Karlsruher Institut für Technologie

Lamb.da - Das Spiel

Testbericht

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

26. März 2015

Inhaltsverzeichnis

1 Einleitung

Nach der Implementierungsphase unserer Applikation „Lamb.da“, war sie schon zum größten Teil fertig, jedoch gab es noch einige funktionale Fehler. Die Qualitätssicherung wurde verwendet um diese Fehler zu finden und zu beheben. Ein anderer wichtiger Teil dieser Phase war die grafische und spielerische Optimierung des Produkts. So wurden Benutzeroberflächen angepasst und die Level des Spiels teilweise stark überarbeitet und eine große Menge, einfach verständlicher, Tutorials mit entsprechenden Beispielsgrafiken eingefügt. Wie die Applikation getestet wurde, welche Verbesserungen es gab und welche Bugs gefunden und behoben wurden ist im folgenden Bericht dokumentiert.

2 Verbesserungen

2.1 package lambda.model.shop

- Einfügen weiterer Items
  - Im Laufe der Qualitätssicherung wurde der Shop mit Items erweitert, sodass nun insgesamt 5 Musik-Items, 2 Background-Items, sowie 2 Elemente-Items erwerbbar sind.

2.2 package lambda.model.level

- Neue Exception: InvalidLevelIdException
  - Im Laufe der Qualitätssicherung wurde das Projekt um eine Exception erweitert, welche geworfen wird, falls man ein Level mit einer invaliden Id beantragt.
- Schwierigkeitsgrade
  - Die Schwierigkeitsgrade wurden im Laufe der Qualitätssicherung so ausgebaut, dass nun jede Stufe seine eigene Musik, sowie seinen eigenen Hintergrund bereitgestellt bekommt.

3 Änderungen am Pflichtenheft

4 Testwerkzeuge

4.1 Statische Werkzeuge zur Codeanalyse

Checkstyle Checkstyle ist ein freies statisches Codeanalyse-Werkzeuge, welches verwendet wird, um die Java-Codequalität zu verbessern. Dieses statische Analyse-Werkzeug lässt sich mit einer geeigneten Datei konfigurieren, um zu entscheiden nach welchen Kriterien die Qualität verbessert wird. Der von uns verwendete Checkstyle ist im Repository vorzufinden und prüft unter anderem die Methodenlänge, Dateilänge, Anzahl an Methodenparameter, richtige Formatierung des Codes (Whitespaces etc.), Namenkonventionen

FindBugs FindBugs ist ein freies statisches Codeanalyse-Werkzeug, welches verwendet wird, um insbesondere Fehlermuster in Java-Code bzw. im Java-Bytecode zu finden. Diese Fehlermuster können unter anderem auf wirkliche Fehler hindeuten, müssen sie aber nicht. Wir haben dieses Werkzeug verwendet, um die eben angesprochenen Fehlermuster aufzudecken und zu überprüfen, ob diese zu Fehlern führen oder nicht. Alle gefundenen Bugs waren recht klein und führten demnach nicht zu Fehlern, weshalb eine Beseitigung nicht nötig war.

4.2 Dynamische Werkzeuge zur Codeanalyse

JUnit JUnit ist ein Java-Testframework, welches verwendet wird, um Java-Programme zu testen. Dazu werden Tests geschrieben, welche man immer wieder automatisiert durchlaufen lassen. Des Weiteren stellt dieses Framework einige Methoden bereit, um Attribute der einzelnen Klassen auf Richtigkeit zu überprüfen. Dieses Werkzeug wurde von uns verwendet, um die Zusammenarbeit einzelner Module zu testen, sowie die Überprüfung der Attribute und weiterhin auch, um durch EMMA (siehe nächster Punkt) eine Testüberdeckung veranschaulichen zu können.

Code Coverage mit EMMA EMMA ist ein dynamische Codeanalyse-Werkzeug, welches verwendet wird, um die Code Coverage (Testüberdeckung) von JUnit-Tests zu messen. Mit dieser Testüberdeckung lässt sich oft Code herausfiltern, welcher niemals ausgeführt wird und weiterhin die Abdeckung von diversen Modultests. EMMA basiert auf einer partiellen Zeilenüberdeckung, in der teilweise ausgeführte Zeilen registriert werden, z.B. bei verzweigenden Anweisungen. Dieses dynamische Werkzeug wurde von

uns verwendet, um die Testüberdeckung der JUnit-Model-Tests zu beobachten und zu analysieren.

