Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Konzeptionierung und Entwicklung eines Spiels zur Wissensvermittlung über Hardware-Komponenten im PC

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft (HTW) Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang *Internationale Medieninformatik*

1. Gutachter: Prof. Dr. Gefei Zhang
2. Gutachter: Henning Müller

Eingereicht von Pauline Röhr [569735]

04.04.2022

Zusammenfassung

Das Gebiet der Rechnerarchitektur ist abstrakt, sehr umfassend und enthält Subthemen von hoher Komplexität. Demnach tendiert die Lehre der zugehörigen Inhalte zu einem ähnlichen Abstraktionsgrad und kann von Verbildlichungen sowie Mitteln zur Steigerung der Lernmotivation profitieren. Ein Lernspiel, das der Vermittlung dieser Inhalte dient, bietet eine Plattform zur Konkretisierung der abstrakten Elemente. Unter Anwendung erprobter Methoden für den Lehr- und Spielentwurf kann ein resultierendes Spiel die Lehre von Rechnerarchitektur sowohl bereichern als auch vereinfachen.

Abstract

The field of computer architecture is abstract, very extensive, and contains subtopics of high complexity. Accordingly, teaching the related content tends to a similar level of abstraction, and can benefit from imagery as well as means to enhance the motivation to learn. An educational game designed to teach this content provides a platform for concretization of the abstract elements. Using proven methods for instructional and game design, a resulting game can both enrich and simplify the teaching of computer architecture.

Hinweis

In dieser Arbeit wird für eine bessere Lesbarkeit auf die Verwendung weiblicher und männlicher Sprachformen verzichtet und stattdessen das generische Maskulin gewählt. Dieses bezieht sich ausdrücklich auf alle Geschlechteridentitäten gleichermaßen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Obwohl Rechner allgegenwärtig sind, ist das Gebiet der Rechnerarchitektur recht abstrakt. Nicht jeder Interessierte kann es sich leisten, einen Blick in einen PC zu werfen oder einen eigenen PC zusammenzubauen. Viele Personen haben Angst, die fragilen Komponenten zu beschädigen und trauen sich deswegen nicht, ihren PC unter die Lupe zu nehmen. Um tiefergehendes Wissen über Rechnerarchitektur zu erlangen reicht es allerdings nicht, lediglich die Komponenten zu betrachten. Die Ebene der digitalen Logik basiert hauptsächlich auf Modellen, und auch wenn diese Logik auf einer physischen Ebene umgesetzt wird, sind die dafür genutzten Transistoren in modernen Chips viel zu klein, um sie mit bloßem Auge zu erkennen, geschweige denn ihre Aufgaben und Funktionsweise durch einfaches Betrachten zu verstehen.

1.1 Problem- und Zielstellung

Ziel dieser Arbeit ist, die Wissensvermittlung über Computerkomponenten anhand eines Spiels zu ermöglichen und zu untersuchen. Das Gebiet der Rechnerarchitektur ist sehr umfassend und enthält Subthemen von hoher Komplexität. Deshalb muss im Rahmen dieser Arbeit entschieden werden, welche Zielgruppe ein solches Lernspiel erreichen soll und welche spezifischen Lernergebnisse angestrebt werden.

Um zu ermöglichen, dass das Spiel auch nach dem hier stattfindenden Entwicklungsprozesses erweitert und verbessert werden kann, ist es außerdem Teil der Zielstellung erweiterbaren, wartbaren, und sauberen Code zu schreiben.

1.2 Stand der Technik

Die Idee, Rechnerarchitektur über ein Spiel zu vermitteln ist nicht gänzlich neu. Seit der Einreichung des Themas dieser Arbeit wurde beispielsweise das Spiel *Turing Complete*[1] veröffentlicht, in welchem dem Spieler ermöglicht wird einen eigenen Turing-vollständigen Rechner aufzubauen. Da dieses Spiel zu Beginn dieser Arbeit jedoch noch nicht veröffentlicht war, konnte Spielerfeedback und Erfolgsquote bisher nicht mit einbezogen werden.

Ein weiteres Lernspiel für Rechnerarchitektur wurde im Rahmen der Veröffentlichung „Improving learning computer architecture through an educational mobile game“ [16] entwickelt. In ihrer Arbeit konnten die Autoren feststellen, dass ihr Spiel den

[1] `https://turingcomplete.game/`

Lernprozess von Rechnerarchitektur interessanter und vergnüglicher gemacht hat. Allerdings steht das hieraus entstandene Spiel nicht öffentlich zur Verfügung, weswegen interessierte Lerner nicht davon profitieren können.

Das kostenpflichtige Spiel *PC Building Simulator*[2], welches 2018 erstmalig erschienen ist, bietet dem Spieler die Möglichkeit seinen eigenen (Gaming-)PC zusammenzubauen. Dabei wird hauptsächlich Wert auf das Aussehen und die Qualität der Komponenten gelegt, und weniger auf die Funktionsweise oder einen Lerneffekt. Jedoch zeigt der Erfolg[3] dieses Spiels, dass ein Interesse an dem Aufbau und der grundlegenden Architektur eines Rechners besteht.

1.3 Aufbau der Arbeit

Um die Ziele dieser Arbeit zu erreichen, wurde in mehreren Phasen gearbeitet: Recherche, Konzeptionierung, Umsetzung, Testung. Um dies auch für den Leser sinnvoll darzustellen, sollen zunächst die Grundlagen und Voraussetzungen für das Projekt geklärt werden. Darauffolgend wird die in der Konzeptionierungsphase durchgeführte Problem- und Anforderungs-Analyse, sowie das resultierende Konzept, vorgestellt. Es folgt die technische Planung, sowie ein Entwicklungsbericht mit Beschreibungen der technischen Umsetzung. Anschließend werden die Ergebnisse aus der Testphase wiedergegeben und ausgewertet.

[2] `https://store.steampowered.com/app/621060/PC_Building_Simulator/`

[3] PC Building Simulator hat eine „Sehr Positiv“ Wertung auf der gängigen Spieleplattform Steam, bei über 30 000 Bewertungen.

2 Grundlagen

Um mit der Konzeptionierung eines Lernspiels über Hardwarekomponenten im PC zu beginnen, wurde zunächst eine Reihe an Recherchearbeiten getätigt, die sinnvolle und schlüssige Design-Entscheidungen für das Spiel ermöglichen sollten. Zusätzlich wurde im Rahmen der Recherche bestehendes Wissen über Rechnerarchitektur, Software-Projektplanung und Spieleentwicklung wiederholt und vertieft. Das erlangte Wissen aus dieser Recherche, sowie vorher existierendes, relevantes Wissen, soll in diesem Kapitel als Grundlage zusammengefasst werden.

2.1 Wissensvermittlung durch Spiele

Spiele für die Wissensvermittlung (auch Serious Games genannt) zu verwenden bietet den Lernenden eine Vielzahl an Vorteilen, "wie beispielsweise die freie Wahl des Lernortes, eine flexible Zeiteinteilung, die Wahl des Lernzeitpunktes, das Eingehen auf das individuelle Lerntempo des einzelnen Lerners, das selbständige Erarbeiten der Spielsituation und das damit verbundene selbstgesteuerte Lernen"[4, S. 63]. Um ein Spiel zu entwerfen, welches effizient Wissen vermitteln kann, soll auf erprobte und wissenschaftlich begründete Methoden zurückgegriffen werden.

2.1.1 Lern- und Motivationstheorien

Auf Basis verschiedener Lern- und Motivationstheorien aus dem Bereich der Psychologie und Didaktik ist es möglich, Design-Prinzipien für Serious Games zu erheben. Tabelle 2.1 ist eine übersetzte Teil-Abschrift aus *The Gamification of Learning and Instruction* [6]. Sie bietet eine Übersicht über verschiedene Lerntheorien und die möglichen Schlussfolgerungen für das Game Design von Lernspielen.

2.1.2 Wissensarten und zugehörige Lehrstrategien

Je nach Art des zu lehrenden Wissens ergeben sich unterschiedliche Techniken und Mechaniken, um Dieses zu vermitteln. Zu den vier Wissensarten nach Anderson und Krathwohl [7] gehören deklaratives, konzeptuelles , prozedurales und metakognitives Wissen. Die Lehre von metakognitivem Wissen ist dabei für diese Arbeit nicht von Relevanz.

Tabelle 2.1: *Lerntheorien und ihre Auswirkung auf Gamification, übersetzte Teil-Tabelle aus The Gamification of Learning and Instruction [6, S. 74]*

Theorie	**Auswirkung auf das Gamification Design**
Cognitive Apprenticeship	Der Rahmen und die Umgebung sollte authentisch sein und Feedback und Anleitung für die Aktivität des Lernenden bieten.
Lepper's Instructional Design Principles for Intrinsic Motivation	Einbeziehung von Elementen für Kontrolle des Lernenden, Herausforderung, Neugierde und Kontextualisierung.
Die Taxonomie von intrinsischen Motivationen zum Lernen	Einbeziehung von intern und extern motivierenden Elementen wie zum Beispiel Herausforderung, Kontrolle, Fantasie, Kooperation, Wettbewerb und Anerkennung.
Selbstbestimmungstheorie	Darbietung von verschiedenen Autonomie-Möglichkeiten, ein Gefühl der Kompetenz und der Verbundenheit mit anderen.
Distributed Practice	Zeitliche Verteilbarkeit des Spiels, um verteilte Wiederholung der Inhalte zu ermöglichen

Deklaratives Wissen

Deklaratives Wissen beinhaltet Fakten, spezifische Details und Terminologie und lässt sich hauptsächlich durch Auswendiglernen einprägen. Übliche Strategien, um diese Art von Wissen zu vermitteln sind Ausarbeitung, Organisation, Assoziation und Wiederholung. Diese Strategien lassen sich wiederum in Spielmechaniken umsetzen. Zur Ausarbeitung von neuem Wissen bietet es sich beispielsweise an eine Geschichte zu erzählen, welche dem Lernenden die Möglichkeit bietet, neue Fakten in einen bekannten Kontext zu bringen. Sortieren und Zuordnen (Matching) sind häufig verwendete Spielmechaniken, die dem Spieler Organisation und Assoziation erlauben. Wiederspielbarkeit hingegen ermöglicht die Wiederholung des erlangten Wissens. Ein typischer Spieltyp für die Vermittlung von deklarativem Wissen sind Trivia-Spiele, da diese ebenfalls eine (regelmäßige) Wiederholung des Wissens, sowie Assoziation und Organisation ermöglichen [6, S. 168-171].

Konzeptuelles Wissen

Zu konzeptuellem Wissen gehören Klassifizierungen, Kategorien und Generalisierungen sowie Prinzipien, Theorien, Modelle und Strukturen. Es behandelt die Beziehungen zwischen den Elementen einer größeren Struktur. Um Konzepte zu vermitteln werden häufig Metaphern verwendet, da diese eine Verknüpfung zwischen bekannten Ele-

menten und dem zu lernenden Konzept bieten. Eine simplere Strategie konzeptuelles Wissen zu lehren, bietet das Darlegen von Beispielen und Gegenbeispielen, durch welche der Lernende in der Lage ist, Merkmale eines Konzepts abzugrenzen. Die Klassifizierung dieser Merkmale bietet eine weitere Strategie zur Lehre von Konzepten. Übersetzt man diese Strategien wiederum in Spielmechaniken, so bieten sich erneut Sortier- und Matching-Spiele durch die Möglichkeit der Kategorisierung und erneuten Zuordnung der Attribute des Konzepts an. Spiele bieten dem Spieler außerdem eine Plattform dafür, das Konzept sowohl in der korrekten als auch in einer inkorrekten Form zu erleben. Beispiele und Gegenspiele lassen sich so in einer gegebenenfalls metaphorischen Umgebung darstellen [6, S. 172-177].

Prozedurales Wissen

Wissen über fachspezifische Techniken, Methoden, Fähigkeiten und Algorithmen sowie Wissen über die angemessenen Anwendungsfälle dieser, wird als prozedurales Wissen bezeichnet. Um prozedurales Wissen zu vermitteln, empfiehlt es sich zunächst einen Überblick über den Ablauf zu geben, sodass der Lernende die Prozedur und ihre einzelnen Schritte in einen Kontext bringen kann. Sobald dies geschehen ist sollte gelehrt werden, wie und warum einzelne Schritte der Prozedur ausgeführt werden. Sollte ein Problem in der zukünftigen Ausführung der Prozedur auftreten, ermöglicht das Wissen über die Gründe der einzelnen Schritte die Erarbeitung von Ausweichlösungen und somit die Fähigkeit sich den Umständen anzupassen [6, S. 181].

2.1.3 Entwicklung von Spielen für die Wissensvermittlung

Für den Design- und Entwicklungsprozess von Instruktions- bzw. Lernspielen gilt es darauf zu achten, die bisher genannten Eigenschaften des zu vermittelnden Wissens zu erkennen und passende Methoden und Strategien anzuwenden. Das ADDIE-Instruktionsdesign-Modell ist ein Rahmenplan für diese Prozesse, basierend auf den fünf Schritten Analyse, Design, Entwicklung, Implementierung und Auswertung [6].

Eine Analyse der Problem- und Zielstellung ist der erste Schritt des ADDIE-Modells. Sie soll eine Zielgruppe festlegen und die Fragen beantworten, wie alt Teilnehmer sind, welches Vorwissen diese mitbringen, wie hoch die zu erwartende Lernmotivation ist und ob diese eher intrinsisch oder extrinsisch ist. Außerdem sollte die Analyse die Inhalte, Aufgaben und Lernziele auffassen, sowie die technischen Voraussetzungen, benötigte Kompetenzen und benötigte Ressourcen beinhalten [15, S. 8f].

Im Design-Schritt werden angemessene Instruktionsstrategien ausgewählt und geplant. Es ist üblich, diesen Schritt anhand der Fertigung eines Design-Dokuments umzusetzen. Dies ist auch im Entwicklungsprozess von Spielen üblich[1]. Ein Design-

[1] siehe zum Beispiel die Aufteilung des kreativen Prozesses in der Spieleentwicklung nach K. Oxland in Gameplay and Design [10]

Dokument für Instruktionsdesign soll eine Inhaltsübersicht sowie Informationen über gewählte Instruktionsstrategien bieten. Ein Game-Design-Dokument (GDD) enthält hingegen Informationen über den Spielverlauf, die Spielziele und die geplante Ästhetik und Stimmung des Spiels. Das finale Dokument soll die Basis für die Entwicklung darstellen, jedoch soll es nicht als festes Konstrukt betrachtet werden, da sich das beschriebene Design und die daraus folgenden Anforderungen im Rahmen der Entwicklungsphase ändern können [6, S. 205].

Sobald die Design-Phase abgeschlossen ist, kann auf Basis der Design-Entscheidungen mit der Entwicklung begonnen werden. Für Videospiele ist es hierbei empfehlenswert auf Basis eines zusätzlichen technischen Designs zu arbeiten, sodass die geplanten Spielfunktionen in einem organisierten Zeitrahmen und in einer schlüssigen Software-Architektur entwickelt werden können. Im Rahmen der Entwicklungsphase sollte regelmäßig getestet und bei möglichen Fehlern im Design Anpassungen an diesem vorgenommen werden.

Die darauf folgende Implementierungsphase beinhaltet hauptsächlich den tatsächlichen Einsatz der entwickelten Lernumgebung. Dabei soll darauf geachtet werden, dass die Zielgruppe wie geplant auf das Produkt zugreifen kann und wenn nötig Unterstützung zur Nutzung erhält.

Die Evaluation beinhaltet die Auswertung von "technisch-funktionale[n] Aspekte[n] wie: Stabilität, Übersichtlichkeit, Funktionalität, Ästhetik"[15, S. 10], sowie didaktische Fragen, welche schon in den früheren Phasen vorbereitet und bei der Planung einbezogen werden sollen. Dazu gehört einerseits eine formative Evaluation, welche während der Design- und Entwicklungsphasen stattfindet und neues Material sowie erstes Feedback für benötigte Änderungen verwendet. Andererseits beinhaltet diese Phase auch die summative Evaluation, welche erst am Ende der Entwicklung stattfinden kann und den Wert des Endprodukts ermitteln soll. Dazu bietet es sich an, eine Umfrage unter den Teilnehmenden durchzuführen. „Gefragt werden sollte

- nach der Akzeptanz (Handhabung, Interesse, Motivation, Inhalt),
- nach den Lernprozessen und -Ergebnissen (Erwartungen, welche Lernprozesse finden statt und wie werden sie bewertet? Werden die formulierten Lehrziele erreicht?),
- und nach Verbesserungsvorschlägen sowie einem Fazit“ [15, S. 11].

2.2 Rechnerarchitektur

Anhand der soeben aufgeführten Informationen wird in dieser Arbeit ein Spielkonzept aufgesetzt, für welches jedoch zusätzlich Wissen über Rechnerarchitektur benötigt wird. Ein Grundwissen über die Verwendungszwecke von Computern wird zwar vorausgesetzt, jedoch soll der grobe Aufbau und die wichtigsten Informationen über die Funktionsweise von modernen Rechnern zunächst wiederholt werden.

