Wirtschaft
Hauptcampus

H O C H
S C H U L E
T R I E R

Medienprojekt: Interdisziplinäre Spielentwicklung

PLATFORM FIGHTER

Betreut von
Prof. Dr. Rezk-Salama und
Prof. Dr. Linda Breitlauch

Name: Gregor Germerodt, Zishan Ahmed
Matrikelnummer: ***
Fachbereich: Informatik
Studiengang: Informatik Digitale Medien und Spiele - Spiele
Abgabetermin: 21.08.2024

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im Rahmen unseres Medienprojekts haben wir das Spiel *Platform Fighter* der *Plattform-Fighter*-Genre entwickelt, inspiriert von bekannten Spielreihen und Titeln wie *Super Smash Bros.*, *Nickelodeon All-Star Brawl* und *Brawlhalla*. Unsere Motivation entsprang einer gemeinsamen Leidenschaft für diese Spiele und dem Wunsch, die technischen Konzepte hinter ihrer Entwicklung besser zu verstehen. Aufgrund der begrenzten Teamgröße, der Abwesenheit von Mitgliedern aus gestaltungsorientierten Studiengängen sowie der parallelen Verpflichtungen zu anderen Universitätsmodulen, setzten wir uns bewusst ein realistisches Ziel: die Entwicklung eines Prototyps im Stil von *Super Smash Bros. 64* (*Abbildung 2*). Dieses frühe Spiel der Reihe zeichnet sich durch einfachere Gestaltung und weniger differenzierte Fighter (Charakter) aus.

Abbildung 1: Dragon King: The Fighting Game (1998) (Super Smash Bros. 64 Prototyp)

Unser Ergebnis ist ein Spiel für zwei Spieler, das sich auf ein Spiegelmatch beschränkt, bei dem beide Spieler einen eigenen Fighter mit dem gleichen Moveset (die Menge aller möglichen Bewegungen eines Fighters) steuern. Trotz der vereinfachten Zielsetzung konnten wir mehrere bemerkenswerte technische Errungenschaften realisieren:

- **ACMD (Animation Commands)-Framework**: Mit diesem selbst entwickelten Framework haben wir die Bewegungsabläufe eines Fighters programmiert.
    - Umgesetzt von Zishan Ahmed

- **Eigenes Inputsystem**: Dieses System verwaltet und weist Controllern und der Tastatur die Steuerung der Fighter zu.
    - Umgesetzt von Zishan Ahmed und Gregor Germerodt

- **Fighter-Controller**: Ein eigens entwickelte Charakter-Controller-Klasse mit individuellen Berechnungen zur Positionsbestimmung der Spielfiguren.
    - Umgesetzt von Zishan Ahmed

- **Unity-Tool – Parametertabelle**: Ein Entwicklungstool, das schnelle Änderungen an den Parametern der Fighter ermöglicht.
    - Umgesetzt von Gregor Germerodt

- **Verwendung von Motion-Capture-Animationen**: Verwendete Animationen wurden mithilfe von Motion-Capture-Technologie aufgenommen, in einer 3D-Anwendung verarbeitet und ins Spiel importiert.
    - Umgesetzt von Zishan Ahmed und Gregor Germerodt

- **Eigenständig modellierte Stage**[1]: Die Stage wurde vollständig in Eigenarbeit in einer 3D-Anwendung modelliert.
    - Umgesetzt von Gregor Germerodt

Mit diesen technischen Entwicklungen konnten wir unsere Kenntnisse im Bereich der Spielentwicklung erheblich erweitern und ein spielbares Produkt erstellen, das unsere ursprünglichen Zielsetzungen größtenteils erfüllt.