MonkeyRunner MonkeyRunner ist ein dynamisches Werkzeug, welches mit Python-Programmen ausgeführt wird. Das Werkzeug funktioniert für Android-Geräte oder solche Emulatoren. Dieses dynamische Werkzeug führt vorgeschriebene Python-Dateien aus und klickt automatisiert an die in der Python-Datei definierten Stellen, wobei es immer wieder Screenshots aufnimmst. MonkeyRunner wurde von uns verwendet, um unsere definierten globalen Testfälle und Systemtests nachzustellen, um diese nicht immer wieder per Hand durchzugehen.

Monkey Monkey ist ein dynamisches Werkzeug, welches gerne für Belastungs- und Stresstests für Android-Applikationen verwendet wird. Dieses Tool läuft entweder direkt auf Android-Geräten oder auf solchen Emulatoren. Dieses dynamische Werkzeug generiert zufällige und hochfrequentierte Eingaben und prüft somit, ob irgendwelche unerwarteten Exceptions oder Fehler auftreten bzw. ab welcher Frequenz von Eingaben die Applikation abstürzt. Dieses Tool wurde von uns verwendet, um die eben beschriebenen Belastungs- und Stresstests durchzuführen.

5 Unittests

5.1 package lambda.model.shop

ShopModelTest Diese Testklasse des ShopModels überdeckt alle Models vom `package lambda.model.shop`. Die einzelnen Tests überprüfen das Zusammenwirken vom `ShopModel` mit den diversen Kategorien (`ShopItemTypeModel`), sowie den verschiedenen Items (`ShopItemModel` bzw. `MusicItemModel`, `BackgroundImageItemModel` und `ElementUIContextFamily`).

Coverage package lambda.model.shop: 98,0%

5.2 package lambda.model.level

LevelModelTest Diese Testklasse überprüft, ob nach einem erstellten Level alle Attribute richtig übertragen und gesetzt wurden, sowie ob die Auswahl der Reduktionsstrategien und benutzbaren Elementen korrekt ist.

DifficultySettingTest Diese Testklasse überprüft, ob nach einer erstellten Schwierigkeitsstufe der Grad der Stufe und die Dateinamen der Musik-Dateien und Bild-Dateien korrekt ist.

LevelContextTest Diese Testklasse testet das Zusammenwirken der verschiedenen Klassen im package. `LevelContext` umfasst ein `LevelModel`, sowie eine `DifficultySetting`. In dieser Testklasse werden alle Levels und Schwierigkeitsgrade über den `LevelManager` geladen, der Shop komplett gesetzt und danach durch diverse Tests alles auf Richtigkeit überprüft.

Coverage package lambda.model.level: 88,2% Die Testüberdeckung ist beinahe maximal, da einige Zeilen wie beispielsweise der Header einer Enumeration nicht mit in die Überdeckung einbezogen werden.

6 Integrationstests

7 Systemtests

Die globalen Testfälle des Pflichtenhefts wurden mithilfe des MonkeyRunner-Werkzeuges umgesetzt. Jeder Testfall wurde in einer, durch MonkeyRunner ausführbaren, Python-Datei abgelegt. Benannt sind sie entsprechend der Testfälle (z.B. Testfall T110 in der T110.pyDatei). In ihnen befinden sich Anweisungen und Vorraussetzungen der Testfälle. Ist die Applikation auf einem Android-Gerät in den, in der Datei, beschriebenen Startzustand gebracht, kann das Testskript ausführt werden. MonkeyRunner geht dann automatisch das entsprechende Testszenario durch.

In der Implementierung bzw. Qualitätssicherung kam es zu kleinen, gewollten Abweichungen zu den Testfällen.

-

- `/T110/ Erstmaliges Starten des Programms`
  Während dem Laden des Programms werden im Ladebildschirm keine Texte oder Comics mehr angegeben.

- `/T120/ Starten des Programms, nachdem mindestens ein Profil bereits erstellt wurde`
  Der Begrüßungsbildschirm wurde hierbei verworfen. Die Applikation zeigt diesen nur noch an nachdem ein neues Profil erstellt wurde

- `/T150/ Beispiellevel zur Eingabe-Bestimmung`
  Die Art und Weise wie Spielelemente platziert werden, wurde grafisch geändert. Es wird jetzt kein transparentes Spielelement angezeigt. Stattdessen zeigt das Spiel weiße Markierungen an allen möglichen Stellen an, an denen man etwas ablegen kann. Ist man mit dem Element in der Nähe der Markierung färbt sie sich grün und signalisiert somit, dass man das Element dort beim Loslassen platziert.

