

Major Project

Modulnummer: SAE6302

Model Name: *Major Project*

Abgabedatum: *25.02.2021*

Abschluss: *Bachelor of Science (Hons.) Games Programming*

Semester: *September 2020*

Name: *Nils Walther*

Campus: *Hamburg*

Land: Deutschland

Wortanzahl: *10286*

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Hamburg, 25.02.2021

Ort, Datum

Unterschrift Student

Praxisprojekt zur Realisierung einer Charakter-Modifikation, durch die Programmierung einer unterstützenden App für das Pen & Paper Regelwerk „Hell Over Mind“ mithilfe von Unity.

I. Inhaltsverzeichnis

I. Inhaltsverzeichnis	2
II. Abbildungsverzeichnis	4
III. Tabellenverzeichnis	6
1. Einleitung	7
1.1 Motivation	7
1.2 Ziele	8
1.3 Marktrelevanz	8
2. Kontext	9
2.1. Rollenspiele und Pen & Paper(Grundkonzept)	9
2.2. Referenz Apps für P&P(Medium)	10
2.2.1 Universal Character Sheet	10
2.2.2 Fifth Edition Character Sheet	11
2.2.3 Character Sheet for any RPG	13
2.3. Entwicklung von Apps	14
2.3.1 Android in Unity	14
2.3.2 UI und UX für Mobile Endgeräte	15
2.3.3 Speicherungen auf Datenbanken	17
3. Methodik	19
3.1. Prototypen Erstellung	19
3.1.1. Projekt aufsetzen	19

3.1.2. Vergleich mit Referenzprojekten	19
3.2. Iterative Überarbeitung	19
3.2.1. Scrum Prinzip	19
3.2.2. Usability-Test	20
3.4. Expertengespräche	20
3.4. Hermeneutik	20
3.5. Meilensteine	21
4. Durchführung	22
4.1. Aufbau und Ideen:	22
4.2. Prototyp:	24
4.2.1. Erste Benutzerschnittstelle:	24
4.2.2. Apache Web Server:	26
4.2.3. C# Skripte:	28
4.3. Visuelle Überarbeitung nach Android UI Design:	31
4.3.1. Externe Arbeiten:	32
4.3.2. Prefabs:	34
4.3.3. Änderung am Erscheinungsbild:	38
4.3.3.1. Login Scene:	38
4.3.3.2. Menu Scene:	39
4.3.3.3. Fragebogen Scene:	39
4.3.3.4. Charakterbogen Scene:	41
4.4. Der Cloud Server:	44
4.4.1. Neue Utility Skripte:	44
4.4.2. Hetzner Server:	46
4.4.3. Weitere PHP Skripte:	49
4.5. Benutzerfreundlichkeit:	52
4.5.1. Tests:	52
4.5.2. Anwendung des Feedbacks:	53
5. Ergebnisse	60
5.1. Präsentation:	60
5.2. Ziele:	64
5.3. Kritische Reflexion:	65
6. Rekapitulation:	66
IV. Literaturquellen:	67
V. Bildquellen :	69
VI. Tabellen Quellen :	73
VII. Anhang :	73

II. Abbildungsverzeichnis

Abbildung 1: Hell Over Mind Buch	8
Abbildung 2: D&D Chainmail Buch	10
Abbildung 3: Universal Character Sheet	12
Abbildung 4: Fifth Edition Character Sheet	13
Abbildung 5: Character Sheet for any RPG	14
Abbildung 6: Unity Engine	15
Abbildung 7: UI und UX Unterschiede	16
Abbildung 8: Lineares Layout	17
Abbildung 9: SQL	19
Abbildung 10: Grundkonzept	24
Abbildung 11: LoginScene	25
Abbildung 12: Fragebogen	26
Abbildung 13: XAMPP	27
Abbildung 14: CREATE DATABASE	27
Abbildung 15: CREATE TABLE	28
Abbildung 16: Connect	28
Abbildung 17: PHP Variablen	28
Abbildung 18: PHP query	29
Abbildung 19: Netzwerk Koroutine	30
Abbildung 20: Dropdown Klasse	30
Abbildung 21: Erstellung der Dropdowns	31
Abbildung 22: Text Lesen	31
Abbildung 23: JSON Klasse	32
Abbildung 24: JSON Klasse	32
Abbildung 25: Gimp Button	33
Abbildung 26: Standard Button	34
Abbildung 27: Input Field	34
Abbildung 28: Oswald Regular	34
Abbildung 29: Farbpalette	34
Abbildung 30: HOM Icon	35
Abbildung 31: Text Prefab	35
Abbildung 32: BildButton	35

Abbildung 33: NormalButton	36
Abbildung 34: InputFieldPlaceholder	36
Abbildung 35: InputFieldTop	36
Abbildung 36: Dropdown	37
Abbildung 37: GrundwertBox	37
Abbildung 38: PopUp	37
Abbildung 39: Swipe	38
Abbildung 40: ButtonSwitch	38
Abbildung 41: LoginScene	39
Abbildung 42: MenuScene	40
Abbildung 43: Geburt	41
Abbildung 44: Kultur	41
Abbildung 45: Jugend	41
Abbildung 46: Berufung	42
Abbildung 47: Erscheinung	43
Abbildung 48: Psyche	43
Abbildung 49: Interaktion	44
Abbildung 50: Fertigkeiten	44
Abbildung 51: Ausstattung	44
Abbildung 52: Save Data	46
Abbildung 53: Data Keys	47
Abbildung 54: Hetzner Server	48
Abbildung 55: Lupari Befehle	48
Abbildung 56: charsheet	49
Abbildung 57: WinSCP anmelden	49
Abbildung 58: Internet Pfad	50
Abbildung 59: php INSERT	51
Abbildung 60: php SELECT	51
Abbildung 61: php UPDATE	52
Abbildung 62: php DELETE	52
Abbildung 63: neue Farbpalette	55
Abbildung 64: Tarotkarte	55
Abbildung 65: App Hintergrund	55
Abbildung 66: Button Hintergründe	56
Abbildung 67: Charakterbögen	56
Abbildung 68: Volks Infos	57
Abbildung 69: PupUp Nachfrage	57

Abbildung 70: Werte speichern	57
Abbildung 71: Obyektz Anzeige	58
Abbildung 72: Liste füllen	58
Abbildung 73: Obyektz Optionen	59
Abbildung 74: Aktionen und Reaktionen berechnen	59
Abbildung 75: Zeige Würfel	60
Abbildung 76: Würfel	60
Abbildung 77: LoginScene komplett	61
Abbildung 78: MainMenu Charaktererstellung	62
Abbildung 79: MainMenu Charakterbögen	62
Abbildung 80: Charaktererstellung Nr.1	63
Abbildung 81: Charaktererstellung Nr.2	63
Abbildung 82: Charakterbogen Person	64
Abbildung 83: Charakterbogen Grundwerte	65
Abbildung 84: Charakterbogen Fähigkeiten und Ausstattung	65

III. Tabellenverzeichnis

Tabelle 1: Offline Tests	53
Tabelle 2: Online Tests	54
Tabelle 3: Erreichte Ziele	66

1. Einleitung

In der folgenden Arbeit, wird die Entwicklung einer App für das *Pen & Paper* Regelwerk „Hell Over Mind“ gezeigt. Dabei wird besonders auf die Entwicklung mit Unity und das Speichern von Daten auf eine Datenbank eingegangen.

1.1 Motivation

Schon vor dem Studium, entwickelte ich einen der Welt von „Hell Over Mind“. Es entstand Ideen von ein Brettspiel und Sammelkartenspiel. Jedoch fehlte mir das Wissen diese zu entwickeln und deswegen fing ich mit meinem Studium am SAE-Institute Hamburg an. Mein erstes Projekt dort, war ein Textbasiertes Abenteuer. Zum Ende des Studiums entwickelte ich endlich die Welt mit einer kleinen Gruppe und verfasste ein *Pen & Paper* Regelwerk (siehe Abb. 1). Dabei wurde mir bewusst, wie alt das Genre ist und trotz hoher Beliebtheit, selten gespielt wird. Was an den neuen Möglichkeiten liegen könnte, Spiele zu spielen. Mein Ziel war es nun die größtmögliche Zielgruppe mit einer App zu erreichen. Um das zu erreichen, war es mein Ziel das Studium mit dieser App abzuschließen, weil ich durch die wissenschaftliche Arbeit, an dem Thema, viel lernen werde und in der Zukunft anwenden kann.

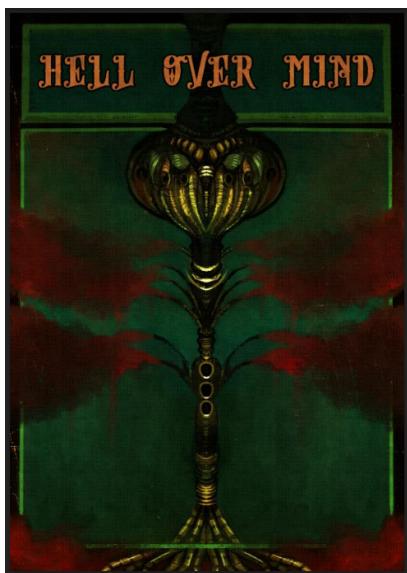


Abb. 1: Hell Over Mind Buch

1.2 Ziele

Das Projektziel ist die digitale Version eines Charakterbogens für „Hell Over Mind“. Dabei sollen sowohl neue, wie auch erfahrene Spieler diese App verstehen und nutzen können. Um dies zu erreichen wird die App mithilfe der *Game-Engine Unity* entwickelt und von Testern überprüft. Deswegen wird das Projekt in zwei Arten von Zielen unterteilt.

Soll-Ziele:

- Erstellung eines „Hell Over Mind“ Charakters
- Modifizierung des Charakters
- Anzeigen und Interaktion von Werten
- Speichern von Daten auf der Datenbank
- Laden der Daten von der Datenbank
- Usability-Tests

Kann-Ziele:

- Erstellen und verändern von Fertigkeiten
- Erstellen und verändern von Items
- Funktionierendes *Hekatem* System
- Eigene Klassen und Rassen
- Verändern der Daten von der Datenbank
- Löschen von Daten auf der Datenbank
- Praxis-Test

Die Soll-Ziele werden benötigt, um eine funktionierende und nutzerfreundliche App zu haben, während die Kann-Ziele nette Features sind, die nicht unbedingt notwendig sind.

1.3 Marktrelevantz

Das *Pen & Paper* Genre hat im Vergleich zu seinem Alter wenig gute Ableger im App Bereich. Das liegt daran, dass es nur Stift und Papier benötigt. Stattdessen haben sich andere Arten von Rollenspielen im digitalen Bereich entwickelt. Um zwischen den anderen Arten von Rollenspielen und vielen durchschnittlichen Apps hervorzustechen, wird eine einzigartige App benötigt. Diese muss durch Aussehen, Inhalt oder Benutzerfreundlichkeit überzeugen. Außerdem bietet sie einen optimalen Einstieg in das neuartige Regelwerk „Hell Over Mind“.

2. Kontext

Zuerst wird erklärt, worauf das Projekt aufbaut, um zu verstehen, wozu die App gebraucht wird. Danach werden Referenzprodukte betrachtet, um ein Bild zu bekommen, wohin die Reise ungefähr gehen soll. Abschließend werden die Techniken und Werkzeuge gezeigt, mit denen gearbeitet wird.

2.1. Rollenspiele und Pen & Paper(Grundkonzept)

Pen & Paper begann, Mitte des 19. Jahrhundert an Popularität zu gewinnen.

Es treffen sich Gruppen von Spielern, meistens unter speziellen Regeln und eigenem Namen. Diese Spiele bestehen aus einem Spielleiter, der alle anderen Spieler durch eine fiktive Welt führt. Der Spielleiter bestimmt die Regeln, die die Mitspieler beeinflussen.

Diese Gruppen spielen nach eigenen Regeln, da es kein einheitliches Regelwerk gibt. Dies ändert sich durch Gary Gygax und Dave Arneson. Sie entwickeln die Bücher „Chainmail“ (vgl. Abb 2), ein wichtiger Grundstein für *Pen & Paper* und auch „Dungeons & Dragons“ (*D&D*), das bisher bekannteste Spiel dieses Genres.

Regelwerke bestehen aus kleinen Geschichten, aber auch aus Regeln zu fantastischen, mittelalterlichen Gegenständen und Magie. Dadurch haben die Spieler Möglichkeiten, sich Welten um und mit diesen Objekten/ Regeln zu erschaffen. Normalerweise schlüpfen die Spieler dabei in Charaktere, die zu der Geschichte passen (vgl. Riggs 2017).

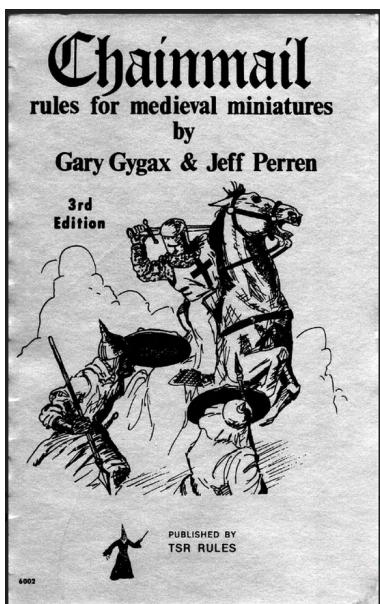


Abb. 2: D&D Chainmail Buch

Beispiel: Ihr könnt neben dem Volk eurem Charakter mit unterschiedlichen Hintergrundgeschichten oder bereits vorhandene Ausbildungen, Persönlichkeit einhauchen. Wie war die Kindheit eures Charakters? Wie ist er oder sie aufgewachsen? In „Hell Over Mind“ gibt es etliche verschiedene Wege eurem Charakter eine persönliche Note zu verleihen. Denkt daran, dass ihr diesen Charakter in eurem Abenteuer verkörpern müsst (Walther et al. 2020: S.18).

Da dieses Prinzip so simpel ist, haben sich viele Spiele daraus gebildet. Rollenspiele sowohl für Brett als auch Videospiele. Außerdem gibt es unzählige neue Regelwerke, „Das Schwarze Auge“ ist dabei ein deutsches Werk als Antwort auf den zuvor genannten Titel *D&D* (vgl. Neukirchner 2014). „Call of Cthulhu“ ist ein Regelwerk, das sich auf die Werke von H.P. Lovecraft bezieht oder „Cyberpunk Red“, das die Spieler in eine Sci-Fi Welt entführt. Die Möglichkeiten sind unendlich und nur durch die Fantasie der Spieler begrenzt (vgl. Elsam 2020).

2.2. Referenz Apps für P&P(Medium)

Was 1976 noch Stift und Papier war, ist heute mobil. Die Welt ist im Wandel, fast jeder besitzt ein Smartphone und einen Internetzugang. Die meisten Regelwerke sind mittlerweile Online erhältlich oder darüber ersichtlich. Außerdem gibt es einige Apps, die das Spiel erleichtern sollen. Drei, für das Projekt, ausgewählte Apps zeigen, was auf dem Markt momentan erhältlich ist. Im Folgenden werden die Apps kurz erklärt und ihre Funktionen erläutert. Eines haben die Beispiele gemeinsam, sie sind alle in Englisch geschrieben.

2.2.1 Universal Character Sheet

Diese App gestattet dem Spieler, unabhängig vom Regelwerk, einen Charakter zu erstellen. Wenn die App geöffnet wird, erscheint ein dunkles leeres Feld. Rechts oben gibt es einen kleinen Knopf aus drei Punkten, um das Menü zu öffnen. Dort können die Charakterbögen sortiert, auf die Grundeinstellungen zugegriffen werden und es gibt einen Knopf, in dem die App beschrieben wird. Um einen Charakterbogen zu erstellen, muss auf den einzigen anderen *Button* auf der Seite gedrückt werden. Ein kleines schwarze Plus in einem roten Kreis. Hier gibt es die Möglichkeit, die Grundwerte des Charakters zu erstellen. Dazu gehören Name, Klasse, Rasse und ein Bild, das von der App vorgegeben wird. In diesem erstellten Charakterbogen, hat der Nutzer die Möglichkeit auf vier Reiter zu drücken: Werte, Inventar,

Fähigkeiten und Notizen. Bei den Werten können, wieder durch den Plus Knopf, neue Werte erstellt werden. Dieser Wert ist nun ein kleiner Balken mit Namen, Zahl, Plus und Minus Knöpfe, welche die Zahl ändern. Im Inventar gibt es wiederum drei Unterteilungen. Zuerst die Möglichkeit, den Charakter mit ausgewählte Ausrüstungen zu versehen. Die Ausrüstungen besitzen Name, Typ, Rüstung und Effekte. Als Zweites gibt es eine Anzeige für Geld, Erfahrungspunkte und das Level des Charakters. Im letzten Abschnitt können Items über ihre Namen gespeichert werden. Bei den Fertigkeiten wird durch den roten Knopf eine Anzeige von Bildern aufgerufen, um sie der neuen Fertigkeit zuzuordnen. Danach kommen Schaden, Kosten und spezielle Notizen. Zuletzt kommt der Notiz *Button*. Dort wird durch den roten Knopf ein Fenster aufgerufen, um eine Notiz mit Titel und Text zu erstellen. Über den Zurück-Knopf des Handys oder einen kleinen Pfeil *Button* oben links, gelangt der Nutzer ins Menü zurück. Dort wird der neue Charakter angezeigt (vgl. Awkwardly Developed Apps 2019).

