



INTERNATIONALE  
HOCHSCHULE

## Projektplan – Projekt Software Engineering

**Erstellt von:** Jan Wunderlich  
**Matrikelnummer:** 92205002  
**Studiengang:** Master of Science Informatik  
**Tutor:** Prof. Dr. Markus Kleffmann  
**Datum:** 28.12.2023

## **Inhaltsverzeichnis**

1	Ziele, Umfang und angestrebtes Ergebnis des Projekts .....	3
2	Anvisierte Zielgruppe.....	5
3	Potenzielle Projektrisiken und Gegenmaßnahmen .....	6
4	Zeitplan und Meilensteine .....	7

## **1 Ziele, Umfang und angestrebtes Ergebnis des Projekts**

Das Ziel des Projekts besteht in der Entwicklung eines sogenannten Endless-Runner- oder Infinite-Runner-Spiels. In diesem Genre übernimmt der Spieler die Kontrolle über eine Figur, die ununterbrochen läuft, und versucht, so lange wie möglich zu überleben. Dies erreicht er, indem er geschickt sämtlichen Hindernissen ausweicht und auftauchende Gegner bezwingt. Der Spieler verliert, sobald die Spielfigur von einem Hindernis oder Gegner getroffen wird. In diesem Moment endet das Spiel bzw. der aktuelle Versuch. Folgende Komponenten sind integraler Bestandteil des Projekts und sollen in der finalen Anwendung nahtlos integriert sein:

### 1. Grundgerüst:

- Initialisierung des Spiels durch Pygame, Erschaffung eines Fensters und Laden aller erforderlicher Ressourcen, einschließlich Grafiken für Hintergrund, Spieler, Gegner, Power-Ups und mehr.
- Das Design soll über alle Spielelemente hinweg, wie den Hintergrund, die Spieler, die Gegner und das Menü im Cyberpunk-Style gehalten werden.
- Definition von Kernparametern wie die Scrollgeschwindigkeit des Hintergrunds, die Sprunghöhe des Spielers, die Schussgeschwindigkeit und die Häufigkeit von Hindernissen oder Gegnern.

### 2. Spieler:

- Spielerbewegungen einschließlich links, rechts, springen und ducken oder滑行 ermöglichen.
- Umsetzung der Schussmechanik, wobei die Möglichkeit geschaffen wird, mehrere Kugeln gleichzeitig abzufeuern, welche ein Spieler mithilfe eines Power-Ups freischalten kann.

### 3. Gegner:

- Zufällige Erzeugung von Gegnern auf der rechten Seite des Bildschirms.
- Definition und Implementierung von unterschiedlichen Fähigkeiten für Gegner wie Laufen, Springen oder das Abfeuern von Waffen.

### 4. Power-Ups und Items:

- Zufällige Generierung von Power-Ups, die der Spieler einsammeln kann.
- Power-Ups bieten verschiedene Effekte, wie ein zusätzliches Leben oder verbesserte Waffen.

### 5. Spiellogik und Fortschrittsanzeige:

- Anzeige von zurückgelegten Metern sowohl während des Spiels als auch im Endbildschirm mithilfe eines Meterzählers.
- Umsetzung einer Highscore-Verwaltung, um Spielergebnisse zu speichern und zu aktualisieren.

- Spielende-Logik, um ein Spiel zu beenden, wenn der Spieler von einem Gegner getroffen wird oder gegen ein Hindernis läuft (Kollisionserkennung).

6. Startbildschirm und Optionen:

- Hauptmenü-Bildschirm, über den der Spieler das Spiel starten kann.
- Integration von Optionen zur Steuerung der Musik und Soundeffekten.

7. Benutzerinteraktion:

- Tastatursteuerung, um Interaktion mit der Spielfigur zu realisieren.
- Maussteuerung für die Navigation im Spielmenü.

8. Spielneustart, -pause und -beendigung:

- Option zum Neustart des Spiels nach dem Tod des Spielers.
- Spieler können Spiel pausieren und zum Hauptbildschirm zurückkehren.

Diese Komponenten bilden den groben Umfang des zu entwerfenden Endless-Runner-Spiels und können im Laufe des Projekts, sollte ein entsprechender Bedarf erkannt werden, ggf. um weitere Elemente ergänzt werden. Das angestrebte Ergebnis dieses Software Engineering Projekts ist die Entwicklung einer voll funktionsfähigen, ansprechenden und unterhaltsamen Endless-Runner-Anwendung, die alle definierten Komponenten umfasst sowie dem Nutzer ein fesselndes Spielerlebnis beschert.

## **2 Anvisierte Zielgruppe**

Bei der Definition der Zielgruppe für das Endless-Runner-Spiel sind mehrere entscheidende Überlegungen anzustellen, die maßgeblich in die Planung und Entwicklung des Spiels einfließen. Neben der Altersgruppe, die mit dem Spiel angesprochen wird, spielen auch die Interessen und Vorlieben der Nutzer eine besonders wichtige Rolle. Ebenso darf die gewünschte Spielerfahrung nicht außer Acht gelassen werden.

Obwohl das Endless-Runner-Spiel grundsätzlich von einem vielfältigen Publikum gespielt werden kann, richtet es sich primär an Gamer und Personen, die zumindest gelegentlich Computer- oder Videospiele nutzen. Die Hauptzielgruppe sind folglich Jugendliche und junge Erwachsene. Die Spielmechanik und -logik werden bewusst so gestaltet, dass das Spiel leicht zugänglich ist, aber dennoch eine gewisse Herausforderung bietet. Eine progressiv zunehmende Schwierigkeit ermöglicht es sowohl Einsteigern als auch erfahrenen Spielern, sich auf ihrem eigenen Niveau zu verbessern und Fortschritte zu erzielen.

Das im Cyberpunk-Style gehaltene Design des Spiels spricht darüber hinaus insbesondere Science-Fiction-Fans an. Eine futuristische Ästhetik, gepaart mit innovativen Spielelementen, bietet eine einzigartige Spielerfahrung. Die spielerischen, visuellen und inhaltlichen Merkmale der Anwendung werden dabei gezielt ausgerichtet, um die Interessen der beschriebenen Zielgruppe bestmöglich zu bedienen.

### 3 Potenzielle Projektrisiken und Gegenmaßnahmen

Für die Realisierung eines adäquaten Risikomanagements sind drei aufeinanderfolgende Schritte erforderlich. Zunächst erfolgt im Rahmen der Risikoidentifikation eine umfassende Analyse, um potenzielle Risiken während der Projektdurchführung zu identifizieren. Anschließend erfolgt eine Risikobewertung, die unter anderem die angenommene Eintrittswahrscheinlichkeit sowie die möglichen Auswirkungen des Risikos auf Kosten oder Ergebnisqualität ermittelt. Der letzte Schritt besteht in der Maßnahmenplanung, bei der geeignete Gegenmaßnahmen zur Prävention oder Korrektur der Risiken entwickelt werden. Die Ergebnisse dieser Analyse werden in einem Risikoinventar zusammengefasst, das in der nachfolgenden Tabelle abgebildet ist. Die Risikokennzahl setzt sich dabei aus dem Produkt der Eintrittswahrscheinlichkeit und der Schadenshöhe des jeweiligen Risikos zusammen. Die Gegenmaßnahmen sind in 2 Arten, nämlich präventive (p) und korrektive (k) Maßnahmen, untergliedert.

Name des Risikos	Tragweite	Schadenshöhe (1-5)	Eintrittswahrscheinlichkeit (1-5)	Risikokennzahl	Maßnahmen (k oder p)
Technische Komplexität	Schwierigkeiten bei der Implementierung	4	3	12	Entwicklungsprozess in kleine Komponenten unterteilen (p), Funktionalitäten priorisieren und ggf. Abstriche machen (k)
Ressourcenknappheit	Verzögerungen bei Implementierung oder Dokumentation	2	2	4	Effiziente Planung und Priorisierung der Aufgaben (p), Projektdauer verlängern (k)
Unvorhergesehene Änderungen im Projektumfang	Zusätzlicher Arbeitsaufwand und Zeitbedarf	3	4	12	Detaillierte Anforderungsanalyse (p), flexible Anpassung der Anforderungen / Aufgaben (k)
Qualitätsprobleme	Fehler in der Anwendung und ggf. Leistungseinbußen	2	2	4	Einzelnen Funktionen testen (p), intensive Fehleranalyse und -korrektur (k)
Ungenügende Planung	Unklare Ziele und Projektverzögerungen	3	2	6	Durchführung einer gründlichen Projektplanung (p), flexibles Projektmanagement (k)
Ungenügende Testabdeckung	Unentdeckte Fehler und Qualitätsprobleme	4	1	4	Festlegung klarer Testziele und -kriterien (p), Erweiterung der Testabdeckung (k)

Tabelle 1: Risikoinventar

## **4 Zeitplan und Meilensteine**

Die Erstellung eines klaren und verständlichen Zeitplans mitsamt Meilensteinen ist entscheidend für den Erfolg eines jeden Projekts. Um diese Komplexität zu bewältigen und den zeitlichen Ablauf optimal zu planen, wird in diesem Software Engineering Projekt auf ein Gantt-Diagramm zurückgegriffen. Dieses visuelle Instrument ermöglicht nicht nur eine übersichtliche Präsentation der Projektzeitleiste, sondern auch eine gezielte Identifikation von kritischen Meilensteinen sowie Abhängigkeiten und ist aus Übersichtlichkeitsgründen auf der folgenden Seite im Querformat abgebildet.

Bei genauerer Betrachtung des Gantt-Diagramms wird deutlich, dass für die Implementierung des Quellcodes in der Erarbeitungs- und Reflexionsphase zwar eine grobe Zeitspanne vorgesehen ist, jedoch keine spezifischen Startdaten, Dauerangaben oder Priorisierungen für einzelne Komponenten festgelegt sind. Dies resultiert aus der Entscheidung für ein agiles Vorgehensmodell in der Entwicklung, bei dem die exakte Reihenfolge und Priorisierung der Funktionen innerhalb des geplanten Zeitrahmens erst im Verlauf des Projekts festgelegt wird. Weitere Details zu diesem Vorgehensmodell werden in der Projektdokumentation ausführlich erläutert.

Projekt: Software Engineering

## Endless-Runner-Game

Projektanfangsdatum: 13.11.23

The Gantt chart illustrates the project timeline from November 2023 to January 2024, divided into four main phases:

- Konzeptionsphase:** Tasks include Projektplan erstellen, Anforderungsdokument erstellen, and Projekttdokumentation erstellen (erste Version). All tasks are completed by November 13th.
- Erarbeitungs- und Reflexionsphase:** Tasks include GitHub Projekt einrichten, Tools, Libraries aufsetzen und konfigurieren, Quellcode schreiben, Grundgerüst, Spieler, Benutzerinteraktion / Steuerung, Gegner, Spiellokig und Fortschrittsanzeige, Power-Ups und Items, Startbildschirm und Optionen, Spielneustart, -pause und -beendigung, Design Pattern in Projekttdokumentation ergänzen, 2 UML-Diagramme erstellen und in Projekttdokumentation aufnehmen, Testfälle aufstellen, durchführen und Testprotokoll schreiben. Most tasks are completed by December 18th, except for the last two which are ongoing.
- Finalisierungsphase:** Tasks include Anwendung fertigstellen, Dokumentationen aktualisieren / fertigstellen, Benutzeranleitung schreiben, Abstract schreiben, and Puffer. These tasks are scheduled for completion between December 30th and January 10th.
- Puffer:** A final phase from January 10th to January 16th.

Key milestones and events marked on the timeline include:

- Urlaub + Feedback eiholen:** A break period from December 23rd to January 4th.
- Feedback eiholen:** Collection of feedback during the break period.