Öffnet man einen modernen Desktop-PC so findet man dort üblicherweise folgende Komponenten:

- Mainboard
- Central Processing Unit (CPU)
- Primärer Speicher
- Sekundärer Speicher
  - Hard Disk Drive (HDD)
  - Solid State Drive (SSD)
- Gegebenfalls Grafikkarte

Das Mainboard (auch Hauptplatine) beinhaltet Steckplätze für alle Komponenten und Schnittstellen für verschiedene Anschlüsse und Peripheriegeräte. Es dient demnach dazu, jegliche Komponenten zusammenzuführen und die Kommunikation zwischen diesen zu ermöglichen. „Die CPU oder der Prozessor [...] führt die im Hauptspeicher abgelegten Programme aus, indem sie deren Befehle nacheinander abruft, analysiert und dann ausführt“ [14, S. 75]. Der primäre Speicher (auch Hauptspeicher, Arbeitsspeicher) ist meist flüchtig und beinhaltet die genannten aktiven Programme, während der sekundäre Speicher nicht flüchtig ist und somit langzeitig Daten speichern kann. In modernen Rechnern sind häufig zwei verschiedene Arten von sekundären Speichern auffindbar: HDDs und SSDs. Dabei sind SSDs die modernere, und teurere Variante, bieten dafür jedoch schnellere Zugriffszeiten. In vielen PCs (insbesondere Gaming-PCs) findet man außerdem eine Grafikkarte, welche eine für die Berechnungen von Grafikverarbeitungen spezialisierte Graphics Processing Unit (GPU) beinhaltet.

Da die CPU dafür zuständig ist, jegliche Befehle auszuführen und somit die Funktionsweise eines Rechners am besten beschreibt, ist es hilfreich, die Funktionsweise der CPU genauer zu betrachten. In modernen CPUs wird üblicherweise die Architektur eines Von-Neumann-Rechners umgesetzt. Dieser ist in fünf Funktionseinheiten unterteilt: das Steuerwerk, das Rechenwerk, den Speicher sowie einem Eingabe- und Ausgabewerk. "Das Steuerwerk (die Steuereinheit - Control Unit) ruft Befehle aus dem Hauptspeicher ab und bestimmt den Befehlstyp. Das Rechenwerk (Arithmetic Logic Unit, ALU) führt Operationen aus, die für den jeweiligen Befehl erforderlich sind, beispielsweise Addition oder boolesche AND-Verknüpfung"[14, S. 75]. Der Speicher besteht aus mehreren Registern, welche sich innerhalb der CPU befinden und somit mit hoher Geschwindigkeit gelesen und beschrieben werden können. Einige Register haben eine bestimmte Funktion, zum Beispiel das Befehlsregister, welches den momentan ausgeführten Befehl aufnimmt, und der Befehlszähler, welcher auf den nächsten auszuführenden Befehl im Hauptspeicher zeigt [14, S. 75].

Um Befehle auszuführen, arbeitet eine CPU eine Reihe von Schritten ab, welche sich im sogenannten Abrufen-Decodieren-Ausführen-Zyklus anordnen. Dieser besteht aus den folgenden Schritten:

1. „Den nächsten Befehl aus dem Speicher in das Befehlsregister abrufen.
2. Den Programmzähler ändern, damit er auf den nächsten Befehl zeigt.
3. Den Typ des gerade eingelesenen Befehls bestimmen.
4. Falls der Befehl ein Speicherwort benötigt, dessen Position bestimmen.
5. Das Wort bei Bedarf in ein CPU-Register laden.
6. Den Befehl ausführen.
7. Zu Schritt 1 gehen, um den nächsten Befehl auszuführen.“ [14, S. 77]

Um den Rahmen dieser Arbeit einzuhalten, wird weiterführendes Wissen hier nicht aufgezeigt. Ein grobes Verständnis für die Inhalte aus Kapitel 1-3 des Buches *Rechnerarchitektur* [14] ist jedoch empfehlenswert.

2.3 Organisation von Software-Projekten

Die Planung und Organisation eines Software-Projekts findet üblicherweise durch einen Projektmanager als Teil des Entwicklungsteams statt. Die hierfür getätigten Arbeiten sind ebenfalls ein Teil des folgenden Entwicklungsprozesses, weswegen ein Grundwissen über Strategien, Methoden und Tools erforderlich ist. In Vorbereitung auf spätere Kapitel sollen drei Methoden genauer betrachtet werden: Feature Driven Development (FDD), Extreme Programming und Kanban.

2.3.1 Feature Driven Development

„FDD ist eine kundenorientierte Entwicklungsmethode mit kleinen Teilresultaten“ [3, S.3]. Zur Umsetzung eines Projekts mit dieser Strategie wird in FDD anhand von fünf Prozessen gearbeitet:

1. „Entwickeln eines Gesamtmodells
2. Erstellen einer Feature-Liste
3. Planung pro Feature
4. Entwurf pro Feature
5. Implementierung pro Feature “[3, S.5]

Feature Driven Development bietet damit verschiedene Vorteile. Der Fokus liegt auf dem Erreichen von Einzelergebnissen, wodurch ein stetiger Fortschritt im Projekt erzielbar und erkennbar ist. Außerdem sind Fehler so frühzeitig erkennbar [8].

2.3.2 Extreme Programming

Die Methode des Extreme Programming basiert auf der Idee, iterative Entwicklungsprozesse zu nutzen und auf Zeitinvestitionen für tiefgründige Designs und Pläne zu verzichten, um auf regelmäßige Anforderungsänderungen reagieren zu können. Damit auch ohne detailreiches Design ein geordneter und zielführenden Entwitcklungsprozess geboten werden kann, beinhaltet Extreme Programming eine aus fünf Werten hergeleitete Reihe von Techniken und Regeln, an welchen sich Teammitglieder orientieren sollen:

- Einfachheit
- Kommunikation
- Feedback
- Respekt
- Mut

Die zugehörigen Techniken und Regeln lassen sich auf der Extreme Programming Website finden [18]. Ähnlich zum Feature Driven Development bietet Extreme Programming eine hohe Anpassungsfähigkeit an sich ändernde Anforderungen sowie frühe Fehlererkennung. Außerdem kann durch Extreme Programming überflüssige Programmierarbeit vermieden werden [2].

2.3.3 Kanban

Das Kanban-System bietet einen organisierten Produktionsprozess für die Optimierung der Verwendung von verfügbaren Ressourcen, wozu hauptsächlich die Kapazitäten des zugehörigen Teams betrachtet werden. Die wichtigsten Methoden dieses Systems sind:

- Visualisierung der Arbeitsprozesse
- Limitierung der aktiven Aufgaben (Work in Progress, WIP)
- Verwaltung der Arbeitsabläufe (Flow)
- Verwendung explizit formulierter Praktiken und Regeln
- Implementierung von Feedback-Zyklen
- Gemeinsame Verbesserung durch experimentelles Vorgehen [1]

Für die Umsetzung dieser Methoden wird häufig ein Kanban-Board verwendet, in welchem zugehörige Aufgaben erstellt, formuliert und sortiert werden können. Üblicherweise unterteilt sich ein solches Board in drei Spalten: Eine Spalte mit offenen Aufgaben (Backlog, To Do), eine Spalte mit derzeitig aktiven Aufgaben (In Progress) und eine Spalte mit den bisher abgeschlossenen Aufgaben. Dies lässt sich beliebig an die geplanten Prozesse anpassen.

2.4 Technische Grundlagen

Für die Entwicklung eines Spiels bietet sich die Verwendung einer Game Engine an, da diese eine vollständige Entwicklungsumgebung anbieten kann, sowie schon vorgefertigte Funktionen für Spiele, wie zum Beispiel Physik, die Grafik-Darstellung (Render-Pipeline), und Werkzeuge zur Verwendung von Ein- und Ausgabe. Aus früheren Erfahrungen heraus wird die Unity Engine gewählt, da diese die Möglichkeit bietet, in kurzer Zeit und mit keinem bis wenig Budget ein vollständiges Spiel zu entwickeln. Gearbeitet wird demnach mit der Programmiersprache C-Sharp (C#).

2.4.1 Unity

Ein Grundverständnis der Unity Engine wird für diese Arbeit vorausgesetzt. Dennoch sollen im Folgenden die wichtigsten Elemente der Spieleentwicklung mit Unity zusammengefasst werden.

Der Editor

Der Unity Editor ist das User-Interface der Engine. Er bietet eine Vielzahl an Basisfunktionen und Einstellungsmöglichkeiten, welche für die Spieleentwicklung verwendet werden können. Dazu gehört eine Übersicht über die Projektordner und -dateien, der Datei- bzw. Objektinspektor, eine Szenenansicht mit zugehöriger Objekthierarchie sowie eine Spielansicht, in welcher es möglich ist, das Spiel zu starten. Der Inspektor bietet dabei spezifische Funktionen zur Anpassung ausgewählter Elemente im Projekt, wie zum Beispiel das Hinzufügen von Skriptlogik oder die Anpassung der Position eines Objekts.

Eine Szene in Unity bezeichnet eine Datei, in welcher Spielumgebungen erstellt und angepasst werden können. Die zugehörige Szenenhierarchie bietet die Möglichkeit, sogenannte GameObjects in diesen Szenen zu gruppieren und zu sortieren. So lassen sich verschiedene Elemente ineinander verschachteln und kombinieren. GameObjects können in diesem Zusammenhang vieles sein: ein einfacher Würfel, eine Kamera, ein Licht oder sogar ein leeres Objekt. Für alle GameObjects gilt jedoch, dass sie eine `transform` Komponente haben. In dieser wird die Position, Rotation und Skalierung, sowieso die jeweilige Hierachie des GameObjects beschrieben.

Die von Unity zur Verfügung gestellten Funktionen des Editors sind sehr umfassend und lassen sich nur schwer zusammenfassen. Dennoch kann es sein, dass bestimmte Funktionen fehlen oder für neue Objekte neue Funktionen hinzugefügt werden sollen. Für diese Fälle ist es möglich, Editor-Skripts zu schreiben, die die Benutzeroberfläche erweitern und verändern können.

MonoBehaviour

Um einem Objekt in einer Szene Logik hinzuzufügen, kann ein C#-Skript erstellt werden, welches von der Klasse MonoBehaviour erbt. Diese Klasse stellt eine Blaupause für das Erstellen von Objekt-Typen dar, welche an GameObjects gehangen werden kann [17]. Alle GameObjects durchlaufen einen gewissen Zyklus im Spielprozess, auf welchen durch die MonoBehaviour-Klasse zugegriffen werden kann. Dazu gehören beispielsweise die `Update` Methode, welche in jedem Frame des Spiels ausgeführt wird, und die `Start` Methode, welche Teil des Ladeprozesses eines Unity Spiels ist und somit für Initialisierungprozesse verwendet werden kann.

Prefabs

Es kommt häufig vor, dass GameObjects in mehrfacher Ausführung verwendet werden und somit eine Vorlage für diese angelegt werden soll. Dafür gibt es in Unity sogenannte Prefabs. Diese kann man anhand schon existierender GameObjects als Datei im Projekt anlegen, in welcher jegliche Informationen über das GameObject gespeichert werden. Das Prefab kann dann als Vorlage für Objekte in einer Szene oder in anderen Prefabs dienen. Wird das Prefab in der zugehörigen Prefab-Ansicht des Editors bearbeitet, so wird diese Veränderung für alle Verwendungen der Vorlage ebenfalls angewendet.

ScriptableObjects

Im Gegensatz zur MonoBehaviour-Klasse bieten ScriptableObjects die Möglichkeit Daten zu speichern und Logik zu implementieren, die nicht an ein bestimmtes Objekt gebunden sein sollen. Klassen, die von `ScriptableObject` erben, können verwendet werden, um zugehörige Asset-Dateien im Projekt zu erstellen. Häufiges Ziel dabei ist die Auslagerung von großen Datenmengen.

2.4.2 Softwarearchitektur

Für die saubere Entwicklung eines Spiels gelten die gleichen Richtlinien und Regeln, wie für andere Software auch. Bestimmte Entwurfsmuster sind für die Spieleentwicklung und folglich für diese Arbeit jedoch von besonderer Relevanz, weswegen diese grob beschrieben werden sollen.

Event-Driven Architecture

Eine Event-Driven Architecture ist ein Software-Entwurfsmuster, in welchem verschiedene Ereignisse asynchron verarbeitet werden. Die in diesem Projekt verwendete Broker-Topologie besteht hauptsächlich aus zwei Komponenten: dem Broker und dem

Event-Prozessor. Der Broker beinhaltet dabei alle Eventkanäle, über welche die Ereignisse vom Hervorrufer zum Event-Prozessor gelangen. Der Event-Prozessor ist anschließend dafür zuständig, das Event wie gewünscht zu verarbeiten [11].

Dieses Entwurfsmuster bietet für die Entwicklung in Unity einige Vorteile. Durch die Entkopplung des Event-Hervorrufers vom Event-Prozessor ist eine Event-Driven Architecture stark skalierbar und sehr performant [11]. Sie ermöglicht in Unity fast vollständig auf die Verwendung der `Update` Methode zu verzichten, welche bei übermäßigem Gebrauch zu Performance-Problemen führen kann. Da solche Probleme in Spielen sofort anhand der Bildfrequenz (Frame Rate) auffallen, müssen diese dringend vermieden werden.

State Machines

> „Bei einem sogenannten Endlichen Zustandsautomaten (engl. finite state machine, kurz FSM) handelt es sich um die Realisation eines Steuerungskonzeptes, welches eine abstrahierte Maschine zum Vorbild hat, die eine Reihe von Zuständen besitzt, durch die sich ihre Funktion definiert. Diese Maschine arbeitet, indem sie von einem Zustand in einen anderen übergeht und bei den Zustandsübergängen oder dem Verharren in Zuständen bestimmte Aktionen ausführt. Dabei ergibt sich der Folgezustand aus der Kombination des momentanen Zustands und einem externen Ereignis, z. B. einem Tastendruck oder einem Signal aus anderen Bereichen der Software.“ [13]

Die verschiedenen Zustände, die in Spielen eingenommen werden (beispielsweise Hauptmenü, Pausierung, im Spiel), beinhalten üblicherweise viele dazugehörige Informationen. Eine Finite State Machine kann dabei helfen Zustandsübergänge zu vereinfachen und fehlerfrei zu halten.

3 Analyse der Problem- und Zielstellung

Auf Basis des ADDIE-Modells beginnt der Entwicklungsprozess eines Spiels für die Wissensvermittlung mit einer Analyse der Problem- und Zielstellung. Um ein möglichst vollständiges und breitgefächertes Design erstellen zu können, werden die Ziele zunächst sehr viel ambitionierter gewählt, als im Rahmen dieser Arbeit erreichbar ist. Für die tatsächliche Entwicklung und erste Testphase des Spiels werden nur einige der geplanten Features ausgewählt und umgesetzt.

3.1 Zielgruppe

Die Hauptzielgruppe des Spiels beschränkt sich auf Schüler und Studierende, welche im Bereich der Informatik tätig sind, da Rechnerarchitektur ein Teilgebiet der technischen Informatik und somit auch Gegenstand der allgemeinen Informatik ist. Das Spiel kann den hier inbegriffenen Zielpersonen Unterstützung im Lernprozess für entsprechende Klausuren bieten oder den Lehrkräften entsprechender Kurse ein Hilfsmittel in der Lehre darstellen. Die Zielgruppe lässt sich zusätzlich durch technisch-interessierte PC-Gamer erweitern, da auch diese Interesse am Aufbau und an der Funktionsweise ihres Rechners haben können (siehe beispielweise *PC Building Simulator* in Kapitel 1.2).

Personen aus dieser kombinierten Zielgruppe befinden sich vermutlich im Altersbereich von jungen Erwachsenen. Es ist annehmbar, dass die Spieler technisch affin sind und zumindest schon einige Erfahrung mit der Bedienung von PCs gesammelt haben. Sie besitzen somit alle ein Verständnis für die Funktion von Rechnern und warum es für sie persönlich wichtig ist, dass ihre Rechner plangemäß funktionieren. Das Vorwissen im Gebiet der Rechnerarchitektur kann jedoch von Spieler zu Spieler stark variieren. Um die Zielgruppe möglichst offenzuhalten, soll das Spiel für all diese Spieler Lernmöglichkeiten bieten.

Auch die Art und Stärke der Lernmotivation kann zwischen den Spielern aus dieser Gruppe variieren. Für Studierende, die das Spiel zum Lernen für eine Klausur verwenden wollen, ist die Motivation eher extrinsisch. Andere greifen das Spiel eher aus purem Interesse auf und sind somit eher intrinsisch motiviert. Unter beiden Umständen wird jedoch eine mittlere bis hohe Lernmotivation angenommen.

3.2 Voraussetzungen für die Entwicklung

Für die ermittelte Zielgruppe lässt sich annehmen, dass jeder der zugehörigen Personen ein verwendbarer PC mit mittlerer bis hoher Leistungsstärke, sowie mit Bildschirm, Maus und Tastatur zur Verfügung steht. Dieser kann beispielsweise aus der entsprechenden Lehrinstitution oder aus persönlichem Besitz stammen. Über das Betriebssystem der PCs lässt sich jedoch keine Aussage treffen, weswegen darüber nachgedacht werden sollte, die Verfügbarkeit des Spiels für Windows-, MacOS- und Linux-Nutzer zu ermöglichen. Die Unity-Engine bietet die Möglichkeit Spiele für Web sowie für die genannten Betriebssysteme zu bauen, weswegen diese für die Umsetzung verwendet werden soll.

