Institut für Softwaretechnologie
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 47

Konzeption und Entwicklung eines aufbauenden, durchgängigen Programmierbeispiels zum Einsatz in der Lehre

Verena Käfer

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Stefan Wagner
Betreuer:	Dipl.-Ing. Jan-Peter Ostberg
begonnen am:	15. April 2013
beendet am:	15. Oktober 2013
CR-Klassifikation:	J.0

Kurzfassung

In dieser Bachelorarbeit wird der Entwurf und die Implementierung eines durchgängigen Programmierbeispiels für den Einsatz in den begleitenden Übungen der Vorlesung Programmentwicklung an der Universität Stuttgart beschrieben. Des Weiteren wird erörtert, inwiefern ein solches Beispiel die Motivation und damit die Prüfungsergebnisse der Studierenden verbessern kann.
Diese Arbeit zeigt die Durchführung des Projekts sowie das fertige Ergebnis. Bei der Erörterung ergab sich, dass ein durchgängiges Beispiel sehr wahrscheinlich die Motivation erhöhen wird, eine genaue Aussage dazu jedoch erst nach der Durchführung einer entsprechenden Studie getroffen werden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Quelltextausschnitte

1. Einleitung

Bei der Programmentwicklung (PE) handelt es sich um eine Pflichtvorlesung für alle Softwaretechnikerinnen und Softwaretechniker sowie Studierende der Wirtschaftsinformatik der Universität Stuttgart im dritten Semester. In ihr werden elementare Grundlagen gelehrt, die für das gesamte weitere Studium von Bedeutung sind. Diese sind beispielsweise UML oder Swing. Umso wichtiger ist es, dass eben diese Grundlagen verstanden und verinnerlicht werden.

Begleitend zur Programmentwicklung gibt es eine Übung, in der die wichtigsten Themengebiete praktisch angewandt und verinnerlicht werden sollen. Aufgrund meiner Tätigkeit als Tutorin in einer dieser Übungen im letzten Semester wurde mir jedoch bewusst, dass dies oft nicht der Fall ist. Die Übungen werden von den Studierenden oft nicht bis wenig vorbereitet und auch während den Übungen fehlt es an Wissen und Verständnis. Die einzelnen Übungen sind zusammenhangslos und die Beispiele sind klein und nur zu Übungszwecken konzipiert, ohne wirklichen Praxisbezug. Die Studierenden wirken unmotiviert und gelangweilt und besuchen die Übungen erst gar nicht. Als Folge dessen sind auch die Prüfungsergebnisse häufig unbefriedigend.

Insgesamt fehlt es also zur Zeit an Motivation und Lernerfolg.

Positive Erfahrungen aus der Vorlesung Softwarequalität haben gezeigt, dass ein durchgängiges Programmierbeispiel die Motivation und den Lernerfolg langfristig erhöhen kann. Es ist daher das Ziel dieser Arbeit, alle übungsrelevanten Themen in einem durchgängigen Programmierbeispiel zu vereinen.

Dies soll durch das Entwerfen und Programmieren eines einfachen Spiels geschehen. Das Spiel kann nach Abschluss der Vorlesung von den Studierenden weiter genutzt werden und hat daher einen persönlichen Nutzen. Dadurch erhöht sich die Motivation. Außerdem entsteht ein großes, zusammenhängendes Programm und man ist so motivierter, alles fertigzustellen, statt nur Einzelteile zu haben. Durch das praktische Anwenden und selbstständige Einarbeiten werden die Vorlesungsinhalte vertieft und dadurch natürlich auch das Verständnis des Lernstoffs verbessert. Als Ergebnis daraus werden sich auch die Prüfungsergebnisse verbessern.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Das Zielprogramm
Hier wird das Zielprogramm mit den notwendigen Übungsinhalten beschrieben.

Kapitel 3 – Projektablauf
Dieses Kapitel beschreibt den Ablauf dieser Bachelorarbeit.

Kapitel 4 – Entwurf
Dieses Kapitel stellt den Entwurf und meine Überlegungen bei der Planung des Programms dar.

Kapitel 5 – Programmaufbau
In diesem Kapitel wird das fertige Programm beschrieben.

Kapitel 6 – Schwierigkeiten
Hier werden die Schwierigkeiten beschrieben, die während dieser Arbeit auftraten.

Kapitel 7 – Theoretische Grundlagen
Dieses Kapitel legt die Grundlagen der Lernmotivation dar.

Kapitel 8 – Zusammenfassung und Ausblick
Die Zusammenfassung der Ausarbeitung und ein Ausblick auf weitere Möglichkeiten.

2. Das Zielprogramm

Die PE-Vorlesung gliedert sich in sechs aufeinanderfolgende Übungen, in denen jeweils der Fokus auf einem Themengebiet liegt. Das Programmierbeispiel ist so aufgebaut, dass es nach und nach ergänzt werden kann, sodass in jeder Übung genau ein Themengebiet abgearbeitet werden kann. Vor jeder neuen Übung wird eine Musterlösung für die Studierenden, die die Aufgaben nicht vollständig lösen konnten, herausgegeben.
In diesem Abschnitt möchte ich erläutern, welche Themengebiete in dem fertigen Programm untergebracht werden sollen, was sie jeweils beinhalten und inwieweit sie von einander abhängen. Des weiteren werde ich den geplanten Aufbau des fertigen Programms und die Spielregeln erläutern.

2.1. Übungsinhalte

Folgende Punkte sind Teile der Übungen und können in dieser Reihenfolge nacheinander ergänzt werden.

Entwurf und UML Der Entwurf einer Software ist ein wichtiger Bestandteil des Entwicklungsprozesses. Die Erfahrung hat gezeigt, dass den Studierenden gerade der Entwurf sehr schwer fällt. Da er auch im SoPra [1] benötigt wird, bietet es sich an, die Studierenden das Prinzip des Entwurfs anhand des Programmierbeispiels trainieren zu lassen. In der Übung kann dann über Unklarheiten oder verschiedene Vorgehensweisen diskutiert werden.
Beim Entwurf muss vor allem die externe Komponente (siehe Kapitel 4) sowie die von Swing vorgegebene Model-Delegate-Aufteilung beachtet werden. Zur Darstellung des Entwurfs werden UML-Diagramme verwendet. Diese können so mit einem realen Beispiel geübt werden. Da sich nicht alle UML-Diagramme anhand des Programmierbeispiel optimal üben lassen, wird es zusätzlich noch kleine Ergänzungsaufgaben zu einzelnen UML-Schwerpunkten der Vorlesung geben. Der Entwurf ist die Grundlage für jede Software und muss daher vor dem Programmieren erstellt werden.

Swing-Grundlagen In der Vorlesung werden die wichtigsten Grundlagen für die Oberflächenprogrammierung unter Java mit Swing gelehrt. Diese beinhalten u. A. das Erstellen von Fenstern, Menüs, Knöpfen und Dialogen. Daher sollen die Studierenden für das

[1] Das Software Praktikum ist für alle Softwaretechniker Pflicht und beinhaltet die Implementierung eines Programms in Gruppen. Weitere Informationen unter: http://www.iste.uni-stuttgart.de/se-old/lehre/softwarepraktika.html

Programm das Startfenster erstellen. Das Fenster enthält Knöpfe für den Spielstart sowie einen (noch leeren) Platzhalter für eine Highscoreliste und benötigt natürlich Layout-Manager. Des Weiteren sollen die Studierenden die Menüleiste im Hauptfenster implementieren. Hier spielen Actions und Listener eine Rolle. Zusätzlich kommt der Dialog hinzu, der am Ende eines Spiels zeigt, wer gewonnen hat und ob ein neues Spiel gestartet werden soll. Hier wird die Verwendung eigener Dialoge geübt. Dadurch werden alle für dieses Thema relevanten Inhalte abgedeckt. Die Fenster sind die Grundlage zur Verwendung des Spiels und müssen daher als allererstes implementiert werden.

Listen und Tabellen Listen und Tabellen sind ein weiterer übungsrelevanter Punkt. Diese Themen lassen sich hervorragend durch eine Highscoreliste kombinieren. Die Studierenden erstellen ein eigenes Table-Model und füllen dieses mit den Daten der Highscoreliste. Diese Highscoreliste kann nach Belieben in einer sortierbaren Tabelle angezeigt und bei Bedarf geleert werden. Die Tabelle kann an zwei verschiedenen Stellen angezeigt werden, daher ist es wichtig, das Model aktuell zu halten. Die lokale Highscoreliste ist die Vorversion zur Verwendung einer Datenbank.

XML Lesen und Schreiben von XML-Dateien gehört ebenfalls zu den Grundlagen. Daher soll es möglich sein, ein begonnenes Spiel zu laden und zu speichern. Den Studierenden ist es dabei selbst überlassen, welche API sie zum Lösen des Problems benutzen. Zusätzlich sollen die Studierenden überlegen, in welchen Situationen abgefragt werden sollte, ob das begonnene Spiel gespeichert werden soll. Beispiele wären hierbei beim Beenden des Spiel oder beim Laden eines gespeicherten Spiels. An den entsprechenden Stellen sollte dann eine entsprechende Abfrage kommen.

Datenbanken In der Vorlesung werden auch Grundlagen für die Verwendung von Datenbanken gelehrt. Die bereits oben erwähnte Highscoreliste soll daher in eine bestehende Datenbank eingegliedert werden. Beim Spielstart wird der Speicherort der Datenbank abgefragt und bei Bedarf eine neue Datenbank erstellt. Diese wird durch ein Passwort geschützt. Die Highscorelist wird während des Spiels mit der Datenbank synchron gehalten. Das Leeren der Datenbank kann nur ausgeführt werden, wenn der Benutzer das richtige Passwort eingibt und damit seine Rechte bestätigt. Diese Übung baut auf die bestehende Highscoreliste auf.

