

# **Programmentwurf „Computergrafik“**

## **Projektarbeit Belegarbeit**

in Computergrafik im dritten Studienjahr

an der Fakultät für Technik  
im Studiengang Informationstechnik

an der DHBW Ravensburg  
Campus Friedrichshafen

von  
Gary Lude, Julian Yaman, Lars Strölin

23.11.2021

Abgabedatum: 5. Theoriesemester  
Kurs: TIT19, Informatik  
Matrikelnummern: ?, ?, 4556017  
Betreuer: Dr. Jürgen Schneider

# **Sperrvermerk**

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017 in der Fassung vom 25.07.2018.

„Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung vom Dualen Partner vorliegt.“

---

Ort, Datum

---

Unterschrift

# **Selbständigkeitserklärung**

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017 in der Fassung vom 25.07.2018.  
Wir versichern hiermit, dass ich unsere Projektarbeit mit dem Thema

## **Programmentwurf „Computergrafik“**

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

---

Ort, Datum

---

Unterschrift

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>VI</b>
<b>Abbildungsverzeichnis</b>	<b>VII</b>
<b>Tabellenverzeichnis</b>	<b>XI</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Hinführung . . . . .	1
1.2 Zielsetzung . . . . .	1
<b>2 Unreal Engine</b>	<b>2</b>
2.1 Materials . . . . .	3
2.2 Level Editor . . . . .	3
2.3 Blueprints . . . . .	5
2.4 Behavior Trees . . . . .	7
<b>3 Technische Grundlagen</b>	<b>8</b>
<b>4 Konzeptionierung</b>	<b>11</b>
4.1 Vorgaben . . . . .	11
4.2 Struktureller Aufbau . . . . .	12
<b>5 Umsetzung</b>	<b>13</b>
5.1 3D-Landschaft . . . . .	13
5.2 Source-Code . . . . .	14
5.2.1 MainMenu . . . . .	15
5.2.2 AlchemistsHouse . . . . .	15
5.2.3 Coin . . . . .	16
5.2.4 Teapot . . . . .	16
5.2.5 FirstPersonCharacter . . . . .	16
5.2.6 FirstPersonHUD . . . . .	17
5.2.7 Animals . . . . .	17
5.3 User Interface . . . . .	18
<b>6 Fazit</b>	<b>19</b>
<b>A Anhang: Map</b>	<b>21</b>
<b>B Anhang: MainMenu Widget</b>	<b>28</b>

<b>C Anhang: AlchemistsHouse</b>	<b>41</b>
<b>D Anhang: CoinBP</b>	<b>53</b>
<b>E Anhang: TeapotBP</b>	<b>57</b>
<b>F Anhang: FirstPersonCharacter</b>	<b>60</b>
<b>G Anhang: FirstPersonHUD</b>	<b>66</b>
<b>H Anhang: Animals</b>	<b>68</b>
<b>I Anhang: MainMenu Interface</b>	<b>76</b>

## **Abkürzungsverzeichnis**

RPG ..... Rollenspiel (engl. Role-Playing Game)