[1] Stage – Grundplattform des Kampfgeschehens

2 Spielprinzip

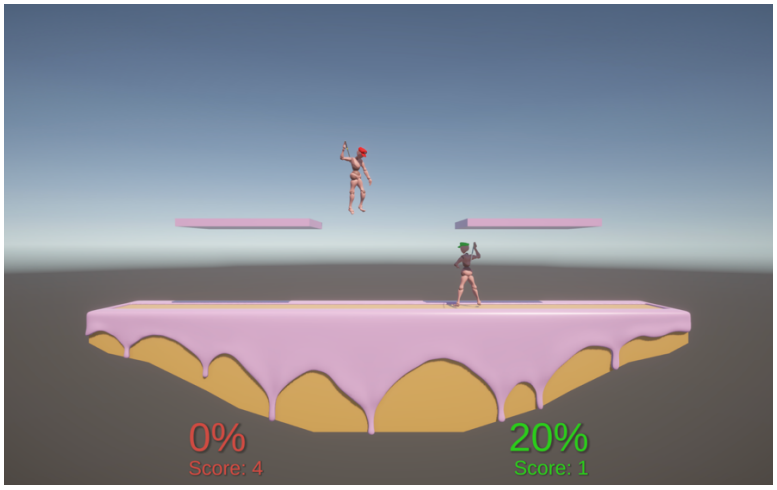

Abbildung 2: Screenshot aus dem Spiel Platform Fighter

In unserem *Platform Fighter* treten zwei Spieler gegeneinander an, um ihre Fähigkeiten im Kampf zu messen. Das Spiel startet direkt nach dem Laden, wobei Spieler 1 den Charakter mit der roten Kappe und Spieler 2 den mit der grünen Kappe steuert. Wie in jedem anderen Spiel der *Plattform-Fighter*-Genre, gilt das Ziel, den Gegner durch Angriffe aus der Arena zu befördern. Jeder Charakter sammelt dabei Schaden in Form von Prozenten, die am unteren Bildschirmrand angezeigt werden. Je höher der Schadenswert, desto weiter fliegt der Charakter bei einem Treffer. Wird ein Spieler aus der Arena geschleudert, bekommt der andere Spieler einen Punkt. Die Prozentanzeige des geschleuderten Spielers wird auf 0 Prozent zurückgesetzt und er kehrt sofort von oben auf die Stage zurück.

Nachdem zwei Eingabegeräte verbunden wurden, ist die Steuerung der Fighter relativ einfach. Die Spieler bewegen ihre Charaktere mit dem linken Stick (oder W, A, S und D auf der Tastatur). Mit den Tasten X und Y auf dem Controller (Xbox-Controller Layout) (oder der Leertaste auf der Tastatur) kann ein Sprung ausgeführt werden. Es wird zwischen zwei verschiedenen Bodensprüngen unterschieden, wird die Sprungtaste nur kurz getippt so wird ein kurzer Sprung ausgeführt, wird die Sprungtaste etwas länger gedrückt so wird ein hoher Sprung ausgeführt. Jeder Fighter verfügt in der Luft über zwei Doppelsprünge. Angriffe werden mit der A-Taste auf dem Controller (oder C auf der Tastatur) ausgeführt. Es gibt sowohl eine Boden- als auch einen Luftangriff, je nach Position des Charakters. Ein weiteres wichtiges Element ist die Möglichkeit, durch Plattformen zu fallen. Dies geschieht, indem der

Spieler den Stick nach unten hält (oder die S-Taste auf der Tastatur gedrückt wird), während er über einer Plattform ist. Sollte das Spiel neu gestartet werden müssen, kann dies durch Drücken der F1-Taste auf der Tastatur erfolgen, wodurch die Szene zurückgesetzt wird und auch die Controllerzuweisungen zurückgesetzt werden.

3 Umsetzung

3.1 ACMD (Animation Commands)-Framework

Für unser Spiel haben wir ein ACMD (Animation Commands)-Framework entwickelt, das auf einem Konzept basiert, welches in der *Super Smash Bros.*-Reihe verwendet wird. Auf dieses Framework sind wir durch die Modding-Community des Spiels *Super Smash Bros. Ultimate* gestoßen. Diese Community hatte eine gewisse Ahnung über die inneren technischen Abläufe des Spiels und veröffentlicht regelmäßig neue Modifizierungen für das Spiel.

