Diploma Thesis

Höhere Technische Bundeslehranstalt Leonding
Abteilung für Medientechnik

Writing the LaTeX Way under consideration of long titles which must not spoil the title page

Submitted by:	**Peter Bauer, 5AHIF**
	Richard Kainerstorfer, 5BHIF
	Alfred Wiedermann, 5AHBG
	Wolfgang Holzer, 5AHEL
Date:	**April 4, 2018**
Supervisor:	**Peter Bauer**
Project Partner:	**MathConsult**

Declaration of Academic Honesty

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such.

This paper has neither been previously submitted to another authority nor has it been published yet.

Leonding, April 4, 2018

Peter Bauer, Richard Kainerstorfer, Alfred Wiedermann, Wolfgang Holzer

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorgelegte Diplomarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Gedanken, die aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Leonding, am 4. April 2018

Peter Bauer, Richard Kainerstorfer, Alfred Wiedermann, Wolfgang Holzer

Abstract

Here it is described what the thesis is all about. The abstract shall be brief and concise and its size shall not go beyond one page. Furthermore it has no chapters, sections etc. Paragraphs can be used to structure the abstract. If necessary one can also use bullet point lists but care must be taken that also in this part of the text full sentences and a clearly readable structure are required.

Concerning the content the following points shall be covered.

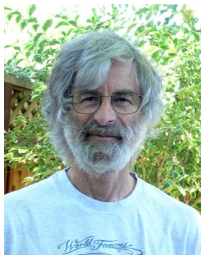
1. *Definition of the project:* What do we currently know about the topic or on which results can the work be based? What is the goal of the project? Who can use the results of the project?

2. *Implementation:* What are the tools and methods used to implement the project?

3. *Results:* What is the final result of the project?

This list does not mean that the abstract must strictly follow this structure. Rather it should be understood in that way that these points shall be described such that the reader is animated to dig further into the thesis.

Finally it is required to add a representative image which describes your project best. The image here shows Leslie Lamport the inventor of LaTeX.



Zusammenfassung

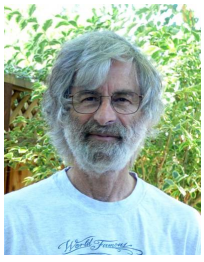
An dieser Stelle wird beschrieben, worum es in der Diplomarbeit geht. Die Zusammenfassung soll kurz und prägnant sein und den Umfang einer Seite nicht übersteigen. Weiters ist zu beachten, dass hier keine Kapitel oder Abschnitte zur Strukturierung verwendet werden. Die Verwendung von Absätzen ist zulässig. Wenn notwendig, können auch Aufzählungslisten verwendet werden. Dabei ist aber zu beachten, dass auch in der Zusammenfassung vollständige Sätze gefordert sind.

Bezüglich des Inhalts sollen folgende Punkte in der Zusammenfassung vorkommen:

- *Aufgabenstellung:* Von welchem Wissenstand kann man im Umfeld der Aufgabenstellung ausgehen? Was ist das Ziel des Projekts? Wer kann die Ergebnisse der Arbeit benutzen?
- *Umsetzung:* Welche fachtheoretischen oder -praktischen Methoden wurden bei der Umsetzung verwendet?
- *Ergebnisse:* Was ist das endgültige Ergebnis der Arbeit?

Diese Liste soll als Sammlung von inhaltlichen Punkten für die Zusammenfassung verstanden werden. Die konkrete Gliederung und Reihung der Punkte ist den Autoren überlassen. Zu beachten ist, dass der/die LeserIn beim Lesen dieses Teils Lust bekommt, diese Arbeit weiter zu lesen.

Abschließend soll die Zusammenfassung noch ein Foto zeigen, das das beschriebene Projekt am besten repräsentiert. Das folgende Bild zeigt Leslie Lamport, den Erfinder von LaTeX.



Acknowledgments

If you feel like saying thanks to your grandma and/or other relatives.

Contents

Chapter 1

Introduction

1.1 Initial Situation

Common word processors do not prepare print-like documents in so far as these programs do not reflect the rules of professional printing which have been grown over centuries. These rules contain clear requirements for balancing page layouts, the amount of white space on pages, font-handling, etc. Donald Knuth's TeX package (see [Knu84]) is a word processor which conforms to these printing rules. This package was enhanced by Leslie Lamport by providing more text structuring commands. He called his package LaTeX [Lam85].

When preparing a thesis, we want not only to have our content on a top level, we also want to commit to a high level of formal criteria. Therefore, we request our students to use one of these professional printing production environments like TeX or LaTeX.

Furthermore students should train their scientific writing skills. This includes a clear and structured break-down of their ideas, a high-level and clear wording, and the training of transparent citations of ideas from other sources than from theirs. A good source for more information concerning technical and scientific writing can be found in [Rec06].

1.2 Goals

The general goals and objectives of the project are described here. Care must be taken that the goals documented here are purely project goals and have nothing to do with individual goals of the team members. If individual goals should be part of the thesis they are listed in appendix B.

1.3 Overview

Details of the diploma thesis have to be aligned between student and supervisor. This should be a basic structure to facilitate the first steps when students start to write their theses.

Figure 1.1: Don Knuth, the inventor of TEX

Never forget to add some illustrative images. Images must not be messed up with your normal text. They are encapsulated in floating bodies and referenced in your text. An example can be seen in figure 1.1. As you can see, figures are placed by default on top of the page nearby the place where they are referenced the first time. Furthermore you can see that a list of figures is maintained automatically which can be included easily by typing the command `\listoffigures` into your document.

1.4 Basic Terminology

As usual the very basic terminology is briefly explained here. Most probably the explanations here only scratch a surface level. More detailed explanations of terminology goes into chapter 2.

1.5 Related Work and Projects

Here a survey of other work in and around the area of the thesis is given. The reader shall see that the authors of the thesis know their field well and understand the developments there. Furthermore here is a good place to show what relevance the thesis in its field has.

1.6 Structure of the Thesis

Finally the reader is given a brief description what (s)he can expect in the thesis. Each chapter is introduced with a paragraph roughly describing its content.

Chapter 2

Theoretical Background

The details of the structure of your thesis have to be aligned with the supervising teacher. However, most of the theses require to have some description of the models used or some other theoretical background necessary to understand the rest of the text.

Since there is enough space here a table is added to show the basic usage of tables in a scientific document. Similarly to images these are also kept outside the normal text flow in a so-called floating body. Table 2.1 shows different options.

Body type	Floats
Image	Always
Table	Always
Algorithm	Sometimes

Table 2.1: Different types of floating bodies

Chapter 3

Summary

Here you give a summary of your results and experiences. You can add also some design alternatives you considered, but kicked out later. Furthermore you might have some ideas how to drive the work you accomplished in further directions.

Chapter 4

Verwendete Technologien

4.1 Unity

Unity ist eine Entwicklungsumgebung hauptsächlich gedacht für die Spieleentwicklung. Eigentümer des Produkts ist Unity Technologies, die ihren Sitz in San Francisco haben. Das Unternehmen ist seit 2004 in Betrieb und arbeitet ausschließlich an Unity und dessen Services. Die wichtigsten Services hierbei sind Unity Multiplayer, Remote Settings, der Unity Asset Store und Unity Collaborate.

4.1.1 Unity Multiplayer

Unity Multiplayer soll es Entwicklern ermöglichen leicht und schnell, auf Mehrspieler basierende, Spiele zu entwickeln. Der Service bietet Support für die Synchronisierung von Objektpositionen zwischen den Geräten, Serverseitige Commands, die Cheating deutlich erschweren, eine Unterscheidung zwischen Client und Server im Code, sodass Entwickler auch für Einzelspieler gedachte Projekte leicht umbauen können, und vieles mehr. Der wohl größte Vorteil ist die Inkludierung eines Lobbysystems. Spieler können Lobbys erstellen und diese für andere Spieler freischalten, sodass diese der Gruppe beitreten können. Immer mehr Mehrspielerspiele, wie „Battlefield“, „Counterstrike“ oder „Destiny“, benutzen Matchmaking um Spieler, die in derselben Region spielen, in Lobbys zusammenzuschließen, um sie zu füllen. Die Spieleindustrie geht immer mehr in die Richtung, der Servicespiele. Der Grund ist, dass Spiele immer teurer werden und große Publisher nicht mehr mit den 60 Euro pro Spiel auskommen. Sie versuchen die Spiele weiter zu monetarisieren, indem sie Spielinhalte ohne zusätzliche Kosten erst spät oder gar nicht erreichbar machen. Je langlebiger die Spiele sind, desto eher ist die Wahrscheinlichkeit, dass Spieler bereit sind mehr dafür zu zahlen. Erreicht wird diese Langlebigkeit durch periodische Updates und nicht zu Letzt durch Mehrspielermodi. Spiele wie „The Division“, „Destiny“ oder „Anthem“ werden in vergleichbar schlechten Zustand veröffentlicht, mit einer großen Karte zum Erkunden und einem Mehrspielermodus. Während der nächsten Jahre werden Bugs behoben und Inhalte hinzugefügt um Spieler zurück zu bringen. Dieser Trend ist in der jetzigen Form zwar nicht wünschenswert, es ist jedoch trotzdem wichtig, dass Entwickler die Möglichkeit dazu haben.

4.1.2 Remote Settings

Viele Mehrspielerspiele, wie „First-Person-Shooter“, MOBAs oder MMOs, benötigen regelmäßige Patches, um die Spielbalance zu verändern. Dieses Thema ist enorm komplex, da viele Räder ineinander greifen. „League of Legends“ ist seit über 10 Jahren, mit

über 200 Millionen registrierten Nutzern, aktiv. Unabhängig davon wird alle 14 Tage ein Patch ausgerollt, der bestimmte Charaktere stärker und andere wieder schwächer macht. Diese Patches will der Service „Remote Settings“ ablösen, indem er Entwicklern die Möglichkeit gibt, bestimmte Werte mit der Cloud zu synchronisieren, sodass diese jederzeit über ein Webinterface, ohne zusätzliche Patches, geändert werden können. Patches sind somit nur mehr für Fehlerbehebungen und zusätzliche Inhalte nötig.

