

Maturitätsarbeit 2023
Dokumentation

Laneclash



Version: 0.1
Datum: 2022-10-23 20:31:23+02:00

Team: Marc Honegger (C6a)
Elia Schürpf (C6a)

Begleitperson: Martin Hunziker

Gymnasium
KZO - Kantonsschule Zürcher Oberland

Inhaltsverzeichnis

I Ziel	1
1 Vorwort	2
2 Ideen	3
2.1 Skizzen	3
2.2 Mindmap	5
2.3 Überlegungen zum Spielprinzip	5
3 Anforderungen	7
3.1 Anforderungen an die Maturitätsarbeit	7
3.2 Anforderungen an das Spiel	8
3.2.1 Notwendig	8
3.2.2 Anvisiert	9
3.2.3 Möglich	10
3.2.4 Bereits früh verworfen	12
4 Risikoanalyse	13
4.1 Hoch	13
4.2 Mässig	14
4.3 Tief	14
II Produkt	16
5 Resultat	17
5.1 Maturitätsarbeit	17
5.2 Spiel	17
5.2.1 Notwendig	17
5.2.2 Anvisiert	23
5.2.3 Möglich	23
5.2.4 Fürs Erste verworfen	24

6 Architektur	27
6.1 Klassendiagramm	27
6.2 Sequenzdiagramme	28
6.2.1 Nahkampftruppe	28
6.2.2 Suizidtruppe	29
6.2.3 Fernkampftruppe	29
6.2.4 Gift	31
6.3 GitHub file system / source / version control	31
7 Zukunft	32
7.1 Ein weisses Blatt	32
7.2 Alpha- und Closed Beta-Testing	32
7.3 Veröffentlichung	33
III Organisation	34
8 Technologien	35
8.1 Unity	35
8.2 GitHub	36
8.3 Jetbrains Rider	36
8.4 Jira	36
8.5 LaTeX	36
8.6 Stable Diffusion	37
8.7 Visual Studio Code	37
8.8 Gource	37
9 Team Management	38
9.1 Anfänge	38
9.2 Betreuungsperson	38
9.3 Kommunikation	39
10 Time Management	40
10.1 Roadmap	40
10.2 Time-Tracking	41
IV Reflexion	42
11 Elia	43
11.1 Rückblick	43
11.2 Nächstes Mal	44

12 Marc	45
12.1 Rückblick	45
12.2 Nächstes Mal	45
13 Team	47
13.1 Rückblick	47
13.2 Nächstes Mal	48
V Glossar	50
Glossar	51

Teil I

Ziel

Kapitel 1

Vorwort

Unsere Vision:

Ein funktionsfähiges Videospiel, welches jede vergleichbare Maturitätsarbeit in den Schatten stellt

Die Grundidee für dieses Game war nicht neu, sondern hatte sich vor einigen Jahren in Marcs Kopf geformt. Jedoch war für die Umsetzung ein Team notwendig. Zusammen mit Elia bildete sich ein gamebegeistertes, programmieraffines und engagiertes Duo. Uns war dennoch von Anfang an bewusst, dass wir auch zu zweit nicht alle Funktionen erfolgreich umsetzen werden. Dennoch haben wir uns dieses hohe Ziel gesetzt. Interessiert und unterstützend begleitete uns Herr Martin Hunziker (IT-Koordinator der KZO). Vieles konnten wir schlussendlich auch umsetzen. Das grösste Wagnis war der Multiplayer und das haben wir dann auch an eigenem Leib erfahren. Alleine für die Erstellung des Multiplayer-Modus investierten wir über 100 Stunden.

Letztendlich haben wir es geschafft, ein funktionsfähiges und lustiges Spiel zu entwickeln.
Einige Kommentare:

'Sick Game' - Marc Honegger
'Absolut Krass' - Elia Schürpf
'Masterpiece' - Martin Hunziker

Wie das Spiel jetzt aussieht, wie es zur aktuellen Version kommen konnte, aber auch wie es in Zukunft damit weiter geht, ist in dieser Arbeit beschrieben.

Kapitel 2

Ideen

2.1 Skizzen



Abb. 2.1: Aufbau der Startseite

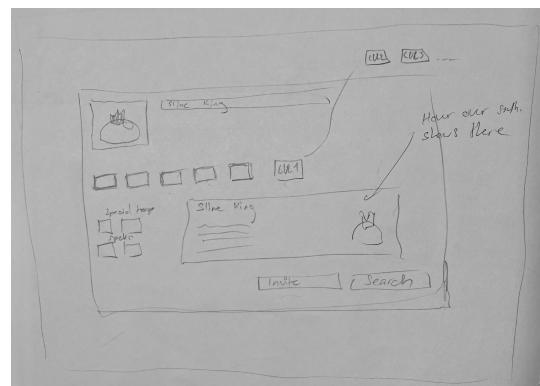


Abb. 2.2: Aufbau bei der Deckerstellung



Abb. 2.3: Spieleszenen mit UI

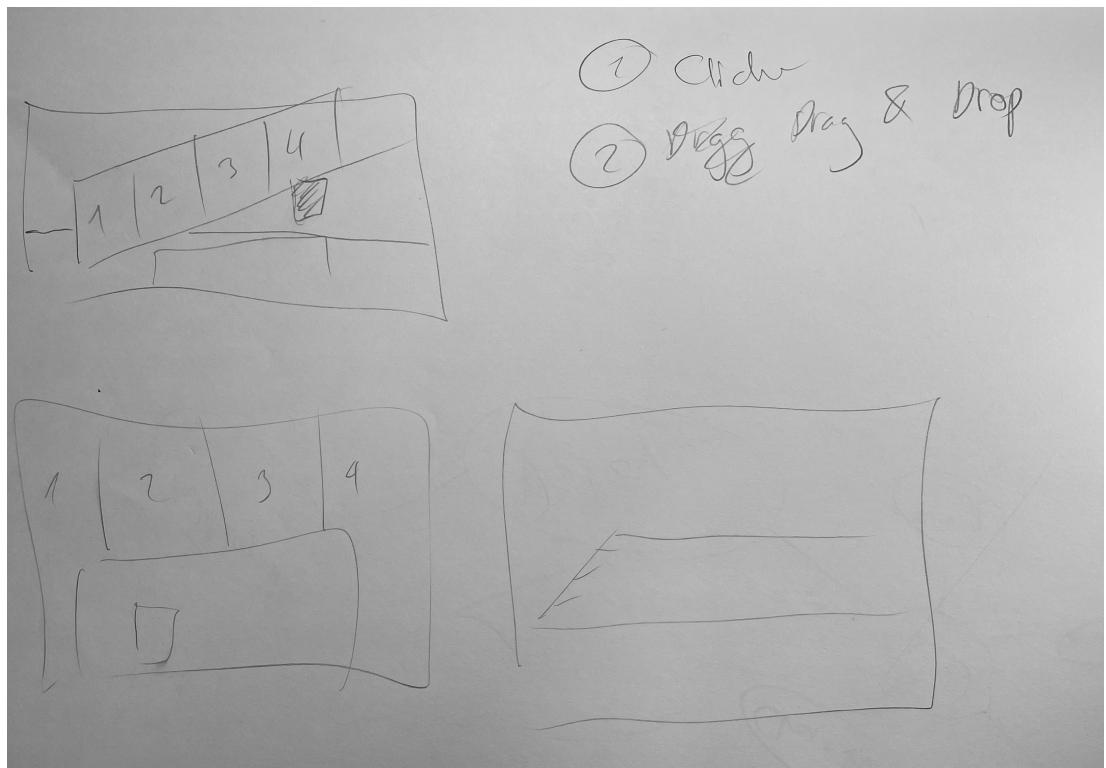


Abb. 2.4: Mechanik der Karten / Spawnen von Truppen

2.2 Mindmap

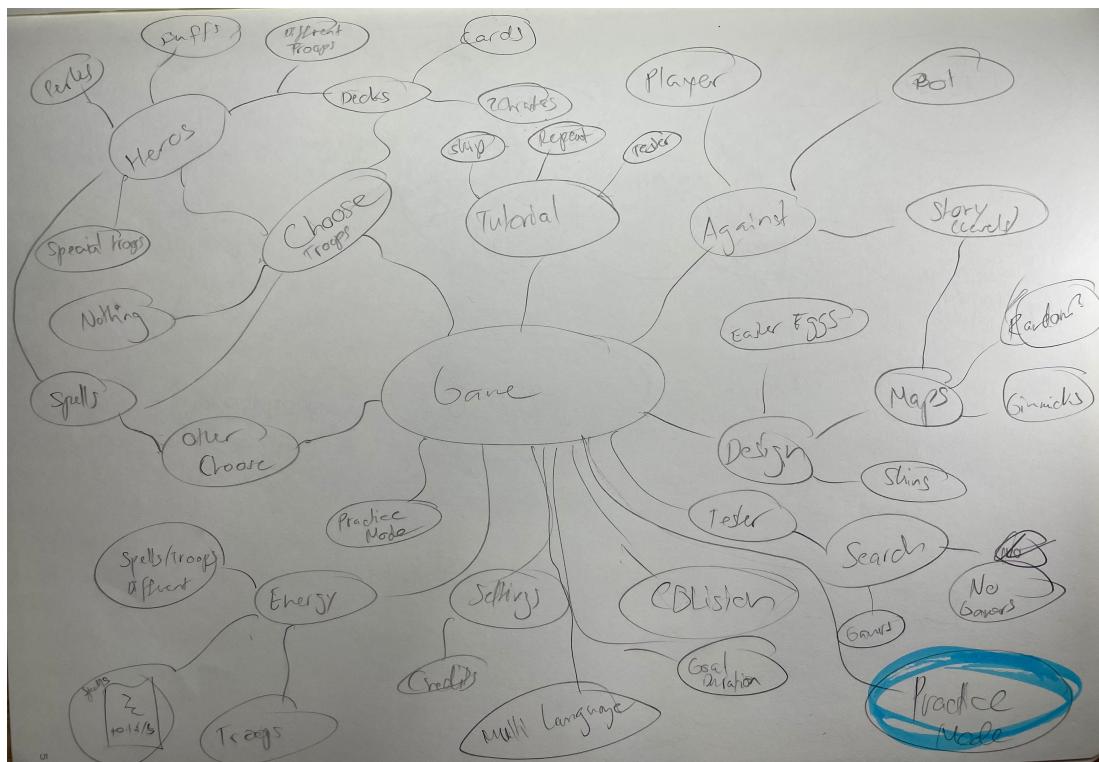


Abb. 2.5: Mindmap

2.3 Überlegungen zum Spielprinzip

Die Idee des Spielprinzips wird im folgenden Unterkapitel grob beschrieben.

- In unserem Spiel treten zwei Spieler*innen gegeneinander an. Dabei befinden sich die Startbereiche der Spieler*innen auf der linken bzw. rechten Seite einer steinernen Brücke.
- In die Schlacht begleitet wird jede*r Spieler von vier Helden. Diese Helden schreiten zu Beginn des Spieles durch die magischen Portale und bilden einen ersten Verteidigungswall. Sterben jene Helden, lösen sie einen Zauber aus, der alle gegnerischen Truppen auf dieser Linie der Brücke ausradiert.
- Der*die Spieler*in kann zudem Unterstützung aus dem eigenen Lager befehligen. Die verfügbaren Truppen erhält der*die Spieler*in in Form von Karten. Der*die Spieler*in kann mit der Verschiebung dieser Karten auswählen, durch welches Portal sie schreiten sollen.

- Jede Truppe verlangt eine unterschiedliche Menge an Kraft (Mana, spirituelle Energie), um das Portal zu durchschreiten. Die zur Verfügung stehende Mana wird oben recht angezeigt.
- Nach jeder Durchschreitung muss der*die Spieler*in eine bestimmte Zeit warten, bis er die nächste Truppe auswählen kann.
- Gelingt es einer verbündeten Einheit das gegnerische Portal zu durchschreiten, ist das Spiel vorbei und man hat gewonnen.
- Mit den Pfeiltasten kann der*die Spieler*in steuern, welchen Teil des Schlachtfelds er überblicken will.

