Implementierungsbericht: RetroMachines, RetroFactory

Luca Becker, Henrike Hardt,
Larissa Schmid, Adrian Schulte,
Maik Wiesner

1. Dezember 2014

Inhaltsverzeichnis

1 Einleitung

2 Änderungen am Entwurf

Falls nicht anders angegeben handelt es sich bei den Methoden um public Methoden.

2.1 Package: data

2.1.1 AssetManager

Hinzugefügte Methoden

Getter und Setter Um auf die Spielelemente gezielt zugreifen zu können, haben wir uns entschieden, diese im AssetManager zu verwalten. Mit Übergabe der jeweiligen TiledMap werden daraufhin die Spielelemente ausgelesen und mithilfe der get-Methoden zurückgegeben.

2.2 Package: models

2.2.1 Model

Hinzugefügte Methoden

destroy Löscht den Eintrag, der aktuell durch das Model repräsentiert wird, durch das Model aus der Datenbank.

Modifizierte Methoden

getStatement - *protected* Erstellt ein Statement, das zur Ausführung von Abfragen auf der Datenbank ausgeführt werden kann.

2.2.2 Profile

Hinzugefügte Methoden

getAllProfiles Erstellt ein String-Array, welches alle Profile enthält, die sich aktuell in der Datenbank befinden.

getProfileNameIdMap Erstellt eine HashMap, welche als Abbildung von einem Profilnamen auf dessen Identifikationsnummer dient.

setStatistic Ändert die Statistik des Profils zur übergebenen Instanz und speichert diese in der Datenbank

2.2.3 Setting

Hinzugefügte Methoden

Konstruktor Es wurde ein Konstruktor eingefügt, der das Erstellen eines Setting-Objektes mit Standardwerten erlaubt.

2.2.4 Statistic

Hinzugefügte Methoden

Konstruktor Es wurde ein Konstruktor eingefügt, der das Erstellen eines Statistic-Objektes mit Standardwerten erlaubt.

2.3 Package: game

2.3.1 RetroLevel und LevelBuilder

Aufgabe der Klasse

RetroLevel gehört zum Model-Teil des Spiels und enthält alle wichtigen Informationen zum jeweiligen Level. Es wird mithilfe von der statischen, inneren Klasse LevelBuilder erstellt, die auch alle anderen Informationen wie die Map, die das jeweilige Level enthält, weitergibt. RetroLevel selber kann nicht direkt instanziiert werden durch einen privaten Konstruktor, um Inkonsistenzen zu vermeiden.

Methoden von RetroLevel

isValidGameElementPosition vergleicht, ob an der Stelle Platz ist, um ein GameElement abzulegen

placeGameElement platziert ein GameElement

removeGameElement entfernt ein GameElement

getTiles gibt ein Array von Rechtecken um die angegebene Position zurück, aus denen man ablesen kann, ob dort freier Platz ist oder etwas steht auf der Map

allDepotsFilled gibt zurück, ob alle Ablagen gefüllt sind

getGameElement gibt ein GameElement an der übergebenen Position zurück

getMap gibt die Map zurück, die das Level hält

getLambdaUtil gibt das LambdaUtil, das das Level hält, zurück

2.4 Package: game.controller

2.4.1 GameController

Hinzugefügte Methoden

update Führt einen Schritt der Spielwelt aus.

Entfernte Methoden

Evaluation Aufgrund unseres Ziels des modularen Designs und der leichten Austauschbarkeit von Klassen bzw. Abläufen haben wir alle Methoden, die die Evaluation direkt betreffen, ausgelagert in die Lambda-Datenstruktur, welche nun also auch die Auswertung des Lambda-Terms vornimmt. Dies betrifft folgende Methoden:

evaluationClicked

enterEvaluationScreen

checkPlacementOfElements

buildLambdaTree

evaluate

checkEvaluationResult

betaReduction

alphaConversion

updateEvaluationScreen

2.4.2 EvaluationController

Anstelle der Methoden im GameController steht nun der EvaluationController.

Methoden

enterEvaluation Zeigt dem Benutzer den Evaluationsbildschirm und holt sich den LambdaTree vom Level.

startEvaluation startet die Evaluation

2.4.3 ProfileController

Hinzugefügte Methoden

deleteProfile

2.4.4 Lambda-Datenstruktur

Im ursprünglichen Entwurf bestand der Baum, der den Lambda-Term darstellt, aus Instanzen der Klasse *Vertex*. Diese hatten als Attribut den jeweiligen Typ (Applikation, Abstraktion, Variable). Wir haben uns entschieden, das zu ändern und stattdessen Polymorphie und dynamische Bindung zu verwenden. Die Klasse *Vertex* ist nun abstrakt und hat die Subklassen *Application*, *Abstraction*, *Variable* und *Dummy*, welche für die Elemente des Lambda-Terms stehen. Dabei sind Instanzen von *Dummy* Platzhalter (leere Knoten), mit denen der LevelTree zu Beginn initialisiert wird. Falls das JSON des Levels nun zwar an sich eine gültige JSON-Syntax hat, diese aber nicht mehr als Lambda-Term korrekt ist, wird nun eine InvalidJsonException (auch in util.lambda) verwendet, um diesen Fall angemessen zu behandeln.

