

Studienprojekt
**Entwicklung eines Farming Simulators auf
einer zeitgemäßen Game Engine**

im Studiengang Softwaretechnik und Medieninformatik
der Fakultät Informationstechnik
Wintersemester 2023/24

Pavithra Sureshkumar

Zeitraum: 14.11.2023 - 15.02.2024
Prüfer: Prof. Dr.-Ing. Harald Melcher
Zweitprüfer: Prof. Dr.-Ing. Andreas Rößler

Inhaltsverzeichnis

1	Einleitung	1
1.1	Kurze-Zusammenfassung	1
1.2	Motivation/Einblick	1
2	Grundlagen der Godot Game Engine	2
2.1	Einführung in Godot	2
2.1.1	Vorteile und Besonderheiten	2
2.1.2	Erste Schritte	3
2.2	Die Oberfläche von Godot	5
2.2.1	Der Projekt Manager	5
2.2.2	Die Panelübersicht	6
2.3	Szenen und Nodes	8
2.3.1	Szenen	8
2.3.2	Nodes	8
2.3.3	Hierarchie von Nodes	9
2.4	GDScript	11
2.4.1	Was ist GDScript?	11
3	Realisierung des Farming Simulators	13
3.1	Konzeptualisierung des Farming Simulators	13
3.1.1	Spielidee und Motivation	14
3.1.2	Gestaltung des Hauptcharakters	16
3.1.3	Gestaltung des Inventarsystems	18
3.1.4	Aufbau der Datenstruktur des Inventarsystems	19
3.2	Praktische Umsetzung des Farming Simulators	21
3.2.1	Nomad Sculpt	21
3.2.2	Realisierung der Modelle mit Nomad Sculpt	23
3.2.3	Mixamo von Adobe und Blender	30
3.2.4	Blender und Godot	34
3.2.5	Umsetzung der Charakteranimationen: "Idle" und "Walk" in Godot	35
3.2.6	Umsetzung des Inventarsystems	42
4	Schluss	43
	Literatur	44

Abbildungsverzeichnis

2.1	Godot Logo	2
2.2	Godotengine macOS download Seite	3
2.3	Godot Projekt Manager	5
2.4	Godot Panelübersicht	6
2.5	Godot Scene	8
2.6	Godot Nodes Hierarchie	9
2.7	GDScript Code Beispiel Sprite2D	11
3.1	Mindmap: Gedanken zum Spiel	14
3.2	Animal Crossing Body	16
3.3	Charakter Skizze	17
3.4	Minecraft Inventarsystem	18
3.5	Einfaches Inventory System	18
3.6	Skizze von Datenstruktur des Inventarsystems	19
3.7	Nomad Logo	21
3.8	Interface Nomdad Sculpt	21
3.9	Erster Prototyp Charakter mit Nomad Sculpt	23
3.10	Zweiter Prototyp Charakter mit Nomad Sculpt	24
3.11	Charakter ohne richtigen Torso	25
3.12	Charakter mit richtigen Torso	26
3.13	Charakter ohne Paint	27
3.14	Charakter mit Paint	28
3.15	Charakter UV-Unwrap UV-Atlas	29
3.16	Mixamo Startpage	30
3.17	Hochladen des 3D Charakters	30
3.18	Animation Library von Mixamo	31
3.19	Mixamo exportieren der Animation	32
3.20	Mixamo und Blender Actions	33
3.21	Importieren des 3D-Modells und Animationsplayer in Godot	34
3.22	Godot Asset Library: Basic 3D Starter	35
3.23	Godot Project Settings	36
3.24	Godot Script	37
3.25	Godot AnimationTree	40
3.26	Godot AnimationTree Editor	40

Listings

3.1 Character movement	38
----------------------------------	----

Kapitel 1

Einleitung

Die vorliegende Arbeit befasst sich mit der Entwicklung eines 3D Farming Simulators. Das Spiel wird unter Verwendung einer zeitgemäßen Game Engine (Godot) erstellt und die 3D-Assets mit NomadSculpt, Blender und Mixamo erstellt.

1.1 Kurze-Zusammenfassung

Das Ziel dieses Projektes ist die Entwicklung eines Farming Games unter Verwendung der Game Engine Godot v4.2.1.stable.official.

Die vorliegende Arbeit dokumentiert den gesamten Prozess, unter anderem der Planung, Erstellung und Bewältigung von Herausforderungen im Zusammenhang mit dem Projekt. Dabei sind die wichtigen Aspekte im Fokus: die Planung, die Umsetzung mit der Game Engine Godot und 3D Blender sowie die Erstellung des Spiels.

1.2 Motivation/Einblick

Das Projekt wird durchgeführt, um die heute zeitgemäße Game Engine Godot kennenzulernen. Das Farming Game soll zum Entspannen dienen und einem das Gefühl geben, in einer anderen Welt einzutauchen.

Kapitel 2

Grundlagen der Godot Game Engine

Um ein besseres Verständnis über der Godot Game Engine zu erlangen, muss man ein paar Grundlagen wissen. Dieses Kapitel gibt eine kleine Einführung zu Godot und erklärt grob wie die Game Engine zu bedienen ist und was man alles mit ihr erreichen kann.

2.1 Einführung in Godot

Godot ist eine freie open-source Game Engine und wurde von Juan Lienentsky und Ariel Manzur in 2007 entwickelt. Im Jahr 2014 wurde die Game Engine unter der MIT-Lizenz veröffentlicht und kann von der öffentlichen Seite von Godot heruntergeladen werden ([1](#), S.1).



Abb. 2.1: Godot Logo ([2](#))

2.1.1 Vorteile und Besonderheiten

Die Godot Game Engine ist eine Open-Source-Plattform für die Entwicklung von Computerspielen. Sie bietet viele Vorteile, wie die Erstellung von 2D-Plattformspielen und 3D-Plattformspielen. Außerdem ist die Benutzeroberfläche der Game-Engine sehr benutzerfreundlich gestaltet.

Alles, was mit Godot erschaffen wird, gehört zu 100 Prozent den Entwicklern, im Gegensatz zu anderen kommerziellen Game-Engines. Bei anderen Game Engines ist in

der Regel eine vertragliche Zusammenarbeit erforderlich. Der Entwickler kann selbst entscheiden, wie und wo das Spiel vertrieben wird. Viele kommerzielle Game Engines haben strikte Voraussetzungen für das Veröffentlichen von aufwendigen Spielen und erfordern den Kauf einer Lizenz (3, S.4).

Godot ist auch sehr transparent, denn Entwickler können selbst entscheiden, ob sie eine bestimmte Eigenschaft der Engine modifizieren oder neue Eigenschaften implementieren möchten, ohne eine spezielle Genehmigung einholen zu müssen. Diese Eigenschaft ist besonders hilfreich bei der Entwicklung größerer Projekte, da man vollen Zugriff auf die internen Funktionen der Engine hat (3, S.5).

Für viele Entwickler ist Godot daher die bessere Lösung für die Spieleentwicklung. Godot wird nicht von einem Unternehmen entwickelt, sondern ist den Entwicklern gewidmet, die ihre Zeit und Erfahrung investieren, um die Engine zu erstellen, zu testen und Fehler zu beheben. Außerdem führen sie ausführliche Dokumentationen über die Game Engine. Viele Entwickler erstellen auch einige Prototypen von Starter-Gameszenen, um mit Godot den ersten Schritt zu machen (3, S.5).

2.1.2 Erste Schritte

Um mit Godot durchzustarten, muss man die neuste Version von der Godot Seite herunterladen <https://godotengine.org/download/macOS/>. Die jetzige Version sollte im Bereich von Godot 4.x.x sein. Auf der Seite wird auch .NET angeboten, diese wird aber nur gebraucht, wenn man mit C# arbeiten möchte. Alternativ kann man Godot auch per Steam, itch.io oder mit dem Paketmanager (Homebrew, Scoop, Snap) der jeweiligen OS installieren (3, S.6).



Abb. 2.2: Godotengine macOS download Seite (2)

Nach dem klassischen download wird dann ein zip Ordner bereitgestellt, diese wird dann entpackt. Alternativ kann man auch kann man diese auch zu dem Programm

Ordner oder dem Applikationen Ordner rüberziehen. Nach dem öffnen der Applikation sieht man den Godot Projekt Manager Fenster (3, S.6).

Die folgenden Kapitel handeln von der Godot Engine, wie sie zu bedienen ist, die einzelnen Kernmerkmale der Godot Engine und die verschiedenen Elemente, um einen groben Überblick zu geben. was die Game Engine zu bieten hat. Die Kapiteln dienen als Basis um mit dem Projekt Farming Simulator durchzustarten zu können.

2.2 Die Oberfläche von Godot

Dieses Kapitel beschreibt den Aufbau der Oberfläche von Godot.

2.2.1 Der Projekt Manager

Nach dem Öffnen von Godot erscheint zunächst der Projekt Manager. Hier kann man entscheiden, ob man ein neues Projekt erstellen möchte oder in der öffentlichen Asset-Bibliothek nach Projekten suchen möchte (3, S.7-8).

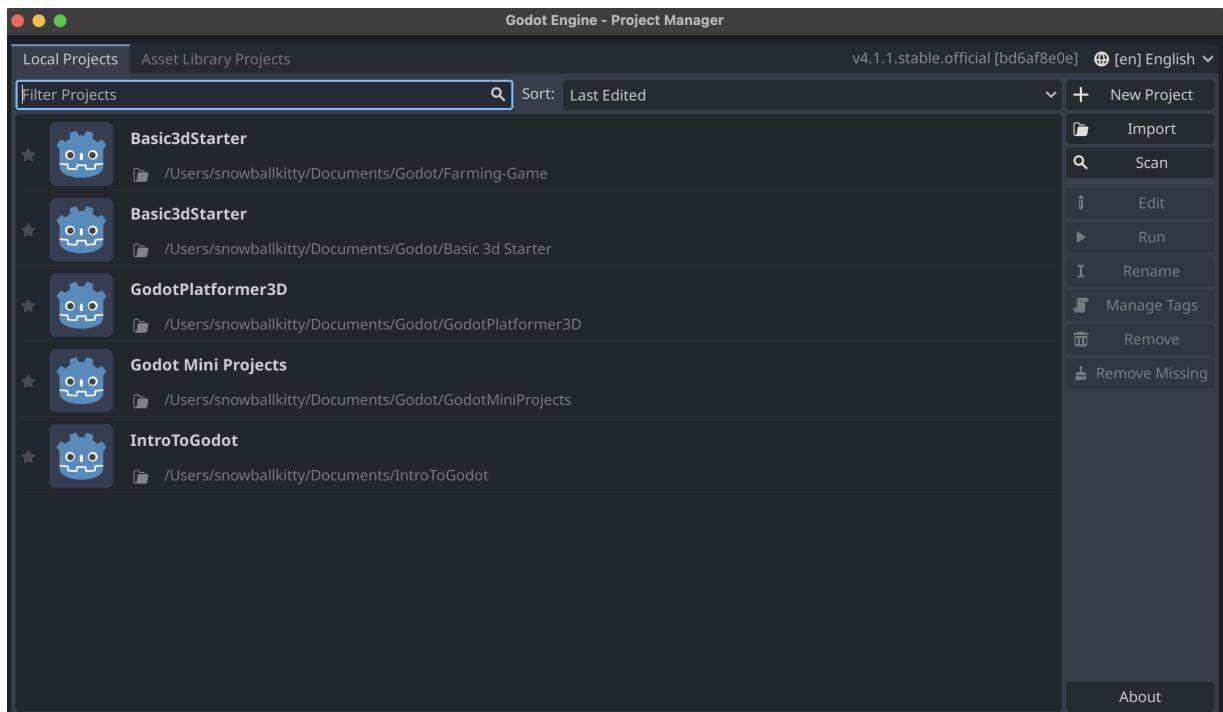


Abb. 2.3: Godot Projekt Manager

2.2.2 Die Panelübersicht

Die Godot-Oberfläche besteht aus mehreren Panels. Die einzelnen Abschnitte wie das Szenenbaum, der Fielssystem-Abschnitt, die Arbeitsfläche und der Inspector haben ihre eigene Funktion (3, S.9-11).