Die Entwicklung des Lernspiels erfordert Kompetenzen in verschiedenen Bereichen. Dazu gehört Management, Instruktionsdesign, Game Design, Rechnerarchitektur, Kunst (Game-Art), Programmieren, Sound, Animation und Level Design. Um diese Gebiete im Rahmen der Entwicklung abdecken zu können, werden entsprechende Ressourcen benötigt. Tabelle 3.1 zeigt den geplanten Tech-Stack, welcher die genannten Bereiche abdecken soll.

Tabelle 3.1: *Der geplante Tech-Stack*

Bereich	**Ressourcen**
Projektmanagement	Trello
Design und Planung	Google Docs
Game-Art	Blender, GIMP
Programmierung	JetBrains Rider (IDE)
Versionskontrolle	Git, GitHub
Sound und Level Design	Unity Editor

3.3 Ziele des Spiels

Das Spiel soll den Lernenden Basis- und Detailwissen über den Aufbau und die Funktionsweise von Rechnern vermitteln. Es soll den Spielern die Möglichkeit bieten, das Aussehen, die Funktionen und die Funktionsweise aller Komponenten eines modernen PCs kennenzulernen. Dementsprechend soll den Spielern auch die Option geboten werden, sich mit der Ebene der digitalen Logik und somit beispielsweise mit Boolescher Algebra und dem Aufbau verschiedener Schaltnetze auseinanderzusetzen. Außerdem sollen Spieler sich mit mathematischen Hintergründen beschäftigen können, sodass auch der Umgang mit Zahlen aus verschiedenen Zahlensystemen (binär, hexadezimal) und Gleitkommazahlen anhand des Spiels erlernt werden kann. Tabelle 3.2 bietet einen Überblick über die Lernziele, sowie eine jeweilige Zuordnung des zu vermittelnden Wissens. Die Tabelle ist an einigen Stellen noch erweiterbar, beispielsweise durch das

Hinzufügen einer Kategorie für Peripherie-Geräte. Allerdings muss für diese Erweiterungen jeweils das entsprechende Fachwissen erlangt werden, bevor ein entsprechendes Design angelegt werden kann.

Tabelle 3.2: *Lernziele und ihre jeweilige Wissensart*

Kategorie	**Lernziel**	**Wissensart**
Grundwissen	Typischer Aufbau eines Modernen Rechners	Deklarativ
	Benennung, Aussehen und Funktion der üblichen Komponenten in modernen PCs	Deklarativ
Geschichte der Rechnerarchitektur	Funktionsweise der ersten mechanischen Rechner	Konzeptuell
	Funktionsweise von Rechnern mit Vakuumröhren	Konzeptuell
	Relevanz von Transistoren für die Entwicklung von Rechnern	Deklarativ
CPU	Benennung und Funktion der Komponenten einer modernen CPU (Von-Neumann-Architektur)	Deklarativ
	Ablauf des Abrufen-Decodieren-Ausführen-Zyklus	Prozedural
Digitale Logik	Funktion von Transistoren	Deklarativ
	Aufbau und Funktion verschiedener Logikgatter, sowie ihre jeweiligen Wahrheitstabellen	Deklarativ
	Anwendung der Regeln der Booleschen Algebra	Konzeptuell
	Verwendung von Gattern für das Erstellen verschiedener Schaltkreise (Multiplexer, Decoder, Komparator, Shifter, Halbaddierer, Volladdierer, Latches, Flip Flops)	Konzeptuell
	Verwendungszweck der genannten Schaltkreise	Deklarativ
Hauptspeicher	Struktur von RAM/DRAM/(DDR-)SDRAM	Deklarativ
	Funktionsweise von Speicherzugriffen (Speicheradressen)	Konzeptuell
	Funktionsweise von Fehlerkorrekturcodes	Konzeptuell
	Funktionsweise des Cache-Speichers	Konzeptuell
Sekundärer Speicher	Arten verschiedener Sekundärspeicher und ihre Unterschiede	Deklarativ
	Funktionsweise von HDDs	Konzeptuell
	Funktionsweise von SSDs	Konzeptuell
Mathematik	Umgang mit verschiedenen Zahlensystemen (Binär, Hexadezimal)	Konzeptuell
	Umgang mit Gleitkommazahlen	Konzeptuell

4 Instruktions- und Game Design

Um die Entwicklung des Spiels starten zu können, wird zunächst eine genaue Planung der möglichen Features benötigt. Um die (Lern-)Ziele nicht zu verfehlen, ist es dabei notwendig, sich an der angefertigten Analyse zu orientieren und entsprechende Instruktionsstrategien sowie zugehörige Spielmechaniken zu wählen. Das erstellte Design Dokument befindet sich im Appendix 1.3 und soll in diesem Kapitel genauer erklärt und begründet werden. An dieser Stelle wurde entschieden, dass das Spiel *DE:LINT* heißen soll.

4.1 Zusammenfassung des Game Designs

Um dem Problem der hohen Abstraktion des Gebiets der Rechnerarchitektur aus dem Weg zu gehen, soll es dem Spieler ermöglicht werden, den Rechner von innen zu betrachten. Da einige PC-Komponenten in Realität relativ klein sind, soll das Spiel nicht maßstabsgetreu gehalten werden. Stattdessen soll der Spieler in einer geschrumpften Form über das Motherboard beziehungsweise durch modellhafte Darstellungen der PC-Komponenten navigieren können. Dies vermeidet nicht nur Abstraktion, sondern ermöglicht dem Spieler sich an den Rechner als Umgebung zu gewöhnen und somit den Aufbau des PCs zu verinnerlichen. Um dennoch ein Gefühl für die Interaktion des Spielercharakters mit der Spielumgebung zu vermitteln, soll das Spiel in der Third-Person-Perspektive gespielt werden.

Auf der Reise durch den PC benötigt der Spieler Feedback und Orientierungshilfe (siehe *Cognitive Apprenticeship* in Tabelle 2.1), wofür ein Hilfscharakter als Lehrer den Spieler begleitet. Um ein authentisches Umfeld, sowie einen sinnvollen Kontext zu bieten (siehe ebenfalls *Cognitive Apprenticeship* und *Lepper's Instructional Design Principles for Intrinsic Motivation* in Tabelle 2.1), soll das Spiel anhand einer kurzen Geschichte eingeleitet werden. Die Geschichte soll dem Spieler nahelegen, sich in einem kaputten PC mit den Komponenten auseinanderzusetzen, um deren Funktionsfähigkeiten wiederherzustellen und den PC vollständig von Staubflusen zu befreien (in Englisch: Lint). Auf Basis dieser Spielziele wurde entschieden, den Hilfscharakter ebenfalls als Staubflusen darzustellen und Linta zu nennen.

Zu Beginn des Spiels soll Linta dem Spieler außerdem das Grundwissen über den PC-Aufbau (siehe Tabelle 3.2) vermitteln, da dieses dem Spieler erst ermöglicht, sich im Rechner bzw. auf dem Motherboard zurechtzufinden. Dies kann durch ein erstes Minispiel geschehen.

Die Wiederherstellung der Funktionsfähigkeit einzelner Komponenten soll im Anschluss die restlichen Lernaufgaben für den Spieler beinhalten. Indem der Spieler Prozesse und Funktionsweisen nachstellt, kann die Komponente ihre Funktion wieder aufnehmen. Diese Spielziele lassen sich ebenfalls als Minispiele in das Hauptspiel integrieren und können so auch in einem gesonderten Design behandelt werden.

Sobald alle Komponenten repariert wurden und der PC komplett gesäubert wurde, kann die Geschichte wieder anknüpfen und dem Spieler zeigen, dass der kaputte Rechner wieder lauffähig ist.

4.2 Auswahl der Instruktionsstrategien

Das gegebene Hauptdesign bietet bereits eine große Menge an benötigten, und möglichen Features, weswegen nur eine gewisse Anzahl der genannten Minispiele im Rahmen dieser Arbeit umgesetzt werden kann. Ein Minispiel zur Vermittlung des Grundwissens ist elementar. Zusätzlich dazu soll die CPU etwas genauer betrachtet werden und demnach ein Minispiel zur Vermittlung des Abrufen-Decodieren-Ausführen-Zyklus hinzugefügt werden.

4.2.1 Vermittlung des Grundwissens

Der typische Aufbau eines modernen Rechners sowie die Benennung, das Aussehen und die Funktion der üblichen PC-Komponenten ist deklaratives Wissen. Eine übliche Spielmechanik um diese Art von Wissen zu vermitteln ist das Sortieren und/oder Zuordnen der benötigten Fakten. Deshalb soll das Grundwissen-Minispiel auf genau dieser Mechanik aufbauen. Zunächst soll der Spieler für eine gezeigte Komponente die korrekte Bezeichnung und zugehörige Funktion auswählen und kann im Anschluss die Komponente auf dem korrekten Steckplatz auf dem Motherboard platzieren. Das nötige Wissen um diese Aufgabe lösen zu können soll Linta im Voraus zur Verfügung stellen. Eine Wiederholung dieses Wissen findet außerdem dadurch statt, dass der Spieler im Anschluss über das Motherboard laufen kann und die Komponenten genauer aus einem anderen Blickwinkel betrachten kann.

4.2.2 Vermittlung des Abrufen-Decodieren-Ausführen-Zyklus

Auch wenn der Abrufen-Decodieren-Ausführen-Zyklus üblicherweise keineswegs vom Menschen ausgeführt wird, sollen dem Spieler die einzelnen Schritte als Aufgabe übergeben werden. Für die Vermittlung von prozeduralem Wissen, wozu dieser Zyklus zählt (siehe Tabelle 3.2), wird empfohlen, mit einem Überblick über die vollständige Prozedur zu beginnen. Dies geschieht durch eine erneute Erklärung von Linta. Anschließend soll für jeden einzelnen Schritt der Prozedur betrachtet werden, wie und weshalb dieser

ausgeführt wird. Indem der Spieler unter Anleitung des Spiels die Schritte eigenständig ausführt, kann er die Fragen des Wie und Warum selbst beantworten.

Dazu soll die Minispielumgebung die wichtigsten Komponenten und ihre Funktionen für den Zyklus zur Verfügung stellen. Ein Instruktionsregister sowie ein inkrementierbarer Programmzähler werden benötigt. Außerdem braucht der Spieler Zugriff auf die Instruktionen und gegebenenfalls Daten im Hauptspeicher. Diese werden über einen Datenbus bereitgestellt, welcher erlauben muss, mit gegebenen Adressen Daten im Speicher auszulesen und zu schreiben. Zusätzlich benötigt die Umgebung eine ALU, welche zwei gegebene Werte addieren kann und mindestens ein zusätzliches Register, um Ergebnisse zwischenzuspeichern. Der Spieler soll mithilfe dieser Komponenten den Zyklus für MOV und ADD Befehle durchlaufen können.

5 Entwicklung und Implementierung

DE:LINT soll in einem Zeitraum von einem Monat entwickelt und fertiggestellt werden. Unter dieser Voraussetzung würde ein komplexes Vorgehensmodell für die Softwareentwicklung wie Scrum oder RUP (Rational Unified Process) zu viel Overhead mit sich bringen und somit den Prozess stark verzögern. Stattdessen ist der Entwicklungsprozess geleitet von verschiedenen Prozessen und Richtlinien aus Feature-Driven Development, Extreme Programming und Kanban, die auf Ein-Personen-Teams ohne Stakeholder und/oder Kunden anwendbar sind.

Wie in den vorangegangenen Kapiteln schon erwähnt wurde, enthält das bisher entworfene Spielkonzept mehr Features und Details, als im Rahmen dieser Arbeit umsetzbar ist. Deshalb ist es besonders wichtig zu priorisieren und zu entscheiden, welche Features umgesetzt werden sollten. Sobald die Feature-Liste zusammengestellt wurde, kann die Softwarearchitektur geplant werden. Damit kann für die Weiterentwicklung nach Abschluss dieser Arbeit gewährleistet werden, dass der Code erweiterbar ist. Anhand der geplanten Architektur kann ein Backlog für das Kanban Board erstellt werden, welches im Anschluss dazu dient, den Entwicklungsprozess zu begleiten. Einzelne Features werden in diesem Prozess in Hinsicht auf eine Softwarearchitektur nicht vollständig geplant, sondern im Sinne von Extreme Programming in ihrer simpelsten Form umgesetzt und durch Iterationen in eine finale Form gebracht. Sobald das erstellte Backlog abgearbeitet ist, kann die Implementierung des Lernspiels geplant und durchgeführt werden.

5.1 Aufgabenformulierung und Priorisierung

Tabelle 5.1 bietet einen Überblick über mögliche Features für DE:LINT, ihre jeweilige Priosierung nach MoSCoW und die entsprechende Entscheidung, ob das Feature umgesetzt werden soll. Ein Großteil der Features ist Teil des Designs und hat deshalb eine hohe Priorität. Ein kleinerer Teil der Features stammt aus dem Grundaufbau eines Spiels, wie zum Beispiel das Haupt- und Pausemenü, sowie ein Speicher- und Ladesystem.

Viele Spieltypen erfordern kein Speicher- und Ladesystem; dazu zählen insbesondere kleinere Spiele. Da DE:LINT nach Abschluss dieser Arbeit jedoch erweitert werden soll und zu einem größeren Spiel heranwachsen kann, wäre es für die Spieler ein hilfreiches Feature. Ein Speicher- und Ladesystem ermöglicht dem Spieler, die Lernprozesse

wie gewünscht aufzuteilen und bietet somit ein Gefühl von Autonomie (siehe *Self-Determination Theory* und *Distributed Practice* in Tabelle 2.1). Aufgrund dessen wird Speichern und Laden als „should have“ eingestuft. Die technische Umsetzung eines solchen Systems sollte zu Beginn der Entwicklung stattfinden. Jegliche zu speichernde Daten müssen eine entsprechende Form haben, weswegen zugehörige Features auf der Umsetzung des Speicher- und Ladesystems aufbauen. Aufgrund dieser Abhängigkeit wird das entsprechende Feature in die Planung dieser Arbeit aufgenommen.

Alle anderen Features mit „should have“ oder niederer Priorisierung sind für die Beantwortung der Fragestellungen nicht notwendig und werden deshalb in dieser Arbeit nicht umgesetzt.

Tabelle 5.1: *Mögliche Features und ihre Priorisierung nach MoSCoW*

Feature	**Priorität**	**Umsetzung**
Hauptmenü	Must have	Ja
Pause Menü	Must have	Ja
Speichern und Laden	Should have	Ja
Ende Menü	Should have	Nein
Spieler- und Kamerabewegung	Must have	Ja
Begehbare PC Umgebung	Must have	Ja
PC-Säuberung	Must have	Ja
Dialoge	Must have	Ja
Einleitungsgeschichte	Must have	Ja
Quests	Must have	Ja
Umgebungsinteraktionen	Must have	Ja
Mini-Spiel Grundwissen	Must have	Ja
Mini-Spiel Befehlszyklus	Must have	Ja
Andere Mini-Spiele	Could have	Nein
Mini-Map/Übersichtsskarte	Could have	Nein
Sound	Should have	Nein
Optionen Menü	Could have	Nein
Tutorial	Could have	Nein
Ladebildschirm	Could have	Nein
Charaktererstellung/-anpassung	Could have	Nein

5.2 Technisches Design

Anhand der gegebenen Feature-Liste kann nun das technische Design und in diesem Sinne eine grobe Softwarearchitektur erstellt werden. Zu diesem Schritt gehört außerdem die Formulierung eines Programmierstils und die Planung einer Dateistruktur in Unity.

5.2.1 Spielarchitektur

Abbildung 5.1 zeigt die wichtigsten Features von DE:LINT und ihre Abhängigkeiten zueinander. Durch die gezeigte Architektur wird eine Hierarchie aufgebaut, die Zugriffe auf Daten und Methoden ausschließlich von oben nach unten ermöglichen. Ziel dieses Designs soll sein, dass Features alleinstehend entwickelt und getestet werden können, sodass ihre Implementierung zu späteren Zeitpunkten variabel ist. Dass Features in der Hierarchie durch Pfeile miteinander verknüpft sind, bedeutet jedoch nicht, dass auch im Code eine Verknüpfung notwendig ist. Es bedeutet vielmehr, dass das hierarchisch höhere Feature eine Kategorie für die Kind-Features bietet und somit die Erlaubnis hat, diese zu manipulieren.