3 Muss- und Wunschkriterien

3.1 Nicht umgesetzte Wunschkriterien

Auslieferung in mehreren Sprachen

Challengemode mit Zeitdruck

Werkstatt zum Umbauen von Maschinen

Highscore-Tabelle

Mehrere Spielmodi

Begleitende Story

Steuerung durch Wischgesten

3.2 Umgesetzte Wunschkriterien

Mehrere Spielcharaktere

Erweiternde Spielelemente (Einstellung des Steuerungsmodus: Rechts- oder Linkshänder)

Pixelgrafik / Retrolook / reduzierter Farbraum

4 Zeitlicher Ablauf

4.1 Adrian

4.2 Henrike

4.3 Larissa

4.4 Luca

21.01. - 26.01. Implementierung der Pakete *data* und *data.model*.

28.01. - 30.01. Fehlerbehebung

30.01. - 01.02. Implementierung des Pakets *game.controllers*

04.02. - 08.02. Implementierung der Klasse *GameScreen* und Fehlerbehebung des Pakets *game.controllers*

08.02. - 13.02. Fehlerbehebung und Implementierungsbericht

4.5 Maik

5 jUnit-Tests

5.1 Package: models

5.1.1 GlobalVariablesTest

Testcase zum Testen der GlobalVariables-Klasse

testCorrectValue Testet, ob die GlobalVariables-Klasse den richtigen default-Wert zurückgibt.

testAssignValue Testet, ob die GlobalVariables-Klasse einen Wert unter einem Key speichert.

5.1.2 ProfileTest

Testcase zum Testen der Profile-Klasse

testCreateProfile Testet, ob ein Profile erstellt werden kann und diesem eine Statistic- sowie Settings-Instanz zugewiesen werden kann.

testDBFetch Testet, ob das Test-Profil aus der Datenbank geladen werden kann und dessen Werte stimmen.

testOverrideDB Testet, ob ein Profil bearbeitet werden kann und diese Änderungen zurück in die Datenbank geschrieben werden.

testGetAllProfiles Testet, ob die statische Methode „getAllProfiles“ das richtige Profile zurückgibt und ob sich die richtige Anzahl an Profilen in dem Array befindet.

testGetAllProfilesWithCreate Testet, ob die statische Methode „getAllProfiles“ auch auf das Erstellen eines neuen Profiles reagiert.

testGetHashMap Testet, ob die statische Methode „getProfileNameIdMap“ die richtige Anzahl an Einträgen und den richtigen Eintrag beinhaltet.

5.1.3 SettingTest

Testcase zum Testen der Setting-Klasse.

testCreateSetting Testet, ob eine Settings-Instanz erstellt werden kann und die richtigen Werte zugewiesen wurden.

testDBFetch Testet, ob eine Settings-Instanz basierend auf den Demoinformationen der Datenbank erstellt werden kann und über die richtigen Werte verfügt.

testWriteBack Testet, ob eine Settings-Instanz in der Lage ist, geänderte Informationen zurück in die Datenbank zu schreiben.

5.1.4 StatisticTest

testCreateStatistic Testet, ob eine Statistics-Instanz erstellt werden kann und die richtigen Werte zugewiesen wurden.

testDBFetch Testet, ob eine Statistics-Instanz basierend auf den Demoinformationen der Datenbank erstellt werden kann und über die richtigen Werte verfügt.

testWriteBack Testet, ob eine Statistics-Instanz in der Lage ist, geänderte Informationen zurück in die Datenbank zu schreiben.

5.2 Package: controllers

5.2.1 GameControllerTest

5.2.2 ProfileControllerTest

Testcase zum Testen der ProfileController-Klasse.

testLoadLastProfile Testet, ob der ProfileController in der Lage ist, das zuletzt aktive Profile zu laden.

testRightLoading Testet, ob der Controller das richtige Profil über die „loadLastProfile“-Methode geladen hat. Testet außerdem, ob das Laden scheitern kann.

testDoubleUserName Testet, ob der Controller das Erstellen eines Profils mit bereits existierenden Namen erlaubt.

testMaximumProfiles Testet, ob der Controller die Obergrenze an Profilen einhält.

testCreateDeleteProfile Testet, ob der Controller ein Profil erstellt und die Vernichtung dieses Profils im Anschluss vollständig ist.

testChangeOnProfileCreate Testet, ob das aktive Profil des Controllers im Anschluss an die Erstellung auf das neu erstellte Profil gewechselt wird.

testChangeProfile Testet, ob der Controller den Wechsel des aktiven Profils dieses in der Datenbank vermerkt.

testListener Testet, ob der Controller die Klassen, die sich bei ihm angemeldet haben, korrekt benachrichtigt. Dazu benutzt es seinen eigenen MockListener.

5.2.3 SettingController

testListener Testet, ob ein MockListener durch den Controller über Änderungen an den Einstellungen benachrichtigt wird.

5.3 Package: gameelements

5.3.1 RetroManTest

testJumpLock Testet, ob ein erneuter Sprung möglich ist, nachdem RetroMan bereits zuvor gesprungen ist.

testFacing Testet, ob die Figur nach links beziehungsweise nach rechts guckt.

testLeftMovement Testet, ob die Figur korrekt nach links beschleunigt wird.

testRightMovement Testet, ob die Figur korrekt nach rechts beschleunigt wird.

testJumpLockOnFalling Testet, ob die Figur nicht springen kann, wenn sie fällt.