Höhere Technische Bundeslehranstalt Wien 3, Rennweg
IT & Mechatronik
HTL Rennweg: Rennweg 89b
A-1030 Wien, Tel +43 1 24215-10, Fax DW 18

Diplomarbeit

WraithKnight

ausgeführt an der
Höheren Abteilung für
Informationstechnologie/Medientechnologie
der Höheren Technischen Lehranstalt Wien 3 Rennweg

im Schuljahr 2018/19

durch
Jean Awad
Fillip Frackiewicz
Moritz Kusta

unter der Anleitung von

Mitra Bayandor
Franz Breunig

Wien, April 2019

Kurzfassung

Die Aufgabe des Teams war es, ein Spiel im Pixelstil zu entwickeln Im Spiel geht es um einen bereits verstorbenen Ritter, der wiederaufersteht und versucht seine verlorene Ehre wiederzuerlangen. Dabei muss der Spieler mehrere Gegner bezwingen. Das Spiel spielt sich im zweidimensionalen Raum ab und die Kameraperspektive ist von oben nach unten gerichtet. Um dem Spieler ein besonderes Spielerlebnis bieten zu können wurde eine passende Story zu dem Spiel ausgearbeitet.

Jedes Level sollte zufällig generiert werden um das Spiel attraktiver zu machen. Mit zufälligen Levels wird der Spieler jedes Mal neu gefordert und das Auswendiglernen ist nicht möglich. Ein Level ist bestanden, wenn alle Gegner eliminiert wurden und der Charakter den Kampf überlebt hat. Wenn das Level erfolgreich absolviert wurde, wird das nächste zufällig generierte Level um eine Schwierigkeitsstufe schwerer. Falls man bei einem Versuch alle Gegner zu besiegen mit dem Hauptcharakter stirbt, beginnt man bei wieder null und muss alle bereits absolvierten Schwierigkeitsstufen neu machen.

Es sollte ein Projekt gefunden werden, dass einen großen Teil des Wissens der fünfjährigen HTL beinhaltet, um das Erlernte in praktische Arbeit umzusetzen. Um das zu erreichen wurde das Spiel von null an entwickelt. Der Code hinter dem Spiel wurde mithilfe von C# geschrieben. Dabei wurde das Monogame-Framework verwendet. Um weitere medientechnische Aspekte mit in das Projekt einfließen zu lassen wurden alle Grafiken, Animationen und Sounds, die für das Spiel benötigt wurden, ebenfalls selber erstellt.

Abstract

The task of the team was to develop a game in the pixel style. The game is about a deceased knight who is resurrected and is trying to regain his lost honor. The player must defeat several opponents. The game takes place in two-dimensional space and the camera perspective is the top-down perspective. In order to offer the player a special gaming experience, a matching story about the game has been worked out.

Each level should be randomly generated to make the game more attractive. With random levels, players are challenged each time and memorization is not possible. A level is passed when all enemies have been eliminated and the character has survived the fight. When the level has been successfully completed, the next randomly generated level becomes heavier by one difficulty level. If you die in an attempt to defeat all opponents with the main character, you start from zero again and have to redo all difficulty levels already completed.

A project should be found that incorporates a large part of the knowledge of the five-year HTL in order to translate what has been learned into practical work. To achieve that, the game was developed from scratch. The code behind the game was written using C #. The monogame framework was used. In order to incorporate further media technology aspects into the project, all the graphics, animations and sounds needed for the game were also created by the artist.

Ehrenwörtliche Erklärung

Ich versichere,

- dass ich meinen Anteil an dieser Diplomarbeit selbstständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe
- und mich auch sonst keiner unerlaubten Hilfe bzw. Hilfsmittel bedient habe.

Wien, am,

Moritz Kusta

Fillip Frackiewicz

Jean Awad

Präambel

Die Inhalte dieser Diplomarbeit entsprechen den Qualitätsnormen für „Ingenieurprojekte“ gemäß § 29 der Verordnung des Bundesministers für Unterricht und kulturelle Angelegenheiten über die Reife- und Diplomprüfung in den berufsbildenden höheren Schulen, BGBl. Nr. 847/1992, in der Fassung der Verordnungen BGBl. Nr. 269/1993, Nr. 467/1996 und BGBl. II Nr. 123/97.

Liste der betreuenden Lehrer

Prof, DI, Mitra Bayandor Hauptbetreuer
Prof, DI, Franz Breunig Hauptbetreuer Stv.

Liste der Kooperationspartner:

Inhaltsverzeichnis

1 EINLEITUNG 9

1.1 ENTSTEHUNG DER IDEE 9
1.2 WAS IST WRAITHKNIGHT? 9
1.3 DAS TEAM 10
1.3.1 MORITZ KUSTA 10
1.3.2 FILLIP FRACKIEWICZ 10
1.3.3 JEAN AWAD 11

2 PROJEKTINDIZIERUNG 12

2.1 PROJEKTZIELE 12
2.1.1 HAUPTZIELE 12
2.1.2 OPTIONALE ZIELE 14
2.1.3 NICHT ZIELE 15
2.2 UMFELDER 16
2.2.1 GRAFISCHE DARSTELLUNG 16
2.2.2 BESCHREIBUNG DER WICHTIGSTEN UMFELDER 17
2.2.3 RISIKOANALYSE 18
2.2.4 RISIKOPORTFOLIO 19
2.2.5 GEGENMAßNAHMEN 20
2.2.6 CONTROLLING KONZEPT 21
2.3 PROJEKTUMSETZUNG 21
2.3.1 MANAGEMENT METHODE SCRUM 21
2.3.2 PRODUCT BACKLOG 22
2.3.3 SPRINTS 22
2.3.4 KOMMUNIKATIONSMITTEL 22
2.3.5 MEETINGS 23
2.3.6 BERICHTE 24
2.4 ABSCHLUSS 25
2.4.1 ABNAHME 25
2.4.2 LESSONS LEARNED 25
2.5 PROJEKTMANAGEMENTTOOLS 25
2.5.1 TAIGA.IO 25
2.5.2 TOGGL 25

3 ÖFFENTLICHKEITSARBEIT 26

3.1 FACEBOOK 26
3.1.1 CONTENT DER FACEBOOK-SEITE 26
3.2 TAG DER OFFENEN TÜRE 26
3.2.1 FLYER 27
3.2.2 PLAKAT 27
3.2.3 STANDORT AM TAG DER OFFENEN TÜRE 27

4 GRAFIKSOFTWARE 28

4.1 ASEPRITE 28
4.1.1 WAS IST ASEPRITE? 28
4.1.2 BENUTZEROBERFLÄCHE 30
4.1.3 ASEPRITES FEATURES 31
4.2 ADOBE PHOTOSHOP 31
4.2.1 WAS IST ADOBE PHOTOSHOP? 31
4.3 ASEPRITE VERSUS PHOTOSHOP 32
4.3.1 ENTSCHEIDUNG FÜR ASEPRITE 32

5 GRAFIKEN 35

5.1 PIXELART ALS STILMITTEL 35
5.1.1 WIESO IN PIXEL ART? 35
5.2 FARBEN 36
5.2.1 FARBTHEORIE 36
5.2.2 FARBMODELLE 37
5.2.3 FARBPALETTE 38
5.3 ANIMATIONEN 39
5.3.1 PROBLEME 39
5.3.2 ABLAUF DER ERSTELLUNG 41
5.4 ELEMENTE 42
5.4.1 AKTOREN/FIGUREN/CHARAKTERE 42
5.4.2 PROJEKTILE 45
5.5 KULISSE 46
5.5.1 BÄUME 46
5.5.2 PFLANZEN 47

6 WEBSEITE 48

6.1 STRUKTUR 48
6.1.1 NAVIGATION 48
6.1.2 HERO 48
6.1.3 TEAM 48
6.1.4 PRODUKTBESCHREIBUNG 49
6.1.5 CHARAKTERBESCHREIBUNG 49
6.2 DESIGN 49
6.2.1 ZWECK 49
6.2.2 FARBEN 49
6.2.3 LOGO 50

7 SOFTWARE 51

7.1 MONOGAME FRAMEWORK 51
7.1.1 UPDATELOOP 54
7.1.2 DELTATIME 54
7.2 ARCHITEKTUR 55
7.2.1 ECS 56

7.2.2 Gebundene Komponente ... 58
7.3 Assetmanagement ... 59
7.3.1 Monogame Content-Manager ... 59
7.3.2 Dynamische Asset-Bibliothek ... 60
7.4 Entity-Erstellung ... 60
7.5 Input ... 62
7.5.1 Keyboard ... 62
7.5.2 Maus ... 62
7.6 Grafiken in Monogame ... 63
7.6.1 Simple Grafiken ... 63
7.6.2 Komplexe Grafiken ... 63
7.6.3 Simple Animationen ... 63
7.6.4 Komplexe Animationen ... 64
7.7 Bewegung ... 64
7.7.1 Vektoren ... 64
7.7.2 Beschleunigung und Abbremsung ... 65
7.8 Kollision ... 66
7.8.1 AABB ... 66
7.8.2 Minkowski-Summe ... 67
7.8.3 Collision Response ... 68
7.9 Levelgeneration ... 70
7.9.1 Cellular Automata ... 70
7.9.2 Finaler Algorithmus ... 70
7.10 Künstliche Intelligenz ... 73
7.10.1 Generelle Intelligenz ... 73
7.11 Gamescreens ... 74
7.12 Performance ... 74
7.12.1 CPU – WallCollision ... 74
7.12.2 GPU – SpriteBatching ... 75
7.12.3 Memory – Referenz Garbage Collector (GC) ... 75

1 Einleitung

1.1 Entstehung der Idee

Die Idee des Projektes entstand schon sehr früh in der vierten Klasse. Durch Umwege und Umstände fand sich das Team jedoch erst am Anfang des fünften Schuljahres. Jedoch ist das Interesse, ein Spiel zu entwickeln, schon lange vorhanden. Der erste Gedanke des Spieles wurde von mehreren anderen Spielen übernommen, jedoch sollte ein anderes Spielgefühl und eine andere Story übermittelt werden. Um ein besonderes Spielgefühl zu erreichen, hat das Team das Spiel von null auf selber entwickelt. Das heißt der ganze Code, alle Grafiken, alle Animationen und der Ton sollen extra für und an das Spiel angepasst werden. Die Grafiken und zugehörige Animationen sollten in einem Pixel-Art-Stil gezeichnet werden, da der Trend in letzter Zeit mehr zu Pixelspielen zeigt. Ein weiterer essentieller Punkt in der Ideenfindung war, dass das Spiel zufällig generierte Levels haben soll, damit das Spiel nach einer Zeit nicht langweilig wird.

1.2 Was ist WraithKnight?

WraithKnight ist ein Spiel, das im zweidimensionalem Raum stattfindet.
Im Spiel steuert man einen bereits gestorbenen, aber wiederauferstandenen Ritter, deswegen der Name „WraithKnight“ (in Deutsch „Geisterritter)“. Man steuert den Hauptcharakter in einem Wald voll mit Gegnern und Hindernissen und versucht seine verlorene Ehre wiederzuerlangen. Dabei muss man an vielen verschiedenen Gegnern vorbei und sie besiegen.
Das Spiel ist ein sogenanntes „Roguelike”. Bei einem „Roguelike“ wird jedes Level zufällig neu generiert. Wenn das Level erfolgreich absolviert ist, wird das nächste Level neu generiert und zusätzlich um eine Stufe schwieriger. Wenn das Level nicht bestanden wird, beginnt man beim leichtestem Level wieder und man muss sich wieder nach oben arbeiten. Durch die zufällige Levelgeneration soll es zu mehr Abwechslung beim Spielen. Ein auswendig lernen von den einzelnen Levels ist nicht möglich und somit wird das Spielen attraktiver und abwechslungsreicher. Beim Spielen wird man das Geschehen aus der „Vogelperspektive beobachten können. Sprich die Ansicht ist von oben nach unten gerichtet. Mit dieser Ansicht kann das Umfeld am besten betrachtet werden und der Spieler hat eine große Übersicht.

1.3 Das Team

1.3.1 Moritz Kusta

Moritz Kusta ist Projektleiter und nimmt somit auch die Rolle des Scrum-Masters an. Die Aufgabenbereiche des Projektleiters sind abgesehen vom Projektmanagment, zeichnen der Levelgrafiken, erstellen der Sounds und Sozial Media. Um das Projekt zeitgemäß fertigstellen zu können ist es wichtig einen Scrum-Master zu haben, der die Verantwortung für den Scrum-Prozess und dessen korrekte Implementation übernimmt und für die Aktualität aller Scrum-Artefakte wie Product-Backlog und Sprint-Backlog sorgt. Ebenfalls sorgt er für den Informationsfluss zwischen Product Owner und dem Team und leitet alle Meetings zwischen den Mitarbeitern und zwischen Auftraggeber und Team.

Individuelle Aufgabenstellung von Moritz Kusta:

Projektleiter / Grafiker	Leitung des Projekts als Scrum Master und Erstellung der Pixelgrafiken
Aufgabenstellung Auflistung der einzelnen Ziele und Anforderungen	• **Ziel-H9** Graphisches Leveldesign • **Ziel-H10** Graphisches Gegnerdesign • **Ziel-H14** Dokumentation durch Management + Controlling • **Ziel-H15** Diplomarbeitsbuch • **Ziel-H16** Sprachunterstützung • **Ziel-H18** Animierter Hauptcharakter • **Ziel-H20** Social-Media-Kanäle

1.3.2 Fillip Frackiewicz

Fillip Frackiewicz ist der Product Owner des Projektes und ist für die Programmierung des Spieles zuständig. Da Fillip Frackiewicz die Idee des Spieles hatte und diese verwirklichen wollte ist es ideal, dass die Rolle des Product Owners von Ihm übernommen wird. Um die Idee bestmöglich zu verwirklichen übernimmt er auch die Zuständigkeit für die Programmierung des Spieles. Da das Projektmitglied bereits eine Vorkenntnis mit der Programmierung von Spielen hat wurden die Ziele realistisch aber erreichbar gesteckt. Unter anderem gehört dazu die Entwicklung einer Spielengine, das Programmieren einer Zufällige Levelgeneration und eine Auflösungsunabhängigkeit des Spieles.

Individuelle Aufgabenstellung von Fillip Frackiewicz:

Programmierer	Programmierung des Spieles
Aufgabenstellung Auflistung der einzelnen Ziele und Anforderungen	• **Ziel-H1** Funktionsfähiges Hauptmenü • **Ziel-H2** Optionsmenü • **Ziel-H3** Auflösungsunabhängig • **Ziel-H4** Tastatursupport • **Ziel-H5** Maussupport • **Ziel-H6** Soundtrack • **Ziel-H7** Soundeffekte • **Ziel-H8** Zufällige Levelgeneration • **Ziel-H11** implementierte Charakter-Controls • **Ziel-H12** Optimierung auf die PC-Plattform • **Ziel-H13** Schwierigkeitsgrad • **Ziel-H15** Diplomarbeitsbuch • **Ziel-H16** Sprachunterstützung • **Ziel-H18** Animierter Hauptcharakter

1.3.3 Jean Awad

Jean Awad ist Projektmitarbeiter und ist zuständig für das Zeichnen der Charaktere des Spieles und für einen Webauftritt des Projektes. Um möglichen Interessenten eine Möglichkeit zu bieten sich über das Produkt zu informieren ist die Hauptaufgabe des Mitarbeiters eine Webseite zu programmieren. Ein weiterer essentieller Punkt ist es ansprechende Grafiken für den Hauptcharakter und die dazugehörigen Gegner zu haben. Zugehörig zu den statischen Grafiken muss für ein Spiel jeder bewegliche Charakter animiert werden. Animationen wie das Laufen, die Attacken oder der Tod müssen per Hand modelliert und gezeichnet werden.

Individuelle Aufgabenstellung Jean Awad:

Webdeveloper	Erstellung, Administration und Testen der Webseite
Aufgabenstellung Auflistung der einzelnen Ziele und Anforderungen	• **Ziel-H6** Soundtrack • **Ziel-H7** Soundeffekte • **Ziel-H9** Graphisches Leveldesign • **Ziel-H10** Graphisches Gegnerdesign • **Ziel-H15** Diplomarbeitsbuch • **Ziel-H16** Sprachunterstützung • **Ziel-H18** Animierter Hauptcharakter • **Ziel-H19** Webseite

2 Projektindizierung

2.1 Projektziele

2.1.1 Hauptziele

Ziel-H1 Funktionsfähiges Hauptmenü

Das Spiel wird ein Hauptmenu haben, mit Buttons zum Navigieren des Interfaces

Ziel-H2 Optionsmenu

Der Spieler kann Einstellungen verändern, wenn er es möchte. Einstellungen wie Soundeinstellungen und Grafikeinstellungen z.B. Anzahl der Partikel, Auflösung und V-Sync

Ziel-H3 Auflösungsunabhängig

Das Spiel muss unabhängig von der Bildschirmsauflösung des Gerätes sein. Resolution-Independent-Rendering oder Perspektiven-Matrizen bieten sich hier an.

Ziel-H4 Tastatursupport

Die Tastatur wird zur Spielsteuerung verwendet und muss daher unterstützt werden.

Ziel-H5 Maussupport

Die Maus wird zur Spielsteuerung verwendet und muss daher unterstützt werden.

Ziel-H6 Soundtrack

Eine zum Spiel passende, selbstkomponierte Hintergrundmusik wird das Spiel die ganze Zeit begleiten.

Ziel-H7 Soundeffekte

Die Spieleraktionen müssen mit Soundeffekten versehen werden, damit Feedback auf den Input besteht und das „Gamefeel“ besser ist.

Ziel-H8 Zufällige Levelgeneration

Die Levels werden zufällig und automatisch generiert, damit der Wiederspielwert besteht.

Ein „Level“ ist die Kombination von Gegnern, Kulissenpositionierung und Hindernissen.

Ziel-H9 Graphisches Leveldesign

Die Levels werden graphisch gestaltet und mit Kulissenobjekten versehen.

Ziel-H10 Graphisches Gegnerdesign

Die Gegner müssen ein eindeutiges graphisches Design haben und auch vollkommend animiert sein. Es gibt mehrere Gegnerdesigns.

Ziel-H11 Implementierte Charakter-Controls

Das Spiel muss auf Spielerinputs reagieren und den Spielcharakter je nach Input korrekt steuern.

Ziel-H12 Optimierung auf die PC-Plattform

Das Spiel wird für PC-Geräte optimiert und als lokale Applikation laufen.

Ziel-H13 Schwierigkeitsgrad

Von dem Ersten bis zum letzten Level wird der Schwierigkeitsgrad immer erhöht und somit die Gegner besser.

Ziel-H14 Dokumentation durch Management

Die Umsetzung des Projektes wird vom Projektmanager und Mitarbeiter protokolliert und dokumentiert, um die Nachvollziehbarkeit der erledigten Aufgaben und Aufwand zu gewährleisten.

Ziel-H15 Diplomarbeitsbuch

Das Projekt wird als Diplomarbeitsbuch dokumentiert und der Öffentlichkeit zur Verfügung gestellt.

Ziel-H16 Sprachunterstützung

Das Produkt wird in der englischen Sprache zur Verfügung stehen.

Ziel-H17 Grafisch gestaltetes Hauptmenü

Das Spiel wird ein Hauptmenu haben, das graphisch passend zum Spiel entworfen ist und verwendet werden kann.

Ziel-H18 Animierter Hauptcharakter

Das Spiel wird einen Hauptcharakter haben, der mit unseren Grafiken animiert ist.

Ziel-H19 Funktionale Webseite

Zu dem Spiel wird es eine passende Webseite geben, diese wird in einheitlichem Design passend zu dem Spiel gestaltet. Auf der Webseite können wir Interessenten über das Spiel, den Status des Spieles und das Team hinter dem Spiel vorstellen.

Ziel-H20 Social-Media-Kanäle

Um aktuelle Infos und Fortschritte so schnell wie möglich publizieren zu können, werden wir Social-Media-Accounts eröffnen und während des Projektes führen. Da wir gleichzeitig eine breite Masse an Menschen ansprechen wollen, haben wir uns für Sozial Media Kanäle entschieden.

2.1.2 Optionale Ziele

Ziel-O1 Skilltree

Es wird ein Skilltree (Fähigkeitenbaum) entworfen. Es werden mindestens 3 Fähigkeiten eingebaut. Unter einem Skilltree versteht man einen Fähigkeitenbaum, mit dem Spieler ihre Fähigkeiten-Wahl individualisieren können. Die Fähigkeiten im Baum selbst stehen fest und können nicht vom Spieler geändert werden.

