«3D Endless Runner» - Game Entwicklung

Diplomarbeit

Céline Hofstetter
Höhere Fachschule Uster
Klasse 16I

Abschlussjahr 2019 | Betreuer: Mario Sabbatella
Druckdatum: 27.02.2019

Dokumentangaben

Dokumentendetails	
Version	1.0
Gültig bis	Unbestimmt
Status	Gültig
Projekttyp	Diplomarbeit
Dokument Klassifizierung	Öffentlich

Änderungsnachweis

Kalenderwoche	Autor	Version	Änderungen	Veränderte Stellen
KW 41	Céline Hofstetter	0.1	- Dokument neu erstellt - Grobgerüst - Einleitung - Projektplanung - Initiierung	Komplettes Dokument
KW 42	Céline Hofstetter	0.2	- Pflichtenheft - Initiierung	Seiten 8-20
KW 43	Céline Hofstetter	0.3	- Initiierung - Definition	Seiten 12-30
KW 44	Céline Hofstetter	0.4	- Definition - Testkonzept / Testfälle	Seiten 21-30, 48-56
KW 45	Céline Hofstetter	0.4.1	- Definition	Seiten 21-30
KW 46	Céline Hofstetter	0.5	- Realisation	Seiten 31-47
KW 48	Céline Hofstetter	0.5.1	- Realisation	Seiten 31-47
KW 49	Céline Hofstetter	0.5.2	- Realisation	Seiten 31-47
KW 50	Céline Hofstetter	0.5.3	- Realisation	Seiten 31-47
KW 51	Céline Hofstetter	0.5.4	- Realisation	Seiten 31-47
KW 52	Céline Hofstetter	0.5.5	- Realisation	Seiten 31-47
KW 01	Céline Hofstetter	0.5.6	- Realisation	Seiten 31-47
KW 02	Céline Hofstetter	0.6	- Testing	Seiten 48-56
KW 03	Céline Hofstetter	0.6.1	- Testing	Seiten 48-56
KW 04	Céline Hofstetter	0.7	- Realisation - Testing	Seiten 31-56
KW 05	Céline Hofstetter	0.8	- Realisation	Seiten 31-47
KW 06	Céline Hofstetter	0.9	- System Architektur Design - Management-Summary - Reflexion	Seiten 5, 21 und 57
KW 07	Céline Hofstetter	0.9.1	- Glossar - Quellenangaben - Tabellen-/Bildverzeichnis	Seiten 58-60
KW 08	Céline Hofstetter	0.9.2	- Anhang - Diverse Layout Anpassungen	Gesamtes Dokument
KW 09	Céline Hofstetter	1.0	- Finalisierung des Dokuments	Gesamtes Dokument

1 Inhaltsverzeichnis

2 Management Summary

2.1 Ausgangslage

Gameentwicklung – Soll ich oder soll ich nicht? Eine Frage, welche mir seit längerem nicht mehr aus dem Kopf geht.

2.2 Kurzbeschreibung des Projekts / Projektidee

Aus der Frage in der Ausgangslage und basierend auf der Vordiplomarbeit (2D Jump and Run mit dem Java 2D API) resultierte die Projektidee, ein 3D Game mit einer Game Engine zu entwickeln. Der Entscheid fiel auf ein «Endless Runner» Spiel. In dem Spiel geht es darum, dass die Spielfigur stetig läuft und man per Tastatureingabe die Spielfigur steuert. Vor der Spielfigur tauchen Hindernisse auf, welchen korrekt ausgewichen werden muss.

2.3 Lösungsvorgehen

In meinem Umfeld erkundigte ich mich vergebens, ob jemand bereits Erfahrung mit 3D Games oder mit einer Game Engine hat. Somit setzte ich die Analyse und die Abklärungen für das Lösungskonzept im Internet fort. Die erarbeiteten Lösungsansätze waren für diese Arbeit alle gut genug. So konnte ich meine persönlichen Aspekte miteinbringen und eine Game Engine verwenden, welche ich schon lange einmal ausprobieren wollte.

2.4 Schwierigkeiten

Trotz ausgiebiger Analyse fiel mir die Erstellung des Zeitplans für die Realisierungsphase ohne Erfahrungswerte nicht leicht. Deshalb plante ich überall genügend Zeit ein, sodass ich nicht unter Zeitdruck gerate. Um mit Sicherheit auf die 285 Stunden Aufwand zu kommen, definierte ich über die Zeitvorgabe hinaus noch zwei Kann-Anforderungen.

2.5 Resultat und Ausblick der Arbeit

Die Frage in der Ausgangslage kann ich nun definitiv mit ja beantworten. Gameentwicklung macht mir Spass und wäre sicherlich etwas für mich. Es steht für mich zur Option, dass Spiel im privaten noch Mobile fähig zu machen. Ansonsten ist das Spiel in sich komplett abgeschlossen und es sind keine Weiterentwicklungen der Funktionalitäten in Aussicht.

Folgende Abbildung zeigt das Resultat der Arbeit:

Abbildung 1 – Screenshot des Spiels

3 Einleitung

3.1 Titel der Diplomarbeit

«3D Endless Runner» - Game Entwicklung

3.2 Thematik

Ziel dieser Diplomarbeit ist es ein 3-dimensionales «Endless Runner» Game zu realisieren. Dazu soll eine Game Engine und die Programmiersprache C# verwendet werden. Das Spiel soll auf einem Windows PC laufen.

3.3 Ausgangslage / Motivationsgründe

Gaming würde ich nicht als ein Hobby von mir bezeichnen, doch ich spiele öfters mit dem Gedanken, später Richtung Gameentwicklung zu gehen. Vor ca. 1 Jahr programmierte ich ein 2D Jump and Run als Vordiplomarbeit. Ein animiertes 2D Game, welches ohne eine Game Engine nur mit Java entwickelt wurde. Für ein kleines 3D Game mit einer Game Engine sind genügend C# Kenntnisse vorhanden, um mir einen weiteren Einblick in das Thema zu verschaffen. Das «3D Endless Runner» Game ist etwas ganz anderes, deshalb wird es von Grund auf neu entwickelt.

Mein erstes eigenes 3D Spiel mit einer Game Engine zu entwickeln, um ein herauszufinden, ob Gameentwicklung später eine Variante wäre, hat mich motiviert. Nach dieser Arbeit sollte mein Spiel funktionstüchtig laufen. Etwas mit einer Game Engine zu realisieren und mit Grafiken zu arbeiten ist für mich noch Neuland und es wird bestimmt interessant, dies zu erlernen.

3.4 Detaillierte Aufgabenstellung

Entwickeln Sie das 3D Spiel «**Endless Runner**» auf Basis einer Game Engine wie z.B. Unity und der Programmiersprache C#.

Gliedern Sie dabei nach folgenden Teilaufgaben:

- Erstellung des Projektplans
- Erstellen des Pflichtenhefts mit den Anforderungen an die Lösung
- Design der Software Architektur, Spiellogik / -konzept, Spielfiguren, Datenhaltung
- Design und Realisierung der 3D Spiel- und Bedienoberfläche
- Realisierung der Lösung
- Dokumentieren der Test Fälle und der Test Resultate
- Erstellen der Dokumentation

3.5 Technologien, Mittel und Methoden

- C#
- Unity
- Visual Studio 2017
- Blender
- Laptop (Surface - Windows 10)
- Visio
- Office allgemein
- Internet (für Hilfe)

4 Projektplanung

4.1 Rollenverteilung

Dieses Projekt wird von mir als Einzelarbeit durchgeführt.

4.2 Vorgehensmethode

Als Vorgehensweise wählte ich die Wasserfallmethode, welche sich in verschiedene Phasen gliedern lässt. Erst wenn eine Phase abgeschlossen ist, wird die nächsttiefere Phase eingeleitet.

Eine Ausnahme bildet die Dokumentationsphase. Sie wird von Beginn bis zum Ende des Projektes laufend weitergeführt.

Für mein Projekt habe ich diese wie folgt angewendet:

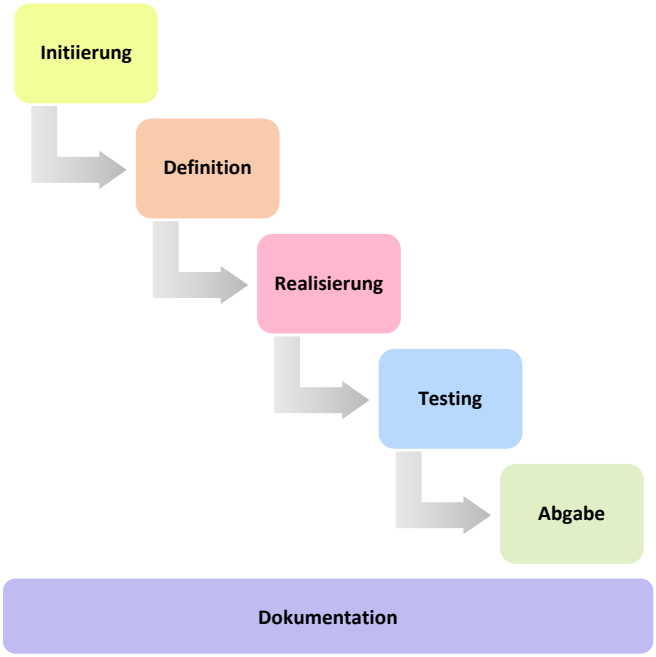

Abbildung 2 – Wasserfalldiagramm

4.3 Versionsverwaltung/ Backup

4.3.1 Code

Der Code wird stetig auf GitLab hochgeladen. So kann man ohne grossen Aufwand auf ältere Versionen zurückgreifen.

4.3.2 Dokumentation

Die Dokumentation wird mit OneDrive in der Cloud gesichert, falls der Arbeitscomputer defekt sein sollte. Für jede Projektphase, ausser für die Abgabe, wurde ein Ordner erstellt, indem die entsprechenden Dokumente, Grafiken, etc. abgelegt werden.

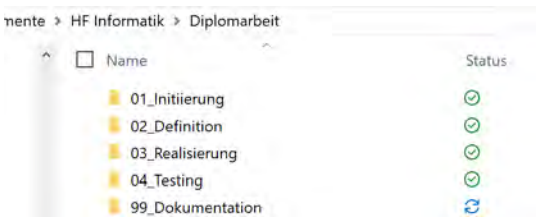

Abbildung 3 – Ordnerstruktur

5 Pflichtenheft

5.1 Allgemeines

5.1.1 Sinn und Zweck

Das Pflichtenheft dient dazu, den Umfang der Arbeit klar zu definieren und abzugrenzen. Ebenfalls soll es die Möglichkeit liefern, daraus eine Aufwandschätzung (Soll-Zeitplan) erstellen zu können. Es dient zudem als Basis für die Realisierung dieses Projekts und ist ein Bestandteil der Diplomarbeit.

5.1.2 Anwendungs- / Einsatzgebiet

Das Spiel ist nur für den privaten Gebrauch gedacht und wird nicht veröffentlicht.

5.1.3 Vorgesehene Erweiterungen

Die Zielplattform der Software sollte ausbaufähig sein, sodass sie später mit geringem Aufwand auch Mobile-fähig ist.

5.2 Allgemeine Beschreibung

5.2.1 Übersicht

5.2.1.1 Ausgangssituation

Für ein kleines 3D Game mit einer Game Engine sind genügend C# Kenntnisse vorhanden, um mir einen weiteren Einblick in das Thema Game Entwicklung zu verschaffen. Das Game «Road Runner» wird von Grund auf neu entwickelt.

5.2.1.2 Abgrenzung des Umfelds

Grafiken werden eher rudimentär dargestellt, der Schwerpunkt liegt deutlich auf den Funktionalitäten.

5.2.1.3 Beziehungen zu anderen Systemen

Es bestehen keine Beziehungen zu anderen Systemen. Es ist eigenständig.

5.2.2 Software

Es wird ein 3D Game namens «Road Runner» realisiert. In dem Spiel geht es darum, dass die Spielfigur rennt und man per Tastatureingabe die Spielfigur steuert. Vor der Spielfigur tauchen Hindernisse auf, welchen korrekt ausgewichen werden muss. Schafft die Spielfigur ein Hindernis nicht, ist das Spiel vorbei. Levels gibt es bei diesem Spiel keine. Um die Schwierigkeit trotzdem laufend zu erhöhen, wird es in regelmässigen Abständen ein bisschen schneller. Das heisst, dass die Spielfigur sich immer schneller fortbewegt. Ebenfalls sollte auf dem GUI eine Punktzahl ersichtlich sein, sodass sich der Benutzer messen kann.

Zum Schluss kann noch eine Highscore Ansicht in das Menu implementiert werden und die Spielfigur mit der Funktion «springen» erweitert werden. Dies sind jedoch keine Muss-Anforderungen.

5.2.3 Übersicht der Funktionen

- Menu
- Spielfigur bewegt sich stetig
- Spielfigur weicht aus
- Hindernisse tauchen auf
- Kollisionscheck
- Game Over
- Geschwindigkeit erhöhen
- Punkte-Zähler
- Kameraperspektive
- *Highscore Ansicht*
- *Spielfigur springt*

5.2.4 Entwicklungsumgebung

Als Entwicklungsumgebung wird eine Game Engine verwendet. Die Evaluation, welche Game Engine sich am besten eignet, wird in der Arbeit stattfinden.

5.3 Technische Anforderungen

5.3.1 Allgemeines

5.3.1.1 Installation

ZIP-Datei auf den gewünschten Computer kopieren und entpacken.

5.3.1.2 Benutzer-/Bedienungsanleitung

Das Spiel ist eher simpel in der Bedienung und sollte nach der ersten Benutzung verständlich sein. Für neue Benutzer wird ein kleines Tutorial in der Dokumentation der Arbeit erstellt.

5.3.1.3 Systemstart und System-Shutdown

Das Programm startet man per Doppelklick auf die EXE-Datei. Um es zu beenden, schliesst man das Fenster wieder oder drückt im Menu auf den Exit-Button.

5.3.1.4 Sprachen

Alle Texte, welche auf der Benutzeroberfläche verwendet werden, sind auf Englisch.

5.3.2 Detaillierte Funktionsbeschreibung

Folgende Anforderungen **müssen** umgesetzt sein:

Anforderung_001
Menu: Es soll eine Menu Ansicht geben, welche einen «Play» und «Exit» Button beinhaltet.

Anforderung_002
Spielfigur bewegt sich stetig: Die Spielfigur soll sich ohne jegliche Benutzereingabe stetig und gerade aus fortbewegen.

Anforderung_003
Spielfigur weicht aus: Die Spielfigur wird mit den Pfeiltasten gesteuert. Sobald der Benutzer die Pfeiltaste links oder rechts drückt, soll die Spielfigur in die entsprechende Richtung ausweichen.

Anforderung_004
Hindernisse tauchen auf: Regelmässig tauchen Hindernisse auf, sodass die Spielfigur ausweichen muss.

Anforderung_005
Kollisionscheck: Es muss überprüft werden, ob den Hindernissen korrekt ausgewichen wurde. Wenn die Spielfigur das Hindernis berührt, ist das Spiel vorbei und die Game Over Ansicht erscheint.

Anforderung_006
Game Over: Bei dem Game Over wird das Spiel gestoppt und über das ganze Applikations-GUI den Text «Game Over» angezeigt inklusive eines Restart-Buttons und des Menu-Buttons.

Anforderung_007
Geschwindigkeit erhöhen: Um das Spiel spannend zu gestalten, wird die Schwierigkeit laufend erhöht, bis ein schnelles Tempo erreicht ist. Anschliessend bleibt die Geschwindigkeit dieselbe.

Anforderung_008
Punkte-Zähler: Der Punkte-Zähler wird laufend um eins erhöht. Ebenso wird die Gesamtanzahl des Zählers auf dem GUI anzeigt.

Anforderung_009
Kameraperspektive: Die Kamerasicht sollte sich aus der Perspektive von oben/hinten der Spielfigur befinden.

Zudem sind folgende Anforderungen **wünschenswert**:

Anforderung_010
Highscore Ansicht: Auf dem Menu sollte es eine Highscore Ansicht geben, wo der höchst erreichte Score bis dahin anzeigt.

Anforderung_011
Spielfigur springt: Die Spielfigur sollte per Leertaste ebenfalls über die Hindernisse springen können.

5.3.3 System- / Fehlermeldungen

Bei einem Game Over wird dies auf dem GUI ausgegeben. Ansonsten sind keine Systemmeldungen vorhanden.

5.4 Datenbasis

Der Highscore des Spiels wird in den PlayerPrefs einem lokalen File, welches von Unity automatisch erstellt wird, abgespeichert. Ebenfalls wird der Asset Store von Unity als Datenbasis für die Laufstrecke, Dekorationen und Hindernissen genutzt.

5.5 Externe Schnittstellen

5.5.1 Benutzeroberfläche

Dem Benutzer wird ein GUI anzeigt, indem er zu Beginn im Menu das Spiel starten kann. Die Spielfigur kann er mit der Tastatur bewegen oder das Spiel nach einem Game Over restarten. Ansonsten hat der Benutzer keine Möglichkeit, in der Applikation zu navigieren.

5.6 Leistungsanforderungen

Für diese Applikation werden folgende Mindestleistungsanforderungen vorausgesetzt:

- **OS:** Windows 7 SP1+
- **Grafikkarte:** Mit DX10 (Shader-Modell 4.0) -Fähigkeiten.
- **CPU:** Support des SSE-Befehlssatzes
- **Zubehör:** Eine Tastatur wird benötigt

Ansonsten kann keine einwandfreie Funktionsvielfalt garantiert werden. (Unity, o.J.)

5.7 Support und Service

Bei Fragen darf man sich gerne an die Entwicklerin Céline Hofstetter wenden.

6 Initiierung

6.1 Analyse

Als erstes muss festgelegt werden, wie mein Lösungsansatz aussieht, sodass ich anschliessend meine Arbeitspakete definieren und den Zeitplan erstellen kann. Da das Spiel von Grund auf neu entwickelt wird und ich es nur für mich privat nutze, bin ich mit der Wahl des Lösungsansatzes völlig frei.

Es wird ein 3D Game namens «Road Runner» realisiert. In dem Spiel geht es darum, dass die Spielfigur rennt und man per Tastatureingabe die Spielfigur steuert. Vor der Spielfigur tauchen Hindernisse auf, welchen korrekt ausgewichen werden muss. Schafft die Spielfigur ein Hindernis nicht, ist das Spiel vorbei. Levels gibt es bei diesem Spiel keine. Um die Schwierigkeit trotzdem laufend zu erhöhen, wird es in regelmässigen Abständen ein bisschen schneller. Das heisst, dass die Spielfigur sich immer schneller fortbewegt. Ebenfalls sollte irgendwo auf dem GUI eine Punktzahl ersichtlich sein, sodass sich der Benutzer messen kann.