## Abbildungsverzeichnis

2.1	Der Level Editor der Unreal Engine. Aus Unreal Engine Docs 2021c. . . . .	4
2.2	Detailansicht eines 3D-Objekts. Aus Unreal Engine Docs 2021c. . . . .	5
2.3	Beispiel eines Blueprints zum Öffnen einer Tür. Aus Unreal Engine Docs 2021g. . . . .	6
2.4	Beispiel eines Behavior Trees, damit Tiere auf der Landschaft laufen können . . . . .	7
3.1	Unterschied zwischen statischen Schatten (1) und dynamischen Schatten (2). Aus Unreal Engine Docs 2021b. . . . .	9
4.1	Konzept-Entwurf der Spielewelt . . . . .	12
A.1	Aufnahme der Landschaft 1 . . . . .	21
A.2	Aufnahme der Landschaft 2 . . . . .	21
A.3	Aufnahme der Landschaft 3 . . . . .	22
A.4	Aufnahme der Landschaft 4 . . . . .	22
A.5	Aufnahme der Landschaft 5 . . . . .	23
A.6	Aufnahme der Landschaft 6 . . . . .	23
A.7	Aufnahme der Landschaft 7 . . . . .	24
A.8	Aufnahme der Landschaft 8 . . . . .	24
A.9	Aufnahme der Landschaft 9 . . . . .	25
A.10	Aufnahme der Landschaft 10 . . . . .	25
A.11	Aufnahme der Landschaft 11 . . . . .	26
A.12	Aufnahme der Landschaft 12 . . . . .	26
A.13	Aufnahme der Landschaft 13 . . . . .	27
B.1	MainMenu Widget: Hierarchy . . . . .	28
B.2	MainMenu Widget: Construct Part 1 . . . . .	28
B.3	MainMenu Widget: Construct Part 2 . . . . .	29
B.4	MainMenu Widget: Construct Part 2 - Check Fullscreen And Vsync . . . . .	29
B.5	MainMenu Widget: Construct Part 3 . . . . .	29
B.6	MainMenu Widget: MainMenuBox Part 1 . . . . .	30
B.7	MainMenu Widget: MainMenuBox Part 2 . . . . .	30
B.8	MainMenu Widget: MainMenuBox Part 3 . . . . .	30
B.9	MainMenu Widget: Save And Load MainMenu Part 1 . . . . .	31
B.10	MainMenu Widget: Save And Load MainMenu Part 2 - Save Menu To Slot Part 1 . . . . .	31
B.11	MainMenu Widget: Save And Load MainMenu Part 3 - Save Menu To Slot Part 2 . . . . .	32
B.12	MainMenu Widget: Save And Load MainMenu Part 4 - Save Menu To Slot Part 3 . . . . .	33

B.13 MainMenu Widget: Save And Load MainMenu Part 5 - Create Save Me-	
nu Slot Part 1 . . . . .	34
B.14 MainMenu Widget: Save And Load MainMenu Part 6 - Create Save Me-	
nu Slot Part 2 . . . . .	35
B.15 MainMenu Widget: Save And Load MainMenu Part 7 - Create Save Me-	
nu Slot Part 3 . . . . .	36
B.16 MainMenu Widget: Save And Load MainMenu Part 8 - Load MainMenu	
From Slot Part 1 . . . . .	37
B.17 MainMenu Widget: Save And Load MainMenu Part 9 - Load MainMenu	
From Slot Part 2 . . . . .	37
B.18 MainMenu Widget: Open GitHub Repo . . . . .	38
B.19 MainMenu Widget: Options . . . . .	38
B.20 MainMenu Widget: Options Example . . . . .	39
B.21 SaveMenuBP . . . . .	40
C.1 AlchemistsTorchBP: Viewport . . . . .	41
C.2 AlchemistsTorchBP: Components . . . . .	42
C.3 AlchemistsHouseBP: Viewport . . . . .	43
C.4 AlchemistsHouseBP: Components And Variables . . . . .	44
C.5 AlchemistsHouseBP: Begin Play Part 1 . . . . .	44
C.6 AlchemistsHouseBP: Begin Play Part 2 . . . . .	45
C.7 AlchemistsHouseBP: Open/Close Entry Door Part 1 . . . . .	45
C.8 AlchemistsHouseBP: Open/Close Entry Door Part 2 . . . . .	45
C.9 AlchemistsHouseBP: Open/Close Entry Door Part 2 - EntryDoor1Timeline	46
C.10 AlchemistsHouseBP: Open/Close Entry Door Part 3 . . . . .	46
C.11 AlchemistsHouseBP: Open/Close Bedroom Door Part 1 . . . . .	47
C.12 AlchemistsHouseBP: Open/Close Bedroom Door Part 2 . . . . .	47
C.13 AlchemistsHouseBP: Open/Close Bedroom Door Part 3 . . . . .	48
C.14 AlchemistsHouseBP: Begin/End Overlap Ambry1 CollisionBox Part 1 .	48
C.15 AlchemistsHouseBP: Begin/End Overlap Ambry1 CollisionBox Part 2 .	49
C.16 AlchemistsHouseBP: Open/Close Ambry1 Doors Part 1 . . . . .	49
C.17 AlchemistsHouseBP: Open/Close Ambry1 Doors Part 2 . . . . .	50
C.18 AlchemistsHouseBP: Begin/End Overlap Chest1 CollisionBox Part 1 .	50
C.19 AlchemistsHouseBP: Begin/End Overlap Chest1 CollisionBox Part 2 .	51
C.20 AlchemistsHouseBP: Open/Close Chest1 Part 1 . . . . .	51
C.21 AlchemistsHouseBP: Open/Close Chest1 Part 2 . . . . .	52
C.22 AlchemistsHouseBP: Manage Lobby Lights . . . . .	52
C.23 AlchemistsHouseBP: Manage Alchemists Room Lights . . . . .	52
D.1 CoinBP: Viewport . . . . .	53