Meilensteinbeschreibung	Fortschritt	Start	Ende	Duration
Projektplan erstellen	100%	13.11.23	13.11.23	0d
Anforderungsdokument erstellen	100%	13.11.23	13.11.23	0d
Projekttdokumentation erstellen (erste Version)	100%	13.11.23	13.11.23	0d
Github Projekt einrichten	0%	04.12.23	04.12.23	0d
Tools, Libraries aufsetzen und konfigurieren	0%	04.12.23	04.12.23	0d
Quellcode schreiben	0%	05.12.23	05.12.23	0d
Grundgerüst	0%	?	?	?
Spieler	0%	?	?	?
Benutzerinteraktion / Steuerung	0%	?	?	?
Gegner	0%	?	?	?
Spiellokig und Fortschrittsanzeige	0%	?	?	?
Power-Ups und Items	0%	?	?	?
Startbildschirm und Optionen	0%	?	?	?
Spielneustart, -pause und -beendigung	0%	?	?	?
Design Pattern in Projekttdokumentation ergänzen	0%	18.12.23	18.12.23	0d
2 UML-Diagramme erstellen und in Projekttdokumentation aufnehmen	0%	19.12.23	19.12.23	0d
Testfälle aufstellen, durchführen und Testprotokoll schreiben	0%	22.12.23	22.12.23	0d
Anwendung fertigstellen	0%	30.12.23	30.12.23	0d
Dokumentationen aktualisieren / fertigstellen	0%	03.01.24	03.01.24	0d
Benutzeranleitung schreiben	0%	07.01.24	07.01.24	0d
Abstract schreiben	0%	09.01.24	09.01.24	0d
Puffer	0%	10.01.24	10.01.24	0d

Abbildung 1: Zeitplanung als Gantt-Diagramm



INTERNATIONALE  
HOCHSCHULE

## Anforderungsdokument – Projekt Software Engineering

**Erstellt von:** Jan Wunderlich  
**Matrikelnummer:** 92205002  
**Studiengang:** Master of Science Informatik  
**Tutor:** Prof. Dr. Markus Kleffmann  
**Datum:** 28.12.2023

## **Inhaltsverzeichnis**

1	Management Summary .....	3
2	Systemumfang und Kontext .....	4
3	Funktionale Anforderungen .....	5
4	Nicht-funktionale Anforderungen .....	9
5	Glossar .....	10

## 1 Management Summary

Dieses Software Engineering Projekt konzentriert sich auf die Entwicklung eines fesselnden Endless-Runner-Spiels, bei dem der Spieler das Ziel verfolgt, eine möglichst große Distanz zu überwinden, ohne von Hindernissen oder Gegnern getroffen zu werden. Das Herzstück des Spiels liegt in seinem ansprechenden Spielerlebnis, welches die Nutzer dazu ermutigt, ihre eigenen Fähigkeiten zu verbessern und ihren Highscore immer wieder zu übertreffen.

Die Funktionalitäten des Spiels erstrecken sich über verschiedene Aspekte. Der Spieler hat die Möglichkeit, die Spielfigur präzise zu steuern, geschickt Hindernissen auszuweichen und Gegner durch den Einsatz einer Waffe zu besiegen. Das Sammeln unterschiedlichster Power-Ups während des Spiels eröffnet dem Spieler zusätzliche Strategien und Möglichkeiten, seinen aktuellen Versuch zu optimieren. Während eines laufenden Versuchs erhält der Spieler kontinuierlich Rückmeldung über seinen Fortschritt, angezeigt in Form der zurückgelegten Meter und gesammelten Punkte. Diese transparente Darstellung ermöglicht es dem Spieler, seine Leistung zu überwachen und den Anreiz zu schaffen, stetig besser zu werden.

Abseits des eigentlichen Spielgeschehens bietet das übersichtliche Hauptmenü dem Spieler eine Vielzahl von Interaktionsmöglichkeiten. Hier kann er nicht nur seine letzten Läufe und Statistiken einsehen, sondern auch die Laustärke von Ton und Musik anpassen. Zusätzlich hat der Spieler die Option, gesammelte Punkte gegen temporäre Power-Ups einzutauschen, die über mehrere Runden hinweg wirksam sind. Ebenso startet der Spieler von hier unkompliziert ein neues Spiel respektive einen neuen Versuch.

In Ergänzung zu den funktionalen Anforderungen liegt ein besonderer Fokus auf nicht-funktionalen Aspekten wie Leistung, Zuverlässigkeit, Ladezeiten und Benutzeroberfläche. Das Ziel ist es, den Nutzern eine umfassende und zufriedenstellende Spielerfahrung zu bieten, die durch hohe Qualität in der Benutzerinteraktion, eine intuitive Benutzeroberfläche und reibungslose Performance geprägt ist. Die Entwicklung zielt darauf ab, ein durchweg ansprechendes und unterhaltsames Spielerlebnis zu garantieren, das den Erwartungen der Spieler gerecht wird.

## 2 Systemumfang und Kontext

Die Anwendung eines UML-Anwendungsfalldiagramms stellt eine effektive Methode dar, um die Abgrenzungen eines Systems zu definieren und den Umfang des Systems auf prägnante Weise zu veranschaulichen. Die klare visuelle Strukturierung von Akteuren und deren Interaktion mit den verschiedenen Anwendungsfällen schafft Transparenz über die Funktionen und Möglichkeiten, die das System bietet. In der nachfolgenden Abbildung ist daher ein UML-Anwendungsfalldiagramm dargestellt, das die Kernaspekte des zu entwerfenden Endless-Runner-Spiels umfasst und dementsprechend Auskunft über die möglichen Aktionen, die der Spieler ausführen kann, gibt.

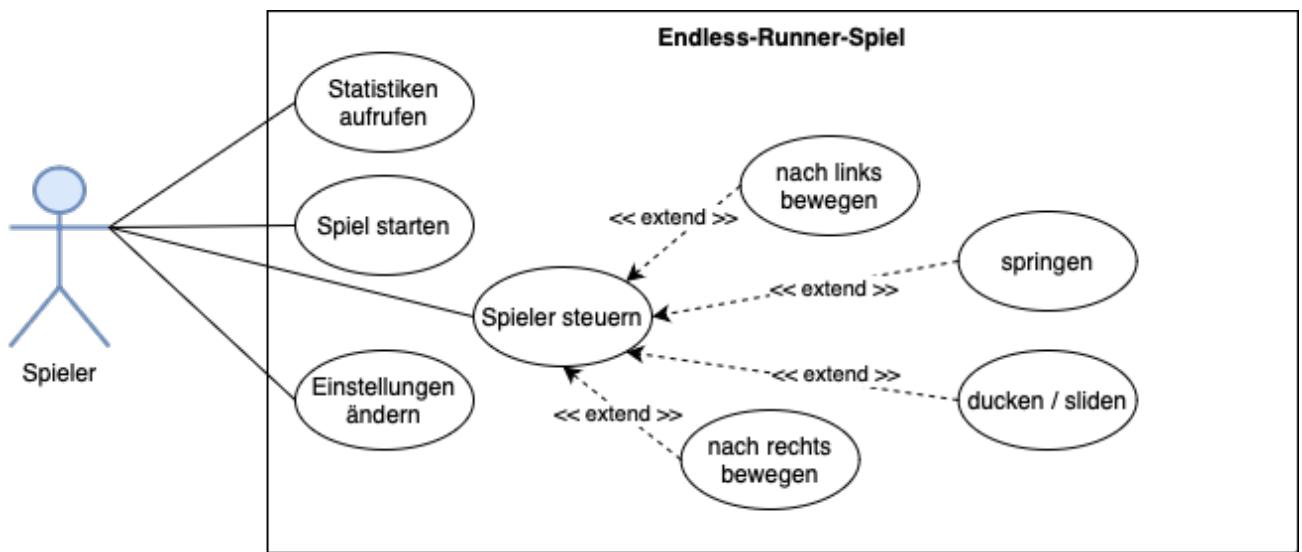


Abbildung 1: UML-Anwendungsfalldiagramm

In diesem UML-Anwendungsfalldiagramm repräsentiert das Endless-Runner-Spiel die klaren Grenzen des Systems. Der einzige Akteur ist der Spieler bzw. Nutzer, der durch die Interaktion mit dem System verschiedene Anwendungsfälle auslöst. Diese Anwendungsfälle umfassen "Statistiken aufrufen", "Einstellungen ändern" und "Spiel starten".

Der Anwendungsfall "Spieler steuern" nimmt eine zentrale Rolle ein, da er sämtliche Bewegungen des Spielcharakters, wie springen, ducken oder in eine Richtung laufen, koordiniert, und stellt daher die Schlüsselkomponente während der Spielausführung dar. Die Anwendungsfälle "Einstellungen ändern" und "Statistiken aufrufen" beschreiben zwei von mehreren Möglichkeiten bzw. Untermenüs, auf welche der User vom Hauptmenü aus zugreifen kann. Zusammengefasst ermöglicht dieses Anwendungsfalldiagramm eine prägnante Darstellung der Systemgrenzen. Es verdeutlicht auf einen Blick, wie der Spieler mit der Endless-Runner-Anwendung interagieren kann, welche Optionen im Menü für den Nutzer verfügbar sind und welche Aktionen im Kontext eines laufenden Spiels ausgeführt werden können.

### 3 Funktionale Anforderungen

Zur präzisen Beschreibung der funktionalen Anforderungen werden sogenannte User Stories verwendet, welche im Zuge des Projektmanagements häufig für diesen Zweck eingesetzt werden. User Stories konzentrieren sich auf spezifische Benutzerinteraktionen und Bedürfnisse, wodurch die Entwicklung unmittelbar auf die Anforderungen des Endnutzers ausgerichtet wird. Definierte Akzeptanzkriterien für jede User Story dienen als klare Maßstäbe zur Beurteilung der erfolgreichen Umsetzung, indem sie messbare Ergebnisse und klare Erwartungen festlegen. Die Einführung von Story Points ermöglicht zudem eine relative Schätzung des Aufwands jeder einzelnen User Story. Dieses Schätzverfahren unterstützt dabei, den Entwicklungsaufwand abzuschätzen und Prioritäten festzulegen. Die nachfolgenden Abbildungen repräsentieren die wichtigsten User Stories dieses Projekts in Form von Karten. Die Story Points reichen hierbei von eins bis fünf und spiegeln eine Bandbreite von geringstem bis höchstem Aufwand wider.

ID: 1	<h3 style="color: #0070C0; margin: 0;">Benutzeroberfläche anzeigen</h3> <p><b>Als Spieler (Rolle)</b> möchte ich eine Benutzeroberfläche für das Spiel sehen (<b>Funktion</b>), damit ich die Anwendung bedienen kann (<b>Nutzen</b>).</p> <p><b>Akzeptanzkriterien:</b></p> <ul style="list-style-type: none"><li><input type="checkbox"/> Fenster öffnet sich beim Starten der Anwendung</li><li><input type="checkbox"/> Fenster besitzt festgelegte Größe und passenden Hintergrund</li></ul> <div style="text-align: right; border-radius: 50%; width: 40px; height: 40px; background-color: #0070C0; color: white; font-weight: bold; font-size: 10px; padding: 5px; margin-top: 10px;">Story Points: 1</div>
-------	---

Abbildung 2: User Story - Benutzeroberfläche anzeigen

ID: 2	<h3 style="color: #0070C0; margin: 0;">Spieler steuern</h3> <p><b>Als Spieler (Rolle)</b> möchte ich meinen Charakter bewegen können (<b>Funktion</b>), damit ich Hindernissen und Gegnern ausweichen kann (<b>Nutzen</b>).</p> <p><b>Akzeptanzkriterien:</b></p> <ul style="list-style-type: none"><li><input type="checkbox"/> Spielercharakter reagiert auf Tastatureingaben und bewegt sich</li><li><input type="checkbox"/> Bewegung des Spielers wird durch Bildschirmgrenzen limitiert</li></ul> <div style="text-align: right; border-radius: 50%; width: 40px; height: 40px; background-color: #0070C0; color: white; font-weight: bold; font-size: 10px; padding: 5px; margin-top: 10px;">Story Points: 3</div>
-------	---

Abbildung 3: User Story - Spieler steuern

ID: 3

## Gegner mit verschiedenen Attacken

### Als Spieler (Rolle)

möchte ich die Fähigkeiten meiner Gegner kennenlernen (Funktion), damit ich meine Strategie anpassen kann (Nutzen).

### Akzeptanzkriterien:

- Gegner sind sichtbar und visuell einfach zu unterscheiden
- Gegner besitzen verschiedene Fähigkeiten (Springen, Schießen)

Story  
Points:  
4

Abbildung 4: User Story - Gegner mit verschiedenen Attacken

ID: 4

## Funktionale Spiellogik

### Als Spieler (Rolle)

möchte ich das ein Spiel korrekt beendet und neugestartet wird (Funktion), damit ich mich verbessern kann (Nutzen).

