Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Oliver Niebsch

Entwicklung und Evaluierung einer Spieleumgebung
als Framework für aktivierendes Lernen

Oliver Niebsch

Entwicklung und Evaluierung einer Spieleumgebung als Framework für aktivierendes Lernen

Abschlussarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Birgit Wendholt
Zweitgutachter : Prof. Dr. Olaf Zukunft

Abgegeben am 06.05.2016

Oliver Niebsch

Thema der Arbeit

Entwicklung und Evaluierung einer Spieleumgebung als Framework für aktivierendes Lernen

Stichworte

Java, Rollenspiel, Framework, Aktivierendes Lernen, Spiel, Motivation

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der Frage, wie man Studierende beim Erlenen von Programmierfähigkeiten unterstützen kann. Um das zu erreichen wird ein Framework für Rollenspiele entwickelt, welches als Grundlage für Übungsaufgaben dient. Durch die Spiele-Thematik soll die Motivation für die Übungen und die Lust auf das Programmieren erhöht werden. Nach der Realisierung des Frameworks erfolgt abschließend noch eine Evaluierung, um zu überprüfen, ob das entwickelte Konzept sein Ziel erfüllt.

Oliver Niebsch

Title of the paper

Development and evaluation of a gaming environment as a framework for accelerated learning

Keywords

Java, role-playing game, framework, accelerated learning, game, motivation

Abstract

This paper deals with the question how to support students learning programming skills. To reach this a framework for role-playing games is developed which provides a basis for exercises. By game topic the motivation for the exercises and the desire to program is due to raise. After the implementation of the framework an evaluation is performed to check whether the developed concept fulfils his aim.

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit beschäftigt sich mit der Frage, wie man Studierende beim Lernen von Programmierkenntnissen und Programmierkonzepten motivieren kann. Ein entwickeltes Spieleframework soll das Interesse wecken und eine Hilfestellung zum selbstständigen lernen bieten. So soll durch Spaß und ein aktivierendes Lernverhalten der Lerneffekt von Aufgaben verstärkt werden, ganz nach Konfuzius:

„Was du mir sagst, das vergesse ich.
Was du mir zeigst, daran erinnere ich mich.
Was du mich tun lässt, das verstehe ich.“

1.1 Motivation

Die Idee zu dieser Arbeit kam durch Beobachtungen des Verfassers von Studierenden in Tutorien. Gerade Programmieranfängern fehlt es am Anfang an Motivation, viel Zeit in das Üben der Programmiersprache und der grundlegenden Konzepte zu investieren. Häufige Gründe dafür sind:

- Die Aufgaben erscheinen als zu einfach, sodass die Studierenden keine Herausforderung darin sehen.
- Die Aufgabenstellungen sind nicht anschaulich oder zu komplex formuliert, da sie auf die Übung eines ganz bestimmten Konzeptes abzielen.
- Der Inhalt der Aufgaben ist nicht interessant für die Studierenden oder auf Dauer zu eintönig.

Um insbesondere dem zweiten und dritten Punkt entgegen zu wirken, wurde für diese Arbeit der Ansatz eines visuell aufbereiteten Spiels gewählt. Untersuchungen von Robin Hilton haben bereits 2006 gezeigt, dass alle Auslöser, die junge

Menschen motivieren, unter anderem in einer Spielumgebung integriert werden können (vgl. Hilton 2006, S.16). Zudem trifft die Thematik eines Computerspiels auch das Interessengebiet vieler junger Leute. Eine Umfrage des Bitkom Research 2015 hat ergeben, dass 81 Prozent der 14 bis 29 Jährigen digitale Spiele spielen (Bitkom 2015). So kann durch die Spieleumgebung nicht nur die Motivation erhöht werden, sondern es wird etwas programmiert mit dem die meisten Studierenden vertraut sind. Das ermöglicht es spielespezifische Ausdrücke ohne großartige Erklärungen zu verwenden, da diese vielen Studierenden bereits bekannt sind. Mit Hilfe solcher Ausdrücke können dann Aufgabenstellungen einfach und anschaulich gestaltet werden.

1.2 Zielsetzung

Experimente mit sogenannten „Serious Games“ haben bereits gezeigt, dass durch einen spielerischen Ansatz sowohl die Grundlagen des Programmierens als auch Verständnis für Programmierkonzepte mit Spaß erlernt werden können (vgl. Mitamura 2012 S.1817 Conclusion). Aufbauend auf diesen Erkenntnissen ist es das Ziel dieser Arbeit ein Java-Spieleframework zu entwickelt, welches eine grundlegende Spieleumgebung beinhaltet und Schnittstellen anbietet, um das Spiel individuell anzupassen und erweitern zu können. Durch die Arbeit mit diesem Framework sollen Informatik-Studierende mit Vorkenntnissen in der Programmierung ihre Fähigkeiten verbessern und festigen können. Für Lehrende soll das Spiel ein Werkzeug sein, um Programmierkonzepte anschaulich zu vermitteln und üben zu lassen.

Neben der Möglichkeit Übungsaufgaben rund um die Spieleumgebung zu entwickeln und den Studierenden zur Verfügung zu stellen, wird ein großer Wert darauf gelegt, dass die Studierenden mit ihrem Wissensstand auch die Funktionsweise der grundlegenden Spieleumgebung verstehen und nachvollziehen können. Zur Erreichung dieses Ziels wird auf die Komplexität der verwendeten Modelle geachtet. Deshalb handelt es sich bei dem Spiel um ein Singleplayer-Spiel in dem sich der Spieler in einer modular aufgebauten Welt bewegt.

Zudem wird auf eine integrierte Entwicklungsumgebung verzichtet, da die Studierenden meist durch Vorlesungen und Praktika bereits mit einer konkreten Entwicklungsumgebung vertraut sind und so das Framework einfach in ihren Studienalltag einbinden können. Diese Entscheidungen sollen den Studierenden

eine möglichst autonome Arbeit mit dem Framework ermöglichen. So wird durch selbständige Arbeit ein aktivierendes Lernverhalten gefördert.

1.3 Aufbau der Arbeit

Kapitel 2 beschäftigt sich mit der Idee und der Besonderheit dieser Arbeit. Hierzu werden vergleichbare Arbeiten und deren Umfang beleuchtet. Zusätzlich wirft das Kapitel einen Blick auf den Inhalt von Rollenspielen und nimmt anschließend eine Abgrenzung dieser Arbeit zu den vorgestellten Ansätzen vor.

In Kapitel 3 werden zunächst die Anforderungen an das Framework dieser Arbeit beschrieben. Anschließend wird in Kapitel 4 die Umsetzung dieser Anforderungen erläutert. Dafür werden der Aufbau des Frameworks beschrieben, grundsätzliche Entwurfsentscheidungen erklärt und die einzelnen Komponenten vorgestellt.

Um zu testen, ob mit Hilfe des entwickelten Frameworks ein aktivierendes Lernen unterstützt werden kann, beschäftigt sich Kapitel 5 mit der Evaluation einer Übungsaufgabe und einer dazugehörigen Umfrage an Studierenden.

Zum Schluss fasst Kapitel 6 die wesentlichen Ergebnisse dieser Arbeit zusammen und gibt ein Ausblick auf mögliche Verbesserungen und Erweiterung des entwickelten Konzeptes.

2 Einordnung der Arbeit

In diesem Kapitel wird der Hintergrund der Idee dieser Arbeit beleuchtet. Das Kapitel 2.1 beschreibt in fünf Unterkapiteln Arbeiten mit vergleichbaren Ansätzen und zeigt deren Funktionsumfang auf. Kapitel 2.2 wirft einen Blick auf klassische Rollenspiele und beschreibt deren Gemeinsamkeiten. In Kapitel 2.3 erfolgt aufbauend auf den vorhergehenden Kapiteln eine Abgrenzung des Konzepts dieser Arbeit.

2.1 Vergleichbare Arbeiten

Die Idee, das Lernen von Programmieren mit Hilfe visueller Darstellung oder spielerischen Komponenten zu erleichtern, ist nicht neu. Es finden sich zahlreiche Ansätze und Projekte, um für Programmieranfänger verschiedener Altersklassen einen Einstieg in die Welt des Programmierens zu schaffen. Die nachfolgenden Kapitel geben anhand von Beispielen eine Übersicht über häufig genutzte Ansätze.

2.1.1 Logo

Umfang
Die Programmiersprache Logo ist einer der ersten Ansätze, Programmieren für Anfänger, insbesondere für Kinder leicht erlernbar zu machen. Hauptbestandteil von Logo ist die sogenannte *Turtle*, die man mit Hilfe von Befehlen über den Bildschirm bewegen kann. Die Turtle wird dabei durch eine Grafik dargestellt und kann bei ihren Bewegungen über den Bildschirm Linien malen. Durch unterschiedliche Bewegungen entstehen so komplexe geometrische Figuren oder Muster.

Mittlerweile gibt es viele unterschiedliche Varianten von Logo und dafür passende Entwicklungsumgebungen. Je nach Variante kann der Umfang der Programmier-sprache unterschiedlich ausfallen. Gemeinsam ist bei allen ein einfacher Befehlssatz, um die Turtle zu bewegen, sowie eine Zählschleife und die Möglichkeit Code in Methoden zusammen zu fassen. Da es sich bei Logo um eine Script-Sprache

handelt, sorgt die überschaubare Anzahl an Befehlen für einen einfachen und schnellen Start. Zudem gibt es Bücher, Tutorials und Anleitungen, wie man mit Logo in die Programmierung einsteigen kann.

Zielgruppe
Ziel von Logo ist es nicht alltägliche oder mathematische Probleme durch Programmierung zu lösen, sondern erste Erfahrungen im Umgang mit dem Computer zu machen und algorithmisches Denken zu fördern. Der didaktische Ansatz dies durch das Zeichnen von Bildern zu erreichen, richtet sich sehr stark an Kinder ohne Programmiererfahrung. Sie können mit Hilfe der Turtle ihrer Kreativität freien Lauf lassen und trainieren dadurch das logische Denken und Grundelemente der Programmierung, wie das Prinzip *Teile und Herrsche*.

Erweiterbarkeit
Durch das Erstellen eigener Methoden lassen sich die Fähigkeiten der Turtle beliebig erweitern. Zudem kann man diese Methoden speichern und zu einem späteren Zeitpunkt laden, sodass z.B. ein erweiterter Befehlssatz den Lernenden zur Verfügung gestellt werden kann. Einige Eigenschaften, wie beispielsweise die Farben mit der die Turtle zeichnen kann, lassen sich jedoch in den meisten Logo-Varianten nicht verändern.

2.1.2 Roberta

Umfang
Roberta ist ein vom Frauenhofer IAIS entwickeltes Konzept, um junge Menschen für Technik und Naturwissenschaft zu begeistern. Mit Hilfe von LEGO Mindstorm Robotern werden Jungen und vor allem Mädchen in die Welt der Technik und des Programmierens eingeführt (vgl. Roberta Home). Die Arbeit mit Robotern soll die Motivation erhöhen und durch die selbstständige Konstruktion die Kreativität fördern. Zudem bietet es einen einfachen Einstieg, da keine Kenntnisse der Programmierung erforderlich sind, um einen LEGO Roboter zusammen zu bauen.

Innerhalb von Roberta finden sich viele didaktisch aufbereitete Kurse für verschiedene Altersgruppen. Diese Kurse werden von speziell geschulten und zertifizierten Personen, den Roberta-Teachern, gehalten. Neben dem Zugriff auf Lernmaterialien haben die Teacher auch außerhalb der Schulungen die Möglichkeit sich untereinander europaweit auszutauschen. So soll sich Roberta stetig weiter entwickeln (vgl. Roberta Konzept).

Für die Programmierung können verschiedene Programmiersprachen genutzt werden. Unter anderem kommt auch eine visuelle Programmiersprache zum Einsatz. Diese ermöglicht es per Drag-and-Drop von Anweisungs- und Steuerungsblöcken das Verhalten des Roboters zu programmieren. So können spielerisch grundlegende Programmierkonzepte vermittelt werden. Mit Hilfe von höheren Programmiersprachen, wie z.B. Java, lassen sich jedoch auch weiterführende Konzepte vermitteln, sofern der Lehrende über die nötigen Grundkenntnisse der Programmiersprache verfügt oder sich diese aneignet.

Zielgruppe
Roberta ist vor allem als Programmiereinstieg für Kinder und Jugendliche gedacht. Zum einen spricht das Experimentieren mit LEGO Robotern eher jüngere Leute an, zum anderen sind die Schulungen für Lehrer an Grund-, Real- und Gymnasialschulen ausgelegt. Außerdem richtet sich das einfache Programmieren mit Blöcken in der visuellen Programmiersprache in erster Linie an Programmieranfänger und Kinder, da man keinerlei Vorkenntnisse oder Wissen über Besonderheiten der Sprache, wie z.B. die Syntax braucht.

Erweiterbarkeit
Entsprechend dem Konzept von LEGO, lassen sich auch die Roboter fast beliebig erweitern. Was die Programmierung angeht, stehen etliche Sensoren zur Verfügung, die der Benutzer verwenden kann. Der Lehrende kann sich Parcours oder Aufgaben ausdenken, welche die Lernenden mit Hilfe dieser Sensoren automatisiert von ihren Roboter lösen lassen sollen. Das Roberta-Konzept umfasst jedoch bereits sehr viele Programmierkonzepte, sodass diese Möglichkeit der Erweiterung eher zum Erstellen von zusätzlichen Übungen interessant ist.

2.1.3 Java-Hamster-Modell

Umfang
Das Java-Hamster-Modell ist ein von Dr.-Ing. Dietrich Boles entwickeltes „didaktisches Modell, mit dessen Hilfe [man] Grundkonzepte der Programmierung auf spielerische Art und Weise erlernen [kann]“ (Java Hamster). Durch selbst geschriebenen Code kann man einen Hamster durch selbst erstellte Labyrinthe bewegen.