Ein ACMD beschreibt eine Bewegung (Move) in einer abstrahierten Methode im Code. In einer Methode ist ein Move in verschiedene Codebereiche (Scopes) unterteilt, welche jeweils nur bei bestimmten Frames des parallellaufenden Frame-Zählers des Spiels ausgeführt werden. Innerhalb eines Scopes wird die Geschwindigkeit des Spielers kontrolliert und aktualisiert, Zustände werden geändert, um in einen anderen Move und somit in ein anderes ACMD zu wechseln, Animationen werden in gesetzten Geschwindigkeiten abgespielt, und Flags () werden gesetzt, um zusätzliche Daten festzuhalten, wie z.B. ob der Spieler bereits alle Doppelsprünge verbraucht hat. Ein Fighter hat stets nur ein aktives ACMD. Zusätzlich zu den ACMDs haben wir auch sogenannte GACMDs (General Animation Commands) eingeführt. Jeden Frame werden alle GACMDs eines Fighters ausgeführt und enthalten Abläufe, die stets ausgeführt werden müssen, wie z.B. der Zustandsautomat, Verbindungen zwischen Zuständen und den entsprechenden ACMDs, sowie Interrupts (z.B., wenn der Spieler sein Lauf mit einem Sprung unterbricht). Das Konzept der GACMDs ist eine eigene Idee von uns und wird vermutlich in dieser Form in den Spielen der *Super Smash Bros*-Reihe nicht verwendet. Alle Flags, States (Zustände), ACMDs und GACMDs müssen vorher deklariert und mit einem String benannt werden, diese bilden gemeinsam ein sogenanntes Moveset. Durch die Benennung der Elemente mit einem String ließen sich die Elemente leichter voneinander unterscheiden und erleichterten zu dem das Debuggen. Es ist theoretisch möglich, Movesets miteinander zu addieren, wodurch es potenziell möglich ist, ein allgemeines Moveset zu programmieren und darauf ein spezielleres Moveset zu konkatenieren, um das redundante Programmieren derselben Abläufe verschiedener Fighter zu vermeiden. Das in diesem Spiel für den Fighter entwickelte Moveset weist eine gewisse Unordnung auf, da die Effektivität der

verschiedenen Code-Praktizierung innerhalb des ACMD-Frameworks bisher von uns nicht vollständig ermittelt werden konnten.

Unser ACMD-Framework stellt eine zentrale Komponente unseres Spiels dar und ermöglicht es uns, komplexe Bewegungsabläufe und Zustandsänderungen zu implementieren und dabei sie möglichst abstrahiert und weniger fehleranfällig zu halten. Trotz der Herausforderungen, wie z.B. die Abwesenheit einer Autovervollständigung der Bezeichnungen der Elemente in der IDE[2] und der noch bestehenden Optimierungspotenziale konnten wir mit diesem Framework einen soliden Ansatz für die Grundlage für das Bewegungsverhalten eines Fighters schaffen.

3.2 Eigenes Inputsystem

Unser Inputsystem ist für die Zuweisung und Verwaltung von Eingabegeräten zuständig. Das Spiel lässt sich sowohl mit Tastatur als auch mit Controller steuern, jedoch wird mindestens ein Controller benötigt, um das Spiel zu zweit spielen zu können. Kurz nach dem Starten des Spiels befindet sich das Inputsystem stets im Zuweisungsmodus: Beim erstmaligen Betätigen einer Taste auf der Tastatur oder eines Knopfes auf einem Controller wird das entsprechende Gerät dem ersten Spieler zugewiesen. Sobald eine Taste oder ein Knopf auf einem anderen Eingabegerät betätigt wird, wird dieses Gerät dem zweiten Spieler zugewiesen, während weitere Controller ignoriert werden. Eine erneute Zuweisung der Eingabegeräte kann erfolgen, wenn der entsprechende Controller getrennt wird, im Falle der Tastatur muss das Spiel neu gestartet werden, entweder durch das Schließen und erneute Öffnen des Spiels oder durch das Drücken der Taste F1. Die Zuweisungsreihenfolge verhält sich größtenteils wie ein Stack: Zuerst wird der erste Spieler zugewiesen, dann der zweite Spieler, und so weiter. Sollte sich der Controller während des Spiels trennen, besteht die Möglichkeit, denselben oder einen anderen Controller anzuschließen. In diesem Fall erfolgt die Zuweisung ebenfalls in der Reihenfolge vom Spieler mit der niedrigsten Nummer bis zur höchsten Spielernummer. Dieses System bietet eine reibungslose und intuitive Methode zur Verwaltung der Eingabegeräte, die insbesondere bei der Verwendung mehrerer Controller wichtig ist.

