

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

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

Ralf Zeller

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	Kurzbeschreibung	1
1.2	Motivation	1
2	Grundlagen der Godot Game Engine	2
2.1	Einführung in Godot	2
2.1.1	Besonderheiten	2
2.1.2	Installationsquellen für Godot	3
2.2	Godot Editor	4
2.2.1	Projekt Manager	4
2.2.2	Godot Editor Übersicht	5
2.3	Nodes and Szenes	7
2.3.1	Nodes	7
2.3.2	Szenen	7
2.3.3	Hierarchie von Nodes	8
2.4	GDScript	8
2.4.1	Was ist GDScript?	9
3	Realisierung des Farming Simulators	10
3.1	Spielidee und Motivation	10
4	Raycast	12
4.1	Einführung in Raycast	12
4.2	Spielercharakter mit RayCast3D Node	13
4.2.1	RayCast3D Inspector Einstellungen	14
4.2.2	Rotationskorrektur RayCast3D Code	15
4.2.3	RayCast3D Debug Options	16
5	Gridmap	17
5.1	MeshLibrary	17
5.1.1	Erstellen von Blöcken in Blender	17
5.1.2	Erstellen einer MeshLibrary	19
5.1.3	Gridmap erstellen	21

6	Pflanzbare Gemüse	22
6.1	Erstellen einer Pflanzen Szene	22
6.1.1	Gemüse Szene in Blender vorbereiten	22
6.1.2	Backen von Texturen	23
6.2	GDScript Skript für Gemüse	24
7	Fazit und Ausblick	27
7.1	Fazit	27
7.2	Ausblick	28

Abbildungsverzeichnis

2.1	Godot Logo	2
2.2	Godot Engine MacOs Installer	3
2.3	Godot Projekt Manager	4
2.4	Godot Editor Overview	5
2.5	Godot FileSystem Panel	5
2.6	Godot Scene Panel	6
2.7	Godot Inspector Panel	6
2.8	Nodes Baumstruktur	7
2.9	GDScript Example	9
3.1	Mindmap: Ideen und Gedanken	10
4.1	Spielercharakter Raycast Node	13
4.2	Spielercharakter Raycast Scene Tree	13
4.3	Spielercharakter Inspector Raycast	14
4.4	Spielercharakter Inspector Node3D	14
4.5	Raycast Debug Options	16
5.1	Blender Cube Mesh Scene	18
5.2	Blender Export Meshes	18
5.3	Erstellen einer Szene aus einem .glb	19
5.4	Instanzierte Szene	19
5.5	Create Trimesh Static Body Menüpunkt	20
5.6	Erstellte Nodes nach Menüpunkt	20
5.7	Export als MeshLibrary	21
5.8	Gridmap Editor	21
6.1	Gemüse Szene in Blender	22
6.2	Blender erstellte Texturen	23
6.3	Blender Image Texture	23
6.4	Textur Back Einstellungen	24

Abkürzungsverzeichnis

UI User Interface

API Application Programming Interface

Kapitel 1

Einleitung

Die vorliegende Arbeit beschäftigt sich mit der Entwicklung eines 3D Farming Simulators mit einer zeitgemäßen Game Engine. Für die Erstellung der Game Assets werden

1.1 Kurzbeschreibung

Das Ziel dieses Projektes ist die Entwicklung eines Farming Simulator mit der Game Engine Godot v4.2.1.stable.official unter der Verwendung der aktuellen Methoden und Möglichkeiten .

Die vorliegende Arbeit dokumentiert den gesamten Prozess der Entwicklung des Simulators. Der Prozess beinhaltet sowohl die Planung, die Erstellung des Farming Simulators, als auch die weitere Aspekte, die zur Entwicklung des Projektes beitragen. In den letzten Jahren wurde das Genre Simulator im Bereich Farming von vielen bekannten Game Titeln wie Stardew Valley, Animal Crossing und viele weitere bekannt gemacht.

1.2 Motivation

Das Projekt soll das Erlernen der aktuellen zeitgemäße Game Engine Godot ermöglichen. Das erlernte Wissen soll Prozesse und Strukturen heutiger Game Engines näher bringen.

Kapitel 2

Grundlagen der Godot Game Engine

Im folgendem Kapitel wird ein kurzer Einblick in die Godot Engine und den Godot Editor gegeben. Hiermit soll ein grobes Verständnis darüber gegeben werden, wie man mit Godot entwickelt.

2.1 Einführung in Godot

Die Godot Game Engine ist eine freie Open-source Game Engine. Sie wurde von Juan Linietsky und Ariel Manzur im Jahr 2007 kreiert. Später wurde die Engine unter der MIT Lizenz im Jahr 2014 veröffentlicht ([1, S.1](#)). In der Abbildung [2.1](#) kann man das derzeitige Logo von Godot sehen.



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

2.1.1 Besonderheiten

Man hat die Godot Game Engine mit einer sehr großzügigen MIT Lizenz veröffentlicht. Zusätzlich ist sie komplett kostenlos und Open-source. Alles was mit der Engine entwickelt wird, gehört vollständig einem selbst. Dadurch werden versteckte Lizenzkosten verhindert, wie sie bei anderen Game Engines vorkommen können ([3, S.4](#)).

Die Entwicklung der Engine wird komplett durch eine Open-source Community vorangetrieben. Man kann sich selbst an der Entwicklung beteiligen. Ebenfalls kann man eine Veränderung ohne benötigte Erlaubnis am Source Codes vornehmen und diese für das eigene Projekt nutzen. Somit erhält man sehr viel Freiheit wie man die Godot Engine für das eigene Projekt verwenden möchte (3, S.4-5).

2.1.2 Installationsquellen für Godot

Damit man Godot mit aktuellen Features für die Entwicklung nutzen kann, wird die neueste Version benötigt. Sie wird im Bereich 4.x.x liegen. Man kann die Version unter der Webseite <https://godotengine.org/download/> herunterladen. Es werden zwei Varianten angeboten. Die erste Variante ist die normale Version. Die zweite Variante bietet eine C# Unterstützung. Man sollte die C# Variante nur für Projekte nutzen, die mit der Programmiersprache C# geschrieben werden sollen (3, S.5).