Kapitel 3

Anforderungen

3.1 Anforderungen an die Maturitätsarbeit

Die Anforderungen der Schule (https://www.kzo.ch/fileadmin/internet/pdf_internet/Unterricht/Maturjahr/Reglement_Maturitaetsarbeit_2023.pdf) halten sich in Grenzen. Es gibt nur wenige genauen Anforderungen und die, die es gibt, sind relativ vage formuliert. Die drei Hauptanforderungen der KZO sind:

1. Zeit

Ungefähr eine Lektion pro Woche soll während den Semestern 5.2 und 6.1. als Zeitaufwand veranschlagt werden

Diese Stunde wird bei den Wochenlektionen dieser Semester angerechnet. Jedoch ist diese Anforderung für uns nicht schwer zu erfüllen, denn unser Projekt wird eine sehr umfangreiche und arbeitsintensive Maturitätsarbeit sein. Wir werden diese Zeitanforderung ohne Probleme erreichen und bei Weitem überschreiten.

2. Begleitperson

Jede Maturitätsarbeit benötigt eine Begleitperson.

Diese Lehrperson bewertet die Arbeit und unterstützt die Schüler bei Bedarf. Wir sind bereits früh auf Herrn Arlbert Kern (Informatiklehrer der KZO) zugegangen. Er hat uns an Herrn Martin Hunziker (Leiter der IT der KZO) verwiesen. Dieser nahm uns sehr gerne mit der Anmerkung auf, dass die Umsetzung unserer Idee sehr schwierig sein werde.

3. Plagiat

Arbeiten dürfen nicht einfach kopiert werden.

Wir haben uns vereinzelt bei gewissen Problemen online inspirieren lassen. Unsere verwendeten Lösungen sind jedoch keine Plagiate. Unser gesamter Code sowie unsere schriftliche Arbeit sind von uns selbst geschrieben.

3.2 Anforderungen an das Spiel

Gewisse Features sind für das Funktionieren des Spiels wichtiger als andere. Zum Beispiel sind für uns Einstellungen, Truppen und Multiplayer notwendiger als die Monetarisierung. Deshalb ist der folgende Abschnitt in vier Stufen unterteilt:

1. **Notwendig:** Ohne diese Features läuft das Spiel nicht oder ist sehr mangelhaft, voller Fehler und unspielbar. Ohne diese Funktionen ist die Idee des Spieles nicht erkennbar.
2. **Anvisiert:** Diese Features sind unser Ziel. Sie wurden Herrn Hunziker angekündigt und sind alle in unserem Plan festgehalten. Diese Funktionen sollen dazu führen, dass das Spiel gut spielbar ist und es Spass macht. Sie sind auch dafür essenziell, das Spiel nicht zu monoton zu gestalten und sollten alle bis zum Abgabetermin der schriftlichen Arbeit vollendet sein.
3. **Möglich:** Dies sind Möglichkeiten, den Spielspass auf ein neues Niveau zu heben. Allerdings ist der Zeitaufwand für die Programmierung dieser Funktionen sehr gross. Keine dieser Anforderungen ist für das Spielen nötig, kann das Spielerlebnis jedoch spannender und ausgereifter machen. Diese Ideen sind teilweise nur Konzepte, die noch bis zur mündlichen Präsentation zu erreichen sind. Vereinzelt können diese, wie die anvisierten Anforderungen, noch bis zur schriftlichen Abgabe erreicht werden.
4. **Verworfen:** Weitere Ideen, die wir hatten, wir jedoch als unmöglich oder nicht sinnvoll erachteten.

3.2.1 Notwendig

- **Systemanforderungen**

Unser Spiel soll auf den meisten modernen Computern funktionieren. Es soll auf macOS und Windows fliessend gespielt werden können (mindestens 60FPS).

- **Benutzeroberfläche**

Die Startseite soll Folgendes enthalten: einen "Spielen"-Knopf, einen "Einstellungen"-Knopf, einen "Credits"-Knopf und einen "Verlassen"-Knopf. Die Benutzeroberfläche soll intuitiv bedienbar sein und schön aussehen. Sie soll mit der Maus bedienbar sein. Andere Eingabemöglichkeiten werden nicht unterstützt.

- **Lokaler Multiplayer**

Das Spiel soll im lokalen Netz gespielt werden können, so zum Beispiel mit Freunden oder Familie. Es sollte mithilfe der IP-Adresse innerhalb des gleichen Netzes zusammengespielt werden können.

- **Einstellungen**

Die Auflösung, Vollbild und Lautstärke müssen im UI eingestellt werden können.

- **Kamera**

Die Kamera ist beweglich und das ganze Schlachtfeld ist sichtbar.

- **Truppen**

Es braucht Truppen, die für eine*n Spieler*in kämpfen können. Diese sind für den Spielverlauf notwendig. Und ohne sie hat das Spiel keinen Sinn.

- **Gewinnmöglichkeit**

Es soll eine Möglichkeit geben, das Spiel zu gewinnen oder zu verlieren und es somit zu beenden. Diese Anforderung kann sehr simpel mit einer Linie verwirklicht werden, welche beim Überschreiten einer Truppe das Spiel beendet.

- **Lanes**

Der*die Spieler*in soll auswählen dürfen, an welcher Stelle er seine Truppe heraufbeschwören will. Dabei soll er*sie die Auswahl zwischen vier Linien haben. Diese Linien sind in die Tiefe verschobenen Abgrenzungen.

- **Synchronisation**

Unser Spiel soll in Echtzeit spielbar sein, deswegen müssen die beiden Spieler*innen synchronisiert dasselbe sehen. Eine Desynchronisation wäre verheerend.

- **Crossplay** Wir wollen die Möglichkeit erreichen, einen Windows Rechner und einem macOS Rechner zu verbinden und zusammenzuspielen.

3.2.2 Anvisiert

- **Helden**

Beide Spieler*innen haben eine Art Truppe, welche ihre Siegeslinie beschützt. Sie haben einen Racheeffekt, welcher die Lane ausradiert, damit das Spiel nicht sofort vorbei ist.

- **Design**

Das Spiel sollte vom Aussehen her "was hergeben". Es sollte nicht wie ein Prototyp, sondern wie ein vollendetes Spiel aussehen.

- **Deck**

Zu Beginn können die Spieler*innen ihr eigenes Deck aus einer Auswahl von Karten zusammenstellen. Im Spiel werden aus diesem Kartendeck per Zufall Karten gezogen.

- **Bot**

Dies ist ein sehr simpler Algorithmus, um alleine gegen den Computer zu spielen. Die Schwierigkeitsstufe kann zwischen '*Einfach*', '*Mittel*', '*Schwierig*' gewählt werden. Angepasst an den Schwierigkeitsgrad werden zufällig Truppen geschickt.

- **Truppen**

Es gibt verschiedene Truppenarten:

- **Nahkampftruppe:** Wenig Reichweite
- **Fernkampftruppe:** Greift auf grosse Reichweite an. Sie schiesst Projektiler (z.B. ein Bogenschütze, der mit Pfeilen schiesst).
- **Suizidtruppe:** Stirbt bei Berührung mit einem Gegner und löst einen Effekt aus.

- **Effekte**

- **Gift:** Zeitlich limitierter und wiederholender Schadenseffekt.
Eine visuelle Markierung soll vorhanden sein und der Gifteffekt darf sich nicht kumulieren.
- **Dornen:** Truppen, welche Charaktere mit Dornen attackieren, erhalten Schaden. Der Effekt soll nach Auswahl auf Nahkämpfer, Fernkämpfer oder beides wirken.
- **Rache:** Truppen mit dem Effekt Rache lösen einen Effekt nach ihrem Tod aus. Beispielsweise kann durch diesen Tod eine Truppe beschworen oder Schaden verursacht werden.

3.2.3 Möglich

- **Zauber**

Alternativ zum Herbeirufen einer Truppe können gewisse Karten einen Effekt auslösen (Zauber).

Beispiele für Zauber:

- ”Heile deine Helden um 5 Lebenspunkte”
- ”Gib deinen Helden 5 Rüstungspunkte”
- ”Verursache allen gegnerischen Truppen 5 Schadenspunkte”
- ”Ziehe 3 Karten”

- **Helden**

Eine Auswahl von Helden, mit unterschiedlichem Schaden.

Leben und Effekte des jeweiligen Helden würden das Spiel nochmals spannender machen. Auch sollten gewisse Truppen auf bestimmte Helden limitiert sein.

- **Truppen**

Weitere Truppen können jederzeit erstellt werden. Hier ist kein Limit gesetzt. Leben, Schaden, Darstellung, Effekt und vieles mehr kann angepasst werden.

- **Effekte**

- **Wiederbelebung:** Die Truppe wird sofort an Ort und Stelle oder mit einer Verzögerung am Startpunkt wiederbelebt.

- **Rüstung:** Die Truppe hat zusätzliche Rüstungspunkte zu den Lebenspunkten. Die Rüstung wird zuerst abgezogen und hat im Gegensatz zum Leben keine Limite.
- **Aura:** Die Truppe fügt gegnerischen Truppen in einem gewissen Radius permanent Schaden zu.

- **Design**

Wenn ausreichend Zeit vorhanden ist, kann das Design immer weiter verbessert werden. Hier geht es um den Feinschliff, z.B. mehr Hintergründe, Details und mehr selbsterstellte Designs.

- **Tutorial**

Eine Anleitung für Anfänger*innen wäre sehr schön. Sie soll neuen Spielern*innen das Anfangen erleichtern. Sie sollte aber auch überspringbar sein, damit erfahrene Spieler*innen das Tutorial nicht erneut spielen müssen. Es soll weder allzu lang noch zu schwierig sein. Idealerweise soll es "anhand von Gameplay" alle Mechaniken des Spiels erklären.

- **Mehrsprachig**

Das Spiel soll in Englisch, Deutsch und Französisch verfügbar sein.

- **Ingame-Chatfenster**

Ein Textfeld, in dem man mit dem/der Gegner*in chatten kann, würde einen zusätzlichen Spassfaktor bieten.

- **Monetarisierung**

Es sind uns drei Möglichkeiten eingefallen, wie wir mit dem Spiel Geld verdienen könnten. Es folgen jeweils die Vor- und Nachteile:

1. **Kaufbare Gegenstände**

- + Vielseitige und nachhaltige Monetarisierung. Neue Features bedeuten auch neue Einnahmemöglichkeiten. Die käuflichen Inhalte sollten auch erspielbar sein. Somit kann bezahlen zwar zu Vorteilen führen, diese können jedoch mit einer hohen Spielzeit trotzdem erreichbar sein.
- Pay-to-Win

2. **Skins**

- + Weit verbreitet und sehr beliebt bei Spieler*innen geben sie keinem*keiner Spieler*in Vorteile.
- Wenig Nutzen und sehr aufwendig. Neue Designs zu erstellen, wäre bei uns nicht sinnvoll. Wir möchten noch sehr viele Features implementieren und benötigen dafür viele zeitlichen Ressourcen. Zudem sind wir keine spezialisierten Designer. Der Aufwand für uns, eine ansprechende und brauchbare Darstellung zu erschaffen, steht in keinem sinnvollen Verhältnis zum Mehrwert für das Spiel.

3. Kostenpflichtiges Spiel

- + Einmalige Bezahlung bei Spielkauf, was für viele Nutzer*innen lukrativer ist. Die gilt jedoch vor allem bei Singleplayer-Spielen.
- Wir nehmen an, dass wenige Nutzer*innen bereit wären, Geld für unser Spiel zu bezahlen. Dafür wird es vorerst nicht genug ausgereift sein. Auch ist diese Methodik nicht nachhaltig und führt nur zu einer einmaligen Geldspritz. Viele grössere Spiele führen deshalb später DLCs ein, um das Spiel zu erweitern. Jedoch ist dies bei einem Multiplayer Spiel Pay-to-Win. Wir müssen mit hoher Wahrscheinlichkeit vorerst Geld investieren, um unser Spiel anbieten zu können.

