Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Reimplementation des Videospiel Klassikers Mario

Pflichtenheft

vorgelegt von

Lars Hick, Tim Eckle und Florian Plesker

Betreuer: Simon Eismann

Würzburg 2018

Inhaltsverzeichnis

1 Einleitung

Im Zuge des Programmierpraktikums der Universität Würzburg soll der Spieleklassiker Super Mario Bros. von Nintendo aus dem Jahre 1987 reimplementiert werden. Das Spiel war für drei Jahrzehnte in Folge das meistverkaufte Videospiel und zählt bis heute zu den einflussreichsten Spielen aller Zeiten. Es ist also kein Wunder, dass der Markenname Mario, der sich nach diesem mehr als erfolgreichen Start zu einer ebenso erfolgreichen Videospielreihe entwickelte, bei vielen Menschen Erinnerungen weckt. Jede dieser Personen wird sofort ein Bild oder eine Idee im Kopf haben, was für ihn oder sie persönlich ein Mario-Spiel ausmacht. In unserer Neuimplementation des Klassikers werden wir uns weniger an den aktuellen Titeln orientieren, sondern mehr nach dem Motto „Back to the basics“ agieren und den Klempner auf seiner Reise durch die Spielwelt nur mit dem Nötigsten ausstatten. Dies beinhaltet die simple, jedoch wichtige Fähigkeit, die Spielfigur überhaupt steuern, also mit der Figur durch das Level laufen oder rennen, zu können. Hindernisse im Level, ob Gegner oder Abgründe, werden durch Springen besiegt, beziehungsweise überwunden. Wir hoffen all denjenigen, die unsere Version spielen werden, das Gefühl eines echten Mario Spiels geben zu können. In diesem Sinne:
Let‘s go !

2 Rahmenbedingungen

In diesem Abschnitt werden die Voraussetzungen an die Zielsysteme erläutert und definiert.

2.1 Anforderungen an die Zielmaschine

Folgende Anforderungen muss der Rechner erfüllen damit ein gutes Spiel-Erlebnis garantiert werden kann:

- Windows 10 Rechner.
- x86-64 bit Prozessor.
- 8 GB Arbeitsspeicher.
- Java 8.
- Tastatur.

2.2 Entwicklungsumgebung

- IntelJii
- Java 8 (inkl. JavaFX)
- Windows 10 Home Betriebssystembild 16299.371

2.3 Anwendungsbereiche

Das Endprodukt dient allein der Unterhaltung der Teilnehmer des Softwarepraktikums im Sommersemester 2018 an der Universität Würzburg.
Es hat sonst keine weiteren Anwendungsbereiche.

3 Funktionale Anforderungen

3.1 Essentielle Kriterien

- Die Spielfigur muss sich bewegen können.
- Die Spielfigur muss springen können.
- Beim Kontakt mit Gegnern und durch Herabfallen in Abgründe verliert der Spieler.
- Es muss ein Menü vorhanden sein.
- Das Spiel muss Soundeffekte besitzen.
- Im Laufe der Entwicklung wird es eine unerwartete Requirement-Änderung geben.
- Es muss eine Model-View-Controller-Architektur mit strikter Aufgabentrennung angewandt werden.

3.2 Bedingt notwendige Kriterien

- Das Spiel besitzt ein Ende.
- Es gibt Blöcke mit denen die Spielfigur interagieren kann.
- Der Charakter kann Münzen einsammeln.
- Das Spiel besitzt verschiedene durchspielbare Instanzen.
- Die Spielfigur erhält Animationen.

3.3 Optionale Kriterien

- Der Spielverlauf kann gespeichert werden.
- Im Spielverlauf können verschiedene Power-ups eingesammelt werden.
- Die erreichte Punktzahl wird am Ende des Spiels in einer Rangliste gespeichert.
- Spieler besitzt mehr mehrere Leben.

3.4 Abgrenzungskriterien

- Es sollen keine Graphiken erstellt werden.
- Es muss kein eigener Sound kreiert werden.

