ExecuteReader();
12
13 bool actionOk = false;
14 if (sqlReader.Read())
15         actionOk = (bool)sqlReader[0];
```

Listing 3.21: Ausschnitt aus checkActionRadius (Code gekürzt)

Die Variablen `startX` und `startY` beinhalten die Startposition des Accounts. In `server.UnpaidRectX` bzw `server.UnpaidRectY` wurde zuvor die Größe des Rechtecks eingegeben, in dem Aktionen für nicht freigeschaltete Accounts stattfinden dürfen. Um ein Rechteck der Größe `UnpaidRectX*UnpaidRectY` um die Startposition herum zu bilden, wird zunächst jeweils die Hälfte der beiden Unpaid-Werte von den Startkoordinaten abgezogen. Dann kann mit Hilfe der `Within`-Funktion geprüft werden, ob die Position, an der die Aktion (hier: das Bauen eines Gebäudes) durchgeführt werden soll, innerhalb des Handlungsrah-

mens liegt. Ihr erster Parameter (Zeile 5 des Listings 3.21) bezeichnet das Objekt, für welches geprüft werden soll, ob es innerhalb des Objekts im zweiten Parameter (Zeilen 6-10) liegt. Hier wird also geprüft, ob der Aktionspunkt mit den zuvor übergebenen Koordinaten `(x y)` innerhalb des durch ein Polygon dargestellten, erlaubten Aktionsrechtecks liegt. Die Rückgabe ist `true`, falls der Punkt im Rechteck enthalten ist, `false`, falls nicht.
Alternativ kann die Methode `Contains` verwendet werden, die genau umgekehrt funktioniert.

2. Prüfen, ob die Position oder ihre Umgebung bereits von einem anderen Gebäude bebaut wurden.
Dazu wird die Methode `checkBuildingPosition` benutzt, welche mit Hilfe von `getOccupiedPoints` bestimmt, ob auf der Bauposition bereits andere Gebäude stehen.

```
1 public Building[] getOccupiedPoints(double lon,
2                                      double lat,
3                                      double rectX,
4                                      double rectY)
```

Listing 3.22: Signatur der Methode getOccupiedPoints

In `lon` und `lat` wird die Angabe eines Punktes in geographischer Länge/Breite erwartet. `rectX` und `rectY` bezeichnen das von diesem Punkt aus aufzuspannende Reckteck, in dem geprüft werden soll, welche Gebäude dort gebaut sind. Die Methode arbeitet nach dem selben Prinzip wie `checkActionRadius`, indem sie mit `Within` überprüft, welche Punkte im angegebenen Rechteck enthalten sind. Anstatt nur einen booleschen Wert zu liefern, der angibt, ob dort ein Punkt vorhanden ist, beinhaltet ihr Rückgabewert sämtliche sich dort befindenden Gebäude (Punkte) mit allen Gebäudeparametern, da die Methode auch zur Einzeichnung der Gebäude auf den Kacheln genutzt wird.
Sind bei dem in `checkBuildingPosition` gestarteten Aufruf von `getOccupiedPoints` keine Gebäude zurückgegeben worden, kann davon ausgegangen werden, dass dort gebaut werden kann.

3. Prüfen, ob der Spieler genügend Ressourcen zum Bau des Gebäudes hat.
Hier findet eine einfache Datenbankanfrage nach dem Ressourcenbesitz des Spielers über die Methode `canAfford` statt.

Sind alle drei Bedingungen erfüllt, kann das eigentliche „Bauen“ des Gebäudes erfolgen.
Zunächst bestimmt der Server eine freie Gebäude-ID. Um keine Lücken durch gelöschte Gebäude, Nachrichten, Handels oder Spieler-IDs aufkommen zu lassen, hält der Server immer eine Liste aller freien, in Lücken liegenden IDs für jeden ID-Typ. Beim Starten des Servers müssen einige dieser Listen zunächst erzeugt werden, da alle IDs nur in der Datenbank gehalten und nicht auf dem Server gespeichert werden. Nachdem beispielsweise die Liste aller Gebäude-IDs aus der Datenbank gelesen wurde, analysiert der Server sie und trägt alle in Lücken liegenden IDs in die `ArrayList freeBuildingIds` ein. Wird ein Gebäude gelöscht, wird dessen ID der Liste hinzugefügt.
Ist in dieser Liste eine freie ID gespeichert, so wird sie für das neue Gebäude benutzt, andernfalls wird die maximale ID aller Gebäude + 1 als neue ID gewählt.
Dann folgt über die Methode `addBuilding` der eigentliche Eintrag des Gebäudes in die Datenbank.

```
1 public bool addBuilding(double x,
2                          double y,
3                          int playerId,
4                          int[] buildingResources,
5                          String resourceType,
6                          int buildingId)
```

Listing 3.23: Signatur der Methode addBuilding

`x` und `y` bezeichnen die Position des Gebäudes in geographischer Länge/Breite. `playerId` stellt den Besitzer des Gebäudes dar, `buildingResources` gibt an, welche Ressourcen das Gebäude produzieren kann. Die Speicherung aller Ressourcen ist notwendig, um spätere Upgrades des Gebäudes zu ermöglichen. Wichtig ist auch die Angabe von `resourceType`, da dadurch zum einen die zu produzierende Ressource, als auch das Aussehen des Gebäudes gesetzt werden. Das Aussehen eines Gebäudes hängt immer vom beim Bau gewählten Rohstoff ab und ändert sich grundlegend nicht. Es können lediglich weitere Produktionsicons hinzukommen.

Insgesamt erfordert das Bauen eines Gebäudes Eintragungen in drei Tabellen:

1. **buildings**
   - Spieler-ID
   - Gebäude-ID
   - Baukoordinaten als `POINT`
   - aktuelle Goldproduktion
   - aktuelle Ölproduktion
   - aktuelle Forschungsproduktion
   - aktuelle Stahlproduktion
   - Typ bzw. Aussehen (Gold/Öl/Forschung/Stahl)

   Die Tabelle `buildings` ist die Haupttabelle für einen Gebäudeeintrag und beinhaltet alle aktuellen Produktionswerte sowie Position, Besitzer und das zu zeichnende Icon für ein Gebäude.

2. **buildingpotential**
   - Gebäude-ID
   - mögliche Goldproduktion
   - mögliche Ölproduktion
   - mögliche Forschungsproduktion
   - mögliche Stahlproduktion

   In `buildingpotential` wird festgehalten, welche Fördermengen mit einem Gebäude möglich sind. Sie werden beim Bau des Gebäudes anhand der auf dem Client beim Bau berechneten Werte gesetzt.

3. **buildingupgrades**
   - Gebäude-ID
   - produziert Gold? (Boolean)
   - produziert Öl? (Boolean)
   - produziert Forschung? (Boolean)
   - produziert Stahl? (Boolean)
   - Upgradelevel Ressourcenanzahl
   - Upgradelevel Produktionsmenge
   - Upgradelevel Verteidigung

`buildingupgrades` zeigt für ein Gebäude an, um welche Eigenschaften es bereits aufgewertet wurde. Beim Bau eines Gebäudes sind alle drei Upgradelevels auf Stufe 1 und nur einer der Ressourcenparameter ist `true`.

Nach dem Eintragen eines neuen Gebäudes muss die Menge der pro Spieltag gewonnenen Ressourcen des Spielers durch die Methode `updateResourceGain` angepasst werden. Sie aktualisiert die Tabelle `resourcegain`, indem sie die Produktion des neuen Gebäudes den täglich gewonnenen Ressourcen hinzufügt. Nach erfolgreichem Bau werden dem Spieler die Baukosten mit Hilfe der Methode `subtractResources` abgezogen und der Spieler wird über den Status des Bauvorgangs informiert. Abschließend ist es notwendig, alle Clients über das neue Gebäude zu informieren, damit sie ggf. ihr Kartenmaterial aktualisieren können um die Änderung auf der Karte sichtbar zu machen (siehe 3.4.5).