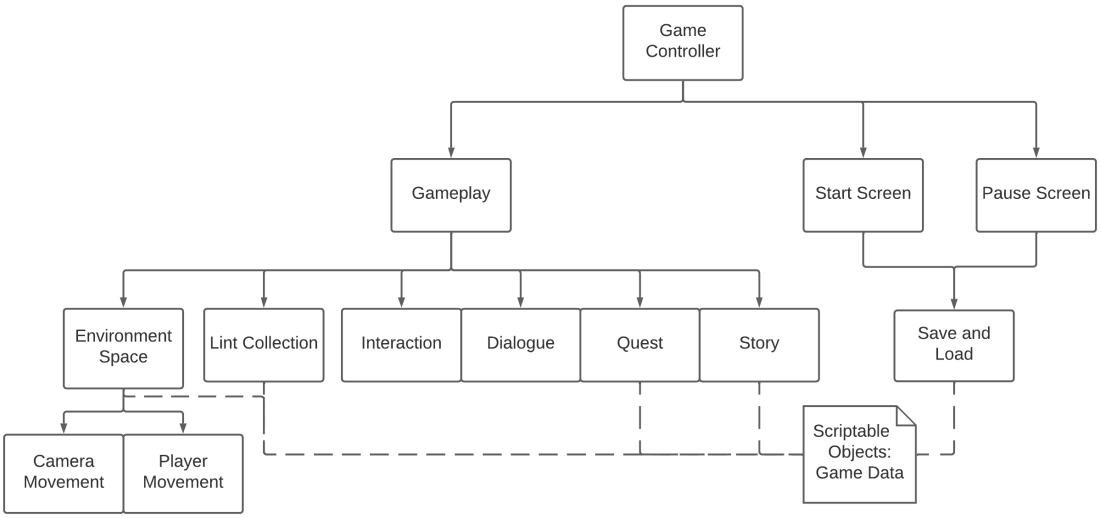


Abbildung 5.1: *Grobe Spielarchitektur; Quelle: Eigene Darstellung*

Ganz oben in der Hierarchie befindet sich dabei der GameController. Dieser ist dafür zuständig die Basisfunktionen des Spiels zu schalten, also das Hauptmenü, Gameplay, Pause Menü und Beenden des Spiels. Da diese Basisfunktionen jeweils einen gesonderten State darstellen und nicht gleichzeitig aktiv sein können, bietet sich für den GameController eine State Machine an. Eine entsprechende Darstellung der States und möglichen Transitions ist in Abbildung 5.2 zu sehen. Im Rahmen der Gesamtarchitektur ist der GameController das einzige Feature, welches Zugriff auf alle anderen Features hat. Jedoch kann kein anderes Feature auf den GameController zugreifen, oder diesen manipulieren.

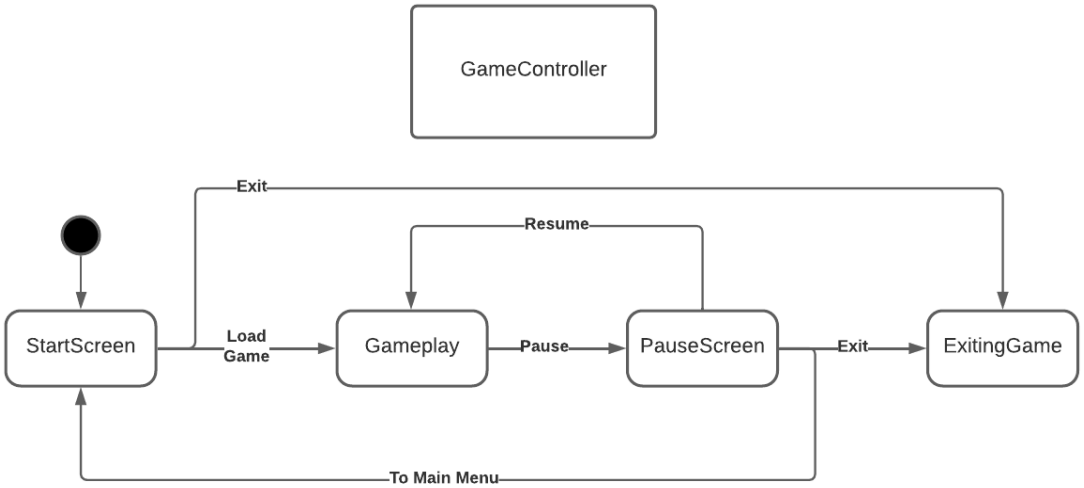


Abbildung 5.2: *Diagramm des Game Controllers mit möglichen State Transition; Quelle: Eigene Darstellung*

Wie zuvor schon erwähnt ist die Entwicklung eines Speicher- und Ladesystem einer der ersten Schritte, die im Entwicklungsprozess stattfinden sollte. Um zu vermeiden, dass Features mit relevanten Daten eng an dieses System gekoppelt werden und somit die Erweiterbarkeit des Spiels beeinträchtigt wird, soll das Speicher- und Ladesystem keinen Zugriff auf diese Features haben. Stattdessen sollen jegliche zu speichernden Daten in ScriptableObjects geschrieben werden und im Ladeprozess wieder ausgelesen werden. Somit schreibt das Speicher- und Ladesystem einen Formfaktor für die zu speichernden Daten vor, hat jedoch keine Verknüpfung mit den zugehörigen Features. Lediglich das Start- und Pausemenü haben Zugriff auf die Speicher- und die Ladefunktion des Systems, wie in Abbildung 5.1 zu sehen ist.

Die Mini-Spiel-Features und die begehbare Motherboard-Umgebung sind in dieser Architekturdarstellung als EnvironmentSpace zusammengefasst. Ziel dieser Kombination ist es, im Gameplay einen Wechsel zwischen verschiedenen Umgebungen zu ermöglichen. Nach Design ist geplant, von einer Start-Umgebung für die Einleitungsgeschichte in das Grundlagen-Mini-Spiel zu wechseln. Im Anschluss soll das Motherboard betreten und von dort aus ein Umgebungswechsel in die Mini-Spiel-Umgebung für den Befehlszyklus ermöglicht werden. Je nach Umgebung verhält sich die Kamera- und Spielerbewegung anders, da für die Mainboard-Umgebung und das CPU-Zyklus-Spiel ein Third-Person-Controller angedacht ist und für das Grundlagen-Minispiel keine Spielerbewegung benötigt wird. Die anderen Gameplay-Features wie Interaktionen oder das Questsystem sollen jedoch unabhängig von der aktuellen Umgebung verwendbar sein.

5.2.2 Programmierstil

Der Programmierstil für die Entwicklung von DE:LINT basiert hauptsächlich auf den Codekonventionen für C#[1] sowie üblichen Entwurfsmustern und Programmierprinzipien für objektorientierte Programmierung[2]. Jegliche Abweichungen von und Zusätze zu diesem Standard sollen im Folgenden ausgeführt und begründet werden.

Viele Abweichungen vom C#-Standard stammen aus üblichen Java-Code-Konventionen[3]. Dazu gehört die Benennung von `public`, `private`, `internal` und `static` Feldern. Dies ist weniger eine logisch begründbare Entscheidung, sondern vielmehr eine persönliche Präferenz für die Lesbarkeit des Codes.

Zusätzlich wird entschieden, dass ScriptableObject Klassen `_SO` als Suffix erhalten und GameEvents das Präfix `on` verwenden sollen.

Wie im vorangehenden Kapitel schon erwähnt wurde, sollen Features alleinstehend entwickelt werden und möglichst wenig Abhängigkeiten zu anderen Features haben. Um dies zu gewährleisten, ist ein ScriptableObject- und event-basierter Ansatz für DE:LINT sinnvoll. Ein solcher Ansatz wird im Unite Austin 2017 Talk von Ryan Hipple [5] beschrieben und soll in angemessener Form für das Projekt eingesetzt werden.

5.2.3 Ordnerstruktur

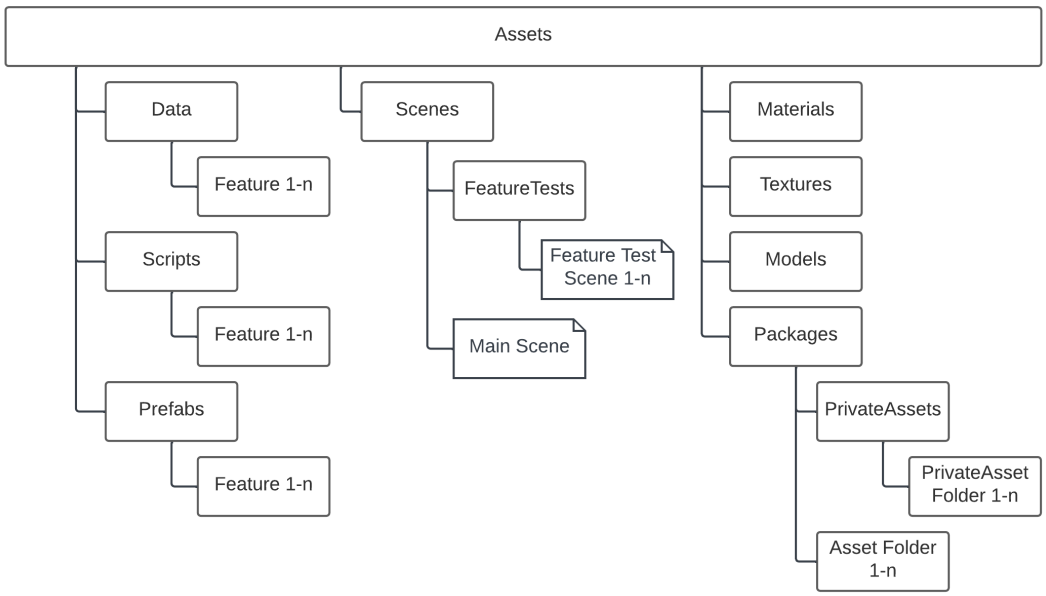


Abbildung 5.3: *Darstellung der geplanten Ordnerstruktur; Quelle: Eigene Darstellung*

[1] Eine Beschreibung der C# Codekonventionen ist unter `https://docs.microsoft.com/de-de/dotnet/csharp/fundamentals/coding-style/coding-conventions` auffindbar.

[2] Eine gute Zusammenfassung solcher Prinzipien bietet das Buch *Clean Code* von Robert C. Martin [9]

[3] Java Codekonventionen sind unter `https://google.github.io/styleguide/javaguide.html` einsehbar.

Unity schreibt Spieleentwicklern keine konkrete Ordnerstruktur vor, weshalb Projekte schnell unübersichtlich werden können. Um dies zu vermeiden, wird für die Entwicklung eine Ordnerstrukur nach Abbildung 5.3 vorgeschrieben. Diese unterstützt die geplante Trennung einzelner Features, indem für jedes Feature Ordner für Daten, Skripte und Prefabs erstellt werden und eine Test-Szene angelegt wird.

Hierbei gilt auch zu beachten, dass für die Entwicklung gegebenenfalls bezahlte Assets (beispielsweise aus dem Unity Asset Store) verwendet werden. Diese dürfen nicht veröffentlicht werden, weswegen der entsprechende Ordner nicht mit im öffentlichen Quellcode in GitHub liegen darf. Stattdessen wird hierfür ein gesonderter Ordner `PrivateAssets` angelegt, welcher in ein privates Git Submodule ausgelagert werden kann.

5.3 Entwicklungsbericht

Mit Fertigstellung des technischen Designs ist es möglich, das Backlog im Kanban-Board zu füllen und entsprechend zu sortieren. Ein Export des ursprünglichen Backlogs befindet sich im Appendix 1.4. Im Folgenden wird der Entwicklungsprozess einzelner Features mit besonderem Augenmerk auf aufgetretene Probleme und entsprechende Lösungen dargelegt. Jeglicher beschriebener Code ist im verlinkten Github Repository des Appendix 1.1 auffindbar.

5.3.1 Basiscode

Im ersten Schritt der Entwicklung wurde ein leeres Unity-3D-Projekt angelegt und mit dem für die geplante Scriptable-Objekt-Architektur benötigten Basiscode aus vorangehenden Projekten versehen. Dazu gehören hauptsächlich die Variablen und Events, wie sie im zuvor erwähnten Unite-Talk von Ryan Hipple [5] beschrieben sind. Diese beiden Datenstrukturen (`DataStructures`) wurden im Laufe des Projekts immer wieder verwendet und spielten für den Entwicklugsprozess eine große Rolle.

Da zu diesem Zeitpunkt noch nicht absehbar war, welche Variablen-Typen benötigt werden, wurde zunächst eine `AbstractVariable` angelegt. Zu späteren Zeitpunkten konnten verschiedene Variablen-Typen wie `BoolVariable` und `IntVariable` von dieser erben. Die entsprechende Umsetzung ist in Listing 5.1 zu sehen.

```
1 [Serializable]
2 public abstract class AbstractVariable<T> : ScriptableObject
3 {
4     public T value;
5 }
```

Listing 5.1: *AbstractVariable Klasse*

Für die Umsetzung der Events wurden Action Delegates verwendet; Listing 5.2 zeigt den Code für die erstellte `GameEvent` Klasse. Im Sinne einer Event-Driven-Architektur

stellt diese Klasse durch die enthaltenen `Action listeners` den Event Broker dar. Andere Klassen können diese Events als ScriptableObjects erhalten und durch die Verwendung der `Raise` Methode sowohl als Event Producer agieren, als auch eigene Methoden als EventConsumer registrieren.

```
[CreateAssetMenu(fileName = "GameEvent", menuName = "Data/Events/GameEvent")]
public class GameEvent : ScriptableObject
{
    private Action listeners;

    public void Raise()
    {
        listeners?.Invoke();
    }

    public void RegisterListener(Action listener)
    {
        listeners += listener;
    }

    public void UnregisterListener(Action listener)
    {
        listeners -= listener;
    }
}
```

Listing 5.2: *GameEvent Klasse*

Für diese Klasse wurde außerdem ein Editorskript erstellt, welches ermöglicht hat, die `Raise` Methode während des Spiels manuell im Editor auszuführen. Dies hat insbesondere bei späteren Tests des Spiels geholfen, da so bestimmte Schritte übersprungen werden konnten und nicht in jedem Testlauf das komplette Spiel gespielt werden musste.

Im Verlauf der Entwicklung kam es häufig zu der Überlegung, eine weitere Klasse für Events mit Parameterübergabe hinzuzufügen. In jedem dieser Fälle war es jedoch möglich, die Übergabe der gewünschten Daten anhand von ScriptableObjects zu programmieren, weswegen dies schlussendlich nicht umgesetzt wurde.

5.3.2 StateMachine und GameController

Unter Verwendung der vorhandenen Events konnte nun der GameController, wie in Abbildung 5.2 dargestellt, als StateMachine programmiert werden. Um die Wiederverwendbarkeit der StateMachine zu garantieren, wurde diese in einem eigenen Namespace entwickelt und im Anschluss vom GameController implementiert. Eine Darstellung der endgültigen Umsetzung der StateMachine und des darauf aufbauenden GameControllers ist in Abbildung 5.4 zu sehen. Wie in 5.2.1 geplant, hat der

GameController Zugriff auf relevante Features, die tiefer in der Hierachie stehen, und somit die Befähigung diese Features in jeweiligen State-Wechseln zu verwalten.

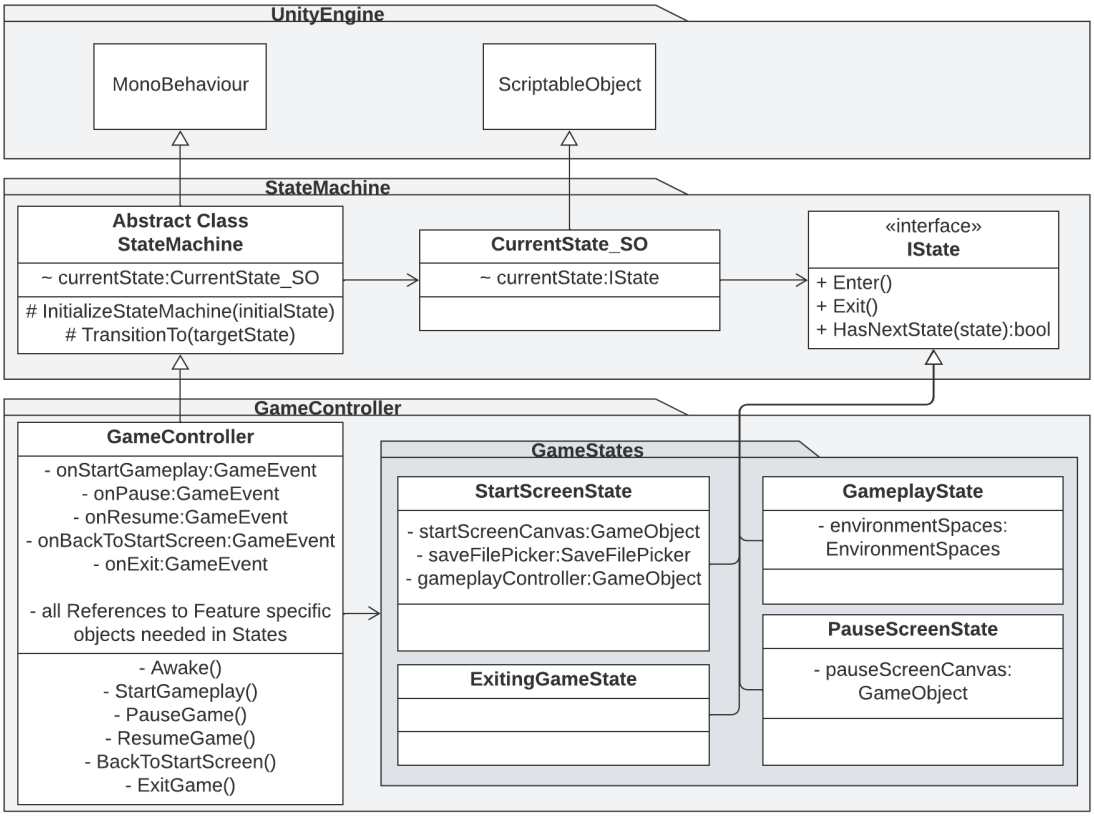


Abbildung 5.4: *Klassen-UML für die abstrakte StateMachine und die Implementierung durch den GameController; Quelle: Eigene Darstellung*

Nicht dargestellt ist hierbei die Beziehung zwischen den States. In der Methode `HasNextState(IState state)` muss für jeden GameState programmiert sein, ob eine Transition zum gegebenen State möglich ist. Eine beispielhafte Implementierung für diese Methode ist im Listing 5.3 zu sehen.

```
private readonly List<Type> nextStates = new List<Type>
{
    typeof(GameplayState), typeof(StartScreenState), typeof(ExitingGameState)
};

public bool HasNextState(IState nextState)
{
    return nextStates.Contains(nextState.GetType());
}
```

Listing 5.3: *HasNextState() Implementierung aus PauseScreenState*

5.3.3 Speicher- und Ladesystem

Im nächsten Entwicklungsschritt wurde das Speicher- und Ladesystem vorbereitet. Aufgrund geringen Vorwissens im Bereich von Datenserialisierung und -sicherung wurde hierfür eine Anleitung von *AmalgamateLabs* [12] zur Hilfe herangezogen. Der finale `SaveAndLoadController` enthält die Identifikationsnummer des derzeitig gültigen Speicherstands (`saveFileNumber`), eine Liste von ScriptableObjects mit den zu speichernden Daten und vier Hauptfunktionen: Speichern, Laden, Speicherstand-Vorlage erstellen und aus Vorlage laden. Diese sollen im Folgenden einzeln erklärt werden.

