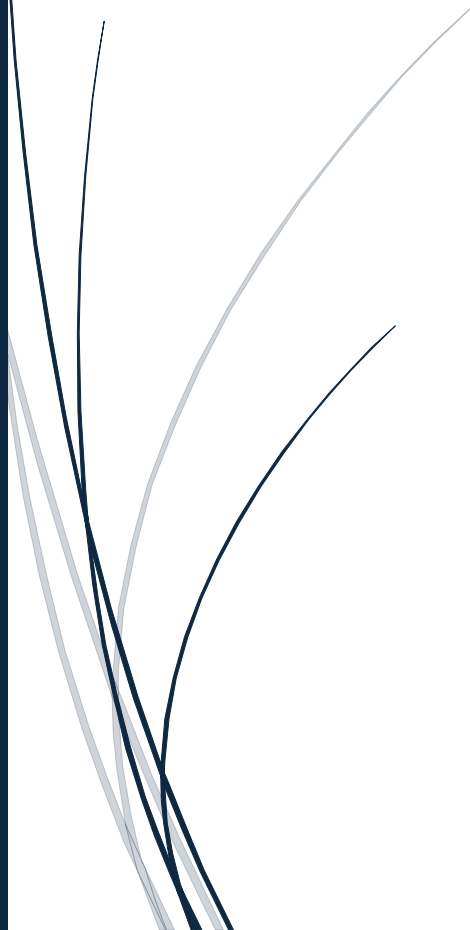


Freitag, 13. Dezember  
2024

# Get Out

Report Anfängerpraktikum



Skrobanek, Timo  
Schäfer, Christian  
Costeniuc, Andrei

## Inhalt

1 - Einführung und Grundlagen .....	3
2 – Dokumentation / Architektur .....	4
2.1 – Storyboard.....	4
2.2 – Use Case.....	6
2.3 – Use Cases Beschreibung .....	6
2.4 – State Machine – Diagramm.....	7
2.5 – Anforderungsanalyse .....	12
2.6 - Doxygen.....	15
3 – Implementierung.....	16
3.1 - Allgemeines .....	16
3.2 - Zuhause.....	17
3.3 - Im Büro.....	17
3.4 - Im Traum .....	17
3.5 - Zuhause (im Traum) .....	19
3.6 - Finale .....	19
3.7 - Hauptmenü.....	18
3.8 - Optionen.....	19
4 – Development Process .....	19
4.1 - Anfangsprozess .....	19
4.1.1 - Meetings .....	19
4.1.2 - Prototype Development .....	20
4.1.3 - Split Task.....	20
4.1.4 - Detailed Requirement Analysis for T.....	21
4.1.5 - Tests for T.....	21
4.1.6 - Code and Documentation for T.....	22
4.1.7 - Test T and Code Review for T .....	23
4.2 - Prozessaktualisierung .....	23
4.2.1 - Meetings .....	23
4.2.2 - Prototype Development .....	24
4.2.3 - Split Task.....	25
4.2.4 - Detailed Requirement Analysis for T.....	25
4.2.5 - Tests for T.....	26
4.2.6 - Code and Documentation for T.....	26
4.2.7 - Test and Code Review for T .....	27

4.2.8 - Grafik des aktualisierten Prozesses .....	28
5 - Fazit .....	29
5.1 - Selbsteinschätzung.....	29
5.2 - Zeitplan .....	29
5.3 - Learnings .....	30
6 - Appendix.....	31
Requirements Document.....	31

# 1 - Einführung und Grundlagen

“The goal of the internship is to learn agile development processes similar to SCRUM and to further optimize the own process in order to achieve a 0-error level.”<sup>1</sup>

Für das Projekt wurde die Spiel-Engine Unity benutzt.

Die wichtigsten Grundbegriffe in Unity sind **Szenen**, **GameObjects**, **Komponenten** und **Scripts**. Ein Spiel in Unity wird in Szenen unterteilt, welche verschiedene Teile des Spiels darstellen, so zum Beispiel ein Level, ein Hauptmenü oder eine Benutzeroberfläche. In unserem Spiel stellen die Szenen Spielbereiche oder Menüs dar. Jede Szene kann eine Vielzahl von GameObjects enthalten, weshalb eine Szene auch gleichzeitig ein Level oder Menü sein kann.

**GameObjects** sind grundlegende Bausteine der Engine. Ein GameObject kann alles Mögliche darstellen, wie ein Charakter, ein Modell, ein Licht oder auch ein UI-Element (beispielsweise ein Button oder Text). Diese GameObjects sind leer, bis ihnen Komponenten hinzugefügt werden, die ihr Verhalten bestimmen.

**Komponenten** sind spezialisierte Module, die einem GameObject hinzugefügt werden, um es funktional zu machen. Eine Transform-Komponente bestimmt z.B. die Position eines GameObjects in der Szene, während Renderer-Komponente dafür sorgt, dass es sichtbar wird. Andere wichtige Komponenten umfassen sogenannte Collider, die zur physischen Interaktion dienen und Rigidbody, die für die physikbasierte Bewegungen sorgen.

**Scripts** sind in Unity erstellte C#-Dateien, die das Verhalten der GameObjects steuern. Sie werden oft verwendet, um benutzerdefinierte Logik zu implementieren, wie etwa die Steuerung eines Spielers oder das Auslösen einer Aktion, wenn auf einem Button geklickt wird.

Der **Szenenbaum** fasst alle GameObjects der Szene zusammen. Das heißt, der Szenenbaum ist eine hierarchische Darstellung aller Objekte, die in einer Szene vorhanden sind. In Unity wird dies als **Hierarchy**-Fenster dargestellt. Dort sieht man die Struktur der GameObjects und ihre Parent-Child-Beziehung.

Unity bietet allgemein eine Vielzahl von **vorgefertigten Komponenten** und **Klassen**, die für das Projekt verwendet wurden. Dazu gehören unter anderem:

- **GameObject, Transform, Rigidbody, Collider** (für die grundlegenden Strukturen und Physik-Interaktionen)
- **UI-Elemente** wie **Button, Text, Image** (zur Erstellung von Benutzeroberflächen)
- **Animator, AudioSource** und **Camera** (für Animationen, Audio und Kamera-Steuerung).

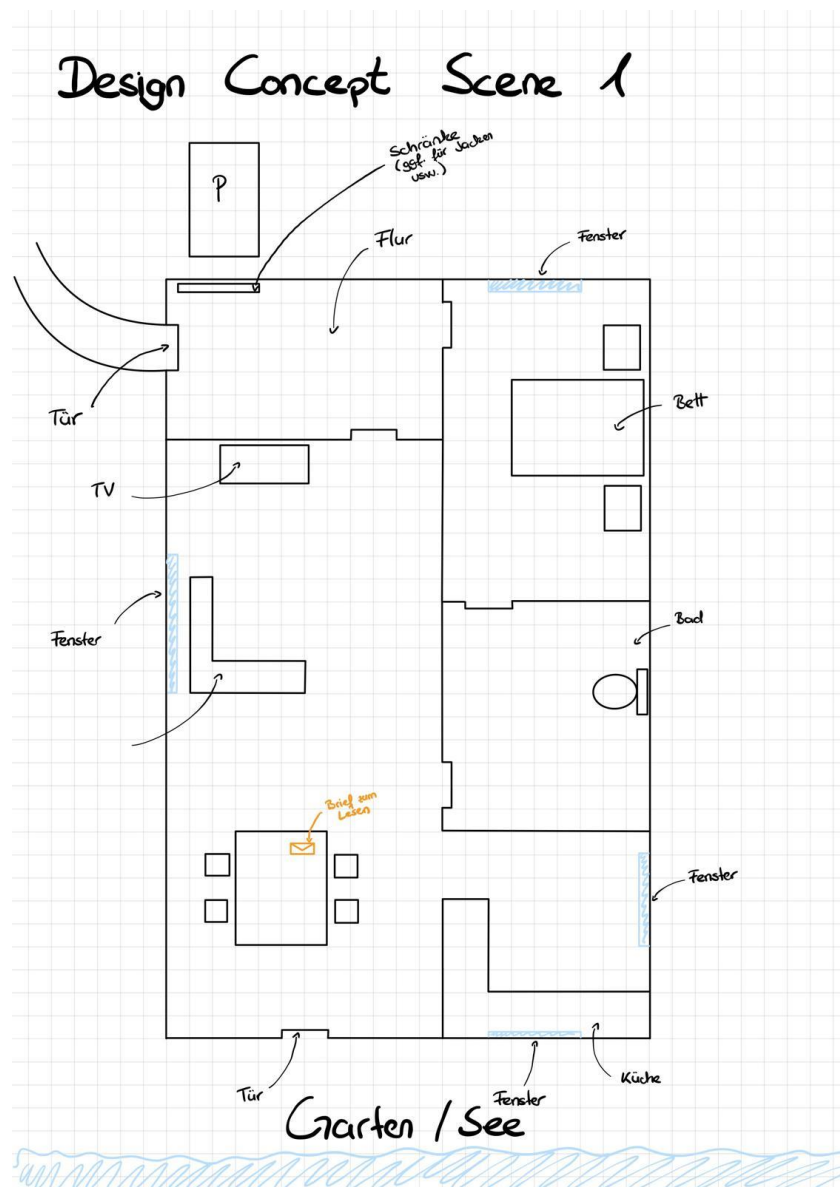
Darüber hinaus verwendet Unity **Packages**, die als zusätzliche Frameworks oder Erweiterungen dienen können. Zum Beispiel bietet das **Cinemachine-Package** erweiterte Kamerasteuerungsfunktionen oder das **Post-Processing Stack** für erweiterte graphische Effekte. Für unser Projekt wurde jedoch kein spezifisches Package oder Framework verwendet.

## 2 – Dokumentation / Architektur

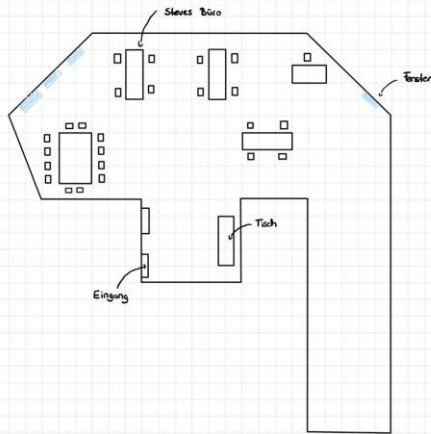
### 2.1 – Storyboard

Eine anfängliche Darstellung des Spiels ist im Storyboard festgehalten. Dies beinhaltet die einzelnen Szenen des Spiels, das heißt, die gewünschte Oberfläche sowie spezifische Spielmechaniken, die später implementiert werden sollten.