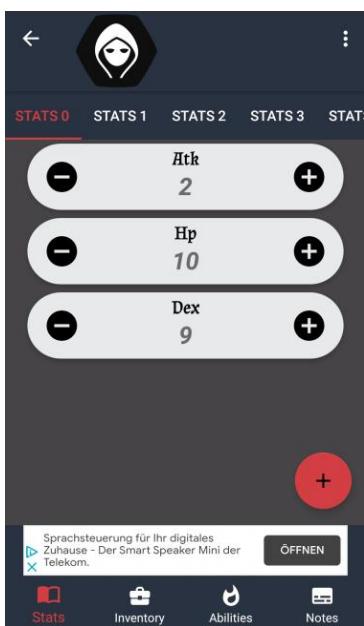


Abb. 3: Universal Character Sheet

2.2.2 Fifth Edition Character Sheet

Hier hat der Spieler die Möglichkeit einen Charakterbogen für die Fünfte Edition von *D&D* zu erstellen. Beim Öffnen der App wird dem Nutzer sofort ein fertiger Charakterbogen präsentiert. Es gibt jedoch eine Leiste oberhalb des Bogens mit drei Knöpfen. Der rechte führt zu einem Menü, das vier Grundfunktionen vermittelt. Dem Nutzer wird die Möglichkeit geboten, sich für Geld erweiterte Funktionen zu kaufen, eine Fähigkeit aus dem Regelwerk zu

aktivieren und die Möglichkeit einen Charakterbogen über eine Datei zu speichern oder zu laden. Zuletzt wird Hilfe angeboten, einiges erklärt und andere Apps beworben.

Der Knopf in der Mitte der oberen Leiste lässt den Nutzer schon bestehende Charakterbögen löschen, wobei der linke Knopf neue erstellt.

Beim Erstellen des Charakters wird gefragt, ob der Nutzer einen *Creator* benutzen möchte oder nicht. Bei Verneinung ist der Charakter sofort erstellt. Falls der *Creator* benutzt wird, wird man zu den Grundwerten des Charakters (Rasse, Sub-Rasse, Klasse, Hintergrundgeschichte) geleitet. Danach werden die Grundwerte dem Regelwerk entsprechend verteilt und zwei besondere Talente der Klasse sind auswählbar. Es folgen weitere Klassenspezifische Fragen, sowie die Verteilung von Lebenspunkten und *Items*. Zuletzt wird gefragt, ob der Nutzer fertig ist und der erstellte Charakter wird angezeigt.

Der Charakterbogen ist *D&D* spezifisch aufgebaut und benötigt Grundwissen des Regelwerks, um verstanden zu werden. Es gibt insgesamt fünf Seiten, welche sich durch Wischen nach links oder rechts auf dem Bildschirm erreichen lassen. Auf der ersten Seite stehen die Grundwerte des Charakters sowie die Klasse und der Name. Mit einem Rechtswisch öffnet sich zuerst die Seite mit den Talenten, danach die *Items* mit den Werten und eine Seite weiter die Fähigkeiten. Auf der letzten Seite finden sich viele Informationen über den Charakter, seine Klasse, sowie seine *Items*, Fähigkeiten und Talente. Alle Informationen in der App sind in farbige Kästen (vgl. Walter Kammerer 2014).



Abb. 4: Fifth Edition
Character Sheet

2.2.3 Character Sheet for any RPG

Auf der Startseite hat der Nutzer die Möglichkeit, einen neuen Charakterbogen zu erstellen oder einen bestehenden wieder zu benutzen. Ein neuer Charakter wird mittels eines kleinen *Button* mit einem Plus, typisch für *Android Apps*, erstellt. Der Nutzer hat die Möglichkeit sich ein eigenes *Template* oder Vorlage zu erstellen, ein schon vorhandenes zu nutzen oder ohne Vorlage weiter zu machen. Die Vorlagen beinhalten die vier Regelwerke „Pathfinder 2e“, „Starfinder“, „DND 5e“ und „Vampire: The Masquerade V5“. Nach der Auswahl der Vorlagen erscheint dem Nutzer ein leerer Charakterbogen. Um das Menü zu öffnen, gibt es in der oberen, linken Ecke einen *Button* oder der Bildschirm wird nach links gewischt. Im Menü gibt es acht Funktionen, die von Veränderung der App, Spenden an den Entwickler, ein *Button* der nach „Reddit“ führt und speichern sowie verlassen der App gehen. Außerdem gibt es die Möglichkeit neue Seiten zu erstellen. Sie sind das Kernelement der App und bieten unzählige Funktionen. Zusammenfassend sind es Objekte mit vielen möglichen Eigenschaften. In Abb. 5 ist nur ein Beispiel der Möglichkeiten dargestellt (vgl. Serge Shustoff 2020).

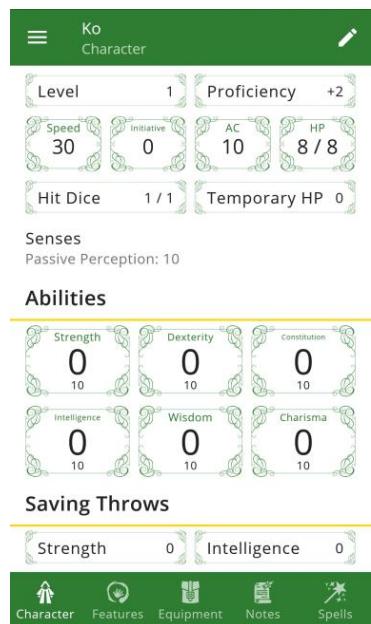


Abb. 5: Character Sheet
for any RPG

2.3. Entwicklung von Apps

2.3.1 Android in Unity

Für das Projekt wird die Entwicklungsumgebung *Unity* genutzt. Seit 2008 steht sowohl für *Android* als auch *iOS* eine große Palette an Apps und Spielen zur Verfügung. Seitdem ist *Mobile Gaming* nicht mehr wegzudenken und jedes Jahr wächst es. Durch diese stetige Entwicklung ist es natürlich auch möglich mit der *Unity Engine*, Apps für den Markt oder auch privaten Gebrauch zu nutzen. Dank der Vielseitigkeit und Masse an *Features*, ist *Unity* sogar zu der beliebtesten *Engine* für *Android* geworden.

Wenn man zur Entwicklung einer App *Unity* nutzt, gibt es drei Punkte, die zu beachten sind.

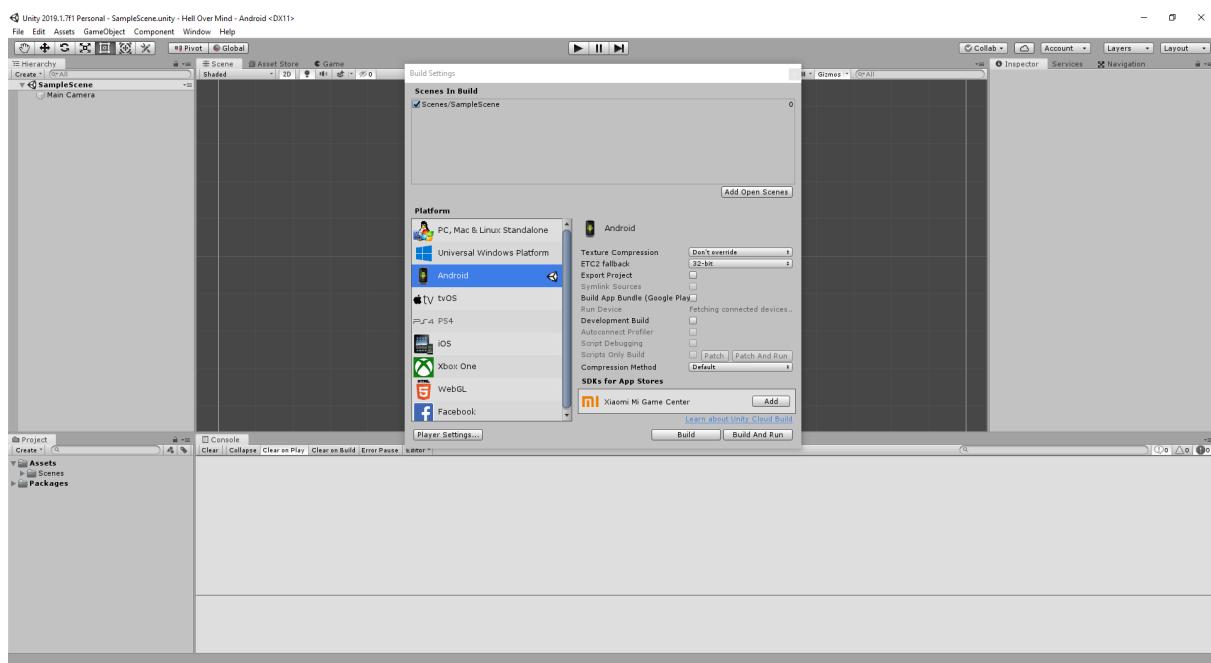


Abb. 6: Unity Engine

Als Erstes muss das Projekt, *Android* kompatibel entwickelt werden, indem ein neues Projekt aufgesetzt und dort in den *Build Settings*, *Android* als Plattform ausgewählt wird. Dies ist wichtig, da *Unity* normalerweise für die Standalone-Entwicklung eingestellt ist. Was *Unity* von der gängigen Entwicklungsumgebungen, *Android Studio* unterscheidet, ist die Benutzeroberfläche. Sie erleichtert die Entwicklung um ein Vielfaches und gibt die Möglichkeit, alles in der *Engine* zu verändern. Außerdem muss dadurch nicht jedes visuelle Objekt der App programmiert werden, sondern kann einfach erstellt werden.

Damit kommen wir zum dritten Punkt, die Sprache. In *Unity* wird hauptsächlich mit C# gearbeitet. Somit muss nicht zwischen Sprachen gewechselt werden, was die Entwicklung der Oberfläche um ein vielfaches vereinfacht (vgl. Talbert 2019; Sinicki 2020).

2.3.2 UI und UX für Mobile Endgeräte

Die *UI*, *User Interface* oder auch Benutzeroberfläche, begegnet uns heutzutage bei fast allen technischen Produkten. Sie dient der Kommunikation zwischen Nutzer und Programm. Dafür benötigt es entweder einen visuellen oder auditiven Part (vgl. Thornsby 2016: S.8).

Die visuellen Funktionen werden mit der Hand bedient. Dazu gehören Knöpfe, Bilder und Texte. Diese können sich durch den Input der Nutzer jederzeit ändern. Anders ist es beim auditiven Teil, der durch Sprachsteuerung gelenkt und durch Töne oder Sprachausgabe wiedergegeben wird. Die Töne werden auch gerne zusammen mit den visuellen Funktionen genutzt, um den Nutzern ein besseres Feedback zu geben.



Abb. 7: UI und UX Unterschiede

Um die Funktionen zu nutzen, wird ein gutes *UI* benötigt, denn selbst die besten Funktionen und Apps haben keinen Nutzen, wenn kein Nutzer sie bedienen kann. Am Ende bleibt es jedem Entwickler überlassen, wie die App entwickelt werden soll, allerdings gibt es Regeln, an die man sich halten sollte. Dazu gehört es, wie im letzten Kapitel gezeigt, Referenzen zu finden und das Produkt diesen entsprechend anzupassen. Ein wichtiger Punkt ist, dass die Apps konsistent sind und der Nutzer sie problemlos bedienen kann. Dies wird durch ähnlichen Aufbau oder einhalten der Regeln erreicht (vgl. Thornsby 2016: S.14–27; 51–76).

Im Folgenden wird auf die wichtigen Aspekte des *UIs* eingegangen. Dabei liegt der Fokus auf dem visuellen *UI*, da dies den Referenzen aus 2.2 entspricht und im Gegensatz zum auditiven für das Projekt notwendig ist.

Wenn der Nutzer die App öffnet, wird ihm das *UI* angezeigt, das er betrachten oder womit er interagieren kann. Die einzelnen Elemente werden als sogenannte *Views* angezeigt und sollten im optimalen Fall einem gewissen Muster folgen, dem *Layout*.

Es gibt zwei gängige *Layouts*. Zum einen das lineare *Layout*. Es sortiert alle Elemente entweder horizontal oder vertikal in der Anzeige und sorgt somit für gleichmäßigen Platz für jedes Element. Ein lineares *Layout* bezieht sich immer auf Gruppen von Elementen und ist von der Größe, der Anzeige und der Objekte in dieser Anzeige, sowie der Anzahl von Objekten in der Anzeige abhängig. Anders verhält es sich da mit dem relativen *Layout*. Bei ihm orientiert sich jedes Objekt selber, im Verhältnis zu seinem *Parent* oder einem anderen beliebigen Element. Dies wird benötigt, um *Performance* zu sparen. Da jedes Element einzeln in sich bewegt wird und sich nicht mehr mit anderen verbinden muss, um zu erfahren, wo es positioniert werden soll. Diese Positionierung ist auch der Punkt, der hier Vorteile bietet. Wo man bei einem linearen *Layout* feste Regeln hat, bietet das Relative viel mehr Möglichkeiten und ist somit flexibler. Die Konsequenz dieser Freiheit ist der Mehraufwand. Bei vielen Elementen sollte auf das lineare *Layout* zurückgegriffen werden, um Zeit zu sparen (vgl. Thornsby 2016: S.28–42).

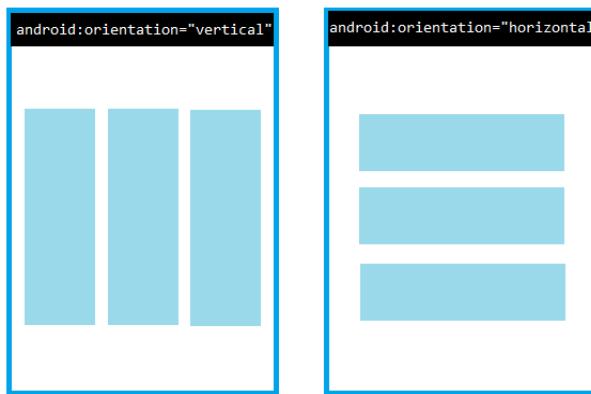


Abb. 8: Lineares Layout

Mit dem fertigen *UI*, gibt es Elemente zum Bedienen. Diese sind auf der Anzeige der App dargestellt und bereit für die Interaktion. Zur Optimierung wird das *UX* oder auch die „User Experience/Nutzer Erfahrung“ benötigt. Diese führt dazu, dass die Nutzer mit der App umgehen können, sie gerne nutzen oder wissen, wie sie zu bedienen ist. Nicht zu vergessen ist, dass jegliche Nutzererfahrungen immer subjektiver Natur sind und somit es unmöglich ist, die perfekte App für jeden zu machen. Trotzdem gibt es Regeln, welche sich im *Android* Markt etabliert haben, um eine gewisse Konsistenz an den Tag zu legen. Dazu gehört eine passende Farbpalette mit miteinander harmonierenden Farben. Diese sollten für alle Elemente und den Hintergrund verwendet werden. Um der Benutzeroberfläche mehr Tiefe zu geben, empfehlen sich Schatten, damit kann man z.B. Objekte hervorheben. Je nach

Wichtigkeit werden die Elemente in unterschiedliche Ebenen gesetzt. Bei Interaktion mit den Elementen gibt es *Feedback*, um den Nutzer interessiert zu halten. Wichtig ist es zu beachten, dass das *UX* nie dem *UI* im Weg stehen darf. Somit darf eine Ladeanimation nicht zu lange gestreckt werden oder ein Farbwechsel muss dem Nutzer positiv auffallen (vgl. Thornsby 2016: S.108–134).

2.3.3 Speicherungen auf Datenbanken

Bei der Erstellung von Charakterbögen, fallen ständig neue Daten an, die der Nutzer speichern möchte. Dazu gibt es zwei Möglichkeiten, wobei im Vorfeld schon beschlossen wurde, nur eine zu nutzen. Der Vollständigkeit halber werden hier beide erwähnt. Die erste Möglichkeit ist die lokale Speicherung in einem Ordner auf dem Handy. Die andere Möglichkeit, auf diese wird im Folgenden eingegangen, ist die Speicherung auf einer Datenbank oder auch Server.

„What is data? Data is just about anything you can imagine and quantify. For example, the records and numbers you keep in a checkbookregister are data. The stats you write down if you keep score at a baseball game are data (Steve Suehring 2002 S.105).“

Daten werden schon seit langem auf Datenbanken gespeichert. Bekannt dafür, sind Unternehmen, welche entweder ein Produkt verkaufen oder einen bestimmten Service anbieten. Der Grund, sie können die Produkte besser anbieten und finden heraus, was welchem Kunden gefällt. Dies ist in der heutigen Welt immer leichter, da jedes elektronische Gerät Daten über den Nutzer sammelt und diese zur Verfügung stellt. Das Gleiche kann mit einem Regelwerk gemacht werden, da mit hilfe der Datenbank herausgefunden werden kann, welche Klassen und Rassen am interessantesten sind. Alle Daten werden sauber in der Datenbank sortiert und sind bereit, ausgewertet zu werden. Dies bietet in der Weiterentwicklung einen enormen Vorteil. Außerdem hat es positive Aspekte für den Nutzer. Sein Handy-Speicher wird nicht von Daten belastet, stattdessen wird alles extern auf einem Server gespeichert und kann beliebig von jedem Gerät aus vom Kunden gezogen werden. Dadurch bietet es auch eine hohe Flexibilität, solange das Gerät mit dem Internet verbunden ist (Steve Suehring 2002 S.106 ff).