Ziel-O2 Controllersupport

Controllerinputs, wie vom Xbox- oder PlayStation-Controller, können verwendet werden.

Ziel-03 Kontaktaufnahme mit Industrie

Professionelle Firmen und Einzelunternehmen der Spieleindustrie werden kontaktiert um mögliche Kooperationen abzuschließen.

Ziel-O4 Unterstützung von mehreren Sprachen

Neben Englisch können andere Sprachen für die Dialoge und für das Menü gewählt werden.

Ziel-O5 Kontaktaufnahme mit Publishern

Publisher werden kontaktiert. Der Kontakt wird stätig gepflegt, um nach der Diplomarbeit das Spiel professionell zu vermarkten.

Ziel-O6 Vereinbarung mit Publishern

Eine vorzeitige Vereinbarung führt zu einem Zugriff auf externe Ressourcen und Quellen.

2.1.3 Nicht Ziele

Ziel-N1 **Es ist ein vollständiges Spiel**

Das Spiel ist in einer finalen Version und mit weitern nicht in diesem Antrag enthaltenen Features versehen.

Ziel-N2 **Es wird veröffentlicht**

Das Spiel wird auf einer Spiele-Plattform veröffentlicht.

Ziel-N3 **Es wird eine Game Engine verwendet.**

Das Spiel verwendet eine Game Engine (Unity, Unreal).

Ziel-N4 **Es wird auf nicht-PC Plattformen optimiert**

Das Spiel wird für andere Plattformen als den PC optimiert oder bereitgestellt.

2.2 Umfelder

Die Umfeldanalyse dient zur Ermittlung der Einflussfaktoren in einem Projekt und deren zukünftige Entwicklung. Sie soll Chancen und Risiken für die Entwicklung des Projektes aufweisen und hervorheben. In der folgenden grafischen Darstellung werden drei Bereiche Analysiert: Technik, Extern und Intern. [1]

2.2.1 Grafische Darstellung

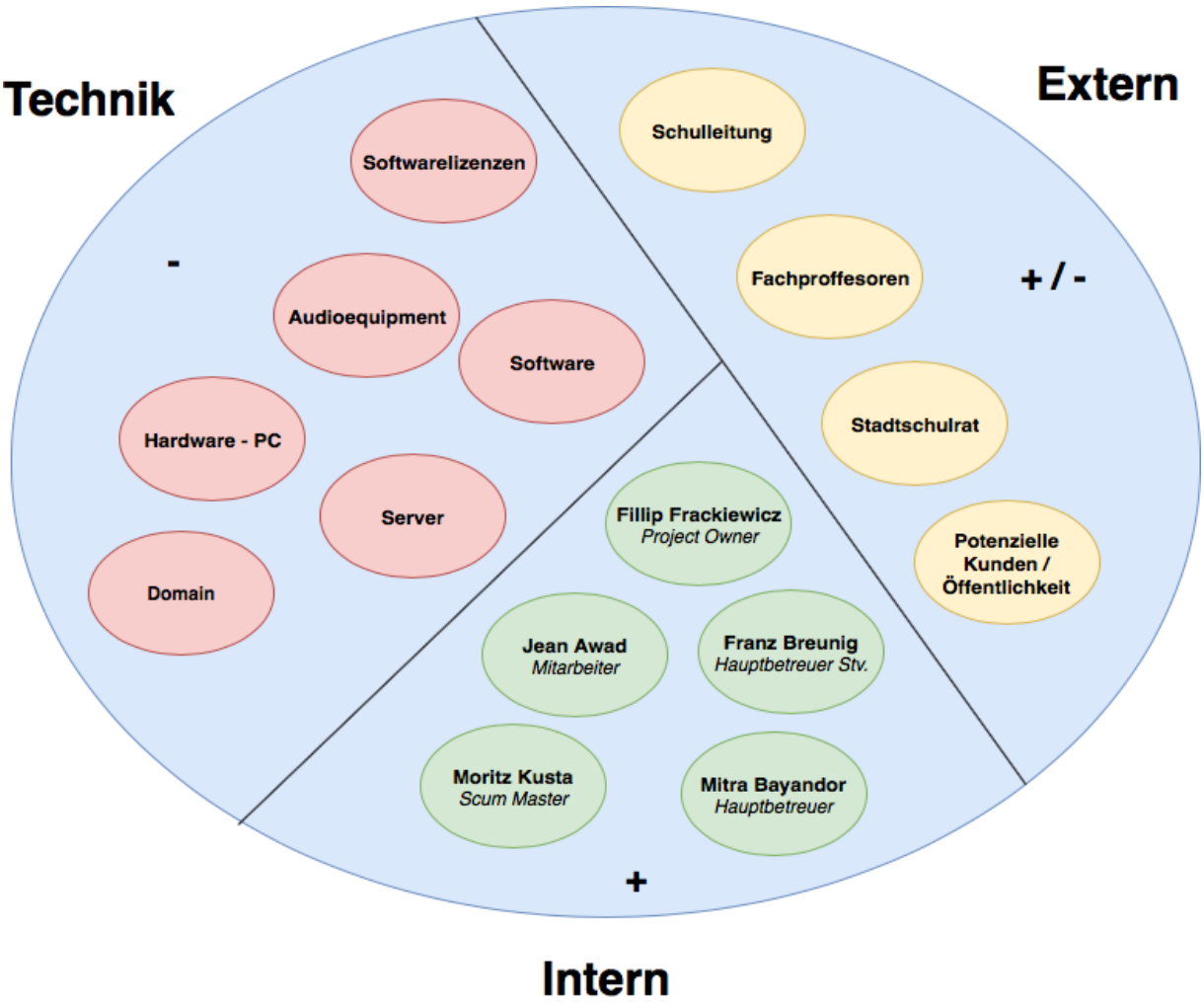

Abbildung 1 Grafische Darstellung der Umfelder

[1] https://www.controllingportal.de/Fachinfo/Grundlagen/Umfeldanalyse.html

2.2.2 Beschreibung der wichtigsten Umfelder

#	Bezeichnung	Beschreibung	Bewertung
1	Moritz Kusta	Scrum Master managet das ganze Projekt und behält alles im Überblick	+
2	Fillip Frackiewicz	Programmierer hat Verantwortung für das Spiel	+
3	Jean Awad	Verantwortlich für die Webseite	+
4	Mitra Bayandor	Auftraggeber steht uns zur Verfügung um technische Fragen zu stellen	+
5	Franz Breunig	Betreuer hilft uns bei der technischen Ausführung des Projektes	+
6	Schulleitung	Muss das Projekt absegnen und erlauben, kann uns jedoch auch Unterstützen	+ / -
7	Fachprofessoren	Können uns technisch helfen, haben jedoch nicht immer Zeit	+ / -
8	Potenzielle Kunden / Öffentlichkeit	Können uns nach einem Test Hinweise zu Fehlern geben, können aber auch unzufrieden sein	+ / -
9	Stadtschulrat	Muss das Projekt erlauben, kann uns jedoch auch Unterstützen und mit Sponsoren verbinden	+ / -
10	Domain	Die Domain könnte schon verwendet sein oder teurer werden	-
11	Server	Der Server könnte abstürzen und die Webseite nicht verfügbar sein	-
12	Hardware - PC	Die PCs könnten ein technisches gebrechen haben und nicht verwendbar sein	-
13	Software	Die Software könnte auf unserem Betriebssystem nicht funktionieren	-
14	Audioequipment	Das Equipment könnte kaputtgehen oder die Treiber nicht auf unserem PC laufen	-
15	Softwarelizenzen	Softwarelizenzen könnten während dem Projekt ablaufen	-

Tabelle 1 Umfeldanalyse - Beschreibung der wichtigsten Umfelder

2.2.3 Risikoanalyse

#	Bezeichnung	Beschreibung des Risikos	P	A	RF
7	Fachprofessoren	Haben nicht immer Zeit	40	40	800
6	Schulleitung	Muss das Projekt absegnen und erlauben	10	75	750
11	Server	Der Server könnte abstürzen und die Webseite nicht verfügbar sein	15	50	750
8	Potenzielle Kunden / Öffentlichkeit	Können mit dem Produkt unzufrieden sein	30	20	600
9	Stadtschulrat	Muss das Projekt erlauben	5	100	500
14	Audioequipment	Das Equipment könnte kaputtgehen oder Treiber nicht auf unserem PC laufen	5	55	275
10	Domain	Die Domain könnte schon verwendet sein oder teurer werden	5	50	250
12	Hardware - PC	Die PC's könnten ein technisches gebrechen haben und nicht verwendbar sein	5	45	225
13	Software	Die Software könnte auf unserem Betriebssystem nicht funktionieren	5	40	200
15	Softwarelizenzen	Softwarelizenzen könnten während dem Projekt ablaufen	25	5	125

Tabelle 2 Umfeldanalyse - Beschreibung der Risiken

P….Eintrittswahrscheinlichkeit des Risikos

A….Schadensausmaß bei Eintritt des Risikos

RF….berechneter Risikofaktor

2.2.4 Risikoportfolio

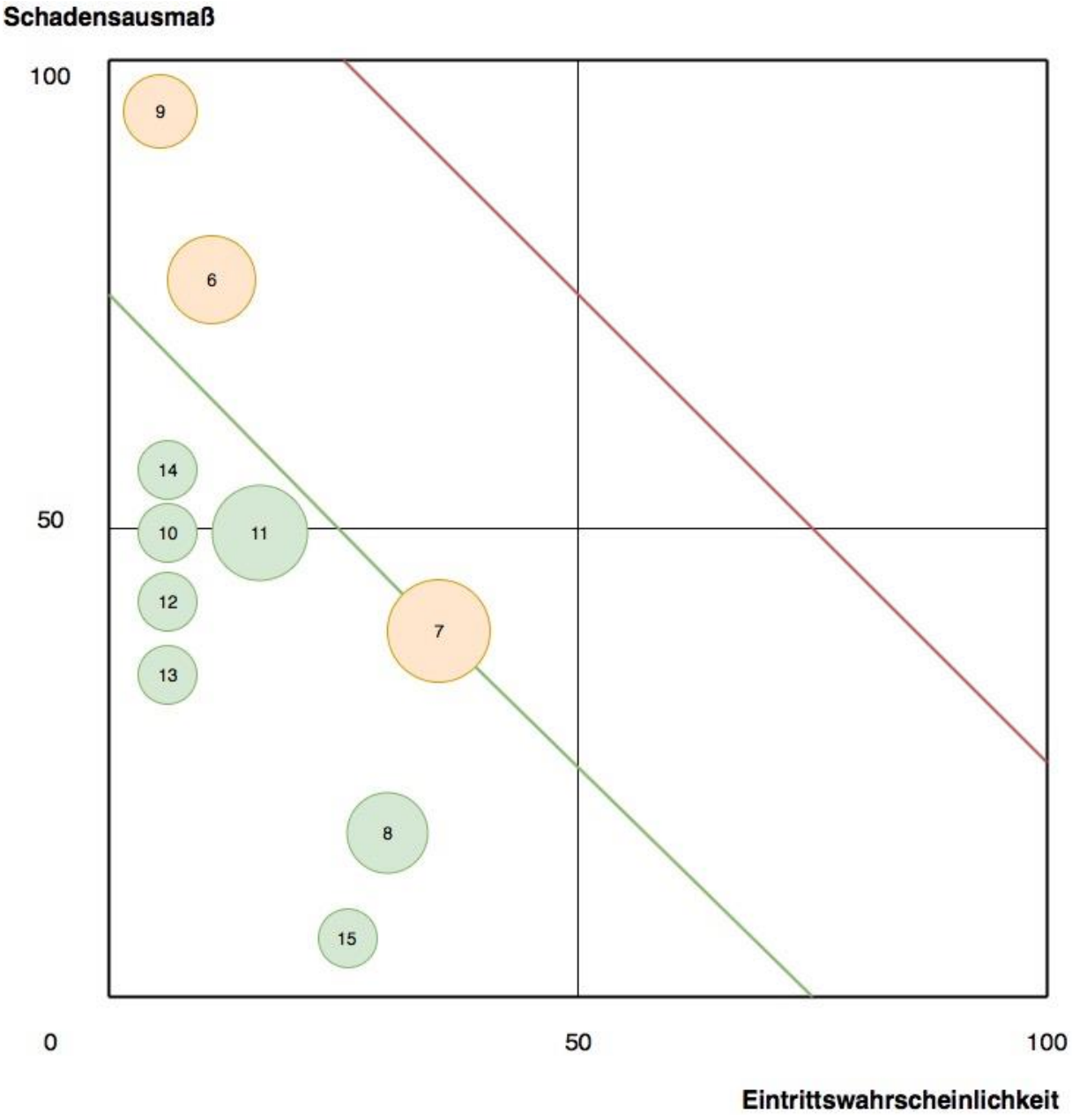

Abbildung 2 Risikoportfolio

2.2.5 Gegenmaßnahmen

#	Bezeichnung	Gegenmaßnahme
6	Schulleitung	Alle Termine einhalten und regelmäßige Updates geben
7	Fachprofessoren	Vor einem Treffen eine kurze Absprache mit dem Lehrer halten
8	Potenzielle Kunden	Über den Status des Projektes Informieren und um Feedback bitten
9	Stadtschulrat	Termine einhalten und Antrag verständlich Formulieren
10	Domain	Rechtzeitig kaufen – andere Domain wählen
11	Server	Anderen Betreiber wählen, wenn dieser Unzuverlässig ist
12	Hardware - PC	Hardware überprüfen – im Notfall auswechseln
13	Software	Software rechtzeitig ausprobieren
14	Audioequipment	Audioequipment überprüfen und vorsichtig behandeln
15	Softwarelizenzen	Lizenzen rechtzeitig updaten

Tabelle 3 Risikoanalyse - Gegenmaßnahmen

2.2.6 Controlling Konzept

Unter Projektcontrolling fallen alle Tätigkeiten, die Notwendig sind um ein Projekt über die ganze Laufzeit zu steuern und somit auch im grünen Bereich zu halten. Um Controlling erfolgreich durführen zu können gibt es einige Voraussetzungen. Für das Controlling Konzept des Projektes wurden folgende Aspekte in den Vordergrund gestellt.

- Transparente Zielplanung mit Zielen nach SMART-Kriterien
- Ehrlichkeit und Offenheit
- Fehler sind erlaubt

Ohne diese Voraussetzungen ist es einem Projektleiter unmöglich ein Projekt im grünen Bereich zu halten. Deswegen wurde in dem Projekt besonders viel Wert auf diese drei Aspekte gelegt. Schon am Anfang des Projektes wurden Ziele nach den SMART-Kriterien definiert umso ein erfolgreiches Projekt zu starten. Ehrlichkeit und Offenheit wurde bei Besprechungen oder anderen Kommunikationswegen immer beachtet, da so Gefahren für das Projekt erkannt und beseitigt werden konnten. Der letzte Aspekt des Controlling Konzepts ist die Toleranz bei Fehlern. Da alle Teammitglieder kaum technische Erfahrung in einem Projekt haben ist es klar, dass Fehler passieren. Diese werden toleriert und mit dem zweiten Aspekt, Ehrlichkeit und Offenheit, als Team ausgebessert.[2]

2.3 Projektumsetzung

2.3.1 Management Methode Scrum

Da Flexibilität ist bei der Entwicklung eines Spieles sehr wichtig ist, da es immer wieder während des Prozesses zu neuen Ideen oder Verbesserung kommen kann. Deswegen wurde die Projektmethode Scrum ausgewählt. Bei dieser Projektentwicklungsmethode wird Flexibilität und hohe Effektivität in den Vordergrund gestellt. In der folgenden Auflistung werden Vorteile von Scrum aufgezeigt:

- Hohe Transparenz um rechtzeitig entgegensteuern zu können
- Hohe Flexibilität und Agilität
- Starke Eigenverantwortung im Team
- Wenige Regeln, daher leicht verständlich

[2] https://www.projektmanagementhandbuch.de/handbuch/projektrealisierung/projektcontrolling/
https://www.online-projektmanagement.info/agiles-projektmanagement-scrum-methode/tools-und-tipps-fuer-scrum/vor-und-nachteile/

2.3.2 Product Backlog

Der Product Backlog besteht aus allen Anforderungen des Produktes. Um die Qualität des Produktes so hoch wie möglich zu halten ist die Qualität des Backlogs essenziell, daher sollte bei der Planung so genau wie möglich gearbeitet werden um keinen Aspekt des Produktes zu vergessen. Der Backlog wird während der ganzen Laufzeit des Projektes gepflegt und wenn es benötigt wird erweitert. Die Anforderungen werden nacheinander bearbeitet und in die verschiedenen Sprints eingeplant. [3]

2.3.3 Sprints

Scrum Projekte werden inkrementell entwickelt und dabei in Zeitboxen eingeteilt. Anschließend wird jedem Sprint ein Name gegeben, Tasks abgeschätzt und zugeteilt. Die Sprintdauer beträgt 3 Wochen. Vor jedem Sprint gibt es ein Sprint Planning Meeting wo die oben angegebenen Punkte besprochen werden. Nach jedem Sprint gibt es ein Sprint Review Meeting, wo der Projektleiter alle Tasks auf Vollständigkeit überprüft und Schwierigkeiten/Probleme im Team behoben/besprochen werden. Zu den beiden Meetings wird ein zugehöriges Meeting-Protokoll angefertigt.[4]

2.3.4 Kommunikationsmittel

Da die Kommunikation im Team reibungslos funktionieren soll sind verschiedene Mittel für die Kommunikation wichtig. Um für alle Dateien und Informationen den richtigen Kommunikationsweg finden zu können wurden einige Applikationen dafür ausgewählt und während des Projektes sehr häufig und intensiv genutzt.

2.3.4.1 E-Mail

Für die schriftliche formelle Kommunikation mit den Betreuern und dem Team wurden E-Mails als Kommunikationsmittel ausgewählt, da es sich sowohl für lange und kurze Nachrichten eignet, alle Mitglieder per Mail erreichbar sind und E-Mails ein renommierter Kommunikationsweg im Arbeitsalltag darstellt. Außerdem werden Protokolle von Meetings an die Betreuer mittels E-Mail weitergegeben. Bei E-Mails wird großen Wert auf eine Begrüßung und eine Verabschiedung gelegt, deswegen wurden ebenfalls Footer für den externen E-Mail-Verkehr angefertigt.

[3] https://www.inloox.de/unternehmen/blog/artikel/scrum-grundlagen-einfach-erklaert-der-product-backlog/

[4] https://scrum-master.de/Scrum-Meetings/Sprint

2.3.4.2 WhatsApp

Für die schriftliche interne Kommunikation im Team wurde WhatsApp gewählt, da diese Applikation von allen Mitgliedern verwendet wird und alle Mitarbeiter am schnellsten über WhatsApp erreicht werden können. Es wurde eine Gruppe gegründet, wo alle Teammitglieder Information austauschen konnten.
Ebenfalls wurde WhatsApp für die kurzfristige Kommunikation mit der Hauptbetreuerin genutzt, falls ein Problem kurzfristig behoben werden musste.

2.3.4.3 Discord

Discord ist ein Online-Chatraum, der es dem Team ermöglicht miteinander, von Zuhause aus und eng beisammen zu arbeiten. Bei Discord können Räume eröffnet werden, wo man mit den Teammitgliedern wie bei einem Sprachanruf untereinander kommunizieren kann. Ebenfalls kann der Bildschirm eines Users übertragen werden, um vor allem bei den Grafikarbeiten das Produkt von anderen Teammitgliedern anschauen lassen zu können.

2.3.4.4 Persönlich

Das wahrscheinlich wichtigste Kommunikationsmittel in einem Projekt ist die persönliche Kommunikation zwischen den Teammitarbeitern. Egal ob bei Meetings oder beim Zusammentreffen um zu arbeiten. Um die Stimmung untereinander positiv zu halten wurde großen Wert auf einen respektvollen Umgang gelegt. Deswegen wurden Meinungsverschiedenheiten immer in einem persönlichen Gespräch besprochen und aus der Welt geschaffen.

2.3.5 Meetings

Während des Projektes gab es zwei verschiedene Arten von Meetings. Einerseits gab es interne Meetings zwischen den Teammitgliedern und andererseits gab es externe Meetings mit den Betreuern. Die externen Meetings haben alle drei Wochen nach dem Sprint stattgefunden. Das Ziel dieser Meetings war es, den Sprint zu besprechen, die Ergebnisse zu sichten, Verbesserungsvorschläge zu bekommen und um ein Status-Update geben zu können. Die internen Meetings haben je nach Verlangen stattgefunden. Um das Produkt so gut wie möglich zu optimieren hat sich das Team öfters auch auf Online-Plattformen „getroffen“, um zusammen zu arbeiten oder um ein Thema zu besprechen. Um im Team besser von den anderen Teammitgliedern Bescheid zu wissen wurde zwei Mal wöchentlich ein Statusupdate mündlich vermittelt. Damit der Projektlenkungsausschuss auch um den Status des Projektes Bescheid weiß wurde Jede Woche ein mündliches Statusupdate an die Hauptbetreuerin weitergegeben.