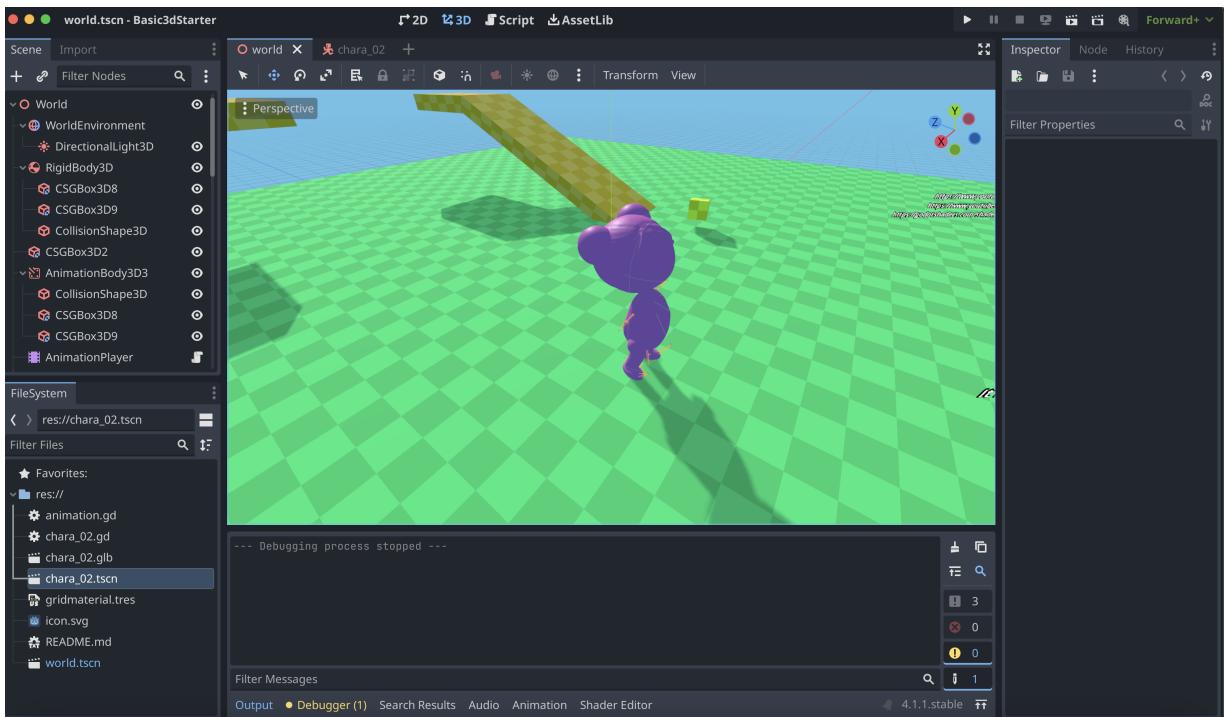


Abb. 2.4: Godot Panelübersicht

In Abbildung 2.4 ist zu erkennen, dass der größte Teil des Editor-Panels das Viewport ist, welches sich in der Mitte befindet. Der Viewport dient der Übersicht über die einzelnen Elemente, die ein Entwickler bearbeitet (3, S.9).

Im oberen Abschnitt der Oberfläche befindet sich die Liste der Arbeitsplätze. Hier kann zwischen 2D-, 3D- und Script-Modus gewechselt werden. Im Script-Modus wird der Spielcode bearbeitet. Der Tab AssetLib ist für das Herunterladen von Add-ons und Beispielprojekten zuständig, die von der Godot-Community bereitgestellt werden (3, S.10).

Über dem Viewport ist die Toolbar sichtbar. Hier können verschiedene Tools verwendet werden, um mit dem Viewport und den einzelnen Elementen zu interagieren (3, S.10).

In der oberen rechten Ecke befindet sich der Player, um das Spiel zu testen. Beim Testen des Spiels erscheint ein neues Fenster, das das Spiel debuggt (3, S.10).

In der linken unteren Ecke des Panels findet man das Dateisystem. Hier werden alle Dateien des Projektordners angezeigt. Alle Ressourcen werden im relativen Pfad res:// lokalisiert, was dem Stammverzeichnis des Projekts entspricht (3, S.10).

In der oberen linken Hälfte ist der Szenenbaum zu sehen, in Abbildung 2.4 auch als Scene bezeichnet. Dieser zeigt die aktuelle Szene an, die im Viewport bearbeitet wird (3, S.11).

Auf der rechten Seite befindet sich der Inspector, in dem die Eigenschaften der Spielobjekte angepasst werden können (3, S.11).

Mit diesen Informationen ist man nun bereit, mit der Godot Engine zu arbeiten. Die Funktionsweise der einzelnen Elemente in Godot lässt sich jedoch nur durch das Bearbeiten von Projekten erfahren. Die folgenden Kapitel geben einen noch tieferen Einblick in die Elemente von Godot, um das Projekt "Farming Simulator" zu starten.

2.3 Szenen und Nodes

Im Kapitel 2.2.2 wurden die einzelnen Elemente der Godot-Oberfläche grob aufgezeigt. Eines der wichtigen Elemente ist der Szenenbaum, welches in diesem Kapitel näher behandelt wird.

2.3.1 Szenen

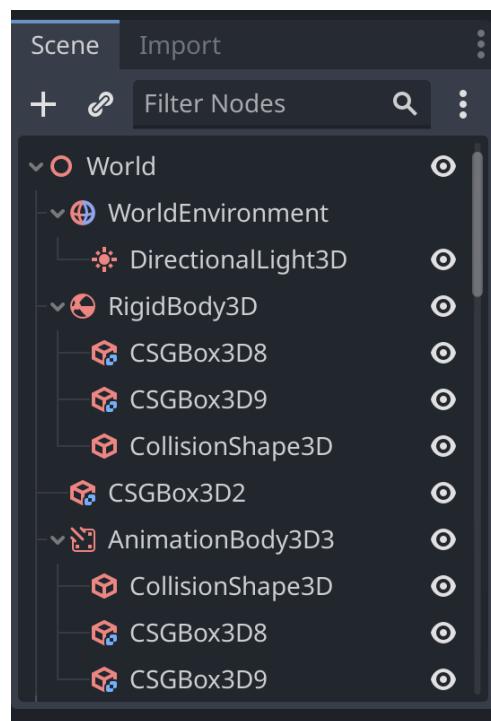


Abb. 2.5: Godot Scene

Eine Szene 2.5 wird in Godot verwendet, um die verschiedenen Spielobjekte im Projekt zu erstellen und zu ordnen. Zum Beispiel kann eine Szene nur für den Spieler erstellt werden, mit den dazugehörigen Nodes und Skripten, um den Spieler zu steuern. Eine andere Szene könnte für die Spielwelt erstellt werden, in der der Charakter mit den erstellten Objekten und Hindernissen in der Welt interagiert. Am Ende werden die Szenen kombiniert, um ein finales Spiel zu erstellen (3, S.12).

2.3.2 Nodes

Nodes sind die Grundbausteine von Godot, um Spiele zu erstellen. Eine Node ist ein Objekt, das verschiedene spezialisierte Spielfunktionen haben kann. Zum Beispiel kann eine Node ein Typ von einer Bildanzeige sein, um eine Animation abzuspielen, oder

ein 3D-Modell darstellen. Eine Node kann eine Sammlung von Eigenschaften besitzen, die dann verwendet werden können, um das Verhalten der Node anzupassen. Welche Nodes ein Entwickler verwendet, ist ihm überlassen, je nach der gewünschten Funktionalität. Es handelt sich um ein modulares System, das dem Entwickler genügend Freiraum gibt und flexibel ist, um die Spielobjekte zu erstellen (3, S.11).

Nodes bringen hauptsächlich ihre eigenen Eigenschaften und Funktionen mit. In Godot kann der Entwickler jedoch das Verhalten oder die Funktionalität der Nodes erweitern, indem er Skripte zu den jeweiligen Nodes hinzufügt. Diese Möglichkeit ermöglicht es dem Entwickler, mehr aus dem Standard-Node herauszuholen. Zum Beispiel kann man zum Sprite2D-Node eine Eigenschaft hinzufügen, die ein Bild anzeigen soll. Möchte man jedoch, dass das Bild beim Klicken verschwindet und beim erneuten Klicken wieder erscheint, muss man ein Skript einfügen, um dieses Verhalten zu erzeugen (3, S.12).

2.3.3 Hierarchie von Nodes

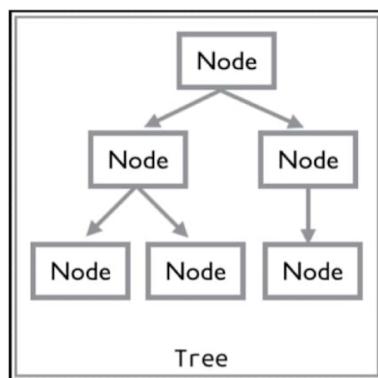


Figure 1.10: Nodes arranged in a tree

Abb. 2.6: Godot Nodes Hierarchie
(3, S.11)

In Abbildung 2.6 werden die Nodes als eine Baumstruktur dargestellt. In einem Baum werden die Nodes als Kinder von anderen Nodes hinzugefügt. Eine bestimmte Node kann viele Kinder haben, aber nur eine Eltern-Node. Wenn eine Gruppe von Nodes zu einem Baum zusammengefügt wird, nennt man dies eine Szene (3, S.11). Wird eine Szene abgespeichert, kann diese als eine neue Node zu einem anderen Node als Kind hinzugefügt werden (4).

Nodes sind ein mächtiges Werkzeug. Das Verständnis dieses Tools ist eine Voraussetzung, um die Objekte in Godot zu begreifen. Doch ohne eine richtige Spiellogik sind Nodes nicht viel wert. Eine Spiellogik legt Regeln fest, die die Objekte befolgen sollen

(3, S.12). Im nächsten Kapitel wird die Skriptsprache GDScript behandelt, die eine der Sprachen ist, die man bei Godot verwenden kann, um eine Spiellogik aufzubauen.

2.4 GDScript

Dieses Kapitel behandelt die verfügbaren Programmiersprachen in Godot, wobei hauptsächlich auf die Skriptsprache GDScript eingegangen wird.

In Godot kann man zwischen zwei Programmiersprachen wählen, um die Nodes zu scripten: GDScript und C#. GDScript ist die dedizierte, integrierte Sprache mit der engsten Integration zur Engine und ist die unkomplizierte Sprache, um Godot zu nutzen. Alternativ kann man auch mit C# programmieren, indem man die entsprechende Version von Godot herunterlädt, die diese Sprache unterstützt. Godot selbst ist in C++ geschrieben, und man kann durch die Verwendung von C++ mehr Leistung und Kontrolle über die Funktionen der Engine direkt erhalten (3, S.12).

Die beste Wahl zum Scripten mit Godot ist GDScript, da es eng mit der Godot - Programmierschnittstelle integriert ist und speziell für schnelle Entwicklung entwickelt wurde (3, S.5).

