Karlsruher Institut für Technologie

Lamb.da - Das Spiel

Implementierungsbericht

Farid El-Haddad, Florian Fervers, Kai Fieger,
Robert Hochweiß, Kay Schmitteckert

4. März 2015

Inhaltsverzeichnis

1 Einleitung

Die Applikation „Lamb.da“ soll Kindern im Grundschulalter auf eine spielerische Art und Weise die wesentlichen Aspekte des untypisierten Lambda-Kalküls und damit auch die Grundlage der funktionalen Programmierung vermitteln.

In diesem Bericht ist der Verlauf der Implementierung unserer Applikation und die in dieser Phase entstandenen Änderungen zum ursprünglichen Entwurf festgehalten. Als erst wird vorgestellt welche der Muss- und Wunschkriterien umgesetzt beziehungsweise nicht implementiert wurden. Danach folgt eine Einsicht in die Änderungen am Programmcode im Vergleich zum Entwurf. Es werden dort hauptsächlich neue und auch wieder entfernte Methoden und Klassen behandelt. Ebenfalls werden im Bericht andere Dinge, wie die für das Programm geschriebenen Testfälle und Exceptions, behandelt.

2 Umsetzung von Muss- und Wunschkriterien

2.1 Musskritierien

- folgende Musskriterien wurden implementiert
  - Bedienen über ein Smartphone per Toucheingabe
  - Auflösen von Lamm-Konstellationen(Analogie zu λ-Termen)
  - Bestimmen eines Endresultats bei einer bereits gegebenen, vollständigen Anordnung von Lämmern
  - Vervollständigen einer Anordnung von Lämmern, um ein gegebenes Endresultat zu erreichen
  - Erstellen, Konfigurieren und Löschen von mehreren Spielerprofilen
  - Verfolgen des Lernfortschritts durch Eltern oder Lehrer durch eine Statistik
  - Aufrechterhaltung der Langzeitmotivation des Spielers
  - Interaktive und intuitive Einführung zur Erklärung des Spiels und seiner Modi

2.2 Umgesetzte Wunschkriterien

- folgende Wunschkriterien Kriterien wurden implementiert
  - Für Tablet-Computer angepasste Version
  - Spiel als Desktop-Anwendung
    - Windows-Anwendung (für Windows 7 und Windows 8)
    - Mac OS X-Anwendung (ab Mac OS X 10.9)
  - Anzeigen von Hinweisen zur Lösung des Level-Ziels

- Sandbox, in der eigene Level erstellt und simuliert werden können
- Verfolgen des Lernfortschritts durch Eltern oder Lehrer durch ein Achievementliste
- Belohnungen für das erstmalige Lösen der Level
- Eintauschen von Belohnungen gegen höherwertige Belohnungen in einem In-Game-Shop
- Vermittlung einer kindergerechten Hintergrundgeschichte mittels Animationen
- Englisch als unterstütze Sprache
- Französisch als unterstütze Sprache
- Optionale Ausführung von unterschiedlichen Reduktionsstrategien bei der β-Reduktion von Lamm-Konstellationen (Normalreihenfolge ist die Standard-Reduktionsstrategie)
  - Applicative-Order
  - Call-By-Name
  - Call-By-Value

2.3 Nicht umgesetzte Wunschkriterien

- folgende Wunschkriterien haben wir aus Zeitmangel fallen lassen :
  - Option für Farbenblinde, durch die Farbenblinde Hilfestellungen beim Spiel bekommen
  - Lehreroption, durch die die richtigen λ-Terme beim Abspielen der Level angezeigt werden

3 Änderungen gegenüber dem Feinentwurf

3.1 package lambda

3.1.1 public class **LambdaGame** extends Game

Neue Methoden

- **getController**()
  Gibt die ViewController-Instanz des Spiel zu einer übergebenen Klasse zurück.

- **setScreen**()
  Setzt den aktuell angezeigten Bildschirm.

- **addViewController**()
  Fügt dem Spiel einen neuen ViewController hinzu.

- **createViewControllers**()
  Nachdem alle Assets geladen sind initialisiert ein Anruf dieser Methode alle ViewController des Spiels.

Entfernte Methoden (Ersetzt durch **getController**())

- **getAchievementMenuVC**()

- **getDropDownMenuVC**()

- **getStatisticVC**()

- **getMainMenuVC**()

- **getSettingsVC**()

- **getShopVC**()

- **getShopItemVC**()

- **getProfileSelectionVC**()

- **getLangEditVC**()

- **getNameEditVC**()
- **getAvatarEditVC**()
- **getLevelSelectionVC**()
- **getEditorVC**()
- **getReductionVC**()

3.1.2 public class **Observable**<Observer>

Allgemein

Da Android Java 8 nicht unterstützt wurden Lambda Expressions von Java durch anonyme innere Klassen ersetzt, was hauptsächlich das Observable betraf. Dies geschah mit Hilfe eigener Consumer und Supplier Interfaces. (ebenfalls in `package lambda` zu finden.)