D.2	CoinBP: Components And Variables . . . . .	54
D.3	CoinBP: Begin Play Part 1 . . . . .	55
D.4	CoinBP: Begin Play Part 2 . . . . .	55
D.5	CoinBP: Tick . . . . .	55
D.6	CoinBP: Circle Coin Animation Part 1 . . . . .	55
D.7	CoinBP: Circle Coin Animation Part 2 . . . . .	56
D.8	CoinBP: Begin Overlap Collision Sphere Part 1 . . . . .	56
D.9	CoinBP: Begin Overlap Collision Sphere Part 2 . . . . .	56
E.1	TeapotBP: Viewport . . . . .	57
E.2	TeapotBP: Components And Variables . . . . .	58
E.3	TeapotBP: Begin Play . . . . .	59
E.4	TeapotBP: Start Animation Part 1 . . . . .	59
E.5	TeapotBP: Start Animation Part 2 . . . . .	59
F.1	FirstPersonCharacter: Viewport . . . . .	60
F.2	FirstPersonCharacter: Components And Variables . . . . .	61
F.3	FirstPersonCharacter: Begin Play . . . . .	62
F.4	FirstPersonCharacter: Tick Part 1 . . . . .	62
F.5	FirstPersonCharacter: Tick Part 2 - Check Z And Change Environment Color . . . . .	62
F.6	FirstPersonCharacter: Movement Input . . . . .	63
F.7	FirstPersonCharacter: Mouse Input . . . . .	64
F.8	FirstPersonCharacter: Jump . . . . .	65
F.9	FirstPersonCharacter: Return To MainMenu . . . . .	65
F.10	FirstPersonCharacter: Interact With Objects . . . . .	65
G.1	FirstPersonHUD Widget: Designer . . . . .	66
G.2	FirstPersonHUD Widget: Hierarchy . . . . .	66
G.3	FirstPersonHUD Widget: Functions And Variables . . . . .	67
G.4	FirstPersonHUD Widget: Construct . . . . .	67
G.5	FirstPersonHUD Widget: Change Interact Text Visibility . . . . .	68
H.1	Animals: Chicken Folder Animations . . . . .	68
H.2	Animals: Chicken Folder AnimBP . . . . .	69
H.3	Animals: Chicken Folder Skeletal . . . . .	69
H.4	Animals: AnimalStruct . . . . .	69
H.5	Animals: DataTable . . . . .	70
H.6	Animals: AnimalsMasterBP - Viewport . . . . .	70
H.7	Animals: AnimalsMasterBP - Components And Variables . . . . .	71
H.8	Animals: AnimalsMasterBP - Begin Play Part 1 . . . . .	72
H.9	Animals: AnimalsMasterBP - Begin Play Part 2 . . . . .	72

---

H.10 Animals: AnimalsMasterBP - Get Animal Variables With DataTable . . . . .	72
H.11 Animals: AnimalsMasterBP - Setup Animal With Variables . . . . .	72
H.12 Animals: AnimalsMasterBP - Choose And Set Skin Variation . . . . .	72
H.13 Animals: AnimalsMasterAiController - Components And Variables . . . . .	73
H.14 Animals: AnimalsMasterAiController - Begin Play . . . . .	74
H.15 Animals: AnimalsMasterAiController - On Target Perception Updated Ai Perception . . . . .	74
H.16 Animals: Behavior Tree Services - Is Player In Radius . . . . .	74
H.17 Animals: Behavior Tree Tasks - Find Random Location In Radius . . . . .	75
H.18 Animals: Behavior Tree Blackboard . . . . .	75
H.19 Animals: Behavior Tree Overview . . . . .	76
I.1 MainMenu Interface: Startup . . . . .	76
I.2 MainMenu Interface: Options - Video . . . . .	77
I.3 MainMenu Interface: Credits . . . . .	77