2.4.1 Was ist GDScript?

Die Syntax der Skriptsprache GDScript ist stark an die Python-Sprache angelehnt. Personen, die bereits Erfahrung mit anderen dynamischen Programmiersprachen wie JavaScript haben, sollten es relativ einfach finden, GDScript zu erlernen. Ähnlich wie Python ist GDScript eine benutzerfreundliche Programmiersprache, besonders geeignet für Anfänger.

```
extends Sprite2D
var speed = 200

func _ready():
    position = Vector2(100, 100)

func _process(delta):
    position.x += speed * delta
```

Abb. 2.7: GDScript Code Beispiel Sprite2D (3, S.13)

GDScript ist eine dynamisch typisierte Skriptsprache, das bedeutet, dass man keine Variablenarten deklarieren muss, wenn man eine Variable erstellt. Stattdessen verwendet GDScript Leerzeichen (Einrückungen), um Codeblöcke zu kennzeichnen. Der größte Vorteil von GDScript liegt in der engen Integration mit der Game Engine, was zu einer schnellen Entwicklung führt und somit wenig Code erforderlich macht.

Die Abbildung 2.7 zeigt grob, wie GDScript aussieht. Der Code repräsentiert ein Sprite2D, das von links nach rechts über den Bildschirm bewegt wird.

Falls weitere Fragen zur GDScript-Sprache auftreten, sollten entsprechende Informationen in der Godot - Dokumentation zu finden sein: <https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/index.html>.

Im weiteren Verlauf des Projekts werden weitere Code-Snippets in GDScript eingeführt.

Kapitel 3

Realisierung des Farming Simulators

Dieses Kapitel befasst sich mit der Umsetzung des Farming Simulators mithilfe der Godot Engine. Dabei werden nicht nur die technischen Aspekte erörtert, sondern auch die Gestaltung des Spiels. Es werden Fragen behandelt, wie beispielsweise: "Warum sollte überhaupt ein Farming Simulator-Spiel entwickelt werden?"

3.1 Konzeptualisierung des Farming Simulators

In diesem Unterkapitel werden hauptsächlich die Spielkonzepte diskutiert. Die technischen Aspekte und die praktische Umsetzung werden im nächsten Kapitel behandelt.

3.1.1 Spielidee und Motivation

Zunächst werden die Gedanken zum Spiel sortiert, dies wurde mithilfe einer Mind Map realisiert 3.1.

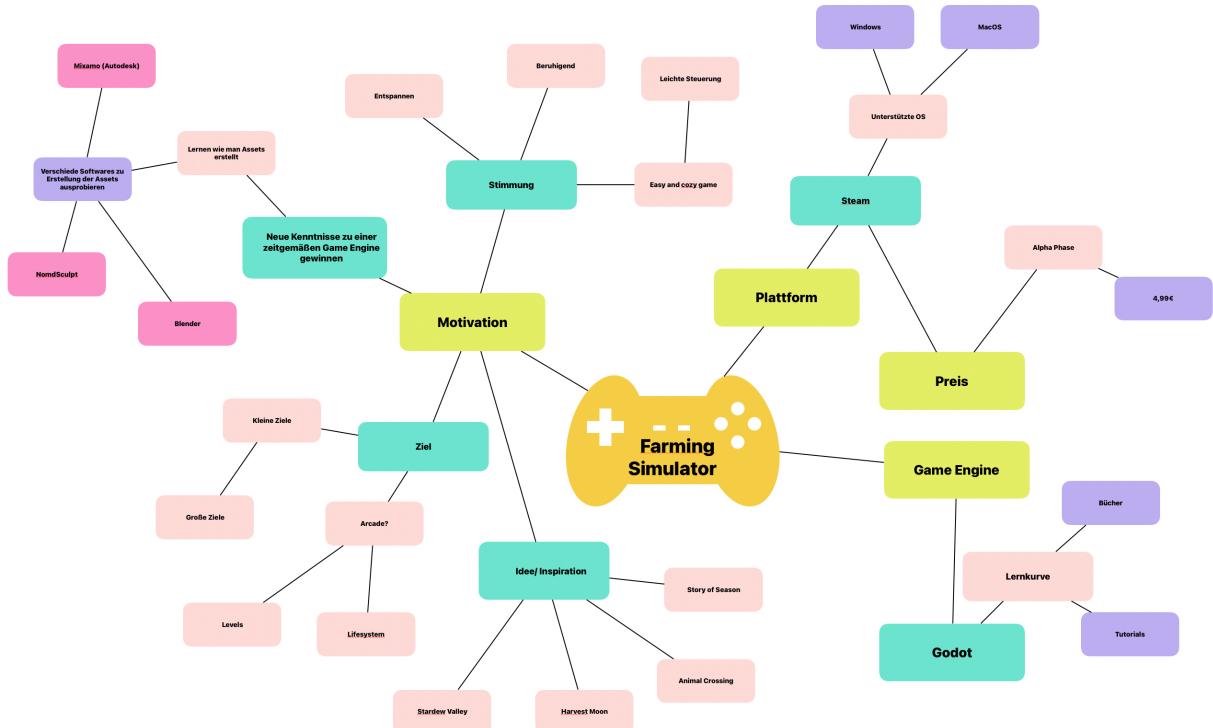


Abb. 3.1: Mindmap: Gedanken zum Spiel

Als erstes stellt sich die Frage: Warum überhaupt ein Farming Simulator Spiel?

Die Motivation hinter diesem Spielkonzept besteht darin, eine bestimmte Stimmung zu erzeugen. Die Idee wurde von bekannten Farming-Simulatoren wie Story of Seasons, Animal Crossing, Stardew Valley und Harvest Moon inspiriert. Bei all diesen Spielen liegt der Fokus auf "Easy Gaming" und "Cozy Gaming". Das Ziel ist es, dem Spieler eine beruhigende Atmosphäre zu vermitteln und ihn auf eine besondere Weise zu entspannen.

Natürlich gibt es auch Entwicklungsziele, die als Motivation dienen. Dazu gehört das Erlernen zeitgemäßer Game Engines sowie das eigenständige Erstellen von Assets unter Verwendung neuer Softwaretools wie Mixamo, NomadSculpt oder Blender. Das übergeordnete Ziel bei der Spielentwicklung ist es, kleine Schritte zu unternehmen und diese schließlich zu großen Fortschritten zu führen.

Im Verlauf des Entwicklungsprozesses stellte sich auch die spannende Frage: "Sollte man ein Arcade-Element integrieren?" Bei diesem Überlegungsschritt wurden Gedanken darüber angestellt, welches Ziel der Spieler im Spiel verfolgt. Ein Spiel besteht oft

aus kleinen Zielen, die der Spieler erreichen möchte. Dazu gehören nicht nur Level oder ein Lebenssystem, sondern auch der Zyklus bestimmter Aufgaben. Im Farming Simulator wird dies beispielsweise durch das Einpflanzen von Samen, das Ernten der Pflanzen und das erneute Erlangen von Samen dargestellt.

Die Auswahl der richtigen Game Engine ist ein entscheidender Aspekt für Entwickler, insbesondere für diejenigen, die wenig Erfahrung haben. Godot wird als ausgezeichnete Wahl für Anfänger empfohlen, insbesondere für die Entwicklung kleiner Indie-Spiele. Dennoch erfordert das Erlernen der verschiedenen Elemente der Game Engine erheblichen Aufwand, und die Lernkurve ist eher schleifend und nicht stetig steigend. Es ist ratsam, die Godot-Dokumentation zu studieren, Tutorials durchzugehen und auch Bücher zu lesen, um sich mit der Plattform vertraut zu machen.

Wenn es darum geht, das Spiel auf einer Plattform zu veröffentlichen, stellt sich die Frage: "Wo soll es hochgeladen werden?" Eine Möglichkeit besteht darin, es auf Plattformen wie Steam zu veröffentlichen. Auch die Auswahl des Betriebssystems, auf dem das Spiel laufen soll, ist entscheidend. Die Frage nach dem Verkaufspreis ist ebenfalls wichtig. In der aktuellen Alpha-Phase könnte eine angemessene Preisgestaltung bei etwa 4,99 Euro liegen.

All diese Überlegungen dienen dazu, einen groben Überblick über das Spiel zu erhalten und das angestrebte Ziel zu erreichen.

3.1.2 Gestaltung des Hauptcharakters

In diesem Kapitel wird die Gestaltung des Charakters behandelt.

Wie bereits im vorherigen Kapitel 3.1 besprochen, wurde das Farmingspiel von Animal Crossing inspiriert.



Abb. 3.2: Animal Crossing Body (5)

Dabei wurde die Figur 3.2 von Animal Crossing genauer betrachtet. Auf den ersten Blick fällt auf, dass die Figur sehr einfach und dennoch einheitlich gestaltet ist. Der Kopf ist eine Kugel, der Körper ein Zylinder, der oben leicht eingezogen ist, und die Hände sind einteilig, das heißt, es gibt keine Finger, sondern nur eine Kugel als Hand.

Nach der Analyse wurde eine grundlegende Skizze des Körpers mit Procreate angefertigt.

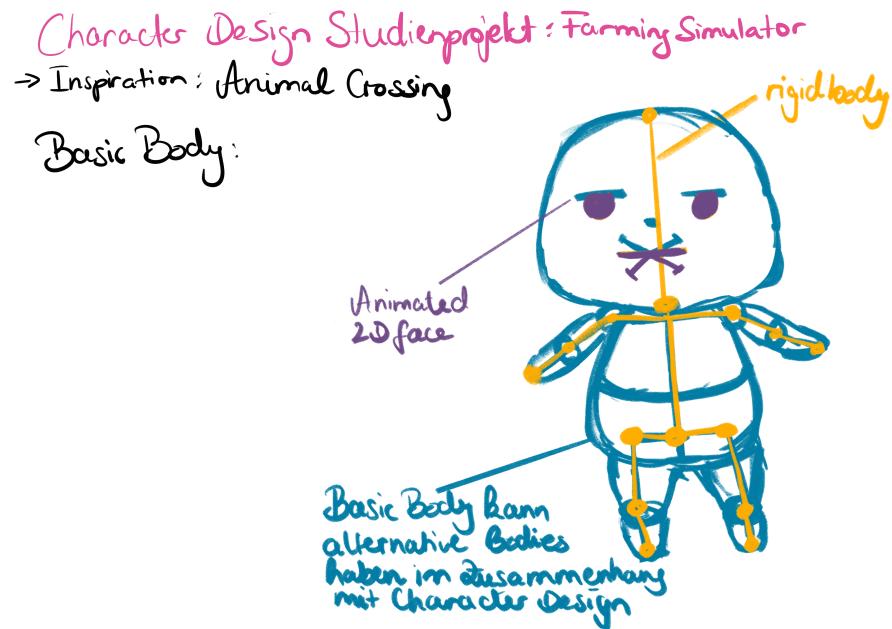


Abb. 3.3: Charakter Skizze

In Abbildung 3.3 wird der Charakter grob skizziert. Das grundlegende Körpermodell wird je nach Bedarf angepasst. Der Rigid Body wird später für Animationen verwendet. Das Gesicht kann entweder statisch oder in 2D animiert sein, abhängig von den Anforderungen.

3.1.3 Gestaltung des Inventarsystems

In diesem Kapitel wird das Konzept des Inventarsystems behandelt.



Abb. 3.4: Minecraft Inventarsystem (6)

In Abbildung 3.4 sieht man die Inspiration für das Inventarsystem, das im Farming Game verwendet wird. Es weist ein einfaches Design mit Slots und einem Ausrüstungsslot auf.

