Projektdokumentation

Mustafa Günes (40417) - Edv Nr: 113400
Yekta Kücük (40384) - Edv Nr: 113468
Aykon Kücük (40250) - Edv Nr: 113468
Rometh Umic-Senol (43376) - Edv Nr: 113400

Parallel Universe

Einführung

Das Projekt "Parallel Universe" ist ein 2D-Jump 'n' Run-Action-Plattformer mit verschiedenen Spielelementen und Gegnern, die eine innovative Spielmechanik bieten. Der Spieler navigiert zwischen zwei parallel laufenden Welten, um Fortschritte zu erzielen und Hindernisse zu überwinden. Das Spiel kombiniert klassische Jump 'n' Run- und Action-Elemente, um den Spielern ein herausforderndes Spielerlebnis zu bieten.

In Parallel Universe bewegt sich der Spieler durch mysteriöse Welten, um verlorene Portal-Scherben zu sammeln und die beschädigten Portale wiederherzustellen. Diese Portale dienen als Verbindung zu neuen und aufregenden Welten, die darauf warten, erkundet zu werden.

Technologien und Entwicklung

Die Entwicklung von "Parallel Universe" erfolgt mithilfe der Unity-Engine, die sich als vielseitige Plattform für 2D-Spiele bewährt hat. Das Projekt wird über die Versionskontrollsoftware Github verwaltet, was eine effiziente Zusammenarbeit im Entwicklerteam ermöglicht. Die Skripte des Spiels werden in der Programmiersprache C# in Visual Studio geschrieben, um eine robuste und wartbare Codebasis zu gewährleisten. Für die Grafiken des Spiels werden Adobe-Programme eingesetzt, während Aseprite für die Pixel Art-Gestaltung verwendet wird.

Zielgruppe

Die Zielgruppe von "Parallel Universe" sind Gaming-Enthusiasten mit einem besonderen Interesse an Indie-Spielen und Retro-Spielmechaniken. Das Spiel richtet sich an Spieler, die Herausforderungen in Jump 'n' Run-Plattformern suchen und sich von der einzigartigen Spielmechanik des Wechselns zwischen parallelen Welten angezogen fühlen.

Qualitätsziele

1. **Stabile Performance**: Das Spiel sollte eine stabile und konsistente Framerate aufweisen, um ein flüssiges Spielerlebnis zu gewährleisten, unabhängig von der Hardware, auf der es ausgeführt wird.
2. **Ästhetische Gestaltung**: Ein ansprechendes visuelles Design und eine konsistente ästhetische Gestaltung tragen zu einem positiven Spielerlebnis bei.
3. **Kreative Spielelemente**: Das Spiel sollte innovative und unterhaltsame Spielelemente bieten, die das Interesse der Spieler wecken und ein einzigartiges Spielerlebnis ermöglichen.
4. **Bugfreie Funktionalität**: Ziel ist es, Bugs und technische Fehler so weit wie möglich zu minimieren, um eine reibungslose und fehlerfreie Interaktion mit dem Spiel zu ermöglichen.
5. **Gute Balance und Herausforderung**: Ein ausgewogenes Gameplay und eine angemessene Schwierigkeitskurve sind entscheidend, um die Spieler zu motivieren und sie vor Herausforderungen zu stellen, ohne sie zu überfordern.
6. **Feedback und Iteration**: Regelmäßiges Feedback von Testern und Teammitgliedern sollte in den Entwicklungsprozess integriert werden, um das Spiel kontinuierlich zu verbessern und auf die Bedürfnisse der Spieler einzugehen.

Minimum Viable Product

2D-Spielumgebung:

- Die Spielwelt besteht aus 2D-Leveln, die als Side-Scrolling-Umgebungen gestaltet sind.
- Das Spiel enthält zwei parallele Welten, zwischen denen der Spieler wechseln kann.

