HOCHSCHULE
MITTWEIDA
University of Applied Sciences

BACHELORARBEIT

Frau
Elly Müller

Anforderungsanalyse einer Game Engine auf Basis eines Game Design Dokumentes (GDD)

Mittweida, April 2023

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Anforderungsanalyse einer Game Engine auf Basis eines Game Design Dokumentes (GDD)

Autorin:
Elly Müller

Studiengang:
Medieninformatik und interaktives Entertainment

Seminargruppe:
MI17w1-B

Erstprüfer:
Prof. Dr. rer. nat. Tobias Czauderna

Zweitprüfer:
Dr. rer. nat. Rico Beier-Grunwald

Einreichung:
Mittweida, 08.04.2023

Verteidigung/Bewertung:
Mittweida, 2023

Faculty of **Applied Computer Sciences & Biosciences**

BACHELOR THESIS

Requirement analysis of a game engine based on a game design document

Author:
Elly Müller

Course of Study:
Media informatics and interactive entertainment

Seminar Group:
MI17w1-B

First Examiner:
Prof. Dr. rer. nat. Tobias Czauderna

Second Examiner:
Dr. rer. nat. Rico Beier-Grunwald

Submission:
Mittweida, 08.04.2023

Defense/Evaluation:
Mittweida, 2023

Bibliografische Beschreibung:

Müller, Elly:
Anforderungsanalyse einer Game Engine auf Basis eines Game Design Dokumentes (GDD). – 2023. – 50 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Bachelorarbeit, 2023.

Referat:

Seit 2018 befindet sich das Projekt „BuggyTech Engine“ an der Hochschule Mittweida in studentischer Entwicklung. Auf dieser Game Engine soll das von der Autorin konzipierte Spiel „Neon Nova“ laufen, wofür sie ein Game Design Document (GDD) erstellte. Das GDD stellt als grundlegendes Dokument, welches sämtliche relevanten Aspekte eines zu entwickelnden Spiels beschreibt, ein Herzstück in der Videospielentwicklung dar. Es dient als zentrales Werkzeug für das Entwicklerteam, um gemeinschaftlich auf ein klares Ziel hinzuarbeiten. In der klassischen Softwareentwicklung werden Anforderungen und Spezifikationen an eine zu entwickelnde Software detailliert in Form eines Lastenheftes verschriftlicht, welches für die Entwickler ähnliche Funktionen hat, wie das GDD. Diese Konzepte werden, neben der Erarbeitung zusätzlicher Wissensgrundlagen, zunächst dargestellt und auf Inhalte, sowie Vor- und Nachteile untersucht. Im Hauptteil dieser Arbeit wird die Eignung eines GDD als Grundlage für die Anforderungsanalyse einer Game Engine am Beispiel von Neon Nova und der BuggyTech Engine untersucht. Der Prozess der Erstellung des Lastenheftes wird dargelegt und die Ergebnisse vorgestellt. Zudem wurden Datenmodelle und UI Layouts für die spezifische Anwendung in de BuggyTech Engine konzipiert. Abschließend wird ein Fazit über den Prozess und die Geeignetheit der aufeinander aufbauenden Arbeitsweise der beiden Dokumente gezogen, welche normalerweise nicht miteinander in Kontakt kommen. Das Ergebnis der Arbeit ist ein vollständiges, für die Weiterentwicklung der BuggyTech Engine nutzbares Lastenheft.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

3D . . . . . dreidimensional

ASUS . . . . . ASUSTek

AU . . . . . Audiosystem

ECS . . . . . Entity-Component-System

ERM . . . . . Entity-Relationship-Modell

EV . . . . . Eventsystem

GB . . . . . Gigabyte

GDD . . . . . Game Design Document

GL . . . . . Game Logik

GR . . . . . Grafik

HUD . . . . . Heads Up Display

IDE . . . . . Integrated Development Environment

IM . . . . . Input Management

KI . . . . . Künstliche Intelligenz

NPC . . . . . Non-player character

PE . . . . . Profiling/Entwicklertools

PK . . . . . Physik und Kollision

RAM . . . . . Random-Access Memory

RM . . . . . Ressourcenmanagement

TA . . . . . Technische Anforderungen

VUCA . . . . . volatile, uncertain, complex, ambiguous

1 Einleitung

1.1 Projekt BuggyTech Engine

Im Rahmen des Moduls „Wissenschaft und Wirtschaft 3“ im Studiengang Medieninformatik und interaktives Entertainment wurde im Jahr 2019 die Idee geboren, einen Arcade Automaten von Grund auf zu bauen, einschließlich des Baus eines Gehäuses und der Entwicklung eines Spiels. Aufgrund der räumlichen Einschränkungen eines Arcade-Automaten wurde ein platzsparender Einplatinencomputer verwendet. Diese Hardwareeinschränkung verhinderte jedoch die Nutzung bestehender Entwicklungsumgebungen. Deshalb bestand die Notwendigkeit der Entwicklung einer eigens für den Arcade-Automaten und sein Spiel konzipierten Engine, die „BuggyTech Engine“. Dabei handelt es sich um eine Game Engine die als Software-Plattform speziell für die Entwicklung und Erstellung von Videospielen entwickelt wird [1]. Sie bietet eine Vielzahl von Tools und Funktionen, die Entwicklern helfen, Spiele schnell und effizient zu erstellen, ohne dass sie jedes Mal von Grund auf neu codieren müssen. Im Rahmen ihrer Bachelorarbeit beschäftigte sich Christina Klaus mit dem Entwurf der Enginearchitektur, sowie dem Entwurf und der Implementation der beiden Klassen „Game Loop“ und „Game Time“. Diese Konzepte legten den Grundstein für die Hauptschleife des Spiels, da sie für die Aktualisierung der Spielwelt und die Verwaltung zeitabhängiger Funktionen erforderlich sind. Theo Stötzer schrieb 2021 seine Bachelorarbeit über die Fortsetzung der Arbeit an der BuggyTech Engine. Er implementierte Audio- und Eventsubsysteme sowie einen Input-Manager für Testzwecke. Daniel Stockmann lieferte durch seine Zuarbeit die Grundlagen für das Grafiksystem der Engine. Es liegen also Vorarbeiten dreier verschiedener Autoren vor, welche aufeinander aufbauen und jeweils Teile der Engine implementierten. Dadurch ergibt sich eine unorganisierte Projektumgebung, welche bisher nicht strukturiert wurde. Dies steht einem reibungslosen Entwicklungsprozess im Weg. Zur Verbesserung des Ablaufes der Softwareentwicklung ist es von Vorteil, ein strukturiertes Vorgehen mit einem Fahrplan und einem zentralen Dokument zu etablieren, wie es bei üblichen Softwareentwicklungsprojekten der Fall ist.

1.2 Zielstellung und Aufgabenbeschreibung der Arbeit

Zielstellung der Arbeit war die Ableitung eines Lastenheftes aus einem Game Design Dokument (GDD). Während der Erarbeitung stellte sich heraus, dass das Lastenheft zusätzlich durch ein Datenmodell und den Entwurf eines Interfacelayouts unterstützt werden sollte.

Ein GDD dient als Ausgangspunkt für die Konzeption und Entwicklung eines Videospiels und fungiert als zentrales Dokument zur Erfassung aller game-designbezogenen und inhaltlichen Aspekte des Spiels.

Ein Lastenheft stellt in der klassischen Softwareentwicklung das zentrale Anforderungsdokument dar, das alle Anforderungen an eine zu entwickelnde Software in ihrer Gesamtheit erfasst und dokumentiert.

Lastenhefte sind übliche Anforderungsdokumente in der klassischen Softwareentwicklung, während GDDs typischerweise für die Spieleentwicklung verwendet werden. Obwohl sie unterschiedliche Zwecke haben, weisen sie Ähnlichkeiten in Bezug auf ihre Funktion auf, da sie strukturiert Anforderungen und Ziele dokumentieren, die für das zukünftige Entwicklerteam von Nutzen sind. Beide sind von entscheidender Bedeutung, um eine klare Richtung vorzugeben und sicherzustellen, dass alle Beteiligten innerhalb des Projekts am selben Strang ziehen.

In der Regel wird ein Game Design Document erst erstellt, nachdem die Wahl der Game Engine bereits getroffen wurde. Aus diesem Grund fallen üblicherweise keine technischen Anforderungen in den Aufgabenbereich des GDDs, da die Entwicklung einer Game Engine speziell für ein Spiel eine unübliche Praxis darstellt. Sollte jedoch eine Engine speziell für das Spiel entwickelt werden, müssen die Anforderungen des Spiels in Bezug auf seine Funktionalitäten und Komplexität im Vorfeld betrachtet werden. So unterscheidet sich die Entwicklung einer Engine für ein komplexes dreidimensional (3D)-Simulations-Spiel grundlegend von der Entwicklung einer Engine für ein Textadventure. Es ist daher von entscheidender Bedeutung, dass die gewünschten Funktionalitäten des Spiels betrachtet und auf die Anforderungen für die Engine übertragen werden. Hierfür bietet sich eventuell ein GDD an, da es diese Funktionalitäten detailliert beschreibt.

Obwohl die Herangehensweise unkonventionell ist und die Vermischung zweier normalerweise nicht zusammen genutzter Dokumente erfolgt, wird in dieser Arbeit die Verwendbarkeit eines selbst erstellten GDDs für die Erstellung einer strukturierten Anforderungsdokumentation in Form eines Lastenheftes untersucht. Das Hauptziel dieser Bachelor-Arbeit besteht darin, die Anforderungsanalyse der BuggyTech Engine zu erstellen und die Entwicklung durch die Erstellung von Datenmodellen und Datenstrukturen zu unterstützen. Die zentrale Fragestellung und Aufgabe der Arbeit besteht darin zu untersuchen, ob ein GDD als Leitfaden für die Erstellung einer Anforderungserfassung in Form eines Lastenheftes für die BuggyTech Engine dienen kann und somit Aufschlüsse über die Verwendbarkeit beider Dokumente in einer synergetischen Erweiterung ihrer üblichen Einsatzgebiete liefert.

1.3 Aufbau der Arbeit

Die Arbeit leitet das Thema mit einer Vor- und Gegenüberstellung klassischer und agiler Softwareentwicklungsmethoden ein. Es werden Begriffserklärungen und Überblicke über die relevantesten Punkte der Arbeit gegeben. In Kapitel 4 erklärt die Autorin ihr Vorgehen vor und bei der Erstellung des Lastenheftes und erläutert, wie die einzelnen Dokumente, wie z.B. Entity-Relationship-Modell (ERM) und Assethierarchie entstanden sind und weshalb bestimmte Programmiermuster Anwendung finden. Das Ergebnis dessen, das Lastenheft, welches lediglich die identifizierten Programmanforderungen enthält, befindet sich, ebenso wie das GDD, im Anhang der Arbeit. Der Schluss befasst sich mit einem Rückblick auf die erstellte Arbeit, evaluiert den Erfolg der Aufgabenstellung und die Anwendbarkeit eines GDDs als Basis für die Anforderungsanalyse für eine Game Engine.

2 Grundlagen

In der Regel beginnt jede Softwareentwicklung mit einem Problem, das gelöst werden muss, oder einer Aufgabe, die optimiert werden soll. Wenn eine Idee für eine Software tatsächlich in die Entwicklungsphase übergeht, wird zunächst eine Anforderungsanalyse durchgeführt [2, S. 5]. In diesem Schritt, dem sogenannten „Requirements Engineering“, werden aus den gewünschten Funktionalitäten des Programms Anforderungen abgeleitet, zusammengefasst und verschriftlicht. Auf dieser Grundlage können dann konkrete Lösungen vorgeschlagen werden, welche im Gegensatz zu den Anforderungen technisch formuliert sind und häufig bereits Codeabschnitte, Blockdiagramme und andere Methoden zur Illustration und dem direkten Aufzeigen von Lösungsansätzen enthalten. Dieser Schritt wird als Systemspezifikation bezeichnet [2, S. 5]. Anschließend erfolgt die Umsetzung des Designs durch das Entwerfen und Realisieren der einzelnen Softwarekomponenten und ihrer Bestandteile. Diese Komponenten werden zusammengeführt und getestet, bevor in Abhängigkeit des gewählten Prozessmodells ein lauffähiges Programm entsteht. Dieses Programm wird dem Kunden zur Abnahme und Inbetriebnahme übergeben.

Es existieren unterschiedliche Prozessmodelle für die Softwareentwicklung, welche sich hinsichtlich ihrer Reihenfolge, Dimensionen und Terminologien teilweise erheblich unterscheiden. Während früher vor allem die klassische Softwareentwicklung verbreitet war, werden seit der Jahrtausendwende vermehrt agile Entwicklungsprozesse eingesetzt. Im Rahmen dieses Kapitels werden zunächst die klassische Softwareentwicklung am Beispiel des Wasserfall- und V-Modells erläutert und deren Nachteile benannt. Anschließend wird anhand des Scrum-Modells erklärt, wie agile Softwareentwicklung diese Nachteile kompensiert. Im letzten Abschnitt wird begründet, aus welchen Gründen trotz der Vorteile agiler Methoden eine klassische Herangehensweise bei der BuggyTech Engine Entwicklung gewählt wurde und welche spezifischen Vorteile dies mit sich bringt.

2.1 Gegenüberstellung klassischer und agiler Softwareentwicklung

Das Wasserfallmodell ist aus dem Bedürfnis nach Vereinheitlichung und Systematisierung der Arbeit entstanden und beinhaltet sequentielle Phasen mit Meilenstein-Dokumenten als Freigabe für den Übergang in die nächste Phase [3]. Die sequentielle Arbeitsweise erlaubt es, Ursachen von Problemen zu beheben, anstatt nur Symptome zu bekämpfen. Aufgrund seiner strukturierten Natur und umfassenden Dokumentation erfordert das Wasserfallmodell im Vergleich zu anderen Herangehensweisen im Software-Entwicklungsprozess weniger Kundenkontakt. Zu Beginn des Projekts werden die Anforderungen und Spezifikationen gemeinsam mit dem Kunden erarbeitet und präzise festgelegt. Dies gewährleistet beidseitig eine klare Vorstellung darüber, was der Kunde erwartet und ermöglicht dem Dienstleister eine genaue Bestimmung dessen, was er zu leisten hat. Aufgrund dieser klaren Struktur ist die Arbeitsweise mit diesem Modell gut organisierbar. Allerdings kann der geringe Kundenkontakt zu einer mangelnden Qualitätssicherung führen, da die Anforderungen möglicherweise nicht ausreichend validiert werden oder Änderungen im Laufe des Projekts nicht angemessen berücksichtigt werden können, da sie nach der Anforderungsanalyse nicht vorgesehen sind.

Das V-Modell zeichnet sich insbesondere durch seine integrierte Qualitätskontrolle der einzelnen Schritte aus, was ihm gegenüber dem Wasserfallmodell einen großen Vorteil verschafft [2, S. 6]. Vor der eigentlichen Implementation werden Teilbereiche getestet und validiert. Durch feste Bezeichnungen und umfassende Versionsdokumentation wird die projektinterne Kommunikation verbessert. Allerdings besteht aufgrund der streng sequentiellen Arbeitsweise nach wie vor das Risiko, dass Probleme das gesamte Projekt aufhalten können. Sich ändernde Kundenwünsche können auch im V-Modell lediglich im ersten Schritt der Entwicklung beachtet werden, wodurch es ähnlich starr wie das Wasserfallmodell ist.

Herausforderungen der klassischen Softwareentwicklung

Aktuelle Softwareprojekte sind jedoch nicht ausschließlich von sich ändernden Kundenwünschen betroffen. Die Bedingungen des Marktes und der Stand der Forschung ändern sich rasant und häufig innerhalb von Tagen, während Projekte und deren Planung oft über Jahre laufen. Veränderungen auf dem Weltmarkt oder neu entwickelte Materialien können daher sofortige Änderungen der Projektanforderungen verlangen. Die klassischen Modelle haben früher eine Daseinsberechtigung gehabt, da Projekte nicht derselben Sprunghaftigkeit und den hohen Anforderungen an Software unterlagen wie heute. Eine Arbeitsweise, die den Kunden nicht einbindet und nicht auf veränderte Anforderungen reagieren kann, ist heutzutage für Softwareprojekte üblicherweise nicht mehr gefragt. Im Kontext der Softwareentwicklung hat sich die agile Arbeitsweise aufgrund der VUCA-Problematik etabliert, die für „volatile“, „uncertain“, „complex“ und „ambiguous“ steht [4, S. 8] [5], was übersetzt „sprunghaft“, „unsicher“, „komplex“ und „mehrdeutig“ bedeutet. In vielen Projekten ist die VUCA-Problematik von enormer Bedeutung und betrifft nicht nur den Bereich der Spieleentwicklung. Die klassische Arbeit nach einem starren Lasten- und Pflichtenheft hat in vielen Bereichen bereits an Bedeutung verloren. Insbesondere im Kontext der Softwareentwicklung hat sich aufgrund eines dynamischen und volatilen Marktes eine agile Arbeitsweise als wirksames Modell bewährt, das im Gegensatz zur klassischen Vorgehensweise steht. Projekte, die in einer Umgebung mit hohen Veränderungen durchgeführt werden, erfordern eine schnelle Anpassungsfähigkeit und Flexibilität, um wettbewerbsfähig und marktgerecht zu sein. Dokumentation und Struktur sind wichtig, können jedoch nicht den Vorrang vor Effizienz und Sinnhaftigkeit haben. In der gegenwärtigen Zeit wird die strikte Umsetzung von Anforderungen, wie sie im Lasten- und Pflichtenheft festgelegt sind, in der Regel nur noch bei vergleichsweise statischen Projekten wie Bauvorhaben angewendet. Dies bedeutet jedoch nicht, dass bei agilen Methoden keine Anforderungsanalyse durchgeführt wird, denn auch ein Product Backlog stellt eine Sammlung von Anforderungen dar. Der Unterschied liegt vielmehr in der Veränderbarkeit und Anpassungsfähigkeit dieser Anforderungen. Ein Bauvorhaben ist aufgrund der unveränderlichen Gesetze der Physik und Statik von vornherein weniger anfällig für Veränderungen. In Projekten, die wahrscheinlich nur wenige Änderungen erfahren werden und relativ stabil sind, ist eine klassische Herangehensweise nach wie vor sinnvoll.

Ein weiteres Problem bei der Entwicklung von Softwareprojekten ist die Ambivalenz mancher Worte, da die menschliche Sprache nur bis zu einem gewissen Grad konkret werden kann. Begriffe wie „Nutzerfreundlichkeit“ sind interpretierbar und somit subjektiv. Für unterschiedliche Nutzer können verschiedene Aspekte der Nutzerfreundlichkeit von Bedeutung sein, beispielsweise die Übersichtlichkeit, die Anzahl der verfügbaren Optionen oder die Farbwahl. Ein minimalistisches Menü kann für den einen Nutzer benutzerfreundlicher sein, während ein anderer Nutzer mehr Optionen bevorzugt, um das bestmögliche Spielerlebnis zu erzielen. Ein Designer kann auch die Farben der Spielsteine berücksichtigen und gedeckt halten, um die Augen der Nutzer zu schonen, aber dies kann für Far-

benblinde ein Problem darstellen, da sie Schwierigkeiten haben, die Spielsteine zu unterscheiden. Diese Beispiele betonen die Komplexität, die selbst einzelne Wörter einbringen können, sowie die Notwendigkeit, sich dieser Problematik bewusst zu sein und solche Ausdrücke in Dokumenten gezielt zu vermeiden oder zu konkretisieren.

Agile Softwareentwicklung

Die agile Softwareentwicklung, die sich um die Jahrtausendwende herum etabliert hat, legt den Schwerpunkt auf den Kundeneinbezug und die Flexibilität und weniger auf umfangreiche Dokumentation. Änderungen und persönliche Kommunikation haben Vorrang vor Plan und Prozessen. Die lauffähige Anwendung und die Erfüllung der Kundenbedürfnisse sind wichtiger als der Vertrag und die Einhaltung von Richtlinien. Entwicklerteams haben eine flache Hierarchie und eine hohe Autonomie [6]. Zu Beginn eines Scrum-Projekts, das ein Beispiel für agile Softwareentwicklungsprozesse darstellt, wird ein Product Backlog erstellt, das eine änderbare und ständig aktualisierte Anforderungsliste darstellt, sortiert nach Dringlichkeit [7]. Einzelne Anforderungen werden mit Hilfe von „User Stories“ konkretisiert, die aus Sicht des Nutzers beschreiben, was er tun kann und warum das für ihn wichtig ist. Zusätzlich werden oft „Definition of Done“ formuliert, die festlegen, wann eine User Story oder Anforderung genau erfüllt ist. Scrum arbeitet mit festgelegten Zeiteinheiten von ein bis vier Wochen, die als Sprint bezeichnet werden [8, S. 320]. Zu Beginn jedes Sprints werden Anforderungen aus dem Product Backlog ausgewählt und als Ziele für den Sprint festgelegt. Diese Sprint-Backlogs werden dann während des Sprints bearbeitet und die Ergebnisse werden am Ende dessen präsentiert und evaluiert. Die Scrum-Methode erfordert tägliche Treffen der Entwickler, bei denen sie sich in einer vorgegebenen Zeitspanne über den Stand ihrer Arbeit austauschen. Dies führt zu einer verbesserten Zuweisung von Zuständigkeiten, da jeder im Team über die jeweiligen Aufgaben der anderen informiert ist und bei sich überschneidenden Themen der richtige Ansprechpartner direkt identifiziert werden kann. Diese optimierte interne Kommunikation reduziert die Problematik ambivalenter Begriffe durch ständige Kommunikation mit dem Kunden und Teammitgliedern. Am Ende jedes Sprints steht ein Produktinkrement, das eine lauffähige Anwendung darstellt, welche in den darauffolgenden Inkrementen Stück für Stück verbessert wird [9]. Die Intention dahinter ist es einerseits, den Kunden frühzeitig ein Produkt zu präsentieren, um somit Feedback und Anpassungen zu ermöglichen. Andererseits trägt das Erstellen eines lauffähigen Produkts am Ende jedes Sprints dazu bei, die Datenintegrität zu erhöhen, da die Zusammenführung von Oberfläche, Datenmodell und Datenbank oft als Herausforderung empfunden wird. Agile Methoden lösen viele Probleme, die klassische Methoden zur Softwareentwicklung mit sich bringen.

Anwendbarkeit der Prozesse im spezifischen Projektumfeld der BuggyTech Engine und Vorteile klassischer Entwicklung

Obwohl allgemein Bedenken gegen klassische Softwareentwicklung bestehen, spielen diese in manchen Fällen eine untergeordnete Rolle. Im konkreten Projektumfeld der BuggyTech Engine ergeben sich keine Vorteile durch die agilen Methoden im Vergleich zu klassischen Entwicklungsprozessen. Die Bedingungen für die Entwicklung des Spiels sind relativ konstant und die Wettbewerbsfähigkeit sowie Aktualität spielen keine Rolle. Das zu entwickelnde Spiel ist in seiner Komplexität limitiert und orientiert sich an Spielen, die vor 40 Jahren entwickelt wurden. Auch Marktbedingungen und wissenschaftliche Entwicklungen haben geringe Relevanz für das Produkt, wodurch das Projekt wenig Druck ausgesetzt ist. Es besteht kein projekttypischer Zeitdruck und die Motivation ist eher forschender und experimenteller Natur. Eine verbesserte Kundenkommunikation ist nicht nötig und der bei

Scrum vorhandene Vorteil, dem Kunden direkt zu Beginn eine lauffähige Anwendung präsentieren zu können, spielt keine Rolle, da die Software bereits aus einzelnen Komponenten besteht. Die immens verbesserte Teamkommunikation durch Sprint Meetings ist für große Projekte von Vorteil, jedoch nicht nötig aufgrund der geringen Anzahl an Entwicklern.

Um die Zukunftschancen des Projekts BuggyTech Engine zu verbessern und die Entwicklung zu optimieren, wird eine strukturierte Grundlage benötigt. Derzeit besteht das Projekt aus verschiedenen Teilbereichen, die von verschiedenen Entwicklern umgesetzt wurden und es fehlt an einer klaren Zielstellung und Orientierungshilfe. Nachfolgende Programmierer müssen sich mit dem Projektstatus auseinandersetzen und die erforderlichen nächsten Schritte eigenständig erarbeiten, was Zeit- und Motivationsverluste sowie Fehleranfälligkeit mit sich bringt.

In diesem Zusammenhang können die Vorteile der klassischen Entwicklungsmethodik genutzt werden, die sich durch eine akkurate Strukturierung und Dokumentation auszeichnen. Aufgaben werden umfassend und detailliert erfasst, was Vorhersagbarkeit und Kontrolle der zu erledigenden Aufgaben ermöglicht. Das Lastenheft wird als Lösung vorgeschlagen, um eine klare Zielsetzung und eine To-do-Liste für die Entwickler zu schaffen. Indem einzelne Aufgaben aus dem Projekt herauskristallisiert werden, wird die Umsetzung wesentlich realistischer und einfacher. Die Umsetzung einzelner, klar definierter Aufgaben in einem Programmierprojekt führt zu einer höheren Effizienz und Zeitersparnis im Entwicklungsprozess. Eine unzureichende Analyse der Anforderungen kann dazu führen, dass wichtige Aufgaben übersehen werden, was in der Folgezeit zu Komplikationen und Verzögerungen führen kann, insbesondere im Falle von Abhängigkeiten zwischen verschiedenen Aufgaben. Das Lastenheft soll dies verhindern und dient als Anlaufstelle und zentraler Sammelpunkt von Aufgaben und ermöglicht einen Überblick über den Fortschritt des Gesamtprojekts. Es soll ausschließlich als Leitfaden für die studentische Entwicklung der Game Engine dienen und hat keinen konkreten Zeitplan. Es existieren keine klassischen Auftraggeber und -nehmer. Aus diesem Grund wird auf gegenseitige Rechte und Pflichten im Lastenheft nicht eingegangen. Unter Berücksichtigung der genannten Argumente ist es im spezifischen Kontext des BuggyTech Engine-Projekts von Vorteil, ein Lastenheft als zentrale Anlaufstelle und Aufgabenverwaltung für Programmierer zu verwenden, ähnlich wie ein GDD. Aufgrund der isolierten Arbeitsweise der Projektbeteiligten und der Abwesenheit von VUCA-Nachteilen kann eine gewisse Inflexibilität durchaus als Vorteil genutzt werden.

2.2 Begriffserklärung Lastenheft

„Ein Lastenheft ist die vom Auftraggeber festgelegte Gesamtheit der Forderungen an die Lieferungen und Leistungen eines Auftragnehmers innerhalb eines (Projekt-)Auftrags.“ [10, S. 9]

Wenn ein Unternehmen plant, einen Dienstleister mit der Entwicklung eines Projekts zu beauftragen, wenn also eine Dienstleitung von Dritten bezogen wird, wird üblicherweise ein Lastenheft verfasst, das die Bedürfnisse und Anforderungen des Kunden an das Produkt beschreibt. Es beinhaltet insbesondere das „Was und Wofür“ [11, S. 2] des Projekts und wird häufig durch grafische Darstellungen, Tabellen und andere Formen der Illustration unterstützt, um die Anforderungen möglichst genau darzustellen. Es dient dem potenziellen Auftragnehmer als Basis für die Evaluierung, ob er die geforderten Bedingungen erfüllen kann.

Ist dies der Fall, erstellt der Auftragnehmer seinerseits ein Pflichtenheft mit konkreten Lösungsansätzen für die verlangten Funktionen [10, S. 10]. Nach Abgabe des Lastenhefts sollten alle Fragen seitens der Entwickler geklärt sein und der Auftraggeber kann im Pflichtenheft genau einsehen, welche Leistungen er erhält. Diese Grundlage schafft ein gemeinsames Verständnis zwischen beiden Parteien und ermöglicht es ihnen, den Kosten- und Zeitaufwand des Projekts abzuschätzen und einen Vertrag abzuschließen.