[2] Integrierte Entwicklungsumgebung (engl. integrated development environment))

3.3 Fighter-Controller

Die Fighter-Controller-Klasse (Charakter-Controller eines Fighters) musste auf eigene Weise umgesetzt werden, da sich die Fighter in unserem 3D-Spiel nur entlang der x- und y-Achse bewegen können. Der Fighter-Controller stellt mehrere Methoden zur Verfügung, um die Geschwindigkeit des Fighters zu setzen, zu addieren oder durch eine gegebene (Gegen-)Beschleunigung die Geschwindigkeit des Fighters einer Geschwindigkeit anzunähern. Diese Methoden werden in den ACMDs und bei Kollisionen verwendet, um den Spieler gezielt und kontrolliert zu steuern. Ein gutes Beispiel hierfür ist das Springen: Der Spieler beschleunigt beim Absprung schneller, als er beim Fallen abbremst. Dies wird erreicht, indem der Fighter beim Absprung eine höhere Beschleunigung erhält und am höchsten Punkt schnell gegenbeschleunigt wird. Danach fällt er mit einer im Vergleich kleineren Gravitationsbeschleunigung und kleineren Endgeschwindigkeit zurück zum Boden. Die Gravitation wird dabei aufgrund von dem Vorteil der erweiterten Kontrolle über den Einfluss von einem GACMD behandelt und nicht vom Fighter-Controller selbst. Der Fighter-Controller ist auch für die Erkennung eines Bodens unter dem Fighter zuständig. Hierfür wird ein einzelner Strahl von knapp über den Füßen bis unter die Füße des Fighters geschossen. Die Länge des Strahls hängt von der aktuellen Geschwindigkeit des Fighters ab, da dieser bei sehr hoher Geschwindigkeit sonst durch den Boden hindurchfallen könnte. Der Fighter-Controller projiziert somit die zukünftige Position des Fighters und prüft, ob auf dem Weg ein Hindernis vorhanden ist. Wird ein Boden unter dem Spieler gefunden, so wird dieser für den aktuellen Frame an die besagte Bodenstelle teleportiert und seine vertikale Geschwindigkeit wird annulliert. Wird unter dem Fighter eine Plattform erkannt, so hat der Spieler durch das Halten des linken Sticks nach unten die Möglichkeit durch die Plattform durchzufallen, um somit leichter auf den Grundboden der Stage zu gelangen. Diese Möglichkeit wurde durch andere Spieler der *Plattform-Fighter*-Genre inspiriert und bietet dem Spieler ein besseres Gefühl der Kontrolle über den Fighter.

3.4 Unity-Tool – Parametertabelle

Die Parametertabelle-Klasse fügt dem *Unity Editor* ein weiteres *Unity*-Tool (vgl. Inspektor, Konsole, etc.) hinzu. Dieses *Unity*-Tool ist eine Tabelle, mit welcher ein Entwickler sämtliche Parameter eines Fighters anzeigen und ändern kann. Die Klasse

liest dafür die Werte aus einer CSV-Datei aus, welche die Werte für die Fighter persistent abspeichert. Alle numerischen Werte der Tabelle sind sowohl temporär als auch permanent änderbar. Weiterhin verfügt die Tabelle über zwei Knöpfe. Der erste Knopf lässt die in der Tabelle stehenden Werte in die CSV-Datei abspeichern und der zweite Knopf setzt die Tabelle auf den Inhalt der CSV-Datei zurück. Alle Änderungen an der Tabelle aktualisieren gleichzeitig die Werte in einem *ScriptableObject* (ein Datencontainer zur Speicherung großer Datenmengen unabhängig von Klasseninstanzen) und haben eine sofortige Auswirkung auf das Spiel. Aus diesem Grund erleichtert die Tabelle das Austesten der Fighter mit neuen Parametern. Der Sinn dahinter ist eine unabhängige Datenbank der einzelnen Fighter-Parameter (CSV-Datei), welche skriptgebunden und bei hoher Anzahl unterschiedlicher Fighter sehr unübersichtlich werden könnten. Da diese Parametertabelle zum Spielen des Spiels nicht gebraucht wird, wird der dazugehörige Code beim Export des Spiels ausgeschlossen. Über die *ScriptableObjetcts* greifen andere *Unity*-Skripts auf die Parameter der Tabelle zu. Zudem werden durch sie, im exportierten Spiel, die zuletzt gespeicherten Werte der Tabelle verwendet.