4.1.3 Unity Asset Store

Der Asset Store ist direkt in die Engine eingebaut und ist somit ein zentraler Aspekt von Unity. Er ist vor allem für Prototypen enorm wichtig und hilft Entwicklern schnell ihre Ideen zu testen und verbessern. Im Store können Nutzer Scripts, Modelle, Animationen, Tools und vieles mehr hochladen und auch herunterladen. Eine simple, aber effektive Review-Implementation ist vorhanden um hochwertige Assets zu präferieren. Inhalte sind für fast jedes Szenario und jede Idee vorhanden und deren Preise reichen von kostenlos bis zu mehr als 100 Euro. Es sind viele Modellpakete vorhanden, die oft gratis sind und für die Prototypenphase, als Platzhalter, verwendet werden. Viele Entwickler laden auch selbst entwickelte Tools, für die Kameraführung oder die Terraingestaltung, hoch. 2017 sparten Entwickler über 1 Milliarde Dollar durch die Benutzung von Assets aus dem Store. Der Store ist aber auch zum Teil Schuld an der schlechten Reputation, die Unity, seit einiger Zeit, plagt. Unzählige Spiele werden täglich in die mobilen App Stores hochgeladen, die nur aus Asset Store Inhalten bestehen, um billige Kopien bekannter Spiele zu machen. Auch ein Grund ist, die Verwendung der Marke Unity selbst. Unity bietet eine Gratisversion, dessen größte Unterscheidung die Inkludierung des Unity Logos, am Start des Spiels, ist. Bei Bezahlversionen hat man die Wahl und da die Erscheinung des Logos meist für billige Spiele steht, entschließen sich die meisten dagegen. Die Zeit des Hochfahrens wird bei Spielen meist für die zur Schau Stellung der Technologien verwendet und die Gameengines in Verwendung entwerfen oft angepasste Logos für wichtige Spieleveröffentlichungen. Unity ist die meist verwendetste Engine am Markt und auch die erste Wahl für viele Indie Entwickler. Spiele, wie „Ori and the Blind“, „Cuphead“, „Superhot“, „Hollow Knight“, „Subnautica“ oder „Overcooked“, sind eine der größten Indie Erfolgsgeschichten in den letzten Jahren aber auch Smartphone Klassiker, wie „Angry Birds“, „Temple Run“, „Crossy Road“ oder „Pokemon Go“ sind mit Unity entwickelt worden. Jedoch erscheint das Unity Logo kaum bei Spielstart, weswegen man diese Spiele nicht sofort damit in Verbindung bringt.

4.1.4 Unity Collaborate

Unity Collaborate ist Unitys in die Engine eingebaute Git Alternative, die ein einfaches Zusammenarbeiten ermöglichen soll. Es gibt gewisse Einschränkungen, die unter gewissen Bedingungen KO-Kriterien sein können. Für Teams bis vier Personen ist der Dienst zwar gratis, werden Accounts jedoch auch mehreren Geräten genutzt, kann es zu Fehlern kommen, die behoben werden können, sofern man sich für eine der Bezahloptionen entscheidet. Unity Collaborate ist ein simples Tool, jedoch ist das neben der größten

Stärke auch eine nicht zu vernachlässigende Schwäche. Ohne Erweiterungen kann der Dienst keine Dateiinhalte lesen und so Dateien zusammenfügen, wie es Git zum Beispiel kann. Es kann nur Unterscheiden in geändert oder eben nicht. Je nach Workflow kann das zum Problem werden. Ist eine strikte Arbeitsteilung vorhanden, so reicht diese Implementierung aus und macht den Dienst, so zur idealen Wahl. Arbeiten oft mehrere Leute an der selben Datei, so kann man Erweiterungen einbauen, die zusätzliche Funktionalität bieten können. Leider bietet Unity keine solchen Erweiterungen und man ist auf Dritte angewiesen.

4.1.5 Unity Preispolitik

Unity bietet hier drei Optionen, die sich abhängig von der Teamgröße, im Preis ändern können. Für Einsteiger und Solo Entwickler ist die Gratis Version von Unity ausreichend. Man ist in keiner Weise in den Plattformen, für die Spiele entwickelt werden können, eingeschränkt und auch der Großteil der Services ist gratis verfügbar. Support wird von Unity nicht bereit gestellt. Um diesen in eingeschränkter Form genießen zu können, wird die, 25 Dollar pro Monat kostende, Plus Version benötigt. Zusätzlicher Cloud Speicherplatz und Zugang zu einem Übungskurs sind zusätzliche Boni. Um sowohl einen „Dark Mode“, als auch ein T-Shirt zu bekommen, benötigt man die, 125 Dollar kostende, Pro Version, die alle Features, wie 20 Prozent Rabatt auf Premium Assets, mehr Cloud Speicherplatz, Zugang zu Entwicklerkonferenzen und vieles mehr freischaltet. Für größere Teams bietet Unity individuelle Lösungen. Selbst wenn man die zusätzlichen Features der Bezahlversionen nicht benötigt, ist man ab einem gewissen Umsatz dazu verplichtet.

4.1.6 Plattformen

Unity unterstützt seit vielen Jahren alle neuen Plattformen, selbst wenn diese selbst noch in Entwicklung sind. Neben den Standardplattformen, wie X-Box, Playstation, Nintendo Switch, Android, Ios, Windows, Linux und Mac werden auch Fernseherbetriebssysteme, wie Samsungs Tizen OS, Apples tvOS oder Googles Android TV unterstützt. Plattformen der Zukunft wie Microsoft Mixed Reality, Gear VR, Google Cardboard, AR-Core und viele mehr, die sich auf Augmented Reality oder Virtual Reality fokussieren, werden zahlreich unterstützt, weshalb Unity hier eine Marktdominanz zeigt. Spiele werden meist für mehrere Plattformen veröffentlicht und ein Vorteil, den Unity bietet, ist es, die selbe Codebasis für alle Plattformen zu verwenden. Ludimus hat sowohl einen Android, als auch einen Windows Client, dennoch können beide Plattformen mit dem selben Code funktionieren. Muss oder soll es jedoch Unterscheidungen zwischen Plattformen geben, so bietet Unity auch hier eine Lösung. Plattformspezifischer Code wird für alle nicht relevanten Plattformen, aus dem Code heraus geschnitten. Häufige Anwendungen hierfür sind Filesysteme und deren verschiedene Pfade.

4.1.7 Mobile Entwicklung

Neben der Virtual und der Augmented Reality, ist Unitys Hauptverwendungszweck die mobile Entwicklung für Ios und Android. Der Vorteil der gleichen Codebasis ist hier enorm wichtig, da diese zwei Systeme sehr unterschiedliche Entwicklungsworkflows haben und es nicht effizient ist, für mehrere Plattformen einzeln zu entwickeln. Selbst große Publisher wie Square Enix, greifen bei ihrem „Tomb Raider“ Smartphone Ableger auf Unity zurück, da sie laut eigener Angabe schnell einen vorzeigbaren Prototypen bauen konnten. Auch Blizzard benutzte Unity zur Entwicklung von ihrem Kartenspiel „Hearthstone“, welches sowohl einen Android und Ios, als auch einen Windows Client, besitzt und auch sie erwähnen die schnelle Entwicklungszeit. Es sind Erfolgsgeschichten, wie diese weswegen Unity unter Entwicklern einen sehr guten Ruf genießt, auch wenn die Öffentlichkeit davon nichts mitbekommt.

4.1.8 Community

Durch den großen Erfolg bildete sich schon zu Beginn eine riesige Community, die durch Tutorials und Livestreams von Unity selbst inspiriert wurde. Hunderte hochqualifizierte Entwickler teilen deren Wissen mit Anfängern und helfen ihnen auf ihrem Weg zum ersten Erfolg. Die Tutorials reichen von Programmierkursen, über die Modellierung, bis hin zur Integration anderer Dienste in Unity. Die Gemeinschaft ist groß genug, sodass jedes gewünschte Verhalten dokumentiert worden ist, oder jeder aufgetretene Fehler behoben worden ist.

4.1.9 Integration

Unity bietet Integrationen mit vielen Modellierungs-, Zeichen- und Audiotools. Die zwei am meisten vorkommenden Kombinationen sind wohl: Unity und Blender und Unity und Photoshop.

4.1.9.1 Unity und Blender

Blender ist eine gratis Open Source Modellierungssoftware, die in viele Formate exportieren kann und auch eine ähnliche Community, wie Unity, bietet. In vielen Tutorials werden Unity und Blender gemeinsam verwendet, da beide Programme zwar Einsteigerfreundlich sind, jedoch auch enormes Potenzial haben, sofern gebraucht.

4.1.9.2 Unity und Photoshop

Für die 2D Spieleentwicklung sind Sprites, also Bilder, enorm wichtig. Viele Guides empfehlen hier meist Photoshop, den sowohl verschiedene Layer als auch Animationen werden perfekt unterstützt, um einen reibungslosen Workflow zu garantieren.