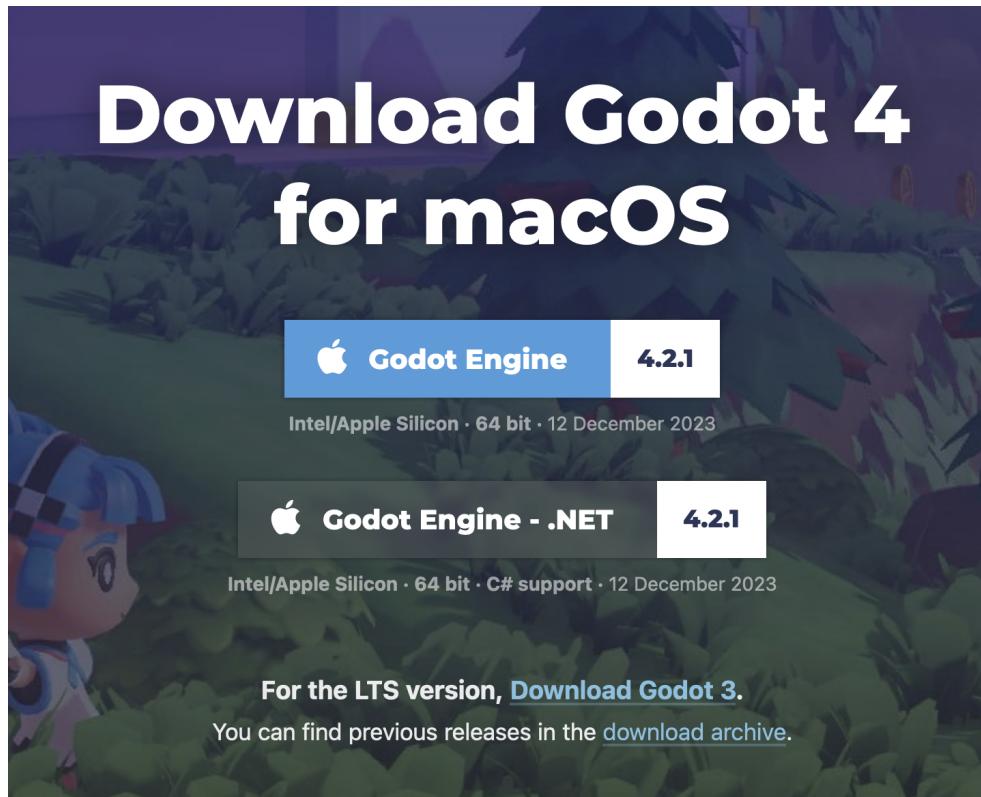


Abb. 2.2: Godot Engine MacOs Installer (4)

In der Abbildung 2.2 wird der Download für MacOs angezeigt. Auf der Download Seite von Godot gibt es weitere Versionen für andere Betriebssysteme wie Linux und Windows. Sie können beim Herunterscrollen auf der Downloadseite ausgewählt werden. Die Godot Game Engine kann über Steam, itch.io oder verschiedene Package Manager installiert werden (3, S.5-6).

2.2 Godot Editor

In diesem Abschnitt soll näher auf den Godot Editor eingegangen werden.

2.2.1 Projekt Manager

Beim Ausführen von Godot wird zuerst der Projekt Manager angezeigt. Von hier aus kann man Entscheiden, ob eine neues Projekt erstellt werden soll oder ein bestehendes geöffnet werden soll. Beim Erstellen eines neuen Projekts kann man ein online verfügbares Template nutzen oder man nutzt den Standard Dialog zum Erstellen des Projektes. Bereits bestehende oder importierte Projekte kann man im Projekt Manager bearbeiten (3, S.7-8).

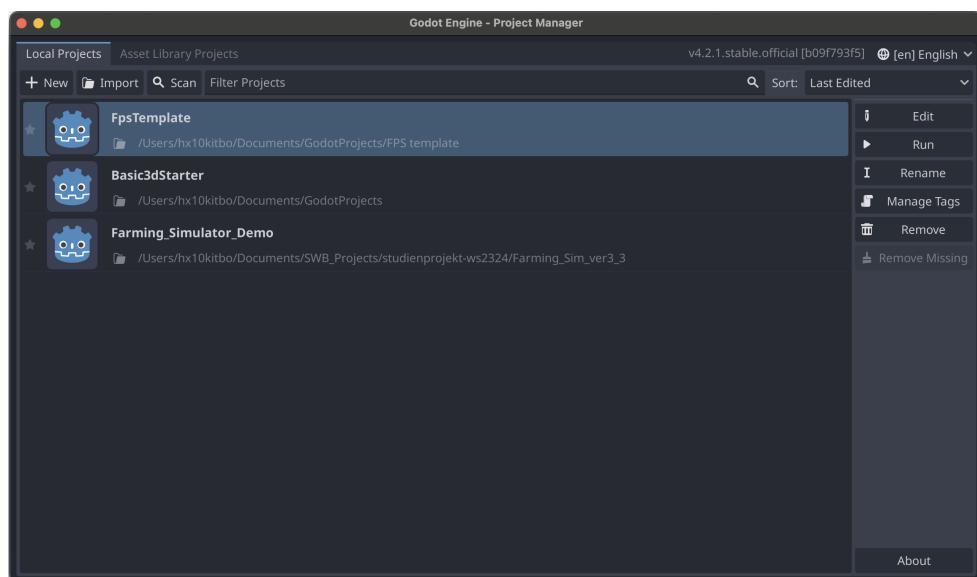


Abb. 2.3: Godot Projekt Manager

In der Abbildung 2.3 wird der Projekt Manager wie er aussehen könnte dargestellt.

2.2.2 Godot Editor Übersicht

Der Godot Editor hat vier Hauptansichten. Die vier Hauptansichten sind 2D, 3D, Script und AssetLib. In der Abbildung 2.4 können die Hauptansichten in der Mitte der Fensterleiste ausgewählt werden. Wählt man eine der Ansichten wird das Panel in der Mitte auf die ausgewählte Ansicht gewechselt. Beispielsweise wird die Ansicht Script gewählt, dann wird in der Abbildung 2.4 die 3D Ansicht auf eine Script Ansicht gewechselt.

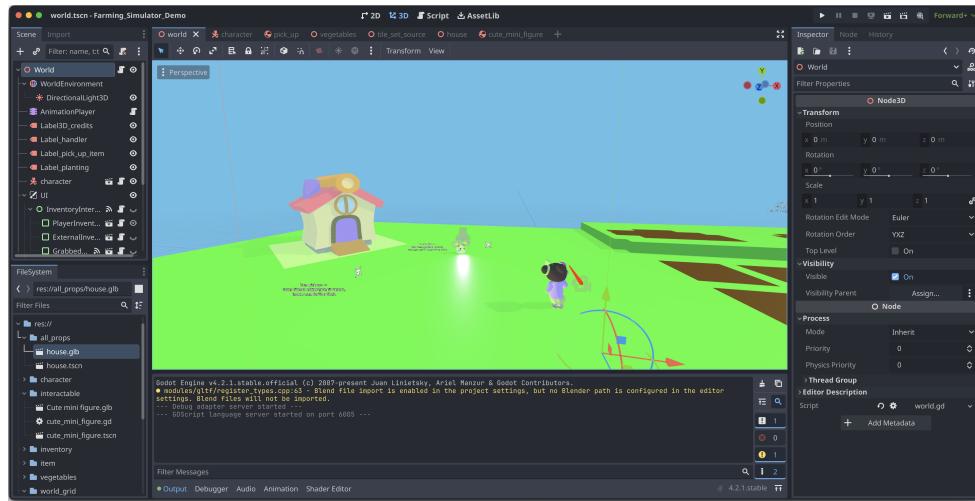


Abb. 2.4: Godot Editor Overview

Der gesamte Editor besteht aus vereinzelten Panels. Den größten Platz nimmt der Viewport ein, der durch die Auswahl einer Hauptansicht gewechselt werden kann (3, S.9).

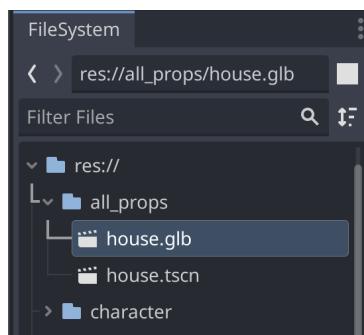


Abb. 2.5: Godot FileSystem Panel

Das Filesystem Panel zeigt alle Ressourcen des Projektes an. Es lässt sich bedienen wie das Filesystem eines Betriebssystems. Die Ressourcen des Projektes können relativ zum Pfad `res://` im Projekt zugegriffen werden (3, S.10). In der Abbildung 2.5 kann man eine Möglichkeit zur Ordnung von Ressourcen sehen.