Charaktersteuerung:

- Der Spieler kann einen Hauptcharakter steuern, der in der Lage ist zu laufen, zu springen und zu kämpfen.
- Die Steuerung erfolgt über Tastatur (W-A-S-D oder die Pfeiltasten) oder den Controller.

Parallelwelt-Mechanik:

- Der Spieler kann wenn er genug “Shards” sammelt, zwischen den beiden parallelen Welten wechseln.
- Hindernisse und Rätsel in der Level Gestaltung erfordern den Wechsel zwischen den Welten, um fortschreiten zu können.

Jump 'n' Run-Elemente:

- Das Spiel bietet klassische Jump 'n' Run-Elemente, wie präzises Springen über Abgründe, Ausweichen von Gefahren und das Sammeln von Power-Ups.

Grafik und Audio:

- Das Spiel verfügt über Pixel Art-Grafiken, die den Retro-Stil vermitteln.
- Passende Soundeffekte und Musik werden das Spielerlebnis ergänzen.

Randbedingungen

Technische Randbedingungen:

- Das Projekt "Parallel Universe" wird in der Unity Game Engine entwickelt.
- Die Programmiersprache für die Skripte ist C#.
- Für die Grafiken werden Adobe Programme verwendet, insbesondere für die Pixel Art Gestaltung wird Aseprite eingesetzt.
- Das Spiel soll als eigene Anwendung laufen.
- Zielplattformen sind Desktop-Computer (Windows, macOS, Linux).

Organisatorische Randbedingungen:

- Das Projektteam besteht aus 4 Studierenden, welche von einem Professor betreut wird.
- Die Projektlaufzeit beträgt 6 Monate. Das Projekt begann und wurde am 31.03.2023 angemeldet. Am 10.Juli wurde erstmals ein laufender Prototyp vorgestellt. Das letzte Gespräch und die Abgabe fand am 31.07.2023 statt.
- Die Entwicklung des Projekts erfolgt in einem wiederholenden und schrittweisen Prozess, der alle zwei Wochen stattfindet.
- Jeder Zyklus beginnt und endet mit einer Besprechung.
- Bevor wir mit der Entwicklung beginnen, legen wir die grundlegenden Funktionen fest, die unser Produkt haben muss (das sogenannte Minimum Viable Product oder MVP) und erfassen die Anforderungen, die das Projekt erfüllen soll.
- Alle Fortschritte, Ziele oder Probleme werden in einem Gruppenchat kommuniziert und kategorisiert

Entwicklungswerkzeuge

Unser Projekt nutzt Github als Entwicklungsplattform mit der Versionsverwaltung Git. Github dient hauptsächlich der Verwaltung des Quellcodes sowie für Pull Requests.

Wir haben unsere Aufgabenaufteilung als Gruppe in Discord kommuniziert und dokumentiert. Wir haben hierfür einen Server erstellt, bei dem wir verschiedene Kanäle für Aufgaben, Termine, Probleme, Fortschritte und etc. hinzugefügt haben.

Adobe Illustrator und Photoshop: Diese Programme wurden verwendet, um hochwertige Grafiken und Assets für das Spiel zu erstellen.

Aseprite: Aseprite ist eine spezialisierte Software für die Pixelgrafikbearbeitung, die für die Erstellung von Pixelart-Grafiken häufig verwendet wurde.

Für die Überprüfung unseres Fortschritts haben wir stets die neueste Version des Projekts abgerufen und diese in Unity getestet.

Dokumentation

Die Verwendung des deutschsprachigen arc42-Templates ermöglicht eine strukturierte und einheitliche Dokumentation der Architektur des Projekts.

Clean Code

Es werden uns vorgestellte und bekannte Design Patterns sowie Best Practices umgesetzt, um einen sauberen, wart- und erweiterbaren Quellcode zu produzieren

Sprache

Der Quellcode, einschließlich der Kommentare, wird vollständig in Englisch verfasst. Bei Commit-Nachrichten und Merge-Requests hatten alle Gruppenmitglieder die Wahl zwischen Englisch und Deutsch, da sie beide Sprachen gut genug beherrschen. Die Dokumentation wurde hingegen vollständig auf Deutsch verfasst.