Abb. 2.1 zeigt die beiden Komponenten des Java-Hamster-Programms. Auf der linken Seite sieht man den einfachen Quelltext-Editor in dem der Inhalt eines vorgegebenen Projekte-Ordners aufgelistet wird und Quelltext-Dateien bearbeiten kann. Bei den Quelltext-Dateien handelt es sich nicht um spezielle .ham-Dateien. Neben den Elementen der Programmiersprache Java hat man in .ham-Dateien noch die Möglichkeit über spezielle Methoden die Funktionalitäten des Hamsters zu steuern. Diese Methoden lassen sich nutzen, ohne im Quelltext das Hamster-Objekt zu kennen oder erzeugt zu haben. Die .ham-Dateien werden vom Programm vor der Ausführung in kompilierbare .java-Dateien umgewandelt.

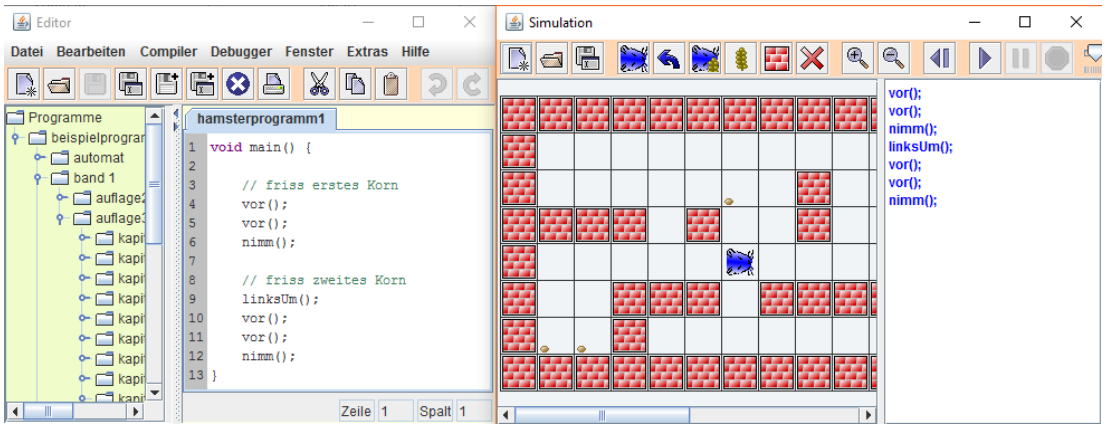

Abb. 2.1 Das Java-Hamster-Programm

Die rechte Seite der Abb. 2.1 zeigt das Simulations-Fenster des Java-Hamster-Programms. In diesem Fenster lassen sich die Labyrinthe für den Hamster gestalten. Hierfür stehen dem Benutzer drei Elemente zur Verfügung: Mauern, Körner und der Hamster. Im Simulations-Fenster wird auch der geschriebene Code simuliert. Dabei werden die Aktionen des Hamsters im Labyrinth dadurch ersichtlich, dass Körner auf den Feldern verschwinden, da sie vom Hamster aufgenommen wurden, oder sich das Hamster-Icon im Labyrinth bewegt. Alle ausgeführten Methoden des Hamsters werden im rechten Bereich des Fensters während der Ausführung des Codes aufgelistet.

Für das Java-Hamster-Modell sind insgesamt drei Bücher erschienen (Stand März 2016) in denen am Beispiel des Hamsters verschiedene Konzepte der Programmierung vermittelt werden (Boles 2013; Boles 2010; Boles 2009). Zudem findet man im Internet Tutorials und Aufgaben, die mit dem Hamster arbeiten und teilweise auch fortgeschrittene Programmiertechniken behandeln.

Zielgruppe
Zur Zielgruppe des Java-Hamster-Modells gehören alle Programmieranfänger, die Java lernen wollen. Die einfach gehaltene grafische Oberfläche und die Vorstellung man lasse einen Hamster durch ein Labyrinth laufen, richten sich jedoch in erster Linie an junge Leute. Die Bücher hingegen sind sehr neutral gehalten und nicht explizit auf Kinder oder Jugendliche zugeschnitten. Sie sollen die interessierte Öffentlichkeit ansprechen.

Erweiterbarkeit
Beim Java-Hamster-Modell ist man an zwei Einschränkungen gebunden: Die eingeschränkte Gestaltungsmöglichkeit der Labyrinthe und die begrenzten Fähigkeiten des Hamsters. Erweiterungen des Modells, wie z.B. Heuballen als bewegliche Wände, sind nicht vorgesehen und können deshalb nicht hinzugefügt werden. Die Fähigkeiten des Hamsters lassen sich durch Erstellen einer eigenen Hamster-Klasse erweitern, jedoch muss man dann immer auf diese Klasse zurückgreifen und verliert die Vorteile der .ham-Dateien.

2.1.4 CodeCombat

Umfang
CodeCombat (CodeCombat) ist ein Browser-Onlinespiel in dem man mit Hilfe von Code einen Helden durch vorgegebene Level steuert. Die anfangs sehr einfach gehaltenen Levels werden nach und nach immer schwerer und zeigen einem Stück für Stück neue Programmierkonzepte mit denen sich die Aufgaben lösen lassen. Durch den erfolgreichen Abschluss der Levels gewinnt man außerdem neue Gegenstände, die bei dem Helden neue Funktionen freischalten, wie z.B. ein Schwert zum Angreifen von Gegnern.

Abb. 2.2 Lösen einer Aufgabe auf der CodeCombat-Websiete

Abb. 2.2 zeigt den Aufbau der CodeCombat-Webseite während der Lösung der Aufgaben eines Levels. Ähnlich wie beim Hamster-Modell (Kapitel 2.1.3) gibt es auch hier die Zweiteilung von Code-Editor (rechts) und der Darstellung der Welt (links). Als Programmiersprachen stehen fünf Script-Sprachen zur Auswahl mit denen sich die Aufgaben lösen lassen (Stand März 2016).

Neben dem Online-Modus in dem man eigenständig anhand der Levels seine Programmierfähigkeiten erweitern kann, gibt es auch die Möglichkeit für Lehrende so genannte Klassenräume einzurichten und Kurse zu kaufen. Die Kurse behandeln jeweils bestimmte Programmierkonzepte und durch den Klassenraum können die Lehrenden die Fortschritte der Lernenden im Auge behalten. Die Webseite bietet zudem Hilfestellungen für Lehrende, die kaum Programmiererfahrungen haben.

Zielgruppe
CodeCombat richtet sich hauptsächlich an Schülerinnen und Schüler. Da es im kostenlosen Online-Modus nicht möglich ist, Level zu überspringen, eignet sich dieser in erste Linie für komplette Anfänger. Das bezahlte Angebot ist für Lehrende an Schulen ausgelegt und soll eine Möglichkeit bieten, Programmieren mit in die schulische Lehre einzubinden.

Erweiterbarkeit
Durch die Open-Source-Lizenz von CodeCombat und die Offenlegung des Quellcodes (https://github.com/codecombat/codecombat), lässt sich das Spiel beliebig erweitern, sofern man die nötigen Programmierkenntnisse besitzt. Zudem gibt es eine Community mit ehrenamtlichen Helfern und Programmierern, die verschiedene Bereiche von CodeCombat weiter entwickeln. Schließt man sich dieser Community an, kann man von den anderen Programmierern Tipps und Hilfestellungen erhalten. Unter anderem gibt es die Möglichkeit eigene Level zu designen oder vorhandene Level zu verändern. Hierfür steht ein kostenloses Online-Tool zur Verfügung, welches sich jedoch nach eigenen Angaben noch in einem Alpha-Stadium befindet.

2.1.5 Greenfoot

Umfang

Bei Greenfoot handelt es sich um eine Java-Entwicklungsumgebung mit deren Hilfe sich objektorientiertes Programmieren lehren lässt. Es bietet die Möglichkeit für die Programmierung interaktive Welten zu gestalten, in denen sich verschiedene Elemente befinden. Durch die visuelle Darstellung der Welt und der Elemente soll das Verständnis und der Umgang mit Objekten in der Programmierung erleichtert werden. Zudem lassen sich die Welten inklusive der enthaltenen Objekte animieren, um so Spiele oder Simulationen zu bauen. Abb. 2.3 zeigt links die visuelle Darstellung der Welt und rechts den integrierten Quelltext-Editor.

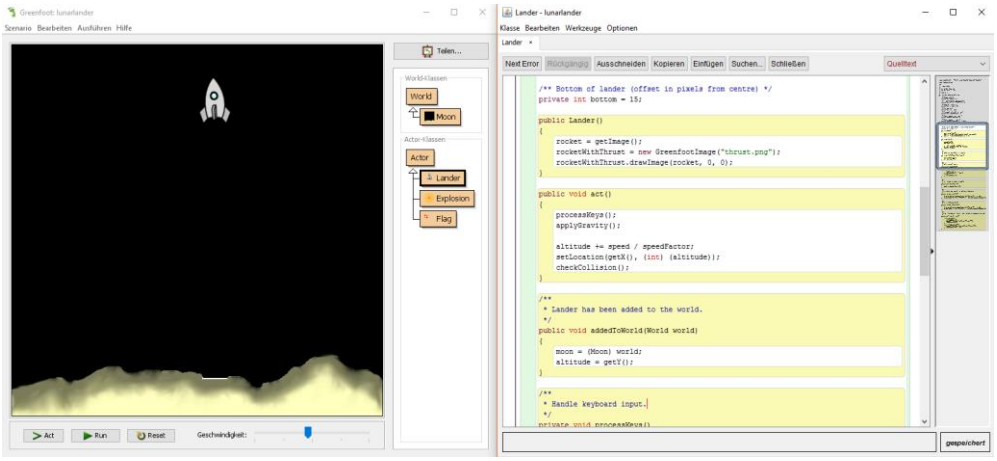

Abb. 2.3 Beispielanwendung in Greenfoot

Die Vorlage für den Quelltext-Editor bietet BlueJ. BlueJ ist nach eigenen Angaben ebenfalls eine Entwicklungsumgebung mit der sich Java-Programme schnell und einfach entwickeln lassen (vgl. BlueJ). Im Gegensatz zu Greenfoot gibt es bei BlueJ jedoch keine vorgefertigten Funktionen, um eine interaktive Welt zu gestalten. Der Editor ist einfach gehalten und bietet dennoch die nötigen Funktionen zum Bearbeiten von Text und Kompilieren von Quellcode, sowie ein übersichtliches Syntax-Highlighting.

Auf der Webseite von Greenfoot können sich Nutzer in einem Forum austauschen und in der Galerie eigene Projekte veröffentlichen oder über andere Projekte diskutieren (vgl. Henriksen 2010 S.3). Neben der Möglichkeit Greenfoot als Hilfsmittel in der Lehre einzusetzen, bieten das Forum und die Galerie eine Hilfestellung für Benutzer, die sich mit Greenfoot Java im Selbststudium aneignen

wollen. Neben dem Lernen von Programmierkonzepten fokussiert Greenfoot die objektorientierten Konzepte von Java und grundlegende Überlegungen zur Modellierung. Bevor man sein Programmierwissen ausprobieren kann, muss man sich zuerst ein Projekt suchen oder sich selbst Gedanken über die Modellierung der Welt und der Elemente machen. Wenn die Welt und die darin befindlichen Elemente erstellt sind, kann man das Verhalten der Elemente programmieren.

Zielgruppe
Greenfoot versucht mit seinem sehr offenen Konzept eine breite Masse an Programmierern und Programmieranfängern verschiedener Altersklassen anzusprechen. Junge Leute können sich spielerisch ihre eigene Welt gestalten und dabei erlerntes Programmierwissen umsetzen. Lehrende können sich mit Hilfe von Greenfoot eine Welt erstellen, um Programmierkonzepte anschaulich zu vermitteln. Erfahrenere Programmierer können das Framework verwenden, um Spiele oder Simulationen zu erstellen. Einzig der Editor ist für ehrfahrende Programmierer nicht so gut geeignet, da er nicht so viele hilfreiche Zusatzfunktionen umfasst, wie man es aus komplexere Editoren, z.B. Eclipse oder NetBeans, gewohnt ist.

Erweiterbarkeit
Da es sich bei Greenfoot um ein Framework handelt, ist Erweiterbarkeit eine Eigenschaft, die sehr stark ausgeprägt ist. Bei der Entwicklung von eigenen Welten ist man keinerlei Einschränkungen unterlegen und auch eigene Klassen, die nicht direkt zur Welt gehören, werden in der GUI mit aufgeführt. Zudem beinhaltet Greenfoot Tools, um das Repertoire an vorgegeben Grafiken und Sounds zu erweitern. Darüber können einfach selbstständig Sounds aufgenommen oder eigene Grafiken für die Gestaltung der Welt hinzuzufügt werden.

2.2 Computer Rollenspiele

Wirft man einen Blick auf Computer Rollenspiele, so entdeckt man beim den grundlegenden Eigenschaften der Spiele sehr viele Gemeinsamkeiten. Basierend auf der Ausarbeitung von S. Lindenberg (vgl. Lindenberg 2009, 2.1.4.2. und 2.1.5.2) werden im Folgenden einige dieser Eigenschaften näher beleuchtet.

Jeder Spieler besitzt eine Figur als Repräsentant im Spiel. Diese Figur lässt sich zu Beginn gestalten und im Laufe des Spiels mit Gegenständen erweitern. Bei diesen Gegenständen handelt es sich beispielsweise um Rüstungen, Waffen und magische Ringe. Der Spieler kann seine Figur durch eine vorgegebene Welt bewegen und

diese erkunden. Hierbei trifft er auf unterschiedliche Landschaften wie Höhlen, Dörfer oder Wälder. Neben der eigenen Figur existieren weitere Wesen oder Objekte in der Welt des Spiels mit denen der Spieler interagieren kann. Feindlich gesinnte Wesen kann man bekämpfen und mit Freundlichen beispielsweise reden.