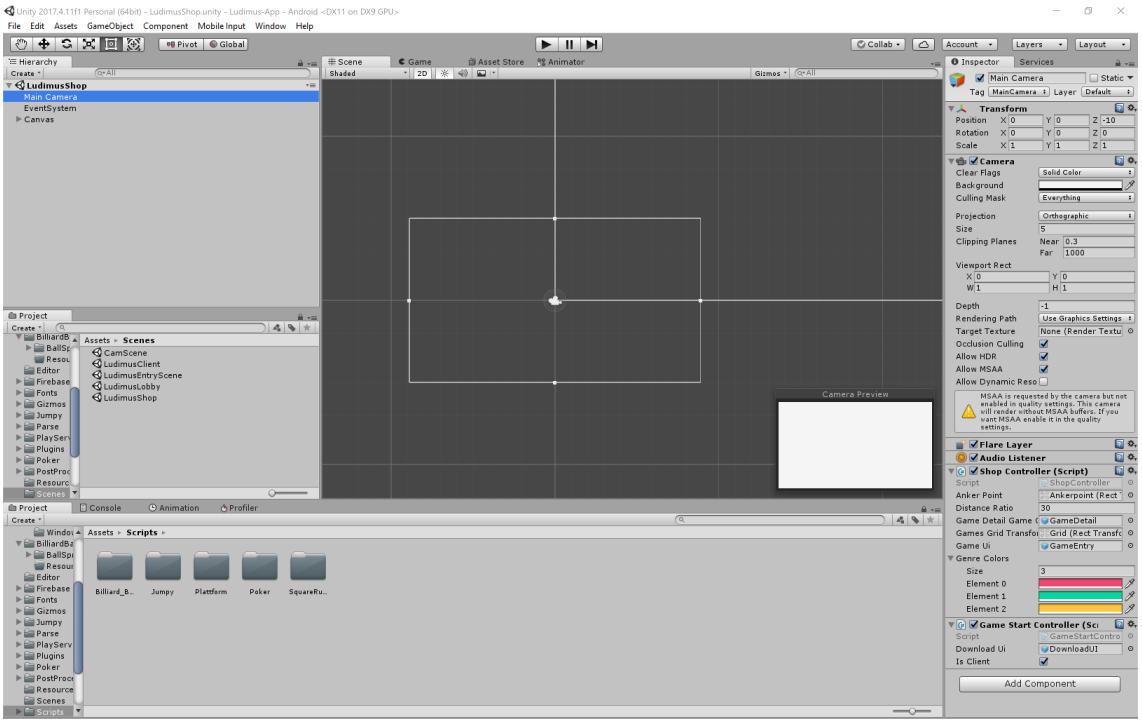


Figure 4.1: Unity Editor

4.2 Entwicklung mit Unity

Die Entwicklung in Unity ist in viele kleine Bereiche aufgeteilt. Logik implementiert man jedoch immer in C-Sharp, sofern man keine explizite Lizenz für C++ erworben hat. Im Laufe des Jahres 2019, soll jedoch eine Visual Scripting Lösung getestet werden. Visual Scripting ist eine Drag and Drop Art der Programmierung. Ein Programm besteht hier aus Nodes. Diese Nodes haben bestimmte Funktionen, die miteinander verbunden werden können. Entwickler können eigene Nodes schreiben, um die Grundfunktionalität zu erweitern. Das bekannteste Beispiel dieser Entwicklungsmethode ist die Unreal Engine 4, die dieses Konzept schon seit Beginn nutzt.

4.2.1 Editor

Der Editor ist in Tabs gegliedert, wie in Abbildung 4.1 zu sehen, die beliebig angeordnet werden können. Wichtige Tabs sind:

4.2.1.1 Hierarchy

Hier werden alle Objekte, die in der aktuell ausgewählten Szene sind, angezeigt. Objekte können verschachtelt sein oder unabhängig voneinander herumverschoben werden. Die Details der ausgewählten Elemente werden im Inspector angezeigt.

4.2.1.2 Scene

Das Scene Fenster zeigt die Welt, die aktuell ausgewählte Szene, in der die Objekte existieren. Alle Objekte haben eine eindeutige Position, mit einer x, y und z Koordinate. Der Benutzer kann sich in diesem Dreidimensionalen Raum frei bewegen und Objekte anordnen, um seine Level zu gestalten. Da Unity sowohl 2D als auch 3D Entwicklung unterstützt, kann man die Perspektive einstellen.

4.2.1.3 Project

Der Project Tab die In Engine Version des Fileexplorers. Er verfügt über eine schnellere Suchfunktion und ein Favoritensystem. Hier ausgewählte Objekte können sowohl in die aktuelle Szene hineingezogen werden oder die Details, im Inspector, angesehen werden.

4.2.1.4 Console

Die Konsole zeigt alle Konsolen Einträge auf, gruppiert nach Art, sprich „log“, „warning“ oder „error“. Wählt man einen Eintrag aus, so sieht man dessen Details und kann auch zur Zeile im Code springen, der diesen ausgelöst hat. Die Standard Lösung von Unity verfügt über keine Suchfunktion, jedoch gibt es Tools im Asset Store, die diese Lücke füllen.

4.2.1.5 Animation

Soll sein Spiel über bewegliche Inhalte verfügen, so sind Animationen oft eine gute Lösung. Unity verwendet hier das Keyframe Prinzip, bei dem alle Eigenschaften zu zwei bestimmten Zeitpunkten, Keyframes, gespeichert werden, und anschließend alle Änderungen über die Zeit verteilt durch geführt werden. Unity unterstützt diese Art der Animation für 3D-Modelle, Sprites und UI-Elemente. 3D-Modelle und Sprites können mit einem Skelett versehen werden, dass bewegt werden kann um auch das Modell selbst zu bewegen.

4.2.1.6 Game

Ist es Zeit das Programm zu testen, so kann man es mithilfe des großen Spielen-Knopf starten. Unity wechselt danach automatisch in den Game Tab. Hier sieht man seine Szene durch eine Kamera, genauso wie sie Spieler später sehen würden. Dieser Modus ist für schnelle Testläufe ideal. Es gibt jedoch unterschiede zwischen dieser spielbaren Version und einer gebauten App, weshalb es ratsam ist, beide Versionen zu überprüfen.

4.2.2 Komponenten

Objekte in Unity bestehen aus Komponenten. Diese geben ihnen ihre Funktionalität und bieten gleichzeitig Flexibilität. Alle vom Entwickler generierten Scripts erben automatisch von der Basisklasse MonoBehaviour, wodurch sie als Komponenten verwendet werden können. Als Basis bietet Unity jedoch schon eine Reihe an Komponenten an,

um Beispielsweise ein Objekt auf Gravitation reagieren zu lassen. Diese bereitgestellten Komponenten sind eingeteilt in: Mesh, Effects, Physics, Navigation, Audio, Video, Rendering, UI, AR und vielen mehr.

4.2.2.1 Mesh

Diese Gruppe besteht aus 3 verschiedenen und essentiellen Komponenten, für die 3D Entwicklung. Der Mesh Filter gibt an welches Modell dieses Objekt darstellen soll. Der Mesh Renderer hingegen beschreibt, welche Materialien das Objekt bekommt oder ob es Schatten erhalten oder werfen soll. Um 3D Text in der Szene anzuzeigen, benötigt man den Text Mesh Komponenten.

4.2.2.2 Effects

Egal ob Lagerfeuer, kleine Luftsteifen, um Bewegung deutlicher zu machen, oder Lens Flare Effekte, wenn der Spieler in die Sonne sieht, Effekte sind in Spielen häufig verwendet und Unity bietet hier eine gute Basis um seine eigenen Effekte zu erzeugen.

4.2.2.3 Physics

Dieser Punkt ist unterteilt in 2D und 3D Physics, um für Klarheit zu sorgen.

4.2.2.3.1 2D Physics Um die vorher erwähnte Gravitation für dieses Objekt zu aktivieren, benötigt man einen Rigidbody, hier konkret Rigidbody2D. Unity bietet eine globale Vektor Variable für die Gravitation, mit einer x und einer y Richtung. Im Rigidbody kann man die Stärke des Effekts auf dieses eine Objekt verändern. Die Werte für Masse, Trägheit und verschiedene Physics Materialien können eingetragen werden. Letztere bestimmen, wie stark das Objekt von Reibung beeinflusst wird und wie federnd es ist. Objekte können auch in ihrer Bewegung oder Rotation für verschiedene Achsen eingeschränkt werden. Zusätzlich zum Rigidbody findet man hier alle Arten von Collidern, die Unity für 2D zu bieten hat. Diese Collider beschreiben die Zone, in der Kollisionen registriert werden. Collider können miteinander verknüpft werden, um die Objekte perfekt einzuhüllen. Box-, Circle-, Edge- und Polygon-Collider sind die am häufigsten verwendeten. Während Box- und Circle-Collider nur geometrische Formen beschreiben und lediglich in deren Größe geändert werden können, bieten sowohl Edge-, als auch Polygon-Collider Möglichkeiten, die für 3D nicht denkbar sind. Edge-Collider werden oft für Plattformen verwendet, da sie nur eine Linie darstellen, die an verschiedenen Punkten abgebogen werden kann. Das selbe Prinzip verfolgt der Polygon-Collider, nur verbindet er Start- und Endpunkt miteinander. Hinzukommt eine automatische Erkennung, der Form, bei Sprites. Als dritte Gruppe dieser Kategorie gelten die Joints. In verschiedenen Ausführung regeln sie das Verhalten zwischen zwei Objekten. Räder, Dämpfungen, Seile und mehr können so realisiert werden. Zu guter Letzt gibt es noch Effectors. Diese regeln die Bewegungen, wenn zwei Objekte miteinander kollidieren. So können „One-Way“ Plattformen oder Förderbänder geschaffen werden.

4.2.2.3.2 3D Physics Die Rigidbody Implementierung für den dreidimensionalen Raum stellt eine abgespeckte Version, im Vergleich zur 2D, dar. Hier kann die Gravitation nur ein- oder ausgeschaltet werden und keine Physics Materialien zugeteilt werden. Bei den Collidern finden sich Implementierungen der Standard Formen, wie Würfel oder Kreis ,wieder, hinzugekommen sind jedoch der Mesh Collider, der eine ähnliche Funktionsweise, wie der Polygon-Collider, darstellt und der Wheel Collider, der detaillierte Entscheidungsmöglichkeiten bezüglich Reifendruck, Federung, Reibung, in verschiedene Richtungen, und vielen mehr, bietet. Auch Effectors werden hier wieder aufgelistet, jedoch ist sticht die Cloth Komponente hier heraus. Sie ermöglich es Kleidung zu simulieren, um zum Beispiel im Wind zu flattern.

4.2.2.4 Navigation

Navigation ist nur für 3d Projekte verwendbar und ermöglicht es Entwicklern für jeder Objekt einzustellen, ob es begehbar ist, um anschließend Gegnerobjekten, oder anderen Objekten, die Wegfindung benötigen, einen sogenannten Nav Mesh Agent zu geben. Dieser Agent enthält Werte für Höhe, die maximale Schräge, die er bezwingen kann, und die maximale Stufenhöhe. Der Entwickler muss nur noch ein Ziel übergeben und das Objekt kann sich selbstständig dorthin bewegen.

4.2.2.5 Audio

Hier können Entwickler Objekte mit Fähigkeiten für das Zuhören oder das Sprechen ausstatten. Bestimmte Audiofilter für Echoeffekte können auch erstellt werden.

4.2.2.6 Video

Als einziges Element in dieser Gruppe, ist der Video Player das Werkzeug zum Präsentieren von vorher gerenderten Zwischenszenen.

4.2.2.7 Rendering

In jedem Spiel wird eine virtuelle Kamera verwendet, durch deren Linse gespielt wird. Diese Kamera Komponente ist unter dem Begriff Rendering, mit einer Skybox-, einer Licht- und der Sprite Renderer Komponente zu finden.

