Shadow Tag

First-Person-Shooter
Interaktive Lasertag-Geisterbahn
Technische Beschreibung

Herberich Sandro
Bahnhofstraße 4
76698 Ubstadt-Weiher

Abgabedatum: 19.04.2023

Inhalt

1. Manager

1.1 Game Manager

Der Game Manager soll sich um die Einstellungen die der Spieler trifft kümmern. Dazu gehören die Sensitivity Einstellungen von Maus und Controller außerdem die FOV und Gamma Einstellungen. Von hier aus wird auch das Ingame Menu gesteuert, um die Cut Scenes zu überspringen.

Zu erstellen ist eine zentrale GameManager-Komponente in Unity, die verschiedene Funktionen bereitstellt, um das Spiel zu steuern. Das Skript sollte als Singleton implementiert werden, um sicherzustellen, dass es nur eine Instanz im Spiel gibt. Es sollte Empfindlichkeit und Videoeinstellungen verwalten, ein Menüsystem bereitstellen, das das Spiel anhalten und den Cursor steuern kann, eine Funktion zum Anwenden der Gamma-Korrektur auf das Volume-Objekt und eine Funktion zum Überspringen des Intro-Videos und Aktivieren des Spielers und der UI enthalten.

1.2 Audio Manager

Erstellen Sie eine zentrale AudioManager-Komponente in Unity, die das Audio-Management des Spiels steuert. Das Skript sollte als Singleton implementiert werden, um sicherzustellen, dass es nur eine Instanz im Spiel gibt. Es sollte eine Methode enthalten, um die Lautstärke-Einstellungen des Spiels zu steuern und eine Referenz auf das AudioMixer-Objekt haben, das verwendet wird, um die Audio-Effekte im Spiel zu mischen. Die SetAudio-Methode sollte die Lautstärke-Einstellungen aus dem Settings-Skript erhalten und auf das AudioMixer-Objekt anwenden.

1.3 Scenes Manager

Zu erstellen ist eine zentrale ScenesManager-Komponente in Unity, die verschiedene Funktionen bereitstellt, um Szenen im Spiel zu steuern. Das Skript sollte als Singleton implementiert werden, um sicherzustellen, dass es nur eine Instanz im Spiel gibt. Es sollte eine Funktion zum Laden von neuen Spielen und zum Laden von Szenen bereitstellen, eine Ladebildschirm-UI mit Fortschrittsanzeige enthalten, ein Menüsystem bereitstellen, das das Spiel anhalten und den Cursor steuern kann, eine Funktion zum Aktualisieren der Audio- und Videoeinstellungen enthalten und beim Laden einer neuen Szene die Player-UI entsprechend ändern.

1.4 UI-Steuerung und Einstellungen

1.4.1 Settings

Zu erstellen ist eine Settings-Komponente in Unity, die verschiedene Optionen für das Spiel bietet. Das Skript sollte als Singleton implementiert werden, um sicherzustellen, dass es nur eine Instanz im Spiel gibt. Die Komponente sollte Schieberegler und Buttons enthalten, um die Empfindlichkeit, das Sichtfeld, die Gamma-Korrektur und die Audioeinstellungen zu steuern.
Ein Zurück-Button sollte hinzugefügt werden, um das Menü zu schließen und zum Spiel zurückzukehren. Die Komponente sollte auch eine Funktion haben, um das Fenster des Menüs zu deaktivieren, wenn es geschlossen wird, und eine Funktion, um das Spiel anzuhalten, wenn das Menü geöffnet ist.

1.4.2 Credits

Zu erstellen ist eine Credits-Komponente in Unity, die eine Schaltfläche zur Verfügung stellt, um das Fenster zu schließen. Das Skript sollte eine Methode bereitstellen, um das Credits-Fenster zu deaktivieren, wenn die Schaltfläche gedrückt wird. Die Methode sollte auch das Spiel pausieren und den GameManager darüber informieren, dass das Spiel nicht mehr im Menümodus ist.