Refactoring Bei der Softwareentwicklung spielt auch das Refactoring eine Rolle. Es soll daher ein bewusst schlecht gehaltenes Stück Code von den Studierenden verbessert werden. Dazu gibt es in der Musterlösung zwei Klassen, in denen die Extract-Method-Methode angewandt werden kann.

KI Die KI der Musterlösung ist eine denkbar einfache. Die Studierenden sollen daher die Möglichkeit haben, eine schlauere KI zu entwickeln. Im Rahmen einer Vortragsübung können diese dann gegeneinander antreten. Die Vortragsübung findet getrennt von den anderen Übungen statt und behandelt keine spezifischen Themen. Die Studierenden haben die Möglichkeit, die eigene KI zu verbessern und sich mit den anderen Studierenden zu messen. Dies fördert den Spaßfaktor.

2.2. Das Spiel

Im Rahmen dieser Arbeit habe ich das Spiel „Flood it“entwickelt. Es handelt sich um ein bereits bekanntes Spielkonzept aus dem Internet, das die erforderlichen Übungsinhalte enthält bzw. um sie erweitert wurde.

2.2.1. Regeln

Die Regeln des Spiels sind recht einfach. Das Anfangsspielfeld besteht aus vielen zufällig eingefärbten Feldern. Der menschliche Spieler beginnt links unten, der Computergegner rechts oben (siehe Abbildung 2.1).

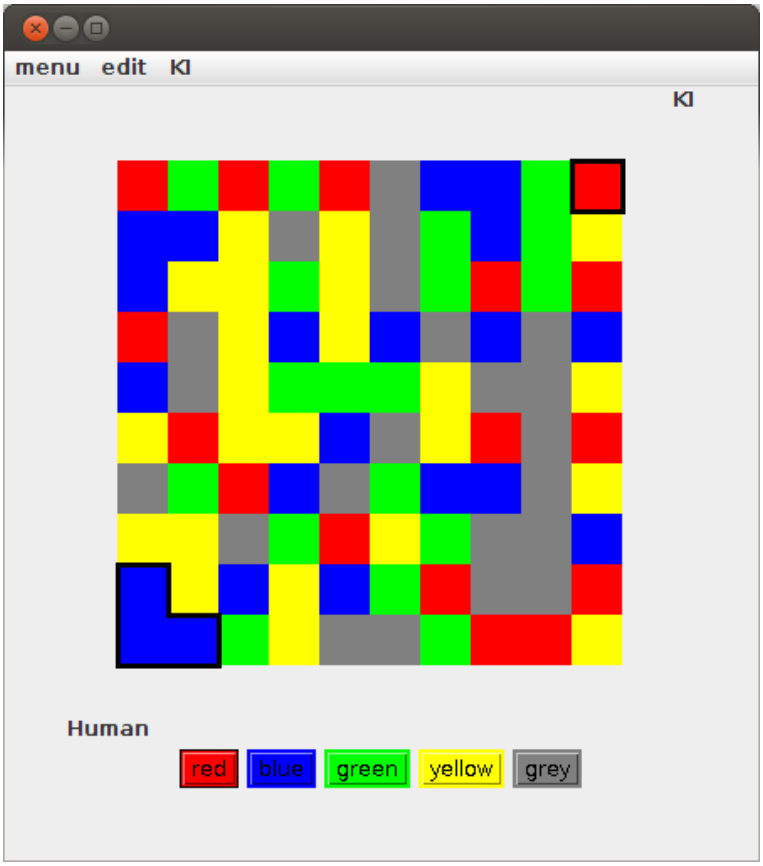

Abbildung 2.1.: Der Anfangsbildschirm

Die schwarz umrandeten Felder gleicher Farbe gehören bereits den Spielern. Zuerst ist der menschliche Spieler am Zug. Er wählt eine beliebige neue Farbe, zum Beispiel gelb. Nun färben sich die eigenen Felder um und die horizontal oder vertikal an die eigenen Felder angrenzenden Felder der gewählten Farbe gehen in seinen Besitz über (siehe Abbildung 2.2).

Es ist natürlich taktisch klug, eine Farbe zu wählen, bei der möglichst viele neue Felder in den eigenen Besitz über gehen. Sollte es keine angrenzenden Felder der gewählten Farbe geben, so ist der Gegner an der Reihe.

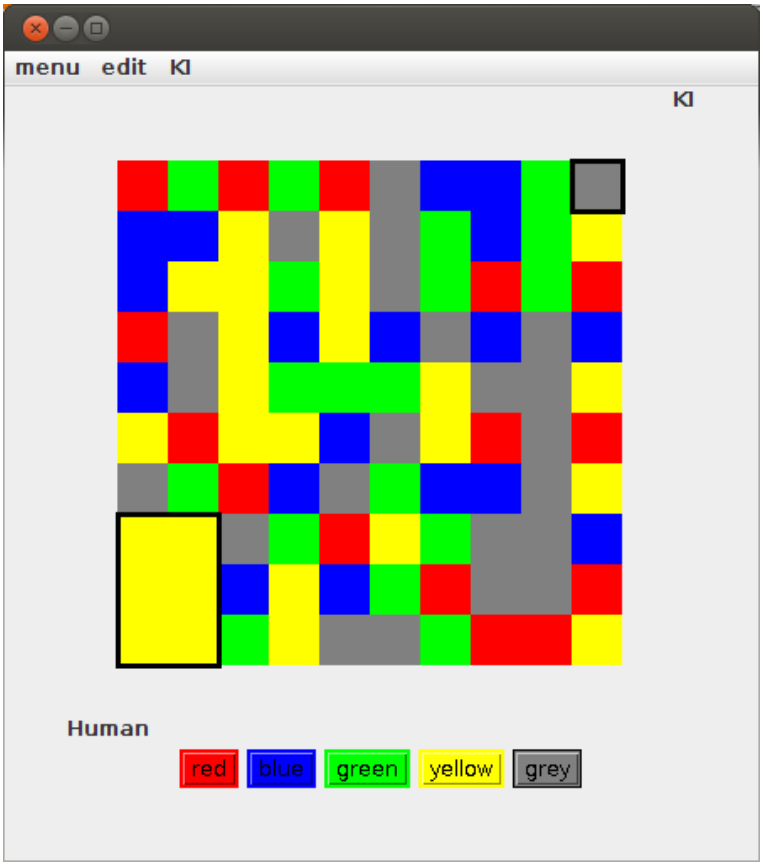

Abbildung 2.2.: Spielverlauf: Nach dem ersten Zug gehen die angrenzenden gelben Felder in den eigenen Besitz über.

Nun ist der Computergegner am Zug.
Das Spiel endet, wenn alle Felder vergeben sind. Wer mehr Felder besitzt, hat gewonnen und ein neues Spiel beginnt.

Sonderfall:
Sobald die eigenen Felder an die des Gegners angrenzen, ist es nicht mehr erlaubt, die gleiche Farbe wie der Gegner zu wählen (siehe Abbildung 2.3).

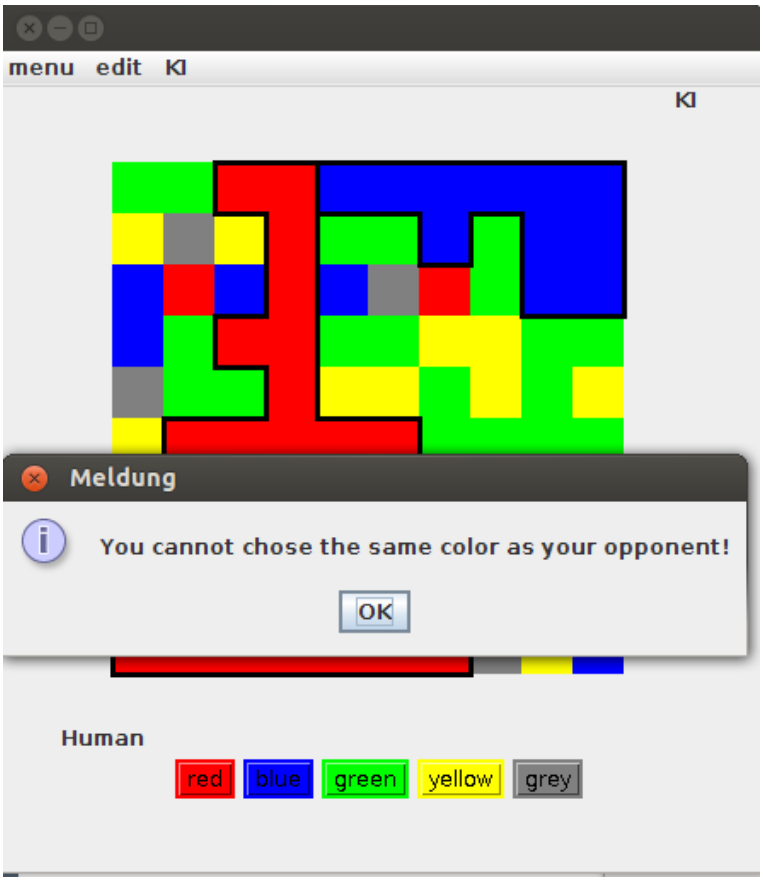

Abbildung 2.3.: Der Gegner hat zur Zeit blau gewählt. Da die eigenen Felder an die gegnerischen angrenzen, darf nun nicht auch blau gewählt werden.

3. Projektablauf

In diesem Kapitel möchte ich den Projektablauf dieser Bachelorarbeit darstellen.