Kontextabgrenzung

Schnittstelle	Bedeutung	Technischer Kontext
Benutzer	Der Benutzer kann Inputs ans Spiel senden, die Auswirkungen auf das Spiel haben.	Menschlicher Benutzer
Unity	Game Engine für die Entwicklung des Spiels	Unity Game Engine
Visual Studio	Integrated Development Environment (IDE) für die C#-Skriptentwicklung	Visual Studio
Adobe Programme	Zur Erstellung von Grafiken und Assets	Adobe Programme
Aseprite	Für die Pixel Art Gestaltung	Aseprite

Fachlicher Kontext

Yekta Leon Kücük	• Projektorganisation • Unity-Entwicklung • C#-Skripte schreiben • Level Design
Mustafa Günes	• Unity-Entwicklung • C#-Skripte schreiben • Level Design
Aykon Kücük	• Grafik Design • Logo Design • Game Elemente entworfen und gestaltet • Entwurf des Plakates zur MediaNight
Rometh Umic-Senol	• Grafik Design • Titelbildschirm Design • In-Game UI Design und Funktionalität

Entwurfs- und Architekturmuster

Für jedes Objekt haben wir ein entsprechendes C#-Skript erstellt, das die spezifischen Funktionen und Aktionen des Objekts enthält. Zum Beispiel gibt es für das Player-Objekt mehrere Skripte wie ein Movement-Skript oder Health-Skript.

Unser Team hat alle Grafikobjekte eigenständig gestaltet, darunter Charaktere, Gegner, Hintergründe und Level-Tile-Objekte, die wir selbst entworfen und implementiert haben. Wir haben hierzu Anfangs klare optische Vorstellungen festgelegt und uns an diesen orientiert.

Technologieentscheidungen

Das Projekt wird hauptsächlich in Unity entwickelt. Unser Code, also die Skripte, wurden hauptsächlich in der Programmiersprache C# geschrieben.

Für die Erstellung der Grafiken haben wir die Adobe Programme Illustrator und Photoshop verwendet. Der Großteil der Sprites wurde jedoch in Aseprite entworfen und anschließend in Unity implementiert.

Organisatorische Entscheidungen zur Erreichung der wichtigsten Qualitätsanforderungen

Das Projekt wurde zu Beginn auf GitHub erstellt. Damit konnten alle Teammitglieder parallel entwickeln und anschließend in das main repository mergen.

Das Erstellen eines Servers, auf dem wir kommunizieren konnten, hat uns dabei geholfen, die Aufgaben aufzuteilen und stetig den aktuellen Stand zu kommunizieren. Hierfür haben wir Discord verwendet. Auf unserem eigenen Server haben wir dann immer wieder die Aufgaben jedes Teammitglieds notiert, Probleme angesprochen und Ziele dokumentiert. Wir haben ein wöchentliches Meeting ausgemacht, in dem wir uns über den aktuellen Stand unserer Arbeit, aufgekommene Fragen und neue Aufgaben ausgetauscht haben.

Querschnittliche Konzepte

Erweiterbarkeit

Unser Spiel ist darauf ausgelegt, leicht erweitert zu werden. Wir bieten die Möglichkeit, viele weitere Level einzubauen und verschiedene Erweiterungen wie Power Ups oder ähnliches hinzuzufügen. Auch das Einfügen neuer Hindernisse und Gegner ist dank vorhandener Objekte und Skripte einfach möglich. Die Verwendung einer Tile-Map ermöglicht es uns, unzählige weitere Level ohne Probleme zu gestalten und das Spielerlebnis kontinuierlich zu erweitern.