4.2.2.7.1 Camera Hier können wichtige Einstellung bezüglich der Darstellung getroffen werden. Von oben beginnend, kann der Entwickler gleich den Hintergrund festlegen. Standardmäßig ist er blau. Neben Farben können Entwickler jedoch auch Skyboxen auswählen. Mit der Option der Culling Mask, können bestimmte Schichten aktiviert oder deaktiviert werden. Die Projektion beschreibt die Art, wie mit entfernten Objekten umgegangen werden soll. Während Perspective Objekte, die in der Ferne liegen kleiner darstellt und so für 3D Spiel ideal ist, behält Orthograpfic die Originalgröße der Objekte unabhängig von deren Entfernung zu Kamera bei und eignet sich somit für 2D Spiele. Abhängig davon kann der Entwickler das Field of View oder die Kameragröße

einstellen. Um zu verändern wie nah oder fern Objekte von der Kamera entfernt sein dürfen um gesehen zu werden, dient die Clipping Planes Option. Die restlichen Optionen sind für verschiedene Rendering Arten oder ob das gerenderte Bild gespeichert werden soll, um eine Karte zu erzeugen.

4.2.2.7.2 Skybox Die Skybox Komponente wird verwendet, um einen Himmel zu simulieren. Dazu benötigt man ein Skybox Material, welches dann hier verwendet werden kann.

4.2.2.7.3 Light Die Licht Komponente wird verwendet um Sonnenlicht, Licht von Lampen, Lagerfeuer und mehr zu simulieren. Dieses breite Aufgabengebiet wird mittels Untertypen realisiert.

PointLight Punktlichter erleuchten einen bestimmten Bereich von einem Punkt aus in alle Richtungen. Lagerfeuer oder Lampen sind hier die Einsatzzwecke.

Arealight Licht wird von einem bestimmten Punkt in einer Kegelform, in eine Richtung, versendet. Diese Methode ist für Scheinwerfer oder Spezielle Straßenlaternen sehr nützlich.

Directionlight Um die Sonne zu simulieren, werden direktionale Lichter verwendet. Diese werfen Licht von überall in eine bestimmte Richtung, weshalb die Position keine Rolle spielt.

4.2.2.7.4 Sprite Renderer Diese Komponente ermöglicht es Sprites in der Welt anzuzeigen. Das ausgewählte Sprite kann in der Farbe im Editor geändert werden, weshalb Sprites oft in Grautönen eingelesen werden, um sie in der Engine anzupassen, durch die Änderung des Order Layer vor oder hinter anderen Sprites angezeigt werden oder durch Materialen bestimmte Effekte, wie Wasserreflexionen, erhalten.

4.2.2.8 UI

Alle Komponenten, die mit der Erstellung eines Userinterfaces zusammen hängen, sind hier untergebracht. Es sind Presets für Buttons, Dropdown Menüs, Eingabefelder und vieles mehr vorhanden. UI-Elemente werden wie normale Objekte im dreidimensionalen Raum gesehen, weshalb Elemente auch nicht mit Code, wie html oder css, erzeugt und verändert werden können.

4.2.2.9 AR

Je nach Plattform ändert sich die Größe dieses Menüs. SDKs für Geräte ähnlich der HoloLens bieten hier Komponenten für Spatial Recognition und mehr.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class UnityShowCase : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9
10      }
11
12      // Update is called once per frame
13      void Update () {
14
15      }
16  }
```

Figure 4.2: Unity Basisscript

4.2.3 Scripts

Diese Komponenten bieten ein solider Grundkonstrukt. Um Funktionalität in ein Spiel zu bringen werden jedoch, eigens geschriebene, Scripts benötigt, die wie folgt aufgebaut sind. Wie bereits erwähnt, erben diese Programmteile von der Basisklasse MonoBehaviour. Lässt man sich die Scripts in der Engine generieren, so erhält man ein File ähnlich dem in Abbildung 4.2.

4.2.3.1 Lifecycle Methoden

Für Einsteiger sind die zwei generierten Methoden, Start und Update, kommentiert. Unity bietet verschiedene Lifecycle Methoden, die unterschiedliche Zwecke und Anwendungsbereiche haben, an.

4.2.3.1.1 Awake Die erste Methode, die für jedes Objekt aufgerufen wird ist Awake. Hier soll Code ausgeführt werden, der nur einmal ausgeführt werden darf.

4.2.3.1.2 OnEnable Objekte in Unity können jederzeit aktiviert oder deaktiviert werden. Werden Objekte aktiviert wird diese Methode aufgerufen. Diese Methode kann genutzt werden um andere Objekte auch zu aktivieren.

4.2.3.1.3 Start Start wird immer nach OnEnable aufgerufen und wird primär für die Verknüpfung zwischen Komponenten benutzt.

4.2.3.1.4 FixedUpdate Unity unterscheidet in zwei verschiedene Updatemethoden. FixedUpdate wird, wie der Name vermuten lässt, immer in selben Zeitintervallen

aufgerufen, während Update immer dann verwendet wird, sobald ein neues Bild erschienen ist. FixedUpdate wird für Physikberechnungen und Eingabeüberprüfungen verwendet, die zwischengespeichert werden um in der Update Methode verwendet zu werden.

4.2.3.1.5 Update Hier wird die Logik des Spiels behandelt. Sollen Objekte bewegt werden, soll dies mit den vorher eingelesenen Werten hier geschehen. Haben bestimmte Aktivitäten eine Abklingzeit, bevor sie wieder verwendet werden können, so werden die Überprüfungen hier durchgeführt.

4.2.3.1.6 LateUpdate Wird für Kamerabewegungen genutzt und wird nach allen anderen Updates aufgerufen. Anschließend wird das aktuelle Bild erzeugt und der Kreislauf beginnt bei FixedUpdate erneut.

4.2.3.1.7 OnDisable Wird ein Objekt während dieses Bildes deaktiviert, so wird diese Methode verwendet um zu reagieren.

4.2.3.1.8 OnDestroy Wird dieses Objekt zerstört, zum Bespiel sobald der Spieler verloren hat, so dient diese Methode um den Score zu speichern oder ein bestimmtes UI aufzurufen.

4.2.3.2 Kollisionen

Kollisionen passieren, wenn zwei Collider desselben Typs, sprich 2D mit 2D und 3D mit 3D, miteinander kollidieren. Abhängig von deren Eigenschaften werden verschiedene Events aufgerufen.

4.2.3.2.1 OnCollisionEnter Kollidiert ein Collider mit einem anderen und beide haben die Standardeinstellungen ausgewählt, so wird diese Methode während dem Bild der Berührung aufgerufen.

4.2.3.2.2 OnCollisionEnter2D Ist das 2D Abbild der oben beschriebenen Methode.

4.2.3.2.3 OnCollisionStay/OnCollisionStay2D Solange zwei Collider miteinander kollidieren, wird diese Methode benutzt.

4.2.3.2.4 OnCollisionExit/OnCollisionExt2D Trennen sich die zwei Objekte nun wieder, kommt diese Methode zum Einsatz.

```
GameObject newObject = Instantiate(Resources.Load("myObject")) as GameObject;
```

Figure 4.3: Objekt erzeugen über Resources

```
GameObject newObject = Instantiate(gameObject);
```

Figure 4.4: Objekt erzeugen durch Klonen eines Objektes

4.2.3.3 Trigger

Jeder Collider hat die Option ein Trigger zu sein oder nicht. Trigger registrieren zwar auf Kollisionen im Code, prallen jedoch nicht voneinander ab. Kollidiert ein Objekt, welches einen Collider mit einem aktiven Trigger hat, mit einem anderen Objekt, unabhängig von dessen Optionen, so werden folgende Events stattdessen ausgeführt.

OnTriggerEnter/OnTriggerEnter2D

OnTriggerStay/OnTriggerStay2D

OnTriggerExit/OnTriggerExit2D

4.2.3.4 Objekte erzeugen

Im Editor können Objekte mit Drag and Drop in die Szene hineingebracht werden. Will man jedoch zur Laufzeit Objekte erzeugen, so funktioniert dieser Ansatz nicht. Die Methode Instantiate erledigt dies, es gibt jedoch verschiedene Möglichkeiten diese zu benutzen. Durch die letzten Parameter kann eingestellt werden, wo und als Kind welchen Elements das Objekt erzeugt wird. Im ersten Parameter muss man das Objekt selbst angeben, wobei der Entwickler hier 3 Optionen hat.

4.2.3.4.1 Resources Bestimmte Ordnernamen sind mit speziellen Funktionen belegt. Der StreamingAssets Ordner zum Bespiel ist für nicht zu kompilierende Inhalte bereitgestellt. Um Objekte einfach zu erzeugen, dient der Resources Ordner. Gespeicherte Objekte können aus diesem über ihren Name erzeugt werden mit der Zeile aus Abbildung 4.3.

4.2.3.4.2 Klonen Um Objekte schnell klonen zu können, dient die Zeile aus 4.4.

4.2.3.4.3 Referenzen Die selbe Syntax, jedoch einen anderen Verwendungszweck hat die Erstellung von Objekten mittels Referenzen. Ein Objekt kann als öffentliche Variable gespeichert werden, um später eine Instanz davon zu erstellen.