## **Tabellenverzeichnis**

# 1 Einleitung

Die folgende Dokumentation, welche von einer Projektgruppe von drei Personen ausgearbeitet wurde, befasst sich mit dem Programmentwurf einer animierten 3D-Szene, welche im Zuge des Informatik-Moduls „Computergrafik“ entwickelt und im Folgenden beschrieben wurde.

## 1.1 Hinführung

3D-Animationen finden wir in unserem heutigen Alltag fast überall. Es gibt sie in Computerspielen, Werbespots und auch in vielen Filmen. Somit ist die Animation von 3D-Objekten und die grafische Visualisierung von Bildinhalten ein essenzieller Bestandteil der Computergrafik und somit der Informatik. Im Rahmen der Vorlesung soll eine 3D-Animation erstellt werden, um das theoretisch erlangte Wissen praktisch anzuwenden.

## 1.2 Zielsetzung

Ziel der Projektarbeit ist es eine animierte 3D-Szene zu erstellen, in welcher die Objekte der Szene über Hilfstoools oder Spiele-Engines importiert werden können und in welcher sich der Betrachter innerhalb der Szene frei bewegen kann. Für diese Aufgabe wurde die Unreal Engine als Basis des Programmentwurfs gewählt, da sie für Einsteiger verhältnismäßig einfach zu bedienen ist und viele Möglichkeiten bietet. Außerdem deckt diese durch die Objekte einiger Packages Teilgebiete der Anforderungen wie Licht- und Schattenberechnung bereits ab.

Als Szene soll eine kleine Landschaft mit einem Gebäude, Bäumen, Tieren und einem See kreiert werden, in welcher der Benutzer eine Spielfigur steuert, die laufen und springen kann, sowie bestimmte Elemente wie Türen öffnen und schließen können soll.

## 2 Unreal Engine

Die *Unreal Engine* ist eine sogenannte Spiel-Engine und wird vom Unternehmen „Epic Games“ entwickelt. Spiel-Engines dienen der Entwicklung von Computerspielen und helfen bei der visuellen Darstellung der Inhalte, beispielsweise einer Spielewelt. Auch werden durch Spiel-Engines bereits Entwicklungsumgebungen mitgeliefert, die es dem Entwickler leichter machen, Logiken eines Spieles oder bestimmter Charaktere mit Code zu integrieren. Die Unreal Engine ist hierbei eine beliebte Option für Spiele-Entwickler sowie für ganze Entwicklerstudios (vgl. Unreal Engine 2021).

Eine Engine erleichtert die Implementierung von Audio, Grafiken und Logiken. In der modernen Computerspielentwicklung wird durch den Einsatz von Spiel-Engines die Entwicklungszeit erheblich gekürzt (vgl. Funes 2020). Der Fokus verlagert sich mehr auf die Bereiche der Kreativität und Funktionalitäten innerhalb des Spiels anstatt auf die technische Implementierung grundlegender Komponenten. Somit wird beispielsweise die Simulation von Lichtquellen und Reflexion stark vereinfacht. Es wird der Engine nur mitgeteilt, dass von einer bestimmten Koordinate aus das Licht auf ein Objekt gerichtet wird oder es sich in der gesamten Umgebung verteilen soll. Die eigentliche Berechnung erfolgt im Hintergrund und muss nicht mehr aktiv implementiert werden. Engines sind somit als eine Art „Framework“ zu verstehen, die bereits die technischen Grundlagen der Computergrafik abdeckt und implementiert, damit auf ihnen ein Spiel einfacher programmiert werden kann (vgl. Valencia-García u. a. 2016, S. 146).