Durch konkrete Anforderungen werden Missverständnisse und Unstimmigkeiten reduziert, die oft zu Verzögerungen und zusätzlichen Kosten führen können. Das Ziel des Lastenhefts ist es daher, alle Anforderungen klar und präzise zu beschreiben und so den Weg für ein möglichst vollständiges Produktkonzept zu ebnen und Entwicklungsaufwand zu reduzieren. „Ein Lastenheft ist nur dann korrekt, wenn jede darin angegebene Anforderung eine ist, die die Software erfüllen soll.“ [12, S. 4]

2.3 Game Design Document: Bedeutung und Inhalt

Das GDD fungiert als das grundlegende Dokument für die Entwicklung eines Videospiels, welches alle relevanten Informationen über das Spiel, seine Funktionsweise und das Ziel enthält [13, S. 436]. Es dient als Orientierung für alle an der Entwicklung des Spiels beteiligten Abteilungen und Entwickler, um eine gemeinsame Vision des Endprodukts zu schaffen und möglichst viele offene Fragen zu klären. Eine präzise Zielvorstellung ist entscheidend, damit alle Entwickler am selben Spiel arbeiten können.

Insbesondere bei Spielen mit einer narrativen Komponente, müssen alle relevanten, fundamentalen Aspekte der Geschichte im GDD abgedeckt werden. Wenn die Handlung durch die Entscheidungen des Spielers beeinflusst werden kann, muss genau festgehalten werden, welche Handlung zu welchem Ergebnis führt. Wenn der Spieler Superkräfte erlangen kann, müssen diese und ihre Auswirkungen ebenfalls im GDD dokumentiert werden.

Es ist jedoch wichtig zu beachten, dass das GDD seine Grenzen hat. Während der Entwicklung können sich viele ursprünglich geplante Ideen als nicht umsetzbar, unterhaltsam oder förderlich für das Spielgeschehen herausstellen. Daher ist das Design-Dokument kein unveränderlicher Endzustand, sondern vielmehr ein Leitfaden, der offen für Veränderungen ist und im Laufe der Entwicklung des Spiels aktualisiert werden kann. Aufgrund der komplexen Natur des Mediums Spiel gibt es keine allgemein gültige Vorlage für ein GDD und jedes ist einzigartig [13, S. 437]. So wird eines für einen Tetris-Klon vermutlich keinen Abschnitt über die Geschichte des Spiels enthalten, während ein GDD für ein rein textbasiertes Abenteuerspiel mit hoher Wahrscheinlichkeit kein Moodboard aufweist.

Basierend auf der iterativen Natur der Spieleentwicklung, Playtesting und Balancing sind präzise Werte wie Lebens- oder Angriffskraft in einem GDD selten spezifiziert [14]. Eine umfassende Auflistung aller Aspekte von hundert NPCs wird im GDD nicht enthalten sein. Anstelle davon ist es erforderlich die Rahmenbedingungen für Nebencharaktere den Entwicklern, die für das Erstellen neuer Charaktere verantwortlich sind, zugänglich zu machen. Bei einem Spiel, das hauptsächlich Rätsel und Puzzles bietet, müssen nicht alle Details jedes einzelnen Rätsels im GDD niedergeschrieben werden, da dies das Dokument unübersichtlich gestalten würde. Stattdessen müssen die Rahmenbedingungen der Mechaniken festgelegt werden, einschließlich der Objekte, die Teil des

Rätsels sein können, ihrer Verhaltensweisen und wie der Spieler sie beeinflussen kann. Diese Voraussetzungen ermöglichen es auch Personen, die nicht Teil des Projekts sind, eigenständig Rätsel zu entwickeln.

In GDDs sind gewisse Eckdaten wie Spieldauer, Zielgruppe oder Elevator Pitch häufig anzutreffen, unterliegen jedoch teilweise Änderungen und sind auch nicht für jedes Spiel relevant. Jedes Spiel arbeitet jedoch mit mindestens einer Spielmechanik. Es existieren, insbesondere bei Videospielen, immer Objekte und Interaktionsmöglichkeiten mit diesen und dies muss dem Spieler grafisch und / oder auditiv vermittelt werden. Bei der Analyse von GDDs bestehender Spiele konnten einige Hauptpunkte identifiziert werden, die in den meisten Design-Dokumenten vorkommen. Diese umfassen:

Überblick

Eine Übersicht und Zusammenfassung des Spieles, oft mit sogenannten Elevator Pitches. Die Hauptmotivation des Spielers und das Feeling des Spieles werden hier umrissen. Meist finden sich hier die Schwerpunkte des Designs und die wichtigsten Aspekte des Spieles.

Gameplay

Dieser Abschnitt erläutert den Anreiz des Spieles für den Spieler, wie das Spiel auf ihn wirken soll und welche Emotionen es auslösen soll. Außerdem wird hier dargelegt, welche Ziele er kennen und verfolgen soll, sowie wie und in welcher Reihenfolge er diese erreicht. Es wird beschrieben „wie sich das Spiel spielt“.

Mechaniken

Dieser Abschnitt befasst sich mit den Regeln des Spieles. Alle Interaktionsmöglichkeiten, welche als Spielmechanik bezeichnet werden können, werden hier in Gänze und mit allen Eventualitäten erklärt. Wie der Spieler mit Objekten, oder diese Objekte untereinander interagieren können und wie Fortschritt erzielt wird, ist hier verzeichnet.

Level und Umgebungen

Die meisten Spiele arbeiten mit räumlich oder thematisch getrennten Abschnitten, welche sich nicht selten nach dem Fortschritt des Spielers richten bzw. analog fortschreiten. Es ist daher üblich, diese im GDD voneinander getrennt aufzuschlüsseln und jeweilige Mechaniken, Personen und Objekte den entsprechenden Levels zuzuordnen.

Interface

Das User Interface befasst sich mit der Interaktion zwischen Spieler und Spiel. Steuerung, Menüs, Displays und sämtliche Inhalte der Nutzeroberfläche werden hier beschrieben. Dies bezieht sich nicht ausschließlich auf Grafik, sondern kann auch z.B. auditive Signale beinhalten.

Stil und Gestaltung

Dieser Abschnitt beschreibt und stellt die Ästhetik des Spieles anhand von Moodboards, Skizzen, Beispielen aus anderen Spielen, Farbpaletten uvm. dar. Audiodesign fällt ebenfalls in diesen Bereich, wird aber häufig getrennt oder untergliedert.

Geschichte und Charaktere

Die Geschichte und Hintergründe der Spielwelt werden erklärt und möglichst detailreich beschrieben, ebenso wie Hauptcharaktere oder tragende Personen. Je nach Umfang der Geschichte und Relevanz und Anzahl der Charaktere macht dieser Abschnitt oft den größten Teil des GDDs aus.

Technische Anforderungen

Technische Spezifikationen enthalten Informationen über die Zielplattform, sowie verwendete Hard- und Software.

Zusammenfassend lässt sich konstatieren, dass ein GDD sämtliche Informationen umfassen sollte, die dem Leser eine Vorstellung des Endproduktes vermitteln und dessen Entwicklung ermöglichen, ohne die Komplexität und den Umfang unnötig zu erhöhen oder den Kreativprozess einzuschränken. Typischerweise beinhalten die meisten GDDs jedoch alle erwähnten Punkte mit entsprechender Detailtiefe.

2.4 Die Verbindung von GDD und Lastenheft

Die Rolle des GDD bei der Erstellung von Lastenheften für Game Engines

In Bezug auf ihre Funktion für das zu erstellende Produkt weisen das Lastenheft und das GDD eine signifikante Ähnlichkeit auf. Beide Dokumente dienen dazu sicherzustellen, dass alle Entwickler auf dasselbe Ziel hinarbeiten, indem sie über die Zielsetzung, Inhalte sowie relevante zusätzliche Informationen des Endproduktes informieren.

Die Konzeption einer Game Engine für ein spezifisches Spiel erfordert ein tiefes Verständnis der Anforderungen an das Spiel. Während handelsübliche Entwicklungsumgebungen darauf ausgerichtet sind, eine breite Palette von Spielen zu unterstützen, ist dies bei fallspezifischen Engines aufgrund der begrenzten personellen und zeitlichen Ressourcen nicht möglich und nötig. Ohne Plan und klare Vorstellung des Endergebnisses zu programmieren wäre nicht zielführend.

Ein GDD kann deshalb als nützlicher Ausgangspunkt für die Erstellung von Lastenheften für Game Engines dienen, da es wichtige Informationen zu Spielmechaniken, Grafikstil, Benutzeroberfläche und weiteren Funktionen enthält. Das Entwicklerteam muss genau verstehen, welche Mechaniken und Features im Spiel vorhanden sein sollen und wie diese miteinander interagieren. Durch die detaillierte Beschreibung der Mechaniken und Features im GDD können die Anforderungen und Aufgaben der Game Engine abgeleitet werden.

Dabei können bestimmte Informationen, wie z.B. die Geschichte oder die Charaktere, für die Programmierung vernachlässigt werden, da sie für den Code nicht relevant sind. Ein Moodboard kann beispielsweise lediglich als Orientierung für die Artists dienen. Allerdings können Spezifizierungen des Art Stils für Programmierer durchaus relevant sein, da sie bestimmte Anforderungen an die Game Engine stellen können, zum Beispiel eine hohe Detailgenauigkeit bei vielen Assets oder eine dynamische Pixelbeleuchtung. In diesen Fällen kann das GDD als Referenz für das Entwicklerteam dienen und eine Identifikation technisch relevanter Herausforderungen erleichtern.

Ein Arbeitslauf, welcher in der Anforderungsanalyse auf den Informationen des GDDs aufbaut, kann sicherstellen, dass das fertige Produkt den Anforderungen des Spieldesigns entspricht und dass die richtige Engine für das Spiel entwickelt wird. Dies kann dazu beitragen, den Entwicklungsprozess zu optimieren und das Endergebnis zu verbessern.

Problematik Herangehensweise über das GDD

Eine ausschließliche Konzentration auf das GDD und die alleinige Verwendung dieses als Grundlage für die Erstellung des Lastenheftes kann möglicherweise dazu führen, dass wichtige Aspekte für die Game Engine übersehen werden. Ein GDD für ein Open-World Spiel, das dem Spieler ermöglicht, sich schnell durch die Spielwelt zu bewegen, wird dies lediglich als Feature erwähnen. Für das Entwicklerteam bedeutet dies jedoch, dass bereits früh ein Stress- und / oder Lastentest durchgeführt werden muss, um festzustellen, ob und in welchem Umfang eine schnelle Reise möglich ist. Ein Debug-Log ist für den Prozess der Erstellung einer Engine, sowie die Programmierung des eigentlichen Spieles, durchaus nützlich, wenn nicht sogar unverzichtbar. Aus dem GDD lässt sich dies jedoch nicht ableiten und seine Notwendigkeit ist nicht offensichtlich, für die Engine jedoch elementar. Es lässt sich durchaus argumentieren, dass ein Programmierer, welcher mit einem Lastenheft arbeiten wird, eigenständig das erforderliche Wissen und Verständnis besitzen sollte, um die Notwendigkeit bestimmter Schritte in der Entwicklungsphase zu erkennen. Allerdings kann nicht geleugnet werden, dass ein umfangreiches Lastenheft den Entwicklungsprozess erheblich erleichtern und dazu beitragen kann, dass potenzielle Probleme frühzeitig erkannt und beseitigt werden können. Selbstverständlich kann ein Lastenheft niemals zu 100% technisch vollständig sein und fertige Lösungen enthalten, da ein beträchtlicher Teil der Arbeit letztendlich vom Entwickler selbst übernommen werden muss. Der Designer muss beispielsweise nicht bis ins kleinste Detail darüber informiert sein, welche spezifischen Code-Stücke und Elemente zur Engine gehören und wie sie miteinander interagieren. Trotzdem kann nicht bestritten werden, dass eine beträchtliche Menge an Zeit auf Seiten des Entwicklers eingespart werden kann, wenn der Designer zumindest ein grundlegendes Verständnis von Engines besitzt oder wenn ein Programmierer direkt in die Erstellung oder Überarbeitung des Lastenhefts einbezogen wird. Das GDD für ein Spiel ist ein guter Ausgangspunkt für die Erstellung des Lastenheftes und beschreibt viele wichtige Punkte. Es konzentriert sich jedoch auf die fertigen Ideen und setzt sich nicht mit der Umsetzung und Implementierung dieser Ideen auseinander oder mit den technischen Herausforderungen, die aus manchen Mechaniken resultieren. Eine Zusammenarbeit mit den späteren Entwicklern bei der Erstellung des Lastenheftes ist daher empfehlenswert, um möglichst vollständig zu sein.

Die Verfasserin dieser Arbeit erkennt, dass ihre bisherige Arbeitsweise, bei der sie sich bei der Erstellung des GDDs ausschließlich auf Designaspekte und das Endergebnis konzentrierte und die Programmierung der Engine vernachlässigte, Nachteile aufweist. Dies führte dazu, dass nicht alle Anforderungen explizit formuliert wurden und zusätzlicher Aufwand nötig war, um die technischen

Aspekte des Lastenheftes zu erarbeiten. Erst durch eine detaillierte Auseinandersetzung mit der Audio-Komponente wurde ihr bewusst, dass es einen Unterschied macht, ob ein Sound lediglich abspielbar sein muss oder ob er dynamisch mit anderen Soundeffekten interagieren oder je nach Spielerposition variieren soll. Die Möglichkeit, mehrere Sounds gleichzeitig abspielen zu können, nahm sie als selbstverständlich an, während dies für einen Programmierer, der ausschließlich auf das Lastenheft angewiesen ist, möglicherweise nicht offensichtlich ist.

Zusammenfassend lässt sich festhalten, dass eine ausschließliche Verwendung eines GDDs als Grundlage für ein Lastenheft einer Spiel-Engine nur dann möglich ist, wenn der Verfasser über ausreichend technisches Wissen im Bereich der Engine-Entwicklung verfügt oder in Zusammenarbeit mit einem Experten arbeitet. Andernfalls ist es erforderlich, sich intensiver mit der Programmierung und Entwicklung von Engines zu beschäftigen, um sicherzustellen, dass das Lastenheft vollständig und korrekt ist. In jedem Fall ist es wichtig, dass das Lastenheft die technischen Anforderungen präzise definiert, um eine erfolgreiche Umsetzung des Spiels zu gewährleisten.

Lastenheft für Engine, nicht für Spiel

Es ist anzunehmen, dass das Erstellen eines Lastenhefts für ein Spiel einfacher wäre als das für die Engine, auf der es ausgeführt werden soll. Das Konzept eines Spiels kann umfangreicher sein, jedoch ist es nicht so technisch komplex wie eine Game Engine. In der Regel werden Spiele mit Engines programmiert, die grundlegende Funktionen für die Entwickler übernehmen. Details, wie die genaue Schwerkraft, werden von den Entwicklern justiert, aber das Schwerkraft-Framework selbst existiert bereits.

Ein Lastenheft ist ein Dokument, das einem fremden Team ermöglicht, ein Softwareprojekt zu entwickeln. Spiele sind subjektive Medien und werden nicht allein von Programmierern entwickelt. Sie bedürfen der Überwachung, Bewertung und Modifikation durch Game Designer, die teilweise auch Neukonzeptionierungen durchführen müssen. Ein Spiel muss ständig getestet und angepasst werden. Die Notwendigkeit häufiger Änderungen schließt die Verwendung eines endgültigen Entwicklungsdokuments aus. Die iterative Entwicklung von Videospielen spricht zudem gegen ein vollständiges Outsourcing, was ein Lastenheft obsolet macht.

Eine Game Engine verhält sich jedoch anders und unterliegt diesen Einschränkungen nicht. Unabhängig von genauen Werten des Spieles muss sie konkrete Anforderungen und Bedingungen an den Umgang mit diesen erfüllen. Diese Grundanforderungen lassen sich präzise formulieren und als objektive Aufgaben beschreiben. Eine Audiodatei kann zum Beispiel entweder abgespielt werden oder nicht. Wenn eine Funktion ohne grobe Fehler implementiert wurde, besteht keine Notwendigkeit für spätere Anpassungen. Zudem bietet die Engine als Grundlage für ein Spiel viele Veränderungsmöglichkeiten. Wenn konkrete Werte von Spielobjekten angepasst werden müssen, ändert sich nichts an der Architektur der Engine, sondern lediglich an den Werten, mit denen sie umgeht oder die sie berechnet. Die Engine legt die Grundlage für die Funktionen des Spiels, die jedoch nicht endgültig sondern veränderbar sind.

Basierend auf der variablen Natur von Videospielen während des Entwicklungsprozesses und der Möglichkeit, Enginefunktionen nach ihrer Erstellung zu modifizieren, ist das Lastenheft ein geeignetes Werkzeug für die Entwicklung einer Engine, aber weniger für die Entwicklung eines Spiels.

2.5 Das Spiel „Neon Nova“

In „Neon Nova“ steuert der Spieler ein Raumschiff durch feindliche Raumstationen, leeren Weltraum und fremde Planeten und stößt dabei immer wieder auf feindlich gesinnte Raumschiffe und Monster, denen er sich in interplanetaren Feuergefechten stellen muss. Hierfür benötigt er viel Geschick, schnelle Reaktion und Treffsicherheit, um die Gegner möglichst schnell auszuschalten. Am Ende eines Levels muss er sich Boss-Gegnern stellen, welche mit zusätzlichen Waffen, Fähigkeiten und Lebenspunkten aufwarten und dem Spieler sein ganzes Können abverlangen, um sie besiegen.

„Neon Nova“ wurde von der Autorin entworfen und soll mithilfe der BuggyTech Engine umgesetzt werden. Es handelt sich um einen Scrolling-Shooter in Top-Down-Ansicht, bei dem der Spieler seine Spielfigur durch ein vertikal bewegendes Spielfeld steuert und auf immer stärkere Gegner trifft, die es in zu besiegen gilt. Im Arcade-Stil gehalten, weist das Spiel eine stark gepixelte Darstellung auf, die jedoch durch eine Vielzahl von schnellen, bunten und aufregenden Effekten ergänzt werden soll. Es orientiert sich stilistisch und in der Spielweise grob an bekannten Arcade-Titeln wie „Truxton“[15]. Das Genre des Arcade-Shooters wurde bewusst für die Umsetzung mit der BuggyTech Engine ausgewählt. Sobald diese Zielsetzung erreicht ist, bietet die Engine die Möglichkeit einer zukünftigen Erweiterung und Ausbau. Zu Beginn des Projekts wird es jedoch als vorteilhaft erachtet, ein eher simples Spiel auszuwählen, welches die Engine weder mit großen Datenmengen noch mit komplexen dreidimensionalen Berechnungen unnötig belastet. Der Vorteil von Arcade Shootern besteht darin, dass das Spielprinzip erprobt und beliebt ist und durch seine Simplizität leicht umsetzbar ist. Infolge des Einsatzes von 2D-Grafiken anstelle von aufwändigen 3D-Modellen mit Rigs, ergibt sich ein Vorteil in Bezug auf die Durchführung von Tests, da die Anforderungen an die Testumgebung reduziert werden. Derartige Assets können zudem einfacher erstellt werden und erfordern keine spezielle Hard- oder Software. Die Entwicklung einer Spiel-Engine stellt eine anspruchsvolle Aufgabe dar, welche eine Vielzahl von technischen Herausforderungen und Anforderungen mit sich bringt. Daher sollten alle verfügbaren Möglichkeiten genutzt werden, um die Komplexität zu reduzieren und die Entwicklungszeit sowie den Aufwand zu minimieren. Dies soll durch die Wahl eines simplen Spielmodells gewährleistet werden.

2.6 Definition semantisches Datenmodell

Ein Datenmodell ist eine strukturierte Organisation und Verknüpfung von Daten [16]. Verschiedene Arten von Datenmodellen haben den Zweck, die Bedeutung und den Zusammenhang von Daten verständlich zu machen, indem sie die Eigenschaften und Beziehungen von Objekten modellieren [17, S. 65]. Datenmodelle können textuell oder grafisch als Diagramme dargestellt werden und helfen bei der Planung und Entwicklung von Datenbanken sowie anderen Anwendungen, die Daten speichern und verwalten müssen. Die Modelle beinhalten Entitäten, Attribute und Beziehungen. Entitäten stellen Abstraktionen von Objekten oder Dingen dar, während Attribute Eigenschaften oder Merkmale von Entitäten beschreiben. Beziehungen definieren Verbindungen zwischen verschiedenen Entitäten und beschreiben ihre Natur. Das semantische Datenmodell ist im Unterschied zu anderen Datenmodellen auf einer höheren Abstraktionsebene und beschreibt die Objekte, ihre Eigenschaften und Methoden, aus denen das System besteht. Es ist also anders als andere Datenmodelle, die sich hauptsächlich auf die Beziehungen zwischen den Daten und mathematische Grundlagen konzentrieren. Das semantische Datenmodell trägt zur einheitlichen und eindeutigen Definition der Daten bei, was die Datenqualität und -zuverlässigkeit steigern kann. Es bietet eine abstrakte Sicht auf die Daten eines

bestimmten Anwendungsbereichs und wird oft bei der Entwicklung von Datenbanken verwendet, um die Anforderungen der Benutzer und die Beziehungen zwischen den Datenobjekten zu erfassen. Das Entity-Relationship-Modell kann als ein Beispiel einer Anwendung eines semantischen Datenmodells betrachtet werden, das auf objektorientierten Konzepten basiert [18, S. 9-10].

2.7 Definition Softwareentwurfsmuster

Softwareentwurfsmuster sind standardisierte Lösungen für wiederkehrende Probleme in der Softwareentwicklung[19, S. 27]. Sie dienen der sinnvollen Strukturierung von Code und bieten einen lösungsorientierten Ansatz. Softwareentwurfsmuster können modular eingesetzt werden, um die Entwicklung von Software zu erleichtern und zu standardisieren [20, S. 53].

Das Flyweight Muster [19, S. 220] ist ein solches, in der Softwareentwicklung etabliertes Entwurfsmuster, welches dem Zweck dient, den Speicherbedarf einer Anwendung durch die gemeinsame Nutzung von Objekten mit identischen Merkmalen zu reduzieren. Komplexe Objekte werden dabei aus einzelnen Komponenten zusammengesetzt und enthalten nur noch extrinsische Daten [19, S. 220]. Dieses Muster ist insbesondere bei der Erstellung großer Anwendungen mit vielen gleichartigen Objekten von Relevanz, da durch die Verwendung von Flyweights der Speicherbedarf und somit auch die Ausführungsgeschwindigkeit optimiert werden können.

3 Einbindung unterstützender Programmiermuster und Erstellung der Datenmodelle

Da es sich bei einer Game Engine um ein datenverarbeitendes System handelt, ist eine möglichst exakte Beschreibung dieser Daten vonnöten. Das GDD beschreibt die zu verarbeitenden Objekte in einer nicht-formalen Weise, z.B. durch Fließtext, Aufzählungen oder Skizzen. Daher wird im folgenden Kapitel die Erstellung eines semantischen Datenmodells beschrieben, um eine formale Beschreibung der von der Engine zu verarbeitenden Daten und deren Strukturen zu erhalten. Im Folgenden werden Programmiermuster beschrieben, mit welchen die Datenmodelle in Programmcode umgesetzt werden können.

3.1 Programmiermuster und Entity-Relationship-Modell

Um eine reibungslose Ausführung des Programms bei einer großen Anzahl von Objekten zu gewährleisten und Cache-Misses zu reduzieren, wählte Frau Klaus für die Strukturierung der BuggyTech Engine ein auf Entitäten und Komponenten basierendes System [21, S. 41].

Im Entity-Component-System (ECS) werden Spielobjekte aus einer Kombination von Komponenten erzeugt, die an einer Entität befestigt werden. Eine Komponente definiert dabei die Eigenschaften eines Spielobjekts wie beispielsweise dessen Geschwindigkeit oder Lebenspunkte. Eine Entität ist ein Container, der verschiedene Komponenten enthält, um ein bestimmtes Objekt darzustellen. Durch dieses Konzept können Objekte aus einzelnen Teilen flexibel erstellt und modifiziert werden [22].

Das Flyweight-Muster ist eine Implementierung des ECS, das bei der Verarbeitung von großen Objektmengen eingesetzt wird. Um die Effizienz des Systems zu erhöhen und den Speicherbedarf der Anwendung zu reduzieren, können das Flyweight- und das Prototype-Muster verwendet werden. Diese Techniken ermöglichen die Erzeugung und Verwendung von Objekten mit geringem Speicherverbrauch und minimieren die Anzahl der notwendigen Objekterzeugungen, was zu einer effizienteren Verarbeitung der Daten führt. Flyweights sind Objekte, die gemeinsame Daten nutzen und daher keine eigenen Kopien des Datensatzes vorhalten, sondern alle auf denselben Datensatz verweisen. Dadurch wird eine effiziente Nutzung von Speicherressourcen ermöglicht und der Gesamtspeicherbedarf reduziert. Eine Möglichkeit zur Implementierung des Flyweight Musters besteht in der Verwendung einer Flyweight Factory [19, S. 220-233], welche einen Objektpool erstellt und verwaltet. Dieser Objektpool enthält bereits erstellte Objekte der entsprechenden Art, welche alle gemeinsamen Merkmale teilen. Durch die Wiederverwendung dieser Objekte kann der Speicherbedarf reduziert werden. Wenn ein Client ein Objekt anfragt, überprüft die Flyweight Factory, ob ein solches Objekt bereits im Objektpool vorhanden ist und gibt es zurück oder erstellt ggf. ein Neues. Die individuellen Merkmale, welche für jedes Objekt unterschiedlich sind, werden erst bei Verwendung des Objekts als Parameter an den Client übergeben.

Spieler und Gegner weisen viele unterschiedliche Merkmale auf, teilen jedoch ähnliche Fähigkeiten und greifen teilweise auf dieselben Ressourcen zurück. Der Aufbau der Objekte ist prinzipiell gleich, jedoch unterscheiden sie sich in ihren intrinsischen Daten, weshalb sich die Anwendung des Flyweight-Musters anbietet. Dadurch können Eigenschaften, die allen Objekten des Typs „Raumschiff“ gemeinsam sind, effizient individualisiert werden.

Prototype Pattern

Das Prototype-Muster eignet sich zur effizienten Erstellung einer Vielzahl ähnlicher Objekte, welche sich lediglich in ihren Attributen oder Werten unterscheiden, also zum Beispiel Schadenspunkte oder Sprite. Es ähnelt dem Flyweight-Muster, da ein Objektmodell als Grundlage für verschiedene Variationen dient. Im Unterschied zum Flyweight-Muster wird beim Prototype-Muster ein Prototypobjekt erstellt, das kopiert und anschließend modifiziert wird. Durch diese Vorgehensweise können viele Varianten eines Basisobjektes ohne vollständige Neuerstellung erstellt werden. Das Prototype-Muster ist insbesondere bei der Erstellung komplexer Objekte sinnvoll, die viel Zeit und Ressourcen erfordern, indem sich die Anzahl der Unterklassen reduzieren können, welche sich lediglich insofern unterscheiden, als dass sie ihre Objekte unterschiedlich instanziieren.

Im Kontext der BuggyTech Engine empfiehlt sich die Anwendung von sowohl dem Flyweight als auch dem Prototype Muster insbesondere bei der Erstellung von Schiffen, Projektilen und Boden Tiles, da diese Objekte vergleichsweise einfach gestaltet sind und sich lediglich in ihren spezifischen Attributwerten unterscheiden. Darüber hinaus können durch die Verwendung dieser Muster die Effizienz der Objekterstellung verbessert und der Speicherbedarf reduziert werden. Beispielsweise beschränkt sich bei den Hintergründen die Unterscheidung lediglich auf die Position und das Sprite.

3.2 Entity Relationship Modell

Ein ERM definiert Entitäten, ihre Beziehungen und Attribute. In der entity-component-basierten Architektur der BuggyTech Engine eignet sich das ERM gut zur Darstellung der Entities und Komponenten. Die Verwendung eines ERMs erleichtert die logische Strukturierung und Organisation von Datenbanken. Durch die grafische Darstellung der Zusammenhänge können doppelte Daten identifiziert und reduziert werden, was die Anwendung des ECS und der in 3.1 beschriebenen Programmiermuster vereinfacht.