3.1.3 public interface **Consumer**<T>

Allgemein

Neues Interface, das dem Consumer Interface von Java 8 entspricht. Da Android kein Java 8 unterstützt wurde Consumer selber Implementiert um auf die Lambda Expressions des Entwurfs verzichten zu können.

Methoden

- **accept**()
  Entsprechend des Consumer Interfaces von Java 8.

3.1.4 public interface **Supplier**<T>

Allgemein

Neues Interface, das dem Supplier Interface von Java 8 entspricht. Da Android kein Java 8 unterstützt wurde Supplier selber Implementiert um auf die Lambda Expressions des Entwurfs verzichten zu können.

Methoden

- **`get()`**
  Entsprechend des Supplier Interfaces von Java 8.

3.2 package `lambda.viewcontroller`

Notizen

- Controller in AssetViewController umbenannt.
- AssetViewController jetzt in lambda.viewcontroller.assets zu finden.

3.2.1 `public class` **`AudioManager`** `implements ProfileManagerObserver, SettingsModelObserver`

Allgemein

Neue Klasse zum Verwalten bzw. Abspielen der Geräusche und Musik im Spiel. Wurde eingeführt, um die Tonausgabe zu vereinfachen.

Methoden

- **`queueAssets()`**
  Wird aufgerufen, um die Sound-Assets und die Standardmusik im Spiel zu laden.
- **`init()`**
  Initialisiert den AudioManager nachdem **queueAssets**() aufgerufen wurde und die Assets fertig geladen sind.
- **`setLoggedIn()`**
  Wechselt den AudioManager in eingeloggten bzw. ausgeloggten Zustand. (Unterschiedliche Tonausgabe in beiden Bereichen)
- **`playSound()`**
  Wird benutzt um ein Geräusch (z.B. Button-Klick) abzuspielen.
- **`playDefaultMusic()`**
  Spielt die Standardmusik des Spiels in einer Schleife ab.

- **`playMusic`**`()`
  Spielt gegebene Musik in einer Schleife ab.

3.2.2 `public abstract class` **`ViewController`** `implements Screen, ProfileManagerObserver`

Allgemein

(Umbenennung) Ursprünglich die Controller-Klasse des Entwurfs. Implementiert jetzt das ProfileManagerObserver-Interface, da dies sonst selbst von fast allen Unterklassen implementiert wird.

Neue Methoden

- **`queueAssets`**`()`
  Wurde eingeführt, damit beim Ladevorgang jeder Unterklasse von **ViewController** ihre benötigten Assets laden kann. Die Methode übergibt dabei benötigte Assets in die Warteschlange des AssetManagers.

- **`create`**`()`
  Wird nach dem Ladevorgang aufgerufen, um die ViewController, mit den jetzt geladenen Assets, zu initialisieren.

3.2.3 `public abstract class` **`StageViewController`** `extends ViewController`

Allgemein

Neue Klasse. Spezialisierter **ViewController**, welcher einen eigenen Bildschirm im Spiel repräsentiert. **ViewController**, die einen Bildschirm darstellen, haben viele häufig identische Methoden. Der **StageViewController** gibt den vom **ViewController** geerbten Methoden eine solche Standardimplementierung, wodurch Redundanz verhindert wird.

Neue Methoden

- **`getStage`**`()`
  Liefert die Stage, die den kompletten Bildschirm darstellt (auf ihr sind alle GUI-Elemente verankert) zurück.

- **`getLastViewController`**`()`
  Liefert eine Referenz auf den ViewController zurück, der gezeigt werden soll, wenn die Zurücktaste von Android gedrückt wird.

- **`setLastViewController`**`()`
  Setzt den ViewController, der gezeigt werden soll, wenn die Zurücktaste von Android gedrückt wird.

3.3 package lambda.viewcontroller.assets

3.3.1 `public class` **`AssetViewController`** `extends StageViewController`

Neue Methoden

- **`getManager`**`()`
  Gibt den AssetManger zurück, der vom Spiel verwendet wird, um alle Assets zu laden.

Entfernte Methoden

- **`getLoadingImage`**`()`
  Wurde nicht benötigt.

- **`loadProgressChanged`**`()`
  Wurde nicht benötigt.