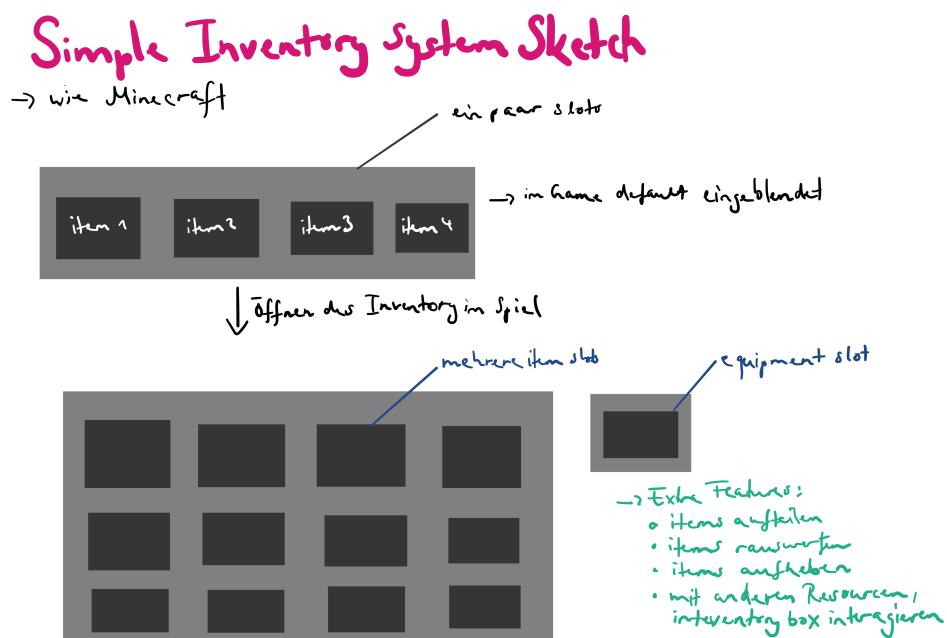


Abb. 3.5: Einfaches Inventory System

Wie in Abbildung 3.4 zu sehen ist, soll ein Inventarsystem geschaffen werden, das dem Spiel Minecraft ähnelt. Dazu wurde eine Skizze erstellt (3.5). Die Skizze zeigt, dass im Standardmodus das Inventar eingeklappt ist und nur die Gegenstände der ersten Reihe angezeigt werden. Beim Aufklappen des Inventars werden mehrere Slots sichtbar.

Die Anzahl der Slots pro Reihe kann nach den Wünschen des Entwicklers festgelegt werden. Ebenso ist ein Ausrüstungsslot vorhanden, der nur Werkzeuge des Charakters aufnimmt. Das bedeutet beispielsweise, dass Früchte, Gemüse oder Samen nicht in den Ausrüstungsslot gelegt werden können.

Die Funktionen des Inventarsystems umfassen:

- Aufteilen von Gegenständen durch Klicken
- Auswerfen von Gegenständen
- Einsammeln von Gegenständen
- Interaktion mit anderen Ressourcen wie Inventarboxen

3.1.4 Aufbau der Datenstruktur des Inventarsystems

Bevor man sich auf die Gestaltung der Szenen und die Erstellung von Skripten konzentriert, ist es wichtig, eine allgemeine Datenstruktur für das Konzept zu definieren. Wie sind die einzelnen Elemente zu realisieren?

Sketch Konzept Inventorysystem Datenstruktur



Abb. 3.6: Skizze von Datenstruktur des Inventarsystems

In der Abbildung 3.6 wird eine einfache Darstellung der Umsetzungsmöglichkeit der Datenstruktur eines Inventarsystem gezeigt. Ein Slot speichert eine Referenz von den Item-Daten sowie die Menge des jeweiligen Items. Die Items werden im Slot-Daten gespeichert. Das Inventar ist letztendlich eine Speicherung von Slot-Daten als Array.

Jede Szene die erstellt wurde, ist dann mit den entsprechenden Daten verbunden. Die Szene kommuniziert mit den jeweiligen Daten über Signale.

3.2 Praktische Umsetzung des Farming Simulators

In diesem Unterkapitel wird die praktische Umsetzung des Farming Simulators behandelt. Dabei werden die verwendeten Technologien näher betrachtet und erläutert.

3.2.1 Nomad Sculpt

Der Charakter wird in Nomad erstellt, einer 3D-Sculpting-App, die auf vielen Geräten läuft. Allerdings ist sie besonders für Tablets mit drucksensitivem Stift optimiert, wie beispielsweise den Apple Pencil oder den Samsung Galaxy Tablet-Stift (7).



Abb. 3.7: Nomad Logo (7)

Inspiriert wurde die App von anderen Desktop-Anwendungen wie ZBrush und Blender. Der Fokus von Nomad liegt auf der einfachen Bedienung der Benutzeroberfläche, ohne dabei wichtige Funktionen zu vernachlässigen (7).

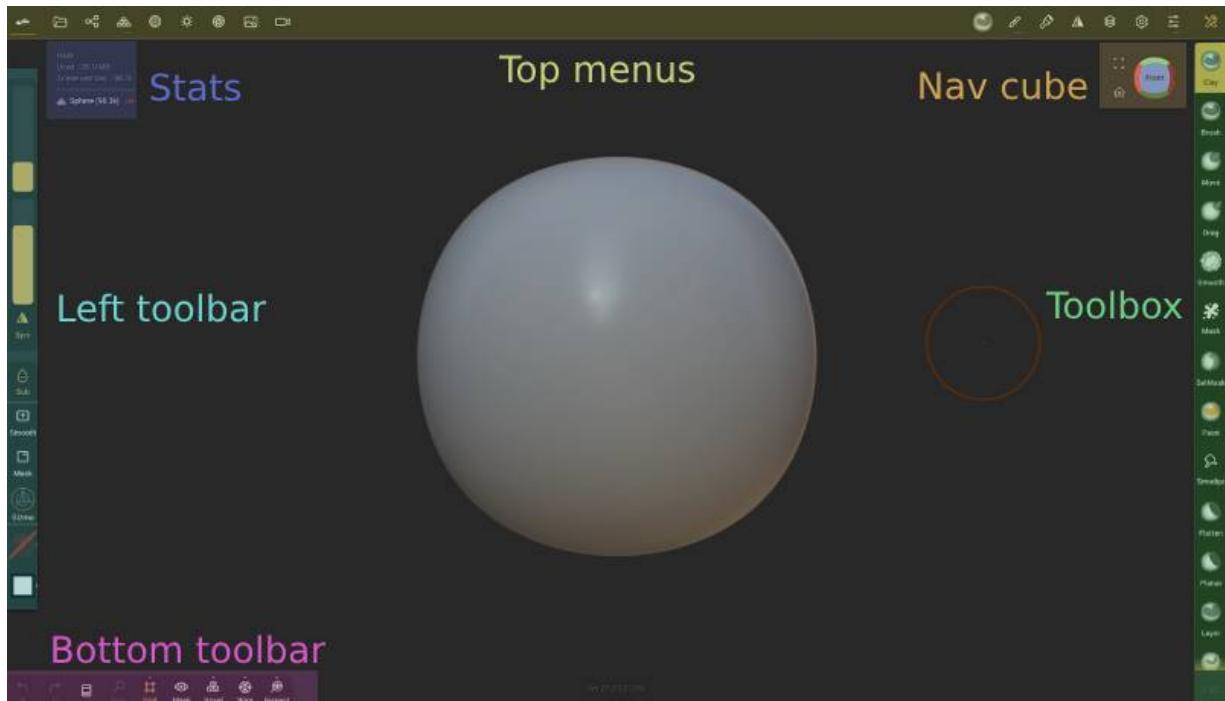


Abb. 3.8: Interface Nomad Sculpt (7)

In der Abbildung 3.8 ist das Interface von Nomad zu sehen, einschließlich der einzelnen Elemente zum Navigieren der verschiedenen Tools.

Im Top-Menü sind die meisten Nomad-Features zu finden. Auf dem oberen linken Menü befinden sich die Szene- und Objekt-Features, während auf der rechten Seite die Menüs für die Tools angeordnet sind. Auf kleineren Bildschirmen klappen diese Menüs zusammen, um Platz zu sparen. Das Stats-Fenster liefert Informationen zur Szene, dem ausgewählten Objekt, dem Maskenstatus und dem Speicherverbrauch. Der Nav Cube unterstützt bei der Orientierung und zeigt an, auf welcher Seite der Skulptur man sich befindet. Diesen kann man auch als Shortcut verwenden, um zu einer bestimmten Ansicht zu springen. Durch Ziehen des Cubes dreht sich die Ansicht entsprechend. Ein Klick auf das Rahmensymbol ermöglicht es, auf das ausgewählte Objekt zu fokussieren oder zur Standard-Heimansicht zurückzukehren. Die Toolbox ist scrollbar und gewährt Zugriff auf die einzelnen Tools. In der linken Toolbar findet man Slider für den Radius oder die Intensität der meisten Tools, kontextbasierte Knöpfe für andere Tools, Shortcuts für Symmetrie, den Alt/Sub-Modus, Maskierung, Glättung, das Gizmo und die Maloption (7).

Das Bottom-Toolbar besteht aus einem klassischen Undo-Button, der die vorgenommenen Operationen rückgängig macht, und einem Redo-Button, der die Operationen wiederherstellt. Der History-Button ermöglicht das Anzeigen der Historie. Der Solo Toggle-Button zeigt entweder das aktuelle Objekt oder alle Objekte an. Das Grid-Button zeigt die Gitter an, und durch langes Halten erscheinen mehrere Optionen für das Grid. Der Mask-Button verbirgt die maskierte Region des ausgewählten Objekts, was nützlich ist, um sich auf einen bestimmten Bereich zu konzentrieren. Der Voxel-Button dient als Shortcut für den Voxel Remesher. Durch langes Halten auf den Voxel-Button können Optionen für die Voxel Remesh-Qualität eingestellt werden. Der Wire-Button schaltet das Drahtgitter ein und aus. Das Inspect-Button zeigt zusätzliche Daten zum aktuellen Mesh an. Die Standardanzeige zeigt die UVs an, aber durch langes Drücken oder Wischen nach oben können die Eigenschaften inspiziert werden, sofern vorhanden, und ob diese im Hintergrund oder auf dem Mesh angezeigt werden (7).

Um mehr über Nomad zu erfahren, empfiehlt es sich, die offizielle Seite von Nomad zu besuchen: <https://nomadsculpt.com/manual/>.

3.2.2 Realisierung der Modelle mit Nomad Sculpt

Da es einige Vorarbeit erfordert, die Anwendung Nomad Sculpt zu verstehen, wurden verschiedene Prototypen erstellt.

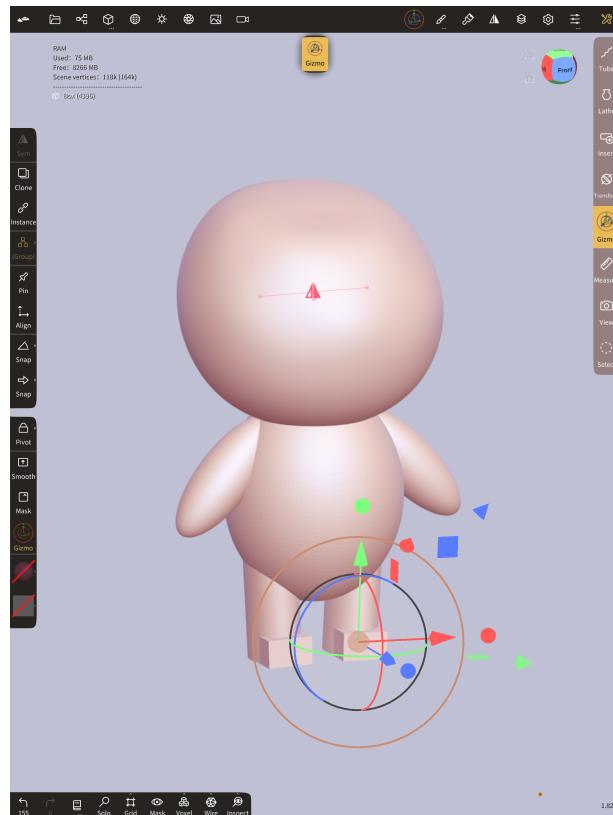


Abb. 3.9: Erster Prototyp Charakter mit Nomad Sculpt

Beim Prototypen in Abbildung 3.9 ist zu erkennen, dass der Charakter sehr einfach gehalten wurde. Obwohl hier mit Nomad Sculpt experimentiert wurde, wurde anschließend ein neuer Charakter erstellt, da dieser nicht den Anforderungen entsprach, die der Charakter erfüllen sollte.