Das ERM kann dazu beitragen, Objekte zu identifizieren, die gemeinsame Daten teilen und somit von mehreren Instanzen des Objekts gemeinsam genutzt werden können. Die Aggregation ist eine Möglichkeit, Entitäten, Komponenten und deren Zusammensetzung darzustellen. Das Flyweight-Muster und dessen Anwendungsfälle können effektiv durch eine Visualisierung und Planung mittels des Entity-Relationship-Modells unterstützt werden, indem Datensätze identifiziert werden, die eine gemeinsame Nutzung ermöglichen. Ein weiterer Vorteil von ER-Modellen ist die Fähigkeit, Beziehungen zwischen Daten zu identifizieren und sicherzustellen, dass Datentypen konsistent sind, was die Datenintegrität gewährleistet. Darüber hinaus sind ER-Modelle äußerst flexibel und können bei Bedarf leicht modifiziert werden, um Veränderungen zu integrieren und neue Anforderungen zu erfüllen. Aus diesem Grund wurde für die Objekttypen „Ship“, „Projectile“ und „Floor Tile“ jeweils ein ERM entworfen. Diese Objekttypen haben die Verwendung von Audio-, Grafik- und Game-Logikkomponenten gemeinsam. Daher wurden auch für diese Komponenten ER-Modelle entworfen.

Bei der Erstellung der Modelle wurde jeder Objekttyp detailliert untersucht, indem das GDD konsultiert und alle potenziellen Funktionen und Features berücksichtigt wurden. Dies stellte eine Herausforderung dar, da viele Attribute nicht offensichtlich waren und nur durch eine sorgfältige Analyse des GDD erschlossen werden konnten. Beispielsweise erfordert die Möglichkeit, Schaden zu nehmen, nicht nur die Implementierung einer aktuellen Lebenspunkteanzahl des Objekts, sondern auch einen Startwert, von dem Schaden abgezogen werden kann. Während der Modellierung stellte sich heraus, dass der Einsatz des ERM bereits den gewünschten Effekt zeigte. Zunächst wurden den Objekten ein „Sprite Sheet Counter“-Attribut zugewiesen, das die aktuelle Framezahl der Animation enthalten sollte. Es wurde jedoch erkannt, dass alle Objekte dieses Attribut benötigten und es daher in die Grafik-Komponente aggregiert werden konnte. Das „Ship“-Objekt hatte ursprünglich elf verschiedene „is a“-Beziehungen, eine für jeden möglichen Schifftyp. Dies konnte jedoch effizienter als „Ship Type“-Attribut zusammengefasst werden, weshalb derartige Beziehungen in den ERM nicht modelliert wurden. Bei der Erzeugung neuer Game Objekte nach dem Prototype Muster während der Laufzeit muss daher nur nach dem Attribut „Ship Type“ unterschieden werden.

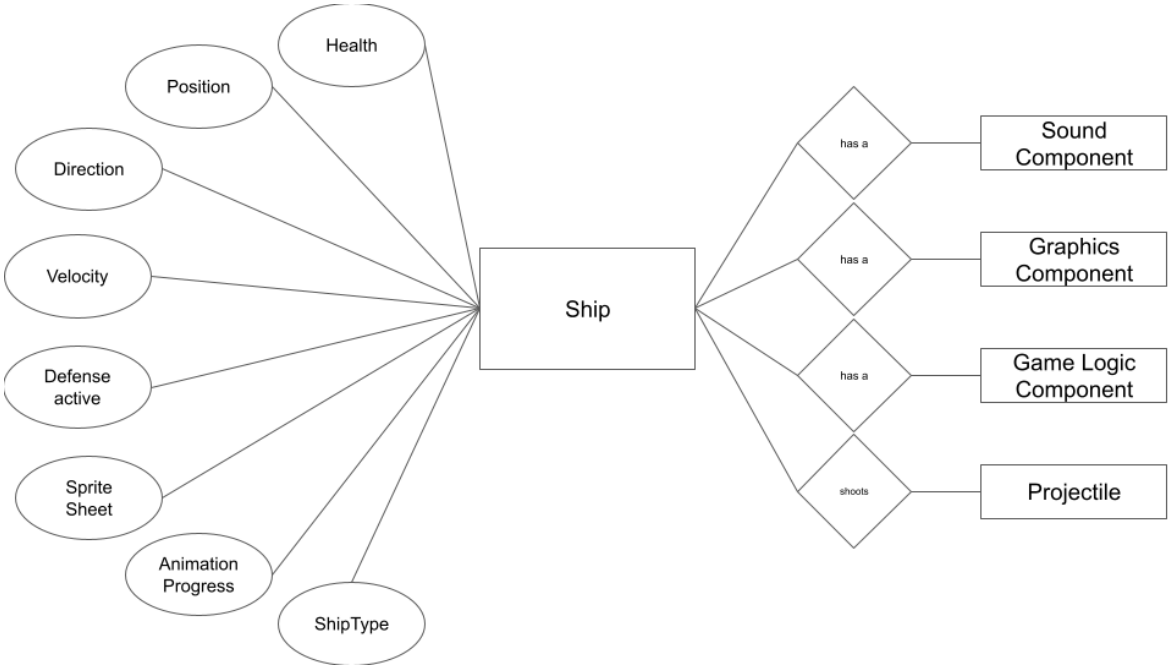

Abbildung 3.1: Datenmodell Raumschiff.

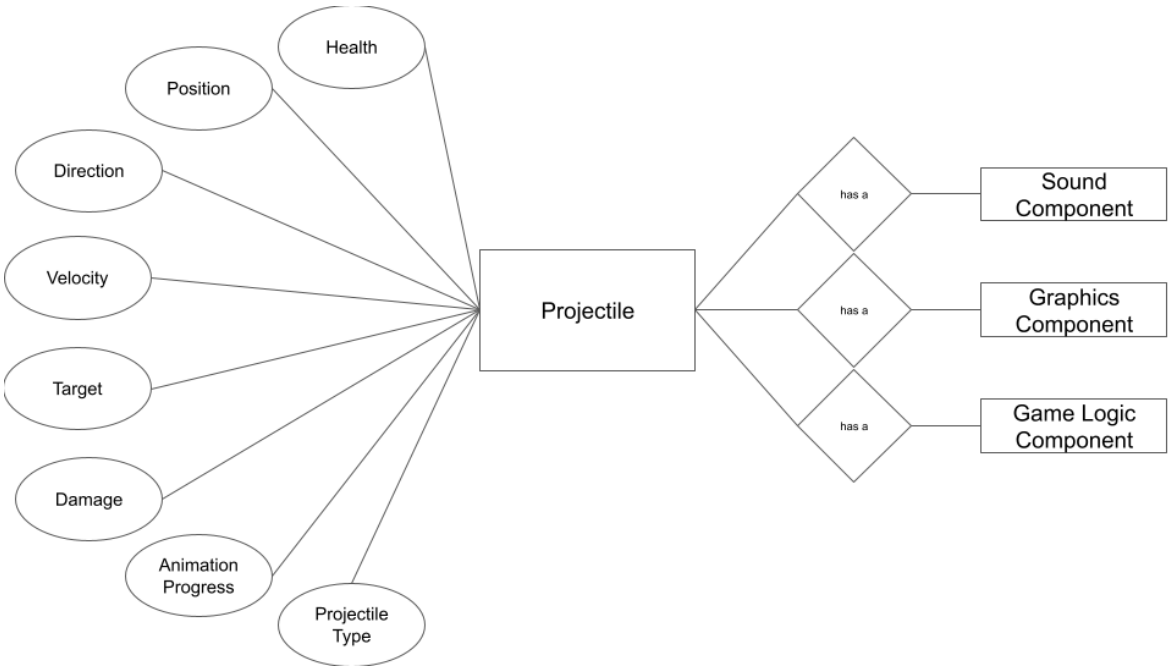

Abbildung 3.2: Datenmodell Projektil.

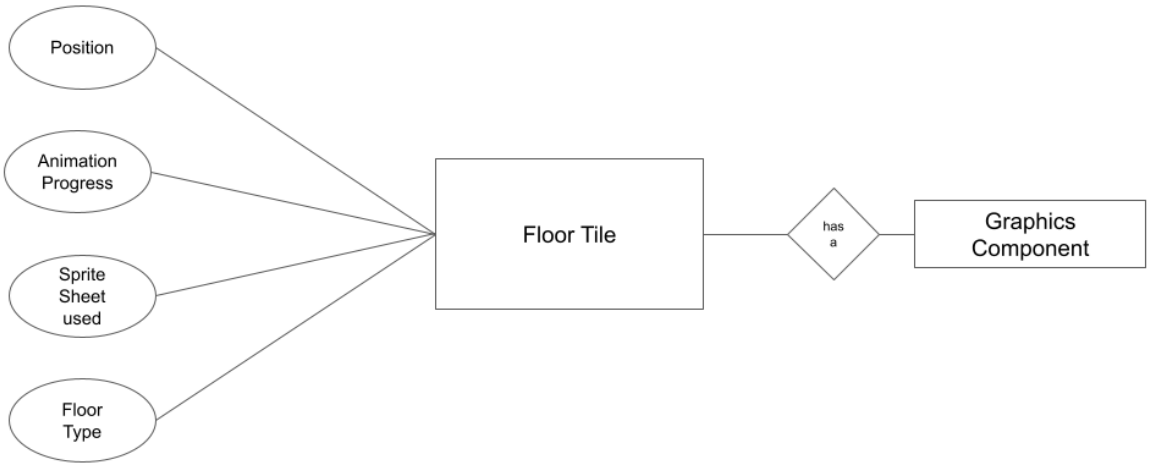

Abbildung 3.3: Datenmodell Bodentile.

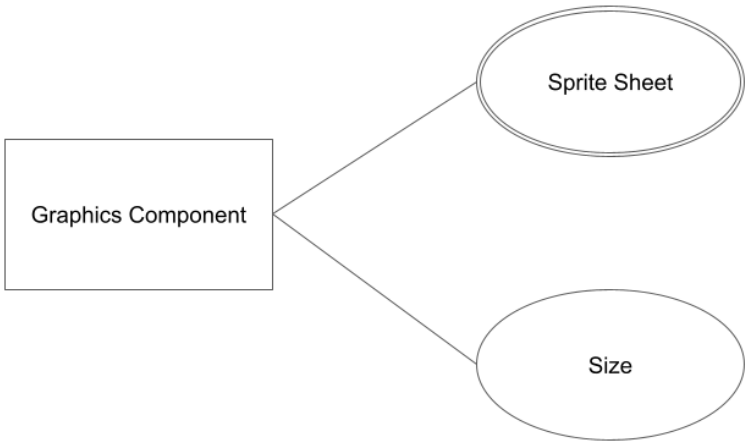

Abbildung 3.4: Datenmodell Grafik Komponente.

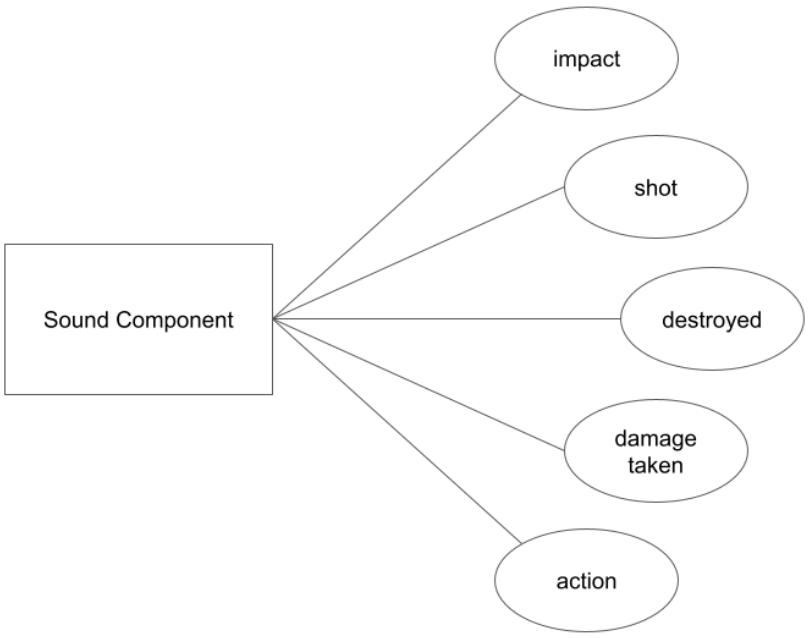

Abbildung 3.5: Datenmodell Audio Komponente.

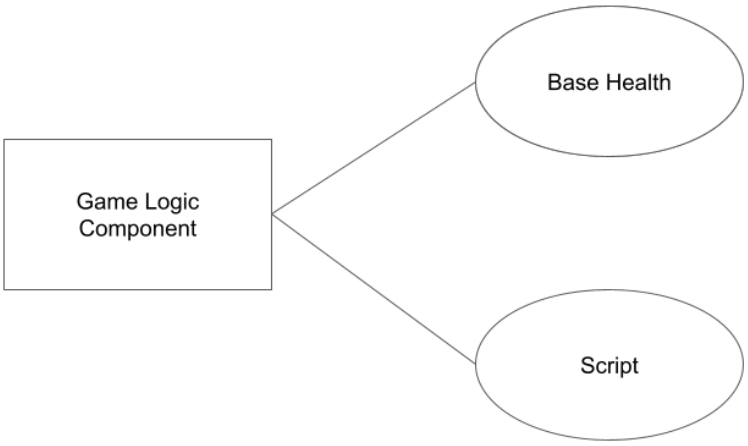

Abbildung 3.6: Datenmodell Spiellogik Komponente.

3.3 Erstellung der Assethierarchie

Eine Animation in einem 2D Spiel ist eine Folge von Bildern welche nacheinander schnell genug dargestellt werden, um eine, für das menschliche Auge als flüssig wahrgenommene Bewegung darzustellen. Die einzelnen Bilder einer solchen Animation sind Sprites und unterscheiden sich nicht von nicht animierten Sprites. In der Assethierarchie sind die benötigten Assets jedoch nach „Sprite“ und „Animation“ sortiert. Dies hat zum Zweck, das Dokument übersichtlicher zu gestalten und den zukünftigen Workload für Artists besser einschätzen zu können, da eine Animation signifikant arbeitsintensiver ist, als eine unbewegliche 2D Grafik.

Gemäß dem GDD hängt die Verwendung von vielen Spielobjekten voneinander ab und die Spielinhalte sind stark logisch getrennt. So haben Gegner jeweils drei verschiedene Assets, deren Verwendung vom aktuellen Level abhängt. Zur Darstellung dieser Abhängigkeiten und zur Erleichterung der Programmierung wurde eine Hierarchie aller zu verwendenden Assets erstellt. Obwohl es sich um eine Hierarchie von Entitätstypen handelt, wird sie im Folgenden als Objekthierarchie bezeichnet.

Zur Generierung der Assethierarchie erfolgte zunächst eine Bestandsaufnahme aller für das Spiel erforderlichen Assets anhand des vorliegenden GDDs. Die erfassten Daten wurden anschließend zur Erstellung einer umfassenden Liste im Anhang aggregiert. Das Game Design Dokument erleichterte diese Aufgabe erheblich, da alle Entitätstypen bereits darin erfasst waren. Anschließend musste die Assetübersicht in eine logische Hierarchie gebracht werden. Das Wichtigste, da stets vorhandene, Objekt im Spiel ist zunächst der Spieleravatar, weshalb die Liste mit dem Spieler und allen korrelierenden Assets beginnt. In der Relevanz für die Spielerfahrung absteigend folgten daraufhin alle Gegner- und Levelassets. Zuletzt wurde die Objekthierarchie um die Interfaceelemente ergänzt. Sobald ein Objekt/Entitytyp identifiziert wurde, wurden daraufhin Variationen und Abhängigkeiten dieser aufgeschrieben. Zum Beispiel gibt es das Spieler-Raumschiff in drei Varianten, wobei je nach deren Spezialfähigkeiten verschiedene Sprites und Animationen benötigt werden. Die Abhängigkeit der Assets voneinander ist in diesem Zusammenhang von Vorteil, da die Auswahl des Raumschiffs

die verwendbaren Fähigkeiten bedingt und diese wiederum die benötigten Assets beeinflussen. Auf dieser Grundlage kann eine Hierarchie von Leveln als Knotenpunkte erstellt werden, welche wiederum alle levelspezifischen Assets als Kindknoten einschließen. Die Verwendung von kontextuellen oder bedingten (conditional) Assets führt zu einer Optimierung der Baumstruktur der Objekthierarchie, da weniger ungenutzte Assets geladen werden müssen. Das Vorgehen bei den nachfolgenden Abschnitten war ähnlich. Zunächst wurden die allgemeinen Objekttypen aufgelistet und anschließend deren Abhängigkeiten erfasst, um zu bestimmen, welche Assets für welche Zwecke benötigt werden.

Um die Liste in eine verwendbare Baumstruktur zu konvertieren, musste das Spiel in logische Abschnitte untergliedert werden. Dies musste in Hinblick auf die Funktion der Objekthierarchie für das Grafik-Subsystem geschehen, welches für das Laden und Löschen der Assets zuständig ist. Durch das Game Design ergeben sich Einschränkungen und Abhängigkeiten einzelner Assets voneinander und von Levelabschnitten. So ist im GDD beschrieben, dass es verschiedene Umgebungen gibt, in welchen Level spielen können. Diese sind künstlich, organisch, natürlich und hybrid, welche eine Mischung aus zwei Arten darstellen. Dementsprechend wurden die Umgebungsassets nach ihren Vorkommen sortiert. Da Menüelemente in einem vom Gameplay getrennten Kontext verwendet werden, sind diese separat gelistet. Die Option in Zukunft Cutscenes oder Textpassagen einzubauen wird im GDD zwar nicht erwähnt, wurde jedoch in die Assethierarchie integriert und die Möglichkeit somit offen gehalten.

Um die Knoten einer Asset-Hierarchie in einer Druckversion besser erkennbar zu machen, wurde die Baumstruktur in mehrere Teile aufgespalten.

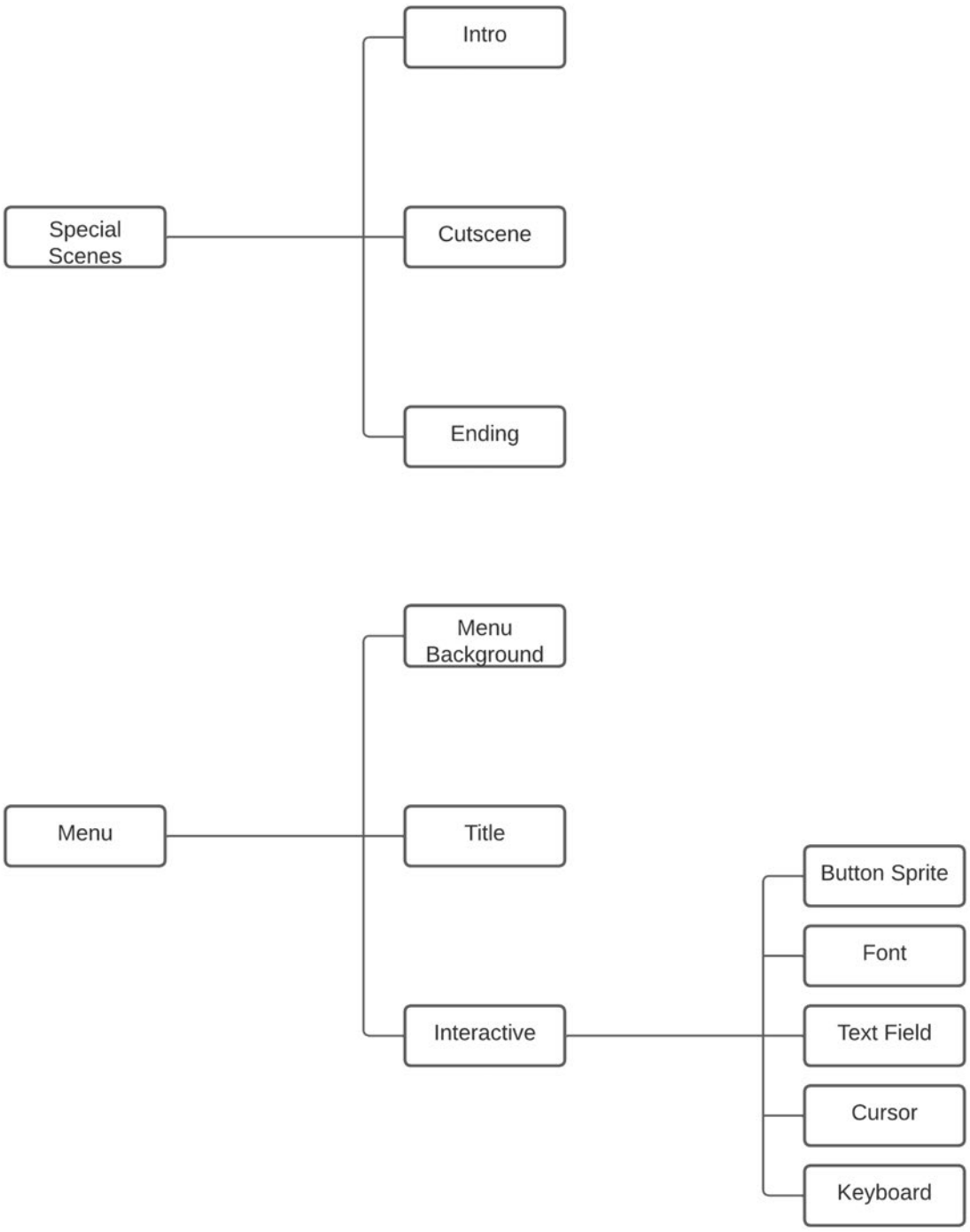

Abbildung 3.7: Die Assethierarchie von Menü- und Cutscenes.

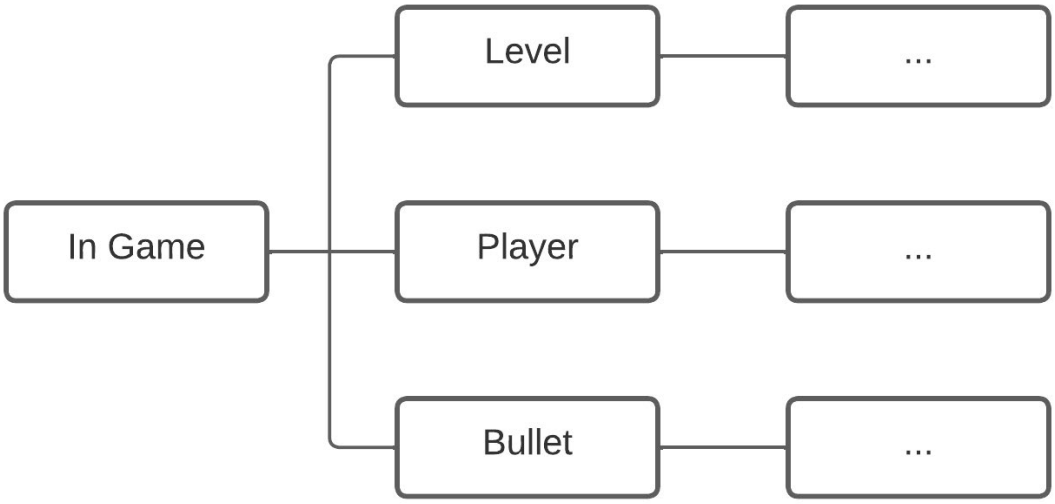

Abbildung 3.8: Die Assethierarchie von In-Game Assets.

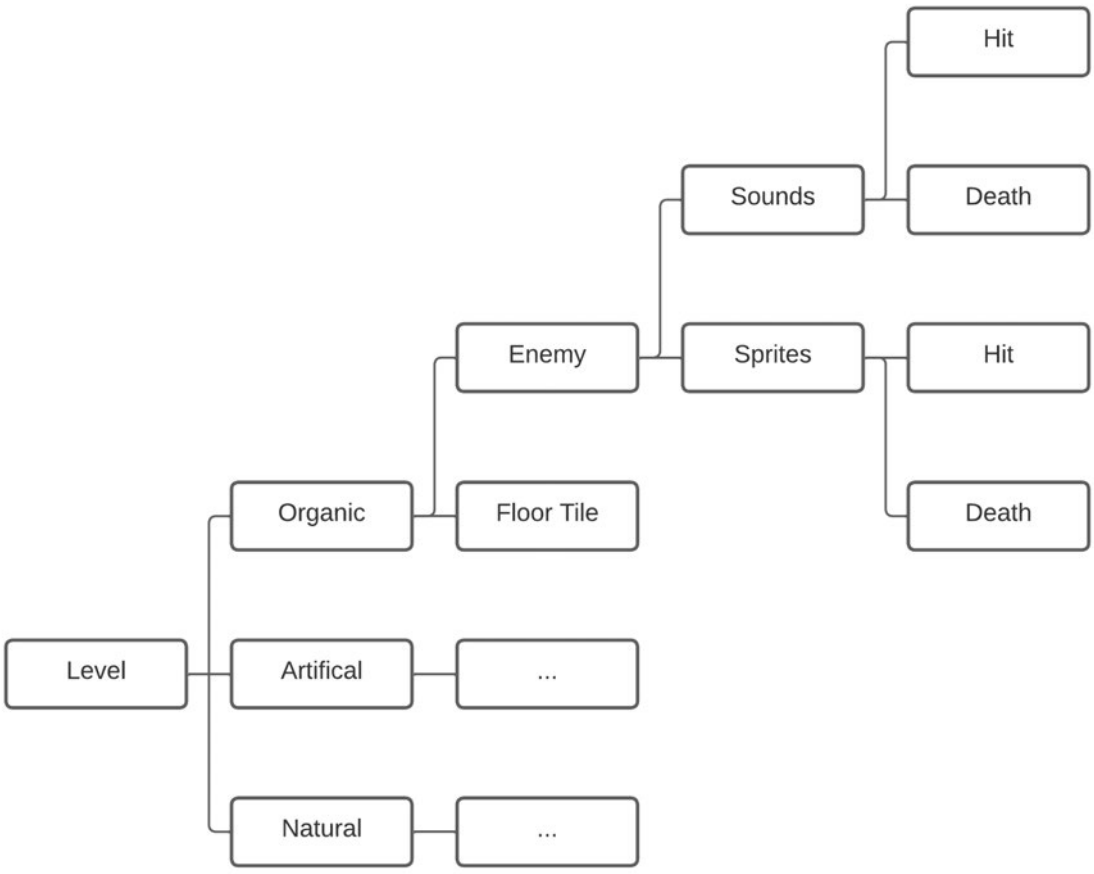

Abbildung 3.9: Die Assethierarchie und Abhängigkeiten von Levelinhalten.

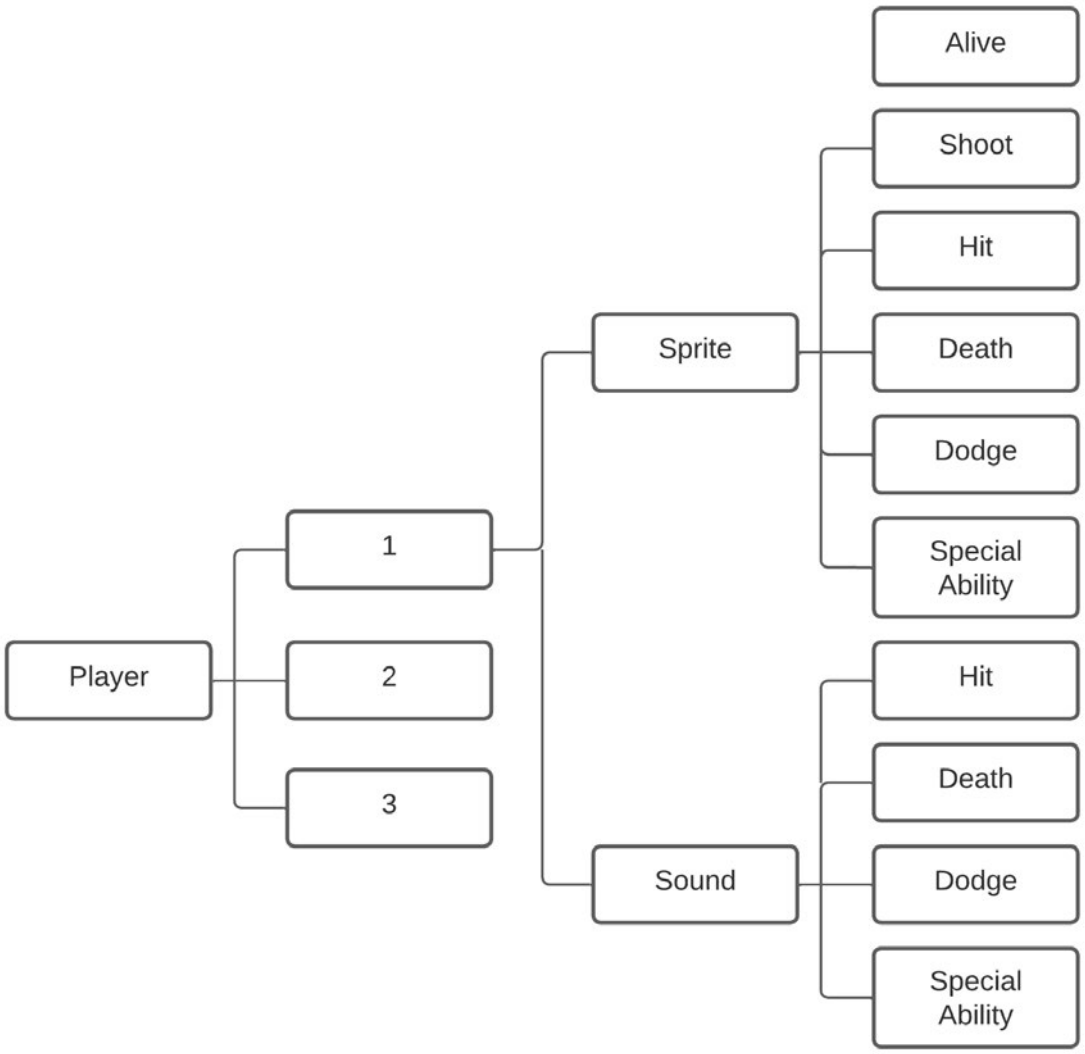

Abbildung 3.10: Die Assethierarchie des Spielers.