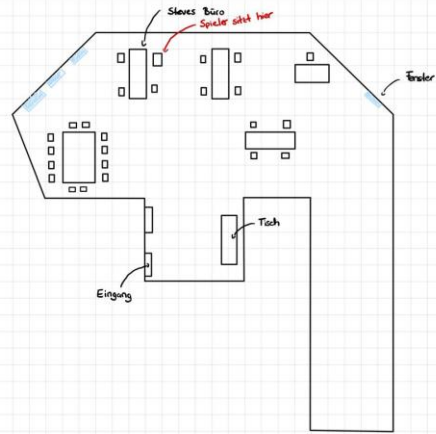
In diesem Storyboard werden die verschiedenen Szenen des Spiels detailliert dargestellt. Jede Szene entspricht einem Abschnitt des Spiels und enthält Informationen zur Atmosphäre, Spielmechanik und optionalen Elementen. Zum Beispiel startet die erste Szene im zuhause des Spielers, wo er unterschiedliche Aufgaben erledigen muss, bevor er das Haus verlassen kann. Die Szenenübergänge sind klar definiert und beinhalten wichtige Details wie die Wechsel von Aufgaben im Haus zu Szenen außerhalb.



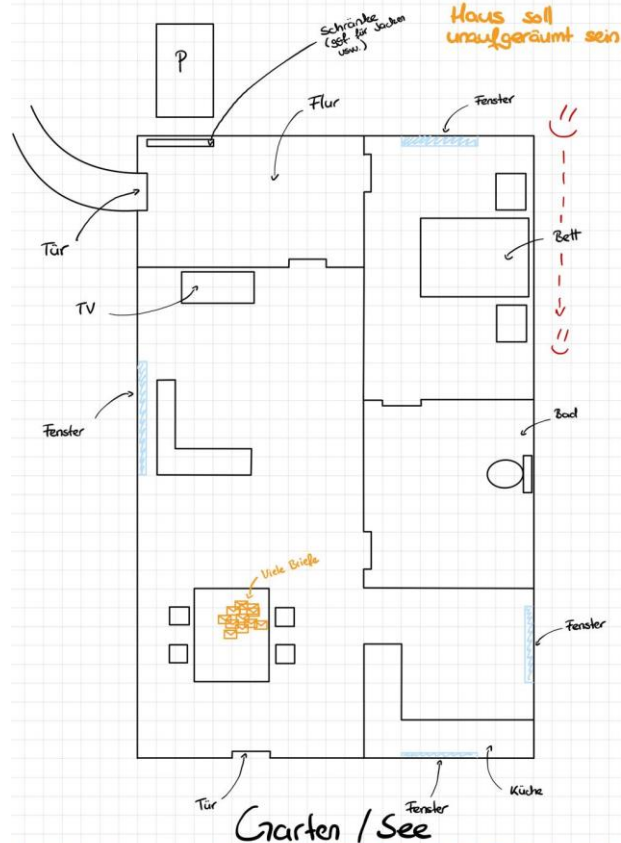
## Design Concept Scene 2



## Design Concept Scene 3



## Design Concept Scene 4



## 2.2 – Use Case

Nachdem das Storyboard erstellt wurde, diente es als Grundlage zur Ableitung der Use Cases. Diese Use Cases werden verwendet, um die Interaktionen des Spielers mit dem Spiel zu definieren. Darüber hinaus lassen sich aus den Use Cases die Funktionen und Features ableiten, die das Spiel enthalten soll, und diese können in der Anforderungsanalyse weiter spezifiziert werden.

Um sicherzustellen, dass die Use Cases diesen Anforderungen entsprechen und ihren Zweck erfüllen, wurden Use Case Diagramme für die jeweiligen Szenen erstellt. Der Akteur in den Use Case Diagrammen ist der Nutzer des Systems, beziehungsweise des Spiels. Die Use Cases sind die Interaktionsszenarien zwischen Akteur und System. Außerdem können sie Beziehungen untereinander haben, welche durch Assoziationen dargestellt sind.

## 2.3 – Use Cases Beschreibung

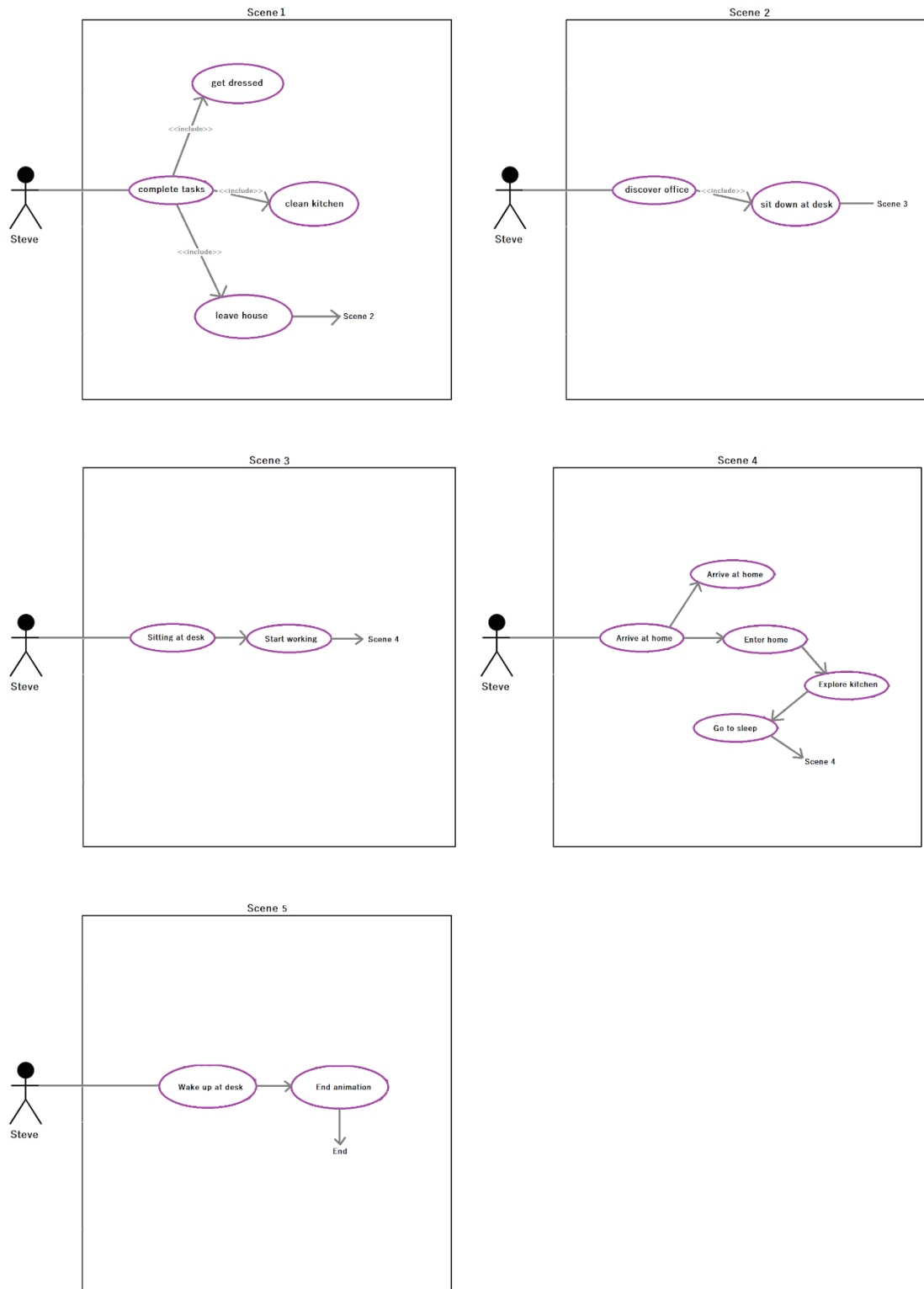
In diesem psychologischen Horrorspiel beginnt die Hauptfigur Steve, ein geschiedener Mann, der mit innerer Zerrissenheit zu kämpfen hat, seinen Tag mit dem Aufwachen auf seinem Sofa. Der Spieler, der die Welt durch Steves Augen sieht, muss alltägliche Aufgaben erledigen, wie sich anziehen, den Müll in seinem Haus aufräumen oder die Küche putzen, bevor er das Haus verlassen und zur Arbeit fahren kann.

Die zweite Szene wechselt in Steves Büro, wo der Spieler das Büro erkunden kann oder sich an seinen Schreibtisch zu setzen kann, wodurch die Handlung fortgesetzt wird.

In der dritten Szene sitzt der Spieler am Schreibtisch und arbeitet. Durch Klicken auf den Bildschirm schläft die Spielfigur ein und tritt in eine surreale Traumwelt ein.

In der vierten Szene kehrt der Spieler in Steves Haus zurück, aber jetzt wird die Atmosphäre noch unheimlicher. Der Spieler wird mit unheimlichen Gestalten bereits außerhalb des Hauses konfrontiert. Wie z.B. ein dunkler lachender Smiley. Dann muss er das Haus erkunden, in dem viele optische Illusionen auftreten, wie z.B. viele Kameras, die an den Wänden hängen oder Tische und Stühle, die an der Decke kleben. Beim Erkunden der Küche erscheint eine Figur, die die geschiedene Frau des Spielcharakters Steve darstellen soll. Der Spieler muss als nächstes ins Schlafzimmer gehen und sich dort ins Bett legen, um wieder aus der Traumwelt zu entkommen.

Das Spiel endet mit einer fünften und letzten Szene: einer Zwischensequenz, in der Steve an seinem Schreibtisch aufwacht und von seinem Chef angemotzt wird. Eine Kameraanimation soll darstellen, dass Steve gerade wieder aufgewacht ist. Die Zwischensequenz wird schwarz und beendet das Spiel mit einer eindringlichen Note.