Die Tests werden anhand der Anforderungen erstellt. Zur Testabnahme müssen alle Tests bestanden sein. Vorgeführt wird schlussendlich das fertige Spiel.

6.1.1 Anforderungen

Aus der Analyse liessen sich folgende Anforderungen bilden.

Folgende Anforderungen **müssen** umgesetzt sein:

Anforderung_001
Menu: Es soll eine Menu Ansicht geben, welche einen «Play» und «Exit» Button beinhaltet.

Anforderung_002
Spielfigur bewegt sich stetig: Die Spielfigur bewegt sich ohne Benutzereingaben stetig gerade aus.

Anforderung_003
Spielfigur weicht aus: Die Spielfigur wird mit den Pfeiltasten gesteuert. Sobald der Benutzer die Pfeiltaste links oder rechts drückt, soll die Spielfigur in die entsprechende Richtung ausweichen.

Anforderung_004
Hindernisse tauchen auf: Regelmässig tauchen Hindernisse auf, sodass die Spielfigur ausweichen muss.

Anforderung_005
Kollisionscheck: Es muss überprüft werden, ob den Hindernissen korrekt ausgewichen wurde. Wenn die Spielfigur das Hindernis berührt, ist das Spiel vorbei und die Game Over Ansicht erscheint.

Anforderung_006
Game Over: Bei dem Game Over wird das Spiel gestoppt und über das ganze Applikations-GUI den Text «Game Over» angezeigt inklusive eines Restart-Buttons und des Menu-Buttons.

Anforderung_007
Geschwindigkeit erhöhen: Die Geschwindigkeit des Spieles wird laufend erhöht, bis ein schnelles Tempo erreicht ist. Anschliessend bleibt die Geschwindigkeit dieselbe.

Anforderung_008
Punkte-Zähler: Der Punkte-Zähler wird laufend um eins erhöht.

Anforderung_009
Kameraperspektive: Die Kamerasicht sollte sich aus der Perspektive von oben/hinten der Spielfigur befinden.

Zudem sind folgende Anforderungen **wünschenswert**:

Anforderung_010
Highscore Ansicht: Im Menu sollte der Highscore ersichtlich sein.

Anforderung_011
Spielfigur springt: Die Spielfigur sollte per Leertaste ebenfalls über die Hindernisse springen können.

6.1.2 Lösungskonzept

6.1.2.1 Game Engine

Für meine Recherchen nutzte ich hauptsächlich das Internet. Ich suchte nach verschiedenen Lösungsansätzen, um ein 3D Game zu entwickeln. Schlussendlich notierte ich mir die drei bekanntesten Game Engins. Da es sich um ein eher simples 3D Game handelt, wäre es bestimmt mit fast jeder Game Engine möglich, dies zu entwickeln. Jedoch spielte da noch ein persönlicher Aspekt mit, dass ich sehr gerne Unity oder eine andere bekannte Game Engine ausprobieren möchte.

Um einen Entscheid fällen zu können legte ich für das Auswahlverfahren diverse notwendige und wichtige Entscheidungskriterien fest. Zusätzlich dazu noch zwei für mich wichtige Punkte. Zum einen das persönliche Interesse und zum anderen, ob es später mit wenigen Anpassungen auf Mobile Plattformen ausgebaut werden könnte.

Abbildung 4 – Unreal Engine Logo (Wikimedia, o.J.)

Unreal Engine

Die Unreal Engine wurde erstmals 1998 von Epic Games veröffentlicht und wird für Konsolen- und Computerspiele (2D/3D) eingesetzt. Nach der ersten Veröffentlichung folgten drei weitere Versionen. Durch die lange Existenz der Engine ist die Community ziemlich gewachsen und gut in der Qualität. Die neuste Version ist kostenlos nutzbar, jedoch fallen für kommerzielle Produkte nach den ersten 3’000 USD Bruttoumsatz pro Spiel und pro Quartal 5% Gebühr an. Für die nichtkommerzielle Nutzung ist die Lizenz kostenlos.

Die Engine wurde mit C++ entwickelt und die Skriptsprache ist UnrealScript. Als Entwicklungsumgebung kann auf Windows Visual Studio verwendet werden. Folgende Betriebssysteme werden als Zielplattform unterstützt:

Windows, Linux, SteamOS, macOS, PlayStation, Xbox, Android, Dreamcast, iOS, Nintendo Switch, webOS, tvOS, WebGL, HTML5, WebAssembly, Google Native Client und Windows Mixed Reality. (Wikipedia, o.J.)

Unity

Die Unity Game Engine erschien im Jahr 2005 von Unity Technologies. Damit werden nebst 2D und 3D Computerspielen auch Spiele für mobile Geräte, Konsolen und Webbrowser entwickelt. Die Engine wird laufend weiterentwickelt und pro Jahr folgen mehrere Updates der Versionen. Durch die grosse Verbreitung von Unity ist die Community enorm gewachsen und gut dokumentiert.

Abbildung 5 – Unity Logo (Wikimedia, o.J.)

Es gibt drei verfügbare Lizenzmodelle. Eines für Hobbyentwickler für 20€ pro Monat, eines für Teams und Freiberufler für 115€ pro Monat und eines für Anfänger welches kostenfrei nutzbar ist. Das Anfängerpacket ist jedoch nur kostenlos, solange der Umsatz pro Jahr nicht höher als 100'000$ ist. (Unity, o.J.)

Die Engine wurde mit C++ entwickelt und als Skriptsprache wird hauptsächlich C# verwendet. Als Entwicklungsplattform kann Windows, Mac oder Linux verwendet werden. Visual Studio wird standardmässig als Entwicklungsumgebung genutzt. Folgende Zielplattform werden unterstützt:
Windows, Mac, Linux, PS4, Xbox One, Wii U, Nintendo Switch, tvOS, HoloLens, iOS, Android, PSV und Webbrowser über WebGL oder Plug-in: Unity Web Player.

Mit der Unity Engine wurden bekannte Games wie Assassins Creed: Identity und Temple Run Trilogy entwickelt. (Wikipedia, o.J.)

Cry Engine

Abbildung 6 – CryEngine Logo (Wikimedia, o.J.)

Die CryEngine wurde im Jahr 2002 von Crytek veröffentlicht. Darauf folgte 2004 das erste mit dieser Spiel-Engine entwickelte und sehr bekannte Spiel namens Far Cry. Auf die erste Version folgten vier weitere Versionen. Wie auch bei den anderen zwei Spiel-Engins ist die Community über die Jahre hinweg ziemlich gewachsen. Das Lizenzmodell der CryEngine wurde schon mehrfach komplett umgestellt. Aktuell ist die Spiel-Engine gratis bis zu 5'000 $ pro Jahr und pro Game. Falls dieser Jahresumsatz übertroffen wird, werden 5% Lizenzgebühr berechnet. (CryEngine, o.J.)

Die CryEngine wurde mit C++ entwickelt und als Skriptsprache kann C#, C++ oder Lua verwendet werden. Zur Entwicklung mit der CryEngine wird Windows 7 als Minimum-Betriebssystem vorausgesetzt. Folgende Zielplattform werden unterstützt: Windows PC, Oculus Rift, Xbox One und Playstation 4. (Wikipedia, o.J.)

6.1.2.2 Grafiktool

Ebenfalls bauche ich ein Tool, um die Grafiken zu erstellen. Ich entschied mich aus zeitlichen Gründen, nur die Spielfigur von Grund auf selbst zu gestalten und die Hindernisse, Dekorationen und Laufstrecke aus einem Asset Store zu entnehmen. Für den Entscheid, mit welcher Software die Spielfigur erstellt wird, habe ich eine zweite Entscheidungsmatrix mit drei der beliebtesten Grafik Softwares erstellt.

Maya

Maya ist eine Software zur Erstellung von 3D-Animationen und -Visualisierungen. Sie wird in der Gameentwicklung und in der Film- und Fernsehindustrie eingesetzt. Die Software erschien anfangs 1998 durch das damalige Unternehmen Alias, heute Autodesk. Im Bereich 3D-Modellierung, Computeranimation und Rendering zählt sie zu den bekanntesten und meistgenutzten Softwareprodukten. Somit ist auch die Community sehr gut fortgeschritten.

Abbildung 7 – Maya Logo (Wikimedia, o.J.)

Die aktuelle Version unterstützt Windows, Linux und Mac OS X als Betriebssysteme. Die erstellten Grafiken können einwandfrei in einem Unity Projekt eingebettet werden. (Wikipedia, o.J.)

Nach dem gratis Testmonat kostet die Software pro Jahr 1'505 $. Studenten könnten jedoch drei Jahre von der gratis Version profitieren. (Autodesk, o.J.)

Für Anfänger von Grafiktools wird die Software eher nicht empfohlen. Maya hat einen sehr grossen Funktionsumfang, was der Einstig nicht gerade leicht macht.

Blender

Abbildung 8 – Blender Logo (Wikimedia, o.J.)

Blender ist eine 3D-Grafiksoftware zum Modellieren, animieren und texturieren. Die Software wurde 1994 von der Blender Foundation auf den Markt gebracht. Seit der Veröffentlichung folgten zahlreiche Versionen mit Verbesserungen. Als eines der bekanntesten 3D-Grafiktool hat sich eine dichte Community über die Jahre aufgebaut.

Windows, Linux, MacOS, Solaris, FreeBSD und IRIX werden als Betriebssysteme mit der aktuellen Version unterstützt. Die generierten Grafiken können einfach in ein Unity Projekt importiert werden.

Blender wurde in den Anfangsjahren unter eine freie Softwarelizenz (GPL – General Public License) gestellt. Somit kann sie kostenfrei genutzt, studiert, geändert oder sogar verbreitet (kopiert) werden.

Blender wird öfters auch von Anfänger von Grafiktools verwendet, da es ein bisschen beschränkter im Funktionsumfang ist, als zum Beispiel Maya. Trotzdem kann es sehr viel und ist, als einer der wenigen oder sogar einzigen Software in diesem Bereich, kostenfrei. (Wikipedia, o.J.)

ZBrush

ZBrush ist eine Grafiksoftware, welche für 2D und 3D Grafiken verwendet wird. Pixologic brachte die erste Version von ZBrush 1999 heraus. Trotz ähnlicher Lebenslänge wie Maya oder Blender ist die Community nicht ganz so ausgebaut. (Wikipedia, o.J.)

Abbildung 9 – ZBrush Logo (Zendesk, o.J.)

Die aktuelle Version der Software kostet für einen Einzelbenutzer einmalig 895 $ und ist mit Windows und Mac OS X kompatibel. Ebenfalls harmonieren die erstellten Grafiken mit der Unity-Game-Engine. (Pixologic, o.J.)

Anfänger von Grafiktools benützen meist zu Beginn andere Tools, welche vollständig kostenlos sind oder einen gratis Testmonat anbieten.

6.1.3 Entscheid

Schlussendlich sah meine Entscheidungsmatrix für die Game Engine wie folgt aus:

Kriterien	Gewichtung	Unreal Engine		Unity 3D		Cry Engine	
		Bewertung	Total	Bewertung	Total	Bewertung	Total
Programmiersprache	20	1	20	3	60	3	60
Entwicklungsumgebung Windows fähig	20	3	60	3	60	3	60
Zielplattform Windows	20	3	60	3	60	3	60
3D untersützt	15	3	45	3	45	3	45
Community	10	3	30	3	30	3	30
Zielplattform ausbaufähig (Mobile)	5	3	15	3	15	1	5
Kosten	5	3	15	3	15	3	15
Persöndliches Intresse	5	2	10	3	15	1	5
Total	**100**		**255**		**300**		**280**

Abbildung 10 – Entscheidungsmatrix für die Game Engine

Jedes Kriterium wurde von 1-3 bewertet (1 = nicht gut, 2 = neutral, 3 = gut).

Bei der Unreal Engine ist die Skript Sprache UnrealScript, da ich diese Programmiersprache nicht beherrsche, musste ich diesen Punkt mit einer 1 bewerten. Dazu musste ich bei der Unreal Engine einen Punkt bei dem persönlichen Interesse abziehen, da mir diese Engine weniger zusage als Unity.

Die Cry Engine hatte bei meinen persönlichen Punkten etwas einzubüssen. Mit dieser Engine ist es leider nicht möglich, die Software für Mobile-Endgeräte zu generieren, somit entsprach sie ebenfalls nicht ganz meinen Interessen.

Die maximale und somit auch die beste Bewertung erhielt Unity 3D. In allen Punkten trifft sie voll und ganz zu, weshalb ich mich auch für Unity 3D entschied.

Anschliessend schaute ich mehrere Tutorials an, wie man ein 3D Game mit Unity entwickelt, sodass ich mir einen Überblick verschaffen konnte, was dies für die einzelnen Arbeitspakete an Aufwand bedeutet. Ebenfalls erhielt ich dabei einen ersten Einblick, wie ich beim Programmieren vorgehen könnte.

Meine Entscheidungsmatrix für die Software, um die Spielfigur zu erstellen, sah folgendermassen aus:

Kriterien	Gewichtung	Maya		Blender		ZBrush	
		Bewertung	Total	Bewertung	Total	Bewertung	Total
Zielplattform Unity	20	3	60	3	60	3	60
Software Windows fähig	20	3	60	3	60	3	60
3D untersützt	20	3	60	3	60	3	60
Animation erstellen	20	3	60	3	60	3	60
Nutzung für Beginner	10	2	20	3	30	2	20
Kosten	5	3	15	3	15	1	5
Community	5	3	15	3	15	2	10
Total	**100**		**290**		**300**		**275**

Abbildung 11 – Entscheidungsmatrix für die Grafiksoftware

Die Kriterien wurden anhand der Informationen aus dem Internet gleich wie oben bewertet. Ganz knapp lag die finale Entscheidung bei der Blender-Software. Jedoch hätte man es sicherlich mit allen drei gut machen können. Mir war es jedoch wichtig, dass die Nutzung möglichst simpel ist, da Grafiken erstellen noch Neuland für mich ist.

6.2 Arbeitspakete

Ich erstellte mir anhand der Aufgabenstellung die Arbeitspakete und teilte sie den verschiedenen Projektphasen zu.

Initiierungsphase			
Bezeichnung	**Beschreibung**	**SOLL (h)**	**IST (h)**
Aufgabe analysieren	Aufgabenstellung durchlesen und eine kurze Analyse davon erstellen. Anschliessend ermitteln, wie die Aufgabe am besten umgesetzt werden kann.	5	5
Lösungskonzept ausarbeiten	Das Lösungskonzept ist mit Minimum Varianten und Entscheidungskriterien auszuarbeiten.	8	9
Arbeitspakete erarbeiten	Anhand der Aufgabenstellung die Arbeitspakete erstellen und den verschiedenen Projektphasen zuweisen.	1	1
SOLL-Zeitplan erstellen	Der SOLL-Zeitplan wird anhand der Arbeitspakete und Meilensteinen erstellt. Die Pakete werden auf die Kalenderwochen bis zur Abgabe verteilt und die benötigte Zeit dafür wird geschätzt.	2	1

Definitionsphase			
Bezeichnung	**Beschreibung**	**SOLL (h)**	**IST (h)**
Software Architektur Design	Eine Software Architektur wird erstellt. Darunter Use Cases, Package-Diagramm, Klassendiagramm und ein Activity Diagramm.	8	10
Benutzeroberfläche Design	Das Design für die Benutzeroberfläche wird entworfen.	4	2
Testkonzept und Testfälle	Das Testkonzept wird erstellt und die Testfälle anhand der Analyse der Aufgabenstellung erstellt.	6	5

Realisierungsphase			
Bezeichnung	**Beschreibung**	**SOLL (h)**	**IST (h)**
Spielfigur Grafik	Die Grafik für die Spielfigur wird erstellt.	10	19
Laufstrecke Grafik	Die Grafik für die Laufstrecke wird erstellt.	3	3
Hindernisse Grafik	Die Grafik für die Hindernisse wird erstellt.	3	3
Spielfigur «läuft» stetig	Die erstellte Spielfigur soll sich laufend fortbewegen.	4	4
Spielfigur links/rechts bewegen	Es soll implementiert werden, dass wen der Benutzer eine Pfeiltaste drückt, die Spielfigur sich nach links/rechts bewegt.	8	3
Kamera Sicht	Die Kamerasicht, wie der Benutzer des Spiels das Spiel sieht, soll eingestellt werden.	4	3
Diverse Laufstrecken-Abschnitte erstellen inkl. Hindernisse	Um das Spiel spannender zu gestalten, sollen weitere Laufstreckengrafiken erstellt werden.	14	16
Laufstrecken-Abschnitte wiederholen	Um die Spielfigur unendlich laufen zu lassen, sollen sich die Laufstrecken-Abschnitte laufend wiederholen.	6	5
Überprüfung, ob dem Hindernis korrekt ausgewichen wurde	Es soll überprüft werden, ob einem Hindernis korrekt ausgewichen wurde. Falls eines nicht korrekt bewältigt wurde, soll es ein Game Over ausgeben.	7	8
Menu	Für das Spiel soll ein kleines und simples Menu erstellt werden.	10	8
Score Ansicht	Es soll ein Score angezeigt werden, welcher laufend hoch zählt.	4	3
Spielfigur Geschwindigkeit erhöhen	Um die Schwierigkeit zu erhöhen, soll die Spielgeschwindigkeit in regelmässigem Abstand laufend erhöht werden.	6	6
Game Over Ansicht	Eine Game Over Ansicht wird implementiert, um dem Spieler anzuzeigen, wann das Spiel vorbei ist.	8	7
Highscore Ansicht in Menu	Im Menu soll es den eigenen Highscores anzeigen.	8	2
Spielfigur springt per Tastatureingabe	Es soll implementiert werden, dass wen der Benutzer eine Taste drückt, die Spielfigur einmal in die Luft springt.	12	15
Code Review	Der Code wird nochmals auf Fehler, Code-Styling und Kommentare überprüft und evtl. ergänzt.	20	20

Testphase			
Bezeichnung	**Beschreibung**	**SOLL (h)**	**IST (h)**
Re-/ Tests durchführen	Anhand der Testplanung werden die Tests erstellt und durchgeführt. Die fehlgeschlagenen Testfälle werden nochmals getestet.	10	8
Fehlerbehebung	Falls Fehler auftauchen, werden diese behoben.	15	3

Dokumentation			
Bezeichnung	**Beschreibung**	**SOLL (h)**	**IST (h)**
Anlegen und Grobstruktur erstellen	Das Dokument wird mit Inhaltsverzeichnis als Grobstruktur erstellt.	2	2
Dokumentieren	Das Projekt wird laufend dokumentiert.	100	99

Abgabe			
Bezeichnung	**Beschreibung**	**SOLL (h)**	**IST (h)**
Reserve + Abgabe (Bericht drucken & binden)	Die Dokumentation wird gedruckt, gebunden und ist abgabebereit. Falls es irgendwo eine Zeitverzögerung geben sollte, ist hier noch ein bisschen Reserve-Zeit eingeplant.	2	1

Präsentation			
Bezeichnung	**Beschreibung**	**SOLL (h)**	**IST (h)**
Präsentation vorbereiten	Die gesamte Präsentation wird vorbereitet und geübt.	15	-

Tabelle 1 – Arbeitspakete

6.3 Zielsetzungen und Meilensteine

Durch meine gewählte Vorgehensweise entschied ich mich, am Ende jeder Projektphase einen Meilenstein zu setzen. Dieser muss zuerst erreicht werden, um weiter zu arbeiten.