### Akzeptanzkriterien:

- Spiel wird bei Kollision mit Gegner oder Hindernis beendet
- Neustart startet Spiel / Versuch neu

Story  
Points:  
5

Abbildung 5: User Story - Funktionale Spiellogik

ID: 5

## Fortschritt anzeigen

### Als Spieler (Rolle)

möchte ich einen Meterzähler sehen (Funktion), damit ich meinen Fortschritt verfolgen kann (Nutzen).

### Akzeptanzkriterien:

- Meterzähler erhöht sich basierend auf zurückgelegter Strecke
- Korrekte Zurücksetzung bei Spielneustart

Story  
Points:  
1

Abbildung 6: User Story - Fortschritt anzeigen

ID: 6

## Hauptmenü anzeigen

Als Spieler (Rolle)

möchte ich ein Hauptmenü sehen (Funktion),  
damit ich das Spiel starten oder Optionen auswählen kann (Nutzen).

Akzeptanzkriterien:

- Hauptmenü wird bei Anwendungsstart angezeigt
- Spieler kann Spiel starten oder Einstellungen anpassen

Story  
Points:  
2

Abbildung 7: User Story - Hauptmenü anzeigen

ID: 7

## Power-Ups einsammeln

Als Spieler (Rolle)

möchte ich Power-Ups einsammeln können (Funktion),  
damit die Verbesserung des Highscores leichter fällt (Nutzen).

Akzeptanzkriterien:

- Power-Ups erscheinen zufällig auf dem Bildschirm
- Einsammeln eines Power-Ups schaltet jeweilige Funktion frei

Story  
Points:  
3

Abbildung 8: User Story - Power-Ups einsammeln

ID: 8

## Belohnungen für Leistungen

Als Spieler (Rolle)

möchte ich Belohnungen für meine Leistungen erhalten (Funktion),  
damit ich Anreize für kontinuierliches Spielen habe (Nutzen).

Akzeptanzkriterien:

- Spieler erhält Punkte oder Münzen für erreichte Distanz
- Spieler kann Münzen für Upgrades oder temporäre Power-Ups nutzen

Story  
Points:  
2

Abbildung 9: User Story - Belohnungen für Leistungen

ID: 9

## Waffe abfeuern

Als Spieler (Rolle)

möchte ich eine Waffe abfeuern können (Funktion),  
damit ich Gegner abschießen kann (Nutzen).

Akzeptanzkriterien:

- Spieler kann Waffe per Tastendruck abfeuern
- Gegner verliert leben / verschwindet, wenn er getroffen wird

Story  
Points:  
3

Abbildung 10: User Story - Waffe abfeuern

## 4 Nicht-funktionale Anforderungen

Während funktionale Anforderungen direkt mit bestimmten Features oder Funktionen in Verbindung stehen, beziehen sich nicht-funktionale Anforderungen auf andere Aspekte eines Systems wie qualitative Eigenschaften, Randbedingungen oder Leistungsmerkmale. Hier werden also keine Handlungen oder Ergebnisse definiert, sondern stattdessen wird beschrieben, wie das System bestimmte Qualitäten erfüllen soll. Nachfolgend sind die wichtigsten Punkte aufgelistet, die das Endless-Runner-Spiel, welches in diesem Software Engineering Projekt entwickelt wird, erfüllen soll.

- Leistung
  - Das Spiel sollte flüssig und reaktionsschnell laufen, ohne Verzögerungen oder Ruckeln, um ein optimales Spielerlebnis zu gewährleisten.
- Benutzeroberfläche
  - Die Benutzeroberfläche sollte benutzerfreundlich und intuitiv gestaltet sein, um eine einfache Interaktion und Navigation für Spieler zu ermöglichen.
- Zuverlässigkeit
  - Das Spiel sollte stabil und zuverlässig sein, ohne häufige Abstürze, Fehler oder Bugs, um eine positive Spielerfahrung sicherzustellen.
- Barrierefreiheit
  - Das Spiel sollte barrierefrei und für Spieler mit unterschiedlichen Fähigkeiten oder Erfahrungen geeignet sein.
- Plattformkompatibilität
  - Die Anwendung sollte sowohl unter Windows 10 als auch unter Windows 11 vollumfänglich und ohne Probleme funktionieren.
- Wartbarkeit
  - Der Quellcode des Spiels sollte gut dokumentiert und strukturiert sein, um zukünftige Wartungen und Aktualisierungen bzw. Anpassungen zu erleichtern.
- Ladezeiten
  - Das Spiel sollte schnelle Ladezeiten aufweisen, um die Wartezeit für die Spieler zu minimieren und ein reibungsloses Spielerlebnis zu realisieren.

## 5 Glossar

Begriff	Beschreibung / Erklärung
<b>Akzeptanzkriterien</b>	Klar definierte Kriterien, die erfüllt sein müssen, damit eine User Story als abgeschlossen bzw. erfüllt gilt.
<b>Cyberpunk</b>	Ein Genre in der Science-Fiction, das oft eine dystopische, technologisch fortschrittliche Welt mit starken visuellen Elementen betont.
<b>Endless-Runner</b>	Ein Spielgenre, bei dem der Spieler eine Figur steuert, die unendlich lange durch eine sich ständig verändernde Umgebung läuft.
<b>Gantt-Diagramm</b>	Eine visuelle Darstellung eines Projektzeitplans, die die zeitliche Abfolge von Aufgaben und Meilensteinen zeigt.
<b>Highscore</b>	Die höchste erreichte Punktzahl eines Spielers.
<b>Kollisionserkennung</b>	Die Fähigkeit des Spiels, Kollisionen zwischen verschiedenen Objekten im Spiel zu erkennen, wie z.B. der Spielerfigur und Hindernissen.
<b>Power-Ups</b>	Objekte oder Elemente im Spiel, die dem Spieler temporäre Vorteile oder Fähigkeiten verleihen.
<b>Story Points</b>	Ein Maß für den Aufwand, der benötigt wird, um eine bestimmte User Story abzuschließen.
<b>User Story</b>	Eine kurze, benutzerzentrierte Beschreibung einer Funktion oder eines Features aus der Sicht des Benutzers.



INTERNATIONALE  
HOCHSCHULE

## **Projektdokumentation – Projekt Software Engineering**

**Erstellt von:** Jan Wunderlich  
**Matrikelnummer:** 92205002  
**Studiengang:** Master of Science Informatik  
**Tutor:** Prof. Dr. Markus Kleffmann  
**Datum:** 28.12.2023

## **Inhaltsverzeichnis**

1	Vorgehensmodell .....	3
2	Technologien und Tools .....	5
3	UML-Diagramme .....	6
3.1	Klassendiagramm .....	6
3.2	Aktivitätsdiagramm .....	8
3.3	Sequenzdiagramm.....	9
4	Design- und Implementierungsentscheidungen .....	11
4.1	Singleton Entwurfsmuster .....	11
4.2	Sonstige .....	12
5	Benutzeranleitung .....	14

## 1 Vorgehensmodell

Im Rahmen der Entwicklungsphase für das vorliegende Projekt wird die agile Methode Scrum eingesetzt. Scrum ist ein bewährtes und flexibles Vorgehensmodell, das sich besonders gut für Softwareprojekte eignet, bei denen sich Anforderungen ändern können. Die Wahl von Scrum als Vorgehensmodell für die Erstellung des Endless-Runner-Spiels erfolgt aufgrund spezifischer Überlegungen, die die besonderen Anforderungen dieses Vorhabens berücksichtigen.

Scrum zeichnet sich durch eine hohe Flexibilität aus, die besonders der Natur von Endless-Runner-Games entgegenkommt. Diese Spiele unterliegen häufig sich ändernden Anforderungen, sei es zur Verbesserung des Gameplays, zur Integration neuer Features oder zur Anpassung aufgrund von Testergebnissen. Scrum ermöglicht dementsprechend eine agile Reaktion auf diese Veränderungen, indem sie in den Product-Backlog integriert und in den kurzen Sprints priorisiert werden.

Die iterative und inkrementelle Struktur von Scrum stellt sicher, dass die Implementierung des Endless-Runner-Spiels in handhabbare Sprints unterteilt werden kann. Dies ermöglicht nicht nur eine kontinuierliche Entwicklung, sondern auch die schnelle Sichtbarkeit von Fortschritten. In kurzen Entwicklungszyklen können so regelmäßig spielbare Teile des Spiels erstellt, getestet und verbessert werden. Die Sprintdauer für dieses Projekt wird auf 5 Tage festgelegt, um eine ausgewogene Balance zwischen Effizienz, Flexibilität und regelmäßiger Leistungsbewertung sicherzustellen. Ein weiterer Vorteil von Scrum liegt im klaren Fokus auf die Benutzererfahrung und die kontinuierliche Anpassung des Produkts. Diese Aspekte sind besonders relevant für die Entwicklung eines Endless-Runner-Games, bei dem die Spielerfahrung im Mittelpunkt steht. Die Möglichkeit, Gameplay, Grafiken und Benutzeroberfläche iterativ zu verbessern, sorgt dafür, dass das Spiel am Ende den Erwartungen der Spieler entspricht.

Obwohl Scrum ursprünglich für die Teamarbeit konzipiert wurde, bietet es auch für Einzelentwickler klare Vorteile. Eine transparente Kommunikation sowie regelmäßige Selbstreflexion ermöglichen es, den eigenen Fortschritt zu bewerten, Schwierigkeiten zu identifizieren und gezielte Anpassungen vorzunehmen. Die Verwendung eines Kanban-Boards und des Product-Backlogs innerhalb des Scrum-Frameworks unterstützt zudem die effiziente Verwaltung von Aufgaben und Prioritäten. Das Kanban-Board verbessert die Übersichtlichkeit und zeigt den Status aller Aufgaben des aktuellen Sprints an. Das Product-Backlog dient als zentrale Sammelstelle für Ideen und Aufgaben, die in den kommenden Sprints organisiert werden.

Im Projekt wird die agile Methode Scrum als Einzelperson angewandt, wobei alle drei Scrum-Rollen, also Scrum Master, Product Owner und Developer, in Personalunion ausgeführt werden. In der Rolle des Scrum Masters liegt der Fokus auf klarer Kommunikation, der Lösung von Hindernissen und der Gewährleistung eines reibungslosen Ablaufs. Als Product Owner werden Backlog-Priorisierung, Anforderungsformulierung und die Betonung des Endproduktwerts übernommen. Als Developer wird technisches Know-how für die letztendliche Umsetzung eingebracht. Eine transparente

Kommunikation, regelmäßige Selbstreflexion und effiziente Tools ermöglichen die Dynamik der Scrum-Rollen auch als Einzelperson optimal zu gestalten.

Insgesamt schafft Scrum eine ausgewogene und flexible Umgebung, maßgeschneidert für die Herausforderungen der Einzelentwicklung eines Endless-Runner-Spiels. Es ermöglicht nicht nur eine effiziente und iterative Entwicklung, sondern auch die stetige Anpassung des Spiels, um eine optimale Spielerfahrung zu gewährleisten. Die kurzen, regelmäßigen Iterationen fördern dabei eine dynamische Anpassung an sich verändernde Anforderungen und eine kontinuierliche Verbesserung der Spielerzufriedenheit im Verlauf des Entwicklungsprozesses.

## 2 Technologien und Tools

Um eine effiziente und erfolgreiche Umsetzung des Projekts zu gewährleisten, werden in der Entwicklungsphase verschiedene Technologien, Tools und Frameworks eingesetzt. Als Programmiersprache wird Python gewählt, da persönliche Erfahrungen mit dieser Sprache vorliegen. Python bietet eine klare und leserliche Syntax, was die Entwicklung und Wartung erleichtert. Zudem verfügt Python über umfangreiche Bibliotheken, die speziell für die Spieleprogrammierung genutzt werden können. Pygame (siehe <https://www.pygame.org/wiki/about>) wird als Bibliothek für die Spieleentwicklung integriert, da es sich als äußerst geeignet für die Umsetzung eines Endless-Runner-Games erwiesen hat. Pygame bietet Funktionen und Werkzeuge, die besonders auf die Anforderungen von 2D-Spielen zugeschnitten sind. Die Bibliothek vereinfacht die Handhabung von Grafiken, Animationen, Kollisionserkennung und andere relevante Aspekte der Spieleentwicklung. Die einfache Integration von Bildern, Sounds und weiteren Ressourcen macht Pygame zu einer optimalen Wahl für die Umsetzung eines Endless-Runner-Spiels.