Im Markt der Spiele-Engines gibt es zahlreiche Anbieter, die sich für die Entwicklung eines 2D- beziehungsweise 3D-Spieles eignen und ihre eigenen Vorteile liefern. Für diesen Programmentwurf fiel die Wahl auf die Unreal Engine aufgrund der bereits vorhandenen Erfahrung, die mit anderweitigen Projekten gesammelt wurde. Die Unreal Engine eignet sich nicht nur für die Darstellung von Szenen in Videospielen, sondern auch für Filme und Serien. Beispielsweise wurde die Spiele-Entwicklungsumgebung in Szenen der bekannten Science-Fiction-Serie „The Mandalorian“ genutzt (vgl. Grubb 2019). Neben umfassenden Funktionalitäten für die Gestaltung hochdetaillierter 3D-Landschaften und Szenerien, bietet die Engine auch zahlreiche Tools zur Interaktion und Programmierung dieser Szenen an. Einige davon werden auch im Zusammenhang mit diesem Projekt benutzt und in den folgenden Abschnitten näher betrachtet. Dabei werden die erwähnten Funktionen nur allgemein beschrieben. Eine detaillierte Beschreibung über die Anwendung dieser im Rahmen des Projekts ist in der Umsetzung im Kapitel 5 zu finden.

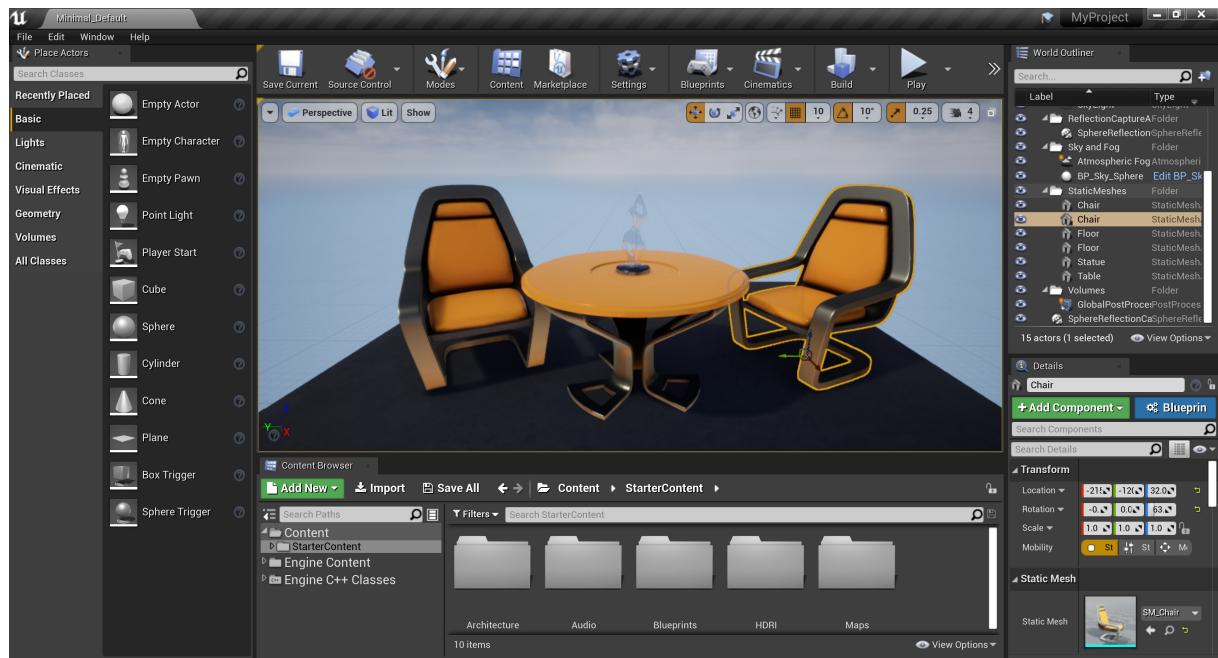
## 2.1 Materials

Eine wichtige Funktionalität für die Gestaltung der 3D-Objekte sind „Materials“. Ein „Material“ bearbeitet die visuelle Darstellung der 3D-Objekte. Sie können bestimmen, wie stark ein 3D-Objekt reflektiert und berechnet die dabei entstehenden Interaktionen mit dem Licht (vgl. Unreal Engine Docs 2021d). Die dahinter liegenden Berechnungen werden von der Engine selbst übernommen. Auch wenn Materials eher eine kleine Funktionalität der Engine darstellt, zeigt es trotzdem, dass grundlegende Berechnungen nicht mehr vom Entwickler berücksichtigt werden müssen. Umso verstärkt sich der Fokus der Entwickler und Spiel-Designer auf die Bearbeitung der Texturen und ihre visuelle Darstellung. Dies bedeutet jedoch nicht, dass der Entwickler gar keinen Einfluss mehr auf die Berechnungen hat. Sie lassen sich weiterhin durch eingegebene Werte und Texturen bearbeiten sowie durch das Hinzufügen mathematischer Ausdrücke, um bestimmte spezielle Effekte zu erzielen, die der Entwickler sich wünscht.