Somit haben sich folgende Meilensteine ergeben:

Nr.	Bezeichnung	Beschreibung	Zeitpunkt
1	Initiierungsphase abgeschlossen	Die Aufgabe ist analysiert, die Arbeitspakete erarbeitet, die Meilensteine definiert und der SOLL-Zeitplan erstellt.	KW 43, 28.10.2018
2	Definitionsphase abgeschlossen	Das komplette Design ist entworfen, der Programmablauf steht und das Testkonzept sowie die Testfälle sind erstellt.	KW 44, 04.11.2018
3	Realisierungsphase abgeschlossen	Alle Anforderungen sind vollständig umgesetzt.	KW 01, 06.01.2019
4	Testphase abgeschlossen	Die neue Funktion wurde getestet, das Ergebnis dokumentiert, Fehler behoben und erneut getestet.	KW 04, 27.01.2019
5	Dokumentation abgeschlossen	Die Dokumentation ist vollständig und bereit zum Druck.	KW 08, 24.02.2019
6	Projekt abgegeben	Projekt Abgabe.	KW 09, 03.03.2019

Tabelle 2 – Meilensteine

6.4 Zeitplan

Zeitplan - Diplomarbeit			Initiierung			Definition	Realisierung									Testing			Dokumentation				Abgabe	Präsentation
Arbeitspaket	**Aufwand in h**		KW 41	KW 42	KW 43	KW 44	KW 45	KW 46	KW 47	KW 48	KW 49	KW 50	KW 51	KW 52	KW 1	KW 2	KW 3	KW 4	KW 5	KW 6	KW 7	KW 8	KW 9	KW 10 - KW 13
Initiierung	**SOLL**	**16**																						
	IST	**16**																						
Analyse	SOLL	5	5																					
	IST	5	5																					
Lösungskonzept ausarbeiten	SOLL	8		4	4																			
	IST	9		5	4																			
Arbeispakete erstellen	SOLL	1	1																					
	IST	1	1																					
SOLL-Zeitplan erstellen	SOLL	2	2																					
	IST	1	1																					
Meilenstein 1: Initiierungsphase abgeschlossen					M1: 28.10.2018																			
Definition	**SOLL**	**18**																						
	IST	**17**																						
Software Architektur Design	SOLL	8			6	2																		
	IST	10			6	4																		
Benutzeroberfläche Design	SOLL	4				4																		
	IST	2				2																		
Testkonzept und Testfälle	SOLL	6				6																		
	IST	5				5																		
Meilenstein 2: Definitionsphase abgeschlossen						M2: 04.11.2018																		
Realisierung	**SOLL**	**127**																						
	IST	**125**																						
Spielfigur Grafik	SOLL	10					10																	
	IST	19					10	9																
Laufstrecke Grafik	SOLL	3						3																
	IST	3						0		3														
Hindernisse Grafik	SOLL	3								3														
	IST	3								3														
Spielfigur «läuft» stetig	SOLL	4						4																
	IST	4						4																
Spielfigur links/rechts bewegen	SOLL	8								8														
	IST	3								3														
Kamera Sicht	SOLL	4								4														
	IST	3								3														
Abschnitte erstellen inkl. Hindernisse	SOLL	14								14														
	IST	16								16														
Laufstrecken-Abschnitte wiederholen	SOLL	6								6														
	IST	5								5														
Hindernis korrekt ausgewichen wurde	SOLL	7								7														
	IST	8								8														
Menu	SOLL	10									10													
	IST	8									8													
Score Ansicht	SOLL	4									4													
	IST	3									3													
Spielfigur Geschwindigkeit erhöhen	SOLL	6										6												
	IST	6										6												
Game Over Ansicht	SOLL	8										4	4											
	IST	7										4	3											
Highscore Ansicht in Menu	SOLL	8											5	3										
	IST	2											2	0										
Spielfigur springt per Tastatureingabe	SOLL	12												12										
	IST	15												15										
Code Review	SOLL	20												5	15									
	IST	20												5	15									
Meilenstein 3: Realisierungsphase abgeschlossen															M3: 06.01.2019									
Testing	**SOLL**	**25**																						
	IST	**11**																						
Tests durchführen	SOLL	10														5	3	2						
	IST	8														4	3	1						
Fehlerbehebung	SOLL	15														8	4	3						
	IST	3														2	1	0						
Meilenstein 4: Testphase abgeschlossen																		M4: 27.01.2019						
Dokumentation	**SOLL**	**102**																						
	IST	**101**																						
Anlegen und Grobstruktur erstellen	SOLL	2	2																					
	IST	2	2																					
Dokumentieren	SOLL	100	4	8	5	4	5	5		2	2	2	2	2	5	1	3	20	10	8	6	6		
	IST	99	4	8	5	4	5	5		3	3	2	2	2	5	2	4	18	10	8	6	3		
Meilenstein 5: Dokumentation abgeschlossen																						M5: 24.02.2019		
Abgabe	**SOLL**	**2**																						
	IST	**1**																						
Reserve + Abgabe (Bericht drucken & binden)	SOLL	2																					2	
	IST	1																					1	
Meilenstein 6: Projekt abgegeben																							M6: 03.03.2019	
Präsentation vorbereiten	**SOLL**	**15**																						15
	IST	**15**																						*15*
Gesamt	**SOLL**	**305**	14	12	15	16	15	12	0	44	16	12	11	22	20	14	10	25	10	8	6	6	2	15
	IST	**286**	13	13	15	15	15	18	0	44	14	12	7	22	20	8	8	19	10	8	6	3	1	15

Hinweis: Für eine bessere Lesbarkeit wird der Zeitplan elektronisch abgegeben. (Céline Hofstetter, 2015)

7 Definition

7.1 Software Architektur Design

7.1.1 Use Cases

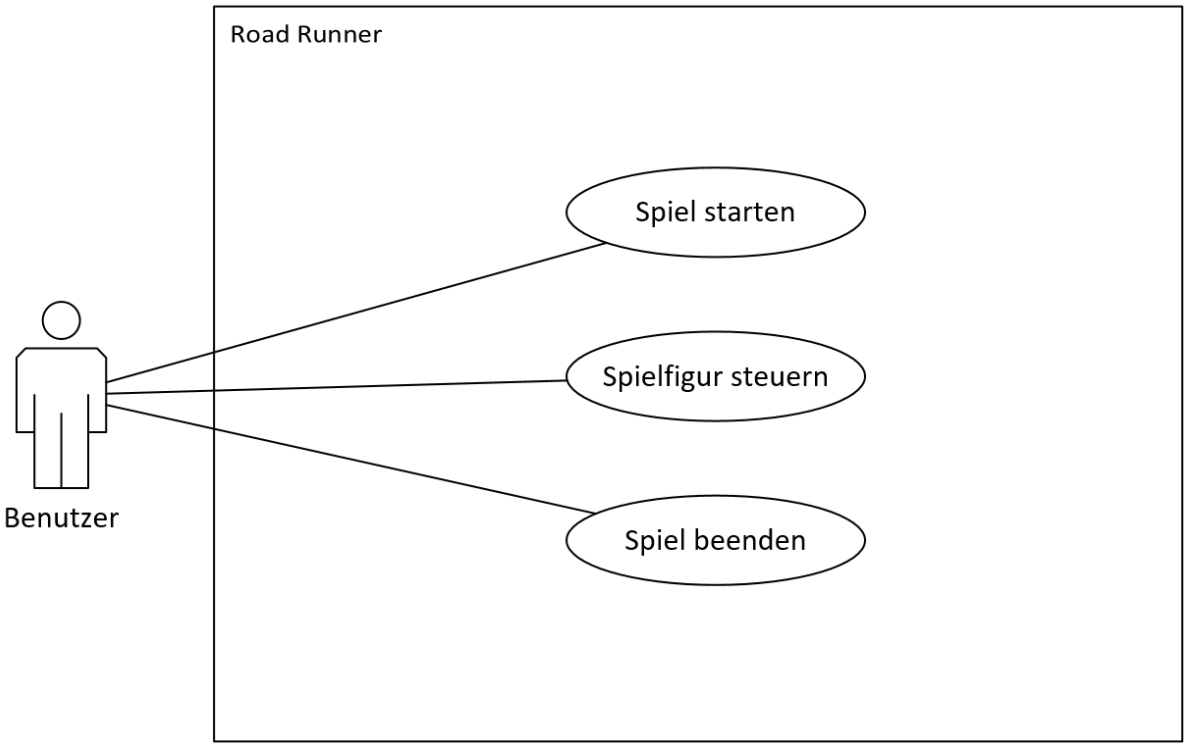

Abbildung 12 – Use Cases

7.1.1.1 Use Case Beschreibung

	Spiel starten
Beschreibung	Der Benutzer möchte das Spiel «Road Runner» starten.
Beteiligte Akteure	Der Benutzer
Vorbedingung	• Applikation ist gestartet
Input	Play-Button wird gedrückt.
Output/Ergebnis	Das Spiel startet und die Spielfigur läuft stetig weiter, bis sie mit einem Hindernis kollidiert.
Standardablauf	1. Benutzer drückt auf den Play-Button 2. Spiel wird gestartet 3. Spielfigur läuft stetig geradeaus und Score wird hochgezählt
Alternativablauf	-

	Spielfigur steuern
Beschreibung	Der Benutzer möchte die Spielfigur steuern.
Beteiligte Akteure	Der Benutzer
Vorbedingung	• Spiel ist bereits gestartet
Input	Als Input werden die Pfeiltasten links/rechts oder die Leertaste verwendet.
Output/Ergebnis	Die Spielfigur bewegt sich in die gewünschte Richtung. Während die Spielfigur springt, reagiert die Spielfigur auf keine weiteren Benutzereingaben.
Standardablauf	1) Die Spielfigur läuft geradeaus 2) Der Benutzer drückt die a) linke Pfeiltaste b) rechte Pfeiltaste c) Leertaste 3) Falls sich die Spielfigur noch am Boden und a) nicht am linken Rand befindet → weiter bei Punkt 4, sonst zurück zu Punkt 1 b) nicht am rechten Rand befindet → weiter bei Punkt 4, sonst zurück zu Punkt 1 4) Die Spielfigur a) bewegt sich nach links b) bewegt sich nach rechts c) springt in die Luft 5) Weiter bei Punkt 1
Alternativablauf	1) Die Spielfigur läuft geradeaus 2) Spielfigur kollidiert mit einem Hindernis 3) Game Over Ansicht erscheint

	Spiel beenden
Beschreibung	Der Benutzer möchte das Spiel beenden.
Beteiligte Akteure	Der Benutzer
Vorbedingung	• Applikation ist gestartet
Input	Exit-Button wird gedrückt.
Output/Ergebnis	Die Applikation wird beendet.
Standardablauf	1. Benutzer drückt auf den Exit-Button 2. Applikation wird beendet
Alternativablauf	-

Tabelle 3 – Use Case Beschreibung

7.1.2 System Architektur

Ein Unity-Game besteht immer aus folgenden Hauptelementen.

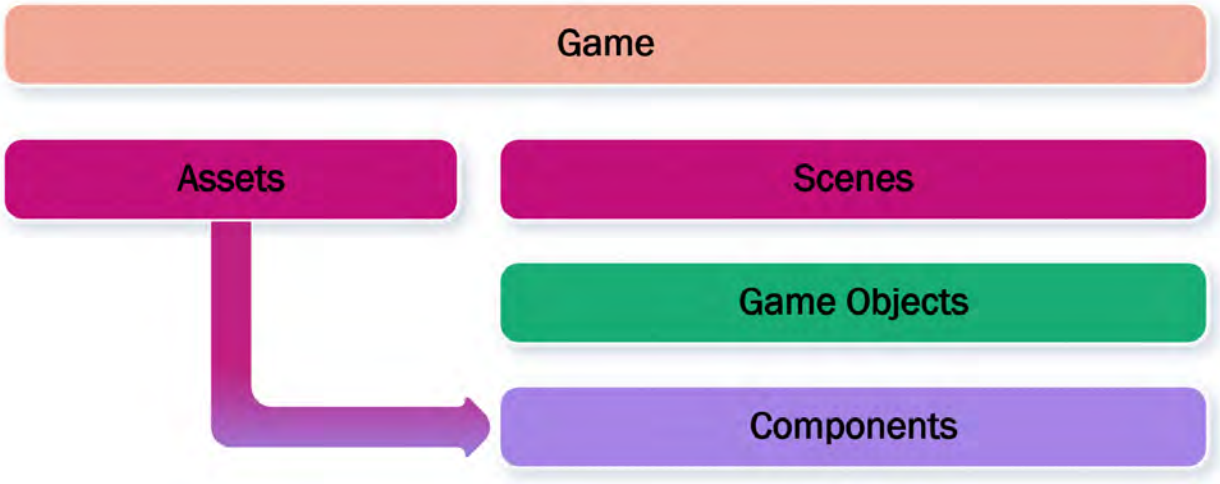

Abbildung 13 – High Level Unity Game Architektur

Assets:

Ein Asset ist ein Element, welches im Spiel verwendet werden kann. Ein Asset kann vom Asset Store von Unity kommen, wie meine Hindernisse oder von einem externen Tool, z.B. wie meine Spielfigur von Blender. Manche Assets, wie z.B. der Animator-Controller können auch direkt in Unity erstellt werden.

Scenes:

Ein Unity Game besteht aus ein oder mehreren Scenes. In jeder Scene kann die entsprechende Umgebung, mit Hindernissen und Dekorationen (sogenannten Game Objects) erstellt werden.

Game Objects:

Die Game Objects sind die fundamentalen Objekte in Unity, welche Charakteren und Gegenstände in den Scenes darstellen. Sie dienen hauptsächlich als Container für die verschiedenen Komponenten und können selbst nicht viel. Standardmässig verfügt jedes Game Object über eine Transformationskomponente. Diese definiert die Position, Drehung und Skalierung eines Objektes. Ohne diese Angaben kann das Objekt nicht erstellt werden.

Components:

Die Komponenten sind die wichtigsten Bestandteile eines Game Objects, denn sie bestimmen ihre Eigenschaften. Unity deckt mit ihren vielen verschiedenen integrierten Komponenten schon sehr viel ab, man könnte aber auch eigene erstellen. Die Kombination aus ein oder mehreren Komponenten definieren das Aussehen und Verhalten eines Game Objects. Per Skript Komponente werden auch eigene Skripte einem Objekt zugewiesen, um zum Beispiel die Spielfigur laufend fortzubewegen.

7.1.3 Skript Lifecycle

Der Lifecycle dient später als kleine Programmierhilfe für die eigenen Skripte um zu wissen, in welcher Reihenfolge die Methoden durchlaufen werden. Wichtig ist zu entnehmen, dass die physikalischen Events unabhängig der Framerate mittels Timer aufgerufen werden. Dazu gehört auch die Kollisionsmethode.

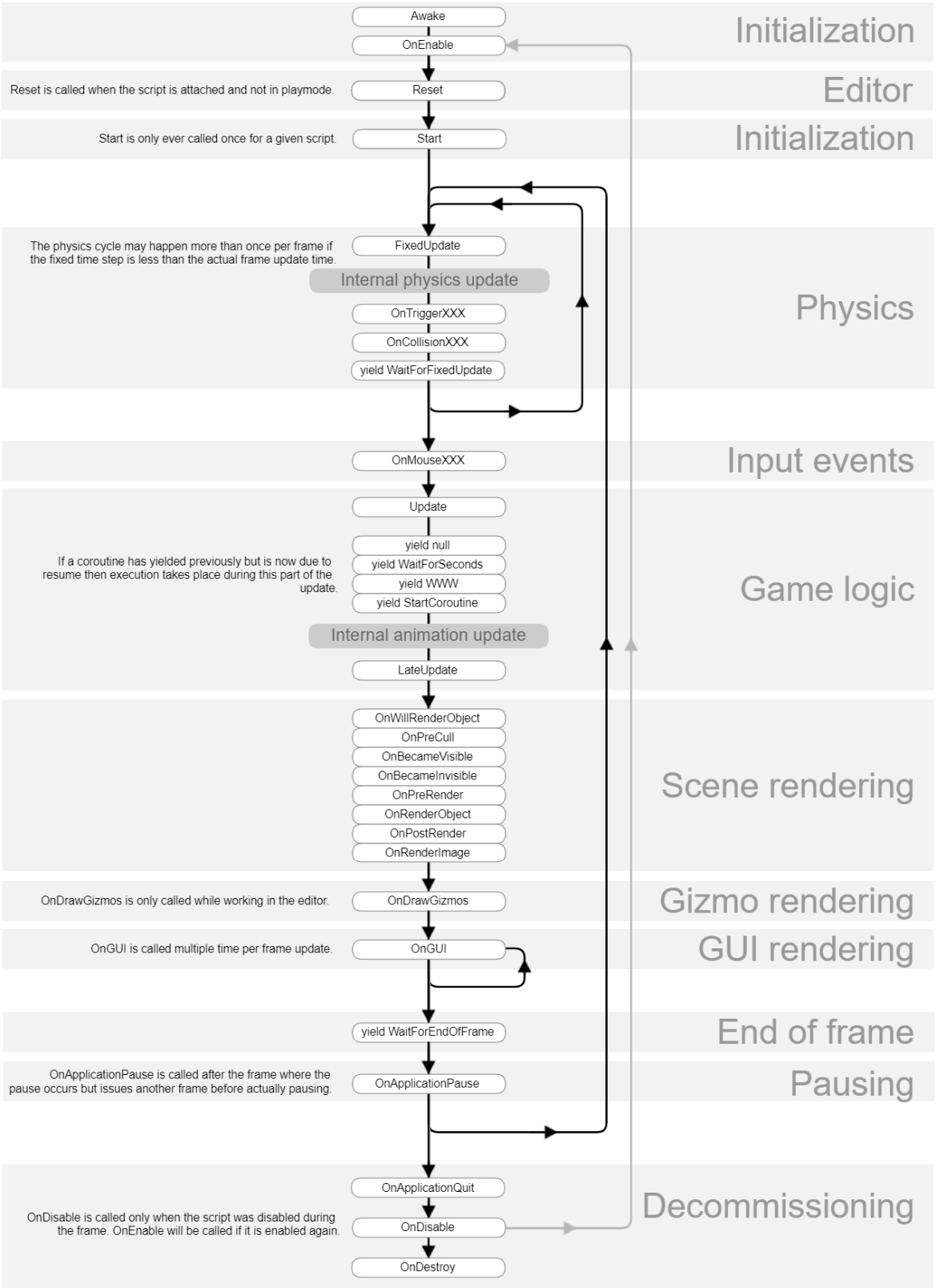

Abbildung 14 – Unity Script Lifecycle Flowchart (Unity, o.J.)