3.1. Planung

Als allererstes begann ich mit der Planung meiner Arbeit. Dazu traf ich mich mit meinem Betreuer um festzulegen, welche übungsrelevanten Themen in meinem Programm vorhanden sein sollten. Dann überlegten wir, was sich als Programmierbeispiel eignen würde. Nach der Entscheidung für das Spiel flood it überlegte ich dann, wie sich die Übungsthemen am Besten in das Spiel integrieren lassen und stimmte dies mit meinem Betreuer ab.
Danach erfolgte meine persönliche Zeitplanung. Nachdem alle Anforderungen geklärt waren setzte ich meine Arbeit mit dem Entwurf fort.

3.2. Entwurf

Von Anfang an war klar, dass das Programm klar strukturiert sein muss, damit es sich in die einzelnen Musterlösungen aufteilen lässt. Dementsprechend strukturierte ich meinen Entwurf (siehe Kapitel 4). Meinem Entwurf entsprechend folgte dann die Implementierung.

3.3. Implementierung

Bei der Implementierung ging ich in der Reihenfolge der Übungen vor. Zuerst baute ich das Basisprogramm mit der Spielfelddarstellung, gefolgt von der Spiellogik und der Basis-KI. Daraufhin folgte die Oberfläche. Nach und nach fügte ich die Highscoreliste, laden und speichern, die Datenbankanbindung und die Möglichkeit eines Refactorings hinzu, ebenso die Möglichkeit zwei KIs gegeneinander spielen zu lassen. Am Schluss exportierte ich das externe Paket als .jar, und importierte es wieder, damit der Code des Pakets für die Studierenden nicht einsehbar ist (siehe Kapitel 4).

Als das Programm fertig war, begann ich, es in einzelne Projekte aufzuteilen. Jedes Projekt entspricht dabei einer Musterlösung der vorherigen Übung mit den TODOs für die nächste Übung. Für jede Musterlösung nahm ich die bis dahin nicht bekannten Teile heraus und

markierte mit TODOs an den entsprechenden Stellen, was für die nächste Übung zu implementieren ist. Zwei Beispiele hierzu finden sich im Anhang A. Beispiel A.1 zeigt dabei eine vollständig neu zu implementiereden Klasse und Beispiel A.2 eine Aufforderung zu Ergänzungen im Code.

4. Entwurf

In diesem Kapitel möchte ich erläutern, welche Entwurfsüberlegungen in das fertige Spiel eingeflossen sind.

4.1. Grundüberlegungen

Modularisierung Von Beginn an war klar, dass das Programm ein hohes Maß an Modularisierung innehaben muss. Dadurch wird die geplante Erweiterung des Programms durch die Studierenden vereinfacht.

Model-Delegate Die Verwendung von Java und Swing legt die Verwendung des Model-Delegate-Prinzips nahe, welches ich dann auch verwendet habe, um meinen Code zu strukturieren.

Externe Komponente Wie bereits in Kapitel 2.1 erklärt, soll es eine nicht einsehbare externe Komponente geben. In dieser Komponente sind die Model- sowie die Spielfeldklasse gekapselt.

Singletons Das Datenmodel sowie das Model der Highscoreliste sollten als Singleton implementiert werden.

Erweiterbarkeit Das Programm sollte nach Fertigstellung auf Wunsch weiterhin erweiterbar sein. Deshalb soll nicht darauf gebaut werden, dass es drei feste Schwierigkeitsstufen mit quadratischen Spielfeldern gibt, sondern es soll die Möglichkeit geben, individuell große Felder verwenden zu können.

4.2. Anwendungsfalldiagramm

Im Folgenden wird ein Anwendungsfalldiagramm zu „Flood it“gezeigt und erklärt. Zur besseren Übersichtlichkeit sind im Text alle Akteure und Anwendungsfälle beim ersten Vorkommen fett markiert.

Abbildung 4.1 zeigt die Hauptanwendungsfälle des Spiels. Es gibt zwei Hauptakteure, den **Mensch** und die **KI**. Diese können einige Funktionen gemeinsam nutzen und werden deshalb als Akteur **Spieler** zusammengefasst.

Der menschliche Spieler hat zu Beginn mehrere Möglichkeiten. Zum einen könnte er

das Spiel natürlich sofort wieder **beenden**. Er kann auch ein **Spiel laden**, sich die **Highscoreliste anzeigen** lassen oder diese **leeren**. Als Hauptfunktion kann er entweder ein Spiel **KI gegen KI** oder ein Spiel **Mensch gegen KI** starten. Ein Spiel Mensch gegen KI kann auf Wunsch **gespeichert** werden. Sobald ein Spiel gestartet oder geladen wurde, muss **gezogen** werden.

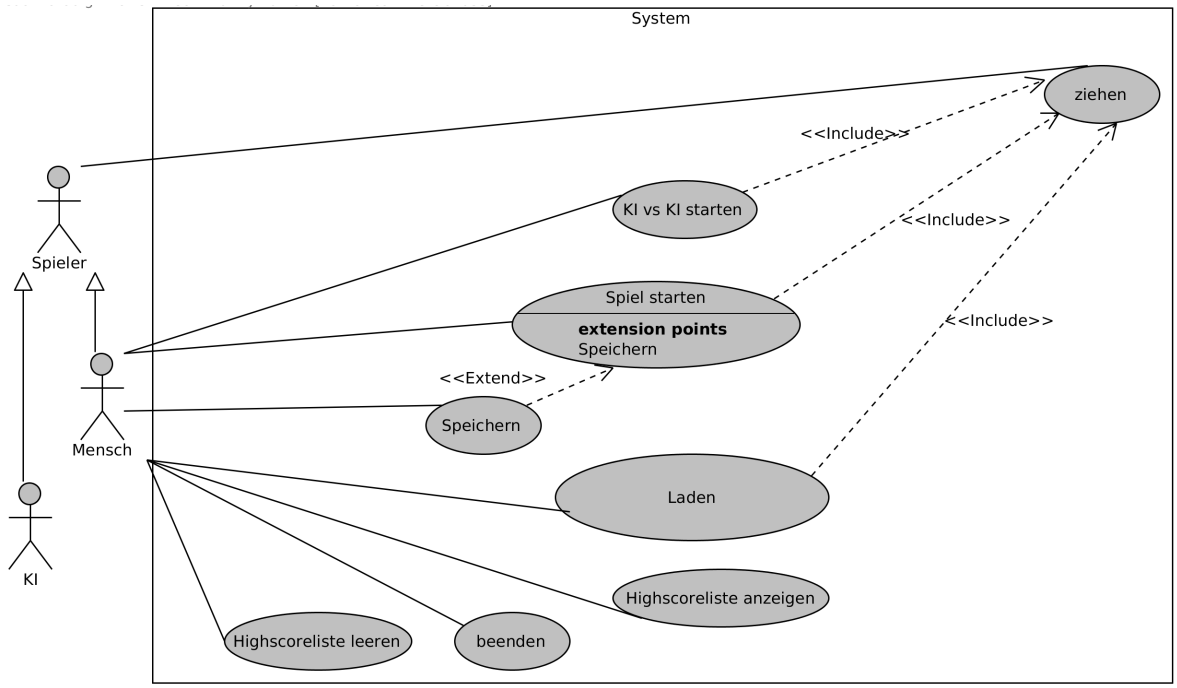

Abbildung 4.1.: Die Hauptanwendungsfälle

4.3. Komponentendiagramm

Abbildung 4.2 zeigt die Aufteilung der Pakete in vier Hauptkomponenten. Die grünen Komponenten werden den Studierenden bereits fertig übergeben, die gelben werden von ihnen erweitert.
In der IO-Komponente sind die IO-Klassen zusammengefasst. Die Komponente implementiert die Funktionalitäten des Speicherns, des Ladens sowie die Kommunikation mit einer Datenbank.
Die View-Komponente enthält alle Klassen, die für die Darstellung des Spiels notwendig sind. Die Main-Klasse zeigt den Startbildschirm sowie das Hauptfenster des Programms. Das Highscore-Pakte bietet die Möglichkeit, sich die Highscoreliste anzeigen zu lassen. Die EndOfMatch-Klasse schließlich implementiert einen Dialog, der am Ende eines Spiels anzeigt, wer gewonnen hat und abfragt, ob ein neues Spiel gestartet werden soll.
Die Controller-Komponente enthält die Spiellogik. Das Logic-Paket implementiert den Spielablauf sowie das Einhalten der Spielregeln. Das KI-Paket hält eine abstrakte KI-Klasse sowie

eine voll funktionsfähige KI. In der Configurations-Klasse letztendlich befinden sich alle im Code verwendeten Konstanten und Einstellungen.
Die externe Komponente kapselt das Model und die Darstellung des Spielfelds in einer für die Studierenden nicht einsehbaren Komponente. Diese Komponente wird an die Studierenden als Bytecode ausgegeben, zusammen mit einer Javadoc-Dokumentation[1].

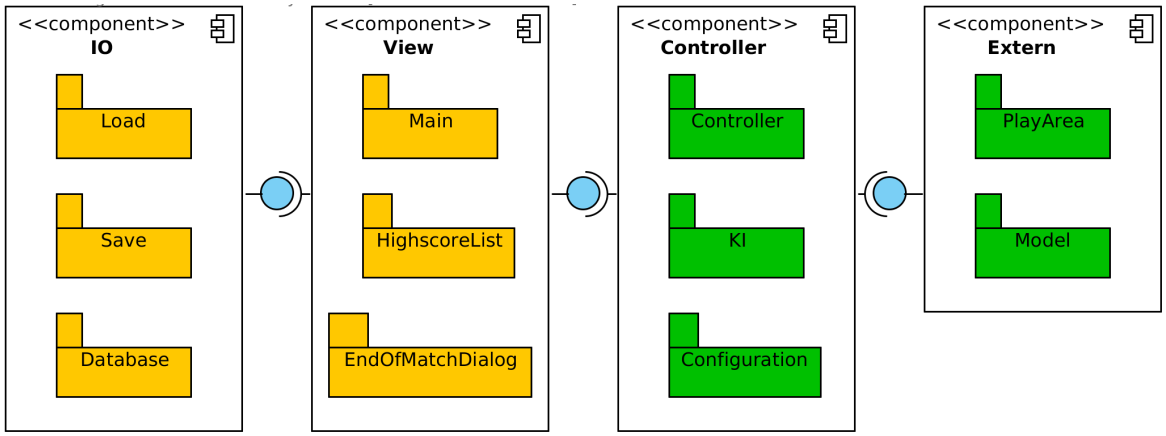

Abbildung 4.2.: Die grünen Komponenten werden fertig vorgegeben, die gelben von den Studierenden erweitert.

