Karlsruher Institut für Technologie

Lamb.da - Das Spiel

Implementierungsbericht

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

2. März 2015

Inhaltsverzeichnis

1 Einleitung

2 Umsetzung von Muss- und Wunschkriterien

2.1 Musskritierien

2.2 Umgesetzte Wunschkriterien

2.3 Nicht umgesetzte Wunschkriterien

3 Änderungen gegenüber dem Feinentwurf

3.1 package lambda.viewcontroller

Notizen

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

3.1.1 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.

3.1.2 `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.

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

Allgemein

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.

3.2 package lambda.viewcontroller.assets

3.2.1 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.

3.3 package lambda.model.profiles

3.3.1 public interface **ProfileModelObserver**

Entfernte Methoden

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

3.3.2 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.

4 Eigene Exceptions

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