1.4.3 UIMainMenu

Zu erstellen ist ein UIMainMenu-Skript in Unity, das ein Hauptmenü bereitstellt, um das Spiel zu starten, Einstellungen zu ändern, das Spiel zu beenden und die Credits anzuzeigen. Es sollte als Singleton implementiert werden, um sicherzustellen, dass es nur eine Instanz im Spiel gibt. Das Skript sollte auch verschiedene Buttons enthalten, um das Spiel zu starten, das Spiel zu laden, Einstellungen zu ändern, das Spiel zu beenden und die Credits anzuzeigen. Es sollte auch das Laden von Spielen und das Verwalten von Audioeinstellungen ermöglichen.

1.4.4 UIGameMenu

Zu erstellen ist ein UIGameMenu-Skript in Unity, das ein Ingamemenü im Spiel bereitstellt. Das Skript sollte verschiedene Funktionen bereitstellen, wie z.B. die Möglichkeit, das Spiel fortzusetzen, das Hauptmenü zu laden, das Spiel zu speichern und Einstellungen zu ändern. Es sollte auch eine Schaltfläche für Credits bereitstellen, um den Spielern Anerkennung zu geben. Das Skript sollte mit den verschiedenen UI-Elementen wie Buttons und Textfeldern verknüpft werden, um die Funktionen zu steuern. Außerdem sollte das Skript als Singleton implementiert werden, um sicherzustellen, dass es nur eine Instanz im Spiel gibt.

2. Player

Es wird für das Gesamte Projekt mit Ausnahme des (Esc) für das Ingame Menu das Neue Input System verwendet

2.1 Player Look

Zu erstellen ist eine PlayerLook-Komponente in Unity, die es dem Spieler ermöglicht, sich in der Spielwelt umzusehen. Das Skript sollte eine Referenz auf die Kamera des Spielers haben und die Empfindlichkeit der Mausbewegungen sowohl horizontal als auch vertikal steuern. Außerdem sollte es eine Funktion bereitstellen, um das Sichtfeld des Spielers zu ändern. Das Skript sollte auch sicherstellen, dass der Cursor im Spiel gesperrt ist, um eine unterbrechungsfreie Spielerfahrung zu gewährleisten.

2.2 Player Movement

Zu erstellen ist eine PlayerMovement-Komponente für die Bewegung des Spielers in Unity, dass die grundlegende Steuerung des Spielers ermöglicht. Das Skript sollte den CharacterController verwenden, um die Bewegung des Spielers zu steuern und die Eingaben des Spielers aus dem InputManager verarbeiten. Es sollte dem Spieler erlauben, zu laufen, zu springen, sich zu ducken und zu sprinten. Das Skript sollte auch eine Möglichkeit bieten, die Position des Spielers zu speichern und zu laden, um den Fortschritt des Spielers zu speichern und wiederherzustellen.

2.3 Player UI

Zu erstellen ist eine Klasse namens "PlayerUI", die von der "MonoBehaviour" Klasse erbt und das "IDataPersistance" Interface implementiert. Die Klasse sollte eine öffentliche statische Instanzvariable "Instance" haben. Die Klasse sollte private Variablen für die aktuelle Punktzahl, das aktuelle Labyrinth-Level und die TextMeshProUGUI-Objekte für die Anzeige der Meldungen, Punktzahlen, Munition und des aktuellen Labyrinth-Levels haben. Die Klasse sollte eine "Awake()" Methode haben, die sicherstellt, dass es nur eine Instanz der Klasse gibt. Die Klasse sollte auch eine "Start()" Methode haben, die das Objekt der Liste der "dataPersistenceObjects" im "SaveManager" hinzufügt und die Daten lädt. Die Klasse sollte Methoden zum Aktualisieren der Punktzahl, Munition und Textanzeige haben. Außerdem sollte es Methoden zum Speichern und Laden von Daten geben. Die "OnDisable()" Methode sollte das Objekt aus der Liste der "dataPersistenceObjects" im "SaveManager" entfernen.