Besonderheiten beim Löschen und Upgrade von Gebäuden:

- Das Entfernen eines Gebäudes erfordert ein Neuzeichnen der Kachel, auf der das Gebäude stand. Daher muss beim Löschen eines Gebäudes beachtet werden, dass die Koordinaten dieser Kachel vor der Löschung gespeichert werden, um nachher diese Kachel neu laden zu können.
- In die Liste der freien Gebäude-IDs des Servers muss nach der Löschung ein Eintrag erfolgen.
- Das Upgrade eines Gebäudes erfordert eine weitere Prüfung über den Upgradelevel des Gebäudes. Jedes Gebäude darf maximal drei mal aufgewertet werden.

Einzeichnen der Spielgebäude

Fordert ein Spieler ein Kartenstück an, so wird es, wie unter 3.5.2.3 beschrieben, entweder aus dem internen Cache oder vom WMS bezogen. Ist eine Kachel noch nicht vorhanden oder hat sie sich durch Neubau, Entfernung oder Upgrade eines Gebäudes geändert, müssen auf das vom WMS gelieferte Rohbild vor dem Versenden die auf dieser Kachel gebauten Spielgebäude eingezeichnet werden. Diese Aufgabe übernimmt die Methode `paintBuildingIcons` der Klasse `WMSMap`. Sie bezieht mit Hilfe von `getOccupiedPoints` alle im Bereich der gewählten Kachel gesetzten Gebäude aus der Datenbank. Zu den dabei beschafften Gebäudeinformationen gehören der Standort, die produzierten Ressourcen, das Aussehen, der Besitzer und die ID des jeweiligen Gebäudes. Dann können anhand die-

ser Daten die den einzelnen Gebäuden entsprechenden Icons, sowie Besitzer und Gebäude-ID an der passenden Position auf das Kartenstück gezeichnet werden. Falls ein Gebäude durch seine Größe über den Kachelrand hinausragen würde, wird dessen Zeichenposition angepasst, so dass es nicht über mehrere Kartenstücke verteilt gezeichnet wird. Daraufhin ist das Bild vollständig vorbereitet und kann dem anfordernden Client zugestellt werden.

3.6.4 Spielerdaten

Neben den Informationen über die Spielgebäude werden in der Datenbank alle weiteren wichtigen Spielerdaten gehalten.

Tabelle accounts

- Spieler-ID
- Passwort
- Kontaktinformationen: Name, Vorname, E-Mail
- Status der Freischaltung
- Abrechnungsinformationen

Tabelle messages

- Nachrichten-ID
- Absender
- Empfänger
- Datum/Uhrzeit
- Status (gelesen/ungelesen)
- Nachrichtentext

Tabelle players

- Spieler-ID und Nickname
- Kostenfaktoren für Gold, Öl, Forschung, Stahl
- Angriffs-, Spionagewert
- Startposition (Position des ersten Gebäudes)

Tabellen resourcegain und totalresources

`Resourcegain` beinhaltet den Gold-, Öl-, Forschungs- und Stahlwert jedes Spielers, der pro Spieltag vergeben wird, während in `totalresources` der Gesamtbesitz eines Spielers gespeichert wird.

3.7 Spielelogik

Im Abschnitt Spielelogik geht es um die Umsetzung und Funktionsweise der Angriffs- und Einnahmefunktion, sowie die Auswirkungen der Charaktereigenschaften auf das Spiel. Der Unterpunkt 3.7.4 erklärt, wie aus der Position des Spielers eine Ressourcenverteilung bestimmt wird.

3.7.1 Angriff und Einnahme von Gebäuden

Um ein Gebäude angreifen zu können, muss es innerhalb der Spionagereichweite, die durch den Spionagewert des Spielers bestimmt wird, liegen. Für jede Spionagestufe erhöht sich die Entfernung, in der angegriffen werden kann, um 100 Pixel. Ein Spieler der Stufe 5 kann also beispielsweise im Umfeld von 500x500 Pixeln alle Gebäude angreifen.
Außerdem bestimmt die Spionagestufe die Menge an Details, die ein Spieler über die Gebäude, die angegriffen werden können, erfährt:

- Stufe 1: Gebäude-ID, Besitzer, Besitzer-ID
- Stufe 2-3: zusätzlich Menge der Gold- und Ölproduktion der Gebäude
- Stufe 4-5: zusätzlich Menge der Forschungs- und Stahlproduktion der Gebäude
- ab Stufe 6: zusätzlich Verteidigungsstufe der Gebäude

Wenn ein Spieler den Angriff oder die Einnahme eines Gebäudes wählt, führt der Server die Aktion als `FightCommand` aus. Hier finden die Unterscheidung zwischen Angriff oder Einname sowie alle nötigen Berechnungen statt. Die Entscheidung über Sieg oder Verlust wird dabei aufgrund von Wahrscheinlichkeiten bestimmt, die durch die Werte Gebäudeverteidigung und Spielerangriffswert festgelegt werden. Auf dem Server wird dazu eine Zufallszahl zwischen 0 und 1 bestimmt und für den Spieler wird aus den genannten Werten ein Angriffswert berechnet. Ist dieser Angriffswert höher oder gleich der zufällig gewählten Zahl, so gewinnt der Spieler. Damit steigt mit dem Angriffswert auch die Wahrscheinlichkeit für einen Sieg.

Berechnung des Angriffswerts

$$\frac{\text{Angriffsstufe des Spielers}}{\text{Verteidigungsstufe des Gebäudes} * 10}$$

Somit liegt für einen Spieler der Angriffstufe 1 beim Angriff auf ein Gebäude der Stufe 1 die Wahrscheinlichkeit zum Sieg bei 10%. Für einen Spieler der Stufe 10 ist dagegen der Sieg beim Angriff auf ein Gebäude der Stufe 1 garantiert, da die Wahrscheinlichkeit hier bei 100% liegt.
Von der Angriffstufe des Spielers hängt auch die Menge der geraubten Ressourcen ab. Pro Stufe des Spielers werden dem Besitzer des Gebäudes 100 Punkte jeder Ressource, die das Gebäude produziert, abgezogen. Besitzt der beraubte Spieler nicht genügend Ressourcen, so wird der höchstmögliche Wert auf den Angreifer übertragen.

Berechnung des Einnahmewerts

$$\frac{\text{Angriffsstufe des Spielers}}{\text{Verteidigungsstufe des Gebäudes} * 2 * 10}$$

Wie man sieht ist die Wahrscheinlichkeit ein Gebäude einzunehmen geringer und auch bei höchster Angriffsstufe nicht immer erfolgreich.

Schlägt ein Angriffs- oder Einnahmeversuch fehl, so nimmt der Angreifer Schaden in Form von Ressourcenabzug. Beim normalen Angriff werden Forschungspunkte abgezogen, ein misslungener Einnahmeversuch bewirkt zusätzlich zum Forschungsverlust den Abzug von Goldpunkten. Dieser Schaden ist abhängig von der Stufe des angreifenden Spielers. Auf Angriffsstufe 1 umfasst der Schaden zunächst nur den auf dem Server eingestellten Standardkostenwert. Alle zwei Angriffsstufen verdoppelt sich dieser Wert. Entgegenkommen kann man dieser Erhöhung durch Aufbessern der Kostenstufen des Spielers. Weiteres dazu erklärt der folgende Abschnitt.

3.7.2 Kostenfaktoren

Zu Beginn des Spiels hängt der Ressourcenbedarf noch vollständig von den Basiswerten des Servers ab. Im Laufe des Spiels erhöhen sich die Kosten für alle Ressourcen dann durch Steigerung der Gebäudeanzahl, der Upgrades oder der

Charakterstufen des Spielers. Über die Kostenstufen lassen sich die Ausgaben für sämtliche ressourcenbenötigende Spielaktionen senken, indem der Verbrauch der vier Ressourcen Gold, Öl, Forschung und Stahl verringert wird. Für jede Stufe (außer der Basisstufe 1) eines Kostenfaktors wird der Verbrauch der jeweiligen Ressource um 5% gesenkt.