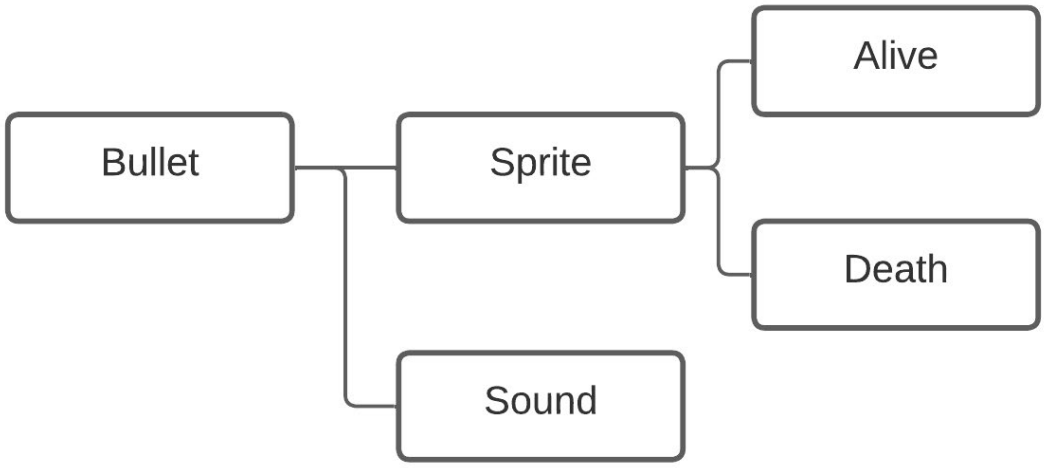

Abbildung 3.11: Die Assethierarchie eines Projektils.

4 Prozess der Anforderungsanalyse

Im vorliegenden Kapitel wird der Prozess der Erstellung des Lastenheftes beschrieben. Dabei wird die gewählte Strukturierung und Reihenfolge erläutert und das GDD auf relevante Inhalte untersucht. Die Autorin setzt sich zudem intensiv mit den bestehenden Komponenten auseinander und untersucht diese auf ihre einzelnen Anforderungen, welche im Lastenheft zu erwähnen sind, insbesondere im Hinblick auf das geänderte Projektumfeld.

Es ist zunächst erforderlich, ein klares Ziel zu definieren und gegebenenfalls wichtige Vorabinformationen darzulegen, bevor der Leser das Lastenheft liest. In der Regel wird zuerst die Ist-Situation beschrieben, gefolgt von der Soll-Situation. Um jedoch die Vollständigkeit zu gewährleisten und sich nicht allein auf vorhandene Systeme zu verlassen oder wichtige Aspekte zu übersehen, wurden zunächst alle Anforderungen erarbeitet und anschließend hinsichtlich ihrer Erfüllung durch vergangene Implementierungen untersucht.

4.1 Zielsetzung definieren und Strukturierung finden

Zielsetzung der Software

Basierend auf dem Lastenheft soll eine Game Engine entwickelt werden, die in der Lage ist, einen im GDD genauer beschriebenen Scrolling Shooter auszuführen. Hierfür soll zunächst ein Prototyp erstellt werden, der die Nutzbarkeit der wichtigsten Mechaniken gewährleistet. Sobald der Prototyp fertiggestellt und funktionsfähig ist, sollen die übrigen Anforderungen implementiert werden. Die endgültige Engine soll flüssig laufen und in der Lage sein, alle im GDD beschriebenen Mechaniken zu implementieren.

Aufgrund der erhöhten Komplexität der Engine ist es zunächst notwendig zu überlegen, welche Funktionalitäten die Engine benötigt, um die Basisanforderungen für den Prototypen des Spieles zu erfüllen. Das Lastenheft soll jedoch nicht nur für den Prototypen, sondern auch für das fertige Spiel gelten. Daher müssen alle wichtigen Anforderungen dafür im Lastenheft berücksichtigt und entsprechend markiert werden. Um eine klare Differenzierung der Anforderungen zu ermöglichen, ist das Dokument nach Relevanz geordnet und die einzelnen Anforderungen innerhalb der Kategorien nach ihrer Wichtigkeit sortiert. Die Identifikation der Notwendigkeit einer systematischen Sortierung der Anforderungen im Lastenheft wurde erst nachträglich erkannt, was zu einem zusätzlichen Aufwand führte, der darin bestand, die Anforderungen im Nachhinein zu sortieren.

Um dem Lastenheft eine logische Struktur zu geben wurde es an den von Christina Klaus identifizierten Subsystemen der Engine orientiert und entsprechend untergliedert. Die vorliegende Anordnung der Kapitel im Lastenheft ist teilweise von der Vorgehensweise der Autorin bei der Identifikation der Anforderungen inspiriert, jedoch nicht streng an diese gebunden.

Um ein vollständiges Lastenheft zu verfassen, sind zudem die Beschreibung der Rahmenbedingungen und gegebenen Umstände sowie die Präzisierung spezifischer Formulierungen erforderlich, um die Struktur des Dokuments zu erläutern und das Lesen zu erleichtern.

Projektumfeld/ Organisation Lastenheft

Die Erstellung der Soll-Situation beginnt mit einer gründlichen Analyse des GDD. Hierbei werden drei zentrale Punkte identifiziert, in die die Anforderungen des Lastenhefts aufgeteilt werden. Physik und Kollision (PK) und Künstliche Intelligenz (KI) stellen hierbei als Teile der Spielmechanik den ersten Abschnitt dar. Obwohl die KI kein Teil der Engine ist, wird ihr aufgrund ihres hohen Aufwands und ihrer inhaltlichen Abgrenzung zu anderen Gameplay-Elementen ein eigenes Kapitel im Lastenheft zugewiesen. Anschließend folgt ein umfassender Abschnitt zur Grafik (GR), der sowohl grundlegende als auch spezifische Anforderungen an die Engine und das Spiel enthält. Die Spezifikation der Interface-Anforderungen schließt sich an. Zur Veranschaulichung wurden Mock-Ups des Interface-Layouts erstellt. Obwohl das User Interface kein Teil der Engine ist, wird ihm aufgrund seiner Bedeutung für das Gameplay ein eigenes Kapitel im Lastenheft zugewiesen. Da das GDD das Interface-Layout nicht ausreichend beschreibt, wurde das Lastenheft um dieses ergänzt. Die Erarbeitung der Soll-Situation endet mit der Kategorie Input Management (IM).

Das Lastenheft dient als Grundlage für die schrittweise Entwicklung der Engine, wobei die einzelnen Anforderungen nach ihrer Relevanz sortiert sind. Die minimal notwendigen Anforderungen sind mit einem Sternchen gekennzeichnet und dienen als Basis für die Entwicklung des Prototyps, der grundlegende Mechaniken und Funktionen testet. Die beschriebenen Voraussetzungen sind ausreichend für das Testen des rudimentären Gameplays. Im nächsten Schritt werden alle Anforderungen hinzugefügt, die den Prototypen in ein gut spielbares Spiel verwandeln und alle geplanten Mechaniken enthalten, um den Zielvorgaben zu entsprechen. Der letzte Schritt ist der Feinschliff, bei dem Features hinzugefügt werden, die die Nutzererfahrung verbessern und die Bedienung erleichtern. Formulierungen im Kapitel „Soll-Situation“, die besagen „Die Engine kann ...“ sind als Bedingung zu verstehen, während die Formulierung „Die Engine muss ... können“ aus Gründen der Lesbarkeit vermieden wurde. Die Kapitel „Grafik“ und „User Interface“ sind jeweils in „Allgemeine Anforderungen“ und „Spezielle Vorgaben“ unterteilt. Das Kapitel „KI“ enthält neben allgemeinen Anforderungen „Höhere Taktik“. Dabei ist zu beachten, dass die Relevanz der einzelnen Anforderungen unabhängig von denen im jeweils anderen Abschnitt betrachtet wird. Vielmehr gilt es lediglich die Reihenfolge innerhalb des jeweiligen Abschnittes zu berücksichtigen.

Die wesentlichen Bedingungen, die für den Prototypen erforderlich sind, um die Grundlagen der Engine zu testen, sind mit einem Sternchen gekennzeichnet. Dies sollte ausreichen, um rudimentäres Gameplay zu testen und sicherzustellen, dass Kernmechaniken und -funktionen vorhanden und funktionsfähig sind. Alle weiteren Anforderungen sind in absteigender Reihenfolge von notwendig über bedingt bis hin zu optional weich gestaffelt. Anforderungen, die direkt nach den Prototyp-Anforderungen aufgeführt sind, sind wesentlich für das fertige Spiel, aber bedingt für den Prototypen. Die Entscheidung über die Wesentlichkeit oder Optionalität der einzelnen Anforderungen obliegt den zukünftigen Entwicklern. In der Regel werden die Funktionen und Daten eines Produkts gemäß dem Schema /LF10/ für Funktionen bzw. /LD10/ für Daten erfasst. Allerdings spielen spezielle Datentypen in diesem Anwendungsfall eine untergeordnete Rolle im Lastenheft. Um die Kategorisierung nach Grafik, Physik und anderen Aspekten deutlicher zu gestalten, wurde das Lastenheft um die folgende Spezifizierung erweitert. Das Lastenheft ist nach folgender Sortierung strukturiert und die einzelnen Produktfunktionen sind entsprechend dem folgenden Schema benannt:

Die Anforderungen können bei Bedarf ergänzt werden, ohne die Sortierung zu beeinträchtigen.

Bereich	Anforderungscode
Physik und Kollision	/PK10/
KI	/KI10/
Grafik	/GR10/
User Interface	/UI10/
Input Management	/IM10/
Audio System	/AU10/
für Prototyp notwendig	/.. ..10/*

Tabelle 4.1: Übersicht der Anforderungscodes nach Bereichen

Es ist außerdem bekannt, dass jede Vorgabe den Entwicklungsprozess erleichtert, indem Entscheidungen im Vorfeld getroffen werden und somit weniger Abstimmungsaufwand notwendig ist. Aus diesem Grund, und um zukünftige Analyse- und Testzwecke zu erfüllen, entwickelte die Autorin zusätzliche Assets, fällte bestimmte Design-Entscheidungen, wie beispielsweise die Schriftart, und wählte bereits einige frei verwendbare Audiodateien aus, um das Lastenheft zu vervollständigen. Die Grafiken wurden speziell für das Spiel gemäß den Guidelines des GDDs entworfen. Die Audiodateien sind thematisch passend und entsprechen ebenfalls den GDD-Vorgaben, wurden jedoch nicht selbst erstellt und können, auch bei einer möglichen Kommerzialisierung des Spiels, ohne Copyrightprobleme genutzt werden. Eine Schriftart, die ebenfalls ohne Copyright ist, wurde ebenfalls zu Testzwecken ausgewählt.

4.2 Erarbeitung der Soll-Situation

Um eine adäquate Analyse der Soll-Situation durchzuführen, ist eine sorgfältige Betrachtung des Game Design Documents unerlässlich. Das GDD beschreibt die Zielvorstellungen des Spiels und somit auch die Anforderungen an die BuggyTech Engine. Es ist daher wichtig, die erforderlichen Kriterien für die Analyse des GDD festzulegen und die Inhalte angemessen zu strukturieren. Auf der Grundlage dieser Analyse müssen dann zusätzliche Anforderungen ermittelt werden, die im GDD nicht explizit genannt sind und in das Lastenheft aufgenommen werden müssen.

4.2.1 Inhalt des Game Design Documents

Der Inhalt des GDDs, welcher für die Erstellung eines Lastenhefts relevant ist, variiert je nach geplantem Spiel. Allerdings gibt es drei grundlegende Elemente, die für alle Videospiele von zentraler Bedeutung sind:

- **Spielmechaniken**
  Jedes Spiel verfügt über spezifische Regeln und Funktionsweisen, die das Spielen überhaupt erst ermöglichen. Diese Regeln beschreiben die Interaktionen zwischen Objekten, ihre gegenseitigen Reaktionen sowie die möglichen Ergebnisse bestimmter Aktionen.
- **Medienausgabe**
  Zur Realisierung einer Spielerfahrung und zur effektiven Kommunikation des aktuellen Spielzustands mit dem Benutzer ist es unerlässlich, dass das System entsprechendes Feedback durch visuelle und / oder auditive Mittel bereitstellt.

- **Nutzerschnittstelle**
  Um dem Spieler die Möglichkeit zur Interaktion mit dem Spielgeschehen zu geben, bedarf es einer Schnittstelle, die es ihm erlaubt, mit den Spielobjekten zu interagieren und deren Zustand zu verändern.

Die genannten Elemente stellen essenzielle Grundpfeiler des Mediums Videospiel dar, da eine virtuelle Spielerfahrung ohne ihr Vorhandensein nicht realisierbar wäre. Deshalb müssen mindestens diese in geeigneter Differenziertheit beschrieben und erläutert werden.

Für das Spiel Neon Nova wurden diese Minimalanforderungen im GDD betrachtet und es wurden Überlegungen angestellt, wie diese umgesetzt werden können. Hierbei wurden die benötigten Ressourcen und die daraus resultierenden Anforderungen identifiziert und berücksichtigt. Ein Platformer mit orthographischer Kamera bringt andere Rendering-, Asset-Pipeline- und Collision-Detection-Herausforderungen mit sich, als eine komplexe dreidimensionale Physik-Sandbox. Solche auf das Spiel bezogenen Anforderungen müssen durch das GDD identifiziert werden.

Feedback für den Nutzer ist grundlegend für ein Programm und muss in Form von visuellen Darstellungen auf dem Bildschirm umgesetzt werden können. Bevor Funktionen programmiert und getestet werden können, müssen die Auswirkungen auf dem Bildschirm sichtbar sein. Die Engine muss in der Lage sein, etwas auf dem Bildschirm darzustellen. Die Objekte auf dem Bildschirm müssen beweglich sein und, im Falle des Spielercharakters, steuerbar sein. Dazu muss die Engine in der Lage sein, Input zu verarbeiten. Spieler und Gegner müssen in der Lage sein, aufeinander zu schießen und Schaden zu verursachen. Dies schließt die KI-gesteuerte Bewegung und Aktivität von nicht vom Spieler steuerbaren Objekten ein. Diese Minimalanforderungen wurden in folgende Kategorien unterteilt:

- Physik und Kollision (Spielmechaniken)
- KI (Spielmechaniken)
- Grafik (Medienausgabe)
- Audio System (Medienausgabe)
- User Interface (Grafik und Nutzerschnittstelle)

Aus jedem Abschnitt des GDDs wurden dann alle identifizierbaren Anforderungen in den entsprechenden Abschnitt des Lastenhefts übertragen.

4.2.2 Spielmechaniken

Physik und Kollision

In der Entwicklung von Videospielen kommen sogenannte Hitboxen zum Einsatz [23]. Diese dienen dazu, einem Objekt eine geometrische Form zuzuweisen, anhand derer die Spiel-Engine Kollisionen erkennen und berechnen kann. Um eine reibungslose Kollisionserkennung zu gewährleisten, müssen sämtliche Objekte in der Kollisionstabelle mit einer entsprechenden Hitbox ausgestattet sein. Auf diese Weise kann die Engine das Verhalten der Objekte im Falle einer Kollision präzise und korrekt berechnen.

Aus dem GDD sowie der in ihm befindlichen Kollisionstabelle geht hervor, dass das Zusammentreffen bestimmter Objekte unweigerlich Auswirkungen auf deren Lebenspunkte hat. Dies impliziert wiederum, dass es möglich sein muss, dass Objekte aufeinandertreffen. Diese müssen also mit Hitboxen versehen werden können. Darüber hinaus stellen genannte Spezialfähigkeiten, wie die „Black Hole Instance“, eindeutige Anforderungen an die Implementierung bestimmter physikalischer Kräfte dar.

/PK10/* Game Objekte / Assets können mit Hitboxen (also Form und Größe) versehen werden.

/PK20/* Eine grundlegende Unterstützung von Kollisionsabfrage und deren Verarbeitung, basierend auf Position, Form und Größe, ist gewährleistet. Das heißt, dass das Aufeinandertreffen von zwei Objekten, basierend auf Position, Form und Größe der Hitbox erkannt und behandelt werden kann.

/PK30/* Wenn eine Kollision erkannt wird, kann ein entsprechendes Event ausgelöst werden.

Aufgrund einer gründlicheren Auseinandersetzung mit Kollisionserkennung in Game Engines fand die Autorin heraus, dass es auch empfehlenswert ist, Methoden zur Kollisionslösung bereitzustellen [24, S. 112]. Diese Anforderung wäre allein auf Basis des GDDs nicht ins Pflichtenheft aufgenommen worden, da ihre Notwendigkeit nicht unmittelbar erkennbar war.

/PK40/* Das Kollisionssystem stellt Methoden bereit, um Kollisionen zu lösen.

/PK50/* Die Hitboxen der Objekte (und physikalische Kräfte) sind während der Laufzeit modifizierbar, also an- und ausschaltbar.

/PK60/ Das Physik- und Kollisionssystem ermöglicht die Implementierung aller im GDD aufgezählten Gameplay Mechaniken. Die Engine kann insbesondere Gravitation simulieren, um die Mechanik „Black Hole Instance” zu ermöglichen.

/PK70/ Die Engine unterstützt Kollisionsgruppen, sodass bestimmte Objekte mit anderen kollidieren können, während Kollisionen mit anderen Gruppen ignoriert werden.

/PK80/ Die Game Engine kann verschiedene physikalische Kräfte wie Geschwindigkeit und Beschleunigung simulieren.

/PK90/ Objekte können zusätzlich zu ihrer Hitbox mit zusätzlichen physikalischen Eigenschaften wie Masse und Gewicht versehen werden. Diese werden bei der Berechnung von Kollisions- und Beschleunigungsvorgängen berücksichtigt.

/PK100/ Die Game Engine kann zusätzliche physikalische Eigenschaften wie z.B. Reibung simulieren können. Dies ermöglicht neue Mechaniken, wie z.B. eine Wolke, in welcher sich alle beweglichen Objekte langsamer bewegen.

KI - Allgemeine Anforderungen

Um ein anspruchsvolles Spielerlebnis zu gestalten, welches mehr erfordert als bloßes geradeaus Schießen, ist es erforderlich, den Gegnern in Neon Nova eine künstliche Intelligenz zu verleihen. Eine simple Hin- und Herbewegung auf einer Achse reicht hierbei nicht aus. Vielmehr müssen die Gegner in der Lage sein, eigenständig Entscheidungen zu treffen, beispielsweise wann sie schießen oder wie sie feindlichen Angriffen ausweichen können. Die verschiedenen Gegnertypen verhalten sich zudem unterschiedlich, wie im GDD im Kapitel „Gegner“ beschrieben. Einige fliegen selbstzerstörerisch auf den Spieler zu, während andere nur schießen. Dies erfordert eine präzise Zielführung und das Wahren von Abstand, während sich der erste Gegnertyp lediglich an die Position des Spielers bewegen muss. Die KI muss in der Lage sein, auf die Eigenschaften und den Status des Spielers sowie ggf. anderer Gegner zuzugreifen, um angemessene Entscheidungen treffen zu können. Aus diesem Grund muss die Engine eine geeignete Schnittstelle bereitstellen, über die die KI auf diese Daten zugreifen kann.

/KI10/* Die Engine stellt Funktionen bereit um Objekte mittels KI-Logik zu bewegen und zu steuern. Dafür sind Position, Status und weitere Informationen von Objekten abfragbar.

Darüber hinaus muss die KI in der Lage sein, auf verschiedene Ereignisse im Spielgeschehen zu reagieren, beispielsweise wenn ein Bossgegner erscheint oder der Spieler eine Spezialfähigkeit einsetzt. Hierfür muss die Engine entsprechende Ereignisse und Benachrichtigungen bereitstellen, die von der KI verarbeitet werden können.

/KI20/* Die KI kann auf das Eventsystem der Engine zugreifen.

KI - Höhere Taktik

Das GDD erwies sich als äußerst hilfreich für die Identifikation der Verhaltensmuster der KI, welche erforderlich sind, wenn Spezialfähigkeiten aktiviert werden. Im Abschnitt „Mechaniken und Spezialfähigkeiten“ werden die Eigenschaften der besonderen Spielerfähigkeiten genau beschrieben, sodass abgeleitet werden kann, wie sich die Gegner in jeder Situation verhalten sollten. Ein Beispiel für eine solche Fähigkeit ist „Decoy“, die es dem Spieler ermöglicht, eine Kopie seines Raumschiffs zu erstellen, die dann zum Hauptziel von feindlichen Angriffen wird, solange sie existiert. Folglich muss die KI der Gegner, sobald diese Fähigkeit aktiviert wird, auf das Decoy zielen, nicht auf den Spieler. Die genannten Anforderungen gehören nicht inhaltlich zu den Anforderungen an die Game Engine, sind jedoch Bestandteil der Game Logik (GL) und wurden als zusätzliche Konkretisierung dem Lastenheft beigelegt.

Da die hier genannten Anforderungen dem Bereich des Game Designs zuzuordnen sind, unterliegen sie einem gewissen Maß an Veränderbarkeit und Anpassung. Die genauen Spezifikationen und Prioritäten können im Laufe des Entwicklungsprozesses variieren und sollten stets den aktuellen Bedürfnissen und Zielen des Projekts angepasst werden.

/KI30/ Objekte mit KI (z.B. Gegner) können im Level navigieren. Das heißt, sie können sich an bestimmte Positionen bewegen oder, mit dem Zweck der Kollisionsvermeidung, ausweichen.

/KI40/ Objekte mit KI können Ziele mit Schüssen treffen oder auf sie zufliegen.

/KI50/ Objekte mit KI können Projektilen des Spielers, Hindernissen und Gegnern ausweichen bzw. vermeiden, in bestimmte andere Objekte hineinzufliegen oder sich von ihnen wegbewegen.

/KI60/ Nutzt der Spieler die Spezialfähigkeit „Decoy“, zielen und schießen die Gegner ausschließlich auf dieses, solange es existiert.

/KI70/ Fliegt eine „Rakete“ auf einen Gegner zu, versucht dieser, ihr auszuweichen.

/KI80/ Ist der Spieler, aufgrund der Fähigkeit „Teleport“ gerade nicht angreifbar, schießen die Gegner nicht auf ihn.

/KI90/ Gegner versuchen aktiv von einer „Black Hole Instance“ wegzufliegen.

/KI100/ Ein Gegner schießt nicht, wenn sich ein weiterer Gegner in seiner direkten Schussbahn befindet.

4.2.3 Medienausgabe

Die Ausgabe von Medien kann in zwei Hauptkategorien unterteilt werden: das Audio- und das Grafiksubsystem. Um eine strukturierte Bearbeitung der Anforderungen zu gewährleisten, wird der Abschnitt über das Audio-Subsystem zuerst behandelt, da er voraussichtlich weniger umfassend ist. Das Grafiksubsystem erfordert hingegen eine Vielzahl von Spezifikationen und wird zudem in allgemeine und explizite Anforderungen unterteilt.

Audio

Im Abschnitt „Audio“ erfolgt keine explizite Beschreibung von Klängen, sondern es werden lediglich die Anforderungen an das Soundsystem sowie die Bedingungen beschrieben, unter denen bestimmte Geräusche abgespielt werden müssen.

Um eine angenehmes Sounderlebnis zu erzeugen und die Wiedergabe von Soundeffekten zum richtigen Zeitpunkt zu gewährleisten, ist es erforderlich, dass Geräusche wiedergegeben werden können, idealerweise in Stereo, um durch die Verwendung von Kopfhörern ein noch immersiveres Erlebnis zu schaffen. Die in das Lastenheft übertragene Anforderung lautet:

/AU10/* Das Audio System kann mehrere Sounds gleichzeitig stereo abspielen.

Um eine Anpassung der Lautstärke während der Entwicklung zu gewährleisten, wird ebenfalls folgende Anforderung an die Engine gestellt:

/AU20/* Die Lautstärke einzelner Soundchannel ist editierbar (vom Entwickler, nicht vom Spieler).

Grafik - Allgemeine Anforderungen

In diesem Abschnitt werden die konkreten Anforderungen an die Darstellung von Grafiken, einschließlich möglicher Transformationen sowie die Technische Anforderungen (TA) an das Grafik-Subsystem, beschrieben. Darüber hinaus werden konkrete Pläne für grundlegende grafische Festlegungen, wie beispielsweise die Ausmaße aller Assets, dargelegt. Im zweiten Teil werden spezifische Informationen in Bezug auf das Spiel in Form von quantitativen Werten und Größenangaben präsentiert. Diese Angaben sind nach Möglichkeit als verbindliche Vorgaben zu verstehen und dienen den Entwicklern als Orientierung während der Umsetzung des Projekts.

Die Fähigkeit der Engine zur dynamischen Darstellung von Objekten auf dem Bildschirm sowie deren Beweglichkeit und Überlagerung von Assets stellen grundlegende Anforderungen dar. Eine wesentliche Rolle spielen dabei Animationen, deren flüssige Wiedergabe durch die Engine gewährleistet werden muss. Ein zusätzliches Feature ist der Parallax Effekt, welcher im GDD als stilistische Anforderung formuliert ist und dementsprechend in die Anforderungsliste aufgenommen wurde.

/GR10/* Die Engine besitzt ein Grafiksubsystem.

/GR20/* Die Engine kann ein Fenster im Vollbildmodus oder im Fenstermodus öffnen.

/GR30/* Die Engine unterstützt das Rendering von 2D Pixel Art Grafiken.

/GR40/* Sprites, Hintergründe und Tilemaps können gerendert und auf dem Bildschirm bewegt, skaliert und rotiert werden.

/GR50/* Die Game Engine ist in der Lage, Animationen darzustellen und die Frames der Animation zuverlässig zu durchlaufen, auch während schneller Bewegungen über den Bildschirm.

/GR60/* Einem Objekt können mehrere Assets zugeordnet werden.

/GR70/* Alpha Blending / Transparenz sind auf Assets anwendbar. Die Game Engine ist in der Lage, Transparenz darzustellen und Alpha Blending zu berechnen / beim Rendering anzuwenden. Assets können über- und untereinander dargestellt werden.

/GR80/ Um den Parallax Effekt darzustellen, ist das Grafiksystem in der Lage, verschiedene Ebenen zu rendern, welche sich in unterschiedlichen Geschwindigkeiten bewegen. Den Ebenen können verschiedene Objekte und Assets zugeordnet werden.

/GR90/ Partikelschwärme können dargestellt und animiert werden. Dabei können verschiedene Parameter wie Größe, Form, Farbe und Bewegung der Partikel festgelegt werden.