## 2.4 – State Machine – Diagramm

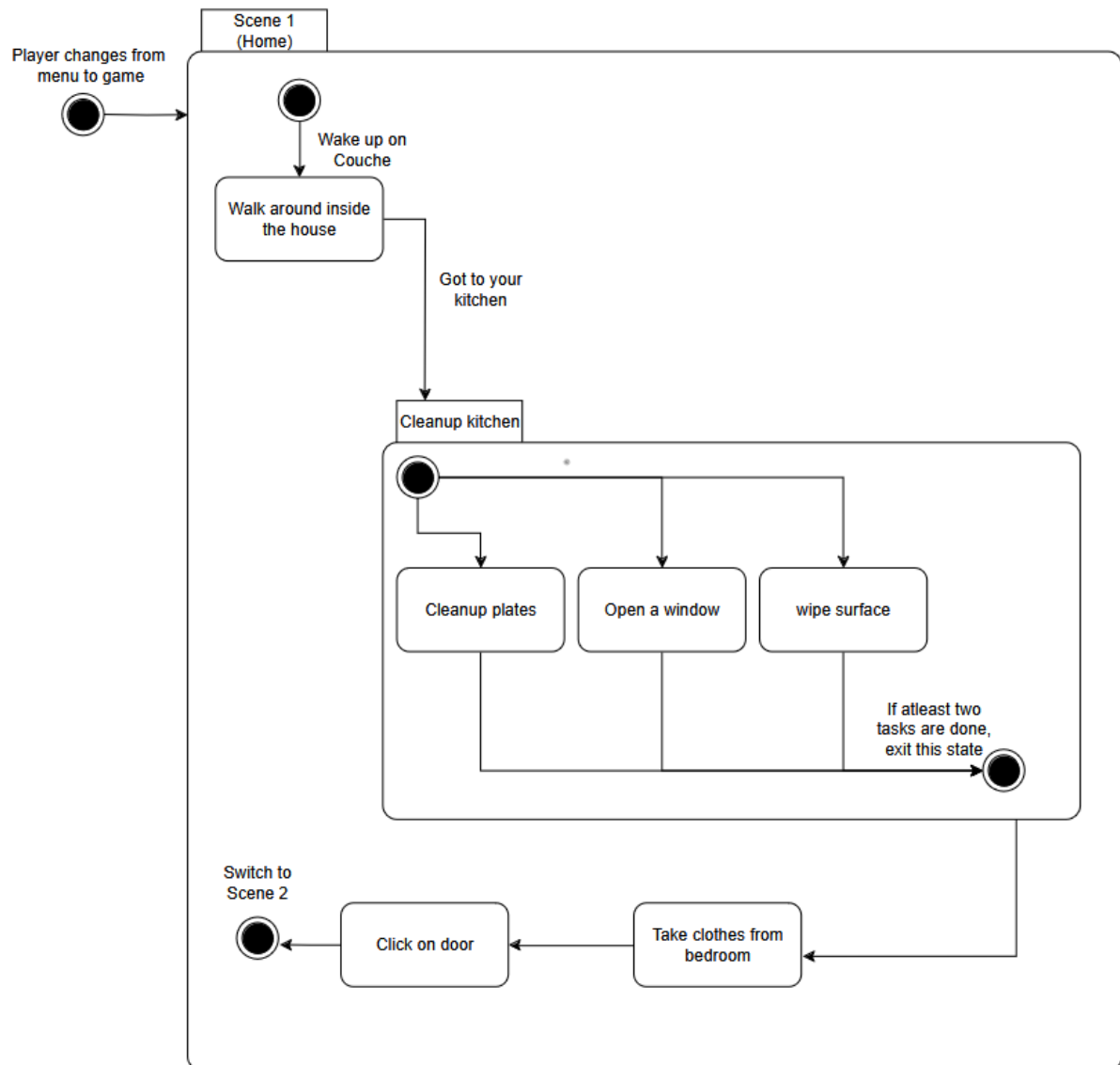
Für unser Spiel haben wir State Machine Diagramme verwendet, um den Ablauf und die Interaktionen in den einzelnen Szenen zu steuern. Eine State Machine ist ein Modell, das den

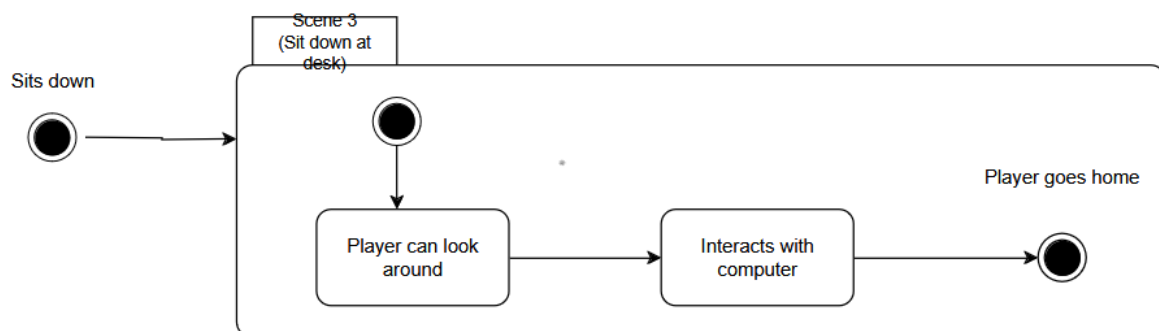
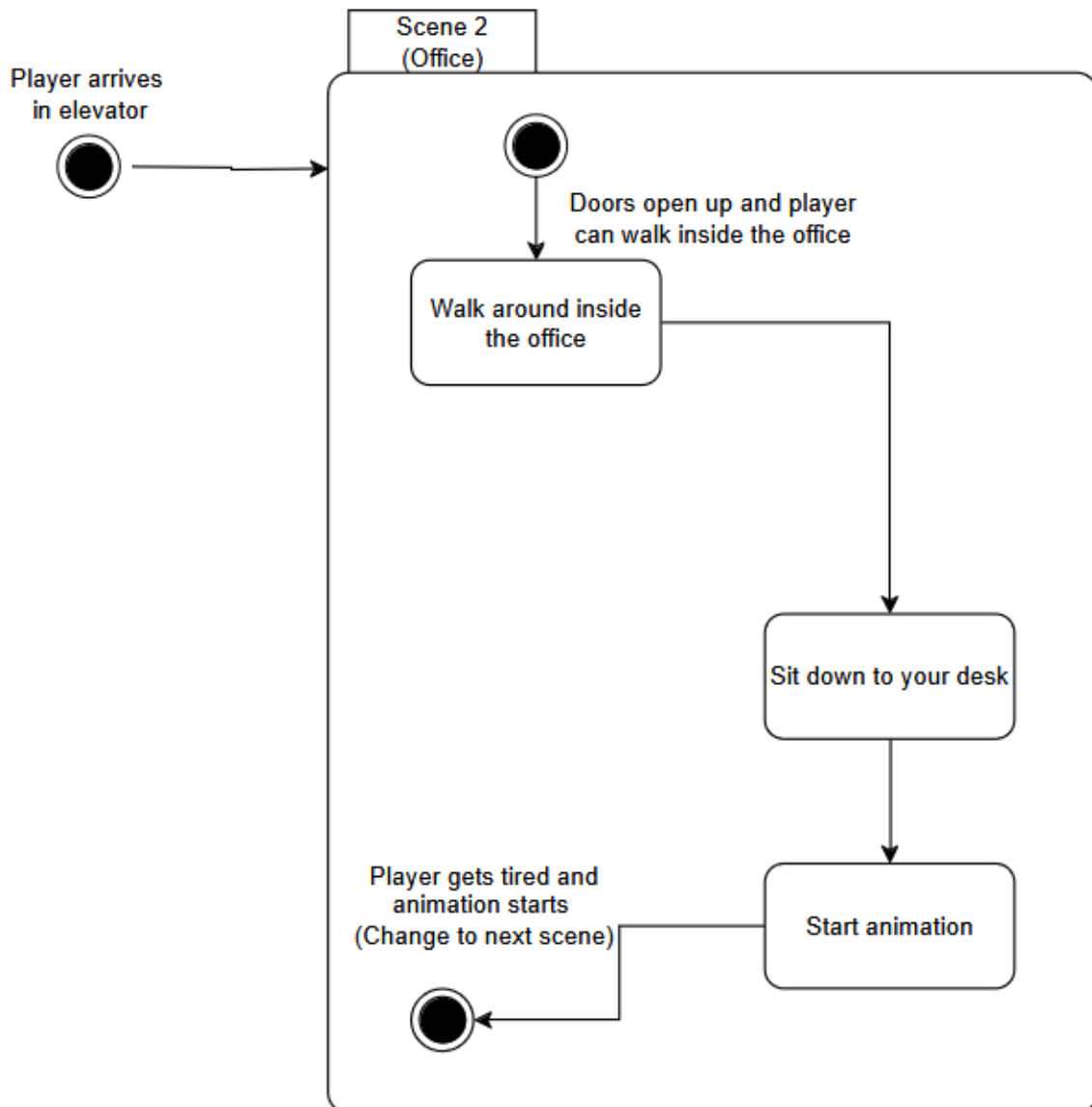


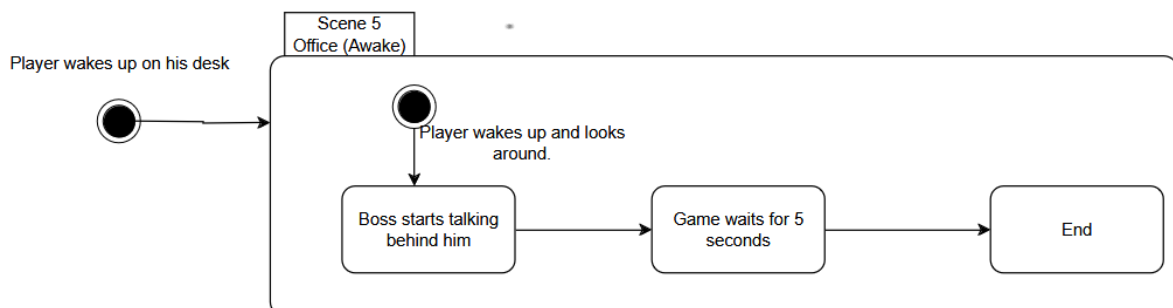
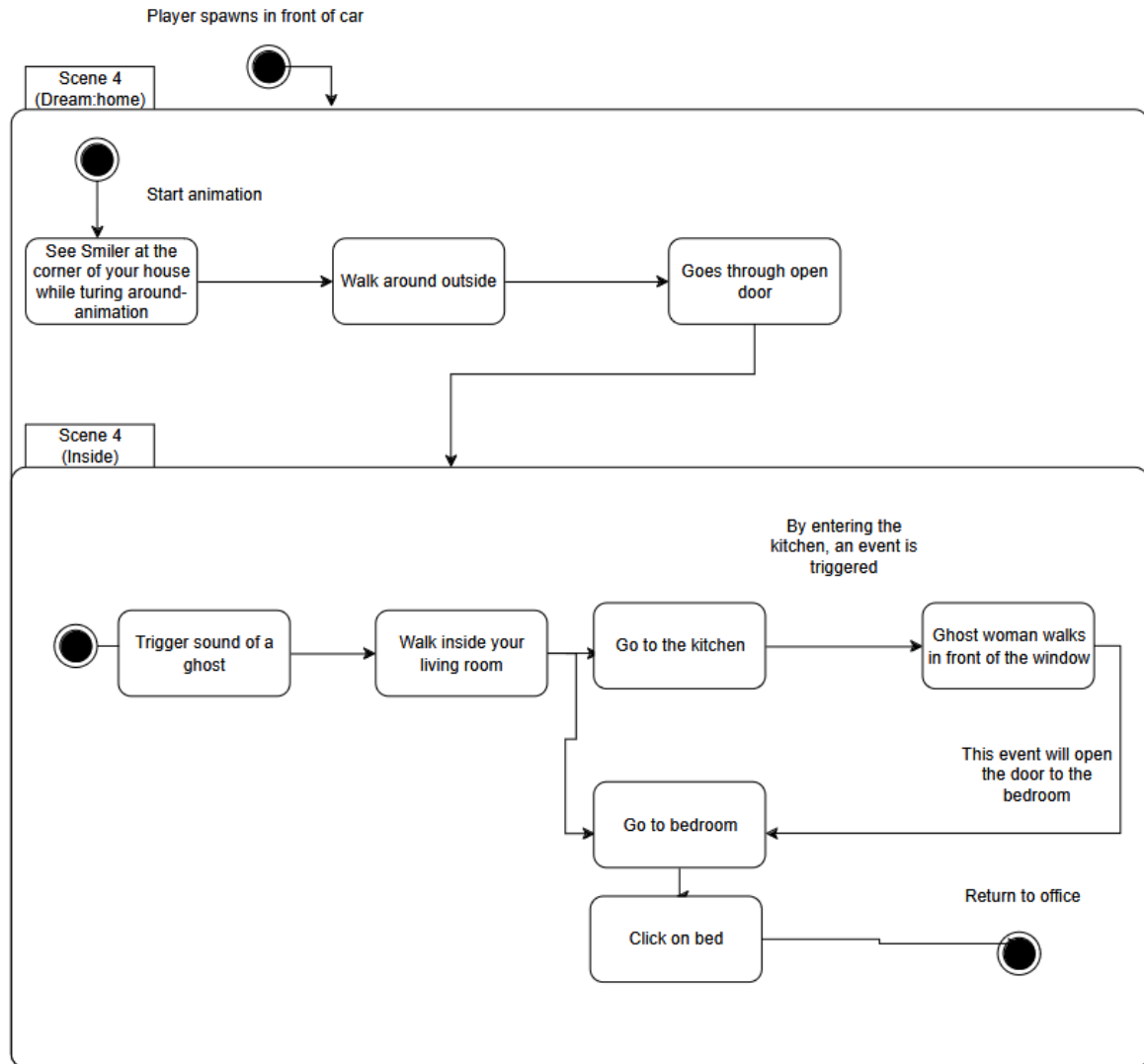
aktuellen Zustand eines Objekts oder Systems beschreibt und darstellt, wie es auf verschiedene Ergebnisse oder Aktionen reagiert. Zustandsübergänge werden durch bestimmte Bedingungen oder Ereignisse ausgelöst, die im Spiel auftreten, wie zum Beispiel das Lösen von Aufgaben oder das Interagieren mit der Umgebung.