3.5 Verwendung von Motion-Capture-Animationen

Aufgrund unserer begrenzten Erfahrung in der Erstellung synthetischer Animationen und unseres eigenen Interesses sowie Neugierde, haben wir uns während der Spielentwicklung dazu entschlossen, die Motion-Capture-Technologie zu nutzen. Dafür wurde uns am Gestaltungscampus der Hochschule Trier ein Motion-Capture-Raum zur Verfügung gestellt. Innerhalb von 8 Stunden wurde ein großer Vorrat an Animationen für die Fighter erstellt, inspiriert nach den Move-Animationen aus dem Spiel *Super Smash Bros. Ultimate*, wobei in unserem Spiel nur ein Drittel aller aufgenommenen Animationen genutzt wurden. Die Motion-Capture-Aufnahmen wurden mit der Software *Motive* aufgenommen und später in der 3D-Anwendung *Blender* bearbeitet. Es wurde ein bereits geriggtes[3], humanoides Model von der

[3] Unter Rigging versteht man den Prozess, bei dem ein digitales Skelett in einem 3D-Modell erstellt wird, um die Bewegung und Animation von Charakteren oder Objekten zu ermöglichen.

Plattform *Mixamo* verwendet. Nach dem Anpassen der Keyframes[4] der Animationen in *Blender* wurden diese in *Unity* importiert und verwendet.

3.6 Eigenständig modellierte Stage

Die Stage wurde mithilfe der 3D-Anwendung *Blender* erstellt. Dafür wurden diverse *Blender*-Tools aus *Object*, *Edit* und *Sculpting Mode* benutzt, um die Stage in Form und Farbe zu bringen. Für die richtigen Maße sorgte eine einfache 2D-Vorlage mit Messwerten aus dem Internet. Das Design soll an einen mit Zuckerguss bedeckten Donut erinnern.

4 Retrospektive und Ausblick

Unser größter Anreiz in diesem Projekt war der Lernerfolg. Dazu haben wir uns als Ziel gesetzt, die Grundkonzepte eines *Plattform-Fighter*-Spiels kennenzulernen und eine einfache Nachstellung zu programmieren. Wir wollten erfahren, wie Entwicklungstools in *Unity* erstellt werden und welche Vorteile dies mit sich bringt. Wir wollten mit Blender erlernen, wie schwer das Modellieren und Animieren von Charakteren ist. Und weiterhin wollten wir anhand der Motion-Capture-Technologie die Kämpfe im Spiel realistischer aussehen lassen und herausfinden, ob es sich gegenüber synthetischen Animationen in Bezug auf Aufwand mehr lohnt.

All diese Ziele konnten wir größtenteils bewerkstelligen. Dabei war das Zeitintensivste das Experimentieren und Ausprobieren in den verschiedenen Bereichen, um die möglichst besten Lösungen zu finden. Aus diesem Grund könnten theoretisch noch unzählige Optimierungen und Erweiterungen vorgenommen werden und wir deswegen das Spiel aufgrund der Zeit und des Aufwands in einem recht simplen Zustand lassen und vermutlich nicht weiter entwickeln werden.

Bei der Entwicklung des Spiels gab es auch einige Konzepte und Dinge, welche aus verschiedenen Gründen, aber vor allem Zeit, nicht umgesetzt oder durch andere Konzepte ersetzt wurden. Eins davon ist die Physik-Logik, welche das Abprallen der Fighter an Oberflächen erlaubt, sofern sie dagegen geschleudert werden. Ein weiteres Konzept ist die sogenannte zweidimensionale *Environmental Collision Box* (ECB),

[4] Ein Keyframe ist ein entscheidender Zeitpunkt in einer Animation, der die Anfangs- oder Endposition eines Objekts definiert. Zwischen diesen Keyframes wird die Animation interpoliert, um fließende Bewegungen zu erzeugen.

welche für die Kollision mit der Umwelt gebraucht wird. Stattdessen wurde ein einfacher *BoxCollider* aus der *Unity*-Bibliothek genutzt. Ursprünglich war es geplant jeden einzelnen ACMD in eine eigene Datei zu extrahieren, stattdessen wurde entschieden, diese in Movesets zusammenzufassen, um einen besseren Überblick über diesen zu haben. Weiterhin mussten viele Animationen aus der Motion-Capture-Session verworfen werden, da wir in der Zeit nicht alle Moves umsetzen konnten. Im Spiel wurden nur 9 von 28 Motion-Capture-Animationen genutzt.

Trotz der genannten Dinge sind wir mit unserem Endprodukt und vor allem mit dem Lernerfolg sehr zufrieden und sehen unsere Arbeit als Erfolg an.