3.2.4 Bereits früh verworfen

- **Online Multiplayer**

Besser als ein Multiplayer, der limitiert auf dasselbe Netz ist, ist ein Multiplayer, der weltweit verfügbar ist. Jedoch ist die direkte Verbindung zwischen zwei Computern, die sich in unterschiedlichen Netzen befinden, dank der heutigen Firewalls von Routern und Computern für uns nahezu unmöglich. Es würden sich viele weitere Probleme ergeben, wie zum Beispiel Port-Forwarding. Dies ist einer der Gründe, weshalb ein Server sehr praktisch wäre. Mit diesem wäre dieses Problem gelöst. Server sind aber teuer und aufwendig zu programmieren und zu warten.

- **Shop**

Nicht alle Karten sind von Beginn an freigegeben, sondern müssen freigespielt werden. So kann zum Beispiel eine Ingame Währung erspielt werden und damit im Shop Karten oder sonstige Dinge gekauft werden. Der Shop kann mit Zahlungsmethoden ausgestattet werden und so zur Monetarisierung beitragen.

- **Kampagne**

Im Singleplayer spielt man gegen bestimmte vorprogrammierte Gegner. Sie sollen immer stärker werden und bestimmte Herausforderungen mit sich bringen. Bei einem Sieg werden Belohnungen verteilt.

- **Anti-Cheat**

In allen grösseren Spielen ist eine sehr komplexe Software vorhanden, um das Schummeln zu verhindern. Teilweise geschieht dies auf Systemebene oder sogar auf Kernel-Level.

- **Errungenschaften**

Durch Erfüllung von Aufgaben erhält man Belohnungen (z.B. Freischalten von bestimmten Karten). Die möglichen und bereits erreichten Errungenschaften sollen in einer Übersicht dargestellt werden. Diese Erweiterung ist keinesfalls notwendig und ein absolutes nice-to-have Feature.

Kapitel 4

Risikoanalyse

Wir haben mögliche Risiken in verschiedene Stufen eingeteilt. Die Einstufung geschah basierend auf der Eintrittswahrscheinlichkeit (unwahrscheinlich, wahrscheinlich, sehr wahrscheinlich) und Auswirkung (unbedenklich, kritisch). Um das Risiko zu managen, haben wir für jedes Risiko eine Reduktionsstrategie definiert.

In den folgenden Teilkapiteln dokumentieren wir die identifizierten Risiken nach folgendem Schema:

- Risiko (Eintrittswahrscheinlichkeit und Auswirkung)
 1. Reduktionsstrategie: Strategie, um die Wahrscheinlichkeit und/oder die Auswirkung zu reduzieren.
 2. Reduziertes Risiko: Risiko nach der Umsetzung der Reduktionsstrategie.
 3. Tatsächliches Outcome: Das tatsächliche Endresultat

4.1 Hoch

1. Vergangene Maturitätsarbeiten, die ein Videospiel als Ziel hatten, sind gescheitert (Wahrscheinlich und Kritisch)
 - (a) Reduktionsstrategie: Klare Anforderungsspezifikation, Planung und Aufteilung der Arbeitspakete, laufende Überprüfung der Planung/Zielerreichung, regelmässiger gegenseitiger Austausch zum Fortschritt, Bereitschaft den empfohlenen Zeitaufwand massiv zu überschreiten.
 - (b) Reduziertes Risiko: Die Eintrittswahrscheinlichkeit ist reduziert. Beim Eintreten ist der Erfolg der Arbeit allerdings immer noch gefährdet.
 - (c) Tatsächliches Outcome: Risiko nicht eingetreten. Die Maturitätsarbeit wurde zusammen mit funktionierendem Videospiel fertiggestellt.
2. Beide Teammitglieder haben keine Ahnung von der Entwicklung eines netzwerkfähigen Multiplayer-Spiels (Wahrscheinlich und Kritisch)

- (a) Reduktionsstrategie: Erstellen der grundsätzlichen Multiplayerfunktionalität in einer einzelnen Spielinstanz, danach Ausbau auf mehrere Spielinstanzen über das lokale Netzwerk (Host und Client).
 - (b) Reduziertes Risiko: Spiel am selben Rechner ist sehr wahrscheinlich möglich, primäres Ziel der Arbeit ist auch ohne Netzwerkfähigkeit erfüllt. Das Risiko ist selbst reduziert immer noch sehr hoch.
 - (c) Tatsächliches Outcome: Risiko nicht eingetreten. Mehrere Spieler können übers Netz zusammen spielen. Das Ziel wurde nur mit beträchtlichem Aufwand erreicht (1/4 der gesamten Entwicklungszeit).
3. Spiel enthält unentdeckte / unbekannte Fehler, die den Spielspass verderben (Wahrscheinlich und Kritisch)
- (a) Reduktionsstrategie: So viel testen wie möglich/sinnvoll, Bugs kategorisieren, sich auf kritische Bugs konzentrieren, Code Reviews.
 - (b) Reduziertes Risiko: Wir schätzen das reduzierte Risiko immer noch als hoch ein, da selbst wenn die Eintrittswahrscheinlichkeit stark reduziert ist; kann dieser eine unentdeckte Fehler den ganzen Spielspass verriesen.
 - (c) Tatsächliches Outcome: Nach Abschluss der Alpha Testphase sind keine Fehler bekannt. Wir drücken nun die Daumen, dass Herr Hunziker keine Bugs findet.

4.2 Mässig

1. Elia hat noch nie mit Unity gearbeitet (Sehr wahrscheinlich und Unbedenklich)
 - (a) Reduktionsstrategie: Elia befasst sich zu Beginn mit Unity, Marc hat Vorwissen und kann Elia unterstützen.
 - (b) Reduziertes Risiko: Falls Elia auf Schwierigkeiten stösst, kann dies zu zeitlichem Mehraufwand führen.
 - (c) Tatsächliches Outcome: Risiko nicht eingetreten. Elia konnte alles Nötige zu Beginn erlernen. Bei Problemen konnte er diese entweder durch Internetrecherchen klären oder Marc fragen.

4.3 Tief

1. Wir werden selbst nicht mit unserem Spiel zufrieden sein. (Wahrscheinlich und Unbedenklich)
 - (a) Reduktionsstrategie: Sich bewusst sein, dass wir nur zwei Schüler sind, die keine bisherigen Erfahrungen hatten. Wir dürfen uns nicht an etwas "festfressen".

- (b) Reduziertes Risiko: Realistische Erwartungshaltung
- (c) Tatsächliches Outcome: Basierend auf der uns zu verfügbaren Zeit und unserem Wissen sind wir zufrieden mit unserem Produkt. Es gibt immer noch Dinge, die wir gerne verbessern und hinzufügen würden. So könnten wir noch Tausende Stunden investieren.

Teil II

Produkt

Kapitel 5

Resultat

Kurz gesagt: Wir konnten fast alle unsere Anforderung erfüllen.

5.1 Maturitätsarbeit

1. Zeit

Geplant war ungefähr eine Lektion pro Woche. Wir haben die geschätzten 100 Stunden (50 Stunden pro Person) um den Faktor vier überschritten. Unser gemessener Zeitaufwand während des aktiven Programmierens beträgt über 420 Stunden. Hier sind die Stunden, in denen wir uns in der Schule über das Projekt unterhalten haben, nicht notiert.

2. Begleitperson

Wir hatten glücklicherweise kein Problem eine Begleitperson zu finden. Martin Hunziker war gerne bereit, unsere Arbeit im Blick zu behalten.

3. Plagiat

Einige Grafiken sind zwar aus dem Internet, jedoch sind alle nutzungsfrei. Wir haben keinen Code geklaut und verwendet.

5.2 Spiel

In den folgenden Teilkapiteln wird die Erfüllung der in Abschnitt 3.2 gestellten Anforderungen gruppiert dargestellt.

5.2.1 Notwendig

- **Systemanforderungen**

Unser Spiel funktioniert sowohl auf macOS als auch auf Windows. Es läuft zudem mit einer FPS von 300, bis teilweise 500, bei den meisten modernen Rechnern.

- Benutzeroberfläche

- Startseite

Unsere Startseite hat einen "Spielen"-Knopf, einen "Einstellungen"-Knopf, einen "Credits"-Knopf und einen "Verlassen"-Knopf. Unsere Tester bewerteten die Startseite als intuitiv verständlich. Bedienbar mit der Maus, hatten die Tester keine Probleme sich zurechtzufinden.



Abb. 5.1: Startseite

- Credits

Es gibt die Möglichkeit, die Credits vorzeitig zu verlassen. Dies geschieht durch das Drücken des Hausbuttons. Die Tester fanden dies auch sehr intuitiv.



Abb. 5.2: Credits

- Settings

Man kann hier die Auflösung ändern und auswählen, ob das Spiel als Vollbild dargestellt werden soll. Auch gibt es hier die Möglichkeit, die Musik lauter

oder leiser zu stellen oder zu muten. Wenn man Einstellungen geändert hat, muss man das Speichern bestätigen.



Abb. 5.3: Settings

Nach Bestätigung der Änderung der Grafikoptionen, werden die Änderungen angewendet und man hat zehn Sekunden Zeit erneut zu bestätigen. Sonst werden die Änderungen zurückgesetzt. Dies wurde implementiert, um eventuellen Problemen vorzubeugen.



Abb. 5.4: Resolution

– Server Management

Nach Bedienung des "Start"-Knopfs gibt es die Möglichkeit, einen Server zu hosten oder einer bereits existierenden Lobby als Client beizutreten. Außerdem kann man automatisch nach bereits existierenden Servern suchen.

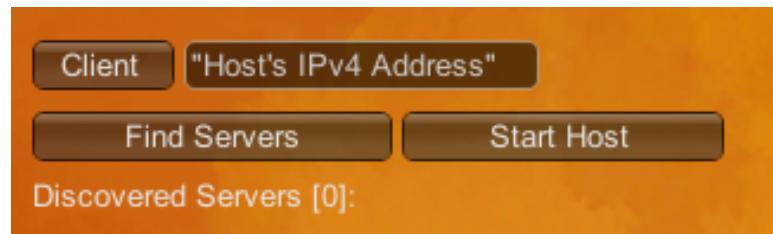


Abb. 5.5: Server Management

Den Server kann man mit einem Linksklick auswählen.



Abb. 5.6: Server Discovery

– Lobby Management

Player (1) ist dabei immer der Host, und Player (2) immer der Client. Das eigentliche Spiel beginnt erst, sobald beide Spieler*innen bereit, also "ready", sind. Der Status "ready" wird dabei immer aktualisiert dargestellt. Der Host hat ausserdem die Möglichkeit, den Spieler (2) aus der Lobby zu entfernen.



Abb. 5.7: Lobby mit Host-View



Abb. 5.8: Lobby mit Client-View

– InGame-UI

Der Host hat die Möglichkeit, das Spiel zu beenden. Er kann auch direkt aufhören zu hosten. Auch der Client hat durchgehend die Möglichkeit, den Server zu verlassen.



Abb. 5.9: Leave-Options mit Host-View

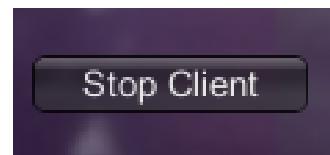


Abb. 5.10: Leave-Options mit Client-View

Im Pausenmenu kann man ausserdem leicht die Musiklautstärke und die Stärke des Schneefalls ändern. Dieses Menu kann mit ESC geöffnet und geschlossen werden.