Beispiel:
Ein Spieler mit 4 Gebäuden hat die Goldkostenstufe 3. Bei einem Basiskostenwert von 50 würde der Bau eines fünften Gebäudes 100 Goldpunkte kosten, da sich die Kosten für ein Gebäude nach jeweils 4 Gebäuden verdoppeln. Durch die Goldkostenstufe 3 erhält er jedoch 2 * 5% = 10% Vergünstigung auf den Bau (die erste Stufe ist bereits zu Beginn des Spiels vorhanden und wird nicht mitgezählt). Dadurch erfordert der Bau des fünften Gebäudes für diesen Spieler nur noch 90 Goldpunkte.

3.7.3 Ölpunkte

Damit alle Gebäude pro Spieltag Ressourcen fördern, ist es notwendig genügend Ölpunkte zu besitzen. Am Ende eines Spieltages wird serverseitig für alle Spieler geprüft, wie viele Gebäude und Gebäudeupgrades sie besitzen, um daraus den Ölverbrauch zu bestimmen. Pro Gebäude werden die eingestellten Basiskosten für Öl benötigt, während jedes Gebäudeupgrade die Hälfte der Basiskosten in Anspruch nimmt. Die so berechneten Kosten werden dann zunächst abgezogen, bevor alle weiteren, durch Gebäude neu verdiente Ressourcenpunkte, addiert werden.
Falls ein Spieler nicht genügend Öl besitzt, findet keine Produktion durch seine Gebäude statt. Er erhält jedoch einen Ölpunkt pro Tag, um einem vorzeitigen Spielende durch Ölmangel vorzubeugen.

3.7.4 Ressourcenberechnung

Die Berechnung der möglichen zu fördernden Ressourcen erfolgt auf Clientseite, da sie vollständig von den geographischen Koordinaten der Spielerposition abhängt. Sie erfolgt in der Methode `calculateResources` der Klasse `ResPanel`, welche für die Darstellung der oberen Statusleiste des Clients zuständig ist (siehe 3.8.2.1).

Bevor mit den Werten der geographischen Länge und Breite gerechnet werden kann, müssen sie zunächst grob abgerundet werden, da das GPS-Gerät kontinuierlich neue Positionen übermittelt, welche sich nur geringfügig unterscheiden. Dies hätte zur Folge, dass einmal gefundene, vorteilhafte Positionen nur schwierig wieder auffindbar wären, da man exakt auf der selben Position stehen müsste. Durch das Abrunden vergrößert sich der Bereich, in dem die gleiche Ressourcenverteilung angezeigt wird, um wenige Meter.
Um aus den zwei gegebenen Zahlen der geographischen Breite und Länge vier eindeutige Ressourcenwerte zu erzeugen, bedient sich der Prototyp der MD5-Hashfunktion. Ihre Berechnung ist im .NET Framework über die Klasse `MD5CryptoServiceProvider` verfügbar. Außerdem erzeugt sie 16 Bytes lange Hashwerte, die sich gut zur Zerlegung in vier Einzelwerte eignen. Zunächst werden die geographischen Koordinaten in Bytearrays konvertiert, dann werden diese Bytearrays aneinander gehängt, um einen einzigen Parameter für die MD5-Funktion zu erhalten. Das Ergebnis der Hashfunktion wird anschließend in vier einzelne Bytearrays zerlegt, welche in positive Integerwerte konvertiert werden. Die sich daraus ergebenden Zahlen sind noch sehr groß und insgesamt sollen pro Position maximal 10 Ressourcenpunkte zur Verfügung stehen. Daher wird zunächst der prozentuale Anteil jeder Zahl an der Gesamtsumme der vier Werte bestimmt, um schließlich die 10 Ressourcenpunkte anteilsmäßig auf die vier Ressourcen verteilen zu können.
Dadurch wird erreicht, dass jede Position eine eindeutige Ressourcenverteilung besitzt, die auch reproduzierbar ist. Außerdem wird durch die Verwendung der Hashfunktion gewährleistet, dass sich nah beieinander liegende Positionen trotz ihrer ähnlichen Werte in der Menge und Verteilung der Spielressourcen sehr voneinander unterscheiden können.

3.8 Benutzerschnittstelle

3.8.1 Server

Die Serveranwendung des Prototypen ermöglicht zum einen die Protokollierung und Überwachung der Abläufe im Spiel und zum anderen die Einstellung wichtiger technischer, als auch spielrelevanter Parameter. Die eingestellte Konfiguration wird in der Datei `mobico_server_settings.xml` im Verzeichnis der Anwendung gespeichert.

3.8.1.1 Hauptfenster

Die drei Hauptfenster der Anwendung dienen zur Protokollierung der Aktionen des Servers.

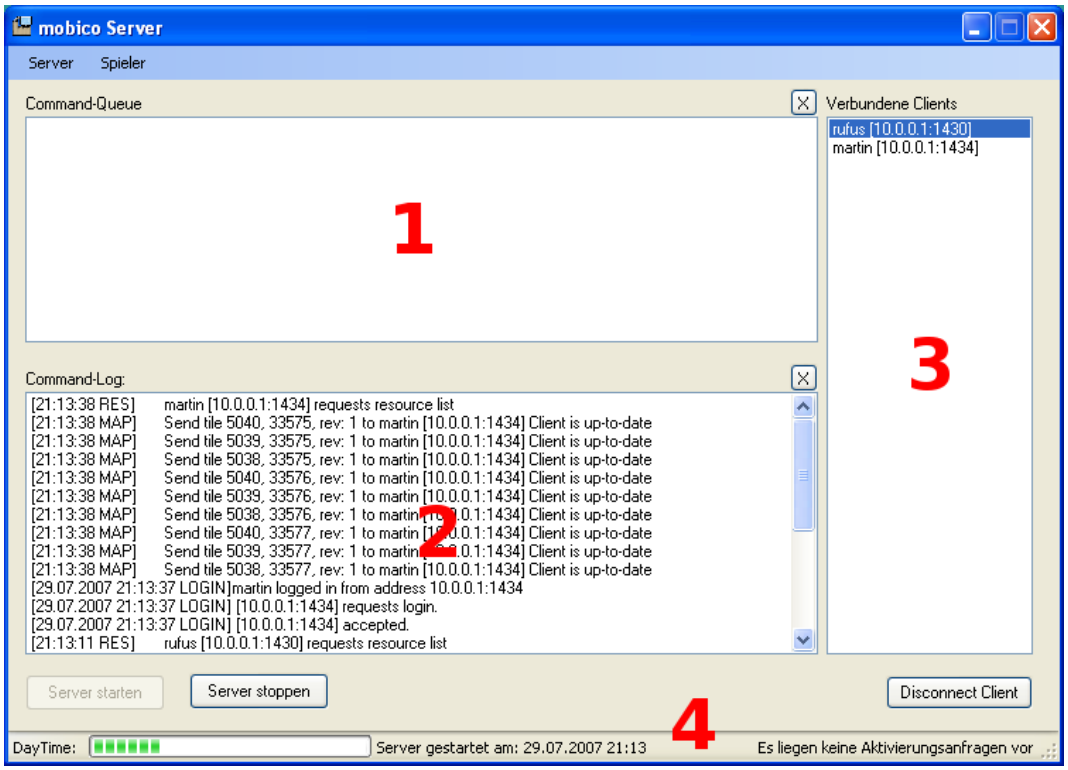

Abbildung 3.7: Screenshot der Serveranwendung

In Liste 1 werden alle Befehle angezeigt, die noch zur Bearbeitung durch das Serverprogramm anstehen. Auf dem Screenshot 3.7 ist diese Liste leer, der Server hat also aktuell keine Anfragen von Clients zu verarbeiten. Nachdem eine Aufgabe vollständig abgearbeitet wurde, wird sie im Command-Log (Liste 2) eingetragen.

