Studienarbeit

Serverseitige Komponenten eines mixed-reality Spiels

Stefan Curow

19. Juni 2011

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

Gutachter:
Prof. Dr. Jens-Peter Redlich

Inhaltsverzeichnis

1 Einleitung

In dieser Arbeit werde ich das Thema „Serverseitige Komponenten eines mixed-reality Spiels“ betrachten. Zu Grunde liegt dazu ein konkretes Spiel „Cityy“, dessen technische Umsetzung, sowie die Durchführung von Performance-Tests Bestandteil dieser Arbeit ist. „Cityy“ ist ein ungewöhnliches Spiel, das ein Spielsystem an den Tag legt, welches bis zum Zeitpunkt dieser Ausarbeitung kein vergleichbares Pendant kennt. Der künstlerische Aspekt steht deutlich im Vordergrund, und stellt das Spielen an sich (Kampf um den Gewinn) in den Hintergrund. Was dabei so interessant und beachtenswert ist sollte daher zu Beginn dieser Ausarbeitung gezeigt werden.
Ich lege somit folgende Thematisierung für die kommenden Kapitel fest. Zuerst wird das Spielsystem vorgestellt. Bis dahin wird die Richtung, in die wir laufen schon deutlich erkennbar werden. Darauf aufbauend werden Anforderungen und Ziele definiert, die das System erfüllen muss. Nach kurzer Analyse der Gegebenheiten wird das System in Aufgabengebiete unterteilt. Darauf folgend wird die Umsetzung der einzelnen Teile vorgestellt. Um Aussagen über die Performance des Systems treffen zu können, werde ich anschließend die Programmteile in (Komplexitäts-) Klassen unterteilen, Tests vorstellen und die Resultate präsentieren. Abschließend werde ich Ausbaumöglichkeiten und Verbesserungsvorschläge erläutern.

2 Das Spiel

2.1 Hintergrund

Das Spiel „Cityy“ wurde von zwei Studenten der visuellen Kommunikation an der Weißensee Kunsthochschule Berlin als Diplomarbeit entwickelt. Das komplette Spiel (Idee, Spielsystem, Komponenten, Vorgänge und Wertigkeiten) und deren grafische Umsetzung für den Spieler sind der Praxis-Bestandteil der Diplomarbeit von **Christoph Mille** und **Constantin Andiel**.
Man sollte das Spiel also eher als ein Experiment betrachten. Es ist keineswegs komplett ausgereift, um als markttauglich gelten zu können. Manche Entscheidungen der technischen Umsetzung wurden genau aus diesem Grund getroffen. Wir werden in den nächsten Kapiteln sehen, welche Implementations-Entscheidungen davon betroffen sind und Änderungsvorschläge präsentieren, falls eine kommerzielle Verwertung geplant sein sollte. Vorerst werden wir aber annehmen, dass im Rahmen der Diplomarbeit das Spiel einmalig von 10-15 Personen gespielt werden wird. Das Spielsystem sieht vor, dass dazu ein territorial begrenzter Raum gewählt werden muss. Im Rahmen der Diplomarbeit wird dies die Stadt Berlin sein.
Um bei „Cityy“ mitspielen zu können benötigt ein Spieler ein Smartphone oder ein äquivalentes Gerät. An dieses werden folgende Mindestanforderungen gestellt:

- bestehende Internetanbindung
- integrierte Kamera
- Sensor zur Bestimmung der Lage/Ausrichtung
- Sensor zur Bestimmung der GPS-Koordinate

Diese Voraussetzungen erfüllt nahezu jedes heutzutage erhältliche Smartphone und sollte daher keine Restriktionen auf einen bestimmten Hersteller oder Smartphone-Variante darstellen.

2.2 Mixed Reality

„Cityy“ erfindet keine neuen technischen Gerätschaften, die ein Spieler erst erforschen, bzw. deren Funktionsweise begreifen muss. Es wird ausschließlich auf bestehende Technologie zurückgegriffen, die in geschickter Art und Weise miteinander in Kontakt gebracht wird. Der Begriff „Mixed Reality“ findet nach der üblichen Definition auch in diesem Spiel Bedeutung. Er meint, dass das Spielsystem als Verbindung aus einem realen Anteil

und einem virtuellen Anteil zu verstehen ist.
Das klassische Beispiel für eine Vermischung der realen und virtuellen Welt findet man in einem Smartphone-Reiseführer. Nehmen wir an, dass ein Smartphone seine GPS-Position durch einen eingebauten Mechanismus bestimmen kann, einen Lage-Sensor besitzt, sowie eine Kamera integriert hat. Informationen über Gebäude können entweder vormals auf dem Smartphone gespeichert worden sein oder just-in-time aus dem Internet geladen werden. All diese Mechanismen sind als gewöhnliche Ausstattung eines Smartphones bereits enthalten.
Steht ein Tourist vor dem Roten Rathaus der Stadt Berlin, so können ihm auf seinem Gerät Informationen über das Bauwerk, etc. angezeigt werden. Dazu muss der Nutzer lediglich die integrierte Kamera auf das Gebäude richten. Das Smartphone ist in der Lage in Zusammenarbeit mit dem Lage-Sensor herauszubekommen auf welches Gebäude der Nutzer sein Gerät richtet und zeigt ihm dann die Informationen darüber an.
„Mixed Reality“ kann hier als Mischung von dem realen Raum (der Tourist steht vor dem Gebäude) und dem virtuellen Raum (Informationen, Videos, Bilder, Baupläne, Daten des Gebäudes) verstanden werden. Das Spiel „Cityy“ ist ein eben solches System.

2.3 Motivation

Auf die Motivation kam ich in der Einleitung schon zu sprechen. Denn obwohl dieses Spiel als ein Experiment gesehen werden soll, kann man davon aus gehen, dass es trotzdem Spaß machen wird und eine sehr gute Alternative zu Familien-, Computer- & Konsolen- oder sonstigen Freizeitspielen darstellen kann. In „Cityy“ rückt der künstlerische Aspekt in den Vordergrund. Der Spieler muss primär geistige und kreative Finesse aufweisen.
Im Verlauf des Spiels erschaffen die Spieler Kunstwerke, die sie in der Öffentlichkeit installieren müssen. Somit werden die Ergebnisse des künstlerischen Schaffens die Stadt (oder einen anderen gewählten Raum) mit Kunstwerken verschönern. Bezüglich dieses Aspekts kann man davon ausgehen, dass es nicht nur Ziel eines Spielers sein wird, das Spiel zu gewinnen, sondern besonders gute und schöne Gebilde zu erschaffen, die das Stadtbild verbessern und prägen können.

2.4 Spielprinzip – Einführung

Der reale Raum, in dem der Spieler auftritt, wird mittels dem virtuellen Raum zu einem System zusammengefügt. Es entsteht dabei ein Aufbau-Strategie-Spiel. Ähnlich zu rein virtuellen Aufbau-Strategie-Spielen, wie „Age of Empires“, „Die Siedler“ oder „Civilisation“, wird auch in „Cityy“ das Spiel durch Sammlung/Produktion bestimmter Ressourcen vorangetrieben. Doch im Gegensatz zu den genannten virtuellen Spielen handelt es sich in „Cityy“ um nur eine Ressource: Energie. Analog zu z.B. „Age of Empires“ beginnt der Spieler mit einem Startkapital an Ressource(n).
Ebenso besteht eine Analogie zu dem Raum, in dem sich der Spieler ausbreiten, entwickeln und seine Gebäude (bzw. in „Cityy“ sind es ja Gebilde) bauen/setzen kann. Doch

genau hier fängt in „Cityy“ der reale Raum an. Der Spieler wählt mittels seines Smartphones, an welcher Position er das Spiel starten möchte. Dazu bewegt er sich mit dem Smartphone an die Position in der Stadt, startet die Spiel-Oberfläche und bestätigt seine aktuelle Position als Startpunkt. Diese GPS-Koordinate wird als „Home-Position“ des Spielers im Spiel festgehalten. Im virtuellen Raum entsteht ein neuer Avatar, der mit dem realen Spieler assoziiert ist. Der Avatar schwebt über der „Home-Position“ und kann einen vorerst nur kleinen Bereich in der virtuellen Welt überblicken, und sich dort ausbreiten/entwickeln. Später wird der Avatar dann höher schweben, und einen größeren Bereich überblicken können.
Des Weiteren muss ein Spieler einmalig ein Team-Logo festlegen. Über den gesamten Verlauf des Spiels wird er dieses Logo behalten. Wir werden später sehen, an welcher Stelle das Team-Logo benötigt wird.

2.5 Spielprinzip – Der Avatar

Der Avatar ist ein dreidimensionales Gebilde aus farbigen Würfeln. Er schwebt über der Home-Position des Spielers. Der Avatar ist für jeden Mitspieler sichtbar.
Die Gestalt des Avatar wird im Verlauf des Spiels von seinem Besitzer verändert. Es werden nach und nach zusätzliche Würfel zur Verfügung stehen, die der Spieler in das Gebilde einordnen kann. Dazu wird es in der Spiel-Oberfläche auf dem Smartphone eine geeignete Möglichkeit geben.

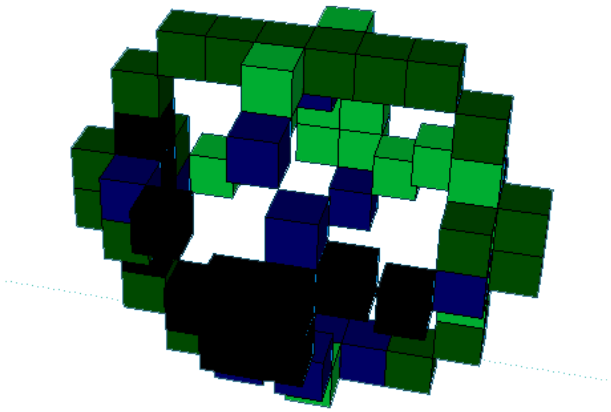

Abbildung 2.1: Beispiel eines entwickelten Avatars

Auf dem Smartphone können die virtuelle Welt und die Reale zusammengeführt werden. Da die GPS-Koordinate des Avatars definiert ist kann das Smartphone die Gestalt des

Avatars jedem Spieler anzeigen, der die integrierte Kamera des Geräts auf die korrekte Position richtet.

2.6 Spielprinzip – Sprossen

Wie bereits erwähnt treibt man das Spiel mittels Energie voran. Je mehr Energie man besitzt, bzw. produziert hat, desto höher schwebt der Avatar und desto größer wird der Ausbreitungsbereich, den ein Spieler im realen Raum einnehmen und besiedeln kann. Die primäre Quelle für Energie sind Sprossen. Sprossen sind real erschaffene Kunstwerke, die ein Spieler kreiert und in seinem Ausbreitungsbereich installiert. Damit eine Sprosse für den Spieler Energie produziert, muss er das Objekt in das Spielsystem einfügen. Dazu fotografiert er die Sprosse und übermittelt Foto und GPS-Koordinate an das Spielsystem. Um die Kreativität des Spielers nicht einzuschränken, ist es jedem selbst überlassen welche Materialien benutzt werden und welche Art von Kunstwerk am Ende heraus kommt. Es ist komplett irrelevant, ob der Spieler ein Stabile aus Holz erschafft oder eine Statue in Stein meißelt.

Abbildung 2.2: Eine Sprosse

Ein einziges Kriterium muss jedoch erfüllt sein. An dem Kunstwerk und auf dem Foto muss das Team-Logo des Spielers erkennbar sein. Dies wird benötigt um die Zugehörigkeit zu einem Spieler zu kennzeichnen. Im weiteren Verlauf dieser Arbeit werden wir sehen warum das Logo benötigt wird.
Das Einfügen einer Sprosse in das System kostet den Spieler Energie. Es besteht eine Ähnlichkeit zu dem Bau von Gebäuden in rein virtuellen Aufbau-Strategie-Spielen, wie

in z.B. „Age of Empires“. Der Bau von Gebäuden kostet Ressource(n). Man benötigt Einheiten von Stein, Holz und Arbeitskraft, um Bauwerke zu errichten. In „Cityy“ benötigt es lediglich Einheiten der Ressource „Energie“. Doch der Bau einer Sprosse wäre nicht lohnenswert, wenn dadurch keine positive Auswirkung im Spiel erreicht werden könnte. Sprossen sind die primäre Energiequelle, die einem Spieler zur Verfügung steht. In einem konstant festgelegten Zeitintervall produziert eine Sprosse Energie, die dem Spieler auf sein Konto gut geschrieben wird. Zunächst ist das Intervall auf 10 Minuten festgelegt worden. Das Energie-Konto eines Spielers hat unbegrenzt großes Volumen. Ein Spieler wird also nicht dazu gezwungen Energie auszugeben, damit die zehnminütlich neu produzierte Menge der kostbaren Ressource seinem Konto gut geschrieben werden kann.
Gleichzeitig wird mit den Kosten, die beim Erschaffen einer neuen Sprosse anfallen verhindert, dass Spieler eine übermäßig hohe Zahl an Sprossen generieren und in das System einpflegen. Sonst könnte sehr schnell ein starkes Ungleichgewicht zwischen den Spielern entstehen.
Um das Spiel interessanter und kniffliger zu gestalten, muss für eine eingefügte Sprosse ein Typ festgelegt werden. Dies erfordert gleichzeitig ein gesundes Maß an Entscheidungskraft mit Bedacht auf das Spielgeschehen und Allgemeinwissen in regionaler und überregionaler Hinsicht.
Sprossen unterschiedlichen Typs produzieren ein verschieden starkes Maß an Energie. Zusätzlich sind die Sprossentypen an regionalen Gegebenheiten orientiert, die dazu führen können, dass zwei Sprossen gleichen Typs an verschiedenen Standorten unterschiedlich viel Energie produzieren können. Ein einfaches Beispiel kann anhand der „grünen Sprosse“ erläutert werden. Stellt ein Spieler eine grüne Sprosse an einem Ort mit viel Grünfläche, Bäumen oder sogar einem Wald auf, so wird die Sprosse mehr Energie für ihn produzieren, als würde sie in einem Plattenbau-Wohnviertel installieren, in dem es eine überdurchschnittlich hohe Kriminalitätsrate gibt. Eine „Crime Sprosse“ würde in diesem Gebiet genau das Gegenteil bewirken und die bessere Wahl sein.
Der Spieler muss zur richtigen Wahl des Sprossentyps zusätzlich noch mit einberechnen, dass jeder Typ von Sprosse einen unterschiedlich hohen Betrag an Energie kostet.
Die folgende Liste zeigt alle Arten von Sprossen auf.

Sprossentyp	**Beschreibung**
Standard	Produziert konstanten Wert an Energie, unabhängig von regionalen Gegebenheiten.
Wind	Produziert an windigen Tagen mehr Energie als gewöhnlich.
Grün	Produziert mehr Energie, wenn die Sprosse an Orten mit viel Grünfläche, Bäumen oder Wäldern aufgestellt wird. In urbanen Gebieten ohne Erholungsmöglichkeiten in der Natur wird keinerlei Energie erzeugt.
Regen	Produziert an regenreichen Tagen mehr Energie als gewöhnlich.

Wasser	Produziert mehr Energie, wenn die Sprosse an Flüssen, Seen, Bächen oder am Meer aufgestellt wird. Ist am Aufstellungsort keinerlei Wasser vorhanden, so wird keinerlei Energie produziert.
Elektrizität	Energieproduktion orientiert sich am tagesbedingten Energieverbrauch des Stadtteils, in dem die Sprosse aufgestellt wurde.
Hartz 4	Energieproduktion orientiert sich an der Arbeitslosenquote des Stadtteils, in dem die Sprosse aufgestellt wurde. Je höher die Arbeitslosenquote, desto mehr Energie wird produziert.
Rumpel	Produziert mehr Energie, wenn die Sprosse an größeren Straßen oder Autobahnen aufgestellt wird. Eingeschränkte Energieproduktion an Nebenstraßen, und keinerlei Energieproduktion in Parks, auf großen Grünflächen und sonstigen Gebieten, ohne Straßen.
DB	Produziert nur Energie, wenn die Sprosse an einem Bahnhof aufgestellt wird. Die Höhe der Produktion orientiert sich an der Anzahl der S-& U-Bahn Linien, sowie Regional- und Fernbahnlinien, die an diesem Bahnhof halten.
Ackermann	Energieproduktion orientiert sich an den Tageswerten des DAX.
Benzin	Energieproduktion orientiert sich an den Tageswerten des an der Börse geführten Rohölpreises.
Imbiss	Sondersprosse! Nimmt man selbst, oder ein Mitspieler einen Imbiss an dem Geschäft, an dem die Sprosse aufgestellt wurde und informiert das Spielsystem mittels Foto über dieses Vorgehen, so erhält der Besitzer einen Bonus an Energie.
Party	Sondersprosse! Besucht man selbst, oder ein Mitspieler die Party an der Stelle, an der die Sprosse aufgestellt wurde und informiert das Spielsystem mittels Foto über dieses Vorgehen, so erhält der Besitzer und der Party-Besucher einen Bonus an Energie.
Crime	Produktion an Energie hängt von der am Vortag veröffentlichten Anzahl von kriminellen Geschehnissen in dem Stadtteil ab, in dem die Sprosse gesetzt wurde.
Politik	Produziert mehr Energie, wenn die Sprosse in Gebieten mit höherer politischer Orientierung gesetzt wurde, als in politisch inaktiveren Gebieten.
Schwaben	Energieproduktion orientiert sich am Mietspiegel für die Straße in der die Sprosse gesetzt wurde. In Gebieten mit sehr geringem Mietspiegel wird keinerlei Energie produziert.