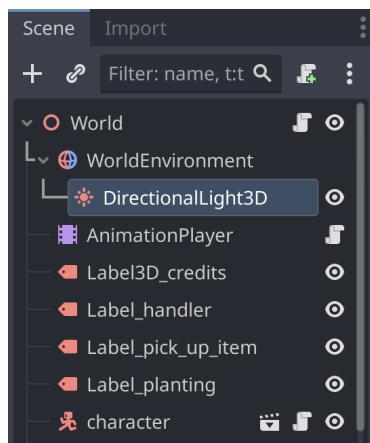


Abb. 2.6: Godot Scene Panel

Das Scene Panel zeigt die aktuell geöffnete Szene an. In ihr werden alle Objekte angezeigt, die eine Szene erzeugen. Die Abbildung 2.6 zeigt eine Szene in der World, der Hauptknoten der Szene ist.

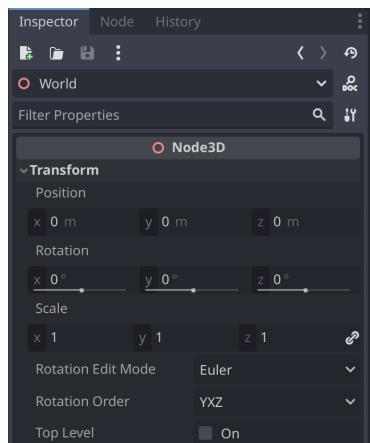


Abb. 2.7: Godot Inspector Panel

Das Inspector Panel zeigt alle Eigenschaften eines ausgewählten Objektes einer Szene. Bei der Abbildung 2.7 wird ein Teil der Eigenschaften, die vom ausgewählten Objekt World verändert werden können, dargestellt.

2.3 Nodes and Szenes

In diesem Abschnitt wird näher auf Szenen und Nodes in Godot eingegangen.

2.3.1 Nodes

In Godot werden Nodes als Grundbaustein genutzt. Jede Node kann spezialisierte Eigenschaften und Funktionen haben. Möchte man die Fähigkeiten einer Node erweitern, wird ein Skript benötigt mit dem das Verhalten definiert wird. Dadurch kann ein bestimmtes Verhalten von einzelnen Nodes erzeugt werden. Beispielsweise können Nodes Elemente des User Interface (UI) darstellen, einen bewegbaren Spielcharakter oder vieles mehr. In einer Baumstruktur organisierte Nodes nennt man Szene (3, S.11-12).

2.3.2 Szenen

In Godot strukturiert und organisiert man Spielinhalte in Szenen. Eine Szene kann aus verschiedenen Elementen, wie z.B. einem Spielcharakter, einer Lebensanzeige und vieles mehr bestehen (3, S.11-12). Die Engine ermöglicht die Verwaltung einzelner Spiellevel in separaten Szenen. Dadurch werden nur die benötigten Spielinhalte für das aktuelle Spiellevel geladen. Jede Szene benötigt eine Root-Node von der einzelne Nodes abzweigen. In der Abbildung 2.6 wird eine Szene im Scenen Panel mit Baumstruktur dargestellt.

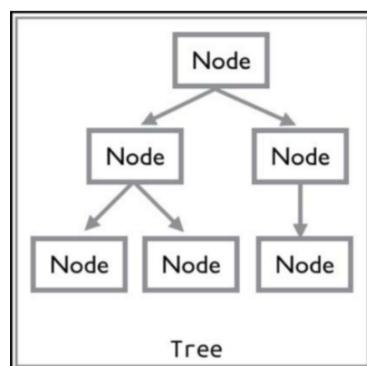


Abb. 2.8: Nodes Baumstruktur (3, S.11)

2.3.3 Hierarchie von Nodes

Die Godot Engine ermöglicht das Gestalten einer Spielszene aus Unterszenen. Somit kann eine Szene genutzt werden zur Beschreibung des Spiellevels und eine weitere Szene für die Gestaltung des Spielcharakters. Dadurch kann der Spielcharakter sehr einfach in eine andere Spielszene gewechselt werden (3, S.11-12). Die Abbildung 2.8 zeigt eine Anordnung von Nodes in einer Baumstruktur. In der Anordnung kann jede angehängte Node eine Szene sein. Das ermöglicht das Erstellen von sehr komplexen Spielinhalten. Beispielweise könnte als Root-Node eine Welt genutzt werden, der eine Lebensanzeige anhängt ist. Die Lebensanzeige ist eine eigenständige Szene in der alle ihre Funktionen definiert sind. Sie könnte aus mehreren einfachen 2D Bildinhalten bestehen, die per Skript einzelne Elemente je nach Stand der Lebenspunkte, einblendet.

2.4 GDScript

In Godot werden zum Schreiben von Skripten die Sprachen GDScript und C# angeboten. Beide Sprachen werden offiziell unterstützt. GDScript ist sehr stark in die GODOT Application Programming Interface (API) integriert (3, S.12).

Die Engine ist in C++ geschrieben und kann um weitere Funktionalitäten erweitert werden. Dadurch erhält man mehr Kontrolle über die Fähigkeiten der Engine und kann die Performenz verbessern (3, S.12).

Es empfiehlt sich die Verwendung von GDScript. Sie ermöglicht einen einfachen und schnellen Start im Erstellen von Spielen mit Godot (3, S.12).