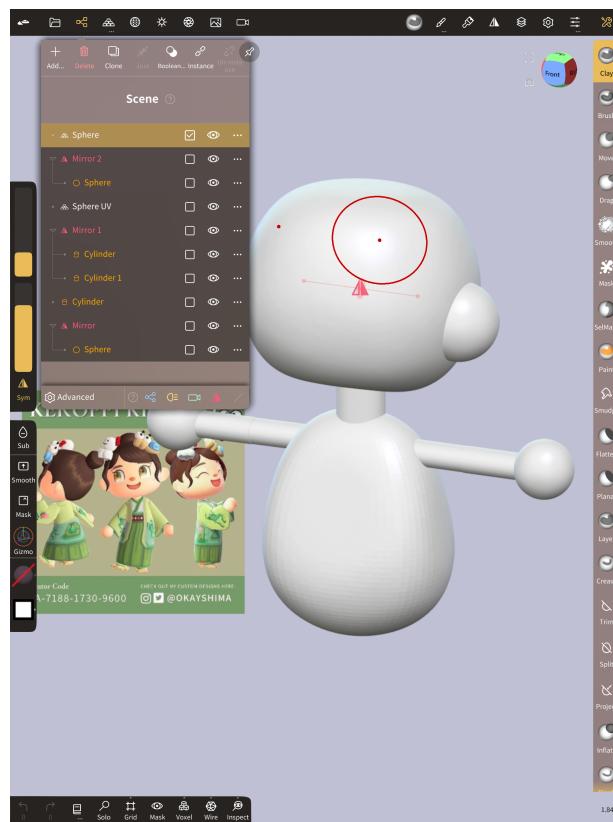


Abb. 3.10: Zweiter Prototyp Charakter mit Nomad Sculpt

In Abbildung 3.10 ist zu sehen, dass der zweite Prototyp des Charakters einen verbesserten Aufbau im Vergleich zum ersten Prototypen in Abbildung 3.9 aufweist. Als Referenz dient die Abbildung von Animal Crossing 3.2 aus dem Konzeptualisierungskapitel. Das Modell hat nun zwei Arme mit kugelförmigen Händen, Ohren und einen Hals.



Abb. 3.11: Charakter ohne richtigen Torso

In Abbildung 3.11 ist der Charakter in einem etwas fortgeschrittenen Zustand zu sehen. Der Charakter verfügt nun über Beine und Füße. Ein Problem, das sich bei diesem Charakter jedoch herausstellte, war der Torso. Dieser sollte eine Einkerbung haben, um den Ober- und Unterkörper zu definieren. Ohne einen gut strukturierten Torso traten Schwierigkeiten beim Einsatz der Mixamo-Software von Adobe auf.



Abb. 3.12: Charakter mit richtigen Torso

In der oberen Abbildung 3.12 ist der Charakter mit einem gut strukturierten Torso zu sehen. Beim Animieren des Charakters oder beim Einsatz des Auto-Rigging mit Mixamo treten nun weniger Probleme auf. Diese Erkenntnis war jedoch vorhersehbar, da die meisten Modelle in Mixamo einen menschlichen Torso mit einer klaren Abtrennung zwischen Oberkörper und Hüftbereich aufweisen.



Abb. 3.13: Charakter ohne Paint

In der Abbildung 3.13 ist der fast fertige Charakter mit einem Kimono zu sehen. Auf den Kopf wurden zwei Accessoires hinzugefügt, einmal eine Schleife und einmal eine Katzenmaske als Haarreif, sowie Sandalen. Hier ist auch zu erkennen, dass der Charakter noch nicht bemalt wurde.



Abb. 3.14: Charakter mit Paint

In Nomad ist es möglich, Objekte mit dem Paint Tool zu bemalen. In der Abbildung 3.14 sind nun Farben zu erkennen. Das Design wurde hier bewusst einfach gehalten, und das Gesicht wird ein statisches Lächeln darstellen. Dabei wurde darauf geachtet, dass der Charakter, ähnlich wie in Animal Crossing 3.2, verniedlicht wird.



Abb. 3.15: Charakter UV-Unwrap UV-Atlas

Um das Material zusammen mit dem 3D-Modell zu exportieren, muss das Modell mit dem UV-Atlas von Nomad entfaltet werden. In Abbildung 3.15 ist der UV-Unwrap Prozess hinter dem 3D-Charakter zu sehen. Nach diesem Vorgang ist es nun möglich, das Modell mit dem Material und den Texturen als glTF (Graphics Library Transmissions Format) zu exportieren. Das glTF-Format ermöglicht die Verwendung des Modells im Spiel und kann leicht in andere Spiele oder Anwendungen exportiert werden.

3.2.3 Mixamo von Adobe und Blender

Dieses Unterkapitel behandelt die Umsetzung der Animation des Charakters mithilfe von Mixamo und Blender.

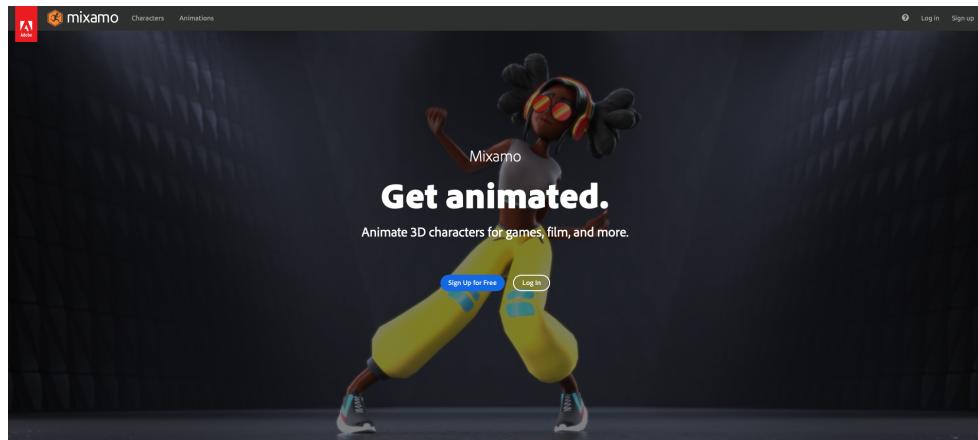


Abb. 3.16: Mixamo Startpage (8)

Mixamo ist ein Online-Service von Adobe, der es ermöglicht, 3D-Charaktere hochzuladen und zu animieren. Zudem bietet Mixamo auch eigene 3D-Charaktere an, die ebenfalls genutzt werden können.

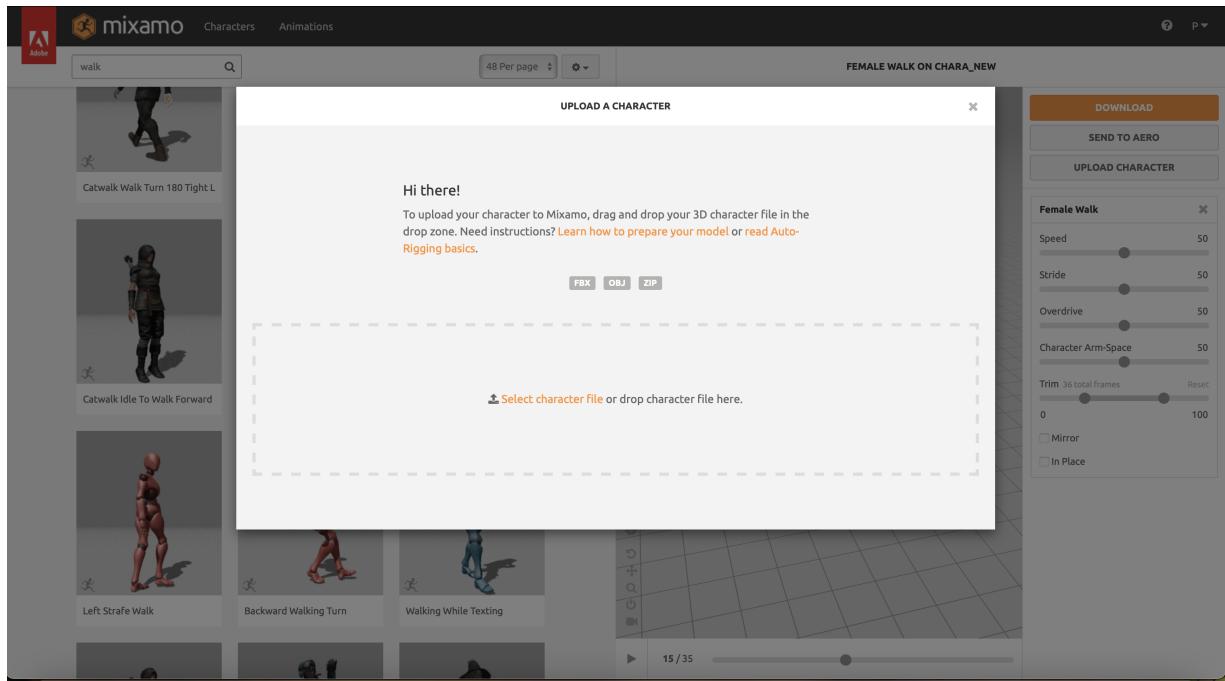


Abb. 3.17: Hochladen des 3D Charakters

Beim Hochladen kann man das Modell als FBX, OBJ oder ZIP-Datei hochladen. Der zuvor erstellte Charakter wurde als glTF von Nomad exportiert. Anschließend wurde

das Modell in Blender importiert und erneut als FBX in Blender exportiert. Nach dem Export wurde das 3D-Modell auf Mixamo hochgeladen. Durch das Auto-Rigging von Mixamo kann das Modell geriggt werden und anschließend mit verschiedenen Animationen versehen werden.