History	An historisch wertvollen Gebieten wird mehr Energie produziert als gewöhnlich.
King	Sondersprosse! Wird die Sprosse an oder neben künstlerisch wertvollen Gebieten, Gebäuden, Skulpturen oder Denkmälern errichtet, so produziert die Sprosse mehr Energie. Das Urteil über die künstlerische Wertigkeit liegt bei den Erfindern des Spiels. Damit ist nicht die Art oder Arbeit an der Sprosse selbst, sondern lediglich die Umgebung gemeint.
Todesstreifen	Eine Sprosse dieses Typs erzeugt nur Energie, wenn sie am ehemaligen Grenzverlauf der DDR zur BRD gesetzt wird.
Sentinel	Sondersprosse! Diese Sprosse zeichnet gegnerische Bewegungen in ihrem unmittelbaren Umkreis auf. Der Besitzer kann die Informationen jederzeit abrufen. Die Sprosse produziert keine Energie. Es wird kein Foto benötigt, um eine Sentinel-Sprosse zu errichten.
Pionier	Sondersprosse! Die Sprosse kann erst nach einiger Zeit gesetzt werden. Sie kann außerhalb des eigenen Ausbreitungsbereichs gesetzt werden, um neue/weitere Gebiete zu erschließen. Dies Ermöglicht den Bau von Sprossen in einem neuen Ausbreitungsbereich. Die Pionier-Sprosse produziert keine Energie. Es wird kein Foto benötigt, um eine Pionier-Sprosse zu errichten.

Tabelle 2.1: Sprossentypen und deren Bedeutung

Sicherlich ist zu erkennen, dass manche Sprossentypen für die Stadt Berlin interessanter oder überhaupt von Bedeutung sind, als für eine andere Stadt oder ein nicht urbanes Gebiet.
Erwähnenswert ist außerdem, dass die „King Sprosse“ bisher von den Erfindern des Spiels auf künstlerische Wertigkeit beurteilt wird. Dies verursacht jedoch hohen Arbeitsaufwand der Erfinder. Falls das Spiel veröffentlicht werden sollte, so muss diese Entscheidung den Mitspielern überlassen werden. Es würde ein Spielmitglieder-basiertes Interface benötigt werden, in dem gegenseitig über Wertigkeiten diskutiert und abgestimmt werden müsste.
Bisher habe ich noch nicht erwähnt, wie hoch oder niedrig die Kosten für eine Sprosse sind, oder wie viel Energie sie zehnminütlich produzieren wird. Das liegt daran, dass die konkreten Wertigkeiten zum Einen noch nicht vollständig fest stehen, und zum Anderen auch den Spielern nicht zu Spielbeginn mitgeteilt werden. Das Programm auf dem Smartphone wird mit Absicht genau so funktionieren. Die Spieler sehen zu Anfang weder, welche Arten von Sprossen es geben wird, noch wird ihnen verraten, warum und wie viel Energie eine bestimmte Art von Sprosse produziert. Dies müssen sie während des Spiels mittels „Learning by Doing“ herausfinden. Dies betrifft Beispielsweise die Art der

Sprosse, die ein Spieler setzen kann. Hat der Spieler nicht genug Energie, um eine „Todesstreifen Sprosse“ zu setzen, so wird ihm diese Möglichkeit im Smartphone-Programm auch nicht angezeigt. Weitere Restriktionen können ebenso dazu führen, dass verschiedene Möglichkeiten erst zu späteren Zeitpunkten offenbart werden. Eine solche Bedingung könnte zum Beispiel sein, dass ein Spieler eine „Ackermann Sprosse“ erst erschaffen kann, wenn er schon mindestens 10 Sprossen besitzt.

2.7 Spielprinzip – Aktionen

Natürlich ist es wünschenswert, dass die Spieler untereinander um Energie, Ruhm und Mächtigkeit konkurrieren. Darin liegt der Spaß, den man während des Spielens haben kann. Das Spiel gibt den Spielern die Möglichkeit, im realen und auch im virtuellen Raum sowohl in positiver, als auch in negativer Weise miteinander zu interagieren. Dazu stehen verschiedene Aktionen zur Verfügung. Wie auch die Arten von Sprossen und Wertigkeiten der Energieproduktion werden dem Spieler zu Beginn nicht alle Möglichkeiten offen gelegt.
Wie man bereits richtigerweise erahnen kann sind Aktionen die primäre Energiesenke. Keine Aktionen auszuführen ist weitestgehend sinnfrei. Energie auf dem Konto aufzubewahren, und nie zu benutzen hat in „Cityy“ keine positiven Auswirkungen. Der Spieler wird also zur Interaktion mit seinen Mitspielern auf natürlichem Wege verleitet. Und das ist im Spielsystem eindeutig erwünscht. Um die Entscheidung zur Verwendung von Aktionen einfacher, aber die Wahl der einzusetzenden Aktion kniffliger zu machen, sind unterschiedlich hohe Beträge an Energie zu bezahlen. Manche Aktionen sind kostenlos. Im Spielsystem wird vorgesehen, dass die kostenlosen Aktionen die Elementarsten sind, und daher am häufigsten genutzt werden sollen. Um Aktionen auf gegnerische Sprossen anzuwenden (wir werden diese gleich kennen lernen) muss sich der Spieler vor der Sprosse befinden.
Dem Spieler stehen im realen Raum die folgenden Aktionen zur Verfügung:

Aktion	**Beschreibung**
Tracker	Zeigt alle Mitspieler-Sprossen im Umkreis der aktuellen GPS-Position auf der im Spiel integrierten (Land-)Karte an. Diese Aktion ist kostenlos. Erkannte Sprossen werden für einen begrenzten Zeitraum im Spielsystem für den Spieler als „verortet“ gespeichert.
Pfadfinder	Der Pfadfinder erfüllt die selbe Funktionalität, wie der Tracker, doch arbeitet auch im Hintergrund. Der Spieler kann einfach durch die Stadt fahren / laufen, und wird über neu erkannte Sprossen in seiner Umgebung informiert.

Sonar	Nutzt der Spieler diese Aktion, so wird ihm etwas Zeit gegeben, um in der Umgebung herum zu fahren / laufen. Die dabei zurückgelegte Folge von GPS-Koordinaten kann zu einem Track zusammen gefasst werden. Verbindet man Anfangs- und Endpunkt, so entsteht eine Fläche. Dem Spieler werden alle Mitspieler-Sprossen in der produzierten Fläche angezeigt.
Radar	Zeigt alle Mitspieler-Sprossen an, die in der Nähe des eigenen Ausbreitungsbereichs gesetzt wurden. Die dabei benutzte Entfernung ist eine definierte Größe.
Sentinel	Das Setzen einer Sentinel-Sprosse bewirkt, dass die Sprosse alle gegnerischen Bewegungen in ihrem unmittelbaren Umkreis aufzeichnet. Der Besitzer kann kostenfrei alle Aufzeichnungen abrufen und auf der (Land-) Karte anzeigen lassen.
Sprosse töten	Der Spieler kann eine Mitspieler-Sprosse töten, indem er sie in der realen Welt zerstört. Im Anschluss muss ein Beweisfoto dem Spielsystem übermittelt werden. Ist eine Sprosse getötet, so produziert sie keine Energie mehr für ihren Besitzer. Eine Sprosse gilt auch als getötet, wenn das Team-Logo entfernt, oder die Sprosse von ihrer Platzierung entfernt wird. Es kann ebenso passieren, dass aufgrund von Regen oder anderen Witterungsbedingungen ein Kunstwerk zerstört wird. Eine Sprosse lebt so lange (und produziert Energie), bis ein Mitspieler ein Beweisfoto an das Spielsystem übermittelt, das das Gegenteil aufzeigt. Das Töten einer Sprosse ist eine kostenlose Aktion.
Sprosse wiederbeleben	Wurde eine Sprosse getötet, so kann sie reaktiviert werden. Dazu muss ein Beweisfoto an das Spielsystem übermittelt werden. Für das Wiederbeleben gilt das Selbe, wie für das Einfügen einer neuen Sprosse: Das Team-Logo muss erkennbar auf dem Foto sein. Die GPS-Koordinate darf beim Wiederbeleben nicht verändert werden. Ist dies (vielleicht zwangsweise) nötig, so muss eine neue Sprosse eingefügt werden. Dabei entsteht kein Nachteil. Die Anwendung der Wiederbelebungs-Aktion ist kostenlos.
Party-Besuch	Hat ein Spieler eine „Party Sprosse“ an einem passenden Ort installiert, so kann jeder Mitspieler ein Foto von sich auf der Party und der Sprosse machen, und die Party-Besuch Aktion ausführen. Sowohl der Besucher, als auch der Sprossen Besitzer bekommen einen Energie Bonus. Der Eigentümer kann natürlich auch ein Foto von sich auf der Party neben seiner eigenen Sprosse ins Spielsystem einpflegen.

Imbiss-Besuch	Hat ein Spieler eine „Imbiss Sprosse“ an einem passenden Ort installiert, so kann jeder Mitspieler ein Foto von sich und seiner Nahrung an der Sprosse machen, und die Imbiss-Besuch Aktion ausführen. Der Besitzer der Imbiss Sprosse bekommt einen Energie Bonus.

Tabelle 2.2: Aktionen im realen Raum

Im virtuellen Raum sind vom Spieler die folgenden Aktionen anwendbar:

Aktion	**Beschreibung**
Boost	Diese Aktion ist kostenfrei, und kann sowohl auf eigene, als auch auf gegnerische Sprossen angewendet werden. Die nächste Energieproduktion der gewählten Sprosse wird einmalig um einen positiven Faktor verbessert.
Tarnkappe	Anwendbar auf eine eigene Sprosse. Befindet sich eine Sprosse in diesem Zustand, so können Mitspieler das Foto der Sprosse nicht sehen. Damit fällt es Ihnen schwerer die Sprosse im realen Raum zu finden, um Aktionen darauf anzuwenden. Die Aktion kostet den Inhaber Energie.
Anti-Tarnkappe	Anwendbar auf eine gegnerische Sprosse im Zustand „Tarnkappe“. Die Sprosse wird von dem Zustand „Tarnkappe“ befreit. Das Foto ist wieder sichtbar. Um die Aktion durchzuführen wird keinerlei Energie benötigt, sondern es muss ein Mini-Spiel gelöst werden.
Freeze	Anwendbar auf eine gegnerische Sprosse, die sich nicht bereits im Zustand „Freeze“ befindet. Die Energieproduktion der Sprosse wird um einen festgelegten Faktor reduziert. Diese Aktion kostet den Anwender Energie.
Auftauen	Anwendbar auf eine eigene Sprosse, die sich im Zustand „Freeze“ befindet. Der Zustand „Freeze“ wird entfernt. Die Energieproduktion läuft normal weiter.
Parasit	Anwendbar auf eine gegnerische Sprosse, die sich nicht bereits im Zustand „Parasit“ befindet. Die Energieproduktion der Sprosse wird um einen festgelegten Faktor reduziert. Nach einem Zeitraum t wird die Zustandsveränderung automatisch entfernt. Der Parameter t ist in der Aktion wählbar. Je höher man den Parameter t wählt, desto mehr Energie muss man zur Ausübung dieser Aktion bezahlen.

Mückenschutz	Anwendbar auf eigene Sprossen. Der Spieler muss in einem bestimmten Zeitintervall einen GPS-Track bewältigen, der nach Abschluss durch Verbinden der ersten und letzten Koordinate geschlossen wird. Alle in dem Track eingeschlossenen Sprossen bekommen den Mückenschutz. Alle mit dem „Parasit“ befallenen Sprossen werden von den Parasiten befreit. Alle anderen Sprossen werden, für Gegenspieler nicht sichtbar, von einem Mückenschutz umgeben, der den nächsten parasitären Befall verhindert.
Tag	Anwendbar auf eine gegnerische Sprosse, die maximal von 3 Mitspielern gleichzeitig als „Tag“ vermerkt worden ist. Setzt man einen Tag auf eine gegnerische Sprosse, so wird sie für den Spieler auf der Karte angezeigt und er kann Aktionen ausführen, auch wenn sich der Spieler nicht in der unmittelbaren Nähe der Sprosse befindet. Des Weiteren können zusätzliche Aktionen auf die Sprosse angewendet werden, die bis zum Zeitpunkt dieser Ausarbeitung noch nicht weiter spezifiziert worden sind. Das Setzen eines Tags belastet das Energie-Konto des „Taggers“.
Anti-Tag	Anwendbar auf eine eigene Sprosse, die von Gegenspielern in den Zustand „Tag“ versetzt wurde. Der Spieler bezahlt einmalig einen konstanten Betrag an Energie, um alle „Tags“ dieser Sprosse zu beseitigen.
Sprengfalle	Anwendbar auf eine eigene Sprosse. Es kostet den Spieler Energie, um eine seiner Sprossen in den Zustand „Sprengfalle“ zu versetzen. Wird ein Gegenspieler in der Nähe dieser Sprosse gesichtet, so muss er ein sofort gestartetes Mini-Spiel bewältigen. Im Anschluss wird die Sprengfalle entfernt. Ist der Gegenspieler bei der Bewältigung des Mini-Spiels erfolglos, so wird ihm Energie von seinem Konto abgezogen.
Sekurit	Beim Sekurit handelt es sich um ein von den Sprossen unabhängiges System. Zum Setzen eines Sekurits (oder besser: einer Sekurit-Linie) muss Energie bezahlt werden. Der Spieler bekommt dann etwas Zeit seine GPS-Position zu verändern, wobei nur die erste und letzte GPS-Position in Betracht gezogen werden. Die direkte Verbindung zwischen den beiden Koordinaten bildet die Sekurit-Linie. Sekurit-Linien sind für Gegenspieler auf der Karte nicht sichtbar. Durchläuft ein Gegenspieler eine gesetzte Sekurit-Linie, so muss dieser ein Mini-Spiel bewältigen. Äquivalent zu der Sprengfalle wird die Sekurit-Linie entfernt, und dem Gegenspieler wird Energie vom Konto abgezogen, sollte er das Mini-Spiel verlieren.

Antisekurit	Dies ist ein Zustand, in den ein Spieler durch konsequentes Bezahlen von Energie versetzt werden kann. Befindet man sich im Antisekurit Zustand, so werden gegnerische Sekurit-Linien auf der Karte angezeigt. Nun kann man diese Umgehen. Der Zustand muss vom Spieler selbst entfernt werden, ansonsten wird ihm ein gewisser Betrag an Energie für ein festgelegtes Zeitintervall vom Konto abgerechnet.

Tabelle 2.3: Aktionen im virtuellen Raum

3 Systemanalyse & Entwurf

3.1 Zentrales System

In den vorherigen Kapiteln haben wir gelernt, dass das Spiel aus mehreren Objekten besteht. Die realen Objekte werden mit Digitalfotos und GPS-Koordinaten zu virtuellen Objekten. Aus dem Spielprinzip können wir schließen, dass es eine Steuerung und Kontrollfunktionen über die Spielobjekte geben muss. Da das Entwickeln eines verteilten Speicher-/Organisationssystems für deutlich mehr Aufwand bedeuten würde, entschied ich mich das System zentral zu gestalten. Wir werden daher alle Objekte an einer zentralen Stelle verwalten, sowie Anfragen und Aktionen über Selbige abhandeln.