7.1.4 Package Diagramm

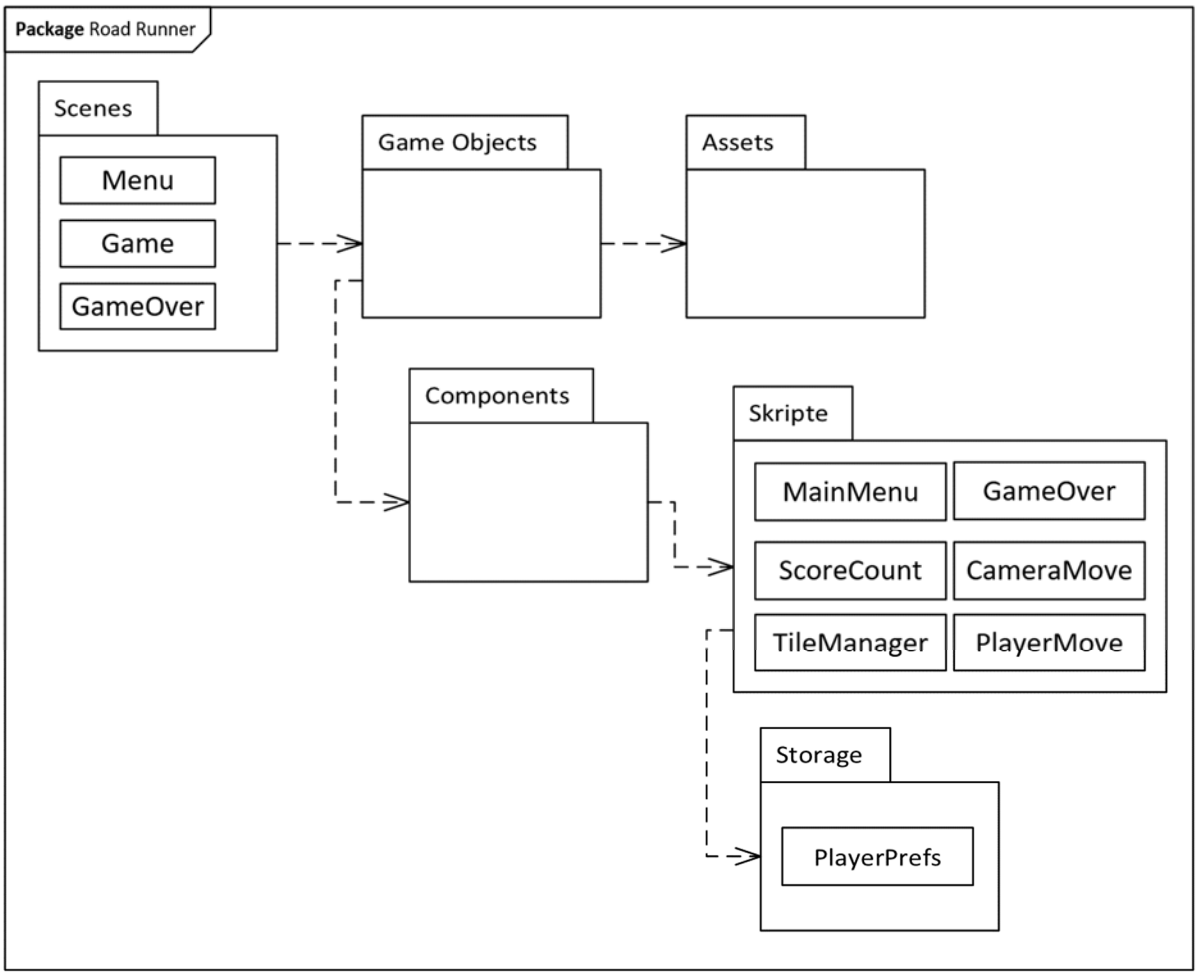

Abbildung 15 – Package Diagramm

Das Package Diagramm besteht aus dem klassischen 3-Schichten Modell.

Scenes (Präsentation)

In dieser Schicht, sind alle GUI Ansichten untergebracht. Den verschiedenen Scenes sind diverse Game Objects und Assets zugewiesen.

Components (Business Logik)

Die Kombination aus ein oder mehreren Komponenten definieren das Aussehen und Verhalten eines Game Objects. Eine Components-Variante ist die Skript Komponente.

Skripte (Business Logik)

Den Skript Komponenten werden die erstellten Skripte zugewiesen, um die Game Objects zu steuern.

Storage (Modell)

In dieser Schicht ist die ganze Datenhaltung untergebracht. Von den Skripten wird auf das automatisch von Unity generierte File PlayerPrefs zugegriffen.

7.1.5 Klassendiagramm

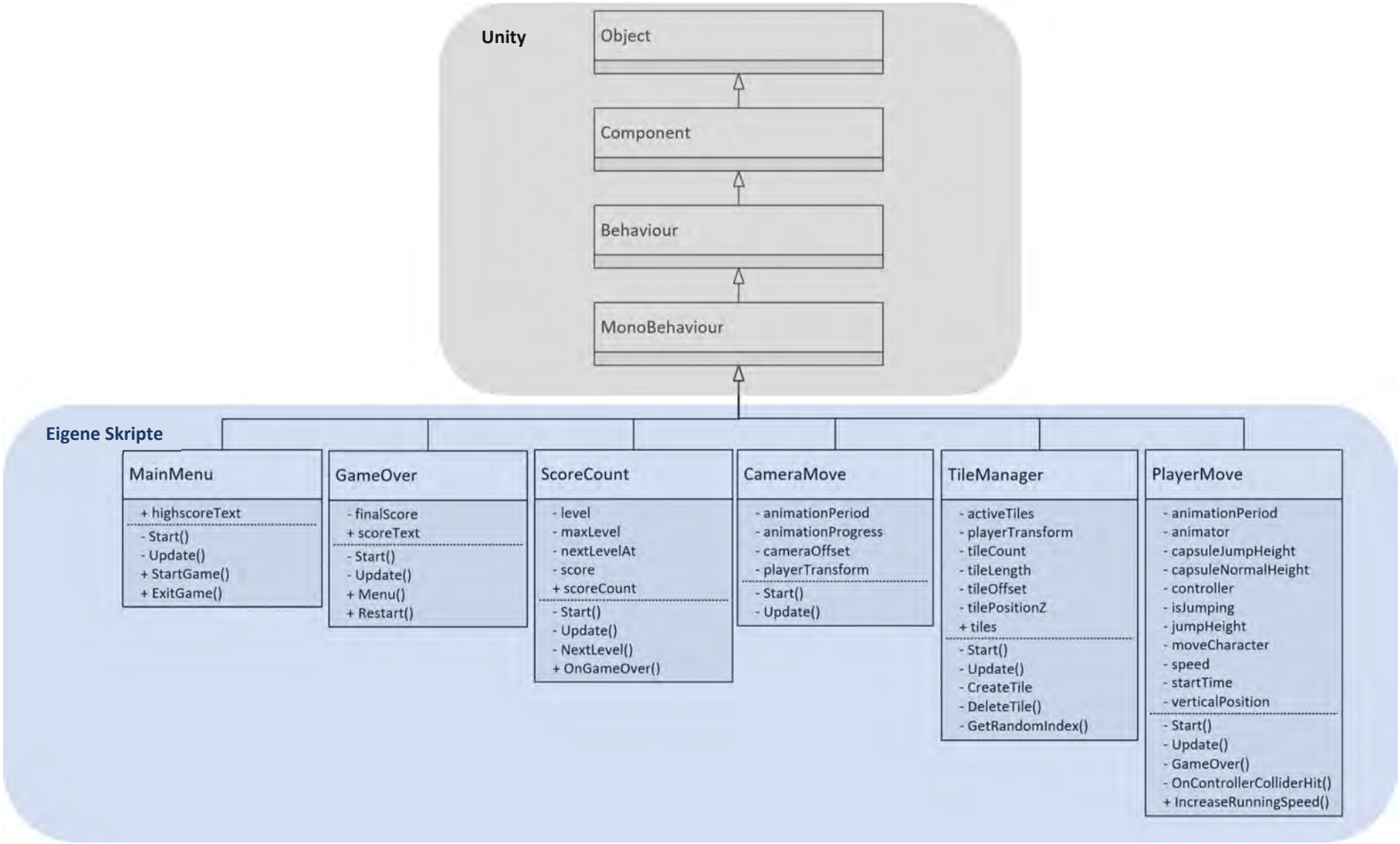

Abbildung 16 – Klassendiagramm

7.1.6 Konzept / Klassenbeschreibung

Im Spiel wird ein Menu, sowie ein Game Over Screen implementiert. Somit wird es drei Scenes geben, eine für das Menu, eine für die Game Over Ansicht und eine für das Spiel an sich. Für diese drei Scenes werden folgende Skripte erstellt.

Klassenname	Beschreibung
MainMenu	Dieses Skript wird für die Menu-Scene verwendet. Beim Starten der Scene zeigt es den Highscore aus den PlayerPrefs an. Ebenfalls beinhaltet das Skript die StartGame-Methode für den Play-Button, welche die Game Scene lädt und die ExitGame-Methode, um das Spiel zu beenden.
GameOver	Das Skript wird in der GameOver-Scene verwendet. Beim Start der Scene wird überprüft, ob der erreichte Score den Highscore überschreitet oder nicht. Allenfalls wird dieser in den PlayerPrefs ersetzt. Zusätzlich enthält das Skript zwei weitere Funktionen für den Restart-Button und den Menu-Button, um entweder die Game-Scene oder die Menu-Scene zu laden.
ScoreCount	Das Skript ist für den Punktezähler verantwortlich und wird in der Game-Scene verwendet. Der Punktezähler wird pro Sekunde immer um die Levelstufe erhöht. Sobald eine gewisse Score Grenze erreicht ist, wird das Level um eines erhöht, bis das Maximallevel erreicht ist. Wenn ein neues Level erreicht wird, wird über die SetSpeed-Methode von dem PlayerMove-Skript die Geschwindigkeit der Spielfigur erhöht. Beim einem Game Over wird der erreichte Score in den PlayerPrefs gespeichert und die GameOver-Scene geladen.
CameraMove	Das Skript ist für die Kamera-Sicht in der Game-Scene zuständig. Anhand der Position der Spielfigur wird laufend die Position der Kamera berechnet. So ist die Spielfigur für den Benutzer der Software immer von der gleichen Perspektive zu sehen.
TileManager	Das Skript ist für die Laufstreckenabschnitte in der Game-Scene verantwortlich. Dem Skript werden alle gewünschten Laufstreckenabschnitte mitgegeben. Es zeigt stetig sieben Abschnitte an. Da die Spielfigur stetig läuft, werden in der Ferne laufend Abschnitte hinzugefügt und diejenigen hinter der Spielfigur immer wieder gelöscht. Welcher Abschnitt hinzugefügt wird, wird mittels Random-Methode entschieden.
PlayerMove	Das Skript steuert in der Game-Scene die Spielfigur. Darin wird die Animation angepasst, je nachdem ob die Spielfigur rennt oder springt. Ebenfalls wird gesteuert, wie schnell und in welche Richtung sich die Spielfigur fortbewegt. Sobald die Spielfigur mit einem Hindernis kollidiert, wird dies abgefangen und die GameOver-Scene geladen.

Tabelle 4 – Klassenbeschreibung

7.1.7 Activity Diagramm

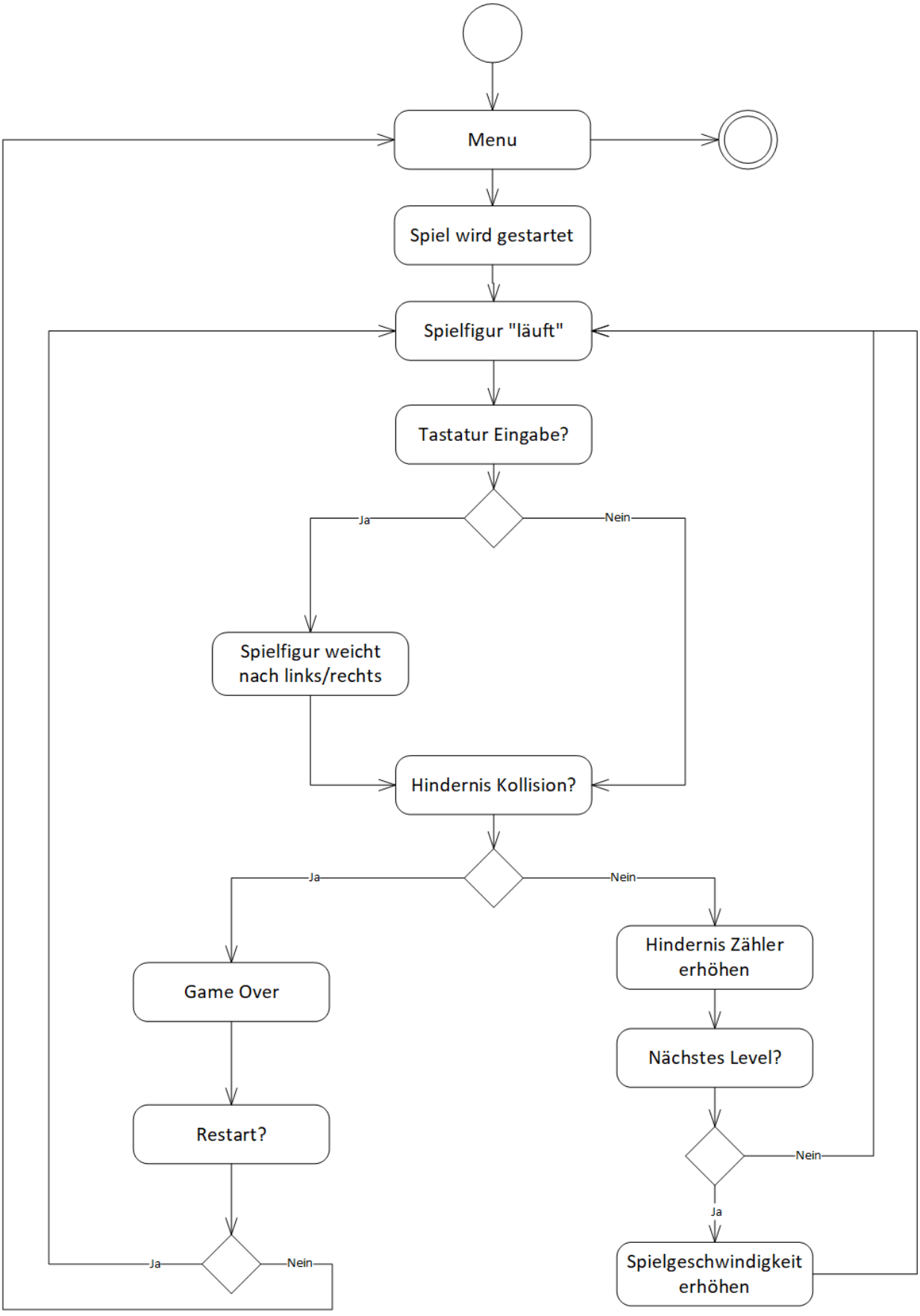

Abbildung 17 – Activity Diagramm, um den gesamten Ablauf des Spiels aufzuzeigen

7.2 Benutzeroberfläche Design

Da ich mit dieser Arbeit mein erstes 3D Game mit einer Game Engine entwickle, geht es zu Beginn erstmal darum, alle Funktionalitäten und Anforderungen umzusetzen. Dafür erstelle ich ein Entwurf für ein Design, welches später noch detaillierter ausgebaut werden könnte.

Generell wird die Umgebung eher düster gehalten und die Spielfigur wird als Ironman realisiert. Die Laufstrecke ist eine Strasse und als Hindernisse werden diverse Strassensperrungen verwendet.