```
foreach (GameObject g in GameObject.FindGameObjectsWithTag("myTag"))
{

}

foreach (object o in GameObject.FindObjectsOfType<MyType>())
{

}
```

Figure 4.5: Beschaffung aller Objekte mit einer bestimmten Eigenschaft

```
[SerializeField]
private float myField;
```

Figure 4.6: Privates Feld mit SerializeField Annotation

4.2.3.5 Öffentliche Variable und Verlinkung zwischen Komponenten

Unity bietet mehrere Möglichkeiten für die Erstellung von Referenzen zwischen Komponenten, wobei deren Einsatz sich deutlich unterscheidet. Um auf Komponenten eines Objektes zugreifen zu können, benötigt man zuerst eine Referenz auf dieses Objekt selbst. Die Methode GetComponent() gibt anschließend den gewünschten Komponenten zurück. Referenzen auf Objekte kann man durch Filtern aller Objekte bekommen, zusehen in Abbildung 4.5. Objekte werden meist mit Tags versehen, weshalb die erste Methode öfter verwendet wird. Will man jedoch nur ein bestimmtes Objekt, so ist es nicht ratsam alle Objekte auf deren Namen zu überprüfen. Hier bietet Unity ein extrem brauchbares Features, welches bei öffentlichen Variablen Standard ist, jedoch mit der Annotation SerializeField auch für private Felder verwendet werden kann, siehe Abbildungen 4.6 und 4.8. Die Werte dieser Felder können über den Inspector gesetzt werden, wie in 4.7 und 4.9 zu sehen.

Erstellt man öffentliche Felder mit den Komponenten oder einem Objekt als Typ, wie in Abbildung 4.8, so kann man diese im Inspector per Drag and Drop belegen, zu sehen in 4.9.

Diese Methode wird verwendet, wenn Referenzen auf andere Objekte nötig sind. Den eigenen Rigidbody, sollte man jedoch nicht so referenzieren. Hier sollte eine private Variable angelegt werden, die mit GetComponent() in der Start Methode initialisiert wird, siehe Abbildung 4.10.

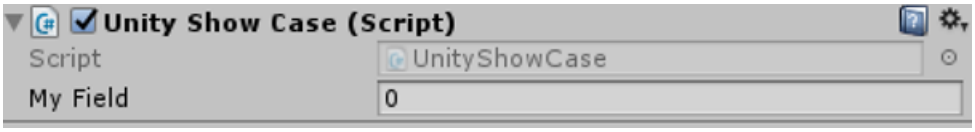


Figure 4.7: Inspector Ansicht von Feldern

```
public Rigidbody MyRigidbody;

public GameObject MyGameObject;
```

Figure 4.8: Öffentliche Felder mit komplexen Datentypen

Figure 4.9: Inspector Ansicht von mehreren komplexen Feldern

4.2.4 Alternativen

Unity ist nicht die einzige Spieleengine und andere Optionen wie die Unreal Engine 4 oder Gamemaker Studio, wären valide Optionen gewesen. Unity bietet jedoch soliden Support für sowohl 2D als auch 3D, wodurch wir in keinster Weise eingeschränkt waren. Den größten Vorteil der Unreal Engine, die unglaublich mächtige Rendering Pipeline, würde von unseren Spielen nie ausgereizt worden, weshalb wir uns für die Unity Game Engine entschieden, nicht zuletzt wegen unserer schon vorhandenen Erfahrung und der riesigen Community.