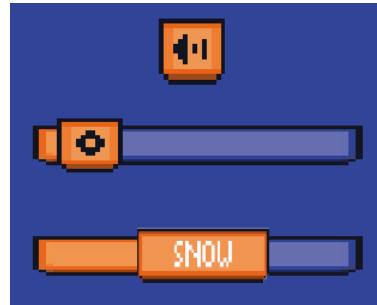


Abb. 5.11: Pausemenu

- **Lokaler Multiplayer**

Das Spiel kann im lokalen Netz gespielt werden. Zum Testen kann man das Spiel auch zweimal parallel auf demselben Gerät laufen lassen. Als Protokoll haben wir TCP gewählt. Folgende vier Punkte waren entscheidend für die Wahl von TCP gegenüber UDP:

- **Zuverlässigkeit**

Es gibt keine Probleme mit verloren gegangenen Paketen. Ging ein Paket verloren, sendet TCP es einfach erneut. Entweder werden alle Daten erfolgreich übermittelt, oder es folgt eine Errormeldung.

- **Sequenziert**

TCP stellt sicher, dass jede Nachricht in der gleichen Reihenfolge eintrifft, in der sie gesendet wurde. Wenn man "a" vor "b" sendet, erhält man auf der anderen Seite garantiert auch zuerst "a" und dann "b".

- **Verbindungsorientiert**

TCP hat sich auf das Konzept einer Verbindung spezialisiert. Eine Verbindung bleibt so lange geöffnet, bis entweder der Client oder der Server sich dazu entscheiden, sie zu schliessen. Anschliessend werden sowohl Client als auch Host benachrichtigt, dass die Verbindung beendet wurde.

- **Überlastungskontrolle**

Wenn ein Server überlastet wird, drosselt TCP selbstständig die Daten, um einen Zusammenbruch durch Überlastung zu verhindern.

- **Einstellungen**

Auflösung, Vollbild, Lautstärke und Stärke des Schneefalls können eingestellt werden.

- **Kamera**

Die Kamera ist beweglich und das ganze Schlachtfeld ist sichtbar. Mit den Pfeiltasten kann man die Kamera frei bewegen und hinein- bzw. hinauszoomen.

- Truppen

- Spielende

Um zu gewinnen, muss eine eigene Truppe die Portale des Gegners durchschreiten. Dies verhindern zuerst sogenannte Helden. Sterben jene Helden, erledigt ihr Racheeffekt alle Truppen, die sich auf jener Linie des Helden befanden.

- Lanes

Unser Spiel hat vier sogenannte Linien ("Lanes"). Diese unterschiedlichen Linien wurden in der Tiefe nach hinten verschoben. Der*die Spieler*in hat die Auswahl, in welcher Reihe er*sie seine Truppen auf den Gegner zuschicken will.

- Synchronisation

Unser Spiel synchronisiert alle Daten in Echtzeit. Dadurch sehen alle Spieler*innen immer dasselbe. Wird der Client aus einem unbekannten Grund disconnected, kann er versuchen, sich erneut zu verbinden, und der Server wird dann direkt alle Daten synchronisieren. Auch hier hilft uns das gewählte Protokoll TCP. Geht ein Paket verloren, wird es automatisch erneut gesendet. Dies hilft uns dabei, die Synchronisation zu jeder Zeit aufrechtzuerhalten.

- **Crossplay** Es ist möglich, sich mit einem Windows Rechner und einem macOS Rechner zu verbinden. Allerdings muss dem Spiel dazu die Kommunikation durch die Firewall teilweise erlaubt sein. Bei einer eher aggressiveren Firewall wird zudem teilweise der Server-Broadcast geblockt. Dies hätte zur Folge, dass man die Server nicht automatisch finden kann. Verbinden kann sich allerdings trotzdem noch.

5.2.2 Anvisiert

- **Helden**

Beide Spieler*innen haben bei unserer aktuellen Version den gleichen Heldentyp. Dies ist ein magischer Turm. Er hat keine Angriffsmöglichkeiten. Allerdings kann er sich passiv heilen und so als eine temporäre Grenze zwischen den gegnerischen Truppen und den eigenen Portalen agieren. Die eigenen Truppen werden durch diese Portale hinter den Helden gespawned.

- **Design**

Wir haben uns für ein 2.5d Design entschieden. Der Vorteil ist, dass wir 2d Grafiken benutzen können. Die Sprites werden dabei um 45 Grad gedreht. Auch die Kamera schaut in einem 45 Grad Winkel auf das Geschehen. Dadurch ergibt sich die Illusion einer Perspektive, in der man von der Seite auf das Geschehen schaut.

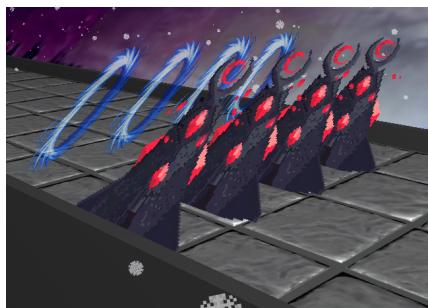


Abb. 5.12: Grafiken von der Seite



Abb. 5.13: Grafiken von vorne

- **Effekte**

- **Gift:** Zeitlich limitierter und sich wiederholender Schadenseffekt. Eine visuelle Markierung ist vorhanden und der Gifteffekt kumuliert sich nicht (hier: Pilz-Truppe).
- **Rache:** Truppen mit Rache haben einen Effekt nach ihrem Tod. Sie können zum Beispiel eine Truppe beschwören oder Schaden verursachen (hier: Helden mit Feuer).

5.2.3 Möglich

- **Monetarisierung**

1. **Kostenpflichtiges Spiel**

- Wir haben eventuell vor, unser Spiel nach Vollendung der Entwicklungsphase zu veröffentlichen. Allerdings wird der Preis auf keinen Fall hoch sein.

- **Design**

Wir werden auch in Zukunft das Design noch verbessern.

- **Truppen** Weitere Truppen können jederzeit erstellt werden. Hier ist kein Limit gesetzt: Leben, Schaden, Darstellung, Effekt und vieles mehr kann angepasst werden. Wir haben alle Programme mit dem Hintergedanken geschrieben, dass es jederzeit ohne grössere Anpassungen möglich ist, neue Truppen hinzuzufügen.

5.2.4 Fürs Erste verworfen

- **Online Multiplayer**

Um sich mit Rechnern ausserhalb des eigenen Netzwerkes verbinden zu können, müssten wir das ganze Multiplayer-System unseres Spieles anpassen, was einen kompletten Recode zur Folge hätte. Außerdem, würde sich die Frage nach der Finanzierung stellen, falls wir einen externen Server kaufen / mieten würden.

- **Shop**

Ein Shop ohne Anti-Cheat und/oder Server ist die perfekte Angriffsfläche für Cheater und ist somit für uns nicht sinnvoll. Wir müssten dann ausserdem ein Ranking-System hinzufügen, damit wir die Spieler*innen belohnen können.

- **Kampagne**

Unser Spiel ist vor allem auf den Multiplayer ausgelegt. Eine Kampagne setzt ausserdem voraus, dass man einen offline Gegner ("Bot") hat. Wir haben uns bereits früh gegen eine Kampagne entschieden, da diese in diesem Stadium leider unsere Kapazität überschreitet.

- **Anti-Cheat**

Wir haben keine Erfahrung in diesem Bereich und es ist ein extrem komplexes Thema. Allerdings hat der Host sozusagen über fast alles die Macht, der Client kann also fast gar nichts kaputt machen. Wir hoffen ausserdem auf die Ehrlichkeit der Menschen, die unser Spiel spielen.

- **Errungenschaften**

Beloohnungen zu erhalten ist immer toll. Wir haben zwei Arten von möglichen Belohnungen diskutiert und verworfen: eine Währung und / oder Pokale. Für die Währung bräuchten wir allerdings auch einen Shop. Wir müssten ausserdem neue Gegenstände hinzufügen, die man zuerst erspielen muss. Für die Pokale bräuchten wir ein Ranking-System.

- **Bot**

Ein guter Bot passt sich der Spielweise des Gegners an. Einen so guten Bot zu programmieren ist eine Herausforderung für sich. Wir hatten die Idee, dass der Bot immer zufällige Truppen zu zufälligen Zeiten spielt. Allerdings würde dies auch eine Menge Balancing benötigen, wofür wir im Moment leider zu wenig Zeit haben.

- **Helden**

Wir hatten als Ziel, mehrere verschiedene Helden zu gestalten und zu programmieren. Allerdings stellte sich relativ früh heraus, dass wir die nötigen Ressourcen lieber an einer anderen Stelle investieren. Wir haben nun einen einzigen Helden, allerdings würde sich das Ausbauen der Helden nach der Abgabe als spannend gestalten. Die Grundlage für den Ausbau ist bereits vorhanden.

- **Ingame-Chatfenster**

Wir hatten eine funktionierende Version erstellt und implementiert. Allerdings stellte sich dies als überflüssig heraus, da man normalerweise neben der Person sitzt, mit der man im Moment spielt.

- **Mehrsprachig**

Das Spiel ist im Moment nur in Englisch verfügbar. Eventuell werden wir dies nach der Abgabe der schriftlichen Arbeit noch anpassen.

- **Effekte**

- **Wiederbelebung:** Die Möglichkeit einer Wiederbelebung würde sich nur für spezielle Truppen anbieten. Er würde den*die Spieler*in wahrscheinlich ein Vielfaches davon kosten, als wenn er neue Truppen losschicken würde.
- **Rüstung:** Uns schien dies nicht nötig zu sein. Dies würde sich erst lohnen, sobald wir mehr Truppen, Effekte oder anderes hinzugefügt hätten.
- **Aura:** Das Prinzip einer Aura klingt faszinierend. Allerdings hätte dies sehr wahrscheinlich einen Recode des Damage-Systems zur Folge und würde unsere Ressourcen sprengen.

- **Tutorial**

Da wir bei der Entwicklung einen Schwerpunkt auf die intuitive Bedienbarkeit gelegt haben, konnten wir auf ein Tutorial und den damit verbundenen immensen Zeitaufwand verzichten.

- **Monetarisierung**

Ohne Anti-Cheat leider vollkommen nutzlos.

- 1. **Kaufbare Gegenstände**

- Neben Schummelmöglichkeiten stellt sich hier ausserdem die Gefahr eines möglichen Pay-to-Win.

- 2. **Skins**

- Der*die Spieler*in bräuchte ein Inventar, das mit einem Server kontrolliert und synchronisiert wird.

- **Zauber**

Wir hatten bereits zu einem frühen Zeitpunkt eine funktionierende Testversion. Der Zauber wurde dem*der Spieler*in durch Karte verteilt.

Im Laufe der Zeit haben wir das Damage-System allerdings so umgeändert, dass der Prototyp schon fast nutzlos war. Ein kompletter Recode des Prototypen wäre nötig.

- **Dornen:** Nachdem wir den Zauber ”Gift” hinzugefügt hatten, erschien und das zusätzliche Hinzufügen der Dornen überflüssig.

- **Deck**

Die Möglichkeit, sich ein eigenes Deck zusammenzustellen, aus dem man dann die Karten zieht, wurde leider aus Zeitgründen fürs Erste verworfen. Allerdings können wir fast schon garantieren, dass wir dies bis zu unserer Präsentation noch hinzufügen werden.

Kapitel 6

Architektur

Die folgenden UML-Diagramme sind nicht vollständig. Wir versuchen einen kleinen aber dennoch guten Einblick in unsere Architektur zu liefern. Dieser Abschnitt sicher spannend für technikinteressierte Leser. Man muss jedoch kein Informatiker sein, um dieses Kapitel zu verstehen.

6.1 Klassendiagramm

Folgend ist der Klassenaufbau unseres Spieles zu sehen. Das rote (S) steht für Singleton. Dies sind Klassen, die im ganzen Spiel nur genau einmal existieren können. Die erste Graphik zeigt den momentanen Stand, wobei mit der Entwicklung des Deckbauers die neue Klasse "DeckManager" hinzukommen wird.