Hauptziel vieler Spiele ist es die eigene Figur zu trainieren und bestimmte Gegner auszuschalten. Auf dem Weg zum großen Ziel trifft der Spieler immer wieder auf kleinere Handlungsstränge, wodurch sich weitere Nebenziele ergeben. In Online-Rollenspielen wird häufig auf ein großes Hauptziel verzichtet und verstärkt mehr Wert auf eine Vielzahl von kleineren Zielen oder Herausforderungen gelegt.

2.3 Zusammenfassung und Abgrenzung

Die Programmiersprache Logo ist recht simpler Ansatz, das Lernen von Programmierkonzepten einfacher zu gestalten. Durch bestimmte Befehle in einer Scriptsprache können auf einer Zeichenfläche geometrische Figuren und Muster gezeichnet werden. Durch die Einfachheit dieses Konzeptes bietet es sich sehr gut an, um erste Erfahrung mit Programmiersprachen zu machen und logisches Denken zu üben.

Das Roberta-Projekt vereint die Vielseitigkeit der Konstruktion von Robotern mit dem Erlenen von Programmierkonzepten, um Aufgaben automatisiert lösen zu lassen. Das Experimentieren mit LEGO-Robotern soll besonders Kinder ansprechen. Ebenfalls eher für Kinder geeignet ist das Java-Hamster-Modell. Dabei wird der Hamster mit Hilfe von Programmcode durch erstellte Labyrinthe gesteuert. Bei beiden Ansätzen lässt sich die Herausforderung im Rahmen der Fähigkeiten des Roboters bzw. Hamsters mit verschiedenen Elementen gestalten.

CodeCombat beinhaltet bereits Eigenschaften eines Rollenspiels. Anhand von vorgegebenen Levels werden nach und nach Programmierkonzepte gelernt und geübt, um mit seiner Figur die immer schwerer werdenden Aufgaben zu meistern. Hierbei ist die Erweiterung des Spielprinzips oder das Erstellen neuer Levelinhalte nur mit einem gewissen Grad an Programmierkenntnissen möglich.

Das Konzept von Greenfoot kommt der Idee dieser Arbeit am nächsten. Es ist ein Framework mit dem sich Welten zum Lehren und Lernen der Programmiersprache Java erstellen lassen. Die Tools für Grafiken und Sound, sowie der integrierte

Quelltext-Editor sorgen dafür, dass man direkt mit dem Programmieren beginnen kann.

Im Gegensatz zu Greenfoot ist das in dieser Arbeit entwickelte Framework auf Rollenspiele spezialisiert und bietet weniger, jedoch spezialisierte Schnittstellen an. Ähnlich wie bei den Ansätzen von Roberta und dem Java-Hamster-Modell beinhaltet das Framework eine lauffähige Spieleanwendung, in welcher der spätere Programmierer Elemente mit Quellcode beeinflussen kann. Dieses Spiel beinhaltet bereits die vorgestellten Eigenschaften eines Rollenspiels. Um Programmierkonzepte zu üben, müssen die Benutzer jedoch keinen Parcours oder, wie bei CodeCombat, Aufgaben meistern. Im Vordergrund steht die Weiterentwicklung und Gestaltung der Spieleanwendung durch die Erstellung neuer Landschaften oder dem Hinzufügen von Elementen.

Die Arbeit mit und an einem Rollenspiel fördert das explorative und spielerische Lernen und soll so die Motivation zum Programmieren steigern. Um das Framework einfach ins Studium einbinden zu können, wird auf einen integrierten Quelltexteditor verzichtet. Der Quellcode des Frameworks ist frei zugänglich und kann so leicht in bereits vorhandene und genutzte Entwicklungsumgebungen eingebunden werden.

Die Anbietung weniger, jedoch flexibler Schnittstellen sorgt für eine große Gestaltungs- und Erweiterungsmöglichkeit. Haben die Studierenden die Mechanik zur Implementierung der Schnittstellen einmal verstanden, können sie unterschiedliche Ideen im Spiel umsetzen und so verschiedene Programmierkonzepte trainieren. Die Vielzahl der möglichen Erweiterungen sorgt dabei dafür, dass das Programmieren nicht schnell eintönig wird.

3 Anforderungsanalyse

Dieses Kapitel beschäftigt sich mit den Anforderungen an das Framework. Zuerst beschreibt Kapitel 3.1 die Zielgruppe, die durch das Framework erreicht werden soll. Dann erklärt Kapitel 3.2 die wichtigsten Eigenschaften von Frameworks und Kapitel 3.3 fasst noch einmal die Eigenschaften von Lehrsprachen zusammen. Kapitel 3.4 zeigt anschließend auf, welche funktionalen und nichtfunktionalen Anforderungen sich durch die Kombination von Rollenspiel, Framework und Lehrsprache für diese Arbeit ergeben.

3.1 Zielgruppe

Die Zielgruppe besteht vorrangig aus jungen Studierenden mit Grundkenntnissen in der Programmierung. Durch die Arbeit mit dem Framework sollen sie spielerisch ihre Programmierkenntnisse festigen und erweitern. Eine Möglichkeit ist, dass die Studierenden vorgegebenen Aufgaben mit dem Framework lösen. Diese Aufgaben könnten beispielsweise im Rahmen eines Praktikums oder eines Tutoriums gestellt werden. Andererseits können die Studierenden auch ohne konkrete Aufgaben das Spiel weiterentwickeln. Die Elemente des Frameworks dienen als Grundlage, sodass nach kurzer Zeit Erfolge in Form eines funktionierenden Spiels zu sehen sind.

3.2 Eigenschaften eines Frameworks

Bei dem Design und der Entwicklung eines Frameworks gibt es einige grundlegende Dinge zu beachten. In erster Linie stellt „*Einfachheit* […] eine unverzichtbare Eigenschaft eines jeden Frameworks“ (Cwalina 2007) dar. Diese Eigenschaft zu erreichen ist nicht leicht, insbesondere wenn ein Framework weiter wächst und so neue Funktionalitäten hinzukommen. Zudem soll ein Framework für möglichst viele ähnliche Probleme eingesetzt werden. Die konkreten funktionalen Anforderungen des einzelnen Problems können dabei sehr verschieden sein. Deshalb bildet die *Erweiterbarkeit* eine weitere wichtige Eigenschaft jedes Frameworks. Die Entwickler des Frameworks sollten sich bei jeder Funktion die Frage stellen, wie groß der Grad

der Abstraktion der Schnittstellen sein muss, um alle nötigen Sonderfälle abdecken zu können.

Zur Unterscheidung der verschiedenen Aspekte eines Frameworks werden bei Cunningham und Tadepalli die Begriffe *frozen spots* und *hot spots* eingeführt. Frozen spots sind die festen Bestandteile des Frameworks, die in jeder entstehenden Applikation wiederverwendet werden. Hot spots sind die variablen Bestandteile des Frameworks, die für jede konkrete Anwendung spezialisiert werden (vgl. Cunningham 2006). Die frozen spots beinhalten das Grundwissen über den Problembereich, wohingegen die hot spots als Schnittstellen realisiert sind. Eine klare *Definition der hot spots* sollte bei einem Framework gegeben sein.

Um das Framework *benutzerfreundlich* zu gestalten, sollten die Schnittstellen möglichst k*onsistent* sein. Dies bedeutet, dass wenig unterschiedliche Mechanismen zur Verwendung der Schnittstellen verwendet werden dürfen. So kann sich der Benutzer des Frameworks auf eine Arbeitsweise einstellen, um verschiedene Aspekte der konkreten Anwendung zu realisieren. Zudem ist eine gewisse *Fehlertoleranz* der frozen spots wichtig. Diese sollten fehlerfrei funktionieren, unabhängig ob die hot spots korrekt bzw. sinnvoll umgesetzt sind oder nicht.

3.3 Lehrsprachen

Viele Lehrsprachen versuchen durch visuelle Repräsentation des Codes und meist spielerische Elemente das Programmieren insbesondere für Anfänger interessant zu gestalten. Häufig wird der Funktionsumfang der Programmiersprachen auch beschränkt oder erweitert. Die Beschränkungen sollen dafür sorgen, dass man sich am Anfang nicht in der Komplexität der Programmiersprache verirrt. Erweiterungen sind meistens spezifisch auf die jeweilige visuelle Umgebung angepasst oder vereinfachen die Syntax, sodass die ersten Aufgaben mit wenigen Codezeilen lösbar sind. Zusätzlich zu der Vereinfachung der Programmiersprachen sind auch die GUIs einfach und übersichtlich aufgebaut, um die Einarbeitungszeit so kurz wie möglich zu halten.

3.4 Anforderungen

Aufbauend auf den Erkenntnissen der Kapitel 2.2, 3.2 und 3.3 ergeben sich die folgenden funktionalen und nicht funktionalen (Kapitel 3.4.3) Anforderungen. Die

funktionalen Anforderungen sind unterteilt, je nachdem ob sie die frozen spots (Kapitel 3.4.1) oder die hot spots (Kapitel 3.4.2) betreffen.
Framework steht dabei für das in dieser Arbeit entwickelte Framework. *Spieler* bezeichnet einen Studierenden, der mit dem Framework entwickelt und das entwickelte Spiel spielt. Mit *Charakter* ist die Figur eines Spielers im Spiel gemeint.

3.4.1 Funktionale Anforderungen an frozen spots

Lauffähige Spieleanwendung
Das Framework stellt eine lauffähige Spieleanwendung zur Verfügung. Die Anwendung lässt sich fehlerfrei starten und beenden. Zudem sind die im Folgenden beschriebenen Aktionen im Spiel möglich.

Charaktergestaltung
Beim Starten der Anwendung kann sich der Spieler einen Charakter gestalten. Die Eigenschaften, wie z.B. Geschlecht, Hautfarbe oder Frisur des Charakters, kann der Spieler aus einer Menge von Vorgaben auswählen.

Bewegung und Animation des Charakters
Der Charakter kann vom Spieler gesteuert werden und sich so durch die Welt bewegen. Die Bewegungen des Charakters sind animiert.

Speicherung des Spiels
Beim Beenden der Anwendung wird der Zustand des Spiels automatisch gespeichert, sodass der Spieler das Spiel zu einem späteren Zeitpunkt an der gleichen Stelle wieder aufnehmen kann.

3.4.2 Funktionale Anforderungen an hot spots

Eigenschaften zum Charakter hinzufügen
Der Spieler kann über Schnittstellen dem Charakter Eigenschaften hinzufügen. Diese Eigenschaften werden automatisch zusammen mit dem Charakter gespeichert und sind für alle Elemente des Spiels abrufbar.

Gestalten der Welt
Der Spieler kann die Welt des Spiels, also den Hintergrund auf dem sich der Charakter bewegt, gestalten. Dies umfasst die Änderung und Erweiterung einer eventuell bereits existierenden Welt, sowie die Erstellung neuer Welten. Die

Grafiken zur Darstellung der Welt können dem Spiel hinzugefügt werden. Der Aufbau der Welt ist in Konfigurationsdateien zu beschreiben.

Interaktive Objekte hinzufügen
Der Spieler kann dem Spiel beliebige Objekte hinzufügen. Diese Objekte werden neben dem Charakter in der Welt dargestellt und können ebenfalls animiert sein. Der Charakter kann mit den Objekten interagieren. Die Form der Interaktion ist soweit flexibel, dass beispielsweise die in Kapitel 2.2 genannten Interaktionsformen Reden, Handeln und Kämpfen über eine Schnittstelle realisierbar sind. Die Positionierung der Objekte erfolgt über Konfigurationsdateien.

3.4.3 Nichtfunktionale Anforderungen

Erweiterbarkeit der Spieleanwendung
Neue oder geänderte Konfigurationsdateien werden beim Starten der Anwendung automatisch mit berücksichtigt und die Welt entsprechend dargestellt.

Einfache Benutzbarkeit
Die Steuerung des Spiels ist einfach gehalten. Der Charakter wird mit wenigen Tasten gesteuert, andere Elemente des Spiels sind durch Klicken mit der Maus bedienbar. Die Eingabe von Texten erfolgt über die Tastatur.

Fehlertoleranz gegenüber Eingaben
Bei fehlerhaften oder unsinnigen Eingaben reagiert das Spiel der Situation angemessen. Durch falsche Eingaben darf es nicht zu einem Absturz des Spiels kommen.

Fehlertoleranz gegenüber fehlerhafter Erweiterungen
Fehlerhafter oder fehlender Quellcode, der eine Erweiterung der Spieleanwendung darstellt, darf nicht zu einem Absturz des Spiels führen. Fehlerhafte Erweiterungen werden ignoriert oder, wenn möglich, die Fehler behandelt.

Portierbarkeit
Die Spieleanwendung lässt sich auf unterschiedlichen Betriebssystemen starten und benutzen. Zum Entwickeln der Erweiterungen wird kein konkretes System vorausgesetzt. Es muss nicht gewährleistet werden, dass das Spiel auf jedem System flüssig läuft.

4 Entwurf

In diesem Kapitel werden der Entwurf und der Aufbau des Frameworks erläutert. Dafür wird in Kapitel 4.1 der generelle Aufbau des Frameworks skizziert. Kapitel 4.2 beleuchtet die grundlegenden Eigenschaften des Frameworks und erklärt mit Hilfe welcher Entwurfsmuster diese Eigenschaften sichergestellt werden. Am Ende dieses Kapitels wird in 4.3 – 4.6 näher auf die einzelnen Komponenten des Frameworks und deren Zusammenspiel eingegangen.