Zusätzlich zum Text der eingegangenen Anfrage werden einige erledigte Aufgaben außerdem mit einem Status versehen. Auch Fehlermeldungen werden hier dargestellt. Anhand der Kürzel lassen sich die Nachrichten in Kategorien einteilen:

- `RES`: Anforderung Ressourcenliste
- `MAP`: Anfrage nach einem Kartenstück
- `REG`: Registrierung
- `LOGIN/LOUT`: Aus/Einloggen eines Spielers
- `MSG`: Nachrichten-/Chataktion
- `TRA`: Handelsaktionen
- `BUILD`: Gebäudebau/Gebäudeliste
- `COST`: Kostenanfrage
- `UPGR`: Gebäudeaufwertung
- `ATT`: Angriff/Angriffsbenachrichtigung
- `LEV`: Spielerwerte/Spieleraufwertung
- `INFO`: Statistiken/Accountdetails

In Liste 3 werden alle zur Zeit verbundenen Clients mit Nickname, IP-Adresse und verbundenem Port angezeigt. Über die Schaltfläche unter der Client-Liste lassen sich Clients zu Testzwecken trennen. Über die Schaltflächen *Server starten* und *Server stoppen* besteht die Möglichkeit, den Server vorübergehend in seiner Arbeit anzuhalten. Dadurch werden alle Clients getrennt. Eine Neuverbindung ist nicht möglich, so lange der Server gestoppt ist, da die Listener-Funktionalität des Servers ebenfalls unterbrochen wird.
Die Statusleiste (4) am unteren Rand der Bedienoberfläche zeigt in der *DayTime*-Leiste die aktuelle „Uhrzeit“ des Spielgeschehens an. Hat die Leiste ihren Maximalwert erreicht, ist ein Spieltag vergangen. Damit werden allen Spielern, die gerade eingeloggt sind, Ressourcen nach ihrer aktuellen Gesamtproduktion zugewiesen. Die *DayTime*-Leiste wird anschließend zurückgesetzt. Neben der Anzeige der Spielzeit befindet sich die Anzeige des Zeitpunktes, seit dem der Server online ist. Außerdem gibt die Statusleiste Auskunft über eingegangene Aktivierungsanfragen von Clients.

3.8.1.2 Optionen

Ist eine Aktivierungsanfrage eingegangen, kann sie über den Menüpunkt *Spieler → Aktivieren*, wie in Abbildung 3.8 dargestellt, eingesehen werden.

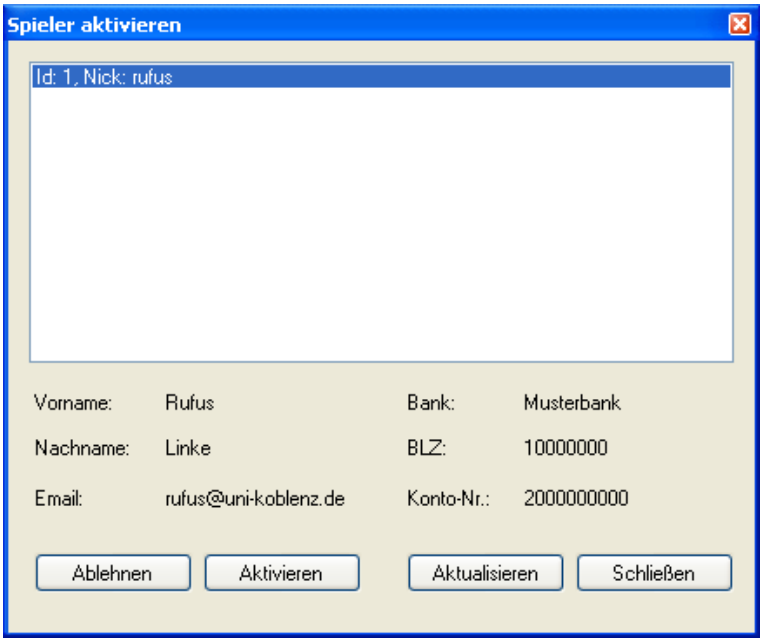

Abbildung 3.8: Aktivierungsfenster des Servers

Der Serverbetreiber erhält durch dieses Fenster Auskunft über alle eingegangenen Aktivierungen, sowie die vom Client übermittelten Angaben. Wurde eine der Optionen *Aktivieren* oder *Ablehnen* gewählt, erhält der Spieler eine Benachrichtigung über den Status seiner Anfrage.
Im Menü *Spieler* befindet sich außerdem eine Kurzübersicht aller registrierten Spieler. Eine Gesamtauflistung aller Informationen über einen Spieler, sowie die Funktion ihn aus der Datenbank zu löschen, sind nach Auswahl eines Spielers über den Button *Details* zugänglich.

Der Menüpunkt *Server* beinhaltet das Einstellungen-Fenster und die Anzeige des auf dem Server gespeicherten Tile-Cache.
Im Fenster *Einstellungen* (Abbildung 3.9) sind alle wichtigen Optionen zur Konfiguration des Servers untergebracht.

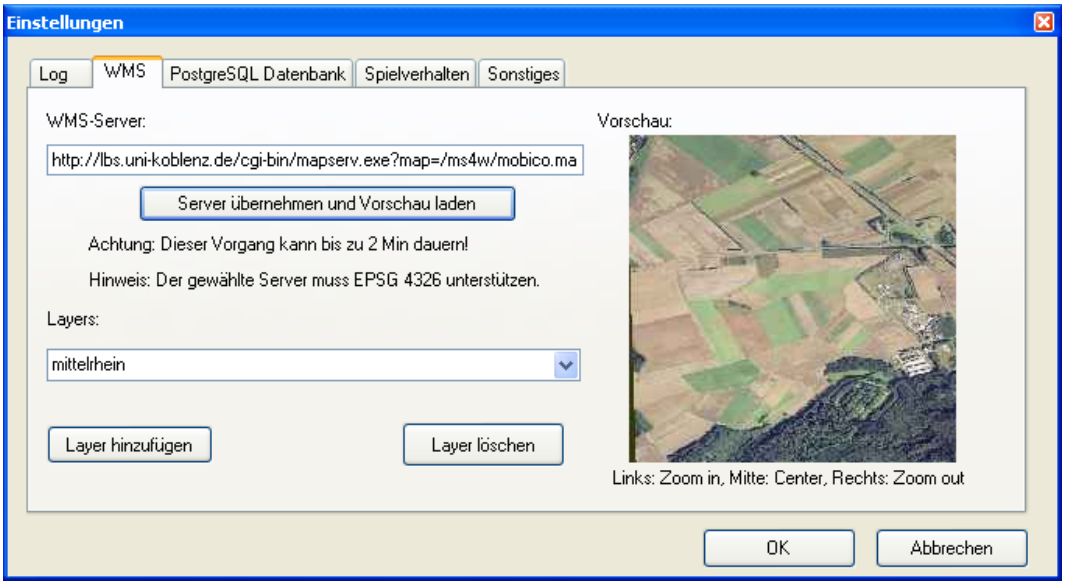

Abbildung 3.9: Einstellungsfenster des Servers

Log

Hier wird eingestellt, welche Aktivitäten der Server protokollieren soll. Die Auswahl von *Fehlermeldungen* führt dazu, dass Fehler wie Verbindungsabbrüche und Fehler beim Zugriff auf WMS in die gewählte Logdatei geschrieben werden. *Spielkommandos* loggt sämtliche Spielaktionen aller Spieler und *Login/Logout-Vorgänge* schreibt die Verbindung und das Trennen von Clients in die gewählte Logdatei. Standardmäßig befindet sich die Logdatei im Verzeichnis der Anwendung.

Die Log-Einstellung betrifft nicht die Aufgabenliste im Hauptfenster des Servers.

WMS

Dieses Menü bietet Einstellungen für den für die Spielkarten zu verwendenden WMS-Server. Dabei ist zu beachten, dass der Prototyp dafür ausgelegt wurde, Koordinaten in geographischer Länge und Breite zu verarbeiten. Der gewählte WMS-Server muss demnach EPSG:4326 unterstützen. Falls der angegebene Server einen sehr großen Bereich abdeckt, kann der Vorgang bis zur Anzeige der Vorschau bis zu zwei Minuten dauern. Danach wird der Vorgang aufgrund eines Timeouts abgebrochen, eine Vorschau ist dann nicht möglich. Über die Layerauswahl können alle später im Spiel sichtbaren Layer gesetzt werden. Weiterhin bietet die Vorschau einfache Navigation über die Maus.