3.4 package lambda.viewcontroller.editor

3.4.1 `public class` **`EditorViewController`** `extends StageViewController`

TODO

3.4.2 `public class` **`HelpDialog`** `extends Dialog`

Allgemein

Neue Klasse. Stellt einen Hilfsdialog für den EditorViewController dar, der die Funktion dessen Buttons erklärt. Wurde zur Übersichtlichkeit des Codes erstellt.

3.4.3 public class **HintDialog** extends Dialog

Allgemein

Neue Klasse. Stellt den Hinweisdialog für den EditorViewController dar, der ein Hinweis für das Level anzeigt. Wurde zur Übersichtlichkeit des Codes erstellt.

3.4.4 public class **TargetDialog** extends Dialog

Allgemein

Neue Klasse. Stellt den Zieldialog für den EditorViewController dar, der das Levelziel anzeigt. Wurde zur Übersichtlichkeit des Codes erstellt.

3.5 package lambda.viewcontroller.level

Allgemein

In den Klassen **VariableUIContext**(), **AbstractionUIContext**() und **ParanthesisUIContext**() wurden die angegebenen Sprites durch TextureRegions ersetzt, um eine bessere Ansteuerung der einzelnen Teilelemente wie z.B. Front-, Mittel- und Hinterteil und den dazugehörigen Masken des Lammes zu gewährleisten. Des Weiteren werden die Animationen nun von der Klasse **LevelContext**() im Package package lambda.model.level gehalten, da diese immer wieder und für jedes Element-Item benötigt werden und man sie somit nicht in jeder einzelnen Familie halten muss.

3.6 package lambda.viewcontroller.shop

Allgemein

Die Kontrolle über die Status der Klasse **DropDownMenuViewController**, ob diese "geöffnetöder "geschlossenïm Shop angezeigt wird, wurde der Klasse **ShopViewController** überschrieben, da diese für die allgemeine Anzeige im Shop zuständig ist und die Buttons für die **DropDownMenuViewController** hält

3.6.1 public class **ShopItemViewController**<T extends ShopItemModel> extends Actor implements ShopItemModelObserver

Allgemein

Die Klasse wurde zu einer generischen Klasse umgeschrieben, da es so einfacher und übersichtlicher ist für jeden Item-Typ einen ViewController zu erstellen. Des Weiteren wurde die Klasse um einen TextButton erweitert, der je nach Status des Items angepasst und im Shop angezeigt wird.

Hinzugefügte Methoden

- public void **setCurrentState**()
  Diese Methode wird aufgerufen, wenn sich der Status eines Items ändert und passt demnach den TextButton dieses Items an, damit visuell der neue Status im Shop signalisiert wird.

3.7 package lambda.viewcontroller.reduction

3.7.1 public class **ReductionViewController** extends StageViewController

TODO

3.7.2 public class **HelpDialog** extends Dialog

Allgemein

Neue Klasse. Stellt einen Hilfsdialog für den ReductionViewController dar, der die Funktion dessen Buttons erklärt. Wurde zur Übersichtlichkeit des Codes erstellt.

3.8 package lambda.model.level

Allgemein

Diesem Package wurden zwei Loader hinzugefügt, welche der von libGDX bereitgestellte AssetManager benötigt, um die LevelModels, sowie die DifficultySettings zu laden und zu halten. Es handelt sich dabei um die Klassen **LevelModelLoader**(), sowie um **DifficultySettingsLoader**() und benötigen keiner weiteren Beschreibung.

3.8.1 `public class` **`LevelManager`**

Allgemein

Neue Klasse, um alle benötigten Information für die Levels zu managen und zu setzen. Sie wurde der Übersichtlichkeit wegen eingeführt und weiterhin um die Komplexität der Klasse LevelModel zu eliminieren. Die Klasse ist als Singleton implementiert, um schnellen und globalen Zugriff auf Levelinformationen zu garantieren.

Methoden

- **`LevelManager`**`()`
  Instanziiert ein neues und gleichzeitig das einzige Objekt dieser Klasse und wird von `public LevelManager` **`getLevelManager`**`()` aufgerufen.

- `public LevelModel` **`getLevelModel`**`(int id)`
  Gibt das LevelModel mit der übergebenen Id zurück.

- `public DifficultySetting` **`getDifficulty`**`(int id)`
  Gibt die DifficultySetting mit der übergebenen Id zurück.

3.8.2 `public class` **`LevelContext`**

Allgemein

Diese Klasse hält nun auch noch die Animationen, die vorher in den jeweiligen ElementUIContexts - also den Spielelementen - gehalten worden wären. Grund dafür ist, dass die Animationen für jede Element-Familie gleich sind und somit nicht in jeder Familie gehalten werden müssen.