4.4. Datenhaltung

Das Model eines Spiels hält ein großes Array, das die Größe des Spielfelds hat. In diesem Array werden die einzelnen Spielfelder gehalten. So kann von der Position in der Darstellung unmittelbar auf das Feld im Array geschlossen werden. Jedes Feld kennt seine Farbe sowie seinen Besitzer.
Änderungen werden vom Model selbst direkt an diesem Array vorgenommen und danach wird das ganze von der Spielfelddarstellung neu gezeichnet.

4.5. Spielfelddarstellung

Das Spielfeld wird mit Hilfe von AWT gezeichnet. Für jedes Feld im Array des Models wird ein entsprechend eingefärbtes Quadrat an die korrekte Position gemalt. Bei Änderungen des Models wird das neue Array an die Spielfelddarstellung übergeben und die bisherige Darstellung übermalt.

[1] http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

4.6. Sequenzdiagramm

In diesem Abschnitt möchte ich den Ablauf der wichtigsten Funktionen des Spiels näher darstellen.

4.6.1. Ein neues Spiel wird gestartet

Was geschieht nun, wenn ein neues Spiel gestartet wird? In Abbildung 4.3 ist der Ablauf dargestellt. Der Spieler wählt im Menü aus, dass er ein neues Spiel starten möchte. Die Main-Klasse gibt dies an den Controller weiter. Dieser erzeugt nun eine neue KI. Nun wird die Schwierigkeit vom Benutzer abgefragt. Basierend auf dessen Eingabe wird ein neues Array der passenden Größe erzeugt. Dieses Array wird daraufhin mit zufällig gefärbten Feldern gefüllt. Nun wird überprüft, ob es zufälligerweise bereits in den Ecken aneinander grenzende Felder gibt, die einem der beiden Spieler gehören. Ist dies der Fall, so werden sie zum Spieler gehörend markiert. Das Array wird nun zurückgegeben und vom Controller an die Spielfelddarstellung weitergegeben. Diese zeichnet das Array und das Spiel kann beginnen.

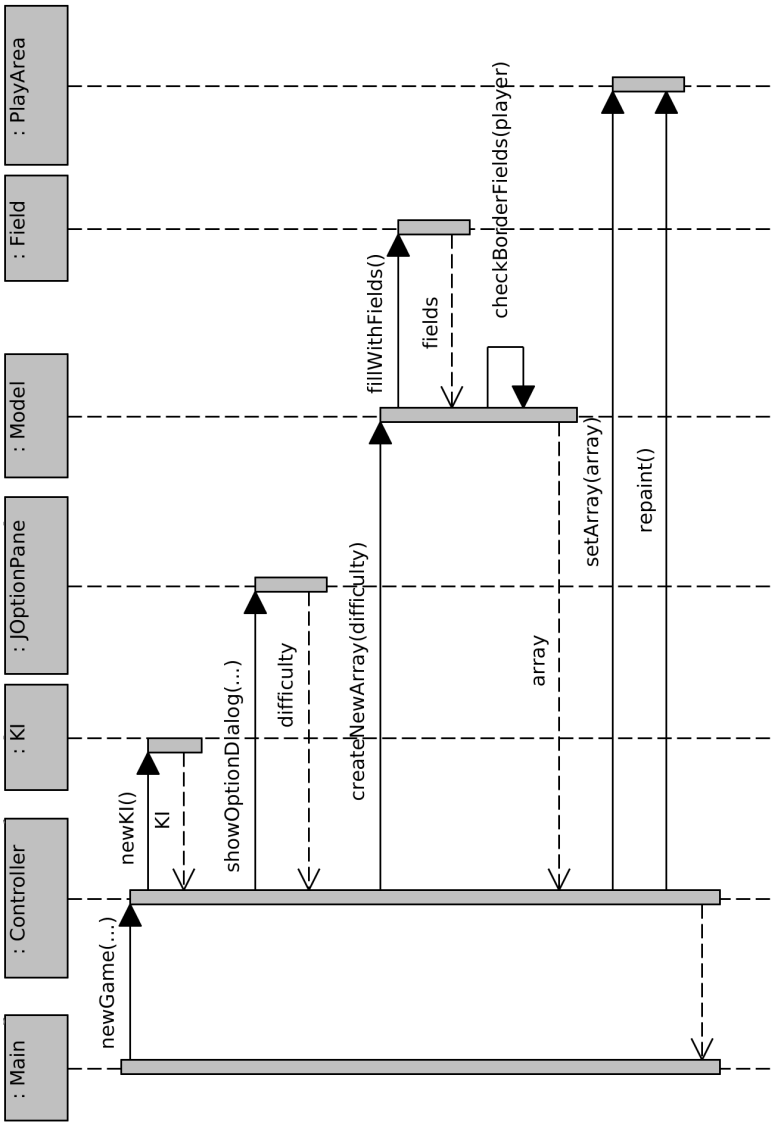

Abbildung 4.3: Ein neues Spiel wird gestartet

4.6.2. Ein Spielzugpaar

Abbildung 4.4 zeigt den Ablauf eines Spielzugpaares. Der menschliche Spieler beginnt. Er entscheidet sich für eine Farbe und drückt auf den entsprechenden Knopf. Die Farbe wird an das Model weitergegeben. Dieses überprüft nun, welche angrenzenden Felder zum Spieler gehören und markiert die Felder intern entsprechend. Daraufhin wird das Spielfeld neu gezeichnet.

Nun wird dem Controller bekannt gegeben, dass der Spielzug beendet ist. Dieser überprüft nun, ob alle Felder vergeben sind und das Spiel gegebenenfalls gewonnen wurde.
Ist dies nicht der Fall, so werden die Knöpfe für den menschlichen Spieler ausgegraut, sodass klar ist, dass die KI am Zug ist und der Spieler keine Farbe wählen kann. Die KI wird nun benachrichtigt und wählt eine Farbe. Diese Farbe wird wie oben bereits beschrieben übergeben und das Spielfeld neu gezeichnet. Der Übersichtlichkeit halber wurden diese Punkte in der Darstellung weggelassen. Nun wird wieder überprüft, ob es einen Gewinner gibt. Der Einfachheit wegen wurde hier nur der negative Fall angegeben. Gibt es keinen Gewinner, so werden die Buttons wieder aktiviert, und der menschliche Spieler ist am Zug. Solle es nach einem der beiden Spielzüge einen Gewinner geben, so wird zuerst ermittelt, mit wie vielen Punkten er gewonnen hat. Dies wird in einem Dialog dargestellt und auf Wunsch des Spielers ein Highscoreeintrag erstellt. Danach hat der Spieler die Möglichkeit, sofern er gewonnen hat, ein neues Spiel zu starten oder zum Startbildschirm zurückzukehren.