Für die IDE bzw. Entwicklungsumgebung wird PyCharm verwendet. Diese Wahl beruht auf persönlichen Erfahrungen und Vorlieben im Umgang mit PyCharm. PyCharm bietet verschiedenen Funktionen wie Code-Intelligenz, Debugging-Tools oder eine Codeformatierung unter Beachtung des integrierten Standards PEP-8 für Python, die bei der Entwicklung und Fehlersuche helfen. Als zentrales Tool für die Umsetzung des Scrum-Vorgehensmodells wird JIRA eingesetzt. JIRA vereint verschiedene Funktionen, die für die agile Softwareentwicklung entscheidend sind. Es ermöglicht die Verwaltung des Product-Backlogs, die Planung von Sprints und die Organisation bzw. Nachverfolgung von Aufgaben durch ein Kanban-Board. Durch die Integration von JIRA wird eine transparente und strukturierte Umsetzung des Scrum-Frameworks gewährleistet, was die effektive Planung und Durchführung des Entwicklungsprozesses ermöglicht. Für das vorliegende Projekt respektive das zu implementierende Endless-Runner-Game wurde ein neues eigenständiges Projekt in JIRA angelegt, welches unter folgendem Link zu erreichen ist: <https://pse-endlessrunnergame.atlassian.net/jira/software/projects/PSE/boards/1/reports/burnup>.

Als Versionierungstool wird GitHub bzw. Git eingesetzt. GitHub ist eine Plattform, die auf Git basiert und eine zentrale sowie kollaborative Umgebung für die Codeverwaltung bietet. Die Integration von Git ermöglicht eine effiziente Verwaltung von Code, das Nachvollziehen von Änderungen sowie das Wechseln von verschiedenen Entwicklungsständen, sollte ein entsprechender Bedarf bestehen. Für das Endless-Runner-Game wurde bereits ein neues GitHub Repository erstellt, das unter [https://github.com/jan-wun/PSE\\_Endless-Runner-Game](https://github.com/jan-wun/PSE_Endless-Runner-Game) betrachtet werden kann.

Die bewusste Auswahl von Technologien und Tools, darunter Python als Programmiersprache, Pygame für die Spieleentwicklung, PyCharm als IDE, JIRA für das Scrum-Vorgehensmodell und GitHub als Codeversionierungstool, schafft eine harmonische Tool-Landschaft. Diese Tools unterstützen nicht nur die spezifischen Anforderungen des Projekts, sondern gewährleisten auch einen reibungslosen Entwicklungsprozess durch effektive Codeverwaltung und agile Planung.

## 3 UML-Diagramme

### 3.1 Klassendiagramm

Zur Visualisierung der Struktur des geplanten Endless-Runner-Spiels wird ein UML-Klassendiagramm verwendet. Dieses Diagramm bietet einen klaren und verständlichen Überblick über die Klassen, ihre Attribute, Methoden und die Beziehungen zwischen ihnen. Durch die Nutzung eines UML-Klassendiagramms kann folglich ein schneller Einblick in die Architektur des Softwareprojekts gewonnen werden. Aus Übersichtlichkeitsgründen enthält das Diagramm für das vorliegende Software Engineering Projekt, das auf der nächsten Seite im Querformat dargestellt ist, nicht sämtliche Attribute, Methoden und Beziehungen zwischen allen Klassen, sondern beschränkt sich auf das Wichtigste.

Die Struktur des Endless-Runner-Spiels wurde im UML-Klassendiagramm sorgfältig entworfen, um die verschiedenen Komponenten und ihre Interaktionen übersichtlich darzustellen. Die Hauptklassen für die Realisierung des Gameplays sind *Game*, *Player*, *Enemy*, *PowerUp*, *Obstacle*, *Weapon* und *Projectile*. Das *Game*-Objekt koordiniert alle Abläufe im Spiel, während die anderen Klassen spezifische Spielentitäten repräsentieren und von der Klasse *Entity* gemeinsame Attribute und Methoden vererbt bekommen. Die Klasse *Player* modelliert die Spielfigur, die Klassen *Enemy* und *Obstacle* stehen für Gegner und Hindernisse, *Weapon* modelliert die verschiedenen Waffen im Spiel, *Projectile* die Schüsse und die Klasse *PowerUp* die verschiedenen Items, welche das Spiel bietet.

Zusätzlich dazu zeigt das Diagramm verschiedene Menüs, die allesamt von der Klasse *Menu* erben. Die einzelnen Menüs stellen verschiedene Seiten respektive Zustände im Spiel dar. Auch wenn alle Menüs von *Menu* erben, sind lediglich die fünf Klassen *MainMenu*, *SettingsMenu*, *ShopMenu*, *ControlsMenu* und *StatsMenu* nach dem gleichen Layout aufgebaut. Das Game-Over-Menü und das Pause-Menü unterscheiden sich absichtlich, um diese Menüs thematisch bzw. logisch besser vom Rest abzugrenzen zu können.

Neben den Klassen für die Implementierung der Spiellogik und den Menüs beinhaltet das Diagramm noch zwei weitere wichtige Klassen, nämlich die Klasse *SaveLoadManager* und die Klasse *Assets*. Während Erstere zum Speichern und Laden von Spieldaten, die vor einer Änderung abgesichert sind, verwendet wird, dient Letztere zum Einladen sämtlicher Spielressourcen wie Bildern, Audio-Dateien oder Schriftarten. Außerdem werden in dieser Klasse Konfigurationsparameter eingelesen, über welche, verschiedenste Spielvariablen, die direkte Auswirkungen auf das Spielerlebnis haben, festgelegt werden können.

Darüber hinaus beinhaltet das Diagramm verschiedene Enums. Die drei Enums *GameState*, *PlayerState* und *EnemyState* bilden den aktuellen Zustand des Spiels, Spielers oder der jeweiligen Gegner ab. Diese Zustände werden innerhalb der entsprechenden Klasse verwaltet und beeinflussen das Spielgeschehen entscheidend. Diese klaren Hierarchien und Verbindungen ermöglichen eine bessere Strukturierung und Wartbarkeit des Codes während der Entwicklung.

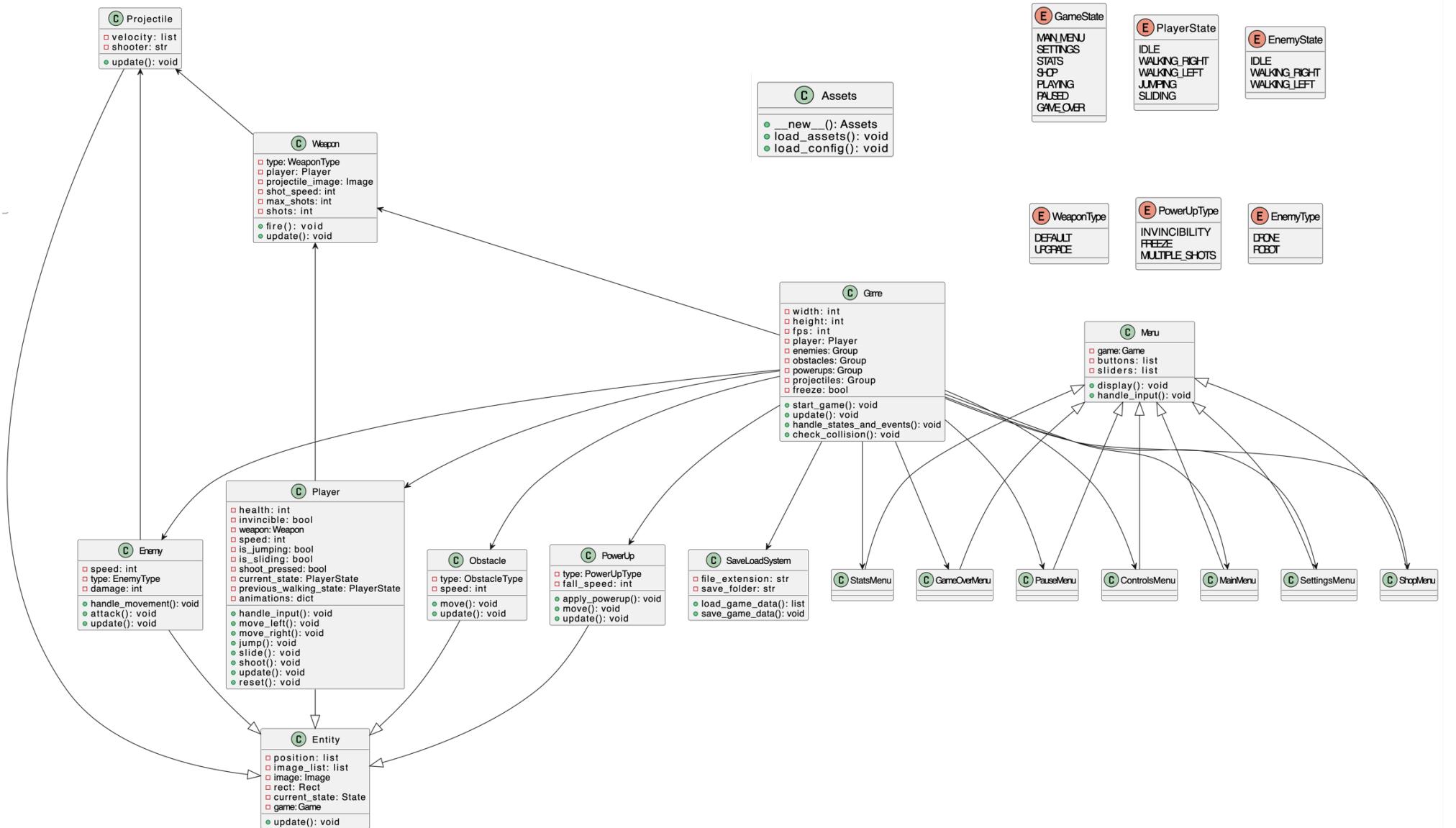


Abbildung 1: UML-Klassendiagramm

## 3.2 Aktivitätsdiagramm

Um den zentralen Ablauf der Hauptspiel-Schleife in der Game-Klasse zu visualisieren, wird ein UML-Aktivitätsdiagramm eingesetzt, welches in Abbildung 2 dargestellt ist. Die Hauptspiel-Schleife stellt das Kernstück des Spiels dar und orchestriert die wesentlichen Prozesse, die für das reibungslose Funktionieren des Spiels erforderlich sind. Das Aktivitätsdiagramm dient dazu, den iterativen Charakter der Hauptspiel-Schleife und seine wesentlichen Elemente zu verdeutlichen. Es beginnt mit der Initialisierung des Spiels, einschließlich der Konfiguration von Spielobjekten, Zuständen und Variablen. Anschließend erfolgt eine dauerhafte Schleife, die so lange läuft, wie der Spieler sich in der Anwendung bzw. im Spiel befindet und dieses nicht aktiv verlassen hat. Innerhalb dieser Schleife werden Zustände und Ereignisse behandelt, um den aktuellen Spielzustand zu überwachen.

Wenn sich das Spiel im Zustand „PLAYING“ befindet, werden die Spielobjekte aktualisiert und gerendert, um eine nahtlose Spielerfahrung zu gewährleisten. Falls das Spiel in den Zustand „GAME\_OVER“ wechselt, wird der Game Over-Bildschirm angezeigt. An dieser Stelle hat der Benutzer die Möglichkeit, das Spiel zu beenden oder es neu zu starten. Für den Fall, dass der Spieler das Spiel verlassen möchte, wird das Spiel geschlossen und die Schleife wird nicht erneut ausgeführt. Bei einem Neustart werden hingegen alle erforderlichen Schritte durchgeführt, um das Spiel in einen spielbaren Zustand zurückzuversetzen. Das Diagramm endet mit einer Schleifenrückkehr, die sicherstellt, dass der Ablauf in der Hauptspielschleife fortgesetzt wird, solange das Spiel läuft.