```
private Rigidbody rb;
// Use this for initialization
void Start ()
{
    rb = GetComponent<Rigidbody>();
}
```

Figure 4.10: Rigidbody des eigenen Objektes erhalten

4.3 Firebase

Firebase ist eine einfach zu bedienende Serveralternative, die Datenbank, Fileserver, Authentifizierung, Website Hosting und mehr über ein Webinterface vereint. Im Vergleich zu regulären Servern macht es Firebase einfacher für den Entwickler, sodass dieser mehr Zeit in sein Produkt selbst investieren kann. Firebase bietet fortgeschrittene Sicherheitsmechanismen, Synchronisierung mehrerer Serverstandorte, Erreichbarkeit immer und überall und Offline Funktionalität. Alle Produkte sind mit verschiedensten Anmeldeoptionen verknüpfbar, wie Google Sign-In, Facebook Sign-In, Email/Passwort, Google Play, usw..

4.3.1 Datenbank

Firebase bietet hier zwei Möglichkeiten zur Speicherung von Daten, wobei beide auf NoSql-Datenbanken zurückgreifen. Daten können über das Webinterface angesehen, geändert und auch hinzugefügt werden.

4.3.1.1 Cloud Firestore

Diese neue Datenbanklösung ist zwar momentan noch in der Beta, ist jedoch von Beginn an für Skalierbarkeit optimiert. Wie für NoSql-Datenbanken üblich, wird enormer Fokus auf Lesegeschwindigkeiten und weniger auf Schreibgeschwindigkeiten gelegt. Verstärkt wird dieser Effekt dadurch, dass Indexe auf alle Felder gelegt werden um die gleichen Datenzugriffszeiten für 100 als auch für 100 Millionen Daten zu garantieren. Cloud Firestore bietet momentan SDKss für Android und Ios Apps, Web, Node, Java, Python und Go an. Der Service verfügt unter Android, Ios und Web über die Möglichkeit, Daten lokal zu zwischenspeichern, wenn die Verbindung einmal verloren geht und diese dann mit dem Server zu synchronisieren, sobald das Problem behoben worden ist. Cloud Firestore ist eine dokumentenbasierte Lösung, was bedeutet das Objekte als Dokumente in einer Collection gespeichert werden. Diese Art der Speicherung wird oft als semistrukturierte Datenspeicherung bezeichnet. Dokumente haben verschiedene Felder, wobei Felder wieder Collections mit Dokumenten enthalten können. Diese sogenannten Subcollections ermöglichen in der Theorie zwar ein 1:1 Abbild der Datenstruktur aus Objektorientierten Sprachen, da diese jedoch nur einzeln gequeried werden können und nicht inkludiert werden können, verursacht diese Methode nur mehr Komplikationen. Eine Realisierung mittels Arrays von Referenzen, wäre hier die einfachere Variante, da Arrays inkludiert

werden und, sofern das Programm das einlesen dieser händelt, einfach geparst werden können. Dieser Aufbau ähnelt sehr einer Relationalen Datenbankstruktur.

4.3.1.2 Realtime Database

Realtime Database bietet viele der selben Vorteile, wie Cloud Firestore. Nur der Fokus auf Skalierbarkeit ist nicht so stark ausgeprägt, jedoch ist das System stabiler, da es schon länger auf dem Markt verfügbar ist. SDKs gibt es für Android, Ios, Web, C++ und Unity. Neben der besseren Verfügbarkeit der SDK für Unity war ein Mitentscheidungsgrund die Stabilität, die die Realtime Database bietet. Die Synchronisierung Offline geänderter Daten ist für Android und Ios verfügbar, nicht jedoch für Webanwendungen. Wir benutzen jedoch keine der drei Plattformen, und selbst wenn können unsere Nutzer keine Daten ändern oder einfügen, weshalb dieses Feature keine Bedeutung für uns hatte. Die Skalierbarkeit ist momentan noch kein Bedenken und der Umstieg auf Cloud Firestore ist auch während der Entwicklung noch leicht genug. Die Realtime Database benutzt die Key-Value Notation für die Datenspeicherung. Diese Schreibweise ähnelt JSON sehr, wodurch das einlesen und auslesen von Daten deutlich erleichtert wird.

Wir entschieden uns für die Realtime Database aus mehreren Gründen. Die Verfügbarkeit der SDK für Unity ist ein KO-Kriterium, da diese Technologie nicht änderbar ist und die Verwendung der Rest-Schnittstelle von Cloud Firestore außer Frage stand. Zusätzliche Features wie bessere Skalierbarkeit, bessere Offline Funktionalitäten oder strukturiertere Daten bringen wenig bis keinen Vorteil, da unsere Datenbank nur die Spiele speichert, die zur Verfügung stehen, diese keine komplexen Attribute haben, welche Referenzen benötigen würden und wir keine Offline Funktionalität brauchen. Falls die Datenbank jedoch über eine gewisse Größe hinweg wachsen würden, wäre ein Umstieg auf Cloud Firebase denkbar, da die Preispolitik diesen Usecase unterstützt.

4.3.2 Storage

Firebase bietet ein simples Filesystem, mit dem man Dateien in Ordnerstrukturen speichern kann. Von jeder Datei wird die Größe, der Typ, das Erstell- und Änderungsdatum und die Download URL angezeigt. Durch die SDKs ist die Navigation durch die Ordnerstruktur und das downloaden von Dateien unkompliziert. Als Administrator erhält man eine Übersicht über alle Downloads, die momentane Speichergröße und die momentane Anzahl an Elementen, für einen bestimmten Zeitraum.

4.3.3 Hosting

Firebase ermöglicht es auch eine Webseite zu hosten. Eine Übersicht über alle Uploads mit deren User ist vorhanden um auch zwischen Versionen zu wechseln. Man kann seine Seite auch mit einer eigenen Domain verbinden.

4.3.4 Authentifizierung

Wie bereits erwähnt bietet Firebase hier viele Möglichkeiten für Entwickler Ihre User anzumelden. Anmeldeoptionen von etablierten Netzwerken wie Facebook, Twitter, GitHub, Google Play oder Google sind unterstützt, ebenso wie Telefonauthentifierung, Gastaccounts oder Authentifizierung über Email und Passwort. Es stehen auch Templates für Email-Adressen Bestätigung, Passwort zurücksetzen, usw. zur Verfügung. Im Webinterface ist eine Auflistung aller registrierten Accounts, mit deren Anmeldeoption, dem Datum der Erstellung und dem Datum des letzten Logins, ersichtlich. Werden Telefonbestätigungen verwendet, sieht man diese als Graph dargestellt. Die Einbindung der Authentifizierung in andere Firebaseprodukte ist über sogenannte Regeln implementiert. Hier kann der Admin einstellen ob, oder wer Schreibzugriff auf die Datenbank hat. Kann jeder schreiben, der eingeloggt ist, so gibt es die globale Variable „auth“, deren Wert man mit null vergleicht. Diese Regeln können mit einem Simulator getestet werden um Fehler zu vermeiden.

4.3.5 Preispolitik

Firebase bietet drei verschiedene Modelle, wobei eines kostenlos ist und die restlichen beiden kostenpflichtig sind. Der „Spark Plan“ ist die gratis Version. Man hat Zugriff auf Datenbank, Fileserver, Website Hosting, Authentifizierung und den Großteil der restlichen Produkte. Jedoch sind bestimmte Nutzungsgrenzen für die verschiedenen Tätigkeiten festgelegt. Die Datenmenge der Realtime Database, darf zum Beispiel nicht über ein Gigabyte groß sein und es dürfen auch nicht über zehn Gigabyte heruntergeladen werden. Für Ludimus sind das jedoch astronomische Grenzen, die nur mit extrem vielen Spielen erreicht werden können. Unsere Anwendung verbraucht vielmehr Fileserverkapazitäten, die bei diesem Modell bei fünf Gigabyte liegen und es dürfen maximal ein Gigabyte pro Tag heruntergeladen werden. Mit genügend Nutzern sind das die ersten Schwellpunkte die überschritten werden. Für 25 Dollar pro Monat kann man Subscriber des „Flame Plans“ sein. Dieser hebt die Nutzungsgrenzen aller Produkte um einiges. Da wir diese Erhöhung jedoch nur für ein Produkt brauchen, und wir hier nicht allein sind, ist die dritte Möglichkeit, der „Blaze Plan“, „Pay as you go“, heißt man zahlt nur, was man benötigt. Zur Schätzung der Kosten bietet Firebase hier einen Simulator und selbst wenn Ludimus viele Nutzer, mit vielen Spielen hätte, würde dieser Plan nur 20 Dollar kosten. Zusätzlich dazu verfügt man dann jedoch auch über alle Funktionen, die Firebase bietet. Bigdata Analysen und mehrere Datenbanken- und Storage-Buckets für Parallelität sind nur einige davon.

4.4 Sonstige Technologien

4.4.1 Figma

Figma ist ein Online Mockup Tool zum schnellen Erstellen von Userinterfaces. Entwürfe werden über die Cloud synchronisiert und können von mehreren Benutzern kollaborativ verwendet werden. Durch die Verfügbarkeit im Browser und das ermöglichen von Gruppenarbeiten, entschieden wir uns für Figma anstatt für Illustrator, welches wir noch zu Beginn benutzten.

4.4.2 Blender