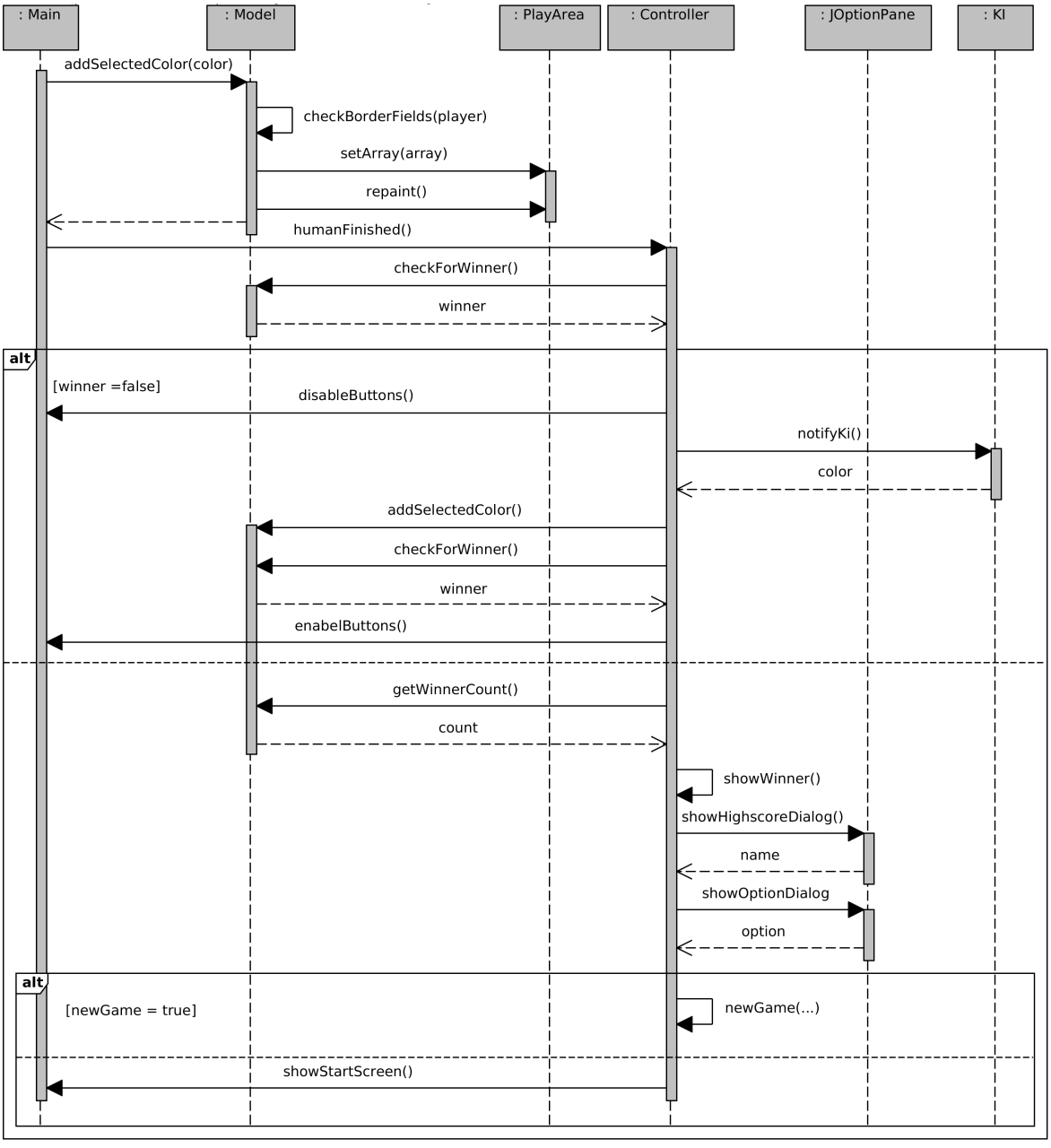

Abbildung 4.4.: Darstellung eines Spielzugpaares

5. Programmaufbau

In diesem Kapitel möchte ich die einzelnen Bestandteile des fertigen Programms darstellen.

5.1. Spielstart

Wird das Spiel gestartet, so erscheint zuerst eine Abfrage, wo die zu nutzende Datenbank liegt bzw. wo sie erstellt werden soll, siehe Abbildung 5.1.

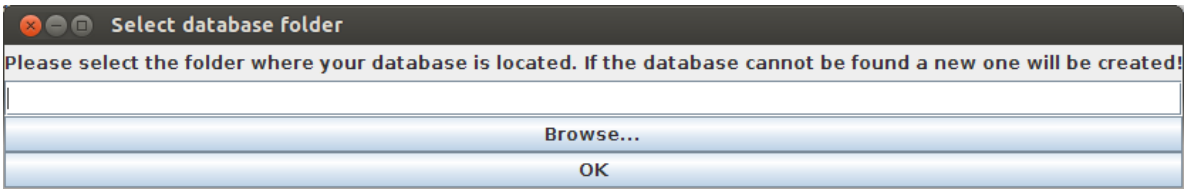

Abbildung 5.1.: Die Datenbank-Abfrage

Ist im angegebenen Ordner keine Datenbank zu finden, so wird eine neue erstellt und ein Passwort beim Benutzer abgefragt. Es ist auch möglich, kein Passwort zu setzen. Danach öffnet sich das Startfenster, wie in Abbildung 5.2 zu sehen ist. Auf der linken Seite gibt es vier Schaltflächen zur Auswahl des Spielmodus. Erstens kann ein neues Spiel gestartet werden. Zweitens kann ein neues KI gegen KI Spiel gestartet werden. Hier treten zwei KIs gegeneinander an. Drittens kann man ein bereits vorhandenes Spiel laden und viertens kann das Spiel sofort wieder beendet werden.
Auf der rechten Seite ist die aktuelle Highscoreliste zu sehen.

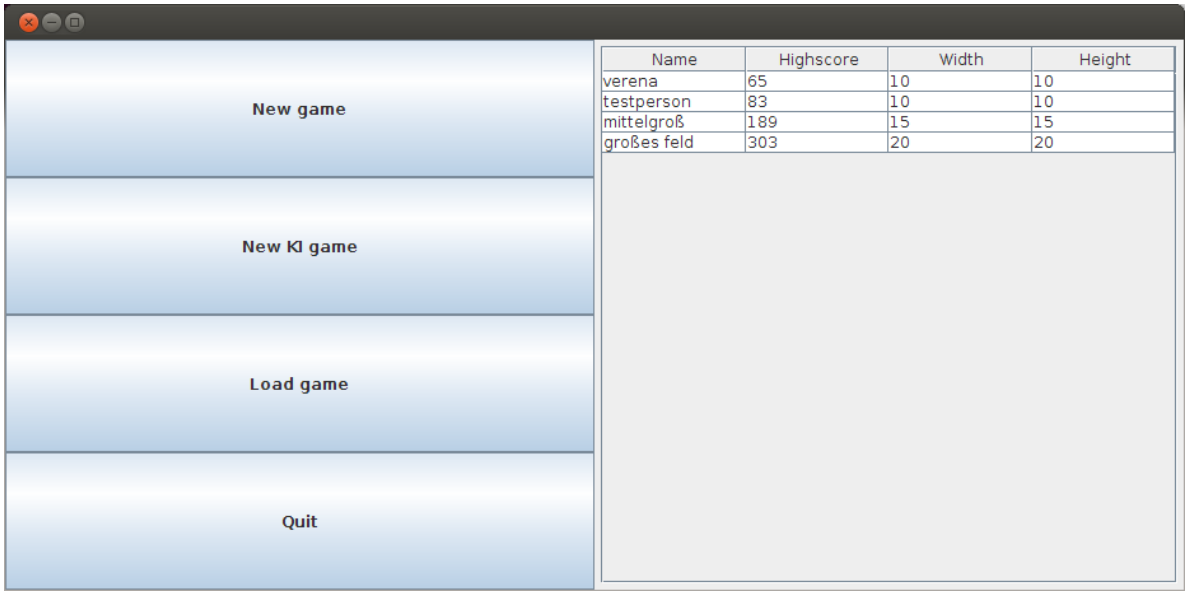

Abbildung 5.2.: Das Startfenster

Startet man nun ein neues Spiel, so muss man eine Schwierigkeitsstufe angeben, wie in Abbildung 5.3 zu sehen ist.

Abbildung 5.3.: Die verschiedenen Schwierigkeitsstufen

Daraufhin öffnet sich ein neues Fenster mit dem eigentlichen Spielfeld, siehe Abbildung 5.4.

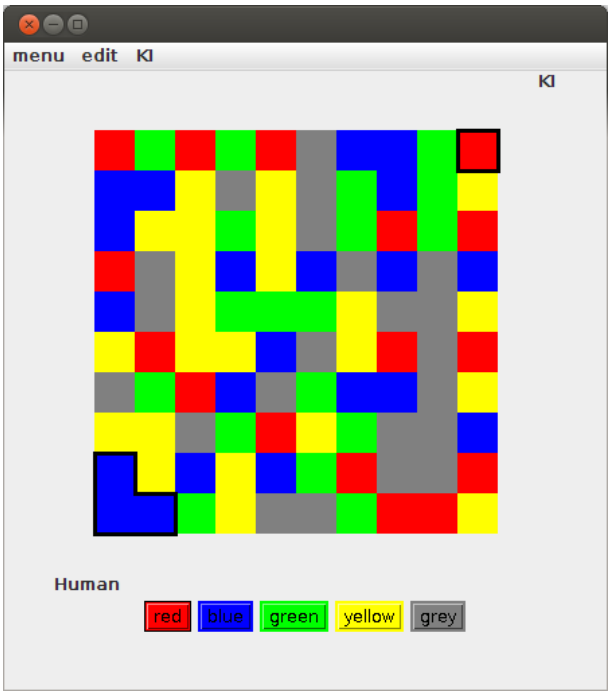

Abbildung 5.4.: Das Hauptfenster mit dem Spielfeld

Dieses aktualisiert sich nach jedem Spielzug. Sobald sich die Bereiche der Spieler berühren (siehe Abbildung 5.5), dürfen die Spieler nicht mehr die gleiche Farbe wie ihr Gegner wählen.

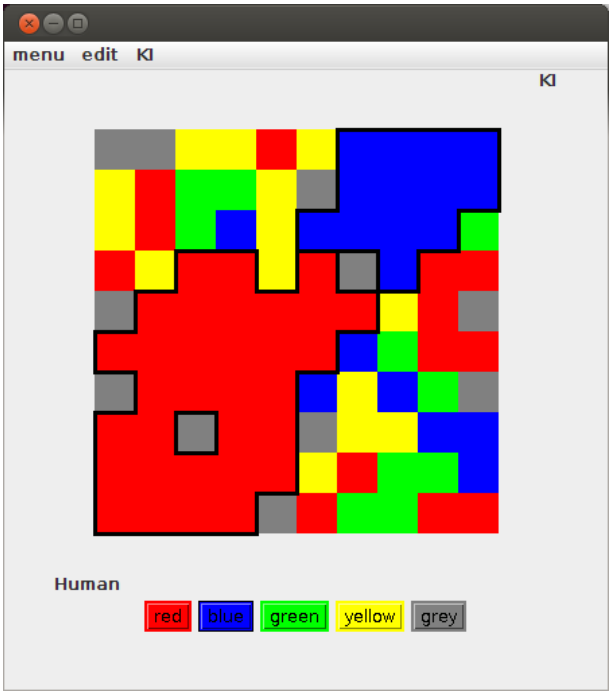

Abbildung 5.5.: Die Bereiche der Spieler berühren sich

Sollten sie dennoch die gleiche Farbe wählen, so erscheint eine Fehlermeldung wie in Abbildung 5.6 gezeigt.