/GR100/ Die Geschwindigkeit der einzelnen Ebenen im Parallax Effekt ist abhängig von der Bewegungsgeschwindigkeit des Spielers..

Grafik - Spezielle Vorgaben

Es sind Anforderungen aus dem GDD ablesbar, welche nicht direkt mit den Anforderungen der Engine korrespondieren. Die Autorin entschied sich, diese dennoch im Lastenheft zu erwähnen, da sie die Umsetzung des Projekts erleichtern können.

/GR110/ Die Engine stellt eine Top-Down Ansicht dar.

/GR120/ Die Auflösung des Spieles beträgt 640 x 480, was einem Verhältnis von 4:3 entspricht. Diese muss ganzzahlig dem Bildschirm angezeigt werden. Das Spiel kann im Fenster laufen oder im Vollbild, mit Rändern, welche das 4:3 Verhältnis herstellen, auch wenn der Monitor diesem nicht entspricht.

/GR130/ Die Assettypen sind gemäß den in der Tabelle 4.2 spezifizierten Abmessungen definiert.

/GR140/ Die Assets in Abbildung 4.1 sind für die jeweiligen Objekte im Prototypen zu nutzen.

/GR150/ Die Schriftart „pixel-love-font“ wird für sämtliche In-Game Texte verwendet. Davon ausgenommen sind Fullscreenanzeigen, z.B. vom Titelschriftzug.

Höhe in Pixel	Breite in Pixel	Objekttypen
8	8	Projektile, Partikel, Sprungschuss Spur, Raketen Spur
16	16	Große Projektile, Sprungschuss Projektil, Fadenkreuz Black Hole Instance, Fadenkreuz Teleport
32	32	Sehr große Projektile, Raketen, Kleine Gegner
64	64	Spieler, Gegner, 2 Spielerleben, 2 Boss Leben, Fähigkeitensymbol, Effekte, Decoy, Hindernisse
128	128	Highscore, Große Gegner, Große Fähigkeiten, Schutzschilde, Zonen
256	256	Bosse, Black Hole Instance
640	480	Screen Overlays

Tabelle 4.2: Übersicht über die Assetgrößen

4.2.4 Nutzerschnittstelle

User Interface - Allgemeine Anforderungen

Dieser Abschnitt widmet sich der Spezifikation der Anforderungen an das Benutzerinterface. Er ist ähnlich wie der Abschnitt „Grafik“ unterteilt und umfasst allgemeine sowie spezifische Vorgaben. Die allgemeinen Anforderungen betreffen Funktionen der Engine, während die spezifischen Anforderungen Layout-Mockups und Funktionen der Interface-Elemente enthalten.

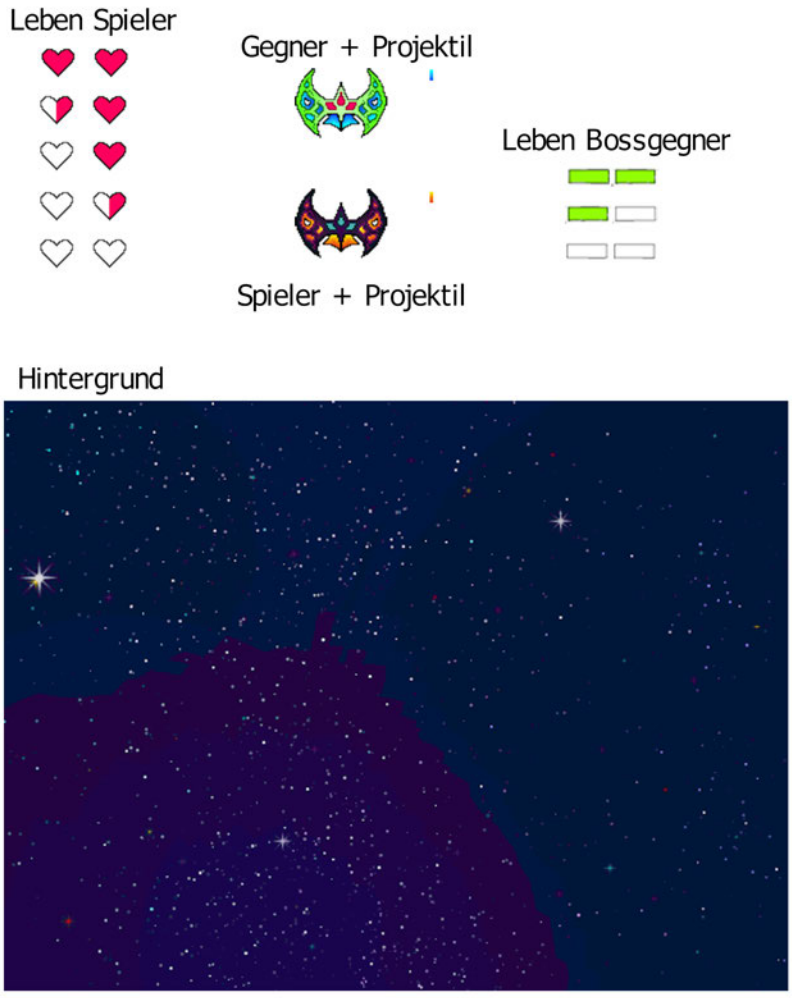

Abbildung 4.1: Eine Auswahl der zu verwendenden Assets.

Das User Interface muss dem Nutzer alle relevanten Informationen über das aktuelle Spielgeschehen in einer ästhetisch ansprechenden und einfachen Form vermitteln. Die Benutzeroberfläche sollte intuitiv und selbsterklärend sein, andernfalls müssen klare Anleitungen in Form eines Tutorials zur Verfügung gestellt werden. Das Interface ist ein wesentlicher Teil eines Spiels, der einer umfassenden Testphase und möglichen Anpassungen unterliegt. Aus diesem Grund ist es schwierig, konkrete, endgültige Vorschläge zu machen. Der folgende Text wurde dem Lastenheft hinzugefügt, um sicherzustellen, dass das Layout und die Funktionen des Interfaces den Entwicklern vorbehalten bleiben.

„Die UI Elemente betreffen sowohl das Grafik- als auch das Input Subsystem. Die Bedienung und Darstellung der Interfaces soll für den Spieler intuitiv, leicht navigierbar und optisch ansprechend sein. Sollte sich im Verlauf der Projektentwicklung herausstellen, dass dies nicht der Fall ist, sind Änderungen, auch grundlegender Art, am Interface und dessen Layout vorbehalten."

Das geplante Interface erfordert die permanente Anzeige bestimmter Daten für den Spieler in Form eines Heads Up Display (HUD). Dabei müssen die HUD-Elemente stets an der gleichen Stelle des Bildschirms positioniert sein. Hierfür muss die Engine in der Lage sein, die entsprechenden Assets anzuzeigen und regelmäßig zu aktualisieren. Der Lebensbalken von Boss-Gegnern muss bei Bedarf ein- oder ausgeblendet werden können. Diese Funktionalität kann auch für zukünftige Interface-Elemente nützlich sein.

/UI10/* Die Engine kann UI Elemente über dem Spielgeschehen darstellen.

/UI20/* Die HUD Elemente (wie Ladebalken für Fähigkeiten) werden regelmäßig aktualisiert und entsprechend der jeweiligen Werte dargestellt.

Neben dem In-Game Interface ist das Menü der zweite wichtige Bestandteil des Interfaces. Die Hauptanforderung an das Menü besteht darin, dass der Spieler in der Lage ist, damit zu interagieren.

/UI30/ Die Engine kann UI Elemente wie Textfelder und Knöpfe darstellen. Der Nutzer kann mit diesen interagieren (Buttons sind anklickbar, Textfelder können beschrieben werden).

/UI40/ Während des Spieles muss ein Pause Menü jederzeit aufrufbar und anzeigbar sein. Der Nutzer muss zwischen verschiedenen Bildschirmen, wie dem Startbildschirm, Game Over und Pause Bildschirm navigieren können.

/UI50/ Der Nutzer kann in den Einstellungen Änderungen an Schriftgröße, Größe der UI Elemente und Größe des HUD vornehmen. Die Einstellungen erlauben zudem eine Anpassung von Farben und Kontrasten, um Farbblindheit auszugleichen.

User Interface - Spezielle Vorgaben

Das GDD enthält keine ausreichend spezifische Beschreibung des Interfaces und lässt wichtige Aspekte wie die Positionierung der Elemente unklar. Um eine bessere Vorstellung des zukünftigen Interfaces zu vermitteln und konkrete Orientierungspunkte zu bieten, hat sich die Autorin eingehend

mit dem Oberflächendesign des In-Game-Interfaces beschäftigt und konkrete Layouts erstellt, die umgesetzt werden sollen. Bei der Erstellung des Designs wurden die Nutzerfreundlichkeit und die ästhetische Übereinstimmung mit bereits erstellten Spielinhalten beachtet. Da alle Assettypen bereits festgelegt waren, standardisierte sie alle Assets entsprechend ihrer Größen, wie in Tabelle 4.2 dargestellt. Zusätzlich wurden technische Vorteile berücksichtigt, wie beispielsweise die Tatsache, dass Bodentiles mit ihren Abmessungen von 96 x 128 Pixel genau sechsmal in das Fenster passen. Die folgenden Anforderungen beinhalten die bei der Erstellung des User Interfaces festgelegten Bestimmungen.

/UI60/ Die Positionierung der Interface-Elemente des HUD entspricht der Abbildung 4.2. Die linke obere Ecke jedes Assets befindet sich an den in Abbildung 4.3 markierten Positionen.

/UI70/ Der Lebensbalken des Spielers füllt sich von rechts nach links. Die Interfacesprites des Spielerlebens haben einen Abstand von 10 Pixeln zueinander.

/UI80/ Der Lebensbalken eines Bossgegners füllt sich von links nach rechts. Die Interfacesprites des Bosslebens haben keinen Abstand zueinander.

Abbildung 4.2: In-Game Ansicht.

Input Management

Die im GDD erwähnte Nutzung von Maus und Tastatur impliziert, dass der Spieler das Menü mittels Mausklick bedienen kann, was dem Standard von gängigen Softwareanwendungen entspricht.

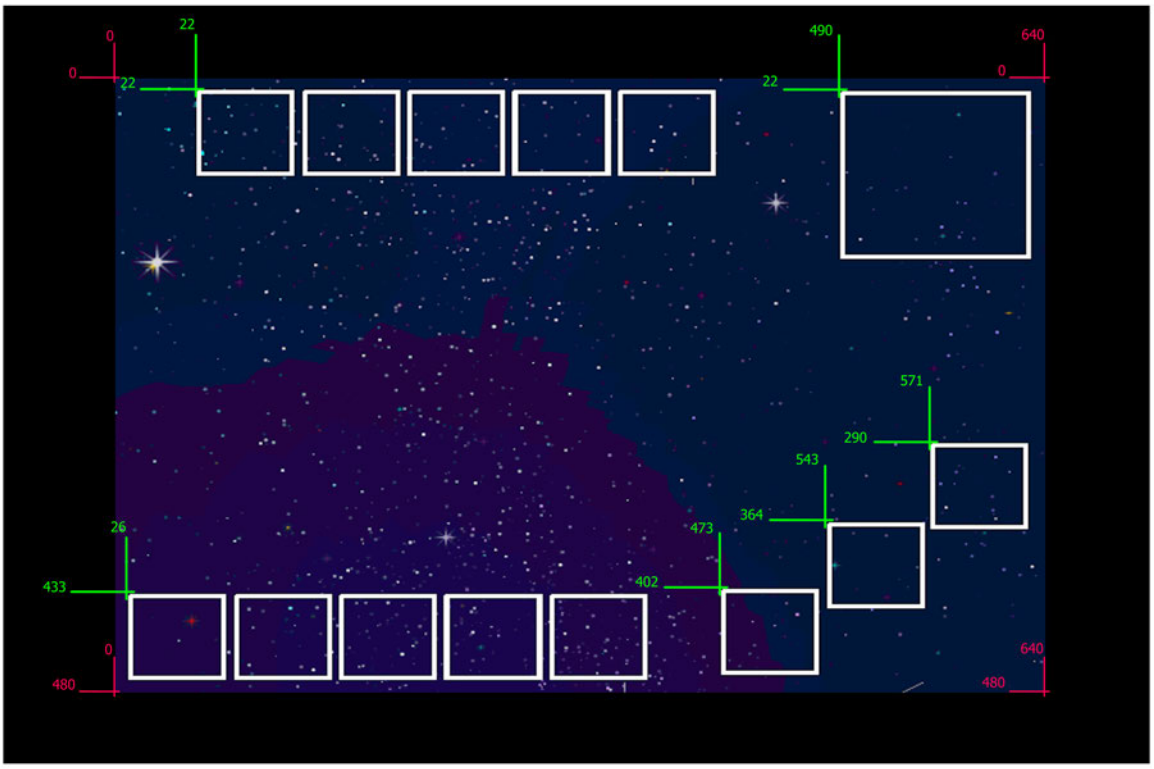

Abbildung 4.3: Koordinaten der Interfaceelemente auf dem Bildschirm.

/IM10/* Die Engine erkennt Nutzereingaben über Maus und Tastatur. Der Spieler kann Elemente auf dem Bildschirm auswählen, seine Spielerfigur steuern und in Textfelder schreiben / Eingaben tätigen.

Die Latenzzeit bei Nutzereingaben des Spielers muss auf ein Minimum reduziert werden, um eine optimale Spielerfahrung mit responsivem Steuerverhalten und ungestörter Immersion zu gewährleisten. Da dies eine grundlegende Anforderung darstellt, wurde die folgende Anforderung mit hoher Priorität in das Lastenheft aufgenommen. Die Autorin führte eine Recherche durch, um einen numerischen Grenzwert für die Verzögerung von Nutzereingaben zu finden. Dabei stieß sie auf ein Experiment, bei dem Probanden ab einer Verzögerung von 115 ms bereits starke Einschränkungen ihrer Reaktionsfähigkeit erlebten und ein schlechteres Spielerlebnis berichteten, wie in Abbildung 4.4 dargestellt. Als Folge wurde dieser Wert als maximal akzeptierte Verzögerung in die Anforderungen aufgenommen, mit einer ideellen Maximalverzögerung von 90 ms, bei der keine signifikanten Auswirkungen auf das Spielerlebnis oder die Reaktionszeit beobachtet wurden.

/IM20/* Nutzereingaben müssen effizient be- und verarbeitet werden und dürfen keine merklichen, den Spielfluss behindernden oder Immersion brechende, Verzögerungen haben. Die Eingabeverzögerung darf 115 ms nicht überschreiten. Im Idealfall beträgt sie unter 90 ms [25, S. 5].

Die Steuerung des Spielers beeinflusst die Geschwindigkeit der Spielfigur gemäß den Anforderungen des GDDs. Konkret wird dort beschrieben, dass das vollständige Bewegen der Achse eines Joysticks oder das vollständige Drücken einer Taste schnellere Bewegungen auslösen sollen, als bloßes Antippen. Daher muss die Engine in der Lage sein, zwischen diesen verschiedenen Eingabearten zu unterscheiden, um eine korrekte Umsetzung der Spielerbewegungen zu gewährleisten.

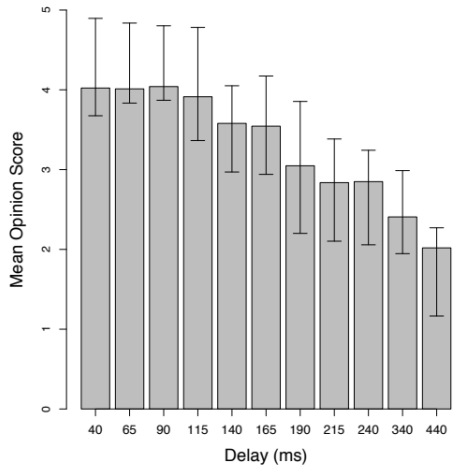

Abbildung 4.4: Bewertung der Spielerfahrung in Zusammenhang mit Eingabeverzögerung [25, S. 5].

/IM30/ Die Engine muss bei der Tastatursteuerung zwischen kurzen Tastenanschlägen und dem gedrückten Halten selbiger unterscheiden und diese als numerische Werte interpretieren können.

Die Implementierung der GDD-Anforderungen zur Nutzung von Gamepads oder Joysticks als alternative Steuerungsoption wird aufgrund ihrer niedrigeren Priorität erst zu einem späteren Zeitpunkt durchgeführt. Die in /IM30/ beschriebene Funktion zur steuerungsabhängigen Bewegungsgeschwindigkeit wird hierbei separat für die Verwendung mit einem Gamepad beschrieben. Hierbei ist zu beachten, dass Arcade-Joysticks keinen Soft-Pull haben und diese Funktion daher bei dieser Steuerungsoption nicht verfügbar sein kann.

/IM40/ Die Engine erkennt Nutzereingaben über ein Gamepad. Der Spieler kann das Spiel mit dem Gamepad bedienen.

/IM50/ Um eine korrekte Steuerung mit einem Gamepad zu ermöglichen, muss die Game Engine eine entsprechende Funktionalität implementieren, die den sogenannten Soft Pull oder Partial Pull des Gamepads korrekt identifizieren und als verwendbaren, numerischen Wert interpretieren kann. Hierfür muss die Engine die Signale des Gamepads auslesen und entsprechend verarbeiten, um die korrespondierenden Bewegungen im Spiel korrekt ausführen zu können.

/IM60/ Die Engine erkennt Nutzereingaben über einen (über USB angeschlossenen) 8-Wege-Joystick. Der Spieler kann das Spiel mit dem Joystick und Knöpfen bedienen.

Ist die Engine in der Lage, verschiedene Eingabegeräte zu erkennen, müssen diese entsprechend priorisiert werden. Es ist notwendig festzulegen, welcher Eingabetyp in welchen Situationen bevorzugt werden soll, um eine einheitliche und konsistente Steuerung für den Spieler zu gewährleisten. Diese Priorisierung ist insbesondere wichtig, wenn der Spieler bereits im Spiel ist oder ein Eingabegerät aktiviert, obgleich schon ein anderer Eingabemodus aktiv ist.

/IM70/ Die Game Engine ist in der Lage, mehrere Eingabegeräte gleichzeitig zu erkennen und zu unterscheiden. Für die Spielersteuerung sind sowohl Gamepads als auch Tastatur und Maus vorgesehen. Wenn ein Spieler eine dieser Optionen auswählt, muss die Engine sicherstellen, dass der Input des anderen Gerätes blockiert wird und keine Auswirkungen auf die Steuerung hat.

4.3 Erarbeitung der Ist-Situation

Das Anforderungsschema wird durch die Ergänzung der in Tabelle 4.3 aufgeführten Punkte erweitert. Die von Frau Klaus entwickelten Subsysteme werden in die Ist-Situation integriert. Technische Anforderungen, die sich hauptsächlich aus bereits vorhandenen Programmteilen ableiten und für die Arbeit an anderen Systemen benötigt werden, werden präzisiert. Profiling/Entwicklertools (PE) wurde aufgrund der Engine-Architektur von Frau Klaus hinzugefügt und stellen lediglich empfohlene Features dar, um zukünftigen Entwicklern die Arbeit zu erleichtern. Ressourcenmanagement (RM) wurde als wichtiger Bestandteil identifiziert und ebenfalls als Kategorie aufgenommen, um eine bessere Unterscheidung zwischen technischen Anforderungen und Ressourcenverwaltung zu ermöglichen. Darüber hinaus werden einige Anforderungen des Grafiksystems konkretisiert und bezüglich ihres Erfüllungsgrades aktualisiert. Der Abschnitt zum Audiosystem (AU) wird um einige Vorschläge von Herrn Stötzer ergänzt, die für den Prototypen nicht zwingend erforderlich sind. Zudem wird die Kategorie Eventsystem (EV) hinzugefügt. Abschließend werden die Änderungen des Projekts seit Projektbeginn betrachtet und die Auswirkungen auf die definierten Anforderungen untersucht.

4.3.1 Analyse vorangegangener Arbeiten und Implementationen

Nach Abschluss der in 4.2.1 genannten Anforderungen wurde der Autorin bewusst, dass die Liste um technische Anforderungen an die Engine erweitert werden musste. Insbesondere im Hinblick auf die noch nicht konkretisierten Subsysteme der Engine stieß die Autorin bei der Erstellung der Systemanforderungen auf Grenzen des GDDs, da dieses keine technischen Spezifikationen enthält, sondern diese lediglich voraussetzt oder impliziert. Die Entwicklung der Game Engine und ihrer Subsysteme wurde bereits von Christina Klaus und Theo Stötzer unter Anleitung von Herrn Daniel Stockmann vorangetrieben. Im Rahmen einer Untersuchung der Ist-Situation mussten die beiden Bachelorarbeiten auf ihre Relevanz für die zukünftige Engineentwicklung analysiert werden. Hierfür werden bereits fertiggestellte Programmteile vorgestellt und die Signifikanz ihrer einzelnen Bestandteile untersucht. Dieser Schritt ist vonnöten, um die Ist-Situation im Lastenheft adäquat darzustellen und den Leser über den bisherigen Fortschritt des Projekts in Kenntnis zu setzen. Da Christina Klaus und Theo Stötzer Teile dieser Systeme bereits erstellt hatten, wurde es als ein günstiger Zeitpunkt betrachtet, mit der Analyse der Ist-Situation zu beginnen und das Lastenheft um eventuell übersehene Anforderungen zu ergänzen. Das Lastenheft wurde um folgende Bereiche und die Anmerkung „i-“ für bereits existente Funktionen ergänzt:

Die Inhalte dieser Tabelle 4.3 befinden sich gemeinsam mit denen aus Tabelle 4.1 im fertigen Lastenheft.

Zunächst beschäftigte sich die Autorin mit der chronologisch früheren Arbeit von Frau Klaus, die sich mit der Konzeptionierung und Implementierung von „Game Loop“ und „Game Clock“ befasste. Diese beiden Konzepte stellen auch ein gutes Beispiel für Anforderungen dar, die nicht direkt aus dem

Bereich	Anforderungscode
Technische Anforderungen	/TA10/
Game Logik	/GL10/
Eventsystem	/EV10/
Ressourcenmanagement	/RM10/
Profiling/Entwicklertools	/PE10/
Ist-Situation, erledigte Anforderung	/i-.. ..10/

Tabelle 4.3: Übersicht der hinzugefügten Anforderungscodes nach Bereichen

GDD abgeleitet werden können. Als Vorbereitung auf den zentralen Teil ihrer Arbeit, der Entwicklung der Game Engine Architektur und der Implementierung eines grundlegenden Frameworks, erstellte Frau Klaus eine grafische Übersicht aller notwendigen Systeme [21, S.39]. Diese ist beizubehalten und ist, in der von Herr Stötzer aktualisierten Version, in Abbildung 4.5 dargestellt. Die Darstellung wird als Referenz für zukünftige Entwickler in das Lastenheft übernommen.

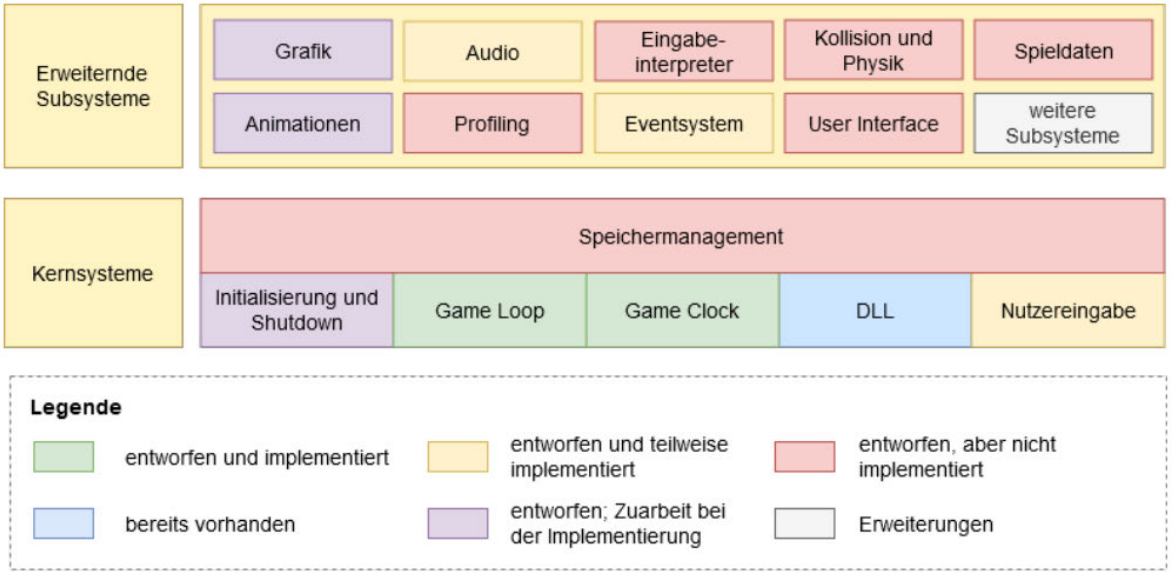

Abbildung 4.5: Von Christina Klaus entworfene und von Theo Stötzer aktualisierte Darstellung der Engine Architektur.

Zudem beschäftigte sich Frau Klaus mit den technischen Anforderungen des Arcade-Automaten, welche sich jedoch, aufgrund der Änderung der Zielplattform, auf die Programmiersprache und die Verwendung bestimmter, von ihr genutzter Frameworks und Bibliotheken reduzieren.

/TA10/* Code ist in C++ zu schreiben.

/i-TA10/* „Der Funktionsumfang von C++11 wurde durch die Verwendung der Standard Template Library (STL) um Algorithmen zur Verwaltung von Datenstrukturen und Strings erweitert.“ [21, S. 3]

/i-TA20/* „Die grafische Darstellung wird mit Hilfe der OpenGL for Embedded Systems 3.1 (OpenGL ES 3.1)-Bibliotheken umgesetzt.“[21, S. 3]

Darüber hinaus hat Frau Klaus eine Liste wichtiger Anforderungen an die Engine erstellt, die auf dem Game Design des damaligen Spiels basierte [21, S. 34-35]. Die von ihr identifizierten Anforderungen sind noch immer aktuell, mit Ausnahme derer, die bereits umgesetzt wurden, wie beispielsweise das grundlegende Mastering von Audiodateien. Im Abschnitt zur Ressourcenverwaltung wird zudem die Notwendigkeit einer Datenbank erwähnt [21, S. 14], die der Autorin zuvor nicht bekannt war und deshalb bisher nicht im Lastenheft erwähnt wurde.

/RM10/* Die Game Engine umfasst eine Datenbank, welche alle Assets beinhaltet. Diese ist vom Entwickler editierbar.

Frau Klaus schrieb im Rahmen ihrer Arbeit die „main“-Klasse und implementierte den Game Loop, sowie die Game Time. Der Game Loop ist ein zentraler Bestandteil der Spieleentwicklung und ermöglicht durch regelmäßige Aktualisierungen das Darstellen des Spielgeschehens und die Berechnung der Spiellogik. Der Game Loop sorgt dafür, dass alle Spielobjekte in einem bestimmten Rhythmus aktualisiert werden und somit das Spielgeschehen vorangetrieben wird [26, S. 37]. Die Game Time ist ein Zeitmessinstrument, das unabhängig von der tatsächlich vergangenen Zeit die In-Game-Zeit misst und somit die Steuerung von zeitabhängigen Vorgängen im Spiel ermöglicht [27, S. 10]. Ein Beispiel dafür ist die Verwendung eines Cooldowns, der eine bestimmte Anzahl von Spielzeit-Einheiten benötigt, bevor er erneut aktiviert werden kann. Aus diesen Gründen wurden die folgenden Anforderungen in das Lastenheft aufgenommen.

/PE10/ Es gibt ein Debug-Log für Debugging- und Testingzwecke. Eingaben werden aufgezeichnet.

/PE20/ Die tatsächlich vergangene Zeit zwischen zwei festlegbaren Events kann gestoppt und ausgegeben werden..

/PE30/ Ein Designer kann, nach Einweisung, eigenständig Level in der Engine konzipieren.

Da diese Systeme keiner der bestehenden Kategorien im Lastenheft zuzuordnen waren, wurde ein neuer Abschnitt mit dem Titel „Game Logik“ hinzugefügt. Diese Erweiterung ermöglicht es, künftige Anforderungen nach dem Schema /GL10/ präzise diesem Abschnitt zuzuordnen.