3.8.3 `public class` **`LevelLoadHelper`**

Allgemein

Diese Klasse wurde vom Package `package lambda.utils` in dieses Package verschoben, da die verschiedenen Helfer-Klassen nun direkt bei den dazugehörenden Models liegen.

3.9 package lambda.model.profiles

3.9.1 public interface **ProfileModelObserver**

Entfernte Methoden

- **changedAvatar**()
  Wurde nicht benötigt.

3.9.2 public class **ProfileModel** extends Observable<ProfileModelObserver>

Hinzugefügte Konstruktoren

- public **ProfileModel**(String newName, ProfileModel oldProfile)
  Erstellt ein neues Profil mit dem gegebenen Namen und den Werten des Alten.

3.10 package lambda.model.shop

Allgemein

In diesem Package wurde lediglich die Klassen **SpriteModel**(), ersetzt durch die Klasse **ElementUIContextFamily**(), die sich vorher im Package package lambda.viewcontroller.level befand. Grund dafür ist, dass es sich hierbei sowohl um die Spielelemente im Editormodus handelt als auch um ein Item, welches im Shop erwerbbar ist. Somit wurde entschieden diese Klasse zusammen mit den anderen Items in diesem Package zu belassen.

Entfernte Klassen

SpriteModel()

Hinzugefügte Klassen

ElementUIContextFamily()

3.10.1 `public class` **`ShopModel`**

Allgemein

Die Klasse wurde als Singleton implementiert, um einen einfachen und globalen Zugriff auf die Items zu garantieren.

Hinzugefügte Methoden

- `public ShopModel` **`getShopModel`**`()`
  Instanziiert ein neues und gleichzeitig das einzige Objekt (falls noch keines existiert) dieser Klasse und gibt dieses zurück.

- `public void` **`loadAllMusictems`**`(AssetManager assets)`
  Erstellt alle Musik-Items und setzt diese in die Liste der dazugehörigen Kategorie.

- `public void` **`loadAllImagetems`**`(AssetManager assets)`
  Erstellt alle Hintergrund-Items und setzt diese in die Liste der dazugehörigen Kategorie.

- `public void` **`loadAllElementtems`**`(AssetManager assets)`
  Erstellt alle Element-Items und setzt diese in die Liste der dazugehörigen Kategorie.

- `public void` **`setAllAssets`**`(AssetManager assets)`
  Setzt alle Assets in die richtigen Items, sowie auch alle Default-Items, welche aktiviert sind, wenn noch kein Item gekauft wurde.

4 Eigene Exceptions

4.1 `InvalidProfilesException`

Eine **InvalidProfilesException** wird geworfen, wenn die Datei, die das Profil darstellt, korrekt gelesen werden kann, aber im größeren Zusammenhang Fehler auftreten. Zum Beispiel: 2 Profile werden geladen, die beide den gleichen Namen haben. Dies ist nicht erlaubt und löst eine **InvalidProfilesException** aus.

4.2 `InvalidJsonException`

Eine **InvalidProfilesException** wird geworfen, falls eine Json Datei, wie ein Level, gelesen werden kann, aber inhaltlich falsch ist.

4.3 `InvalidLambdaTermException`

Eine **InvalidLambdaTermException** wird geworfen, falls es ein Problem gibt während eine Operation auf einem LambdaTerm ausgeführt wird, das dadurch bedingt ist, dass dieser LambdaTerm ungültig ist.

5 Änderungen gegenüber der Datenstruktur

6 Levelspezifikationen

6.1 Endgültiges Levelformat

6.2 Levelbeschreibungen

7 Unit Tests

7.1 package lambda.model.profiles

7.1.1 public class **ProfileEditModelTest** implements ProfileEditObserver

Beschreibung

Testklasse für das ProfileEditModel

Tests

- **testLanguageNextPrev**()
  Testet, ob die Auswahl der nächsten/vorherigen Sprache funktioniert und entsprechend **changedLanguage**() der Observer aufgerufen wurde.

- **testAvatarNextPrev**()
  Testet, ob die Auswahl des nächsten/vorherigen Avatars funktioniert und entsprechend **changedAvatar**() der Observer aufgerufen wurde.

- **testLanguageCycle**()
  Testet, ob die Auswahl der Sprache zyklisch ist. D.h. ob man falls man lang genug die nächste/vorherige Sprache wählt, wieder am Anfang ankommt.

- **testAvatarCycle**()
  Testet, ob die Auswahl der Avatare zyklisch ist. D.h. ob man falls man lang genug den nächsten/vorherigen Avatar wählt, wieder am Anfang ankommt.