2.3.6 Berichte

Um dem Projektlenkungsausschuss regelmäßig über den Status des Projektes zu informieren wurden mehrere Arten von Berichten an den Hauptbetreuer weitergegeben. Zu jeder Sprintplanung gab es ein „Sprint Planning Protokoll“, zu jedem Sprintende gab es ein „Sprint Review“ und zu jedem Meeting mit den Betreuern wurde ebenfalls ein Protokoll geschrieben.

2.3.6.1 Sprint Planning Protokoll

In diesem Dokument wurde das Planning Meeting protokolliert. Im Sprint Planning Meeting wurde als erstes eine grobe Übersicht über den derzeitigen Stand des Projektes gegeben und über aktuelle Termine und Deadlines informiert. Anschließend wurde die Dauer des Sprints definiert und das Team hat einen passenden Namen zu dem Sprint gefunden. Danach wurden die User Stories aus dem Product Backlog entnommen und Tasks daraus gebildet. Die User Stories und deren Story Points wurden währenddessen protokolliert.

2.3.6.2 Sprint Review

Um die Projektauftraggeberin nach jedem Sprint über den aktuellen Status des Projektes zu informieren, wurde ein Sprint Review angefertigt. In dem Protokoll wurden folgende Informationen weitergegeben:

- Projektstatus mithilfe einer Ampel (Grün → alles nach Plan, Orange → kleine Probleme, Rot → Weiterführung in Gefahr)
- Beschreibung der Teammotivation
- Probleme im Projekt und deren Lösungsstrategien
- Erledigte Arbeiten und anstehende Arbeiten

2.3.6.3 Sprint Retrospektive

Nachdem der Sprint abgeschlossen wurde, muss dafür gesorgt werden, dass es Optimierungen im nächsten Sprint gibt. Um diese zu erreichen gibt es eine Sprint Retrospektive. Dabei wird im Anschluss des Sprint Review Meetings (siehe 2.3.6.2) eine Diskussion gestartet, bei dem rückblickend über den vergangenen Sprint diskutiert wird. Es werden positive und negative Aspekte des Sprints herausgesucht und analysiert.

Probleme werden besprochen und gelöst umso im nächsten Sprint produktiver oder angenehmer arbeiten zu können. [5]

2.4 Projektmanagementtools

2.4.1 Taiga.io

Taiga ist ein freies, mehrfach prämiertes Open Source Projektmanagmenttool, welches uns in der Durchführung und in der Planung unterstützt hat. Das Tool kann online auf der Webseite taiga.io verwendet werden. Taiga wurde benutzt, um den Backlog und die Sprints zu verwalten. Um den Backlog in Taiga haben zu können muss dieser am Anfang des Projektes eingetragen werden. Anschließend sind alle User-Stories als Seitenelemente vorhanden und man kann diese einfach in einen neuen Sprint ziehen. Außerdem hat jedes Teammitglied Zugriff auf das Tool. Das heißt User-Stories können zugeteilt werden. Wenn man am Ende des Projektes Statistiken und andere Details des Projektes näher kennen lernen will, kann man alles wichtigen Daten exportieren und danach damit weiterarbeiten.

2.4.2 Toggl

Toggl ist ein freies Zeiterfassungstool mit einer besonders simplen Lösung, mit nur einem Mausklick kann man die Zeiterfassung starten und stoppen. Am Ende jeder Woche, Monat oder Sprint können anschließend einfach Reports und Statistiken exportiert werden.
In dem Projekt wurde Toggl je nach Fälligkeit benutzt. Wenn in dem Projekt viel weitergebracht wurde, wurde Toggl nicht benötigt, jedoch wurde im Falle eines Verzuges ausgemacht, dass Toggl daraufhin aktiviert wird, um sicherzustellen, dass effektiv an dem Projekt gearbeitet wird.[6]

[5] https://scrum-master.de/Scrum-Glossar/Sprint_Retrospective_Meeting
https://www.openpm.info/display/openPM/Sprint+Retrospective

[6] https://www.computerwoche.de/a/professionelle-zeiterfassung-in-der-cloud,3067627,3

3 Öffentlichkeitsarbeit

Bei Öffentlichkeitsarbeit auch „PR“ (Public Relations) genannt geht es primär darum, eine Beziehung zwischen Öffentlichkeit und einem Projekt zu schaffen. Durch diese Beziehung kann das Projektteam direkt auf Konsumenten eingehen und kann daraus einige Vorteile ziehen. Welche Vorteile hat Öffentlichkeitsarbeit?

- Steigerung der Bekanntheit
- Generierung von neuen Interessenten
- Schnelle und flexible Reaktion
- Geringe Kosten jedoch eine große Wirkung[7]

3.1 Facebook

Um potentielle Kunden und Fans über den aktuellen Stand und andere Neuigkeiten zu informieren, hat sich das Team dazu entschieden eine Facebook-Seite zu eröffnen und auf diesem, regelmäßig Content zu posten. Durch Facebook kann der Kontakt mit potentiellen Kunden einfach aufgebaut werden. Da Facebook eine Kommentar, Chat und Like Funktion hat kann der direkte Kontakt mit „Follower“ aufgebaut werden und auf die Interaktion der Menschen eingegangen werden.

3.1.1 Content der Facebook-Seite

Um unseren „Likern“ einen Ansprechenden Content bieten zu können, haben wir uns für mehrere Arten von Content entschieden. Einerseits wurden immer wieder Updates gegeben und andererseits wurden alle Charaktere vorgestellt und beschrieben.

3.2 Tag der offenen Türe

Um beim Tag der offenen Türe für möglichst viel Aufmerksamkeit zu generieren, haben wir uns dazu entschieden Flyer und ein Plakat zu drucken. Flyer und Plakate wurden dazu verwendet, einen ansprechenden und interessanten Stand aufzubauen. Die Grafiken wurden dabei alle in Photoshop entworfen.

[7] https://www.gruenderszene.de/lexikon/begriffe/public-relations-pr
https://www.aufgesang.de/pr/leistungen/pressearbeit/9-gruende-fuer-pr/

3.2.1 Flyer

3.2.1.1 Vorderseite

Auf der Vorderseite des Flyers ist ein Farbverlauf zu erkennen. Von oben nach unten ist der Verlauf von einem Dunkelblau bis zu einem Hellblau. Die Farben wurden aus unserer „Corporate Identity“ entnommen, um einen großen Wiedererkennungswert zu haben. Die Überschrift „Wraithknight“ ist mit der Schriftart *SLAYTANIC* geschrieben. Um dem Interessenten einen kleinen Geschmack auf das Spiel geben zu können wurde der Spruch „Kämpfe dich durch Mengen an Feinden um deine verlorene Ehre wiederzuerlangen. Als untoter Ritter wurde dir nur das Leben hinterlassen, dir wird alles abverlangt, während deines Kampfes zur Normalität.“ Auf den Flyer gedruckt. Unterhalb dieses Textes wurde noch auf unsere Webseite www.wraithknight.at verlinkt.

3.2.1.2 Rückseite

Auf der Rückseite wurde das Logo unseres Spieles auf einem dunkelblauen Hintergrund gedruckt. Die Farben wurden natürlich aus unserer „Corporate Identity“ entnommen. Um während des Betrachtens gleichzeitig neugierig zu werden hat sich das Team während dem Entwurf überlegt, dass man mit dem Logo Interesse weckt. Vor allem wenn der Flyer umgedreht auf einem Tisch liegt soll sich der Interessent denken, was wohl auf der Vorderseite steht, da das Design so mystisch ist.

3.2.2 Plakat

Um beim Tag der offenen Türe eine richtigen „Eyechatcher“ zu haben, hat sich das Team dazu entschieden ein A0 großes Plakat zu drucken. Das Plakat ist einmal horizontal in der Hälfte geteilt. Die obere Hälfte ist Dunkelblau und die Untere ist Hellblau. Die Farben wurden aus unserem „Corporate Identity“ entnommen. Um dem Interessenten unser Spiel näher zu bringen wurde der aktuelle Stand der Charaktere auf das untere Feld des Plakates gedruckt. Um den Namen des Spieles nicht zu vergessen wurde in der oberen Hälfte wieder „Wraithknight“ mit der Schriftart *SLAYTANIC gedruckt.*

4 Grafiksoftware

4.1 Aseprite

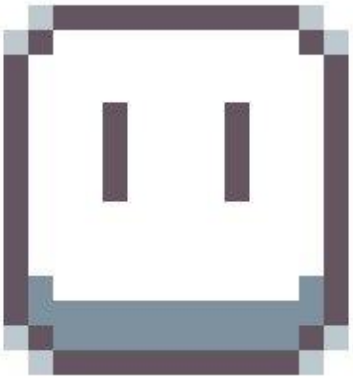

Abbildung 3 Aseprite Logo Quelle: https://pbs.twimg.com/profile_images/875731732389146624/-UznwnAx_400x400.jpg

Heutzutage gibt es etliche Action-Rollenspiele. Diese Spiele können sich nur unwesentliche durch ihre Inhalte voneinander abheben, aber in Hinblick auf Design und Grafiken kann es doch beachtliche Unterschiede geben. Um unserem Spiel etwas Besonderes zu verleihen, beschlossen wir, unsere Bilder in einem Pixelgrafik Programm zu erstellen.

4.1.1 Was ist Aseprite?

Aseprite ist ein kostenloses Pixelgrafikprogramm, das hauptsächlich für das Erstellen und Bearbeiten von Pixel-Grafiken und Animationen verwendet wird. Diese erstellten Grafiken werden auch in Action-Rollenspielen und aus Point-an-Click Abenteuerspiele verwendet. Zwei mehr verbreitete Spiele, wo Aseprite Grafiken benutzt werden, sind „OpenTTD“ und „Battle For Wesnoth“ [8]

Der Nutzer hat die Möglichkeit, den kompletten Vorgang, welcher Erstellung bzw. Modellierung und Animation von Pixelgrafiken beinhaltet, in einem Programm abzudecken

Mit Aseprite lässt sich sehr einfach 2D-Animationen für Spiele zu erstellen. Von Sprites über Pixel-Art bis hin zu Grafiken im Retro-Stil. Ein Sprite ist ein erstelltes Grafikobjekt. Der Retro Stil ist ein Stil aus historischer Vergangenheit.

Aseprite ist ein sehr stabiles Programm für Pixelgrafiken und hat eine anwenderfreundliche und triviale Benutzeroberfläche. Die Benutzeroberfläche ist in einem Pixel-Design dargestellt. Aseprite ist ein aktives und aktuelles Entwicklerprogramm,

[8] https://wiki.ubuntuusers.de/ASEPRITE/

welches eine Vielzahl von Werkzeugen, Ebenen und zweidimensionalen Animationen anbietet. Derzeit ist Aseprite nur in englischer Sprache verfügbar und das Programm ist frei und Open Source.

4.1.1.1 Bedienung

Die Bedienung von Aseprite ist genauso wie der von anderen Bildbearbeitungsprogrammen. Das Programm bietet ähnliche Werkzeuge, deren Einstellungen einzeln angepasst werden können. Hinter den meisten Werkzeugen auf der Werkzeugleiste sind weitere Werkzeuge verborgen. Drückt man auf das Symbol mit einem simplen Links-Klick, werden zusätzliche Werkzeuge sichtbar und auswählbar.

4.1.1.2 Animationszeitfenster

Eine hilfreiche Unterstützung sind die Ebenen (Layers), die Ebenen, werden in der Timeline, einfach mit den angegebenen Namen gekennzeichnet und angezeigt. Im Animations-Editor kann man Ebenen ausblenden, in der Reihenfolge (Hierarchie) neu anordnen oder für die Bearbeitung sperren, sodass bei späterer Arbeit nichts versehentlich verändert wird. Im Animations-Editor Bereich kann man Bildsequenzen und Ebenen gleichzeitig verwalten.

4.1.1.3 Bearbeitungsfenster

Wie auch bei 3D-Programmen, bietet Aseprite auch die Möglichkeit, das Bearbeitungsfenster horizontal und vertikal zu teilen, oder auch mehrfach, wenn nötig. Diese Einstellung ist ungewöhnlich für ein Bildbearbeitungsprogramm. Somit sind Bilddateien in verschiedenen Größen bearbeitbar, bei einer hohen Zoomstufe kann sehr auf Detailgetreue geachtet werden, Bereiche als Farbauswahl genutzt werden oder sogar verschiedene Bilddateien übereinander zu legen und zu überlappen oder auch nebeneinander anlegen lassen und somit Vergleiche ziehen.

4.1.1.4 Farbpalette

Aseprite generiert beim Öffnen eines vorhandenen Sprites automatisch eine passende Farbpalette, die aus den Farben des Bildes entsteht. Diese Farbpalette kann auch abgespeichert werden und bei Bedarf bei späterem Gebrauch nochmals benutzt werden.

4.1.2 Benutzeroberfläche

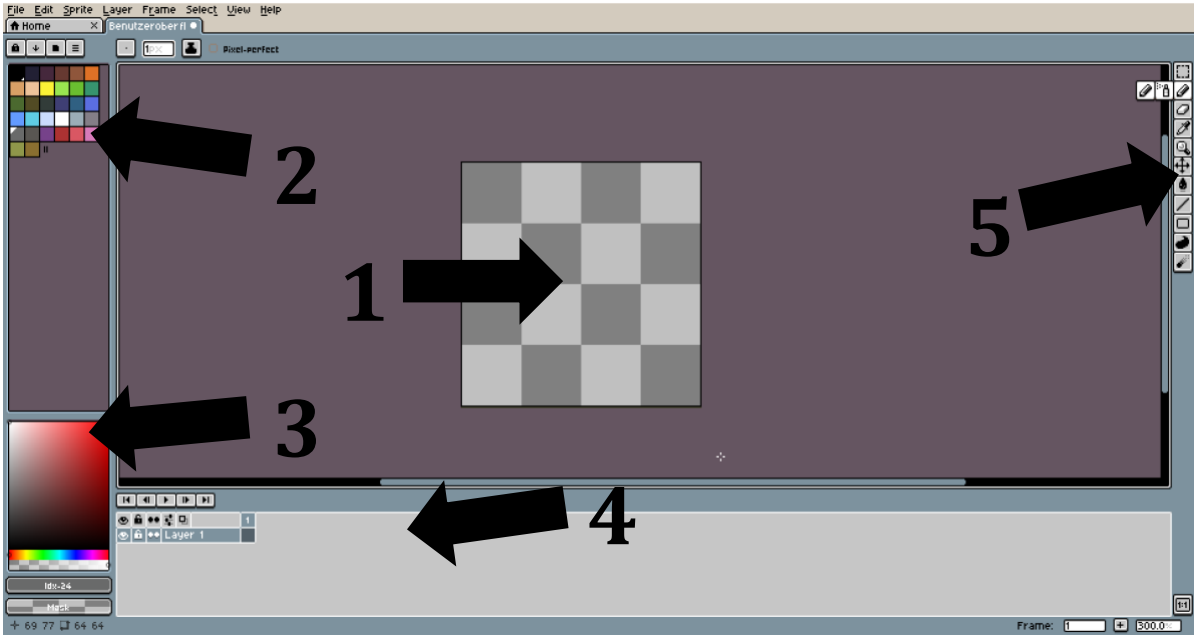

Abbildung 4 Aseprite Benutzeroberfläche

Die Benutzeroberfläche lädt durch ihre Struktur und das Pixeldesign ein, die Pixelgrafiken kreativ zu gestalten, somit ist es die ideale Umgebung für solche Arbeiten. Alles hat seinen angenehmen Platz und dadurch können Anfänger vereinfacht den Bezug zu Pixelgrafiken bekommen. Siehe Aseprites Benutzeroberfläche (**Error! Reference source not found.** Die Benutzeroberfläche besteht aus der Arbeits- bzw. Zeichenfläche (Nr. 1), der Farbpalletten-Box (Nr. 2), der Farbauswahl-Box (Nr. 3), der Timeline (Nr. 4) und aus der Tool-Box (Nr. 5). Die Zeichenfläche ist der Schwerpunkt, der die Szenen graphisch darstellt bzw. worin die Grafiken gezeichnet werden. Sie besteht aus einem Canvas und ist der sichtbare Bereich des Sprites. Die Canvas-Size ist immer veränderbar.

Die Farbpaletten-Box beinhaltet eine schon generierte oder selbst erstellte Farbpallette. Bei der Farbpalletten-Box können selbst- oder fertig erstellte Palletten kopiert und eingesetzt werden und für die Pixel-Art benutzt werden. Die Farbauswahl-Box erlaubt es Farben in den Farbräumen RGB, HSV und HSL zu selektieren. Die Timeline besteht aus einem Bereich, welcher eine bestimmte Zeit von einer Szene in einzelne Bilder aufteilt, wie auch die dazugehörigen Ebenen. Die Bilder oder Zellen in der Timeline können für Animationen kreiert, kopiert, bewegt, verlinkt und eingesetzt werden. Diese können auch in verschiedene Ebenen aufgeteilt werden und diese beschriftet werden, um die Animationen leichter zu organisieren. Die Animation kann mit Vor- oder Rückwärts abgespielt werden oder im Ping-Ping Modus (Vor-und Rückwärts).

Die Toolbox beinhaltet alle Werkzeuge, die benötigt werden, um ein 2D Objekt zu erzeugen oder zu bearbeiten.

4.1.3 Aseprites Features

Aseprite ist für die Arbeit mit Pixeln konzipiert und erleichtert das Arbeiten mit Animationen.

4.1.3.1 Timeline

Die Komponenten von dem Objekt sind aufgeteilt auf verschiedene Ebenen, Frames und Zellen. Dies erleichtert es die einzelnen Ebenen und Frames zu bearbeiten. In anderen Programmen besteht dieses Feature nicht.

4.1.3.2 Animations-Wiedergabe

Es ist möglich während der Animation, beim Vor- oder Rückwärts spielen die Animation bzw. das Objekt gleichzeitig zu bearbeiten. Außerdem besteht die Möglichkeit, die Wiedergabegeschwindigkeit der Loops und die Geschwindigkeit jedes einzelnen Frames zu ändern. Es gibt drei Wiedergabemodi: Vorwärts, Rückwärts und Ping-Pong.

4.2 Adobe Photoshop

Abbildung 5 Adobe Photoshop Logo
Quelle:
https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Adobe_Photoshop_CC_icon.svg/768px-Adobe_Photoshop_CC_icon.svg.png

4.2.1 Was ist Adobe Photoshop?

Adobe Photoshop ist ein kostenpflichtiges Pixelgrafikprogramm des riesen Softwareunternehmen Adobe Systems. Dieses Pixelgrafikprogramm ist spezialisiert auf Bildbearbeitungen und für Vorprozesse des Drucks, durch die Datenaufbereitung und

Retuschierung. Photoshop wird von den meisten als „das Bildbearbeitungsprogramm“ angesehen. Sie wird von Profis, wie Photographen, Grafik-Designern, WEB- und Videoexperten und Künstler eingesetzt. Photoshop stellt, digitale Bildbearbeitungen zwei & dreidimensional (2D, 3D), Videobearbeitung und Bildanalyse zur Verfügung. Außerdem kann man mit Photoshop Bildeigenschaften oder den Content manipulieren, Bilder optimieren, simulieren, retuschieren, Bilder erzeugen oder anders nach eigenem Wunsch gestalten und vieles mehr.

Es stehen auch eine Vielzahl von Plug-Ins zur Verfügung, die neue Effekte hinzufügen, die vorhandene Funktionalität verbessern und den Arbeitsablauf vereinfachen.

Photoshop verfügt über Werkzeuge für RAW-Bildeinstellungen, Objektivkorrektur, Retuschieren, Bildheftung, HDR, Fixieren von Rahmen. Es unterstützt die meisten professionellen Farbmodi und Dateiformate. Es enthält umfangreiche Listen mit Filtern, Stilen, Effekten, Schriftarten sowie Werkzeugen zum Malen, Skizzieren und zur Typografie. Es versteht sowohl Raster- als auch Vektorgrafiken. Es enthält sogar Tools für die Videobearbeitung, das Arbeiten mit 3D-Objekten und die Unterstützung für den 3D-Druck.