4.1 Aufbau des Frameworks

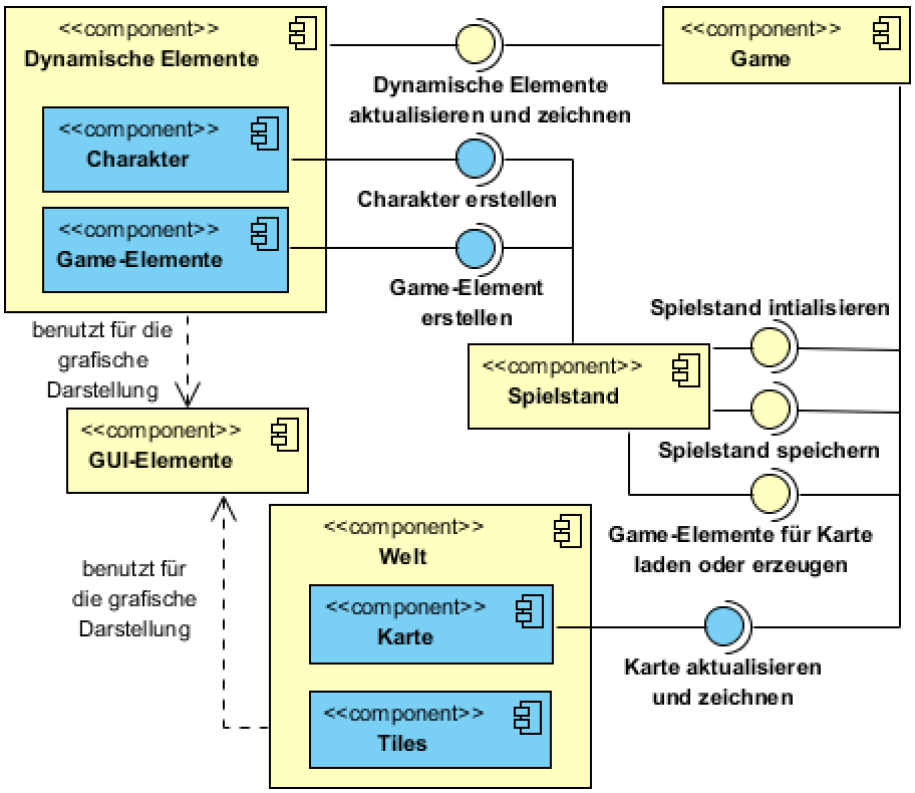

Abb. 4.1 Komponentenmodell des Frameworks

Abb. 4.1 zeigt die Struktur des Frameworks als Komponentendiagramm, sowie wesentliche Schnittstellen zwischen den Komponenten. Die Komponente *Game* beinhaltet die allgemeine Spiellogik und steuert den Spielablauf. Die Komponente *Welt* besteht aus mehreren *Karten*, welche wiederum aus verschiedenen Grafiken, den *Tiles*, zusammengesetzt sind. Jede Karte besitzt Informationen über ihren Aufbau, anhand dieser beim Erstellen einer Karte die interne Zusammensetzung aus verschiedenen Tiles erzeugt wird. Die einzelnen Karten bilden den Hintergrund des Spiels und geben das Raster vor, in dem sich die d*ynamischen Elemente* befinden. Diese teilen sich auf in *Game-Elemente* und den *Charakter*. Pro Spiel gibt es genau einen Charakter, der vom Spieler gesteuert und so durch die Welt bewegt werden kann. Die Game-Elemente befinden sich an fixen Positionen auf den einzelnen Karten, wobei es beliebig viele Game-Elemente pro Karte geben kann. Sowohl die Welt, als auch die dynamischen Elemente nutzen für ihre grafische Darstellung die Vorlagen aus der Komponente *GUI-Elemente*. Diese Vorlagen sind für die Spieleanwendung spezialisiert. Die Komponente *Spielstand* verwaltet alle zustandsbehafteten Objekte und ist verantwortlich für die Persistierung. Der Spielstand wie auch der Charakter sind als Singleton realisiert.

Zur Steuerung des Spieleablaufs nutzt die Komponente Game die Schnittstellen der anderen Komponenten. Zu Spielbeginn bzw. Spielende veranlasst die Game-Komponente das Initialisieren bzw. Speichern des Spielstandes. Bei der Initialisierung kann der Spieler zwischen bereits gespeicherten Spielständen wählen oder einen neuen Spielstand erstellen. Wird ein neuer Spielstand erstellt, kümmert dieser sich um die Erstellung eines neuen Charakters. Während des Spiels sorgt die Game-Komponente dafür, dass die Karte und die dynamischen Elemente regelmäßig aktualisiert und neu gezeichnet werden. Wird die Karte gewechselt (siehe Kapitel 4.5), veranlasst die Game-Komponente außerdem, dass alle Game-Elemente auf der neuen Karte geladen oder erzeugt werden. Dafür besitzt der Spielstand Informationen über die auf den Karten vorhandenen Game-Elemente. Sind die jeweiligen Game-Elemente im Spielstand gespeichert, können sie direkt geladen werden. Sind sie nicht gespeichert, werden sie neu erstellt. (Details siehe Kapitel 4.7)

4.2 Grundlegende Entwurfsentscheidungen

Wichtige Eigenschaften eines jeden Frameworks sind Abstraktion und Erweiterbarkeit. Abstraktion sollte soweit gewährleistet sein, dass der Benutzer des Frameworks mit wenig Aufwand komplexe Konzepte einer Programmiersprache

nutzen oder größere Programme entwickeln kann. Dabei muss es dennoch möglich sein das Framework mit eigenen Funktionalitäten zu ergänzen, um damit nicht nur eine konkrete Aufgabe, sondern eine ganze Klasse von gleichartigen Aufgaben lösen zu können.

Zur Erfüllung dieser beiden Eigenschaften wird in dieser Arbeit unter anderem das *Inversion of Control* Pattern verwendet. Dieses Pattern besagt, dass ein Framework nicht nur Funktionalitäten bereitstellt, die in einem Programm verwendet werden. Vielmehr stellt das Framework selbst ein lauffähiges Programm dar, welches von Programmierern durch verschiedene Schnittstellen erweitert und angepasst werden kann. Zur Realisierung der Schnittstellen und additiven Erweiterung des Frameworks dieser Arbeit werden *Callback-Methoden* und Konfigurationsdateien verwendet.

Durch das Konfigurieren von Dateien mit kommaseparierten Werten kann der Programmierer leicht die Welt gestalten oder mit neuen Karten erweitern (siehe Kapitel 4.5). Das Framework liest automatisch die Konfigurationsdateien ein und überreicht die Informationen an *MapCreator*-Methoden. Dabei handelt es sich um spezielle Methoden, die durch einer Java-Annotation[1] markiert sind und sich um die Erstellung von konkreten Grafiken anhand der übergebenen Informationen kümmern. Dem Programmierer sind bei der Komplexität der Dateiwerte oder der Grafiken keinerlei Grenzen gesetzt.

Das Hinzufügen von Game-Elementen (Kapitel 4.6.2), um die Welt z.B. mit Hindernissen oder interaktiven Objekten zu erweitern, erfolgt ebenfalls über CSV-Konfigurationsdateien. Hierbei muss nur die Position des Game-Elements innerhalb der Welt und eine konkrete Klasse, die sich um die Darstellung und das Verhalten des Game-Elements kümmert, angegeben werden. Die Implementation der konkreten Klasse erfolgt auf Basis eines Interfaces. Dieses Interface definiert bereits Callback-Methoden, die vom Spiel bei der Interaktion mit Game-Elementen ausgeführt werden. So wird es durch Vorgabe weniger Methoden möglich, nahezu beliebige Elemente in das Spiel zu integrieren.

[1] Annotationen sind zusätzliche Informationen innerhalb des Programmcodes, die jedoch nicht zum Code gehören und diesen auch nicht direkt beeinflussen. Die zusätzlichen Informationen können zur Compilezeit oder auch zur Laufzeit des Programms genutzt werden, um abhängig von ihnen diverse Funktionen auszuführen (vlg. https://docs.oracle.com/javase/tutorial/java/annotations/ Abruf: 21.12.2015).

4.3 Die Komponente Game

Die Game-Komponente stellt eine lauffähige Applikation zur Verfügung. Diese beinhaltet ein Fenster mit Zeichen-Canvas und kümmert sich um die Initialisierung aller Komponenten sowie dem Starten der Mainloop. Die Mainloop sorgt analog zu anderen Spieleumgebungen für die zyklische Aktualisierung der Karte und der dynamischen Elemente.

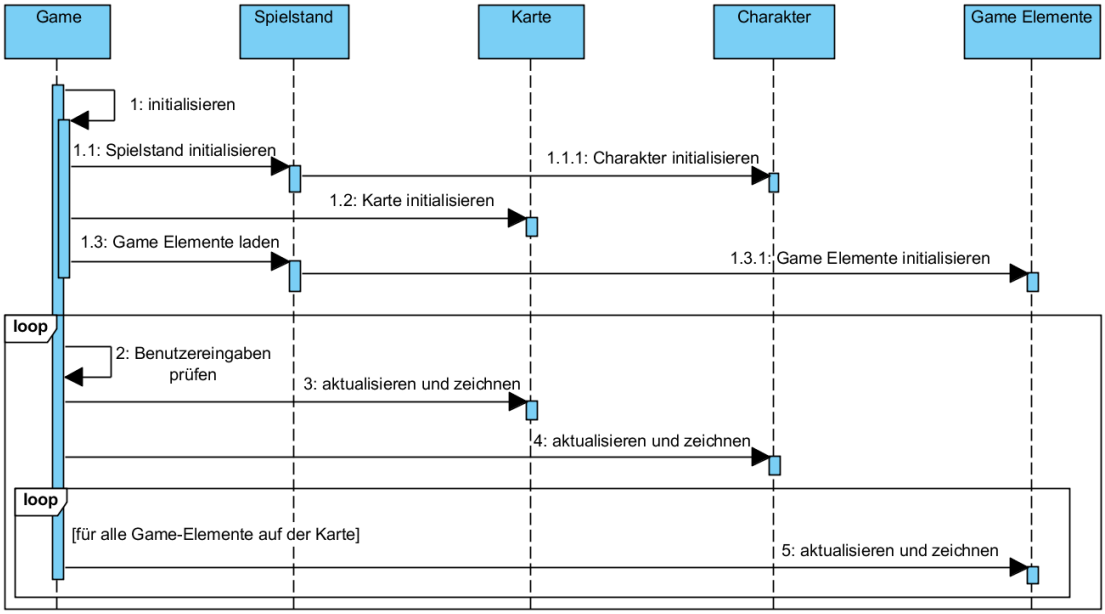

Abb. 4.2 Initialisierung und Mainloop der Game-Komponente

Abb. 4.2 zeigt die Funktionsweise der Applikation. Die Klasse Game bildet den Startpunkt des Spiels und beinhaltet die Spiellogik zum Initialisieren und Aktualisieren aller Komponenten. Nach der Initialisierung des Spielstands, der Karte des Charakters und der Game-Elemente startet die Mainloop. In dieser wird zunächst geprüft, ob es Eingaben vom Benutzer gab. Anschließend werden die darstellbaren Elemente aktualisiert und neu gezeichnet. Das Aktualisieren der einzelnen Elemente kann auf Benutzereingaben reagieren, um die zum Zeichnen benötigten Daten zu berechnen. Berechnungen sind beispielsweise nötig, wenn sich der Charakter bewegen soll oder die Karte animierte Elemente enthält, die für den nächsten Frame berechnet werden müssen[2]. Die Schleife sorgt dafür, dass die

[2] Animationen bestehen aus Einzelbildern, sodass jedes animierte Element für jeden Frame berechnen muss, welches konkrete Bild dargestellt wird.

Aktualisierungen möglichst gleichmäßig angestoßen werden, sodass alle Elemente mit einer festen Framerate aktualisiert werden und Animationen aus Einzelbildern für den Spieler flüssig erscheinen.

4.4 Die Komponente GUI-Elemente

Die Komponente *GUI-Elemente* definiert GUI-Controls, die für die Spieleanwendung spezialisiert sind. Es wird ein Wrapper um das GUI-Framework Java FX geschaffen, welches zum Erstellen der *Application* und des *GraphicsContext* genutzt wird. Die GUI-Controls abstrahieren spieletypische Funktionen und bilden Vorlagen für häufig gebrauchte GUI-Elemente, wie z.B. animierte Grafiken oder Buttons. Darüber hinaus gibt es auch komplexere GUI-Controls, wie Dialoge mit verschiedenen Antwortmöglichkeiten, die mehrere JavaFX-Elemente zusammenfassen. Das alles soll ein einfaches Arbeiten mit der Spieleumgebung ermöglichen, ohne mit JavaFX in Berührung zu kommen.