Blender ist ein Open Source 3D-Modellierungstool. Es ist gratis nutzbar und bietet neben der Modellerstellung noch viele weitere wichtige Features, wie Animationen, Materialerstellung oder die Kolorierung von Objekten. Durch die in Punkt 4.1.9.1 angesprochene Synergie, zwischen Blender und Unity, ist der Importprozess reibungslos.

4.4.3 Vectr

Vectr ist ein einfaches Tool zum Erstellen von Vector Grafiken. Die Funktionalität ist zwar auf ein paar Formen und Textfelder beschränkt, jedoch war dies ausreichend für all unsere Icons, weshalb wir von Photoshop zu Vectr wechselten.

4.4.4 Illustrator

Zu Beginn des Projektes benutzten wir Illustrator noch für die Mockup Erstellung. Wie in Punkt 4.4.1 erwähnt, änderte sich dies jedoch und Illustrator wurde nur für die Erstellung des Billiardtisches, aufgrund der Mustererstellung für das Tuch, die Holzränder und die Metallecken, genutzt.

Chapter 5

Ausgewählte Aspekte der Systemerstellung

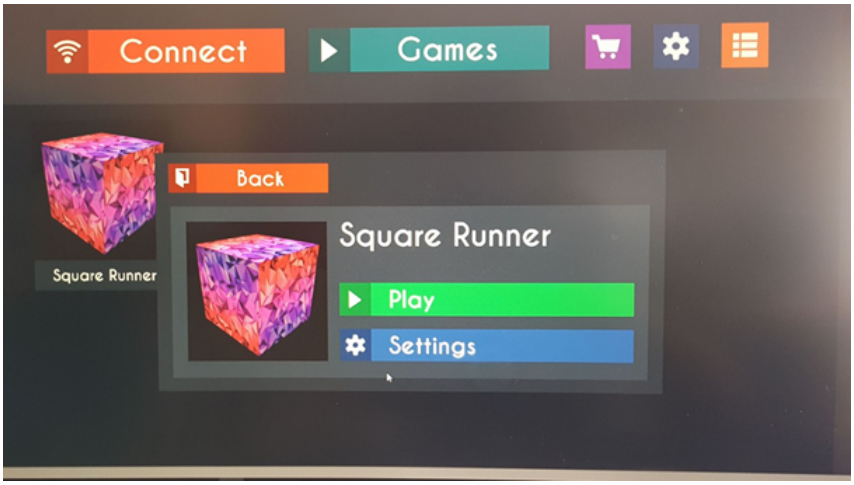


Figure 5.1: Erster Shop Prototyp

5.1 Designentscheidungen

Ludimus war von Beginn an als ein Produkt für Konsumenten gedacht, weshalb wir viele Überlegungen und anschließende Änderungen bezüglich der Benutzeroberfläche anstellten. Viele der Abwandlungen sind mit dem Hintergedanken der Benutzbarkeit der Plattform verbunden und konzentrierten sich auf Farben, sowie Form und Anordnung. Im Folgenden Abschnitt werden generelle Designentscheidungen beschrieben, die sich auf alle Aspekte es Projekts ausgewirkt haben. Änderungen die nur einen Teilbereich umfassten, werden in Punkt 5.1.2 genauer erwähnt.

5.1.1 Generelle Designentscheidungen

Für die ersten Iterationen des Projekts wählten wir dunkle Farben um auch das Spielen in den späteren Stunden zu ermöglichen. Die Grafik 5.1 zeigt das Server Interface, wobei hier das Shop-Interface zu sehen ist, sowie die Detailansicht eines Spieles.

Um einerseits mehr Platz für den eigentlichen Inhalt zu haben und andererseits moderner auszusehen, änderten wir die Proportionen, die Abstände und Teils die Farben, um an die Kacheln von Windows 8, zu erinnern, wie in der Abbildung 5.2 zu sehen. Hier ist die Lobbyansicht ausgewählt.

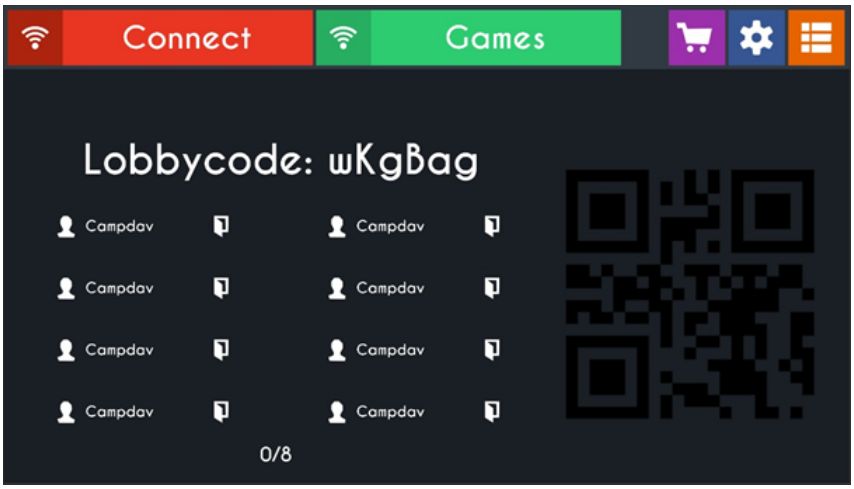


Figure 5.2: Lobby Prototyp

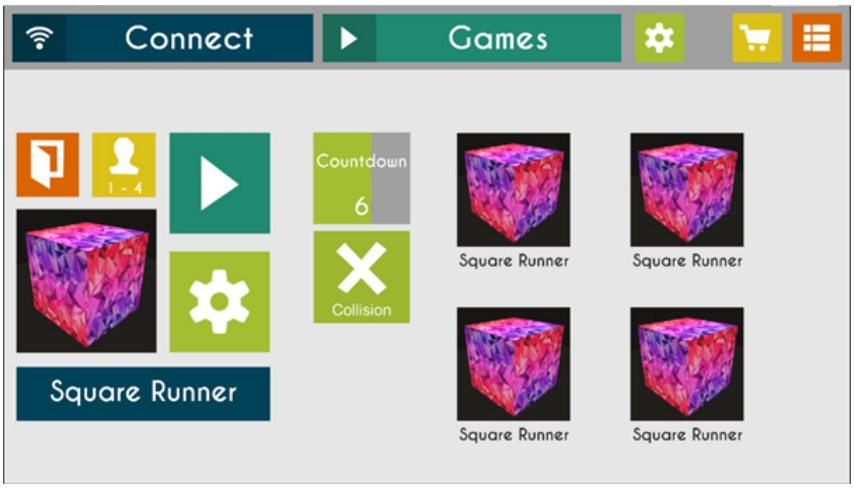


Figure 5.3: Shopdetailansicht mit Einstellungen und hellem Farbschema

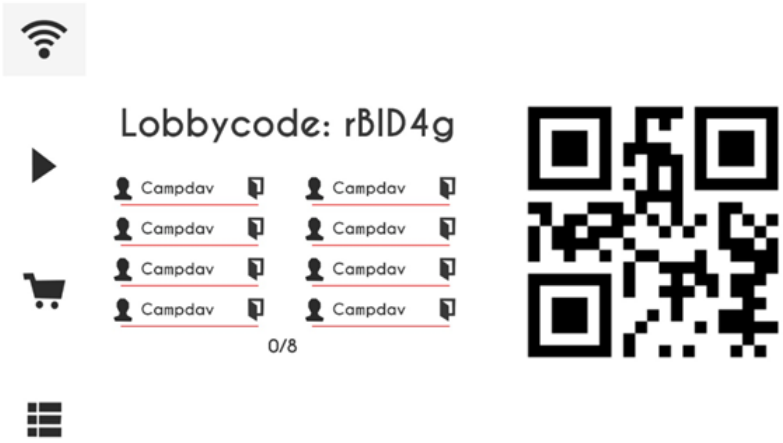


Figure 5.4: Schwarz-Weiß Designentwurf

Ludimus war Primär als Produkt für Familien mit Kindern gedacht. Dunkle Farben übermittelten nicht das Gefühl, das wir transportieren wollten, weshalb wir uns für helle Farben auf weißen Hintergrund entschieden. 5.3 zeigt auch eine Umgestaltung der Detailansicht, dazu jedoch in Punkt 5.1.2.1 mehr.

Einen gewissen Rückschritt in Sachen Farben stellte unser nächstes UI dar, zu sehen in 5.4. Der Hintergedanke hierbei war die Vereinfachung der Plattform und ein schwarzweißes Design half uns dabei. Nutzer sahen nun auch erstmals, in welchem Menü sie sich befanden, da dieses nun einen dunkleren Hintergrund bekam. Globale Einstellungen für die Plattform selbst als eigener Punkt, wurden mit diesem Design entfernt.

In den nächsten Iterationen brachten wir einerseits Farbe zurück, rundeten andererseits aber auch alle Buttons ab und gaben ihnen einen Schatten, wie in 5.5 zu sehen. Hatte man in allen bisherigen Versionen noch eine Menüleiste und einen Inhaltsteil, so entfernten wir Erstere um nur den Inhalt darstellen zu können. Drückt der Nutzer nun auf einen der vier Menüpunkte, so schwenkt die Kamera in dessen Ecke und der Inhalt wird dort angezeigt. Mit einem Zurück-Knopf kann der Benutzer anschließend wieder zum Hauptmenü gelangen. Wir implementierten zusätzlich noch eine Farbauswahl für das gesamte Interface. Benutzer konnten sich bestimmte Farbkombinationen auswählen, um sowohl die vier Primärfarben, als auch den Hintergrund zu ändern. Diese Entscheidungen hatten jedoch alle mehrere Nachteile, weshalb wir alles wieder verändern mussten. Die Kameraanimationen waren zwar interessant anzusehen, jedoch verlangsamten sie den Navigationsprozess Unnötigerweise. Zusätzliche Probleme machten verschiedene Formate,

Figure 5.5: Buntes Hauptmenü mit Schatten und abgerundeten Ecken

da die Animationen nur auf 16:9 optimiert waren. Obwohl das Farbschema wechseln gut funktioniert hat und Menschen mit verschiedenen Präferenzen die Möglichkeit gab das UI zu verändern, mussten wir das Feature entfernen, da uns die Erkennung unserer Marke wichtig war und wir erreichen wollten, dass sobald Familien unser Logo oder unsere Farben sehen an uns denken können. Wenn jeder seine eigenen Farben festlegen kann, ist dies nicht möglich.
Bisher waren eine Übersicht der Lobby, der Shop, alle Heruntergeladenen Spiele und ein Menü für Optionen in der Serveransicht und der Client konnte nur Name und Lobbycode eingeben und sich anschließend verbinden. Wollte eine Gruppe also spielen, mussten sich zuerst alle Spieler auf deren Handys zum großen Bildschirm verbinden und danach musste ein Spieler das Tablet, den Laptop oder den Fernseher bedienen um ein Spiel zu starten, oder zuerst herunterladen. Speziell beim Spielen im Wohnzimmer über den Fernseher gibt es hier enorme Probleme, die mit der Steuerung von Menüs mit der Fernbedienung zu tun haben. Um diese unnötige Interaktion mit dem großen Bildschirm zu entfernen, gestalteten wir die Aufteilung der Menüs komplett um. Der Server sollte nur noch die Lobby anzeigen, da diese keine Interaktion forderte. Sowohl Spielernamen, als auch der Lobbycode in schriftlicher und grafischer Form, konnten so größer dargestellt werden. Zusätzlich zu einem Login Fenster, welches wir bei App-Start dem Client hinzugefügt haben, wurden Spieler wie immer aufgefordert ihren Namen und den Lobbycode einzugeben um sich anschließend zu verbinden. Dieser Vorgang blieb für alle Spieler bis auf den ersten gleich. Dieser hat nun jedoch die Kontrolle über die Lobby und kann sowohl neue Spiele herunterladen, als auch bereits heruntergeladene Spiele starten. Um das Interface zu vereinfachen, vereinten wir den Shop mit den heruntergeladenen Spielen. Durch den Verzicht einer einheitlichen Trennung kann der Nutzer nicht unterscheiden ob ein Spiel zuerst heruntergeladen werden muss und probiert so viel mehr neue

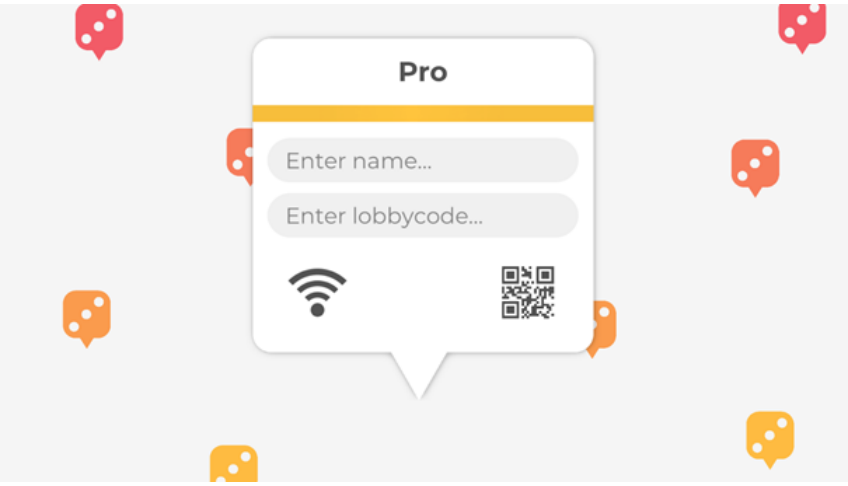


Figure 5.6: UI zum Verbinden zu einer Lobby

Spiele. Die Downloadzeit überschreitet nur selten 10 Sekunden, wodurch dies nur noch verstärkt wird.

Wie in 5.6 zu sehen, ist unser Userinterface mit dem Ludimus Logo eng verknüpft, um es so in die Köpfe der Nutzer zu bekommen. Das UI zum Verbinden zu einer Lobby ist in eine Form, die dem Ludimus Logo ähnelt, eingebettet. Zusätzlich steigen im Hintergrund Ludimus Ballone auf, die ihre Farbe in Abhängigkeit des Subskription Levels, des eingeloggten Nutzers, ändern.

5.1.2 Spezielle Designentscheidungen

5.1.2.1 Shop

Der Shop änderte sich über die verschiedenen Versionen am meisten, da die richtige Präsentation der Spiele essentiell für den Kauf eines teureren Subskription Plans ist. Die Änderungen gliedern sich in die Anzeige aller Spiele und die Anzeige eines Spieles. Zu Beginn wurden alle Spiele in einem Gitter mit Zeilen und Spalten angezeigt, wie in 5.1 zu sehen. Überlegungen bezüglich Sortierung oder Suche wurden zu diesem Zeitpunkt noch nicht angestellt. Um Nutzern zu ermöglichen nur Spiele einer bestimmten Art zu sehen, erweiterten wir deren Datenmodell um das Feld „Genre“. Zusätzlich vergaben wir einen Farbcode für jedes Genre, um dem Nutzer schnell ersichtlich zu machen um welche Art von Spiel es sich handelt. Da wir die Plattform für externe Entwickler öffnen wollten, um mehr Spiele anbieten zu können, fügten wir die Reiter „New“, „Top“ und „New Update“ hinzu. Der Gedanke hierbei war neue Spiele eine kurze Zeit unter „New“ anzuzeigen und Spiele, die oft gespielt werden sollten anschließend unter „Top“

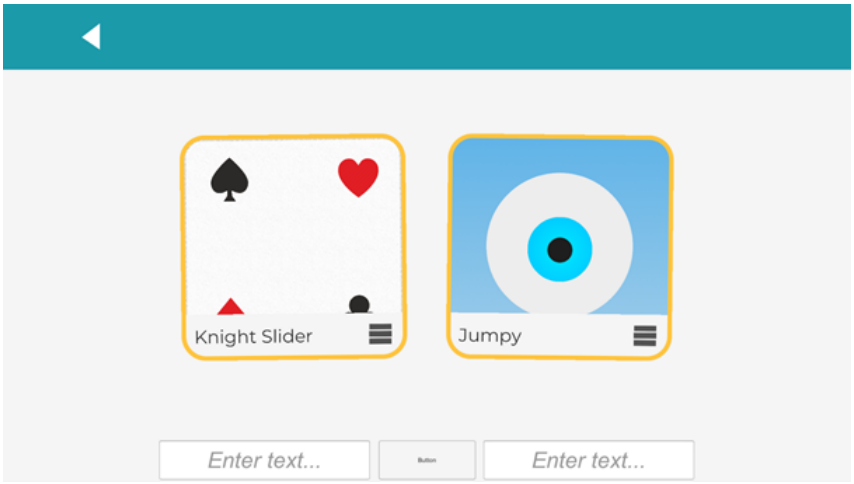


Figure 5.7: Finale Shoppräsentation mit Debug Modus