Wir haben uns dazu entschieden, State Machines zu verwenden, da es uns geholfen hat, die Interaktionslogik des Spiels zu strukturieren und sicherzustellen, dass das Verhalten des Spiels konsistent und nachvollziehbar bleibt. Dadurch konnten wir sehr einfach den Fortschritt des Spielers durch verschiedene Zustände (wie „Aufwachen“, „Küche aufräumen“, „Haus verlassen“) modellieren und zu kontrollieren, wann und wie der Spieler von einem Zustand in den nächsten übergeht.

Die Entwicklung der State Machines basierte direkt auf den Use Case Diagrammen des Spiels, die die einzelnen Szenen und Interaktionen beschreiben. Für jede Szene gibt es spezifische Aufgaben und Zustände, die den Fortschritt des Spielers steuern. Zum Beispiel in Szene 1, wo der Spieler in seinem Zuhause Aufgaben erledigen muss, gibt es Zustände wie „Anziehen“ oder „Küche aufräumen“. Sobald diese Aufgaben abgeschlossen sind, wird der Zustand „Haus verlassen“ freigeschaltet, der den Übergang in die nächste Szene ermöglicht.







## 2.5 – Anforderungsanalyse

### Zusammenfassung der Funktionalen Anforderungen

Das Spiel basiert auf einem interaktiven Erlebnis, bei dem der Spieler in einer First-Person-Perspektive durch verschiedene Umgebungen navigiert und Aufgaben löst. Der Spieler hat die Möglichkeit, „Objekte“ zu untersuchen und mit ihnen zu interagieren, indem bestimmte Tasten oder die Maus verwendet werden. Jede Interaktion im Spiel löst spezifische Ereignisse aus, die den Fortschritt des Spiels beeinflussen.

Die Spielmechanik basiert auf einer konstanten Bewegung innerhalb der Spielwelt, bei der der Spieler frei durch die Umgebungen navigieren kann. Der Fortschritt des Spiels wird durch verschiedene Speicherpunkte gespeichert, die es dem Spieler ermöglichen, jederzeit zu einem früheren Stand zurückzukehren oder das Spiel von einem Checkpoint aus fortzusetzen.

Eine besondere Anforderung ist die Implementierung eines funktionalen SaveGameManagers, der dafür sorgt, dass der Fortschritt des Spielers in regelmäßigen Abständen gesichert wird. Dadurch kann der Spieler seinen Fortschritt beibehalten, auch, wenn er das Spiel verlässt oder die Szene wechselt.

Nun folgt eine detaillierte Beschreibung der funktionalen Anforderungen und deren Umsetzung in den verschiedenen Bereichen des Spiels.

#### *Generelles*

#### Perspektive und Spielmechanik

Das Spiel ist in 3D designed und hat dementsprechend auch die Bewegungsrichtungen des Spielers zu beachten.

Der Spieler kann sich durch die Verwendung der Tasten W, A, S, D in alle Richtungen bewegen, während bestimmte Aktionen wie das Aufnehmen von Gegenständen oder das Interagieren mit Objekten durch Mausklicks oder Tastendruck ausgelöst werden.

#### Interaktion mit der Umgebung

Das Spiel bietet eine Vielzahl an interaktiven Elementen, die der Spieler erkunden kann. Von alltäglichen Aufgaben, wie das Aufräumen der Küche oder das Einsammeln von Briefen, bis hin zu entscheidenden Momenten, in denen der Spieler innerhalb der Geschichte voranschreitet, entwickelt sich die Geschichte durch die Aktionen, die der Spieler ausführt. Jede Interaktion hat das Potential, die Umgebung zu verändern oder den Fortschritt in der Geschichte zu beeinflussen.

#### Szenen und Zustandsübergänge

Das Spiel ist in verschiedenen Szenen unterteilt, die der Spieler nacheinander durchläuft. In jeder Szene gibt es spezifische Aufgaben, die abgeschlossen werden müssen, um den Übergang zur nächsten Szene zu ermöglichen. Die Zustände des Spiels werden durch eine State Machine verwaltet, die den Fortschritt des Spielers in Abhängigkeit von seinen Aktionen verfolgt.

### Bildschirm Anforderungen

Das Spiel wird nur im Querformat unterstützt und kann im Vollbildmodus gespielt werden.

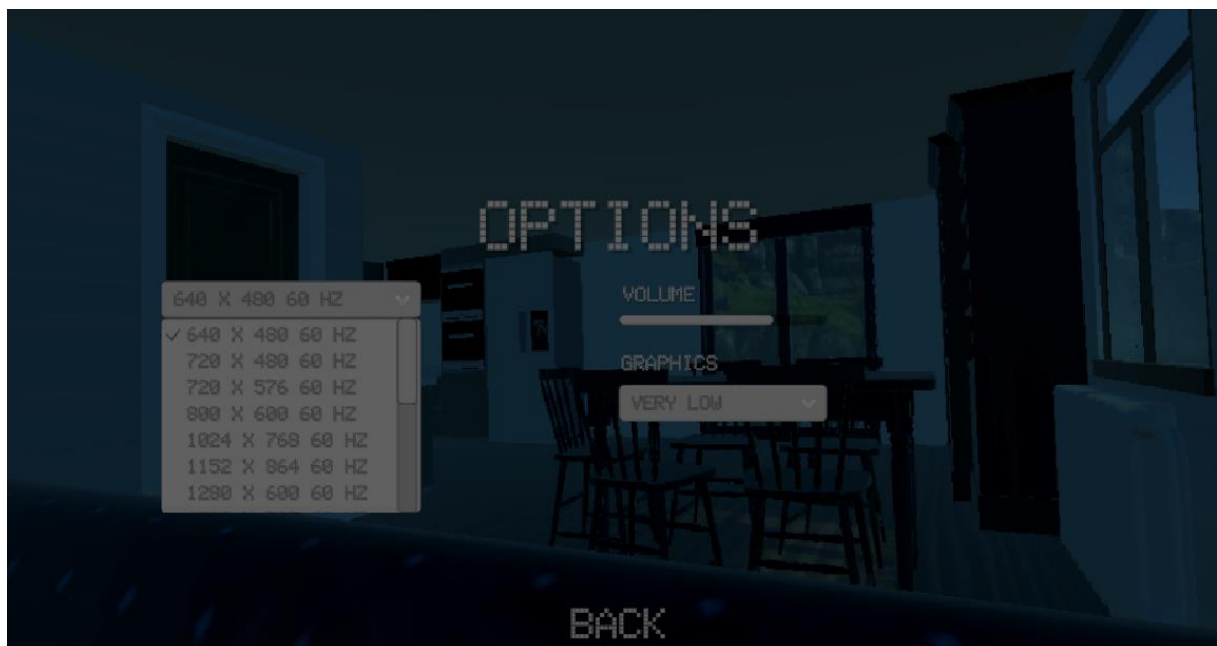
### Willkommen-Screen / Hauptmenü

Das Programm startet mit einem Willkommen-Screen beziehungsweise mit dem Hauptmenü. Das Hauptmenü beinhaltet Buttons für den Spielstart, Optionen und Programmbeenden.

### Optionen

In den Optionen kann die Lautstärke des Spiels angepasst werden, und diese Änderungen werden sofort übernommen. Die Einstellungen bleiben über die gesamte Spieldauer hinweg erhalten.

Darüber hinaus verfügt das Spiel über eine automatische Erkennung der angeschlossenen Monitore. Beim Start des Spiels wird die maximale Auflösung jedes Bildschirms ermittelt, sodass der Spieler die höchste, von seinem Monitor unterstützte Auflösung auswählen kann.



## *Spielanforderungen*

### Hauptmenü

Beim Start des Spiels erscheint unser Hauptmenü (s. oben für Beschreibung). Sobald auf „Play“ gedrückt wird, geht die erste Szene mit einer Animation des Spielcharakters los.

### Szenenauswahl

Der Spieler durchläuft mehrere Szenen, die jeweils eine eigenständige Umgebung und Aufgaben beinhalten. Jede Szene hat spezifische Aufgaben, die der Spieler abschließen muss, um zur nächsten Szene überzugehen.

### Interaktion mit der Umgebung

Der Spieler kann Objekte in der Umgebung durch Mausklicks oder die „E“-Taste auf der Tastatur untersuchen und mit ihnen interagieren. Bestimmte Objekte lösen Ereignisse aus, die den Fortschritt im Spiel beeinflussen.

### Objekt-Interaktionen und Hinweise

Verschiedene Objekte und Details in der Umgebung bieten Hinweise, die den Spieler tiefer in die Geschichte eintauchen lassen. So können beispielsweise verstreute Gegenstände Hinweise auf die Vergangenheit des Hauptcharakters geben.