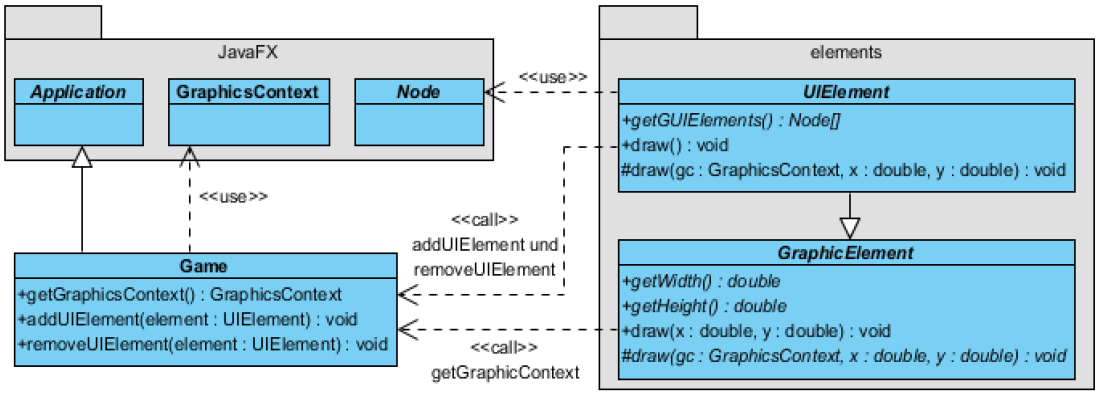

Abb. 4.3 Hauptklassen der Komponente "GUI-Elemente"

Das Klassendiagramm in Abb. 4.3 zeigt die beiden Hauptklassen der Komponente GUI-Elemente und den Zusammenhang zum JavaFX Framework. Die Klassen *UIElement* und *GraphicElement* bilden die beiden Superklassen von denen alle anderen GUI-Controls ableiten. Die Klasse GraphicElement dient als Superklasse für beliebig darstellbare Elemente. Von Subklassen wird nur verlangt, dass sie sich auf dem JavaFX GraphicsContext zeichnen können. UIElement ist eine spezialisiertere Superklasse für GUI-Controls, die eine Sammlung von JavaFX Nodes kapselt. Durch Ableiten von den Klassen GraphicElement und UIElement ist es ebenfalls möglich, eigene GUI-Controls zu definieren, die zusammen oder anstelle der vorgegebenen GUI-Controls verwendet werden können.

Die linke Seite der Abb. 4.3 zeigt, dass auch die Game-Klasse aus der Komponente Game (Kapitel 4.3) für ihre grafischen Funktionen auf das JavaFX Framework zugreift. Durch das Ableiten von der Klasse *Application* wird automatisch eine JavaFX *Stage* erstellt, zu dem sich ein Element mit einem GraphicsContext und beliebige Nodes hinzufügen lassen. Der GraphicsContext von Game wird von Objekten der Klasse GraphicElement zur Darstellung der Welt, des Charakters und der Game-Elemente genutzt und kann über die Methode *getGraphicsContext* ermittelt werden. Objekte der Klasse *UIElement* nutzen die Möglichkeit zu Stage beliebige Nodes hinzuzufügen, um z.B. Sprech- und Handelsdialoge oder Menüs einzublenden. Die Klasse Game bietet dafür die Methoden *addUIElement* zum Hinzufügen und *removeUIElement* zum Entfernen der entsprechenden Objekte an.

4.5 Die Komponente Welt

Die Welt, in der sich der Spieler mit seinem Charakter bewegt, ist in verschiedene Karten aufgeteilt. Die einzelnen Karten sind in einem Gitter angeordnet, sodass jede Karte maximal acht Nachbarkarten besitzen kann. Jede Karte besitzt ebenfalls wieder eine Gitterstruktur, in welcher sich der Charakter bewegt. Dieser allgemeine Weltaufbau ist in Abb. 4.4 bespielhaft dargestellt. Blau umrandet sind insgesamt neun Karten in einem drei mal drei großen Raster angeordnet. Durch dieses Raster ergeben sich, ausgehend von der mittleren Karte, auch die Kartenpositionen. Die orangen Pfeile zeigen, auf welche Nachbarkarten ein Charakter, der sich auf der mittleren Karte befindet, wechseln kann.

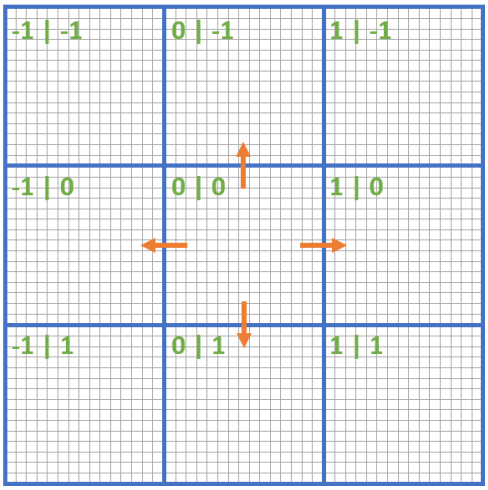

Abb. 4.4 Allgemeiner Aufbau einer bespielhaften Welt

Die Bewegung des Charakters ist auf vier Himmelsrichtungen begrenzt. Es ist weder möglich sich mit seinem Charakter auf einer Karte diagonal zu bewegen, noch die Karte diagonal zu verlassen. Will ein Charakter von der Karte an der Position 0|0 zur Karte an der Position 1|-1, so muss dieser Charakter zuerst auf die Karte an der Position 0|-1 oder 1|0 wechseln, um von dort aus an sein Ziel zu kommen.[3]

Abb. 4.5

Abb. 4.5 zeigt den Zusammenhang zwischen der Karte und den Tiles.[4] Wie in Abb. 4.4 durch die graue Gitterstruktur dargestellt, ist jede Karte in 15x15 gleich große Quadrate unterteilt. Jedes dieser Quadrate kann man mit mindestens einem Tile befüllen, sodass eine Karte aus mindestens 15x15 = 225 Tiles besteht. Alle Tiles besitzen die gleiche Größe und beinhalten je genau eine Grafik. Mit Hilfe von transparenten Bereichen innerhalb dieser Grafiken ist es möglich, verschiedene Landschaften durch Überlagerung zu erzeugen. Bei den Grafiken der Tiles handelt es sich um Dateien, die im Zuge des *Liberated Pixel Cup (LPC)* entstanden sind. Der LPC ist ein Online-Wettbewerb zur Erstellung von frei lizensierten Grafiken und anschließender Entwicklung von Spielen unter Verwendung dieser Grafiken (vgl. LPC 2016). Durch den LPC wird neben dem Dateityp PNG auch die Größe der Grafiken mit 32x32 Pixeln festgelegt, was sich in dieser Arbeit durch die fixe Größe der Tiles und die fehlende Möglichkeit der Skalierung widerspiegelt.

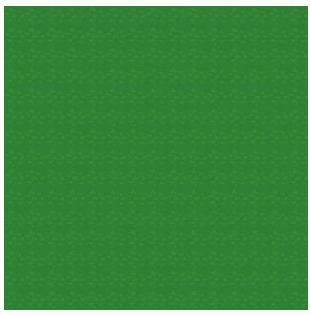

Abb. 4.6

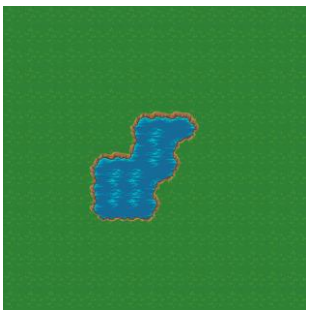

Abb. 4.7

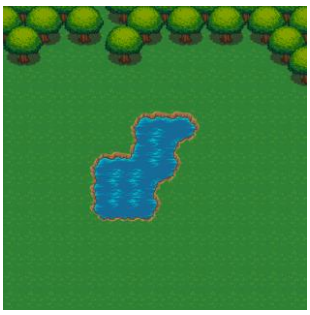

Abb. 4.8

Sequentieller Aufbau einer Karte mit Gras, einem See und Bäumen

[3] Die Begrenzung ist durch die verwendeten Grafiken vorgegeben, da diese nur Animationen in vier Himmelsrichtungen beinhalten.

[4] Aus Gründen der Übersichtlichkeit stellt Abb. 4.5 nur einen Teil des Klassendiagramms dar.

Abb. 4.8 zeigt den beispielhaften Aufbau einer Karte mit einer Wiese, einem See und Bäumen. Betrachtet man die einzelnen Tiles pro Quadrat auf der gesamten Karte, so ergeben sich mehrere Ebenen, die die einzelnen Aspekte einer Karte beinhalten. Abb. 4.6 zeigt nur die unterste Tile-Ebene, welche die grundlegende Graslandschaft beinhaltet. Abb. 4.7 und Abb. 4.8 demonstrieren, wie sich durch das Hinzufügen von weiteren Ebenen mit einem See und Bäumen die Karte Stück für Stück aufbauen lässt. Dabei muss nicht jede Ebene aus 15x15 Tiles bestehen und somit die komplette Karte abdecken. Die zweite Ebene mit dem See besitzt beispielsweise nur Tiles im mittleren Bereich der Karte und beeinflusst die anderen Kartenbereiche nicht.

Die Tiles eines Quadrates steuern, ob Bereiche einer Karte betreten werden können. Geht man davon aus, dass ein Charakter nicht durch Bäume gehen kann, ist ein Verlassen der Karte aus Abb. 4.8 nach Norden nicht möglich. Somit ist es auch nicht direkt möglich auf die nördliche Nachbarkarte zu wechseln. Diese Einschränkung der Zugänglichkeit von manchen Karten kann zum einen genutzt werden um Inhalte des Spiels, wie z.B. eine Stadt, die von einer Mauer umgeben ist, zu modellieren. Andererseits können durch solche Beschränkungen auch die Grenzen der Welt fixiert werden.

Neben dem Zusammenhang zwischen der Karte und den Tiles macht Abb. 4.5 deutlich, woher eine Karte die Informationen ihres Aufbaus erhält. Beim Erzeugen einer neuen Karte, liest diese die Dateien mit den Tile Informationen ein und erzeugt daraus die einzelnen Tiles. Jede Datei beschreibt genau eine Tile-Ebene und besteht aus maximal 15x15 kommaseparierten Werten, die jeweils für ein bestimmtes Tile stehen. Das Erzeugen der konkreten Tiles anhand dieser Werte übernehmen bestimmte statische Methoden, so genannte *MapCreator*, die durch eine Java Annotation gekennzeichnet sind. Die Annotation enthält zudem Informationen über die Ebenen, für die die Methode zuständig ist. Zur Unterscheidung der einzelnen Tile-Ebenen werden diese durchnummeriert. Die unterste Ebene erhält die Nummer „0“.

Das Sequenzdiagramm in Abb. 4.9 stellt die Erzeugung von Tiles einer Ebene mit Hilfe von MapCreator-Methoden als zweistufigen Vorgang dar. Die Klasse „Parser“ steht repräsentativ für alle Klassen, die eine MapCreator-Methode beinhalten. Zuerst liest die Karte die entsprechende Datei ein und erstellt eine Tile-Matrix, die mit *UndefinedTiles* (siehe Abb. 4.10) befüllt wird. Diese Tiles dienen als Platzhalter in der Matrix und als Container für die einzelnen Werte aus der Datei. Anschließend werden alle MapCreator-Methoden, die für die aktuelle Tile-Ebene zuständig sind, nacheinander aufgerufen. Jede Methode verändert dabei die Matrix, indem die UndefinedTiles anhand der gekapselten Werte durch konkrete Tile-Objekte ersetzt werden. Diese veränderte Matrix wird wiederum der nächsten MapCreator-Methode übergeben, sodass nach und nach alle UndefinedTiles in der Matrix ersetzt werden. Nachdem die Matrix durch alle MapCreator-Methoden verändert wurde, wird sie als neue Tile-Ebene der Karte hinzugefügt.

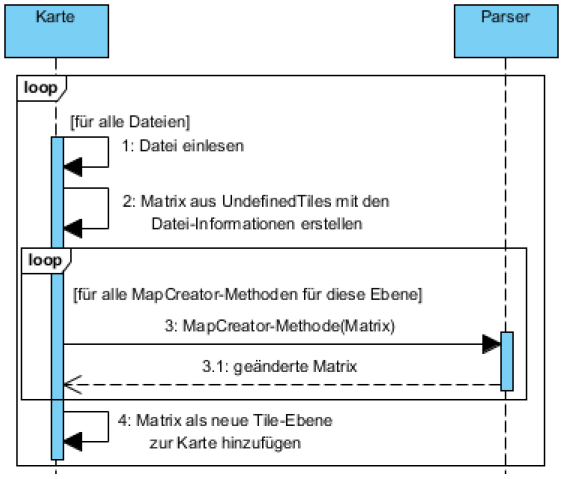

Abb. 4.9 Erstellen von Tiles einer Ebene

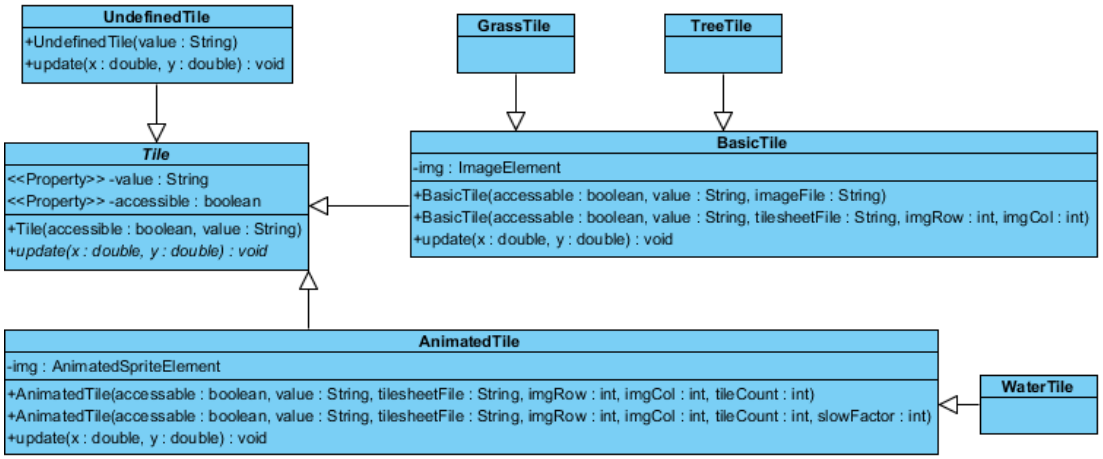

Abb. 4.10 Tile-Klassen

Abb. 4.10 zeigt die verschiedenen Tile-Klassen. Die abstrakte Klasse *Tile* definiert die Schnittstelle aller Tiles. Die Eigenschaft *value* beinhaltet den zu dem Tile gehörenden Wert aus einer Datei und die Eigenschaft *accessible* gibt an, ob der

Charakter das Tile betreten kann. Die *update*-Methode dient, falls notwendig, zur zyklischen Aktualisierung und zur Zeichnung des Tiles auf dem GraphicsContext (Kapitel 4.4). *GrassTile*, *TreeTile* und *WaterTile* sind konkrete Tile-Klassen, die zu Test- und Vorführungszwecken implementiert wurden. Sie stellen eine Möglichkeit dar Tiles zu spezialisieren, indem die Eigenschaften in einer separaten Klasse beschrieben werden. Eine weitere Möglichkeit besteht darin, direkt Objekte der Klassen *BasicTile* oder *AnimatedTile* zu erzeugen und alle relevanten Informationen bei der Initialisierung mitzugeben. Objekte der Klasse BasicTile nutzen zur Darstellung ein ImageElement (beinhaltet eine einfache Grafik), wohingegen Objekte der Klasse AnimatedTile ein AnimatedSpriteSheet (beinhaltet eine animierte Grafik bestehend aus Einzelbildern) zeichnen können. Objekte der Klasse UndefinedTile werden beim Zeichnen als schwarze Quadrate dargestellt, um sicher zu gehen, dass jede Karte darstellbar ist, auch wenn beim Parsen der Karten-Dateien ein Wert auftritt, der von keiner MapCreator-Methode verarbeitet werden kann.