zu finden sein. Um auch ältere Spiele wieder beliebt zu machen oder Entwicklern eine zweite Chance geben zu können, falls der ursprüngliche Release nicht funktioniert hat, würden Spiele die neue Updates erhalten unter „New Update“ gezeigt werden. Den Plan externe Entwickler mit einzubeziehen verworfen wir jedoch, wodurch eine solche Gliederung nicht mehr nötig war, da zu wenig Spiele dafür verfügbar wären. Wie in Abbildung 5.7 zu sehen, vereinfachten wir zusätzlich die Anzeige der Spiele, in dem wir das Icon des Spieles größer machten, die Detailansicht hinter einem Menü versteckten und wir das Genre des Spieles durch die Farbe der Umrandung klar darstellten. Die zwei Eingabefelder, am unteren Ende des Bildschirms, mit dem Knopf in der Mitte gehören zum Debug Modus, der nur für Entwickler ist, dazu mehr in 5.3.

Die in Abbildung 5.3 zu sehenden spielspezifischen Einstellungen wurden entfernt und durch Optionenmenüs im Spiel selbst ersetzt. Durch Änderungen im Startprozess von Spielen wurden Spiele mit mehreren Szenen ermöglicht, wodurch die alte Methode überflüssig wurde.

5.2 Spielstart

Sobald der erste verbundene Spieler ein Spiel ausgesucht hat, indem er darauf geklickt hat, und die Gruppengröße für das Spiel in Ordnung ist, wird die ID des Spieles an den Server geschickt. Dieser sucht sich sowohl die Downloadlinks für die Client-Dateien, als auch für seine eigenen Server-Dateien. Anschließend überprüft er, ob das Spiel schon alle nötigen Dateien für die aktuelle Version heruntergeladen hat. Falls nicht wird das nachgeholt und anschließend werden die Download URLs für die Clients an alle verbundenen Spieler geschickt. Diese überprüfen auch deren Existenz und laden fehlende herunter. Verfügt ein Client über alle Dateien, so schickt er ein „ok“ an den Server. Sobald der Server Bestätigungen aller Clients erhalten hat, startet er seine Szene. Und schickt ein „ok“ an alle Clients, worauf hin diese auch deren Szenen starten. Dieser Ablauf ähnelt dem zwei Phasen Commit bei verteilten Datenbanken, wodurch die Sicherheit dieses Systems bewiesen ist.

5.2.1 Probleme

Spiele bestehen drei Komponenten, die gemeinsam dafür sorgen, dass die Spiele funktionieren. Sie bestehen aus Szenen, Modellen und Scripts. Szenen und Modelle, auch Texturen, Materialien, usw. sind in sogenannten „Assetbundles“ gespeichert. Alle Szenen des Clients sind in so einem „Assetbundle“ zusammengespeichert, ähnlich einer Zip-File. Alle Szenen des Server sind ebenfalls so persistiert. Die Unterscheidung zwischen Client und Server ist zwar nicht zwingend nötig, jedoch ist es Ressourcenverschwendung, wenn der Server auch Szenen der Clients speichert, da er diese ohnehin nicht startet, und umgekehrt. Die selbe Speicherung gilt für alle Modelle. Der Grund für die Trennung zwischen Client und Server bleibt gleich, jedoch wäre es deutlich einfacher, wenn Clientszenen und Clientmodelle zusammen gespeichert werden und alle Serverszenen und Servermodelle ebenfalls. Leider unterstützen „Assetbundles“ in Unity diese Funktionalität nicht. Als dritter Baustein gelten die Scripts, in denen die Logik der Spiele, das eigentliche Programm, beschrieben ist. Scripts können nicht zu „Assetbundles“ gemacht werden, jedoch gibt es unter Android und Windows die Möglichkeit, sie zu bündeln und dann zur Laufzeit eines Spieles zu verwenden. Unter Ios ist eine derartige Methode nicht möglich, da kein externer Code zur Laufzeit von Apps hinzugefügt werden kann oder darf. Diesen Sicherheitsmechanismus wird Apple nicht entfernen, was bedeutet, dass jeder Code immer in der App vorhanden sein muss. Für den Workflow des Erstellens eines neuen Spieles würde das bedeuten, dass wir alle verwendeten Scripts in das Hauptprojekt transferieren müssen und dann eine neue App Version in den App Store hochladen müssen. Apple User sind ein großer Anteil an unserer Zielgruppe, weswegen ein Ausschluss der Plattform nicht möglich ist. Zwei Speichervarianten für zwei Plattformen, wäre undenkbarer Mehraufwand, weshalb wir uns entschlossen, dass wir für jedes neue Spiel ein Update mit dessen Scripts für alle Plattformen herausbringen.

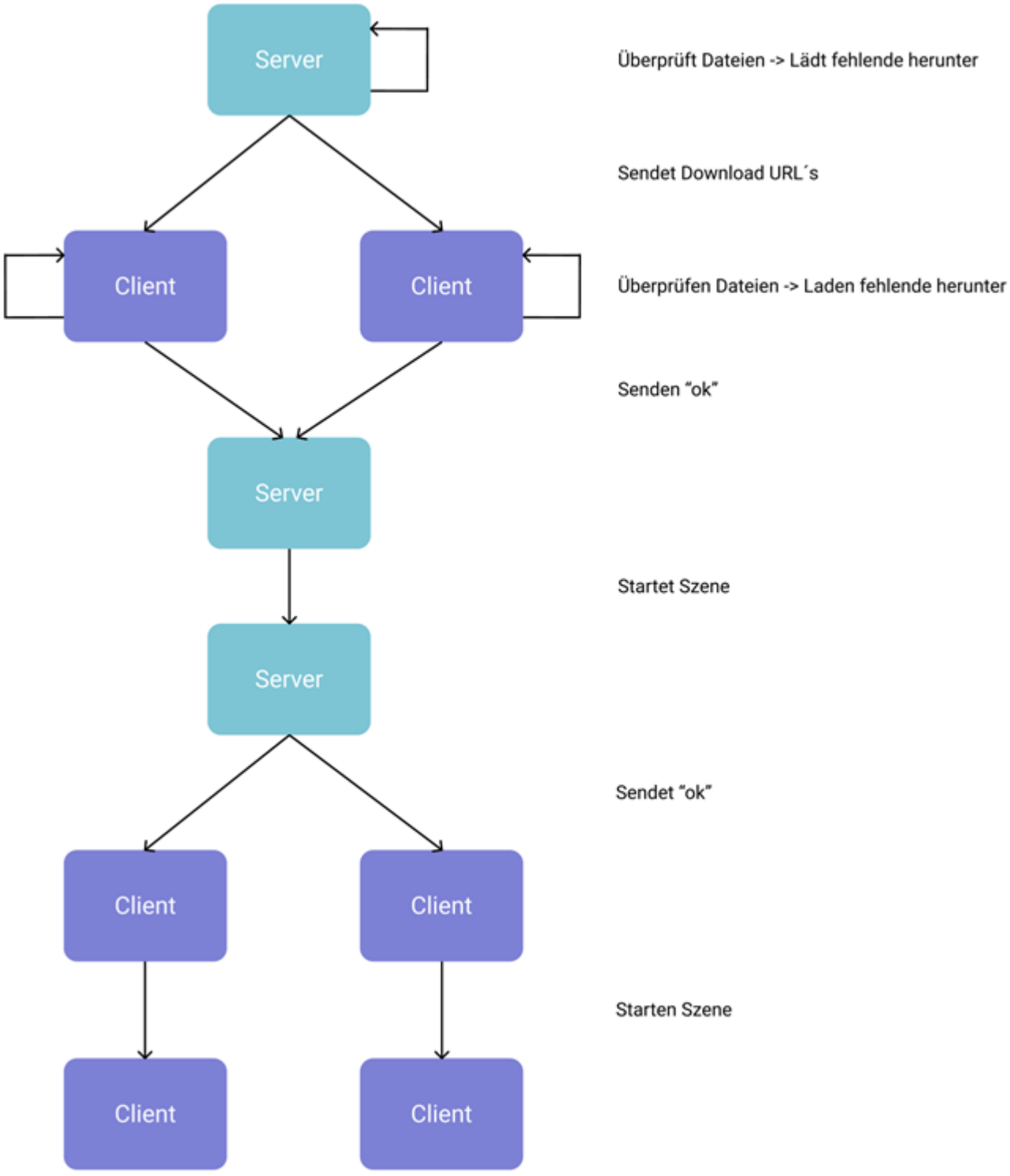


Figure 5.8: Vorgang bei Spielstart

5.2.2 Optimierungsansätze

Der Fileserver von Firebase hat mit unserem Gratisbezahlplan eine maximale Downloadgröße und eine Grenze für Downloads pro Tag unabhängig von der Größe der Dateien, die heruntergeladen werden. Das ist der Grund, warum nur so viele Zugriffe gemacht werden sollten, wie nötig. Eine Maßnahme war es, dass der Server alle Download URLs beschafft und an die Clients weitergibt. Dieses Prinzip wurde auch für den eigentlichen Download selbst getestet. Im konkreten bedeutet das, dass der Server alle Plattformen der Clients überprüft und dann die Clientdateien für eben diese Plattformen herunterlädt, zwischenspeichert und anschließend den Clients über das lokale Netzwerk sendet. Dieser Ansatz entlastet den Fileserver, da dieser nur einen Downloadrequest verarbeiten muss, der Rest passiert nur zwischen den Spielern. Die gesamte Downloadzeit steigt jedoch dadurch, dass Dateien zwei Mal pro Spieler heruntergeladen werden, anstatt einmal und Clients können während dieser Zeit nur eingeschränkt kommunizieren, weshalb wir uns für die oben beschriebene Variante entschieden.

5.3 Debug Modus

Das in 5.2 beschriebene System funktioniert einwandfrei mit fertigen Spielen. Da jedoch Spiele sowohl Datenbankeinträge, als auch hochgeladene Dateien am Fileserver benötigen, ist diese Lösung komplett ungeeignet für Spiele in Entwicklung. Der Debug Modus löst dieses Problem und ermöglicht es, auch kleine Änderungen, mit erheblich geringerem Zeitaufwand, zu testen. Vor dem Debug Modus, wurden die Scripts der fertigen Spiele in das Hauptprojekt hinein kopiert. Die Entwicklung passierte in einem externen Projekt. Diese externen Projekte hatten ein Gerüst, das sich um die Spielerhandhabung und um die Kommunikation zwischen Client und Server im generellen kümmerte. Ein Testdurchlauf der aus „Assetbundles“ für Szenen und Modelle, jeweils wieder unterschieden in Client und Server, bauen, auf Fileserver hochladen, Scripts in Hauptprojekt kopieren und neuste Version für Smartphone und PC bauen dauerte mehrere Minuten. Mit dem Debug Modus fällt das externe Projekt mit Kommunikationsgerüst, das Bauen der „Assetbundles“ und das hochladen auf den Fileserver weg. Für neue Spiele wird nur ein Ordner im Hauptprojekt und eine Client und Server Startszene angelegt. Die Kommunikation zwischen Server und Client ist mit Delegates abrufbar, wodurch man aktuelle Spieler, deren Inputs und vieles mehr gleich verwenden kann. Bestimmte Aspekte, wie Physics, UI oder Gegner-AI sind Client unabhängig und können somit nun, auch ohne diese, in der Engine, mit einem Knopfdruck, getestet werden. Werden sowohl Client, als auch Server benötigt, die Clients für Smartphone und PC bauen. Der letzte Schritt ist nur bei fast fertigen Versionen nötig, da es sonst eher ratsam ist, nur einen Client zu bauen und den Server in Engine zu debuggen. Als erster verbundener Spieler kann man in der Spielanzeige das neue Spiel zwar nicht sehen, über die zwei Eingabefelder, kann man jedoch sein Spiel nun starten. Im linken Feld wird der Name der Startszene des Client angegeben und im rechten die des Servers. Drückt der Entwickler nun auf den Knopf in der Mitte wird sein Spiel gestartet. Die Zeiteinsparen und die Erleichterung der Entwicklung sind enorm, weshalb alle fertigen Spiele mit dieser Methode entwickelt worden sind.

Bibliography

[Knu84] Donald E. Knuth. *The TeXbook*. Addison-Wesley Professional, Reading, Mass, 1 edition, January 1984.

[Lam85] Leslie Lamport. *Latex Document Preparation System Users*. Addison Wesley Publishing Co, Reading, Mass, spi edition, October 1985.

[Rec06] Peter Rechenberg. *Technisches Schreiben: (nicht nur) für Informatiker*. Carl Hanser Verlag GmbH & Co. KG, München, 3 edition, August 2006.

List of Figures

List of Tables

Project Log Book

Date	Participants	Todos	Due

Appendix A

Additional Information

If needed the appendix is the place where additional information concerning your thesis goes. Examples could be:

- Source Code
- Test Protocols
- Project Proposal
- Project Plan
- Individual Goals
- ...

Again this has to be aligned with the supervisor.

Appendix B

Individual Goals

This is just another example to show what content could go into the appendix.