2.4.1 Was ist GDScript?

Die Syntax der Skriptsprache GDScript wurde nach dem Vorbild Python nachgebildet. Jemand der Erfahrung mit der Sprache Python hat sollte sich mit GDScript zurecht finden können. Die Sprache GDScript verwendet eine dynamische Typisierung. Dadurch müssen die Datentypen der Variablen nicht mit einem bestimmten Typ wie beispielsweise int definiert werden. Es werden Einrückungen verwendet zum Markieren von Codeblöcken (3, S.13).

```
extends Sprite2D
var speed = 200

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

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

Abb. 2.9: GDScript Example (3, S.13)

In der Abbildung 2.9 sieht man einen Beispielcode in GDScript. Der Code lässt ein Sprite2D von links nach rechts laufen (3, S.13). Eine Referenz zu GDScript Sprache findet man in der offiziellen Dokumentation unter der Adresse: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html finden.

Kapitel 3

Realisierung des Farming Simulators

In diesem Kapitel wird beschrieben welche Überlegungen in die Entwicklung des Farming Simulators einfließen.

3.1 Spielidee und Motivation

Als erstes wurden die Gedanken und Ideen in der folgenden Mindmap 3.1 gesammelt. Die Mindmap dient dazu einen Überblick über die Entwicklung und Konzeption des Farming Simulators zu erhalten.

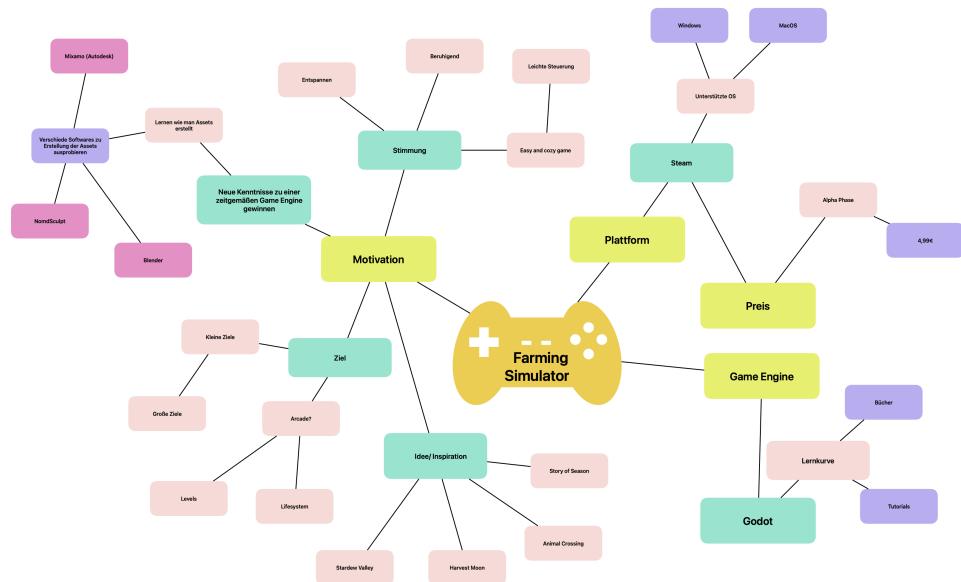


Abb. 3.1: Mindmap: Ideen und Gedanken

Die erste Frage mit der sich das Projekt beschäftigt ist: "Warum überhaupt ein Farming Simulator?"

Die Motivation für Spieler, die dieses Spielkonzept bietet ist, dass der Spieler selbst bestimmen kann in welcher Geschwindigkeit der Fortschritt in der Spielwelt vornanschreitet. Zusätzlich wird in diesem Konzept oft kein gefährlicher Gegenspieler vermittelt, der den Spieler in zeitliche Bedrängnis bringt. Die Idee wurde in bekannten Farming-Simulatoren wie Animal Crossing, Harvest Moon, Stardew Valley und Story of Seasons realisiert. Dem Spieler wird eine beruhigende und entspannende Atmosphäre vermittelt.

Für das Projekt gibt es Entwicklungsziele. Mit dem Projekt sollen neue Kenntnisse in einer zeitgemäßen Game Engine gewonnen werden. Es sollen verschiedene Programme wie Blender oder Nomadsculpt zur Erstellung von Game Assets verwendet werden. Bei der Entwicklung sollen vorrangig kleine Ziele verfolgt werden, die in der Summe auf ein großen Fortschritt hinsteuert.

Während des Entwicklungsprozesses stellte sich folgende Frage: "Sollte man ein Arcade-Element integrieren?" Auf diese Frage hin wurden Überlegungen darüber angestellt, welche Motivation einen Spieler am Spiel halten könnte. In Rollenspielen wird dem Spieler kleine Aufgaben gegeben, die der Spieler lösen kann. Bei einem Farming Simulator muss der Spieler eine Pflanze mit einem Samen einpflanzen und wartet bis die Pflanze ausgewachsen ist. Die ausgewachsene Pflanze kann geerntet werden und gibt weitere Samen, die genutzt werden können zum Anpflanzen weiterer Pflanzen. Für den Spieler ergibt sich hierbei ein Zyklussystem aus Einpflanzen und Ernten.

Ein weiterer Aspekt ist die richtige Auswahl der Game Engine. Vor allem für Neulinge in der Spieleentwicklung ist die Wahl nicht sehr leicht. Godot wird für Anfänger und kleine Indie Spieleentwicklung empfohlen. Das Erlernen der Engine ist nicht stetig steigend und manche Elemente erfordern einen erheblichen Zeitaufwand. Damit das Erlernen erleichtert wird gibt es eine offizielle Godot Dokumentation, die man bei Lernen und Problemen studieren sollte. Zusätzlich sollte man sich Bücher und Video Tutorials für die Themen, die einen Interessen besorgen.

Bei der Frage, auf welcher Platform soll es veröffentlicht werden, ergibt sich die weitere Frage: "Wo soll es hochgeladen werden?" Es gibt mehrere Platformen auf dem das Spiel veröffentlicht werden könnte. Steam wäre eine Möglichkeit zur Veröffentlichung des Spiels. Ein weitere Eigenschaft die man beim Spiel beachten sollte, ist auf welchen Betriebssystem das Spiel laufen soll. Möchte man sich mit dem Verkaufspreis beschäftigen. Dann wäre für die Alpha-Phase ein Preis der etwa bei 4,99€ liegt, gut angesetzt.

Mit allen vorangegangenen Überlegungen erhält man einen groben Überblick über die Aspekte der Entwicklung eines Farmings Simulators.

Kapitel 4

Raycast

In diesem Kapitel wird beschrieben wie ein Raycast in einem 3D Spiel zur Interaktion mit Objekten der Spielwelt verwendet werden kann.

4.1 Einführung in Raycast

Ein Raycast ermöglicht das Interagieren mit anderen Objekten in der Spielwelt. Beispielsweise wird ein Raycast von einem Spieler abgesendet, wird der Strahl Informationen über eine Kollision mit einem Objekt aus der Spielwelt als Antwort an den Spieler zurückgeben.

In Godot gibt es zwei Arten von Raycasts. Die erste Variante des Raycasts wird mit der Node RayCast3D realisiert. Die zweite Variante verwendet einen Zugriff auf den World-space und ermöglicht durch den Zugriff eine Kollision in der Welt herauszulesen. Für das Projekt wurde die erste Variante verwendet. Dadurch wird die zweite Variante in diesem Projekt vernachlässigt.

4.2 Spielercharakter mit RayCast3D Node

Damit der Spieler RayCast3D Node in Godot verwenden kann, muss die Node an dem Spieler Charakter angebracht werden. In der Abbildung 4.1 sieht man die angebrachte RayCast3D Node an der Position der Kamera des Spielers.

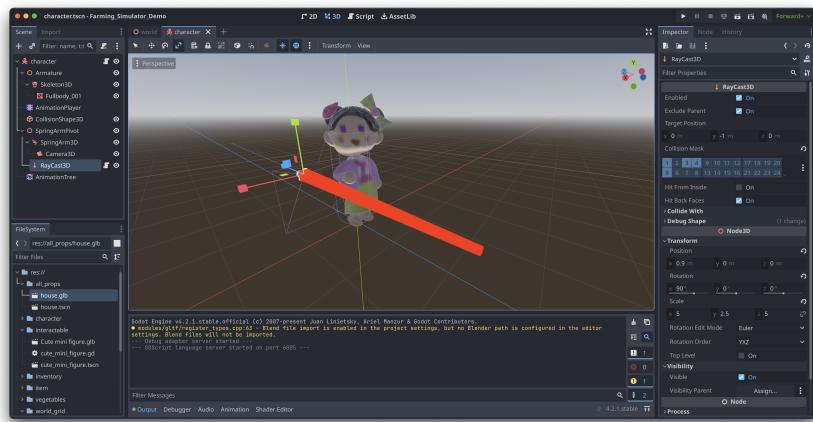


Abb. 4.1: Spielercharakter Raycast Node

In der Szene ist der RayCast3D Node am SpringArmPivot angebracht. Dadurch wird dem Spieler ermöglicht den Raycast mit der Kamera gleichzeitig zu drehen.

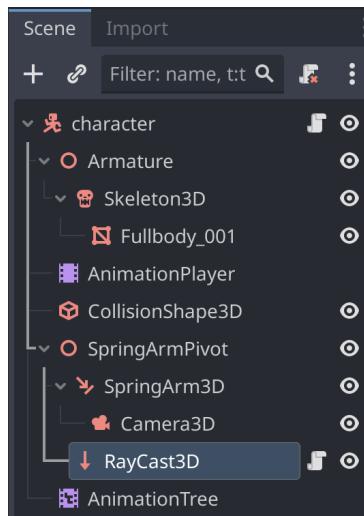


Abb. 4.2: Spielercharakter Raycast Scene Tree

Auf der Abbildung 4.2 wird die Struktur der Szene des Spielcharakters gezeigt.

4.2.1 RayCast3D Inspector Einstellungen

In den Inspector Einstellungen der RayCast3D Node können mehrere Einstellungen vorgenommen werden.

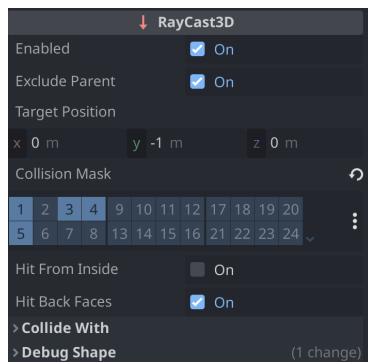


Abb. 4.3: Spielercharakter Inspector Raycast

Die RayCast3D spezifischen Einstellungen betreffen die Erfassung von anderen Objekten in der Spielwelt. Hier kann ausgewählt werden auf welche Kollisionsmasken die Node eine Kollisionsantwort geben soll. In der Abbildung 4.3 kann man einen Teil der Einstellungen dder Charakter Szene sehen

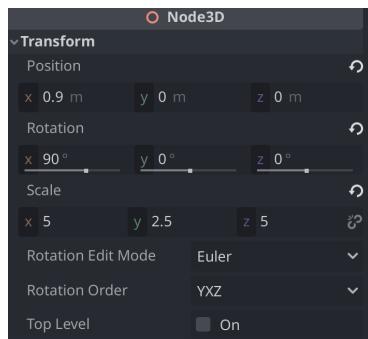


Abb. 4.4: Spielercharakter Inspector Node3D

Die Node3D Einstellungen des RayCast3D Nodes ermöglichen die Transformation im dreidimensionalen Bereich. Unter den Einstellungspunkten kann die Größe des Strahls bestimmt werden.

4.2.2 Rotationskorrektur RayCast3D Code

Die RayCast3D Node muss zusätzlich im verbundenen Skript mit der Character Szene, um Rotationskorrekturen ergänzt werden. Dadurch kann die Node die aktiven Kamerabewegungen nachmachen und ermöglicht eine Third-Person Steuerung des Charakters.

List. 4.1: character.gd