Bei der Entwicklung mit *Unity* bieten sich zwei Datenbank-Management-Systeme oder auch *DBMS* an, *MySQL* und *SQLite*. Das *SQL* steht dabei für „Structured Query Language“ und bezieht sich dabei auf die Programmiersprache, welche für *DBMS* genutzt wird. Diese sind nur zwei von einigen Hundert auf dem Markt. *DBMS* wird in zwei Teile unterteilt. Zuerst *NoSQL*, diese bieten nicht das typische System von Datenbanken und sind noch in der Entwicklung. Grundlegend wird hier auf *SQL* verzichtet und es gibt keine Tabellen. Die

zweite Art sind die relationalen *DBMS* oder auch *RDBMS*. Zu diesen gehören auch *MySQL* und *SQLite*. Zur ersten Frage: Warum bieten sich diese beiden *DBMS* für *Unity* am besten an? Ganz einfach, es sind die wohl bekanntesten und auch beliebtesten auf dem Markt. Es finden sich etliche Seiten, *Tutorials* und Videos über diese im Internet. *SQLite* eignet sich perfekt, um kleine Projekte ohne externen Server aufzusetzen, um direkt am Speicherort zu agieren und zu testen. *MySQL* hingegen besitzt eine viel größere Menge an Datentypen und ist eher für große Mengen von Daten ausgelegt. Es benötigt einen externen Server und auch mehr Aufwand. Deshalb sollte auf *MySQL* zurückgegriffen werden, wenn das Projekt groß ist oder in Zukunft in die Größe skaliert werden könnte (vgl. Stabinskas 2019).

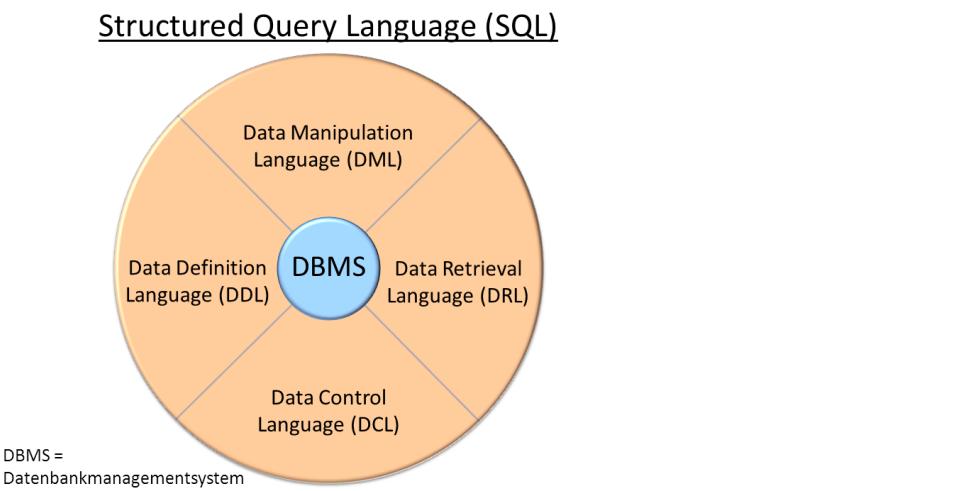


Abb. 9: SQL

Wenn eine *DBMS* ausgewählt ist, gilt zu bedenken, dass der Aufwand, für eine Datenbank immer vom Projekt und den Ressourcen abhängt. Für ein kleineres Projekt sollte der Fokus auf den Datenbanken, dem Kontakt und der Speicherung liegen, während größere Projekte noch Sicherheit und Datenauswertung in den Fokus nehmen können. Das Gleiche gilt bei der Programmierung von Datenbanken. Diese können sehr speziell und kompliziert entwickelt werden, allerdings nur, wenn die Zeit dafür reicht. Ansonsten verschwendet man Zeit und Ressourcen. Wenn eine Datenbank mit *SQL* aufgesetzt werden soll, gibt es vier Schritte, denen man folgen sollte (Steve Suehring 2002 S.108 f.).

1. Die Anforderungen und Ziele der Datenbank werden geprüft. Dabei wird auf die Entität, Eigenschaften und Verbindungen der Daten geachtet.
2. Es wird ein grobes *Layout* erschaffen, um alle Daten zu sammeln. Aus diesem *Layout* werden alle Tabellen der Datenbank normalisiert und schließlich wird die Datenbank mit allen Tabellen mithilfe von *SQL* geschrieben.
3. Die Datenbank wird auf Fehler überprüft und optimiert.
4. Die Datenbank wird in das Projekt integriert, um Daten zu sammeln.

3. Methodik

3.1. Prototypen Erstellung

3.1.1. Projekt aufsetzen

Zu Beginn des Projekts muss eine Idee entwickelt werden. Es müssen Pläne aufgesetzt werden, um sowohl zeitlich, als auch inhaltlich die Vorgaben zu planen und einzuhalten. Um das Projekt erstellen zu können, wird eine *Engine* benötigt, welche nach einiger Recherche funktionsbereit sein sollte. Dabei kann auf Vorlieben oder Erfahrungen zurückgegriffen werden. Zuletzt müssen Organisations- und Speicherungswerkzeuge beschaffen werden, um eine geordnete Übersicht von dem Projekt zu haben (vgl. refa.de 2021).

3.1.2. Vergleich mit Referenzprojekten

Während der Prototyp erstellt wird, soll er mit ausgewählten Referenzprojekten verglichen werden. Dabei wird nach der Art der Relation getestet. Anhand der inneren Selbstwahrnehmung werden die Vergleichsprojekte jeweils mit Gemeinsamkeits- und Unterschiedsvergleichen getestet. Dabei soll herausgefunden werden, was verbessert werden kann, bevor die eigentlichen Tests der eigenen App beginnen. Mithilfe der Vergleichsprodukte werden Skizzen erzeugt, welche alle wichtigen Funktionen beinhalten (vgl. Vergleichsmethode.wordpress.com 2020).

3.2. Iterative Überarbeitung

3.2.1. Scrum Prinzip

Nach der Erstellung des Prototyps wird mithilfe der Befragungen die App immer weiter verbessert. Außerdem werden zu den Befragungen auch immer weitere Funktionen hinzugefügt. Somit wird aus der Zusammenarbeit zwischen den Spielern und dem Entwickler eine App entstehen, welche den Spielern direkt beim Spiel helfen soll. Falls Funktionen fehlerhaft oder unbrauchbar sind, können sie durch dieses Vorgehen schnell beseitigt werden. Die Nachteile dabei ist allerdings die Mehrarbeit, welche in den Prozess einläuft. Es kann passieren, dass die App nicht fertig wird, da sie immer weiter verbessert wird und es somit nie zu einem fertigen Zustand kommt (vgl. Wirsing 2006 S. 8 i. V.m. S. 12).

3.2.2. Usability-Test

Für stetigen Fortschritt wird in gleichmäßigem Abstand das Produkt einer Testgruppe mitsamt einem Testspiel unterzogen. Wobei die Funktionalität überprüft wird, die von Test zu Test verbessert werden soll. Außerdem werden schneller Probleme entdeckt und durch das Verhalten der Nutzer lernt man die Anwendung durch die Augen eines anderen Nutzers kennen.

Der Test wird drei Abschnitte beinhalten. Zuerst wird der Aufgabenbogen vom Testleiter erstellt. Der Testbogen wird den Testern vorgelegt. Der Testleiter beobachtet und interviewt die Tester. Diese Beobachtungen werden durchgehend dokumentiert. Im besten Fall werden die Beobachtungen sofort umgesetzt oder mit den Testern besprochen.

Die Tester haben zwei Aufgaben. Sie geben dem Leiter am besten passendes Feedback zu ihrem Test. Dies machen sie, nachdem und während sie ihre Aufgaben vom Aufgabenbogen machen. Außerdem werden themenspezifische Daten über die Tester festgehalten, um sie zu kategorisieren (vgl. Moran 2019).

3.4. Expertengespräche

Da diese Arbeit ein großes Spektrum abdeckt, werden Experten für spezielle Bereiche benötigt. Diese werden anhand ihrer Schwerpunkte oder persönlichen Beziehungen ausgesucht und zum Projekt herangezogen. Sie sollen mit ihrem Spezialwissen nicht die Arbeit übernehmen, sondern unterstützen, bewerten und im Nachhinein ein Feedback geben, was Verbesserungswürdig ist. Die Experten werden besonders zum Anfang und Ende des Projekts benötigt.

3.4. Hermeneutik

„Die hermeneutische Antwort darauf ist mit der Analyse von Verstehen und Auslegung gegeben: Sofern das Verstehen immer ein Schon-Verstanden-Haben ist, Vertrautheit mit einem sich überliefernden Lebens- und Weltzusammenhang, ist die gegenwärtige Lebens- und Weltoffenheit allein durch die Überlieferung erschlossen. Alles Begegnende tritt in diese Offenheit ein und wird insofern immer nach ihren Möglichkeiten aufgenommen und integriert (Figal 2009).“

Die Recherche ist eine Methode, die auf eine zuvor durchgeführte Methode folgt. Sie dient dazu neues Wissen zu suchen, verstehen und anzuwenden. Dies wird permanent bei der Arbeit an dem Projekt angewendet, um mögliche Probleme schnell zu beheben.

Grundsätzlich geht es um schnelle Lösungen zu kleinen Fehlern, diese werden aus Kapazitätsgründen nicht erfasst. Große Fehler werden dokumentiert und entsprechend durch eine Überschrift kenntlich gemacht.

3.5. Meilensteine

Da es bei dem Projekt um einen Versuch geht, eine App zu erstellen, für ein Spiel, das sehr vom Offline Verhalten seiner Nutzer geprägt ist, hängt sehr viel von dem Feedback der Nutzer ab. Es gibt ein paar grobe Meilensteine, welche die Erstellung der App und Feedback Runden beinhalten. Bei diesen Meilensteinen geht es nicht um bestimmte Ziele in der App, die erreicht werden sollen, stattdessen sollen bestimmte Funktionen testbereit sein oder Fehler behoben werden. Deswegen wird die Produktion sehr stark vom iterativen Vorgehen geprägt sein und dem Feedback der Tester und Betreuer (vgl.projekte-leicht-gemacht.de 2015). Die Meilensteine des Projekts sehen wie folgt aus:

1. Konzeptphase
 - a. Prototyperstellung
 - b. Datenbankverknüpfung
2. Iterative Arbeit
 - a. *Layoutkonzept*
 - b. *UI Basics*
 - c. *Art Style*
 - d. Funktionen
 - e. Datenbankverbindung
3. Fertigstellung der App
 - a. *UI* abrunden
 - b. Export Test auf Plattformen

4. Durchführung

Ein halbes Jahr vor dem Beginn der Arbeit begann schon die Planung. Zu dieser Zeit entstand das *Pen & Paper* Regelwerk „Hell Over Mind“. Dazu gab es eine Webseite und eine App sollte folgen. Zwar wurde damals hauptsächlich an dem Regelwerk gearbeitet, jedoch bestand die Idee der App und entwickelte sich mit dem Regelwerk zu einem groben Plan. Es stand fest, dass die App den Spieler und somit den Nutzer der App, dabei helfen soll, das Regelwerk einfacher zu verstehen und den Charakterbogen zu ersetzen.

4.1. Aufbau und Ideen:

Um das Projekt zu verwirklichen, werden viele Programme benötigt. Eine davon ist *Unity 2020.1.6f1* als *Engine* um die App zu erstellen. Dazu muss bei der Erstellung als *Target* Plattform *Android* ausgewählt werden. Für Ideen, Tests und Konzepte kommt Google Docs zum Einsatz durch seine Vielseitigkeit und Speichermöglichkeit in der *Cloud*. Ebenfalls zum Speichern benötigt wird *GitHub* und *GitHub Desktop*, um das Projekt mit allen Ordner doppelt zu speichern. Bei visuellen Aufgaben wird das Grafikprogramm *Gimp 2.10.12* benutzt. Dazu zählen Einfärben, Größe ändern oder Bearbeitung von Bildern, sowie von Texten. Außerdem wird ein *Cloud-Server* benötigt, um alle Daten des Nutzers online auf einer Datenbank zu speichern. Dieser wird in einem späteren Kapitel erwähnt.

Das Grundkonzept (siehe Abb. 10) zeigt, wie die *HOM* App über Seiten verteilt wird, Werte sammelt und nutzt. Dabei sind die Wörter in den roten Kreisen, die jeweiligen Seiten, auf denen sich der Nutzer bewegt und die Wörter in den türkisen Vierecken Methoden oder Werte, mit denen der Nutzer arbeitet, um sich einen Charakter zu erstellen oder einen schon bestehenden zu bearbeiten. Die Werte und Methoden sind aus dem Regelwerk entnommen und stimmen mit der Spielweise aus dem Regelwerk überein.

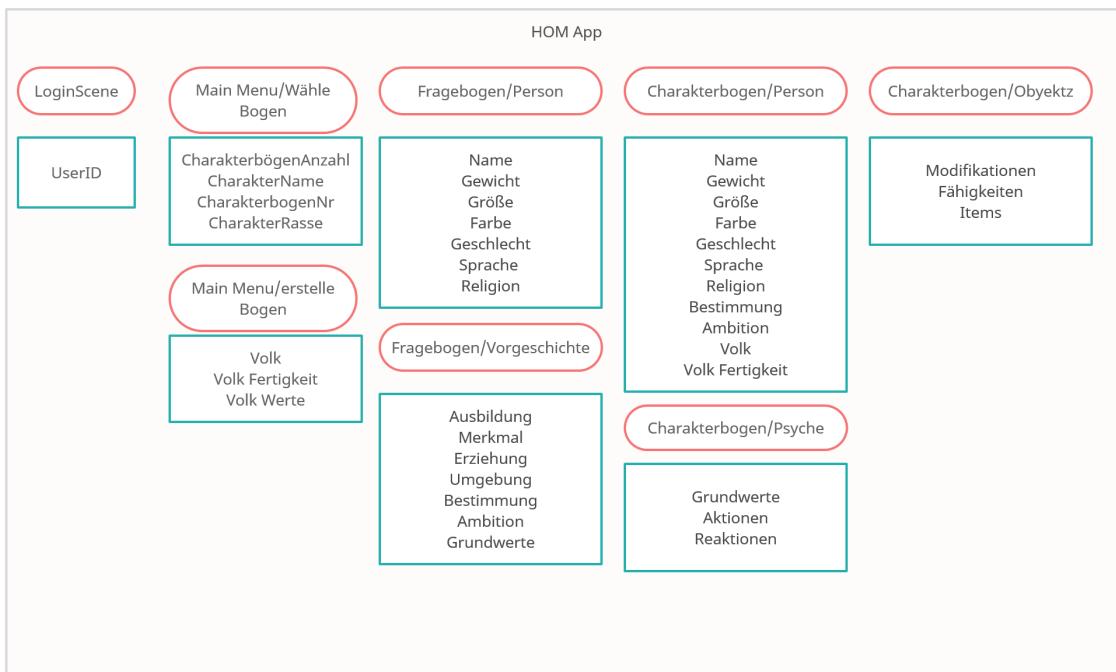


Abb. 10: Grundkonzept

In der *LoginScene* registriert oder meldet sich der Benutzer an, um eine einzigartige *UserID* zu erhalten. Diese ist nötig, um alle Werte passend zu dem Nutzer auf einer Datenbank zu speichern. Nachdem der Nutzer seine *UserID* erhalten hat, wird er in das Hauptmenü weitergeleitet, dort kann er zwischen dem Erstellen eines neuen Charakters oder einen schon bestehenden Charakter wählen. Wenn der Benutzer einen neuen Charakter erstellen möchte, werden ihm die Völker mit einzigartigen Fähigkeiten, Grundwerten und Geschichten angezeigt. Falls er sich für einen bestehenden Charakterbogen entscheidet, werden ihm alle bestehenden Bögen angezeigt mit Namen, Nummer und Volk. Die dritte Szene beinhaltet den Fragebogen, dort erstellt der Nutzer neue Charaktere. Dafür gibt es zwei Abteilungen, zuerst die Person, dort kann der Benutzer Daten eingeben, welche im späteren Spiel wenig Einfluss haben. Anders verhält es sich mit der Vorgeschichte, dort erhält der Benutzer für jede Wahl Werte in Form von Zahlen, welche die Stärke seines Charakters ausmacht. In der vierten und letzten Szene, steht es dem Benutzer frei seinen Charakter zu bearbeiten. Hierbei lässt sich zwischen drei Bereichen unterteilen. Im ersten, Person, geht es um alle persönlichen Daten des Charakters, vom Namen und Rasse, bis zur Bestimmung und Ambition. Der zweite Bereich, Psyche, beinhaltet alle Grundwerte des Charakters, um das Spiel zu spielen und auch die Möglichkeit mit diesen zu interagieren. Im letzten Bereich, *Obyektz*, kann der Benutzer eigene Modifikationen, Fähigkeiten und *Items* erstellen, speichern, sowie laden.

4.2. Prototyp:

Der Prototyp, benötigt als Erstes ein *Frontend*, weil es sich bei der App um Datenspeicherung und Wiedergabe handelt, wird eine Szene mit von *Unity* vorgegebenen *TMP Input Field* und *TMP Buttons* versehen. *TMP* bezeichnet dabei *TextMeshPro*, welches ein *Package* von *Unity* ist, um dabei zu helfen Texte schöner und performanter darzustellen (vgl. unity3d/textmeshpro 2019). Mit einem funktionierenden *Frontend* besteht die Möglichkeit, Daten des Benutzers auf einer Datenbank zu speichern und Daten vom Regelwerk aus, in der App wiederzugeben.

4.2.1. Erste Benutzerschnittstelle:

Den Anfang macht die erste Szene. Hier ist es wichtig dem Nutzer eine einzigartige *UserID* zu geben. Hinter dieser werden auf der Datenbank alle Daten gespeichert. Dafür ist ein Registrier und Anmelde-System vorgesehen (vgl. seo-analyse.com). Zum Anmelden oder Registrieren werden Nutzernname und Passwort benötigt, die mit der richtigen Kombination zu der gewünschten *UserID* führen. Wie in der *LoginScene* (siehe Abb. 11) gezeigt werden zwei *Input Fields* erstellt, um Daten durch die Eingabe des Benutzers zu erhalten. Durch den *Button* werden diese Daten gespeichert und überprüft (mehr in 4.2.2.). Außerdem gibt es ein vorläufiges Hintergrundbild durch ein *Image* eingesetzt, um die Szene komplett zu haben.

