Anis Talic , Arsenii Gukasov, Taha

FOS21

Platformer

Einführung

Das Pflichtenheft beschreibt die detaillierte Umsetzung der Anforderungen aus dem Lastenheft für einen 2D-Platformer. Für die Erstellung des Spiels wird die Programmiersprache C# verwendet.

Beschreibung des Istzustands

- Es existiert kein bestehendes Projekt.

Beschreibung des Soll-Konzepts

- Es soll ein Spiel im Stil von Souls-Spielen mit endlosen Levels entwickelt werden.

Beschreibung von Schnittstellen

- Keine externen Schnittstellen: Das Spiel läuft standalone, ohne Netzwerkfunktionen.
- Interne Schnittstellen: Es müssen Klassen für Spieler, Gegner, Levels und die Benutzeroberfläche erstellt werden.

Funktionale Anforderungen

- **Flüssige Steuerung über Tastatur:**
    - Die Tasten für die Bewegung des Charakters WASD müssen konfiguriert sein. Leertaste zum Springen und Taste zum Angreifen.
- **Gegner-Erscheinung und Interaktion:**
    - Gegner sollten zufällig mit einer Zufallsfunktion in Intervallen von etwa 1 Minute erscheinen. Für Gegner sollte das KI-Verhalten festgelegt werden.
- **Automatische Level-Generierung:**
    - Die Levels sollten zufällig erstellt werden und sich nicht wiederholen, das Spiel sollte endlos sein und nicht enden, der Schwierigkeitsgrad sollte mit der Zeit zunehmen.
- **Zwei Charakterklassen:**
    - Es sollten definitiv zwei Klassen definiert sein, für die der Spieler spielen kann. Die erste Klasse ist der Krieger, der über einen starken Angriff verfügt, aber auf den Nahkampf beschränkt ist, die zweite ist der Magier, der aus der Ferne kämpfen kann, aber im Nahkampf schwächer ist. Für beide Klassen sollten eigene Fähigkeiten und Waffen ausgearbeitet werden

Benutzbarkeit

- Kostenlos auf itch.io:
- Niedrige Leistungsanforderungen: für Low-End-Hardware (z. B. keine High-Res-Grafiken, effiziente Loops).
- Leichte Verständlichkeit: ein Tutorial-Menü oder On-Screen-Hinweise hinzu (z. B. "Drücke Leertaste zum Springen").

Zuverlässigkeit

- Es muss eine Fehlerbehandlung für die Ein- und Ausgabe implementiert werden, Fehler wie das Verlassen der Karte oder das Respawnen an der falschen Stelle müssen behandelt werden, nach jeder Entwicklungsphase muss ein Test durchgeführt werden, um die Stabilität zu gewährleisten.

Effizienz

- Das Projekt soll dem Nutzer auch auf sehr schwachen Systemen durch Code-Optimierung ein flüssiges Spielerlebnis bieten.

Wartbarkeit

- Der Code sollte strukturiert sein und das Projekt sollte sich an die grundlegende OOP-Struktur halten.

Risikoakzeptanz

- Keine speziellen Vorgaben

Skizze des Entwicklungszyklus und der Systemarchitektur

- **Entwicklungszyklus (Wasserfallmodell):**
    - Anforderungsanalyse: Basierend auf Lastenheft, detaillieren in diesem Pflichtenheft.
    - Entwurf: Skizzieren Sie UML-Diagramme für Klassen und Flowcharts für Game-Loop.
    - Implementierung: Codieren Sie schrittweise: Zuerst Core (Steuerung, Physik), dann Features (Gegner, Levels), zuletzt UI/Highscore.
    - Test: Unit-Tests, Integrationstests und Beta-Tests auf Fehler.
    - Veröffentlichung: Kompilieren, hochladen auf itch.io und announcen.
- **Systemarchitektur:**
    - Main: Update, Render, Input-Handling.

    - Komponenten: Scene-Manager für Menüs/Levels, Entity-System für Spieler/Gegner, Procedural Generator für Levels.
    - Technologien: C#

Lieferumfang

- Fertiges Spiel als Download: .exe-Datei
- Installationsdatei: Selbstextrahierendes Archiv oder einfaches Zip mit Anleitung.

Abnahmekriterien

- Stabile Version auf itch.io
- Keine Fehler: Vollständige Umsetzung der Anforderungen, getestet auf mehreren Systemen.
- Keine zwingenden Updates: Das Spiel ist standalone und funktioniert ohne Patches.