2.4 Input Manager

Zu erstellen ist eine Klasse "InputManager", die das MonoBehaviour erweitert. Die Klasse soll ein Singleton-Muster verwenden und eine statische Instanz "Instance" bereitstellen. Des Weiteren soll die Klasse über ein PlayerInput-Objekt verfügen und die Player-Actions "onWalk" und "onWeapon" definieren. Außerdem soll die Klasse über eine Referenz auf das Script "PlayerMovement", "PlayerLook" und "GunManager" verfügen. Es soll eine Funktion zum Einschalten der Taschenlampe ("TurnOnFlashlight") implementiert werden und die verschiedenen Player-Actions sollen entsprechend mit den passenden Funktionen der anderen Skripte verbunden werden.

2.5 Player Interact

Zu erstellen ist eine Klasse namens "PlayerInteract", die die Interaktion des Spielers mit Objekten in der Spielwelt ermöglicht. Die Klasse sollte ein "Camera"-Feld haben, das auf die Kamera des Spielers verweist, sowie "distance" und "mask" Felder, um die Entfernung und Ebene der Interaktion zu steuern. Die Klasse sollte auch eine Referenz auf die "PlayerUI" und "InputManager" Klassen haben, die benötigt werden, um die Interaktion mit dem Spieler zu steuern.

Genauere Ausführung:

In der "Update()" Methode sollte die Klasse eine Raycast-Abfrage durchführen, um zu bestimmen, ob sich ein interaktives Objekt in Reichweite des Spielers befindet. Wenn ja, sollte die Klasse die "UpdateText()" Methode der "PlayerUI" Klasse aufrufen, um dem Spieler eine Nachricht anzuzeigen, die ihn zur Interaktion auffordert. Wenn der Spieler die Interaktion auslöst, sollte die Klasse die "BaseInteract()" Methode des "Interactable"-Objekts aufrufen, um die Interaktion durchzuführen.

3. Interactable

Zu implementieren ist eine abstrakte Klasse Interactable, welche eine Basis-Interaktionsfunktion bereitstellt und eine promptMessage Eigenschaft hat, welche eine Nachricht speichert, die dem Spieler angezeigt werden kann.

Die Klasse Interactable soll folgende Funktionen bereitstellen:

- Eine promptMessage Eigenschaft, welche eine Nachricht speichert, die dem Spieler angezeigt werden kann.
- Eine BaseInteract Funktion, welche eine abstrakte Methode Interact aufruft.
- Eine Interact Methode, welche in der abgeleiteten Klasse überschrieben werden kann.

Die promptMessage Eigenschaft speichert eine Nachricht, die dem Spieler angezeigt werden kann, um ihm mitzuteilen, dass er mit einem interaktiven Objekt interagieren kann. Die BaseInteract Funktion ruft die abstrakte Methode Interact auf, welche in der abgeleiteten Klasse überschrieben werden muss, um spezifische Interaktionsfunktionen bereitzustellen.

3.1 Interactable Objects

Die Objekte mit welchen der Spieler Interagieren kann sollen von Interactable erben. Je nach Einsatz soll das Objekt spezifisch angepasst werden. Als Beispiel eine Tür die sich öffnen lassen soll.

Zu erstellen ist eine Klasse mit dem Namen "Door", welche von der abstrakten Klasse "Interactable" erbt. In der Klasse sollen folgende Variablen definiert werden:

- Ein GameObject namens "handel"
- Ein boolscher Wert mit dem Namen "isLeft"
- Ein boolscher Wert mit dem Namen "isRight"
- Ein GameObject mit dem Namen "house"
- Ein Animator mit dem Namen "currentAnimator"
- Ein AudioTrigger mit dem Namen "audioTrigger"

Des Weiteren soll in der Start-Methode der aktuelle Animator des Hauses durch "GetComponent" zugewiesen werden. Außerdem soll die Methode "Interact" überschrieben werden, um spezifische Interaktionslogik für die Tür zu implementieren.