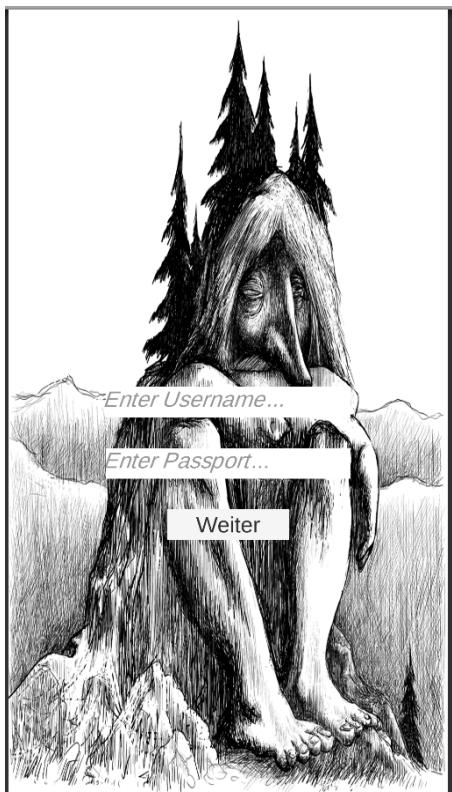


Abb. 11: LoginScene

Der Prototyp benötigt die Möglichkeit, dass der Benutzer einen Charakter erstellen kann. Deswegen ist die zweite Szene der Fragebogen. Dort werden alle persönlichen Werte des Charakters abgefragt. Auf dem Fragebogen (siehe Abb. 12) wird gezeigt, dass zu den *Input Fields* aus der *LoginScene* Text hinzugefügt wurde, welcher den Sinn des Inputs beschreibt. Außerdem steht hinter dem Text *Field* „Volk“ ein *TMP Dropdown*. Genauso wie bei „Sprache“ und „Religion“. Das *Dropdown* funktioniert wie ein *Input Field*, mit vorgegebenen Auswahlmöglichkeiten, von denen der Benutzer eine auswählen kann. Unten links steht „New Text“, dieser wird verändert, wenn im *Dropdown* eine Möglichkeit ausgewählt wird. Darauf wird in 4.2.2 näher eingegangen, was dort geschrieben wird und wo es herkommt. Ebenfalls fallen die drei Felder mit „Button“ auf, welche jedoch keine Funktionen haben und nach dem Prototyp verschwinden.



Abb. 12: Fragebogen

Mit beiden Szenen steht das *Frontend* für den Prototyp, welches alle Funktionen, die benötigt werden ausführen kann. Im Prototyp wird weder auf das *UX* eingegangen noch auf das *Layout*. Es geht um eine einfache Oberfläche zum Testen der ersten Funktionen und die Grundsteine für die Entwicklung der App.

4.2.2. Apache Web Server:

Zur Speicherung der Daten, die der Benutzer für seinen Charakterbogen braucht, ist eine Datenbank nötig. Diese wird für den Prototyp lokal erstellt und soll nur die Möglichkeit bieten, *Scripte* und Datenbankstruktur zu testen. Dazu werden zuerst kleine Datenbanken benötigt. Diese sollen Daten des Benutzers und seines Charakters speichern. Zur Erstellung des Servers wird das Programm Paket *XAMPP* heruntergeladen, das optimal für diese Anforderungen ist.

Mit dessen Hilfe wird sowohl ein *Apache Web Server* als auch *MySQL* auf dem Computer installiert. *Apache* wird benötigt, um eine Verbindung, zwischen *MySQL* und *Unity* herzustellen. In *XAMPP* (siehe Abb. 13), werden die *Services* danach aktiviert. Wodurch über den *Browser* mit der *URL*, `127.0.0.1/phpmyadmin`, die Verbindung mit *phpMyAdmin* aufgebaut werden.

Service	Module	PID(s)	Port(s)	Actions
	Apache	14860 13448	80, 443	Stop
	MySQL	5884	3306	Stop

Abb. 13: XAMPP

Auf der Browserseite, von *phpMyAdmin*, werden die Datenbanken mit *SQL* angelegt und betreut. Mit dem Befehl *CREATE DATABASE* (siehe Abb. 14), wird eine neue Datenbank erstellt, in die weiter Tabellen erstellt werden.

```
CREATE DATABASE charsheet;
```

Abb. 14: CREATE DATABASE

Durch *CREATE TABLE* (siehe Abb. 15) wird eine Tabelle erstellt, in die Datentypen, die gespeichert werden sollen, eingetragen werden. Wichtig ist dabei die richtige Benennung. Die Datentypen des Prototyps beinhalten dabei nur *INT* und *VARCHAR (255)*. Der *INT* speichert ganze Zahlen und *VARCHAR (255)*, 255 Buchstaben. Außerdem kommt in *charcreate*, der primäre Schlüssel *UserID* vor, der in allen Tabellen vorkommen soll, um die Daten einzigartig zu machen.

```
CREATE TABLE charcreate (
    UserID INT,
    CharCiv VARCHAR (255),
    CharName VARCHAR (255),
    CharWeight VARCHAR (255),
    CharHeight VARCHAR (255),
    CharAge VARCHAR (255),
    CharColor VARCHAR (255),
    CharLanguage VARCHAR (255),
    CharReligion VARCHAR (255),
    CharTraining VARCHAR (255),
    CharFeature VARCHAR (255),
    CharEducation VARCHAR (255),
    CharEnvironment VARCHAR (255)
);
```

Abb. 15: CREATE TABLE

Um mit der Datenbank zu interagieren, wird eine Skriptsprache benötigt, welche *queries* erstellen kann (vgl. w3schools.com 2021). *Queries* sind Anfragen an den Server, durch *SQL*. Also könnten sie auch in *SQL* geschrieben werden. Dafür werden *PHP Skripte* erstellt, die in vier Teile aufgeteilt sind. Zuerst wird ein Kontakt mit dem Server aufgebaut, durch die Funktion *mysqli_connect* (siehe Abb. 16). Diese benötigt als Parameter, den *hostname* für den Server. Dann wird durch den *username* und *password* in *phpMyAdmin* eingeloggt und die *database* zeigt die ausgewählte Datenbank an.

```
<?php
$hostname = "localhost";
$database = "charsheet";
$database_username = "root";
$database_password = "";

$connection = mysqli_connect($hostname, $database_username, $database_password, $database);
?>
```

Abb. 16: Connect

Wenn eine Verbindung zum Server besteht, wird damit begonnen, die Daten von *Unity* zu erhalten und in die *query* einzufügen. Dafür werden zuerst Variablen (siehe Abb. 17) mit Werten befüllt, indem sie nach Variablen mit dem *String* hinter *\$_POST* suchen. Diese werden von *Unity* aus gesendet, was später erwähnt wird.

```
$user = $_POST['username'];
$pass = $_POST['password'];
```

Abb. 17: PHP Variablen

Mithilfe der Variablen werden die *Queries* gebaut. Für den Prototyp werden die *SQL* Funktionen *SELECT* und *INSERT* benötigt (siehe Abb.18). *SELECT* nimmt die gesuchte Variable von einer Tabelle, wenn eine Bedingung erfüllt ist. In diesem Fall, wenn es den gewünschten *Username* gibt. *INSERT* erlaubt es, in eine Tabelle, Variablen mit Werten zu versehen. Beide sind eine *query*, welche mit der Funktion *mysqli_query* an den Server mit der zuvor aufgebauten Verbindung gesendet werden. Danach müssen die Funktionen durch *if-Anweisungen* aufgerufen werden und die *query* wird auf dem Server ausgeführt.

```

$queryt = "SELECT UserID FROM login WHERE Username = '$user'";
$cComeback = mysqli_query($connection, $queryt);
if($cComeback)
{
    if(mysqli_num_rows($cComeback) > 0)
    {
        echo "RegisterWrong";
    }
    else
    {
        $query = "INSERT INTO login ( Username, Passport) VALUES ('$user', '$pass')";
        $result = mysqli_query($connection, $query);
        if($result)
        {
            echo "Success";
        }
    }
}

```

Abb. 18: PHP query

4.2.3. C# Skripte:

In *Unity*, wird die objektorientierten Allzweck-Programmiersprache *C#* genutzt. Mit dieser werden Skripte erstellt, um mit der Benutzeroberfläche aus 4.2.1 zu arbeiten. Im Prototyp werden vier Arten von *Skripten* benötigt.

Zuerst *Manager*, welche alle wichtigen Funktionen der Szenen durchführen. Dazu gehören, das Aufrufen von *UI* Elementen, die dem Benutzer beim Start der Szene angezeigt werden sollen, Daten vom Input erhalten oder den Input mit Daten befüllen, Laden von neuen Szenen ermöglichen sowie die Verbindung mit der Datenbank beginnen.

Für die Verbindung mit der Datenbank sind die *Network Skripte* verantwortlich. In diesen werden *Koroutinen* gestartet (siehe Abb.19), welche laufende Funktionen sind. Um mit den *Koroutinen* eine Internetfunktion zu erreichen, wird die Klasse *UnityEngine.Networking* benötigt. Mit dieser wird durch *WWWForm* Daten gesammelt, die durch *UnityWebRequest* an die gewünschte *URL* gesendet werden. Danach wird auf eine Antwort von der *URL* gewartet. Mit der *URL* werden immer *PHP* Dateien auf dem Server aufgerufen.

```

public IEnumerator RegisterUser()
{
    WWWForm form = new WWWForm();
    form.AddField("username", name);
    form.AddField("passport", "12345");
    UnityWebRequest request = UnityWebRequest.Post(urlRegister, form);
    yield return request.SendWebRequest();

    if(request.isNetworkError || request.isHttpError)
    {
        Debug.LogError("Networkerror");
    }
    else
    {
        Debug.Log(request.downloadHandler.text);
        request.Dispose();
    }
}

```

Abb. 19: Netzwerk Koroutine

Neben diesen beiden Skripten, welche für jede Szene gemacht wurden, gibt es zwei Skripte, die zum besseren Umgang entwickelt wurden. Um viele Daten zu lesen der *JsonManager* und für die *Dropdowns* aus dem Fragebogen der *DropdownManager*. Dieser nimmt alle benötigten Daten für den *Dropdown* über den *Inspector* von *Unity* in die Klasse *DropdownInfos* (siehe Abb. 20) auf. Dazu gehören der zu bearbeitende *Dropdown*, wie viele Optionen er anzeigen soll, was in diesen Optionen stehen soll, ein anzuzeigender Text bei Auswahl einer Option, sowie das Textobjekt in dem der Text steht.

```

[System.Serializable]
public class DropdownInfos
{
    public TMP_Dropdown Dropdowns;
    public int OptionsNr;
    public TextAsset OptionsText;
    public TextAsset DescriptionFile;
    public TMP_Text DescriptionObject;
}

```

Abb. 20: Dropdown Klasse

Die Daten von *DropdownInfos* werden in einem *Array* gespeichert, welcher beim Start der Szene in der Funktion *CreateDropdown* (siehe Abb. 21) die Optionen für den Dropdown erstellen. Dies wird durch eine *foreach-Schleife* erreicht, die für jeden *Dropdown*, *CreateOptions* aufruft. Eine Funktion, die für die Anzahl der Optionen, eine *for-Schleife* laufen lässt, in der durch die Methode *Add* die Optionen erhalten werden.

```
void CreateDropdown()
{
    foreach (DropdownInfos i in Dropis)
    {
        i.Dropdowns.ClearOptions();
        CreateOptions(i.OptionsText, i.Dropdowns, i.OptionsNr);
    }
}

void CreateOptions(TextAsset File, TMP_Dropdown Down, int Options)
{
    for (int i = 0; i < Options; i++)
    {
        DropOptions.Add(ParseFile(File, i));
    }
    Down.AddOptions(DropOptions);

    DropOptions.Clear();
}
```

Abb. 21: Erstellung der Dropdowns

Um den Text für die Optionen zu erhalten, wird die Funktion *ParseFile* (vgl. Abb. 22) aufgerufen. Diese liest ein *TextAsset* ein und führt die Methode *Split("n[0]")* auf den Text aus. Dadurch wird der Text in ein *string Array* gespeichert. Um diese Methode zu ermöglichen, müssen die *TextAsset*, welche die Informationen für die Optionen beinhalten, nach jeder Option einen Zeilenumbruch haben.

```
//gibt den Optionen oder dem Text eine Schrift
private string ParseFile(TextAsset File, int Nr)
{
    //TXT Dateien aus dem Server holen
    string[] Text = File.text.Split("\n"[0]);
    return (Text[Nr]);
}
```

Abb. 22: Text Lesen

Der zuvor erwähnte *JsonManager*, funktioniert ähnlich wie *ParseFile*. Hier werden die Informationen aus einer *JSON-Datei* in einer Klasse gespeichert. Aus dieser Klasse wird dann ein *Array* erstellt, genauso wie in der *JSON-Datei* (vgl. Abb.23).

```
[System.Serializable]
public class BaseValues
{
    public BaseValue[] base1;
}

[System.Serializable]
public class BaseValue
{
    public string Name;
    public int[] GW;
}
```

Abb. 23: JSON Klasse

Um aus einer *JSON-Datei* mit den gleichen Werten wie in der Klasse *BaseValue* auslesen zu können, wird die Funktion *ReadJsonCivValues* benötigt (vgl. Abb. 24). Mit dieser wird ein *TextAsset* eingelesen und mit *JsonUtility* in *BaseValue* eingetragen.

```
public int ReadJsonCivValues(TextAsset ai, int line, int value)
{
    string jsoni = ai.text;

    BaseValues destinyInJson = JsonUtility.FromJson<BaseValues>(jsoni);
    int ergebniss = destinyInJson.base1[line].GW[value];
    return (ergebniss);
}
```

Abb. 24: JSON Klasse

4.3. Visuelle Überarbeitung nach Android UI Design:

Mit einer integrierbaren Benutzeroberfläche, Datenbank zum Speichern der Daten und Skripten steht der Prototyp mit seinen Grundfunktionen. Im dritten Kapitel wird die Benutzeroberfläche verändert. Es wird ein einheitliches Aussehen eingerichtet, mit *GIMP* als Grafikprogramm gearbeitet, sowie *Prefabs* erstellt.

4.3.1. Externe Arbeiten:

Mit *GIMP* als freies Grafikprogramm werden Bilder für den Hintergrund und Knöpfe bearbeitet. Dabei werden die Bilder in *Pixelgröße* erstellt. Die Bilder für den Knopf werden alle in der gleichen Datei erstellt. Diese hat eine Größe von **200x200 Pixeln** und eine Auflösung von **300x300 PPI**.

Um einen Knopf zu gestalten (vgl. Abb. 25), wird zuerst die Funktion des Knopfes definiert. Die Definition wird in einem Begriff zusammengefasst, der in ein Textfeld, welches das untere Drittel des Bildes einnimmt, geschrieben wird. Danach wird im Internet nach einem passenden Bild gesucht. Diese Bilder sind einfach gestaltet und übertragen symbolisch die Information. Diese Bilder werden mit zwei Werkzeugen bearbeitet. Zuerst werden sie auf **100x100 Pixel** skaliert. In der passenden Größe wird durch das Ausrichten das Symbol in der Mitte des Bildes eingerichtet.

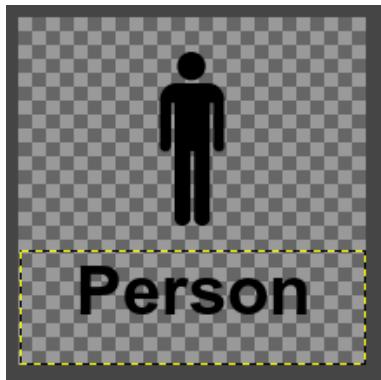


Abb. 25: Gimp Button

Es gibt es zwei Hintergründe, die grundsätzlich überall verwendet werden. Der *Standard Button*, der in der *LoginScene* oder *PopUps* verwendet wird (vgl. Abb. 26) und *InputField* Hintergrund, der bei Eingabefeldern Benutzung findet (vgl. Abb. 27). Diese beiden Hintergründe haben das gleiche Grundformat von *Gimp*. Sie sind **260x67 Pixel** groß und haben ebenfall eine Auflösung von **300x300 PPI**.

Der Standard *Button* hat einen ausgeschnittenen Teil von einem anderen Bild erhalten und beim *InputField*, wurden zwei goldene Striche oben und unten mit dem Stift gemalt.



Abb. 26: Standard Button

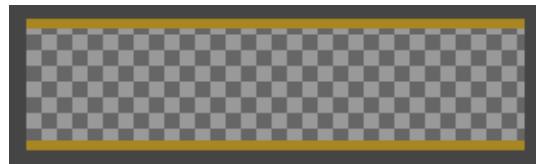


Abb. 27: Input Field

Mit neuen Bildern für *UI* Elemente, muss die Schrift bearbeitet werden. Dafür wurde die gleiche wie im Regelwerk ausgesucht, *Oswald Regular* (vgl. Abb. 28). Diese wird bei den Texten in *Unity* eingefügt.

```
Schriftname: Oswald Regular
Version: 3.0; ttfautohint (v0.95) -l 8 -r 50 -G 200 -x 0 -w "G" -W -c
OpenType-Layout, TrueType Konturen
```

```
abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890.:; ' " (!?) +-*=/=
```

Abb. 28: Oswald Regular

Für die Benutzeroberfläche wird abschließend eine Farbpalette für die Elemente ausgesucht. Dabei wird auf der Website, *flatuiccolors.com*, die Kanadische Palette ausgewählt. Diese wurde von dem Youtuber Game Dev Guide empfohlen (vgl. Game Dev Guide 2020).