### Aufgaben- und Zustandsverwaltung

Jede Szene enthält spezifische Aufgaben, die der Spieler erledigen muss. Durch eine State Machine wird der Spielfortschritt erfasst und auf die Aktionen des Spielers reagiert. Aufgaben wie das Aufräumen der Küche oder das Einsammeln von Briefen führen zu neuen Zuständen, die den Übergang zur nächsten Szene ermöglichen.

### Eingabe

Der Spieler bewegt sich mit den Tasten W, A, S, D durch die von den Entwicklern festgelegten Umgebungen. Die Geschwindigkeit kann mit der Shift-Taste geändert werden. Hierbei wird zwischen Gehen und Rennen gewechselt.

## Bezug zur restlichen Dokumentation

### Bezug zum Storyboard

Das Storyboard dient nicht nur als Grundlage für die Anforderungsanalyse, sondern beeinflusst sie auch maßgeblich. Während des Planungsprozesses für die einzelnen Szenen des Spiels wurden viele wichtige Aspekte deutlich, die in den Anforderungen integriert werden mussten. Die visuelle Darstellung des Spielablaufs im Storyboard

ermöglichte es uns, die benötigten Funktionen und Features abzuleiten, um das Spielerlebnis zu optimieren.

### Bezug zum Use Case Diagramm

Die Use Case Diagramme waren ein zentraler Bestandteil der Anforderungsanalyse, da es uns ermöglichte, die verschiedenen Interaktionen und Funktionen des Spiels klar zu definieren. Während der Erstellung des Diagramms haben wir unterschiedliche Aspekte angesprochen, die für das Spiel erforderlich sind. Die Use Case Diagramme halfen uns, die Hauptfunktionen und Interaktionen des Spiels zu verstehen und sicherzustellen, dass alle relevanten Anwendungsfälle berücksichtigt wurden.

### Bezug zum Klassendiagramm

Die Anforderungsanalyse half bei der Erstellung des Klassendiagramms, da darauf Informationen abgeleitet werden konnten.

## Entwicklung

Da die Anforderungsanalyse ein dynamischer Prozess ist, der sich im Laufe der Entwicklung eines Computerspiels ständig weiterentwickelt, gab es über die Zeit auch Veränderungen in unserer Anforderungsanalyse. Im Folgenden sind die wichtigsten Anpassungen aufgeführt, die wir vorgenommen haben:

### Wechsel von Unreal Engine 5 zu Unity

Ursprünglich haben wir uns vorgenommen, das Spiel in Unreal Engine 5 zu entwickeln. Aufgrund von Herausforderungen wie unklarer Dokumentation und Schwierigkeiten mit dem Blueprint-System, haben wir uns entschieden, auf Unity umzusteigen. Unreal Engine 5 entsprach nicht unseren Anforderungen, Unity hingegen bot uns eine stabilere und benutzerfreundlichere Entwicklungsumgebung, eine größere Community und Zugriff auf zahlreiche hilfreiche Ressourcen, was den Entwicklungsprozess deutlich verbesserte.

### Anpassung des Storyboards

Während der Entwicklung stellten wir fest, dass der ursprüngliche Umfang des Spiels ehrgeizig war, daher reduzierten wir das Storyboard und fokussierten uns auf die wesentlichen Elemente der Geschichte. Diese Entscheidung ermöglichte uns, die wichtigsten Spielinhalte effizienter zu entwickeln.

## 2.6 - Doxygen

### Einführung der Doxygen-Dokumentation

Um den Überblick über den Code zu verbessern, führten wir Doxygen für die Dokumentation ein. Zuvor hatten wir Schwierigkeiten, den Zusammenhang zwischen



verschiedenen Code-Teilen schnell zu verstehen, was die Weiterentwicklung erschwerte. Durch Doxygen konnten wir den Code effizienter nachvollziehen und neue Methoden nahtloser integrieren.

Siehe Appendix.

## 3 – Implementierung

Unser Spiel ist im Allgemeinen ein Zustandsautomat. Nach diesem Prinzip haben wir auch die Inhalte der einzelnen Szenen ausgenommen dem Menü implementiert. Jeder Zustand steuert Elemente, die im Spiel eine zeitliche Abhängigkeit haben (Bsp. Intro-Animationen).

### 3.1 - Allgemeines

Für unseren Zustandsautomaten haben wir folgende Zustände definiert.

- SCENE 1
- SCENE1\_INTRO\_ANIMATION
- SCENE1\_CLEANUP\_DONE
- SCENE1\_NOT\_DRESSED
- SCENE1\_COMPLETED
- SCENE2
- SCENE2\_COMPLETED
- SCENE3
- SCENE3\_OUTRO
- SCENE3\_OUTRO\_DONE
- SCENE4\_INTRO
- SCENE4
- SCENE4\_GHOST\_APPEAR
- SCENES

Diese werden innerhalb der Klasse State, der mit StateManager zusammenarbeitet, verwaltet.

Eine weitere Komponente, für unser Spiel bildet die Interaktion mit verschiedenen Spielobjekten - in Unity GameObject genannt – welche wir durch eine Klasse möglich machen, die uns Grundfunktionalitäten dafür bietet.

Zusätzlich gibt es die Logik für die Navigation im Spiel, welche die Spielerbewegung, Kamerarotation und auch die Kollision mit Objekten beinhaltet.

Außerhalb vom eigentlichen Spiel gibt es noch Skripte zum Navigieren im Menü, sowie dem Vornehmen von Einstellungen oder auch das einfache Anzeigen von Text im und außerhalb vom Spiel.

## 3.2 - Zuhause

Das Spiel startet mit der Szene 1 “Zuhause” und beginnt direkt mit einer Animation, welche dem Spieler signalisiert, dass der Charakter gerade aufgestanden ist. Nachdem diese fertig ist, wird dem Spieler die Steuerung freigegeben und dadurch vom Zustand SCENE1\_INTRO\_ANIMATION zu SCENE1 gewechselt. Nun kann sich der Spieler frei in dem vorgegebenen Bereich bewegen und mit verschiedenen Gegenständen interagieren.

Räumt der Spieler seine Wohnung auf, wechselt das Spiel in Zustand SCENE1\_CLEANUP\_DONE woraufhin, der Spieler eine weitere Aufgabe visuell in Form eines Textes erhält.

Hat der Spieler alle Aufgaben erledigt, wechselt das Spiel in den Zustand SCENE1\_COMPLETED. Dadurch werden Animationen gestartet und die nächste Szene geladen.

## 3.3 - Im Büro

In der Szene vom Büro gibt es zwei Zustände. SCENE2 ist der Zustand, welcher dem Spieler erlaubt sich frei zu bewegen und mit Gegenständen zu interagieren. SCENE2\_COMPLETED wechselt in die dritte Szene, sobald der Spieler mit seinem Arbeitsplatz interagiert.

Diese Szene ist von der Implementierung sehr ähnlich zur ersten Szene und hat keine neuen Herausforderungen in der Implementierung mit sich gebracht.

## 3.4 - Im Traum

Nun kommen wir zu der Szene mit dem größten Implementierungsaufwand. In dieser Szene ist der Spieler in der Traumwelt zuhause.

Die Anforderungen an die Szene haben, verschiedene Elemente in die Programmierung einfließen lassen, die in vorherigen Szenen nicht implementiert wurden, weshalb unterandern das Interaktionssystem zu diesem Zeitpunkt grundlegend geändert werden musste, um erweiterte Funktionalität zu gewährleisten.

In dieser Szene gab es nämlich nicht nur Interaktionen durch Anklicken der Maus, sondern ebenfalls durch Position des Spielers und Rotation der Kamera.

Dazu mussten auch hier wieder die Zustände beachtet werden damit Animationen zur Richtigen Zeit ausgeführt werden.

### 3.5 - Zurück im Büro

Die letzte Szene im Spiel hat keinerlei Interaktionsmöglichkeiten mit Ausnahme der Kopfbewegung.

Die Szene behandelt die Situation, dass der Spieler aus seinem Albtraum aufgewacht ist und nun an seinem Schreibtisch sitzt, wo der Chef von hinten auf ihn einredet, dass er doch nicht bei der Arbeit schlafen kann. Wenn dieser Monolog fertig ist, beendet das Spiel indem zu den Credits gewechselt wird.

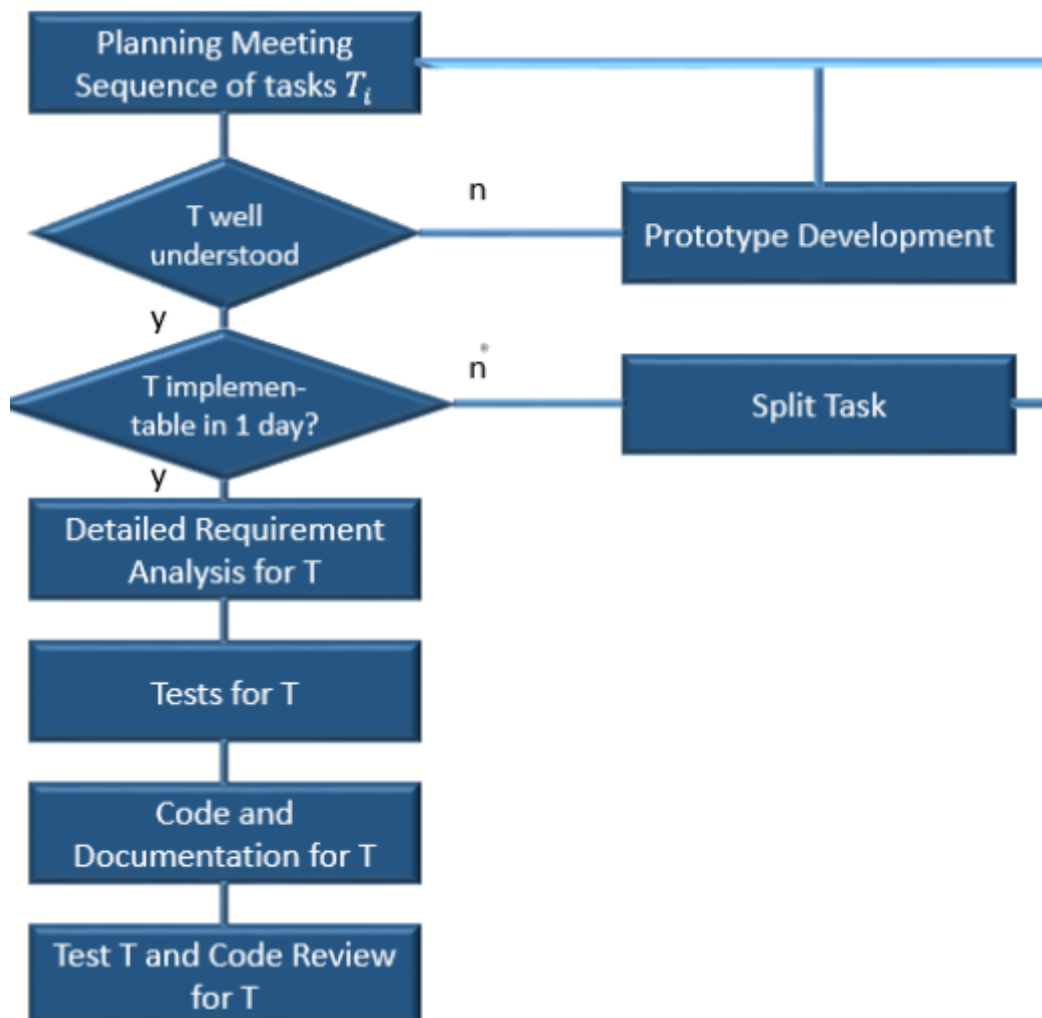
### 3.7 - Hauptmenü

Im Hauptmenü wurden nur die grundlegendsten Funktionen implementiert. Es soll möglich sein, das Spiel zu starten und auch Einstellungen vorzunehmen. Ebenfalls soll das Spiel beendet werden können.