Der Nutzer hat auch die Möglichkeit, den kompletten Vorgang, welcher Erstellung bzw. Modellierung und Animation von Pixelgrafiken beinhaltet, in der Anwendung abzudecken. Mit Adobe Photoshop lassen sich auch 2D-Animationen im Retro-Stil erstellen. [9]

4.3 Aseprite versus Photoshop

Beide Anwendungen (Aseprite und Photoshop) sind für verschiedene Zwecke gedacht. Aseprite ist für Sprites, während Photoshop für die Fotobearbeitung konzipiert ist.
Aseprite ist zu 100% für Pixelart-Entwickler gedacht.
Bei Photoshop sind das Erstellen, Modellieren oder Animieren von Pixel-Sprites eher nur beiläufig bzw. nachträglich entwickelt, da sie sich zunächst auf das Bearbeiten von Fotos konzentrieren.

4.3.1 Entscheidung für Aseprite

4.3.1.1 Animationswiedergabe Einstellung

Bei Aseprite kann man die Wiedergabegeschwindigkeit der Animations-Loops und die Geschwindigkeit jedes einzelnen Frames ändern.

[9] https://whatis.techtarget.com/de/definition/Adobe-Photoshop

4.3.1.2 Zellen Verwaltung

In Aseprite kann man auch den Indikator bzw. Zellen in der Animationszeitleiste kopieren und einzufügen für Animationen. Im Konkurrenzprogramm Photoshop geht dies nicht. Dort muss man den Frame in der gewünschten Form ausschneiden, um ihn dann abzuspeichern und das in ein neues Feld einzufügen. Es können auch verschiedene Teile oder Zellen der Timeline markiert werden und diese dann einzeln durchlaufen lassen. Beispielsweise, wenn verschiedene Animationen für den gleichen Charakter existieren.

4.3.1.3 Tools

Aseprite hat viel mehr Tools und Funktion, welche das Erstellen und Bearbeiten von Pixel vereinfachen. Tools wie Beispielsweise: „Jumble Tool, „Condor Tool“ und „Spray Tool“. Funktionen wie: „Shading-Funktion“.

4.3.1.4 Bedienung

Aseprite ist im Vergleich zu Photoshop sehr trivial. Photoshop kann mühsam sein, wenn man es für andere Zwecke verwendet wie Pixel-Sprites. Photoshop erfordert einige Einstellungen für die Pixel-Grafik.
Da Aseprite auf Pixel-Art spezialisiert ist, kann man die nützlichen Funktionen von Aseprite einfacher und schneller nutzen als mit Photoshop. Die nötigen Tools zum Erstellen und Bearbeiten der Grafiken sind spezialisiert auf Pixel-Grafiken und sind deswegen besser und trivialer zu bedienen.

Alle Tools und die meisten Funktionen verfügen über Tastenkombinationen, so dass Ergebnisse schnell erzielt werden können. Daher ist die Bedienung und Anwenderfreundlichkeit bei Aseprite, um einiges besser als bei Photoshop.

4.3.1.5 Anwenderfreundlichkeit

Aseprite ist ein sehr fokussiertes Programm: Es ist nicht mit Symbolen gefüllt, es gibt keine übermäßigen Funktionen, und Dialogfelder haben im Allgemeinen nur ein paar Optionen, so dass man nie überfordert ist. Aseprite ist sehr leicht zu erlernen und sehr einfach zu bedienen.

Auch die Benutzeroberfläche ist wie schon erwähnt, sehr einladend und anwenderfreundlich, da man das Gefühl hat in der richtigen Umgebung zu sein, durch die Pixel-Art Oberfläche. Alles scheint einen angenehmen Platz zu haben und dadurch könnten sich Anfänger wie zu Hause fühlen.

4.3.1.6 Animationszeitleiste

In meinen Augen das stärkste Argument für Aseprite ist die einfache Bedienung in der Animationszeitleiste beim Animieren der Grafiken. Dadurch, dass Layers, Frames und Zellen alle in einem Fenster liegen, kann man sich die Grafiken bzw. die Animationen sehr einfach organisieren, was bei Photoshop nicht geht. Bei Photoshop liegen Frames und Layer in jeweils eigene Bereiche und somit kann man leicht durcheinanderkommen, da man bei aufwendigen Animationen oder Animationen wo man zahlreiche Layers benutzt schnell die Übersicht verlieren.

4.3.1.7 Fazit

Wenn es also zur Erstellung, Bearbeitung oder Animation von Pixel-Grafiken kommt ist Aseprite hervorragend, das Benutzen der Timeline und den Indikatoren ist um einiges einfacher und besser als mit Photoshop. In der Mehrzahl der Fälle eignet sich Aseprite viel besser für die Erstellung von Pixel Art Grafiken. Die gesamte Anwendung ist darauf ausgerichtet, die bestmögliche Pixelgrafik zu erstellen und dabei weitaus mehr.

5 Grafiken

Eine Grafik ist ein Bild oder eine visuelle Darstellung eines Objekts.

5.1 Pixelart als Stilmittel

Pixel Art ist die Kreation digitaler Kunst durch Editieren von Pixel für Pixel. Ein Pixel ist ein digitales Bildschirmelement in Form eines Quadrats. Das Erstellen von Pixelgrafiken ist das platzieren jedes Pixels von Hand. Je mehr Pixel ein Bild hat, desto detaillierter wird ein Bild.

5.1.1 Wieso in Pixel Art?

Pixel Art eignet sich hervorragend zum Erstellen einfacher Bilder. Gut ausgeführte Pixelbilder bieten beispiellose Klarheit und visuelle Kontrolle, insbesondere auf kleinen Bildschirmen, die nicht immer die höchste Qualität haben. Die palettisierten (auf bestimmte Farben beschränkt) Bilder sind auch äußerst speichereffizient. Hundert Kilobyte sind mehr als genug für ein Spiel. Im Allgemeinen können Pixelbilder eine sehr effiziente Methode sein, um eine große Anzahl von Kunstwerken im selben Stil zu erzeugen. Somit schauen die Grafiken uniform aus.
Es gibt wenige Farben, sodass man Grafiken direkt, schnell und einfach bearbeiten kann. Dies bedeutet, dass das Erstellen der Grafiken von Spielobjekten trivial sein kann. Deshalb ist Pixelkunst eine sehr einfache erlernende Art der digitalen Kunst, da man vollste Kontrolle über jeden Pixel hat.

Außerdem können die Grafiken in Pixel Art von allen Programmen gelesen werden, da das Dateiformat der Grafiken JPG oder PNG ist. Dies ist jedoch nicht der Fall bei Vektordateiformaten. Vektorgrafiken ist die Verwendung von Linien, Punkten, Kurven und Formen, die alle auf mathematischen Ausdrücken basieren, um Bilder in der Computergrafik darzustellen. Ein weiterer Nachteil bei Vektorgrafiken ist, dass oftmals kleine Zeichnungsfehler sichtbar sind, wenn die Bilder bis zu einem gewissen Grad vergrößert werden. Ein solcher Vorfall kann die Qualität der Bilder beeinträchtigen, insbesondere wenn es im Animationsbereich verwendet werden. Ein weiterer Nachteil von Vektorbildern ist, dass sie zeitaufwändig sind und auch spezielle Techniken für die Erstellung benötigt werden. Beispielsweise die Bézierkurve. Bei Pixelgrafiken ist dies nicht der Fall, da man ohne Vorwissen auch gutaussehende Grafiken zeichnen kann. [10]

[10] https://liamjardimlacey.weebly.com/pixel-art.html

5.2 Farben

Farben sind eine visuelle Wahrnehmung, die hervorgerufen werden, wenn das Licht in einem für das menschliche Auge sichtbaren Bereich liegt. Die Farbwahrnehmung ist individuell, denn dies kann von Mensch zu Mensch unterschiedlich wahrgenommen werden. Durch die Bestand-, und Beschaffenheit von Augen und Empfindlichkeit der Rezeptoren können Farben unterschiedlich wahrgenommen werden.

5.2.1 Farbtheorie

Farben sind das wichtigste Mittel, das man bei Grafiken oder Webseiten einsetzen kann. Sie können verschiedenste Stimmungen in Menschen wecken und liegen in einem bestimmten Farbraum.

5.2.1.1 Farbpsychologie

Es gibt eine große Anzahl von emotionalen Reaktionen, die mit Farben verbunden sind. Einige Farben können unterschiedliche Bedeutungen haben, sogar gegensätzliche Bedeutungen, die vollständig auf Kontext und Anwendung basieren. Farben können was Bestimmtes assoziieren. Durch die Farben kann man festlegen, welche Assoziation bei Menschen hervorgerufen wird. Die Vermittlung von Stimmungen oder auch die Beeinflussung von Stimmungen, kann durch das Einsetzen bestimmter Farben nochmals verstärken. Um damit richtig umgehen zu können, muss man wissen welche Farbe, was für Stimmung bei Menschen hervorrufen. [11]

5.2.1.1.1 Blau

Mit Blau ist verbindet man Wissen und Gelassenheit, sowie Kühle, Sympathie, Harmonie, Verständnis, Freundschaft, Geduld, Gesundheit, Ergebenheit, Ehrlichkeit, Ehre, Loyalität, Frieden, Weisheit, das Unendliche, Formalität, Treue, Mitgefühl. Es ist mit dem Himmel, dem Meer und Eis verbunden. Das Symbol für Wahrheit und Reinheit.
Es verlangsamt den menschlichen Stoffwechsel und wirkt beruhigend. Blau ist stark mit Ruhe und Gelassenheit verbunden.
Mit Dunkelblau verbindet man Tiefe, Stabilität und Wissen, Mysteriös, geheimnisvoll.
Mit Hellblau verbindet man Kälte, Gefühlslosigkeit, Härte, Stolz und Abweisung.

Unser Hauptmerkmal in den Grafiken oder auf der Webseite sind die verschiedenen Abstufungen aus dem Farbwert Blau. Das Ziel war es ein kühles bzw. kaltes Gefühl für den User zu bewirken, welches auch gleichzeitig mysteriös und geheimnisvoll ist.

[11] https://zevendesign.com/color-association/

5.2.1.1.2 Rot

Mit Rot verbindet man Wärme, Ärger, Grausamkeit, Aufregung, Kraft, Stärke, Liebe, Hass, Leidenschaft, extreme Emotionen, Kampf, Lust, Gefahr, Energie.
Rot ist eine sehr emotionale Farbe und symbolisiert auch Heiß, Feuer, Hitze, Blut

5.2.1.1.3 Grün

Mit Grün verbindet man Frische, Geheimnis, Neid, Hoffnung, ewiges Leben, Freude, Zurückhaltung Ausdauer, Fruchtbarkeit, Glück, Erfolg, Wachstum, Ehrgeiz, Gier. Es ist die Farbe der Natur, wie man sie normalerweise sieht. Grün mildert Spannungen und bedeutet Ruhe und ist eine beruhigende Farbe für das menschliche Auge

5.2.1.2 Farbharmonie

Einige Farben arbeiten großartig zusammen und andere dagegen nicht. Farben könnten gemeinsam gut miteinander harmonieren, sie können in Kontrast zueinanderstehen oder disharmonieren. Die Kombination aus verschiedenen Farben harmonieren erst dann, wenn sie für den Menschen als angenehm empfunden werden. Das liegt natürlich am Betrachter, was angenehm für Auge ist. Eine Harmonie wäre zum Beispiel eine Abweichung oder Abstufung einer Farbe.
Kontraste sind Farben, die sich stark voneinander abheben. Kontraste kommen durch Komplementärfarben zustande. Komplementärfarben sind Farben, die sich im Farbkreis gegenüberliegen. Die Wirkung daher ist, dass die Farben sich gegenseitig verstärken.
Da sich Farben gegenseitig beeinflusse, muss man sehr darauf achten welche Farben man kombiniert. Farben könnten unrein wirken oder im Auge wehtun.

5.2.2 Farbmodelle

Der Bereich von Farben, die in einem bestimmten Verfahren dargestellt werden, nennt sich Farbraum, Farbmodell oder Farbsystem. Ein Farbraum ist ein definierter Farbbereich. Bekannte Farbräume sind RGB, CMYK und HSV. Die Verwendung von Modellen stellt eine enorme Vereinfachung des visuellen Systems dar.

5.2.2.1 RGB

Der RGB-Farbraum besteht aus den 3 Primärfarben Rot, Grün und Blau. Man spricht hier auch vom additiven Farbraum, da durch die Kombination der drei Grundfarben, ihre Strahlungsenergie addiert. Aus Ihnen kann man alle Farben mischen wie beispielsweise: Wenn man Rot mit Blau mischt, dies Magenta ergibt oder durch Rot und Grün Yellow entsteht. Mischt man Blau mit Grün, erhält man Cyan. Die Farben, die durch Kombination aus zwei Primärfarben entstehen, heißen sekundär Farben. Mischt man nun alle

Farbkanäle (Rot + Grün + Blau) miteinander entsteht die Farbe Weiß. Jedoch kann aus dem Mix durch andere Farben nicht Primärfarben erstellt werden.

Durch, erhaltenen Farben (Cyan, Magenta und Yellow) stellt sich nun der CMYK-Farbraum her.

5.2.2.2 CMYK

Der CMYK-Farbraum besteht aus den Farben Cyan (C), Magenta (M), Yellow (Y) und Black (K). CMYK nutzt das subtraktive Farbsystem aus, indem Farben auf einer Oberfläche abgeschieden werden, um bestimmte Lichtfarben selektiv zu absorbieren. Das Licht, das von den Farben nicht absorbiert wird, wird in das Auge eines Betrachters reflektiert, was dazu führt, dass sie die beabsichtigte Farbe sehen. CMYK ist am besten geeignet für den Druck-Bereich.

5.2.2.3 HSV

HSV steht für Hue (H), Saturation (S) und Value (V)

HSV basiert auf drei Werte: Farbton, Sättigung und Helligkeit. Dieser Farbraum beschreibt Farben anhand ihrer Sättigung und ihres Values (Helligkeitswert).

Farbton ist der Farbanteil des Farbmodells.

Die Sättigung ist die Intensität der Farbe. Wenn die Sättigung gegen Null geht, wird mehr Grau in die Farbe eingeführt, dann hat das den Effekt, dass die Farbe ein wenig bleicher ist, je mehr Sättigung die Farbe bekommt, steigt die Intensität der Farbe.

Der Value (Wert) arbeitet in Verbindung mit der Sättigung und beschreibt die Helligkeit der Farbe von 0 bis 100 Prozent, wobei 0 vollständig schwarz ist und 100 die hellste ist und die meiste Farbe zeigt.

5.2.3 Farbpalette

Eine Farbpalette ist eine Ansammlung von ausgewählten Farben. Das Ziel ist es, ein Bild in einer begrenzten Anzahl von Farben zu produzieren. Das Erstellen einer Farbpalette sollte möglichst wenige Farben haben, um konsistente und einheitliche Grafiken zu erstellen.

Bei jeder Palette besteht die Absicht darin, möglichst viele der am häufigsten benötigten Farben, anstelle der weniger nützlichen Farben bereitzustellen. Wenn der gesamte RGB-Bereich (oder ein anderer großer Farbraum) zur Verfügung steht, besteht der Punkt der Verwendung einer Palette im Gegensatz zur direkten Auswahl aus dem gesamten verfügbaren Farbraum darin, die Auswahl der Farben zu vereinfachen - je weniger Optionen vorhanden sind, desto einfacher ist es, eine gute Entscheidung zu treffen. Das Erstellen harmonischer Bilder erfordert die Auswahl einer noch kleineren Untermenge der Palette. Harmonischere Paletten, bei denen der Großteil der Palette für ein bestimmtes

Projekt wahrscheinlich nützlich ist, sind naturgemäß in der Art der Bilder eingeschränkt und unterliegen einem bestimmten Limit.

Abbildung 6 Wraithknight's Farbpalette

Deswegen ist die Farbpalette für die Grafiken limitiert und nur mit den Farben versehen, die auch benötigt werden. Auch Farben die gut miteinander harmonieren wie gut in (**Error! Reference source not found.**) zu sehen ist. Durch die Einschränkung ist das Zeichnen der Grafiken um einiges müheloser und simpler. Die Grafiken werden dadurch außerdem einheitlicher, man hat bessere Kontrolle über die Kontraste und ist einfacher für Schattierungen.

5.3 Animationen

Animation sind eine Simulation der Bewegung, erstellt durch eine Reihe von Bildern, oder der Anzeige von Bildern. Der Unterschied zwischen Animation und Video, ist das ein Video eine fortlaufende Bewegung durchführt und es in einzelne Frames zerlegt ist, bei Animationen beginnt die Animation mit unabhängigen Bildern und fügt sie zusammen, um die Illusion einer kontinuierlichen Bewegung zu bilden.

5.3.1 Probleme

Das Animieren der Charaktere hatte anfangs sehr viele Startschwierigkeiten und reichlich viele Probleme. Probleme wie: Zwecklose Farben, Missachtung physikalische Gesetze, Keine Einhaltung der Proportionen und Brechung der Formen.

5.3.1.1 Zwecklose Farben

Problem: Unklarheit der Formen.

Ursache: Formen wie Arme und Beine hatten überflüssige Farben, die nicht ausschlaggebend oder nützlich waren. Überzählige Farben, die der ganzen Form keine Klarheit beschaffen hat und somit auch wirkungslos waren.

Lösung: Farbpalette limitiert und Formen maximal zwei Farben geben.

5.3.1.2 Missachtung physikalische Gesetze

Problem: Physik hinter Animationen nicht korrekt.

Ursache: Die Gravitation wurde beispielsweise bei der Sterbeanimation beim Fallen nicht regelrecht berücksichtigt, indem der Charakter weiter nach vorne fiel. Somit sah die Animation aus als ob der Charakter leicht abspringen würde.

Lösung: Mit Übereinstimmung der Gravitation, fällt der Charakter direkt nach unten.

5.3.1.3 Keine Einhaltung der Proportionen

Problem: Verhältnisse von Formen ungenau.

Ursache: Die Größe von Beinen oder Armen haben in den Nachfolgenden Frames ihre Größe verändert. Auch nur minimalistisch mit ein oder zwei Pixeln zu dick oder dünn.
Lösung: Skelett-Form für den jeweiligen Charakter gezeichnet und als Basis genommen. (Abbildung)

5.3.1.4 Brechung der Formen

Problem: Formen wie Arme und Beine waren nicht einheitlich.
Ursache: Beim Laufen oder bei Attacken haben die Arme und Beine der Charaktere in den Nachfolgenden Frames Unterschiede in den Formen. Somit hatte das den Effekt, dass Arme oder Beine gebrochen waren.

Lösung: Skelett-Form für den jeweiligen Charakter gezeichnet und als Basis genommen. (Abbildung)

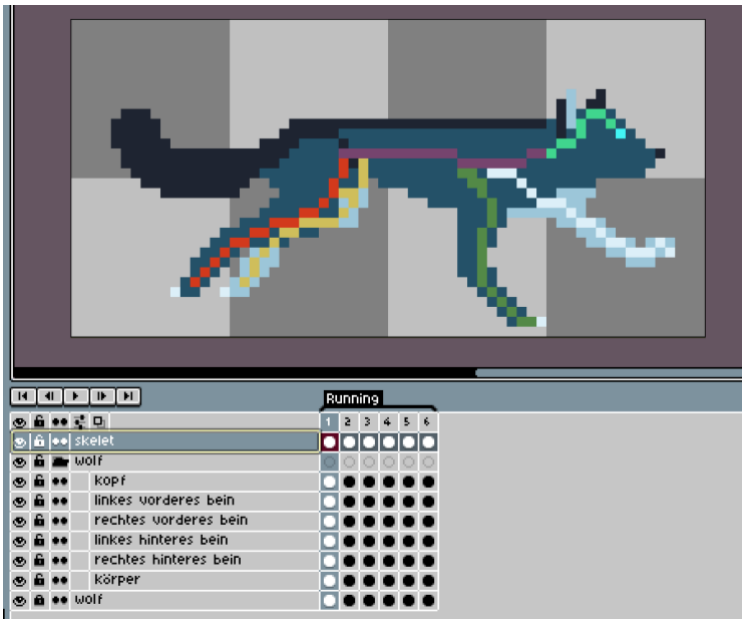

Abbildung 7 Wolf Skelett

Lösung der Probleme: Ein Skelet, welches in jeder der Formen aufgeteilt ist und dies als Basis verwendet wird. Somit konnte genauer auf Formen und Proportionen geachtet werden. Das aufteilen auf verschiedene Layers ist auch eine enorme Hilfestellung, da man dann die verschiedenen Teile in den Layers unabhängig zum Rest der Grafik bearbeiten, verschieben, zeichnen kann.