Abb. 29: Farbpalette

Um die App mit Elementen aus dem Regelwerk zu schmücken, wurden Werke aus dem Regelwerk von den Artists besorgt. Dazu gehören die Bilder der Völker, Bestimmungen und Logo. An denen wurde nicht gearbeitet, sondern sie wurden eins zu eins eingefügt. Außerdem entwickelt die Artistin Anou ein *Icon* für die App (vgl. Abb. 30). Dieses wurde in *Unity* über die Projekteinstellungen eingefügt und wird auf dem Smartphone angezeigt.



Abb. 30: HOM Icon

4.3.2. Prefabs:

Um die Materialien aus dem letzten Kapitel zu nutzen, wurden *Prefabs* erstellt. *Prefabs*, sind *GameObjects*, die mit allen Werten und Komponenten als *Asset* gespeichert werden und somit als Vorlage für die Objekte in der Szene dienen (vgl. Unity3d.com/Prefabs). Ein weiterer Vorteil ist, dass die Objekte in der Szene sich verändern, wenn das *Prefab* geändert wird. Dadurch wird die Arbeit erheblich erleichtert. Allerdings muss jedes *Prefab*, nach dem Erstellen, entsprechend dem Ort, angepasst werden.

Bei den vier Grund Objekten der *UI*, gehört der Text (vgl. Abb. 31) an die erste Stelle. Dieser beinhaltet ein *TMP_Text* als Komponente. Dort wurde die Schriftfarbe auf Knochen gesetzt und die Schrift zu *Oswald Regular* geändert.



Abb. 31: Text Prefab

Bei dem *BildButton*(vgl. Abb. 32) gibt es eine *Image* und *Button* Komponente. Die *Transition* beim *Button* wird von *Color Tint* zu *Sprite Swap* bei der Aktivierung geändert . Dieses *Prefab* dient als Grundlage für alle Menü Knöpfe.



Abb. 32: BildButton

Der *NormalButton* (vgl. Abb. 33) ist eine Variante vom *BildButton*. Dabei wurde der *BildButton* verlängert und das *Image* wurde verändert. Außerdem besitzt der Knopf ein *TMP_Text* als Objekt, um dem *Button* Text in *Unity* zu geben.



Abb. 33: NormalButton

Das *InputFieldPlaceholder* (vgl. Abb. 34), ist das erste *Prefab* für das *TMP_InputField*. Es besitzt ein neues Hintergrundbild und hat eine goldenen Schriftfarbe. Außerdem besitzt es ein Objekt mit dem Namen *PlaceholderName* und ein Skript für dieses. Der *PlaceholderName*, hat eine Textkomponente und erhält den Text aus *Placeholder*, wenn das *InputFieldPlaceholder* ausgewählt wurde.



Abb. 34: InputFieldPlaceholder

Das *InputFieldTop* (vgl. Abb. 35), ist identische mit dem *InputFieldPlaceholder*, jedoch besitzt es ein *TopName* statt *PlaceholderName*. Der *TopName* besitzt eine Textkomponente, die dauerhaft angezeigt wird.

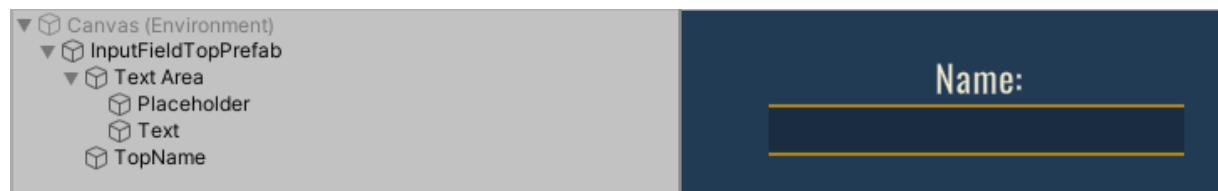


Abb. 35: InputFieldTop

Das Letzte *TextMeshPro* Objekt ist das *Dropdown* (vgl. Abb. 36). Es besitzt ein *TMP_Dropdown* und zwei zusätzliche Objekte. Genauso wie *InputFieldTop*, besitzt es ein *TopName* und dazu noch ein neuer Pfeil im *ImageDropPfeil*, das ein *Image* beinhaltet. Außerdem hat das *Dropdown* eine goldene Schrift und das *InputHintergrund* Bild.



Abb. 36: Dropdown

Die *GrundwertBox* (vgl. Abb. 37), ist eine Verbindung von mehreren *Prefabs*. Sie dient zur Darstellung der Grundwerte im Charakterbogen und beinhaltet vier *Prefabs*. Ein Text *Prefab* mit einem weiteren Text *Prefab* als *Child*. Diese zeigen die Grundwert abgekürzt und ausgeschrieben an. Die Abkürzung ist golden und hat eine *Font Size* von 94. Die ausgeschriebene Form ist Knochen weiß mit einer *Font Size* von 40. Außerdem liegt sie direkt unter der Abkürzung. Neben dem Text *Prefab*, liegen drei aneinander gereihte *InputFieldOnly* *Prefabs*. Diese sind nur *TMP_Dropdown*. Sie besitzen drei unterschiedliche *Images* um sie zu unterscheiden. Der Grundwert ist dunkel, der addierende Wert ist in grün und der abziehende Wert in Rot.



Abb. 37: GrundwertBox

Um dem Nutzer Feedback auf die Aktionen zu geben, werden *Pop Ups* benötigt. Deswegen gibt es ein *PopUp Prefab* (vgl. Abb. 38). Es besteht aus zwei Ebenen. Die erste ist der Hintergrund, der ein *Image* besitzt, welches sich über das gesamte Objekt zieht. Das *Image* hat die Farbe schwarz und die *Opazität* liegt bei 100 von 255. Die zweite Ebene enthält ein weiteres *Image*, mit einer höhere *Opazität*. Außerdem einen Text *Prefab* und *NormalButton* *Prefab*. Das *PopUp* wird über die *Manager* aufgerufen und verändert.

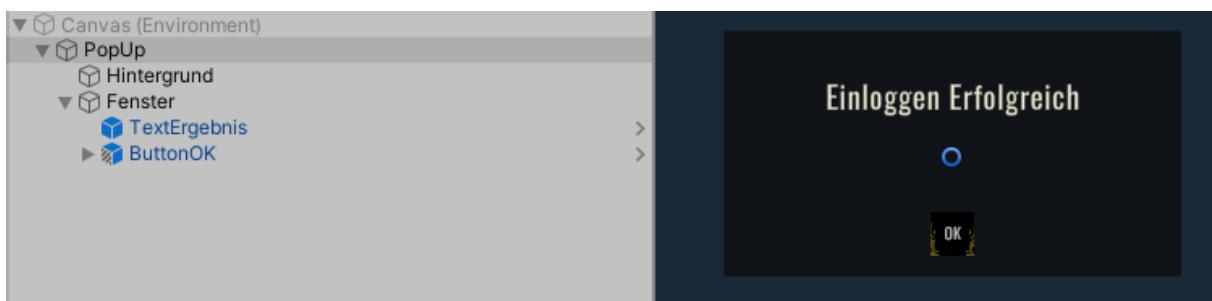


Abb. 38: PopUp

In der App gibt es die Möglichkeit zwischen Seiten zu wechseln. Dafür wird das Prefab *Swipe* (vgl. Abb. 39) benötigt. Dieser ist komplizierter, als die bisherigen *Prefabs*. Der *Swipe* teilt sich in zwei Objekte auf. Der *SwipeHeader* besitzt das *ButtonSwitch* (vgl. Abb. 40) Skript. Es ist für die Handhabung des *Swipe* verantwortlich. Dort werden die benötigten Seiten angegeben und dazu entsprechend viel *BildButton1* hinzugefügt. Der *BildButton1* ist dann in der Lage zu der ihm zugewiesenen Seite zu wechseln, durch die Funktion *SwitchSitesArrow* mit der Seitenzahl. Außerdem gibt es die *BildButton* „Zurück“ und „Vorwärts“. Sie gehen in den Seiten zurück oder vorwärts, durch die Funktion *GetSite* mit der 1 für vorwärts und -1 für zurück.

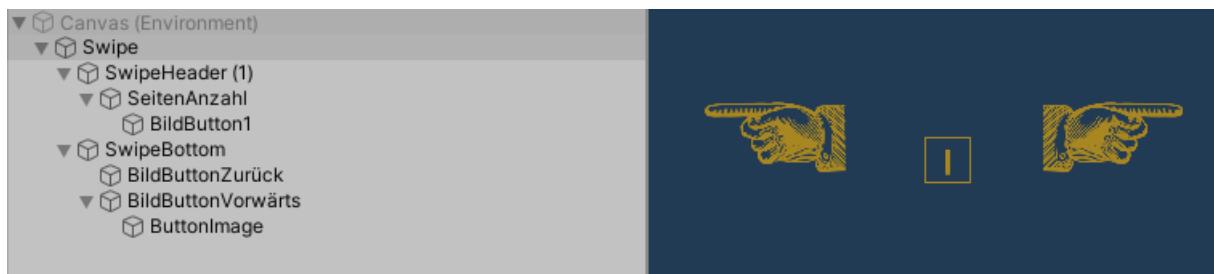


Abb. 39: Swipe

```
public void SwitchSitesArrows(int Site)
{
    side = Site;
    CheckArrows();
    ChangeSite();
}

public void GetSite(int multi)
{
    side = side+multi;
    CheckArrows();
    ChangeSite();
}

void ChangeSite()
{
    ReturnColor();
    Active.gameObject.SetActive(false);
    Active = Sites[side];
    Active.SetActive(true);
    NormalColor(side);
}

void CheckArrows()
{
    if (side <= 0)
    {
        Arrows[1].SetActive(true);
        Arrows[0].SetActive(false);
    }
    else if (side >= max)
    {
        Arrows[0].SetActive(true);
        Arrows[1].SetActive(false);
    }
    else
    {
        Arrows[0].SetActive(true);
        Arrows[1].SetActive(true);
    }
}
```

Abb. 40: ButtonSwitch

4.3.3. Änderung am Erscheinungsbild:

Mit den *Prefabs*, einer Farbpalette und neuen Bildern, ist es möglich eine App zu gestalten, die von Testern geprüft und weiter verbessert werden kann. Dazu werden die *Prefabs* aus dem letzten Abschnitt in die Szenen eingearbeitet und die Szenen mit *Layouts* und Farben verschönert. Die *Layouts* sind von *Unity* vorgegeben und sind ein Teil vom Auto-*Layout* (vgl. Unity3d/UIAutoLayout). Sie sind zur Anordnung und gleichmäßigen Verteilung von Objekten gedacht.

4.3.3.1. Login Scene:

Die erste Szene (vgl. Abb. 41) der App erhält zwei Texte als Überschrift, dazu das Logo vom Spiel in der Mitte und zwei *NormalButton* um sich anzumelden oder zu registrieren. Als Hintergrund wurde ein schwarzes Bild mit einem roten Farbverlauf in der Mitte erstellt. Beim Betätigen der *NormalButton*, wird das aktive Objekt deaktiviert und ein neues Objekt aktiviert. In dem neu aktiven Objekt sind *InputFields* um dem Benutzer eine *ID* zu geben.



Abb. 41: LoginScene

4.3.3.2. Menu Scene:

Die *MenuScene* (vgl. Abb. 42) besteht aus zwei Seiten. Von denen eine funktioniert. Der Hintergrund ist in drei Teile aufgeteilt, diese bestehen aus Objekten mit grauen *Images*. Im obersten Balken, sind drei Knöpfe, die zwischen den Menüseiten wechseln. Die Knöpfe werden als *Childs* durch eine *Horizontal Layout Group*, von *Unity*, geordnet. Der zweite Balken zeigt das Volk an, aus dem ein neuer Charakter geschaffen wird. Dazu gibt es ein *Dropdown*, das alle Völker anzeigt und je nach Auswahl das Bild ändert. Im untersten Balken erscheint ein Knopf, um in die Charaktererstellung zu gelangen, wenn ein Volk ausgewählt wurde.



Abb. 42: MenuScene

4.3.3.3. Fragebogen Scene:

Der Fragebogen, besitzt insgesamt vier Seiten, die alle auf dem gleichen Hintergrund abgespielt werden. Dieser besteht, wie das Menü aus drei Grauen Balken (vgl. Abb. 43). Am dunkelsten ist der linke Balken, der als Hintergrund für die Buttons der Seiten fungiert. Auf ihm liegen die vier Knöpfe der Seiten Person, Kultur, Jugend und Berufung. Diese werden durch eine *Vertical Layout Group* geordnet und in der oberen Ecke ein Knopf zum Speichern des Fragebogen. Der zweite Balken wird von dem Linken abgeschnitten und liegt im oberen Teil der Szene. Er dient als Hintergrund für die Überschrift, die in Gelb den Sinn der Seite erklärt. Der größte und hellste Balke, bietet den Platz für den Inhalt der Seiten.

Bei der Geburt (vgl. Abb. 43), werden die persönlichen Daten (Name, Gewicht, Größe, Alter, Haarfarbe, Hautfarbe und Geschlecht) erhoben. Sie werden durch sieben *InputFields* einer *Vertical Layout Group* geordnet. Kultur (vgl. Abb. 44), besitzt zwei *Dropdowns*, mit eigenen

Texten. Bei Veränderung vom *Dropdown*, werden die Texte dem Inhalt des *Dropdown* entsprechend angepasst. Die Inhalte sind alle aus dem Regelwerk, wurden jedoch stark vereinfacht. Es sind die vier Religionen und vier Sprachen auswählbar. Die Jugend (vgl. Abb. 45), besitzt vier Seiten, auf jeder Seite ist ein *Dropdown* mit eigenem Text, der sich entsprechend dem Inhalt ändert. Der Text ist dem Regelwerk nachempfunden und beinhaltet die Grundwerte, die durch die Auswahl erhalten werden. Vorhanden sind die Umgebung, Merkmal, Erziehung und Ausbildung.



Abb. 43: Geburt



Abb. 44: Kultur



Abb. 45: Jugend

In der Berufung (vgl. Abb. 46), gibt es zwei Seiten. Auf beiden sind *Dropdowns*, jedoch nur die erste Seite besitzt ein Bild. Dort werden die Tarotkarten der Bestimmungen angezeigt. Außerdem durch Texte die Inhalte des *Dropdown*. Auf der ersten Seite sind die acht Grundbestimmungen und auf der zweiten Seite alle Ambitionen



Abb. 46: Berufung

4.3.3.4. Charakterbogen Scene:

Der Charakterbogen (vgl. Abb. 47) hat den gleichen Hintergrund wie der Fragebogen. Jedoch besitzt er fünf Knöpfe, die zu den Seiten, Person, Psyche, Handlung, Fähigkeit und Inventar führen. Außerdem hat der *BildButton* zum Speichern ein neues Bild.

Die erste Seite des Charakterbogens ist ähnlich aufgebaut wie die erste Seite des Fragebogen. Die sieben persönlichen Eigenschaften, werden in *InputFieldsTop* angezeigt. Bei der Psyche (vgl. Abb. 48), kommt das *Prefab* der *GrundwertBox* zum Einsatz. Sie wird für die sechs Grundwerte, Agilität, Kraft, Ausdauer, Reflexe, Gesundheit und Verstand eingesetzt. Als Überschrift steht Inneres in Gelb und über den *GrundwertBoxen*, wurden vier Texte eingebaut. Diese vier Texte sind in einem *GameObject* vereint und dienen als Überschrift für die *GrundwertBox*. Jede *GrundwertBox* ist in einem *Game Object* mit einer *Vertical Layout Group*.



Abb. 47: Erscheinung

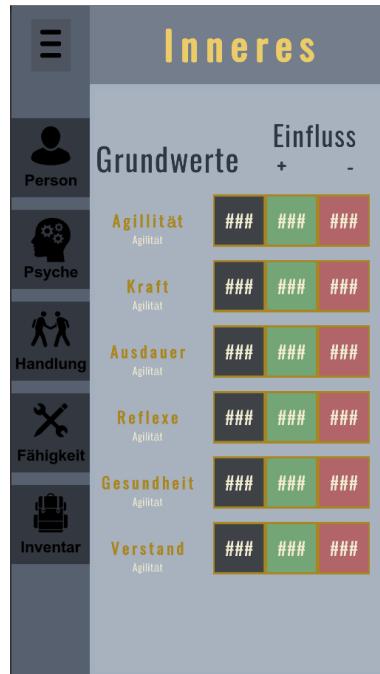


Abb. 48: Psyche

Bei der Interaktion (vgl. Abb. 49), werden die Aktionen und Reaktionen von „Hell Over Mind“ in *NormalButton* eingefügt. Diese *NormalButton*, erhalten das *Image* vom *InputField* und ein weiteres Textobjekt, um den Wert der Interaktion anzuzeigen. Mit den modifizierten *NormalButton*, werden Aktionen und Reaktionen gebildet. Diese sind zwei Objekte mit *Vertical Layout Group*, welche nebeneinander die Szene ausfüllen. Als Überschrift erhalten sie ein Text und zwischen den *NormalButton*, wird ein dünnes *Gameobject* mit einem grauen *Image* gezogen, zur Abtrennung. Die Fertigkeiten (vgl. Abb. 50), sind Daten, die unendlich oft erstellt werden können. Dafür wird auf der Seite mit drei *NormalButton*, die Art der Fertigkeit ausgewählt. Diese lässt sich durch das *Dropdown* noch detaillierter auswählen. Das *Dropdown* ändert seine Optionen je nach Fertigkeiten Art. Durch *InputFields* werden den Fertigkeiten Name, Erschöpfung und Effekt gegeben. Um die Fertigkeit auf der Datenbank zu speichern, wird der *NormalButton* „Erlernen“ gedrückt. In Ausstattung (vgl. Abb. 51), werden ebenfalls Daten zum Erstellen von Objekten gesammelt. Diese Daten werden über *InputFields* eingegeben und durch den *NormalButton* mit „Item speichern“ auf die Datenbank übertragen. Ein *Item* hat einen Namen, Type, Gewicht und Beschreibung. Außerdem wird die Belastung, durch die *Items* über *InputFields* angegeben.