Im Speicherprozess wird für jedes referenzierte ScriptableObject eine gleichnamige .dat Datei im Ordner des aktuell aktiven Speicherstands angelegt oder überschrieben. Die Daten im ScriptableObject werden zunächst in ein Json-Format umgewandelt. Anschließend wird ein `BinaryFormatter` verwendet, um eine mögliche Bearbeitung der Speicherdateien durch den Spieler zu verhindern. Dieser Formatter serialisiert den gegebenen Json-String in ein Binärformat und schreibt dies anschließend in die entsprechende Datei.

Um den derzeitig aktiven Speicherstand zu laden, muss erneut über jedes referenzierte ScriptableObject iteriert werden. Es wird zunächst eine gleichnamige .dat Datei aus dem Speicherstand-Ordner geöffnet und ihr Inhalt anschließend durch einen `BinaryFormatter` deserialisiert, sodass die zuvor gespeicherten Json-Daten wieder zur Verfügung stehen. Mithilfe der `JsonUtility` können diese Daten wieder zurück in das zugehörige ScriptableObject geschrieben werden. Voraussetzung hierfür ist, dass die zuvor gespeicherten Daten serialisierbar waren. Dies stellte an einigen Stellen im Projekt eine Herausforderung dar, da beispielsweise Verlinkungen zwischen ScriptableObjects nicht geladen werden konnten. Somit musste für alle zu speichernden Daten auf diese Anforderung geachtet werden und gegebenenfalls einzelne Werte ausgelagert werden.

Eine Speicherstand-Vorlage wurde hinzugefügt, um beim Erstellen eines neuen Speicherstands jegliche Daten auf ihren ursprünglichen Zustand zurückzusetzen. Die Vorlage wird genau einmal bei erstmaligem Start des Spiels mit der gleicher Methodik wie im Speicherprozess angelegt. Wird zu einem späteren Zeitpunkt ein neuer Spielstand erstellt, so wird der Inhalt des Template-Ordners in einen neuen zugehörigen Ordner kopiert und die ursprünglichen Daten geladen. Dadurch wird sichergestellt, dass neue Spielstände nicht den Fortschritt aus zuvor geöffneten Spielständen übernehmen.

5.3.4 Spieler- und Kamerasteuerung

Die Entwicklung eines PlayerControllers ist aufwändig und benötigt viel Zeit, um Feinheiten wie die Sprunghöhe, Laufgeschwindigkeit und den Umgang mit verschie-

denen Hindernissen korrekt einzustellen. Unity stellt im Asset Store selbst einen ThirdPersonController zur Verfügung[4], weswegen keine eigene Entwicklung des Features stattfinden sollte. Der verwendete Controller enthält einen vollständig animierten Humanoiden (siehe Abbildung 5.5), dessen Aussehen für DE:LINT beibehalten wurde.

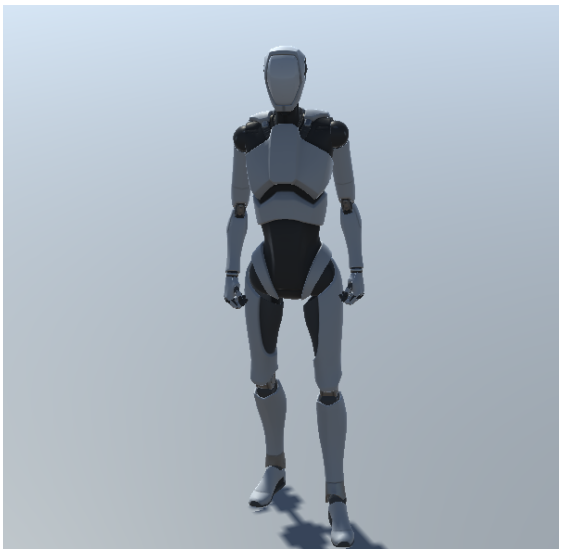


Abbildung 5.5: *Der humanoide Spielercharakter aus dem Unity ThirdPersonController; Quelle: Eigene Darstellung*

Aufgrund des von Unity gewählten Ansatzes zur Verarbeitung von Spieler-Input kam es im Laufe des Spiels zu unterschiedlichen Problemen. Das zuvor angelegte Input-Handling musste angepasst werden, da bisher GameEvents verwendet wurden. Der heruntergeladene Input Manager hingegen verwendete ein Messaging-System, welches sich nur durch zusätzlichen Code mit den GameEvents kombinieren ließ. Außerdem musste erneut evaluiert werden, welche GameObjects für das Handling von Input zuständig sein sollen, da mehrere gleichzeitig aktive Handler zu Fehlern führten. Da im Pause-Menü oder im Start-Menü jedoch kein ThirdPersonController aktiv ist, kann dieser nicht immer für das InputHandling zur Verfügung stehen. Aus diesem Grund wurde für jeden GameState ein passender InputHandler entwickelt. Der GameController wurde danach dafür eingesetzt, die zugehörigen Objekte jeweils aktiv oder inaktiv zu schalten.

Mit Aktivierung des ThirdPersonControllers wird der Mauscursor ausgeblendet, sodass Benutzeroberflächen nicht mehr per Maus bedienbar waren. Dieses Problem musste gelöst werden, sodass der Cursor bei einer Pausierung des Spiels oder zu Beginn eines Dialogs aktiviert und die mausgesteuerte Kamerabewegung deaktiviert wurde. Dafür wurde für begehbare Spielumgebungen die Klasse `ThirdPersonEnvironmentSpace`

[4]Zu finden unter `https://assetstore.unity.com/packages/essentials/starter-assets-third-person-character-controller-196526`

mit den zugehörigen Methoden `DeactivateThirdPersonInput` und `ActivateThirdPersonInput` angelegt. Diese Methoden werden vom GameController oder als Reaktion auf ein GameEvent jeweils zu Dialogbeginn/-ende ausgeführt. Listing 5.4 zeigt den verwendeten Code.

```
internal void DeactivateThirdPersonInput()
{
    thirdPersonInput.DeactivateInput();
    gameplayInputManager.cursorLocked = false;
    gameplayInputManager.cursorInputForLook = false;
    gameplayInputManager.ResetCursorState();
}

internal void ActivateThirdPersonInput()
{
    thirdPersonInput.ActivateInput();
    gameplayInputManager.cursorLocked = true;
    gameplayInputManager.cursorInputForLook = true;
    gameplayInputManager.ResetCursorState();
}
```

Listing 5.4: *Methoden aus der Klasse ThirdPersonEnvironmentSpace*

5.3.5 Dialoge, Interaktionen und Quests

Dialogsystem

Für die Umsetzung von Dialogen wurde zunächst betrachtet, welche Anforderungen diese erfüllen sollen. Um den Spielern weiterhin ein Gefühl von Autonomie zu vermitteln, wurde ein Dialogsystem mit verschiedenen Auswahlmöglichkeiten für den Spieler umgesetzt. Abbildung 5.6 zeigt einen Beispieldialog mit drei Auswahlmöglichkeiten.

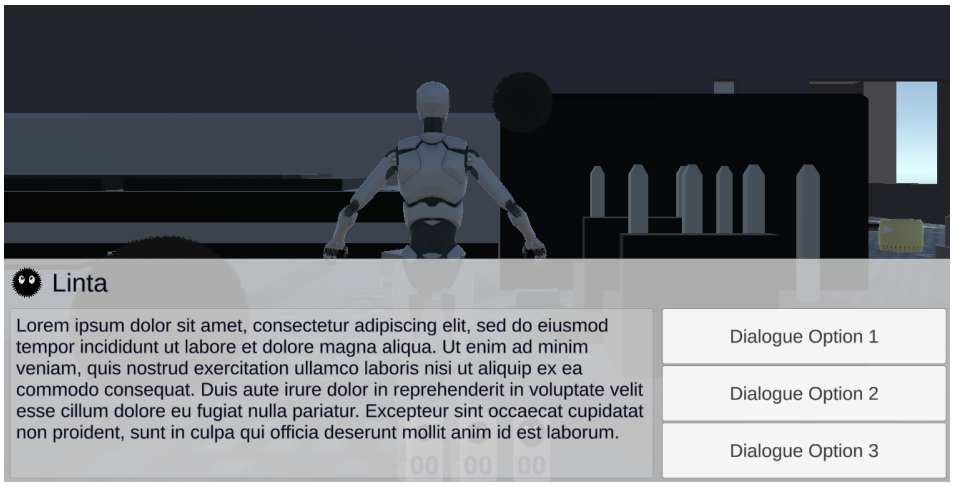


Abbildung 5.6: *Beispieldialog aus DE:LINT; Quelle: Eigene Darstellung*

Gemäß der Anforderungsanalyse müssen die Dialogoptionen verschiedene Funktionen erfüllen können: die nächste zugehörige Dialognachricht anzeigen, oder den Dialog anhand eines GameEvents beenden. Abbildung 5.7 zeigt die finale Umsetzung anhand eines Klassen-UML. Da die anpassbaren Elemente eines Dialogs hierbei ScriptableObjects sind, ist es jederzeit möglich neue Dialoge hinzuzufügen, oder schon erstelle Dialoge anzupassen, ohne dabei den Code verändern zu müssen.

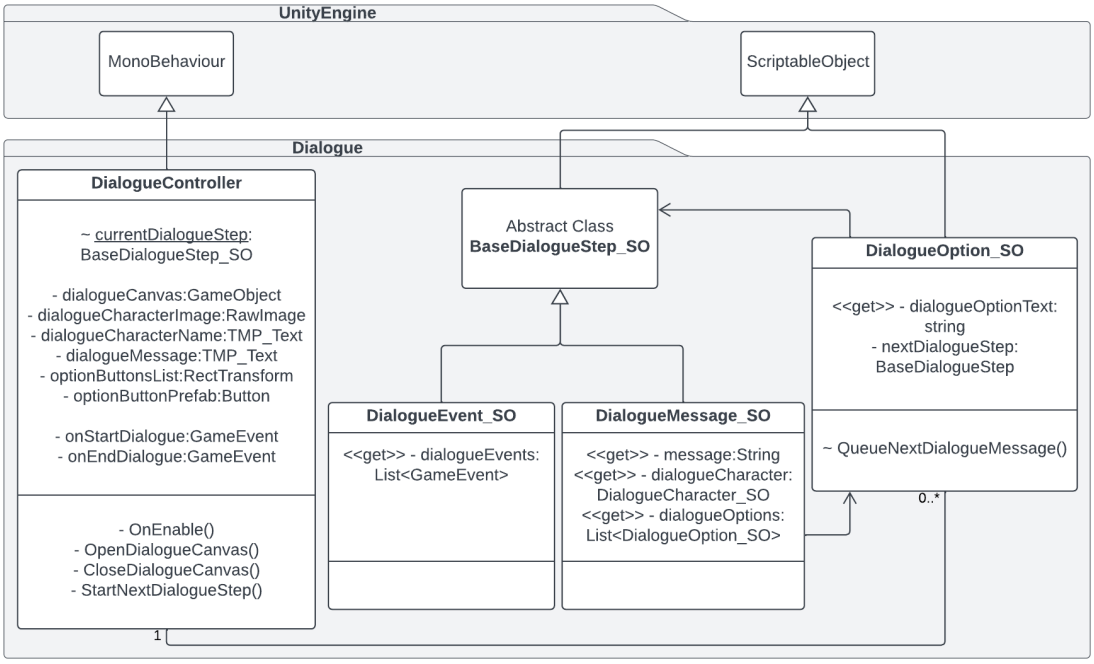


Abbildung 5.7: *Klassen-UML für das Dialog Namespace; Quelle: Eigene Darstellung*

Interaktionssystem

Ein nächster Schritt war die Entwicklung des Interaktionssystems. Als Funktion dieses Systems soll dem Spieler ermöglicht werden, Objekte in einer Third-Person-Spielumgebung zu bewegen, Dialoge zu starten oder in eine andere Umgebung bzw. in ein Minispiel zu wechseln. Um die Interaktionen an den ThirdPersonController anzubinden, wurden andere Spiele als Beispiel herangezogen. Insbesondere das Spiel *Genshin Impact*[5] hat hier als Inspiration gedient. In diesem werden alle Interaktionsoptionen auf dem HUD (Head-up-Display) in einer Liste angezeigt. Durch Hoch- und Runterscrollen mit der Maus ist es dem Spieler möglich, die Listenelemente nach unten oder oben zu verschieben. Wird der Interagieren-Knopf (F) auf der Tastatur bedient, so wird die oberste Interaktionsoption ausgeführt. Abbildung 5.8 zeigt die beschriebene Umsetzung aus *Genshin Impact*.

[5]Siehe `https://genshin.hoyoverse.com/`

Abbildung 5.8: *Interaktions-HUD in Genshin Impact; Quelle: Eigene Darstellung*

Für eine entsprechende Umsetzung in DE:LINT wurde `InteractableObjectsManager` geschrieben, welcher die Liste der `interactableObjects` in einem ScriptableObject verwaltet. Listing 5.5 zeigt den entsprechenden Code.

```
public class InteractableObjectsManager : MonoBehaviour
{
    [SerializeField] private GameEvent onScrollUp;
    [SerializeField] private GameEvent onScrollDown;
    [SerializeField] private GameEvent onInteract;
    [SerializeField] private InteractableObjects_SO interactableObjects;

    private void Awake()
    {
        interactableObjects.Reset();

        onScrollUp.RegisterListener(interactableObjects.ShiftRight);
        onScrollDown.RegisterListener(interactableObjects.ShiftLeft);
        onInteract.RegisterListener(Interact);
    }

        private void Interact()
    {
        if (interactableObjects.IOList.Count > 0)
        {
            interactableObjects.IOList[0].Interact();
        }
    }
}
```

Listing 5.5: *InteractableObjectsManager Klasse*

Interagierbare Objekte befinden sich dabei in der begehbaren Spielumgebung. Trifft der Spielercharacter auf den zugehörigen Collider eines InteractableObjects, so fügt sich dieses Objekt zur IOList im ScriptableObject hinzu. Solange sich eines oder mehrere Elemente in der IOList befinden, wird ihr zugehöriger Interaktionstext auf dem HUD für den Spieler angezeigt. Entfernt sich der Spielercharakter aus dem Collider des interagierbaren Objekts, so entfernt sich dieses Objekt wieder aus der IOList.

Questsystem

Nach Fertigstellung der Dialog- und Interaktionssysteme konnte mit der Entwicklung eines Questsystems begonnen werden. In diesem sollte es möglich sein, begonnene Quests einzusehen und durch das Erledigen der beinhalteten Aufgaben die Quests abzuarbeiten. Eine Quest besteht damit im finalen System aus einer Folge einzelner Aufgaben. Der Abschluss einzelner Aufgaben wird durch GameEvents gekennzeichnet. Wird das zugehörige `onFinishTask` Event ausgeführt, so wird der nächste Quest-Schritt aktiviert, oder gegebenfalls die Quest beendet und aus dem Quest Log entfernt. Das Quest-Log zeigt den Titel der Quest und eine Beschreibung der derzeitigen Aufgabe für jede aktive Quest, wie in Abbildung 5.9 zu sehen ist. Es lässt sich mit dem Drücken der J-Taste (J für Journal) öffnen und schließen.