- `/T170/ Das Einkaufsmenü benutzen`
  Items aktivieren sich nicht mehr automatisch nachdem sie gekauft wurden. Die Testsequenz wurde so abgeändert, dass nach dem Kaufen der 2 Items Item 2 und dann 1 aktiviert werden. Die Aktivierung von Item 1 deaktiviert wie gewohnt Item 2. Aktivierte Items werden ebenfalls nicht mehr mit Haken, sondern mit einer farblichen Veränderung des entsprechenden Knopfes gekennzeichnet.

- `/T180/ Optionen auswählen`
  Lehrermodus und Farbenblindenmodus sind schon in vorherigen Phasen entfernt worden. Dadurch ändert sich der Testfall entsprechend.

Anmerkung:

- `/T210/ Ein Profil ist eindeutig durch den Namen gekennzeichnet. Es kann nicht mehrere Profile mit demselben Namen geben.`
  Dieser Testfall zur Datenkonsistenz wurde nicht wie oben durch MonkeyRunner umgesetzt. Seine Einhaltung ist aber schon durch die Unit-Tests gegeben.

8 Statistik

9 Lasttests

Expandierende Lambda-Terme Zunächst wurde die Auswertung eines immer weiter wachsenden Lambda-Terms im Reduktionsmodus in einer Endlosschleife durchgeführt. Dies führte dann häufig zu StackOverflowExceptions. Dieses Problem behoben wir, indem wir eine Maximalzahl an Knoten (100) in den Lambda-Termen und damit in den Elementen einführten, die die Reduktion der Lambda-Terme begrenzt.

Große Lambda-Terme Die Auswertung von Lambda-Termen, welche aus sehr vielen Lämmern und Edelsteinen bestehen, wurde ebenfalls getestet. Dabei kam es zu keinen größeren Performanceeinbußen, jedoch führten sehr große Lambda-Terme teilweise ebenfalls zu StackoverflowExceptions, was so wie bei expandierenden Ausdrücken durch Einführung einer Maximalanzahl Von Elementen behoben wurde.

Willkürliche Bildschirmeingaben Mithilfe des monkey-Werkzeugs wurde getestet wie sich die Applikation bei zufälligen und hochfrequenten Bildschirmeingaben verhält. Das monkey-Werkzeug wurde also für Stress-Tests verwendet. Beim mehrmaligen Testläufen mit unterschiedlichen Event Sequenzen konnten unter normalen Bedingungen keine Auffälligkeiten festgestellt werden, es kam zu keinen Abstürzen oder sonstigen unerwarteten Verhaltensweisen.

10 Behobene Bugs

10.1 Shop

- Anzeige von mehreren aktivierten Items pro Kategorie
  - Der Bug im Shop, wenn man zwei verschiedene Items einer Kategorie aktivierte, dass beide als aktiviert angezeigt werden, wurde behoben und somit wird der Status jedes Items nun richtig angezeigt.
- Fehlende Anzeige aktivierter Items nach Programmneustart
  - Der Bug im Shop, dass ein bereits aktiviertes Item nach einem Neustart des Programms nicht mehr als aktiviert angezeigt wird, wurde behoben und nun werden beim Laden eines Profils, wechseln des Profils sowie beim Neustarten alle Zustände der Items richtig angezeigt.
- Schließen offener Kategorien nach Item-Einkauf
  - Der Bug im Shop, wenn ein Item gekauft wurde und direkt nach dem Kauf alle offenen Kategorien geschlossen wurden, wurde behoben und der aktuelle Zustand einer Kategorie (ob offen oder geschlossen) wird nach einem Kauf nun beibehalten.

11 Glossar

Statische Codeanalyse Statische Codeanalyse ist ein statisches Testverfahren für Software, welches zur Compilerzeit durchgeführt wird. Hierbei wird der Quellcode formaler Prüfungen unterzogen bei welchen Fehler in Form von Code-Qualität oder Formatierung entdeckt werden können.

Dynamische Codeanalyse Dynamische Codeanalyse ist ein dynamisches Testverfahren für Software, welcher zur Laufzeit durchgeführt wird. Hierbei können Programmfehler aufgedeckt werden, welche durch evtl. variierende Parameter oder variierender Nutzer-Interaktion auftreten können.

Tool / Werkzeug Tools bzw. Werkzeuge beschreiben die Mittel, mit welchen entweder statische oder dynamische Codeanalyse durchgeführt werden können.