Interaktion		
	Aktionen	Reaktionen
Person	Angriff #	Argumentieren #
	Verkleiden #	Absorbieren #
	Unterhalten #	Beten #
Psyche	Bewegung #	Verteidigen #
	Steuern #	Lügen #
	Wuchten #	Ausweichen #
Handlung	Analysieren #	
	Interagieren #	
	Kanalisieren #	
Fähigkeit		
Inventar		

Abb. 49: Interaktion

Fertigkeiten	
Person	Physisch Magni Synti Voodoo
	Erlernen
	Name:
Psyche	Kategorie:
	Option A
Handlung	Erschöpfung:
	Effekt:
Fähigkeit	
Inventar	

Abb. 50: Fertigkeiten

Ausstattung	
Person	Belastung: Maximal Momentan
	Item speichern
	Name
Psyche	
	Type
Handlung	
	Gewicht
Fähigkeit	
	Beschreibung
Inventar	

Abb. 51: Ausstattung

4.4. Der Cloud Server:

Im letzten Kapitel wurden die Oberflächen gezeigt, mit allen Möglichkeiten Daten einzugeben oder zu laden. Die Daten vom Menü oder Fragebogen müssen gespeichert und auf dem Charakterbogen wieder geladen werden. Außerdem müssen die Daten des Charakterbogens jederzeit bearbeitbar sein. Dazu wird ein *Cloud Server* benötigt. Der lokale *Apache Web Server* ist für Testzwecke ausreichend, jedoch wird für das fertige Produkt ein Server benötigt, der von überall aufgerufen werden kann. Dafür wird zuerst eine neue Methode der Datenspeicherung gezeigt. Danach der *Cloud Server* um den gearbeitet wird. Zuletzt die Kommunikation zwischen dem Server und *Unity*.

4.4.1. Neue Utility Scripte:

Um die Daten von *InputField* oder *Dropdown* zu speichern, wird ein *Dictionary* erstellt (vgl Abb. 52). In diesem *Dictionary* sind zwei *Strings* enthalten, der erste *String* speichert den Namen der Daten als *Key* (vgl. Abb. 53) und der zweite *String* die Daten vom *Input*. Dieses *Dictionary* liegt auf dem *ScriptableObject*, *PlayerData*, welches in den *Assets* liegt und dort die Daten zwischen speichert. Um auf diese Daten zuzugreifen, muss über *PlayerData* der *Key* aufgerufen werden. Die *Keys* beinhalten alle Daten, mit denen in dem Projekt gearbeitet wird.

```
public Dictionary<string, string> data = new Dictionary<string, string>();  
  
public void SaveDataInput(TMP_InputField input)  
{  
    string key = input.gameObject.GetComponent<DatabaseData>().id.ToString();  
    string value = input.text;  
  
    SaveDataString(key, value);  
}  
public void SaveDataText(TMP_Text input)  
{  
    string key = input.gameObject.GetComponent<DatabaseData>().id.ToString();  
    string value = input.text;  
  
    SaveDataString(key, value);  
}  
  
public void SaveDataString(string key, string value)  
{  
    if (data.ContainsKey(key))  
    {  
        data[key] = value;  
    }  
    else  
    {  
        data.Add(key, value);  
    }  
}
```

Abb. 52: Save Data

```

public enum DataId
{
    //NutzerDaten
    UserName, UserPassport, UserID,

    //Anzahl der Bögen
    UserCharSheets, SheetNr,

    //Speicherung der festen String Werte eines Charakters
    CharRace, CharRaceAbility, CharName, CharWeight, CharHeight, CharAge, CharHairColor, CharSkinColor, CharGender,
    CharLanguage, CharReligion, CharDestiny, CharDestinyLevel, CharAmbition, ModiAmount, AbilityAmount, ItemAmount,

    //wird für die GW Auswertung benötigt
    CharFeature, CharEducation, CharEnvironment, CharTraining,

    //Daten eine Modifikation
    ModiNr, ModiName, ModiPotenz, ModiLvl,

    //Daten eines Items
    ItemName, ItemType, ItemWeight, ItemDescription,

    //Daten einer Fertigkeit
    AbiName, AbiType, AbiSchool, AbiRange, AbiCost, AbiLength, AbiEffect,

    //Grundwert Daten
    AG, AGplus, AGminus, KR, KRplus, KRminus, AU, AUplus, AUminus, RE, REplus, REminus, GE, GEplus, GEminus, VE, VEplus, VEminus
}

public DataId id;

```

Abb. 53: Data Keys

4.4.2. Hetzner Server:

Mit der Möglichkeit, die Daten lokal zu speichern, wird ein *Cloud Server* zum Zwischenspeichern benötigt (vgl. Abb. 54). Dieser erfüllt die Mindestanforderungen und wird nur zum Speichern von Daten und *PHP* Dateien benötigt. Mit einer *IP-Adresse* des Servers und einem *Root-Passwort* wird über *PuTTY* auf den Server zugegriffen. *PuTTY* ist eine Applikation, die auf die Kommandozeile des Servers zugreift und über diese den Server verändert (vgl. Putty.org).

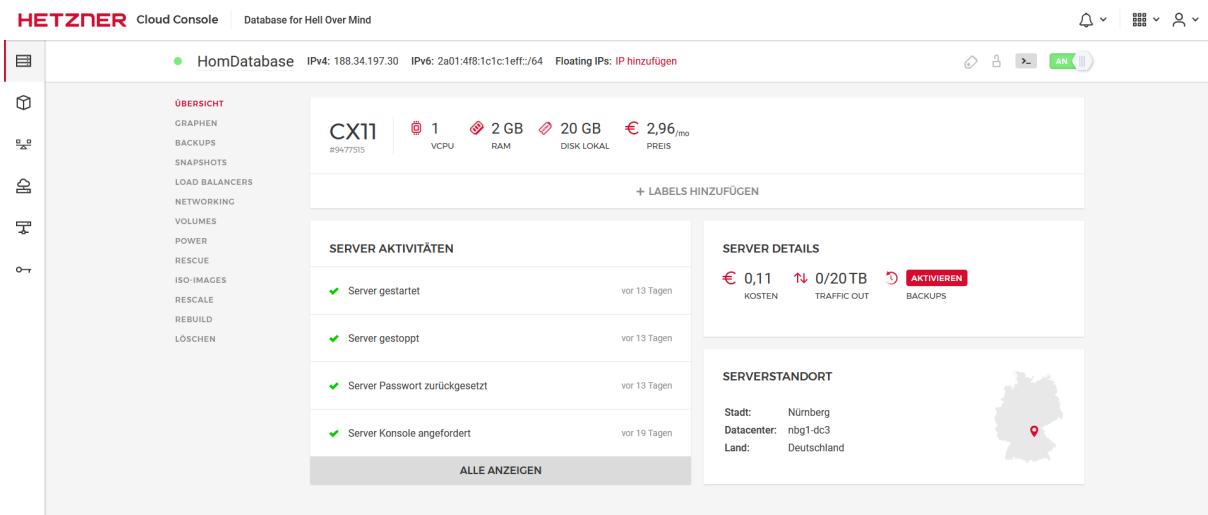


Abb. 54: Hetzner Server

Über *PutTY* werden *Apache*, *MariaDB-Server* und *PhpMyAdmin* installiert (vgl. Abb. 55). Dadurch ist der Server einsatzbereit und benötigt die neue Datenbank. Vom Prototyp wird *Login* übernommen und *charsheet* (vgl. Abb. 56) neu erstellt. *Charsheet* dient dazu, die Daten von *PlayerData* geordnet zu speichern.

```
apt-get update
apt-get upgrade
apt-get install sudo
apt-get install curl nano
apt-get install apache2
apt-get install mariadb-server mariadb-client
mysql_secure_installation
apt-get install php7.0 php7.0-cli php7.0-curl php7.0-gd php7.0-intl php7.0-json php7.0-mbstring
php7.0-mcrypt php7.0-mysql php7.0-opcache php7.0-readline php7.0-xml php7.0-xsl php7.0-zip
php7.0-bz2 libapache2-mod-php7.0 -y
service apache2 reload
apt-get install phpmyadmin
mysql -u root
CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON *.* TO 'username'@'localhost' WITH GRANT OPTION; (username und
password - Durch eure Daten ersetzen)
service apache2 reload
sudo ln -s /usr/share/phpmyadmin /var/www/html
```

Abb. 55: Lupari Befehle

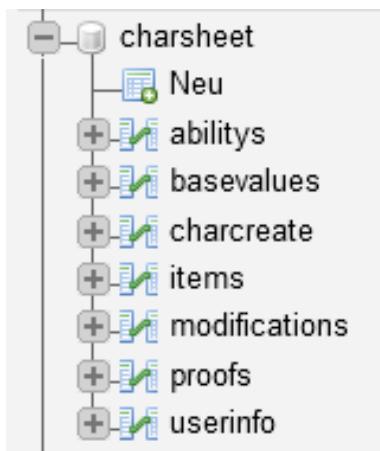


Abb. 56: charsheet

Die Bearbeitung des Servers über die Kommandozeile ist sehr aufwendig. Deswegen wird aus *WinSCP* zurückgegriffen, das eine benutzerfreundliche Oberfläche bietet und somit die Arbeit erleichtert. Wie bei *PutTY* wird mit der *IP*, dem *root* und Passwort angemeldet (vgl. Abb. 57). Mit dem Server verbunden, wird der Dateipfad „var/www/html/“ gesucht. Über diesen Ordner lässt sich mit dem Server interagieren. Deswegen werden die *php* Dateien zusammen mit *phpmyadmin* dort abgelegt (vgl. Abb. 58). Um auf diese Dateien des Servers zuzugreifen, wird zuerst die *IP* eingegeben, gefolgt von dem Dateinamen.

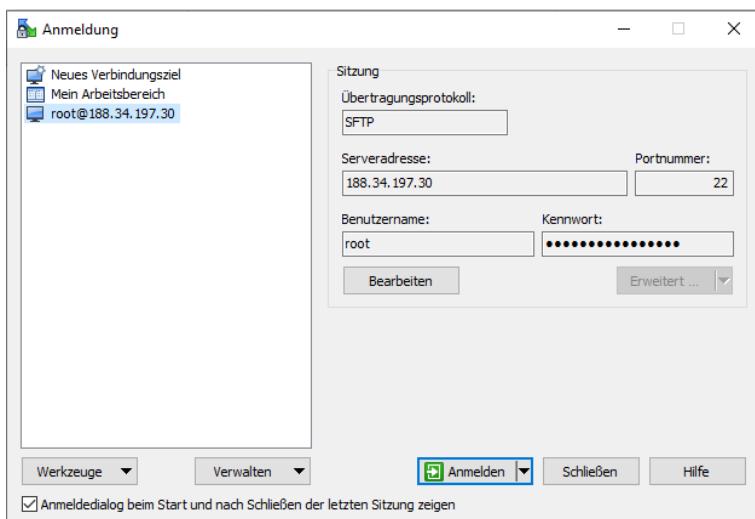


Abb. 57: WinSCP anmelden

Name	Größe	Geändert	Rechte	Besitzer
phpmyadmin		13.01.2021 00:02:08	rwxr-xr-x	root
DeleteAbility.php	1 KB	13.01.2021 00:12:17	rwxrwxrwx	root
DeleteItem.php	1 KB	20.02.2021 03:22:45	rw-r--r--	root
DeleteModi.php	1 KB	20.02.2021 03:23:08	rw-r--r--	root
GetAmounts.php	1 KB	19.02.2021 04:54:50	rw-r--r--	root
GetGW.php	1 KB	06.02.2021 16:04:17	rw-r--r--	root
GetPersonal.php	2 KB	06.02.2021 15:50:46	rw-r--r--	root
GetSheetInfos.php	2 KB	14.02.2021 04:29:31	rw-r--r--	root
GetUserInfos.php	1 KB	15.02.2021 02:49:13	rw-r--r--	root
index.html	11 KB	13.01.2021 00:02:09	rw-r--r--	root
LoadAbility.php	1 KB	19.02.2021 06:08:31	rw-r--r--	root
LoadItem.php	1 KB	19.02.2021 06:08:46	rw-r--r--	root
LoadModi.php	1 KB	19.02.2021 06:08:36	rw-r--r--	root
Login.php	2 KB	20.01.2021 15:35:06	rw-r--r--	root
Register.php	2 KB	27.01.2021 09:08:46	rw-r--r--	root
SafeAbility.php	1 KB	18.02.2021 19:51:56	rw-r--r--	root
SafeGW.php	2 KB	28.01.2021 17:45:58	rw-r--r--	root
SafeItem.php	1 KB	09.02.2021 01:27:44	rw-r--r--	root
SafeModi.php	1 KB	09.02.2021 23:18:32	rw-r--r--	root
SafeProof.php	2 KB	21.01.2021 23:02:48	rw-r--r--	root
SafeSheet.php	3 KB	27.01.2021 21:06:51	rw-r--r--	root
Test.php	1 KB	22.01.2021 19:11:30	rw-r--r--	root
UpdateAbility.php	1 KB	18.02.2021 19:53:19	rw-r--r--	root
UpdateAmounts.php	1 KB	11.02.2021 03:03:05	rw-r--r--	root
UpdateBaseValues.php	2 KB	14.02.2021 04:53:28	rw-r--r--	root
UpdateItem.php	1 KB	18.02.2021 20:31:27	rw-r--r--	root
UpdateModi.php	1 KB	18.02.2021 19:54:10	rw-r--r--	root
UpdatePersonal.php	2 KB	14.02.2021 04:18:28	rw-r--r--	root
UpdateUserInfo.php	1 KB	27.01.2021 09:21:21	rw-r--r--	root
UserInfoSave.php	1 KB	27.01.2021 02:11:39	rw-r--r--	root

Abb. 58: Internet Pfad

4.4.3. Weitere PHP Skripte:

Im Internetpfad (vgl. Abb. 58), sind viele *php* Dateien, um mit den Daten der App zu interagieren. *Login* und *Register* sind vom Prototypen. Diese benutzen die *INSERT* Methode (vgl. Abb. 59), genauso wie alle php Dateien, die mit *Safe* starten. Sie überträgt Daten von *Unitym, VALUES* auf die Datenbank *INTO*.

```
$UserID = $_POST['UserID'];
$SheetNr= $_POST['SheetNr'];

$query = "INSERT INTO userinfo (UserID, UserCharSheets) VALUES ('$UserID','$SheetNr') ";

$result = mysqli_query($connection, $query);
if($result)
{
    echo "User Saved";
}
else
{
    echo "lol";
}
```

Abb. 59: php INSERT

Die andere Methode ist *SELECT* (vgl. Abb. 60), die bei *php* Dateien mit *Load* und *Get* vorkommt. Dabei werden *UserID* und *SheetNr*, *WHERE*, von *Unity* an einen bestimmten *table*, *FROM*, gesendet und von dort bestimmte Daten, *SELECT*, zurückgesendet, über das Sprachkonstrukt *echo*. Zur Abtrennung der Daten wird ein „|“ verwendet. *Load* und *Get* sind die einzigen Funktionen, die Daten zurückgeben.

```
$UserID = $_POST['UserID'];
$SheetNr = $_POST['SheetNr'];

$query = "SELECT ModiAmount, AbilityAmount, ItemAmount FROM charcreate WHERE UserID='$UserID' AND SheetNr='$SheetNr'";
$result = mysqli_query($connection, $query);

if (mysqli_num_rows($result) > 0)
{
    // output data of each row
    while($row = mysqli_fetch_assoc($result))
    {
        echo $row["ModiAmount"]."|".$row["AbilityAmount"]."|".$row["ItemAmount"];
    }
}
else
{
    echo "0 results";
}
```

Abb. 60: php SELECT

Neue Dateien sind die *Update* Dateien, die mit der *UPDATE* Funktion laufen (vgl. Abb. 61). Beim *UPDATE*, werden Daten von *Unity*, *SET*, an eine bestimmte Datenbank gesendet und dort an eine vorgegebene Stelle, *WHERE*, angesetzt.

```

$UserID = $_POST['UserID'];
$SheetNr = $_POST['SheetNr'];
$query = "UPDATE userinfo SET UserCharSheets = '$SheetNr' WHERE UserID = '$UserID'";

$result = mysqli_query($connection, $query);
if($result)
{
    echo "gelungen";
}
else
{
    echo "lol";
}

```

Abb. 61: php UPDATE

Neben dem Verändern von Daten, gibt es das Löschen, durch *DELETE*, ausgeführt durch alle Dateien mit *Delete* am Anfang (vgl. Abb. 62). Dort wird eine Reihe gelöscht, von einer ausgewählten Datenbank, *FROM*, wo die von *Unity* gesendeten Daten, *WHERE*, passen.

```

$UserID      = $_POST['UserID'];
$SheetNr     = $_POST['SheetNr'];
$ModiNr      = $_POST['ModiNr'];

$query = "DELETE FROM modifications WHERE ModiNr = '$ModiNr' AND UserID = '$UserID' AND SheetNr = '$SheetNr'";

$result = mysqli_query($connection, $query);
if($result)
{
    echo "Modifikation gelöscht";
}
else
{
    echo "lol";
}

```

Abb. 62: php DELETE

4.5. Benutzerfreundlichkeit:

Für eine benutzerfreundliche App wird die Hilfe der Tester benötigt. Um die Informationen der Tester zu erhalten und auswerten zu können, wurden Fragebögen entwickelt. Diese sind wie folgt strukturiert. Zuerst werden die persönlichen Informationen erfragt, um die Probanden einordnen zu können. Wie gut ist die Erfahrung mit Apps? Wie viel Zeit wird am Smartphone verbracht und gibt es Erfahrungen mit dem Genre *Pen&Paper*? Danach werden den Testern Aufgaben gestellt, entweder sollen sie bestimmte Dinge in der App erledigen, sie begutachten oder gezielt nach Fehlern suchen. Wenn die Nutzer ihre Aufgaben geschafft haben, geben sie ein kurzes Feedback, das mit den Notizen des Testleiters zu einem Gesamtfeedback wird. Die Notizen sind Beobachtung der Tester, bei der Handhabung der App. Die entscheidenden Informationen sind das Gesamtfeedback, nachdem sich der folgende Feinschliff und Expertengespräche richten werden.