Menu:

Abbildung 18 – Benutzeroberflächen Design - Menu

Standardzustand:

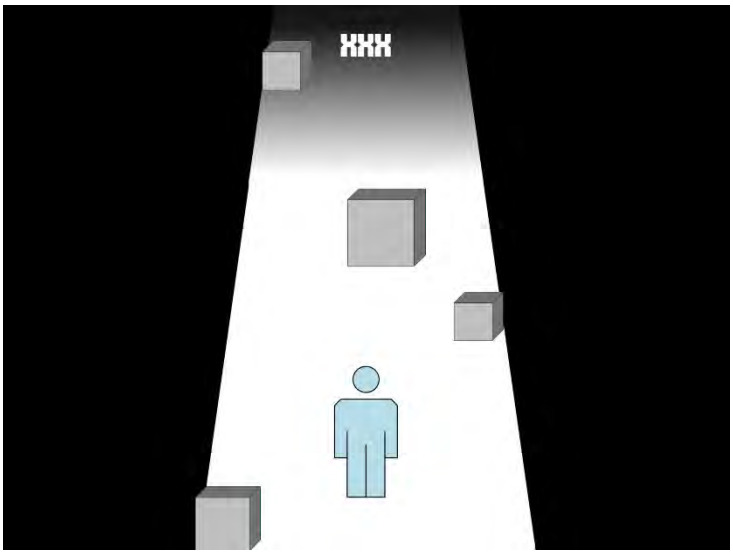

Abbildung 19 – Benutzeroberflächen Design - Standardzustand

Hindernis ausweichen:

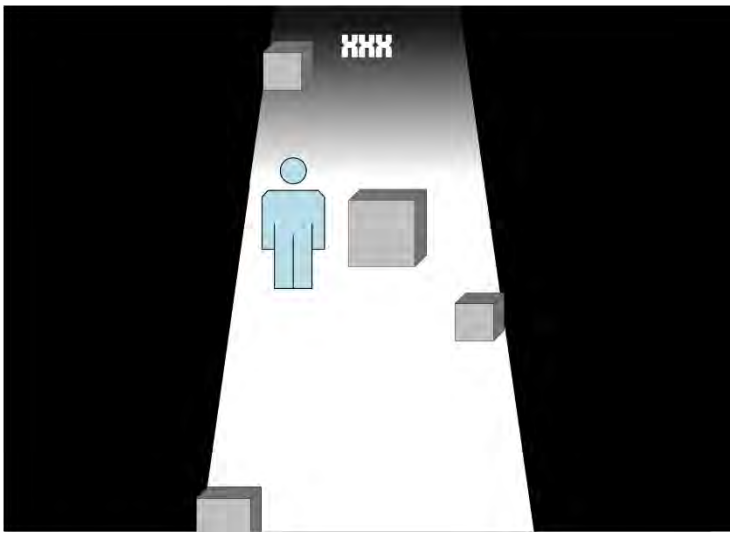

Abbildung 20 – Benutzeroberflächen Design – Hindernis ausweichen

Kollision mit Hindernis:

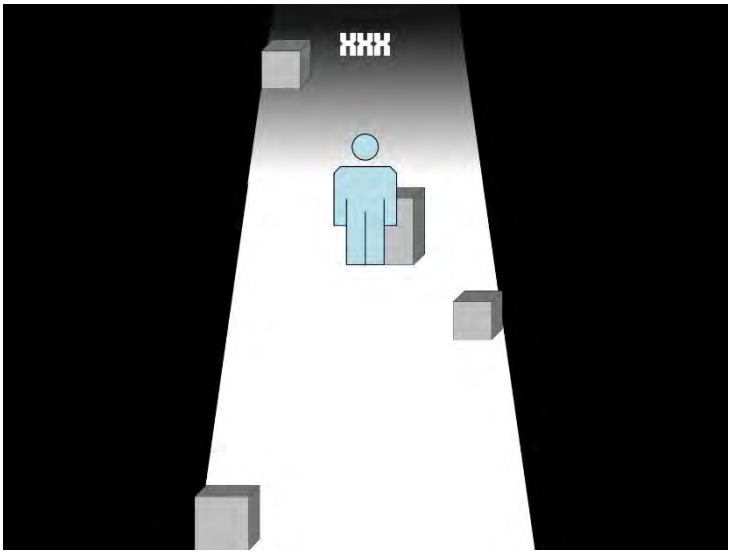

Abbildung 21 – Benutzeroberflächen Design - Kollision

Game Over:

Abbildung 22 – Benutzeroberflächen Design – Game Over

8 Realisierung

Hinweis: Zur vereinfachten Darstellung wurden bei den Code-Snippets die Kommentare zum Grossteil entfernt.

8.1 Spielfigur Grafik

Als erstes brauchte ich eine 3D Grafik für die Spielfigur. Wie in der Definitionsphase erarbeitet, entschied ich mich für das Tool «Blender». Um zu verstehen, wie und was ich mit Blender realisieren kann, schaute ich zu Beginn im Internet mehrere Tutorials. Anspruchsvoller als im Voraus gedacht, arbeitete ich die Spielfigur so gut wie es ging aus. Da es sich um keine Grafikarbeit handelt, feilte ich es aus zeitlichen Gründen nicht bis ins Detail aus. Des Weiteren fehlt mir die dazu benötigte Erfahrung. Folgende Bildabfolge habe ich für den Grundriss als Vorlage verwendet:

Abbildung 23 – Vorlage für Grundgerüst der Spielfigur (Artstation, o.J.)

Für die Einfärbung der Spielfigur, schaute ich diverse Bilder von Ironman an und machte einen für mich passenden Mix daraus. Denn auf jedem Bild ist er farblich ziemlich verschieden dargestellt. Schlussendlich sah die Spielfigur so aus:

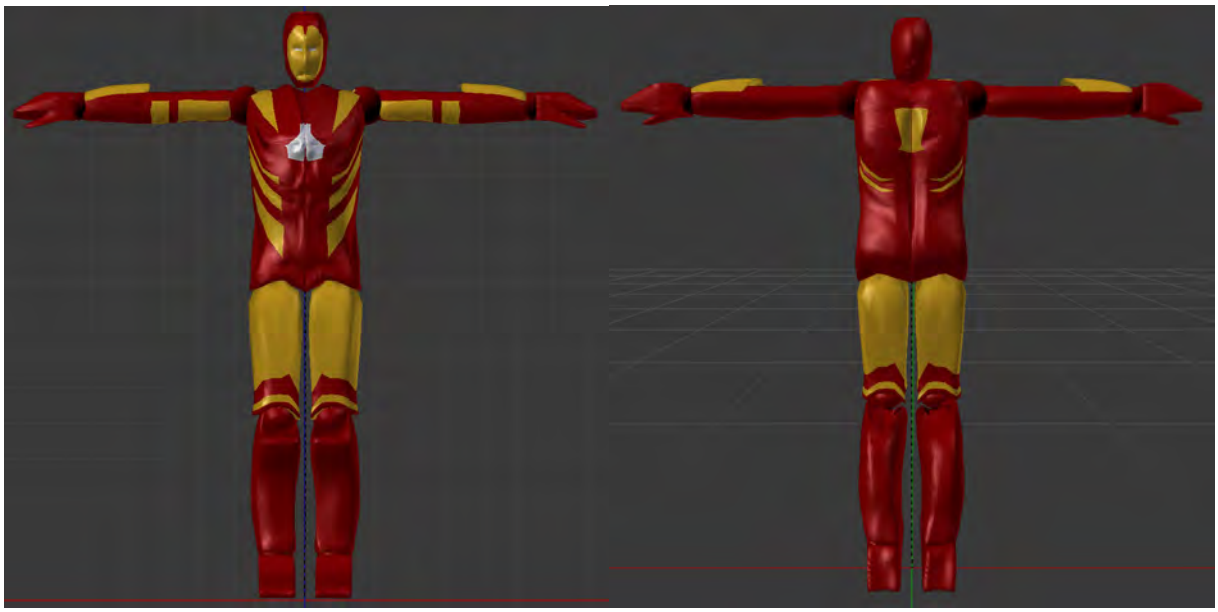

Abbildung 24 – Grafik der Spielfigur

Die Animation konnte ich ebenfalls direkt im Blender erstellen. Dafür erstellte ich eine Abfolge von diversen Lauf- und Sprungpositionen. Mit den erstellten Knochen der Spielfigur, konnte ich sie in die gewünschte Position bringen. Natürlich könnte man der Figur noch weitere Knochen zuteilen, jedoch fand ich einen rudimentären Knochenbau für einen Roboter passend. Mit ein wenig experimentieren fand ich die entsprechenden Posen heraus, sodass die Lauf- und Sprunganimation möglichst real aussah.

Im Blender sah dies folgendermassen aus:

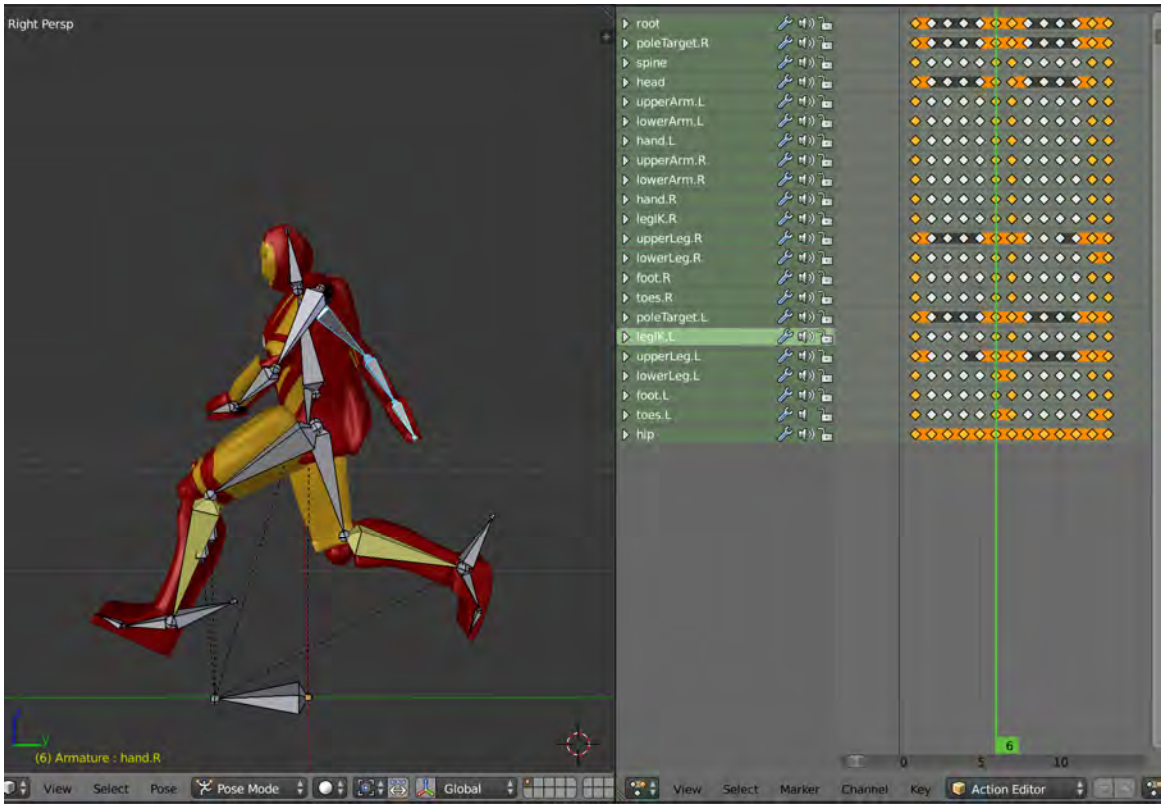

Abbildung 25 – Spielfigur Animation erstellen

8.2 Laufstrecke Grafik

Wie bereits erwähnt, entschloss ich mich aus zeitlichen Gründen nur die Grafik der Spielfigur selbst zu erstellen. Aus dem Designentwurf war bereits festgelegt, dass die Laufstrecke als Strasse realisiert werden soll. Somit suchte ich im Unity Asset Store nach diversen Strassen. Nach einer Weile habe ich eine breite und leere Strasse gefunden, welche meinen Geschmack entspricht. Diese ist in der unterstehenden Grafik abgebildet.

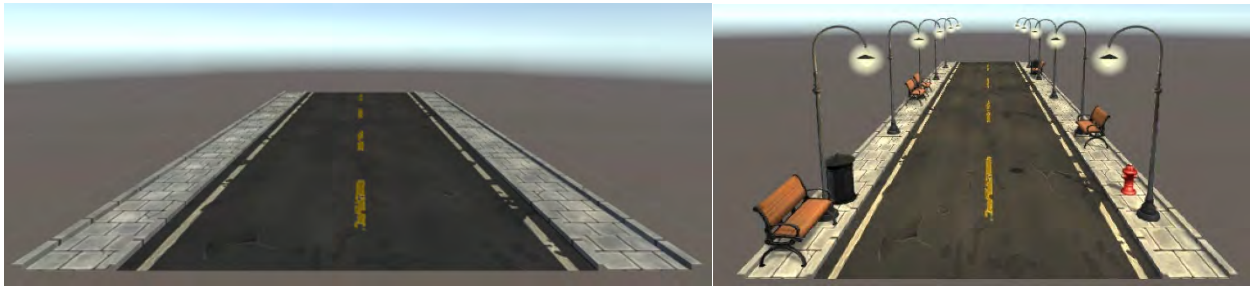

Abbildung 26 – Grafik der Laufstrecke

Diese dekorierte ich anschliessend auf dem Trottoir mit Mülleimern, Strassenbanken und Strassenlaternen. Dazu fügte ich auf beiden Trottoirs eine transparente «Wand» ein, sodass die Spielfigur später nur auf der Strasse laufen kann. (Unity Asset Store, 2018)

8.3 Hindernisse Grafik

Für die Hindernisse habe ich ebenfalls Grafiken vom Unity Asset Store verwendet. Ich suchte mir mehrere, möglichst zusammenpassende Strassenelemente wie diese:

Abbildung 27 – Hindernis Grafiken

8.4 Spielfigur «läuft» stetig

Als erstes erstellte ich eine neue Scene «Game» und importierte die Spielfigur Grafik von Blender in mein Unity Projekt. Danach zog ich per Drag and Drop meine erstellte Spielfigur in die Scene und setzte sie auf jeder Achse auf den Nullpunkt. Anschliessend entfernte ich alle Standard-Komponenten der Spielfigur bis auf «Transform» und «Animator» und fügte ihr ein «Character Controller» hinzu, um die Spielfigur zu steuern.

Beim Importieren der Spielfigur-Grafik von Blender in das Projekt, wurden auch alle erstellten Animationen mitimportiert. Für die Animator-Komponente der Spielfigur erstellte ich eine Animation in Unity, welche per Boolean zwischen «Run» und «Jump» gesteuert werden kann.

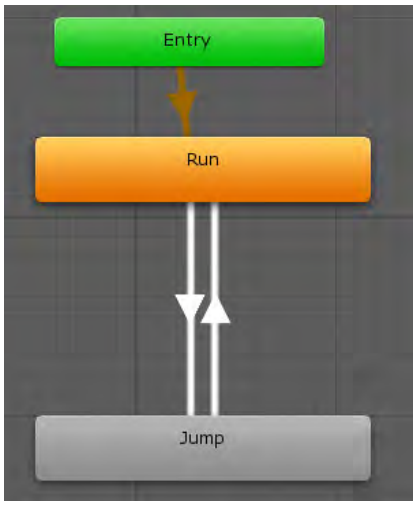

Abbildung 28 – Screenshot Unity Animation

Ich erstellte einen neuen Ordner «Scripts» mit einem C#-Skript `PlayerMove`, welches der Spielfigur als Komponente zugewiesen wurde. Das Skript ist für das Bewegen der Spielfigur zuständig. Um sie stetig geradeaus fortzubewegen, wird folgender Code verwendet.

```
private CharacterController controller;

private float speed = 6.0f;

// Use this for initialization
void Start()
{
    controller = GetComponent<CharacterController>();
}

// Update is called once per frame
void Update()
{
    controller.Move(Vector3.forward * speed * Time.deltaTime);
}
```

Die Variable `controller` greift auf die CharacterController-Komponente zu. Der CharacterController beinhaltet eine Move-Methode mit der das entsprechende Element, in meinem Fall die Spielfigur, bewegt werden kann. Als Parameter wird ein `Vector3` Bewegung erwartet, also `Vector3.forward`, da die Figur sich vorwärtsbewegen soll. Eigentlich würde dies schon ausreichen, um die Spielfigur stetig geradeaus fortzubewegen. Doch mit einem kurzen Gedanken an später, läuft die Figur mit einem Meter pro Frame viel zu schnell. Deshalb entschied ich mich die Figur sechs Meter pro Sekunde zu bewegen. Der positive Nebeneffekt davon ist, dass es auf allen Computer gleich aussieht, egal wie viele Frames der Computer pro Sekunde verarbeiten kann. Somit erstellte ich eine Variable `speed` und multiplizierte `Vector3.forward` mit dem `speed` und mit `Time.deltaTime`.

8.5 Spielfigur links/rechts bewegen

Um die Spielfigur nach rechts und links zu bewegen, erstellte ich eine `Vector3` Variable `moveCharacter`, welcher `Vector3.zero` zugewiesen wird. Der `Vector3` besteht aus den drei Achsen X, Y und Z. Die X-Achse ist für links und rechts, die Y-Achse für hoch und runter und die Z-Achse für vor- und rückwärts verantwortlich. Der X-Achse des Vektors wird `Input.GetAxisRaw("Horizontal")` zugewiesen. Der String `Horizontal` ist im `InputManager` von Unity bereits vordefiniert. Standardmässig sind dem Positiv- und Negativ-Button die Pfeiltasten links und rechts zugewiesen.

Dass die Spielfigur dennoch stetig läuft, wird der Z-Achse die Variable `speed` zugewiesen. Der Parameter für die Move-Methode des Character Controllers wurde wie im Code-Snippet ersichtlich durch `moveCharacter * Time.deltaTime` ersetzt.

```
    moveCharacter = Vector3.zero;
    moveCharacter.x = Input.GetAxisRaw("Horizontal") * speed;
    moveCharacter.z = speed;

    controller.Move(moveCharacter * Time.deltaTime);
```

8.6 Kamerasicht

Zu Beginn erstellte ich ein neues C#-Skript «CameraMove», welches dem Objekt «Main Camera» zugewiesen wurde. Bei dem Spiel sollte die Spielfigur immer von hinten zu sehen sein das heisst, die Kamera sollte laufend der Spielfigur mit demselben Offset folgen. In der Start-Methode wird in der Variable `playerTransform` die Position der Spielfigur gespeichert, sowie der Offset zwischen der aktuellen Position der Kamera in der Scene und der Spielfigur ermittelt.

```
void Start()
{
    playerTransform = GameObject.FindGameObjectWithTag("Player").transform;
    cameraOffset = transform.position - playerTransform.position;
}
```

Ähnlich wie bei der Spielfigur wird eine `Vector3` Variable `moveCamera` erstellt. Dieser wird die Position der Spielfigur plus den Offset zur aktuellen Position der Kamera in der Scene zugewiesen. Die X-Achse von `moveCamera` wird auf Null gesetzt, da die Kamera sich immer mittig von der Laufstrecke befinden soll, auch wenn sich die Spielfigur nach links oder rechts bewegt. Die Position auf der Y-Achse kann sich zwischen den Werten zwei und vier befinden. Diese Differenz wird für die spätere Sprung-Realisierung der Spielfigur verwendet.