```

1    extends CharacterBody3D
2
3    @onready var ray = $SpringArmPivot/RayCast3D
4    ...
5
6    func _unhandled_input(event):
7        if Input.is_action_just_pressed("quit"):
8            get_tree().quit()
9
10       if event is InputEventMouseMotion:
11           spring_arm_pivot.rotate_y(-event.relative.x * .005)
12           spring_arm.rotate_x(-event.relative.y * .005)
13           spring_arm.rotation.x = clamp(spring_arm.rotation.x, -PI/4,
14                                           PI/4)
15
16           # Ray movement with mouse
17           ray.rotate_x(-event.relative.y * .005)
18           ray.rotation.x = clamp(ray.rotation.x, PI/4, PI/2)
19
20           # Character movement with camera
21           armature.rotate_y(-event.relative.x * .005)
22
23               # Character movement with camera
24               armature.rotate_y(-event.relative.x * .005)
25
26   ...

```

In der Code Darstellung 4.1 wird die Rotationskorrektur für die RayCast3D Node abgebildet.

4.2.3 RayCast3D Debug Options

Godot bietet mehrere Optionen zum Debuggen eines Spiels. Eine wichtige Option "Visible Collision Shapes". Mit der aktivierte Option können RayCast3D Nodes in einer interaktiven Spielszene angezeigt werden.

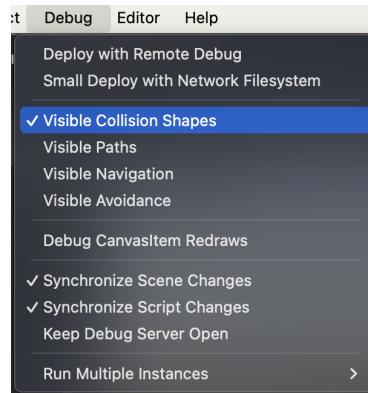


Abb. 4.5: Raycast Debug Options

Die Abbildung 4.5 zeigt den Menüpunkt unter der man die Debug Option im Editor aktivieren kann.

Kapitel 5

Gridmap

In Godot kann man mit einer Gridmap eine Spielwelt erstellen. Die einzelnen Blöcke der Gridmap werden in einer MeshLibrary definiert. Gridmaps ermöglichen ein effizientes Rendern der Spielwelt.

5.1 MeshLibrary

In einer MeshLibrary werden die Blöcke, die in einer Gridmap plaziert werden können, zusammengefasst.

5.1.1 Erstellen von Blöcken in Blender

Mit Blender kann man eine Szene erstellen, die alle Blöcke beinhaltet, die später in die Gridmap platziert werden sollen. Damit der Prozess reibungslos funktioniert wurde die Blender Version 3.6 LTS verwendet. Bei neueren Versionen von Blender können Problemen beim Exportieren vorkommen.

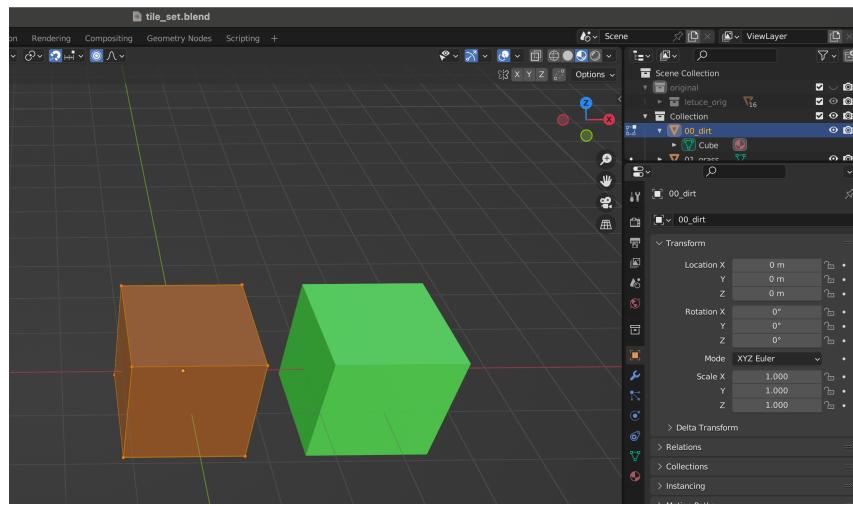


Abb. 5.1: Blender Cube Mesh Scene

Im Editor von Blender muss darauf geachtet werden, dass alle Objekte die Scale von 1.0 haben. In der Abbildung 5.1 kann man die skalierten Objekte erkennen. Bei Objekten die eine andere Scale haben, muss man zuerst "Apply All Transforms" angewendet werden. Sonst können die Objekte später beim Importieren falsch dargestellt werden.

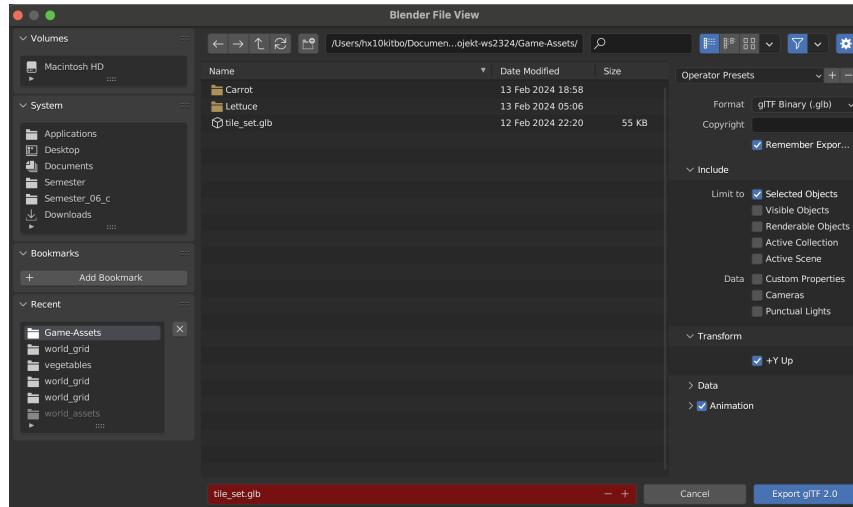


Abb. 5.2: Blender Export Meshes

Bevor die Objekte exportiert werden können, wählt man alle Objekte die exportiert werden sollen aus. Anschließend wird aus dem Menü der Punkt "glTF 2.0 (.glb/.gltf)" ausgewählt. Daraufhin wird ein Dialog geöffnet in dem "Selected Objects" und "+Y Up" ausgewählt sein müssen. Mit den vorgenommenen Einstellungen kann die Szene exportiert werden. Die Abbildung 5.2 zeigt einen möglichen Dialog zum Exportieren der Datei.