3.2 Server – Multi-Client Modell

Aus einem zentralisierten System resultiert, dass ausschließlich gespielt werden kann, wenn die zentrale Stelle für alle Spieler verfügbar ist. Das Internet soll als Kommunikationsschnittstelle dienen, da die Internetanbindung für Smartphones immer günstiger, schneller und an immer mehr Orten verfügbar wird. Um keine Restriktionen über die Art des Smartphones zu treffen, werden wir die Geräte über gängige und erprobte Protokolle mit dem Server kommunizieren lassen.
Es muss berücksichtigt werden, dass die mobile Internetanbindung von Smartphones durch verschiedene Einflüsse unterbrochen werden kann. Kurze Kommunikationsprozesse wären daher von Vorteil. Um die Nutzbarkeit zu steigern wäre es ebenso wünschenswert, nur wenig und selten mit dem Serversystem zu kommunizieren. Der Spielspaß steigt nicht gerade, wenn das Anzeigen der Sprossen in der Umgebung äußerst lange dauert.
Das Serversystem muss auf jeden Fall unterstützen, dass multiple Clients gleichzeitig Anfragen stellen und Funktionen aufrufen können. Das Einfügen von Anfragen in eine Warteschlange und iteratives Abarbeiten wäre nicht akzeptabel.
Des Weiteren gelten Datenverluste als fatal. Ich werde nicht davon ausgehen, dass Hardware ewig hält und nie technische Fehler auftreten. Ein System zur Vorbeugung von Datenverlusten muss auf jeden Fall im Serversystem-Konzept berücksichtigt werden.
Aus der Einführung dieser Ausarbeitung sollte klar geworden sein, dass das Spielsystem bisher keineswegs als ausgereift und unveränderbar gilt. Um in Zukunft simple oder auch schwerwiegende Modifikationen vornehmen zu können, muss das Serversystem leicht bedienbar und komponentenweise austauschbar sein. Eine klare Trennung von Spieldaten und Spiellogik ist daher dringend zu empfehlen.

3.3 Komponentenmodell

Das Projekt wird in folgende Abschnitte unterteilt:

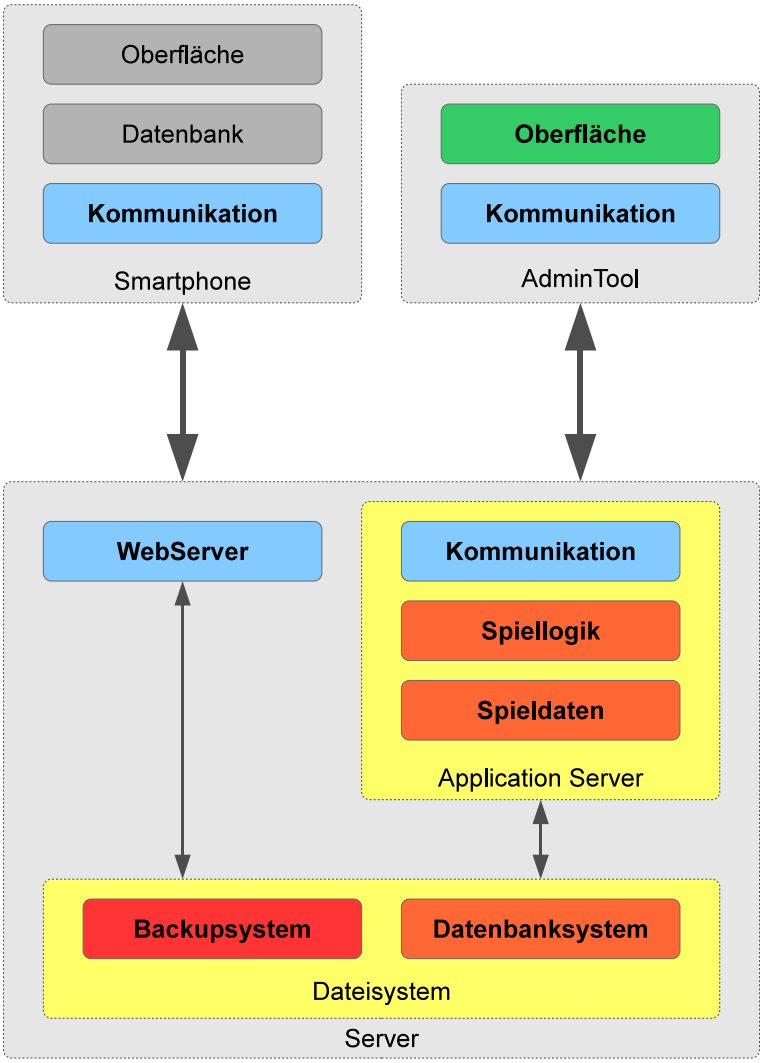

Abbildung 3.1: Komponentenmodell des Systems

Die farbig markierten Komponenten in der Abbildung 3.1 liegen dabei in meinem Aufgabengebiet. In den folgenden Kapiteln werden die Eigenschaften und die Implementation der einzelnen Module vorgestellt. Diese gehören zum Umfang meiner Studienarbeit. Die Funktionsweise der grau markierten Elemente werden im Verlauf dieser Ausarbeitung nur kurz erläutert, da in vereinzelten Fällen ein Ausblick als notwendig erscheinen wird.

4 Implementation

4.1 Hardware

Als Grundlage steht dem Experiment ein gebrauchtes Computersystem, sowie eine gewöhnliche DSL-Heimnetz Internetanbindung zur Verfügung. Dies wird in unserem Experiment zwar ausreichend sein, aber kann nicht als Grundlage für ein markttaugliches Produkt gelten. Bei der Implementation wurde darauf geachtet, dass die Hardware austauschbar, sowie der Standpunkt des Serversystems frei wählbar bleibt.
Da das Computersystem in den folgenden Kapiteln als Referenzsystem für die Performance Tests herhalten muss, werde ich die eingebauten Komponenten hier kurz vorstellen:

Prozessor: Intel Pentium 4 (3.00 GHz)

Speicher: 2 GB DDR-Ram (4 x 512MB)

Festplatten: 3 Stück

- Samsung SP2514N (250GB, SATA)
- Seagate ST3120026AS (120GB, SATA)
- Maxtor STM325031 (250GB, ATA)

Es wurde von mir das Serverbetriebssystem Ubuntu Server 10.04 installiert. Die beiden SATA Festplatten bilden dabei ein RAID-Verbund im Level 1 (Spiegelung), der vom Betriebsystem geleitet wird (Software-RAID). Alle Programme, die ich benutzen werde, sind auf diesem gespiegelten Verbund installiert. Damit versuche ich einem Datenverlust durch Ausfall von Festplatten vorzubeugen. Auf der ATA Festplatte werden ausschließlich Snapshots des Systems gespeichert. Dies dient mir als letzte Rettung, falls wirklich beide Festplatten aus dem Verbund ausfallen, oder ein vergangener Systemstatus (zumindest teilweise) wiederhergestellt werden muss. Das Programm rsnapshot macht dazu in regelmäßigen Abständen Sicherungen von allen Daten, die für das Spielsystem wichtig sind, auf die dafür vorgesehene Festplatte.

4.2 Internet

Der Computer ist durch einen Router mit dem Internet verbunden. Da dieser eine Netzwerk Adress Übersetzung (NAT – Network Address Translation) vornimmt, muss ich mich um die Konfiguration einer Firewall vorerst nicht kümmern. Der Router wird so konfiguriert, dass er nur Ports an den Server weiterleitet, die als Vertrauenswürdig gelten,

oder für das Projekt wirklich notwendig sind.
Da es sich bei der Internetanbindung um eine Heimnetzanlage handelt, muss ich davon ausgehen, dass der Internetanbieter Zwangstrennungen und Neuvergabe der öffentlichen IP durchführt. Dazu wurde ein Dynamischer DNS Name registriert, und der Router darauf eingerichtet.

4.3 Middleware

Der Einsatz eines Middleware Produkts hilft mir bei der Problemlösung der Aufgabenstellung. Das Aufgabengebiet einer Middleware Plattform, das mich interessiert, ist das Verteilen von zentral gelagerten Objekten an multiple Clients über erprobte Schnittstellen. Dabei sollen und können Clients gleichzeitig auf Objekte zugreifen, ohne in Konflikte zu geraten.
Ich habe mich dafür entscheiden den offenen JBoss Application Server zu benutzen, und werde mein Projekt nach Enterprise JavaBeans 3 (EJB3) konformen Richtlinien verfassen. Die EJB Architektur sieht hinreichende Richtlinien vor, die zu den aufgestellten Anforderungen an das Serversystem passen.
Die klare Trennung zwischen Daten- und Logikschicht des Systems hilft nicht nur, das Spielsystem einfach veränderbar zu gestalten, sondern unterstützt ebenso eine rasche Austauschbarkeit des zu Grunde liegenden Datenbanksystems und Möglichkeiten zur Anpassung der Kommunikationsschicht.
Des Weiteren ist es durch das Benutzen der EJB Architektur möglich Anfragen parallel zu bearbeiten. Durch Nutzung des eingebauten Transaktionsmodells und der Java Persistence API (JPA) wird sichergestellt, dass eine Anfrage entweder komplett bearbeitet oder verworfen wird. Dies ist wichtig, um ein Management zwischen den parallel auftretenden Anfragen zu erlangen und Übertragungsfehler aufgrund möglicher Verbindungsprobleme zu tolerieren. Der Java Message Service (JMS) wird dazu benutzt werden, um gewisse Anfragen/Aufgaben verspätet abzuarbeiten, die keine konkreten Rückgabeobjekte erwarten.
Das Serversystem hat die Aufgabe, eine Schnittstelle für die Clients bereit zu stellen. Hier werde ich Gebrauch von WebServices für die Smartphones und RMI-IIOP (Remote Method Invocation über Internet Inter-Orb Protocol) für das AdminTool machen.

4.4 Datenbanksystem

Der JBoss Application Server benutzt standardmäßig das Datenbanksystem HSQLDB (Hyper Structured Query Language Database), auch bekannt als HypersonicSQL. Es zählt zu den eingebetteten Datenbanksystemen, wird im Prozess des Application Servers gestartet, und ist von außen nicht ersichtlich. Es besteht zwar die Möglichkeit, die Funktionsweise des HSQLDB Systems zu ändern, doch gibt es keine Möglichkeit von außerhalb des Prozesses ein Datenbankdump anzufordern. Da diese Funktionalität zum Erstellen

von Backups/Snapshots jedoch dringend benötigt wird, habe ich das unterliegende Datenbanksystem geändert und auf dem Betriebsystem ein MySQL Datenbanksystem installiert.

4.5 Smartphone

Als Smartphone steht dem Projekt ein HTC Desire zur Verfügung. Mit Absicht wurde ein Gerät gewählt, dass mit dem quelloffenen Betriebsystem Android versehen ist. Dieses bietet Entwicklern eine umfassende Entwicklungsumgebung, die viele hilfreiche Funktionen und Bibliotheken enthält. Das Betriebsystem läuft auf Basis eines Linux Kernels, der Prozess- und Speicherverwaltung übernimmt. Selbstentwickelte Programme sind in Java zu schreiben, und werden in einer Java Virtual Machine (JVM) ausgeführt. Nun unterliegt man schnell dem Trugschluss, dass somit die Möglichkeit bestehen könnte bereits entwickelte Java Klassen (nicht Android) in ein Android Projekt einpflegen zu können. Doch leider ist die auf dem Smartphone installierte Java Virtual Machine (Dalvik VM) nicht kompatibel zu Klassen, die für eine Sun/Oracle JVM kompiliert wurden.
Das Android Software Development Kit (Android SDK) enthält Bibliotheken, die speziell für die Dalvik VM kompiliert wurden. Der Umfang dieser Ansammlung von Klassen ist jedoch nicht vergleichbar mit der Menge an frei verfügbaren Sun/Oracle JVM kompatiblen Klassen. Enttäuschenderweise findet sich zum Beispiel keine Möglichkeit RMI-IIOP Aufrufe tätigen zu können.

4.6 WebServices

Die Nutzung von WebServices ist ebenso nicht in den Standard Bibliotheken der Android SDK vorgesehen. Möchte ein Client einen angebotenen WebService benutzen, so ruft er gewöhnlicherweise zunächst ein WSDL (WebService Description Language) Dokument ab. Dieses beschreibt XML-konform den Funktionsumfang des WebServices. Daraus kann automatisiert eine Klasse in der clientseitig benutzen Programmiersprache generiert werden. Die konkrete Instanz dieser Klasse liegt dabei physikalisch im Speicher des Serversystems. Mittels SOAP (Simple Object Access Protocol) können Funktionsaufrufe auf der Instanz von dem entfernten Client aufgerufen werden, indem er Funktionen auf dem generierten Objekt aufruft. Anstelle der Abarbeitung im eigenen Speicher wird die Anfrage an den Server via SOAP übermittelt. Dies bedeutet Senden von Parametern in einem XML-konformen Dokument, zumeist via HTTP (Hypertext Transfer Protocol), an den Server und Erwarten einer Antwort in selbigem Format. Das Resultat kann dann wieder zu einem Rückgabewert in der clientseitig benutzten Programmiersprache automatisiert umgewandelt werden.
Doch leider stehen im Android SDK weder Bibliotheken zum automatischen Generieren von Klassen aus WSDL Dokumenten, noch Senden von Anfragen oder Erhalten von Antworten via SOAP zur Verfügung. Bis zum Zeitpunkt dieser Ausarbeitung findet sich auch

keine offene Bibliothek, die diese Lücke schließt. Nur durch die Nutzung einer offenen, jedoch minimalen Bibliothek (ksoap2) findet sich wenigstens die Möglichkeit, SOAP Anfragen manuell zu erzeugen und via HTTP-Client an den Server zu schicken. Die Antwort dieser Operation muss dann allerdings ebenso manuell von mir verarbeitet werden.
Um mein Aufgabengebiet abzuschließen, habe ich eine Bibliothek erstellt, die jegliche Kommunikation mit dem Serversystem übernimmt, und über eine Schnittstelle alle Möglichkeiten zur Interaktion im Spiel definiert.

4.7 Enterprise Java Beans

4.7.1 Datenschicht

Die EJB Architektur definiert eine klare Trennung zwischen Daten- und Logikschicht. Daher werde ich die beiden Bereiche aus dem Projekt nacheinander vorstellen. Die Datenschicht definiert Klassen von Informationen, die über die einzelnen Spieler, Spielobjekte und Zustände Auskunft geben. Diese Informationen werden vom Application Server persistent in die Datenbank geschrieben. Die Datenschicht muss alle Spielobjekte enthalten und den Informationsgehalt aller im Spiel möglichen Zustände annehmen können. Des Weiteren können Relationen zwischen den Klassen definiert werden, die Zugehörigkeiten oder Referenzierungen bedeuten können.
In unserem Fall besteht die Datenschicht aus dem Spieler (User), dem virtuellen Charakter, den er spielt (Character), den Sprossen (Scion), den Sekurit-Linien (Sekurit), den Spielvariablen (GameVariable) und Systemnachrichten (Log), die zur Analyse von Vorgängen im Server dienen.
Der „User“ bezeichnet den realen Spieler, der sich am System anmelden kann, um Aktionen im Spiel auszuführen. Von einem Nutzer werden Informationen zur Anmeldung (Benutzername, Kennwort, Registrationsdatum) gespeichert, sowie der Zustand, ob der Nutzer aktiv im Spiel teilnimmt. Die Administratoren müssen sich natürlich vorbehalten, Nutzer zu deaktivieren, die sich regelwidrig verhalten. Vom Passwort wird nur der Hashwert aus einer Einweg Hashfunktion gespeichert. Somit wird verhindert, dass ein schädlicher Eindringling das Klartext-Passwort eines Spielers (auf Anhieb) erlangen kann. Das Team-Logo wird ebenso in der Klasse eines Benutzers gespeichert. Da der Spieler einen virtuellen Charakter spielt besteht eine Eins-zu-Eins (OneToOne) Relation zur Klasse „Character“, wobei der User natürlich der Eigentümer des Charakters ist.
Der „Character“ bezeichnet die virtuelle Darstellung des Spielers. Es ist der Avatar, der über der Home-Position schwebt. In dieser Klasse wird demnach die GPS-Position des Avatars gespeichert, natürlich der aktuelle Energie-Level, eine Liste von Zuständen, die ein Charakter annehmen kann und das/die Ausbreitungsgebiet(e). Zu der vorhandenen Relation zum Besitzer kommen noch weitere Relationen zu anderen Klassen hinzu. Der Charakter wird zum Besitzer aller Sprossen, die ein Spieler in das Spielsystem hochlädt. Damit besteht eine Eins-zu-Viele (OneToMany) Beziehung zu den eigenen Sprossen. Selbiges gilt für die Sekurit-Linien, die ein Spieler kreiert hat. Um verortete Sprossen