Bei der Gestaltung der Materials kann ein Entwickler alle Möglichkeiten ausschöpfen. Der Wechsel des Fokus auf die Kreativität anstatt auf die technische Basisimplementierung von Texturen und unterschiedlicher mathematischer Berechnungen für die Darstellung dieser, ermöglicht auch kleineren Entwicklerstudios und Solo-Spieleentwickler hochqualitative 3D-Szenen und Spiele zu produzieren. Dies kann als ein signifikanter Vorteil heutiger Spiele-Engines beschrieben werden und hebt die Standards, die an heutige Spiele und 3D-Darstellungen gestellt werden. Ermöglicht wird dies auch durch die heute zur Verfügung stehenden Rechenleistung, die auch komplexere und realistischere Darstellungen ermöglicht. Natürlich kann ein Spiel nicht allein aus 3D-Objekten bestehen. Diese Objekte müssen auch in einer entsprechenden 3D-Szene platziert und integriert werden, um eine visuelle Darstellung einer Szenerie komplett zu machen.

## 2.2 Level Editor

Für die Gestaltung der 3D-Szenen für den Programmentwurf, bietet die Unreal Engine den „Level Editor“ an. Der Editor wird als Grundfunktionalität der Unreal Engine beschrieben (vgl. Unreal Engine Docs 2021c). Szenen, die in der Unreal Engine im Level Editor erstellt werden, werden als Level beschrieben.



**Abb. 2.1:** Der Level Editor der Unreal Engine. Aus Unreal Engine Docs 2021c.

Im Editor können dem Level 3D-Objekte hinzugefügt und entfernt werden. Zudem lassen sie sich transformieren und ihre Eigenschaften können verändert werden. Der Level Editor bietet dafür verschiedene Bearbeitungs-Modi an, welche jeweils ihre eigene Aufgabe erfüllen.

Beim Wechseln der Modi zeigt die Unreal Engine für den ausgewählten Modus ein eigenes Interface an, die dem jeweiligen Aufgabenbereich zugeschnitten sind. Beispielsweise bietet der sogenannten Landschafts-Modus Werkzeuge an, um die Landschaft in einer 3D-Szene zu bearbeiten. Der Select-Modus wiederum erlaubt die Auswahl und Bearbeitung einzelner 3D-Objekte innerhalb einer Szene. Bei der Auswahl eines 3D-Objektes werden in der Detailansicht die Eigenschaften des Objektes dargestellt.