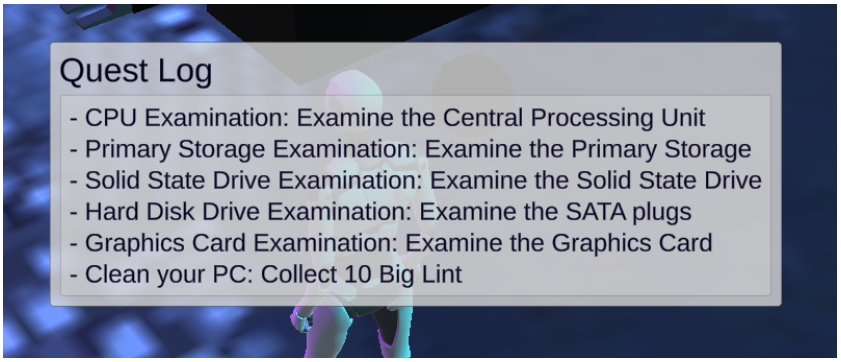


Abbildung 5.9: *Geöffnetes Questlog in DE:LINT; Quelle: Eigene Darstellung*

Für jede im Spiel enthaltene Quest musste im Speicher- und Ladesystem der Aktivitätsstatus sowie der Fortschritt gespeichert werden. Demnach erhält der SaveAndLoadController pro Quest eine BoolVariable als Aussage, ob die Quest aktiv ist und eine IntVariable mit der Identifikationsnummer des derzeitig aktiven Quest-Schritts. Im Ladeprozess des Spiels muss das Quest Log anhand dieser Variablen erneut befüllt werden.

5.3.6 Minispiele

Grundlagen Minispiel

Sortier- und Matchingspiele werden üblicherweise in 2D dargestellt, weswegen dies auch für das Grundlagen Minispiel umgesetzt wurde. Aus diesem Grund wurde für jede der Komponenten ein entsprechendes Bild verwendet, welches vom Spieler auf dem Motherboard platziert werden kann. Die Benutzeroberfläche für das gesamte Spiel sollte so simpel wie möglich gehalten werden, um spätere Anpassungen und Ausbesserungen zu vereinfachen. Abbildung 5.10 zeigt das Minispiel zu einem fortgeschrittenen Zeitpunkt, zu welchem bereits die CPU, HDD, SSD und der Arbeitsspeicher auf dem Motherboard platziert wurden.

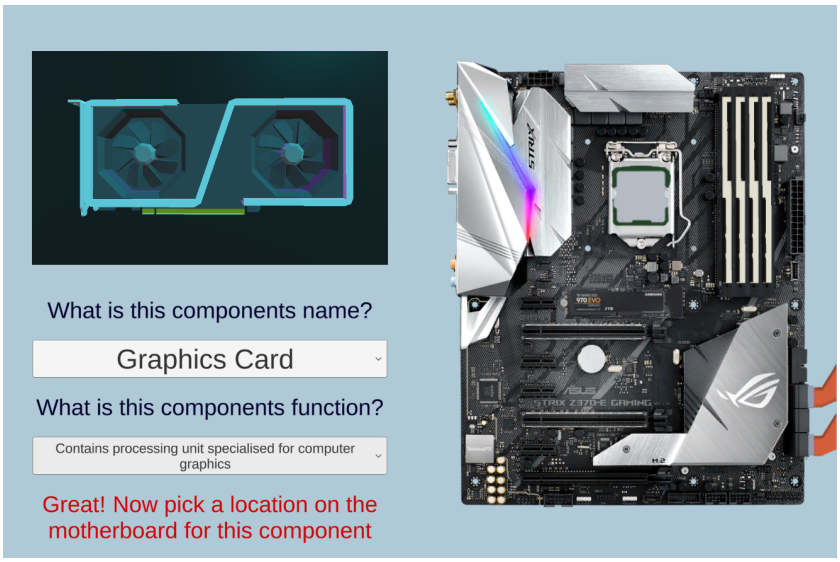


Abbildung 5.10: *Benutzeroberfläche des Grundlagen-Minispiels; Quelle: Eigene Darstellung*

Das hier gezeigte Fenster oben links bietet dem Spieler außerdem die Möglichkeit, die jeweils angezeigte Komponente anzuklicken und durch Bewegung der Maus nach Links oder Rechts zu drehen. Dadurch kann der Spieler zu einem frühen Zeitpunkt die Komponenten wie gewünscht betrachten und besser kennenlernen. Um dies umzusetzen, wurde eine extra Kamera in der Szene angelegt, welche direkt auf die jeweilige Komponente ausgerichtet ist. Diese Kamera schreibt ihren Output in eine RenderTexture[6], welche zur Laufzeit aktualisiert wird und somit die Drehung des Objekts darstellen kann.

[6]siehe zugehörige RenderTexture Unity Dokumentation `https://docs.unity3d.com/Manual/class-RenderTexture.html`

Befehlszyklus Minispiel

Für die Umsetzung des Befehlszyklus im Code war es naheliegend, die zuvor erstellte StateMachine wiederzuverwenden. Für jeden Schritt im Zyklus wurde ein State erstellt, und diese anhand der `HasNextState` Methode aneinandergereiht, wie in Abbildung 5.11 gezeigt wird.

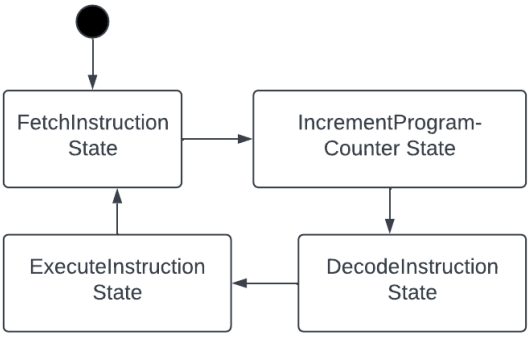


Abbildung 5.11: *Diagramm der Befehlszyklus-StateMachine; Quelle: Eigene Darstellung*

Dem Spieler wurde für dieses Spiel ein Inventar zur Verfügung gestellt, in welches genau ein Wert zum Transport aufgenommen werden kann. Außerdem wurde jede der benötigten Komponenten als interagierbares Objekt mit ihrer entsprechenden Interaktionslogik angelegt. Die ALU bietet beispielsweise die Interaktionsmöglichkeiten Wert A und Wert B einzufügen, sowie anschließend die Summe von Wert A und B zu berechnen und das Ergebnis wieder aufzunehmen.

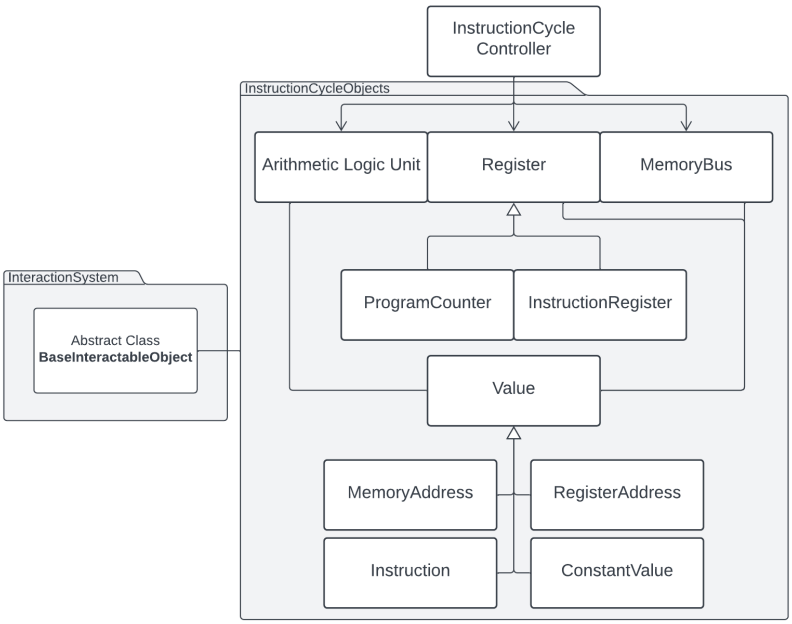


Abbildung 5.12: *Grobe Architektur des InstructionCycleController; Quelle: Eigene Darstellung*

Äquivalent zur Funktionsweise des `GameControllers`, hat der `InstructionCycleController` Kenntnis von allen zugehörigen Komponenten und steht somit in der Hierarchie über allen anderen Elementen des Minispiels. Eine grobe Architektur des finalen Minispiels ist in Abbildung 5.12 zu sehen.

5.3.7 Andere Features

Die bisher beschriebenen Umsetzungen bilden einen Hauptteil des geschriebenen Codes. Zusätzlich wurde für verschiedene andere Features nicht nur Code geschrieben, sondern auch Zeit im Unity Editor investiert. Ein Teil dieser Features sollen im Folgenden kurz erläutert werden.

User Interfaces

Unity bietet Entwicklern die Möglichkeit, User Interfaces anhand von GameObjects zusammenzustellen. Dies wurde in Verbindung mit dem Package TextMeshPro[7] verwendet. Da im Entwicklungsprozess jedoch wenig Wert auf die Ästhetik des Spiels gelegt wurde, sind das Pause- und Startmenü, sowie das HUD, sehr simpel gehalten. Auch in Hinblick auf den zugehörigen Code wurde hier wenig Zeit investiert, da viele Funktionen (zum Beispiel Button-Klicks) durch GameEvents gesteuert werden konnten.

PC-Säuberung

Für das Feature der PC-Säuberung wurden an verschiedenen erreichbaren Stellen in den Third-Person-Umgebungen (Mainboard, Befehlszyklus-Umgebung) Staub/Lint verschiedener Größen platziert, welche durch simples Hindurchlaufen vom Spieler eingesammelt werden können. Außerdem wurde eine Quest im Spiel hinzugefügt, alle vorhandenen Staubflusen einzusammeln. Hierfür musste für jeden platzierten Staubflusen im Speicher- und Ladesystem die Information angelegt werden, ob dieser bereits eingesammelt wurde oder nicht, sodass entfernte Flusen bei einem Neustart des Spiels nicht erneut auftauchten.

3D-Assets

Da der Fokus im Entwicklungsprozess auf der Programmierung des Spiels lag, wurden benötigte 3D-Modelle nicht selbst erstellt. Stattdessen wurde der Unity Asset Store und andere Anbieter für 3D-Assets zur Gestaltung der Spielumgebung durchsucht und passende Modelle im Spiel zusammengefügt. Eine Liste der verwendeten Assets befindet sich in der ReadMe des verlinkten GitHub Repositories (Appendix 1.1).

[7] Siehe TextMeshPro Unity Dokumentation `https://docs.unity3d.com/Manual/com.unity.textmeshpro.html`

Darüber hinaus wurde ein Helfer damit beauftragt, ein 3D-Modell für Linta zu erstellen. Dieses Modell wurde an den Spielercharakter angebunden und mit einer stetigen Auf- und Abbewegung versehen, um eine Schwebebewegung zu simulieren. Das für Linta verwendete Fellmaterial wurde zudem für die im Spiel verteilten Staubflusen wiederverwendet.

Story

Um alle Funktionen zusammenzubinden und dem Spieler einen roten Faden zu geben, wurde außerdem ein `StoryController` entwickelt. Dieser ist dafür zuständig, verschiedene Erklärungsdialoge einzuleiten, Minispiele zu starten, Quests zu aktivieren oder Animation/Veränderungen in der Spielumgebung hervorzurufen. Der erreichte Fortschritt in der erstellten Story musste auch in den `SaveAndLoadController` aufgenommen werden.

5.4 Vorbereitung auf die Implementierung

Mit Abschluss aller Features wurde das gesamte Spiel ausgiebig getestet und auffallende Fehler behoben. Im Laufe des Entwicklungsprozesses wurde entschieden, das Spiel nicht im Web, sondern als ausführbares Programm für Windows und MacOS zur Verfügung zu stellen. Unity stellt die Buildprozesse für diese beiden Betriebssysteme zwar zur Verfügung, jedoch wird für den MacOS Build auch ein Rechner mit dem entsprechenden Betriebssystem benötigt. Anstatt den MacOS Build auf dem bisher verwendeten Windows-Rechner auszuführen, wurde von GitHub Actions[8] Gebrauch gemacht. Die vorgefertigte Unity-Builder Action[9] wurde in das DE:LINT GitHub Repository eingefügt, die zugehörige Unity-Lizenz aktiviert, und der zugehörige Workflow ausgeführt.

Mithilfe eines alten MacOS-Geräts war es möglich, den erstellten Build grob zu testen. Dabei fiel auf, dass für bestimmte Dateien im Spiel die Zugriffsrechte angepasst werden mussten. Dieser Schritt wurde durchgeführt und der angepasste Build gemeinsam mit dem Windows-Build als Release zur Verfügung gestellt.

[8] Siehe `https://docs.github.com/en/actions`
[9] Siehe `https://github.com/marketplace/actions/unity-builder`

6 Evaluation

Im letzten Schritt des ADDIE-Modells soll das fertig entwickelte Spiel in verschiedenen Aspekten ausgewertet werden. Dazu gehört eine technisch-funktionale Evaluation, welche größtenteils auf persönlichen Einschätzungen basiert, sowie eine didaktische Evaluation anhand einer Umfrage. Die erstellte Umfrage soll von allen freiwilligen Spieltestern im Anschluss an ihr Spielerlebnis ausgefüllt werden.

6.1 Technische Evaluation

Wie zuvor schon erwähnt wurde nur ein Teil der denkbaren Features für DE:LINT tatsächlich umgesetzt, weswegen eine Erweiterung und Weiterentwicklung des Spiels nach Abschluss dieser Arbeit angestrebt wird. Dadurch ist die Erweiterbarkeit des geschriebenen Codes und der erstellten Funktionen im Unity Editor ein wichtiges Kriterium für die technische Evaluation. Folgende getroffene Maßnahmen spielen für die Erweiterbarkeit eine unterstützende Rolle:

- Simples Hinzufügen neuer Mini-Spiele, diese können wenn nötig eine komplett eigene Architektur verwenden
- Basisfunktionen basieren auf einer ScriptableObjects-Architektur, weswegen diese ohne Veränderung von Code angepasst und erweitert werden können. Dazu gehören:
  - Dialoge
  - Quests
  - Interaktionen
  - Story-Elemente
- Entwicklung von wiederwendbaren Einheiten, wie die StateMachine

Jedoch fallen einige verbesserungswürdige Codestellen auf, die zukünftige Wartungen und Erweiterungen des Spiels erschweren könnten. Dazu gehört der Aufbau der bisher erstellten Minispiele, insbesondere des Spiels zur Erklärung des Befehlszyklus. Die erstellten Klassen und Methoden haben eine hohe Code-Komplexität und können nach persönlicher Einschätzung nicht als „Clean Code“ betitelt werden. Sollten die Minispiele also zu einem späteren Zeitpunkt verändert werden, wäre ein Refactoring des bisherigen Codes notwendig.

Ein weiterer wichtiger Evaluationsaspekt ist die allgemeine Wiederverwendbarkeit der erstellten Systeme. Wie geplant wurde jedes System isoliert entwickelt und getestet.

Somit können diese mit geringem Aufwand aus DE:LINT extrahiert und für die zukünftige Spieleentwicklung in anderen Projekten wiederverwendet werden. Insbesondere die StateMachine in Kombination mit dem GameController, sowie das Dialog- und Questsystem haben sich als stabile und extrahierbare Systeme hervorgetan.

Im Gegensatz dazu steht das Speicher- und Ladesystem. Die Methodik, jedes ScriptableObject einzeln in die Liste der zu speichernden Daten einzufügen erscheint umständlich. Außerdem muss darauf geachtet werden, dass die ScriptableObjects nicht verschachtelt und insgesamt serialisierbar sind. Deswegen ist eine Übernahme des Speicher- und Ladesystems in andere Projekte nicht empfehlenswert.

Zur technisch-funktionalen Auswertung gehört außerdem die Betrachtung der Übersichtlichkeit und Ästhetik. Wie zuvor schon angemerkt wurde, wurde während der Entwicklung wenig Wert auf Ästhetik gelegt. Daraus resultierend wurden Benutzeroberflächen sehr simpel gehalten und die Bearbeitung der Lichtverhältnisse, Animationen und Texturen ausgelassen. Darunter leidet die Übersichtlichkeit und Benutzerfreundlichkeit des Spiels, weswegen diese Aspekte in zukünftigen Anpassungen dringend einbezogen werden sollten.

6.2 Evaluation des Instruktionsdesigns

Für die didaktische Beurteilung wurde ein Umfrage entworfen, welche freiwilligen Spieletestern zugesendet wurde. Ziel der Umfrage war, den Lehrwert des bisherigen Spiels zu ermitteln, sowie das Potenzial einer Weiterführung zu erforschen.

6.2.1 Entwurf der Umfrage

Da ein vollständiges Durchspielen von DE:LINT eine gewisse Zeit beansprucht, soll die Umfrage so knapp wie möglich gehalten werden und demnach nur die wichtigsten Fragen beantworten. Wie in 2.1.3 beschrieben gehört dazu die Akzeptanz, die Lernprozesse und -ergebnisse sowie Verbesserungsvorschläge und ein Fazit. Auf Basis dieser Themen werden die folgenden Fragen für die Umfrage ausgewählt (aus dem Englischen übersetzt):

1. Bist du ein Informatikstudent, oder hast du Erfahrung im Informatikbereich?
2. Hast du etwas gelernt während du DE:LINT gespielt hast?
3. Wenn du auf die vorherige Frage mit Ja geantwortet hast, was hast du gelernt?
4. Wie viel Spaß hattest du beim Spielen von DE:LINT (auf einer Sterne-Skala von eins bis fünf)
5. Denkst du, dass DE:LINT oder ein Nachfolger von DE:LINT verwendet werden kann, um Rechnerarchitektur zu lehren?