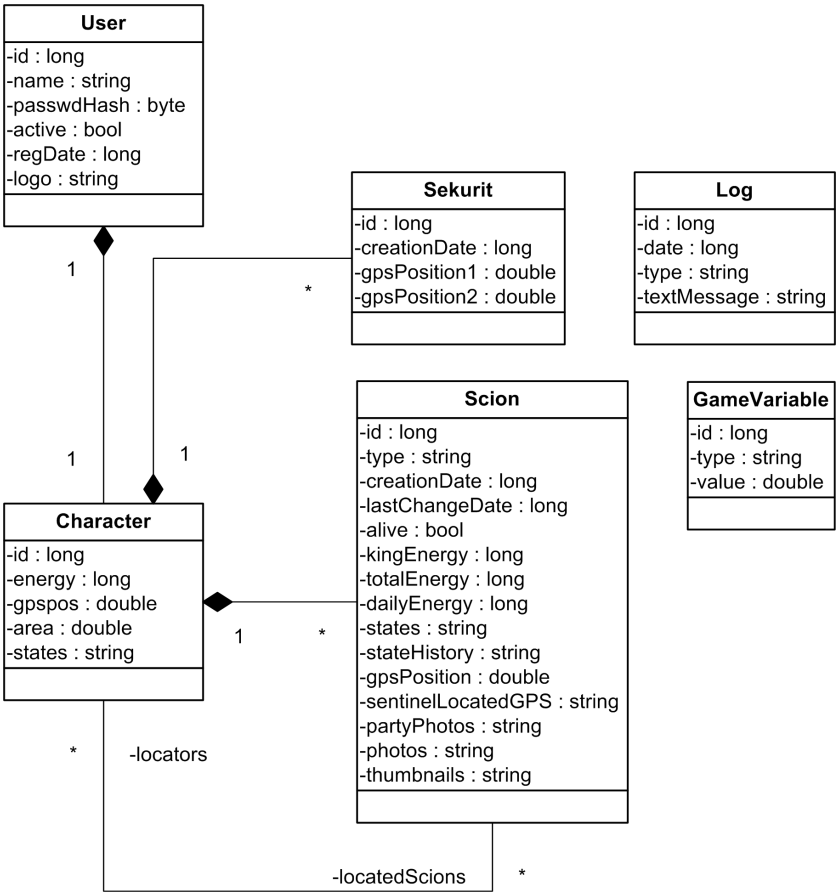

Abbildung 4.1: Klassendiagramm

abzuspeichern, wird eine Viele-zu-Viele (ManyToMany) Beziehung einem Charakter hinzugefügt. Da ein Charakter mehrere Sprossen verorten kann, und eine Sprosse auch von mehreren Charakteren verortet werden kann, muss eine Viele-zu-Viele Beziehung erstellt werden.
Der „Sekurit“ ist die einfachste Klasse. Sie speichert lediglich das Erstellungsdatum und die beiden GPS-Koordinaten, über die die Sekurit-Linie definiert ist. Eine Referenz zu dem besitzenden Charakter schließt die Klasse ab.
Die „Scion“ bezeichnet die virtuelle Darstellung eines realen Sprossen-Objekts. Da sich das Spiel hauptsächlich auf das Erstellen, Interagieren und Zustandsveränderungen von und mit Sprossen konzentriert, ist dies die wichtigste, aber auch größte/mächtigste Klasse. Zu einer Sprosse wird das Erstellungsdatum gespeichert, es muss der Typ der Sprosse, das Foto und die GPS-Position, sowie alle Statusveränderungen festgehalten werden. Das Datum über die letzte Veränderung wird auch gespeichert. Der Sinn dieses Parameters wird später in dieser Ausarbeitung erklärt. Es wird gesichert, ob die Sprosse lebendig oder zerstört ist. Es sind Analyse-Zwecke in der Smartphone Oberfläche für den Spieler

vorgesehen, und somit muss die total produzierte Energie, eine Historie über die Zustandsveränderungen, sowie die täglich produzierte Energie für eine Sprosse abgespeichert werden. Falls es sich um eine Sondersprosse handelt, so muss bei einer King-Sprosse die festgelegte Zusatzenergie festgehalten werden, eine Snack/Party-Sprosse muss zusätzlich zum Sprossen-Foto auch noch die Snack- & Party-Fotos festhalten. Falls es sich um eine Sentinel-Sprosse handelt, so müssen Aufzeichnungen über Benutzeraktivitäten im sichtbaren Umfeld dieser Sprosse abgespeichert werden. Die Relationen zu anderen Klassen wurden bereits genannt. Natürlich sind Relationen zum Besitzer sowie zu allen Charakteren festgehalten, die die Sprosse verortet haben.
Als „GameVariable“ bezeichne ich alle Parameter, die im Spiel festlegbar und änderbar sind. So zum Beispiel gibt es Variablen, die die Kosten für die einzelnen Sprossen definieren. Die Distanzen, die zur Verortung gelten sollen und Kosten für Aktionen, sind in den Instanzen dieser Klasse einstellbar. Dazu benötigt eine Instanz dieser Klasse lediglich einen Typ und einen Wert.

4.7.2 Datenschicht – Fotos

In den vorherigen Absätzen wurde häufig über „Fotos speichern“ gesprochen. Zur Abspeicherung habe ich grundsätzlich zwei verschiedene Möglichkeiten in Betracht gezogen. Entweder ein Foto wird als großes Objekt (BLOB – Binary Large Object) in die Datenbank persistiert oder auf die Festplatte geschrieben. Im zweiten Fall muss dann die Pfadangabe als Zeichenkette in die Datenbank geschrieben werden. Ein großes Objekt in der Datenbank erhöht den Rechenaufwand für das Datenbanksystem, da große Objekte eine oder mehrere Typumwandlungen durchfahren müssen. Dies wird beim Schreiben und Lesen aus der Datenbank nötig. Bei einer Speicherung auf der Festplatte kann das Lesen und zur Verfügung stellen der Datei von anderen Programmen übernommen werden. Die Installation eines erprobten Webservers kann zur Dateiveröffentlichung Kompressions- und Cachingverfahren einsetzen, die sowohl den Kommunikationsaufwand, als auch die Antwortzeit senken können. Jedoch hat dies den Nachteil, dass man möglicherweise einen höheren Wartungsaufwand in Kauf nehmen muss. Ich habe mich trotzdem für die zweite Methode entschieden. Die Senkung des Übertragungsaufwands ist für das Projekt eindeutig wichtiger, als ein eventuell erhöhter Wartungsaufwand für den Administrator. Als Webserver wurde von mir der Apache HTTP Server auf dem Betriebsystem installiert und konfiguriert. An das Smartphone wird somit über den WebService nur eine URL gesendet, von der das angeforderte Bild geladen werden kann.
Es ist außerdem vorgesehen, dass nicht nur das aktuellste Foto einer Sprosse im System verfügbar sein muss, sondern auch alle vorherigen Fotos. Dies bedeutet, dass von einer Sprosse, die getötet und im Anschluss wiederbelebt wird, bereits 3 Fotos im System existieren müssen. In gewissen Fällen ist es besser, nur ein Vorschaubild (Thumbnail) des richtigen Fotos zu übertragen. Das Senden von Vorschaubildern spart Zeit, erhöht jedoch den Speicheraufwand und geringfügig auch die Speichermenge auf der Festplatte. Doch das ist ein Kompromiss, den das Serversystem gerne eingehen wird.

4.7.3 Logikschicht

Die Logikschicht hat die Aufgabe das Spiel zu leiten, indem es die Spielelemente steuert, Zugriffe kontrolliert und Änderungen an der Datenschicht vornimmt. Im Fall von „Cityy“ übernimmt die Logikschicht zusätzlich noch ein Protokoll-System, um über wichtige server-interne Vorgänge zu informieren.

4.7.4 Logikschicht – Energieproduktion

Die Sprossen produzieren zehnminütlich eine Energiemenge, die einerseits von der Art und dem Standort, aber auch einem Zufallsgenerator abhängt. Für jede Art von Sprosse ist ein zufälliger Faktor täglich zu belegen. Im Spielsystem vorgesehen, aber bisher noch nicht erwähnt, ist die Planbarkeit über genaue Energieproduktion einer Sprosse nicht voraussagen zu können. Dazu verwende ich den TimerService, der im EJB3 Standard definiert ist. 24-stündlich wird er ausgelöst, und bewirkt eine Neubelegung der Zufallsfaktoren. Diese kommen als Spiel-Variablen (siehe 4.7.1, Klasse „GameVariable“) vor. Die Energieproduktion einer Sprosse benutzt ebenso einen Timer aus dem TimerService. Nach Ablauf eines Timers reaktiviert sich dieser und fügt eine Nachricht in die JMS (Java Message Service) Warteschlange ein. Der Timeout-Handler hat selbst nicht die Möglichkeit in den Persistenz-Kontext einzugreifen. Da ihm somit Datenbank-Zugriffe verwehrt bleiben, wird eine Message Driven Bean die Benachrichtigung aus der Warteschlange nehmen und eine Aktualisierung der Energie-Level aller Benutzer, sowie die Neubelegung der Zufallsfaktoren veranlassen.

4.7.5 Logikschicht – Verortungen

Sondersprossen bekommen eine leicht abgeänderte Behandlung im Spielsystem. Das Einfügen einer Snack- oder Partysprosse soll genauso schnell von statten gehen, wie das Einfügen einer Standardsprosse. Doch gerade die Snack- und Partysprossen müssen bei jedem Mitspieler verortet sein, damit er sie sofort auf der Karte erkennt, und gegebenenfalls Aktionen an der jeweiligen Sprosse ausführen kann. Um also eine Sprosse dieser Art bei allen Mitspielern zu verorten, wird beim Einfügen eine Nachricht in die JMS Warteschlange gelegt, die verspätet abgearbeitet werden kann. Eine Message Driven Bean sorgt im Anschluss für das Herstellen der geforderten Beziehungen zwischen allen Charakteren und der neu eingefügten Sondersprosse.
Verortungen treten auch bei Sentinel-Sprossen und speziellen Aktionen auf (siehe 2.2, „Tracker“). Im Spiel ist vorgesehen, dass in regelmäßigen Abständen die Smartphones Spielobjekte in ihrer Umgebung abfragen. Dazu muss an den Server die aktuelle GPS-Position des Smartphones gesendet werden, damit eine Liste von benachbarten Objekten erzeugt und geantwortet werden kann. Bei Abfragen dieser Art wird nach selbigem Vorgehen eine Nachricht in die JMS Warteschlange gelegt. Eine Message Driven Bean wird dann verspätet einerseits alle benachbarten Sentinel-Sprossen über Mitspieler-Aktivitäten

unterrichten, und andererseits die Relationen zu bisher nicht verorteten Sprossen herstellen.

4.7.6 Logikschicht – Protokoll-System

Um Vorgänge innerhalb des Serversystems analysieren zu können existiert ein System, das auf dem selben Wege Benachrichtigungen in die Datenbank einpflegt wie das Verortungssystem. Während der Abarbeitung von Spielaktionen werden Nachrichten in die JMS Warteschlange gelegt, die dann durch eine Message Driven Bean verspätet entnommen und deren Inhalt nach Typen sortiert in die Datenbank persistiert werden.

4.7.7 Präsentationsschicht – Rollen

Die Kontaktaufnahme mit dem Spielsystem orientiert sich nach Absicht und Rolle des Clients. Die Clients werden dabei in zwei Gruppen unterteilt.
Zum Einen gibt es die Administratoren, denen ein AdminTool zur Verfügung steht und die sich im physikalisch gleichen Netzwerk wie der Server befinden. Damit stehen ihnen die Benutzung von RMI-IIOP über die spezifizierten Netzwerk-Ports zur Verfügung. Um eine Administration über einen sicheren Kanal auch von außerhalb des physikalischen Netzwerks zulassen zu können, wurde ein VPN-Server auf dem Serversystem installiert und die Clients mit nötigen Zertifikaten und Schlüsseln versorgt.
Die andere Rolle wird von den Spielern eingenommen. Diese befinden sich außerhalb des physikalischen Netzwerks des Serversystems, und es wird eine Netzwerkadress-Übersetzung (NAT) benötigt. Dies erlaubt, dass die Clients nur Netzwerk-Ports für eingehende Netzwerkverbindungen benutzten können, die der Router erlaubt.
Da zwei verschiedene Rollen mit unterschiedlichen Rechten existieren, werden demnach auch zwei unterschiedliche Schnittstellen vom Application Server zur Verfügung gestellt. Die WebServices-Schnittstelle der Spieler (UserInterface) beruht auf der selben Implementierung, wie die RMI-IIOP Schnittstelle der Administratoren (AdminInterface). Jedoch ist sie mit einem eingeschränkteren Funktionsumfang versehen, als es den Administratoren zur Verfügung steht. Die Wahl der selben Funktions-Implementierung erleichtert die Wart- und Überprüfbarkeit des Gesamtsystems.
Den Spielern werden Funktionen zur Verfügung gestellt, die ihnen gestatten Änderungen an ihrem eigenen Account vorzunehmen (Passwort, Logo, Home-Position), öffentliche Informationen über Mitspieler abzurufen, definierte Aktionen im Spiel ausführen zu können, sowie Daten über ihre eigenen Spielobjekte abzurufen. Den Administratoren werden zusätzlich noch Möglichkeiten zum Management des Spiels und der Spielobjekte eingeräumt (Spiel-Variablen anpassen, Nutzeradministration, manuelle nicht Aktions-basierte Sprossenänderungen), sowie das Auslesen des Protokoll-Systems.

4.7.8 Präsentationsschicht – Spielobjekte

Wie in den Anforderungen definiert versuchen wir den Kommunikationsaufwand möglichst klein zu halten. Das Smartphone wird nach Möglichkeit versuchen alle nötigen Informationen über Spielobjekte in einem im Smartphone-Programm enthaltenen internen Datenbanksystem zu speichern. Dazu zählen komplette Sprossenobjekte und vor Allem deren Fotos.
Fragt ein Client also die Sprossen in seiner Umgebung ab, so bekommt dieser lediglich eine Liste von IDs der Sprossen, und deren letztes Änderungsdatum. Ein kurzer Abgleich mit den Daten in seiner Datenbank kann einen erhöhten Kommunikationsaufwand ersparen. Sollte eine Sprosse mit der übermittelten ID nicht existieren, so kann das Smartphone diese vom Serversystem abfragen. Eine Instanz einer Sprosse muss zunächst serverseitig manuell zu einer Zeichenkette kodiert werden, die clientseitig zu einem Sprossen-Objekt dekodiert wird. Gleiches gilt, wenn das Änderungsdatum mit dem aus der internen Datenbank nicht übereinstimmt.
Diese Möglichkeit der Zwischenspeicherung ist natürlich nur optional, und stellt keine Einschränkung an die Auswahl der zugelassenen Clients dar. Sollte ein Smartphone kein internes Datenbanksystem besitzen, so kann dieses Gerät trotzdem benutzt werden. Übermittelte Informationen liegen bei solchen Geräten im flüchtigen Speicher, und gehen bei Beendigung des Programms verloren. Das Serversystem übermittelt beim Reaktivieren des Programms alle benötigten Informationen und Spielobjekte erneut an den Client.
Im Kapitel über die Datenschicht wurde das Vorgehen zu Fotos bereits erläutert. Es gibt grundsätzlich zwei Kommunikationswege, die ein Foto in unserem Spielsystem gehen muss. Zunächst muss ein Foto einer neuen Sprosse an den Server gesendet werden. Der Server speichert dieses Foto dann in einen Ordner auf der Festplatte, den der WebServer per HTTP zur Verfügung stellt. Ein Client muss in der Lage sein ein Foto vom Server abzufragen, welches den zweiten Kommunikationsweg eines Fotos beschreibt.
Fotos sind binäre Dateien, die je nach Format stärker oder schwächer komprimiert werden können. Dazu existieren verlustfreie und verlustbehaftete Kompressionsverfahren. Verlustbehaftete Verfahren können zum Beispiel die Auflösung oder Farbtiefe des Bildes manipulieren, um eine kleinere Datei zu erzeugen. Als Projektvorgabe wurde definiert, dass verlustbehaftete Kompressionsverfahren zur Senkung des Kommunikationsaufwands ausscheiden. Mit dieser Einschränkung bestehen nur noch wenige Möglichkeiten Kompressionsverfahren einzusetzen. Wie schon im Kaptitel über Fotos in der Datenschicht erwähnt, löst der Apache HTTP WebServer das Übertragen eines im Spielsystem vorhandenen Fotos an einen Client. Dazu stehen erprobte Verfahren zur Kompression und zum Caching zur Verfügung. Das Auslesen des Fotos aus dem Datenbanksystem, sowie das korrekte Typ-Umwandeln und ggf. Komprimieren des Fotos ist somit keine Aufgabe des Application Servers mehr. Tests ergaben, dass dieser Vorgang deutlich länger dauert. Doch einmalig muss das Foto vom Smartphone an das Spielsystem übermittelt werden. Da der Kommunikationsweg mit dem Nutzen eines WebServices bereits fest steht sind die Möglichkeiten zur Kompression noch weiter eingeschränkt. Demnach besteht die Schwierigkeit darin, ein binäres Objekt zu einer Zeichenkette zu wandeln, die so wenig

Zeichen wie möglich benutzt. Durch das Benutzen von SOAP (XML über HTTP) besteht keine Möglichkeit, das Foto als binäres Objekt über einen Socket zu transferieren. Eine Umwandlung zu einer Zeichenkette bleibt unvermeidbar. Das verbreitetste Vorgehen ist das Benutzen des Base64-Verfahrens. Dabei werden 64 druckbare ASCII-Zeichen zur (De-)Kodierung benutzt, damit auch Systeme oder Protokolle das Verfahren benutzen können, die keinen kompletten ASCII-Zeichensatz verarbeiten können.
Das Base64-Verfahren ist seit November 1996 standardisiert und zählt zu den populärsten Kodierungsverfahren, das den Zweck des Text-basierten Datenaustauschs erfüllt. Leider entstehen durch das Umwandeln einer binären Datei in etwa 33% mehr Bits, die transferiert werden müssen. Laut dem W3C Gremium wird seit Januar 2005 für moderne WebServices die Verwendung von MTOM (Message Transmission Optimization Mechanism) empfohlen. MTOM benutzt XOP (XML-binary Optimized Packaging), um binäre Daten in SOAP Nachrichten einzubinden. Da die bereits angesprochene ksoap2 Bibliothek die Verwendung von MTOM nicht unterstützt, jedoch das Base64-Verfahren als übliche Technik im MTOM-Mechanismus empfohlen wird, fiel die Wahl auf ein manuelles Kodieren und Dekodieren von binären Daten durch das Base64-Verfahren.