4.5.1. Tests:

Für den Kontakt mit den Testern gibt es persönliche Treffen und online Besprechen. Zuerst wird der persönliche Test (vgl. Tabelle 1 Offline Tests) betrachtet. Dabei beobachtet und dokumentiert der Testende durchgehend und kommt dadurch zu einem Ergebnis. Alle Probanden unterscheiden sich und bieten dadurch einen Vergleich zu einer breiten Zielgruppe. Es zeigt sich, dass sich die Probanden mit viel Smartphone Nutzung oder *P&P* Erfahrung leichter in der App zurecht fanden. Der Nutzer ohne *P&P* Erfahrung ist in der App verloren und benötigte Hilfe vom Testenden. Proband 1 und 2 finden die Funktionen schnell, haben jedoch ein Problem mit Aussehen und Umgang der Seiten.

Name	Nutzer Infos	Aufgaben	Feedback
Proband 1	-Experte mit Apps -ca. 6 h pro Tag -wenig Erfahrung in P&P	- Umgang und Funktionen der App teste -Aufgaben erfüllen	-benötigt eine Möglichkeit, zurück zu gehen -Swipe einbauen -Buttons überarbeiten -Anordnung ändern
Proband 2	-Amateur mit Apps -ca. 1 h pro Tag -etwas Erfahrung in P&P	-Aussehen der App bewerten -Aufgaben erfüllen	-simples Aussehen -Einheitlichkeit -Seiten sind unkenntlich
Proband 3	-Erfahren mit Apps -ca. 1-2 h pro Tag -keine Erfahrung in P&P	-Funktionen und Online Verhalten testen -Aufgaben erfüllen	-Seiten nicht offensichtlich -Buttons sind nicht erklärend -benötigt eine Möglichkeit, zurück zu gehen

Tabelle 1: Offline Tests

Nach der Bearbeitung und Umsetzung des Feedbacks vom ersten Test wird ein zweiter Test (vgl. Tabelle 2 Online Tests) entwickelt. Dort haben alle Nutzer gute Erfahrungen mit Apps, unterscheiden sich aber bei der P&P Erfahrung. Besonders Proband 6 fällt mit sieben Stunden am Handy und großer P&P Erfahrung auf. Der große Nachteil des Online Tests ist die Dokumentation des Testenden. Diese beruht ausschließlich auf der sprachlichen Kommunikation und Wiedergabe des Tests vom Nutzer. Somit kann nicht direkt auf die Handhabung eingegangen werden. Deswegen, werden die Tester angehalten, mehr in das Feedback zu schreiben und jeden Schritt in der App, den sie machen, zu erklären. Das Ergebnis der Tests ergibt, dass die Nutzer das Aussehen schlecht bis durchschnittlich empfinden und die Interaktionsmöglichkeiten unübersichtlich sind.

Name	Nutzer Infos	Aufgaben	Feedback
Proband 4	-Erfahren mit Apps -ca. 3 h pro Tag -keine Erfahrung in P&P	-Aussehen bewerten -Meinung zu Objekten	-Input benötigt Vorgaben -Anordnung fehlerhaft -Szenen übersichtlicher gestalten
Proband 5	-Erfahren mit Apps -ca. 3 h pro Tag -wenig Erfahrung in P&P	-Fragen beantworten -Aufgaben erfüllen -Feedback geben	-simples Aussehen -einige Objekte nicht gefunden -Anfänger freundlich
Proband 6	-Erfahren mit Apps -ca. 7 h pro Tag -große Erfahrung in P&P	-Aussehen bewerten -Meinung zu Objekten	-Benötigt neue Anordnung von Buttons -Überarbeitung der Seiten nötig -neue Farben, Zitat: "sieht aus wie Hitler Jugend"

Tabelle 2: Online Tests

4.5.2. Anwendung des Feedbacks:

Das Feedback der Tester wird sofort angegangen und kleinen Problemen werden sofort beseitigt. Ein großes Problem der Tester ist das Aussehen der App. „Manche Seiten sind nicht zu finden.“, „Es sieht nicht ansprechend aus.“ oder „Wo finde ich diese Funktion?“ sind nur einige Reaktionen der Tester. Deswegen wird die Hilfe von zwei Experten benötigt. Anema als UI Artist hat die benötigte Expertise, um zu beurteilen, was das Problem ist. Alcin als 2D Artist bewertet die Arbeit und gibt Hilfestellung bei der Veränderung.

Eine neue Farbpalette (vgl. Abb. 63) wird eingesetzt, um Hintergrund, Knöpfe und Schrift anzupassen. Gold wird für Ränder, Bilder und die ausgewählte Schrift genutzt. Für die

restliche Schrift dient ein Knochen-farbigen Weiß. Für den Hintergrund wird ein dunkles Rot und bei den Knöpfen werden helles Grün, helles Rot, durchscheinendes Weiß sowie durchscheinendes Schwarz genommen.



Abb. 63: neue Farbpalette

Danach wird der Hintergrund überarbeitet. Die grauen Balken werden durch ein Bild ersetzt, das dem Stil eine Tarotkarte nachempfunden ist (vgl. Abb. 64). Das Bild wird mit Gold und Rot eingefärbt, um einen einzigartigen Hintergrund zu erhalten (vgl. Abb. 65). Dieser Hintergrund wird auf allen Szenen angewendet, mit Ausnahme der *LoginScene*.

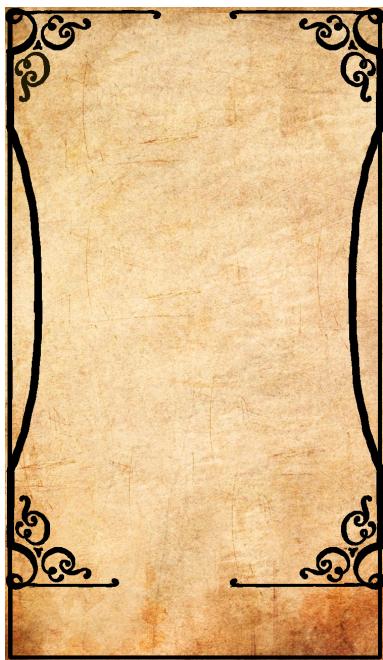


Abb. 64: Tarotkarte



Abb. 65: App Hintergrund

Mit dem neuen Hintergrund werden die Knöpfe (vgl. Abb. 66) überarbeitet. Sie erhalten einen goldenen Rand und vier unterschiedliche Hintergrundfarben. Diese wurden mit 40% Deckkraft aufgetragen und dienen dazu, bei der Aktivierung dem Nutzer ein Feedback zu geben, indem sie ihre Farbe wechseln.

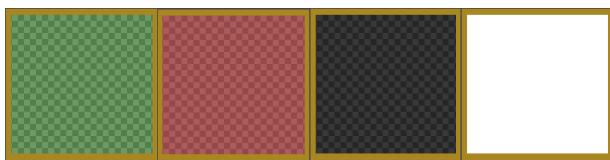


Abb. 66: Button Hintergründe

Außerdem wurden neue Objekte geschaffen. Die Auswahl der Charakterbögen (vgl. Abb. 67), die aus zwei *Grundwertboxen* besteht und einem *Dropdown*. Der *Dropdown* lässt dabei die Informationen in der unteren *Grundwertbox* ändern. Wenn ein Charakterbogen bearbeitet werden soll, wird immer der aus der *Grundwertbox* ausgewählt.



Abb. 67: Charakterbögen

Ähnlich verhält es sich bei der Anzeige von Informationen zu einem Volk (vgl. Abb. 68). Dafür wurde das *Dropdown*, Auswahl der Völker, um die Funktion erweitert, weitere Texte und sechs *InputFields* zu ändern, um aus einer *JSON-Datei* die Daten über das Volk in diese einzufügen.

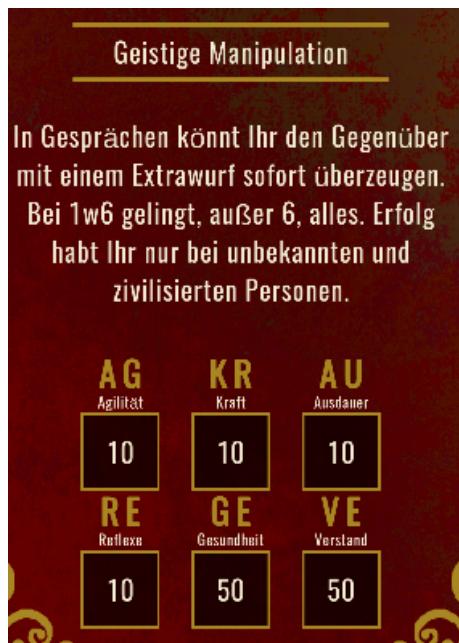


Abb. 68: Volks Infos

Um den Charakterbogen zu speichern, wurde das *PopUp* eingefügt, das den Nutzer doppelt auffordert zu bestätigen, um zu speichern (vgl. Abb. 69). Falls der Nutzer speichert, werden alle Daten aus den *InputFields*, welche sich in einem Array befinden, auf *PlayerData* übertragen (vgl. Abb. 70) und gespeichert.

```
public void Succes()
{
    suceder++;
    if (suceder == 1)
    {
        AreYouReady();
    }
    else if (suceder == 2)
    {
        WantToSave();
    }
    else if (suceder == 3)
    {
        SaveSheet();
    }
}
```

Abb. 69: PupUp Nachfrage

```
void DetermineValues()
{
    foreach (TMP_InputField inp in BaseValuesInputs)
    {
        Debug.Log("1");
        player.SaveDataInput(inp);
    }
    foreach (TMP_InputField inp in PersonInputs)
    {
        Debug.Log("2");
        player.SaveDataInput(inp);
    }
    player.SaveDataString("ModiAmount", amountsInt[0].ToString());
    player.SaveDataString("AbilityAmount", amountsInt[1].ToString());
    player.SaveDataString("ItemAmount", amountsInt[2].ToString());
}
```

Abb. 70: Werte speichern

Außerdem wurde die Möglichkeit eingeführt, kleine *Obyektz* zu speichern, upzudaten und zu löschen. Diese *Obyektz* sind Modifikationen (vgl. Abb. 71), Fähigkeiten und *Items*.



Abb. 71: Obyektz Anzeige

Das Speichern der *Obyektz* funktioniert wie beim Charakterbogen. Dabei werden die Daten aus den *InputFields* auf den Server übertragen. Um die Daten aus der Datenbank zu erhalten, werden sie in einer Liste gespeichert (vgl. Abb. 72). Danach werden die Namen der *Obyektz* in einem *Dropdown* dargestellt (vgl. Abb. 73). Die Optionen werden wie bei dem *DropdownManager* erstellt. Falls kein *Obyektz* vorhanden ist, wird in die *InputFields* und den *Dropdown* „leer“ eingefügt. Falls die Liste aber Daten enthält, werden bei Änderung der Optionen, vom *Dropdown*, werden die *InputFields*, von der Liste, überschrieben. Über den Knopf mit der Tonne, wird das ausgewählte *Obyektz* des *Dropdown* gelöscht und bei dem Knopf mit den Pfeilen, wird das *Obyektz* mit den Daten aus den *InputFields* überschrieben.

```
void ReadObyektz(string text, int type)
{
    string[] textArray = text.Split("|"[0]);
    //modi
    if (type == 0)
    {
        manage.modis = new ModiInfos[manage.amountsInt[type] + 1];
        for (int i = 0; i < manage.amountsInt[type]; i++)
        {
            manage.modis[i] = new ModiInfos();
            manage.modis[i].Nr = textArray[0+(4*i)];
            manage.modis[i].Name = textArray[1 + (4 * i)];
            manage.modis[i].Potenzial = textArray[2 + (4 * i)];
            manage.modis[i].Rank = textArray[3 + (4 * i)];
        }
    }
}
```

Abb. 72: Liste füllen

```

void CreateOptions()
{
    for (int i = 0; i < amountsInt.Length; i++)
    {
        if (amountsInt[i] > 0)
        {
            GiveOptionsAmount(i);
        }
        else
        {
            GiveNothing(i);
        }
    }
}

```

Abb. 73: Obyektz Optionen

Zuletzt hat der Nutzer die Möglichkeit, mit den Grundwerten zu interagieren. Dazu wurden die Grundwerte zu den Werten von jeder Aktion und Reaktion aus dem Regelwerk zusammengerechnet (vgl. Abb. 74).

```

public void ShowValues()
{
    int AG=(int.Parse(BaseValuesInputs[0].text)+int.Parse(BaseValuesInputs[1].text)-int.Parse(BaseValuesInputs[2].text) )/ 10;
    int KR=(int.Parse(BaseValuesInputs[3].text)+int.Parse(BaseValuesInputs[4].text)-int.Parse(BaseValuesInputs[5].text) )/ 10;
    int AU=(int.Parse(BaseValuesInputs[6].text)+int.Parse(BaseValuesInputs[7].text)-int.Parse(BaseValuesInputs[8].text) )/ 10;
    int RE=(int.Parse(BaseValuesInputs[9].text)+int.Parse(BaseValuesInputs[10].text)-int.Parse(BaseValuesInputs[11].text) )/ 10;
    int GE=(int.Parse(BaseValuesInputs[12].text)+int.Parse(BaseValuesInputs[13].text)-int.Parse(BaseValuesInputs[14].text) )/ 10;
    int VE= (int.Parse(BaseValuesInputs[15].text) + int.Parse(BaseValuesInputs[16].text) - int.Parse(BaseValuesInputs[17].text)) / 10;
    int Weapon = 0;
    int Cloth = 0;
    int Armor = 0;
    int MainValue = 0;
    ActionTexts[0].text = (Weapon + GE).ToString();
    ActionTexts[1].text = (Cloth + VE).ToString();
    ActionTexts[2].text = (RE + VE).ToString();
    ActionTexts[3].text = (AG + GE).ToString();
    ActionTexts[4].text = (RE + VE).ToString();
    ActionTexts[5].text = (KR + GE).ToString();
    ActionTexts[6].text = (AU + VE).ToString();
    ActionTexts[7].text = (AG + VE).ToString();
    ActionTexts[8].text = (MainValue + VE).ToString();
    ActionTexts[9].text = (RE + VE).ToString();
    ActionTexts[10].text = (AU + VE).ToString();
    ActionTexts[11].text = (AU + VE).ToString();
    ActionTexts[12].text = (Armor + GE).ToString();
    ActionTexts[13].text = (RE + VE).ToString();
    ActionTexts[14].text = (AG + GE).ToString();
}

```

Abb. 74: Aktionen und Reaktionen berechnen

Die Werte der Interaktionsmöglichkeiten sind *Child* eines *Button* der, wenn er gedrückt wird, anzeigt, was gewürfelt werden muss (vgl. Abb. 75).

```
public void GetValue()
{
    Master = GameObject.FindGameObjectWithTag("Master");
    TMP_Text[] text = gameObject.GetComponentsInChildren<TMP_Text>();
    int value = int.Parse(text[1].text);
    Master.GetComponent<DiceScript>().ShowDice(value);
}
```

Abb. 75: Zeige Würfel

Wenn der Nutzer den Würfel Knopf drückt (vgl. Abb. 76), wird durch einen *Randomizer* das Ergebnis berechnet und dem Nutzer präsentiert.



Abb. 76: Würfel

5. Ergebnisse

Im Folgenden werden die Szenen mit den Seiten der App gezeigt und anschließend verglichen, um festzustellen, welche Ziele erreicht wurden. Zuletzt wird die persönliche Meinung zu dem Projekt abgegeben.

5.1. Präsentation:

Die *LoginScene* (vgl. Abb. 77) besteht aus drei Seiten. Sie bieten dem Nutzer die Möglichkeit sich zu registrieren oder anzumelden. Dadurch erhält der Nutzer eine *ID* und den Zugang zum Menü.



Abb. 77: LoginScene komplett

Das Menü besteht aus zwei Bereichen. Der erste Bereich (vgl. Abb. 78) umfasst die Charaktererstellung. Auf der ersten Seite wird ein Volk ausgewählt, das als Bild präsentiert wird. Zu dem ausgewählten Volk, gibt es auf der zweiten Seite eine kurze Geschichte und auf dritten Seite die Grundwerte, sowie die Volksfähigkeit. Im unteren Teil der Szene sind drei Knöpfe. Die Äußenen wechseln zwischen den Bereichen und der Mittlere wechselt die Szene.



Abb. 78: MainMenu Charaktererstellung

Im zweiten Bereich (vgl. Abb. 79) werden die vorhandenen Charakterbögen gezeigt. Dort wird ein Bogen ausgewählt, der bearbeitet werden kann.



Abb. 79: MainMenu Charakterbögen

Die Charaktererstellung teilt sich in vier Bereiche auf. Diese sind über Knöpfe im unteren Bereich zu erreichen. Beim Beenden der Charaktererstellung kann über den Speicherknopf oben links, die Charaktererstellung beendet werden.

Die drei ersten Bereiche (vgl. Abb. 80) teilen sich für die persönlichen Daten in Geburt und Kultur, sowie Jugend für die Grundwerte auf. In der Berufung (vgl. Abb. 81) kann der Charakter sich einen bestimmten Spielstil aussuchen, dem er folgen möchte.