PostgreSQL Datenbank

Bei einer Änderung der Datenbankeinstellungen ist zu beachten, dass die vom Prototypen benötigten Tabellen bereits vorher in der Datenbank erstellt werden müssen.

Spielverhalten

Im Menü Spielverhalten können alle Parameter, die den Spielverlauf beeinflussen gesetzt werden. Die Einstellung *Startkapital* ist nur für neue Benutzer von Bedeutung. *Aktionsradius* kennzeichnet den Bereich, in dem nicht freigeschaltete Spieler von ihrem ersten Gebäude aus Aktionen durchführen können.

Sonstiges

Der Punkt *Sonstiges* erlaubt bisher nur die Einstellung der Kompressionsrate der Bilder, die zum Client gesendet werden. Zwar werden die einzelnen Bilder schon durch den WMS-Server komprimiert, jedoch werden sie vom Server intern als Bitmaps verwaltet. Erst beim Transfer werden die Bilder erneut komprimiert, um Bandbreite auf dem Übertragungsweg zum Client einzusparen.

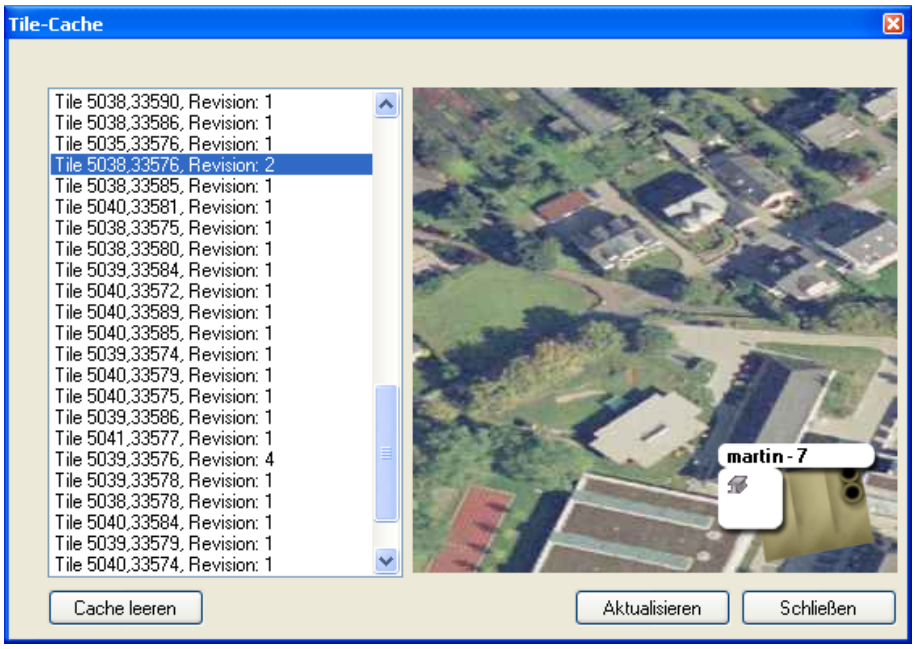

Abbildung 3.10: Tile-Cache des Servers

Im Menüpunkt *Server → TileCache anzeigen* lässt sich ein Überblick über die vom Server bereits geladenen Kacheln gewinnen (Abbildung 3.10). Die Schaltfläche

Cache leeren löscht alle vorgeladenen Bilder vom Server, so dass sie bei Anforderung eines Clients anschließend neu vom WMS-Server bezogen werden. Die Revisionen bleiben dabei erhalten, so dass die Gefahr von Inkonsistenz zwischen den Daten von Server und Client nicht besteht.

3.8.2 Client

Die Benutzerschnittstelle des Clients besteht aus drei Teilen. Am oberen und unteren Rand des Bildschirms befinden sich jeweils Menüleisten. Den Hauptteil des Bildschirms nimmt die Karte mit den Gebäudeinformationen ein.

3.8.2.1 Statusleiste

Die obere Menüleiste (Abbildung 3.11) dient hauptsächlich der Information des Spielers über seinen virtuellen Ressourcenbestand.

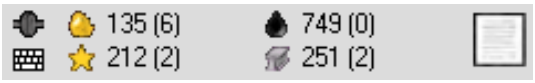

Abbildung 3.11: Die Statusleiste

Je nach Servereinstellung für die Spieltagdauer findet eine Neuberechnung der Ressourcen statt und wird auf allen eingeloggten Clients aktualisiert. In Klammern hinter den Ressourcenzahlen stehen stets die Ressourcenfördermöglichkeiten des aktuellen Standorts.

Abbildung 3.12: Ein Nachrichtenempfang

Weiterhin findet sich in der rechten oberen Ecke ein Symbol zur Anzeige von aktuellen Ereignissen, die den Spieler betreffen. Ein Klick hierauf zeigt dem Nutzer entsprechende Nachrichten (Abbildung 3.12). Diese Ereignisse können unter

anderem den Angriff auf ein eigenes Gebäudes oder ein unterbreites Handelsangebot beinhalten. Außerdem befinden sich in der linken oberen Ecke des Bildschirms zwei weitere Symbole. Das Steckersymbol, dessen Zustände in Abbildung 3.13 zu sehen sind, gibt darüber Auskunft, ob der Spieler momentan mit dem Server verbunden ist. Beim Klick auf das Tastatursymbol öffnet sich die für mobile Endgeräte übliche virtuelle Tastatur.

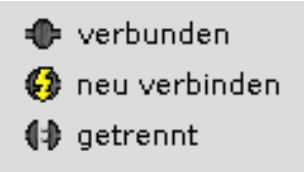

Abbildung 3.13: Zustände der Serververbindung

3.8.2.2 Spielfenster

Das Hauptfenster in Abbildung 3.14 beinhaltet die aktuelle Umgebungskarte mit eventuell vorhandenen Spielgebäuden in der Nähe.

Abbildung 3.14: Der Hauptbildschirm

Die folgende Abbildung 3.15 stellt die möglichen Spielgebäude dar.

Abbildung 3.15: Die Spielgebäude

Der aktuelle Standort des Spielers wird stets in der Mitte des Bildschirms anhand eines weißen Pfeils dargestellt, welcher sich nach der Blickrichtung des Nutzers ausrichtet. Sobald sich der Spieler bewegt, scrollt die Umgebungskarte mit. Alle Interaktionsmöglichkeiten mit dem Spiel finden je nach eingeblendeten Fenstern hier statt. Eine Veränderung des Kartenausschnitts findet nur statt, wenn der Spieler eine Kartenübersicht anfordert, so dass die normale Spielkarte mit einer ausgezoomten Karte der Spielumgebung überlagert wird.

3.8.2.3 Menüleiste

Alle Vorgänge zur Interaktion mit den Spielfunktionen werden in der Menüleiste 3.16 ausgewählt.

Abbildung 3.16: Die Menüleiste

Von links nach rechts beinhaltet dies Funktionen der Kommunikation, des Handels, der Gebäudeinteraktion, der Spielinformationen und der Optionen. Die Kommunikationsmöglichkeiten bestehen aus einem Nachrichtensystem und einem Chatsystem. Der Handel kann durch Angebote anderer Spieler oder durch das

Einstellen eines eigenen Angebots getätigt werden. Die umfangreichsten Möglichkeiten bietet das Baumenü. Hier ist es möglich seine Gebäudeliste abzufragen, eine Fabrik zu bauen oder aufzuwerten und andere Gebäude anzugreifen. Der Angriff wiederum kann ein Diebstahl oder eine Übernahme des Gebäudes sein. Die Spielinformationen bieten eine Spielstatistik und sämtliche Optionen für die Charaktereigenschaften an. Außerdem kann in diesem Menü eine Übersichtskarte ausgewählt werden, die es ermöglicht einen größeren Sichtradius anzuzeigen. In den Optionen werden die Nutzerdaten verwaltet. Hierunter fallen die Bankdaten, die benötigt werden um alle Funktionen des Spiels vollständig auszuschöpfen. Ebenfalls in den Optionen wählt man den Menüpunkt Logout um das Spiel zu beenden.