/i-GL10/ Es existiert ein Game Loop in welchem zukünftig die Spiellogik läuft. Hier wird Input verarbeitet, Physik geupdated und Grafik gerendert.

/i-GL20/ Es existiert eine Game Clock, welche die In-Game-Zeit messen kann.

Die Entwicklung der BuggyTech Engine wurde während der Bachelor Arbeiten von Daniel Stockmann unterstützt. Er konzipierte die Grundlagen für das Grafiksystem der Engine, indem er den OpenGLRenderer und das SDLGraphicsSystem integrierte und die Klassen „Subsystem“ sowie „SubsystemStage“ verfasste, welche als Basis für alle zukünftigen Subsysteme dienen sollen [28].

/i-GR10/ Die Engine besitzt ein Grafiksubsystem.

Zusätzlich konnten folgende Produktfunktionen identifiziert werden:

/i-GR20/ Es wird ein OpenGL 4 Rendering Kontext erzeugt und die Grafikkarte wird von der Engine angesteuert.

/i-GR30/ Es lässt sich ein Fenster im Vollbild oder Fenstermodus öffnen. Ein Canvas kann erstellt und verwendet werden.

Die Befehle, um einzelne Sprites zu zeichnen, fehlen bisher. Diese Funktion wird jedoch in /GR30/* abgedeckt und muss noch implementiert werden.

/TA-20/* Zur Implementierung weiterer Subsysteme werden die Klassen *Subsystem* und *SubsystemStage* verwendet.

Im Rahmen seiner Bachelorarbeit konzipierte und implementierte Theo Stötzer erfolgreich zwei Subsysteme für die Engine, nämlich das Audio- und das Eventsystem. Des Weiteren entwickelte er einen vorläufigen Input Manager, um das Eventsystem zu testen. Das Audiosystem erfüllt bisher grundlegende Anforderungen. Es kann verschiedene Sounddateien gleichzeitig abspielen und deren Lautstärke individuell regeln.

/i-AU10/ Das Audio System kann mehrere Sounds gleichzeitig stereo abspielen.

/i-AU20/ Die Lautstärke einzelner Soundchannel ist editierbar (vom Entwickler, nicht vom Spieler).

Im Ausblick seiner Abschlussarbeit präsentiert Herr Stötzer mehrere ausbaufähige Aspekte des Systems [28, S. 62], von welchen die Autorin, basierend auf dem GDD, zwei für wichtig hält. Ein Spiel muss dem Spieler gewisse Einstellungen zum Anpassen seiner Spielerfahrung ermöglichen, darunter zählt das Anpassen der Lautstärke des Spieles. Es sollte dem Spieler innerhalb der Software möglich sein, die Master-Lautstärke zu steuern.

/AU30/ Die Gesamtlaustärke des Spiels kann durch den Spieler im Menü angepasst werden.

Wünschenswert wäre die Möglichkeit des Anpassens einzelner Channel. Die meisten Spiele ermöglichen es dem Spieler, verschiedene Soundkategorien einzeln zu steuern. Möchte er zum Beispiel die Musik des Spieles genießen, kann er Soundeffekte leiser und die Musik lauter stellen. Versteht er die Dialoge nicht, kann er diese explizit lauter und Musik und Effekte leiser regeln.

/AU40/ Der Nutzer hat die Möglichkeit, im Menü die Lautstärke der einzelnen Kanäle (z.B. Musik, Soundeffekte, Schüsse) anzupassen.

Ein weiterer Punkt Herrn Stötzers, welcher in das Lastenheft übernommen wird, ist das Normalisieren sich überlagernder Sounds [28, S. 62]. Nach dem derzeitigen Status des Audiosystems werden sich überlagernde Sounds lauter. Das GDD skizziert ein schnelles, unübersichtliches Spiel mit rasanten Schusswechseln. Schaut man sich Arcade Vorbilder des Scrolling Shooter Genres an, wird oft bis zu zehnmal in der Sekunde geschossen [29]. Spielt man für jeden Schuss die entsprechende Sound Datei, muss man sicher gehen, dass die Sounds weder gequeued noch lauter werden, wenn sie sich mit anderen überlagern.

/AU50/ Überlagernde Sounds werden durch Normalisierung angepasst.

Sowohl die Behandlung der Überlagerung von Sounds als auch die kanalbasierte Lautstärkeregelung durch den Nutzer sind für den Prototypen nicht unbedingt erforderlich. Jedoch ist die Normalisierung wichtiger und daher priorisiert zu behandeln. Was ebenfalls nicht hinreichend für den Prototypen ist, jedoch eventuell positiv zur Spielerfahrung beiträgt, ist die positionsbasierte Lautstärkeregelung einzelner Soundquellen. Findet eine Explosion also zum Beispiel nah am Spieleravatar statt, würde diese lauter abgespielt, als wenn sie weit entfernt ist. Ebenso würde der Spieler das Geräusch auf dem rechten Kopfhörer hören, wenn die Quelle sich rechts vom Spieler befindet und umgekehrt.

/AU60/ Es ist möglich, die Lautstärke von Sounds auf Basis der Distanz zur Quelle in Bezug auf den Spieler zu modifizieren.

/AU70/ Die Ausgabe der Geräusche erfolgt kanalbasiert, abhängig von der Position des Spielers in Bezug auf die Quelle, entweder im linken oder im rechten Audiokanal.

Die Eignung des Input Managers als Basis für diese Engine-Komponente muss bei der zukünftigen Arbeit am Subsystem untersucht werden. Falls die Grundlage von Herrn Stötzer geeignet ist, wäre eine mögliche zusätzliche Arbeit nicht erforderlich. Jedoch wäre es aus betriebswirtschaftlicher Sicht ratsam, vom schlechtesten Szenario auszugehen und anzunehmen, dass der Input Manager keine praktische Verwendung findet und vollständig neu geschrieben werden muss. Tastaturanschläge können jedoch bisher problemlos erkannt werden [28, S. 58].

/i-IM10/ Die Engine erkennt Tastenanschläge.

Das von Herrn Stötzer implementierte Eventsystem identifiziert Tastenanschläge als Event, und es muss noch evaluiert werden, ob es für die Verarbeitung komplexerer Vorgänge ausreichend ist. Bei Bedarf sind Änderungen vorbehalten.

In der bisherigen Version des Lastenheftes fehlten Anforderungen bezüglich des Eventsystems, da dieser technische Aspekt bei der reinen Analyse des GDD übersehen wurde. Die Autorin hat sich daher mit den eventsystembezogenen Abschnitten der Bachelorarbeit von Herrn Stötzer auseinandergesetzt und sich über die grundlegenden Funktionen von Eventsystemen in Game Engines informiert. Im GDD werden verschiedene Funktionen beschrieben, die als Events definiert werden können, sie werden jedoch auch teilweise impliziert, wie zum Beispiel das Erscheinen eines Bossgegners an bestimmten Stellen im Level. Andere Dinge, wie die Möglichkeit der Abfrage eines Events von verschiedenen Stellen im Code, werden im GDD überhaupt nicht erwähnt, da dies eine ausschließlich code-bezogene Anforderung ist.

/EV10/* Das Eventsystem kann Ereignisse wie Nutzereingaben, Kollisionen und Animationen handhaben, auslösen und verarbeiten.

/EV20/* Die Engine kann Events als Reaktion auf bestimmte Konditionen auslösen, beispielsweise das Aufeinandertreffen von zwei bestimmten Objekten oder das Beenden eines Levels.

/EV30/* Es existiert ein Event Listener. Events müssen von anderen Stellen in der Engine abgefragt und untereinander priorisiert werden können. Dies kann parallel verlaufen, d.h. ein Event kann von mehreren Stellen gleichzeitig abgefragt werden.

/EV40/* Der Entwickler hat die Möglichkeit Events zu definieren, hinzuzufügen und zu bearbeiten.

/EV50/ Es ist möglich, Events je nach Priorität in Warteschlangen zu sortieren und u.U. zu vernachlässigen, um einen Event-Stau zu vermeiden. Redundant gleiche Events müssen aggregiert werden.

Ein Achievementsystem und die Erstellung von Zählern werden weder im GDD spezifiziert, noch sind sie aus technischer Sicht notwendig. Dennoch sind Achievements in der Spielerfahrung ein beliebtes Element, das das Gameplay durch die Integration von Countern positiv beeinflussen kann. Dieses Beispiel zeigt, dass eine kreative Herangehensweise, welche nicht ausschließlich technische Aspekte berücksichtigt, sondern auch die Gestaltungsaspekte des Endproduktes im Auge behält, Vorteile hinsichtlich der sinnvollen Erweiterung des Lastenheftes haben kann.

/EV60/ Es können Zähler für bestimmte Events angelegt und deren Stand temporär oder permanent gespeichert werden. Dies soll ein Achievementsystem und Statistiken ermöglichen.

Sowohl der Input Manager als auch das Audiosystem wurden unter dem SDL-Framework implementiert. Auch Frau Klaus und Herr Stockmann nutzten das SDL Framework zur Implementation.

/i-TA30/ Für die Implementierung von Game Clock, Game Loop und Input Manager wurde das Simple DirectMedia Layer 2 genutzt [21, S. 41]. Für das Audiosystem wurde sich dem SDL_Mixer bedient [28, S. 29].

/i-TA40/ Eine SDL 2 Programmierschnittstelle wird für den Zugriff auf Eingabegeräte (Tastatur, Controller, Joystick, etc.), für Soundausgabe und -mastering, sowie das Rendern von true type fonts verwendet.

Daher wird empfohlen, bei der Entwicklung künftiger Versionen der Engine ebenfalls auf das SDL-Framework zurückzugreifen, um Inkompatibilitäten und Probleme bei der Einbindung neuer Subsysteme zu verhindern. Der weitere Einsatz von SDL wird als Anforderung in das Lastenheft aufgenommen:

/TA30/* Sofern möglich, sollten weitere Subsysteme, die Rendering, Audioausgabe, die Ereignis-Behandlung, die Thread- und Timer-Verwaltung sowie Nutzerinput betreffen oder eine Erweiterung bestehender Programmteile darstellen, unter Einsatz von folgenden Bibliotheken implementiert werden: Standard Template Library (STL), Simple DirectMedia Layer (SDL) und OpenGL for Embedded Systems 3.1 (OpenGL ES 3.1).

Da Frau Klaus und Herr Stötzer zum Kompilieren des Programmes die IDE „Visual Studio 2019 Community Edition“ verwendeten [21, S. 3] [28, S. 58], wird dies auch zukünftig empfohlen

/TA50/ Die IDE „Visual Studio 2019 Community Edition“ ist zum Kompilieren des Programms empfohlen, um Komplikationen zu vermeiden.

4.3.2 Bewertung der Auswirkungen der Änderungen seit Projektbeginn

Änderungen der Hardware Im Verlauf des Projekts hat sich das Ziel gewandelt, da anfangs neben der Entwicklung des Spiels, welches damals noch „Whisper Woods“ hieß, auch die Entwicklung und der Bau eines Arcade-Automaten geplant waren. Die dadurch gegebenen räumlichen und elektronischen Einschränkungen brachten spezielle Anforderungen an Hard- und Software mit sich. Christina Klaus betonte in diesem Zusammenhang, dass eine optimale Nutzung der vorhandenen Ressourcen notwendig sei, um eine gute Performance des Spiels zu gewährleisten [21, S. 32]. Nunmehr sollen die Engine und ihr Prototyp auf einem handelsüblichen Computer laufen. Der Wegfall der Einschränkungen durch die Hardware erleichtert die Softwareprogrammierung durch die erhöhte Leistung und Speicherkapazität. Moderne Computer sind in der Lage, eine große Anzahl niedrig aufgelöster Texturen flüssig darzustellen.

/TA40/* Die Software muss auf Windows 10 oder höher laufen.

Zu Beginn des Projektes sollte das Spiel noch auf dem ASUS Tinker Board laufen. Mit 2 GB RAM [30] hat der Einplatinencomputer weniger Arbeitsspeicher als die meisten Smartphones [31]. Diese Einschränkung erfordert ein intelligentes Ressourcenmanagement und das Laden von Assets zur Laufzeit. Ein Cache-System wäre hierbei von Nöten, um häufig und / oder kürzlich verwendete Ressourcen im Speicher zu halten und wiederholte Ladevorgänge zu vermeiden. Außerdem müssten nicht mehr benötigte Assets identifiziert werden, um den entsprechenden Speicherplatz freigeben zu können. Ein solches Ressourcenmanagement ist auch weiterhin von Vorteil, jedoch nur bei sehr inhaltslastigen oder umfangreichen Spielen oder solchen mit hochaufgelösten Texturen unbedingt notwendig, da gängige Computer über genügend Kapazität verfügen. Die niedrige Auflösung der Assets des geplanten Spiels aufgrund der vorgesehenen Fenstergröße von 640 x 480 Pixeln lässt jedoch viel Spielraum zu. Alle Assets sollten ohne merkbare Performance-Einbußen zu Beginn des Spiels geladen werden können, wodurch ein simples Cache-System ausreichend ist. Allerdings sind ein Cache-System und ein Garbage-System immer noch erforderlich, um eine effiziente Nutzung von Ressourcen zu gewährleisten.

/RM20/* Die Game Engine benötigt ein Cache System.

/RM30/* Die Game Engine verfügt über ein Garbage Collection System.

Die Assethierarchie ist speziell für den Einsatz in der BuggyTech Engine konzipiert und ihre Verwendung wird durch die folgende Produktfunktionen verlangt.

/RM40/* Um das effiziente Laden von Assets zu ermöglichen, arbeitet die Game Engine mit der in 3.3 einsehbaren Assethierarchie. Die Sortierung der Assets nach Szenen ermöglicht es, das Rendern von Game Assets und Objekten zu reduzieren und optimieren, indem nur die aktuell auf dem Bildschirm angezeigten Objekte gerendert werden. Zusätzlich liefert es dem Ressourcencache Informationen darüber, welche Assets in Zukunft benötigt werden, damit diese vorgeladen werden können.

/RM50/ Der Spieler kann seinen Spielstand speichern und zu einem späteren Zeitpunkt abrufen, um weiterzuspielen.

Die nachfolgenden Anforderungen sind nicht unbedingt erforderlich für die Engine. Bei der Untersuchung der Themen Ressourcenverwaltung und Engine-Entwicklung wurde jedoch festgestellt, dass die Implementierung von Funktionen zur Durchsuchung von Datenbanken den Entwicklungsprozess erleichtern kann. Die Integration der beiden darauf folgenden Funktionen wird empfohlen um die Spiel-Performance zu verbessern. Die Funktionen sind jedoch rein optional und nicht für den Prototypen erforderlich.

/RM60/ Die Datenbank kann nach bestimmten oder mehreren Assets durchsucht werden und hat ein grafisches Interface.

/RM70/ Ressourcen können zur Laufzeit bereitgestellt werden. Das Laden und Freigeben von Speicherplatz während der Laufzeit muss effizient genug sein, um den Spielfluss nicht zu beeinflussen und nicht für Frame Rate Drops zu sorgen.

/RM80/ Um ein effizientes Rendern von Assets zu gewährleisten, z.B. durch die Verwendung von Texture Atlanten oder Sprite Batching, kann die Game Engine mit diesen Formaten umgehen. Die Verwendung dieser richtet sich nach der eventuellen Notwendigkeit. Diese kann zum jetzigen Zeitpunkt nicht festgestellt werden. Sind sie nicht notwendig, ist die Anforderung hinfällig.

Ein Cache System und ein Garbage System sind nach wie vor nötig, jedoch ist ein so sparsames Ressourcenmanagement nicht mehr erforderlich. In diesem Kontext, bzw. bei der Recherche diesbezüglich, wurde außerdem die Notwendigkeit einer Datenbank und deren Funktionen klar. Dies geht aus dem GDD nicht hervor, weshalb es nun ergänzt wurde.

Änderungen am Game Design Seit der Konzeption der BuggyTech-Engine hat sich auch das geplante Spiel stark verändert. Anstatt eines Jump & Run- und Rätselplattformers soll nun ein Scrolling Shooter entwickelt werden, bei dem der Spieler sich frei auf dem Bildschirm bewegen kann. Im Vergleich zum ursprünglichen Spiel sind die Mechaniken simpler und weniger zahlreich. In „Whisper Woods“ spielte die Umgebung eine bedeutende Rolle, da zahlreiche Rätsel- und Interaktionsmöglichkeiten geplant waren. Beispielsweise sollte der spielbare Charakter durch Graben in der Erde neue Wege erschließen können, was zusätzliche Animationen und Assets erfordert hätte. Während im ursprünglich geplanten Spiel ein Großteil des Spielerlebnisses auf der Interaktion mit der Umgebung basierte, können in der aktuellen Version Umgebungselemente lediglich zwei Zustände haben - intakt oder zerstört -, abhängig von ihren Lebenspunkten. Die Möglichkeit, Areale zu gestalten, die

die Spielerbewegung beeinflussen, wird zwar in Betracht gezogen, ist aber bisher nicht geplant und auch nicht zwingend notwendig. Somit ist die Interaktion des Spielers mit der Umgebung nicht so komplex wie im ursprünglichen Konzept. Die Programmierung einer Engine bietet den Vorteil, dass grundlegende Funktionen flexibel angepasst werden können. Dementsprechend erfordern die oben genannten Änderungen keine Anpassung der Architektur der Game Engine oder Umstrukturierungen innerhalb der Engine.

5 Zusammenfassung und Ausblick

Nach der Entwurfsphase und Projektplanung in der Videospielentwicklung wird ein Prototyp erstellt, um die grundlegenden Mechaniken effektiv zu implementieren und zu testen. Nach der Überprüfung der Funktionalität und des Spielerlebnisses werden weitere Tests durchgeführt, einschließlich Stress- oder Lastentests, um sicherzustellen, dass das Programm auch in Ausnahmesituationen und bei Überlastung eine stabile Leistung aufweist. Im Rahmen der Entwicklung eines Videospiels treffen in der Regel Game Designer gemeinsam mit Programmierern und gegebenenfalls Stakeholdern Entscheidungen darüber, welche Mechaniken prototypisiert werden und in welcher Reihenfolge. Hierbei orientieren sie sich in der Regel an der Gewichtung der einzelnen Komponenten in Bezug auf die Essenz des Spiels. Die Autorin musste das Lastenheft nachträglich bearbeiten und umstrukturieren, um eine Sortierung der Anforderungen nach Relevanz umzusetzen. Um die wichtigsten Punkte des Game Design Documents zu identifizieren, musste dieses mehrmals überarbeitet werden, was viel Zeit kostete. Eine Sortierung des GDDs nach der Wichtigkeit der einzelnen Mechaniken und Aspekte des Spiels während der Erstellung würde es jedem Leser ermöglichen, die Schwerpunkte auf einen Blick zu erkennen. Dies könnte zudem das Prototyping erleichtern und Zeit sparen. Um die Planung eines Prototyps zu beschleunigen, kann es hilfreich sein, bereits bei der Erstellung des Design-Dokuments auf die Wichtigkeit der einzelnen Mechaniken zu achten. Game Designer haben vor und während der Erstellung eines solchen Dokuments bereits eine sehr klare Vision ihres Ziels und wissen, wo die Schwerpunkte und Alleinstellungsmerkmale des Spiels liegen, welche Aspekte von besonderer Bedeutung sind und welche lediglich Bonus-Features darstellen. Es wäre sinnvoll, einen dokumentübergreifenden Schlüssel zur Markierung und Strukturierung relevanter Inhalte zu verwenden, ähnlich wie beim Lastenheft für die Game Engine. Das geordnete Vorgehen, wie es beim Lastenheft üblich ist, könnte, durch die Anwendung im Game Design Document, auch die Entwicklung von Spielen positiv beeinflussen.

Die vorliegende Arbeit untersucht, ob ein Game Design Document (GDD) als Grundlage für die Erstellung eines Lastenheftes geeignet ist. Die Verwendung eines GDDs hat sich als äußerst hilfreicher Ausgangspunkt erwiesen, um eine Anforderungsanalyse für eine Game Engine durchzuführen. Die konkrete und vollständige Formulierung eines GDDs erleichtert den Überblick über die zu implementierenden Mechaniken und gibt dem Leser eine klare Vorstellung vom Gesamtziel des Spiels. Ein Lastenheft beschreibt das „Was und Warum“ eines Projekts und hat ähnliche Fragen wie ein GDD im Fokus, jedoch aus einer anderen Perspektive. Ein Designer erstellt ein GDD, um die Quintessenz und Feinheiten des geplanten Spiels festzuhalten und legt den Schwerpunkt auf Vollständigkeit und Schwerpunktvermittlung. Ein Spiel wird in der Regel auf einer bestehenden Engine entwickelt, die Auswahl und Arbeit damit fällt nicht in den Zuständigkeitsbereich eines Game Designers.

Die Autorin des Game Design Documents hat als Designerin des Spiels den Fokus auf die Gestaltung der Spielmechaniken gelegt und daher keine oder nur sehr wenige technische Details im GDD aufgenommen, da Implementierungsdetails zum Zeitpunkt der Erstellung des GDDs nicht berücksichtigt wurden. Dies führte bei der Konvertierung in das Lastenheft dazu, dass wichtige technische Informationen fehlten. Während beispielsweise das Layout des Interfaces und die Auflösung des Spieles durchaus behandelt werden können, ist es äußerst unüblich, Pixelpositionen von Assets in einem GDD zu definieren.

Die Bachelorarbeiten, welche von der Autorin in diesem Kontext studiert wurden, lieferten zwar eine solide Basis, jedoch war dennoch eine intensive Recherche über Programmstrukturen, Enginearchitektur und Softwareentwicklung notwendig, um die technischen Defizite, die aufgrund der designorientierten Natur des GDDs bestehen, adäquat auszugleichen. Zusammenfassend kann gesagt werden, dass ein GDD allein ausreichen *kann*, um darauf basierend ein Lastenheft für eine Engine zu entwickeln. Dies kann jedoch nur gelingen, wenn ausreichende technische Kenntnisse über Programmierung und Engineentwicklung vorhanden sind. Ein grundlegendes Verständnis dieser Aspekte ist elementar und sollte ausgebaut werden, um das Fehlen von technischen Konkretisierungen im GDD auszugleichen. Sollte dies nicht gewährleistet sein, sollte ein Lastenheft niemals ausschließlich aus dem GDD erstellt werden. Eine Game Engine ohne spezifische Vorstellungen über das darauf laufende Spiel zu entwerfen, würde höchstwahrscheinlich nicht erfolgreich sein. Eine konkrete Vorstellung des Zieles erleichtert die Erstellung eines Lastenhefts erheblich. Das GDD bietet daher dennoch eine hervorragende Grundlage für die Entwicklung des Lastenheftes, da es grundlegende Informationen über das zu entwickelnde Spiel enthält, welche für die Entwicklung einer Engine von elementarer Bedeutung sind.

Während der Bearbeitung des Dokuments vollzog die Autorin einen Perspektivwechsel. Zuvor hatte sie im Verlauf ihres Studiums GDDs lediglich hinsichtlich ihrer designbezogenen Vollständigkeit verfasst. Durch die Anforderungsanalyse erlangte sie jedoch ein besseres Verständnis für andere Aspekte und widmete der technischen Umsetzung ihrer Ideen mehr Aufmerksamkeit. Sie erkannte, dass eine ganzheitliche Herangehensweise bei der Erstellung sowohl eines Lastenheftes als auch eines Spiels von Vorteil ist, da sich die Vorteile von design- und programmierorientierten Perspektiven ergänzen können.

HOCHSCHULE MITTWEIDA
University of Applied Sciences

GAME DESIGN DOCUMENT

Neon Nova

Frau
Elly Müller

Mittweida, April 2023

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Eckdaten/Basics

1.1 Zielgruppe

Neon Nova soll sowohl der Generation, die derlei Spiele noch selbst am Automaten in der Spielhalle erleben durfte ein nostalgisches Erlebnis darbieten, als auch junge Spieler an das Genre heranführen, indem es ihnen bekannte Elemente im Spiel verbaut. Fans von schnellen, effektgeladenen Actionabenteuern, die ohne viel Geschichte direkt zur Sache kommen wollen, sollen mit Neon Nova auf ihre Kosten kommen.

1.2 Gameplay

Fliegen, Schießen, Ausweichen, wiederholen. Neon Nova ist ein fast paced Scrolling Shooter, der Konzentration mit ansprechenden, bunt leuchtenden Effekten belohnt.

Der Spieler steuert sein Raumschiff über unbekannte Planeten, durch Meteoritenschauer und feindliche Raumstationen.

Das Spiel ist fast paced und lässt nicht viel Zeit zum Nachdenken. Erfolg lässt sich nur erreichen durch schnelles Auffassungsvermögen und Reaktion, sowie durch das Erlernen von Angriffsmustern der verschiedenen Gegnertypen.

1.3 Steuerung

- WASD oder Pfeiltasten einer Tastatur
- Gamepad
- 8-Wege-Joystick

Der Spieler kann sich in zwei Dimensionen bewegen, also horizontal und vertikal auf dem Bildschirm.

Das Ausweichmanöver ermöglicht dem Raumschiff sich unter einem Angriff hin wegzuducken, also theoretisch die Bewegung in die dritte Dimension. Diese ist jedoch nur von kurzer Dauer und in dieser lässt sich die Spielfigur nicht vom Spieler steuern. Der Spieler spielt entweder mit Controller oder mit Tastatur und Maus. Befindet er sich im Spiel lässt sich das Eingabemedium nicht wechseln. Wird ein Eingabemedium während des Spieles getrennt, pausiert das Spiel. Ohne Eingabemedium lässt es sich nicht fortsetzen. Drückt der Spieler, im Pause Menü oder im Startbildschirm, eine Taste auf der Tastatur erscheint ein Tastatur Icon und die weitere Eingabe erfolgt über die Tastatur. Dies gilt gleichermaßen für den Controller.

Der Spieler kann in seinen Einstellungen die Tastenbelegung nach Belieben ändern. Eine Option setzt diese Belegung auf die ursprüngliche zurück.

Bei der Steuerung mit dem Gamepad korrespondiert die Betätigung von Trigger und Joystick direkt mit der Bewegung des Spieleravatars. Wird der Joystick so bewegt, dass sein Achsenwert 1 beträgt, also vollständig betätigt wird, bewegt sich das Raumschiff schneller in die Richtung, als wenn der Joystick nur sanft in diese Richtung geschoben wird. Ist der Trigger nur leicht betätigt, schießt das Raumschiff langsamer, als wenn die Taste komplett durchgedrückt wird.

Dementsprechend bewegt sich, bei der Steuerung mit Tastatur, das Raumschiff schneller, wenn die entsprechende Taste gedrückt gehalten wird, als wenn sie nur kurz oder einige Male angetippt wird.

1.4 Raumschiffe und Spezialfähigkeiten

Man hat vor Spielbeginn die Wahl zwischen drei verschiedenen spielbaren Raumschiffen (diese Zahl kann bei Bedarf noch erhöht werden). Das Raumschiff, mitsamt seiner Fähigkeiten, für das man sich entscheidet, spielt man dann für den gesamten Verlauf des Spieles. Dies soll den Wiederspielwert erhöhen, da der Spieler im besten Fall mit allen Raumschiffen die Reise bestreiten möchte. Was die Steuerung und die Basisfähigkeiten und -werte angeht, sind alle Raumschiffe gleich. Die verschiedenen Raumschiffe haben unterschiedliche Formen und auch die Farbe und das Aussehen ihres Standard Schusses unterscheidet sich.

Der wichtigere Unterschied sind jedoch die Spezialfähigkeiten. Jedes Raumschiff besitzt jeweils eine defensive und aggressive Spezialfähigkeit. Diese ermöglichen es dem Spieler in bestimmten Abschnitten Wege zu beschreiten, welche den anderen Raumschiffen verwehrt bleiben. Zudem werden dadurch Spielstile und andere Herangehensweisen an (Boss-) Kämpfe ermöglicht, welche wiederum den Spielfluss individualisieren.

Die Stärke/Weite/Dauer/Fächerung der Spezialfähigkeit ist abhängig vom Zustand des Ladebalkens/der Anzahl der Kombis/Multiplikatoren. Je leerer, desto schwächer die Attacke bzw. geringer die Reichweite/Zeit/etc. Ein voll geladener Balken/... erzeugt die stärkste Attacke/längste Effektdauer/...