Beim Starten des Spiels soll jedoch immer die erste Szene geladen werden, da wir schlussendlich auf die Implementierung von Save-Games verzichten.

Außerdem gibt es ein extra "Optionen"-Fenster, welches in der Anforderungsanalyse genauer beschrieben wurde.

## 4 – Development Process



### 4.1 - Anfangsprozess

In der Anfangsphase des Projekts sind wir streng nach dem vorgegebenen Development-Schema vorgegangen. Im Verlauf des Projekts haben wir dann Anpassungen und Aktualisierungen vorgenommen, die das Development-Schema effektiver für die Entwicklung unseres Spiels machen.

#### 4.1.1 - Meetings

Rolle:

Ein wesentlicher Bestandteil des Entwicklungsprozesses ist die Koordination der einzelnen Aufgaben und die Vorgehensweise während des Projekts. In den Meetings werden genau diese Fragen beantwortet. Es werden eine Reihe an Tasks definiert und priorisiert, die dann nacheinander individuell abgearbeitet werden. Unklarheiten werden besprochen und es wird sichergestellt, dass das Team weiterhin zielstrebig und erwartungsgemäß arbeitet.

Umsetzung über Videokonferenz:

Durch regelmäßige wöchentliche Meetings auf Discord haben wir uns über den aktuellen Stand des Projekts austauschen können. Wir haben Aufgaben neu priorisiert und diese dann Personen zugeordnet, die diese dann abarbeiten. Mit dem Projektmanagement-Tool Trello haben wir alle erarbeiteten Entscheidungen zeitnah dokumentiert, um einen visuellen Überblick des Projekts beizubehalten. Die grundlegendsten Ziele unserer Meetings war es, sicherzustellen, dass jedes Teammitglied alle Aufgaben versteht und dessen Meinungen und Vorschläge einbringen konnte. Wir haben auch stets die Meetings dazu genutzt, unseren Prozess anhand von aufgetretenen Fehlern anzupassen und zu optimieren.

Umsetzung in Präsenz:

Jedes zweite Meeting haben wir in Präsenz abgehalten. Dafür haben wir uns zu dritt in einem Seminarraum im Mathematikon getroffen und vor Ort an komplizierten Themen gearbeitet. Wir haben den Vorteil der direkten Kommunikation meistens ausgenutzt, indem wir über aufgekommene Implementierungsfehler oder komplizierten Problemen in Präsenz, an meist nur einem Laptop, diskutiert haben.

//TODO Discord als Kommunikationstool anmerken

### 4.1.2 - Prototype Development

Rolle:

Bei komplizierten oder unklaren Anforderungen bzw. Aufgaben können Prototypen erstellt werden, um sicherzustellen, dass die Umsetzung der Implementierung klar verstanden wurde und keine Missverständnisse auftreten. Es können außerdem auch verschiedene Lösungsansätze einfach und effektiv ausprobiert werden. Somit kann man ohne große Arbeit die beste Implementierung ausarbeiten.

Umsetzung:

Wir haben Prototypen genutzt, um die Implementierung grundlegender Funktionen zu erarbeiten. Es war uns sehr wichtig, die häufig aufkommenden Mechaniken im Spiel effektiv programmieren zu können. Vor allem die Interaktionen mit Gegenständen, das Bewegen des Spielers oder das Eventsystem wurden mit Hilfe von Prototypen optimiert und vollkommen nachvollzogen. Dadurch konnten wir sicherstellen, dass diese Funktionen fehlerfrei und effektiv implementiert wurden.

### 4.1.3 - Split Task

Rolle:

Es kann auch vorkommen, dass Anforderungen zu komplex oder zu aufwendig sind, um sie innerhalb eines Tages zu implementieren. In solch einem Fall ist es empfehlenswert, sich für eine Aufteilung der Anforderung in einzelne Teile, zu entscheiden. Somit kann sichergestellt werden, dass die einzelnen Aufgaben nicht einen Arbeitstag (circa 8 Stunden) überschreiten.

Umsetzung:

Um festzustellen, ob eine Aufgabe mehr als einen Tag in Anspruch nimmt, haben wir die Aufgabe im Team nochmals genauer angeschaut. Wir haben aufgezählt, was alles für die Realisierung getan werden muss. Die Einschätzung des zeitlichen Aufwands haben wir basierend auf vergangenen Erfahrungen getroffen.

#### 4.1.4 - Detailed Requirement Analysis for T

Rolle:

Nachdem klargestellt wurde, dass eine Aufgabe innerhalb eines Tages implementierbar ist, wird bei der Detailed Requirement Analysis überprüft, ob alle Details in den Anforderungen der Aufgabe vorhanden sind. Das muss noch vor dem Beginn der Implementierung geschehen, damit dann später im Verlauf keine Missverständnisse entstehen. Denn weitere Details während der Implementierung auszuarbeiten ist extrem ineffizient.

Umsetzung:

Wir haben uns alle Informationen zu der jeweiligen Aufgabe aus den Anforderungen angeschaut und hinterfragt, ob man allein damit die Aufgabe so implementieren kann, wie wir uns das auch vorgestellt haben. Außerdem haben wir überprüft, ob die Anforderungen eindeutig und ohne mögliche Missverständnisse formuliert sind. Falls es Einwände gab, haben wir diese umgehend in den Anforderungen ergänzt bzw. konkretisiert.

#### 4.1.5 - Tests for T

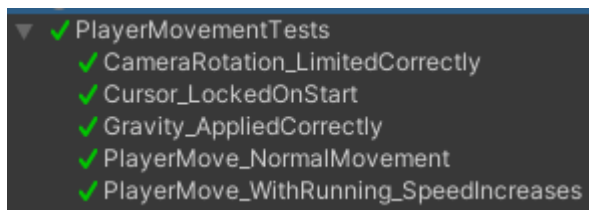
Rolle:

Mit der Entwicklung von Tests kann die Funktionalität der Tasks garantiert werden. Es muss vor der Implementierung noch hinterfragt werden, was für Anforderungen die Task erfüllen muss und wie diese dann getestet werden. Ohne Tests hätte man keine Garantie, dass die geplante Umsetzung der Task auch tatsächlich allen Anforderungen gerecht wird.

Umsetzung:

Zum Testen der Funktionen haben wir uns in erster Linie für Unit-Tests entschieden. Wir haben uns zu einer bestimmten Funktion Test Cases überlegt und eine passende

Testumgebung aufgebaut. Innerhalb dieser Testumgebung haben wir die im Spiel verwendete Funktion ausgeführt und im Nachhinein überprüft, ob die Testwerte den erwarteten entsprechen.



\*Testergebnisse der Unittests in Unity

Für ein besseres User Interface haben wir für die ersten beiden Szenen Usability-tests erstellt, die wir bei bekannten Testpersonen durchgeführt haben. Mit Hilfe dieser Benutzererfahrungen konnten wir überprüfen, ob die Non-Functional-Requirements entsprechend erfüllt wurden.

#### 4.1.6 - Code and Documentation for T

Rolle:

Nachdem Anforderungen entsprechend formuliert wurden und Tests für die Task erstellt wurden, folgt die Implementierung der Task. Das ist der eigentliche produktive Schritt, der im Prozess am Ende das Produkt darstellt. Hierbei geht es darum, dass das aktuelle Projekt um eine bestimmte Funktion iterativ erweitert wird, bis das Endprodukt entstanden ist.

Die Dokumentation des Codes ist wichtig, um die Nachvollziehbarkeit und Wartbarkeit des Codes zu verbessern. Es sollte ohne große Umstände möglich sein, dass noch in ferner Zukunft der Code auch von externen Entwicklern nachvollzogen oder abgeändert werden kann. Dabei spielt eine gründliche Dokumentation eine wichtige Rolle.

Programme, wie Doxygen oder einfache Diagramme, wie z.B. Klassendiagramme helfen bei der Verständlichkeit des Codes.

Umsetzung:

Die Implementierung haben wir entweder allein oder als Paar gemacht, abhängig von der Komplexität der Aufgabe. Bei Meetings in Präsenz hat sich das Pair-Programming angeboten, da es deutlich leichter ist als über Videokonferenzen. Wir haben Unity als Engine benutzt und in C# mit Visual Studio Code als Entwicklungsumgebung programmiert.

Für die allgemeine Dokumentation haben wir Doxygen verwendet und die daraus resultierende index.html Datei auf einem Server gehostet, um immer einen aktualisierten Zugriff auf die Dokumentation von überall aus zu haben. Wir haben zu jeder Funktion in C# einen Doxygen-Kommentar hinzugefügt und die config-Datei

unseren Bedürfnissen entsprechend angepasst. Konkret bei der Programmierung haben wir hauptsächlich mit den Diagrammen gearbeitet, da diese sehr einfach den Sachverhalt dargestellt haben und die Klassen-basierte Vorstellung der Diagramme in C# leicht zu übernehmen war.

#### 4.1.7 - Test T and Code Review for T

Rolle:

Nach einem gesamten Durchlauf des Development-Process für eine Task T wird am Ende nochmal eine Überprüfung der Umsetzung durchgeführt. Hierbei wird sichergestellt, dass der Code auch, den Anforderungen entsprechend, erstellt wurde. Außerdem wird geschaut, dass die Tests auch alle Testfälle abdecken, die Regeln des Clean Codes eingehalten wurden und die Dokumentation vollständig ist. Wenn Fehler oder Probleme gefunden wurden, muss im nächsten Schritt überlegt werden, an welcher Stelle im Prozess dieser Fehler vorgekommen ist und wie man solche Fehler in Zukunft verhindern kann.

Umsetzung:

Wir haben beim Code Review die simple Regel angewandt, dass derjenige, der den Code geschrieben hat, selbst nochmal im Anschluss drüber schaut und hinterfragt, ob der Code alle Anforderungen erfüllt. Nur bei Zweifeln oder Unsicherheiten wurde das Team gefragt. Bei dem Testreview haben wir als gesamtes Team die Tests erstellt und konnten somit auch alle zusammen sicherstellen, dass wir alle Testfälle abgedeckt haben. Für die Dokumentation des Codes haben wir auch immer alle über das resultierende Doxygen Dokument drüber geschaut.