5.1.2 Erstellen einer MeshLibrary

Die exportierte Datei .gltf muss zum Projektordner hinzugefügt werden. Mit einem Rechts-klick auf die Datei im Filesystem Panel, kann die Szene instanziert werden.

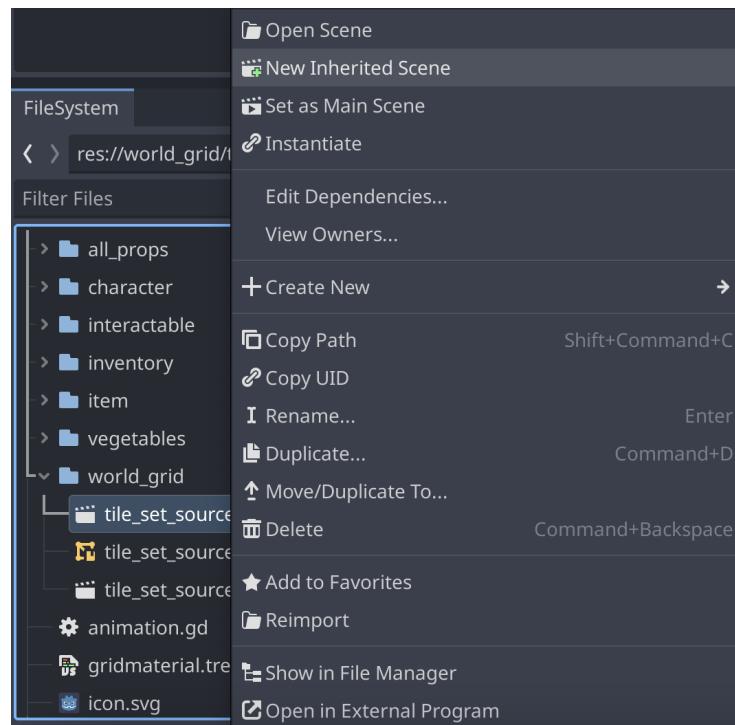


Abb. 5.3: Erstellen einer Szene aus einem .gltf

Die Abbildung 5.3 zeigt das Auswahlmenü. Im Menü sollte der Menüpunkt "New Inherited Scene" ausgewählt werden. Anschließend wird eine neue Szene erstellt. In der neu erstellten Szene werden alle Objekte als Mesh instanziert.

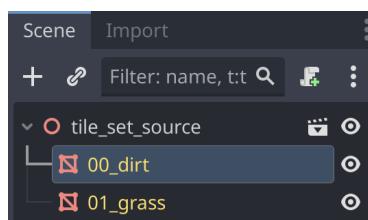


Abb. 5.4: Instanzierte Szene

In der Abbildung 5.4 sind die instanzierten Blöcke als Mesh Nodes sichtbar. Damit in der späteren Welt die Nodes eine Kollision aufweisen, muss man noch einen StaticBody3D und eine CollisionShape3D an jede Mesh anhängen. Sehr einfach geht es indem man eine Node im Scene Panel auswählt. Anschließend im Viewport Panel den Menüpunkt "Mesh" auswählt.

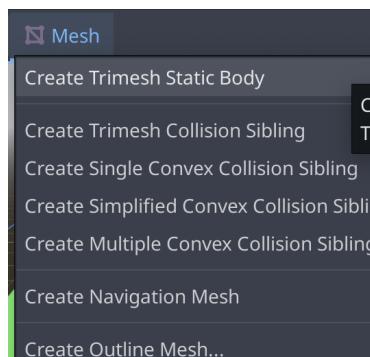


Abb. 5.5: Create Trimesh Static Body Menüpunkt

Die Abbildung 5.5 zeigt die Menüpunkte die nach dem Klicken auf "Mesh" im Viewport angezeigt werden. In der Auswahl wählt man "Create Trimesh StaticBody". Dadurch wird für die aktiv ausgewählte Node ein StaticBody3D und eine CollisionShape3D angehängt. Gefällt einem das Kollisionsverhalten der Node nicht, kann man die CollisionShape3D auswählen der Node auswählen und im Inspector die CollisionShape3D verändern.

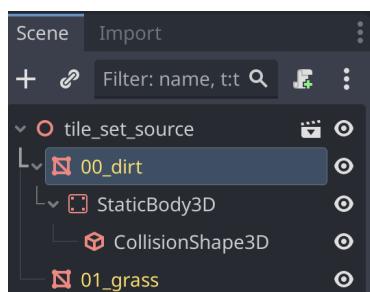


Abb. 5.6: Erstellte Nodes nach Menüpunkt

In der Abbildung 5.6 kann man das Ergebnis im Scene Panel nach erfolgreichem Ausführen des Menüpunkts "Create Trimesh StaticBody" sehen. Nachdem alle Nodes einen eignen StaticBody3D und eine CollisionShape3D haben, kann die Szene als eine MeshLibrary exportiert werden. Damit man einen Import Export erstellen kann, muss der Menüpunkt "Scene -> Export As -> MeshLibrary" aufgerufen werden.

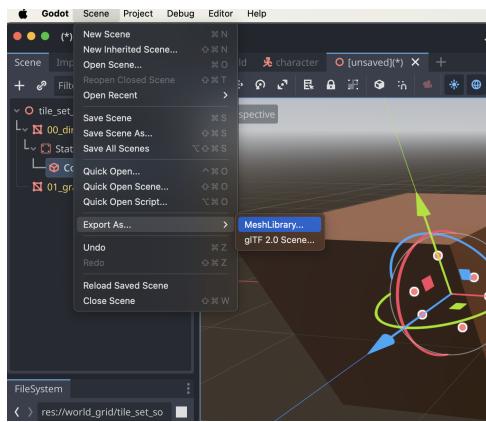


Abb. 5.7: Export als MeshLibrary

In der Darstellung 5.7 kann man einen Screenshot sehen, der das Menü "Export As" und den Menüpunkt "MeshLibrary" anzeigt.

5.1.3 Gridmap erstellen

Auf der Hauptszene erstellt man im Scene Panel eine GridMap Node. Man fügt in der Inspector Ansicht die MeshLibrary der GridMap Node hinzu. Die Größe der einzelnen Blöcke kann in der Inspector Ansicht der GridMap eingestellt werden. Zusätzlich lässt sich auch Orientierung der einzelnen Blöcke auf den Koordinatenachsen einstellen.

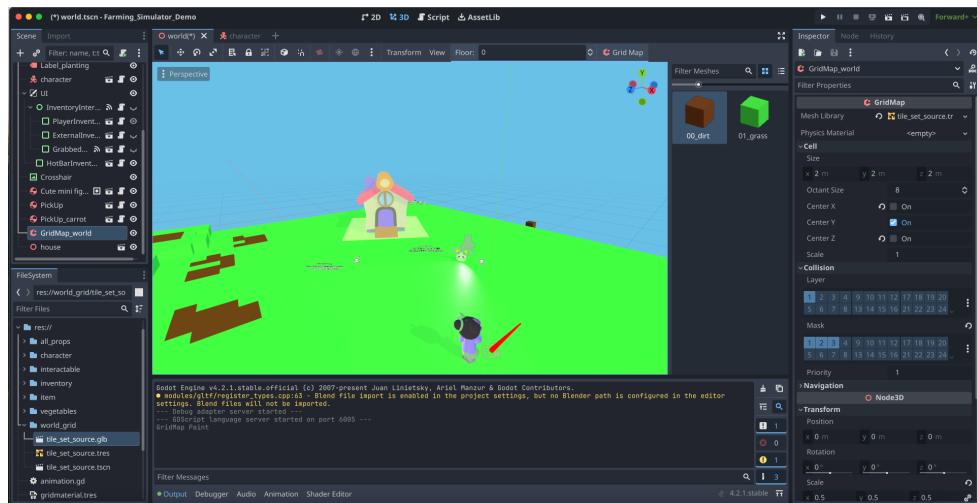


Abb. 5.8: Gridmap Editor

Die Abbildung 5.8 zeigt eine Gridmap, deren Zentrierung der einzelnen Blöcke auf der y-Achse eingestellt ist. Dadurch wird jede gebaute Block in der GridMap auf der y-Achse zentriert. Über den Viewport Editor können verschiedene Meshes aus der Meshlib ausgewählt werden, um diese aktiv in die Gridmap einzufügen.

Kapitel 6

Pflanzbare Gemüse