4.6 Die Komponente Dynamische Elemente

Neben der Welt, welche zu Spielbeginn fest steht und sich während des Spiels nicht verändert, gibt es auch dynamische Elemente. Der Begriff „dynamisch“ bezieht sich auf die Veränderlichkeit der Elemente während des Spiels. Die dynamischen Elemente teilen sich auf in einen vom Spieler steuerbaren Charakter und mehrere Game-Elemente. Der Charakter kann zwischen einzelnen Karten hin und her wechseln und sich so in der ganzen Welt bewegen. Die einzelnen Game-Elemente sind hingegen je genau einer Karte zugeordnet.

4.6.1 Der steuerbare Charakter

Als Repräsentant in diesem Spiel besitzt jeder Spieler wie in vielen Rollenspielen üblich eine Figur, den sogenannten Charakter. Mit dem Charakter kann der Spieler die Welt erkunden oder mit anderen Elementen im Spiel interagieren. Welche anderen Elemente dies sein können und wie diese Interaktion aussieht, ist in Kapitel 4.6.2 beschrieben. Durch solche Interaktionen kann der Spieler direkt Einfluss auf das Spiel nehmen.

Der Charakter ist als Singleton-Pattern realisiert, um dafür zu sorgen, dass nur genau ein Charakter die Benutzereingaben behandelt und sich durch die Welt bewegen kann. Zudem wird ein einfacher Zugriff auf den Charakter auch für spätere

Erweiterungen geschaffen. Zum Erschaffen eines Charakters gibt es zwei Möglichkeiten:

- Das Gestalten des Charakters über eine GUI, dem *Charakter-Editor*.
- Das Laden eines zuvor gespeicherten Charakters aus dem Spielstand, was in Kapitel 4.7 genauer beschrieben wird.

Der Charakter-Editor verwendet das Factory Pattern, um die Erstellung eines *PreviewCharacter* (Abb. 4.11) zu kapseln. Dieser PreviewCharacter beinhaltet den Namen und alle Informationen zur grafischen Darstellung des Charakters und ist die Basis, um einen neuen Charakter zu erstellen. Mit Hilfe des Editors kann sich der Spieler seinen Charakter individuell gestalten und Eigenschaften wie z.B. Geschlecht, Hautfarbe oder Frisur wählen. Die Kleidung ist anfangs für jeden Charakter gleich festgelegt, kann jedoch über Schnittstellen des Charakters verändert werden. Die Verwendung des Charakter-Editors sorgt für einen spielerischen Einstieg in die Arbeit mit dem Framework. Zudem kapselt er alle relevanten Daten zur grafischen Darstellung des Charakters und abstrahiert somit beim Initialisieren diese Teile von der Spielelogik des Charakters. Nach dem Erstellen wird auf Basis des PreviewCharacter durch einen Copy-Konstruktor der *ControllableCharacter* erzeugt.

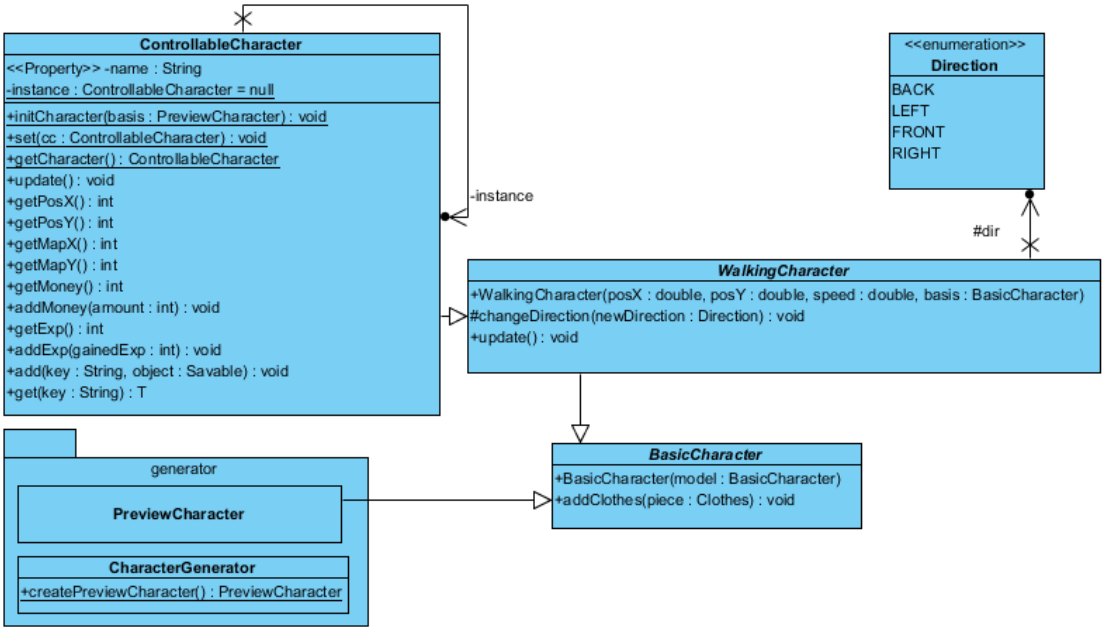

Abb. 4.11 Die Klassen des Charakters

Abb. 4.11 zeigt die Vererbungshierarchie des Charakters als Klassendiagramm[5]. Die Klasse ControllableCharacter ist die Repräsentation des Charakters und erbt von den Klassen *WalkingCharacter* und *BasicCharacter*. BasicCharacter kennt alle Grafiken für die Darstellung des Charakters und seiner Kleidung. Deshalb ist diese Klasse auch die Superklasse von PreviewCharacter. Zudem bietet BasicCharacter die Methode *addClothes* an, mit der sich die Kleidung des Charakters verändern lässt.

Die Klasse WalkingCharacter sorgt für die Animation und Darstellung des Charakters. In der *update*-Methode dieser Klasse erfolgt die Fortbewegung des Charakters durch Änderung seiner Position sowie das Neuzeichnen aller Grafiken, sodass aus Einzelbildern eine Animation entsteht. In der Klasse ControllableCharacter wird die update-Methode erweitert, um eine Steuerung des Charakters zu ermöglichen. Hierfür wird auch die Methode *changeDirection* benutzt, um den Charakter zu drehen. Zusätzlich bietet ControllableCharacter Methoden an, um Eigenschaften des Charakters abzufragen oder zu verändern. Hierzu gehören auch die Methoden *add* und *get*, mit denen der Charakter um beliebige Eigenschaften erweitert werden kann. Diese werden in einer Property-Value-Map abgelegt und zusammen mit dem Charakter gespeichert. Die drei statischen Methoden und die statische Variable *instance* stellen die Realisierung des Singleton-Patterns dar. Die Methode *set* wird vom Spielstand (Kapitel 4.7) benutzt, um die Singelton-Instanz mit einem vorher gespeicherten Charakter zu belegen.

4.6.2 Game-Elemente

Neben dem Charakter, der von einem Spieler gesteuert wird, sieht das Framework noch andere dynamische Elemente, so genannte Game-Element im Spiel vor und bietet dafür ein Interface (Abb. 4.12) an. Der Art der Elemente sind dabei keine Grenzen gesetzt. Denkbar wären einfache Steine, Bienen oder andere Insekten, die durch die Luft fliegen bis hin zu Personen mit denen sich der Charakter in einer festgelegten Art und Weise unterhalten kann. Wie auch der Charakter werden die Game-Elemente auf dem Hintergrund der Karte gezeichnet. Das regelmäßige Neuzeichnen der Elemente, z.B. um diese anhand von Einzelbildern zu animieren, erfolgt innerhalb der Methode *update*. Die Game-Elemente können einzelne Felder oder Bereiche der Karte blockieren, indem sie dem Charakter verbieten durch sie hindurch zu gehen. Ob das Game-Element ein Hindernis darstellt, prüft das Game mit der Methode *isAccessible*.

[5] Das Klassendiagramm ist nicht vollständig. Es wurden zugunsten der Übersicht einige Methoden herausgenommen, die vorrangig innerhalb des Spiele-Frameworks genutzt werden.

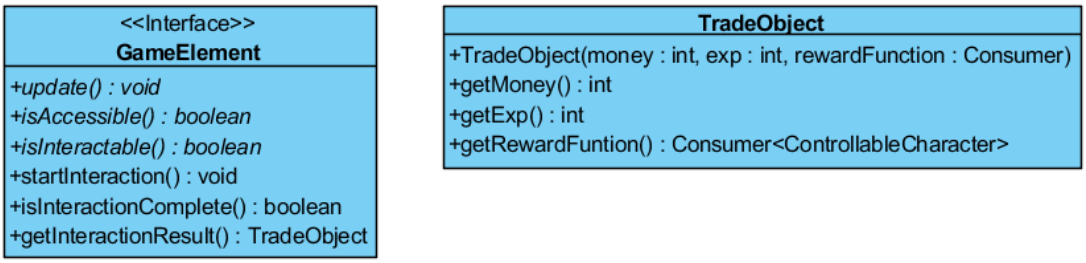

Abb. 4.12 Das GameElement-Interface und die TradeObject-Klasse

Mit Hilfe von Game-Elementen ist es auch möglich interaktive Objekte, z.B. freundliche bzw. feindliche Kreaturen oder Zaubertränke in das Spiel zu integrieren und so dem Charakter mehr ermöglichen, als nur die Welt zu erkunden. Drückt der Spieler eine bestimmte Interaktionstaste, versucht der Charakter mit dem vor sich befindenden Game-Element zu interagieren. Dies gelingt, wenn sich vor dem Charakter ein Game-Element befindet, dessen *isInteractable*-Methode den Wert true liefert. Die anschließende Interaktion besteht aus drei Phasen:

1. Starten der Interaktion
2. Durchlaufen der Interaktion
3. Verarbeiten des Ergebnisses der Interaktion

Die erste Phase beginnt durch den Aufruf von *startInteraction* auf dem Game-Element. Programmiert man beispielsweise mit Hilfe des Game-Element-Interfaces einen Händler, bei dem der Charakter Gegenstände kaufen kann, dient die Start-Phase dazu mit Hilfe bestimmter GUI-Controls (Kapitel 4.4) eine Verkaufsoberfläche zu erstellen und anzuzeigen. Solange der Charakter die zum Verkauf stehenden Gegenstände betrachtet und aussucht, welche er kaufen möchte, läuft die zweite Phase der Interaktion. Das Game fragt innerhalb der Mainloop (Kapitel 4.3) regelmäßig durch die Methode *isInteractionComplete* beim Game-Element nach, ob die zweite Phase der Interaktion beendet ist. Im Beispiel des Händlers ist dies der Fall, wenn der Charakter den Kauf der ausgewählten Gegenstände bestätigt oder abbricht.

Stellt das Game fest, dass Phase zwei einer Interaktion beendet ist, stößt es die dritte Phase an. Die Methode *getInteractionResult* liefert das Ergebnis der Interaktion in Form eines *TradeObject* zurück. Diese Klasse dient als Container für mögliche Änderungen am Charakter. Hat der Charakter beim Händler etwas gekauft, müssen ihm natürlich Geld abgezogen und die gekauften Gegenstände hinzugefügt werden. Geld und Erfahrungspunkte treten häufig als Ergebnis einer

Interaktion, wie z.B. bei Kämpfen oder dem Auffinden von Schätzen, auf und wurden deshalb direkt in TradeObject vorgesehen. Um dem Charakter auch die gekauften Gegenstände zu geben, gibt es zusätzlich noch die Möglichkeit einen *Consumer* anzugeben. Dabei handelt es sich um eine beliebige Methode, die einen ControllableCharacter als Parameter annimmt. Das Game sorgt dafür, dass alle Änderungen, die in einem TradeObject gekapselt sind, als Abschluss der Interaktion auf den Charakter angewandt werden. Dafür werden zuerst dem Charakter über seine Schnittstellen (siehe Abb. 4.11) der Geldbetrag und die Erfahrungspunkte hinzugefügt. Anschließend wird der Consumer ausgeführt, indem der Charakter als Parameter übergeben wird. Innerhalb dieser Methode werden dann weitere Veränderungen am Charakter vorgenommen, wie z.B. die gekauften Gegenstände hinzufügen.

posX	posY	info	class
5	8	100%	potions.HealingPotion
10	13		enemies.Spider

Abb. 4.13 Game-Element-Informationen aus CSV-Datei als Tabelle

Um Objekte verschiedener Game-Elemente automatisch beim Betreten einer Karte erzeugen zu lassen, gibt es für jede Karte eine CSV-Datei mit Informationen über die auf ihr enthalten Game-Elemente. Der Name der Datei beinhaltet die Kartenposition, sodass dadurch eine eindeutige Zuordnung möglich ist. Der Spielstand hat Zugriff auf alle CSV-Dateien mit Game-Element-Informationen und sorgt für die automatische Erstellung der Objekte (Kapitel 4.7). Abb. 4.13 stellt die Informationen einer beispielhaften CSV-Datei als Tabelle dar. Die drei Felder *posX*, *posY* und *class* sind für jedes Game-Element zwingend nötig, da sie die Position auf der Karte und die konkrete Klasse für das Game-Element beschreiben. Das Feld *info* bietet die Möglichkeit dem Objekt der Klasse beim Initialisieren Informationen in Form einer Zeichenkette mitzugeben. Im Beispiel aus der Grafik wird dieses Feld genutzt, um die Stärke der Wirkung eines Heiltranks anzugeben. Eine andere denkbare Verwendung wäre z.B. eine Auflistung von Gegenständen, die ein bestimmter Händler zum Verkauf anbietet.