Innerhalb der "Interact"-Methode soll überprüft werden, ob der Anker der Tür eine bestimmte Rotation hat und ob die Tür links oder rechts ist. Basierend auf diesen Bedingungen soll der aktuelle Animator entsprechend ausgelöst werden, um die Tür zu öffnen oder zu schließen. Zusätzlich soll der AudioTrigger ausgelöst werden.

4. Enemy

Zu erstellen ist eine Klasse namens "Enemy", die von der Klasse "Interactable" erbt. Die Klasse "Enemy" soll Variablen für Gesundheit, Punkte, Lebenszeit, Schleimvarianten, Texturen und Audio enthalten. Die Klasse "Enemy" soll eine Methode zum Hinzufügen von Schaden, eine Coroutine für die Lebenszeit des Gegners und eine Methode zum Aktualisieren der Punkte des Spielers beinhalten. Außerdem soll die Klasse "Enemy" eine Methode enthalten, die verschiedene Audio-Clips abspielt, um verschiedene Situationen wie Erschrecken, Treffer und Tod zu untermauern.

Für Schleim-Gegner soll die Klasse "Enemy" eine Methode zum zufälligen Auswählen der Schleimvariante und zum Setzen der Schleimtextur enthalten. Wenn bestimmte Schleimvarianten ausgewählt werden, sollen auch die Punkte des Spielers entsprechend angepasst werden.

5. Save System

Das Save System soll aus 4 Skripten Bestehen:

5.1 SaveManager

Zu erstellen ist eine Klasse namens "SaveManager" in Unity, die das Speichern und Laden von Spielständen ermöglicht. Die Klasse enthält eine Liste von Objekten, die das IDataPersistance-Interface implementieren, um sicherzustellen, dass alle relevanten Daten bei einem Speichervorgang gespeichert werden. Die Klasse enthält auch Methoden zum Starten eines neuen Spiels, zum Laden eines Spielstands und zum Speichern des aktuellen Spielstands. Wenn das Spiel beendet wird, wird automatisch ein Speichervorgang durchgeführt.

5.2 SaveData

Zu erstellen ist eine Klasse mit dem Namen "SaveData", diese Klasse soll die gespeicherten Daten eines Spiels repräsentieren und serialisierbar sein. Die Klasse enthält die folgenden öffentlichen Felder und Eigenschaften:

- "currentPoints": Ein ganzzahliger Wert, der die Punktzahl des aktuellen Levels speichert.
- "currentLabyrinthLevel": Ein ganzzahliger Wert, der das aktuelle Level des Labyrinths speichert.
- "playerPosition": Eine Vector3-Struktur, die die aktuelle Position des Spielers im Spiel speichert.
- "unlockedWeapons": Eine Liste von booleschen Werten, die speichert, welche Waffen im Spiel freigeschaltet sind.
- "unlockedLevels": Eine Liste von booleschen Werten, die speichert, welche Level im Spiel freigeschaltet sind.

Zudem soll die Klasse eine Standard-Konstruktorfunktion haben, die die oben genannten Felder mit Standardwerten initialisiert. Die Klasse soll durch die Verwendung des "[System.Serializable]"-Attributs serialisierbar gemacht werden.

5.3 FileSaveDataHandler

Zu erstellen ist eine Klasse "FileSaveDataHandler". Diese Klasse ist für das Laden und Speichern von Daten in einer Datei zuständig. Sie benötigt zwei Parameter: dataDirPath, der den Pfad zum Ordner angibt, in dem die Datei gespeichert werden soll, und dataFileName, der den Namen der Datei angibt, in der die Daten gespeichert werden sollen. Die Klasse verfügt über eine "Load()" Methode, die die Daten aus der Datei lädt und eine "Save()" Methode, die die Daten in der Datei speichert. Die Daten werden im JSON-Format gespeichert. Die Klasse enthält Fehlerbehandlung und Debugging-Informationen.