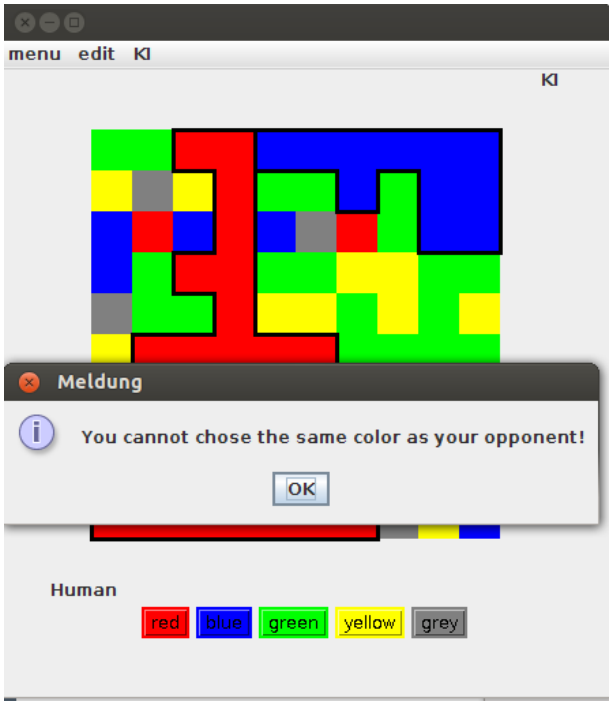

Abbildung 5.6.: Beim Wählen der gleichen Farbe erscheint eine Fehlermeldung

5.2. Laden und speichern

Es ist zu jeder Zeit möglich, das aktuelle Spiel zu speichern. Ebenso ist es jederzeit möglich, ein bereits gespeichertes Spiel zu laden. Sollte es derzeit ein noch nicht beendetes Spiel geben, so wird nachgefragt, ob der Spieler das aktuelle Spiel vorher speichern möchte.

5.3. Highscoreliste

Auf Wunsch kann man sich jederzeit die aktuelle Highscoreliste über einen Menüeintrag anzeigen lassen, wie in Abbildung 5.7 zu sehen ist.

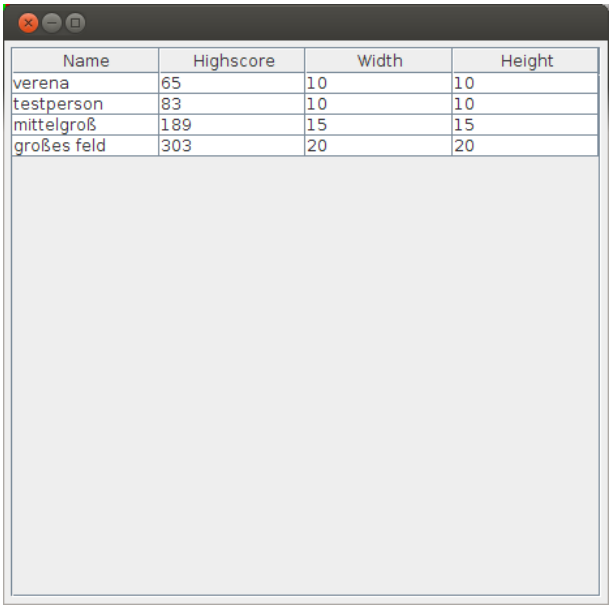

Abbildung 5.7.: Die aktuelle Highscoreliste

Auf Wunsch kann diese über das Menü auch geleert werden. Dazu ist allerdings das Passwort nötig, das beim Erstellen der Datenbank gesetzt wurde, wie in Abbildung 5.8 zu sehen ist.

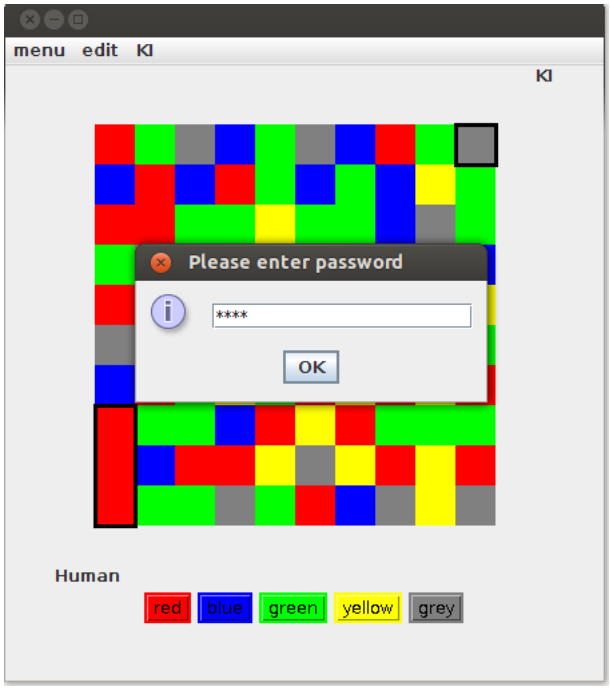

Abbildung 5.8.: Für das Löschen der Highscoreliste wird das Passwort abgefragt.

5.4. KI gegen KI

Für die geplante Vortragsübung können zwei KIs gegeneinander antreten. Dies kann auch über das Menü gestartet werden. Am Ende wird der Gewinner angezeigt. (siehe Abbildung 5.9)

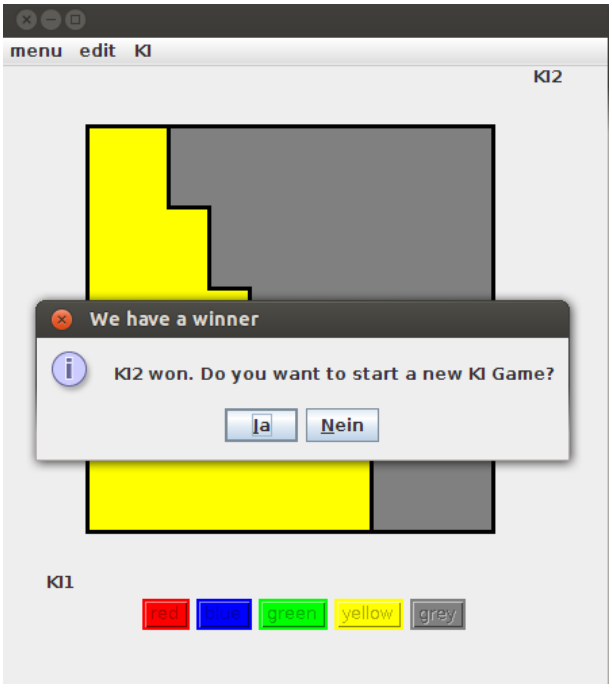

Abbildung 5.9.: KI1 hat gewonnen

6. Schwierigkeiten

In diesem Kapitel möchte ich die Schwierigkeiten beim Erstellen dieser Arbeit aufzeigen.

6.1. AWT

Die größte Herausforderung war die Einarbeitung in AWT, da dies im bisherigen Studium nur als Alternative zu Swing erwähnt, aber nicht vertieft worden war. Nach kurzer Recherche war mir jedoch klar, dass sich das Spielfeld wie ich es mir vorstellte nicht mit Swing zu realisieren war, da man mit Swing nicht zeichnen kann. Ich entschied mich, das Spielfeld mit AWT zu realisieren, da AWT einen großen Teil der GUI-Design-Möglichkeiten unter Java ausmacht und ich für eine Übung für Programmieranfänger keine komplizierten Funktionen von Drittanbietern einbauen wollte.
So musste ich mich erst einmal in AWT einarbeiten und einen Prototyp für die Spielfelddarstellung bauen um herauszufinden, wie sich das Spielfeld am Besten darstellen lässt. Daraufhin war es kein Problem mehr, das Spielfeld zu implementieren. Danach musste ich nur noch den AWT-Teil mit dem Swing-Teil verknüpfen, da die Studierenden als Grundlage Swing lernen und der Hauptteil des Programms daher mit Swing implementiert werden sollte.

6.2. Bug im Speicherdialog

AWT bietet einen Dialog an, der beim Benutzer direkt abfragt, ob eine vorhandene Datei beim Speichern überschreiben werden soll. Wie ich schnell herausfand, wirft dieser Dialog jedoch unter Linux manchmal eine Exception, weil er mit der angebotenen Liste der zuletzt verwendeten Dateien unter Linux nicht zurecht kommt. Nach kurzer Recherche fand ich heraus, dass der Bug bereits bekannt ist[1], jedoch erst in der nächsten Java-Version behoben wird. Deswegen blieb mir nicht anderes übrig, als als Workaround den Standard-SWT-Dialog zu verwenden und die entsprechende Abfrage selbst zu implementieren.

[1] http://bugs.sun.com/view_bug.do?bug_id=7130662

6.3. Synchron halten der einzelnen Musterlösungen

Als das Programm meiner Meinung nach fertig ausgearbeitet war, teilte ich es in einzelne Projekte auf. Jedes Projekt entspricht einer Musterlösung zur vorhergehenden Übung, in der auch schon die Anforderungen an die nächste Übung notiert sind. In jeder Musterlösung entfernte ich daher den bis dahin nicht bekannten Code und fügte entsprechende TODOs ein, um klar zu machen, was für die jeweilige nächste Übung implementiert werden sollte. Trotzdem waren im Nachhinein immer wieder kleine Änderungen nötig (siehe 6.4), die dann natürlich in allen Projekten geändert werden mussten. Das war mit einem erhöhten Arbeitsaufwand verbunden.

6.4. Erfassung aller nötigen Anforderungen

Es war nicht ganz einfach, alle nötigen Anforderungen and das Programmierbeispiel am Anfang zu erfassen. Die Hauptziele waren von Anfang an klar. Beim Diskutieren über die Übungen und Erstellen der Übungsblätter kristallisierten sich jedoch einige Punkte erst gegen Ende heraus, beispielsweise die Verwendung einer Abstract-Action oder das Anlegen eines Administrators für die Datenbank. Dies bedingte dann auch Abschnitt 6.3.