Da die gegebenen Menü-Controls des .NET Compact Frameworks keine Anpassungsmöglichkeiten bieten, wurde eine eigene Menüimplementierung durch das Erzeugen einer Control-Klasse `ImageButton` erreicht. Dieser `Imagebutton` wiederum enthält ein Panel mit Label und je nach Menüeinträgen entsprechend Buttons zur Auswahl. Hierdurch wurde es möglich, transparente Bilder in die Menüleiste zu legen und Farbanpassungen vorzunehmen.

Kapitel 4

Ergebnis

Das vierte Kapitel beinhaltet einen Ausblick und ein Fazit. Der Ausblick behandelt Überlegungen zur Weiterentwicklung des Prototypen, während das Fazit einen Rückblick gibt und die Arbeit abschließt.

4.1 Ausblick

Zuerst werden im Ausblick mögliche Erweiterungen besprochen. Darauf folgen Überlegungen um das Projekt in technischer Hinsicht zu optimieren. Welche Anwendungsszenarien sich durch die Entwicklung des Prototyps ergeben, wird im Anschluß besprochen.

4.1.1 Erweiterung des Spiels

Das Spiel in seiner letztendlichen Form mag dem Anspruch an ein Spiel genügen. Um jedoch eine Massentauglichkeit zu erreichen bzw. eine solide Kundenbasis aufzubauen, bedarf es einiger Veränderungen und Erweiterungen.

Hinzufügen eines/mehrerer Spielziele

Der erste größere Anreiz ein Spiel zu kaufen, welchen dieser Prototyp nicht besitzt, wäre die Schaffung eines oder mehrerer definierten Spielziele. Es ließe sich leicht ein Ziel in der Form finden, welches besagt der Beste in einer Disziplin zu sein oder die höchste Punktzahl für etwas zu bekommen. Da die meisten bekannten Multiplayerrollenspiele meist genau ein solches Ziel haben, wäre dies

ein angemessenes Ziel in diesem Genre für dieses Spiel. Natürlich liegt der Reiz dann in dem Erreichen kleinerer Zwischenziele.

Abrufbare Spieler-/Buddyliste

Ein weiterer wichtiger Punkt in einem Multiplayerspiel ist die Kommunikation innerhalb der Spielergemeinschaft. Es existieren zwar Möglichkeiten der Kommunikation zwischen den Spielern, allerdings ist diese nicht komfortabel genug, weil stets Spielernamen eingegeben werden müssen. Außerdem müssen sich die Spieler untereinander kennen, so dass der Name im Spiel bekannt ist und eine Adressierung erfolgen kann. Die einzige Möglichkeit andere fremde Spieler kennenzulernen bietet der Chat. Allerdings werden hier auch nur Spieler angezeigt, die sich ebenfalls im Chat befinden.
Durch diese Tatsachen wird die Kommunikation unter den Spielern erschwert bzw. das Kennenlernen fast unmöglich gemacht. Hier würden eine Freundesliste oder eine Liste der aktuell verbundenen Spieler Abhilfe schaffen. Man könnte sehen wer online ist und diesen direkt anschreiben.

Angriffe anders gestalten

Um den Spielspaß zu erhöhen, müsste die Interaktion zwischen den Spielern und gegnerischen Gebäuden umfangreicher gestaltet werden. Zum jetzigen Zeitpunkt erfolgt für den Angriff lediglich eine Berechnung nach dem Zufallsprinzip, allerdings läuft dieser Vorgang ohne weitere Einflussnahme einer der beiden Parteien ab. Dieses Problem ließe sich mit verschiedenen Möglichkeiten umgehen. Die Spieler könnten unter der Bedingung, dass beide im Spiel sind, ein Minispiel in Form von Vier gewinnt oder Tic Tac Toe austragen um so den Erfolg der Angriffsaktion zu beeinflussen. Eine andere Möglichkeit wäre, das Ausführen von Angriff und Diebstahl in dem Stil des Spiels Pacman zu gestalten. Der angreifende Spieler müsste somit den Diebstahl oder den Angriff selbst in einem fiktiven Gebäude durchführen.

Anpassung der Spielparameter (Balancing)

Wichtig für diesen Spielprototyp sind ebenfalls weitere Überlegungen zum Gleichgewicht der Spielwerte. Durch unzureichende Tests war es noch nicht möglich, zu analysieren in wie weit die Punktevergabe und die Berechnungen dahingehend zufriedenstellend durchgeführt werden, ohne dass ein Ungleichgewicht und somit eine Bevorteilung eines Spielers stattfindet, das Spiel zu kurzweilig ist oder keinen Anreiz bietet, es über längere Zeit zu spielen.

Weitere Menüs mit Statistiken zwischen den Spielern

Der Vergleich unter den Spielern ist genauso wichtig, um die Nutzer an das Spiel zu binden, wie ausgewogene Spielparameter. Hierfür bieten sich entsprechend dem Spielkonzept hauptsächlich Highscorelisten an, welche beispielsweise aufführen, wer die meisten Ressourcen abgebaut oder die meisten Gebäude errichtet hat. Dieser initiierte Konkurrenzkampf wirkt förderlich auf die Spielkultur und sorgt für eine treue Spielgemeinde.

4.1.2 Technische Optimierung

Kachelkonzept

In technischer Hinsicht lässt sich der Prototyp ebenfalls verbessern. Vordergründig wäre hier das Konzept der Kachelunterteilung zu erwähnen. Aufgrund der Benutzung von WGS84 und der damit verbundenen Aufteilung der Welt in Grad entstehen bei der implementierten Methode quadratische Kacheln, die einen Teil der Erde abbilden. Dieser Ausschnitt kann durch die Kugelform der Erde nicht quadratisch sein. Somit werden stets trapezförmige Ausschnitte der Erde auf quadratische Bilder abgebildet. Wegen der geringen Veränderung innerhalb einer Kachel kann dies vernachlässigt werden. Ein hierdurch größeres Problem ergibt sich aus dem Längengrad. Da die Kachelbreiten nicht automatisch an den Abstand zum Äquator angepasst werden bilden Kacheln aus Deutschland beispielsweise weniger Fläche ab als eine Kachel aus Afrika. Damit entsteht auf einer Kachelbreite mit gleicher Pixelzahl ein Zoomeffekt je höher der Breitengrad wird. Diese Problematik ließe sich mit der Umstellung auf das Gauß-Krüger-Koordinatensystem

verhindern. Hierdurch würde mit Metern statt Grad gerechnet. Die komplizierte Umrechnung der vom GPS-Empfänger gelieferten geographischen Koordinaten in das Gauß-Krüger-System führte allerdings zum Einsatz von WGS84.
In Bezug auf das Spielgeschehen entsteht ein weiteres Problem durch die Kachelunterteilung. Wenn ein Nutzer ein Spielgebäude errichtet und dabei nah an einem Kachelrand steht, so wird das Gebäude im schlechtesten Fall auf 4 Kacheln verteilt gezeichnet. Durch die Implementierung der Kachelaktualisierung und der Versendung von Bilddaten ist es nicht möglich mehr als eine Kachel im gleichen Ablauf zu verändern.
Es müsste weiterhin berechnet werden, wo das Gebäudebild zerschnitten werden muss und welcher Teil des Bildes auf welchem Kachelteil zu zeichnen ist. Dies wird durch eine Verschiebung des Gebäudes auf dem Kartenteil umgangen, so dass das Bild von einem Gebäude stets nur auf einer Kachel gezeichnet wird. Eine andere Möglichkeit dieses Problem zu entschärfen wäre die Vergrößerung einer Spielkachel, damit weniger Kachelgrenzen vorhanden sind. Dabei würden allerdings wieder mehr Ressourcen benötigt und das Grundproblem wäre nicht behoben.