5.3.2 Ablauf der Erstellung

Der Ablauf einer Erstellung einer Animation ist es zunächst einmal viele Sketches einer Figur zur erstellen in ihrer natürlichen Standposition. Ist man zufrieden mit dem Entwurf, kommt die Feinarbeit, durch die genauen Farben und der Feinschliff der einzelnen Pixel in ihren Formen. Zu beachten hierbei ist, dass die ganzen einzelnen Formen wie Kopf, Arme, Beine usw., in einzelne Ebenen aufgeteilt ist, wie auch in (Abbildung) zu erkennen ist. Sobald man mit der Standposition zufrieden ist erstellt man das zugehörige Skelet. Im nächsten Schritt wird nur das Skelet animiert beispielsweise beim Laufen. Sobald die Animation des Skeletts fertig ist, wird sie nach bestimmten Bedingungen geprüft z.B. nach Richtigkeit der Realität oder der Physik. Hierbei ist zu beachten die Formen im selben Verhältnis zu haben, wie auch in der natürlichen Standposition. Zu guter Letzt kommt der Feinschliff durch Farben und einzelner Pixel.

5.4 Elemente

Elemente im Spiel sind Charaktere, Projektile und die Umgebung.

5.4.1 Aktoren/Figuren/Charaktere

Das Spiel hat sechs Charaktere. Die Charaktere sind der Hauptcharakter, ein Wolf, ein Frosch, der Dunkle Ritter, Bogenschütze und der Endgegner. Zu jedem Charakter gehören Animationen, dazu gehören beispielsweise das Laufen, die Attacken und das Sterben. Der Hauptcharakter ist der einzig steuerbare Charakter vom User und die restlichen Charaktere die Gegner, welche automatisch gesteuert werden.

5.4.1.1 Charakterbeschreibung

Jeder der Figuren hat natürlich eine eigne Persönlichkeit und einen Charakter.

5.4.1.1.1 Hauptcharakter

Abbildung 8 Hauptcharakter Standposition

Der Hauptcharakter ist ein wiederauferstandener Ritter. Durch seinen Tod hat er seine Ehre verloren und möchte diese wiedererlangen. Bewaffnet mit einem Langschwert und einem Wurfmesser versucht er sich durchzukämpfen. Durch seine Schnelligkeit und durch seine Kampfstärke ist es ihm möglich gegen mehrere Gegner zu kämpfen und diese im Idealfall zu besiegen.

5.4.1.1.2 Wolf

Abbildung 9 Wolf Standposition

Der Wolf ist durch seine Agilität und Schnelligkeit ein sehr starker Gegner. Er ist extrem flink und ist sehr aggressiv, daher richtet er bei dem Ritter viel Schaden an. Durch seine Intelligenz kann er dem Ritter extrem viel abverlangen, jedoch hat er keine Art von Schutz an seinem Körper, deswegen reicht wenig Schaden aus, um ihn zum Fallen zu bringen.

5.4.1.1.3 Frosch

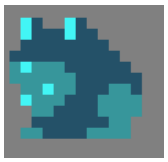

Abbildung 10 Frosch Standposition

Der Frosch ist ein schleimiger Gegner mit wenig Beweglichkeit. Er versucht mit der Seelenruhe, den Ritter abzulenken, damit andere Gegner ihn währenddessen attackieren können. Bei seinem Tod hinterlässt er schleimige Kampfspuren. Er explodiert vor seinem Lebensende, um dem Ritter noch möglichst viel Schaden mit auf seinem weiteren Weg zu geben.

5.4.1.1.4 Dunkler Ritter

Abbildung 7 Dunkle Ritter Standposition

Der Dunkle Ritter ist der Erzrivale des Hauptcharakters. Gefüllt mit dem Seelenfeuer und seiner ganzen Wut, versucht er alles, dass der Ritter nicht an ihm vorbeikommt. Durch seine leichte Rüstung ist er sehr agil und beweglich. Er trägt ein großes Langschwert, welches er, durch das Gewicht, über seine Schulter schwingen muss.

5.4.1.1.5 Bogenschütze

Abbildung 8 Bogenschütze Standposition

Der Bogenschütze ist natürlich mit Pfeil und Bogen ausgestattet und nutzt diese effektiv von der Entfernung um den Ritter zu stoppen. Jedoch ist er sehr langsam und zerbrechlich da er schon ein langes Leben hinter sich hat. Der Bogenschütze ist außerdem leicht paralysiert und ähnelt einem Zombie, der durch die Gegend irrt. Um den Ritter zu stoppen riskiert er sein letztes Blut.

5.4.1.1.6 Endboss

Abbildung 9 Endboss Standposition

Der Endboss ist das Alphatier der Untoten und ist die letzte Instanz um den Ritter zu besiegen. Mit seiner Stärke und seinem Willen den Ritter zu besiegen, ist er der stärkste Gegner im Spiel und setzt alles darauf dem Ritter seine verlorene Ehre nicht wiederzugeben. Gefährlich wird es, wenn der Endboss an den Ritter herankommt, da er sehr gefährlich ist und einen enormen Schaden ausrichten kann.

5.4.2 Projektile

Projektile sind Sprites die zu einer bestimmten Grafik gehören aber als eigenes externes Individuum betrachtet werden. Beispielsweise abgefeuerte Geschosse oder Teile einer Attacke, welche auch animiert sind.

5.4.2.1 Pfeil

Abbildung 10 Pfeil Projektil

Der Pfeil ist das Projektil vom Bogenschützen und ist das abgefeuerte Geschoss aus seiner Attacke.

5.4.2.2 Wurfmesser

Abbildung 11 Wurfmesser Projektil

Das Wurfmesser ist das abgefeuerte Geschoss des Hauptcharakters.

5.4.2.3 Windstoß

Abbildung 12 Windstoß Projektil

Der Windstoß entsteht durch den enormen Schlag des Hauptcharakters. Durch diese Druckwelle entsteht noch weiterer Schaden bei Gegnern.

5.4.2.4 Gebiss

Abbildung 13 Gebiss Projektil

Das Gebiss ist Teil der Angriffsanimation des Wolfes und erzeugt mehr Dynamik und Aggressivität, durch ihre Animation.

5.5 Kulisse

Um für die einzelnen Level eine passende Kulisse zu haben, musste das Team Grafiken erstellen. Da sich das Spiel im Mittelalter abspielt wurde auch das Design und dazugehörig die Farben an das Mittelalter angepasst. Das Setting des Spieles ist in einem düsteren, geheimnisvollen Wald. Daher mussten Bäume, Pflanzen und der Boden passend erstellt werden.

5.5.1 Bäume

5.5.1.1 Zeichnen der Bäume

Wie man an der unteren Grafik (Abbildung 14) sehen kann, ist die Modellierung von Grafiken ein langer Prozess. Wir haben uns dabei entschieden, für das Entwerfen der Bäume 32 Pixel in der Breite zu verwenden und 64 Pixel in der Höhe. Prinzipiell ist es egal wie groß die Bäume sind, da Pixelgrafiken skalierbar sind, jedoch ist es wichtig beim Modellieren, da die Größe der Grafik mit der Detailierung skaliert. Je mehr Pixel desto mehr Details können gezeichnet werden. Im ersten Schritt definiert man die Form des Baumes, je nachdem ob es z.B. ein Laubbaum oder ein Nadelbaum ist. Anschließend werden grobe Schattierungen hinzugefügt. Im Bild werden die Schattierungen von links nach rechts gezogen. Jedoch wurde in unseren Grafiken die hellste Stelle der Schattierung am oberen Ende des Baumes gesetzt, da wir eine Top-Down Perspektive haben. Anschließend werden die Blätter gezeichnet. In der Abbildung kann man sehen, dass alle Blätter in etwa die gleiche Form haben. In unserem Projekt haben wir uns jedoch gegen die Zeichnung von Blättern entschieden, da die Bäume im Spiel so klein sind, dass man sie kaum erkennen würden und dabei ein rauschen entstehen könnte. Anschließend wird noch der Baumstamm hinzugefügt.

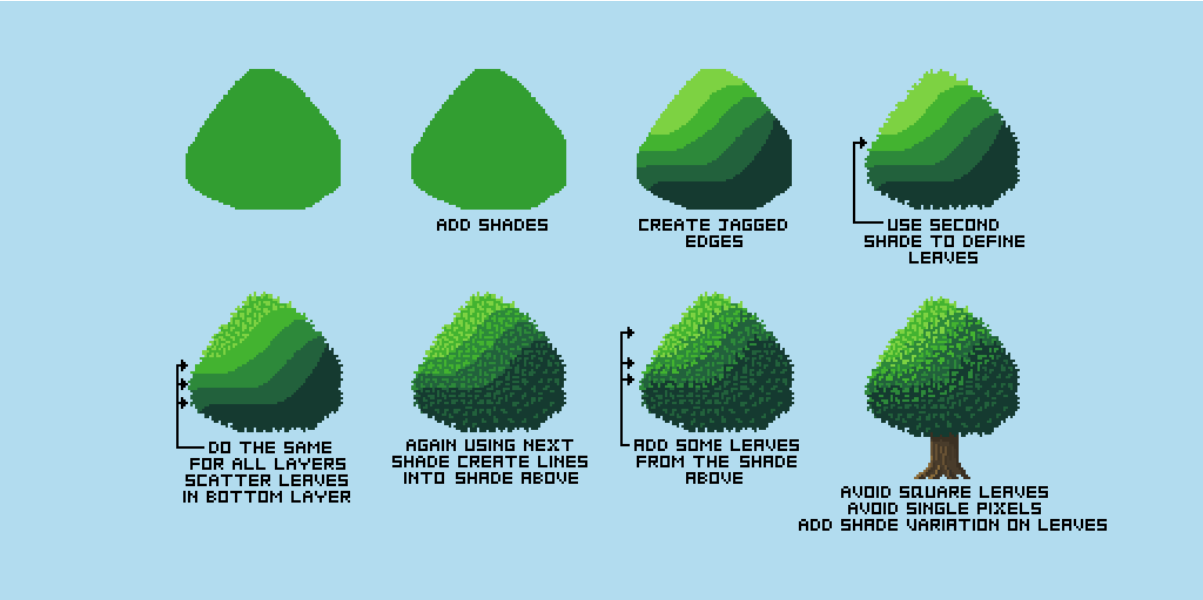

Abbildung 14 Zeichnen von Bäumen (Quelle: https://imgur.com/gallery/eD8NA)

5.5.1.2 Endergebnis

In der unteren Grafik (Abbildung 15) kann man drei verschiedene Bäume erkennen. Von Links gesehen der Erste Baum, ist ein normaler Laubbaum. Der mittlere Baum ist etwas dünnerer Laubbaum und der Letzte ist ein Klassischer Nadelbaum. Mit den verschiedenen Bäumen soll der Wald realistisch simuliert werden.

Abbildung 15 Bäume

5.5.2 Pflanzen

Um dem Wald noch ein bisschen Düsterer wirken zu lassen, wurden noch kleinerer Elemente zu dem Tileset hinzugefügt. Da alle diese Elemente nahe am Boden sind haben sie alle eine braune Farbe. Diese Farbe soll eine drei Dimensionale Vorstellung generieren, obwohl das Spiel nur auf einem zweidimensionalen Raster stattfindet.

Abbildung 16 Pflanzen

.

6 Webseite

Zum Spiel gibt es auch eine begleitende Produktwebseite, die die genaueren Informationen über Spiel und Charaktere noch genauer erläutert.

6.1 Struktur

Bevor man mit der Umsetzung der Webseite beginnt, werden Mockups (Zeichnungen von Layouts) erstellt, um die Vorstellungen im Kopf zu visualisieren. Die Layouts, die man erstellt hat kann man dann auch vereinen um das Beste Resultat zu erzielen oder ein komplett neues erstellen.

Die Struktur der Produktwebseite besteht aus 4 Sektionen, dem Hero, dem Team, der Produktbeschreibung und der Charakterbeschreibung.
Ein Hero ist das erste was man bei einer Webseite sieht, wenn sie geladen wird. Sie bildet den Ersteindruck der Landing Page.

6.1.1 Navigation

Die Navigation ist das Herzstück jeder Webseite. Sie ist eine Art Wegweiser. In der Top bar der Produktwebseite ist eine Circle Navigation, die nicht wie bei den meisten Webseiten eine Navigationsleiste Links oder Rechts ist. Die Webseite soll sich von anderen Spielwebseiten abheben und deswegen, ist in der Navigation in der Mitte auch das Logo von Wraithknight. Zum Logo siehe (Abbildung 17).

Die Icons verlinken auf die Weiteren Sektionen zu Team, Produktbeschreibung und Charakterbeschreibung. Außerdem sind die Icons auch im Pixel-Grafik-Editor erstellt, um gleich am Anfang dem User das Gefühl zu geben, dass unser Spiel ein Pixel Spiel ist.

6.1.2 Hero

Der Besucher soll sofort erkennen, worum es auf der Seite geht, deshalb besteht der Hero aus einem animierten GIF von einem Gameplay. Die Idee dahinter ist, Dynamik in die Webseite zu erzeugen.

6.1.3 Team

Nach dem Hero kommt man zur Team-Sektion, wo die Teammitglieder vorgestellt werden mit ihrer Position, durch Profilfoto, Position im Team und Kurzerklärung.

6.1.4 Produktbeschreibung

Bei der Produktbeschreibung-Sektion erhält man einen gesamten Überblick zum Spiel mit einem Video zu unserem Spiel, welches dies nochmals fürs Auge verdeutlicht. Die Herausforderung hier, ist es nicht mit Text zu überfüllen und möglichst alle Informationen in wenig Worten zu packen. Deswegen wurde der Text öfters optimiert, um damit einen spannenden Text und eine gewisse Spannung zu erzeugen das Spiel zu spielen.

6.1.5 Charakterbeschreibung

Die letzte Sektion ist die Charakterbeschreibung. In dieser Sektion war die Idee, dass nach der Produktbeschreibung wieder etwas fürs Auge zu haben. In dem Bereich werden die Spielfiguren vorgestellt mit Bild und Beschreibung. In dem Bereich sind alle Aktoren graphisch aufgelistet und durch einen Klick erhält man zu dem jeweiligen Charakter eine genauere Beschreibung und eine vergrößerte Ansicht der Figur.

6.2 Design

Ein Design ist die Umsetzung eines Konzepts oder einer Idee aus einem Modell, einem Plan oder einem Muster.

6.2.1 Zweck

Der Zweck eines guten Designs ist es den Besucher so lange wie möglich auf seiner Seite zu haben, damit der Besucher sie nicht augenblicklich verlässt. Das Design ist essentiell bei jeder Webseite, denn die bleibt auch beim Besucher nachdem er/sie die Seite verlassen hat, auch im Kopf hängen. Das Ziel ist es den User durch das Design nochmals auf die Seite zu locken oder noch besser die Webseite mit anderen zu teilen. Jeder Besucher, der auf der Webseite nicht zurechtfindet, klickt auch deswegen schnell weg. Deshalb, ist auch eine gute Usability notwendig. Usability ist die Benutzerfreundlichkeit auf der Seite. Das Ziel der Usability ist es, dem User es so einfach wie möglich zu machen sich auf der Webseite zurecht zu finden und dass der User sich auskennt.

6.2.2 Farben

Die Farben auf unsere Webseite sind übergreifend zu unserem Spiel und unserem Logo angepasst, die alle dieselben Farben haben. Somit sind die Farben im Spiel, sowie auf der Webseite, als auch im Logo alle überlappend und zwar in verschiedenen Farbtönen.

6.2.3 Logo

Abbildung 17 Wraithknight's Logo

Das Logo ist einer der wichtigsten Elemente eines Produktes. Das Logo ist essentiell bei Firmen, Produkte, Webseiten und vieles mehr, denn das Zeichen bleibt nach einer Zufriedenheit lange in Gedanken. Mit dem Logo kann etwas Bestimmtes damit identifiziert werden.

6.2.3.1 Entstehungsgeschichte

Das Logo war schon im Vorhinein klar. Das Ziel war es mit dem Logo ein kühles Seelenfeuer zu erstellen, denn jeder unserer Charaktere ist mit dem Seelenfeuer gefüllt, denn die Seele lebt in unserem Spiel immer weiter. Der Effekt vom Logo, zeigt Kühle, Kälte und auch das Untote in sich.

7 Software

Alle Softwarekomponenten des Spiels wurden in C# innerhalb des Monogame Frameworks geschrieben.

7.1 Monogame Framework

Abbildung 18 Monogame Logo Quelle: https://upload.wikimedia.org/wikipedia/commons/e/e6/MonoGame_Logo.svg

Monogame ist eine Open-Source-Weiterführung vom von Microsoft entwickelten XNA' Framework. Es basiert auf der Programmiersprache C# und unterstützt DirectX als auch OpenGL.

- DirectX ist eine von Microsoft entwickelte Grafik-API. Sie wird von Windows-Systemen und Xbox-Konsolen unterstützt.
- OpenGL ist eine Open-Source Grafik-API. Sie ist plattformübergreifend und kann für Windows, macOS und Linux Systeme verwendet geben.

Diese Bibliotheken werden vom Monogame Framework zum Kommunizieren mit der Grafikkarte verwendet. Beim Erstellen eines Projektes muss man sich für eines dieser zwei entscheiden.
Für dieses Projekt bietet sich OpenGL am besten an, da wir von keinen relevanten Einschränkungen betroffen sind und wir dann mit unserem Produkt mehrere Plattformen gleichzeitig ansprechen können.