7. Theoretische Grundlagen

In diesem Kapitel lege ich dar, warum ich davon überzeugt bin, dass ein durchgängiges Programmierbeispiel die Motivation und den Lerneffekt erhöhen wird. Dazu möchte ich zunächst auf die Grundlagen der Motivation eingehen.

7.1. Motivation

Was ist Motivation? Und wie kann man sie steigern? Im Allgemeinen unterscheidet man zwei Arten der Motivation: intrinsische Motivation und extrinsische Motivation.
„Intrinsisch motivierte Verhaltensweisen können als interessensbestimmte Handlungen definiert werden“[DR93, S. 255]. Die Motivation kommt von einem selbst, von innen heraus. Interesse und innere Werte spielen eine große Rolle. Wenn man sich stark für ein Thema interessiert, dann ist man auch motiviert, sich näher damit zu befassen.

Im Gegensatz dazu steht die extrinsische Motivation. „Extrinsisch motivierte Verhaltensweisen treten in der Regel nicht spontan auf; sie werden vielmehr durch Aufforderungen in Gang gesetzt, deren Befolgung eine (positive) Bekräftigung erwarten lässt, oder die auf andere Weise instrumentelle Funktion besitzen“[DR93, S. 255]. Diese Motivationsart kommt also von außen. Belohnung und Strafe spielt hier mit hinein. Man ist motivierter, sein Zimmer aufzuräumen, wenn man weiß, dass man danach ein Eis bekommt. Ebenso ist man motivierter, die Regeln einzuhalten, wenn man bei Nichtbeachten der Regeln bestraft wird. Jedem wird klar sein, dass die Aussicht auf Belohnung ein deutlich besseres Gefühl hinterlässt, als die Aussicht auf Strafe.

7.1.1. Quellen der intrinsischen und extrinsischen Motivation

Woher kommt nun intrinsische oder extrinsische Motivation? John Barbuto [Bj98] unterscheidet fünf Quellen der Motivation, zwei intrinsische und drei extrinsische.

Diese Quellen sind:

Intrinsisch:

Interne Prozessmotivation (intrinsic process motivation) Bei dieser Quelle geht es darum, dass jemand eine Aufgabe um ihrer selbst Willen bewältigt. Einfach weil es ihm Spaß macht. Vorteile oder Belohnungen spielen keine Rolle. Beispiele hierfür wären zum Beispiel ein Musiker oder ein Künstler. Sie musizieren oder malen einfach, weil sie Spaß daran haben, nicht um damit etwas zu erreichen. Die Tätigkeit selbst ist Motivation genug.

Internes Selbstverständnis (internal self-concept-based motivation) Menschen mit dieser Motivationsquelle orientieren sich an (unbewussten) internen Standards und Maßstäben. Das eigene Handeln dient dazu, diesen Standards gerecht zu werden, sie zu erfüllen.

Extrinsisch:

Instrumentelle Motivation (instrumental motivation) Bei dieser Motivation führt eine Tätigkeit umgehend zu einem sichtbaren, positiven Ergebnis, beispielsweise zu Bezahlung oder einer Beförderung. Die Tätigkeit ist das Mittel, um etwas anderes zu erreichen.

Externes Selbstverständnis (external self concept-based motivation) Im Gegensatz zum internen Selbstverständnis geht es hier um Idealvorstellungen des Umfeldes. Man will dazugehören, einen Status erlangen, und orientiert sich daher stark an dem, was gesellschaftlich akzeptiert ist und erwartet wird.

Internalisierung von Zielen (goal internalization motivation) So motivierte Menschen machen sich die Ziele ihrer Umwelt zu ihren eigenen, beispielsweise die ihres Unternehmens. Sie strengen sich an, weil sie der Überzeugung sind, dass ihre eigene Leistung dazu beiträgt, diese Ziele zu erreichen.

7.1.2. Motivation im Studium

Wie lässt sich das Ganze jetzt auf das Studium übertragen? Wären die Studierenden nur intrinsisch motiviert, so hieße dass, dass sie für jedes Modul aus purem Interesse heraus lernen würden. Aus eigener Erfahrung weiß ich jedoch, dass man nicht an jedem Modul gleich interessiert ist und sich das Interesse für das eine oder andere Pflichtmodul eher in Grenzen hält. Müsste man es nicht belegen, so würde man dies auch nicht tun.
Wären die Studierenden nur extrinsich motiviert, hieße das wiederum, dass sie nur wegen Aussicht auf Belohnung (das bestandene Studium) oder auf Bestrafung (Nichtbestehen einer Prüfung und eventueller Studiumsabbruch) auf Prüfungen lernen würden. Da sich die allermeisten Studierenden freiwillig für ihr Studium entschieden haben, ist allerdings sichergestellt, dass zumindest ein gewisses Grundinteresse an der Thematik vorhanden ist

und die Motivation nicht nur extrinsischer Natur sein kann.

Manfred Prenzel [Pre96] hat nun ein Modell erarbeitet, bei dem intrinsische und extrinsische Motivation nicht strikt polarisiert sind. Dieses ist in Abbildung 7.1 zu sehen.

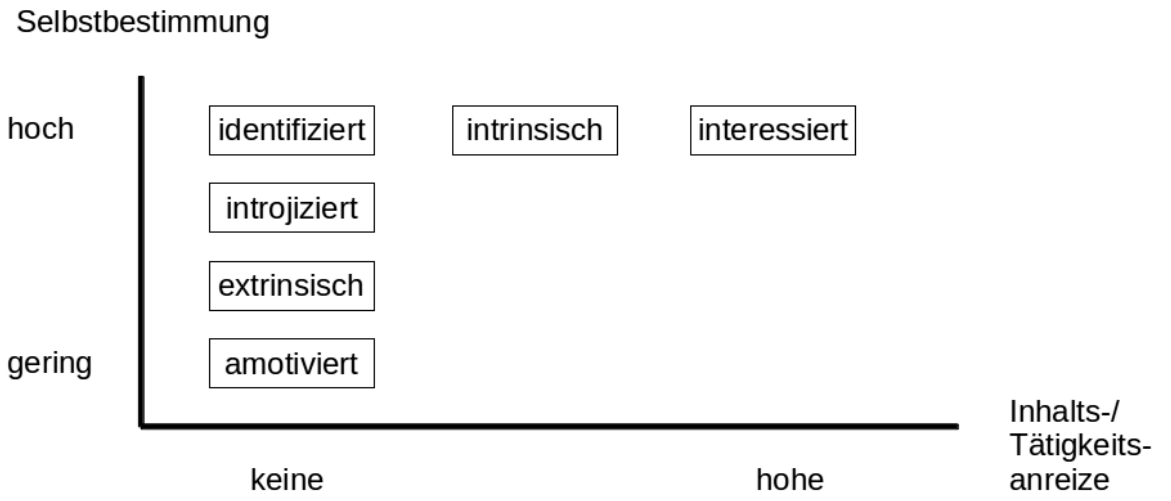

Abbildung 7.1.: Das Model nach Prenzel [Pre96]

Amotiviert Es ist keine Lernmotivation vorhanden. Die Person sieht keine Möglichkeit, die Situation unter Kontrolle zu bringen.

Extrinsisch Die Person lernt nur, um Belohnungen zu erhalten oder Bestrafung zu vermeiden. Ohne diese würde sie nicht lernen. Das Lernen erfolgt nur durch Druck und ist fremdbestimmt.

Introjiziert Die Person lernt, weil sie sonst ein schlechte Gewissen hat. Unmittelbare Sanktionen müssen nicht vorliegen. Sie handelt nicht mehr vollkommen fremdbestimmt aber keineswegs selbstbestimmt.

Identifiziert Die Person lernt, weil das Lernen nötig ist, um eigene Ziele zu erreichen. Der Inhalt oder die Tätigkeit an sich ist wenig reizvoll. Dennoch ist das Lernen selbst motiviert.

Intrinsisch Das Lernen ist selbst-motiviert und wird nicht durch äußeren Druck bestimmt. Das Lernen an sich ist der Anreiz.

Interessiert Die Person ist bereit, sich übermäßig stark mit einem Thema auseinanderzusetzen. Sie ist fasziniert und so gepackt, dass sie es aus freien Stücken weiter erschließen möchte.

Das Modell geht davon aus, dass die Motivation steigt, je höher die Selbstbestimmung ist. Hier bewegen wir uns in den Varianten identifiziert, introjiziert, extrinsisch und amotiviert. Die Motivation kann also steigen, obwohl keine oder gringe inhaltliche Reize gegeben sind. Des Weiteren steigt die Motivation, wenn sie von innen kommt (intrinsisch) oder stark interessensgeprägt ist (interessiert).

Prenzel beschreibt nun, dass es sich günstig auf den kognitiven Prozess beim Lernen auswirkt, wenn man selbstbestimmt lernen kann, also wenn man identifiziert, intrinsisch oder interessiert motiviert ist. Dazu kommt, dass fremdbestimmtes Lernen von Angst und Unlust begleitet ist, während selbstbestimmtes Lernen von positiven Gefühlen begleitet ist.