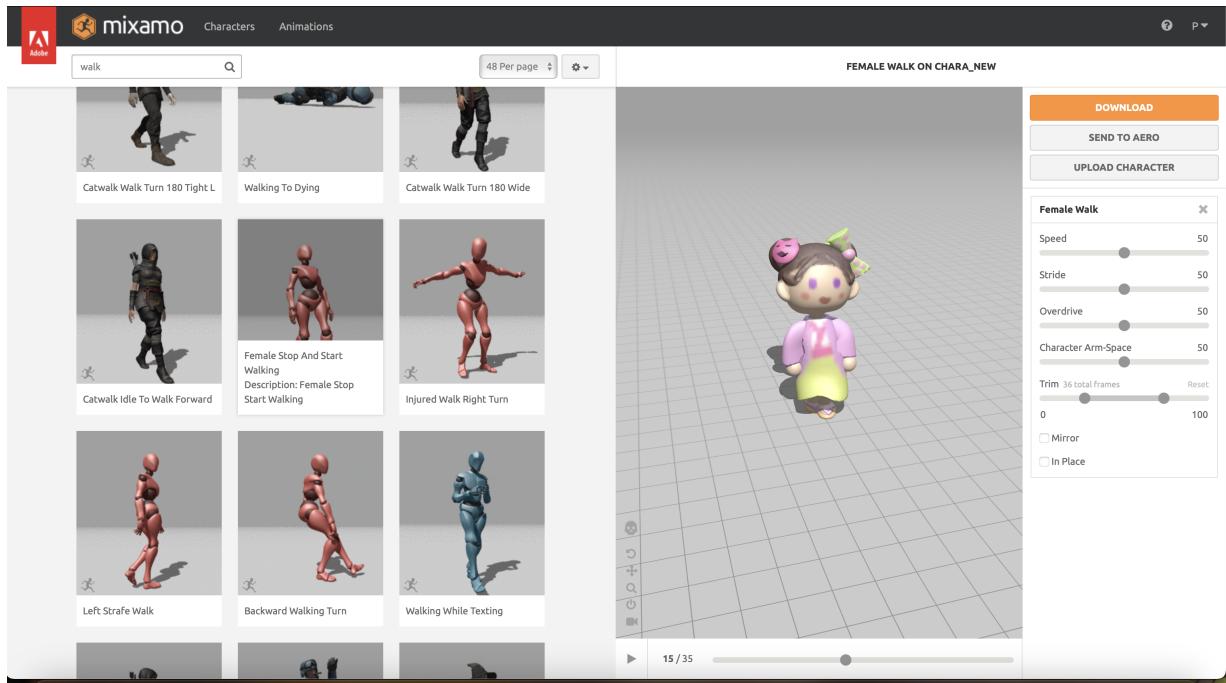


Abb. 3.18: Animation Library von Mixamo

In der Abbildung 3.18 sind verschiedene Animationen zu sehen, die dem Charakter zugeordnet werden können. Auf der rechten Seite sind zusätzliche Optionen sichtbar, wie beispielsweise die Geschwindigkeit der Laufbewegung. Es ist wichtig, dass die Laufanimation im Spiel an Ort und Stelle ausgeführt wird, da die Animation beim Bewegen des Charakters im Loop abgespielt wird. Andernfalls würde der Charakter immer wieder zur Ausgangsposition zurückkehren, und die Laufbewegung im Spiel würde dabei gestört werden.

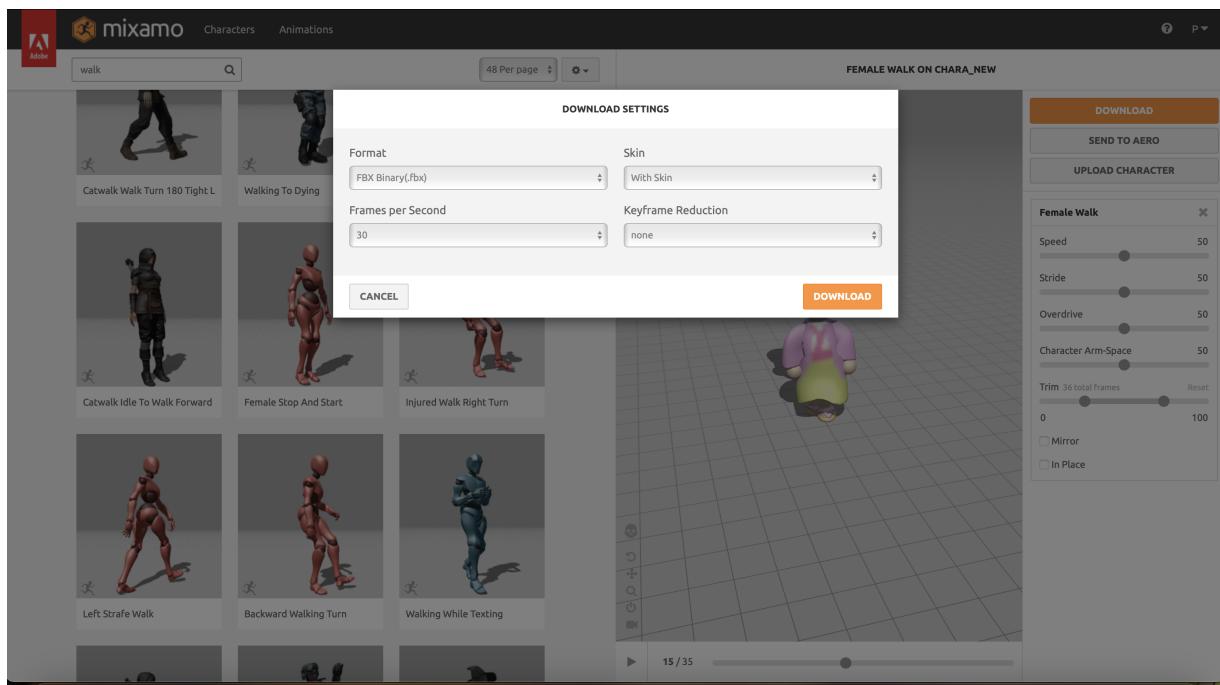


Abb. 3.19: Mixamo exportieren der Animation

In der Abbildung 3.19 ist der Exportvorgang zu sehen. Als Format wird erneut FBX verwendet, wobei auch die Hautinformationen (Skin) mitgenommen werden. Anschließend kann die Anzahl der Frames pro Sekunde festgelegt werden; hier wurden 30 FPS gewählt. Nach den Einstellungen kann das Modell mit der Animation heruntergeladen werden.

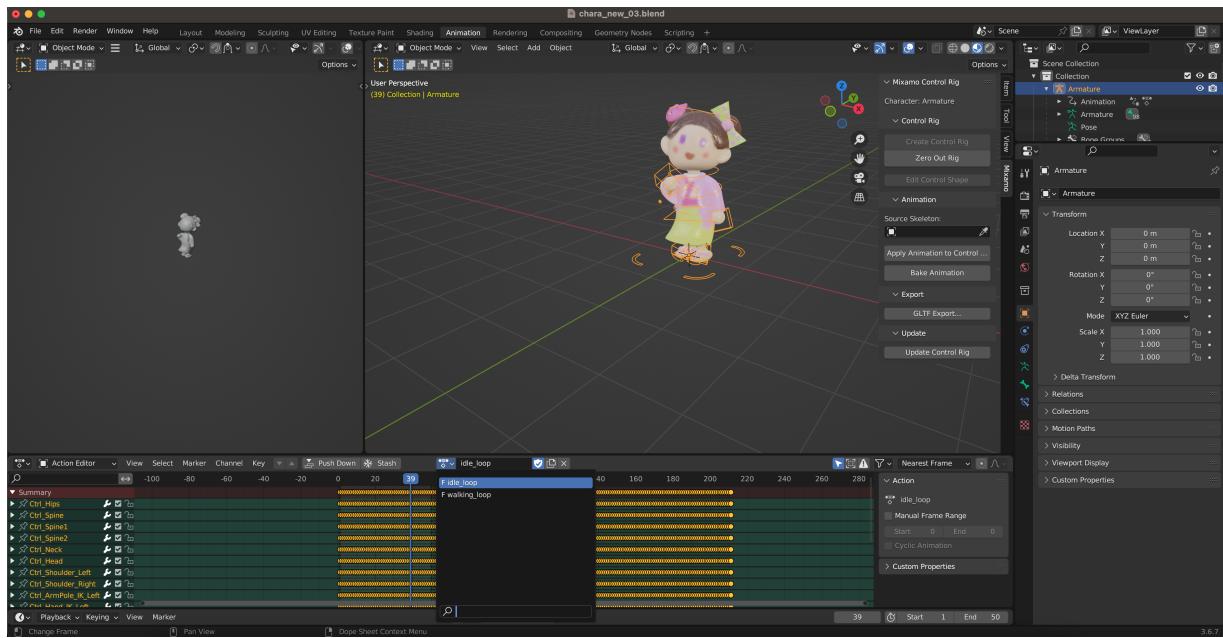


Abb. 3.20: Mixamo und Blender Actions

In Blender ist es dann möglich, die Animationen zu importieren und mithilfe des Mixamo auto control-rig-add-on das Modell zu riggen und die Animationen einzubinden. Zurzeit gibt es Probleme mit der neusten Blender Version und dem Mixamo Plugin, deswegen ist es ratsam die derzeitige LTS Version (v3.6.7) von Blender zu benutzen. Weitere Infos zu dem Auto Rigging Addon findet man auf der Seite <https://substance3d.adobe.com/plugins/mixamo-in-blender/>.

In der Abbildung 3.20 sieht man, dass in Blender der Figur die Animationen "Walking" und "Idle" zugewiesen wurden, indem die Animationen als Aktionen für ein Modell hinzugefügt wurden. Das bedeutet, dass ein 3D-Modell nun die beiden Aktionen "Walking" und "Idle" enthält. Außerdem ist die Bezeichnung des Actions sehr wichtig. Wenn die Actions nicht richtig mit "loop" benannt werden, könnte es Probleme mit der Animation in Godot geben. Denn mit der Bezeichnung "loop", erreicht man automatisch einen Schleifenvorgang beim importieren des Charakters in Godot. Die Animationen sollten alle bei Frame 0 beginnen, um später beim Importieren in Godot ein stotterfreies Abspielen zu gewährleisten.

3.2.4 Blender und Godot

Dieses Unterkapitel behandelt die Verwendung von Blender und Godot.

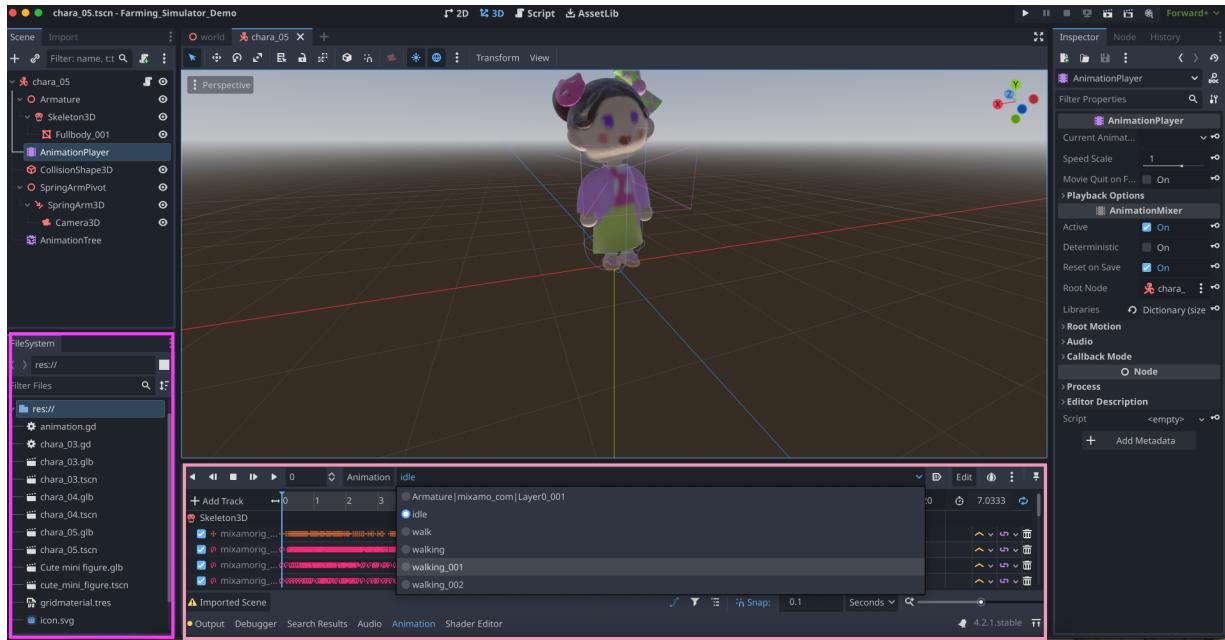


Abb. 3.21: Importieren des 3D-Modells und Animationsplayer in Godot

Um einen 3D-Charakter in Godot zu importieren, muss dieser zuerst im glTF-Format exportiert werden. Der Export erzeugt eine Datei mit der Endung ".gltf", die dann durch einfaches Drag and Drop in das Verzeichnis von Godot importiert werden kann. Nach dem Import erstellt man eine Szene für den 3D-Charakter. Nach dem Import überprüft man, ob die Animationen und Materialien korrekt angezeigt werden. Wie erwartet laufen die Animationen in einer "loop"-Schleife, da in Blender die jeweiligen Aktionen mit der Einstellung "loop" versehen wurden.