Enemy AI

Wir haben eine Enemy AI im Skript definiert, welche den Benutzer erkennt, wenn er nah genug dran ist und diesen dann verfolgt. Er unterbricht die Verfolgung, wenn er einen bestimmten Stop Point erreicht. Wenn der Player nicht erkannt wird, dann hat jeder Enemy einen Patrouillenweg, den er auf und abläuft.

Animationssystem

Ein weiteres essentielles Konzept unseres Spiels ist das Animationssystem. Hierbei haben wir besonderen Wert darauf gelegt, Animationen wie Idle-, Sprung- und Laufbewegungen nahtlos und flüssig zu integrieren. Die Animationen tragen maßgeblich zum immersiven Spielerlebnis bei und verleihen den Charakteren und Umgebungen eine lebendige und visuell ansprechende Darstellung. Die einzelnen Animationen wurden auch von unserem Team entworfen.

Audio- und Sounddesign

Wir haben noch passende Hintergrundmusik hinzugefügt und Sound-Effekte bei verschiedenen Aktionen wie das Springen oder das Zerstören eines Gegners.

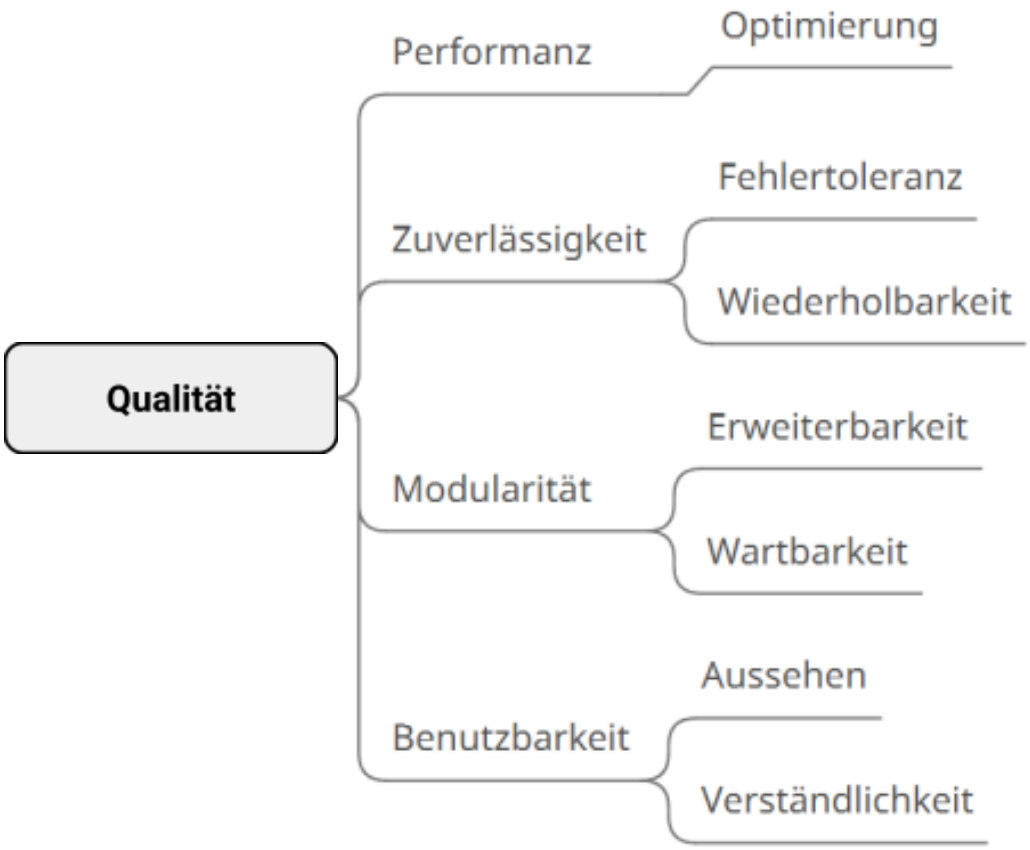
Benutzeroberfläche (UI/UX)

Wir haben ein Hauptmenü erstellt für den User und ein Pausemenü. Zudem haben wir eine In-Game-UI für die Lebensanzeige und die Anzahl der gesammelten Shards.