Abb. 80: Charaktererstellung Nr.1

Abb. 81: Charaktererstellung Nr.2

Nach der Charaktererstellung, ist der Charakterbogen, die letzte Szene. Er ist ähnlich wie der Fragebogen aufgebaut und besitzt sowohl einen Speicherknopf, als auch die unteren Knöpfe zum Wechseln der Seiten. Er teilt sich in fünf Bereiche auf. Im ersten Bereich (vgl. Abb. 82) werden alle Daten aus dem Fragebogen übertragen.



Abb. 82: Charakterbogen Person

Im zweiten Bereich (vgl. Abb. 83), werden die Grundwerte des Charakters in der ersten Seite Inneres angezeigt. In der Tabelle werden die Werte und die positiven sowie negativen Einflüsse angezeigt. Auf der zweiten Seite Inneres kann sich der Nutzer eigene Modifikationen erstellen. Die Interaktionen zeigen die Handlungen mit Aktionen und Reaktionen für das Spiel an. Außerdem kann hier gewürfelt werden.

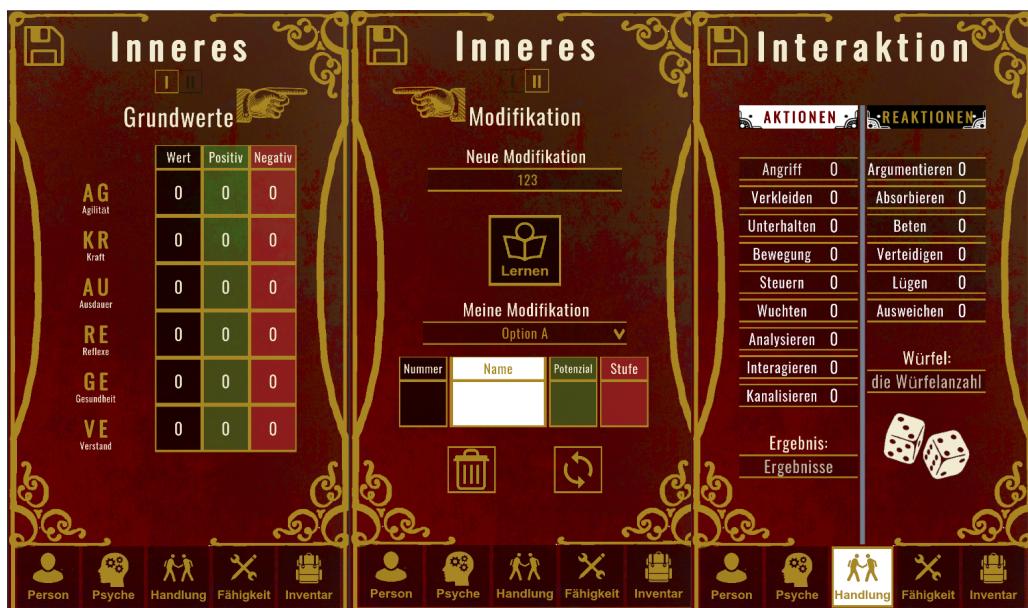


Abb. 83: Charakterbogen Grundwerte

Zuletzt gibt es Fertigkeiten und Ausstattung (vgl. Abb. 84), die es dem Spieler erlauben, sich eigene Fertigkeiten und *Items* zu erstellen und zu modifizieren.



Abb. 84: Charakterbogen Fähigkeiten und Ausstattung

5.2. Ziele:

Wie sich schon in der Präsentation der Ergebnisse zeigt, ist die App erstellt. Somit sind alle Soll-Ziele (siehe Tabelle 3) im Bereich der App erreicht und die Hälfte der Kann-Ziele ebenfalls. Besonders bei der Datenbank sind alle Ziele umgesetzt, während bei den Tests nur 50 % geschafft wurde.

Soll-Ziele:	
Erstellung eines „Hell Over Mind“ Charakters	vorhanden
Modifizierung des Charakters	vorhanden
Anzeigen und Interaktion von Werten	vorhanden
Speichern von Daten auf der Datenbank	möglich
Laden der Daten von der Datenbank	möglich
Usability-Tests	gemacht
Kann-Ziele:	
Erstellen und verändern von Fertigkeiten	möglich

Erstellen und Verändern von Items	möglich
Funktionierendes Hekatem System	nicht integriert
Eigene Klassen und Rassen	nicht geschafft
Verändern der Daten von der Datenbank	möglich
Löschen von Daten auf der Datenbank	möglich
Praxis-Test	noch nicht geschafft

Tabelle 3: Erreichte Ziele

5.3. Kritische Reflexion:

Durch die vielen Werkzeuge, mit denen gearbeitet wurde, war der Einstieg sehr langsam und mühselig. Es mussten neue Sprachen und der Umgang mit *Unity* und *GIMP* gelernt werden. Das Aufsetzen des Servers und die spätere Interaktion waren die umfangreichste Arbeit. Ich verwendete ganze Wochen zu lernen, wie ich damit umzugehen habe. Der positive Aspekt ist im Nachhinein, dass ich sehr viel Neues gelernt habe und nun anwenden kann.

Im Projekt hat mir die direkte Entwicklung an den Skripten und der Umgang mit den Testern, sowie die Tests an sich gefallen. Was mir weniger lag, war das *UI* und die Anordnung der vielen Objekte. Dabei kommt es oft auf millimetergenaue Arbeit an und häufiges Wiederholen. Jedoch war diese Arbeit nichts im Vergleich zu *PHP*. Die Entwicklung der *PHP* Skripte hat die meiste Arbeit gefressen, mit dem niedrigsten Aufwand. Die Skripte bestehen hauptsächlich aus *Copy-and-paste* und dem Senden eines *Strings* durch die *query*. Das Problem ist der *String*, der meistens falsch war. Das Finden des Problems war sehr anstrengend, da es keine Möglichkeit gab, diesen Fehler anzuzeigen. Ich war somit ewig mit der Suche beschäftigt.

Außerdem konnte ich mich schlecht auf die Situation mit der globalen Pandemie einstellen. Mein erster Plan wurde dadurch terminiert und dies hat mir viel Motivation gekostet, was dazu geführt hat, dass ich mir einen Monat Urlaub nehmen musste.

Rückblickend habe ich viel Neues gelernt und auf jeden Fall mehr Lust auf die Entwicklung von Apps bekommen. Mit den neu erlernten Fähigkeiten kann ich hoffentlich in der Zukunft viel erreichen.

Ebenso war das kontinuierliche Arbeiten mit Testern und deren Feedback eine interessante Erfahrung. Das Thema visuelle Aspekte einer App werde ich in Zukunft lieber Experten überlassen und mir bei Bedarf deren Knowhow einholen.

6. Rekapitulation:

Die Entwicklung der App, in allen Facetten, ist ein aufwendiges und kostspieliges Unterfangen. Umso mehr Features in die App einfließen, desto mehr Aufwand und Wissen ist erforderlich. Besonders zu Beginn der Entwicklung ist viel Recherche nötig, um Unmengen an Basiswissen zu sammeln. Es muss eine konkrete Idee geben, mit definierten Zielen. Mit diesen Zielen können Werkzeuge ausgewählt werden, um die Idee umzusetzen. Wichtig ist eine gute und vertraute Entwicklungsumgebung. Diese ist mit *Unity* gegeben, auf der aufgebaut wird. Wenn man allerdings allein eine App entwickelt, genügt *Unity* nicht. Die Gestaltung durch *GIMP* stellt eine große Herausforderung dar. Das Problem ist nicht die Bildbearbeitung, in der *GIMP* sehr gut ist, sondern die Beschaffung der Bilder. Ohne Artists oder eigene Kenntnisse ist man bei diesem Aspekt aufgeschmissen. Ebenso ist die Entwicklung mit Datenbanken, *PHP* und *SQL* für einen Neueinsteiger eine riesige Aufgabe. Es hilft Grundkenntnisse in der Programmierung zu haben und sie anwenden zu können. Außerdem gibt es im Internet und in der Literatur viel Hilfe und somit sollte kein Problem ein Dead-End bedeuten. Heutzutage ist es leicht eine App zu entwickeln, jedoch schwer diese nutzerfreundlich zu gestalten. Deswegen sind die kontinuierlichen Tests mit Nutzern wichtig, um mit ihnen die App zu verbessern. Am Ende steht ein Produkt, das die mühselige Arbeit einen Charakterbogen zu erstellen, erleichtert, um mehr Zeit beim *Pen & Paper* zum Spielen zu haben.

IV. Literaturquellen:

Adams, Vernon (2012), Font Squirrel OSWALD, 30.03.2012,
<https://www.fontsquirrel.com/fonts/oswald>, letzter Zugriff: 24.02.2021.

Apache.org (2004), The Apache Software Foundation, January 2004
<https://www.apache.org/>, letzter Zugriff: 24.02.2021.

Awkwardly Developed Apps, Universal Character Sheet, 2019, letzter Zugriff 9.2.2021

Bitvise (2020), Download PuTTY: latest release (0.74), 2020-06-27, <https://www.putty.org/>,
letzter Zugriff: 24.02.2021.

DocFX (2020), Auto Layout, 06.10.2020,
<https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/comp-UIAutoLayout.html>
, letzter Zugriff: 24.02.2021.

Elsam, Sara (2020), 10 best tabletop roleplaying games out right now, 13.02.2020,
<https://www.dicebreaker.com/categories/roleplaying-game/best-games/best-tabletop-roleplaying-games>, letzter Zugriff: 24.02.2021.

Figal, Günther/Hans Jürgen Wendel (2009), Verstehensfrage, 1. Edition, Tübingen

Game Dev Guide (2020), Making UI That Looks Good In Unity using Color Palettes, Layout Components and a Blur Pane, 18.02.2020,
<https://www.youtube.com/watch?v=HwdweCX5aMI>, letzter Zugriff: 24.02.2021.

Neukirchner, Anne (2014), Ausflug in die Welt von Pen und Paper - Die besten Rollenspiel - Umsetzungen, 29.03.2014,
<https://www.pcgames.de/Spiele-Thema-239104/Specials/Ausflug-in-die-Welt-von-Pen-und-Paper-Die-besten-Rollenspiel-Umsetzungen-1115283/>, letzter Zugriff: 24.02.2021.

Riggs, Ben (2017), The Story of D&D Part One: The Birth, Death, and Resurrection of Dungeons & Dragons, 26.12.2017,
<https://geekandsundry.com/the-story-of-dd-part-one-the-birth-death-and-resurrection-of-dungeons-dragons/>, letzter Zugriff: 24.02.2021.

Serge Shustoff, Character Sheet for any RPG, 2020, letzter Zugriff 9.2.2021

Sinicki, Adam (2020), How to create non-game apps in Unity, 19.01.2020,
<https://www.androidauthority.com/make-unity-apps-1073017/>, letzter Zugriff: 24.02.2021.

SOE-Analyse.com (2004), SEO Lexikon, 2004,
<https://www.seo-analyse.com/seo-lexikon/e/einloggen/>, letzter Zugriff: 24.02.2021.

Stabinskas, Edward (2019), SQLite vs MySQL – What's the Difference, 27.03.2019,
<https://www.hostinger.com/tutorials/sqlite-vs-mysql-whats-the-difference/>, letzter Zugriff: 24.02.2021.

Suehring, Steve (2002), MySQL Bible, 1. Aufl., New York

Talbert, Lance (2019), Introduction to Mobile Development with Unity, 16.01. 2019,
<https://www.red-gate.com/simple-talk/dotnet/c-programming/introduction-to-mobile-development-with-unity/>, letzter Zugriff: 24.02.2021.

Thornsby, Jessica (2016), Android UI Design, 1st. Edition, Birmingham

Unity Dokumentation (2019), TextMeshPro, 04.2019,
<https://docs.unity3d.com/Manual/com.unity.textmeshpro.html>, letzter Zugriff: 24.02.2021.

Unity Dokumentation (2018), Prefabs, 31.07.2018,
<https://docs.unity3d.com/Manual/Prefabs.html>, letzter Zugriff: 24.02.2021.

w3schools.com (1999), PHP MySQL Database, Erscheinungsdatum,
https://www.w3schools.com/php/php_mysql_intro.asp, letzter Zugriff: 24.02.2021.

Walter Kammerer, Fifth Edition Character Sheet, 2014, letzter Zugriff 9.2.2021

Walther, Nils/Phillip Frahne/Sarah M. Carstens/Alcin Narinc/Anou Lange (2020), Hell Over Mind, 1. Aufl., Hamburg

Windolph, Andrea & Dr. Blumenau, Alexander (2015), Die Meilensteintrendanalyse einfach erklärt, 13.05.2015,
<https://projekte-leicht-gemacht.de/blog/pm-methoden-erklaert/die-meilensteintrendanalyse-einfach-erklaert/>, letzter Zugriff: 24.02.2021.

Wirsing, Martin (2006), Projektmanagement: Prozessmodelle, 07.2006
https://www.pst_ifi.lmu.de/Lehre/WS0607/pm/vorlesung/PM-02-Prozess.pdf, letzter Zugriff: 24.02.2021.

WordPress.com. (2009), Was ist ein Vergleich/die Vergleichsmethode? Eine Definition, 13.04.2009, <https://vergleichsmethode.wordpress.com/>, letzter Zugriff: 24.02.2021.

V. Bildquellen :

Abbildung 1	Walther et al., 2019, s. 1
Abbildung 2	https://boardgamegeek.com/image/3353867/chainmail , abgerufen am 24.02.2021
Abbildung 3	Eigene Darstellung
Abbildung 4	Eigene Darstellung
Abbildung 5	Eigene Darstellung
Abbildung 6	Eigene Darstellung
Abbildung 7	https://www.seo-kueche.de/lexikon/user-interface/ , abgerufen am 24.02.2021
Abbildung 8	https://telegra.ph/Linear-layout-08-10 , abgerufen am 24.02.2021
Abbildung 9	https://de.wikipedia.org/wiki/SQL , abgerufen am 24.02.2021
Abbildung 10	Eigene Darstellung
Abbildung 11	Eigene Darstellung
Abbildung 12	Eigene Darstellung
Abbildung 13	Eigene Darstellung
Abbildung 14	Eigene Darstellung
Abbildung 15	Eigene Darstellung
Abbildung 16	Eigene Darstellung
Abbildung 17	Eigene Darstellung
Abbildung 18	Eigene Darstellung
Abbildung 19	Eigene Darstellung
Abbildung 20	Eigene Darstellung

Abbildung 21	Eigene Darstellung
Abbildung 22	Eigene Darstellung
Abbildung 23	Eigene Darstellung
Abbildung 24	Eigene Darstellung
Abbildung 25	Eigene Darstellung
Abbildung 26	Eigene Darstellung
Abbildung 27	Eigene Darstellung
Abbildung 28	Eigene Darstellung
Abbildung 29	https://flatuic平滑色.com/ , abgerufen am 24.02.2021
Abbildung 30	Lange, 2021
Abbildung 31	Eigene Darstellung
Abbildung 32	Eigene Darstellung
Abbildung 33	Eigene Darstellung
Abbildung 34	Eigene Darstellung
Abbildung 35	Eigene Darstellung
Abbildung 36	Eigene Darstellung
Abbildung 37	Eigene Darstellung
Abbildung 38	Eigene Darstellung
Abbildung 39	Eigene Darstellung
Abbildung 40	Eigene Darstellung
Abbildung 41	Eigene Darstellung
Abbildung 42	Eigene Darstellung

Abbildung 43	Eigene Darstellung
Abbildung 44	Eigene Darstellung
Abbildung 45	Eigene Darstellung
Abbildung 46	Eigene Darstellung
Abbildung 47	Eigene Darstellung
Abbildung 48	Eigene Darstellung
Abbildung 49	Eigene Darstellung
Abbildung 50	Eigene Darstellung
Abbildung 51	Eigene Darstellung
Abbildung 52	Eigene Darstellung
Abbildung 53	Eigene Darstellung
Abbildung 54	https://console.hetzner.cloud/projects/685165/servers/9477515/overview , abgerufen am 24.02.2021
Abbildung 55	https://www.youtube.com/watch?v=UBrg1eUGRQ , abgerufen am 14.01.2021
Abbildung 56	Eigene Darstellung
Abbildung 57	Eigene Darstellung
Abbildung 58	Eigene Darstellung
Abbildung 59	Eigene Darstellung
Abbildung 60	Eigene Darstellung
Abbildung 61	Eigene Darstellung
Abbildung 62	Eigene Darstellung
Abbildung 63	Eigene Darstellung
Abbildung 64	Eigene Darstellung

Abbildung 65	Eigene Darstellung
Abbildung 66	Eigene Darstellung
Abbildung 67	Eigene Darstellung
Abbildung 68	Eigene Darstellung
Abbildung 69	Eigene Darstellung
Abbildung 70	Eigene Darstellung
Abbildung 71	Eigene Darstellung
Abbildung 72	Eigene Darstellung
Abbildung 73	Eigene Darstellung
Abbildung 74	Eigene Darstellung
Abbildung 75	Eigene Darstellung
Abbildung 76	Eigene Darstellung
Abbildung 77	Eigene Darstellung
Abbildung 78	Eigene Darstellung
Abbildung 79	Eigene Darstellung
Abbildung 80	Eigene Darstellung
Abbildung 81	Eigene Darstellung
Abbildung 82	Eigene Darstellung
Abbildung 83	Eigene Darstellung
Abbildung 84	Eigene Darstellung

VI. Tabellen Quellen :

Tabelle 1	Eigene Darstellung
Tabelle 2	Eigene Darstellung
Tabelle 3	Eigene Darstellung

VII. Anhang :

USB-Stick mit:

1. Abstract
2. Projektordner
3. PHP Skripten
4. Digitale schriftliche Arbeit
5. Screencast
6. SQL Skripte