In diesem Kapitel wird behandelt wie man pflanzbare Gemüse in Blender erstellt. Die fertigen Modelle aus Blender exportiert und als neue Szene in Godot importiert.

6.1 Erstellen einer Pflanzen Szene

Damit man erstellte Pflanzen aus Blender in die Godot Engine importieren kann, müssen einige Schritte vorab getan werden. Die nachfolgenden Schritte beschreiben wie man ein Blender Modell mit Texturen importierbar für Godot machen kann.

6.1.1 Gemüse Szene in Blender vorbereiten

Als erstes wird in Blender eine Szene mit Gemüse erstellt. Hierbei ist wichtig, dass alle Gemüse Modelle die exportiert werden sollen von Scale 1.0 sind. Sonst müssen auf alle Modelle alle Transformationen angewendet werden.

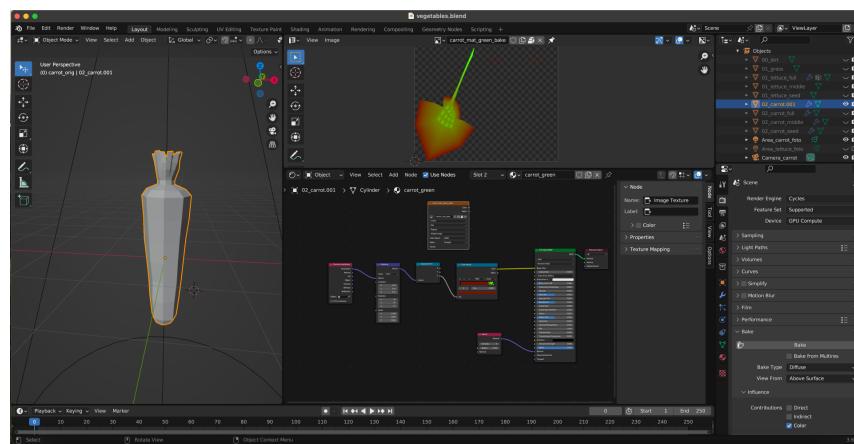


Abb. 6.1: Gemüse Szene in Blender

In der Abbildung 6.1 wird eine fertige Szene in Blender mit Texturen gezeigt. Mit Blender kann man Meshes eine Textur geben. Die Textur kann durch verschiedene Funktionen verändert werden, die man durch Backen extrahieren kann.

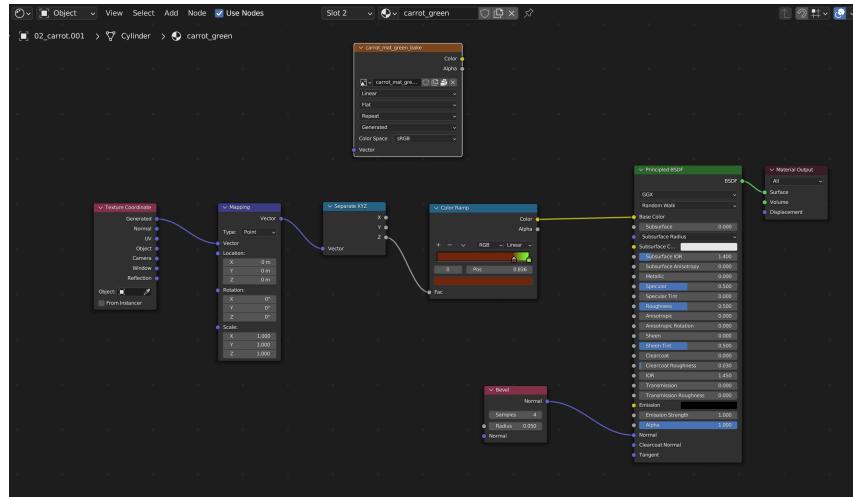


Abb. 6.2: Blender erstellte Texturen

Die Darstellung 6.2 zeigt die Textur einer Karotte. Die Textur wurde durch verschiedene Funktionen verändert.

6.1.2 Backen von Texturen

Damit man die generierte Textur in Blender auch später in Godot verwenden kann. Muss mit Textur Backen eine zusätzliche Textur erstellt werden, die man später in Godot importieren kann. Jedes Mesh des Modells das eine Textur besitzt, braucht für jedes Material eine Blender Material. Beispielsweise bei der Karotte wurden zwei Materialien verwendet. Jeweils ein Material für das grüne und orange einer Karotte.

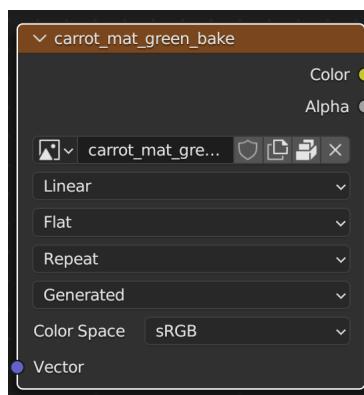


Abb. 6.3: Blender Image Texture

In der Abbildung 6.3 wird eine Image Texture dargestellt, wie sie benötigt wird für das Textur backen. Bevor man die Textur backen kann müssen nur das Modell und die Image Texture Funktionen ausgewählt werden. Anschließend muss in den Rendereinstellungen der Modus Cycle ausgewählt werden.

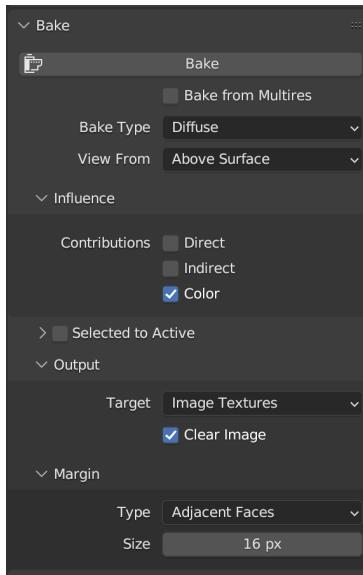


Abb. 6.4: Textur Back Einstellungen

Auf dem Bild 6.4 sind die Rendersettings ausgewählt, die nicht durch äußere Lichtquellen beeinflusst werden. Sollte Bake gedrückt werden dauert es eine Weile bis die Texturen fertig gebacken sind.

Anschließend können die Modelle auf die gleiche Weise wie im Kapitel Gridmap 5.1.1 exportiert werden.

6.2 GDScript Skript für Gemüse

Im Abschnitt 5.1.2 wird erklärt wie man eine exportierte glTF Datei in Godot importiert. Mit den Gemüse wird es auf die gleiche Art getan. Der einzige Unterschied ist bei den Gemüse, dass es nicht exportiert werden muss, sondern es wird als Godot Szene gespeichert.

Zusätzlich muss jedes Modell auf die Position Position die gleiche Position in der Gemüse eigenen Szene transformiert werden. Dadurch kann man später das Mesh und die CollisionShape3D je nach Größe und Art des Gemüses ein- und ausblenden.

List. 6.1: vegetables.gd