Entwurfsentscheidungen

Zweck	Getroffene Entscheidungen	Alternativen
Game Engine	**Unity:** Sehr populäre Engine, relativ zu anderen Engines deutlich einsteigerfreundlicher, Vorerfahrungen mit der Engine	**Unreal**: Größerer Funktionsumfang als Unity, aber deutlich steilere Lernkurve **Godot**: Guter Funktionsumfang, Open Source, aber bietet durch die kleinere Community weniger Startpunkte zum Lernen
Pixel Art	**Aseprite:** Aseprite ist eine beliebte und leistungsstarke Software, die speziell für die Erstellung von Pixel Grafiken und Animationen entwickelt wurde. Die Software bietet viele hilfreiche Tools, um ansprechende Sprites erstellen zu können.	**GIMP:** kostenloses und Open-Source-Bildbearbeitungsprogramm, das aber nicht speziell für Pixelart ausgelegt ist. **GraphicsGale:** benutzerfreundliche Software speziell für Pixelart. Leider nur für Windows verfügbar.
User Interface und Grafikdesign	**Adobe Photoshop:** Leistungsstarkes Grafikbearbeitungsprogramm für UI-Designs. **Adobe Illustrator:** Vektorbasierte Software für skalierbare UI-Elemente.	**GIMP:** kostenloses und Open-Source-Bildbearbeitungsprogramm, das aber nicht speziell für Pixelart ausgelegt ist. **Affinity Photo:** Leistungsstarkes Bildbearbeitungsprogramm mit hoher Benutzerfreundlichkeit und bezahlbarer Einmalzahlung statt Abonnement.

Qualitätsanforderungen



Qualitätsszenarien

Ziel	Szenario
Modularität	Ein Entwickler will einen neuen Charakter oder neue Power-Ups in das Spiel einfügen. Durch die Separation von Charakter und Items kann der Entwickler die neuen Features implementieren, das neue 3D-Modell und die Animationen importieren und die gewünschten Items direkt in die Level implementieren.
Performanz	Ein Benutzer spielt auf einem älteren PC. Da das Spiel durch die niedrige Anzahl an Polygonen und Events geringere Anforderungen an die Leistung der Hardware stellt, kann der Nutzer das Spiel dennoch in einem annehmbaren Rahmen spielen.
Zuverlässigkeit	Ein Spieler steht vor einer schwierigen Jump 'n' Run-Passage im Level, in der schnelle oder viele Inputs gleichzeitig erforderlich sind. Um sicherzustellen, dass das Spiel zuverlässig reagiert, muss das Ergebnis der Aktion dasselbe sein, unabhängig davon, ob der Spieler schnelle und viele Inputs oder nur einzelne Inputs ausführt. Die Steuerung muss präzise und reaktionsschnell sein, um eine konsistente Spielerfahrung zu gewährleisten.