### 4.2 - Prozessaktualisierung

Nach dem Anfangsprozess haben wir revidiert, was für Erfolge und Probleme beim Befolgen des Development-Schemas auftraten. Anhand dessen haben wir das Schema erweitert und für die effektivste und fehlerfreiste Entwicklung gesorgt.

#### 4.2.1 - Meetings

Erfolge:

Durch die wöchentlichen Meetings konnte man sich im Team immer genug austauschen und aufgekommene Probleme zusammen lösen. Wir konnten regelmäßig sicherstellen, dass alle auf dem aktuellen Stand sind und die weitere Vorgehensweise verstanden haben. Durch die Benutzung von Trello konnten wir einzelne Aufgaben übersichtlich anordnen oder Teammitgliedern zuordnen und somit diese nach Prioritäten/Themen abarbeiten.



Besonders zu erwähnen sind die Meetings in Präsenz. Wir haben klare Vorteile feststellen können, wenn man physisch miteinander interagieren konnte. Das Arbeiten zu zweit an einem Laptop, wie z.B. beim Pair-Programming hat uns absolut überzeugt und deshalb haben wir das immer bei komplizierteren Aufgaben praktiziert.

Probleme:

Es gab allerdings auch manchmal Meetings, in denen nicht viel erarbeitet werden konnte. Bei Meinungsunterschieden oder Missverständnissen musste viel Zeit und Energie aufgebracht werden, um diese Probleme zu lösen. Vor allem bei Fragen in der Implementierung oder technischen Problemen war die Kommunikation sehr ineffektiv. Trotz Bildschirmübertragung oder langen Erklärungen über Videokonferenzen konnten Probleme nicht so leicht gelöst werden, wie in Präsenz. Ein weiteres Problem war die falsche Einschätzung von Relevanz einer Anforderung. Wir haben teilweise zu detailreich an einer Anforderung gearbeitet, obwohl diese Entscheidungen zu diesem Zeitpunkt noch absolut irrelevant waren. Die ausgearbeiteten Details mussten im Laufe der Entwicklung verworfen werden, da diese nicht mehr mit dem Gesamtkonzept übereingestimmt haben.

Prozessaktualisierungen:

Wir haben uns gegen die Verwendung von festen Zeitlimits für die Diskussion über ein bestimmtes Thema entschieden, da manche Anforderungen tatsächlich mehr Zeit benötigt haben und trotzdem noch effektiv ausgearbeitet wurden. Da wir mit drei Mitgliedern kein großes Team waren, haben wir einfach alle stets hinterfragt, wie relevant die aktuelle Diskussion auch für die nächsten Aufgaben ist. Bei dem Verdacht, dass es zu sehr ins Detail geht, wurde das ganz klar kommuniziert. Außerdem haben wir in Trello die Aufgaben nach Priorität sortiert und somit wussten wir zu jeder Zeit, auf was wir uns konzentrieren müssen. Um umständliche Erklärungen über Videokonferenzen zu vermeiden, haben wir im Voraus bei komplizierten Funktionen oder ähnliches eine Konferenz in Präsenz geplant, damit wir dann vor Ort das Problem effektiv angehen können.

## 4.2.2 - Prototype Development

Erfolge:

Das Erstellen der Prototypen hat uns nicht nur einen Vorteil in der Implementierung gebracht, sondern hat es uns überhaupt erst ermöglicht, die Funktionsweisen der Mechaniken zu verstehen. Als größtenteils Anfänger in der Spieleentwicklung konnten wir uns die Umsetzung z.B. eines Eventsystems nicht einmal vorstellen. Um dieser Überforderung entgegenzuwirken, haben wir uns strikt daran gehalten, Prototypen zu erstellen. Und mit der Zeit haben wir uns dann im Kleinen eine Vorstellung der

Umsetzung machen können. Das war jedes Mal eine riesige Erleichterung, mit wenig Aufwand und so simpel, wie möglich, die Funktionsweisen zu verstehen.

Probleme:

Manche Prototypen waren nicht umfassend genug. Wir haben versucht, diese so simpel, wie möglich, zu gestalten, was aber in der tatsächlichen Implementierung des Spiels nicht möglich war. Dort wurde es viel umfangreicher und somit auch komplizierter. Es entstanden neue unvorhergesehene Probleme, die den Arbeitsfluss aufgehalten haben.

Prozessaktualisierungen:

Diesen Prozess zu optimieren, war einer der schwersten, da hier unvorbereitete Probleme entstanden, bei einem Thema, wo wir uns nicht auskannten. Wir konnten also nur das Risiko minimieren, indem wir uns intensiver mit der Umsetzung der Mechaniken im Großen auseinandergesetzt haben. Das haben wir meistens mittels Youtube-Tutorials gemacht, da dort anhand von Beispielen manche Mechaniken erklärt wurden.

### 4.2.3 - Split Task

Erfolge:

Das Aufsplitten der Aufgaben hat immer gut funktioniert. Wir haben uns im Team am Anfang jeder Sitzung Zeit genommen und erst mal analysiert, was als nächsten Schritt abgearbeitet werden muss. Beim Aufzählen der einzelnen Schritte, konnte schon ziemlich offensichtlich erkannt werden, ob die Aufgabe an einem Tag machbar war oder nicht. Mit der Zeit wurden die Einschätzungen umso präziser, da wir diese auf Basis von vergangenen Erfahrungen getroffen haben.

### 4.2.4 - Detailed Requirement Analysis for T

Erfolge:

Indem wir zusammen als Team die Anforderungen überprüft haben, konnten wir uns effektiv verschiedene Ansichten der Implementierung überlegen und konnten diese potenziellen Missverständnisse durch klarere Formulierungen verhindern.

Probleme:

Wie auch unten in Kapitel 5 bei Learnings erwähnt, hatten wir hier große Probleme. Wir haben uns sehr schwergetan, sich in einen außenstehenden Entwickler hineinzusetzen. Es gab unklare Aussagen, die nach unserem Erachten eindeutig waren. Manche Missverständnisse haben wir erst bei der Implementierung bemerkt und dann mussten wir mitten in der Entwicklung nochmal Steps zurückgehen und die Details konkretisieren, was sehr viel Zeit und Aufwand gekostet hat.

Prozessaktualisierungen:

Nach den ersten Rückschlägen haben wir uns dazu entschieden, manche Anforderungen nochmal zu überarbeiten mit starkem Fokus auf unklare Aussagen. Wir haben versucht, uns die extremen Missverständnisse auszudenken, um diese dann aus dem Weg zu räumen.

#### 4.2.5 - Tests for T

Erfolge:

Durch das Erstellen der Tests haben wir einen deutlich besseren Einblick bekommen, ob die Anforderungen für die Task klar formuliert wurden, da die Tests auf den Anforderungen basiert haben. Nur mit verständlichen Anforderungen konnten wir passende Tests schreiben.

Probleme:

Zu Beginn war es noch sehr unsicher, für welche Tasks wir Tests erstellen sollen und falls ja, wie intensiv wir die Tests gestalten sollen. Ohne Erfahrung von den weiteren Schritten, die später erst folgen, wie z.B. die Implementierung, war es sehr schwer einzuschätzen, ob uns die erstellten Tests tatsächlich später helfen werden.

Prozessaktualisierungen:

Bevor wir die eigentlichen Tests implementiert haben, haben wir erst mal eine allgemeine Liste aller benötigten Tests verfasst, um einen Überblick zu bekommen. Das hat Struktur in die Abdeckung der Testfälle gebracht und somit konnten wir viel leichter abschätzen, ob wir alle Fälle abgedeckt haben. Erst danach haben wir mit der eigentlichen Implementierung begonnen.

#### 4.2.6 - Code and Documentation for T

Erfolge:

Speziell beim Interaktionssystem haben wir große Erfolge gemacht, da wir direkt von Anfang an eine funktionale Grundlage erstellt haben, die Interaktion jeglicher Art mit Objekten realisiert. Diese konnten wir nutzen, um spezifischere Interaktionen effektiv und schnell zu programmieren. Das hat einwandfrei funktioniert.

Ein weiterer Erfolg war das Übertragen von Prototypen in das fertige Spiel. Bei oft genutzten Funktionen, wie Interaktion oder PlayerMovement haben wir die Prototypen sehr detailliert und weitreichend erstellt. Somit konnten wir ohne großen Aufwand die Prototypen in den Produktiv-Code integrieren und die finale Realisierung war nahezu problemlos.

Die ausführlichen Diagramme, die wir im Voraus erstellt hatten, haben sehr stark geholfen, den Überblick während der Implementierung beizubehalten. Doxygen hat teilweise wie eine Überprüfung gedient. Wir haben unsere erstellten Klassendiagramme

mit den generierten aus Doxygen verglichen und konnten so leicht überprüfen, ob die Programmierung auch unseren ursprünglichen Vorstellungen entsprach. Das war immer eine Befriedigung, wenn das generierte Diagramm dem zuvor geplanten entsprach.

Probleme:

Wie bereits in 2.5 Anforderungsanalyse beschrieben, hatten wir zu Beginn der Programmierung sehr große Probleme mit Unreal Engine 5, da die neuste Version noch keine klare Dokumentation hatte und noch nicht viele Projekte im Internet damit erstellt wurden. Das hat es uns erschwert, als Anfänger in diese Funktionalitäten einzusteigen. Hinzu kommt, dass das Blueprint-System nicht passend für unser programmier-basiertes Projekt war und wir somit von Unreal Engine 5 ablassen mussten.

Was die Dokumentation mit Doxygen angeht, haben wir ungefähr 3 Wochen zu spät angefangen. Wir hatten bereits den ersten Produktiv-Code geschrieben, allerdings ohne Doxygen-Kommentare. Als wir diese dann nachholen wollten, mussten wir zuerst nochmal den früheren Code nachvollziehen und uns zurückerinnern. Das hat sich als sehr ineffizient rausgestellt.

Prozessaktualisierungen:

Nach mehreren Wochen, in denen wir versucht haben, mit Unreal Engine 5 klarzukommen, haben wir uns dazu entschieden, zu Unity zu wechseln, da dort die Dokumentation ausführlicher war und es als einsteigerfreundlicher beschrieben wurde. Dieser verspätete Umschwung erwies sich im Nachhinein als richtige Entscheidung, da wir uns sehr leicht in Unity einarbeiten konnten und damit sehr schnell Erfolge machen konnten.

Sobald wir dann die Doxygen Dokumentation nachbearbeitet hatten, haben wir immer parallel zur Implementierung auch die Doxygen-Kommentare verfasst. Das wurde ein fester und relevanter Punkt in der Implementierung, damit wir immer auf dem aktuellen Stand sind und zu jeder Zeit den aktuellen Code nachvollziehen können.