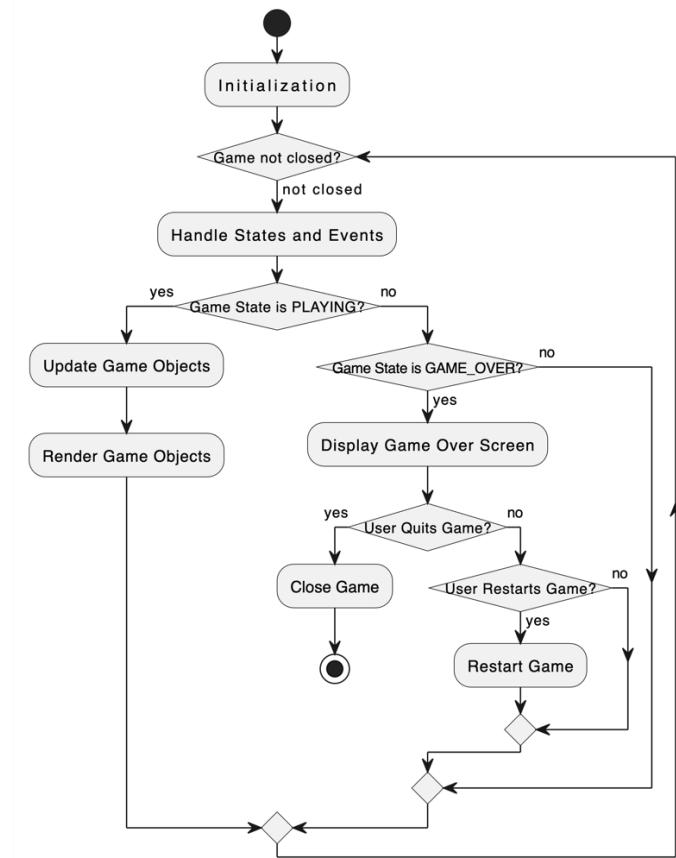


Abbildung 2: UML-Aktivitätsdiagramm

### 3.3 Sequenzdiagramm

Das in Abbildung 3 auf der nachfolgenden Seite illustrierte UML-Sequenzdiagramm zeigt die Interaktion zwischen dem Benutzer, der durch den *User*-Akteur repräsentiert wird, der *Player*-Klasse und der *Game*-Klasse. Das Diagramm dient dazu, den Fluss der Benutzerinteraktionen mit der Spielfigur im Spiel zu visualisieren. Aus Übersichtlichkeitsgründen ist das Sequenzdiagramm bewusst vereinfacht dargestellt und sämtliche Bedingungen, die erfüllt sein müssen, damit eine Tastatureingabe des Benutzers auch eine entsprechende Aktion der Figur nach sich zieht, wurden vernachlässigt. Diese können dem Quellcode sowie den dazugehörigen erklärenden Kommentaren in ebendiesem entnommen werden.

Das Sequenzdiagramm beginnt damit, dass der Benutzer das Spiel startet, was durch den Aufruf der Methode *start\_game()* geschieht. Daraufhin aktiviert sich die Klasse *Game* und führt die Methode *handle\_states\_and\_events()* für jedes auftretende bzw. identifizierte Event aus. In dieser Methode werden allgemeine Spielzustände und Ereignisse behandelt. Im Anschluss daran wird überprüft, ob der aktuelle Zustand das Spielen ist. Bei positivem Ergebnis erfolgt die Ausführung der Funktion *update()*, welche die Spielobjekte aktualisiert. Zudem wird die Klasse *Player* aktiviert und über die klasseneigene Methode *update()* die Funktion *handle\_input()* aufgerufen, die auf Benutzereingaben lauscht.

Das Diagramm zeigt dann die Verarbeitung von Benutzereingaben für verschiedene Tasten, darunter die vier Pfeiltasten (LEFT\_ARROW, RIGHT\_ARROW, UP\_ARROW, DOWN\_ARROW) und die Leertaste (SPACE). Jede dieser Eingaben führt zu entsprechenden Aktionen in der Klasse *Player*, also zu einer Bewegung des Charakters nach links oder rechts, zu einem Sprung, zum Sliden respektive Rutschen oder zum Abfeuern eines Schusses. Das Diagramm illustriert die User-Interaktionen dementsprechend beispielhaft für ein konkretes Szenario, in dem der Spieler die vorgenannten Aktionen in exakt dieser Reihenfolge ausführen muss, um zu überleben. Denkbar wäre ein solches Szenario im Spielkontext folgendermaßen: Der Spieler bewegt sich zunächst nach links, sieht einen Schuss der Drohne von oben kommen und bewegt sich dann nach rechts, um diesem Schuss auszuweichen. Anschließend muss er erst über einen Meteoriten springen und dann滑en, um nicht ein entgegenkommendes Auto zu berühren. Zu guter Letzt, wird ein aufgetauchter Roboter abgeschossen, damit dieser keine weiteren Probleme verursacht.

Nach jeder Benutzerinteraktion wird die Methode *update\_animation()* aufgerufen, um Animationen zu aktualisieren. Animationen werden dadurch realisiert, dass verschiedene Bilder der Spielfigur in schneller Geschwindigkeit hintereinander angezeigt werden. Neben den reinen Spielaktionen werden also auch visuelle Aspekte berücksichtigt, um die verschiedenen Interaktionsmöglichkeiten mit dem Charakter besser voneinander unterscheiden zu können und dem Benutzer allgemein eine bessere Spielerfahrung zu bieten.

Nach diesem Punkt würde sich der Zyklus, anders als im Diagramm einfacheitshalber dargestellt, ab der Methode *handle\_states\_and\_events()* wiederholen. Diese Schleife setzt sich fort, solange

das Spiel läuft. Durch die zyklische Natur wird der fortlaufende Prozess von Benutzerinteraktionen und Spielaktualisierungen während des Spiels repräsentiert.



Abbildung 3: UML-Sequenzdiagramm

## 4 Design- und Implementierungsentscheidungen

### 4.1 Singleton Entwurfsmuster

Bei der Implementierung des Endless-Runner-Spiels wird auf das Singleton-Entwurfsmuster zurückgegriffen, um eine effiziente und konsistente Verwaltung sämtlicher Spielressourcen zu gewährleisten. Das Singleton-Muster gehört zur Kategorie der Erzeugungsmuster und stellt sicher, dass lediglich eine Instanz einer Klasse existiert, was wiederum globalen Zugriff auf diese zentralen Ressourcen ermöglicht. Im vorliegenden Projekt ist die Klasse Assets als Singleton implementiert.

Der Mechanismus des Singleton-Musters wird durch die geschickte Anwendung der `__new__`-Methode realisiert. Diese Methode sorgt dafür, dass zu jedem Zeitpunkt nur eine einzige Instanz der Klasse existiert, was die Konsistenz und Koordination der Spielressourcen optimiert. Diese Ressourcen umfassen nicht nur Bilder für Spieler, Hindernisse, Gegner, Menüs und Power-Ups, sondern schließen auch Audiodateien für Musik und Soundeffekte mit ein. Darüber hinaus werden Konfigurationsparameter, die in der „`config.json`“-Datei hinterlegt sind, sorgfältig eingelesen.

Ein zentraler Fokus liegt auf der übersichtlichen Organisation der Ressourcen, wobei die Assets-Klasse verschiedene Klassenattribute nutzt, um nicht nur auf die Singleton-Instanz, sondern auch auf die Konfigurationsparameter zugreifen zu können. Diese Strukturierung erleichtert nicht nur den Zugriff, sondern ermöglicht auch eine klare Zuordnung und Verwendung der geladenen Ressourcen. Ein anderer nicht unwesentlicher Aspekt dieser Asset-Verwaltung liegt in der Einbindung von Schriftarten (Fonts), die eine visuell ansprechende Gestaltung des Spiels ermöglichen. Durch die Verwendung des Singleton-Entwurfsmusters wird nicht nur die Skalierbarkeit des Spiels verbessert, sondern auch die Pflege und Erweiterung der Ressourcen vereinfacht. Das Singleton-Muster eröffnet somit die Möglichkeit eines globalen, konsistenten Zugriffs auf alle Spielressourcen von verschiedenen Teilen bzw. Komponenten des Spiels aus. Insgesamt bietet die Wahl des Singleton-Entwurfsmusters eine robuste Lösung für die zentrale Verwaltung der für das Spiel benötigten Ressourcen, fördert die Wartbarkeit des Codes und optimiert die Performance des Pygame-Spiels.

## 4.2 Sonstige

Neben der Verwendung des Singleton-Entwurfsmusters gibt es eine Reihe weiterer bedeutsamer Design- und Implementierungsentscheidungen, die auf dem Weg zum fertigen Endless-Runner-Spiel getroffen wurden. Die Entscheidung für eine objektorientierte Design-Struktur ermöglichte eine klare und modulare Organisation des Codes. Durch die Nutzung von Klassen und Vererbung konnte eine hierarchische Struktur geschaffen werden, die die verschiedenen Spielelemente, wie Spieler, Gegner, Hindernisse und Power-Ups, effizient repräsentiert. Die Vorteile der Objektorientierung, wie Wiederverwendbarkeit, Kapselung und leichtere Erweiterbarkeit, waren entscheidend für die Bewältigung der Komplexität des Endless-Runner-Spiels.

Die Klasse Entity spielt eine zentrale Rolle im Design, da sie die gemeinsamen Eigenschaften und Verhaltensweisen aller Spielelemente definiert. Die Implementierung der *update()*-Funktion sowie von Attributen wie der Position und des aktuellen Status oder angezeigten Bilds in dieser Basisklasse ermöglicht unter anderem eine einheitliche Aktualisierung der Positionen der Spielobjekte und reduziert somit redundanten Code. Die Verwendung von Enums zur Repräsentation von Spielzuständen, Spielerzuständen, Gegnertypen und anderen kategorialen Daten trägt erheblich zur Lesbarkeit des Codes bei. Diese Entscheidung hilft nicht nur, den Code verständlicher zu machen, sondern minimiert auch potenzielle Fehlerquellen durch die Vermeidung von magischen Werten.

Ein weiterer entscheidender Punkt ist die Anpassbarkeit des Spiels. Durch die Integration einer *config.json*-Datei können sämtliche Spielparameter, von der Geschwindigkeit der Spielerbewegung bis zu den Zeitintervallen, in denen Gegner, Hindernisse oder Power Ups spawnen, einfach und schnell angepasst werden. Diese externe Konfigurationsdatei bietet nicht nur Flexibilität während der Entwicklung, sondern ermöglicht es auch, das Spiel nachträglich, ohne Änderungen am Quellcode vornehmen zu müssen, zu adjustieren. Dieser Punkt war unter anderem zur schnellen, nahtlosen Einarbeitung von Feedback von Nutzern entscheidend und trägt maßgeblich dazu bei, das Gleichgewicht des Spiels zu wahren und die Spielerfahrung zu optimieren, ohne in den Code eingreifen zu müssen.

Die Implementierung des Spielzyklus und des Aktualisierungsmechanismus spielt eine bedeutende Rolle für die reibungslose Funktionsweise des Endless-Runners. Die *update()*-Methode wird kontinuierlich aufgerufen und ermöglicht die Aktualisierung aller Spielobjekte, was wiederum die Animationen, Bewegungen und Interaktionen im Spiel sicherstellt. Durch die kluge Nutzung dieses Mechanismus konnte eine effiziente und ressourcenschonende Spiellogik entwickelt werden.

Die Integration eines Systems zur Speicherung und zum Laden von Spielständen (*SaveLoadSystem*) eröffnet dem Spieler die Möglichkeit, das Spiel zu beenden respektive zu schließen und später mit gleichem Spielstand fortzufahren. Das bedeutet, dass Dinge wie die gesammelten Coins, erreichte Statistiken oder gewählte Lautstärkeinstellungen auch über die aktuelle Spielsession hinaus gespeichert werden. Die Wahl einer einheitlichen Dateiendung und eines festen Ordnerpfads für die

gespeicherten Daten gewährleistet eine konsistente und benutzerfreundliche Implementierung dieses Features.

Die *Menu*-Klasse, die verschiedene Menüs innerhalb des Spiels repräsentiert, wurde so entworfen, dass sie nicht nur die Anzeige, sondern auch die Handhabung von Benutzerinteraktionen ermöglicht. Hierzu zählt neben dem Drücken von Schaltflächen auch die Verwendung von sogenannten Slidern, mit denen sich im Einstellungsmenü die Lautstärkeoptionen festlegen lassen. Diese Entscheidung erleichtert nicht nur die Erweiterung um neue Menüs, sondern sorgt auch für eine klare Trennung zwischen Spiellogik und Benutzeroberfläche.