4.7.9 Präsentationsschicht – Funktionstests

Um die Korrektheit der Logikschicht zu testen, wurden im Projekt zahlreiche Funktionstests entwickelt. Diese benutzen das JUnit Framework, welches die Rückgabewerte und -objekte darauf überprüft, ob sie den definierten erwarteten Resultaten entsprechen. Die Tests wurden so entworfen, dass Änderungen an der Logikschicht einfach und schnell auf korrekte Funktionsweise überprüft werden können.

4.8 AdminTool

Um den Administratoren eine Möglichkeit zum Überwachen und Steuern des Spielgeschehens zu geben, wurde von mir ein Programm mit grafischer Oberfläche (GUI – graphical user interface) entworfen. Da der Application Server das Verwenden des RMI-IIOP Mechanismus' unterstützt, fiel die Wahl auf eine java-basierte Oberfläche. Natürlich existieren zahlreiche GUI-Builder, die das Erstellen von grafischen Oberflächen erleichtern. Ich entschied mich schlussendlich dagegen, eines dieser Werkzeuge zu benutzen. Ich sehe mich dem Problem gegenüber ein GUI zu entwickeln, das gut aussieht und für dynamisch ändernde Inhalte eine problemlose Bedienung vorsieht.
Meine Entscheidung fiel darauf den Kern der Programmiersprache Processing (`http://www.processing.org`) für diesen Zweck zu benutzen. Processing fasst Komponenten aus der „java.awt“ (Abstract Window Toolkit) Bibliothek für den Benutzer zusammen, die dazu benutzt werden, eine Zeichenoberfläche mit grafischen Objekten zu füllen. Dazu stehen einfache Funktionen zum Darstellen geometrischer Figuren, wie Kreisen, Vierecken, Linien oder Dreiecken zur Verfügung. Natürlich existieren auch Funktionen zum Anzeigen von Texten, Bildern und Videos. Processing bietet die Möglichkeit, eine spezielle

Funktion mit einer zeitlich definierbaren Rate („FrameRate“ - Bildfrequenz) aufzurufen. Dies hilft dem Programmierer, Änderungen an den dargestellten geometrischen Objekten und Fotos auf der Oberfläche in dieser Funktion vorzunehmen. Durch passende Benutzung dieser Funktion lassen sich somit sowohl einfache, als auch komplexe Animationen erstellen.
Das Programm lässt sich in die Kategorien Spieladministration, Sprossen-Management, Nutzerverwaltung, Sekurit-Management und Protokoll-System unterteilen. Die Spieladministration beschränkt sich auf das Starten und Stoppen des Spiels, sowie die Möglichkeit Änderungen an den Spiel-Variablen vornehmen zu können.

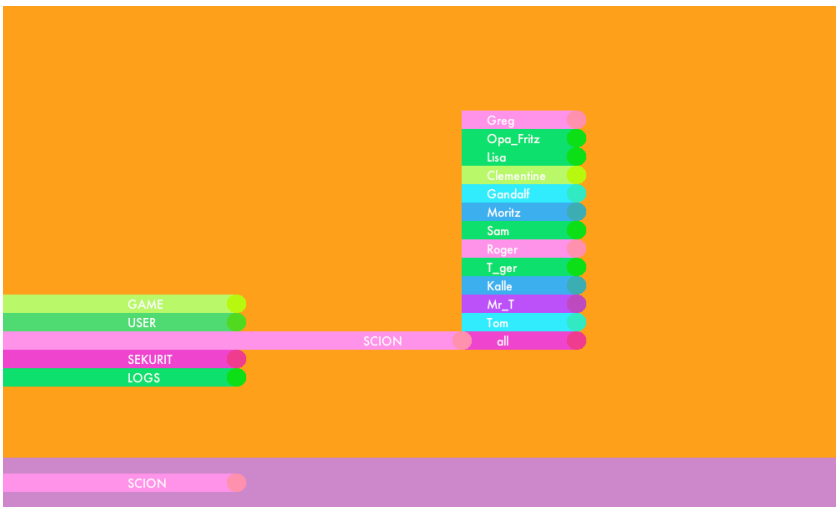

Abbildung 4.2: AdminTool – Startseite, Kategorie „SCION“ ausgewählt

Auf der Startseite befinden sich die fünf genannten Kategorien. Da das Menü in einer Baumstruktur aufgebaut wurde, kann mittels Pfeiltasten einfach navigiert werden. Dynamisch ändernde Inhalte mussten im Programm beachtet werden. Wie auf Abbildung 4.2 erkennbar, sind in der zweiten Ebene des Baums im Zweig „SCION“ alle Benutzer auswählbar. Doch die Anzahl der Benutzer kann variieren. Das Auswählen eines Objektes aus der zweiten Ebene führt zum Wechsel der Seite zur ausgewählten Administrationsebene. Diese werden auf den folgenden Abbildungen gezeigt.

Abbildung 4.3: AdminTool – Spiel starten/stoppen

Auf der Administrationsseite „GAME → start/stop“ (Abbildung 4.3) kann das Spiel gestartet und gestoppt werden. Der eventuelle Start-Zeitpunkt und eine minimale Statistik über die Anzahl der Spieler und Sprossen im System wird angezeigt.

Abbildung 4.4: AdminTool – Spielparameter anpassen

Auf der Administrationsseite „GAME → parameters“ (Abbildung 4.4) können sämtliche Spielparameter angepasst werden.

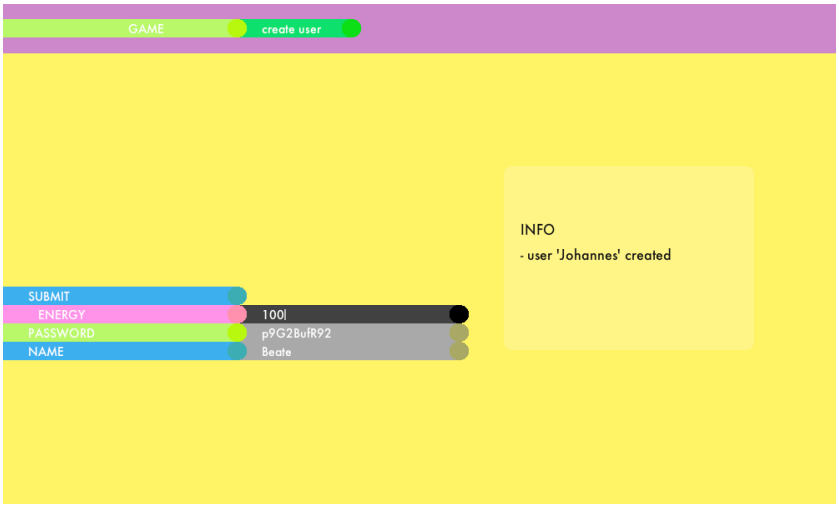

Abbildung 4.5: AdminTool – Spieler anlegen

Das Anlegen von Spielern erfordert einen Benutzernamen, ein Passwort, sowie das Start-Energie-Level und kann auf der Administrationsseite „GAME → create user“ (Abbildung 4.5) getätigt werden. Das Löschen eines Spielers kann unter dem Punkt „GAME → delete user“ veranlasst werden.

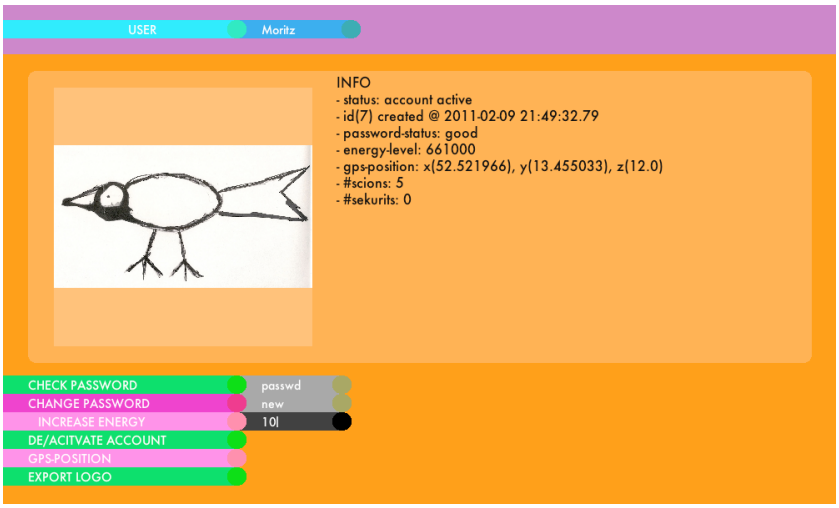

Abbildung 4.6: AdminTool – Nutzeradministration

Die Nutzeradministration (Abbildung 4.6) umfasst Möglichkeiten zur Überprüfung und Änderung des Passwortes. Es existieren Funktionalitäten zum manuellen Ändern des Energie-Levels seines Charakters, die Home-Position kann auf einer Karte angezeigt werden und das Team-Logo des Spielers kann auf die Festplatte des Administrators exportiert werden.

Abbildung 4.7: AdminTool – Sprossenadministration

Die Administrationsseite für Sprossen (Abbildung 4.7) umfasst den größten Funktionsumfang. Zunächst muss in einer Zeile ein Zeitpunkt eingegeben werden, ab dem alle Sprossenobjekte angezeigt werden sollen. Werden in der Liste Sprossen angezeigt, so besteht die Möglichkeit, die Foto-Historie dieser Sprossen zu exportieren. Nach Auswählen einer Sprosse können die Zustandsveränderungen Boost, Freeze, Parasit, Sprengfalle und Tarnkappe hinzugefügt bzw. entfernt werden. Verstoßen Sprossen gegen die Bestimmungen oder wurden von einem Mitspieler unvorschriftsmäßig umgebracht, so haben die Administratoren hier die Möglichkeit, Sprossen wiederzubeleben oder zu töten. Das Festlegen der küstlerischen Wertigkeit für die Sondersprosse „King“ kann ebenso auf dieser Seite vorgenommen werden. In Abbildung 4.7 wurde der Wert auf 70 gesetzt. Die letzte Option, die den Administratoren gegeben wird ist das Anzeigen der GPS-Koordinate auf einer Karte.

Abbildung 4.8: AdminTool – Sekuritadministration

Die Seite zum Administrieren der Sekurit-Linien (Abbildung 4.8) dient lediglich zum Anzeigen und gegebenenfalls Löschen einer Sekurit-Linie.

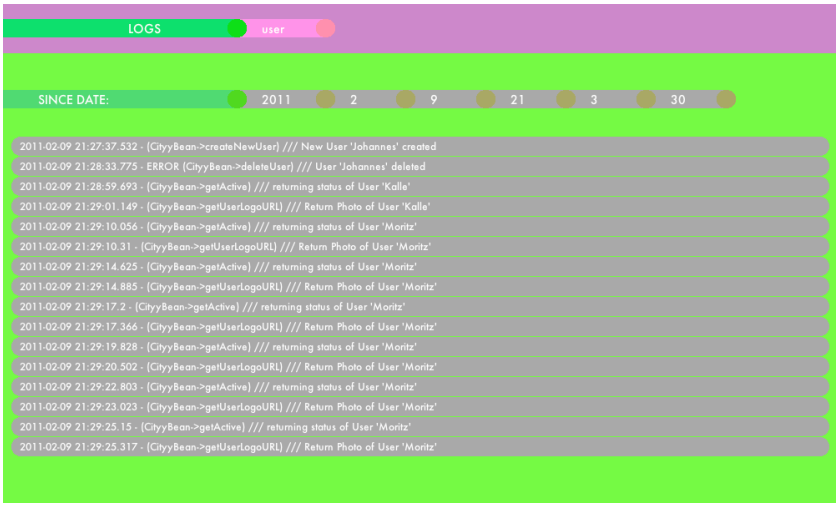

Abbildung 4.9: AdminTool – Protokoll-System

Wie schon auf der Sprossen-Administrationsseite (Abbildung 4.7) gibt man auch auf der Protokoll-System-Seite (Abbildung 4.9) ein Datum ein, ab welchem Zeitpunkt man die Logs angezeigt bekommen möchte. Mehr als durch die Logs zu Blättern ist nicht vorgesehen.

5 Performance

Um Schwachstellen im System zu erkennen und möglicherweise zu verbessern, habe ich zahlreiche Performance-Tests durchgeführt. Die Referenz-Hardware und das zu Grunde liegende Betriebs- und Softwaresystem wurden in Kapitel 4.1 vorgestellt. Der Application Server wird vor die Aufgaben gestellt, parallel auftretende Anfragen beantworten zu müssen. Die Antwortzeit wird dabei notiert.

5.1 Transportschicht

Die Tests fanden in einem Programm auf dem selben Computer statt. Somit wird die Transportschicht nicht beachtet. Dies kann natürlich dazu führen, dass Anfragen erst später beantwortet werden, als in den Tests heraus gekommen ist. Eine langsame Internetanbindung der Smartphones kommt häufig vor. Doch würde eine Einbeziehung der Transportschicht die Resultate verändern, da eine zufällig auftretende Verminderung der Antwortzeit Aussagen über die Performance des Systems verfälscht.

5.2 Datenbanksystem

Das Datenbanksystem MySQL wurde in Kapitel 4.4 erwähnt und ist auch in den Performance-Tests aktiv. Würde man das Datenbanksystem austauschen, könnte es dazu führen, dass Aufgaben schneller oder langsamer abgearbeitet werden können. Das System wurde daher so entworfen, dass seine Komponenten austauschbar bleiben. In keinster Weise wird verlangt, dass das zu Grunde liegende Datenbanksystem eine MySQL Datenbank sein muss.

5.3 Kategorisierung

Die Aufgabengebiete des Application Servers können in drei Kategorien unterteilt werden. Die nach Erwarten am häufigsten benutzte Kategorie ist das Lesen von Daten aus der Datenbank und Übermitteln an den Client. Das zweit-meistbenutzte Aufgabengebiet liegt darin, Daten in der Datenbank zu manipulieren. Und die kompliziertesten Aufgaben gehören in die Kategorie „schreibend auf die Datenbank zugreifen, während man gleichzeitig noch Datei-Ausgaben tätigt“. Diese Kategorie betrifft ausschließlich das Übermitteln von Fotos, die nach Dekodierung auf die Festplatte geschrieben werden müssen. Außerdem muss der Datenbank ein Vermerk über den Verbleib des Fotos hinzugefügt werden.

Zu jeder der drei Kategorien werden die Resultate von drei bis vier Tests vorgestellt. In der Kategorie der lesenden Datenbankzugriffe werden zusätzlich noch zwei weitere Resultate vorgestellt, die vor der Übermittlung der Daten einen gewissen Rechenaufwand an den Server stellen.

5.4 Testablauf

Da die Implementierung des AdminInterfaces via RMI-IIOP äquivalent zum UserInterface via WebService (siehe Kapitel 4.7.7 „Präsentationsschicht – Rollen“) ist, werden die Tests über das AdminInterfaces getätigt. Diese Entscheidung wird das Testresultat eher positiv beeinflussen, da bei der UserInterface Schnittstelle zusätzlicher Aufwand zur serverseitigen Dekodierung der Anfrage nötig ist. Der Application Server muss WebService Anfragen zunächst aus einer XML-Struktur auslesen, und nach der Abarbeitung wiederum eine XML-Struktur generieren. Dies entfällt bei meinen Performance-Tests.
Das Testprogramm initialisiert parallele Threads, die im Anschluss gleichzeitig gestartet werden. Sie notieren den Zeitpunkt, bevor die definierte Anfrage an das Serversystem gesendet wird, und vergleichen den Endzeitpunkt nach Antwort des Application Servers. Dieser Vorgang wird für alle parallelen Threads 100 Mal wiederholt. Die Fülle an Objekten in der Datenbank belaufen sich auf ca. 6300 Sprossen und 6300 Sekurit-Linien, die auf 20 Benutzer aufgeteilt werden. Dabei sind die GPS-Koordinaten der Objekte zufällig über Berlin verteilt.
Die Resultate jedes Threads werden dann in einer globalen Liste gesammelt, und nach Ablauf aller Tests in eine CSV(comma-separated values)-Datei geschrieben. Die folgenden Abbildungen werden nach Einlesen und Mittelwertbildung über alle Testresultate erstellt. Jeder Punkt in den Grafiken entspricht also einem Mittelwert aus $100 * k$ notierten Resultaten, wobei k die Anzahl der Threads ist, die in diesem Test parallel gestartet wurden. Da der Application Server schon ab einer sehr geringen Anzahl an parallelen Anfragen keine Antwort mehr gab, wurden alle Tests auf 1, 2, 4, 7, 11, 16 und 22 parallelen Threads ausgeführt. Diese werden auf der X-Achse dargestellt.