1.4.1 Alle Raumschiffe

Jedes Raumschiff hat jeweils eine aggressive und eine defensive Basisfähigkeit. Diese unterscheiden sich nicht unter den verschiedenen Typen.

Defensive Fähigkeit “Ausweichen”

Das Raumschiff duckt sich in Bewegungsrichtung weg. Dies macht gegnerische Angriffe wie Schüsse für kurze Dauer (X) wirkungslos, schützt den Spieler jedoch nicht vor Schaden welcher durch Schadenszonen und Todeszonen verursacht wird.

Aggressive Fähigkeit “Schuss”

Der simple Angriff ist Schießen. Ein einzelner Schuss macht X Schaden.

Zusätzlich zu den Basisfähigkeiten besitzt jedes Raumschiff jeweils zwei Spezialfähigkeiten:

1.4.2 Raumschiff-1

Defensive Fähigkeit “Decoy”

Das Raumschiff erzeugt eine zweite Hologrammversion von sich selbst, welche nach einiger Zeit mit einer Explosion, welche Gegner in der Nähe verletzt, zerstört wird. Das Decoy schießt geradeaus, bewegt sich aber nicht. Der Spieler wird für den Gegner unsichtbar und das gegnerische Feuer zielt auf die Kopie des Spielers.

Aggressive Fähigkeit “Raketen”

Das Raumschiff verschießt eine bestimmte Anzahl Raketen, welche selbstständig nach Zielen suchen und Gegner schädigen. Dies ist abhängig von der Stärke der Gegner. Der stärkste Gegner wird mit der ersten Rakete angegriffen, der zweitstärkste mit der zweiten Rakete, usw..

1.4.3 Raumschiff-2

Defensive Fähigkeit “Schutzschild”

Das Raumschiff erzeugt für eine bestimmte Zeit eine schützende Hülle um sich selbst. Gegnerische Projektile prallen vom Spieler ab, werden zurückgeworfen und können Gegner treffen. Das Schutzschild hält X Schaden ab/hält X

Aggressive Fähigkeit “Chainshot”

Der Chainshot springt vom ersten getroffenen Gegner zum nächsten (am wenigsten weit entfernt) und macht dabei bei allen gleich viel prozentuellen/numerischen Schaden. Oder Der Chainshot springt vom ersten getroffenen Gegner zum stärksten auf dem Bildschirm, dann zum zweitstärksten, usw. ODER umgedreht (um viele kleine Gegner schnell auf einmal zu zerstören.

1.4.4 Raumschiff-3

Defensive Fähigkeit “Teleport”

Der Teleport ermöglicht es dem Spieler, das Raumschiff für sehr kurze Zeit unsichtbar und untreffbar zu machen. Während dieser kurzen Zeit kann der Spieler mit einem Sucher einen anderen Punkt auf dem Bildschirm ansteuern. Bestätigt er diese Position mit einem Knopfdruck, erscheint das Raumschiff-3 an dieser Position, wird also teleportiert.

Der Teleport ermöglicht es dem Spieler das Raumschiff eine bestimmte Entfernung in die jeweils angesteuerte Bewegungsrichtung zu bewegen und alles auf dem Weg dorthin zu ignorieren.

Aggressive Fähigkeit “Black Hole Instance”

Der Spieler wählt mit dem Fadenkreuz eine Stelle auf dem Bildschirm, an welcher er kurz darauf ein schwarzes Loch erzeugt. Dieses saugt kleine Gegner und Gegnerprojektile komplett ein, große Gegner werden in die Richtung gesaugt und erleiden Schaden.

2 Interface

Dauerhaft auf dem HUD zu sehen sind:

- Leben Spieler
- Spezialfähigkeiten Spieler (Ladestand)

Zeitweilig auf dem HUD zu sehen sind:

- Leben Bossgegner
- Spezialfähigkeiten Spieler (eingesetzt)

- 640*480 Bildschirmgröße

3 Spielablauf

- Level auswählen
- Level durchspielen
- (Storyabschnitt - nicht geplant aber machbar)
- Scoreboard anzeigen (gesammelte Punkte, erledigte Gegner, gefundene Collectibles, ...)
- Raumschiff modifizieren (ästhetisch verändern)
- Level auswählen (durchgespielte Level können im Menü ausgewählt und erneut gespielt werden)

4 Level

Alle Level können in folgende Kategorien eingeordnet werden:

Künstliche Level Der Hintergrund wird dominiert von durch Menschen- bzw. Alienhand geschaffenen Strukturen. Z.B.:

- Raumstationen
- Gebäude
- Städte

Organische Level Der Hintergrund wird dominiert von außerirdischen Pflanzen, Pilzen und natürlichen Gewässern. Z.B.:

- Wälder
- Flüsse
- Wiesen

Natürliche Level Der Hintergrund wird weder von organischen, noch künstlichen Bestandteilen dominiert. Die gezeigte Welt ist natürlich geformt, aber unbelebt.

- Planetenoberflächen
- Asteroidenfelder
- Höhlen

Hybriden Der Hintergrund vereint die obigen Levelformen in sich. Z.B.:
- Überwucherte
- Zerstörte Städte
- Verlassene Raumstationen

5 Gegner

Die hier beschriebenen Gegner existieren in jeweils drei verschiedenen Variationen. Entsprechend der im Abschnitt Level beschriebenen Umgebungen muss für jeden Gegnertyp jeweils eine thematisch und stilistisch zum Level passende Variante existieren. Ideen am Beispiel der Schadenszone: Künstlich: freiliegende Kabel, welche Funken versprühen Organisch: dornenbesetzte Ranken Natürlich: Weltraumschrott

5.1 Hindernisse

- Sind ohne Fremdeinwirkung unbeweglich und schießen nicht auf den Spieler.
- Kleine, schwebende (also nicht fest stehende oder verankerte) Hindernisse können von schwarzen Löchern beeinflusst werden.
- Können durch Schüsse zerstört werden.
- Fügen dem Spieler bei Berührung X Schaden zu.
- Fügen dem Spieler bei Berührung X Schaden zu.
- Haben x Leben.

5.2 Schadenszonen

- Fügen dem Spieler eine bestimmte Menge/Prozentzahl Schaden zu.
- Werden von schwarzen Löchern nicht beeinflusst.
- Manche haben unendliches/unzerstörbares Leben und können nicht zerstört werden.
- Andere haben X Leben und können durch Schüsse zerstört werden.

5.3 Todeszonen

- Der Spieler stirbt bei Berührung./Fügen dem Spieler Schaden in Höhe seiner Lebenspunkte zu
- Werden von schwarzen Löchern nicht beeinflusst.
- Können nicht durch zerstört werden.
- Haben unendliches/unzerstörbares Leben.

5.4 Turrets

- Sind unbeweglich und schießen auf den Spieler.
- Werden von schwarzen Löchern nicht beeinflusst.
- Können durch Schüsse zerstört werden.
- Haben X Leben.

5.5 Kamikaze

- Können nicht schießen.
- Fliegen gezielt auf den Spieler zu.
- Sobald sie ihn berühren, explodieren sie und er erleidet X Schaden.
- Können durch Schüsse zerstört werden.
- Haben X Leben.

5.6 Drohnen

- Können schießen, machen aber nicht viel Schaden.
- Ein Schuss der Drohnen macht X Schaden.
- Können durch Schüsse zerstört werden.
- Haben X Leben.

5.7 Gegnerschiffe-Advanced

- Können schießen und machen genug Schaden, dass der Spieler versuchen muss den Schüssen auszuweichen.
- Gegnerschiffe-Advanced haben zwei verschiedene Schussmuster. Ein normaler Schuss der Gegnerschiffe-Advanced macht X Schaden. Ein Zweitschuss der Gegnerschiffe-Advanced macht X Schaden.
- Gegnerschiffe-Advanced sind in der Lage den Schüssen des Spielers in einem gewissen Maß auszuweichen.
- Können durch Schüsse zerstört werden.
- Haben X Leben.

5.8 Gegnerschiffe-Gepanzert

- Können schießen und machen genug Schaden, dass der Spieler versuchen muss, den Schüssen auszuweichen.
- Ein Schuss der Gegnerschiffe-gepanzert macht X Schaden.
- Sie haben eine Panzerung/ein Schutzschild um sich herum. Die Schüsse des Spielers können diese/n nicht durchdringen. Die Panzerung/Das Schutzschild hat eine Schwachstelle. Diese hat X Leben. Erst wenn die Schwachstelle beschossen wurde fällt die Panzerung/das Schutzschild und der Spieler kann mit seinen Schüssen das Gegnerschiff-gepanzert zerstören.
- Das Gegnerschiff-gepanzert wird, technisch gesehen, also was Stats und Daten angeht, zu einem Gegnerschiff-Advanced oder Gegnerschiff, wenn der Panzer zerstört ist.

- Hat X Leben.

5.9 Boss

- Können schießen und machen genug Schaden, dass der Spieler versuchen muss, den Schüssen auszuweichen.
- Ein Boss hat, im Gegensatz zu Gegnerschiffen, welche jeweils nur ein Schussmuster haben, die Möglichkeit mehrere verschiedene Schussmuster, sowie bosspezifische Schuss- und Angriffsmuster zu nutzen.
- Jeder Boss hat eine Panzerung/einen Schutzschild. Diese/r kann sich auch nach einer Phase erneuern und muss dann erneut zerstört werden.
- Je nach Phase kann ein Boss verschiedene Schussmuster anwenden. Er hat einen normalen Schuss, welcher X Schaden macht, einen Zweitschuss, welcher X Schaden macht, und einen Drittschuss, welcher X Schaden macht.
- Hat X Leben.

6 Spieler/Gegner Interaktion

Abbildung 6.1 stellt eine Kollisionstabelle dar. Diese ist von oben nach unten, also senkrecht zu lesen. Z.B.: trifft ein Spieler (2. Spalte) auf eine Gegnerbullet (Zeile 19), erleidet er (der Spieler) 1 Schadenspunkt. Die Werte sollen lediglich ein ungefähres Verhältnis der Stärken einzelner Objekttypen darstellen und stellen keine Endgültigkeit dar.

Abbildung 6.1: Die Assethierarchie.

7 Collectibles

Der Spieler kann während seines Playthroughs verschiedene Collectibles sammeln.

7.1 Raumschiffteile/Upgrades

Die verschiedenen Teile des Raumschiffes können ausgetauscht werden, die Änderungen sind rein ästhetisch. Die Raumschiffteile können im Verlauf des Spieles gesammelt werden. Vor Start eines neuen Levels hat der Spieler die Möglichkeit die Teile seines Raumschiffes neu auszuwählen. Sammelt der Spieler ein neues Teil ein, pausiert das Spiel und ihm wird die Option gegeben, das gesammelte Stück direkt auszurüsten. Es ist nicht möglich, in einem Playthrough alle Raumschiffteile zu erhalten. Einmal gesammelte Teile bleiben für sämtliche weitere Playthroughs erhalten, sofern der Spieler nicht den kompletten Spielstand löscht, und lassen sich auch bei allen Raumschifftypen ausrüsten, unabhängig davon, in welchem Playthrough das Teil gefunden wurde.

Ein Raumschiff besteht aus 5 verschiedenen Bestandteilen/Bauteilen, welche unabhängig voneinander ausgetauscht werden können (siehe Collectibles):

- Munition
  Aussehen des abgegebenen Schusses
- Körper
  Aussehen des mittleren Teiles des Raumschiffes
- Flügel
  Aussehen der Flügel des Raumschiffes
- Antrieb
  Aussehen der Hinterseite des Raumschiffes
- Kraftstoff
  Aussehen des, von den Triebwerken ausgestoßenen, Partikelschwarmes

7.2 Puzzle

Dieses Feature stell bisher lediglich eine Idee dar. Es muss nicht umgesetzt werden sondern wird nur der Vollständigkeit halber/ als Idee für die Zukunft hier festgehalten.

Der Spieler kann während seines Playthroughs X Puzzleteile sammeln, welche sich nach und nach zu dem Bild eines Raumschiffes zusammenfügen lassen, welches man sich außerhalb des aktiven Spieles anschauen kann. Hat man alle Puzzleteile gesammelt, auf denen eines der 4 Bauteile komplett zu sehen ist, erhält man dieses Bestandteil und kann es, ebenso wie die in den Levels findbaren Bauteile, an seinem Raumschiff ausrüsten.

8 Stil und Ästhetik

Die verpixelte Arcadewelt soll dynamisch beleuchtet werden, was durch die Buggytech Engine erreicht werden soll. Neon Nova soll vollgestopft werden mit leuchtenden, reflektierenden und funkelnden Oberflächen um der Engine möglichst viele Anwendungsmöglichkeiten für die dynamische Beleuchtung zu bieten. Explosionen, leuchtende Geschosse und Partikelleuchten sollen maximal zur Geltung kommen, um die Welt zu beleben. Künstliche Level sollen klinisch, futuristisch und dystopisch-kalt anmuten, natürliche Level bekannt und doch tot und lebensfremd. Organische Level

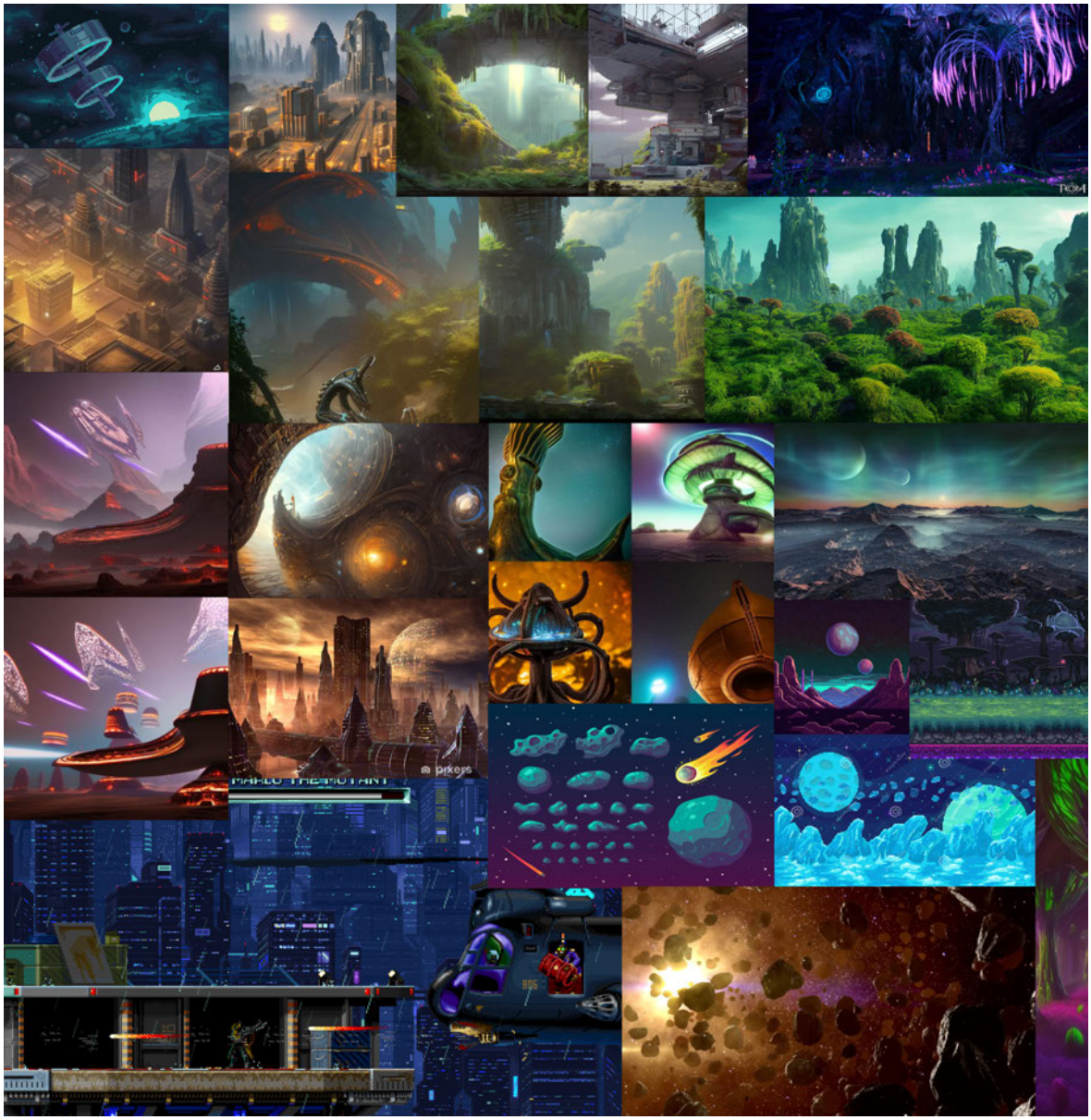

Abbildung 8.1: Moodboard

und deren Hybride bieten jeder Menge außerirdischer Lebensformen und deren Auswüchsen Platz, um so den Spieler zu verzaubern und in fremde Welten zu entführen. Dennoch ist Der Hintergrund nicht das Hauptaugenmerk. Das liegt auf spektakulären Explosionen und jeder Menge Lichteffekten, welche die dynamische Beleuchtung zur Schau stellen sollen.

Für Darstellungen des angestrebten Stiles siehe Abbildung 8.1 Die Bilder des Moodboards wurden mit der "https://creator.nightcafe.studio/ AI erstellt.

Die copyrightfreie Schriftart „pixel-love-font“ wird für In-Game Texte verwendet. Titel- und/ oder Todesbildschirm sowie andere Anzeigen können davon ausgenommen werden.

9 Potenzielle Erweiterungen

Das Spiel soll eine modulare Erstellung von Leveln erlauben, demnach lassen sich diese in beliebiger Menge, Größe und Vielfalt produzieren. Die Anzahl von Raumschiffen und Gegnern lässt sich ebenfalls jederzeit ändern und erweitern, ebenso ihr Aussehen, ihre Varianten und sammelbare Raumschiffteile.

HOCHSCHULE MITTWEIDA
University of Applied Sciences

LASTENHEFT

Anforderungsanalyse für die BuggyTech Engine

Frau
Elly Müller

Mittweida, April 2023

Inhaltsverzeichnis

Abbildungs- und Tabellenverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung/allg. Ziel

1.1 Zielsetzung der Software

Basierend auf dem Lastenheft soll eine Game Engine entwickelt werden, die in der Lage ist, einen im GDD genauer beschriebenen Scrolling Shooter auszuführen. Hierfür soll zunächst ein Prototyp erstellt werden, der die Nutzbarkeit der wichtigsten Mechaniken gewährleistet. Sobald der Prototyp fertiggestellt und funktionsfähig ist, sollen die übrigen Anforderungen implementiert werden. Die endgültige Engine soll flüssig laufen und in der Lage sein, alle im GDD beschriebenen Mechaniken zu implementieren.

1.2 Projektumfeld/Organisation Lastenheft

Das Lastenheft dient als Grundlage für die schrittweise Entwicklung der Engine, wobei die einzelnen Anforderungen nach ihrer Relevanz sortiert sind. Die minimal notwendigen Anforderungen sind mit einem Sternchen gekennzeichnet und dienen als Basis für die Entwicklung des Prototyps, der grundlegende Mechaniken und Funktionen testet. Die beschriebenen Voraussetzungen sind ausreichend für das Testen des rudimentären Gameplays. Im nächsten Schritt werden alle Anforderungen hinzugefügt, die den Prototypen in ein gut spielbares Spiel verwandeln und alle geplanten Mechaniken enthalten, um den Zielvorgaben zu entsprechen. Der letzte Schritt ist der Feinschliff, bei dem Features hinzugefügt werden, die die Nutzererfahrung verbessern und die Bedienung erleichtern. Formulierungen im Kapitel „Soll-Situation“, die besagen „Die Engine kann ...“ sind als Bedingung zu verstehen, während die Formulierung „Die Engine muss ... können“ aus Gründen der Lesbarkeit vermieden wurde. Die Kapitel „Grafik“ und „User Interface“ sind jeweils in „Allgemeine Anforderungen“ und „Spezielle Vorgaben“ unterteilt. Das Kapitel KI enthält neben allgemeinen Anforderungen „Höhere Taktik“. Dabei ist zu beachten, dass die Relevanz der einzelnen Anforderungen unabhängig von denen im jeweils anderen Abschnitt betrachtet wird. Vielmehr gilt es lediglich die Reihenfolge innerhalb des jeweiligen Abschnittes zu berücksichtigen.

Die wesentlichen Bedingungen, die für den Prototypen erforderlich sind, um die Grundlagen der Engine zu testen, sind mit einem Sternchen gekennzeichnet. Dies sollte ausreichen, um rudimentäres Gameplay zu testen und sicherzustellen, dass Kernmechaniken und -funktionen vorhanden und funktionsfähig sind. Alle weiteren Anforderungen sind in absteigender Reihenfolge von notwendig über bedingt bis hin zu optional weich gestaffelt. Anforderungen, die direkt nach den Prototyp-Anforderungen aufgeführt sind, sind wesentlich für das fertige Spiel, aber bedingt für den Prototypen. Die Entscheidung über die Wesentlichkeit oder Optionalität der einzelnen Anforderungen obliegt den zukünftigen Entwicklern.

Das Lastenheft ist nach folgender Sortierung strukturiert und die einzelnen Produktfunktionen sind entsprechend dem folgenden Schema benannt:

Die Anforderungen können bei Bedarf ergänzt werden, ohne die Sortierung zu beeinträchtigen.

Bereich	Anforderungscode
Technische Anforderungen	/TA10/
Game Logik	/GL10/
Ressourcenmanagement	/RM10/
Profiling/Entwicklertools	/PE10/
Physik und Kollision	/PK10/
Eventsystem	/EV10/
KI	/KI10/
Audiosystem	/AU10/
Grafiksystem	/GR10/
User Interface	/UI10/
Input Management	/IM10/
Ist-Situation, erledigte Anforderung	/i-.. ..10/
für Prototyp notwendig	/.. ..10/*

Tabelle 1.1: Kombinierte Tabelle der Anforderungscodes nach Bereichen

2 Ist-Situation

2.1 Spielmechaniken

2.1.1 Technische Anforderungen

/i-TA10/* „Der Funktionsumfang von C++11 wurde durch die Verwendung der Standard Template Library (STL) um Algorithmen zur Verwaltung von Datenstrukturen und Strings erweitert.“

/i-TA20/* „Die grafische Darstellung wird mit Hilfe der OpenGL for Embedded Systems 3.1 (OpenGL ES 3.1)-Bibliotheken umgesetzt.“

/i-TA30/ Für die Implementierung von Game Clock, Game Loop und Input Manager wurde das Simple DirectMedia Layer 2 genutzt. Für das Audiosystem wurde sich dem SDL_Mixer bedient.

/i-TA40/ Eine SDL 2 Programmierschnittstelle wird für den Zugriff auf Eingabegeräte (Tastatur, Controller, Joystick, etc.), für Soundausgabe und -mastering, sowie das Rendern von true type fonts verwendet.

2.1.2 Game Logik

/i-GL10/ Es existiert ein Game Loop in welchem zukünftig die Spiellogik läuft. Hier wird Input verarbeitet, Physik geupdated und Grafik gerendert.

/i-GL20/ Es existiert eine Game Clock, welche die In-Game-Zeit messen kann.

2.2 Medienausgabe

2.2.1 Grafik

/i-GR10/ Die Engine besitzt ein Grafiksubsystem.

/i-GR20/ Es wird ein OpenGL 4 Rendering Kontext erzeugt und die Grafikkarte wird von der Engine angesteuert.

/i-GR30/ Es lässt sich ein Fenster im Vollbild oder Fenstermodus öffnen. Ein Canvas kann erstellt und verwendet werden.

2.2.2 Audio

/i-AU10/ Das Audio System kann mehrere Sounds gleichzeitig stereo abspielen.

/i-AU20/ Die Lautstärke einzelner Soundchannel ist editierbar (vom Entwickler, nicht vom Spieler).

2.3 Nutzerschnittstelle

2.3.1 Input Management

/i-IM10/ Die Engine erkennt Tastenanschläge.

3 Soll-Situation

3.1 Spielmechaniken

3.1.1 Technische Anforderungen

/TA10/* Code ist in C++ zu schreiben.

/TA-20/* Zur Implementierung weiterer Subsysteme werden die Klassen *Subsystem* und *SubsystemStage* verwendet.

/TA30/* Sofern möglich, sollten weitere Subsysteme, die Rendering, Audioausgabe, die Ereignis-Behandlung, die Thread- und Timer-Verwaltung sowie Nutzerinput betreffen oder eine Erweiterung bestehender Programmteile darstellen, unter Einsatz von folgenden Bibliotheken implementiert werden: Standard Template Library (STL), Simple DirectMedia Layer (SDL) und OpenGL for Embedded Systems 3.1 (OpenGL ES 3.1).

/TA40/* Die Software muss auf Windows 10 oder höher laufen.

/TA50/ Die IDE „Visual Studio 2019 Community Edition“ ist zum Kompilieren des Programms empfohlen, um Komplikationen zu vermeiden.

3.1.2 Ressourcenmanagement

/RM10/* Die Game Engine umfasst eine Datenbank, welche alle Assets beinhaltet. Diese ist vom Entwickler editierbar.

/RM20/* Die Game Engine benötigt ein Cache System.

/RM30/* Die Game Engine verfügt über ein Garbage Collection System.

/RM40/* Um das effiziente Laden von Assets zu ermöglichen, arbeitet die Game Engine mit der im Lastenheft einsehbaren Assethierarchie. Die Sortierung der Assets nach Szenen ermöglicht es, das Rendern von Game Assets und Objekten zu reduzieren und optimieren, indem nur die aktuell auf dem Bildschirm angezeigten Objekte gerendert werden. Zusätzlich liefert es dem Ressourcencache Informationen darüber, welche Assets in Zukunft benötigt werden, damit diese vorgeladen werden können.

/RM50/ Der Spieler kann seinen Spielstand speichern und zu einem späteren Zeitpunkt abrufen, um weiterzuspielen.

/RM60/ Die Datenbank kann nach bestimmten oder mehreren Assets durchsucht werden und hat ein grafisches Interface.

/RM70/ Ressourcen können zur Laufzeit bereitgestellt werden. Das Laden und Freigeben von Speicherplatz während der Laufzeit muss effizient genug sein, um den Spielfluss nicht zu beeinflussen und nicht für Frame Rate Drops zu sorgen.

/RM80/ Um ein effizientes Rendern von Assets zu gewährleisten, z.B. durch die Verwendung von Texture Atlanten oder Sprite Batching, kann die Game Engine mit diesen Formaten umgehen. Die Verwendung dieser richtet sich nach der eventuellen Notwendigkeit. Diese kann zum jetzigen Zeitpunkt nicht festgestellt werden. Sind sie nicht notwendig, ist die Anforderung hinfällig.

3.1.3 Profiling/Entwicklertools

/PE10/ Es gibt ein Debug-Log für Debugging- und Testingzwecke. Eingaben werden aufgezeichnet.

/PE20/ Die tatsächlich vergangene Zeit zwischen zwei festlegbaren Events kann gestoppt und ausgegeben werden.

/PE30/ Ein Designer kann, nach Einweisung, eigenständig Level in der Engine konzipieren.

3.1.4 Physik und Kollision

/PK10/* Game Objekte / Assets können mit Hitboxen (also Form und Größe) versehen werden.

/PK20/* Eine grundlegende Unterstützung von Kollisionsabfrage und deren Verarbeitung, basierend auf Position, Form und Größe, ist gewährleistet. Das heißt, dass das Aufeinandertreffen von zwei Objekten, basierend auf Position, Form und Größe der Hitbox erkannt und behandelt werden kann.

/PK30/* Wenn eine Kollision erkannt wird, kann ein entsprechendes Event ausgelöst werden.

/PK40/* Das Kollisionssystem stellt Methoden bereit, um Kollisionen zu lösen.

/PK50/* Die Hitboxen der Objekte (und physikalische Kräfte) sind während der Laufzeit modifizierbar, also an- und ausschaltbar.

/PK60/ Das Physik- und Kollisionssystem ermöglicht die Implementierung aller im GDD aufgezählten Gameplay Mechaniken. Die Engine kann insbesondere Gravitation simulieren, um die Mechanik “Black Hole Instance” zu ermöglichen.