4 Funktionalität

Der Spieler wird sich bei Betätigung der entsprechenden Tasten durch die Spielwelt bewegen können. Abgesehen vom Laufen nach links oder rechts, wird es dem Spieler auch möglich sein mit dem Charakter springen zu können. Diese Fähigkeit dient dazu, Hindernisse im Level, zum Beispiel in der Form von Gegnern, überwinden beziehungsweise beseitigen zu können. Die Kollision mit einem Gegner, sofern diese nicht von oben herab stattfindet, oder der Sturz in einen Abgrund führt zum Verlust eines Lebens. Untermalt wird das Spiel von akkustischen Effekten. Sowohl nach Spielstart, als auch beim Pausieren des Spiels erhält der Spieler Zugriff auf ein Menü, welches ihm erlaubt, aus verschiedenen Unterpunkten zu wählen. Das Gesamte Projekt wird dabei strikt nach der Model-View-Conroller-Architektur implementiert.

4.1 Benutzungsoberfläche

Es wird eine objektorientierte Oberfläche geben, auf der der Spielcharakter so wie die Spielwelt dargestellt werden. Der Charakter wird voraussichtlich mindestens Animationen für das Laufen und Springen erhalten. Zudem wird es verschiedene Anzeigen geben, welche die wichtigsten Informationen -z.B. die aktuelle Lebenszahl- anzeigen. Beim Start, wie auch beim Pausieren des Spiels wird ein Spielmenü angezeigt. Jenes gewährt Zugriff auf die wichtigsten Funktionen, wie beispielsweise die Speicherung des Spielstandes.

4.2 Qualitätsziele

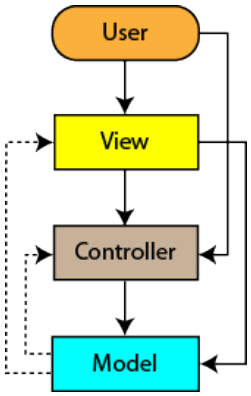
Das Spiel soll mindestens mit einer konstanten Bildrate von 10 Bildern pro Sekunde laufen.

4.3 Abnahmekriterien

Bei Abnahme des Projekts durch den Betreuer Simon Eismann soll mindestens der folgende Ablauf des Spiels ohne kritische Fehler möglich sein:

- Programmstart.
- Spielbeginn.
- Bewegen der Spielfigur.
- Durchlaufen des Levels.
- Bei Springen auf Gegner verschwinden diese.
- Überspringen von Hindernissen ist möglich.
- Beenden des Levels bei Zieldurchlauf.
- Alternatives Ende bei Verlust aller Leben.
- Abzug eines Lebens bei Berührung von Gegnern, die nicht von oben getroffen werden oder bei Stürzen in Abgründe.

5 Gesamtsystemarchitektur



Ein MVC-Paradigma wird VCM-Paradigma genannt, wenn die drei zugehörigen Module folgende Schichtenarchitektur bilden: View, Controller, Model.
Diese Architektur ist für interaktive Anwendungen wie z. B. Spiele sehr gut geeignet.
Die drei Module einer MVC-Komponente sind gemäß dem Schichtenparadigma in Ebenen angeordnet. Die höheren Ebenen können auf tiefergelegene Ebenen zugreifen, aber nicht umgekehrt.
Die unserste Ebene enthält das Modell. Das zugehörige Modul weiß nichts von den über ihr liegenden Module und kann mit diesen nur mit indirekt – z. B. durch Antworten auf Nachrichten oder mit Hilfe des Observer-Patterns – kommunizieren.
Ein Controllermodul kann direkt auf ein Modellmodul zugreifen, um die Inhalte zu manipulieren.
Eine View kommuniziert i. Allg. direkt nur mit Controllermodulen, um diesen Benutzeraktionen, die über die View erfolgen, mitzuteilen. Ein direkter Zugriff auf ein Datenmodul ist nur dann notwendig, wenn die Nachrichten, die von den Datenmodulen verschickt werden, nicht alle relevanten Informationen enthalten. Zugriffe auf Servicemodule sind nicht vorgesehen.
Der Benutzer stellt das „oberste Modul“dar. Er kommuniziert nur mit View- und Controllermodulen.
Man beachte, dass in diesem Paradigma die Logik sowohl im Modell als auch im Controller realisiert werden kann.[34]

[34]https://glossar.hs-augsburg.de/Model-View-Controller-Paradigma#Model_.28Modell.29

6 Schnittstellenarchitektur

Model:

- Spielfiguren:
  - void setPosition(Pos pos)
  - Pos getPosition()
  - void setWalkingSpeed(int speed)
  - int getWalkingSpeed()
  - void setLifes(int lifes)
  - void addLife()
  - void takeLife()
  - int getLifes()
- Level:
  - int getDistance()

View:

- void init()
- void update(Model model, List<Events>)