4.7 Die Komponente Spielstand

Wenn man ein Spiel beendet und später erneut startet, erwarten die meisten Spieler an dem Punkt wieder anknüpfen zu können, an dem sie das Spiel beendet haben. Um diesen Erwartungen auch in diesem Framework gerecht zu werden,

werden alle zustandsbehafteten Objekte in einem Spielstand verwaltet. Beim Beenden des Spiels, wird der aktuelle Spielstand automatisch gespeichert und kann zu einem späteren Zeitpunkt wieder geladen werden. Es ist möglich mehrere Spielstände gleichzeitig zu besitzen, sodass beim Starten des Spiels der Spieler einen bereits vorhandenen Spielstand laden oder einen neuen Spielstand erstellen kann.

Hauptbestandteil des Spielstands bildet der Charakter des Spielers. Durch die Singleton-Eigenschaft des Charakters entsteht eine eins-zu-eins-Beziehung zwischen Spielstand und Charakter. Diese eindeutige Zuordnung wird insofern genutzt, dass die einzelnen Spielstände mit dem Namen des enthaltenen Charakters betitelt werden. Dadurch ist es für den Spieler einfacher, die einzelnen Spielstände zu unterscheiden.

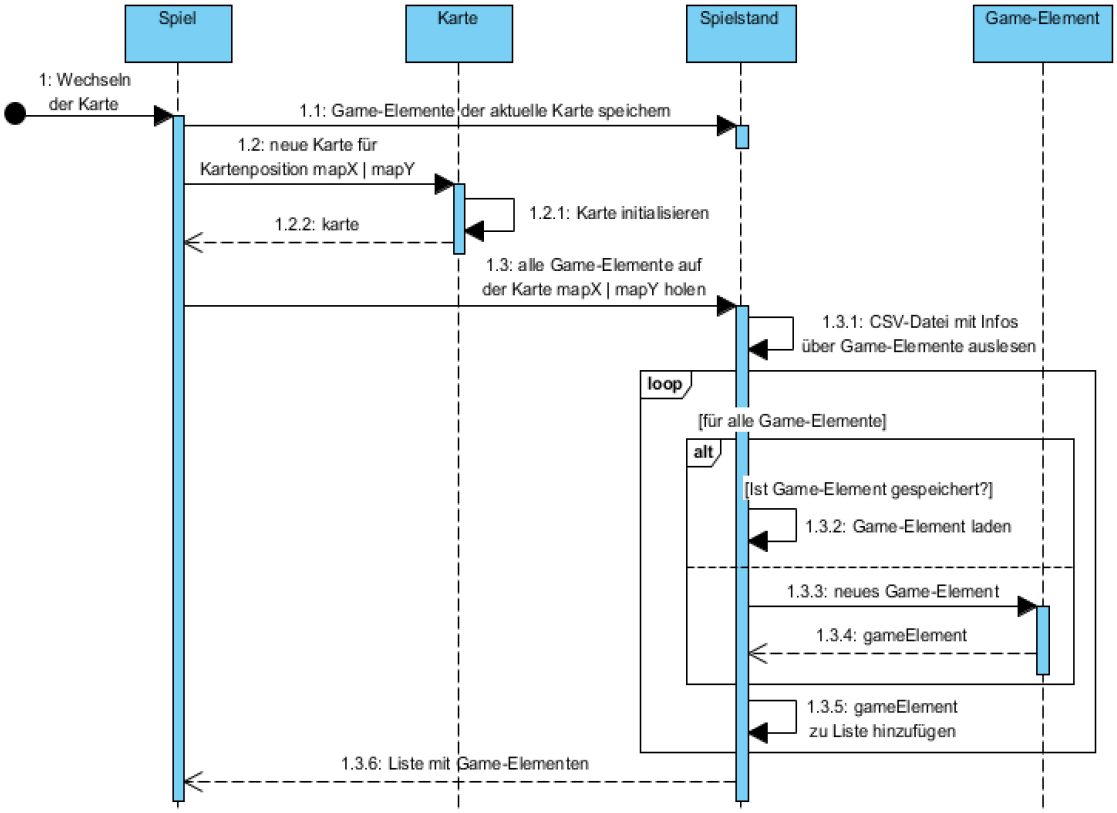

Abb. 4.14 Erstellen von Game-Elementen mit Hilfe des Spielstands

Neben dem Charakter werden auch alle Game-Elemente im Spielstand gespeichert. Abb. 4.14 zeigt, wie beim Wechseln einer Karte zuerst die Game-Elemente der alten Karte gespeichert und anschließend alle auf der neuen Karte befindlichen Game-

Elemente erstellt werden. War der Charakter schon einmal auf der neuen Karte, sind die Game-Elemente bereits im Spielstand gespeichert und werden von dort geladen. Wenn der Charakter die Karte das erste Mal betritt, werden die Game-Elemente anhand der Informationen aus einer CSV-Datei (siehe Kapitel 4.6.2) neu erstellt.

Zu Beginn eines Spiels, wird der vom Spieler ausgewählte Spielstand geladen und aus ihm der Charakter initialisiert. Wählt der Spieler die Möglichkeit, einen neuen Spielstand zu erzeugen, wird zum Initialisieren des Charakters der Charakter-Editor aufgerufen (siehe Kapitel 4.6.1) und der neue Charakter automatisch dem neuen Spielstand hinzugefügt. Beim Beenden eines Spiels wird der Spielstand und mit ihm alle enthaltenen Objekte gespeichert. Vor der Speicherung werden automatisch Callback-Methoden aufgerufen, um z.B. offene Datei-Ressourcen freizugeben. Die Kennzeichnung der Callback-Methoden wird mit Hilfe von Annotationen realisiert.

4.8 Zusammenfassung

Dieses Kapitel hat durch Erläuterung der einzelnen Komponenten des Frameworks die Umsetzung der funktionalen Anforderungen gezeigt. In der Komponente Game (Kapitel 4.3) befindet sich die lauffähige Spieleanwendung, die dafür alle anderen Komponenten benutzt. Wie der Aufbau der Welt und die Gestaltung über Konfigurationsdateien funktioniert, erklärt das Kapitel 4.5. Die Realisierung der Charaktergestaltung und die animierten Bewegungen des Charakters sind in Kapitel 4.6 beschrieben. Zudem werden dort die Schnittstellen erklärt, mit denen man dem Charakter Eigenschaften hinzufügen und interaktive Objekte im Spiel einzubauen kann. Zu guter Letzt befasst sich das Kapitel 4.7 mit der Komponente Spielstand und wie die automatische Sicherung des Spiels funktioniert.

Abgesehen von den Java-Annotationen wurde für die Umsetzung des Frameworks auf leicht verständliche Programmierkonzepte, wie Konfigurationsdateien und Callback-Methoden, gesetzt. Annotationen wurden eingesetzt, da ihre Verwendung simpel und sehr anschaulich ist. Das Framework übernimmt komplexe Aufgaben selbstständig und unterstützt so den Benutzer. Dazu zählt unter anderem das Auslesen von Informationen aus Annotation per Reflection, das Parsen der Konfigurationsdateien und das regelmäßige Aktualisieren aller Game-Elemente.

5 Evaluation

Zum Testen der Praxistauglichkeit des Frameworks wurde eine Aufgabenstellung entworfen, die im Rahmen eines Tutoriums als betreute Übung und von weiteren Studierenden in ihrer Freizeit selbstständig bearbeitet werden konnte. Kapitel 5.1 beschreibt den Inhalt und die Durchführung der Aufgabe. Anschließend werden in Kapitel 5.2 die Ergebnisse einer Umfrage zu der Aufgabe und dem Konzept dieser Arbeit untersucht. Kapitel 5.3 zieht anhand der vorgestellten Ergebnisse ein Fazit über das in dieser Arbeit entwickelte Konzept.

5.1 Testaufgabe

Inhalt der Aufgabe ist es, einige MapCreator-Methoden zum Erstellen der Karten (siehe Kapitel 4.5) zu programmieren. Die Aufgabenstellung ist in vier Teilaufgaben untergliedert, um den Studierenden einen Leitfaden mitzugeben und es zu ermöglichen, dass nach einigen Zeilen Code bereits eine Änderung am Spiel sichtbar ist. Diese Teilaufgaben sind im Nachfolgenden näher beschrieben.

Die Lösung der Aufgabe erfolgte auf zwei unterschiedliche Arten. Ein Teil der Studierenden löste die Aufgabe während eines Tutoriums und konnte sich bei Fragen direkt an den Verfasser dieser Arbeit wenden. Der andere Teil löste die Aufgabe selbstständig alleine oder in kleinen Gruppen außerhalb des Tutoriums. Sie hatten zwar die Möglichkeit Fragen per E-Mail zu stellen, jedoch wurde diese Möglichkeit nicht genutzt.

Teilaufgabe 1 und 2
In den Teilaufgaben 1 und 2 geht es um ein einfaches Abbilden von String-Werten auf Grass- und Wasser-Tiles. Über das Tile-Array der Karte soll iteriert werden und

an den Positionen, an denen in der Textdatei ein „G“ oder „W“ steht, sollen entsprechende Tiles erzeugt und als Werte dem Array zugewiesen werden. Die ersten beiden Aufgaben dienen als Einstieg in das Framework und machen die Studierenden mit den vorgegebenen Methoden und Klassen vertraut. Zudem wird eine anschauliche Arbeit mit mehrdimensionalen Arrays angeregt.

Teilaufgabe 3
Die dritte Teilaufgabe stellt eine Erweiterung der Erzeugung der Wasser-Tiles dar. Um nicht nur Wasser, sondern auch das Ufer korrekt anzeigen zu lassen, müssen den Wasser-Tiles entsprechende Informationen, ob eine Ufergrafik verwendet werden soll oder nicht, bei der Erstellung der Objekte mitgegeben werden. Die Übergabe der Informationen erfolgt durch ganzzahlige Werte, welche die einzelnen Typen von Wassergrafiken darstellen. Es wurden hier keine Enums verwendet, da dieses Konzept zum Zeitpunkt des Semesters noch nicht allen Studierenden bekannt war. Zur Bestimmung der Typen müssen die Studierenden die Nachbarn der einzelnen Tiles bestimmen, was die Arbeit mit den Indizes eines Array fördert und ein Verständnis für den Aufbau der Welt verbessert.

Teilaufgabe 4
Teilaufgabe 4 ist eine Zusatzaufgabe, um Studierenden, die mit den ersten drei Teilaufgaben gut zurechtgekommen sind, noch eine weitere Übungsaufgabe zur Verfügung stellen zu können. Inhalt dieser Teilaufgabe ist die korrekte Darstellung von Bäumen. Da Bäume aus mehreren Tiles zusammengesetzt sind, müssen anhand unterschiedlicher String-Werte aus der Textdatei die verschiedenen Baum-Tiles erzeugt werden. Ziel ist es, dass die Studierenden die Nützlichkeit einer Mehrfachbedingung[6] in dieser Situation erkennen und diese anstatt von geschachtelten bedingten Anweisungen verwenden.

5.2 Auswertung der Umfrageergebnisse

Die Studierenden wurden gebeten nach dem Bearbeiten der Aufgabe an einer anonymen Umfrage teilzunehmen, um das in dieser Arbeit entwickelte Konzept zu bewerten. Insgesamt haben 15 Studierende an der Umfrage teilgenommen, von denen bis auf eine Ausnahme alle das Konzept positiv bewerteten. Im Folgenden werden die einzelnen Ergebnisse der 10 Fragen näher betrachtet.

[6] gemeint ist hiermit z.B. die switch-case-Anweisung in Java

In der ersten Frage wurde nach den Vorkenntnissen der Studierenden in verschiedenen Bereichen gefragt. Hiermit wurde geprüft, ob die Teilnehmer auch in die Zielgruppe dieser Arbeit passen. Abb. 5.1 zeigt die Ergebnisse der ersten Frage. Der Großteil der Befragten gab an, wenig oder mittelmäßig viele Vorkenntnisse in der Programmierung und mit Java zu besitzen. Sie liegen damit in der festgelegten Zielgruppe. Lediglich je 3 Studierende gaben an, viele Vorkenntnisse im Programmieren und mit Java zu besitzen. Eine detailliertere Auswertung der Antworten hat ergeben, dass diese Angaben von den drei gleichen Studierenden gemacht wurden. Diese Studierenden passen zwar nicht genau in die Zielgruppe, bieten jedoch eine gute Referenz, ob das Konzept auch für andere Personen interessant ist.