Projektverlauf und Probleme

Problem: Zur Entwicklung in Unity benötigt man Tiles, die in eine Tile Palette importiert und dann im Level eingesetzt werden. Da aber das Erstellen aller Designs viel Zeit in Anspruch nimmt, würde es zusätzliche Zeit in Anspruch nehmen, auf die fertigen Sprites zu warten, bevor wir mit der Entwicklung in Unity beginnen können. *Lösung:* Um Zeit zu sparen, haben wir "Development Sprites" verwendet, bis das Design Team seine Arbeit erledigt hat. Mithilfe dieser einfachen Sprites konnten wir direkt mit dem Levelbau beginnen und begannen, Skripte für das Spielerbewegungssystem, Items usw. zu schreiben.
Problem: Während der Reviews von Nutzern anlässlich der Media Night sind uns einige Probleme zur Benutzerfreundlichkeit aufgefallen. An einigen Stellen wurde das Spiel als zu schwierig oder sogar unfair wahrgenommen, vor allem aufgrund des Level Designs. Diese Rückmeldungen haben wir erst nach Tests mit einer Vielzahl von Nutzern unterschiedlichen Alters erhalten, da sie während der internen Entwicklung nicht offensichtlich waren. Lösungsansätze: User Testing: Ausführen von regelmäßigen Tests mit einer breiten Palette von Nutzern, um frühzeitig Feedback zu erhalten und potenzielle Probleme zu identifizieren, bevor das Spiel fertiggestellt wird.
Problem: Kollisionen können zu fehlerhaften Berechnungen und unerwartetem Verhalten führen, wenn die Kollisionsformen der GameObjects nicht korrekt konfiguriert sind. In unserem Projekt sind bestimmte Fälle aufgetreten, bei denen solche Kollisionsprobleme auftraten.
Problem: Unsere Architektur erwies sich als unmodular, da wir mit großen, überladenen Skripten arbeiteten. Dies führte zu Schwierigkeiten bei der Implementierung neuer Funktionen und beeinträchtigte die Skalierbarkeit des Projekts.

Erfahrungen und angeeignetes Wissen

Level Design: Wir haben Erfahrungen im Design von Leveln gesammelt und erkannt, wie wichtig eine ausgewogene Schwierigkeitskurve für ein unterhaltsames Spielerlebnis ist.

Benutzerfreundlichkeit: Durch Nutzerreviews und Tests haben wir die Bedeutung einer benutzerfreundlichen Gestaltung des Spiels erkannt und gelernt, wie wir die Schwierigkeit anpassen können, um ein angenehmes Spielerlebnis zu gewährleisten.

KI-Entwicklung: Die Arbeit an der Gegner-KI hat uns gezeigt, dass die Entwicklung intelligenter und herausfordernder Gegner eine komplexe Aufgabe ist, bei der verschiedene Strategien und Verhaltensmuster berücksichtigt werden müssen.

Kollisions- und Hitbox-Optimierung: Wir haben gelernt, wie wichtig präzise Kollisionen und Hitboxen sind, um ein reibungsloses Gameplay und eine genaue Interaktion zwischen Objekten sicherzustellen.

Grafikdesign: Wir haben uns mit dem Erstellen von Grafiken und grafischen Objekten für unser Spiel auseinandergesetzt. Dies beinhaltet das Design von Charakteren, Hintergründen, Animationen und visuellen Effekten. Dabei haben wir neue Techniken und Werkzeuge im Grafikdesign kennengelernt. Besonders der Bereich Pixelart bringt neue Herausforderungen mit sich, die auch bei vorheriger Erfahrung mit Grafikdesign eine komplett neue Umgebung bedeuten. Hierbei stellt die Lernkurve in erster Linie das Verständnis für das richtige Maß an benötigten Details dar, da bei Pixelart gilt: Mehr Detail bedeutet nicht unbedingt ein besseres Design. Auch wenn ein Objekt im Einzelnen in Ordnung erscheint, besteht die Möglichkeit, dass es als Game-Objekt “unpassend” wirkt. Eine solche Situation wirkt sich einerseits auf die zu hohe benötigte Zeit pro Objekt aus, als auch auf die “Stimmigkeit” des gesamten Erscheinungsbildes der Software. Man sollte sich also zu Beginn auf eine feste Auflösung abhängig von der tatsächlichen Größe der Objekte festlegen. Letztendlich führt, um das Verständnis dafür zu entwickeln, kein Weg an der schlichten Erfahrung vorbei.

C#-Programmierung: Wir haben unsere Kenntnisse in der Programmiersprache C# erweitert und neue Konzepte und Techniken erlernt, um das Gameplay und die Funktionalitäten im Spiel zu implementieren. Gleichzeitig haben wir frühere Erfahrungen mit C# aufgefrischt und angewendet.