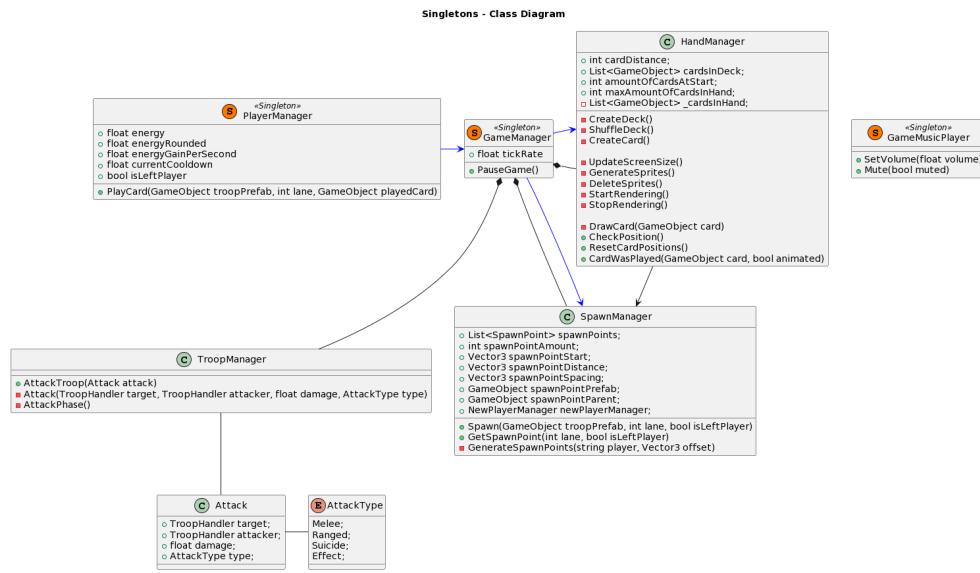


Abb. 6.1: momentaner Aufbau

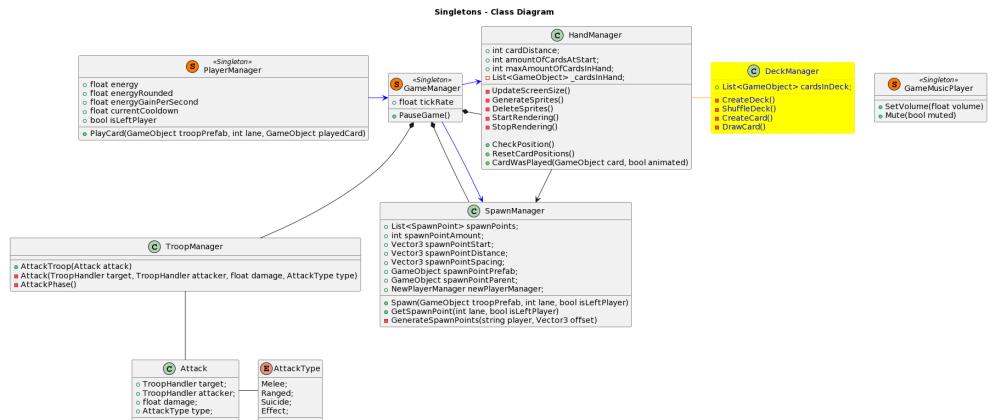


Abb. 6.2: zukünftiger Aufbau

6.2 Sequenzdiagramme

6.2.1 Nahkampftruppe

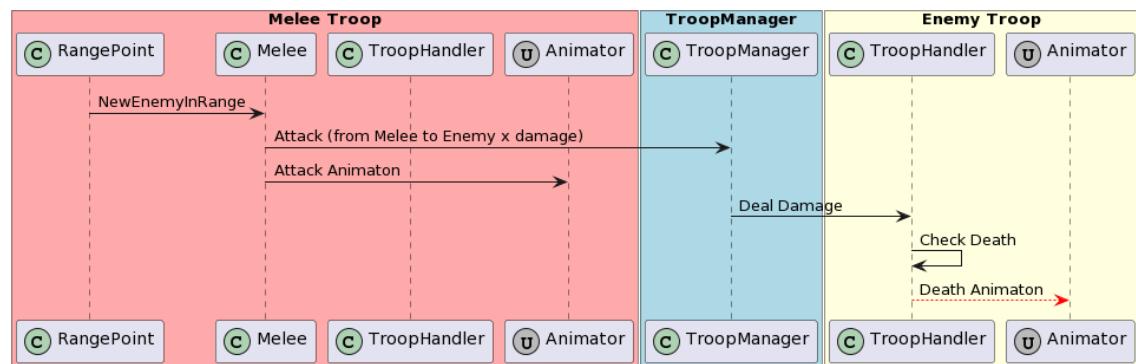


Abb. 6.3: Abfolge, wenn eine Truppe in Reichweite einer Nahkampftruppe kommt

6.2.2 Suizidtruppe

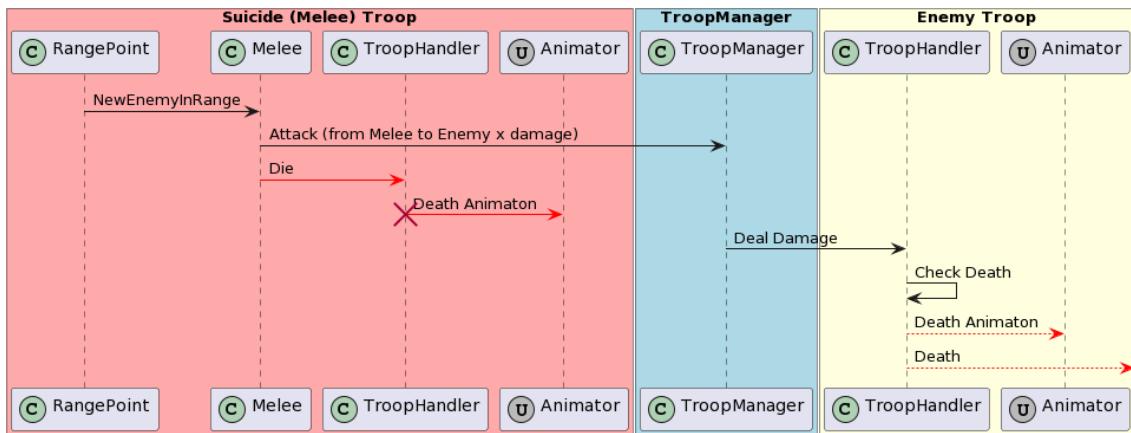


Abb. 6.4: Suizidtruppe mit ähnlichen Prinzipien

6.2.3 Fernkampftruppe

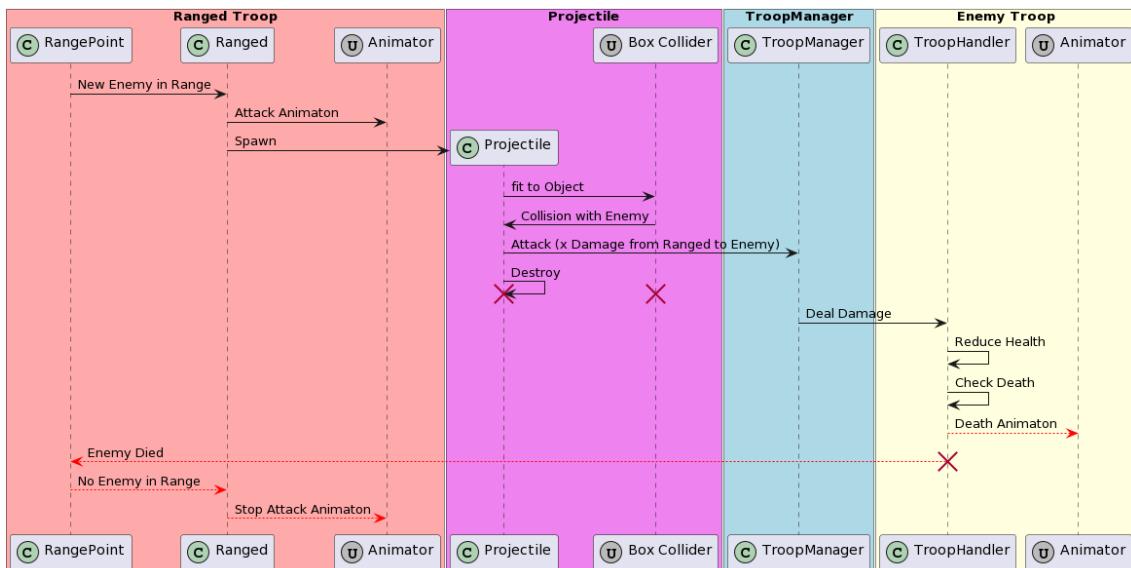


Abb. 6.5: Reaktion einer Fernkampftruppe auf neuen Gegner

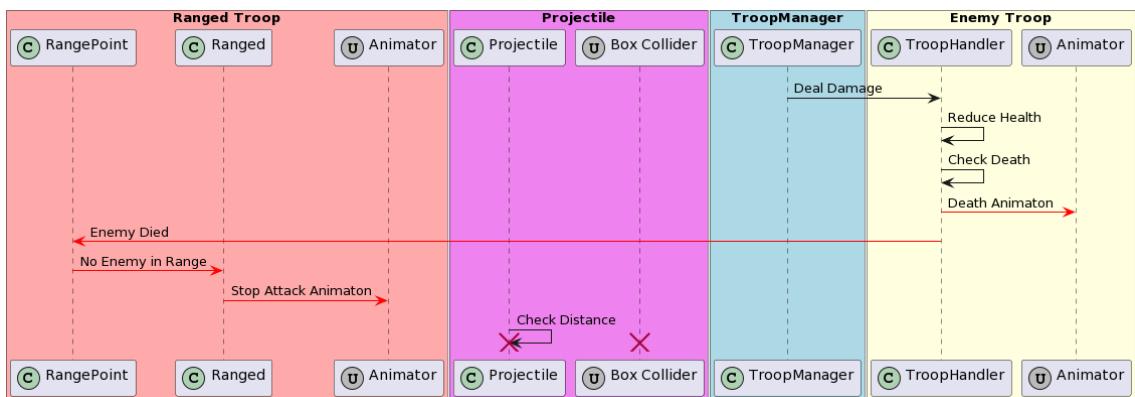


Abb. 6.6: Gegner stirbt bevor das Projektil Schaden anrichten konnte

6.2.4 Gift

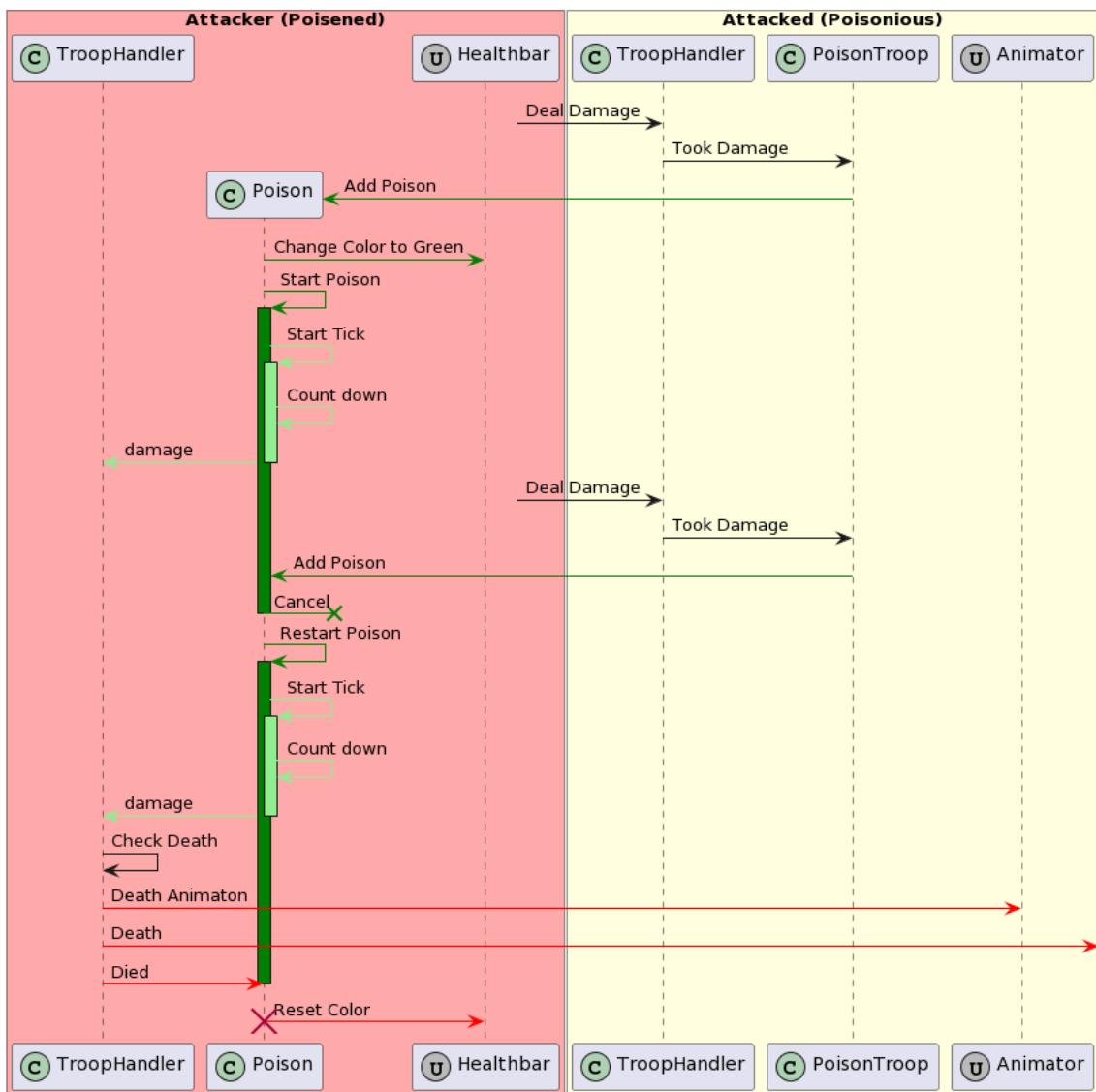


Abb. 6.7: Reaktion einer Truppe die den Effekt Gift besitzt

6.3 GitHub file system / source / version control

<https://www.youtube.com/watch?v=IQT37uwpcg4>

Kapitel 7

Zukunft

”Der beste Weg, die Zukunft vorauszusagen, ist, sie selbst zu gestalten.” (Abraham Lincoln)

7.1 Ein weisses Blatt

- Das Spiel wird bis zu der Präsentation nicht nur ”Staub ansetzen”. Wir haben fest vor (sofern es der Prüfungsdruck der kommenden Monaten erlaubt), das Spiel noch weiterzuentwickeln und zu verbessern. Denn trotz grossen Fortschritts ist Lanceclash noch lange nicht fertig. Obwohl schon sehr viel existiert, haben wir noch sehr viele Ideen, die wir gerne umsetzen möchten.
- Viele Ideen wurden aus Zeitgründen fürs Erste verworfen oder verzögert. Daher mangelt es uns auf keinen Fall an Ideen. Wir wollen diese fehlenden Features auf jeden Fall noch hinzufügen.
- Das Hinzufügen der meisten Ideen sollte sich dabei als nicht sehr schwierig herausstellen. So existiert zum Beispiel die Mechanik für den Effekt ”Dornen” bereits. Sie benötigt lediglich noch einen Feinschliff.
- Hätten wir unendlich viel Zeit, würden wir einen kompletten Recode des ganzen Spiels schreiben. Im Nachhinein wissen wir, dass wir viele Stellen besser hätten schreiben können. Zudem hätten wir den Vorteil, dass wir bereits auf die ungefähre Struktur zurückgreifen können.

7.2 Alpha- und Closed Beta-Testing

- Bis zum Abschluss der Arbeit wurde ein Alpha-Testing in einer kleinen Testgruppe durchgeführt. Bis zu der Präsentation wird das Alpha-Testing ausgeweitet und ein geschlossenes Beta-Testing ist bereits in Planung.

- Das Beta-Testing war ab der ersten Sommerferienwoche geplant. Allerdings waren wir zu diesem Zeitpunkt noch nicht zufrieden mit unserem Produkt und hatten auch noch kein Alpha-Testing durchgeführt. Wir wollten auf keinen Fall etwas unserer Meinung nach Ungenügendes aus der Hand geben. So wurde das Datum des Beta-Testings immer weiter nach hinten verschoben, bis es vor dem Abgabetermin der schriftlichen Arbeit nicht mehr realisierbar war.
- An Tester*innen wird es auf keinen Fall mangeln. Alle Personen in unserem Umfeld wollen das Spiel ausprobieren. Ob Schulkollegen, Freunde oder Verwandte; sie alle wollen unser Spiel in den Händen halten. Wir sind froh, in einem solchen Umfeld arbeiten zu können.
- Wir hatten bereits die Idee, dass sich mögliche Tester*innen auf einer Warteliste eintragen können. Wir hatte sogar angedacht, einen kleinen Server zu mieten, um eine kleine Webseite hochzuschalten, auf der sich die Testinteressierten eintragen könnten. Die Beta-Tester würden dann per Los ausgewählt werden.

7.3 Veröffentlichung

- Wir planen, die erste vollständige Version für die Öffentlichkeit zugänglich zu machen. V 1.0 wird entweder auf itch.io (<https://itch.io/>) oder auf Steam (<https://store.steampowered.com/>) veröffentlicht. Welcher Preis dafür festgelegt wird, ist noch unklar.
- Vielleicht werden wir auch nur einen "Buy Me A Coffee"-Button (<https://www.buymeacoffee.com/>) hinzufügen, über den man uns unterstützen kann.
- Der Source-Code bleibt allerdings ziemlich sicher privat. Es ist möglich, dass wir noch darüber diskutieren werden, ob dies auch so bleiben wird.

Teil III

Organisation

Kapitel 8

Technologien

8.1 Unity

”Unity ist mehr als eine Engine”, schreibt Unity auf der eigenen Webseite <https://unity.com/pages/more-than-an-engine>. Fakt ist, Unity ist die marktführende Game-Engine. Die Laufzeit- und Entwicklungsumgebung ist vor allem bei Indie Developern sehr beliebt. Deswegen ist es auch kein Wunder, dass über 50 Prozent aller Spiele mit Unity erstellt wurden. Doch auch für Grafikdesigner, Architekten, Filmentwickler und viele Weitere ist Unity ein sehr mächtiges Programm. Und wenn selbst die Streitkräfte der Vereinigten Staaten Unity nutzen, wieso sollten wir noch weiter suchen? Trotzdem haben wir einige für uns wichtige Vorteile herausgesucht:

- Physik
 - Unity berechnet einen grossen Teil der physikalischen Eigenschaften automatisch. Dies ist für uns von grossem Vorteil, da wir uns zum Glück nicht mit dem Berechnen von Kollisionen herumschlagen müssen. Ausserdem erlaubt uns Unity die Physik nach Bestreben zu ändern und ganz nach unserem Willen zu formen.
- Grafiken
 - Die Animation von Grafiken ist dank Unity ziemlich einfach. So kann man beispielsweise eine Bildersequenz in Unity laden, und Unity formt daraus automatisch eine Animation. Jedes Spielobjekt kann seinen eigenen Animator haben. In diesem kann man verschiedenen Zuständen des Spielobjekts verschiedene Animationen zuweisen. Greift die Truppe zum Beispiel an, wird von der normalen Animation zur Angriffanimation gewechselt.
- Programmierung

- In Unity kann man Spielobjekten selbst geschriebene Programme, sogenannte Scripts, hinzufügen. Diese werden mit C# geschrieben. Dies war von Vorteil, da wir bereits Erfahrung mit C# hatten.

8.2 GitHub

GitHub ist die grösste webbasierte open-source Plattform. Auf ihr kann man Source Code speichern und teilen. Ein grosser Vorteil von GitHub ist, dass man immer zu vergangenen Versionen zurückgelangen kann. Hinzu kommt, dass man verschiedene Versionen des Source Codes in sogenannten Branches speichern und verändern kann. GitHub speichert ausserdem jede Änderung am Code, die durchgeführt wurde. So entsteht eine History, die jede Änderung nachverfolgen lässt. Ein sehr grosser Vorteil von GitHub ist, dass man im Team in Echtzeit am gleichen Code schreiben kann. Unser grösstes Pro-Argument: Backup-Backup-Backup! Alles wird gespeichert, und es gibt keine Gründe, sich die Festplatte mit Backups vollzuschreiben.

Eine Visualisierung von unserem GitHub Repository kann man unter folgendem Link finden.

8.3 Jetbrains Rider

”Rider ist eine unglaubliche .NET-IDE mit der Power von ReSharper”, schreibt Jetbrains auf der eigenen Webseite. Dies können wir nur bestätigen. Rider hat zudem die Unity API integriert. Dadurch hat Rider eine spezialisierte Codeinspektion für Unity. Treten Errors im Unity Editor auf, kann Rider direkt den Ursprung des Fehlers anzeigen. Das Debugging ist dadurch um einiges leichter und mühseloser. Rider warnt User ausserdem vor schlecht geschriebenen Funktionen, die einen Performance-kritischen Effekt auf das Spiel haben könnten.

8.4 Jira

Jira ist ein webbasiertes Programm für Projektmanagement und Zeiterfassung. Dies ermöglichte uns genau zu erfassen, wie viel Zeit wir an einem bestimmten Problem gearbeitet haben. Uns war klar, dass wir eine Art der Zeiterfassung brauchten, denn im Nachhinein ist dies ein Ding der Unmöglichkeit. Das Dashboard ist für berechtigte Personen unter diesem Link einzusehen.

8.5 LaTeX

LaTeX verwendet das Textsatzsystem TeX. LaTeX ist eine Software, die sich von anderen Textprogrammen unterscheidet. Im Gegensatz zu den gewohnten WYSIWYG (”What You See is what you get”) Editoren (wie z.B. Word) braucht man zur Benutzung von LaTeX einen separaten Editor. LaTeX vereinfacht den Umgang mit TeX durch einfachere

Befehle. Der grösste Vorteil an TeX ist, dass man sehr genau einstellen kann, wie alles aussehen soll.

8.6 Stable Diffusion

Stable-Diffusion ist ein State-of-the-Art, Deep-Learning Text-zu-Bild Modell. Es wurde 2022 vom Start-up Stability AI veröffentlicht. Es wird für die Generation von detaillierten Bildern verwendet. Wir wollten seit Beginn der Arbeit eine sehr neue Technologie verwenden. Dann als Stable Diffusion veröffentlicht wurde, war uns klar, dass wir diese Chance nicht verpassen dürfen. Hier ergab sich, dass wir Stable Diffusion verwendet haben, um einen individuellen Hintergrund und ein individuelles Icon zu generieren. Wir nutzten Ideen, eine schlechte Skizze und viele Stunden an Testen und Verbessern, um den Hintergrund und das Icon nach unserem Geschmack zu generieren.

8.7 Visual Studio Code

Visual Studio Code ist ein kostenloser Editor von Microsoft. Ein grosser Vorteil von Visual Studio Code ist, dass man Erweiterungen hinzufügen kann. Die Erweiterung "LaTeX Workshop" hat dabei das spezifische Syntax-highlighting hinzugefügt. Ohne jene Erweiterung sich die Benutzung von LaTeX um einiges schwerer gewesen.

8.8 Gource

Gource ist eine gratis open-source Software, die perfekt für die Visualisierung von GitHub Repositories geeignet ist. Wir haben Gource für die Erstellung eines Videos genutzt, das als Visualisierung unseres Dateisystems dient.