Benutzerschnittstelle

Weiterhin ließe sich die Interaktion zwischen Spieler und mobilem Endgerät softwareergonomisch verbessern. Da das Spiel nicht ohne Texteingabe auskommt, wird die virtuelle Standardtastatur verwendet. Da sie allerdings viel Platz wegnimmt, kann man sie in der Statusleiste ein- und ausblenden. Das Problem, was bei geöffneter Tastaturansicht entsteht, ist die Überdeckung der Menüleiste und Eingabeformen. Der Nutzer sieht nicht, was er schreibt. Um dies zu beheben, müsste die Menüstruktur angepasst werden. Hierdurch wird allerdings die Benutzerführung erschwert, da die Fenster an unterschiedlichen Stellen erscheinen würden und dies den Nutzer eventuell verwirrt. Ein kompletter Verzicht auf die Tastatureingabe wäre nur durch die Abschaltung der Kommunikationsmöglichkeiten umsetzbar. Wenn die Kommunikation unverzichtbar ist, so könnte man ein Mikrofon zur Aufnahme von Mitteilungen benutzen. Der Aufwand für die Übertragung und Verwaltung der Audiodaten ist allerdings sehr viel höher als es bei geschriebenen Mitteilungen der Fall ist.

Da alle Spielfunktionen über das Menü erreichbar sind, findet die Interaktion mit den Spielgebäuden über Listen mit ihren Identifikationsnummern statt. Ab einer größeren Anzahl Häuser fällt es dem Nutzer schwer, die Gebäude zu unterscheiden. Eine intuitivere Möglichkeit für den Benutzer bestünde in der direkten Anwahl eines Spielgebäudes. Hierdurch wären Verwechslungen ausgeschlossen, allerdings müsste der Spieler sich stets in die Nähe des Spielgebäudes bewegen, damit es auf dem Spielfeld erscheint und anklickbar wird.
Was zur besseren Orientierung des Spielers beiträgt, wäre die Rotation des Spielfeldes, so dass sich stets das Spielfeld an dessen Blickrichtung ausrichtet. Leider unterstützt das C# .NET Compact Framework nur bedingt die Rotation von Bildern. Es ist möglich das Bild um 90, 180 oder 270 Grad zu drehen. Durch das Fehlen der Zwischenschritte würde dies den Spieler allerdings stark verwirren und nicht als Orientierungshilfe dienen. Aus diesem Grund wird ein Pfeil in die Mitte des Bildschirms gezeichnet, der die Blickrichtung anzeigt.

Caching

Die Zwischenspeicherung der Kartenkacheln zum Vermindern der Datenübertragung führt zu verschiedenen Bedingungen, ohne die eine Verwaltung der Kacheln nicht möglich ist. Ein Problem der Synchronisierung entsteht durch die gleichzeitige Speicherung der Bilddaten auf dem Server und dem Client. Der Server speichert während des Betriebes in einer Revisionsdatei, welche Kachel auf welchem Versionsstand ist, so dass Bildinformationen von Versionen unabhängig voneinander gespeichert werden. Hierdurch wird ermöglicht die Kacheln aus dem Cache zu löschen, aber den Fortschritt einer einzelnen Kachel beizubehalten. Die Konsistenz der Daten kann gefährdet werden, wenn der Server nicht ordnungsgemäß in der Lage war seinen Cache zu schreiben oder von außen manuell diese Versionsdatei gelöscht wird, der Client aber immernoch Kacheln im Cache behält.
Beispielsweise hält der Client die Version 5 einer Kachel im Speicher. Wenn der Server diese Kachel nun neu lädt, ohne Zugriff auf die generierte Versionsdatei zu haben, erhält diese Kachel die Version 1 und hat somit eine niedrigere Version als die Clientkachel. Ändert nun ein anderer Client dieses Teil durch einen Gebäudebau so bekommt diese Kachel den Wert 2 vom Server. Der Client mit der Kachelversion 5 fordert solange keine Aktualisierung der Kachel an, bis die

Versionsnummer 5 wieder erreicht ist. Der Client sieht dementsprechend keine Veränderungen auf dem Spielfeld, obwohl sich etwas ändert. Eine Lösung des Problems wäre die Erkennung des Servers, ob die Versionsdatei aktuell ist bzw. regelmäßige Backups dieser Datei. Eine andere Möglichkeit bestünde in einem generellen Rücksetzen der Versionen.

Eine Optimierung des Speicherplatzbedarfs vom Clientcache wäre weiterhin notwendig, da das vom Server gesendete JPEG-Bildmaterial auf dem mobilen Endgerät intern als Bitmap verarbeitet wird. Beim Beenden des Spiels speichert der Cache die Bilddaten auf dem Endgerät. Hierbei ist es aufgrund von Einschränkungen des .NET Compact Frameworks nicht möglich den Komprimierungsgrad von JPEG oder PNG-Dateien festzulegen. Um einen Qualitätsverlust der Bilder durch mehrmaliges Komprimieren zu vermeiden, werden PNG-Bilder mit einer durchschnittlichen Dateigröße von 250KB auf dem Endgerät gespeichert.

4.1.3 Anwendungsszenarien

Das umgesetzte Spiel stellt einen Prototyp dar, der sich abwandeln und so verändern lässt, dass weitere kommerzielle Anwendungsmöglichkeiten entstehen.

„Schnitzeljagd“

Die naheliegendste Anwendung um spielerisch ein GPS-Signal zu verwenden findet sich in der „Schnitzeljagd“ bzw. im Geocaching wieder. Hierbei werden Objekte versteckt und durch Anhaltspunkte wird es möglich das Objekt zu finden. Das mobile Endgerät würde hier nur der Koordinatenanzeige dienen. Durch die erarbeiteten Funktionsweisen mit Luftbildmaterial und eingeblendeten Objekten wäre es möglich, eine Jagd nach virtuellen Objekten spannender und abwechslungsreicher gestalten. Die Kommunikation zwischen den Spielern erweitert hierbei die Spielerfahrung.

Tauschbörse

Unabhängig von zeitlich begrenzten Veranstaltungen ließe sich eine Tauschbörse einrichten. Diese würde es den Nutzern ermöglichen, ortsabhängig digitale

Daten, wie Klingeltöne, Bilder oder Videos zu tauschen. Das bereits bestehende System würde es außerdem ermöglichen, die zu tauschenden Daten virtuell abzulegen und andere Nutzer danach suchen zu lassen. Die Verbindung zum Geocaching und der damit verbundene Reiz des Suchens nach einem Objekt liegt nahe. Dieser Reiz ließe sich durch die Einführung von digitalen Sammelbildern und der Verteilung dieser an verschiedenen Orten verstärken und eine junge Zielgruppe stärker ansprechen.

Informationsdienst

Eine Veranstaltung könnte sich den Prototyp so umfunktionieren, dass Besucher zu Beginn ein mobiles Endgerät erhalten und dieses dazu benutzen, spielerisch die Umgebung zu entdecken. Je nach Veranstaltung könnte man beispielsweise Gewinne ausschreiben für den Nutzer, der als erster alle Informationen an den vordefinierten Standorten abgefragt hat. Messen sowie Museen würden sich hierfür anbieten. In geschlossenen Räumen müsste allerdings eine andere Technik als GPS zur Positionsbestimmung eingesetzt werden.

Bonussystem

Auf Veranstaltungen mit Verkaufs- oder Werbeständen würde sich zusätzlich eine Art Bonussystem integrieren lassen, so dass für das Sammeln von Informationen, die von den werbenden Unternehmen zur Verfügung gestellt werden, Rabatte für den Einkauf von bestimmten Waren oder sonstige Ermäßigungen bzw. Belohnungen vergeben werden können. Eine abgewandelte Form dieses Bonussystem könnte ein Gewinnspiel sein. Mehrere Stationen gilt es hier zu finden und zu passieren. Je nachdem, was das Ziel darstellt, kann man nun anhand von Punkteständen oder ähnlichem einen Gewinner küren.