Eine neu erstellte Monogame-Vorlage sieht folgendermaßen aus:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace Game1
{
    public class Game1 : Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        protected override void Initialize()
        {
            base.Initialize();
        }

        protected override void LoadContent()
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);
        }

        protected override void UnloadContent()
        {
```

```
        }


        protected override void Update(GameTime gameTime)

        {

            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape)) Exit();


            base.Update(gameTime);

        }


        protected override void Draw(GameTime gameTime)

        {

            GraphicsDevice.Clear(Color.CornflowerBlue);


            base.Draw(gameTime);

        }

    }

}
```

Codebeispiel 1 Monogame Vorlage

`Initialize()`	wird für das Laden aller Elemente verwendet, welche nicht von der Pipeline geladen werden. (Services, Spielstände, Datenbanken)
`LoadContent()`	wird für das Laden aller Elemente verwendet, welche von der Pipeline geladen werden. (Grafiken, Sounds, Schriften)
`UnloadContent()`	wird für das Entladen aller Elemente verwendet, welche von der Pipeline geladen wurden. Dies befreit das verbrauchte RAM.
`Update()`	wird für das Aktualisieren der Spiellogik verwendet.
`Draw()`	wird für das Zeichnen mit der Grafikkarte verwendet.

`Initialize()` und `LoadContent()` werden immer beim Programmstart aufgerufen.
`UnloadContent()` wird beim Programmende aufgerufen.
`Update()` und `Draw()` werden ständig aufgerufen und bilden die GameUpdateLoop.

Beim Ausführen dieses Programms wird ein Bildschirm mit einem hellblauen Hintergrund erstellt. Beim Drücken der ESC-Taste oder dem Menüknopf eines Controllers wird das Fenster geschlossen. Diese Vorlage ist alles, was Monogame von uns an Arbeit entnimmt. Monogame verleiht uns ein stabiles Grundgerüst, worauf wir unser Spiel basieren können. Alles andere muss selbst entwickelt werden.

7.1.1 UpdateLoop

Die Funktionen `Update()` und `Draw()` bilden die UpdateLoop, welche jede Bildwiederholung, also für jedes Frame, aufgerufen wird. Logik-Code ist hier vom Grafik-Code getrennt. Dies ist best practice und hat auch technische Gründe.

`Update()`

In der Update-Funktion wird der Logik-Code ausgeführt. Dieser beinhaltet so ziemlich alles, was nicht direkt auf den Bildschirm gezeichnet werden muss. Jeder zeitabhängige Code verwendet die DeltaTime zur Berechnung von Zeitdifferenzen. Der Großteil des Codes wird sich hier befinden.

`Draw()`

In der Draw-Funktion wird der Code zum Zeichnen ausgeführt. Dies wird mithilfe eines Spritebatches erledigt. Ein Spritebatch ist eine Ansammlung von Befehlen, die an die Grafikkarte am Ende eines Updateloops geschickt werden. Diese werden innerhalb eines Batchs organisiert, um die Optimierung von z.B. sich wiederholenden Grafiken zu ermöglichen.

7.1.2 DeltaTime

Wenn man z.B. Bewegung simulieren möchte, benötigt man einen Bezug zu der vergangenen Zeit. Diese Funktionalität gibt uns das `GameTime` Objekt, welches von Monogame bereitgestellt und verwendet wird.

Das Objekt `GameTime` besteht aus Folgenden Attributen:

- `TimeSpan TotalGameTime` Die vergangene Zeit seit Spielstart
- `TimeSpan ElapsedGameTime` Die vergangene Zeit seit letztem `Update()`
- `Bool IsRunningSlowly` Ob das Spiel unter der festgelegten Frequenz rennt

Bei der Berechnung von Zeit gibt es zwei Herangehensweisen:

7.1.2.1 Fixe Wiederholrate

Bei der fixen Wiederholrate wird die Frequenz, mit der sich das Spiel selbst aufruft, limitiert. Sie erlaubt einfachere Kalkulation von Zeitdifferenzen, da diese im Idealfall immer gleich sind. Wenn man sein Spiel auf 60fps limitiert, wäre ein Zeitsprung zwischen zwei Bildern immer 0.0166s (1/60s) lang. Damit umgeht man Berechnungen mithilfe von `ElapsedGameTime`. Wenn die Wiederholrate durch Performanceprobleme unter diese

Schwelle fällt, verhält sich die Spiellogik normal und weißt kein außerordentliches Verhalten auf, da die Zeitdifferenz statisch definiert wurde.

Die Vor- und Nachteile diese Methode sind die folgenden:

- ❖ Zeitberechnung statisch und Logik-Code ist vorhersehbarer
- ❖ Performance-Schwankungen oberhalb der Schwelle werden abgeschnitten
- ❖ Das System überarbeitet sich nicht, was sinnvoll bei Laptops und Mobilgeräten ist, oder bei limitierter Stromversorgung
- Monitore mit Bildfrequenzen über der Schwelle werden nicht ausgenutzt, kommt negativ bei der Kundschaft an
- Unbrauchbar für Online Multiplayer – durch verschiedene Hardware würde es zu Desynchronisierungen kommen

Diese Methode ist mittlerweile veraltet, wird aber dennoch bei kleineren Projekten gerne verwendet.

7.1.2.2 Variable Wiederholrate

Bei der variablen Wiederholrate wird die Frequenz nicht limitiert und hat somit kein Maximum, das sie erreichen kann. Für die Berechnung der Zeit wird die GameTime verwendet. Die seit dem letzten Frame vergangene Zeit kann mit `(float)gameTime.ElapsedGameTime.TotalSeconds` abgefragt werden. Diese ist in Sekunden angegeben und muss als Faktor mit anderen Zahlen multipliziert werden.
Die Vor- und Nachteile diese Methode sind die folgenden:

- ❖ Wird als State of the Art von Konsumenten gesehen
- ❖ Nutzt die Hardware vollkommen aus
- Bei extremen Performance-Schwankungen können Logikprobleme entstehen (siehe Movement)

7.2 Architektur

Da keine etablierte Engine für die Entwicklung des Spiels verwendet wurde, musste diese auch selber gemacht werden. Dies ist normal für Entwickler, die nur ein Framework verwenden.

Unter einer Game Engine versteht man ein wiederverwendbares Stück Software, welches auf der Basis von Daten agiert. Dadurch kann eine Engine bei mehreren Projekten eingesetzt werden, da sich diese nur in den Daten unterscheiden und die Logik gleichbleibt.

Kommerzielle Game Engines beinhalten meist Entwickler-Tools mit graphischer Oberfläche. Bei größeren Teams erlaubt das eine erweiterte Zusammenarbeit, ohne dass sich jeder mit der Programmiersprache auskennen muss.

Da das Wraithknight-Team aber nur einen Programmierer hat, sind diese Funktionalitäten nicht nötig.

7.2.1 ECS

Das **E**ntity **C**omponent **S**ystem Design Pattern setzt Komposition über Vererbung.
Vom Konzept her ist es das polare Gegenteil vom klassischen **O**bjekt-**O**rientierten **P**rogrammieren. ECS wird meist in der Spieleentwicklung verwendet, da gerade dort die Schwächen von OOP sehr früh zu Problemen führen.

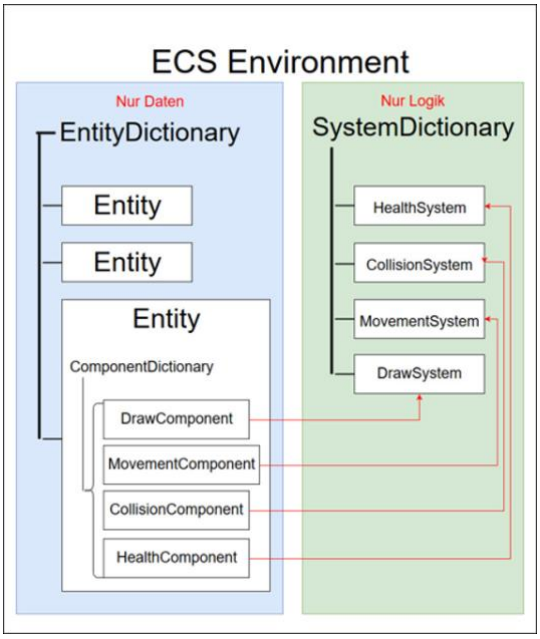

Abbildung 19 ECS Architektur

Die ECS-Architektur befindet sich aber nur innerhalb eines vom Programmierer definierten ECS Environments. Im Falle von Wraithknight wäre das das tatsächliche Spiel. Das ECS Environment übernimmt Aufgaben wie das Entity-Management (das Löschen/Erstellen von Entitäten), das System-Management, sowie die verschiedenen Boot-Routinen[12].

Einige Spielelemente, wie z.B. das Menu, benötigen das ECS nicht, und werden daher "klassisch" im OOP programmiert.

7.2.1.1 Das Vererbungsproblem

Dies ist nur ein Demonstrationsbeispiel und wurde nicht im Projekt Wraithknight angewendet.

Wir haben folgenden Hierarchiebaum im OOP:

[12] Eine Abfolge an Anforderungen die beim Spielstart erfüllt sein müssen (z.B. Levelgeneration)

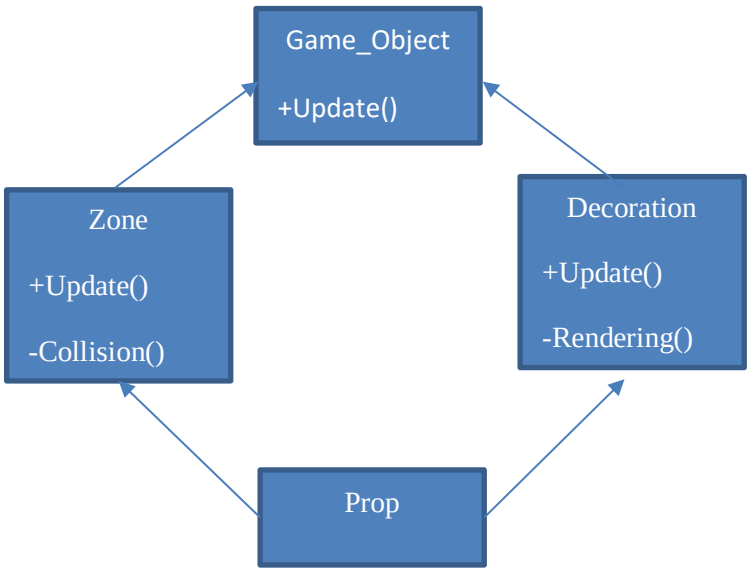

Abbildung 20 Deadly Diamond Szenario

Die `Zone` ist ein integrierbares, aber unsichtbares Objekt. Eine Triggerbox, welche eine Zwischensequenz auslöst, würde darunterfallen. `Decorations` sind nicht interagierbare Objekte, welche aber für den Spieler sichtbar sind, wie z.B. Büsche. `Props` sind interagierbare und sichtbare Objekte, diese wären zum Beispiel Bäume mit Kollision. Damit `Prop` dieses Verhalten erreicht, muss Logik von der `Zone` Klasse und der `Decoration` Klasse beinhaltet werden.

Mit der jetzigen Datenstruktur kann die `Prop` Klasse nur von einen der beiden Klassen vererben. Würde man von beiden vererben, hätte man einen Deadly Diamond. Beide Elternklassen vererben nämlich von der `Game_Object` Klasse und es entsteht ein Logikkonflikt. Beispielweise hat die Klasse `Game_Object` eine Funktion namens `Update()`, welche das Objekt jeden . `Zone`, als auch `Decoration` vererben diese Funktion und modifizieren sie. Wenn `Prop` jetzt von `Zone` und `Decoration` vererbt, welche modifizierte Version soll dann verwendet werden?

Wenn `Prop` dann nur von einer Klasse vererbt, muss die Logik von der anderen Klasse irgendwie in Prop untergebracht werden; man würde sich wiederholen.

Wenn man versucht den Code von den Unterklassen nach `Game_Object` zu verschieben, hat `Zone` Grafik-Code und `Decoration` Kollisions-Code. Es gibt keine saubere Möglichkeit die Logik von `Zone` und `Decoration` für Klassen wiederzuverwenden, welche von `Zone`, als auch von `Decoration` Inhalte übernehmen möchten.
Alle Alternativen weisen Mängel in der Architektur auf.

Jetzt derselbe Fall mit der ECS Architektur:

- Es gibt eine `CollisionComponent` und eine `DrawComponent`.
- Die Klasse `Zone` gibt es nicht mehr, sie ist ein `Game_Object` mit einer `CollisionComponent`.
- Die Klasse `Decoration` gibt es nicht mehr, sie ist ein `Game_Object` mit einer `DrawComponent`.
- Die Klasse `Prop` ist ebenfalls ein `Game_Object`, aber mit einer `CollisionComponent` und einer `DrawComponent`.

Es gibt keine Hierarchie Probleme, weil es keine Hierarchie mehr gibt. Die Entitäten werden durch ihre Komposition definiert, und nicht durch die Vererbung.

7.2.1.2 Entity

Ein Entity ist ein Container für eine beliebige Anzahl an Komponenten. In Wraithknight beinhalten Entities noch Metadaten, wie etwa:

- eine ID
- ein Boolean, ob sie noch am „Leben" ist
- das Team (Freundlich, Feindlich, Neutral)
- den Typ der Entität, der bei der Erstellung verwendet wurde.

Diese Metadaten sind für eine funktionelle ECS-Architektur nicht notwendig, erleichtern aber das Debugging.

7.2.1.3 Component

Eine Komponente ist ein Datensatz, der das Verhalten einer Entity beschreibt. Komponenten beinhalten KEINE SPIELLOGIK und sind mit einer Tabelle einer Datenbank zu vergleichen.
Eine Movement Component hätte z.B. eine (X/Y)-Position und eine Geschwindigkeit, womit die Bewegung ausgedrückt wird. Wenn eine Entity eine aktivierte `MovementComponent` besitzt, ist sie dazu fähig, sich zu bewegen und bewegt zu werden.

7.2.1.4 System

Systeme existieren meistens parallel zu einer Komponentenklasse. Sie führen eine Collection mit allen nötigen Komponenten, die sie überwachen. Innerhalb eines Systems findet man die Logik von den zugehörigen Komponenten.
Ein Movement System hätte eine Liste mit allen Movement Components, die gerade im Spiel vorhanden sind. Jedes Update wird über diese Liste iteriert und die Komponente aktualisiert. In unserem Beispielfall würde die Geschwindigkeit mit der vergangenen Zeit multipliziert werden und dann zu den Positionskoordinaten addiert werden.

7.2.2 Gebundene Komponente

Manchmal ist es nötig, dass eine Komponente über die Informationen einer anderen Komponente verfügen muss. In diesem Fall werden gebundene Komponenten verwendet.

Die Draw Component-Klasse beinhaltet Informationen über das Zeichnen einer Grafik. Das Draw System ist derzeit nur dafür verantwortlich, die Draw Component zu zeichnen. Wenn sich eine Entity aber bewegt, d. h. die Position in der Movement Component wird geändert, kriegt die Draw Component davon nichts mit und wird immer noch auf der alten Position gezeichnet. In dem Fall binden wir die Movement Component an die Draw Component. Dann hat das Draw System Zugriff auf die tatsächliche Position einer Entity, nur indem es eine Draw Component nach ihren Bindings fragt.

7.3 Assetmanagement

Die als PNG exportierten Grafiken müssen in die Spiellogik eingebunden werden. Dabei hilft der Monogame Content-Manager, welcher jedes Asset in eine XNA-Datei umwandelt. Innerhalb des Programmes unterscheiden sich die Ressourcen aber in der Klasse.

7.3.1 Monogame Content-Manager

Der Monogame Content-Manager greift auf Dateien innerhalb des „Content“-Ordners zu und konvertiert diese auf XNA-Dateien um. Diese Konvertierung erfolgt aber nur bei der Kompilierung des Programms. Ressourcen, welche vom Content-Manager geladen wurden, befinden sich nun in der Pipeline.

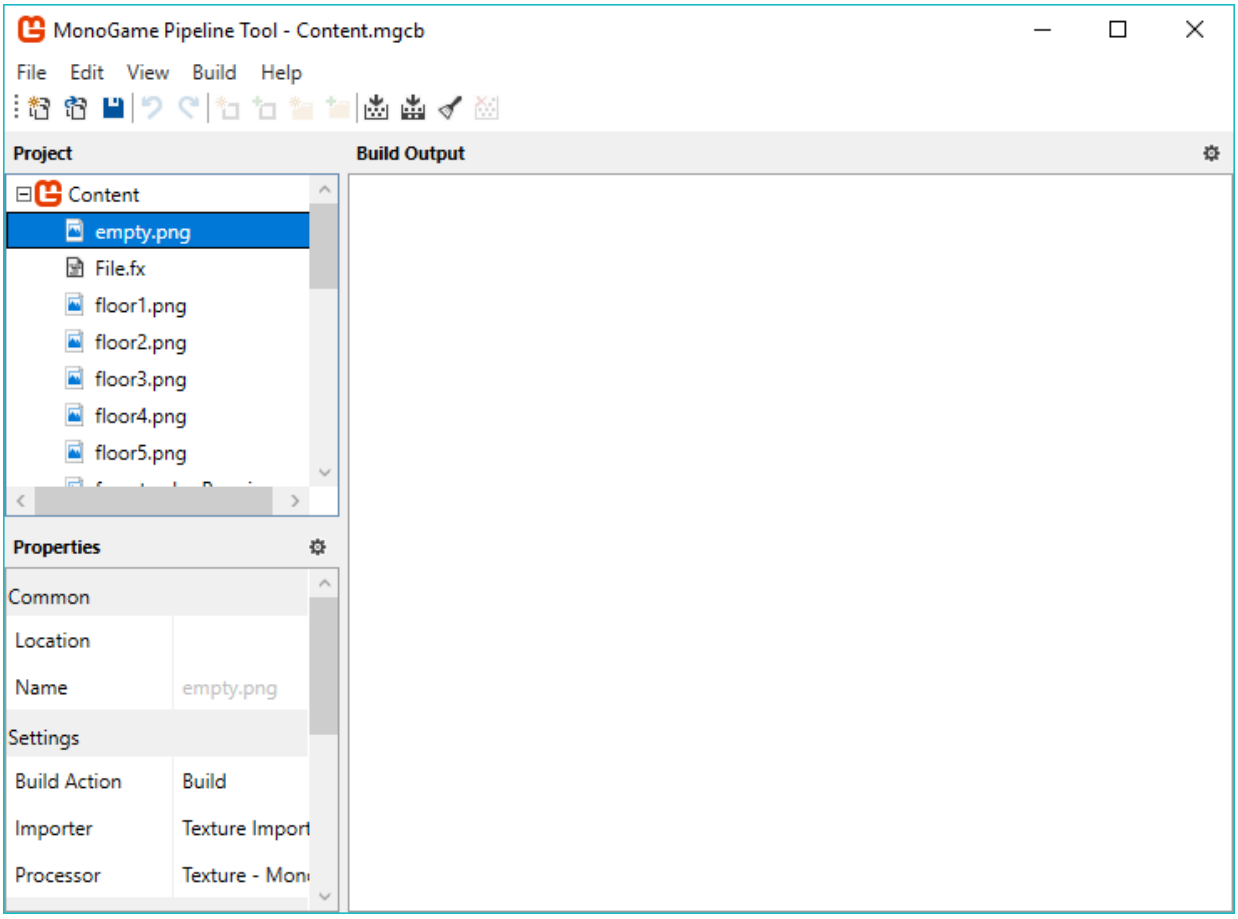

Abbildung 21 Monogame Content-Manager

7.3.2 Dynamische Asset-Bibliothek

Bei größeren Projekten verliert man schnell die Übersicht über eingebundene Ressourcen, wenn man keine ordentliche Verwaltung betreibt.

Die Bibliothek ist eine global erreichbare statische Klasse. Sie führt jeweils eine Collection für eine Art von Ressource. Auf die Bibliothek kann mit Asset.GetTexture() zugegriffen werden.

Eine weitere Funktion der Bibliothek ist das automatische Entladen von nicht mehr verwendeten Assets. Diese Funktionalität ist vor allem bei Spielen mit hohen Grafikauflösungen, somit auch großen Dateien, relevant. In diesem Projekt sind die verwendeten Grafiken von sehr geringer Größe, weshalb das automatische Entladen keinen technisch relevanten Nutzen hat, sondern nur dank der Automatisierung dem Entwickler etwas Arbeit abnimmt.

7.4 Entity-Erstellung

Entitäten bestehen aus einer Liste von Komponenten und einigen Metadaten, welche die Verwaltung vereinfachen.

Die Entitäten eines bestimmten Typs bestehen aber immer aus den gleichen (oder zumindest ähnlichen) Komponenten. Ein Baum besteht immer aus einer Draw Component und einer Collision Component. Bei der Generierung einer Entität werden der Typ und ein paar Startvariablen bekannt gegeben.

Nachdem die Entität mit den Komponenten gefüllt wurde, werden die Metadaten in Relation zu den Inhalten gesetzt. Dies finalisiert die Generierung. Jetzt müssen die Komponenten noch den Systemen bekannt gegeben werden. Jedem System wird eine Liste von zu registrierenden Komponenten gegeben und sie suchen sich die jeweilig relevanten Komponenten aus und verarbeiten sie.

Um eine Entität zu generieren, wird die statische Klasse `ECS_CreateEntity` verwendet. Ihr einziger Inhalt ist die `CreateEntity()` Methode, welche den Typ und ein paar Initialvariablen als Parameter übernimmt.

```
public static Entity CreateEntity(EntityType type, Vector2Ref position = null, Coord2?
speed = null, GameTime gameTime = null, Allegiance allegiance = Allegiance.Neutral)
```

Codebeispiel 2 Entity-Erstellung

Das Hauptmerkmal einer Entität ist die `EntityType`, deswegen wird innerhalb dieser Methode in einem Switch zwischen den Typen unterschieden. Abhängig vom angegebenen Typ wird die Entität mit den jeweils nötigen Komponenten gefüllt.