Um das Spiel ein bisschen schöner einzuleiten, gleitet die Kamerasicht beim Starten von der Vogelperspektive langsam in die normale Position hinab, immer mit dem Blickwinkel auf die Spielfigur gerichtet. Um das zu realisieren, wird ein neuer Vektor `animationOffset` erstellt. Die Kameraposition wird mit einem `Vector3.Lerp` (Interpolation) verschoben, um einen sanften Übergang zwischen Animation Start- und Endpunkt zu erzeugen. Die Startposition setzt sich aus der aktuellen Kameraposition plus dem erstellten `animationOffset` zusammen und der Endpunkt ist gleich der aktuellen Kameraposition. Als Interpolationswert wird `animationProgress` gesetzt, welcher laufend so erhöht wird, dass die gesamte Animation schlussendlich drei Sekunden dauert. Standardmässig zeigt der Kamerablickwinkel immer geradeaus, jedoch sollte dieser immer auf die Spielfigur gerichtet sein. Deshalb wird die `transform.LookAt()` Methode aufgerufen, welcher die Position der Spielfigur plus eineinhalb Punkte darüber mitgegeben wird. Ohne die eineinhalb Punkte darüber, wären die Füsse der Spielfigur in der Mitte des Bildes, was für das Spielgefühl nicht optimal ist.

Sobald die Startanimation vorbei ist, wird die Kamera Position laufend durch `moveCamera` ersetzt und die Kamera folgt der Spielfigur von hinten mit dem gewünschten Offset.

```
void Update()
{
    Vector3 moveCamera = playerTransform.position + cameraOffset;
    moveCamera.x = 0;
    moveCamera.y = Mathf.Clamp(moveCamera.y, 2, 4);

    if (animationProgress < 1.0f)
    {
        // Start Game
        Vector3 animationOffset = new Vector3(0, 5, 5);
        transform.position = Vector3.Lerp(moveCamera + animationOffset, moveCamera, animationProgress);
        animationProgress += Time.deltaTime / animationPeriod;
        transform.LookAt(playerTransform.position + Vector3.up * 1.5f);
    }
    else
    {
        transform.position = moveCamera;
    }
}
```

Während der Startanimation, soll die Spielfigur nicht lenkbar sein. Somit wird die `animationPeriod` in das `PlayerMove` Skript übernommen. In den ersten drei Sekunden des Spiels, wird die Figur nur vorwärtsbewegt. Es wird ein `return` aufgerufen, sodass der restliche Code in der Update-Methode nicht ausgeführt wird.

```
if (Time.time - startTime < animationPeriod)
{
    controller.Move(Vector3.forward * speed * Time.deltaTime);
    return;
}
```

8.7 Diverse Laufstrecken-Abschnitte inklusive Hindernisse erstellen

Um das Spiel spannender zu gestalten, sollen sich auf der Laufstrecke diverse Hindernisse befinden, sodass die Spielfigur ihnen ausweichen muss. Als Grundlage nahm ich für die weiteren Abschnitte die bereits erstellte Strasse mit den sich darauf befindenden Dekorationen. Auf jedem neuen Abschnitt änderte ich die Dekorationen und fügte beliebig Hindernisse ein. Die jeweiligen Abschnitte speicherte ich im Projektordner unter Prafabs/Tiles. Als das Spiel soweit fertiggestellt war, musste ich diverse Hindernisse noch ein bisschen verschieben, da es teilweise zu einfach oder zu schwierig war.

Abbildung 29 – Ausschnitt Laufstrecken Grafik

8.8 Laufstrecken-Abschnitte wiederholen

Um die Spielfigur unendlich laufen zu lassen, müssen sich die Laufstrecken-Abschnitte immer wiederholen. Um dies zu realisieren, erstellte ich ein leeres Game Object namens «TileManager» und setzte alle Achsen-Punkte auf Null. So befindet sich das neu erstellte Objekt direkt unter der Spielfigur, wo die Laufstrecke beginnen soll. Für das TileManager-Objekt erstellte ich ein gleichbenanntes C#-Script.

Um die verschiedenen Laufstreckenabschnitte dem Manager zuzuweisen, wurde ein public GameObject Array erstellt. So kann in der Inspector Ansicht des TileManager-Objects die Grösse des Arrays angeben und die verschiedenen Tiles hineingezogen werden.

Abbildung 30 – Screenshot von der TileManager-Skript Komponente

Nebst dem Array für die verschiedenen Laufstrecken wurden folgende Variablen im TileManager-Skript erstellt.

```
public GameObject[] tiles;

private Transform playerTransform;
private float tilePositionZ = -10.0f;
private float tileLength = 30.0f;
private float tileOffset = 5.0f;
private int tileCount = 3;
private List<GameObject> activeTiles;
```

Im Skript wird in der Start-Methode die Player Transformation ermittelt, denn wie auch die Kamera, positionieren sich die Tiles anhand der Spielfigur.

Der `tilePositionZ` Variable wird der Z-Achsenpunkt zugewiesen, an welchem sich das erste Tile beim Start befinden soll. Dieser Wert liegt im Minusbereich, da die Laufstrecke sich für die Vogelperspektivenanimation der Kamera weiter hinten als die Spielfigur befinden muss. Die effektive Länge der Laufstreckenabschnitte wird in der `tileLength` Variable gespeichert.

Da der Benutzer in seiner Ansicht auch während dem Spiel noch ein bisschen hinter die Spielfigur sieht, wird ein `tileOffset` festgelegt. `TileCount` definiert, wie viele Tiles sich zusammen in der Scene befinden sollen. Da meine Abschnitte ziemlich lange sind, reichen drei Abschnitte aus, um den Platz bis zum Horizont zu füllen.

Die `activeTiles` Liste wird verwendet, um die Abschnitte laufend aufzuräumen, dazu später mehr.

Als erste Laufstrecke soll immer die leere Laufstrecke erscheinen, da der Benutzer während der Kameraanimation den Hindernissen nicht ausweichen kann. Dafür soll sie im Verlaufe des Spiels nicht mehr auftauchen. Die leere Laufstrecke, habe ich als erstes Objekt dem Tiles-Array zugewiesen. Um diese zu erhalten, wird die CreateTile-Methode mit dem Parameter 0 aufgerufen.

Alle anderen Abschnitte sollen sich in zufälliger Reihenfolge laufend wiederholen. Um diese zu erstellen wird der CreateTile-Methode keinen Parameter mitgegeben, per Default wird der Parameter `index` auf 1 gesetzt.

In der Start-Methode werden die ersten drei Tiles per CreateTile-Methode erstellt.

```
private void CreateTile(int index = 1)
{
    GameObject tileObject = Instantiate(tiles[GetRandomIndex(index)]) as GameObject;

    tileObject.transform.SetParent(transform);
    tileObject.transform.position = Vector3.forward * tilePositionZ;
    tilePositionZ += tileLength;
    activeTiles.Add(tileObject);
}
```

Die GetRandomIndex-Methode gibt für die leere Laufstrecke 0 oder einen Random Index der übrigen Laufstrecken des `tiles`-Array zurück. Die entsprechende Laufstrecke wird als `GameObject` instanziiert und dem Game Object «TileManager» als Child zugewiesen.

Als Position wird ihr `Vector3.forward * tilePositionZ` gesetzt was bedeutet, dass das Objekt auf der Z-Achse zu dem Punkt `tilePositionZ` verschoben wird. Anschliessend wird die `tilePositionZ` bereits für den nächsten Abschnitt um die Länge eines Tiles erhöht. Zum Schluss wird es zur Liste `activeTiles` hinzugefügt.

Die Update-Methode sieht folgendermassen aus.

```
private void Update()
{
    if (playerTransform.position.z - tileLength - tileOffset > tilePositionZ - tileCount * tileLength)
    {
        CreateTile();
        DeleteTile();
    }
}
```

Sobald der Z-Achsenpunkt des ersten Tile kleiner als die Position der Spielfigur abzüglich der Länge des aktuell darauf laufenden Tiles und des Offset ist, wird ein neues Tile hinzugefügt und anschliessend eines entfernt.

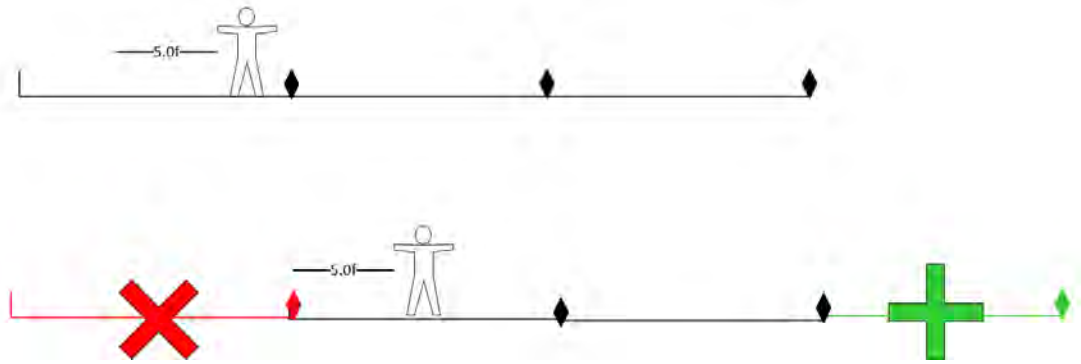

Abbildung 31 – Logik, um Laufstreckenabschnitte hinzuzufügen und zu entfernen

Sobald ein neuer Abschnitt hinzugefügt wird, wird derjenige, welcher nicht mehr notwendig ist, gelöscht. Dazu wird die `activeTiles` Liste verwendet. Das erste Objekt in der Liste wurde als erstes hinzugefügt und ist somit auch dasjenige, welches nicht mehr gebraucht wird. Zuerst wird das Objekt von der Scene entfernt und anschliessend aus der `activeTiles` Liste gelöscht.

```
private void DeleteTile()
{
    Destroy(activeTiles[0]);
    activeTiles.RemoveAt(0);
}
```

8.9 Überprüfung, ob dem Hindernis korrekt ausgewichen wurde

Beim Erstellen der Laufstrecken wurde jedem Hinderniselement ein BoxCollider und der neu erstellte «Obstacles» Tag hinzugefügt. Der CharacterController der Spielfigur beinhaltet automatisch eine sogenannte Kapsel. Diese ist auf dem Bild als dünne, grüne Linie zu erkennen. Ebenfalls in einer grünen Linie sind die Box Collider bei den Hindernissen zu sehen.

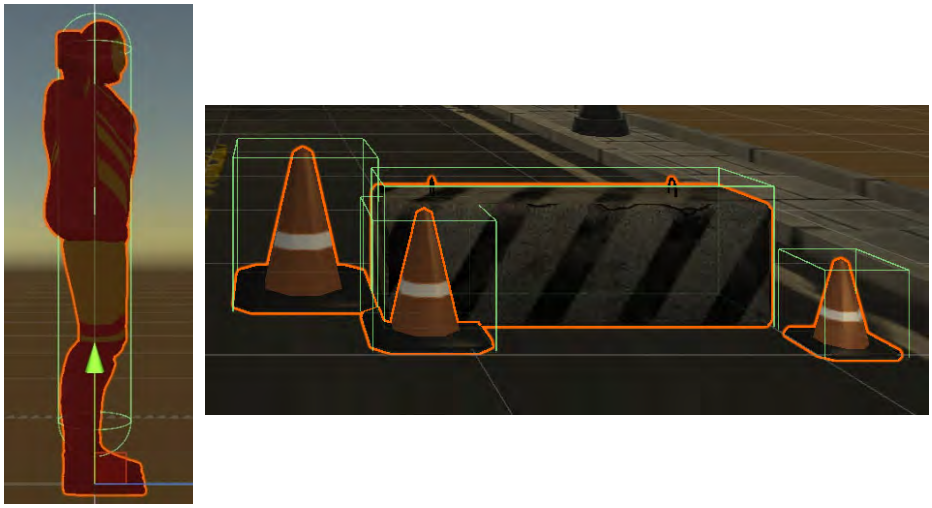

Abbildung 32 – Colliders für die Kollisionsüberprüfung

Denn sobald die Kapsel des Character Controllers der Spielfigur ein anderes Objekt berührt, also zwei grüne Linien aufeinandertreffen, wird automatisch die OnControllerColliderHit-Methode aufgerufen. Mithilfe der Box Collider der Hindernisse kann Unity bestimmen, dass dies ein Objekt ist. Um Fehlkollisionen zu vermeiden, wird bei einer Kollision mithilfe des gesetzten Tags pro Hindernis überprüft, ob es sich wirklich um ein Hindernis handelt. Falls es zu einer Kollision kommt, wird die Game Over Ansicht aufgerufen.

```
private void OnControllerColliderHit(ControllerColliderHit hit)
{
    if (hit.gameObject.tag == "Obstacles")
    {
        GameOver();
    }
}
```

8.10 Menu und Highscore Ansicht in Menu

Das Menu wird beim starten des Programms angezeigt, sowie wenn man bei der Game Over Ansicht auf den Menu-Button klickt. Auf der Menu-Ansicht ist der Titel des Spiels «Road Runner» und der Play-Button zum starten des Spiels zu sehen. Ebenfalls erscheint unten links ein Exit-Button zum Beenden des Spiels und unten rechts den bereits erspielten Highscore des lokalen Benutzers. Dieser wird aus den `PlayerPrefs` herausgelesen und in das entsprechende Textfeld abgefüllt.

```
void Start()
{
    highscoreText.text = "Highscore: " + PlayerPrefs.GetInt("highscore").ToString();
}
```

8.11 Score Ansicht und Spielfigur Geschwindigkeit erhöhen

Während dem Spiel wird ein Punktezähler für den Benutzer eingeblendet. Um diesen zu realisieren wird der Game Scene ein Textfeld `ScoreCounter` beigefügt. Dem Spielfigur Objekt wird eine neue Skript-Komponente mit dem `ScoreCount` Skript hinzugefügt und das neu erstellte Textfeld wird als Parameter mitgegeben.

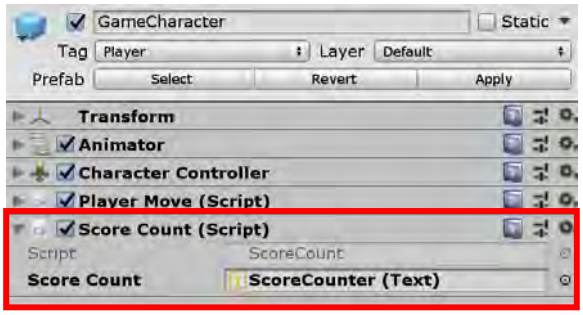

Abbildung 33 – Screenshot von der ScoreCount-Skript Komponente

Im `ScoreCount` Skript startet der Zähler bei 0 und das Level bei 1. Das fünfte Level wird als Maximallevel festgelegt und immer nach 10 Punkten wird das Level erhöht.

In der Update Methode wird überprüft, ob der Score bereits für das nächste Level genügt. Falls ja wird das Level um eines erhöht, insofern das Maximallevel noch nicht erreicht ist. Sobald sich das Level erhöht wird die `IncreaseRunningSpeed` Methode von dem `PlayerMove` Skript aufgerufen, welche mit `speed++` die Geschwindigkeit der Spielfigur um eins erhöht. Der Score wird pro Sekunde um die Zahl der aktuellen Levelstufe erhöht und dem Textfeld `ScoreCounter` als Text mitgegeben.

```
void Update()
{
    if (score >= nextLevelAt)
    {
        NextLevel();
    }
    score += Time.deltaTime * level;
    scoreCount.text = ((int)score).ToString();
}

private void NextLevel()
{
    if (level == maxLevel)
    {
        return;
    }
    nextLevelAt += nextLevelAt;
    level++;

    GetComponent<PlayerMove>().IncreaseRunningSpeed();
}
```

```
public void IncreaseRunningSpeed()
{
    speed++;
}
```

8.12 Game Over Ansicht

Die Game Over Ansicht wird angezeigt, sobald die Spielfigur mit einem Hindernis kollidiert und das Spiel vorbei ist. Auf der Ansicht ist als Überschrift «Game Over», sowie ein Restart- und Menu-Button zu sehen. Ebenfalls erscheint direkt unter der Überschrift, der erreichte Punktestand der aktuellen Runde und ob es ein neuer Highscore ist oder nicht. Falls der Highscore geknackt wurde, wird dieser in den `PlayerPrefs` überschrieben.

```
void Start()
{
    int score = (int)PlayerPrefs.GetFloat("score");

    if (score > PlayerPrefs.GetInt("highscore"))
    {
        PlayerPrefs.SetInt("highscore", score);
        scoreText.text = "New Highscore: " + score.ToString();
    }
    else
    {
        scoreText.text = "Score: " + score.ToString();
    }
}
```

8.13 Spielfigur springt per Tastatureingabe

Die Spielfigur soll beim Drücken der Leertaste einmal in die Luft springen, für dies wird das `PlayerMove` Skript ein wenig angepasst. Es wird um folgende Variablen erweitert.

```
private Animator animator;
private float jumpHeight = 1.0f;
private float verticalPosition = 0.0f;
private float capsuleNormalHeight = 1.6f;
private float capsuleJumpHeight = 0.7f;
private bool isJumping = false;
```

Die `animator` Variable wird verwendet, um der Animation den Boolean mitzugeben, ob die Spielfigur springen soll oder nicht. Mit `jumpHeight` wird die Sprunghöhe auf `1.0f` gesetzt und die `verticalPosition` ist standardmässig auf 0, da die Spielfigur zu Beginn normal läuft. Der `isJumping` Boolean wird verwendet, dass die Spielfigur erst nach erfolgreicher Landung des letzten Sprunges erneut springen kann.