Darüber hinaus wurde die Logik der Kollisionserkennung zwischen den verschiedenen Spielobjekten von einer anfänglichen rechteckigen Kollisionserkennung zu einer pixelgenauen Kollisionserkennung optimiert, was zu einer erheblichen Verbesserung des Spielflusses sowie der Nutzerzufriedenheit beiträgt. Statt der Überprüfung, ob die rechteckigen Bounding-Boxes, welche die Spielentitäten umgeben, überlappen, wird jeder überlappende Pixel dahingehend untersucht, ob er zum jeweiligen Spielobjekt gehört, oder aber transparent ist. Folglich ist das Resultat des Kollisionschecks nur dann positiv, wenn sich zwei Spielentitäten auch wirklich berühren und nicht nur deren Rechtecke. Es wird also sichergestellt, dass es in den überlappenden Bereichen beider Objekte auch wirklich mindestens einen intransparenten Pixel gibt.

Zusammenfassend lassen sich diese Design- und Implementierungsentscheidungen als zentrale Säulen des Endless-Runner-Spiels betrachten. Von der klaren Hierarchie der Klassen über die effektive Nutzung von Enums bis hin zur externen Konfigurationsdatei tragen diese Entscheidungen maßgeblich zur Lesbarkeit, Wartbarkeit und Anpassbarkeit des Codes bei. Sie bilden das Gerüst, auf dem das Spiel aufbaut, und ermöglichen eine flexible und zugleich effiziente Entwicklung.

## 5 Benutzeranleitung

Zur Ausführung der Anwendung respektive des Endless-Runner-Spiels müssen die folgenden Voraussetzungen erfüllt und die aufgeführten Schritte durchgeführt werden.

Voraussetzungen:

- Python ist in der Version 3.11 oder neuer auf dem System installiert.
- Git ist auf dem Rechner installiert.

Ausführung der Anwendung:

- Klonen des GitHub-Repository ([https://github.com/jan-wun/PSE\\_Endless-Runner-Game](https://github.com/jan-wun/PSE_Endless-Runner-Game)).
- Navigation in das geklonte Verzeichnis.
- Starten der Anwendung mithilfe der Batch-Datei „start.bat“ ([https://github.com/jan-wun/PSE\\_Endless-Runner-Game/blob/main/start.bat](https://github.com/jan-wun/PSE_Endless-Runner-Game/blob/main/start.bat)).



INTERNATIONALE  
HOCHSCHULE

## **Testprotokoll – Projekt Software Engineering**

**Erstellt von:** Jan Wunderlich  
**Matrikelnummer:** 92205002  
**Studiengang:** Master of Science Informatik  
**Tutor:** Prof. Dr. Markus Kleffmann  
**Datum:** 25.12.2023

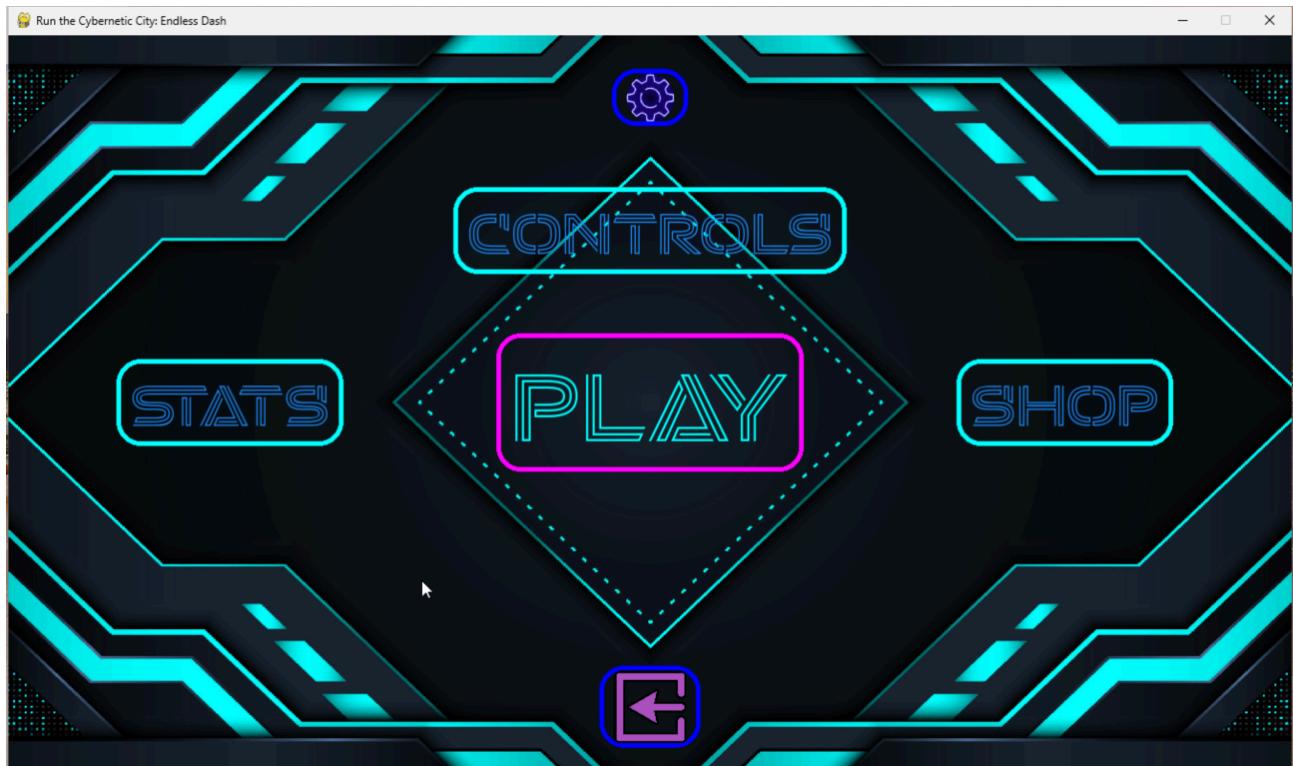
## **Inhaltsverzeichnis**

1	Dokumentation der Testfälle.....	3
1.1	Testfall 1: Anwendung starten und Hauptmenü anzeigen (User Story 1 & 6) .....	3
1.2	Testfall 2: Menüfunktionalität (User Story 6).....	4
1.3	Testfall 3: Spieler steuern (User Story 2).....	7
1.4	Testfall 4: Gegner mit verschiedenen Attacken (User Story 3).....	8
1.5	Testfall 5: Funktionale Spiellogik (User Story 4).....	9
1.6	Testfall 6: Fortschritt anzeigen (User Story 5) .....	10
1.7	Testfall 7: Power-Ups einsammeln (User Story 7).....	11
1.8	Testfall 8: Belohnungen für Leistungen (User Story 8).....	13
1.9	Testfall 9: Waffe abfeuern (User Story 9) .....	15

## 1 Dokumentation der Testfälle

### 1.1 Testfall 1: Anwendung starten und Hauptmenü anzeigen (User Story 1 & 6)

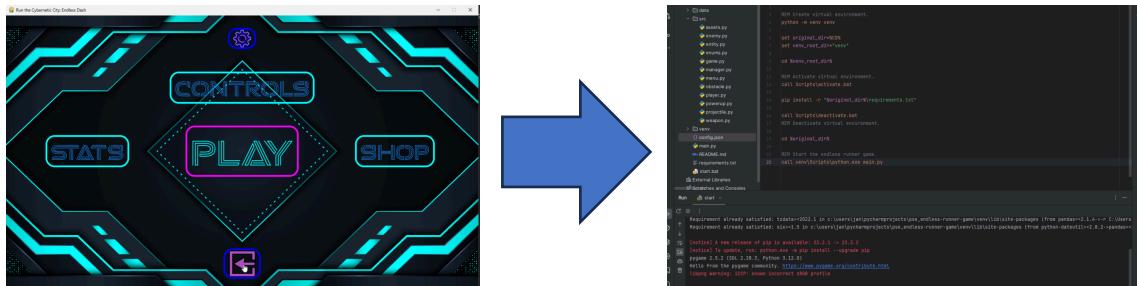
- Kurze Beschreibung
  - Überprüfen, ob sich beim Anwendungsstart ein Fenster mit festgelegter Größe öffnet und das Hauptmenü angezeigt wird.
- Vorbedingungen
  - keine
- Durchführung des Tests
  - Anwendung starten
- Erwartetes Ergebnis
  - Ein neues Fenster öffnet sich, besitzt eine festgelegte Größe und zeigt das Hauptmenü an.
- Tatsächliches Ergebnis



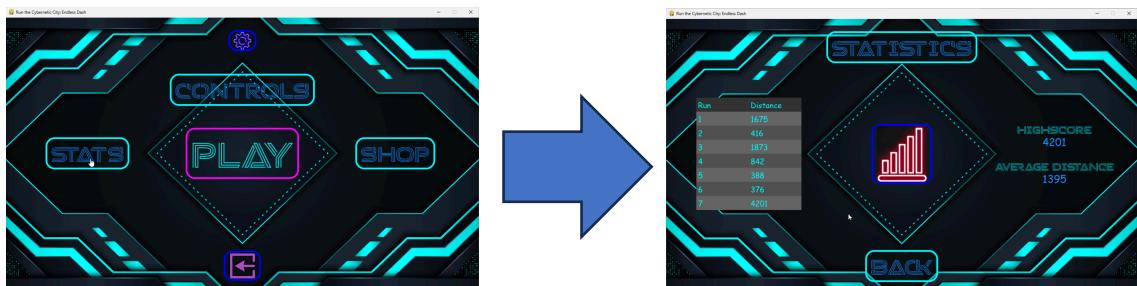
## 1.2 Testfall 2: Menüfunktionalität (User Story 6)

- Kurze Beschreibung
  - Überprüfen, ob beim Klicken auf die jeweiligen Buttons in den verschiedenen Menüs (Hauptmenü, Shop, Einstellungen, Statistiken, Controls, Pause, Game Over) die richtigen Seiten angezeigt werden bzw. die gewünschte Aktion passiert.
- Vorbedingungen
  - Je nachdem, welches Menü getestet wird, vom Hauptmenü in entsprechendes Menü per Buttonklick wechseln.
  - Für Pause und Game-Over-Menü muss das Spiel gestartet worden sein und der Pause Button oder p gedrückt werden bzw. der Spieler sterben.
  - Das Spiel befindet sich im entsprechenden Zustand.
- Durchführung des Tests
  - Anwendung starten
  - Buttons in Hauptmenü testen
  - Zurück-Button in Untermenüs testen
  - Einstellungen in Settings-Menü verändern
  - Spiel starten, Pause-Button drücken und Buttons in Pause-Menü ausprobieren
  - Im Spiel sterben und Buttons im Game-Over-Menü testen
- Erwartetes Ergebnis
  - Hauptmenü: Quit-Button (unten) schließt Anwendung, Stats-Button (links) öffnet Statistiken, Play-Button (Mitte) startet Spiel bzw. Versuch, Settings-Button (oben) öffnet Einstellungen, Controls-Button (zwischen Play-Button und Settings-Button) öffnet Controls-Menü, Shop-Button (rechts) öffnet Shop-Menü
  - Untermenüs: Back-Button führt zurück zum Hauptmenü
  - Einstellungen: Veränderung der Laustärken kann wahrgenommen werden; Musik-Lautstärkenänderung wird direkt wahrgenommen; Sound-Lautstärke erst bei Starten eines Versuchs und Springen / Schießen
  - Pause-Menü: Resume-Button sorgt für Fortsetzung des Spiels, Main-Menu-Button führt zurück zum Hauptmenü; Quit-Button schließt Anwendung
  - Game-Over-Menü: Restart-Button sorgt für Neustart des Spiels bzw. eines Versuchs; Main-Menu-Button führt zurück zum Hauptmenü; Quit-Button schließt Anwendung
- Tatsächliches Ergebnis
  - Hauptmenü:

▪ Quit-Button



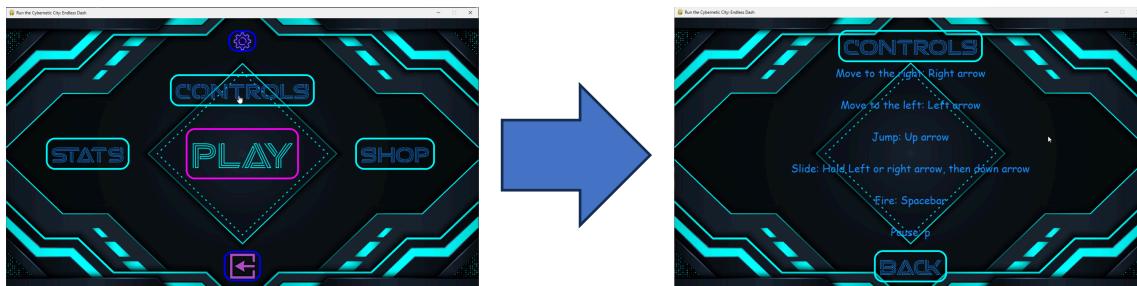
▪ Stats-Button



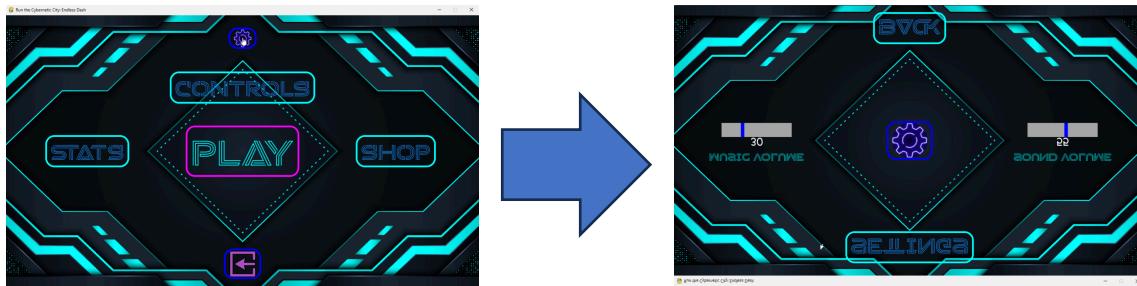
▪ Play-Button



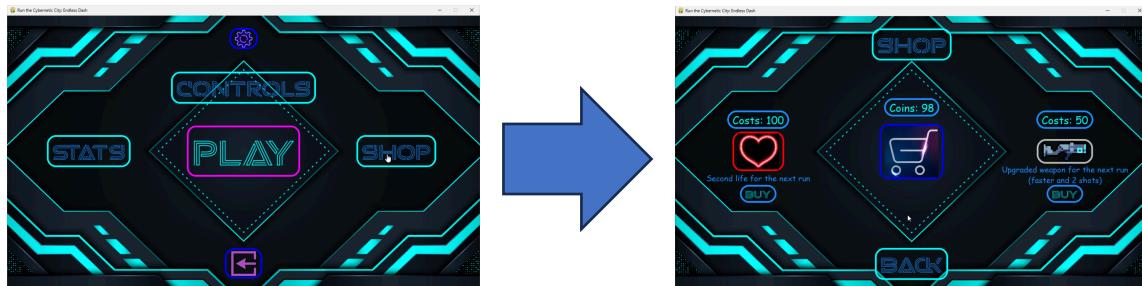
▪ Controls-Button



▪ Settings-Button



- Shop-Button



- Untermenüs (Shop, Settings, Stats und Controls-Menü):
  - Back-Button → Rückkehr zum Hauptmenü
- Einstellungen:
  - Laustärkenanpassung von Musik und Sounds wird korrekt übernommen
- Pause-Menü:
  - Resume-Button → Fortsetzung des Spiels
  - Main-Menu-Button → Rückkehr zum Hauptmenü
  - Quit-Button → Anwendung schließt sich
- Game-Over-Menü:
  - Restart-Button → Neuer Versuch
  - Main-Menu-Button → Rückkehr zum Hauptmenü
  - Quit-Button → Anwendung schließt sich
- Shop-Menü:
  - Wird in Testfall 8 getestet

### 1.3 Testfall 3: Spieler steuern (User Story 2)

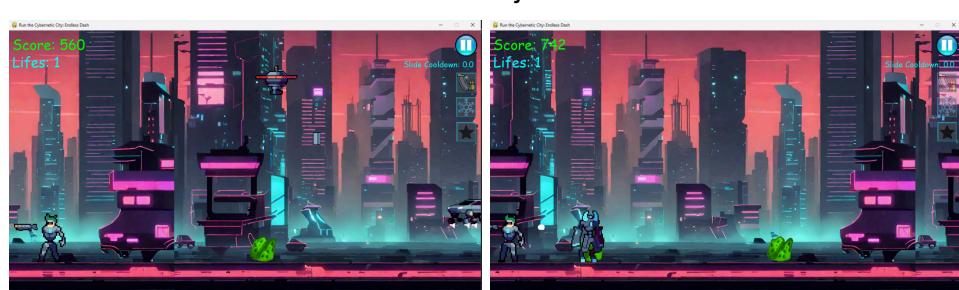
- Kurze Beschreibung
  - Überprüfen, ob Spielercharakter auf Tastatureingaben korrekt reagiert.
  - Überprüfen, ob Bewegungen des Spielers durch Bildschirmgrenzen limitiert werden.
- Vorbedingungen
  - Das Spiel befindet sich im Zustand „PLAYING“
- Durchführung des Tests
  - Anwendung starten & Play-Button in Hauptmenü drücken
  - Spielerbewegungen nach rechts und links ausprobieren (rechte / linke Pfeiltaste)
  - Springen ausprobieren (Pfeiltaste nach oben + zusätzlich rechts / links)
  - Sliden ausprobieren (rechte / linke Pfeiltaste gedrückt halten + Pfeiltaste nach unten drücken)
  - Waffe abfeuern (Leertaste)
  - Versuchen links und rechts über den Bildschirm hinaus zu gehen
- Erwartetes Ergebnis
  - Pfeiltaste links = Spieler bewegt sich nach links
  - Pfeiltaste rechts = Spieler bewegt sich nach rechts
  - Pfeiltaste oben = Spieler springt aus dem Stand nach oben
  - Pfeiltaste oben + rechts = Spieler springt nach oben und rechts
  - Pfeiltaste oben + links = Spieler springt nach oben und links
  - Pfeiltaste unten = nichts passiert
  - Pfeiltaste links gedrückt + Pfeiltaste unten = Spieler slidet nach links
  - Pfeiltaste rechts gedrückt + Pfeiltaste unten = Spieler slidet nach rechts
  - Leertaste = Waffe wird abgefeuert / Schuss löst sich
  - Pfeiltaste links an linkem Ende des Bildschirms = Spieler bewegt sich nicht weiter
  - Pfeiltaste rechts an rechtem Ende des Bildschirms = Spieler bewegt sich nicht weiter
- Tatsächliches Ergebnis
  - Pfeiltaste links = Spieler bewegt sich nach links
  - Pfeiltaste rechts = Spieler bewegt sich nach rechts
  - Pfeiltaste oben = Spieler springt aus dem Stand nach oben
  - Pfeiltaste oben + rechts = Spieler springt nach oben und rechts
  - Pfeiltaste oben + links = Spieler springt nach oben und links
  - Pfeiltaste unten = nichts passiert
  - Pfeiltaste links gedrückt + Pfeiltaste unten = Spieler slidet nach links
  - Pfeiltaste rechts gedrückt + Pfeiltaste unten = Spieler slidet nach rechts
  - Leertaste = Waffe wird abgefeuert / Schuss löst sich
  - Pfeiltaste links an linkem Ende des Bildschirms = Spieler bewegt sich nicht weiter
  - Pfeiltaste rechts an rechtem Ende des Bildschirms = Spieler bewegt sich nicht weiter

## 1.4 Testfall 4: Gegner mit verschiedenen Attacken (User Story 3)

- Kurze Beschreibung
  - Überprüfen, ob Gegner sichtbar und visuell einfach zu unterscheiden sind.
  - Überprüfen, ob Gegner Attacken besitzen, die nicht identisch sind.
- Vorbedingungen
  - Das Spiel befindet sich im Zustand „PLAYING“
- Durchführung des Tests
  - Anwendung starten
  - Play-Button in Hauptmenü drücken
  - Warten bis Gegner spawnen + Hindernissen ausweichen / lange genug leben, um Schüsse von Gegnern zu sehen
- Erwartetes Ergebnis
  - Gegner erscheinen nach einiger Zeit auf dem Bildschirm und lassen sich visuell unterscheiden
  - Es gibt 2 Arten von Gegnern:
    - Drohne → bleibt nach Erscheinen für restliche Spielzeit bestehen; bewegt sich von Seite-zu-Seite und schießt alle paar Sekunden zufällig nach unten
    - Roboter → bewegt sich nach Erscheinen in Richtung des Spielers; schießt alle paar Sekunden zufällig in Richtung des Spielers; kann durch Abschießen getötet werden; erscheint nach Tod nach zufälliger Zeit wieder

### □ Tatsächliches Ergebnis

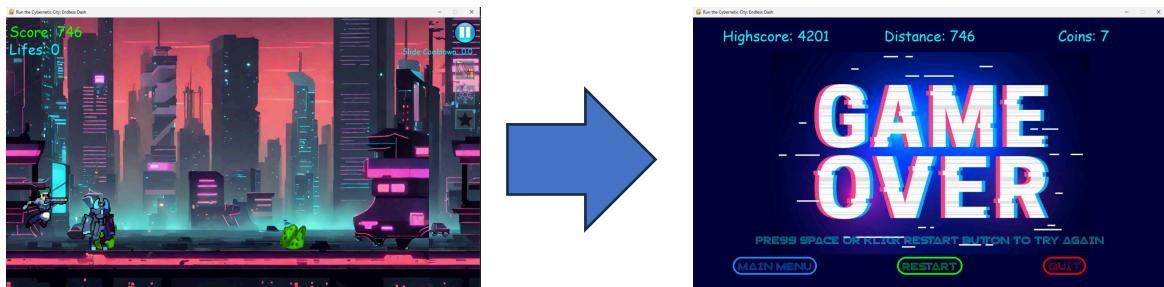
- Gegner erscheinen nach einiger Zeit auf dem Bildschirm und lassen sich visuell unterscheiden
- Bilder von Drohne und Roboter mit Projekttilen



- Drohne → bleibt nach Erscheinen für restliche Spielzeit bestehen; bewegt sich von Seite-zu-Seite und schießt alle paar Sekunden zufällig nach unten
- Roboter → bewegt sich nach Erscheinen in Richtung des Spielers; schießt alle paar Sekunden zufällig in Richtung des Spielers; kann durch Abschießen getötet werden; erscheint nach Tod nach zufälliger Zeit wieder

## 1.5 Testfall 5: Funktionale Spiellogik (User Story 4)

- Kurze Beschreibung
  - Überprüfen, ob Spiel bei Kollision mit Gegner oder Hindernis oder Projektil von Gegner korrekt beendet wird.
  - Überprüfen, ob Neustart das Spiel bzw. den Versuch neustartet.
- Vorbedingungen
  - Das Spiel befindet sich im Zustand „PLAYING“
  - Der Spieler besitzt genau ein Leben
- Durchführung des Tests
  - Anwendung starten
  - Play-Button in Hauptmenü drücken
  - Hindernis berühren (Auto und Meteor)
  - Gegner berühren (Roboter)
  - Von Projektil eines Gegners treffen lassen (Roboter und Drohne)
- Erwartetes Ergebnis
  - Wenn der Spieler ein Hindernis oder den Roboter berührt oder von einem Projektil eines Gegners (Roboter oder Drohne) getroffen wird und ein Leben besitzt, endet das Spiel bzw. der Game-Over Bildschirm wird angezeigt.
- Tatsächliches Ergebnis
  - Wenn der Spieler ein Hindernis oder den Roboter berührt oder von einem Projektil eines Gegners (Roboter oder Drohne) getroffen wird und ein Leben besitzt, endet das Spiel bzw. der Game-Over Bildschirm wird angezeigt.
  - Beispiel → Spieler wird von Projektil des Robotes getroffen



## 1.6 Testfall 6: Fortschritt anzeigen (User Story 5)

- Kurze Beschreibung
  - Überprüfen, ob Meterzähler sich basierend auf zurückgelegter Strecke erhöht.
  - Überprüfen, ob Meterzähler bei Spielneustart zurückgesetzt wird.
- Vorbedingungen
  - Das Spiel befindet sich im Zustand „PLAYING“
- Durchführung des Tests
  - Anwendung starten
  - Play-Button in Hauptmenü drücken
  - Sterben und Spiel neustarten
- Erwartetes Ergebnis
  - Zurückgelegte Distanz wird kontinuierlich erhöht (Anzeige oben links)
  - Bei Neustart wird Distanz auf 0 zurückgesetzt
- Tatsächliches Ergebnis
  - Zurückgelegte Distanz wird kontinuierlich erhöht (Anzeige oben links)
  - Bei Neustart wird Distanz zurückgesetzt