6. Während der Entwicklung des Spiels habe ich mich hauptsächlich auf Funktionalität und Game-Design fokussiert, aber weniger Wert auf Game-Art und UI-Design gelegt. Hast du in Anbetracht dieser Gegebenheiten Bugs gefunden, oder hast du Tipps um aus DE:LINT ein besseres Spiel zu machen?

7. Für den Fall, dass du noch etwas anderes bezüglich DE:LINT sagen möchtest, kannst du dies hier tun.

Die freiwilligen Tester stammen aus dem persönlichen Umfeld, weswegen kein einheitliches Vorwissen von diesen erwartet werden kann. Frage 1 dient deshalb dazu, das Erreichen der Zielgruppe zu überprüfen. Die zweite und dritte Frage können Aufklärung darüber bieten, ob die eingeplanten Lernziele vom bisher entwickelten Spiel erreicht werden. Die vierte Frage hingegen überprüft die Akzeptanz der Spieler in Bezug auf das Spiel. Zum Abschluss ermöglichen Frage 6 und 7 die Formulierung von Verbesserungsvorschlägen und einem Fazit.

6.2.2 Erwartete Ergebnisse

Anhand des Instruktionsdesigns kann eine Zielsetzung für die Ergebnisse der Umfrage aufgestellt werden. Zusätzlich dazu können auch erwartete Abweichungen von diesen Zielen anhand der technischen Evaluation formuliert werden.

Von Testern mit Erfahrung im Bereich der Informatik ist zu erwarten, dass sie sowohl das Basiswissen als auch den Befehlszyklus kennen. Deswegen kann nicht damit gerechnet werden, dass jeder Spieler vom derzeitigen DE:LINT-Spiel lernt. Stattdessen ist zu erwarten, dass die Zahl der Spieler von DE:LINT mit erreichten Lerneffekten mit der Teilnehmerzahl ohne Erfahrung im Informatikbereich korreliert. Es ist auch außerhalb eines Informatikstudiums oder -berufs üblich, den groben Aufbau eines Rechners zu kennen und gegebenenfalls einen eigenen Rechner zusammenzustellen. Deshalb kann erwartet werden, dass die hauptsächlichen Lernergebnisse im Befehlszyklus-Minispiel erzielt werden.

Laut Instruktionsdesign sollen die Lernenden möglichst viel Spaß während der Spielzeit haben - eine Zielsetzung wäre also eine Bewertung von durchschnittlich vier Sternen in Frage 4. Wie in der technischen Evaluation jedoch schon angemerkt wurde, hatte die Ästhetik und Benutzerfreundlichkeit des Spiels eine geringe Priorität während des Entwicklungsprozesses. Dies wirkt sich vermutlich negativ auf den Spielspaß aus, weswegen eine geringere durchschnittliche Wertung zu erwarten ist.

Ziel ist außerdem, dass hundert Prozent der Tester einschätzen, dass DE:LINT oder ein Nachfolger des Spiels als Lehrmittel für Rechnerarchitektur verwendet werden kann. Da der derzeitige Stand der Entwicklung diese Antwort nicht beeinflussen sollte, hat die technische Evaluation keinen Einfluss auf diese Erwartungshaltung.

In Frage 6 und 7 wird in Bezug auf das Instruktions- und Game-Design hauptsächlich

positives Feedback erwartet. Zusätzlich dazu ist jedoch vermutbar, dass die schlechte Benutzerfreundlichkeit, sowie negative Einschätzungen der Spielästhetik angesprochen werden.

6.2.3 Auswertung der Umfrageergebnisse

Im Zeitraum vom 14.03.2022 bis 26.03.2022 haben zwanzig freiwillige Spieltester die Umfrage zur Bewertung von DE:LINT ausgefüllt. Eine Übersicht über alle Beantwortungen ist im Appendix 1.2 verlinkt. Folglich sollen nun die Ergebnisse in Bezug auf die Zielsetzung der Arbeit und die Erwartungen ausgewertet werden.

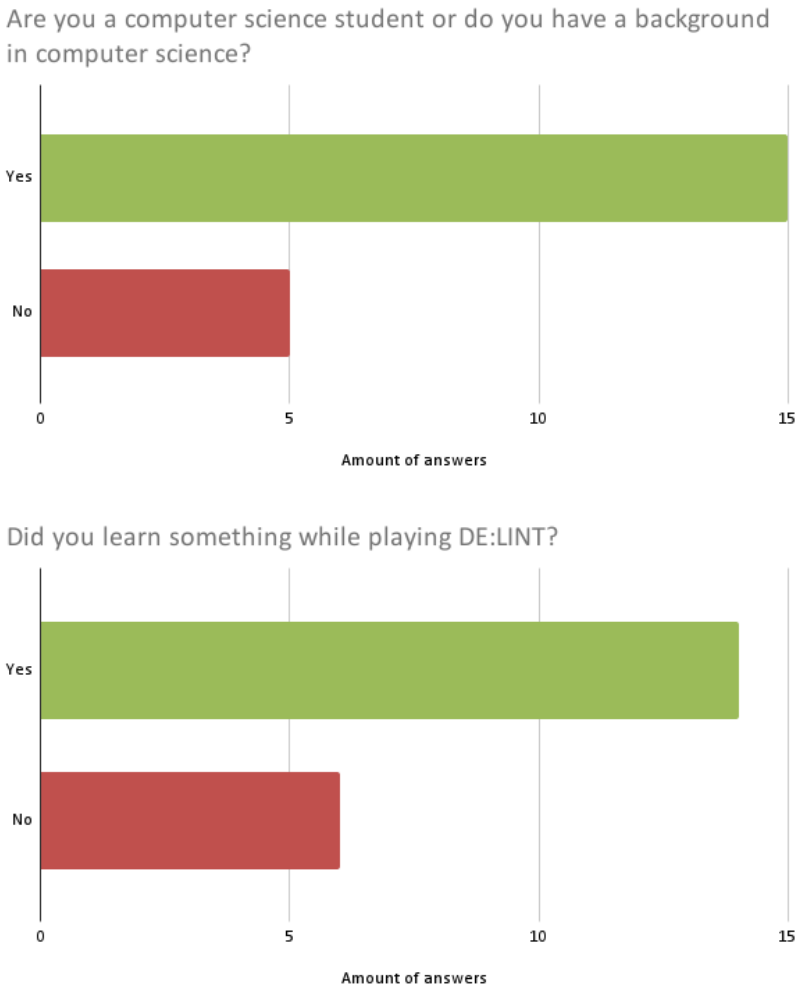


Abbildung 6.1: *Umfrage-Ergebnisse der Fragen 1 und 2; Quelle: Eigene Darstellung*

Abbildung 6.1 zeigt die Ergebnisse der Fragen 1 und 2. Wie erwartet haben Tester an der Umfrage teilgenommen, die keine Erfahrung im Informatikbereich haben. Überraschenderweise haben jedoch mehr Personen angegeben von DE:LINT gelernt zu haben,

wodurch die Erwartungen stark überstiegen werden. Frage 3 liefert etwas Aufklärung über diese Abweichung. Von 14 Testern, die etwas durch DE:LINT gelernt haben, haben vier angegeben, dass das Gelernte vielmehr eine „Wiederholung“ vorhandenen Wissens, als ein Erlangen von neuem Wissen war.

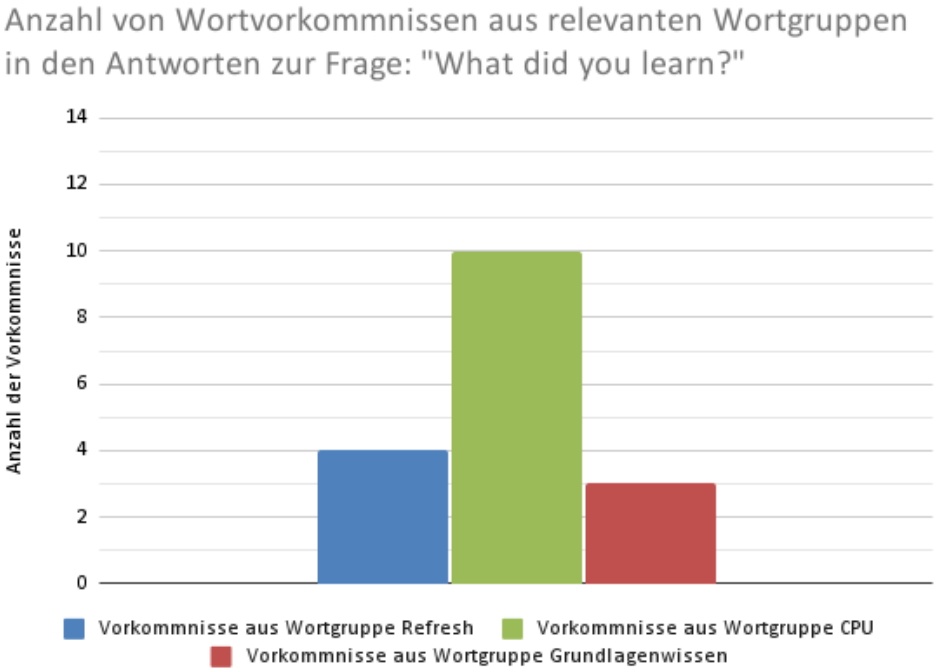
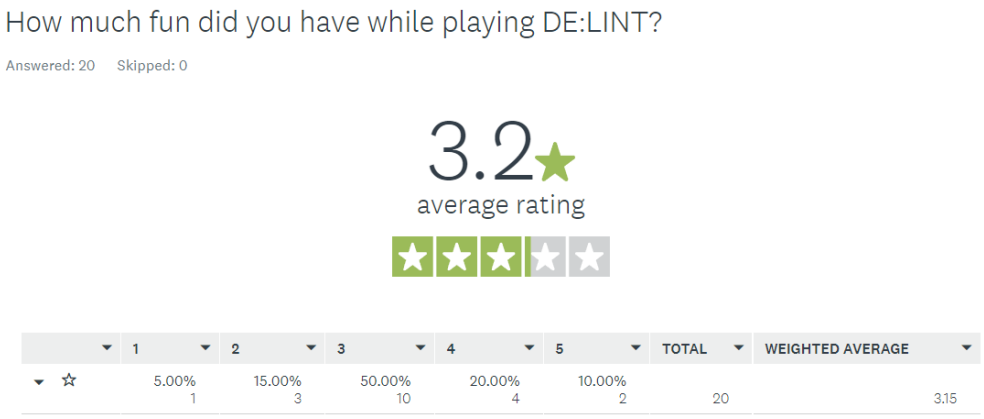


Abbildung 6.2: *Wortvorkommnisse in Umfrage-Frage 3; Quelle: Eigene Darstellung*

Auffällig ist außerdem, dass 10 der 14 Antworten aus Frage 3 den Begriff *CPU* beinhalten, wie Abbildung 6.2 zeigt. Wie vermutet hat die Mehrheit der Lernenden angegeben, dass sie etwas über den Befehlszyklus im CPU gelernt haben. Drei der Tester gaben an, erstmalig ein Grundlagenwissen über PC-Komponenten erlangt zu haben.

Für die Bewertung des Spielspaßes erreicht das Gesamtergebnis nicht das Ziel einer durchschnittlichen 4-Sterne-Bewertung, sondern einen Durchschnitt von 3,2 Sternen, wie in Abbildung 6.3 zu sehen ist. Dies kann durch die bereits erwähnten Probleme in der Benutzerfreundlichkeit und fehlender Ästhetik ausgelöst worden sein. Zur Überprüfung dieser Vermutung können die Antworten zu den Fragen 6 und 7 verwendet werden.



	1	2	3	4	5	TOTAL	WEIGHTED AVERAGE
☆	5.00% 1	15.00% 3	50.00% 10	20.00% 4	10.00% 2	20	3.15

Abbildung 6.3: *Umfrage-Ergebnisse der Frage 4; Quelle: Eigene Darstellung*

Mit einer durchschnittlichen Anzahl von 93.9 Wörtern pro Antwort beinhaltet Frage 6 sehr viel mehr Feedback als erwartet. Da hier nach gefundenen Bugs und Tipps gefragt wurde, sind den Testern im Spielprozess also einige Verbesserungsmöglichkeiten aufgefallen. Folgende Themen stechen dabei besonders hervor:

- Der Reset-Button im Befehlszyklus-Minispiel hat das Spiel nicht korrekt zurückgesetzt, weswegen das Spiel für einige Tester nicht komplettierbar war.
- Die Erklärung des Befehlszyklus erfolgt hauptsächlich durch Linta und viele haben diesen Dialog übersprungen, weswegen sie im Minispiel nicht wussten was sie tun sollen. Eine bessere Anleitung während des Minispiels ist also wünschenswert.
- Der Third-Person-Controller hat an verschiedenen Stellen Probleme hervorgerufen:
  - Es ist möglich, aus dem PC-Gehäuse zu springen. Anschließend ist der Spielercharakter in einem dauerhaften Zustand des Fallens und das Spiel muss neugestartet werden um dies zurückzusetzen.
  - Die Sprunghöhe war teilweise nicht hoch genug, um aus Löchern in der Umgebung wieder rauszuspringen.
  - Durch das Entfernen und Neu-Platzieren von Komponenten kann der Spielercharakter im Boden eingeklemmt werden, wodurch die weitere Fortbewegung verhindert wird.
- Die Orientierung auf dem Motherboard ist einigen schwergefallen. Eine Übersichtskarte wäre dabei hilfreich.
- Den Spielern hat eine Möglichkeit gefehlt, die Minispiele abzubrechen und zu einem späteren Zeitpunkt neu starten zu können.

Diese Punkte entsprechen der Vermutung, dass die Benutzerfreundlichkeit den Spielspaß verringert hat. Interessanterweise wurde hier jedoch keine Aussage über die

Ästhetik des Spiels getroffen. Gegenteilig zu den Erwartungen enthalten die Antworten aus Frage 7 sogar vereinzeltes Lob über die Ästhetik des Spiels: „Lovely idea, loved the art style.“(Teilnehmer 6), „Mir gefallen die Models gut, find auch die Idee mit dem Herumspazieren auf dem Mainboard gut“(Teilnehmer 4).

Als Zusammenfassung der Ergebnisse aus Frage 7 lässt sich außerdem sagen, dass die Idee des Spiels bzw. das Instruktionsdesign von den Testern sehr gut aufgenommen und bewertet wurde. Negatives Feedback wird hauptsächlich durch fehlendes Interesse am Gebiet der Rechnerarchitektur begründet.

Zum Abschluss der Auswertung soll insbesondere die Frage beantwortet werden, ob DE:LINT oder ein entsprechender Nachfolger des Spiels verwendet werden kann, um Wissen über Rechnerarchitektur zu vermitteln. Wie die Ergebnisse aus Frage 5 und die zugehörige Abbildung 6.4 zeigen, haben alle zwanzig Spieltester diese Frage mit „Ja“ beantwortet. Dies entspricht vollkommen der Zielsetzung und Ergebniserwartung dieser Arbeit.

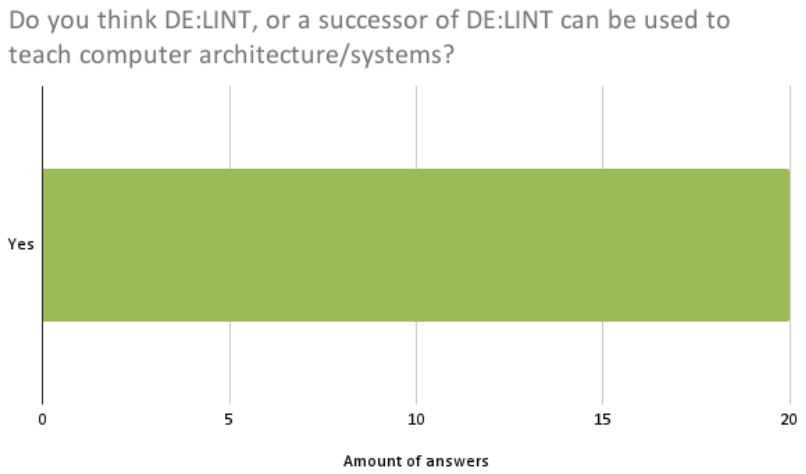


Abbildung 6.4: *Umfrage-Ergebnisse der Frage 5; Quelle: Eigene Darstellung*

7 Zusammenfassung

Als Ergebnis dieser Arbeit ist ein begründetes und effektives Spielkonzept zur Wissensvermittlung über Rechnerarchitektur sowie das erweiterbare Teilspiel DE:LINT auf Basis dieses Konzepts entstanden. Eine Umfrage unter freiwilligen Spieletestern hat nicht nur die Effektivität des Konzepts bestätigt, sondern auch gezeigt, dass ein Interesse an dem Themengebiet im Spielkontext besteht. Dennoch sind im Rahmen der Evaluation noch einige Fehlerstellen und mögliche Optimierungen aufgefallen, welche sich auf den Spielspaß der Tester ausgewirkt haben und für die Weiterentwicklung dringend behoben werden sollten.