	Keine (1)		Wenig (2)		Mittel (3)		Viel (4)	
	Σ	%	Σ	%	Σ	%	Σ	%
im Programmieren	-	-	2x	13,33	10x	66,67	3x	20,00
mit Java	3x	20,00	6x	40,00	3x	20,00	3x	20,00
mit Spielen	3x	20,00	3x	20,00	3x	20,00	6x	40,00
mit der Spieleentwicklung	6x	40,00	7x	46,67	2x	13,33	-	-

Abb. 5.1 Antworten auf die Frage "Wie viel Vorkenntnisse hast du?"

Frage zwei lautete „Hat es dir Spaß gemacht an einem Spiel zu programmieren?“. Diese Frage haben 14 von 15 der befragten Studierenden mit „Ja“ beantwortet. Auch auf die dritte Frage, ob die visuelle Darstellung des Codes durch Änderungen im Spiel hilfreich sei, antworteten 10 der Befragten mit „Ja“ und vier mit „Eher ja“. Diese eindeutige Tendenz der Antworten zeigt, dass ein grundlegendes Ziel erreicht wurde, nämlich durch die Spieleumgebung das Programmieren zu erleichtern und mit Spaß zu verbinden.

Die Fragen vier und fünf zielen auf die Gestaltung der Aufgabe ab. Die Studierenden wurden gefragt, welche der vier Teilaufgaben sie erfolgreich bearbeiten konnten. Alle Studierenden haben die ersten beiden Teilaufgaben lösen können, über die Hälfte haben alle Teilaufgaben erfolgreich bearbeitet. Zudem wurde die Frage nach der Verständlichkeit der Aufgabe gestellt. 80 Prozent gaben an, dass sie einiges mehrmals lesen mussten um es zu verstehen und 20 Prozent haben die Aufgabenstellung sogar als „nicht gut“ eingestuft und gaben an, vieles nicht verstanden zu haben.

Diese Ergebnisse zeigen deutlich, dass eine einfache und verständliche Formulierung der Aufgabenstellung nicht trivial, aber dennoch wichtig ist. Zudem sollte bei der Stellung der Aufgaben der Wissensstand der Studierenden mit berücksichtigt werden, um einen kontinuierlich Fortschritt zu ermöglichen. Um auf unterschiedliche Wissensstände eingehen zu können, kann man beispielsweise Zusatzaufgaben formulieren oder die Aufgaben in Kleingruppen lösen lassen, sodass jeder seine Kenntnisse mit einbringen kann.

Mit den Fragen sechs und sieben wurde der Lernerfolg der Studierenden erfragt. 13 Teilnehmer gaben an etwas bei der Aufgabe gelernt zu haben. Abb. 5.2 zeigt die Verteilung der Antworten auf die konkrete Nachfrage welche angestrebten Aspekte der Programmierung inwieweit geübt wurden. Die Mehrheit der Studierenden gab an, dass sie durch die Übung drei oder mehr Aspekte vertiefen konnten. Somit hatten die Studierenden nach eigener Einschätzung nicht nur Spaß beim Programmieren mit dem Framework, sondern konnten dabei auch noch ihre Programmierfähigkeiten festigen oder verbessern.

	konnte ich vertiefen (1)		habe ich erstmals verstanden (2)		sind noch unklar (3)		kann ich nicht beurteilen (4)	
	Σ	%	Σ	%	Σ	%	Σ	%
Arrays	12x	80,00	-	-	1x	6,67	2x	13,33
Bedingte Anweisungen	12x	80,00	-	-	1x	6,67	2x	13,33
Schleifen	12x	80,00	-	-	-	-	3x	20,00
Mehrfachbedingung (s...	10x	66,67	-	-	1x	6,67	4x	26,67
Methoden	10x	66,67	-	-	1x	6,67	4x	26,67

Abb. 5.2 Antworten auf die Frage "Folgende Aspekte der Programmierung ..."

Die letzten drei Fragen bezogen sich auf die Einsatzmöglichkeit des Frameworks. Den Vorschlag, das Konzept des Programmierens an einem Spiel in einer Vorlesung oder begleitend dazu einzusetzen, fand nur einer der Befragten nicht gut. Alle anderen stimmten diesem Vorschlag zu. Bei der Kategorisierung der Zielgruppe einer Spieleumgebung zum Lernen von Programmierkonzepten kommen fast alle Studierenden zu einem ähnlichen Ergebnis. Beginnend bei Jugendlichen erachten sie eine solche Umgebung für alle Altersklassen als sinnvoll, wenn die entsprechenden Personen Grundkenntnisse in der Programmierung und mit Java besitzen.

5.3 Fazit

Die Auswertung der Umfrageergebnisse hat gezeigt, dass man durch Aufgaben rund um die Spieleumgebung das Schreiben von Programmcode und das Vertiefen von Programmierkonzepten mit Spaß verbinden kann. Einzig das Erstellen von verständlichen Aufgaben stellt dabei eine kleine Hürde dar. Jedoch besteht dieses Problem sehr häufig, wenn ein Lehrender mit guten Fachkenntnissen Übungsaufgaben für Lernende mit geringeren Kenntnissen entwickelt. Nichts desto trotz erachtete der Großteil der Befragten die gesetzte Zielgruppe als sinnvoll und der Vorschlag, das entwickelte Konzept im Studium zu integrieren, wurde positiv aufgefasst. Da das Spieleframework viele Erweiterungsmöglichkeiten bietet, ist die Umsetzung dieser Idee durchaus denkbar. Beispielsweise könnte im Laufe eines Semesters mit kontinuierlichen Übungsaufgaben von den Studierenden Stück für Stück ein interessantes Spiel mit diversen Funktionen entwickelt werden.

6 Schlussbetrachtung

In dieser Arbeit wurde ein Spieleframework zur Unterstützung eines aktivierenden Lernverhaltens beim Erlernen von Programmierkonzepten in Java entwickelt. Mit Hilfe des Frameworks kann ein Rollenspiel entwickelt und gestaltet werden. Dafür beinhaltet es einer lauffähigen Spieleanwendung und Schnittstellen, um bestimmte Elemente des Spiels zu verändern. Die Entwicklung eines Spiels soll bei den Studierenden die Motivation zum Programmieren erhöhen und eine Umgebung bieten, in der sich Programmierkonzepte üben und anschaulich erklären lassen. Die Erreichung des Ziels wurde mit einer Umfrage mit Studierenden überprüft.

Nach einer kurzen Erläuterung der Idee und der Motivation dieser Arbeit in Kapitel 1 erfolgte in Kapitel 2 eine Einführung in das Themengebiet. Hierfür wurden bereits vorhandene Ansätze von Umgebungen zum Lernen von Programmierkenntnissen betrachtet. In vielen dieser Umgebungen wurde, wie auch in dieser Arbeit, eine grafische Oberfläche oder eine eigens kreierte Welt genutzt, die durch Programmcode beeinflusst und verändert werden kann.

Am Ende des zweiten Kapitels erfolgte die Abgrenzung zu den vorgestellten Ansätzen. Der grundlegende Unterschied bestand in der Idee, dass es zwar eine vorgegebene Spielewelt gibt, man diese jedoch nahezu beliebig erweitern und gestalten kann. Dadurch ist ein flexibler Einsatz des Frameworks zum Erklären und Üben verschiedener Programmierkonzepte möglich, ohne dass man sich vorher mit den grundlegenden Problemen und Schwierigkeiten der Spielentwicklung auseinander setzen muss.

Kapitel 3 hat die Zielgruppe beschrieben und grundlegende Eigenschaften von Frameworks und Lehrsprachen aufgezeigt. Aufbauend darauf wurden die Anforderungen an diese Arbeit formuliert. Der Aufbau des entwickelten Spiele-frameworks war Bestandteil von Kapitel 4. Nach der Beschreibung der Struktur des Frameworks und der Erläuterung grundlegender Entwurfsentscheidungen wurde die Umsetzung der funktionalen Anforderungen erklärt. Dazu wurden nacheinander die einzelnen Komponenten und Schnittstellen näher beschrieben.

Kapitel 5 evaluierte anhand einer Aufgabe das entwickelte Konzept. Betrachtet man die Ergebnisse der Umfrage, so erkennt man, dass das Ziel des Konzepts erreicht wurde. Das entwickelte Spieleframework kann helfend beim Üben und Festigen von Programmierkonzepten eingesetzt werden. Zudem ist die Idee, so eine Vorlesung oder ein Praktikum zu gestalten durchaus etwas, was die Motivation der Studierenden am Programmieren erhöhen kann.

6.1 Ausblick

Das Fazit der Evaluation (Kapitels 5.3) beinhaltet bereits eine Idee, wie man das Spieleframework in der Hochschullehre einsetzen kann. Darüber hinaus ist es auch denkbar das Framework bereits in der Schulzeit einzusetzen oder in Form eines Tutorials der interessierten Öffentlichkeit anzubieten. Hierfür sollten jedoch die Schnittstellen des Frameworks vereinfacht oder weiter abstrahiert werden. Beispielsweise kann das Konzept der Annotationen in Java für Programmieranfänger verwirrend sein und sollte für diese Zwecke lieber durch andere Techniken ersetzt werden. Auch der Umgang mit den Konfigurationsdateien zur Beschreibung der Karten und zu Positionierung der Game-Elemente ist nicht trivial. Hier würde sich eine eigene GUI anbieten mit deren Hilfe die Konfigurationsdateien erstellt und bearbeitet werden können.

Eine interessante Erweiterung wäre der Umbau des Frameworks auf ein Multiplayer-Spiel. Um dies zu realisieren müsste man sich zuerst Gedanken über die Synchronisierung der Objekte in den einzelnen Spieleumgebungen machen und z.B. eine Server-Client-Architektur entwerfen. Je nach Einsatzgebiet kommen dabei auch Fragen zur Skalierbarkeit und Lastenverteilung auf, wenn beispielsweise der Server eine bestimmte Menge an Clients nicht mehr verwalten kann. In vereinfachter Form wären einige Multiplayer-Erweiterungen ebenfalls möglich, indem nur ein bestimmter Teil der Spieleumgebungen synchronisiert wird. Denkbar sind hier anfangs Features wie ein Chat oder Teile der Welt in Form von Labyrinthen oder Dungeons.

Zwar könnte durch ein Multiplayer-Spiel die Motivation der Studierenden noch weiter erhöht werden, jedoch wäre eine verteilte Spieleumgebung nicht nur komplexer zu entwerfen, sondern auch schwieriger zu verstehen. Somit würde eine solche Erweiterung auch sehr viele Probleme für das Konzept mit sich bringen. Denn dann übersteigen einige verwendete Konzepte sehr stark den Wissenstand der Programmieranfänger.

7 Literaturverzeichnis

Bitkom 2015
BITKOM e.V. : *Gaming hat sich in allen Altersgruppen etabliert.* https://www.bitkom.org/Presse/Presseinformation/Gaming-hat-sich-in-allen-Altersgruppen-etabliert.html

Boles 2010
BOLES, Dietrich ; BOLES, Cornelia : *Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell*. Springer, 2010. ISBN 978-3-658-04802-0.

Boles 2009
BOLES, Dietrich : *Parallele Programmierung spielend gelernt mit dem Java-Hamster-Modell: Programmierung mit Java-Threads*. Springer, 2009. ISBN 978-3-8351-0229-3.

Boles 2013
BOLES, Dietrich : *Programmieren spielend gelernt mit dem Java-Hamster-Modell*. Springer, 2013. ISBN 978-3834806406.

Java Hamster
BOLES, Dr.-Ing. Dietrich : *Java-Hamster*. http://www.java-hamster-modell.de/index2.html Abruf: 2016-03-08

CodeCombat
CODECOMBAT INC. : *CodeCombat*. https://codecombat.com/ Abruf: 2016-02-17

Cunningham 2006
CUNNINGHAM, H Conrad ; TADEPALLI, Pallavi : *Using function generalization to design a cosequential processing framework*. In : *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*. 2006. p.213b–213b.

Cwalina 2007
CWALINA, Krzysztof ; ABRAMS, Brad : *Richtlinien für das Framework-Design: Konventionen, Ausdrücke und Muster für wiederverwendbare. NET-Bibliotheken*. Pearson Deutschland, 2007.

Henriksen 2010
HENRIKSEN, Poul ; KÖLLING, Michael ; MCCALL, Davin : *Motivating Programmers via an Online Community*. *J. Comput. Sci. Coll.* 2010. Vol. 25, no. 3, p. 82–93. – online verfügbar unter: http://dl.acm.org/citation.cfm?id=1629116.1629132 Abruf: 2016-04-27

Hilton 2006
HILTON, Robin : Gaming as an education tool. *Young Consumers*. 2006. Vol. 7, no. 2, p. 14–19.

Roberta Home
Frauenhofer IAIS : *Roberta Home*. http://roberta-home.de/de Abruf: 2016-04-12

Roberta Konzept
Frauenhofer IAIS : *Roberta Konzept*. http://roberta-home.de/de/konzept Abruf: 2016-04-12

BlueJ
KÖLLING, Michael : *BlueJ*. http://www.bluej.org/about.html Abruf: 2016-01-29

Lindenberg 2009
LINDENBERG, Stefan : *Konzeption und Realisierung eines Frameworks für mobile Rollenspiele mit Android*. HAW Hamburg. – online verfügbar unter: http://edoc.sub.uni-hamburg.de/haw/volltexte/2010/957/pdf/bachelorarbeit.pdf Abruf: 2016-05-05

Mitamura 2012
MITAMURA, Tamotsu ; SUZUKI, Yasuhiro ; OOHORI, Takahumi : *Serious games for learning programming languages*. In : *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*. 2012. p. 1812–1817.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____