5.5 Resultate – Datenbank schreibend

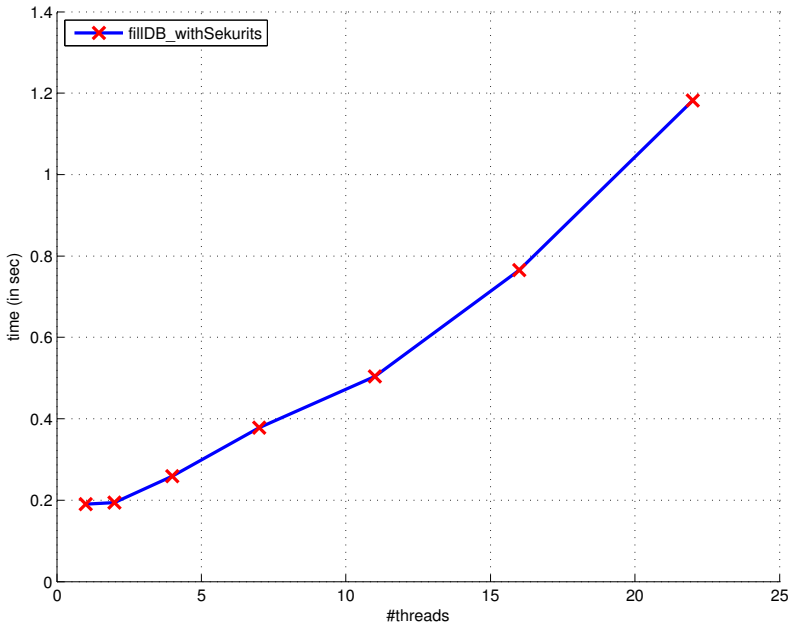

Abbildung 5.1: Performance Test: Sekurit-Linie einfügen

Die Abbildungen (5.1, 5.2, 5.3) aus dieser Komplexitätsklasse zeigen, dass keine marginalen Unterschiede zwischen den Antwortzeiten der Funktionen vorhanden sind. Bei 20 gleichzeitigen Anfragen liegt die durchschnittliche Antwortzeit stets unter einer Sekunde.

5.6 Resultate – Datenbank schreibend & Datei-Ausgaben

Die Abbildungen (5.4, 5.5, 5.6, 5.7) aus dieser Komplexitätsklasse zeigen, dass gleichwertige Härte der Aufgabenstellung unter allen ausgewählten Funktionen gleichwertige Performance-Resultate produziert. Alle vier Funktionen müssen Bilder empfangen, dekodieren, gegebenenfalls eine kleinere Version erstellen, auf die Festplatte speichern und eine verhältnismäßig kleine schreibende Operation in der Datenbank vornehmen. Damit die Funktionen untereinander vergleichbar bleiben, wurde immer das selbe Bild benutzt, das in bereits komprimierter Form ca. 1 MB groß ist. Da die Funktionen aus dieser Komplexitätsklasse am Längsten von allen Funktionen benötigen, kann man sich glücklich schätzen, dass sie zu den seltensten Aufgaben zählen, die das Serversystem gestellt bekommt.

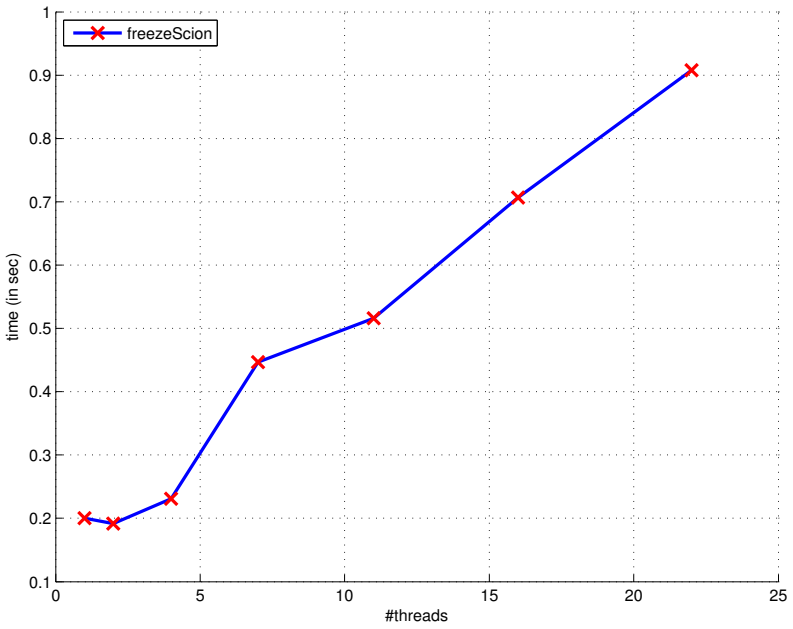

Abbildung 5.2: Performance Test: Sprosse einfrieren

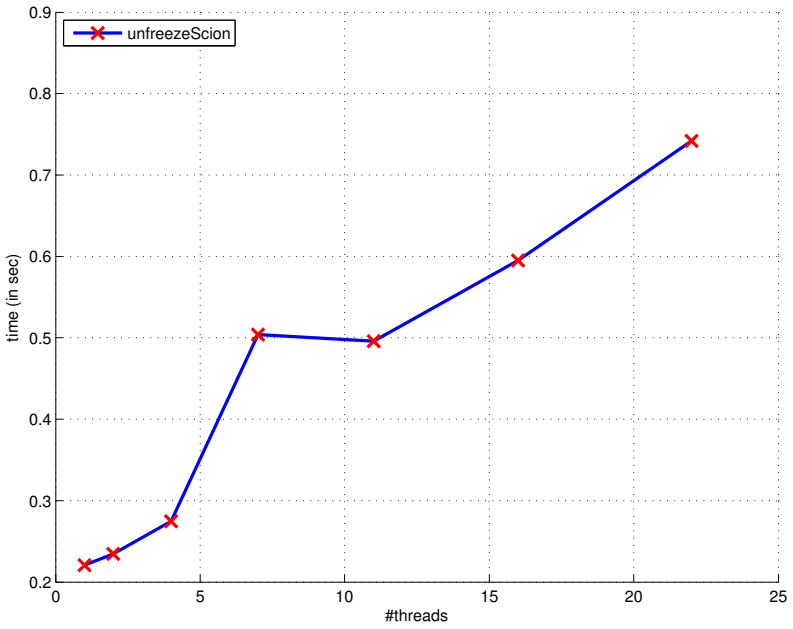

Abbildung 5.3: Performance Test: Sprosse auftauen

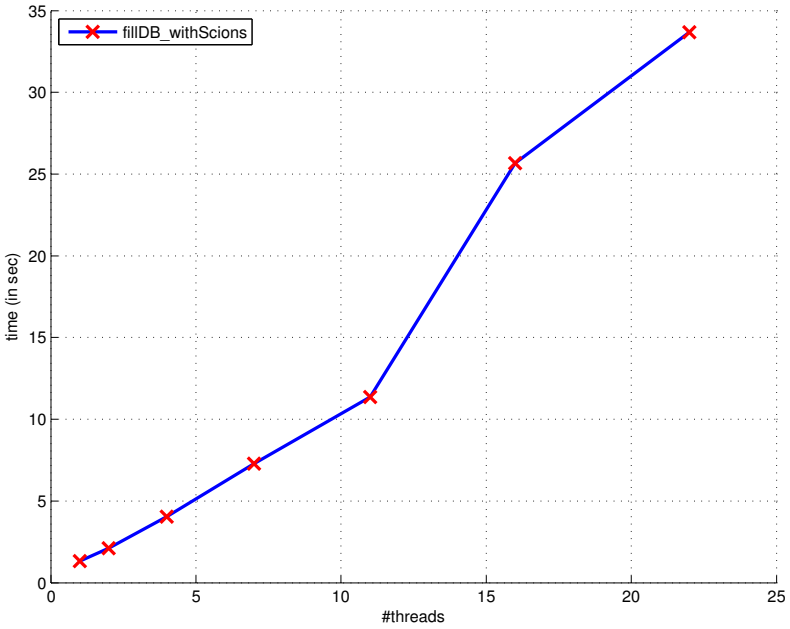

Abbildung 5.4: Performance Test: Sprosse hinzufügen

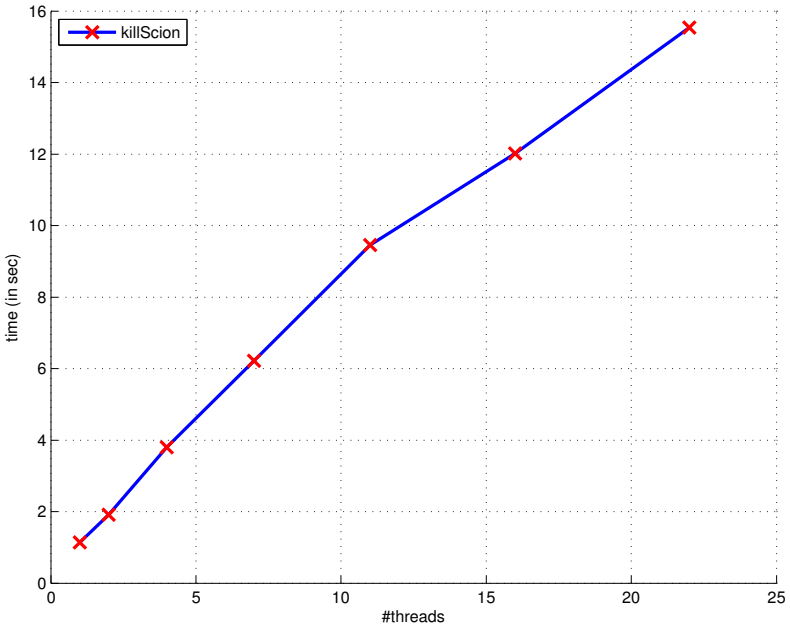

Abbildung 5.5: Performance Test: Sprosse töten

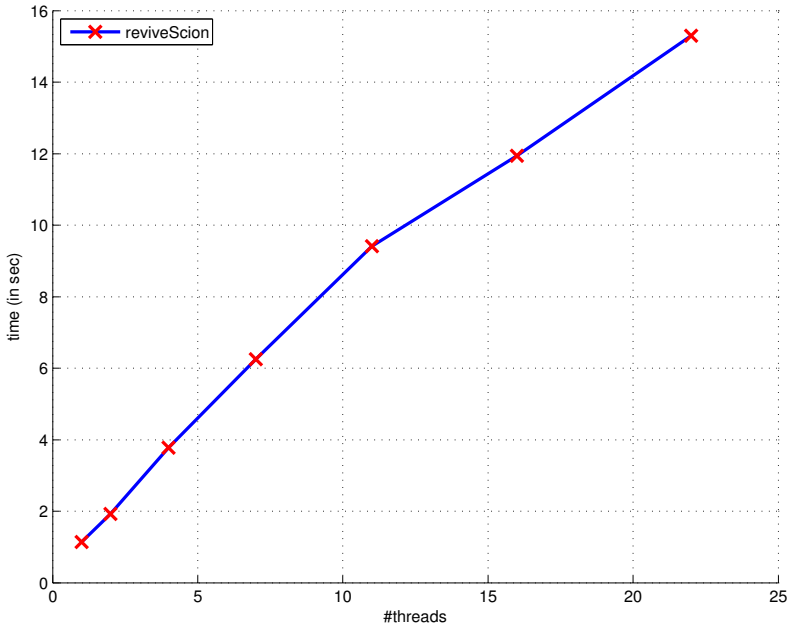

Abbildung 5.6: Performance Test: Sprosse wiederbeleben

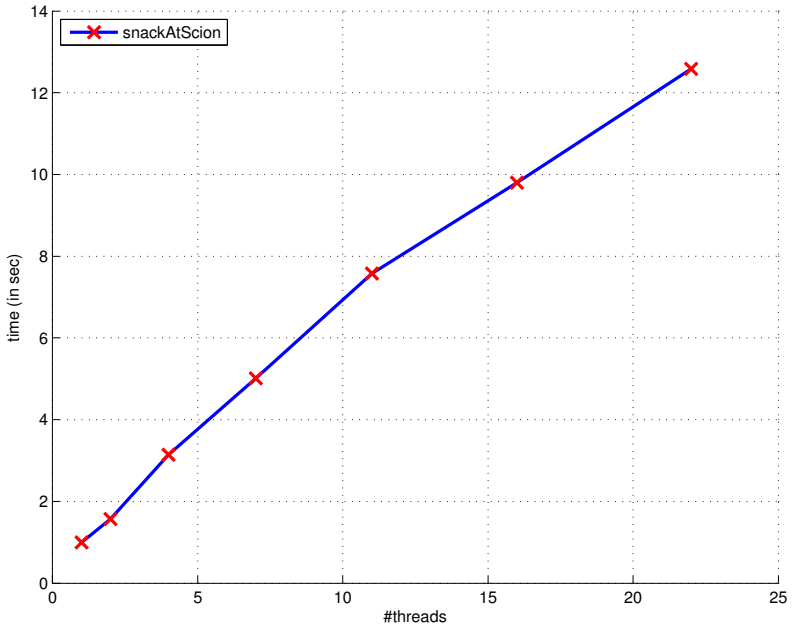

Abbildung 5.7: Performance Test: Snack an einer Sprosse

5.7 Resultate – Datenbank lesend

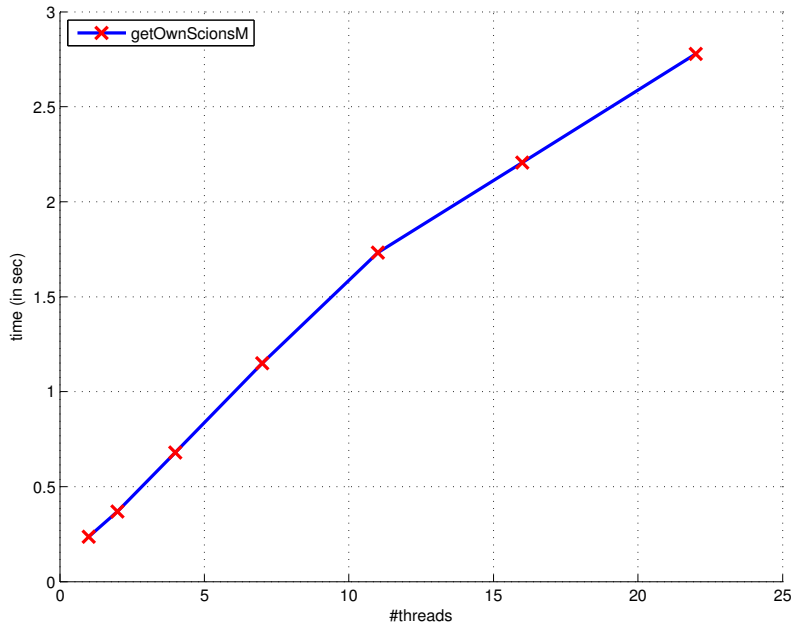

Abbildung 5.8: Performance Test: Eigene Sprossen übermitteln

Die ersten vier Tests (Abbildungen 5.8, 5.9, 5.10, 5.11) zeigen den Verlauf von einfachen lesenden Operationen. Das Auslesen aller eigenen Sprossen benötigt eine Zusammenstellung mehrerer Objekte aus der Datenbank. Daher benötigt diese Operation mehr Zeit. Die letzte Operation (Abbildung 5.12) beinhaltet nach dem Auslesen vieler Objekte aus der Datenbank eine rechenintensive Verarbeitung. Es müssen nur diejenigen Objekte an den Client gesendet werden, die in seiner näheren Umgebung sind. Da dies das Errechnen der Entfernung zwischen jedem Objekt und dem Client benötigt, ist eine höhere Bearbeitungszeit unumgänglich. Auf der Abbildung (5.12) sieht es, als würde die Antwortzeit besser werden, wenn die Anzahl der Threads größer wird. Doch dies liegt lediglich daran, dass an dieser Stelle häufig Anfragen nicht mehr Beantwortet werden, und es zu einem „Timeout“ kommt.