4.2 Fazit

Zusammenfassend lässt sich feststellen, dass alle zu Beginn gesetzten Zielstellungen erreicht wurden. Das Geoinformationssystem zusammen mit der Positionsbestimmung über GPS wurde sinnvoll in den Ablauf des Spiels integriert. Der spannende Ansatz orthographische Luftbilder der realen Spielerumgebung

als Spielfeld zu nutzen wurde mithilfe des Web Map Service umgesetzt. Auch die Speicherung geographischer Features in einer PostGIS-Datenbank findet Anwendung in der Ergänzung des Kartenmaterials durch virtuelle Spielgebäude. Die Anforderungen des mobilen Umfelds wurden durch Kachelung des Kartenmaterials berücksichtigt.
Problemen der mobilen Datenkommunikation versucht der Prototyp mit entsprechenden Lösungsansätzen entgegenzuwirken. Durch geringen Overhead bei der Übertragung der Spieldaten und hohe Komprimierung des Bildmaterials werden deutlich Datenmengen eingespart. Zu dieser Einsparung tragen auch das Vorladen und Zwischenspeichern der Kachelstücke bei. Während des Spiels auftretende Verbindungsausfälle werden durch automatische Neuverbindungen gehandhabt, um einen möglichst störungsfreien Ablauf zu gewährleisten.
Insgesamt hat die Studienarbeit eine interessante Anwendung von mobilen Geoinformationssystemen hervorgebracht, die vor allem durch die Multiplayerinteraktion überzeugen kann. Der prototypische Ansatz ist dabei nicht nur auf den spielerischen Bereich beschränkt, sondern bietet, wie in diesem Kapitel bereits beschrieben, andere nützliche Anwendungsmöglichkeiten.

Anhang A

Protokollübersicht

Die nachfolgende Auflistung ist gekürzt und beinhaltet nur die Beschreibung der Spielnachrichten ohne die dazugehörigen Parameter.

Nachrichten von Server zu Client

Nachricht	**Beschreibung**
<map>	Zustellung eines Kartenstücks
<tileupdated>	Benachrichtigung über geänderte Kachel
<registerstatus>	Status einer Registrierungsanfrage
<activationstatus>	Status einer Aktivierungsanfrage
<buildinglist>	Liste der Spielergebäude
<upgradebuildinglist>	Liste der Spielergebäude mit dazugehörigen Aufwertungen
<buildstatus>	Status einer Bauanfrage
<buildingupgradestatus>	Status einer Aufwertungsanfrage
<chat>	Verteilung einer Chatnachricht
<messagelist>	Liste der Nachrichten eines Spielers
<messagedeleted>	Bestätigung über Löschen einer Nachricht

<messagetext>	Textinhalt einer Nachricht
<messagestatus>	Status einer Nachrichtenzustellung
<tradeofferforward>	Benachrichtigung über ein Handelsangebot
<tradestatus>	Status einer Handelsaktion
<fightlist>	Liste der möglichen Angriffsziele
<fightstatus>	Ergebnis eines Angriffs
<levellist>	Liste der Spielerwerte
<accountdetails>	Accountinformationen
<resources>	Übermittlung des Ressourcenbesitzes
<costresponse>	Kosten für eine Aktion
<statistics>	Spielstatistiken

Nachrichten von Client zu Server

Nachricht	**Beschreibung**
<requestmap>	Anforderung eines Kartenstücks
<requestbuildinglist>	Abrufen der Gebäudeliste
<requestupgradelist>	Abrufen der Aufwertungsliste für Gebäude
<build>	Bauanfrage
<deletebuilding>	Löschen eines Gebäudes
<upgradebuilding>	Aufwertungsanfrage
<chat>	Verschicken einer Chatnachricht
<requestmessagelist>	Abrufen der Nachrichtenliste
<writemessage>	Versenden einer Nachricht

<deletemessage>	Löschen einer Nachricht
<readmessage>	Abfrage des Textinhalts einer Nachricht
<tradeoffer>	Handel anbieten
<tradereply>	Antwort auf Handelsangebot
<tradedelete>	Handel zurückziehen
<requestfightlist>	Abrufen der mögichen Angriffsziele
<fight>	Angriff auf Gebäude
<requestlevellist>	Abrufen der Spielerwerte
<levelup>	Spieleraufwertung
<requestresources>	Abfrage der Gesamtressourcen
<requeststats>	Abfrage der Spielstatistiken
<requestcost>	Kostenabfrage
<requestaccountdetails>	Abrufen der Accountinformationen
<deleteaccount>	Löschen des Accounts
<newpassword>	Änderung des Passworts
<requestactivation>	Aktivierungsanfrage

Literaturverzeichnis

[Bre07] Jack Bresenham. Algorithm for computer control of a digital plotter, Januar 1965, http://www.research.ibm.com/journal/sj/041/ibmsjIVRIC.pdf, Zugriffsdatum : 11.07.2007.

[Dre07] Stuart Dredge. 10 mobile games trends for 2007, Januar 2007, http://www.pocketgamer.co.uk/r/Mobile/feature.asp?c=2075, Zugriffsdatum : 10.07.2007.

[EP07] E-Plus. E-plus online flat, August 2007, http://www.eplus.de/tarife/13/13_1/13_1.asp, Zugriffsdatum : 13.08.2007.

[Gar07] Gartner. Gartner says worldwide mobile gaming revenue to grow 50 percent in 2007, Juni 2007, http://www.gartner.com/it/page.jsp?id=507467, Zugriffsdatum : 10.07.2007.

[IDC07] IDC. Idc says subscription-based purchases of wireless games will gain ground by 2010, Januar 2007, http://www.idc.com/getdoc.jsp?containerId=prUS20504407, Zugriffsdatum : 10.07.2007.

[Nin07] Nintendo. Annual report 2006, April 2006, http://www.nintendo.com/corp/report/06AnnualReport.pdf, Zugriffsdatum : 10.07.2007.

[NME07] NMEA. Nmea 0183 datensätze, August 2007, http://www.nmea.de/nmea0183datensaetze.html, Zugriffsdatum : 13.08.2007.

[Nok07] Nokia. Device specifications, August 2007, http://www.forum.nokia.com/devices/matrix_all_1.html, Zugriffsdatum : 13.08.2007.

[pgF07] pgFoundry. Npgsql: User's manual, August 2006, http://npgsql.projects.postgresql.org/docs/manual/UserManual.htm, Zugriffsdatum : 13.08.2007.

[Pri07] PricewaterhouseCoopers. German entertainment and media outlook: 2006 - 2010, Oktober 2006, http://www.pwc.de/fileserver/EmbeddedItem/german_E&M%20Outlook_2006.pdf?docId=e50b444c3a26bb5&componentName=pubDownload_hd, Zugriffsdatum : 10.07.2007.

[Rea07] RealNetworks. Research reveals casual games provide mental balance, stress relief and relaxation, August 2006, http://www.realnetworks.com/company/press/releases/2006/casgames_research.html, Zugriffsdatum : 10.07.2007.

[Res21] Refractions Research. Postgis documentation, 2007, http://postgis.refractions.net/docs/, Version: 1.2.1.

[Sha09] Sharpmap. Sharpmap documentation, 2006, http://www.codeplex.com/SharpMap/Release/ProjectReleases.aspx?ReleaseId=154, Version: 0.9.

[Sys07] Adobe Systems. Adobe - flash lite, Juni 2007, http://www.adobe.com/products/flashlite/, Zugriffsdatum : 13.08.2007.

[Tec07] Tecchannel. Wachstumsmarkt mobile gaming, Juni 2006, http://www.tecchannel.de/index.cfm?pid=216&pk=440894&p=1, Zugriffsdatum : 10.07.2007.

[Wig03] Andy Wigley. *Microsoft .NET Compact Framework*. Microsoft Press Books, Redmond, Washington, 2003.

[Yao04] Paul Yao. *.NET Compact Framework Programming with C#*. Addison-Wesley, Boston, 2004.

[You07] YourWorldGames. The shroud, Juni 2006, http://www.shroudgame.com, Zugriffsdatum : 10.07.2007.