7.1.3. Bedingungen für selbstbestimmtes Lernen

Folglich lernt also jeder am besten, wenn er es freiwillig tut und sich für das Thema interessiert.
Leider ist das im Studium nicht immer der Fall. Einerseits ist das Lernen im Studium keineswegs selbstbestimmt. Viele Übungen sind Pflicht und in der Prüfungsordnung steht geschrieben, welche Prüfungen man ablegen muss. Natürlich studiert man freiwillig, dennoch handelt es sich in unbeliebten Modulen wohl eher um extrinsiche oder introjizierte Motivation und nicht wie gewünscht um indentifizierte Motivation.
Auch ist das Interesse für einzelne Module nicht immer vorhanden, da es sich um Pflichtmodule handelt. Hierbei wäre intrinsische oder interessierte Motivation wünschenswert, was aber selten der Fall ist.
Rein theoretisch müsste man sich für die perfekte Motivation also alle Module aussuchen können und sich für alle übermäßig interessieren. Doch das ist wohl kaum möglich.

7.1.4. Übertragung auf die Übung

Wie lässt sich nun die Motivation der Studierenden in der PE-Übung erhöhen? Es gibt bereits einen gute Basis. Bei der PE handelt es sich zwar um eine Plichtvorlesung, der Besuch der Übungen ist jedoch freiwillig. Die Selbstbestimmung ist damit bereits höher als in anderen Modulen. Leider fehlt den meisten Studierenden dennoch das Interesse am Thema und die relativ strikte Vorgabe von Aufgaben, die zu lösen sind, verhindert eine größere Selbstbestimmung.
Das fehlende Interesse am Thema ist wohl auch der Grund, warum einige Studierenden die Übungen überhaupt nicht besuchen. Sie sind nur extrinsisch motiviert etwas für die Vorlesung zu tun, weil sie sonst die Prüfung nicht bestehen.
Gibt es noch andere Gründe, warum die Studierenden wenig motiviert sind?

7.1.5. Demotivation

Demotivation

Dazu möchte ich auf die sechs Möglichkeiten zur Demotivation nach Prenzel eingehen [Edu10] :

Keine Autonomie Eine zu fest vorgegebene Lösungsstruktur wirkt demotivierend. Es ist besser, die Lernenden alleine auf individuelle Lösungen kommen zu lassen, anstatt sie zu sehr in eine bestimmte Richtung zu drängen.

Keine Ziele Es ist wichtig, von Anfang an die Ziele einer Lektion darzulegen. Sind diese den Lernenden nicht klar, so wird das Ziel oft nicht erreicht.

Falsches Niveau Die Lehre muss zu dem Niveau der Lernenden passen. Es darf nicht zu niedrig sein, da das demotivierend wirkt, jedoch auch nicht zu hoch.

Fehlendes Zutrauen Es wirkt demotivierend, wenn der Lehrende kein Vertrauen in die Kompetenz der Lernenden hat oder überrascht wirkt, wenn eine Aufgabe richtig gelöst wurde.

Keine soziale Einbindung Sowohl der Lehrende als auch die Lernenden gehören zu einer Gemeinschaft. Es ist demotivierend, wenn man das Gefühl hat, nicht gebraucht zu werden oder nicht dazu zu gehören.

Desinteresse des Lehrenden Es wirkt auf die Lernenden demotivierend, wenn der Lehrende selbst den Stoff als langweilig oder unnütz ansieht. „Die Motivation des Lehrenden kennzeichnet die maximale Motivation der Lernenden.“ [Edu10]

7.1.6. Das durchgängige Beispiel

In den vorhergehenden Abschnitten habe ich gezeigt, dass es mehrere Einflüsse auf die Motivation bzw. Demotivation der Studierenden gibt. Inwiefern hilft nun ein durchgängiges Programmierbeispiel, die Motivation der Studierenden zu erhöhen und die Demotivation zu verringern?
Zuerst möchte ich auf die sechs Ausprägungen von Lernmotivation zurück kommen. Wie bereits weiter oben erwähnt, sind die Übungen bereits freiwillig und die Selbstbestimmung daher bereits teilweise gegeben. Diese wird durch das durchgängige Beispiel noch erhöht. Die Studierenden können selbst entscheiden, wie sie die Aufgaben lösen möchten. Es gibt zwar eine Musterlösung, diese stellt aber nur eine von vielen richtigen Möglichkeiten dar. Zusätzlich wird durch das Programmieren eines Spiels auch das Interesse erhöht. Ein Spiel mach Spaß und man kann es nachher weiterverwenden. Dadurch ist die Motivation deutlich größer, weil der Anreiz ein ganz anderer ist. Außerdem möchte man nachher das komplette Spiel fertig haben und bleibt deshalb am Ball, im Gegensatz zu unabhängigen kleinen Übungsaufgaben.

Dazu kommt, dass die Studierenden weniger demotiviert sind. Sie können autonomer arbeiten, weil es eben keine einzig richtige Lösung gibt. Sie können völlig frei arbeiten, um das Aufgabenziel zu erreichen. Die Ziele der Übungen sind auch von Anfang an klar. Das Gesamtziel, nämlich das fertige Spiel, wird bereits in der ersten Übung dargelegt und auch die Teilziele pro Übung werden auf dem Aufgabenblatt klar dargelegt. Dies war allerdings bereits vorher gegeben. Das Niveau entspricht auch dem Niveau der Studierenden. Einzelne Teilaufgaben steigern das Niveau innerhalb der Aufgabenblätter. Da das Beispiel von vornherein darauf ausgelegt war, genau die Übungsthemen abzudecken, wird genau der übungsrelevant neue Stoff geprüft. Die Tutoren müssen Vertrauen in die Kompetenz der Studierenden haben. Dies ist ein Punkt, an dem jeder Tutor für sich arbeiten muss. Dass die Studierenden freie Wahl haben, wie sie die Aufgaben lösen, zeigt jedoch bereits ein Grundvertrauen in ihre Fähigkeiten. Dazu kommt, dass die Tutoren und die Studierenden eine Gemeinschaft bilden. Oft werden die Aufgaben in Gruppen gelöst, dadurch arbeiten die Studierenden zusammen und alle werden gebraucht. Die Tutoren können dabei Denkanstöße geben, aber nicht mehr. Zu guter Letzt erhöht ein durchgängiges Beispiel auch die Motivation der Tutoren. In meinem Fall, weil ich das Spiel selbst programmiert habe und allgemein, weil man sich besser hineinversetzten kann.
Insgesamt lässt sich also sagen, dass sich ein durchgängiges Programmierbeispiel ziemlich sicher positiv auf die Motivation der Studierenden auswirken wird.

8. Zusammenfassung und Ausblick

Was lässt sich nun zusammenfassend sagen? Die Arbeit ist im Großen und Ganzen glatt verlaufen. Es ist mir gelungen, alle übungsrelevanten Themen in einem durchgängigen Beispiel unterzubringen. Dieses Beispiel kann damit ab dem Wintersemester 2013/14 für die Programmentwicklung eingesetzt werden. Der Motivationstheorie zufolge wird sich ziemlich sicher die Motivation der Studierenden erhöhen und damit sollten sich auch die Prüfungsergebnisse verbessern. Das Ziel dieser Arbeit ist damit erreicht.

Ausblick

Wie das Wort Theorie schon besagt, gibt es noch keine praktischen Ergebnisse, die diese Arbeit belegen. Es gibt für durchgängige Beispiele bereits positive Erfahrungen aus anderen Modulen und damit ist die Chance groß, dass es wieder funktionieren wird. Um das gewünschte Ergebnis zu verifizieren, sollte die nächsten Semester eine Studie durchgeführt werden, die die Einstellungen der Studierenden gegenüber dem Programmierbeispiel erfasst sowie die messbaren Prüfungsleistungen.

A. Anhang

A.1. TODO-Beispiele

```
package io.database;

/**
 * This class holds the methods for communicating with the database
 *
 * TODO This class should also provide the possibility to add a new entry to the
 * database.
 *
 * TODO It must also provide the possibility to clear the whole list.
 *
 * TODO Last don't forget to initialize the database the first time you use it.
 * It would also be nice if the user could decide where the database is stored
 * and which database should be used
 *
 * TODO when the database is created the first time the user must set a password
 * which should be stored in an extra table or for an admin user (part c)
 *
 * TODO to clear the high score list this password must be provided
 *
 * TODO there should be an admin user who can clear the high score list
 *
 * @author Verena Kaefer
 *
 */
public class Database {

}
```

Quellcodeausschnitt A.1: Beispiel für einen Klassenkopf mit TODO-Einträgen

```
highscoreItem.addActionListener(new ActionListener() {

	@Override
	public void actionPerformed(ActionEvent argo) {
		// TODO a new frame which shows the current high score list. It
		// should be sortable
	}

});

clearItem.addActionListener(new ActionListener() {

	@Override
	public void actionPerformed(ActionEvent argo) {
		// TODO clear highscore list
	}

});
```

Quellcodeausschnitt A.2: Beispiel für TODO-Einträge im bestehenden Code

Literaturverzeichnis

[Bj98] J. Barbuto, jr. Motivation sources inventory: development and validation of new scales to measure an integrative taxonomy of motivation. *Physological Reports*, 82:1011–1022, 1998. (Zitiert auf Seite 37)

[DR93] E. Deci, R. Ryan. Die Selbstbestimmungstheorie der Motivation und ihre Bedeutung für die Pädagogik. *Zeitschrift für Pädagogik*, 39:223–238, 1993. (Zitiert auf Seite 37)

[Edu10] EduShift. Sechs Möglichkeiten, Lernende zu demotivieren. online, 2010. URL `http://www.edushift.de/2010/08/19/sechs-moglichkeiten-lernende-zu-demotivieren/`. (Zitiert auf Seite 41)

[Pre96] M. Prenzel. *Lehr- und Lernprobleme im Studium*, Kapitel Bedingungen für selbstbestimmt motiviertes und interessiertes Lernen im Studium, S. 11–22. Verlag Hans Huber, 1996. (Zitiert auf den Seiten 7 und 39)

Alle URLs wurden zuletzt am 10. 10. 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

(Verena Käfer)