/PK70/ Die Engine unterstützt Kollisionsgruppen, sodass bestimmte Objekte mit anderen kollidieren können, während Kollisionen mit anderen Gruppen ignoriert werden.

/PK80/ Die Game Engine kann verschiedene physikalische Kräfte wie Geschwindigkeit und Beschleunigung simulieren.

/PK90/ Objekte können zusätzlich zu ihrer Hitbox mit zusätzlichen physikalischen Eigenschaften wie Masse und Gewicht versehen werden. Diese werden bei der Berechnung von Kollisions- und Beschleunigungsvorgängen berücksichtigt.

/PK100/ Die Game Engine kann zusätzliche physikalische Eigenschaften wie z.B. Reibung simulieren können. Dies ermöglicht neue Mechaniken, wie z.B. eine Wolke, in welcher sich alle beweglichen Objekte langsamer bewegen.

3.1.5 Eventsystem

/EV10/* Das Eventsystem kann Ereignisse wie Nutzereingaben, Kollisionen und Animationen handhaben, auslösen und verarbeiten.

/EV20/* Die Engine kann Events als Reaktion auf bestimmte Konditionen auslösen, beispielsweise das Aufeinandertreffen von zwei bestimmten Objekten oder das Beenden eines Levels.

/EV30/* Es existiert ein Event Listener. Events müssen von anderen Stellen in der Engine abgefragt und untereinander priorisiert werden können. Dies kann parallel verlaufen, d.h. ein Event kann von mehreren Stellen gleichzeitig abgefragt werden.

/EV40/* Der Entwickler hat die Möglichkeit Events zu definieren, hinzuzufügen und zu bearbeiten.

/EV50/ Es ist möglich, Events je nach Priorität in Warteschlangen zu sortieren und u.U. zu vernachlässigen, um einen Event-Stau zu vermeiden. Redundant gleiche Events müssen aggregiert werden.

/EV60/ Es können Zähler für bestimmte Events angelegt und deren Stand temporär oder permanent gespeichert werden. Dies soll ein Achievementsystem und Statistiken ermöglichen.

3.1.6 KI - Allgemeine Anforderungen

/KI10/* Die Engine stellt Funktionen bereit um Objekte mittels KI-Logik zu bewegen und zu steuern. Dafür sind Position, Status und weitere Informationen von Objekten abfragbar.

/KI20/* Die KI kann auf das Eventsystem der Engine zugreifen.

3.1.7 KI - Höhere Taktik

Die genannten Anforderungen gehören nicht inhaltlich zu den Anforderungen an die Game Engine, sind jedoch Bestandteil der Game Logik und wurden als zusätzliche Konkretisierung dem Lastenheft beigelegt.

Da die hier genannten Anforderungen dem Bereich des Game Designs zuzuordnen sind, unterliegen sie einem gewissen Maß an Veränderbarkeit und Anpassung. Die genauen Spezifikationen und Prioritäten können im Laufe des Entwicklungsprozesses variieren und sollten stets den aktuellen Bedürfnissen und Zielen des Projekts angepasst werden.

/KI30/ Objekte mit KI (z.B. Gegner) können im Level navigieren. Das heißt, sie können sich an bestimmte Positionen bewegen oder, mit dem Zweck der Kollisionsvermeidung, ausweichen.

/KI40/ Objekte mit KI können Ziele mit Schüssen treffen oder auf sie zufliegen.

/KI50/ Objekte mit KI können Projektilen des Spielers, Hindernissen und Gegnern ausweichen bzw. vermeiden, in bestimmte andere Objekte hineinzufliegen oder sich von ihnen wegbewegen.

/KI60/ Nutzt der Spieler die Spezialfähigkeit „Decoy“, zielen und schießen die Gegner ausschließlich auf dieses, solange es existiert.

/KI70/ Fliegt eine „Rakete“ auf einen Gegner zu, versucht dieser, ihr auszuweichen.

/KI80/ Ist der Spieler, aufgrund der Fähigkeit „Teleport“ gerade nicht angreifbar, schießen die Gegner nicht auf ihn.

/KI90/ Gegner versuchen aktiv von einer „Black Hole Instance“ wegzufliegen.

/KI100/ Ein Gegner schießt nicht, wenn sich ein weiterer Gegner in seiner direkten Schussbahn befindet.

3.2 Medienausgabe

3.2.1 Audio

/AU10/* Das Audio System kann mehrere Sounds gleichzeitig stereo abspielen.

/AU20/* Die Lautstärke einzelner Soundchannel ist editierbar (vom Entwickler, nicht vom Spieler).

/AU30/ Die Gesamtlaustärke des Spiels kann durch den Spieler im Menü angepasst werden.

/AU40/ Der Nutzer hat die Möglichkeit, im Menü die Lautstärke der einzelnen Kanäle (z.B. Musik, Soundeffekte, Schüsse) anzupassen.

/AU50/ Überlagernde Sounds werden durch Normalisierung angepasst.

/AU60/ Es ist möglich, die Lautstärke von Sounds auf Basis der Distanz zur Quelle in Bezug auf den Spieler zu modifizieren.

/AU70/ Die Ausgabe der Geräusche erfolgt kanalbasiert, abhängig von der Position des Spielers in Bezug auf die Quelle, entweder im linken oder im rechten Audiokanal.

3.2.2 Grafik - Allgemeine Anforderungen

/GR10/* Die Engine besitzt ein Grafiksubsystem.

/GR20/* Die Engine kann ein Fenster im Vollbildmodus oder im Fenstermodus öffnen.

/GR30/* Die Engine unterstützt das Rendering von 2D Pixel Art Grafiken.

/GR40/* Sprites, Hintergründe und Tilemaps können gerendert und auf dem Bildschirm bewegt, skaliert und rotiert werden.

/GR50/* Die Game Engine ist in der Lage, Animationen darzustellen und die Frames der Animation zuverlässig zu durchlaufen, auch während schneller Bewegungen über den Bildschirm.

/GR60/* Einem Objekt können mehrere Assets zugeordnet werden.

/GR70/* Alpha Blending / Transparenz sind auf Assets anwendbar. Die Game Engine ist in der Lage, Transparenz darzustellen und Alpha Blending zu berechnen / beim Rendering anzuwenden. Assets können über- und untereinander dargestellt werden.

/GR80/ Um den Parallax Effekt darzustellen, ist das Grafiksystem in der Lage, verschiedene Ebenen zu rendern, welche sich in unterschiedlichen Geschwindigkeiten bewegen. Den Ebenen können verschiedene Objekte und Assets zugeordnet werden.

/GR90/ Partikelschwärme können dargestellt und animiert werden. Dabei können verschiedene Parameter wie Größe, Form, Farbe und Bewegung der Partikel festgelegt werden.

/GR100/ Die Geschwindigkeit der einzelnen Ebenen im Parallax Effekt ist abhängig von der Bewegungsgeschwindigkeit des Spielers..

3.2.3 Grafik - Spezielle Vorgaben

/GR110/ Die Engine stellt eine Top-Down Ansicht dar.

/GR120/ Die Auflösung des Spieles beträgt 640 x 480, was einem Verhältnis von 4:3 entspricht. Diese muss ganzzahlig dem Bildschirm angezeigt werden. Das Spiel kann im Fenster laufen oder im Vollbild, mit Rändern, welche das 4:3 Verhältnis herstellen, auch wenn der Monitor diesem nicht entspricht.

/GR130/ Die Assettypen sind gemäß den in der Tabelle 3.1 spezifizierten Abmessungen definiert.

/GR140/ Die Assets in Abbildung 3.1 sind für die jeweiligen Objekte im Prototypen zu nutzen.

/GR150/ Die Schriftart „pixel-love-font“ wird für sämtliche In-Game Texte verwendet. Davon ausgenommen sind Fullscreenanzeigen, z.B. vom Titelschriftzug.

Höhe in Pixel	Breite in Pixel	Objekttypen
8	8	Projektile, Partikel, Sprungschuss Spur, Raketen Spur
16	16	Große Projektile, Sprungschuss Projektil, Fadenkreuz Black Hole Instance, Fadenkreuz Teleport
32	32	Sehr große Projektile, Raketen, Kleine Gegner
64	64	Spieler, Gegner, 2 Spielerleben, 2 Boss Leben, Fähigkeitensymbol, Effekte, Decoy, Hindernisse
128	128	Highscore, Große Gegner, Große Fähigkeiten, Schutzschilde, Zonen
256	256	Bosse, Black Hole Instance
640	480	Screen Overlays

Tabelle 3.1: Übersicht über die Assetgrößen

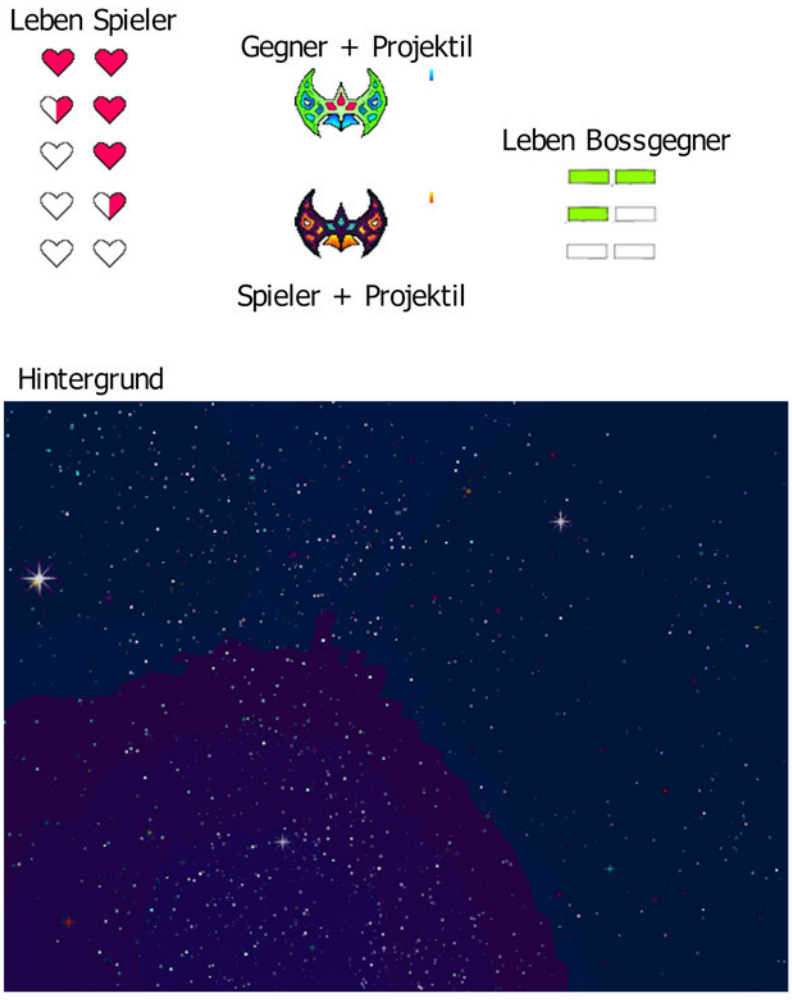

Abbildung 3.1: Eine Auswahl der zu verwendenden Assets.

3.3 Nutzerschnittstelle

3.3.1 User Interface - Allgemeine Anforderungen

Die UI Elemente betreffen sowohl das Grafik- als auch das Input Subsystem. Die Bedienung und Darstellung der Interfaces soll für den Spieler intuitiv, leicht navigierbar und optisch ansprechend sein. Sollte sich im Verlauf der Projektentwicklung herausstellen, dass dies nicht der Fall ist, sind Änderungen, auch grundlegender Art, am Interface und dessen Layout vorbehalten.

/UI10/* Die Engine kann UI Elemente über dem Spielgeschehen darstellen.

/UI20/* Die HUD Elemente (wie Ladebalken für Fähigkeiten) werden regelmäßig aktualisiert und entsprechend der jeweiligen Werte dargestellt.

/UI30/ Die Engine kann UI Elemente wie Textfelder und Knöpfe darstellen. Der Nutzer kann mit diesen interagieren (Buttons sind anklickbar, Textfelder können beschrieben werden).

/UI40/ Während des Spieles muss ein Pause Menü jederzeit aufrufbar und anzeigbar sein. Der Nutzer muss zwischen verschiedenen Bildschirmen, wie dem Startbildschirm, Game Over und Pause Bildschirm navigieren können.

/UI50/ Der Nutzer kann in den Einstellungen Änderungen an Schriftgröße, Größe der UI Elemente und Größe des HUD vornehmen. Die Einstellungen erlauben zudem eine Anpassung von Farben und Kontrasten, um Farbblindheit auszugleichen.

3.3.2 User Interface - Spezielle Vorgaben

/UI60/ Die Positionierung der Interface-Elemente des HUD entspricht der Abbildung 3.2. Die linke obere Ecke jedes Assets befindet sich an den in Abbildung 3.3 markierten Positionen.

/UI70/ Der Lebensbalken des Spielers füllt sich von rechts nach links. Die Interfacesprites des Spielerlebens haben einen Abstand von 10 Pixeln zueinander.

/UI80/ Der Lebensbalken eines Bossgegners füllt sich von links nach rechts. Die Interfacesprites des Bosslebens haben keinen Abstand zueinander.

3.3.3 User Interface - Input Management

/IM10/* Die Engine erkennt Nutzereingaben über Maus und Tastatur. Der Spieler kann Elemente auf dem Bildschirm auswählen, seine Spielerfigur steuern und in Textfelder schreiben / Eingaben tätigen.

Abbildung 3.2: In-Game Ansicht.

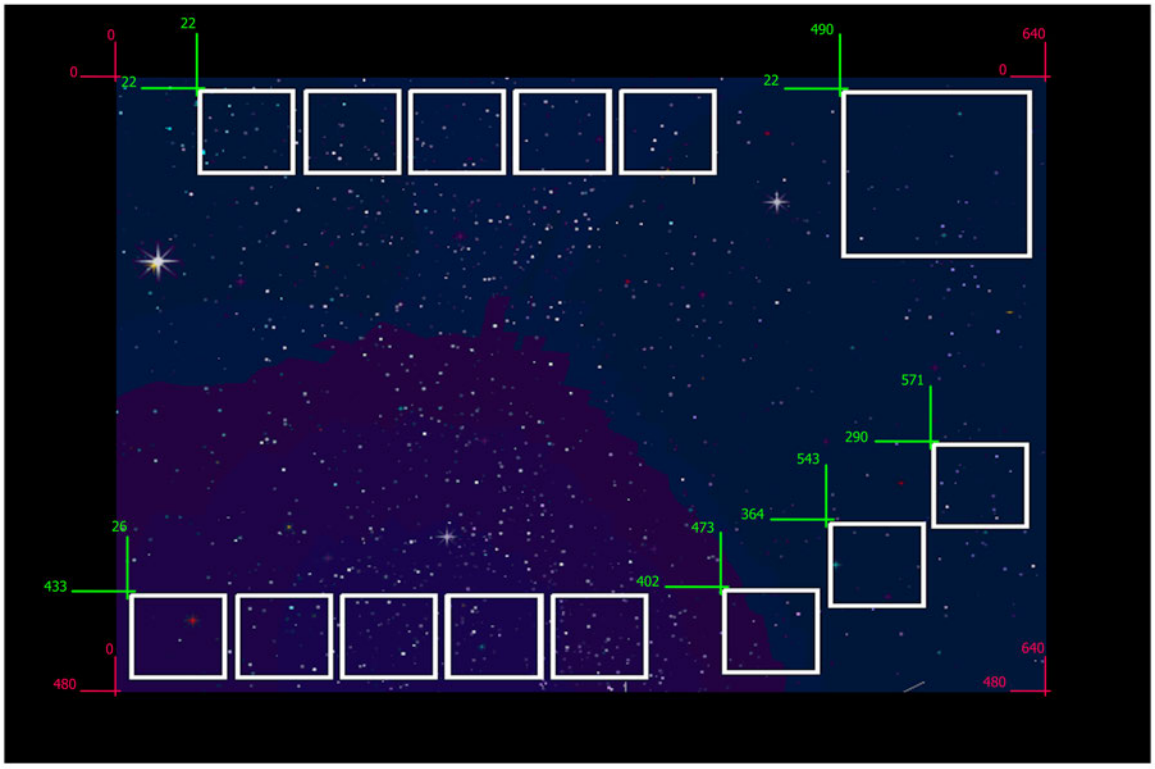

Abbildung 3.3: Koordinaten der Interfaceelemente auf dem Bildschirm.

/IM20/* Nutzereingaben müssen effizient be- und verarbeitet werden und dürfen keine merklichen, den Spielfluss behindernden oder Immersion brechende, Verzögerungen haben. Die Eingabeverzögerung darf 115 ms nicht überschreiten. Im Idealfall beträgt sie unter 90 ms.

/IM30/ Die Engine muss bei der Tastatursteuerung zwischen kurzen Tastenanschlägen und dem gedrückten Halten selbiger unterscheiden und diese als numerische Werte interpretieren können.

/IM40/ Die Engine erkennt Nutzereingaben über ein Gamepad. Der Spieler kann das Spiel mit dem Gamepad bedienen.

/IM50/ Um eine korrekte Steuerung mit einem Gamepad zu ermöglichen, muss die Game Engine eine entsprechende Funktionalität implementieren, die den sogenannten Soft Pull oder Partial Pull des Gamepads korrekt identifizieren und als verwendbaren, numerischen Wert interpretieren kann. Hierfür muss die Engine die Signale des Gamepads auslesen und entsprechend verarbeiten, um die korrespondierenden Bewegungen im Spiel korrekt ausführen zu können.

/IM60/ Die Engine erkennt Nutzereingaben über einen (über USB angeschlossenen) 8-Wege-Joystick. Der Spieler kann das Spiel mit dem Joystick und Knöpfen bedienen.

/IM70/ Die Game Engine ist in der Lage, mehrere Eingabegeräte gleichzeitig zu erkennen und zu unterscheiden. Für die Spielersteuerung sind sowohl Gamepads als auch Tastatur und Maus vorgesehen. Wenn ein Spieler eine dieser Optionen auswählt, muss die Engine sicherstellen, dass der Input des anderen Gerätes blockiert wird und keine Auswirkungen auf die Steuerung hat.

4 Abnahmekriterien

Alle Prototypbedingungen (*) sind erfüllt.

Auf gegenseitige Rechte und Pflichten zwischen Auftragnehmer und -geber wird im Lastenheft kein Fokus gesetzt, da das Projekt voraussichtlich ausschließlich studentisch und non-profitabel entwickelt wird. Demzufolge existieren in diesem Fall Auftraggeber und -nehmer nicht im klassischen Sinne. Das Lastenheft soll lediglich als Leitfaden für die zukünftige Entwicklung der Game Engine dienen, hat keinen konkreten Zeitplan und demnach nicht die, in der Projektentwicklung übliche, Customer-Client-dependance, -kommunikation und -kollaboration.

Glossar

Arcade Automaten öffentlich aufgestellte Geräte auf welchen Videospiele gespielt werden können.
Asset Bestandteil eines Inhalts eines Videospiels.
Assethierarchie für die vorliegende Arbeit entworfene Strukturierung aller Assets.

BuggyTech Engine teilweise umgesetzte Videospiel-Engine.

Cache temporärer Datenspeicher.

Datenmodell abstrahierte Darstellung von Daten und deren Beziehungen in einem System.

Engine Entwicklungsumgebung für Videospiele.
Entity-Relationship-Modell Datenmodell zur Darstellung von Entitäten, deren Attributen und Beziehungen untereinander.

Feature Funktion einer Softwareanwendung.
Flyweight Muster Softwareentwurfsmuster zur Optimierung des Speicherverbrauchs.
Framework Softwaregerüst zur Unterstützung von Anwendungsentwicklung.

Game Design Dokument verschriftlichte Zusammenfassung aller essenziellen Aspekte eines Videospiels.
Gameplay beschreibt das Gesamterlebnis eines Spielers beim Spielen eines Videospieles.

Hitbox nicht sichtbarer Bereich eines Videospielobjektes, welcher zur Bestimmung von Kollisionen genutzt wird.

Ist-Situation beschreibt existierende Produktfunktionen im Lastenheft.

Kollision Berührung von Game-Objekten basierend auf ihren Hitboxen.

Lastenheft Anforderungsdokument in der klassischen Softwareentwicklung.
Layout grafische Anordnung von Elementen einer Anwendung.

Moodboard zur Kommunikation von Design-Konzepten genutzte Collage.

Neon Nova für die BuggyTech Engine konzipiertes Spiel.

Pflichtenheft Dokument in der klassischen Softwareentwicklung, enthält konkrete Lösungsvorschläge für verlangte Anforderungen.
Prototyp frühe Umsetzung elementarer Funktionen eines Softwareprojektes in einer lauffähigen Anwendung, dient zu Test- und Feedbackzwecken.

Rendering der Prozess der Darstellung von grafischen Inhalten im Kontext von Softwareanwendungen.

Softwareentwurfsmuster reproduzierbare Lösung für ein häufiges Problem in der Softwareentwicklung.
Soll-Situation beschreibt geforderte Produktfunktionen im Lastenheft.
Sprite Grafik, welche Teil einer Animation ist.
Subsystem beschreibt im Kontext dieser Arbeit einen Bestandteil der Game Engine.

User Interface Benutzeroberfläche einer Softwareanwendung, umfasst alle digitalen, interaktiven Ein- und Ausgabeelemente.

Literaturverzeichnis

[1] G. Pezzi, *How to Make Your Own C++ Game Engine*, 2022. Adresse: https://www.gamedeveloper.com/blogs/how-to-make-your-own-c-game-engine (besucht am 04. 04. 2023).

[2] C. Johner und T. Geis, „Softwareentwicklung“, in *Praxishandbuch IT im Gesundheitswesen*. Carl Hanser Verlag GmbH & Co. KG, 2009, S. 1–38, ISBN: 978-3-446-41556-0.

[3] V.-J. Gaida, *Wasserfallmodell – klassisches Projektmanagement*, 2023. Adresse: https://www.factro.de/blog/wasserfallmodell/#was (besucht am 02. 04. 2023).

[4] H. F. Barber, „Developing Strategic Leadership: The US Army War College Experience“, *Journal of Management Development*, Jg. 11, Nr. 6, S. 4–12, 1992, ISSN: 0262-1711. DOI: 10.1108/02621719210018208.

[5] U. Reusche, *Warum es nach einer technischen Fortschrittsfokussierung eine menschliche braucht?*, 2022. Adresse: https://www.ifsm-online.com/2022/01/18/warum-es-nach-einer-technischen-fortschrittsfokussierung-eine-menschliche-braucht/ (besucht am 04. 04. 2023).

[6] K. Beck u. a., *Manifesto for Agile Software Development*, 2001. Adresse: http://www.agilemanifesto.org/ (besucht am 02. 04. 2023).

[7] C. Kleczewski, *Product Backlog*, 2022. Adresse: https://scrumguide.de/product-backlog/ (besucht am 02. 04. 2023).

[8] K. Becker und W. Bachmann, „Strategische IT-Beschaffung“, in *Praxishandbuch IT im Gesundheitswesen*. Carl Hanser Verlag GmbH & Co. KG, 2009, S. 307–337, ISBN: 978-3-446-41556-0.

[9] A. Zweimüller, Hrsg., *Das SCRUM Inkrement – Was ist das und wofür ist es gut?*, 2020. Adresse: https://www.agile-heroes.com/de/magazine/scrum-inkrement/ (besucht am 02. 04. 2023).

[10] *DIN 69901-5: Projektmanagement - Projektmanagementsysteme*. Berlin: Beuth, 2009.

[11] Verein Deutscher Ingenieure, *Vorgehensweise bei der Erstellung von Lasten-/Pflichtenheften* (VDI-Richtlinien). Düsseldorf: VDI-Verlag, 1996, Bd. 2519, Bl. 1.

[12] Institute of and Electrical and Electronics and Engineers and Institute of Electrical and Electronics Engineers, *IEEE Recommended Practice for Software Requirements Specifications*, 1998. DOI: 10.1109/IEEESTD.1998.88286.

[13] T. Fullerton, *Game Design Workshop A Playcentric Approach To Creating Innovative Games*, 3. Aufl. CRC Press Taylor & Francis Group, 2014, ISBN: 978-1-4822-1717-9.

[14] Playcent Games, Hrsg., *How to Make a Game Design Document (GDD)*, 2022. Adresse: https://playcentgames.com/how-to-make-a-game-design-document-gdd/ (besucht am 02. 04. 2023).

[15] Bitwave Games, Hrsg., *Truxton bei Steam*, 2023. Adresse: https://store.steampowered.com/app/2022880/Truxton/ (besucht am 05. 04. 2023).

[16] C. Dr. Schulz, Hrsg., *Das Datenmodell – die Strukturen eines Anwendungsbereiches eindeutig erfassen*, 2020. Adresse: https://www.palladio-consulting.de/datenmodell/ (besucht am 02. 04. 2023).

[17] G. Pernul und R. Unland, *Datenbanken im Unternehmen: Analyse, Modellbildung und Einsatz* (Lehrbücher Wirtschaftsinformatik), 2., korr. Aufl. München: Oldenbourg Wissenschaftsverlag, 2003, ISBN: 9783486272109. DOI: 10.1524/9783486594850.

[18] P. P.-S. Chen, „The Entity-Relationship Model—toward a Unified View of Data“, *ACM Trans. Database Syst.*, Jg. 1, Nr. 1, S. 9–36, 1976, ISSN: 0362-5915. DOI: 10.1145/320434.320440.

[19] A. Shvets, *Dive Into Design Patterns*. Kamianets-Podilskyi: Refactoring Guru, 2018.

[20] H. Breuer, J. R. Bessant und S. Gudiksen, *Gamification for innovators and entrepreneurs: Using games to drive innovation and facilitate learning* (Business & economics). Berlin und Boston: De Gruyter, 2022. DOI: 10.1515/9783110725582.

[21] C. Klaus, „Konzeption und teilweise Implementierung einer 2D-Game Engine“, Bachelor Arbeit, Hochschule Mittweida, 2019.

[22] R. Lord, *Why use an Entity Component System architecture for game development?*, 2012. Adresse: https://www.richardlord.net/blog/ecs/why-use-an-entity-framework.html (besucht am 02. 04. 2023).

[23] P. Gordon, „The theory behind first-person hitboxes: Choosing the right hitbox is key to making a great shooter. Here's the difference between them and why they matter“, *Wireframe*, Jg. 22, S. 30, 2019, ISSN: 2631-6722.

[24] I. Millington, *Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for Your Game*. CRC Press Taylor & Francis Group, 2010, ISBN: 978-0123819765.

[25] R. Eg, K. Raaen und M. Claypool, „Playing with Delay: With Poor Timing Comes Poor Performance, and Experience Follows Suit“, in *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, 2018, S. 1–6. DOI: 10.1109/QoMEX.2018.8463382.

[26] R. Nystrom, *Game programming patterns*. Genever Benning, 2014, ISBN: 978-0990582908.

[27] J. Gregory, *Game Engine Architecture*, 2nd edition. Boca Raton, FL: CRC Press Taylor & Francis Group, 2015, S. 1–989, ISBN: 978-1466560017.

[28] T. Stötzer, „Konzeption und prototypische Implementierung eines Event- und Audio-Systems für eine Game Engine“, Bachelor Arbeit, Hochschule Mittweida, 2021.

[29] MadMatty, *Mega Drive Longplay [200] Truxton*, World of Longplays, Hrsg., 2013. Adresse: https://www.youtube.com/watch?v=6iIG_ufZUzc&t=61s (besucht am 22. 03. 2023).

[30] ASUSTeK Computer Inc., Hrsg., *ASUS Tinker-Board - Ein ARM-basierter Single-Board-Computer*. Adresse: https://www.asus.com/de/networking-iot-servers/aiot-industrial-solutions/all-series/tinker-board/ (besucht am 22. 03. 2023).

[31] ScientiaMobile, Hrsg., *How Much RAM is in Smartphones?*, 2022. Adresse: https://www.scientiamobile.com/how-much-ram-is-in-smartphones/ (besucht am 21. 03. 2023).

Eidesstattliche Erklärung

Hiermit versichere ich – Elly Müller – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Mittweida, 06. April 2023 | Elly Müller

Ort, Datum | Elly Müller