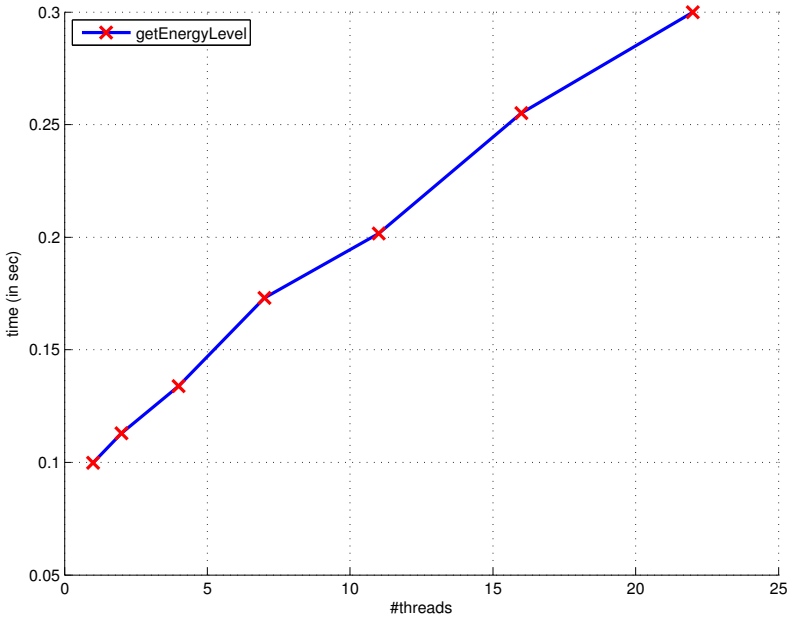

Abbildung 5.9: Performance Test: Energie-Level übermitteln

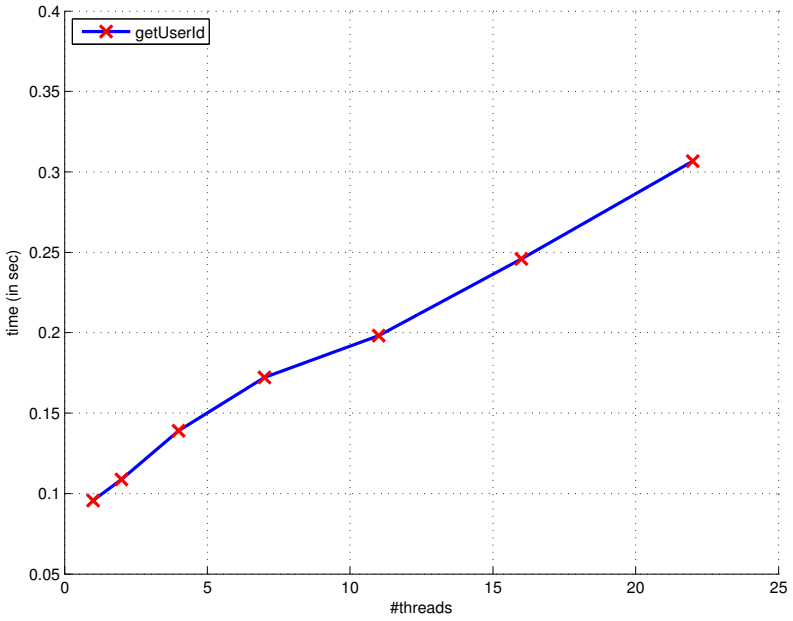

Abbildung 5.10: Performance Test: User-ID übermitteln

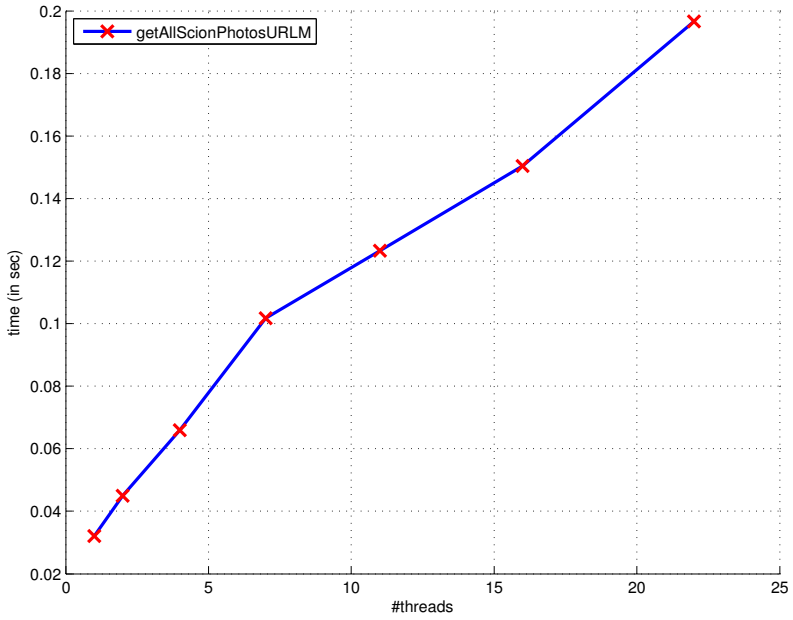

Abbildung 5.11: Performance Test: Die URLs aller Sprossen Fotos übermitteln

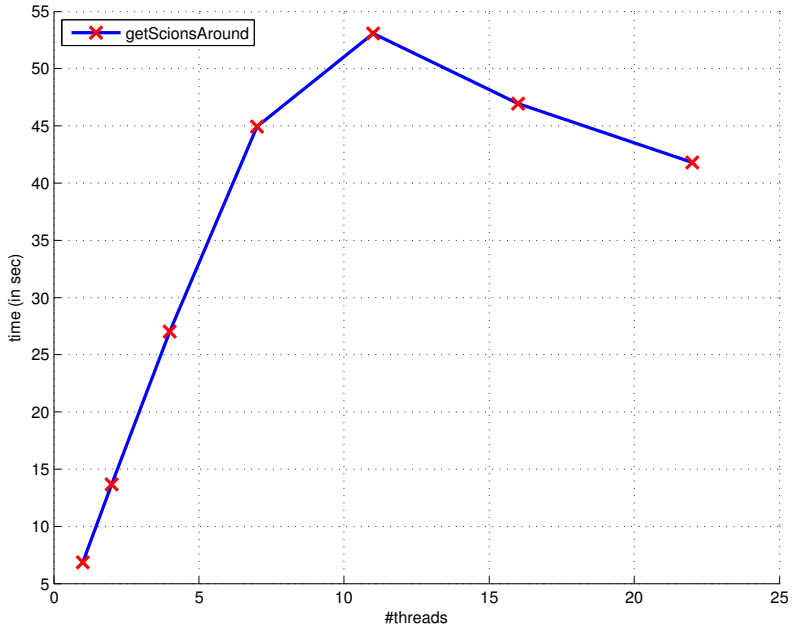

Abbildung 5.12: Performance Test: Sprossen in der Umgebung

5.8 Verbesserung Teil 1

Die Bearbeitungszeit des letzten Tests hat jedoch ein massives Problem offenbart. Die Funktion, die wohl am häufigsten benutzt wird, hat sehr schlechte Antwortzeiten. Zur Verbesserung der Situation können leider keine Hardware-technischen Änderungen vorgenommen werden. Somit fällt die Bildung der Menge der nahen Objekte auf externen Computern aus. Um auch weiterhin die zu Grunde liegende Datenbank austauschbar zu gestalten, habe ich mir überlegt, nur eine minimale Änderung in der Datenschicht vorzunehmen, und keine Datenbank-spezifischen Optimierungsmöglichkeiten zu benutzen. Es werden zwei zusätzliche Datenobjekte „ScionCluster“ und „ScionClusterArray“ hinzugefügt. Wobei Relationen von einem ScionCluster zu mehreren Scions erstellt werden.
Berlin ist ein geografisch beschränktes Gebiet. Ich teile es in 500m Abschnitte auf. Es entstehen rechteckige Gebiete zu 500m*500m. Nun werden alle Sprossen beim Einfügen in diese rechteckigen Gebiete eingeordnet. Es entstehen Cluster von Sprossen. Befinden sich Sprossen in ein und dem selben Cluster, so bekommen sie eine Referenz zu dem selben ScionCluster. Die Gesamtheit aller ScionCluster bezeichne ich als ScionClusterArray. Die Änderungen der Datenschicht führen zum Klassendiagramm in Abbildung 5.13.

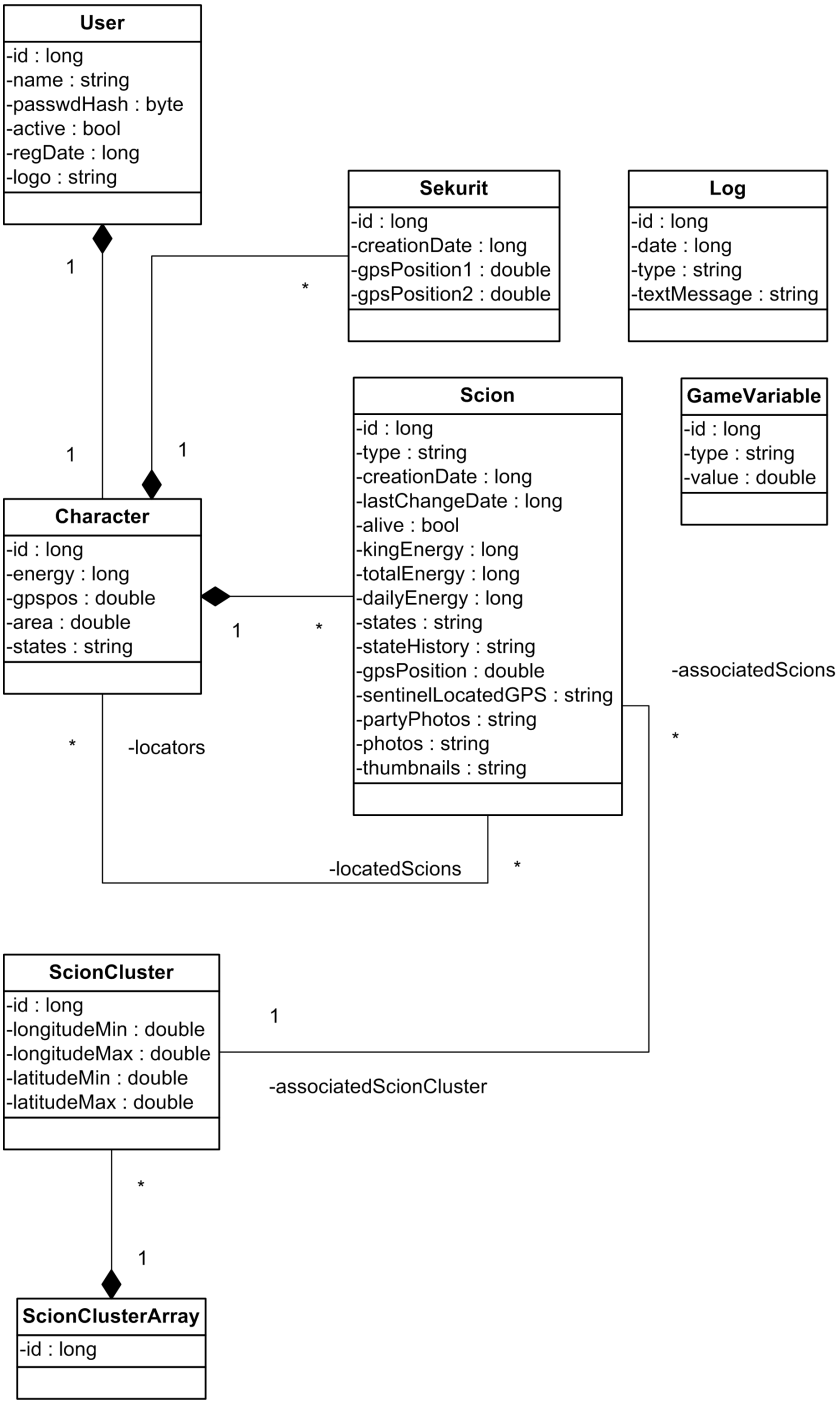

Abbildung 5.13: Klassendiagramm mit ScionCluster und ScionClusterArray

Durch das Zuordnen jeder neuen Sprosse in das passende Cluster erleichtern wir die Suche nach den Sprossen in der näheren Umgebung. Ein Suchvorgang bedeutet nun das Heraussuchen des ScionClusters aus dem ScionClusterArray, in dem sich der Client befindet (blaues Kästchen), und Bildung der Sprossenmenge in diesem und den 8 umgebenden Clustern (graue Kästchen).

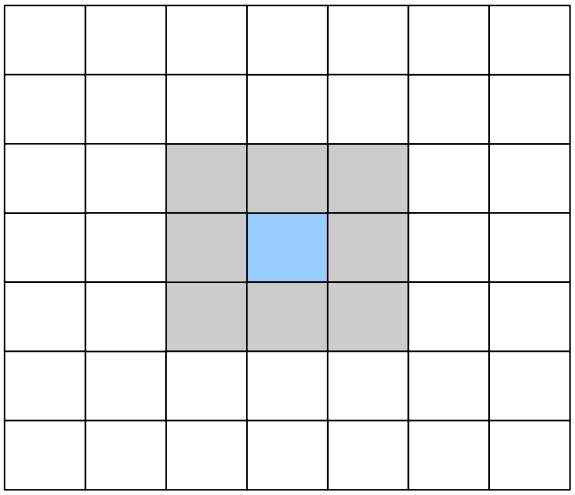

Abbildung 5.14: ScionCluster, die den Suchraum bilden

Somit wird der Suchraum auf 1,5km*1,5km eingeschränkt. Man kann davon ausgehen, dass in den weißen Kästchen keine Sprossen existieren, die nahe genug an der Client-Position dran sind. Somit müssen diese auch nicht durchsucht werden.
Die Suche mit dem Cluster Konzept benötigt zwar einen höheren Aufwand beim Heraussuchen der richtigen Cluster und darin enthaltenen Sprossen, jedoch wird der Suchraum erheblich eingeschränkt. Somit kann sich der Application Server viel Berechnungsarbeit zur Bestimmung der Distanzen zwischen zwei GPS-Positionen ersparen.
Ein zusätzlich aufgesetzter Performance-Test, der unter den selben Konditionen stattfindet wie die Vorherigen, soll den direkten Vergleich der beiden Verfahren aufzeigen.

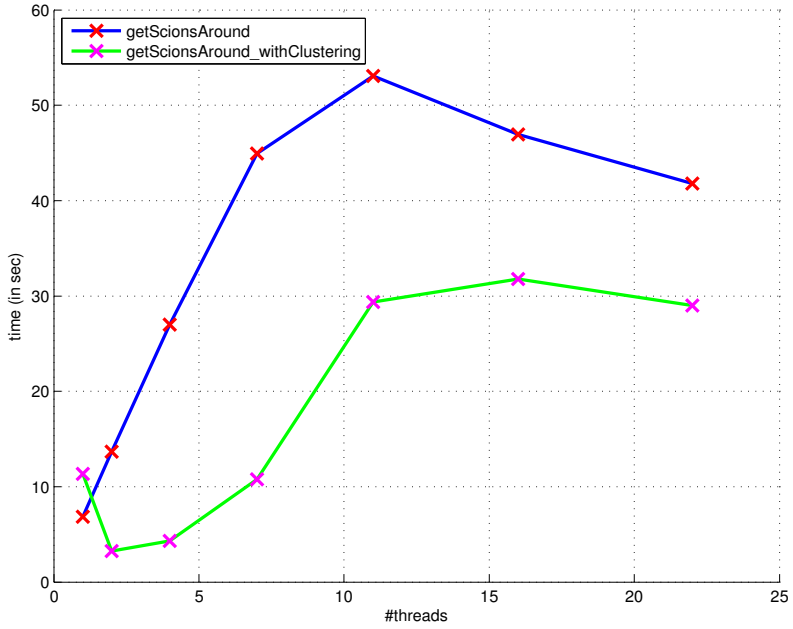

Abbildung 5.15: Performance Test: Sprossen in der Umgebung (mit Cluster Verf ahren)

Die Resultate (Abbildung 5.15) zeigen, dass das Cluster-Verfahren mindestens doppelt so schnell ist, wie das Verfahren ohne Cluster.