3.2.5 Umsetzung der Charakteranimationen: "Idle" und "Walk" in Godot

In diesem Unterkapitel wird die Umsetzung der Charakteranimationen "Walk" und "Idle" in Godot beschrieben.

Das Spiel wurde mit Godot v4.2.1.stable.official entwickelt und sollte dementsprechend auch mit dieser Godot-Version geöffnet werden. Das Öffnen mit einer anderen Version von Godot könnte dazu führen, dass die 3D-Objekte nicht korrekt dargestellt werden und beschädigt werden könnten.

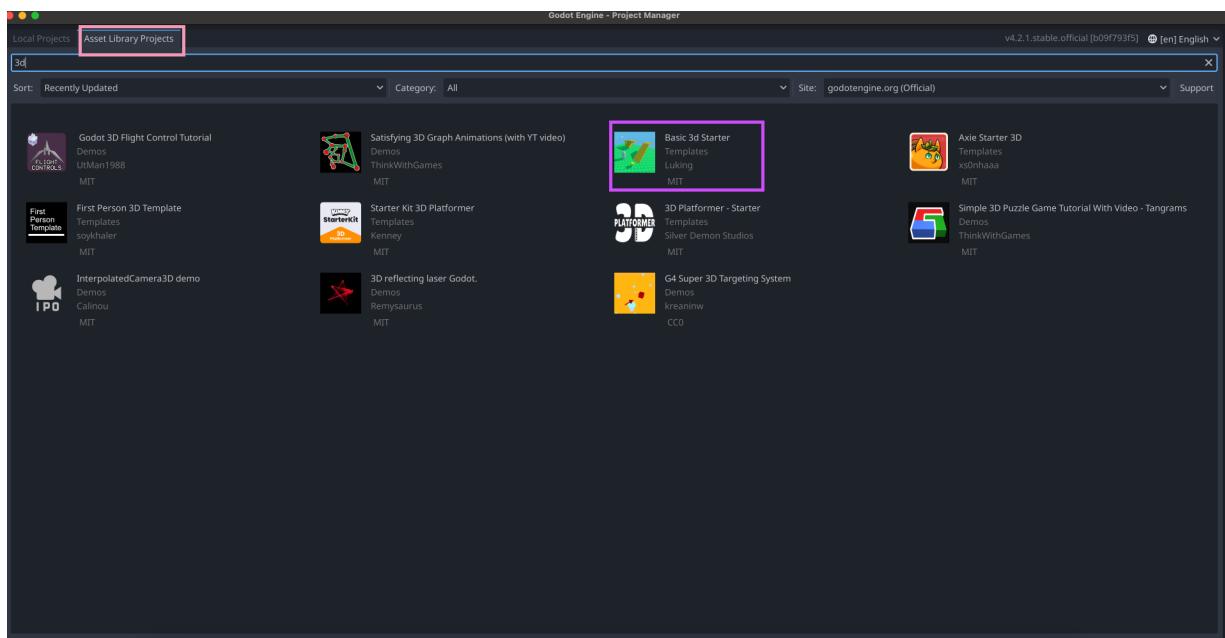


Abb. 3.22: Godot Asset Library: Basic 3D Starter

Bevor der Charakter ins Spiel importiert wurde, wurde als erster Schritt aus der Godot Asset Library der Basic 3D Starter heruntergeladen. Dieses Template diente als Grundlage für die Erstellung des Spiels. Es bietet Anfängern ein Beispieltemplate mit einer Kapsel als Charakter und einer 3D-Plattform. Darüber hinaus enthält das Template weitere Elemente wie animierte Plattformen, ein beeinflussbares Objekt (Würfel) und Textboxen.

In Abbildung 3.21 ist zu sehen, dass nach dem Import ein inherited scene für den Charakter erstellt wurde. In Godot werden erstellte Szenen als ".tscn" bezeichnet.

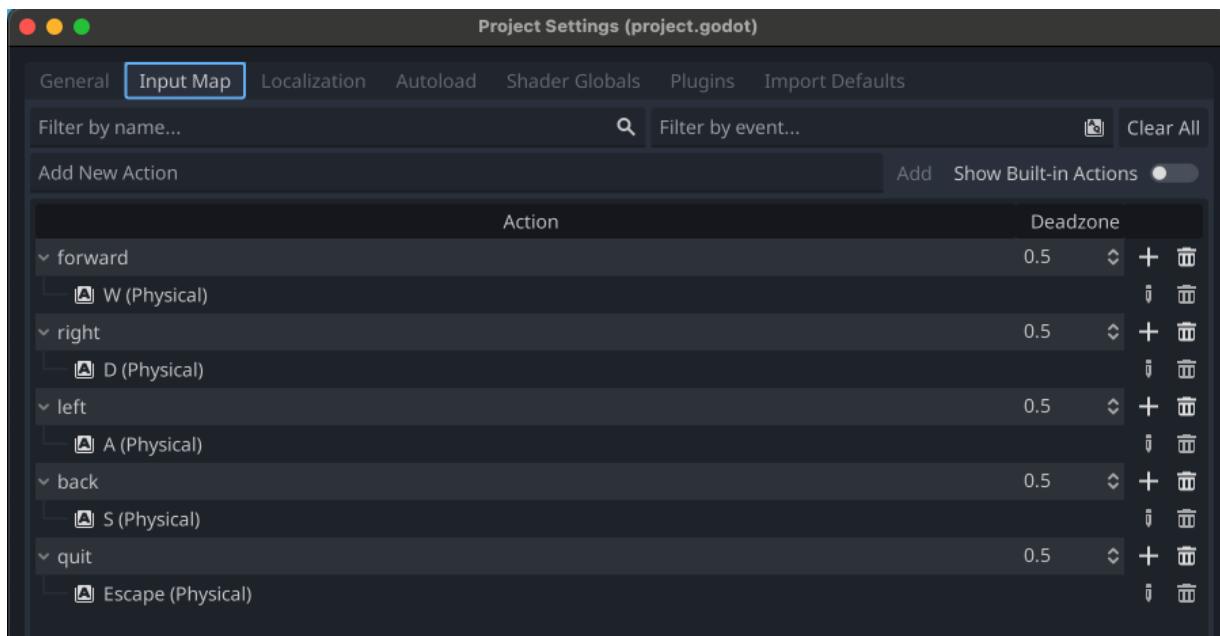


Abb. 3.23: Godot Project Settings

Bevor man mit dem Programmieren und dem Erstellen der anderen Nodes beginnt, legt man in den Project Settings die Input Map fest. Hier können Tasturbefehle auf Actions verteilt werden, wie beispielsweise Vorwärts (W), Rückwärts (S), Rechts (A) und Links (D). Der Tasturbefehl "quit" wird später für das Beenden des Fensters oder des Spiels verwendet.

In der Abbildung 3.21 ist auch das Root Node des Charakters zu sehen. Hierbei handelt es sich um einen CharacterBody3D. Man erkennt zudem, dass der Charakter von einem CollisionShape3D umgeben ist. Dieses dient dazu, mit anderen Objekten in Godot zu kollidieren. Für den Charakter wurde ein CapsuleShape3D ausgewählt und entsprechend skaliert.

Nach der Erstellung des CollisionShape3D wird die Kamera eingerichtet. Diese wird als Root ein Node3D namens SpringArmPivot haben. Diese wird verwendet, um die Kamera zu drehen. Dem SpringArmPivot wird dann ein SpringArm3D zugewiesen, und anschließend bekommt das SpringArm3D eine Camera3D zugewiesen. Das SpringArm3D dient dazu, die Länge des Abstands zwischen der Kamera und dem Charakter festzulegen. Außerdem wird der SpringArm3D mit der Kamera auf die Schulterhöhe des Charakters in der y-Achse verschoben.

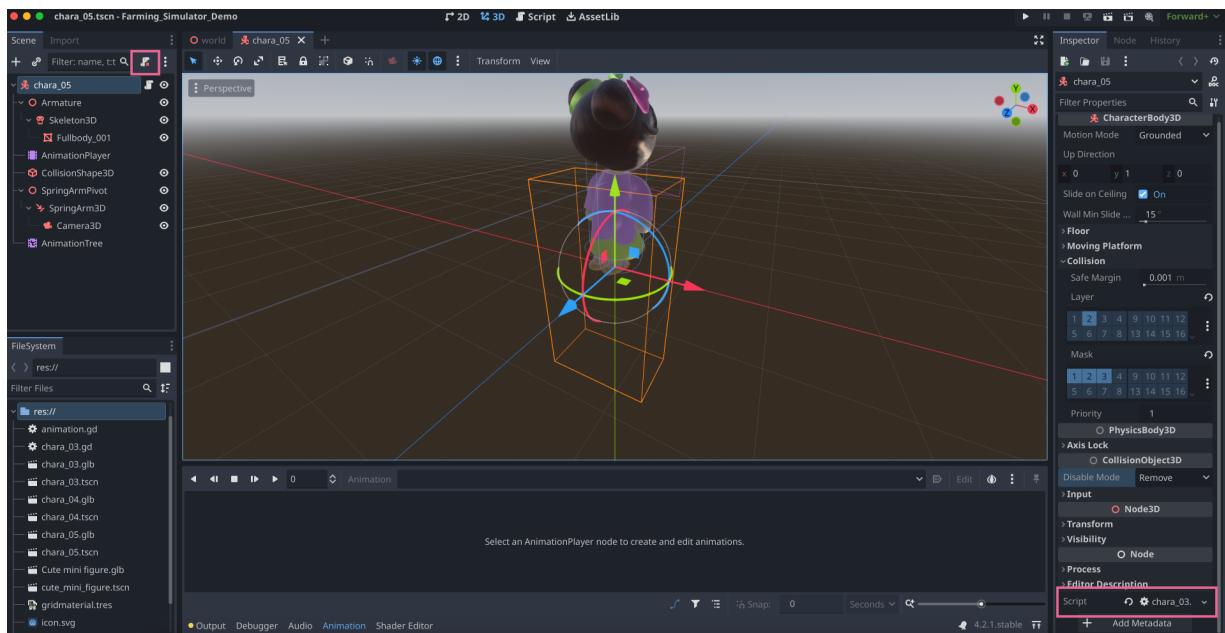


Abb. 3.24: Godot Script

Nun fügt man dem CharacterBody3D ein neues Skript hinzu. In der Abbildung 3.24 sind verschiedene Möglichkeiten dargestellt, wie ein Skript eingebunden werden kann. Man kann in Godot oben links bei der Szene ein Skript zum dazugehörigen Node erstellen oder rechts unten im Inspektor. Als Template nimmt man das, was bereits mit dem CharacterBody3D mitgeliefert wird.

Nachdem das Skript erstellt wurde, zieht man die Charakter-Szene in die World-Szene, um zu überprüfen, ob der Charakter und die Kamerabewegungen funktionieren.