## 1.7 Testfall 7: Power-Ups einsammeln (User Story 7)

- Kurze Beschreibung
  - Überprüfen, ob Power-Ups zufällig auf dem Bildschirm erscheinen.
  - Überprüfen, ob Einsammeln eines Power-Ups jeweilige Funktion freischaltet / aktiviert.
- Vorbedingungen
  - Das Spiel befindet sich im Zustand „PLAYING“
- Durchführung des Tests
  - Anwendung starten
  - Play-Button in Hauptmenü drücken
  - Jeweiliges Power-Up einsammeln
- Erwartetes Ergebnis
  - Power-Ups erscheinen nach zufälligen Zeitintervallen auf der rechten Seite und fallen von oben nach unten
  - 3 Arten von Power-Ups
    - Multiple Shots → Spieler erhält für restlichen Versuch mehr Schüsse (er kann mehr Schüsse hintereinander abfeuern)
    - Freeze → Spiel wird für kurze Zeit eingefroren (alles außer der Spieler)
    - Invincibility → Spieler wird für kurze Zeit unbesiegbar (wenn er Hindernisse oder Gegner berührt, passiert nichts)
  - Einsammeln eines Power-Ups aktiviert jeweilige Funktion
  - Anzeige über aktivierte Power-Ups aktualisiert sich entsprechend (rechts oben); Zeit der restlichen Dauer des Power-Ups wird angezeigt
- Tatsächliches Ergebnis
  - Power-Ups erscheinen nach zufälligen Zeitintervallen auf der rechten Seite und fallen von oben nach unten
  - 3 Arten von Power-Ups
    - Multiple Shots



- Freeze

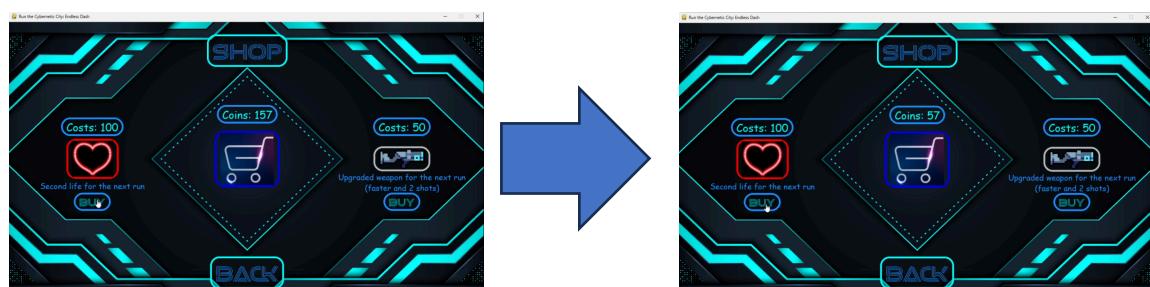


- Invincibility



## 1.8 Testfall 8: Belohnungen für Leistungen (User Story 8)

- Kurze Beschreibung
  - Überprüfen, ob Spieler Münzen / Coins für erreichte Distanz bei Spielende erhält.
  - Überprüfen, ob Spieler Münzen / Coins zum Kaufen von Items im Shop einsetzen kann.
- Vorbedingungen
  - Für 1: Spiel befindet sich im Zustand „GAME\_OVER“
  - Für 2: Spiel befindet sich im Zustand: „SHOP“ und Spieler hat genügend Coins zum Kaufen eines Items
- Durchführung des Tests
  - Anwendung starten
  - Für 1: Play-Button drücken & sterben
  - Für 2:
    - Shop-Button in Hauptmenü drücken
    - Buy-Button beider Items drücken
    - Neuen Versuch starten
- Erwartetes Ergebnis
  - Dem Spieler werden die erreichten Coins gutgeschrieben.
  - Dem Spieler werden die angezeigten Coins abgezogen (hat er zu wenig oder das Item bereits gekauft, erscheint eine entsprechende Warnmeldung).
  - Wird der nächste Versuch gestartet, besitzt der Spieler für diesen einen Versuch die gekauften Items (zweites Leben oder bessere Waffe → mehr und schnellere Schüsse).
- Tatsächliches Ergebnis
  - Dem Spieler werden die erreichten Coins gutgeschrieben.
  - Dem Spieler werden die angezeigten Coins abgezogen (hat er zu wenig oder das Item bereits gekauft, erscheint eine entsprechende Warnmeldung).
  - Wird der nächste Versuch gestartet, besitzt der Spieler für diesen einen Versuch die gekauften Items (zweites Leben oder bessere Waffe → mehr und schnellere Schüsse)





## 1.9 Testfall 9: Waffe abfeuern (User Story 9)

- Kurze Beschreibung
  - Überprüfen, ob Spieler Waffe per Tastendruck abfeuern kann (nur so oft wie er Schüsse hat und Schüsse noch nicht verschwunden sind).
  - Überprüfen, ob Gegner (Roboter) verschwindet, wenn er getroffen wird.
- Vorbedingungen
  - Das Spiel befindet sich im Zustand „PLAYING“
- Durchführung des Tests
  - Anwendung starten
  - Play-Button drücken
  - Leertaste zum Abfeuern eines Schusses drücken
  - Gegner (Roboter) abschießen
- Erwartetes Ergebnis
  - Leertaste feuert einen Schuss ab, wenn der Spieler aktuell noch Schüsse hat
  - Gegner (Roboter) verschwindet, wenn Spieler ihn mit einem Schuss trifft
- Tatsächliches Ergebnis
  - Leertaste feuert einen Schuss ab, wenn der Spieler aktuell noch Schüsse hat
  - Gegner (Roboter) verschwindet, wenn Spieler ihn mit einem Schuss trifft





INTERNATIONALE  
HOCHSCHULE

## Abstract – Projekt Software Engineering

**Erstellt von:** Jan Wunderlich  
**Matrikelnummer:** 92205002  
**Studiengang:** Master of Science Informatik  
**Tutor:** Prof. Dr. Markus Kleffmann  
**Datum:** 28.12.2023

## **1 Einführung und Projektziele**

Das Software Engineering Projekt startete mit dem klaren Ziel, ein fesselndes Endless-Runner-Game zu entwickeln, welches eine Reihe von verschiedenen Funktionalitäten besitzen und unterschiedlichste Anforderungen erfüllen sollte. So sollte der Spieler unter anderem dazu in der Lage sein, eine Spielfigur zu steuern, mit der er geschickt sämtlichen Hindernissen ausweichen und auftauchende Gegner besiegen kann. Außerdem sollte der Spieler verlieren, wenn er von einem Hindernis oder Gegner getroffen wird sowie seinen Highscore, also die höchste erreichte Distanz, sehen können, um eine Motivation zu schaffen, bei jedem neuen Versuch sein Bestes zu geben und einen neuen Highscore aufzustellen.

## **2 Durchführung des Projekts**

Die Umsetzung des Projekts erfolgte in vier fünftägigen Sprints, wobei zunächst eine eingehende Einarbeitung in Pygame notwendig war. Die strukturierte Umsetzung passierte über Tickets in JIRA, die dem jeweiligen Sprint im Rahmen einer Sprintplanung zugeordnet wurden. Hierdurch konnten die einzelnen Anforderungen in gut beherrschbare Aufgabenpakete unterteilt werden und die Implementierung fiel durch den schnell sichtbaren Fortschritt sowie häufige Erfolgserlebnisse leicht. Kam es doch mal dazu, dass innerhalb eines Sprints nicht alle Aufgaben bzw. Tickets abgearbeitet werden konnten, so stellte dies kein Problem dar und ebenjene Tickets wurden einfach mit in den nächsten Sprint genommen.

Dieser Methodik folgend, wurde im ersten Sprint zunächst ein Grundgerüst geschaffen und Hintergrund, Spieler sowie Hindernisse wurden ebenso wie eine erste Kollisionserkennung und ein Fortschrittszähler implementiert. Innerhalb des zweiten Sprints erfolgte dann die Einarbeitung von Gegnern einschließlich ihrer Bewegungen sowie Attacken und es wurde eine Waffe hinzugefügt, die der Spieler zum Töten eines Gegners abfeuern kann. Der dritte Sprint befasste sich dann mit der Erstellung von verschiedenen Menüs, der Realisierung unterschiedlicher Funktionen innerhalb der Menüs und dem Erhalt von Belohnungen für erzielte Leistungen nach Spielende. Im vierten und letzten Sprint wurde dann viel Zeit zur Verbesserung bzw. Optimierung des bis dahin geschriebenen Codes und zur Einarbeitung von Feedback anderer Spieler aufgewendet. Zudem wurden noch Power-Ups sowie ein Entwurfsmuster für die Verwaltung von Spielressourcen umgesetzt. Diese iterative Vorgehensweise ermöglichte eine schrittweise Entwicklung und Anpassung an aufkommende Anforderungen sowie Probleme.

## **3 Herausforderungen**

Die Herausforderungen lagen nicht nur in der Umsetzung des Spiels selbst, sondern auch in der Einarbeitung in das Pygame-Framework sowie in der Definition und Priorisierung von Anforderungen. Besonders anspruchsvoll war die Komplexität des Spiels. Jede Komponente ließ sich einzeln gut oder mit angemessenem Aufwand implementieren. Jedoch gestaltete sich das Zusammenführen der verschiedenen Elemente zu einem Gesamtkonstrukt mit allen notwendigen

Abhängigkeiten als anspruchsvoll. Speziell die Realisierung eines übersichtlichen, leicht wartbaren und vor allem erweiterbaren Codes erwies sich als wesentlich komplexer als ursprünglich angenommen.

## 4 Einhaltung des Zeitplans

Die Zeitplanung konnte insgesamt gesehen eingehalten und sogar unterboten werden. Dies ist hauptsächlich auf das Maß an unerwartet verfügbarer privater Zeit zurückzuführen, die fast täglich in die Entwicklung und Dokumentation des Spiels investiert werden konnte. Die effiziente Nutzung der verfügbaren Zeit ermöglichte nicht nur die rechtzeitige Fertigstellung, sondern auch eine schnelle Umsetzung der Dokumentation und Erfüllung der schriftlichen Anforderungen.

## 5 Kritische Reflexion und gewonnene Erkenntnisse

Rückblickend kann das Projekt zweifelsohne als Erfolg betrachtet werden. Alle definierten Anforderungen wurden innerhalb des festgelegten Zeitrahmens erfüllt, und das Endprodukt bzw. das Spiel, präsentiert sich nicht nur ästhetisch, sondern ist auch voll funktionsfähig. Dennoch ist die anfängliche Unterschätzung des Arbeitsaufwands und der Komplexität der entwickelten Anwendung kritisch zu reflektieren. Trotz erfolgreicher Überwindung von Schwierigkeiten, Problemen und Verzögerungen während der Spielimplementierung, insbesondere durch zusätzliche Zeitinvestitionen und großzügige Zeitreserven für die Dokumentation, zeigt sich bei genauer Betrachtung des Entwicklungsprozesses im UML-Klassendiagramm ein klarer Lernprozess. Die anfängliche Planung, bedingt durch mangelnde Erfahrung mit Pygame, erwies sich nur als bedingt sinnvoll und zutreffend. Die gewonnenen Erkenntnisse verdeutlichen die Bedeutung einer umfassenden und detaillierten Projektplanung. Insbesondere sollte vor dem Start der Implementierungsphase mehr Zeit für eine präzise Planung eingeplant werden. Alle Aspekte, die für die spätere Entwicklung von Relevanz sind, sollten in dieser Phase sorgfältig durchdacht und umfassend dokumentiert werden.

## 6 Fazit

Das Projekt war nicht nur ein technischer Erfolg, sondern auch eine persönliche Lernreise. Die Einarbeitung in Pygame und die Umsetzung der gestellten Anforderungen konnte erfolgreich gemeistert werden. Die Herausforderungen in der Code-Optimierung haben gezeigt, dass ein gut durchdachter Entwurf von entscheidender Bedeutung ist. Die Einhaltung des Zeitplans war ein entscheidender Faktor für den erfolgreichen Abschluss des Projekts.

Insgesamt hat der Abschluss des Moduls nicht nur ein funktionierendes Spiel mitsamt Dokumentation hervorgebracht, sondern auch wertvolle Erkenntnisse über Projektplanung, Problemlösung und Code-Optimierung generiert. Der Weg von der anfänglichen Planung bis zum endgültigen Produkt war eine lehrreiche Erfahrung, die in zukünftigen Projekten von großem Nutzen sein kann.