5.9 Verbesserung Teil 2

Die Antwortzeit aller Funktionen offenbart jedoch noch ein weiteres Problem. Keine der aktuellen Datenbanken benötigt für eine einstellige Anzahl von parallelen Anfragen eine Antwortzeit im Sekundenbereich. Ich habe diesbezüglich eine Untersuchung durchgeführt. Da schlichtweg alle Funktionen dieses Problem offenbaren habe ich zur Analyse eine spezifische Funktion heraus gesucht, anhand dieser ich das Problem eingrenzen möchte. Die ausgewählte Funktion ist das Hinzufügen einer neuen Sprosse. Dabei lässt sich die Abbarbeitung unterteilen in 3 Phasen. In der ersten Phase kommt es zu einer Initialisierung, wobei geprüft wird, ob alle übergebenen Parameter mit korrekten Werten belegt sind. Des Weiteren wird überprüft, ob der User ausreichend Energie hat, um den Vorgang auszuführen. In der zweiten Phase wird das übertragene Bild mit dem Base64-Verfahren (Kapitel 4.7.8) dekodiert, es wird ein Thumbnail erstellt, und beide Bilder werden auf die Festplatte gespeichert. Zuletzt werden neue Datenobjekte und in die Datenbank persistiert. Die zu erwartende Abarbeitungszeit ist in Abbildung 5.16 und 5.17 dargestellt.

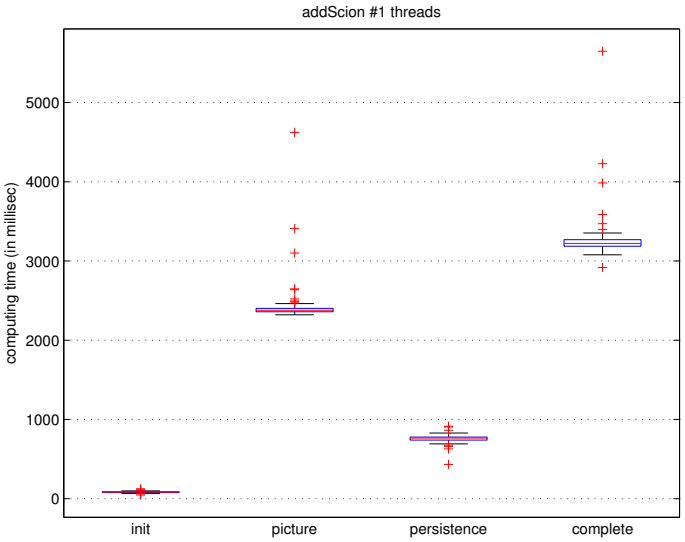

Abbildung 5.16: Sprosse Hinzufügen, Boxplot, 1 Thread

Das Umwandeln eines Bildes und Speichern auf der Festplatte ist eine Design-Entscheidung, die einen großen Teil der benötgten Abarbeitungszeit erklären kann. Doch das größere Problem ist die überaus hohe Zeit, die für das Persistieren (Phase 3) benötigt wird.
Die offensichtlichste mögleiche Problemstelle ist die ausgetauschte Datenbank selbst. Damit Java mit einer mysql Datenbank kommunizieren kann, benötigt es einen entsprechen-

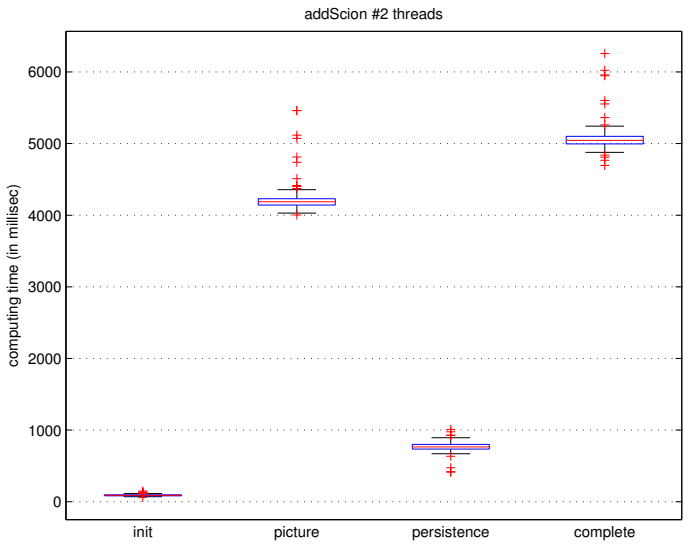

Abbildung 5.17: Sprosse Hinzufügen, Boxplot, 2 Threads

den Connector. Dieser könnte möglicherweise veraltet sein, oder es könnten Versionskonflikte bestehen. Das Rücksetzen auf die im JBoss mitgelieferte Datenbank brachte jedoch keine Verbesserung, und somit scheidet die Annahme aus.
Hintergründige Tätigkeiten, wie die Energieproduktion, und das Erstellen von Snapshots vom Spielsystem fielen auch nicht ins Gewicht.
Die Benutzung einer MySQL Datenbank brachte mit sich, dass für alle nicht-elementaren Java-Datentypen (auch wenn sie das Interface Serializable implementieren) eine BLOB („binary large object“)-Annotation nötig ist. Dabei wird dem OR-Mapper mitgeteilt, dass eine zusätzliche Kodierung und Dekodierung des Wertes der Variable beim Schreiben in, bzw. Lesen aus der Datenbank nötig ist. Die integrierte HSQLB Datenbank benötigt diese Annotation nicht, und man könnte Annehmen, dass die Umwandlung der Werte beträchtlich viel Zeit in Anspruch nimmt. Doch auch ohne diese Annotation in der HSQLB bleibt die Verarbeitungszeit bestehen.
Ein weiterer elementarer Bestandteil in der Abarbeitung jeder Funktion ist das Protokollieren von Vorgängen. Dazu verwende ich stets Message Driven Beans, die sofort Erstellt, aber erst verspätet abgearbeitet werden. Eine Nachricht wird dabei in eine Warteschlange gestellt, und bei Bedarf (zum Beispiel bei wenig Systemlast) entnimmt der Application Server eine Nachricht daraus, und arbeitet sie ab. In meinem Fall wird ein neues Objekt vom Typ Log (siehe Abbildung 4.1) erzeugt und in die Datenbank persistiert. Beim Wiederholen der Tests auf einem anderen Betriebsystem erschienen Fehlermeldungen über die inkorrekte Initialisierung dieser Warteschlange. Der Application Server versuchte dann die Warteschlange ständig erneut zu Initialisieren. Dies führte zu einer endlosen Schleife. Leider wurde diese Nachricht nicht auf dem eigentlichen Zielsystem angezeigt. Das Ent-

fernen der Warteschlange führte schließlich zum Erfolg.
In den Abbildungen 5.18 und 5.19 ist ersichtlich, dass das Persistieren nun im Millisekundenbereich liegt.

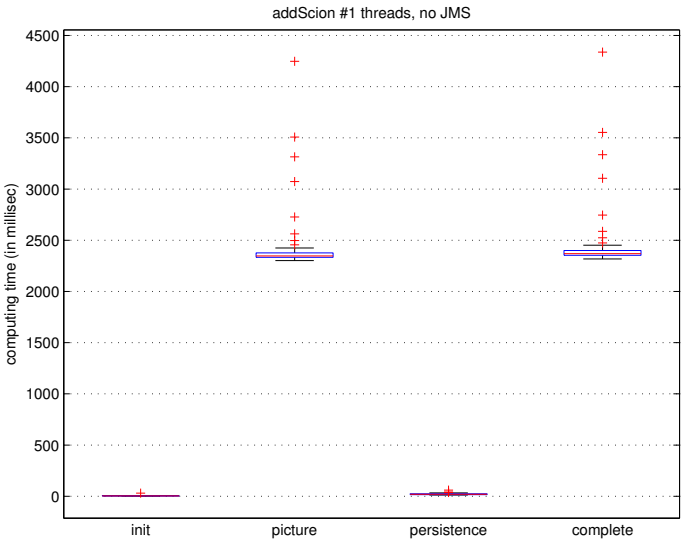

Abbildung 5.18: Sprosse Hinzufügen, Boxplot, 1 Thread, ohne Warteschlange

Die ausgewählte Funktion enthält jedoch ein Hindernis, das bei parallelen Anfragen nun viel eher zu Timeouts führen kann. Wie aus der Grafik ersichtlich ist damit ist das Umwandeln und Abspeichern von Bildern pro Anfragesteller gemeint. Um die Performance des Persistierens ohne Warteschlangen-Nachrichten zu verdeutlichen, zeigen die Abbildungen 5.20 und 5.21 die zu erwartende Antwortzeit beim Einfügen von Sekurit-Linien, abhängig von der Anzahl der Anfragesteller. Deutlich sichtbar ist der Unterschied zur vorherigen Situation. Hat das Persistieren von 7 neuen Sekurit-Linie zuvor im Schnitt 400 Millisekunden benötigt, so persitiert der Application Server nun parallel 500 Sekuritobjekte in selbiger Zeit.

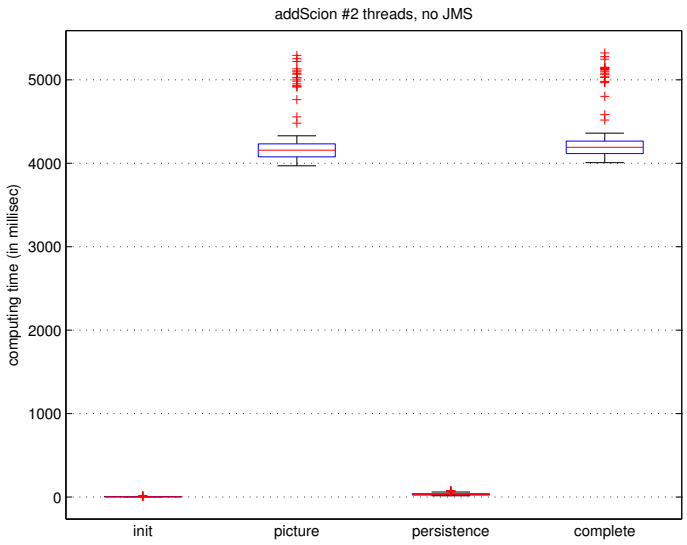

Abbildung 5.19: Sprosse Hinzufügen, Boxplot, 2 Threads, ohne Warteschlange

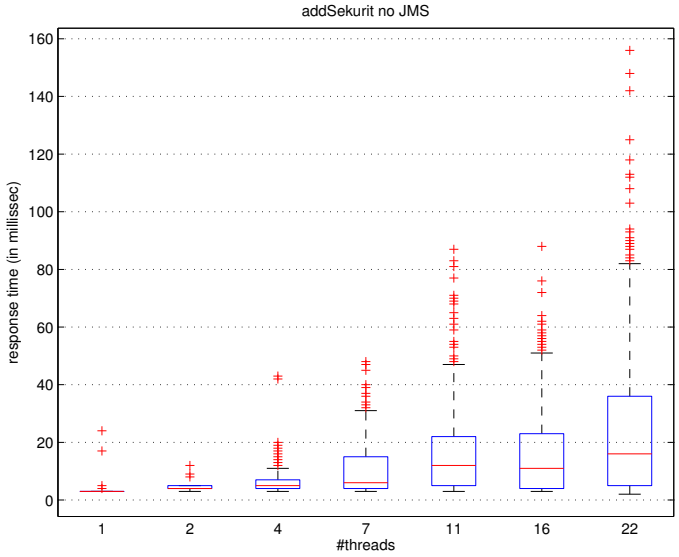

Abbildung 5.20: Sekurit Linie Hinzufügen(1), Boxplot, ohne Warteschlange

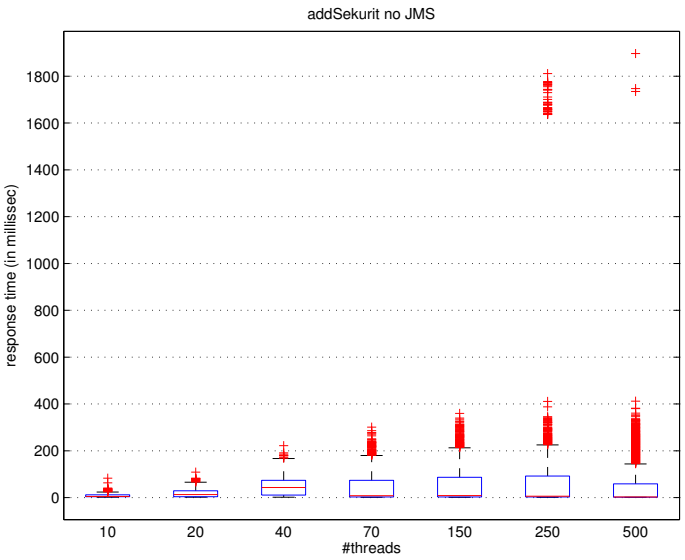

Abbildung 5.21: Sekurit Linie Hinzufügen(2), Boxplot, ohne Warteschlange

Auch Operationen, die lesende Zugriffe an der Datenbank vornehmen verbessern ihre Antwortzeit, wenn sie keine Warteschlangenoperationen ausführen. Abbildung 5.22 zeigt, dass auch bei 500 parallelen Datenbankabfragen die durchschnittliche Antwortzeit unter 500 Millisekunden liegt.

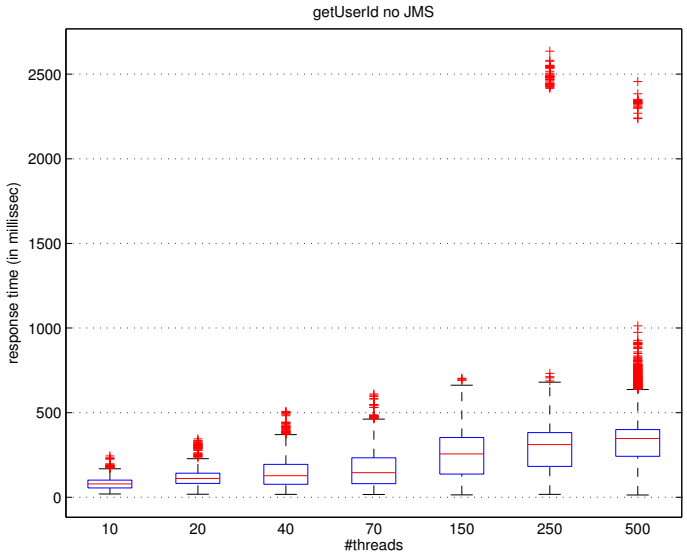

Abbildung 5.22: User ID lesen, Boxplot, ohne Warteschlange

6 Fazit

Das Projekt zeigte, dass Enterprise Java Beans nicht ungeeignet für den Zweck des „mixed reality game“ ist. Die austauschbaren Komponenten sorgen für gute Anpassbarkeit und zahlreiche standardisierte Möglichkeiten in der Präsentationsschicht (RMI-IIOP, WebServices) führen dazu, dass die angebundenen Clients frei wählbar sind.
Smartphones mit dem neuen Android Betriebsystem können Programme starten, die in Java geschrieben wurden. Sie verfügen jedoch bei Weitem nicht über den Funktionsumfang, den man von Java gewohnt ist, da die Java Virtual Machine auf den Android Smartphones nicht vergleichbar mit der Java Virtual Machine von einem Computer ist. Eine Veröffentlichung von freien Bibliotheken, die die Benutzung von WebServices oder RMI-IIOP mit deren kompletten Funktionsumfängen zur Verfügung stellen, wäre äußerst wünschenswert.
Da sich Datenbank-spezifische Anpassungsmöglichkeiten im JBoss Application Server nur schwer realisieren lassen, kann sich eine gut strukturierte Datenschicht äußerst positiv auf die Performance auswirken. Dazu muss, entgegen der EJB Spezifikation, in meinem Fall die Datenschicht auf die Logikschicht abgestimmt sein. Dabei möchte ich keinesfalls Schwachstellen in der EJB Spezifikation andeuten, sondern lediglich zeigen, dass in gewissen Fällen die Antwortzeit der Präsentationsschicht verbessert werden kann, wenn sowohl die Logikschicht geändert, als auch die Datenschicht erweitert wird.
Geschwindikeitsverbesserungen erfordern eine strukturierte Analyse des Sachverhalts. In meinem Projekt habe ich zwei unterschiedliche Schwachstellen offenbart, und daher auch verschiedene Lösungsmöglichkeiten offenbart. Eine Kategorisierung der Aufgabenstellung an das System —und das Untersuchen dieser— ist sehr hilfreich, um Problemstellungen zu erkennen und gezielte Verbesserungsmöglichkeiten zu erkennen.
Um ein markttaugliches Produkt zu entwerfen, müssen allerdings weitere Geschwindigkeits-Verbesserungen eingearbeitet und sicherheitstechnische Anpassungen vorgenommen werden. Dabei muss zumindest das benutzte Protokoll für die WebService Funktionalität durch ein Verschlüsseltes ersetzt werden. Passwörter werden momentan im Klartext übertragen und es ist bisher in keinerlei Hinsicht vorgesehen, dass Aktionen nur aufgerufen werden können, falls ein Spieler durch sein Passwort authentifiziert ist.
Eine Veröffentlichung des Spiels für die Öffentlichkeit wird zum momentanen Zeitpunkt nicht in Betracht gezogen.