7.1 Fazit

Über die Problem- und Aufgabenstellung dieser Arbeit hinweg, sind im Rahmen der Spielentwicklung wiederverwendbare Systeme für spätere Projekte entstanden. Durch die isolierte Entwicklung der Spielsysteme ist es für zukünftige Arbeiten möglich, diese aus dem Code zu extrahieren.

Durch die Beachtung relevanter Code-Prinzipien und die Achtsamkeit den Code sauber zu halten bietet sich DE:LINT als zukünftiges Gruppenprojekt an. Die Erweiterung durch zusätzliche Mini-Spiele ist simpel und die Bearbeitung und Weiterführung von Dialogen, Quests und der Geschichte des Spiels benötigt keine Code-Anpassung. Stattdessen ist es sogar für Nicht-Programmierer möglich, diese Systeme zu verwenden.

Das entstandene Instruktions-Konzept geht außerdem weit über den bisherigen Spielstand hinaus und lässt sich ebenso simpel erweitern, wie der zugehörige Code.

7.2 Limitationen

Gegensätzlich zu den Erwartungen ist die Spielästhetik den Testern nicht negativ aufgefallen. Dennoch sollte für die spätere Entwicklung ein Teammitglied mit Expertise im Game Art und UI-Design-Bereich hinzugezogen werden. Insbesondere durch das fehlende UI-Design hat die Benutzerfreundlichkeit von DE:LINT stark gelitten.

Aufgrund von Zeitmangel wurde außerdem auf einige wichtige Features verzichtet und einige bestehende Features nicht ausführlich genug getestet. Vor der Weiterentwicklung sollten entsprechende Bugs behoben werden und anschließend die wichtigsten Features zur Verbesserung der Übersichtlichkeit hinzugefügt werden. Dazu gehört eine Übersichtskarte und die Anpassung des Befehlszyklus-Minispiels.

7.3 Ausblick

Da DE:LINT mit hoher Wahrscheinlichkeit weiterentwickelt wird, stellt sich die Frage, wo dieses Spiel letztendlich eingesetzt werden kann. Die Umfrage aus dieser Arbeit hat gezeigt, dass neben Personen aus dem Informatikbereich auch Interesse bei fachfremden Personen besteht. Deswegen ist die Veröffentlichung des finalen Spiels auf üblichen Plattformen wie *Steam* oder *Itch.io* denkbar.

Um die Effektivität des Instruktionsdesigns tiefergehend zu testen, würde es sich außerdem anbieten, das finale Spiel zur Unterstützung der Lehre in relevanten Informatikkursen zu verwenden und die Ergebnisse erneut anhand einer Umfrage auszuwerten.

Quellenverzeichnis

[1] John Adam. *The Kanban system for agile software development explained*. Okt. 2021. URL: https://kruschecompany.com/kanban-method-agile-software-development/ (besucht am 29. 03. 2022).

[2] *Extreme Programming: Software development to the extreme*. URL: https://www.ionos.com/digitalguide/websites/web-development/extreme-programming/ (besucht am 29. 03. 2022).

[3] Daniel Gyger. *Feature-Driven Development: Agile vs. klassische Methoden der Software-Entwicklung*. 2003/04. URL: https://files.ifi.uzh.ch/rerg/amadeus/teaching/seminars/seminar_ws0304/08_Gyger_Fdd_Ausarbeitung.pdf (besucht am 29. 03. 2022).

[4] M. Helm und F. Theis. *Digitale Lernwelt - Serious games : Einsatz in der beruflichen Weiterbildung*. Bertelsmann, 2011. ISBN: 9783763948079. URL: https://books.google.de/books?id=IGnuxN6yUGsC.

[5] Ryan Hipple. *Unite Austin 2017 - Game Architecture with Scriptable Objects*. Youtube. 2017. URL: https://www.youtube.com/watch?v=raQ3iHhE_Kk.

[6] K.M. Kapp. *The Gamification of Learning and Instruction: Game-based Methods and Strategies for Training and Education*. Pfeiffer essential resources for training and HR professionals. Wiley, 2012. ISBN: 9781118096345. URL: https://books.google.de/books?id=M2Rb9ZtFxccC.

[7] Anderson LW u. a. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Jan. 2001. ISBN: ISBN: 080131903X.

[8] Rachaelle Lynn. *What is FDD in Agile?* URL: https://www.planview.com/resources/articles/fdd-agile/ (besucht am 29. 03. 2022).

[9] R.C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Robert C. Martin Series. Pearson Education, 2008. ISBN: 9780136083252.

[10] K. Oxland. *Gameplay and Design*. Addison-Wesley, 2004. ISBN: 9780321204677. URL: https://books.google.de/books?id=l05TkZFbS24C.

[11] Mark Richards. *Event-Driven Architecture*. URL: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html (besucht am 29. 03. 2022).

[12] Aaron Salisbury. *Data Persistence: Saving Games Online in Unity WebGL*. Apr. 2016. URL: https://amalgamatelabs.com/Blog/1/data_persistence (besucht am 27. 03. 2022).

[13] *Statemachine*. URL: https://www.mikrocontroller.net/articles/Statemachine (besucht am 29. 03. 2022).

[14] A.S. Tanenbaum und T. Austin. *Rechnerarchitektur: Von der digitalen Logik zum Parallelrechner*. Always learning. Pearson Studium ein Imprint von Pearson Deutschland, 2014. ISBN: 9783868942385. URL: https://books.google.de/books?id=pagrnwEACAAJ.

[15] Redaktionsteam e teaching.org. *Instruktionsdesign*. Sep. 2007. URL: https://www.e-teaching.org/didaktik/theorie/instruktionsdesign/instruktionsdesign.pdf.

[16] Ahmed Tlili, Fathi Essalmi und Mohamed Jemni. „Improving learning computer architecture through an educational mobile game“. In: *Smart Learning Environments* 3 (Mai 2016). DOI: 10.1186/s40561-016-0030-6.

[17] *Unity-Manual: Creating and using scripts*. URL: https://docs.unity3d.com/560/Documentation/Manual/CreatingAndUsingScripts.html (besucht am 29. 03. 2022).

[18] Don Wells. *Extreme Programming: A gentle introduction*. URL: http://www.extremeprogramming.org/ (besucht am 29. 03. 2022).

Glossar

Backlog ist eine Liste von bisher unbearbeiten Aufgaben in einem Projekt. 9, 21, 26

Collider bieten Funktionen zur Überprüfung und Verarbeitung von 2D- und 3D-Objektüberschneidungen in Unity. 34

Delegate ist ein Zeiger auf eine Funktion im Code. 26

Frame ist ein Einzelbild aus Filmen oder Spielen. 11

Game Engine ist ein Framework für die Entwicklung von Videospielen. 10

Git Submodule Ein Submodul eines Git-Repositorys ist ein eingebundenes zweites Repository und kann gesondert bearbeitet und hochgeladen werden. 26

Json (JavaScript Object Notation) ist ein Datenformat. 29

JsonUtility ist eine von Unity zur Verfügung gestellte Klasse zur Verarbeitung von Json-Daten. 29

Minispiel ist ein integriertes, kleines Spiel innerhalb eines Gesamtspiels.. 17, 18, 19, 24, 32, 35, 37, 38, 39, 44

MoSCoW ist ein Priorisierungsverfahren. Der Name steht für *must have, should have, could have, won't have*. 21

Namespace ist ein Paket von zusammenhängendem Code in C#. 27

Overhead ist zusätzlicher Arbeitsaufwand in einem Entwicklungsprozess. 21

Quest stellt ein Ziel oder eine Aufgabe in Spielen dar. 22, 24, 34, 37, 38, 39, 40

Tech-Stack Im Tech-Stack werden die innerhalb eines Projekts verwendeten Technologien aufgelistet. 14

UML (Unified Modeling Language) ist eine Modellierung-Sprache für Software. 32

A Appendix

1.1 Quell-Code

Der Quellcode befindet sich im zugehörigen GitHub-Repository mit der URL `https://github.com/proehr/DE_LINT`.
Das resultierende Spiel wurde dort im Release v1.0.0 hochgeladen, siehe `https://github.com/proehr/DE_LINT/releases/tag/v1.0.0`.
Das Repository wurde am 03.04.2022 archiviert und kann demnach nicht mehr bearbeitet werden.

1.2 Umfrageergebnisse

Die gesamten Umfrageergebnisse sind in einem Tabellenformat auf der Webseite `https://tinyurl.com/yabgohjd` einsehbar.

1.3 Design Dokument

DE:LINT

Instructional Game Design Document

DE:LINT is a learning game that aims to educate CS students and/or PC gamers about the functionality and structure of PC components. Players will have the ability to travel through the PC and its components by shrinking themselves and will be tasked with cleaning any lint that has gathered inside. As the Player reaches maximum cleanliness in all components, their PC is fixed. Bigger learning tasks are indicated by a malfunctioning component and by fixing said component the player will learn how the component functions and gain more cleanliness inside the PC.

Outcome

The Player has memorized and understood the roles and functionality of all basic PC components (CPU, RAM, SSD, HDD, Motherboard, GPU, etc.). If the player has progressed further, they have understood the deeper functionality of singular components (CPU, Primary Storage, Secondary Storage etc.).

Instructional Objectives

At the end of the game, the player will be able to:

Basics

- Explain the structure and levels of a modern computer
- Name most important components of a modern computer and their roles
- Describe what a PC looks like from the inside

History of PC architecture

- Explain functionality of the first mechanical computers
- Explain functionality of computers with vacuum tubes
- Explain why transistor changed the industry and how integrated circuits were introduced

CPU

- Name components of the CPU and their roles
- Explain the Von-Neumann-Architecture

Digital Logic

- Explain what a transistor is used for
- Name multiple Gates and draft their truth tables
- Properly apply rules of Boolean Algebra
- Use gates to set up specific circuits (Multiplexer, Decoder, Comparator, Shifter, Half Adder, Full Adder, Latches, Flip Flops)
- Explain the purpose of those circuits

Primary Storage

- Name components of the primary storage and their roles

Secondary Storage

- Name multiple types of secondary storage and explain functionality of those

Math

- Use number systems of different bases for calculations and transform numbers into other bases

Whether the player is able to do mentioned objectives is dependent on their progression in the game. The Objectives may also be extended, or some may be pushed back into further development, since this game allows for extension in any way.

Description of Characters & Characteristics

The Player is able to pick a basic human character type and customize the eyes, skin, hair and clothing in terms of color. Their player character may be customized later on in the game as well. The picked character is used to walk through the PC and components, as seen from a third-person-perspective.

Linta is a floating piece of Lint, who accompanies the player on their journey through the PC. She takes the role of the instructor in the game, and urges the player to help her fix and clean everything. She looks and behaves similarly to Soot Sprites from the movie Spirited Away.

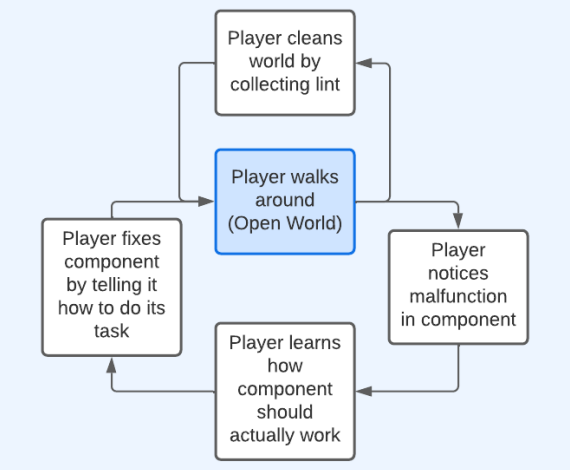
All **components** may be given a personality as well. Their shape and form should still be realistic, though they may be given some facial features and/or arms. Possible character names are “Mother Boardie”, “Proceus”, “Storeos the First” and “Storeos the Second”, “Potentia”, and “Expresseus”. All of those may be designed more deeply when needed.

Description of Game Environment

The Game starts off in a typical office environment, but the player is soon shrunken and is now able to walk around their PC from a third person perspective. The surroundings should look at least similar to actual PC components, though they may be stylized and simplified. The surroundings are not too colorful, though the lighting may introduce some color, since modern PCs often have some (RGB) lighting components in them. The environment depicts liveliness and has a techy vibe.

Description of Game Play

The given diagram shows a rough outline of the core game loop. To give the player a sense of control, the environment should be explorable at their own pace, which is why an open world approach is helpful.



Any processes in this cycle are guided by Linta, who gives the players hints where to look, and gives the players quests to learn something. Linta will eventually be the instructor in those quests and enable the player to automate the tasks they just learned. When the quest is done, she guides the player through the Open World again.

Reward Structure

The player is rewarded with the collection of Lint - the more Lint they have collected, the cleaner their virtual PC is. They are able to collect Lint from the Open World (Gathering/Cleaning Mechanic) and by completing Lintas Quests. Specifically - if a component is malfunctioning and the player has learned and in return taught the component how to do its' thing, the component "spits out" all of the Lint which has been causing the malfunction i.e. "everything has been cleared up". Lint is collectible in different sizes, proportionally to the size of the component/the place where it is collected from. The player can collect Big Lint on the motherboard, Normal Lint inside a component and the smallest lint on digital-logic-level. They can be converted: 1 Big Lint = 100 Normal Lint = 100000 Small Lint. The different sizes of Lint can be exchanged for items to progress deeper into the PC, to explore new areas or to unlock new mechanics (i.e. pulling cables and cleaning the plugs, cleaning more sensitive parts). The final Goal is to collect all Lint (if possible 100 Big Lint), as a metaphor for fixing all components and therefore the PC in general.

Look and Feel of the Game

Since hyperrealistic 3D-Assets of all hardware components are hard to come by, a more stylized approach is desirable. Linta is a comic-y figure, and to make her fit into the surroundings, the game should look comic-y as well. It is completely fine to leave out some detail in the look of the components, instead there should be a focus on giving Linta and all of the components lots of personality, and giving the surrounding a lively feel, as if power is running through everything. While the components and environment may be mostly gray-ish, the lighting can play a huge

role and has the potential to introduce warnings/danger situations. More info about the look of the game can be found in a Mood/Art Board.

Technical Description

Since the Game will be developed in Unity, it can be made available via itch.io for all players in a browser, or as a build for MacOS and Windows. This may be decided later on. If needed, data can be collected via Azure Playfab. More technical info about this project can be found in a technical design document.

Project Timeline

The Game needs to be finished within 1 month of full-time work, or 2 months of half-time work.

1.4 Backlog

Card Name	Card Description
[Dev] GameController	The basic game states should be controlled by a StateMachine. ACs: – See GameStateDiagram – The GameController should not be called by any other class instead the GameController calls all other classes
[Dev] Main Menu	ToDos: - Implement Main Menu functionalities ACs: - "New Game" Button, starts character creation - "Load Game" Button, loads previous game from save file - "Exit" Button
[Dev] Pause Menu	ACs: - "Resume" Button, goes back to Gameplay - "Options" Button, opens Options UI - "Save & Return to Main Menu" Button - "Save & Exit" Button
[Dev] Save and Load	ACs: - Progress is saved automatically into a GameData Scriptable Object - The Player can save progress into a save file - The save file is encrypted - The Player can load a game from a save file
[Dev] Player Movement	ACs: - The Player is able to walk around with WASD from a third person perspective - The PlayerCharacter has a walking animation - During Dialogues and Interactions, the Player has no control over movement - Linta follows the Player wherever they go
[Dev] Camera Movement	ACs: - In Exploration mode, the Player can control the camera with the mouse - During Dialogues and Interactions, the Player has no control over the camera
[Design] Mainboard Map	
[Dev] Cleanup Mechanic	ACs: - The Player is able to collect Lint by walking through it
[Dev] HUD	ACs: - Collected Lint (Big Lint, Normal Lint, Small Lint) - Quest Log
[Dev] Interaction System	ACs: - The Player should be able to interact with objects and NPCs - Accordingly, a dialogue may be initiated
[Dev] Dialogue Interaction	ACs: - Player is able to hold dialogues with NPCs - When needed, Player can pick between dialogue options - Dialogues can unlock quests
[Dev] Quest System	ACs: - The Player can accept quests, and see them in the quest log - The Quest System notifies involved NPCs and Interactable Objects about their sta - When player finishes a quest, the quest is ticked off and removed from the quest l
[Dev] Plug-A-PC Mini Game	ACs: - The player can play a puzzle game to arrange their PC
[Dev] Mini Game Interaction	ACs: - The Player can enter a mini game, inside the main game as an interaction
[Dev] StoryController	
[Design] Start scene & first quest	
[Dev] Instruction Cycle Mini Game	
[Dev] Add Instruction Cycle mini game quest	
[Dev] Add all component examination quests	
[Dev] Linta	
[Dev] Add lint collection quest to story	
[Dev] Remove component interaction	
[Dev] Add enough Lint to finish game	

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

04.04.2022, Berlin,

Datum, Ort, Unterschrift