5.4 (Interface) IDataPersistance

Zu erstellen ist eine Schnittstelle namens "IDataPersistance", die von anderen Klassen implementiert werden kann. Die Schnittstelle muss zwei Methoden enthalten: "LoadData" und "SaveData". Diese Methoden sollen es ermöglichen, Daten aus verschiedenen Quellen zu laden und zu speichern.

6. Gun und Gun Manager

6.1 Gun Manager

Der GunManager soll die Interaktion zwischen dem Spieler und den Waffen bereitstellen.

Zu erstellen ist eine Klasse namens GunManager. Sie sollte die Gesamten Waffen Funktionen verwalten. Hierzu zählen schießen, nachladen, Waffe wechseln und Waffe freischalten. Es soll auch die Möglichkeit bestehen die freigeschalteten Waffen zu speichern und zu laden. Am Anfang sollen alle Waffen erzeugt werden und wenn sie erfolgreich erzeugt wurden gespeichert werden es wird außerdem immer das Aktuelle Gun Skript wie auch der Animator und AudioSource der aktuellen Waffe gespeichert werden.

Von dem GunManager soll eine Instanz erstellt werden und es soll sichergestellt werden, dass dieser nur einmal existiert.

6.2 Gun

Auf jedem Waffen Prefab soll ein Gun Skript eingesetzt werden.

In diesem Script werden hauptsächlich die werte der Waffe gesichert die dann von GunManager verwendet werden sollten es soll lediglich eine Methode geben. Diese dient dazu die HDR-Farbe zu ändern je nachdem wie voll das Magazin der Waffe ist.

Die Variablen die Benötigt werden sind:

- gunVariant (ein enum dass den Gun Type angibt)
- (bool) weaponUnlocked, weaponActive
- (Material) neon
- (Color) currentColor

  <u>Weapon Specks</u>
- (int) damage, ammunition, ammunitionMax
- (float) range, reloadTime, schootCooldown

  <u>Audio Clips</u>
- (AudioClip) reloading, shooting, empty

  <u>Components</u>
- (VisualEffect) effect
- (AudioSource) audioSource
- (Animator) animator

7. Spawner

Der Spawner soll dazu genutzt werden verschiedenste Sachen zu spawnen je nachdem was benötigt wird soll nach isGhostStation, isHub, isPlayer und isWeaponDisplay unterschieden werden.

Je nach Element, dass gespawnt werden soll wird entweder die Position des Spawners verwendet oder die geladene Position des Spielers. Da diese im Hub nicht geladen werden soll, muss dies nur geschehen, wenn der Spieler sich in einem Labyrinth Levels befindet.

Wenn ein Monster gespawnt werden soll, soll der Spawner erst ausgelöst werden, wenn ein bestimmter Bericht mit isTrigger betreten wird. Das Waffen Display soll direkt Spawnen beim erstellen der Map.

8. Portal

Es soll 2 Arten von Portalen geben die sich nicht groß von einander unterscheiden deswegen soll es auch mit einem Skript gelöst werden und nur mit einem bool wert unterschieden werden.

8.1 Die Hub Portale

Diese sollen in ein bestimmtes Level führen, dass zuvor definiert wurde. Dazu soll das aktuelle Level und das Level wohin gereist werden soll im Inspektor auswählbar sein.

Zusatz bei jedem Labyrinth Eingang steht ein Portal, dass zurück in die Hub Welt führt.

8.2 Die Labyrinth Portale

Diese Portale sollen die Funktion erfüllen den Spieler 1 Szene weiter zu führen und das gespeicherte Labyrinth Level um 1 zu erhöhen.