Kapitel 9

Team Management

9.1 Anfänge

- Bereits Mitte Januar 2022 fragte Marc Elia an, ob sie nicht zusammen die Maturitätsarbeit schreiben wollten. Wir hatten beide vor, als Maturaarbeit etwas zu programmieren. Also vereinigten wir unsere Kräfte.
- Die nächste Frage war, was wir genau als Projekt programmieren wollten. Es stellte sich sehr schnell heraus, dass sich Marcs Grundidee (ein Videospiel) perfekt eignen würde. Man kann seiner Fantasie bei der Erstellung eines Videospiels freien Lauf lassen.
- Marc brachte am Anfang direkt den Vorschlag, dass wir unsere Zeit mit Jira tracken würden. Er setzte zudem GitHub und Unity für unseren ersten Gebrauch auf.
- Elia konnte sich sehr schnell in dieser unbekannten Umgebung zurechtfinden. So entstand ein begeisterungsfähiges gleichberechtigtes Duo.

9.2 Betreuungsperson

- Uns wurde vor Beginn der Arbeit mehrfach gesagt, dass einige Schüler*innen Probleme haben, eine Betreuungsperson zu finden. Um sicherzugehen, dass dies uns nicht geschehen würde, suchten wir bereits sehr früh nach einer Lehrperson.
- Elias erster Vorschlag war Herr Albert Kern. Er hat ihn bereits vor längerer Zeit provisorisch für eine individuelle Betreuung angefragt. Was lag näher, als ihn nun auch für eine Gruppenarbeit anzufragen? Wie es der Zufall wollte, begegneten sich Herr Kern und Elia Ende Februar zufälligerweise auf dem Gang und tauschten einige Worte aus. Herr Kern gab allerdings zu, dass er keine Erfahrung mit der Entwicklung von Videospielen hatte und schlug Herrn Martin Hunziker vor. Dieser sei ein begnadeter Videospiel-Fanatiker und wahrscheinlich besser für die Betreuung geeignet.

- Am dritten März schrieben wir per E-Mail eine kurze Anfrage an Herrn Hunziker. Bereits vier Tage später sassen wir mit Herrn Hunziker in seinem Büro und besprachen mit ihm unsere Ideen. Er sagt uns, er habe selbst bisher keine Erfahrung in der Programmierung von Videospielen.
- Eine Woche später gab er sein schriftliches Einverständnis für die Betreuung unserer Maturitätsarbeit.
- Wir informierten ihn regelmässig über unseren Fortschritt und konnten uns bei Fragen an ihn wenden.

9.3 Kommunikation

- Die Kommunikation zwischen Elia und Marc fand vor allem über WhatsApp statt. Dies geschah allerdings nicht nur in Textform, sondern des Öfteren auch in Telefonaten zu allen möglichen, aber auch unmöglichen Uhrzeiten. Zudem konnten wir auf Jira nachschauen, auf welche Aufgabe der Partner seine Zeit getracked hat.
- Zudem trafen wir uns dreimal physisch ausserhalb der Schulzeit, um unsere Arbeit zu organisieren.
- Wir müissen allerdings zugeben, dass die Kommunikation teilweise nicht ausreichend vorhanden war. Manchmal hatten wir keine Ahnung, woran genau der Partner arbeitete. Wir wussten zwar den ungefähren Arbeitsinhalt, allerdings nicht immer alle Einzelheiten.
- Die Kommunikation zwischen Schüler und Lehrer war unserer Meinung nach mit sechs Treffen genügend gut.

Kapitel 10

Time Management

10.1 Roadmap

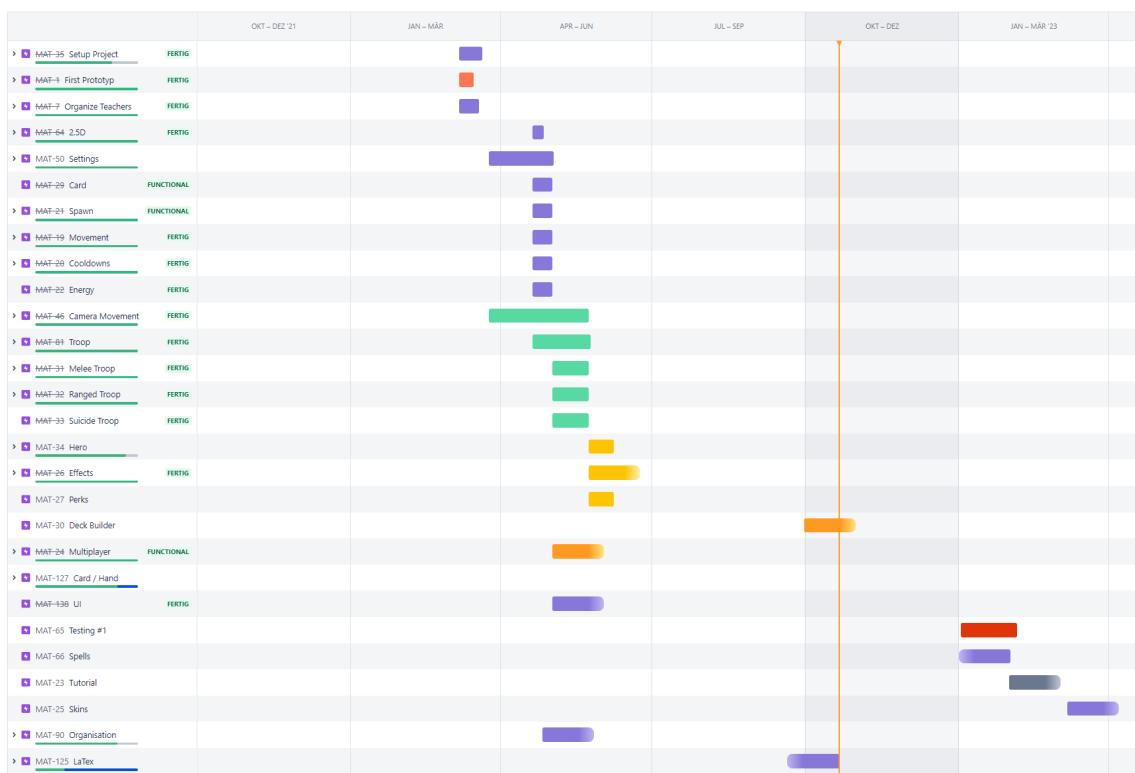


Abb. 10.1: Roadmap (auf Jira)

10.2 Time-Tracking

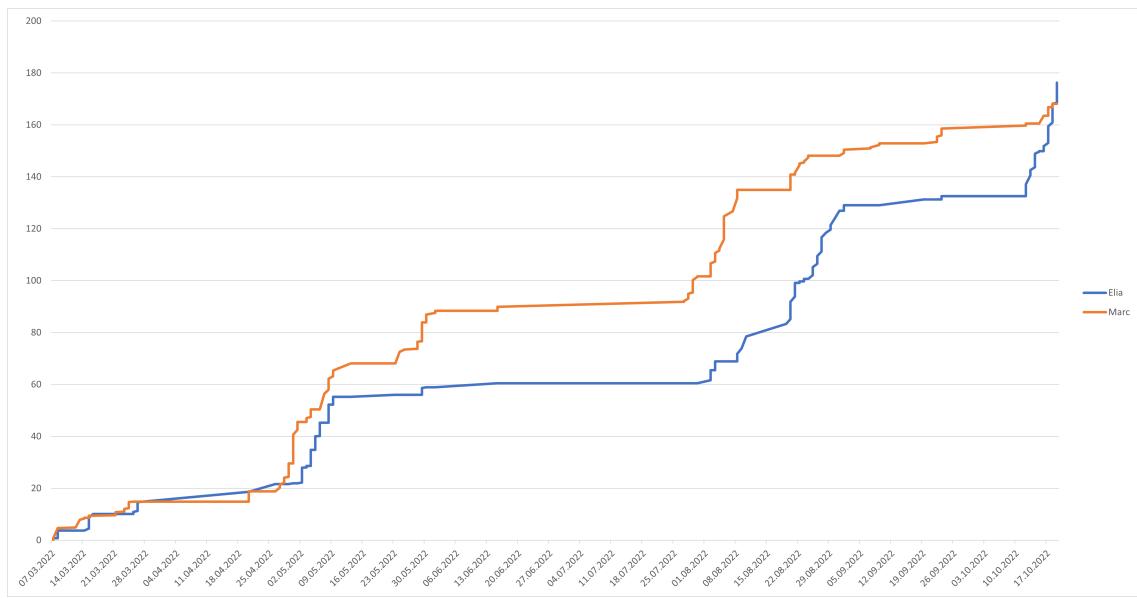


Abb. 10.2: Zeitaufwand seit Beginn der Arbeit

DA NO EXPECTED IN GRAPH (mind und max 100-150)

TEAM HOURS mind =120, max =150

	getrackte Zeit	erwartete Zeit
Elia		120h
Marc		120h
Team		240h

Abb. 10.3: gesamter Zeitaufwand

Teil IV

Reflexion

Kapitel 11

Elia

11.1 Rückblick

Das Produkt dieser Arbeit erfreut mich sehr. Obwohl dem Prototyp unseres Videospiels noch einige Funktionen fehlen, bin ich doch grundsätzlich mit unserem Spiel zufrieden. Dies ist das grösste und zeitaufwendigste Projekt, an dem ich jemals gearbeitet habe. Unser GitHub Repository enthält mehr als 40'000 Zeilen an C# Source-Code. Das ganze Repository ist ca. 1.1 Millionen Zeilen lang und mehrere Gigabytes gross (einschliesslich DLL's etc.).

Alleine hätte ich dies nie geschafft, deshalb bin ich besonders froh, dass wir dies im Team verwirklichen konnten. Die Gruppenarbeit hat mir viel Druck von den Schultern genommen, da ich mich bei Problemen immer auf Marc verlassen konnte. Zudem führte die Zusammenarbeit zu vielen guten Ideen, die wir auch nur zusammen als Team in die Tat umsetzen konnten. Durch diese Gruppenarbeit wurde unsere Freundschaft stark gestärkt. Ich habe zu schätzen gelernt, dass ich mich die ganze Zeit voll und ganz auf Marc verlassen konnte.

Obwohl von Anfang an klar war, dass unser Projekt eine sehr aufwendige Arbeit sein würde, habe ich sie doch unterschätzt. Am meisten unterschätzt habe ich allerdings den Arbeitsaufwand der schriftlichen Arbeit. Sie stellte sich als ziemlich grosser Zeitschlucker heraus.

Ich hatte zudem unterschätzt, was mein geplantes Hardwareupgrade für immensen Einfluss auf meinen Arbeitsfluss haben würde. Der Wechsel von meinem Laptop auf einen hochmodernen Computer brachte viele Vorteile mit sich. Die vervielfachte Leistung hatte den Vorteil, dass ich bei Kompilierung des Codes keine halbe Minute mehr warten musste. Stattdessen konnte ich zwei Sekunden später unser Spiel testen. Diese immense Rechenleistung merkte man auch daran, dass sich mein Zimmer durch die Nutzung des Computers im Laufe eines Tages - trotz geöffnetem Fenster - um mehrere Grade erwärmte.

Auch wenn die Fehlersuche teilweise sehr anstrengend und die Tage lang und Nächte kurz waren, hat mir das Programmieren meistens Spass gemacht. Zwischen Mai und August habe ich leider jegliche Motivation verloren, an der Maturitätsarbeit weiterzuarbeiten.

beiten. Die Ursache dafür war, dass es sich anfühlte, als wären wir bereits sehr weit und dass wir ohne Probleme eine Pause einlegen könnten (vgl. Paretoprinzip). Hinzu kam der immense Prüfungsstress dieser Zeit.

Ich fand es schade, dass ich teilweise den ganzen Tag nur im Zimmer sass und an der Maturaarbeit programmierte.