#### 4.2.7 - Test and Code Review for T

Erfolge:

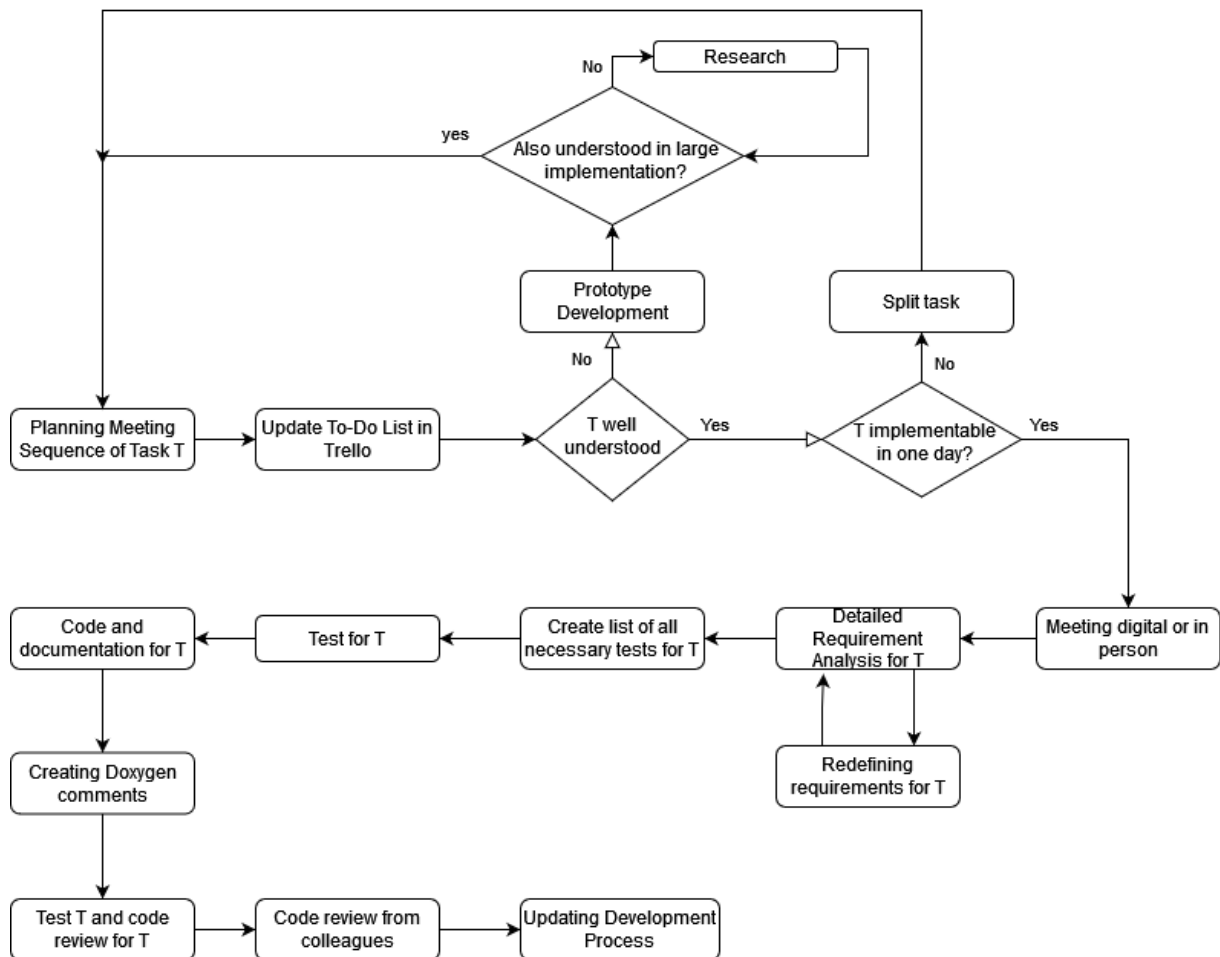
Da die Überprüfung des Codes von derselben Person durchgeführt wurde, die kurz davor den Code geschrieben hatte, war es sehr effizient, da diese Person sich nicht erst noch einarbeiten musste. Außerdem hatten wir fast jedes Mal alle Testfälle abgedeckt, da wir als gesamtes Team dort sehr gründlich durchgegangen sind.

Probleme und Prozessaktualisierungen:

Es hat sich herausgestellt, dass die Überprüfung des selbstgeschriebenen Codes nicht besonders hilfreich ist. Eine weitere Person war hierbei viel wertvoller, da diese neutral

auf den Code schaut und besser den Sinn oder die Code-Struktur hinterfragen konnte. Nachdem dann der Code von anderen angeschaut wurde, haben wir schnell festgestellt, dass die anderen Teammitglieder ebenfalls den Code regelmäßig bewerten sollten. Das haben wir dann auch als festen Bestandteil im Development-Process eingefügt.

#### 4.2.8 - Grafik des aktualisierten Prozesses



## 5 - Fazit

### 5.1 – Selbsteinschätzung

Im Rückblick auf das Projekt ist uns klar geworden, dass eine präzisere Definition der Anforderungen essenziell gewesen wäre. Die Anforderungen an das Spiel wurden von uns zwar recht ausführlich definiert, allerdings führte das trotzdem dazu, dass wir während der Entwicklung häufig auf unerwartete Herausforderungen stießen, insbesondere in Bezug auf den Umfang der Aufgaben und die benötigte Zeit für deren Umsetzung. In einigen Fällen hatten wir das Gefühl, dass wir Aufgaben „on the fly“ anpassen mussten, was zusätzlichen Aufwand und zeitliche Verzögerung verursachte.

Für zukünftige Projekte ist unser Ziel, die Anforderungen noch detaillierter und realistischer zu definieren, um eine präzisere Planung und bessere Ressourcenzuteilung zu ermöglichen. Ein strukturierter Anforderungsprozess würde es uns erlauben, potenzielle Herausforderungen frühzeitig zu identifizieren und zeitaufwendige Umplanungen während der Entwicklung zu vermeiden. Außerdem möchten wir lernen, den Umfang von Aufgaben kritischer zu bewerten und realistischere Zeitpuffer einzuplanen, um flexibel auf unvorhergesehene Probleme reagieren zu können.

Dieser Prozess der Selbstreflexion hat uns gezeigt, wie wichtig es eigentlich ist, einen klar definierten Rahmen für die Projektanforderungen zu haben, um effizienter und gezielter arbeiten zu können. Unser Ziel ist es, aus diesen Erfahrungen zu lernen und sie in zukünftigen Projekten anzuwenden, um die Qualität und die Effizienz unserer Arbeit zu steigern.

### 5.2 – Zeitplan

Insgesamt kann man festhalten, dass unsere ursprünglichen Zeiteinschätzungen definitiv zu optimistisch waren, da wir kaum erkannt hatten, wie umfangreich und anspruchsvoll die meisten der uns zugewiesenen Aufgaben in Bezug auf das Zeitmanagement sei würden. Obwohl wir gegen Ende des Praktikums besser in der Lage waren, unsere Zeiteinschätzungen anzupassen, war unsere erfahrungsgemäß noch immer zu optimistisch, da wir am Ende unserer Arbeitszeiten ziemlich oft immer noch eine Stunde über der ursprünglich geplanten Zeit lagen, und manchmal sogar mehr.

Ein besonderes Beispiel war hier die Modellierung unserer Welt, was sich zeitintensiver herausstellte, als wir ursprünglich gedacht hatten. Wir haben begonnen, unsere Spielwelt, einschließlich des Hauses zu modellieren. Das führte dazu, dass der Aufwand für das Erstellen und Gestalten der Umgebung deutlich größer war, als wir erwartet hatten, und es sah auch nicht besonders schön aus. Die umfangreiche Modellierung hat viele Ressourcen beansprucht, darunter vor allem sehr viel Zeit, was unseren Zeitplan stark belastete. Rückblickend darauf wäre es sinnvoll gewesen, entweder mehr Zeit für

diesen Aspekt einzuplanen oder die Modellierungsaufgaben so aufzuteilen, sodass der zeitliche Rahmen besser eingehalten wird.

## 5.3 - Learnings

Aus diesem Anfängerpraktikum haben wir einige wichtige Erkenntnisse gewonnen, die uns in zukünftigen Arbeiten helfen wird, effizienter und zielgerichteter zu arbeiten:

### Realistische Einschätzung des Arbeitsumfangs und Zeitmanagement

Zu Beginn des Projekts haben wir ehrgeizige Ziele gesetzt und den Zeitaufwand für viele Aufgaben unterschätzt. Während wir in den frühen Phasen den Zeitplan noch einhalten konnten, wurden unsere Zeitvorgaben gegen Ende zunehmend schwammiger. Das hat uns verdeutlicht, wie wichtig es ist, den Arbeitsaufwand realistischer abzuschätzen und ausreichend Pufferzeit für unvorhergesehene Schwierigkeiten einzuplanen.

### Effiziente Organisation von Speicherorten für Dokumente

Im Verlauf des Projekts haben wir festgestellt, dass das Speichern von Dokumenten an zu vielen unterschiedlichen Orten schnell zu Verwirrung führen kann. Wichtige Informationen und Unterlagen gingen verloren und waren schwer auffindbar, was zu unnötigen Verzögerungen führte. Zukünftig möchten wir klare und zentrale Speicherorte festlegen, um die Suche nach Dokumenten zu vereinfachen und wertvolle Zeit zu sparen.

### Flexibilität in der Anfangsplanung

Ein weiteres Learning war, dass eine zu detailreiche Ausarbeitung der Funktionalitäten am Anfang des Projekts nicht immer sinnvoll ist. Während der Entwicklung haben sich einige Anforderungen und Funktionen geändert, was zur Folge hatte, dass viele Details aus der Anfangsphase nochmal überarbeitet werden mussten. In zukünftigen Projekten werden wir darauf achten, flexibler zu bleiben und den Detailgrad der Planung an die jeweilige Projektphase anzupassen, um unnötige Arbeit zu vermeiden.

### Ein besseres Verständnis für die Perspektive des Entwicklers

Ein weiteres Learning aus diesem Projekt ist die Erkenntnis, wie wichtig es eigentlich ist, Anforderungen so zu formulieren, sodass sie für alle Beteiligten klar und eindeutig verständlich sind (insbesondere für Entwickler, die das Hintergrundwissen vom Spiel bzw. vom Code haben). Da wir unsere Anforderungen selbst erstellt und diese direkt umgesetzt haben, traten bei uns keine Verständnisprobleme auf. Allerdings hätten wir stärker darauf achten sollen, die Anforderungen so zu formulieren, sodass sie auch für außenstehende Entwickler, die nur die Anforderungen ohne zusätzliche Erklärungen erhalten, unmissverständlich sind. In zukünftigen Projekten müssen wir daher mehr Aufwand darin investieren, uns in die Perspektive der Entwickler bzw. Programmierer

hineinzuversetzen, dass alle Anforderungen klar, präzise und für andere gut nachvollziehbar sind.

## 6 – Appendix

Doxygen Dokumentation als PDF

Requirements Document