7.1.2 public class **ProfileManagerTest** implements ProfileManagerObserver

Beschreibung

Testklasse für den ProfileManager. Testet anhand Testprofilen Grundfunktionen des ProfileManagers und das Aufrufen der entsprechenden Benachrichtigungsmethoden auf seinen Observern.

Tests

- **testProfileLoad**()
  Überprüft, ob der ProfileManager die Testprofile lädt und **getNames**() deren Namen korrekt zurückgibt.

- **testCurrentProfile**()
  Versucht ein Profil auszuwählen und testet, ob dies korrekt geschehen ist.

- **testRenaming**()
  Überprüft, ob das Umbenennen eines Profils.

- **testDeleteCreateProfile**()
  Testet ein Szenario, bei dem ein Profil gelöscht wird und danach ein neues mit gleichem Namen generiert wird.

- **testDeleteSaveWrongProfile**()
  Versucht Profile, die nicht existieren zu speichern und zu löschen und schlägt fehl falls dies geschieht oder zu einem Fehler führt.

7.1.3 public class **ProfileModelTest** implements ProfileModelObserver

Beschreibung

Testklasse für das ProfileModel.

Tests

- **testNewProfile**()
  Erstellt eine neue ProfileModel-Instanz/ein neues Profil und testet, ob es mit den richtigen Standard-Werten initialisiert wurde.

- **testRenameProfile**()
  Testet das Umbenennen eines Profils durch den entsprechenden Konstruktor. Überprüft dabei, ob das neue Profil den richtigen Namen hat und die sonstigen Werte des Alten korrekt übernommen hat.

- **testSettersGetters**()
  Testet alle Getter und Setter des ProfileModels und stellt sicher, dass dabei die entsprechenden Benachrichtigungsmethoden der Observer aufgerufen wurden.

7.2 package lambda.model.settings

7.2.1 public class **SettingsModelTest** implements SettingsModelObserver

Beschreibung

Testklasse für das SettingsModel.

Tests

- **testSetMusicOn**()
  Testet, ob die Musik an und aus gestellt werden kann und ob dabei entsprechend **changedMusicOn**() der Observer aufgerufen wird.

- **testSetMusicVolume**()
  Testet, ob die Lautstärke der Musik geändert werden kann und ob dabei entsprechend **changedMusicVolume**() der Observer aufgerufen wird.

- **testSetSoundVolume**()
  Testet, ob die Lautstärke der Sounds/Geräusche geändert werden kann und ob dabei entsprechend **changedSoundVolume**() der Observer aufgerufen wird.

- **testForInvalidValues**()
  Stellt sicher, dass **setMusicVolume**() und **setSoundVolume**() trotz Übergabe nicht erlaubter Werte immer nur Werte im legalen Bereich [0,1] annimmt.

8 Zeitplanung

8.1 Vorläufiger Zeitplan

8.2 Endgültiger Zeitplan

9 Glossar

Android Betriebssystem und Softwareplattform für hauptsächlich mobile Geräte. Das Produkt wird für mehreren Plattformen entwickelt, aber in erster Linie für Android.

Asset Asset in ein Sammelbegriff für Grafiken, Musikdaten, Sprachpakete, Videos etc. Assets werden ins Programm geladen (z.B beim Programmstart) und dort verwendet, wobei sie aber generell nicht verändert werden.

Identifizierer Ein Identifizierer oder kurz Id ist eine eindeutig und einmalig vergebene Nummer für ein Objekt, um dieses wieder zu erkennen.

JSON Kurz für "JavaScript Object Notation". JSON stellt ein Datenformat dar, das es ermöglicht Daten fest und in einer einfach lesbaren Form abzuspeichern.

LibGDX LibGDX ist ein auf Java basierendes Framework für die Entwicklung von Multiplattform-Spielen. So erlaubt es mit der gleichen Code-Basis die Entwicklung für Desktop und Mobile Endgeräte wie Windows, Linux, Mac OS X, Android, iOS und HTML5.

10 Anhang

Angehängt wird unser Erzeugnis aus dieser Entwicklungsphase mit Quellcode und Junit-Tests sowie einer ausführbaren Android-Apk-Datei. Dies befindet sich alles unter dem Projekt-Verzeichnis, das sich im gleichen Verzeichnis befindet wie dieses Dokument befindet. Die Vorbedingungen sowie die nötigen Schritte zur Einbindung unseres Projekts in eine IDE sind in einer readMe.txt, die sich ebenfalls im Projekt-Verzeichnis befindet, zusammengefasst.