```
case EntityType.Hero:
{
	entity.SetAllegiance(Allegiance.Friendly);
	entity.SetStateComponent();
	entity.AddComponent(new MovementComponent(accelerationBase: 1250, maxSpeed: 175,
	friction: 800, position: safePosition));

	entity.AddComponent(new AttackBehaviorComponent(new List<AttackComponent>()
	{
		new AttackComponent(EntityType.HeroKnightSlashWeak, AttackType.Primary,
		entity.GetComponent<MovementComponent>().Position, new Vector2(0, 20),
		posOffsetInDirection: 35, startSpeed: 200, attackState: 0,
		attackDelayMilliseconds: 100, attackCooldownMilliseconds: 500,
		blockInputDurationMilliseconds: 250, selfKnockback: -150),

		new AttackComponent(EntityType.HeroKnightThrowingDagger, AttackType.Secondary,
		entity.GetComponent<MovementComponent>().Position, new Vector2(0, 20),
		posOffsetInDirection: 25, startSpeed: 1100, attackState: 0,
		attackCooldownMilliseconds: 250, blockInputDurationMilliseconds: 100)

	}, entity.GetComponent<MovementComponent>().Position));

	entity.AddComponent(new IntelligenceNode(EntityType.Hero,
	entity.GetComponent<MovementComponent>().Position));

	entity.AddComponent(new DrawComponent(Assets.GetTexture("heroIdle"), drawRec: new
	AABB(0, 0, 100, 100), boundPos: entity.GetComponent<MovementComponent>().Position,
	offset: new Vector2(0, -8)), typeof(MovementComponent));

	entity.AddComponent(new AnimationComponent(AnimationStructures.GetAnimationList(type)),
	typeof(DrawComponent));

	entity.AddComponent(new HealthComponent(20), typeof(MovementComponent));

	entity.AddComponent(new CollisionComponent(collisionRectangle: new AABB(safePosition,
	new Vector2(16, 16)), offset: new Vector2(20, 40), isPhysical: true), new List<Type> {
	typeof(MovementComponent), typeof(HealthComponent) });

	entity.AddComponent(new BlinkComponent(3, 2000, 1000, 125), new List<Type> {
	typeof(InputComponent), typeof(MovementComponent), typeof(AttackBehaviorComponent),
	typeof(CollisionComponent), typeof(HealthComponent), typeof(DrawComponent) });

	entity.AddComponent(new InputComponent(true), new List<Type> {
	typeof(MovementComponent), typeof(AttackBehaviorComponent), typeof(BlinkComponent) });
```

Codebeispiel 3 Entity Erstellung

Die Funktion `CreateEntity()` definiert die Daten, laut welchen alle Entitäten erstellt werden. Hier werden die Komponenten eingefügt, Komponenten miteinander verknüpft und Metadaten erstellt. Sind alle Komponenten eingefügt und der Switch-Block verlassen, gilt die Entität als fertig generiert. Jetzt wird sie noch allen Systemen überwiesen, welche dann alle relevanten Komponenten für sich selbst ermitteln und registrieren.

Die Funktion `CreateEntity()` ähnelt einer Datenbank. In Zukunft kann die Funktion auch zu einer Datenbank exportiert werden. Im Idealfall erlaubt das dem User, Spiel-Modifizierungen zu erstellen.

7.5 Input

Unter Input versteht man jedes vom User erstelltes Signal, welches der Computer verarbeiten soll. In der Spieleentwicklung umfasst das hauptsächlich die Maus, die Tastatur und gegebenenfalls den Kontroller. In letzter Zeit tauchten auch andere Input-Möglichkeiten auf, wie zum Beispiel Sprachsteuerung oder Eyetracking, diese befinden sich aber noch in einer Nische oder helfen, behinderten Spielern die Erfahrung zugänglicher zu machen.

In diesem Projekt wird Userinput nur per Keyboard und Maus umgesetzt.
Um das Verwalten von Userinput sorgt sich die InputReader-Klasse.

7.5.1 Keyboard

Monogame hat ein eingebautes Enum namens `Keys` für alle verwendbaren Tastenanschläge.

```
public static bool IsKeyPressed(Keys key)
{
	return _currentKeyboardState.IsKeyDown(key);
}
```

Codebeispiel 4 Keybord

Mit der obigen Funktion kann man abfragen, ob eine Taste auf dem Keyboard gedrückt ist.

Parallel zum `_currentKeyboardState` wird auch `_lastKeyboardState` abgespeichert. Damit kann man Abfragen wie

- Release: `_lastKeyboardState == true && _currentKeyboardState == false`
- Trigger: `_lastKeyboardState == false && _currentKeyboardState == true`
- Active: `_currentKeyboardState == true`

durchführen.
Mit diesen drei Inputabfragen hat man jeden relevanten Keyboardinput abgedeckt.

7.5.2 Maus

Die Position vom Mauszeiger wird als zweidimensionaler Vektor abgespeichert.
Auf die Maustasten wird, wie beim Keyboard, mit einem Enum zugegriffen.

Genauso wie bei dem Keyboard gibt es auch hier die `_currentMouseState-` und `_lastMouseState-`Variablen.
Die Abfragen funktionieren gleich wie beim Keyboard-Schema:

- Release: `_lastMouseState == true && _currentMouseState == false`
- Trigger: `_lastMouseState == false && _currentMouseState == true`
- Active: `_currentMouseState == true`

Auch hier hat man mit diesen drei Inputabfragen jeden relevanten Mouseinput abgedeckt.

7.6 Grafiken in Monogame

Wenn die Grafiken erfolgreich von der Monogame-Pipeline geladen wurden, kann auf sie mithilfe der Asset-Bibliothek zugegriffen werden. Eine Entität bekommt graphische Funktionalitäten mit Hilfe einer `DrawComponent`, welche die geladene Textur speichert. Soll diese dann animiert werden, bindet man eine `AnimationComponent` an die `DrawComponent`. Eine `DrawComponent` alleine weiß nichts über die eigentliche Position der zugehörigen Entität. Möchte man, dass diese sich mit der Entität mitbewegt, muss eine `MovementComponent` an die `DrawComponent` gebunden werden.

7.6.1 Simple Grafiken

Simple Grafiken bestehen nur aus einer einzigen Textur und einer einzigen `DrawComponent`. Sie werden hauptsächlich bei stationären Entitäten wie Kulissen oder Bäumen eingesetzt.

7.6.2 Komplexe Grafiken

Komplexe Grafiken bestehen aus mehreren Texturen mit mehreren `DrawComponents`. Bei einer komplexen Grafik wären z.B. der Oberkörper und die Beine in jeweils separaten `DrawComponents`. Dies ermöglicht auch separate Animation von den einzelnen `DrawComponents`.

Sie finden derzeit keinen Einsatzbereich in der Diplomarbeit, aber die Systeme dazu sind funktionsfähig.

7.6.3 Simple Animationen

Animationen sind eine Abfolge von Frames (Texturen). Jeder Frame hat seine selbst definierte Anzeigezeit. Die Anzeigezeit gibt an, wie lange ein Frame in der Animation sichtbar ist. Ist die Anzeigezeit abgelaufen, wird der nächste Frame der Animation geladen. Ist der letzte Frame abgelaufen, gilt die Animation als beendet.

Animationen werden generell durch bestimmte Trigger ausgelöst. Eine `AnimationComponent` kann die nötigen Trigger aus der `StateComponent` ablesen. Events werden zurzeit nicht verwendet, da diese weniger Kontrolle über komplexes Zustandsverhalten bieten. Jedes System kann potentiell einen Trigger setzen, weshalb die `StateComponent` eine besondere Komponente ist, denn jede andere Komponente hat direkten Zugriff auf sie, wenn sie in einer Entität definiert ist.

Eine Animation hat auch die „nächste“ Animation abgespeichert. Diese kann auf sich selbst verweisen (womit die Animation sich wiederholt) oder auf eine andere Animation verweisen (womit man Animationen „aneinander ketten“ kann, um komplexe Animationen zu bilden).

7.6.4 Komplexe Animationen

Komplexe Animationen sind mehrere aneinander gekettete simple Animationen. Eine komplexe Animation kommt bei der Laufanimation des Helden vor. Wenn er zu laufen beginnt, wird ein Trigger gesetzt, welcher den Übergang in die Laufanimation auslöst. Ist der Übergang fertig, wird der Run-Cycle geloopt. Hört der Held wieder auf zu laufen, wird wieder ein Trigger gesetzt, welcher die Animation für die Abbremsung auslöst.

Abbildung 22 Komplexe und simple Animationen

7.7 Bewegung

Entities, die sich bewegen können, besitzen eine Movement Component. Diese ist auch im Movement System registriert und wird von diesem verwaltet.
Bewegung in Spielen kann sehr komplex werden, mit vielen Nuancen, um sie möglichst realistisch oder „gut anfühlend“ zu machen. Der Spaß von Platformern (nicht unser Genre) kommt hauptsächlich von der Bewegung und Physik. Es fühlt sich gut an, ein physikalisches Objekt mit Gewicht und Momentum durch die Welt zu bewegen.

Da das Spiel zweidimensional ist, wird die Bewegung auch nur in zwei Dimensionen simuliert, in der X- und Y-Achse.

7.7.1 Vektoren

Das Monogame Framework stellt eine `Vector2` Struct zur Verfügung, welche kartesische, zweidimensionale Koordinaten darstellt. Diese eignen sich gut zu der Repräsentation einer Position, aber nicht so gut für Geschwindigkeiten.

Würde man bei Geschwindigkeiten kartesische Koordinaten verwenden, würde es einem schwerfallen, die Geschwindigkeit mit der Zeit abnehmen zu lassen. Im Spiel ist das aber

ein gewünschter Effekt. Der Boden, auf dem sich eine Entität bewegt, bremst sie durch Reibung ab, bis sie zum Stillstand kommt. Bei kartesischen Koordinaten wäre das leicht umzusetzen: Bei jedem `Update ()` wird die X- und Y-Koordinate mit der Abbremsung subtrahiert. Die zwei Achsen sind hier aber voneinander unabhängig, was heißt, dass eine Achse vor der anderen auf 0 abgebremst werden kann. Dieser Ansatz funktioniert bei 2D-Sidescrollern, wie dem klassischen Super Mario. Hier schaut man in einem 90 °-Winkel auf eine waagrechte Spielfläche. Die Y-Achse beschreibt die vertikale Höhe über dem Boden.

Bei Top-down-Spielen, wie unserem, definieren die Achsen positionelle Koordinaten, wie bei dem Spiel „Schiffe versenken“. Um hier eine realistische Abbremsung zu verwirklichen, muss man die kartesischen Koordinaten zuerst in Polarkoordinaten umwandeln und dann die Länge des Vektors mit der Abbremsung verkürzen. Hierfür wurde `Coord2` Struct selbst erstellt, welche aus einem kartesischen `Vector2` und einem polaren `Polar2` (auch selbst erstellt) besteht. Die Achsen von `Coord2` können nur durch die Funktionen manipuliert werden, welche automatisch zwischen polaren und kartesischen Koordinaten umwandeln.

7.7.1.1 Kartesisch

Kartesische Koordinaten bestehen aus einer X- und Y-Koordinate. Das von Monogame vorgegebene `Vector2`-Struct bietet auch einige arithmetische Funktionen, welche die spielrelevanten Berechnungen vereinfachen.
Kartesische Koordinaten eignen sich für die Darstellung von Positionen.

7.7.1.2 Polar

Die polare Implementierung ist selbst gemacht und wurde nicht vom Framework vorgegeben. Sie besteht aus einem Winkel und einer Länge.
Polare Koordinaten eignen sich für die Darstellung von Bewegungen.

7.7.2 Beschleunigung und Abbremsung

Die größte Herausforderung bei dem Entwurf eines guten Bewegungssystems besteht darin, dass sich die Bewegung realitätsnah und befriedigend anfühlt.

Der Userinput kann direkt die Geschwindigkeit des Hauptcharakters beeinflussen. Dies ist der simpelste Ansatz und wird gerne bei Arcade-Style Spielen, wie Pacman, verwendet.

Doch in der realen Welt beschleunigen Objekte nicht sofort auf die gewünschte Geschwindigkeit, denn sie besitzen eine Trägheit. Um dieses Verhalten in der virtuellen Welt zu simulieren kontrolliert der Userinput stattdessen die Beschleunigung des Charakters, welche dann auf die Geschwindigkeit wirkt. Hier muss man auf die Feinheiten der Bewegung achten, damit der Spieler nicht ein „schwammiges“ Feedback wahrnimmt.

Ein gut umgesetztes Moveset trägt eine Menge zum *Game feel*[13] bei.

[13] *Game feel* ist das User Experience vom Spiel. Es besagt, wie das Spiel sich für den Spieler „anfühlt“.

7.8 Kollision

Wenn zwei physikalische Objekte einander berühren, kommt es zu einer Kollision. Wie damit dann umgegangen wird, bezeichnet man als „Collision Response“.

Das Objekt, welches sich in Bewegung befindet und die Kollision auslöst, nennt man „Actor“. Das Objekt, welches von der Kollision betroffen wird, nennt man „Target“.

Im üblichen Fall von physischen Objekten kommt es zu einer Blockade. Dabei behält das stillstehende Objekt, das Target, seine Position, und das bewegende Objekt, der Actor, muss seine Position und Bewegung aktualisieren. Wenn zwei bewegende Objekte kollidieren, wird ein zufälliges als Actor gewählt.

Logische Objekte können ebenfalls miteinander kollidieren, wie zum Beispiel ein Projektil mit einem Gegner. Hier wird eine physische Response allein nicht ausreichen, denn der Gegner muss durch das Projektil auch noch Schaden nehmen. In diesem Fall muss zusätzlich eine logische Response ausgelöst werden, mit dem Projektil als Actor und dem Gegner als Target. Bei Bedarf kann immer noch eine physische Response durchgeführt werden.

7.8.1 AABB

Die **A**xis-**A**ligned-**B**ounding-**B**oxes-Technik ist das simpelste Prinzip zu der Berechnung von Kollisionen. Physische Objekte werden als Boxen dargestellt. Eine Box besteht aus einer Position und den Dimensionen. Die Boxen sind nicht rotationsfähig, da, wie der Name sagt, sie an die Achsen ausgerichtet sind.

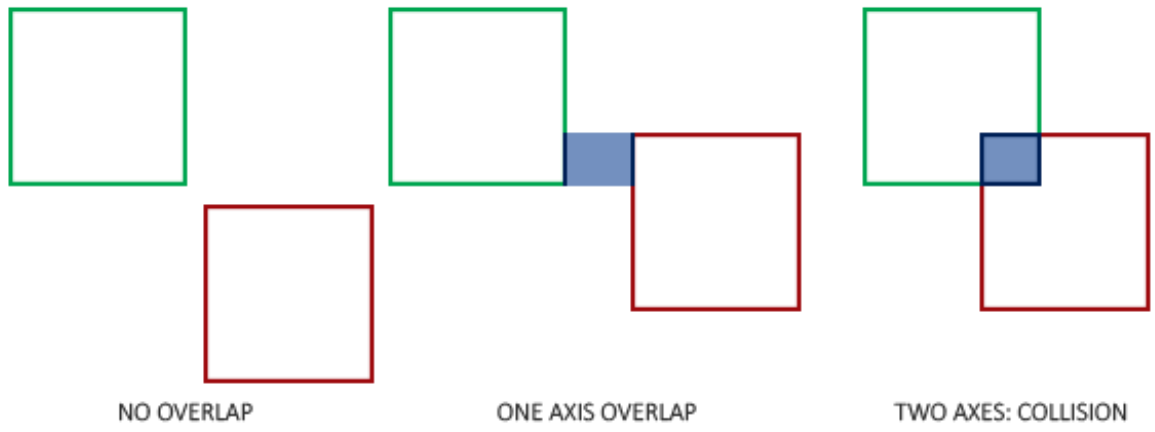

Abbildung 23 AABB Kollision
Quelle: https://learnopengl.com/img/in-practice/breakout/collisions_overlap.png

Ob zwei Boxen miteinander kollidieren, kann man mit folgender Formel berechnen:

```
if(A.Left < B.Right && A.Right > B.Left && A.Top < B.Bottom && A.Bottom > B.Top)
```

A =	*AABB*
B =	*AABB*
Left =	*X*
Right =	*X + Width*
Top =	*Y*
Bottom =	*Y + Height*

Tabelle 4 Bezeichnung der AABB Kollision Formel

Diese Formel stellt nur fest, ob zwei AABBs einander schneiden. Wie damit umgegangen wird, wird nicht beschrieben.

7.8.2 Minkowski-Summe

Die Minkowski-Summe ist eine mathematische Formel, welche bei der Berechnung von geometrischen Körpern verwendet wird. Dabei werden zwei Formen miteinander addiert, um die Summenform zu bilden. Dies ist einer der einfachsten Algorithmen, um den Penetrations-Vektor einer Kollision zu berechnen. Die Summenform kann gegen den Punkt $(0/0)$ getestet werden. Falls sich der Punkt in der Form befindet, besteht eine Kollision.

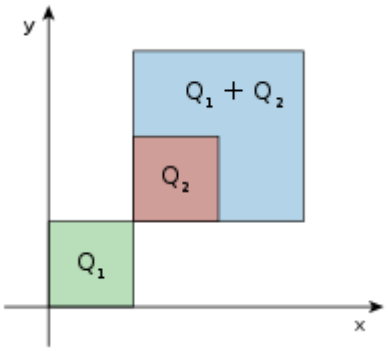

Abbildung 24 . Minkowski-Summe von 2 Quadraten
Quelle:
https://en.wikipedia.org/wiki/Minkowski_addition#/media/File:Minkowski_sum_graph_-_vector_version.svg

Falls eine Kollision besteht, kann man auch die kürzeste Distanz nach außen, den Penetrations-Vektor, berechnen. Der Penetrations-Vektor ist die relative Distanz von (0/0) bis zu der nähesten Seite.

Wenn man den Penetrations-Vektor berechnet hat, kann dieser zu der Position des Actors addiert werden, um den Actor aus der Kollision herauszubewegen.

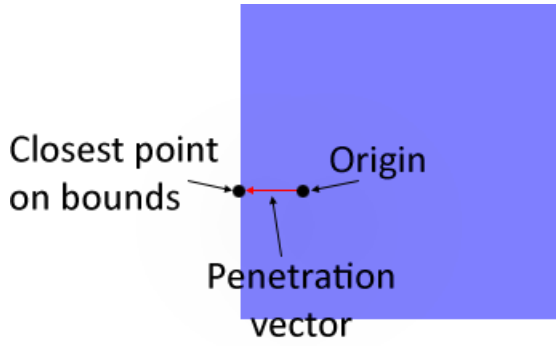

Abbildung 25 Penetrationsvektor
Quelle: https://blog.hamaluik.ca/assets/images/simple-aabb-collision-using-minkowski-difference/penetration_vector.png

7.8.3 Collision Response

Wenn eine Kollision erkannt wurde, muss sie irgendwie gelöst werden. Eine Kollision zählt als gelöst, wenn nach der Lösung die Kollision nicht mehr besteht. Auch wenn eine Kollision mathematisch korrekt gelöst wurde, könnte sie logisch Fehler beinhalten. In Wraithknight bestehen die Bedingungen dazu aber nicht, weshalb dies kein Problem ist.

7.8.3.1 Physische Kollision

Wenn ein physisches Objekt mit einem anderen physischen Objekt kollidiert, entsteht eine physische Kollision, welche eine physische Response benötigt.

Bei physischen Responses unterscheidet man zwischen den folgenden:

7.8.3.1.1 Block

Die üblichste Art von Response. Bei einer Kollision wird der Actor an die Grenzen vom Target gesetzt, und die Bewegung auf der Achse, auf welcher kollidiert wurde, wird auf 0 gesetzt.

7.8.3.1.2 Bounce

Bei einer Kollision wird der Actor an die Grenzen vom Target gesetzt, und die Bewegung auf der Achse, auf welcher kollidiert wurde, wird negiert.

7.8.3.1.3 Stick

Bei einer Kollision wird der Actor an die Grenzen vom Target gesetzt. Der Actor verliert dann die Kollisions-Komponente und bindet sich an die Bewegung vom Target. Sinnvoll bei Wurfmessern, welche in z.B. Gegnern stecken bleiben sollen.

7.8.3.1.4 Disappear

Bei einer Kollision wird der Actor verschwinden. Ohne einen Actor kann es keine Kollision mehr geben, somit gilt sie als gelöst. Sinnvoll bei Partikel-Effekten.

7.8.3.2 Logische Kollision

Wenn ein logisches Objekt mit einem anderen logischen Objekt kollidiert, entsteht eine logische Kollision, welche eine logische Response benötigt. Logische Kollisionen müssen nicht unbedingt gelöst werden, es muss nur auf sie reagiert werden.

Im Umfang der Diplomarbeit werden nur Projektile als logische kollisionsfähige Objekte verwendet. Im gröberen Umfeld der Spieleentwicklung findet man auch noch Trigger (Bereiche, welche auf Kontakt ein Event auslösen).

Um den folgenden Abschnitt zu verstehen, muss man sich zuerst die `ProjectileComponent` genauer anschauen. Für Kollisionen hat die `ProjectileComponent` folgende relevante Attribute:

`public int Power;`	Ist die „Lebensanzahl" eines Projektils.
`public int Damage;`	Ist der Schaden, welcher auf Kontakt verursacht wird.
`public bool IsPhasing;`	Besagt, ob ein Projektil durch andere logische Objekte „durchschwebt". Ein Schwertschlag kann mehrere Gegner gleichzeitig treffen, ein Pfeil nicht.

Bei logischen Responses unterscheidet man zwischen den folgenden:

7.8.3.2.1 Projektil auf Projektil

Kollidiert ein Projektil mit einem anderen Projektil, sollen diese sich gegenseitig beeinflussen. Es gibt zwei Arten von Projektilen:

- Phasing, z.B. Schwertschlag. Diese Projektile können mehre Entities treffen. Bei Schadensunterschuss
- Piercing, z.B. Pfeil. Diese Projektile können nur ein Entity treffen. Bei Schadensüberschuss penetrieren sie durch.

Projektil Phasing auf Projektil Piercing

7.8.3.2.2 Projektil auf Leben

Kollidiert ein Projektil mit einem „lebendem" Objekt (Held, Gegner), soll dieses Schaden nehmen.

Das Projektil versucht so viel Schaden wie möglich:

Wenn `Damage` kleiner ist als `Power`, hat das Projektil genug Kraft, um den vollen Schaden anzurichten. Dabei nimmt die Target `HealthComponent` den vollen `Damage` vom Actor als Schaden.

Ist `Power` kleiner als `Damage`, hat das Projektil nicht genug Kraft, um den vollen Schaden anzurichten. Deswegen nimmt das Target nur die übrige Power vom Actor als Schaden.

Das Projektil verliert so viel an `Power`, wie es Schaden gemacht hat. Dabei kann es nicht mehr Schaden machen, als das Target an Leben übrig hat.

7.9 Levelgeneration

Ein Markenzeichen vom Roguelike-Genre ist die prozedurale Levelgeneration. Es gibt viele Algorithmen für das Generieren von zufälligen Leveln. Eines der einfachsten ist Cellular Automata, das sehr anschauliche Ergebnisse liefert.

7.9.1 Cellular Automata

In Cellular Automata agiert jede Zelle selbstständig, abhängig von ihrer Umgebung. Die Zellen befinden sich in einem Gitter und beginnen in einem zufälligen Startzustand. Das System durchläuft dann mehrere Generationen, wo die Regeln angewendet werden.
Eines der bekanntesten Systeme ist „Conway's Game of Life".

Eine Zelle bestimmt ihren Inhalt in Abhängigkeit von ihren Nachbarn. Überschreiten die aktiven Nachbarn (=Wände) einer Zelle einen Grenzwert, wird die Zelle selbst aktiv. Fallen die aktiven Nachbarn unter einen anderen Grenzwert, wird die Zelle inaktiv. Die Regeln, nach denen eine Zelle ihren Inhalt bestimmt, werden vom Entwickler vorgegeben. Der Vorteil dabei ist, dass man Presets erstellen kann. Man kann problemlos zwischen den Presets wechseln, um z.B. die Schwierigkeit eines Levels zu erhöhen.

7.9.2 Finaler Algorithmus

Der Cellular Automata-Algorithmus kann frei modifiziert werden, um die generierten Level entsprechend anzupassen.

In unserem Fall ist die Levelgeneration in vier Phasen unterteilt.

7.9.2.1 Phase 1: Rohmaterial

In der Rohmaterial-Phase wird ein passendes Rohmaterial für den Cellular Automata-Algorithmus vorbereitet. Angefangen wird mit einem 50%-Rauschen. Man endet mit einem 100x100 Raster, wo 50% der Zellen Wände sind und 50% nicht.

Um den Rand herum wird dann zusätzlich noch ein 80%-Rauschen hinterlegt. Dieses extra Rauschen rundet die Kanten der Karte ab, damit sie um den Rand herum nicht abrupt endet.

Dies vollendet schon die Rohmaterial-Phase und die Cellular Automata Zyklen können angewendet werden.

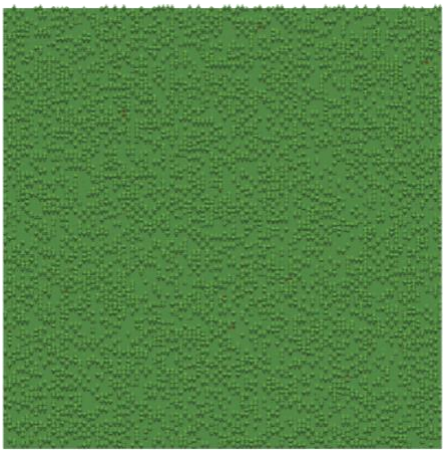

Abbildung 27 50% Rauschen

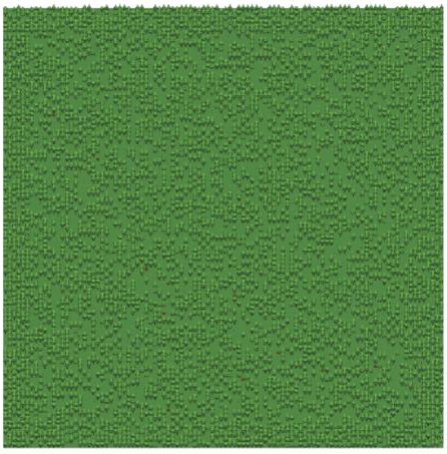

Abbildung 26 80% Rauschen am Rand

7.9.2.2 Phase 2: Cellular Automata Zyklen

In der Phase 2 wird die Karte aus der Phase 1 übernommen und mehrmals durch Cellular Automata behandelt. Die Regeln werden vom Entwickler definiert und können sich auch innerhalb der Generierung noch ändern. Jeder Zyklus erhöht die Generierungszeit, weshalb sie auf vier Zyklen limitiert sind.

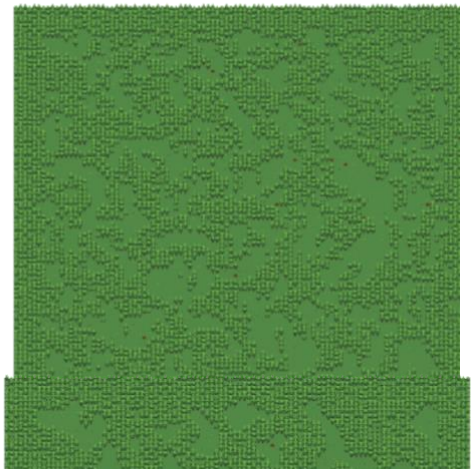

Abbildung 29 1. Zyklus

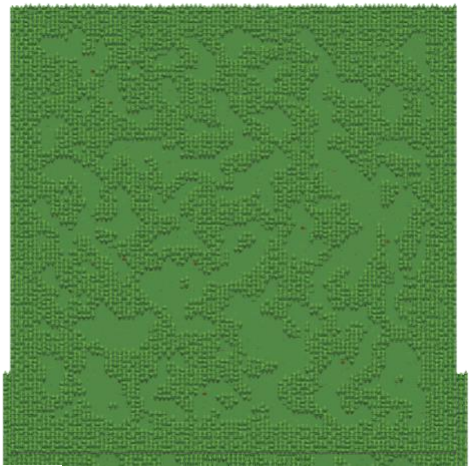

Abbildung 28 2. Zyklus

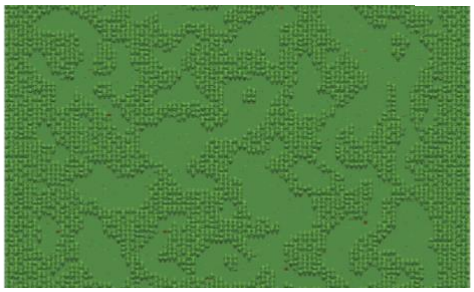

Abbildung 31 3. Zyklus

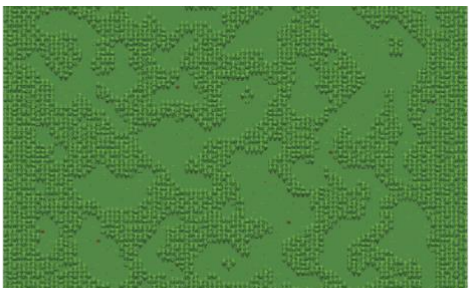

Abbildung 30 4. Zyklus

7.9.2.3 Phase 3: Map-Vorbereitung

Nach dem vierten Zyklus hat man bereits eine verwendbare Basis für die Spielumgebung. Doch sind viele der Räume voneinander getrennt und für den Spieler nicht begehbar. Man kann entweder die Räume miteinander verbinden oder überflüssige Räume löschen. Damit die Karte nicht zu groß wird, werden die überflüssigen Räume in diesem Algorithmus gelöscht.

Hierfür wird zuerst der größte Raum ausgesucht. Wurde der größte Raum identifiziert, werden alle kleineren mit Wänden gefüllt.

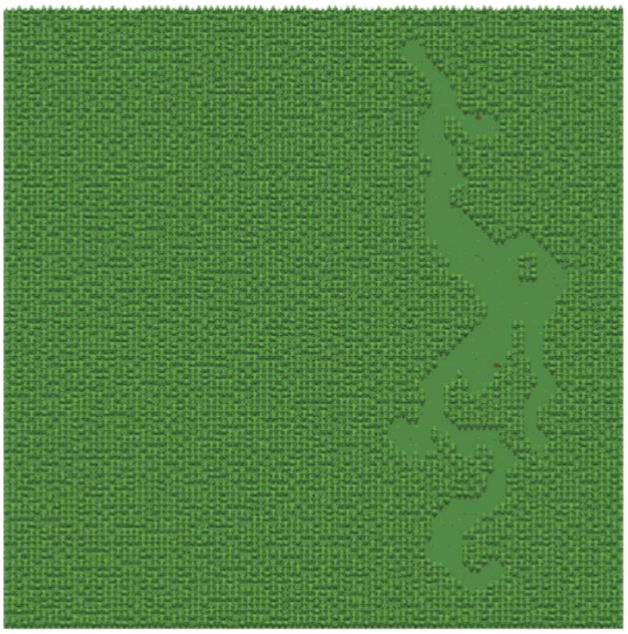

Abbildung 33 Räume wurden entfernt

Abbildung 32 Auf Dimensionen angepasst

Die Spielfläche ist jetzt fertig, aber es gibt noch viele unnötige Wände, welche die Performance des Spiels beeinflussen. Deren Texturen werden gezeichnet und Kollisionsboxen verarbeitet, obwohl sie nicht zum Spielerlebnis beitragen.

Man kann sie aber problemlos entfernen, indem man die Dimensionen des Raumes abmisst und das Level auf diese verkleinert.

7.9.2.4 Phase 4: Entität-Platzierung

Jetzt müssen nur noch die Spielfiguren (Gegner?) in das Level platziert werden. Hierfür wird ein Budget festgelegt. Die Höhe des Budgets wird vom Schwierigkeitsgrad festgelegt. Jede Spielfigur kostet einen bestimmten Wert. Wird die Einheit auf dem Level platziert, wird dieser Wert vom Budget abgezogen.

Ist das Budget aufgebraucht, so ist das Level fertig generiert und kann für das Spiel verwendet werden.

Abbildung 34 Entität Plazierung

7.10 Künstliche Intelligenz

Das Ziel von künstlicher Intelligenz ist es, realistisches Verhalten zu illusionieren. Simulieren ist in den meisten Fällen nicht notwendig, da vor allem in Computerspielen der Spieler von anderen Sachen abgelenkt ist. Um intelligentes Verhalten zu simulieren, braucht man auch die notwendige Rechenleistung dafür, weshalb man sehr sparsam mit künstlicher Intelligenz umgehen sollte.

7.10.1 Generelle Intelligenz

Jedes von einer Intelligenz wahrnehmbare Objekt wird mit einer `IntelligenceNode`-Komponente markiert. Eine intelligenzfähige Entität besitzt eine `IntelligenceComponent`-Komponente, in welcher sich Anweisungen für das Verhalten befinden

```
public IntelligenceOrder(EntityType target, int range, OrderType type, int priority, int updateCooldownMilliseconds, bool reset)
```

```
List<IntelligenceOrder> orders = new List<IntelligenceOrder>();
orders.Add(new IntelligenceOrder(EntityType.Hero, 100, OrderType.Attack1, 1, 1000, true));
orders.Add(new IntelligenceOrder(EntityType.Hero, 300, OrderType.Move, 0, 250, true));
```

Codebeispiel 5 Intelligenz Anweisungen

Die „Gegner Ritter"-Entität besitzt folgende Anweisungen:

- Wenn der Held 100 Einheiten von dir entfernt ist, attackiere ihn und aktiviere einen Cooldown von 1000 Millisekunden

  Wenn dies nicht der Fall ist, springe zur nächsten Anweisung

- Wenn der Held 300 Einheiten von dir entfernt ist, laufe auf ihn zu und aktiviere einen Cooldown von 250 Millisekunden

Der updateCooldownMilliseconds-Parameter bezieht sich nur auf die Intelligenz. Während dieser Cooldown abläuft, ist die Entität immer noch funktionsfähig, nur die Intelligenz-Aufrufe werden limitiert und die Entität trifft für diese Zeit keine Entscheidungen.

7.11 Gamescreens

Gamescreens sind, wie der Name sagt, „Spiel-Bildschirme". Es ist ein abstraktes Konzept, um Spielstadien zu beschreiben. Das Hauptmenü, Pausemenü und das eigentliche Spiel sind Gamescreens. Diese sind auf einem Stack organisiert, wo nur der oberste Screen aktualisiert wird. So kann man ein Pausemenü über ein bereits laufendes Spiel legen. Das Spiel erscheint immer noch im Hintergrund des Pausemenüs, doch es ist eingefroren und übernimmt keinen Input. Stattdessen ist das Pausemenü der aktive Screen, bis es wieder geschlossen wird.

7.12 Performance

Performance war schon immer ein wichtiges Thema bei der Entwicklung von Spielen. Mit der Zeit wurden immer mächtigere Spieleplattformen veröffentlicht, weshalb man bei der Entwicklung von kleineren Spielen jetzt mehr Toleranzraum hat. Bei der Entwicklung für die PC-Plattform bleibt es aber dennoch wichtig, das Produkt möglichst gut zu optimieren, da 10 Jahre alte Laptops auch noch als PC gelten und somit auch Teil der potentiellen Kundschaft sind.Die überwiegende Mehrheit von Performanceproblemen sind Logikfehler. Darunter fällt nicht geeignete Software-Architektur, schlechte Organisation von Daten oder ein ineffizienter Algorithmus. Natürlich hat jede Codezeile einen Einfluss auf die letztendliche Performance des Programms, aber wenn der Einfluss eines „Problems" so klein ist, dass man ihn zwischen Compiler-Optimierungen nicht erkennt, ist er vernachlässigbar. Da Performance Testing ein ständiger Vorgang ist, könnte man allein darüber ein ganzes Diplomarbeits-Buch schreiben. Aus Übersichtsgründen wurden daher die 3 Gebiete der Spiel-Optimierung hervorgehoben, mit je einem Anwendungsbeispiel.

7.12.1 CPU – WallCollision

Im Spiel müssen ständig Kollisionen geprüft werden. Dies ist ein Aufwand von $O(n^2)$ und steigert sich quadratisch mit der Anzahl an Kollisionsboxen. Vor allem die Kollisionsüberprüfung ist ein massiver Leistungsfresser, weshalb jede Optimierung in diesem Bereich langfristig die Performance verbessert. Eine Optimierungsmöglichkeit ist ein Collision Tree.

Ein Collision Tree unterteilt den nach Kollisionen geprüften Bereich in mehrere kleinere Bereiche. Der Actor befindet sich in einem dieser Unterbereiche und wird nur mit anderen Kollisionsboxen innerhalb dieses Unterbereiches geprüft. Um es anhand eines realen Beispiels zu erklären: Ein Auto in Wien muss nicht mit einer Wand in Graz geprüft werden. Die zwei Orte sind so weit voneinander entfernt, dass eine Kollision unmöglich ist. Diese Idee kann man dann weiter und weiter anwenden und die verkleinerten Unterbereiche in noch mehr, noch kleinere Unterbereiche aufteilen.Im Fall von Wraithknight sind alle Wände und Kulissen gleich groß und an einem Grid angelegt. Daher kann man sie ganz einfach in ein Array einspeichern. Durch die Position des Actors kann man die Position im Array berechnen und dann nur noch die acht Nachbarzellen überprüfen.

Die oben erwähnte Implementierung vermindert den Rechenaufwand auf O(1).

Diese Optimierung hat die Wiederholfrequenz um ~40fps gesteigert.

7.12.2 GPU – SpriteBatching

Das Zeichnen auf dem Bildschirm wird von der Grafikkarte umgesetzt. Damit die Grafikkarte eine Textur zeichnen kann, muss sie zuerst in den Speicher geladen werden. Dies verbraucht Zeit und muss für jede Grafik, jedes Update durchgeführt werden. Im schlimmsten Fall muss jedesmal, wenn eine Textur gezeichnet wird, die alte entladen und die neue geladen werden .Diesen Prozess kann man optimieren, indem man alle Grafiken mit derselben Textur direkt hintereinander zeichnet. Wenn die Textur gewechselt wird, werden alle Grafiken nacheinander gezeichnet, welche die neue Textur verwenden. Hiermit minimiert man das Laden und Entladen der Grafikkarte und erlaubt ihr, sich mehr auf ihre eigentliche Aufgabe zu konzentrieren, nämlich Grafiken auf den Bildschirm zu zeichnen.
Diese Optimierung hat die Wiederholfrequenz um ~5fps gesteigert.
Der Effekt ist größer bei schlechteren Grafikkarten.

7.12.3 Memory – Referenz Garbage Collector (GC)

Quelle: https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/
Eine Fähigkeit von C++ ist die manuelle Verwaltung des Arbeitsspeichers. In C# kümmert sich der Garbage Collector automatisch darum. Damit der Garbage Collector seine Arbeit verichten kann, muss er zuerst den gesamten Thread stoppen. Je ineffizienter die interne Datenstruktur des Spieles ist, desto länger dauert dieser Freeze.

Man soll versuchen, so wenige Pointer wie möglich zu verwenden. Auch Redundanz sollte in oft vorkommenden Objekten vermieden werden. Systeme sollten so wenig wie möglich übereinander wissen und keine direkten Pointer aufeinander haben. Architekturen sollten pyramidenförmig sein, mit den Pointern nach unten zeigend.

Durch diese Optimierungen wird die GC nicht mehr wahrgenommen.

Tabellenverzeichnis

Tabelle 1 Umfeldanalyse - Beschreibung der wichtigsten Umfelder 17
Tabelle 2 Umfeldanalyse - Beschreibung der Risiken 18
Tabelle 3 Risikoanalyse - Gegenmaßnahmen 20
Tabelle 4 Bezeichnung der AABB Kollisions Formel 67

Abbildungsverzeichnis

Abbildung 1 Grafische Darstellung der Umfelder 16
Abbildung 2 Risikoportfolio 19
Abbildung 3 Aseprite Logo Quelle:
https://pbs.twimg.com/profile_images/875731732389146624/-UznwnAx_400x400.jpg 28
Abbildung 4 Aseprite Benutzeroberfläche 30
Abbildung 5 Adobe Photoshop Logo Quelle:
https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Adobe_Photoshop_CC_icon.svg/
768px-Adobe_Photoshop_CC_icon.svg.png 31
Abbildung 6 Wraithknight's Farbpalette 39
Abbildung 7 Dunkle Ritter Standposition 43
Abbildung 8 Bogenschütze Standposition 44
Abbildung 9 Endboss Standposition 44
Abbildung 10 Pfeil Projektil 45
Abbildung 11 Wurfmesser Projektil 45
Abbildung 12 Windstoß Projektil 45
Abbildung 13 Gebiss Projektil 45
Abbildung 14 Zeichnen von Bäumen (Quelle: https://imgur.com/gallery/eD8NA) 46
Abbildung 15 Bäume 47
Abbildung 16 Pflanzen 47
Abbildung 17 Wraithknight's Logo 50
Abbildung 18 Monogame Logo Quelle:
https://upload.wikimedia.org/wikipedia/commons/e/e6/MonoGame_Logo.svg 51
Abbildung 19 ECS Architektur 56
Abbildung 20 Deadly Diamond Szenario 57
Abbildung 21 Monogame Content-Manager 59
Abbildung 22 Komplexe und simple Animationen 64
Abbildung 23 AABB Kollision Quelle: https://learnopengl.com/img/in-
practice/breakout/collisions_overlap.png 66
Abbildung 24 . Minkowski-Summe von 2 Quadraten Quelle:
https://en.wikipedia.org/wiki/Minkowski_addition#/media/File:Minkowski_sum_graph_-
_vector_version.svg 67
Abbildung 25 Penetrationsvektor Quelle: https://blog.hamaluik.ca/assets/images/simple-aabb-
collision-using-minkowski-difference/penetration_vector.png 68
Abbildung 27 80% Rauschen am Rand 71
Abbildung 26 50% Rauschen 71
Abbildung 29 2. Zyklus 71
Abbildung 28 1. Zyklus 71
Abbildung 31 4. Zyklus 71
Abbildung 30 3. Zyklus 71
Abbildung 33 Auf Dimensionen angepasst 72
Abbildung 32 Räume wurden entfernt 72
Abbildung 34 Entität Plazierung 73