Sobald die Leertaste gedrückt wird und die Spielfigur sich nicht bereits im Sprung befindet, wird ein Sprung gestartet. Das heisst, `isJumping`, sowie der Boolean des Animators «Jump» wird auf `true` gesetzt. Da die Höhe des Spielfigur Objektes beim Sprung kleiner wird, wird die Höhe während des Sprunges verkleinert, sodass es zu keiner optischen Fehlkollision führt. Als `verticalPosition` wird die festgelegte Sprunghöhe gesetzt.

```
if (!isJumping && Input.GetKeyDown(KeyCode.Space))
{
    isJumping = true;
    controller.height = capsuleJumpHeight;
    animator.SetBool("Jump", true);
    verticalPosition = jumpHeight;
}
```

Anschliessend wird der Spielcharakter während der bereits stetigen Vorwärtsbewegung auf der Y-Achse zu dem Sprunghöhepunkt verschoben. Da eine Person nicht gleich schnell springen wie laufen kann, wird die `verticalPosition` nur mit der Hälfte des `speed` multipliziert. Während eines Sprungs wird die Steuerung der X-Achsenbewegung ignoriert. Sobald die Spielfigur die Sprunghöhe erreicht hat, wird die `verticalPosition` wieder auf 0 gesetzt. Wenn diese den Nullpunkt erreicht hat, wechselt die `isJumping` Variable wieder zu `false` und die Höhe des Spielfiguren Objektes wird wieder zu seiner Standardhöhe zurückgesetzt.

```
if (controller.transform.position.y >= jumpHeight)
{
    animator.SetBool("Jump", false);
    verticalPosition -= jumpHeight;
}

moveCharacter = Vector3.zero;
if (!isJumping)
{
    moveCharacter.x = Input.GetAxisRaw("Horizontal") * speed;
}
moveCharacter.y = verticalPosition * speed / 2;
moveCharacter.z = speed;

controller.Move(moveCharacter * Time.deltaTime);

if (controller.transform.position.y <= 0)
{
    isJumping = false;
    controller.height = capsuleNormalHeight;
}
```

8.14 Abweichungen der Implementation zum Softwareentwurf

Um Planung und Realität zu vergleichen, stellte ich die Screenshots von dem Resultat dem geplanten Design gegenüber. Die Umsetzung des geplanten Designs gelang mir meiner Meinung nach ziemlich gut.

Bei dem Menu und Game Over Screen nahm ich etwas grellere Farbtöne als geplant und eine etwas andere Schriftart. Ebenso fügte ich bei beiden Screens ein Strassenelement und bei dem Game Over Screen zusätzlich ein Hindernis ein. Der Grund dafür war, dass es für mich so ansprechender wirkt.

Die aktive Spielansicht konnte wie geplant umgesetzt werden und generell gab es keine grösseren Abweichungen.

Menu Softwareentwurf:

Abbildung 34 – Geplantes Menu Design

Menu Implementation:

Abbildung 35 – Umgesetztes Menu Design

Aktives Spiel Softwareentwurf:

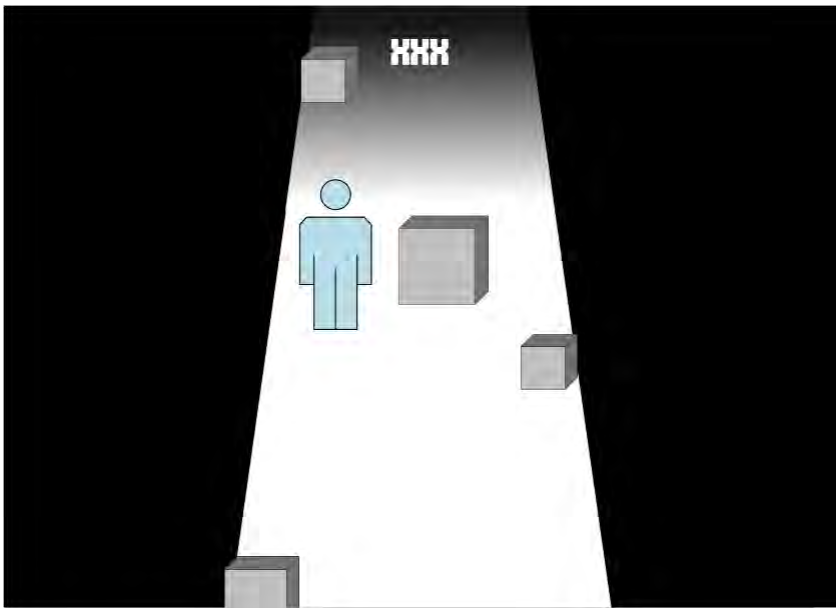

Abbildung 36 – Geplantes aktives Spiel Design

Aktives Spiel Implementation:

Abbildung 37 – Umgesetztes aktives Spiel Design

Game Over Softwareentwurf:

Abbildung 38 – Geplantes Game Over Design

Game Over Implementation:

Abbildung 39 – Umgesetztes Game Over Design

8.15 Benutzer- und Installationsanleitung

1. Der Benutzer erhält eine Zip-Datei mit allen benötigten Dateien. Diese muss er per Rechtsklick auf das Zip-File zuerst entpacken.

Abbildung 40 – ZIP-Datei entpacken

2. Der automatisch neu erstellte Ordner «RoadRunner» beinhaltet folgendes:

Abbildung 41 – Übersicht enpackte ZIP-Datei

3. Per Doppelklick auf die EXE-Datei wird die Spiel-Software gestartet und das Menu erscheint. Die Software kann entweder per EXIT-Button oder mit der Standardfunktion von Windows beendet werden.

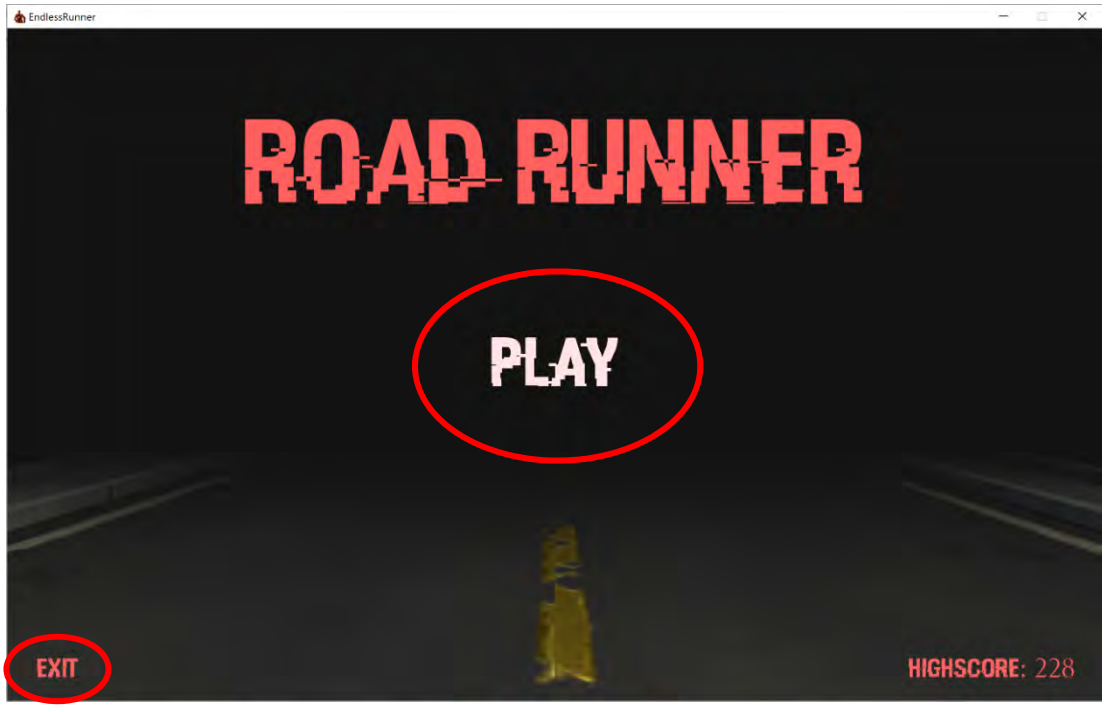

Abbildung 42 – Benutzeranleitung der Menu Ansicht

4. Sobald auf den Play-Button geklickt wird, beginnt das Spiel. Als Benutzer kann man die Spielfigur mit den Pfeiltasten links/rechts steuern oder mit drücken der Leertaste in die Luft springen lassen, um den Hindernissen auszuweichen.

Abbildung 43 – Benutzeranleitung des aktiven Spiels

5. Wenn die Spielfigur mit einem Hindernis kollidiert, ist das Spiel vorbei und folgende Ansicht erscheint. Als Benutzer hat man zwei Optionen. Entweder kann direkt ein neues Spiel gestartet oder zum Menu zurückgekehrt werden.

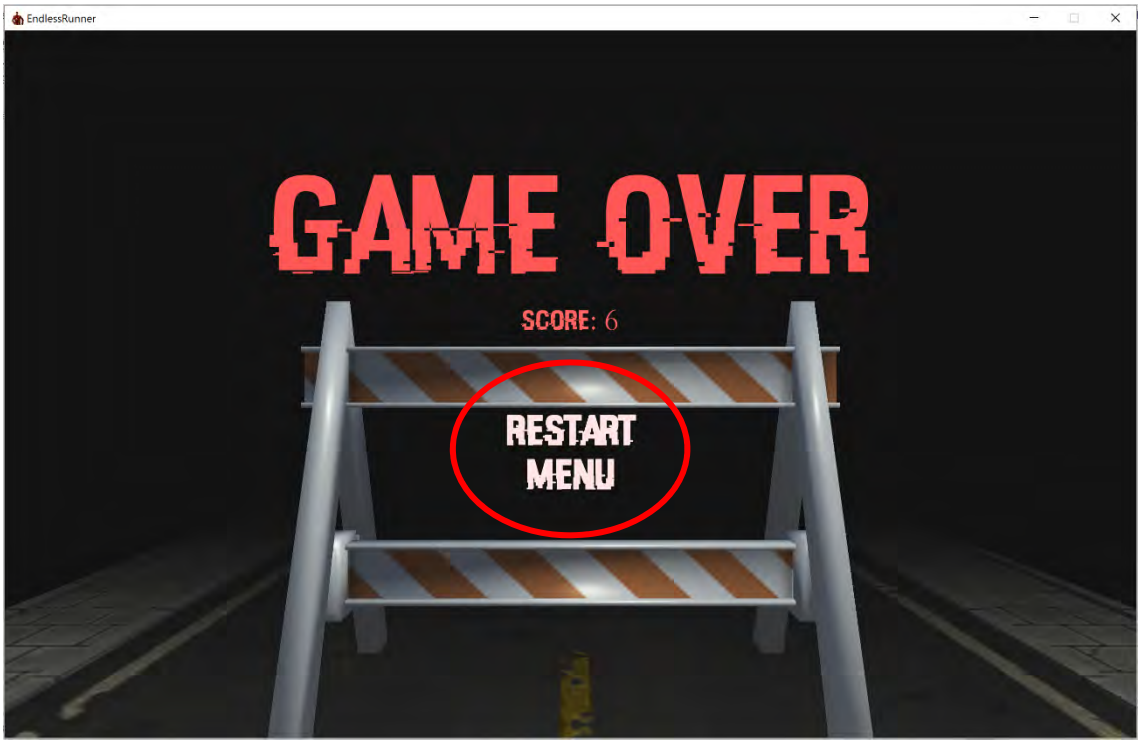

Abbildung 44 – Benutzeranleitung der Game Over Ansicht

9 Testing

9.1 Testkonzept

9.1.1 Umfeld

Getestet wird auf einem Surface Pro 4 mit Windows 10 Pro.

9.1.2 Abgrenzung

Da die Applikation neu ist, gibt es keine Abgrenzungen. Die ganze Applikation wird vollumfänglich getestet.

9.1.3 Testmethoden/-strategie

Die Testfälle werden als Benutzertests ausgeführt. Dies bedeutet, dass ein Tester (bei dieser Arbeit ich selbst) alle Tests wie beschrieben durchführt. White-Box-Tests wurden bewusst weggelassen, da allfällige Fehler bei den Benutzertests direkt auffallen würden und das Spielgefühl nicht durch White-Box-Tests testbar ist.

Anhand aller Anforderungen im Pflichtenheft wurden die Abnahmetests ausgearbeitet, um alle Funktionalitäten zu testen. Die Voraussetzungen für den Testfall, sowie die Eingabe des Benutzers und das erwartete Ergebnis sind definiert.

Ein Testfall ist erst erfolgreich, wenn das Ergebnis mit dem erwartenden Ergebnis übereinstimmt.

9.2 Testfälle

9.2.1 Benutzertests

Testfall-Nr. - Titel	TF_001 – Spiel starten
Beschreibung	Spiel starten.
Anforderungs-Nr.	Anforderung_001
Voraussetzung	• EXE-Datei muss vollständig vorhanden sein
Input	Benutzer führt einen Doppelklick auf die EXE Datei aus.
Erwarteter Output	EXE Datei öffnet sich und das Menu wird angezeigt.

Testfall-Nr. - Titel	TF_002 – Play Button
Beschreibung	Das Spiel beginnt nach dem drücken des Play-Buttons.
Anforderungs-Nr.	Anforderung_001
Voraussetzung	• Spiel befindet sich auf der Menu Ansicht
Input	Benutzer drückt auf den Play-Button.
Erwarteter Output	Das Spiel beginnt und die Spielfigur läuft stetig gerade aus.

Testfall-Nr. - Titel	TF_003 – Exit Button
Beschreibung	Die Applikation wird beendet.
Anforderungs-Nr.	Anforderung_001
Voraussetzung	• Spiel befindet sich auf der Menu Ansicht
Input	Benutzer drückt auf den Exit-Button.
Erwarteter Output	Die Applikation wird beendet.

Testfall-Nr. - Titel	TF_004 – Spielfigur läuft stetig
Beschreibung	Spielfigur läuft ohne Benutzereingabe stetig vorwärts.
Anforderungs-Nr.	Anforderung_002
Voraussetzung	• Spiel ist gestartet
Input	Benutzer macht nichts.
Erwarteter Output	Spielfigur läuft stetig vorwärts und die Laufstrecke «erneuert» sich endlos, bis zu einem Game Over.

Testfall-Nr. - Titel	TF _005 – Keine Eingabe
Beschreibung	Benutzer tätigt keine Eingabe, obwohl ein Hindernis kommt.
Anforderungs-Nr.	Anforderung_002
Voraussetzung	• Spiel ist gestartet
Input	Benutzer macht nichts.
Erwarteter Output	Spielfigur läuft in das nächste Hindernis und der Game Over Screen wird angezeigt.

Testfall-Nr. - Titel	TF_006 – Spielfigur weicht aus
Beschreibung	Spielfigur weicht beim Drücken der Pfeiltaste in die entsprechende Richtung.
Anforderungs-Nr.	Anforderung_003
Voraussetzung	• Spiel ist gestartet
Input	Benutzer drückt die Pfeiltaste nach links.
Erwarteter Output	Spielfigur weicht auf die linke Seite aus.

Testfall-Nr. - Titel	TF_007 – Unerwartete Tastatureingabe
Beschreibung	Benutzer tätigt unerwartete Tastatureingabe.
Anforderungs-Nr.	Anforderung_003
Voraussetzung	• Spiel ist gestartet
Input	Benutzer drückt die Taste R.
Erwarteter Output	Spiel läuft normal weiter.

Testfall-Nr. - Titel	TF_008 – Unerwartete Mauseingabe
Beschreibung	Benutzer tätigt unerwartete Mauseingabe.
Anforderungs-Nr.	Anforderung_003
Voraussetzung	• Spiel ist gestartet (kein Game Over)
Input	Benutzer klickt mit der Maus in das aktive Spiel.
Erwarteter Output	Spiel läuft normal weiter.

Testfall-Nr. - Titel	TF_009 – Hindernis-Abschnitte tauchen auf
Beschreibung	Es tauchen laufend neue Hindernis-Abschnitte auf.
Anforderungs-Nr.	Anforderung_004
Voraussetzung	• Spiel ist gestartet (kein Game Over)
Input	Der Benutzer spielt das Spiel ganz normal ohne spezielle Eingaben.
Erwarteter Output	Es tauchen in zufälliger Reihenfolge laufend neue Hindernis-Abschnitte auf.

Testfall-Nr. - Titel	TF_010 – Kollisionscheck
Beschreibung	Spielfigur berührt ein Hindernis während dem ausweichen.
Anforderungs-Nr.	Anforderung_005
Voraussetzung	• Spiel ist gestartet
Input	Benutzer weicht einem Hindernis so aus, dass die Spielfigur das Hindernis trotzdem noch berührt.
Erwarteter Output	Es führt zu einer Kollision und der Game Over Screen wird angezeigt.

Testfall-Nr. - Titel	TF_011 – Restart
Beschreibung	Nach einem Game Over wird einen Restart des Spiels ausgeführt.
Anforderungs-Nr.	Anforderung_006
Voraussetzung	• Spiel ist gestartet • Benutzer befindet sich auf dem Game Over Screen
Input	Benutzer klickt auf Restart.
Erwarteter Output	Spiel startet sich neu.

Testfall-Nr. - Titel	TF_012 – Menu
Beschreibung	Nach einem Game Over zurück zum Menu.
Anforderungs-Nr.	Anforderung_006
Voraussetzung	• Spiel ist gestartet • Benutzer befindet sich auf dem Game Over Screen
Input	Benutzer klickt auf Menu.
Erwarteter Output	Das Menu wird angezeigt.

Testfall-Nr. - Titel	TF_013 – Geschwindigkeit erhöhen
Beschreibung	Die Spielgeschwindigkeit wird laufend erhöht.
Anforderungs-Nr.	Anforderung_007
Voraussetzung	• Spiel ist gestartet
Input	Der Benutzer spielt das Spiel ganz normal ohne spezielle Eingaben.
Erwarteter Output	Das Spiel wird sporadisch schneller.

Testfall-Nr. - Titel	TF_014 – Score Anzeige
Beschreibung	Dem Benutzer wird während des Spiels laufend der aktuelle Score angezeigt.
Anforderungs-Nr.	Anforderung_008
Voraussetzung	• Spiel ist gestartet
Input	Der Benutzer spielt das Spiel ganz normal ohne spezielle Eingaben.
Erwarteter Output	Je höher das Level, desto schneller wird der Score laufend hochgezählt.

Testfall-Nr. - Titel	TF_015 – Kameraperspektive
Beschreibung	Der Benutzer sieht die Spielfigur immer von der Perspektive diagonal hinten/oben.
Anforderungs-Nr.	Anforderung_009
Voraussetzung	• Spiel ist gestartet
Input	Der Benutzer spielt das Spiel ganz normal ohne spezielle Eingaben.
Erwarteter Output	Die Spielfigur ist für den Benutzer immer von der Perspektive diagonal hinten/oben ersichtlich.

Testfall-Nr. - Titel	TF_016 – Highscore Ansicht
Beschreibung	Dem Benutzer wird im Menu der lokale Highscore angezeigt.
Anforderungs-Nr.	Anforderung_010
Voraussetzung	• Spiel ist gestartet • Benutzer befindet sich auf dem Menu Screen
Input	Der Benutzer spielt das Spiel zweimal bis zu einem Game Over.
Erwarteter Output	Der höchst erreichte Score in den gespielten Runden erscheint in der Menu Ansicht als Highscore.