List. 3.1: Character movement

```

1  extends CharacterBody3D
2
3
4  @onready var armature = $Armature
5  @onready var spring_arm_pivot = $SpringArmPivot
6  @onready var spring_arm = $SpringArmPivot/SpringArm3D
7  @onready var anim_tree = $AnimationTree
8
9  const SPEED = 5.0
10 const LERP_VAL = .15
11
12 # Get the gravity from the project settings to be synced with
   RigidBody nodes.
13 var gravity =
   ProjectSettings.get_setting("physics/3d/default_gravity")
14
15 func _ready():
   Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED)
16
17
18 func _unhandled_input(event):
19   if Input.is_action_just_pressed("quit"):
20     get_tree().quit()
21
22   if event is InputEventMouseMotion:
23     spring_arm_pivot.rotate_y(-event.relative.x * .005)
24     spring_arm.rotate_x(-event.relative.y * .005)
25     spring_arm.rotation.x = clamp(spring_arm.rotation.x, -PI/4, PI/4)
26
27 func _physics_process(delta):
28   # Add the gravity.
29   if not is_on_floor():
30     velocity.y -= gravity * delta
31
32   # Get the input direction and handle the movement/deceleration.
33   # As good practice, you should replace UI actions with custom
   gameplay actions.
34   var input_dir = Input.get_vector("left", "right", "forward",
   "back")
35   var direction = (transform.basis * Vector3(input_dir.x, 0,
   input_dir.y)).normalized()
36   direction = direction.rotated(Vector3.UP,
   spring_arm_pivot.rotation.y)
37   if direction:
38     velocity.x = lerp(velocity.x, direction.x * SPEED, LERP_VAL)
39     velocity.z = lerp(velocity.z, direction.z * SPEED, LERP_VAL)

```

```
40     armature.rotation.y = lerp_angle(armature.rotation.y,
41             atan2(-velocity.x, -velocity.z), LERP_VAL)
42     else:
43         velocity.x = lerp(velocity.x, 0.0, LERP_VAL)
44         velocity.z = lerp(velocity.z, 0.0, LERP_VAL)
45
46     anim_tree.set("parameters/BlendSpace1Dblend_position",
47             Vector2(velocity.x, velocity.z).length() / SPEED)
48
49     move_and_slide()
```

In dem obigen Codeausschnitt sieht man, wie sich der Charakter bewegt. Hier kommen wieder die Input Maps zum Einsatz, die zu Beginn festgelegt wurden. Zu Beginn bewegt sich der Charakter ohne eine Animation durch die Welt. Als nächstes werden die Kameraeinstellungen vorgenommen. Gewünscht ist, dass die Kamera sich mit der Mausbewegung unabhängig vom Charakter bewegt. Deshalb wurden zu Beginn der SpringArmPivot, der SpringArm3D und die Camera3D zur Szene hinzugefügt. Mit dem InputEventMouseMotion-Event werden die Rotationen des Arms verändert. Um eine endlose Rotation zu verhindern, wird die clamp() Methode verwendet. Nun muss die Rotation der Kamera an die des Charakters angepasst werden, was bei der direction geschieht.

Als nächstes versuchen wir, die Animation einzubauen. Dazu muss zunächst programmiert werden, dass der Charakter sich in die Richtung dreht, in die er sich auch bewegt. Dies wird durch das Rotieren des Armature-Knotens erreicht. Für viele Aktionen wird eine Interpolation verwendet, für die ein LERP-Value erstellt wird. Der Charakter wird rotiert, wenn die Ausrichtung (direction) nicht Null ist.

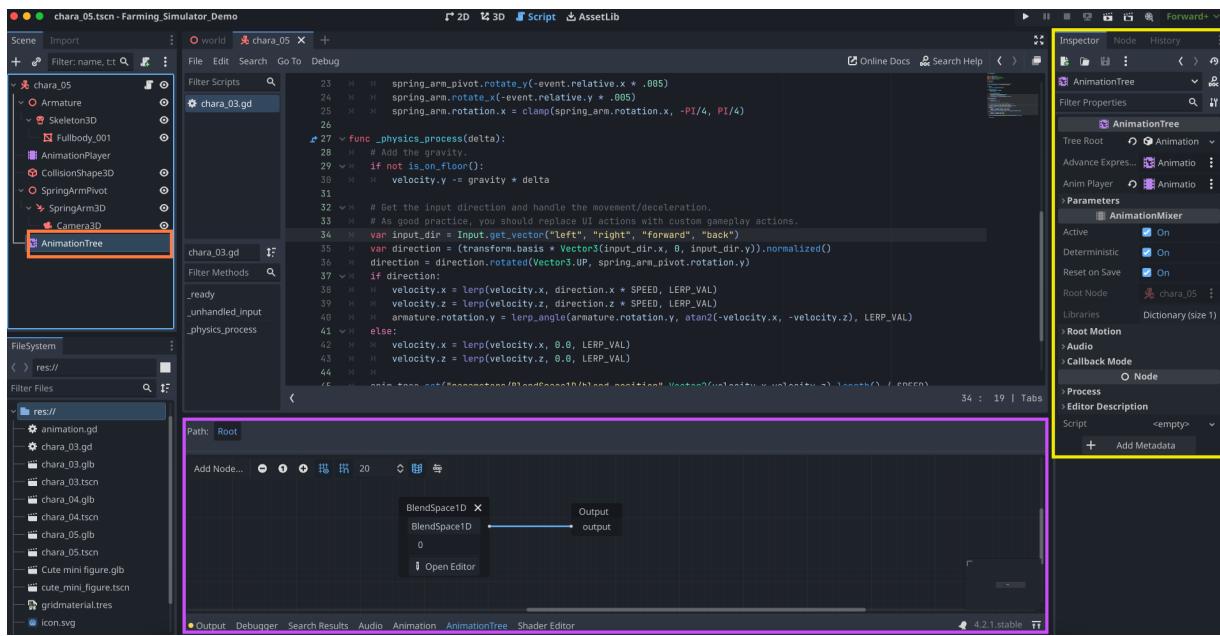


Abb. 3.25: Godot AnimationTree

Wenn der Charakter sich jetzt mit der Ausrichtung bewegt, ist der nächste Schritt das Hinzufügen der Animation. Dies wird durch einen AnimationTree-Knoten erreicht, wie in Abbildung 3.25 zu sehen ist. Zuerst fügt man den AnimationPlayer hinzu und erstellt ein BlendSpace1D.

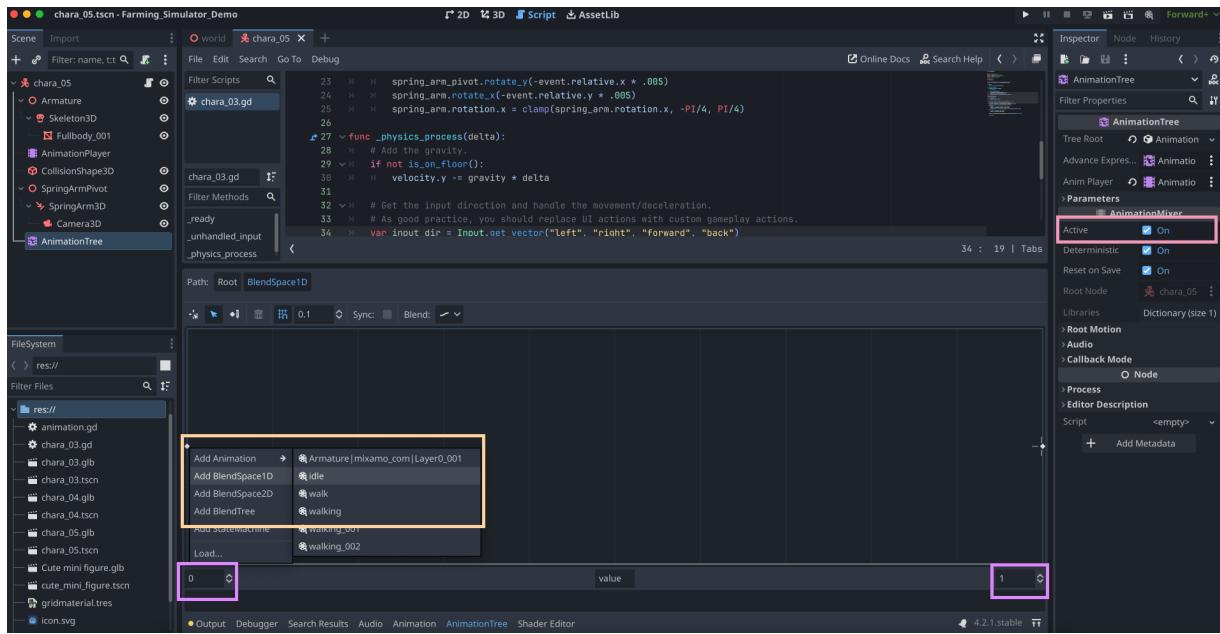


Abb. 3.26: Godot AnimationTree Editor

Danach öffnet man den Editor für das BlendSpace1D und fügt jeweils für Rechts und Links durch Rechtsklick die entsprechende Animation hinzu. Links fügt man die "Idle"-

Animation hinzu und rechts die "Walk"-Animation. Man setzt den Wert für Links auf 0 und für Rechts auf 1. Diese Werte bestimmen die Länge des Geschwindigkeitsvektors des Charakters. Anschließend verbindet man, wie in Abbildung 3.25 gezeigt, den BlendSpace1D mit dem Output und aktiviert den AnimationTree.

Um die Animation flüssig abzuspielen, muss die Geschwindigkeit interpoliert werden, und im Skript muss der Pfad (property Path) des AnimationTrees als Parameter hinzugefügt werden. Nun bewegt sich der Charakter flüssig und wechselt seine Animation je nach Geschwindigkeit zwischen "Idle" und "Walk".

3.2.6 Umsetzung des Inventarsystems

Kapitel 4

Schluss

Ergebnis-Bewertung, Zusammenfassung und Ausblick

Literatur

1. JOHNSON, J. *Godot 4 Game Development Cookbook: Over 50 solid recipes for building high-quality 2D and 3D games with improved performance*. Packt Publishing, 2023. ISBN 9781838827250. Auch verfügbar unter: <https://books.google.de/books?id=FNS-EAAAQBAJ>.
2. *Godotengine download macOs*. [o. D.]. Auch verfügbar unter: <https://godotengine.org/download/macos/>. Zugegriffen: 2023-25-12.
3. BRADFIELD, C. *Godot 4 Game Development Projects: Build five cross-platform 2D and 3D games using one of the most powerful open source game engines*. Packt Publishing, 2023. ISBN 9781804615621. Auch verfügbar unter: <https://books.google.de/books?id=GrfLEAAAQBAJ>.
4. *Godot Docs – 4.2 branch — docs.godotengine.org* [<https://docs.godotengine.org/en/stable/index.html>]. [o. D.]. [Accessed 30-12-2023].
5. *AnimalCrossing* [<https://i.pinimg.com/originals/07/f7/ec/07f7ec705585cd7a9c.jpg>]. [o. D.]. [Accessed 14-01-2024].
6. SAVIOR. *Inventory profiles Mod (1.20.4, 1.19.4) - inventory sorting tweaks*. 2023. Auch verfügbar unter: <https://www.9minecraft.net/inventory-profiles-mod/>. [Accessed 01-02-2024].
7. *Getting Started | Nomad* — nomadsculpt.com/manual/gettingstarted. [o. D.]. [Accessed 16-01-2024].
8. *Mixamo Startpage*. [o. D.]. Auch verfügbar unter: <https://www.mixamo.com/#/>. [Accessed 01-02-2024].