```

1  extends Node3D
2
3  @export var harvest_amount = 1
4  @export var croptype = "carrot"
5
6  # Data about vegetable item
7  const carrot = preload("res://item/items/carrot.tres")
8  const carrot_seed = preload("res://item/items/carrot_seed.tres")
9
10 # Objects and Collision Shape of the vegetables
11 @onready var carrot_full = get_node("02_carrot_full")
12 @onready var carrot_full_col =
13     get_node("02_carrot_full/StaticBody3D/CollisionShape3D")
14 @onready var carrot_middle = get_node("02_carrot_middle")
15 @onready var carrot_middle_col =
16     get_node("02_carrot_middle/StaticBody3D/CollisionShape3D")
17 @onready var carrot_seedling = get_node("02_carrot_seed")
18 @onready var carrot_seedling_col =
19     get_node("02_carrot_seed/StaticBody3D/CollisionShape3D")
20
21 var harvestable = false
22
23 # Value of current growth
24 var growth_val = 0.0
25
26 var growth_time_in_seconds = 2.0
27
28 # Seed values for growth stages
29 var growth_seed_to_middle_val = 2.5
30 var growth_max_val = 5.0
31
32 func _ready():
33     # Instantiate invisible "carrot"
34     carrot_full.visible = false
35     carrot_full_col.disabled = true
36     carrot_middle.visible = false
37     carrot_middle_col.disabled = true
38     carrot_seedling.visible = false
39     carrot_seedling_col.disabled = true
40
41 func _process(delta):
42     if growth_val < growth_max_val:
43         growth_val += delta / growth_time_in_seconds
44
45     if croptype == CropType.CARROT:

```

```
43     # Visibility of the crop from seedling -> full_grown
44     carrot_full.visible = growth_val >= growth_max_val
45     carrot_middle.visible = growth_val < growth_max_val &&
        growth_val >= growth_seed_to_middle_val
46     carrot_seedling.visible = growth_val < growth_seed_to_middle_val
        || 0.0
47
48     # Activate collision shape of the crops according to the
        growing state of the vegetable
49     carrot_full_col.disabled = growth_val < growth_max_val
50     carrot_middle_col.disabled = growth_val >= growth_max_val ||
        growth_val < growth_seed_to_middle_val
51     carrot_seedling_col.disabled = growth_val >=
        growth_seed_to_middle_val
52
53     # Set harverstable if full grown
54     harvestable = growth_val >= growth_max_val
```

Der angezeigte Codeabschnitt 6.1 muss in der Szene Gemüse eingebunden werden. Damit ist es möglich, dass ein gepflanztes Gemüse von Samen bis zum erntereifen Gemüse heranwächst.

Kapitel 7

Fazit und Ausblick

Zum Abschluss dieser Arbeit wird ein Fazit im Folgendem erläutert. Hierbei wird auf die Schwierigkeiten beim Entwickeln des Farming Simulators eingegangen. Anschließend wird auf Verbesserungsmöglichkeiten im letzten Kapitel [7.2](#) eingegangen.

7.1 Fazit

Das Ziel des Studienprojektes war es einen Farming Simulator mit einer zeitgemäßen Game Engine zu entwickeln. Erreicht wurde, dass man im Spiel mit einem Raycast den Ort bestimmen kann an dem ein Gemüsesamen eingepflanzt werden soll. Die Technik einen Raycast zu verwenden war sehr neu mit diesem Projekt. Deswegen mussten mehrere Ansätze getestet werden, bis ein brauchbarer Ansatz zum Interagieren mit der 3D Welt gefunden wurde.

Zusätzlich wurde ein Inventarsystem entwickelt, dass Aufschluss darüber gibt welche Samen und Gemüsesorten man besitzt. Sollte im ersten Slot vom Inventar ein Samen sein, kann man diesen in einem ausgewählten Bereich der Welt einpflanzen. Nachdem die Pflanze vollständig ausgewachsen ist, erhält man Samen und das angepflanzte Gemüse. Somit wurde das Inventarsystem mit Farmingsystem erfolgreich integriert. In dem beide Systeme sehr komplex sind, hat es viel Zeit gekostet beide Systeme miteinander zu verbinden.

In der kurzen Zeit war die Entwicklung des Farming Simulators für zwei Personen eine sehr große Herausforderungen. Vor allem die fehlenden Fähigkeiten haben die Lernkurve steil gehalten. Zwei Features konnten nicht verwirklicht werden. Zu einem ein Hauptmenü mit dem man das Spiel starten und beenden kann. Eigene erstellte Hintergrundmusik die für das Aussehen des Spiels geeignet ist.

Für den zeitlichen Aufwand den man benötigt, um alle Features in das Spiel zu integrieren reichen zwei Entwickler nicht aus. Dennoch ist der entstandene Prototyp des Farming Simulators gelungen.

7.2 Ausblick

Es gibt noch sehr viele Bereiche und Möglichkeiten, die verbessert werden können. Eine Möglichkeit besteht darin mehrere Objekte im Spiel mit einem Inventar auszustatten. Damit man gesammelte Items in beispielsweise Kisten oder Truhen ablegen kann.

Auch wäre es möglich den Spieler mit einer Hungeranzeige auszustatten. Damit immer wieder Gemüse im Spiel konsumiert werden muss.

Weiterhin könnte man das Inventarsystem, um ein Shop System erweitern. Dadurch könnte es möglich sein, Items an NPCs zu verkaufen.

In der Zukunft wäre auf jeden Fall vorstellbar, dass das Spiel um ein Hauptmenü erweitert wird mit dem den Stand der Welt speichern und laden kann.

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. CALABRÓ, Andrea. *Logo with wordmark of the Godot Engine game engine*. 2021. Auch verfügbar unter: https://commons.wikimedia.org/wiki/File:Godot_logo.svg. [Zugegriffen 16-01-2024].
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. *Godotengine download macOs*. [o. D.]. Auch verfügbar unter: <https://godotengine.org/download/macos/>. [Zugegriffen 12-01-2024].