Testfall-Nr. - Titel	TF_017 – Spielfigur springt
Beschreibung	Beim Drücken der Leertaste soll die Spielfigur während dem laufen einmal in die Luft springen.
Anforderungs-Nr.	Anforderung_011
Voraussetzung	• Spiel ist gestartet
Input	Der Benutzer drückt die Leertaste.
Erwarteter Output	Die Spielfigur springt während dem laufen einmal in die Luft.

Testfall-Nr. - Titel	TF_018 – Spielfigur im Sprung steuern
Beschreibung	Während einem Sprung soll die Spielfigur nicht steuerbar sein.
Anforderungs-Nr.	Anforderung_011
Voraussetzung	• Spiel ist gestartet
Input	Der Benutzer drückt die Leertaste und während dem Sprung die rechte Pfeiltaste.
Erwarteter Output	Die Spielfigur springt während dem laufen einmal in die Luft, ohne nach rechts auszuweichen.

Tabelle 5 – Benutzertests

9.3 Testprotokoll

9.3.1 Testdurchlauf 1

Testfall-Nr.	Testfall	Effektives Ergebnis	Status
TF_001	Spiel starten	EXE Datei öffnet sich und das Menu wird angezeigt.	**Erfolgreich**
TF_002	Play Button	Das Spiel beginnt und die Spielfigur läuft stetig gerade aus.	**Erfolgreich**
TF_003	Exit Button	Die Applikation wird beendet.	**Erfolgreich**
TF_004	Spielfigur läuft stetig	Spielfigur läuft stetig vorwärts und die Laufstrecke «erneuert» sich endlos, bis zu einem Game Over. Jedoch läuft die Spielfigur ein wenig in der Luft und nicht direkt auf dem Boden.	**Fehlgeschlagen**
TF_005	Keine Eingabe	Spielfigur läuft in das nächste Hindernis und der Game Over Screen wird angezeigt.	**Erfolgreich**
TF_006	Spielfigur weicht aus	Spielfigur weicht auf die linke Seite aus.	**Erfolgreich**
TF_007	Unerwartete Tastatureingabe	Spiel läuft normal weiter.	**Erfolgreich**
TF_008	Unerwartete Mauseingabe	Spiel läuft normal weiter.	**Erfolgreich**
TF_009	Hindernis-Abschnitte tauchen auf	Es tauchen in zufälliger Reihenfolge laufend neue Hindernis-Abschnitte auf.	**Erfolgreich**
TF_010	Kollisionscheck	Teilweise gibt es eine optische Fehlkollision, da der Character-Collider während des Sprungs über die Spielfigur herausragt.	**Fehlgeschlagen**
TF_011	Restart	Spiel startet sich neu.	**Erfolgreich**
TF_012	Menu	Das Menu wird angezeigt.	**Erfolgreich**
TF_013	Geschwindigkeit erhöhen	Das Spiel wird sporadisch schneller.	**Erfolgreich**
TF_014	Score Anzeige	Je höher das Level, desto schneller wird der Score laufend hochgezählt.	**Erfolgreich**
TF_015	Kameraperspektive	Die Spielfigur ist für den Benutzer immer von der Perspektive diagonal hinten/oben ersichtlich.	**Erfolgreich**
TF_016	Highscore Ansicht	Der höchst erreichte Score in den gespielten Runden erscheint in der Menu Ansicht als Highscore.	**Erfolgreich**
TF_017	Spielfigur springt	Die Spielfigur springt während dem laufen einmal in die Luft.	**Erfolgreich**
TF_018	Spielfigur im Sprung steuern	Die Spielfigur ist während dem Sprung weder links noch rechts steuerbar, jedoch kann sie erneut springen, während sie in der Luft ist.	**Fehlgeschlagen**

Tabelle 6 – Testprotokoll Testdurchlauf 1

9.3.2 Testdurchlauf 2

Anforderungs-Nr.	Testfall	Effektives Ergebnis	Status
TF_001	Spiel starten	EXE Datei öffnet sich und das Menu wird angezeigt.	**Erfolgreich**
TF_002	Play Button	Das Spiel beginnt und die Spielfigur läuft stetig gerade aus.	**Erfolgreich**
TF_003	Exit Button	Die Applikation wird beendet.	**Erfolgreich**
TF_004	Spielfigur läuft stetig	Spielfigur läuft stetig vorwärts und die Laufstrecke «erneuert» sich endlos, bis zu einem Game Over.	**Erfolgreich**
TF_005	Keine Eingabe	Spielfigur läuft in das nächste Hindernis und der Game Over Screen wird angezeigt.	**Erfolgreich**
TF_006	Spielfigur weicht aus	Spielfigur weicht auf die linke Seite aus.	**Erfolgreich**
TF_007	Unerwartete Tastatureingabe	Spiel läuft normal weiter.	**Erfolgreich**
TF_008	Unerwartete Mauseingabe	Spiel läuft normal weiter.	**Erfolgreich**
TF_009	Hindernis-Abschnitte tauchen auf	Es tauchen in zufälliger Reihenfolge laufend neue Hindernis-Abschnitte auf.	**Erfolgreich**
TF_010	Kollisionscheck	Es führt zu einer Kollision und der Game Over Screen wird angezeigt.	**Erfolgreich**
TF_011	Restart	Spiel startet sich neu.	**Erfolgreich**
TF_012	Menu	Das Menu wird angezeigt.	**Erfolgreich**
TF_013	Geschwindigkeit erhöhen	Das Spiel wird sporadisch schneller.	**Erfolgreich**
TF_014	Score Anzeige	Je höher das Level, desto schneller wird der Score laufend hochgezählt.	**Erfolgreich**
TF_015	Kameraperspektive	Die Spielfigur ist für den Benutzer immer von der Perspektive diagonal hinten/oben ersichtlich.	**Erfolgreich**
TF_016	Highscore Ansicht	Der höchst erreichte Score in den gespielten Runden erscheint in der Menu Ansicht als Highscore.	**Erfolgreich**
TF_017	Spielfigur springt	Die Spielfigur springt während dem laufen einmal in die Luft.	**Erfolgreich**
TF_018	Spielfigur im Sprung steuern	Die Spielfigur springt während dem laufen einmal in die Luft, ohne nach rechts auszuweichen.	**Erfolgreich**

Tabelle 7 – Testprotokoll Testdurchlauf 2

9.4 Testauswertung / Konsequenzen

Nach dem ersten Testdurchlauf schlugen nur drei von allen Testfällen fehl. Dies hat sicherlich damit zu tun, das während dem Realisieren fortlaufend getestet wurde und gefundene Fehler sofort eliminiert wurden. Trotz des guten Ergebnisses war ein zweiter Testdurchlauf nötig, um ein 100% erfolgreiches Testergebnis zu erreichen. Alle drei Fehler wurden anschliessend erfolgreich behoben und alle Tests bei einem zweiten Testdurchlauf erneut getestet. Nach dem zweiten Testdurchlauf waren alle Tests erfolgreich, das Testing abgeschlossen und eine vollständige und erfolgreiche Testabdeckung für alle Anforderungen erreicht.

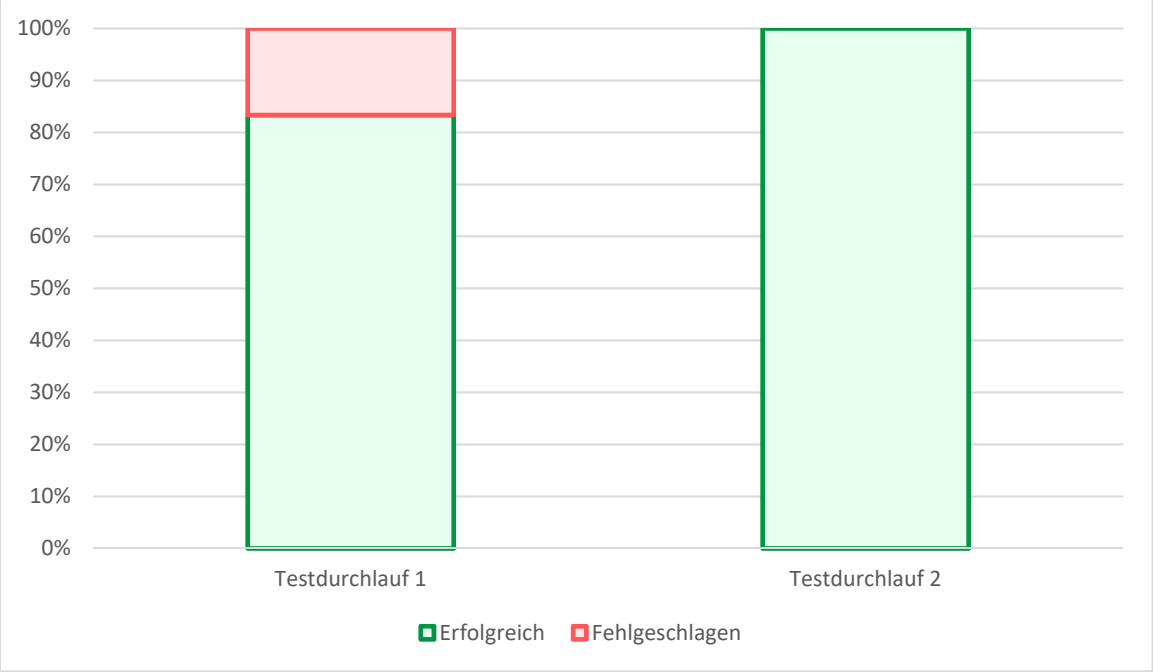

Abbildung 45 – Testauswertung

10 Reflexion

Zum ersten Mal habe ich ein Game mit einer Game Engine realisiert. Trotz ausgiebiger Analyse war es ohne Erfahrungswerte für die Planung sehr schwierig einzuschätzen, wie lange ich für die verschiedenen Realisierungstask brauche. Aus diesem Grund habe über die 285 Stunden Vorgabe hinaus noch zwei Kann-Anforderungen definiert, damit ich sicher auf die benötigten Stunden komme, falls ich mich bei manchen Tasks überschätze.

Der IST-Zustand wich ausgenommen von der Realisierungsphase kaum von der Planung ab. Bei den Realisierungsaufgaben gab es teils grosse Abweichungen. Manches nahm mir die Game Engine schon fast automatisch ab, was in einigen Aufgaben den Aufwand massiv verringerte. Dafür konnte ich beide definierten Kann-Anforderungen zu meiner Erfreunis umsetzten. Das Erstellen der Spielfigur-Grafik mit Blender unterschätzte ich dafür enorm. Es war sehr viel Fleissarbeit nötig, bis die Spielfigur einigermassen so aussah, wie sie sollte und dadurch, dass ich mit dem Tool noch nicht vertraut war, dauerte es nochmals länger.

Schlussendlich glich sich der Zeitplan dank den zwei definierten Kann-Anforderungen gut aus und der IST-Zustand beträgt 286 Stunden.

Ab dem ersten Testdurchlauf war ich positiv überrascht, dass nur drei Tests fehlschlugen. Und dies, obwohl die Arbeit für mich ziemlich Neuland war. Bereits nach dem zweiten Testdurchlauf waren alle Tests erfolgreich bestanden.

Mit dem Endresultat bin ich sehr zufrieden und bin stolz auf mich. Alle geforderten Anforderungen sind vollständig implementiert.

Abschliessend gibt es zu sagen, dass mir noch bewusster wurde, wie wichtig eine gute Analyse für Planung ist. Eine saubere Analyse erleichtert die Planung und später auch die Realisierung. Falls ich ein ähnliches Projekt nochmals durchführen müsste, würde ich die Analyse noch exakter durchführen, sodass mir die Planung leichter fällt.

In den 286 Stunden Aufwand sammelte ich viele Erfahrungen, welche mir im späteren Berufsleben bestimmt weiterhelfen werden.

11 Glossar

Begriff	Erklärung
Assets	Alle Grafikelemente in einem Unity Game werden Assets genannt.
Cloud	Speicherplatz über das Internet
Components	Components sind die Eigenschaften eines Game Objects.
CPU	Englische Abkürzung für Central Processing Unit, im deutschen Prozessor genannt.
Game Engine	Framework für Computerspiele
Game Objects	Game Objects sind ein Container für diverse Components und Assets.
GitLab	Versionsverwaltungssystem
Ironman	Superheldenfigur
o.J.	Abkürzung für «ohne Jahr»
OS	Englische Abkürzung für Operating System, im deutschen Betriebssystem genannt.
PlayerPrefs	Ein automatisch von Unity erstelltes lokales File, um zum Beispiel einen Highscore zu speichern.
Scenes	Eine Scene ist in Unity eine Umgebung.
Screenshot	Bildschirmfoto
Road Runner	Road Runner ist der Name des entwickelten Spiels.
Unity	Unity ist eine bekannte Game Engine.
Unity Asset Store	Der Unity Asset Store wird von Unity zur Verfügung gestellt, um bereits erstellte Assets herunterzuladen.
Zip-File	Ein Zip-File ist ein Dateiformat für verlustfreie und komprimierte Daten.

Tabelle 8 – Glossar

12 Quellenangaben

12.1 Literaturverzeichnis

Artstation. (o.J.). Abgerufen am 9. November 2018 von https://cdnb.artstation.com/p/assets/images/images/002/669/897/large/u-ri-so-zz-body-title1.jpg?1464335522

Autodesk. (o.J.). Abgerufen am 20. Oktober 2018 von https://www.autodesk.com/products/maya/subscribe?geoNavigationPreferredSite=US&plc=MAYA&term=1-YEAR&support=ADVANCED&quantity=1

Céline Hofstetter. (2015). Abgerufen am 12. Oktober 2018 von Individuelle praktische Arbeit

CryEngine. (o.J.). Abgerufen am 20. Oktober 2018 von https://www.cryengine.com/user/registration

Pixologic. (o.J.). Abgerufen am 20. Oktober 2018 von https://store.pixologic.com/zbrush-2018/single-user-license/

Unity Asset Store. (1. Dezember 2018).

Unity. (o.J.). Abgerufen am 20. Oktober 2018 von https://store.unity.com/de

Unity. (o.J.). Abgerufen am 19. Oktober 2018 von https://unity3d.com/de/unity/system-requirements

Unity3d. (o.J.). Abgerufen am 03. November 2018 von https://docs.unity3d.com/Manual/ExecutionOrder.html

Wikimedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://upload.wikimedia.org/wikipedia/commons/thumb/3/3c/Logo_Blender.svg/2000px-Logo_Blender.svg.png

Wikimedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://upload.wikimedia.org/wikipedia/commons/thumb/d/da/Unreal_Engine_Logo.svg/2000px-Unreal_Engine_Logo.svg.png

Wikimedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://upload.wikimedia.org/wikipedia/commons/thumb/1/19/Unity_Technologies_logo.svg/2000px-Unity_Technologies_logo.svg.png

Wikimedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://upload.wikimedia.org/wikipedia/commons/8/8d/CryEngine_Nex-Gen%284th_Generation%29.png

Wikimedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://upload.wikimedia.org/wikipedia/en/7/75/Logo_of_Maya.png

Wikipedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://de.wikipedia.org/wiki/Unreal_Engine

Wikipedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://de.wikipedia.org/wiki/Unity_(Spiel-Engine)

Wikipedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://de.wikipedia.org/wiki/CryEngine

Wikipedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://de.wikipedia.org/wiki/Maya_(Software)

Wikipedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://de.wikipedia.org/wiki/Blender_(Software)

Wikipedia. (o.J.). Abgerufen am 20. Oktober 2018 von https://de.wikipedia.org/wiki/ZBrush

Zendesk. (o.J.). Abgerufen am 20. Oktober 2018 von https://fso.zendesk.com/hc/article_attachments/360000085206/zbrush-logo.png

12.2 Abbildungsverzeichnis

12.3 Tabellenverzeichnis

13 Anhang

- Eigenständigkeitserklärung
- Aufgabenstellung
- Ordnerstruktur und Inhalt auf USB Stick:
  - **Diagramme:** Alle erstellten Diagramme als PDF-Dateien
  - **Dokumente:** Dokumentation und Zeitplan als PDF-Dateien
  - **Software:** Skripte als PDF-Dateien, Installationsdatei und gesamtes Software Projekt

Bildungszentrum Uster
Höhere Berufsbildung Uster
HBU

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Vordiplom- oder Diplomarbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

Verfasst von (in Druckschrift):

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	Vorname(n):
_____	_____
_____	_____
_____	_____
_____	_____

Ich bestätige mit meiner Unterschrift:

- Ich habe keine im „HFU Leitfaden schriftliche Arbeiten“ beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.
- nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	Unterschrift(en)
_____	_____
	_____
	_____
	_____

Bildungszentrum Uster
Höhere Fachschule Uster
Technik Informatik Wirtschaft Führung
Berufsschulstrasse 1
8610 Uster

Höhere Fachschule Uster, Berufsschulstrasse 1, 8610 Uster

Frau
Céline Hofstetter
Am Klusbach 8
8616 Riedikon

6. Oktober 2018

Diplomarbeit 2018-19

Sehr geehrte Frau Hofstetter

Sie erhalten folgende Diplomarbeit zugeteilt, welche Sie selbständig zu lösen haben.

«3D Endless Runner» - Game Entwicklung

Beschreibung
Ziel dieser Diplomarbeit ist es ein 3-dimensionales «Endless Runner» Game zu realisieren. Dazu soll eine Game Engine und die Programmiersprache C# verwendet werden. Das Spiel soll auf einem Windows PC laufen.

Aufgabenstellung
Entwickeln Sie das 3D Spiel «**Endless Runner**» auf Basis einer Game Engine wie z.B. Unity und der Programmiersprache C#.

Gliedern Sie dabei nach folgenden Teilaufgaben:
- Erstellung des Projektplans
- Erstellen des Pflichtenhefts mit den Anforderungen an die Lösung
- Design der Software Architektur, Spiellogik / -konzept, Spielfiguren, Datenhaltung
- Design und Realisierung der 3D Spiel- und Bedienoberfläche
- Realisierung der Lösung
- Dokumentieren der Test Fälle und der Test Resultate
- Erstellen der Dokumentation

Versand der Aufgabenstellung **9. Oktober 2018**
Abgabe der 2 Dokumentationen **4. März 2019**

Freundliche Grüsse

Mario Sabbatella