11.2 Nächstes Mal

Für das nächste grosse Projekt würde ich nicht viel anders machen. Ich möchte mir zu Beginn des Semesters einen Plan jedoch erstellen, der festlegt, dass ich pro Woche mindestens eine Stunde arbeiten werde. Mit diesem Plan wäre es auf keinen Fall zu dieser sehr langen Pause (von Mai bis August) gekommen. Wenn ich erneut die Chance hätte, dieses Projekt in einer Gruppenarbeit zu lösen, würde ich diese Chance ohne zu zögern ergreifen. Allerdings würde ich in Zukunft unsere Organisation zwingend verbessern. Eventuell wäre ein drittes Teammitglied auch keine schlechte Idee (vgl. Abschnitt 13.1). Was diese Erfahrung allerdings mit meinem Plan, Informatiker zu werden, geändert hat, kann ich zu diesem Zeitpunkt nicht beantworten. Fakt ist, ich habe nun auf jeden Fall eine genauere Vorstellung, wie die Tage eines Informatikers aussehen. Zudem habe ich eine Menge an Respekt für jeden Spielentwickler gewonnen. Ein Spiel zu entwickeln ist nicht leicht. Deswegen werde ich ab jetzt immer meine Erfahrungen im Hinterkopf behalten und auch immer an das Dev Team denken, wenn ich über einen Bug oder Crash fluche.

Kapitel 12

Marc

12.1 Rückblick

Ich bin äusserst zufrieden mit unserer Arbeit. Allerdings war meine ursprüngliche Vorstellung unseres Spiels ausgereifter. Dies scheiterte aber nicht an der Arbeit von Elia, mir oder uns beiden, sondern einfach an meinem Hochmut. Ich habe vor Jahren versucht, dieselbe Spielidee auf eigene Faust umzusetzen und kam in relativ kurzer Zeit sehr weit. Ich dachte, der Fortschritt wächst linear. Jedoch stiessen auch wir hier auf das Pareto-prinzip: 20% Aufwand gleich 80% Ergebnis und die restlichen 20% Ergebnis benötigen 80% des Aufwands. Der Feinschliff des UI, der Abschluss der schriftlichen Arbeit und ein bugfreies Spiel waren riesige Zeitfresser.

Die Arbeit mit Elia war voll und ganz erfreulich. Es gab keine Probleme und es hat mir durchgehend Spass gemacht. Wir hatten zwar wenig konkrete Gespräche, dennoch fanden wir uns immer wieder im Gespräch über unsere Arbeit. Ich bin extrem froh Elia als Partner angefragt zu haben, und ich denke er war für das Spiel mindestens gleich bereichernd wie ich.

Gegen Ende wurde die Arbeit doch noch stressig. Ich kann nur von Glück reden, dass Elia auch im Endspurt so ein grosses Engagement. Für mich war es bis kurz vor Schluss sehr schwierig wieder in den Flow zu kommen.

12.2 Nächstes Mal

Vieles würde ich genauso wiederholen: zum Beispiel den Start unserer Arbeit, die Arbeit an sich und die Wahl meines Partners. Aber natürlich lief vieles nicht reibungslos. Das grösste Problem war mangelnde Planung und Routine. Folgende 'Systeme' sollen dem bei meinem nächsten Projekt entgegenwirken:

1. Ich werde mir ein Stundenminimum pro Woche setzen. Dies soll nicht die angepeilten Anzahl Stunden darstellen, sondern kontinuierliches Arbeiten garantieren. So wird verhindert, dass ich komplett aus der Arbeit rausfalle und Schwierigkeiten habe wieder reinzufinden.

2. Ich werde ein Reflexionsystem einführen. In diesem soll wöchentlich oder allenfalls monatlich reflektiert werden. Was lief gut? Was lief schlecht? Was kann ich besser machen?
3. Ich werde wöchentlich brainstormen, alleine oder zu zweit. Dies würde dem Projekt deutlich mehr Agilität und Anpassung bringen. Ich bin auch sicher, dass dies zu vielen und sicher auch interessanten Ideen führt.

Diese Punkte hätten alle dabei geholfen, mir das Ziel vor Augen zu führen. Denn genau dieses Ziel verlor ich zeitweise aus dem Blick. Schlussendlich hatte ich weniger Spass und bekam Schwierigkeiten mich zu motivieren.

Da unser Projekt nicht mit der Abgabe abgeschlossen ist, werde ich gemäss dieser Reflexion die oben genannten Strategien versuchen umzusetzen. Nichtsdestotrotz würde ich beim nächstem Mal eine dritte Person mit ins Boot holen.

Trotz der teilweise noch fehlenden Features bin ich stolz auf uns und denke wir haben für zwei Gymischüler ein brillantes Produkt erstellt.

Kapitel 13

Team

13.1 Rückblick

Teamarbeit

Wir verstanden uns während der gesamten Zeit der Arbeit sehr gut. Es kam nie zu zwischenmenschlichen Problemen. Jedoch haben wir viel Potenzial der Teamarbeit nicht ausgeschöpft. Da wir kein gemeinsames Code Ownership hatten, gibt es nun Softwareteile, die nur einer von uns detailliert kennt und versteht. Diese Spezialisierung führte zu einem teilweise getrennten Arbeitsprozess. Zurückführen kann man dies auch auf die zu seltenen Treffen. Hätten wir uns öfter als dreimal getroffen, wären viele Unklarheiten oder Missverständnisse nicht aufgetreten. So waren wir beide des Öfteren unsicher, wie genau wir unsere verrichtete Arbeit dokumentieren sollten. Es stellte sich zudem die Frage, was genau als investierte Zeit geltet. Unsere grössten Fehler waren, dass wir nie Pull Requests und in die letzten zwei Monate nicht mehr mit Feature Branches gearbeitet haben. Die führte zu verminderter Code-Qualität und entsprechend grossem Zeitverlust bei der Fehlersuche.

Projektplanung

Ein Ziel vor den Augen zu haben und eine Vision zu verfolgen ist wichtig. Jedoch soll es immer möglich sein, diese anzupassen. Bereits im März haben wir sehr viel getüftelt und Ideen entwickelt, die zu diesem Zeitpunkt vollkommen irrelevant sein sollten. So hatten wir beispielsweise bereits acht Effekte "erfunden", bevor es einen Prototyp gab. Ohne es zu wissen, haben wir das Wasserfallmodell angewandt. Das Modell wird in der Informatik teils als eher schlecht erachtet, da man sich in der modernen Entwicklung eher auf einen agilen Arbeitsprozess fokussieren sollte. Im Nachhinein müssen wir zugeben, dass wir fast allen möglichen Problemen des Wasserfallmodells über den Weg gestolpert sind (https://de.wikipedia.org/wiki/Wasserfallmodell#Probleme_und_Nachteile).

Priorisierung

Wir hatten eine teils ungenaue Priorisierung. Obwohl wir bereits früh unsere Anforderungen festlegten, haben wir uns in das eine oder andere Detail verbissen. Dies führte dazu, dass die Entwicklung von wichtigen Anforderungen hinter unwichtigeren Funktionen anstand.

Manpower

Wir hatten gegenüber einer standardmässigen Maturitätsarbeit den Vorteil, dass wir zu zweit mehr Manpower mitbringen. Doch selbst unserem dynamische Duo sind zu zweit Grenzen gesetzt. Aus diesem Grund sind wir uns einig, dass eine dritte Person beträchtlich bei der Arbeit geholfen hätte. Eine Möglichkeit wäre gewesen, dass das dritte Teammitglied die ganze gestalterische Arbeit übernommen hätte. Doch auch ein*e weitere*r Entwickler*in hätte massgebend bei der Arbeit ausgeholfen. Zudem denken wir, dass ein Team mit drei Mitgliedern immer noch recht klein ist. Aus diesem Grund hätte ein*e dritte*r Arbeitspartner*in zu keinen grossen organisatorischen Problemen geführt.

Dokumentation

Wir haben den Beginn der schriftlichen Arbeit zu spät angesetzt. Wir hatten den Zeitaufwand für die Strukturierung und gemeinsame intensive Überarbeitung massiv unterschätzt. Entsprechend kamen wir gegen Ende ins Schwitzen. Ein bekanntes Phänomen in der Spielentwicklung, das Crunchtime ([https://en.wikipedia.org/wiki/Crunch_\(video_games\)](https://en.wikipedia.org/wiki/Crunch_(video_games))) genannt wird.

13.2 Nächstes Mal

Treffen

Eine grössere Anzahl an Treffen und Telefonaten würde das gemeinsame Verständnis zum aktuellen Stand verbessern. Gäbe es eine weitere Arbeit, würden wir uns mindestens monatlich treffen, um den Fortschritt und die nächsten Arbeiten zu besprechen.

Mögliche Fragen, die auf diesem Weg einfacher zu klären sind, sind:

- "An welchem Punkt stehen wir genau?"
- "Was haben die einzelnen Teammitglieder in der nächsten Zeit vor?"
- "Gibt es neue wichtige Ideen für das Spiel?"
- "Sind unsere alten Skizzen noch wahrheitsgetreu?"
- "Benötigen wir neue Skizzen?"
- "Was sind die nächsten Schritte?"

Vier-Augen-Prinzip

Üblich bei der Nutzung von GitHub als Team sind sogenannte Pull-Requests. Diese finden statt, wenn ein Mitglied eine Änderung auf den Hauptordner übertragen will. Ein solcher Pull-Request muss dann von einem weiteren Teammitglied genehmigt werden. Der Grundgedanke ist, dass eine zweite Person den Code überprüft und damit mögliche Fehler sowie Verbesserungen erkennt. Damit werden Fehler reduziert und somit auch Zeit gespart. In einer weiteren Arbeit würden wir dieses Prinzip ohne Ausnahmen anwenden.

Priorisierung

Diesen Punkt anzupassen und anzuwenden ist eher schwierig. Mit einer grösseren Menge an Treffen, Besprechungen und Planung könnte man dieses Problem allerdings in der Zukunft minimieren.

Verschwendete Zeit

Ein Zeitverlust durch Fehler sehen wir nicht als "verschwendet", sondern eher als "Zeit des Lernens". Durch das Beheben unserer Fehler haben wir sehr wahrscheinlich am meisten gelernt.

Wir haben allerdings den Fehler gemacht, dass wir uns zu Beginn acht verschiedene Effekte erarbeitet haben. Dies hat uns wertvolle Planungszeit gekostet, die wir an anderen Stellen besser hätten nutzen können.

Teil V

Glossar

Glossar

Client Die Person, die dem Server einer anderen Person als Mitspieler beitritt.

Crossplay Plattformübergreifendes Spielen. Die Möglichkeit das gleiche Spiel auf zwei unterschiedlichen Plattformen zu spielen (hier: Windows und macOS).

DLC DLC steht für "Downloadable Content", also zusätzlich herunterladbare Inhalte. Diese bieten eine Erweiterung zu der "Standard-Version". DLCs sein teilweise recht teuer.

FPS Frames Per Second. Anzahl Bilder pro Sekunde. Höhere FPS ist meistens gleichbedeutend zu einem flüssigeren Spielerlebnis.

GitHub Repository Ein Repository enthält alle Ordner und Dateien des aktuellen Projekts. Innerhalb dieses Repository kann man die ganze bisherige Arbeit anschauen und kontrollieren. Dabei speichert das Repository auch eine Vergangenheit jeder Datei, sodass man Änderungen einfach zurücksetzen kann.

Host Die Person die das Spiel hostet. Sie ist Server und Client.

Ingame Im laufendem Spiel, nicht auf dem Computer oder beim Start oder Schliessen des Spiels.

Recode Komplette Umprogrammierung und Reprogrammierung einer Funktion.

Skins Kaufbare Darstellungen, welche von dem Standard abweichen. In vielen Spielen nur im Tausch mit echter Währung zu erhalten und bringen nur ästhetische Unterschiede.