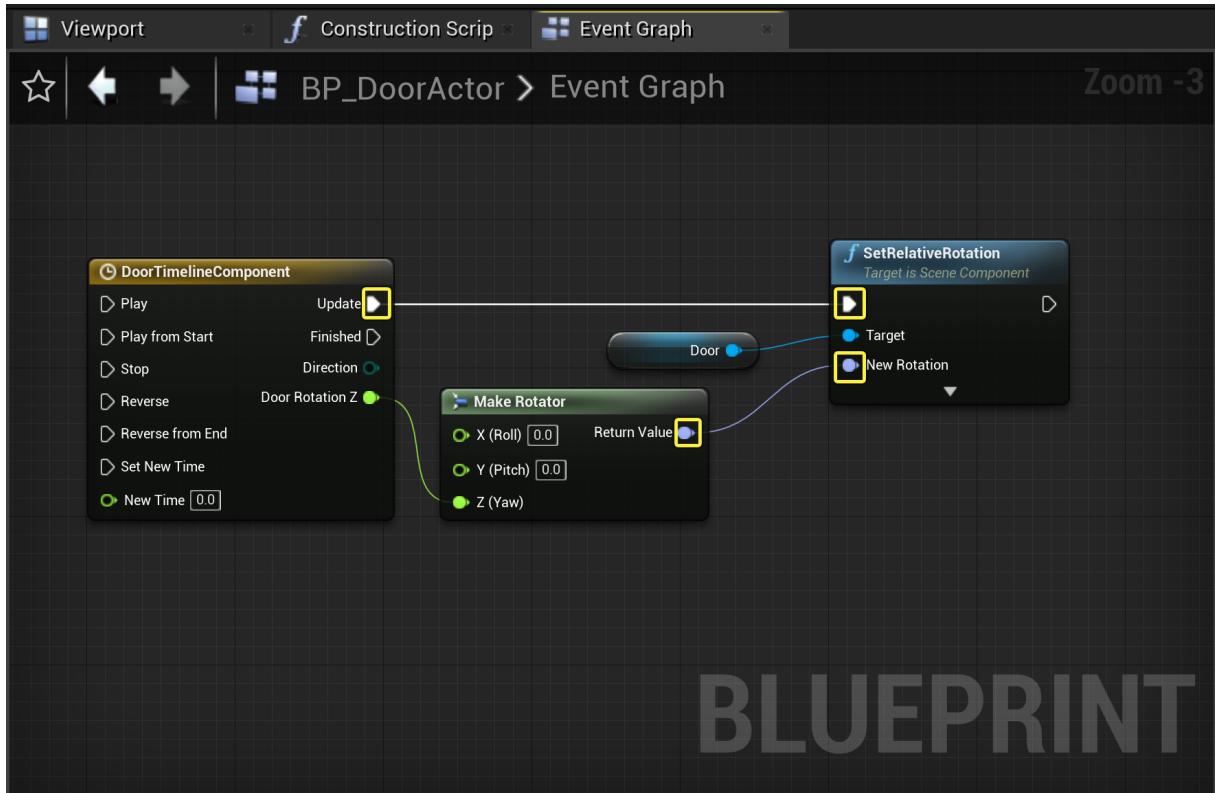
Abb. 2.2: Detailansicht eines 3D-Objekts. Aus Unreal Engine Docs 2021c.

Neben den zwei erwähnten Modi bietet die Unreal Engine weitere Bearbeitungsmöglichkeiten für die Gestaltung einer 3D-Szene an. Die genannten Modi werden bei der Realisierung des Programmentwurfs verwendet. Bislang wurde beschrieben, wie sich eine Szene gestalten lässt. Damit sich jedoch Logiken und Möglichkeiten für Interaktionen in einem Spiel realisieren lassen, bietet die Unreal Engine eine weitere wichtige Funktionalität an, die sogenannten Blueprints.

## 2.3 Blueprints

Blueprints sind als „visuelle Skriptsprache“ zu verstehen und beinhalten C++-Code (vgl. Pv 2021, S. 21f.). Ähnlich wie häufig verwendete Programmiersprachen nutzen Blueprints den objektorientierten Ansatz und verfügen über Klassen und Objekten (vgl. Unreal Engine Docs 2021f). Vorteil der Blueprints ist, dass sie Code und Funktionsab-

läufe wiederverwendbar macht und die Notwendigkeit einer Neuprogrammierung wesentlicher Grundlagen für die Entwicklung von Spielen eliminiert. Die Flexibilität der Blueprints zeigt sich unter anderem dadurch, dass ihr hinterlegter Code nicht fix abgeschlossen ist, sondern beliebig erweiterbar ist, um die Anforderungen der Entwickler gerecht zu werden.



**Abb. 2.3:** Beispiel eines Blueprints zum Öffnen einer Tür. Aus Unreal Engine Docs 2021g.

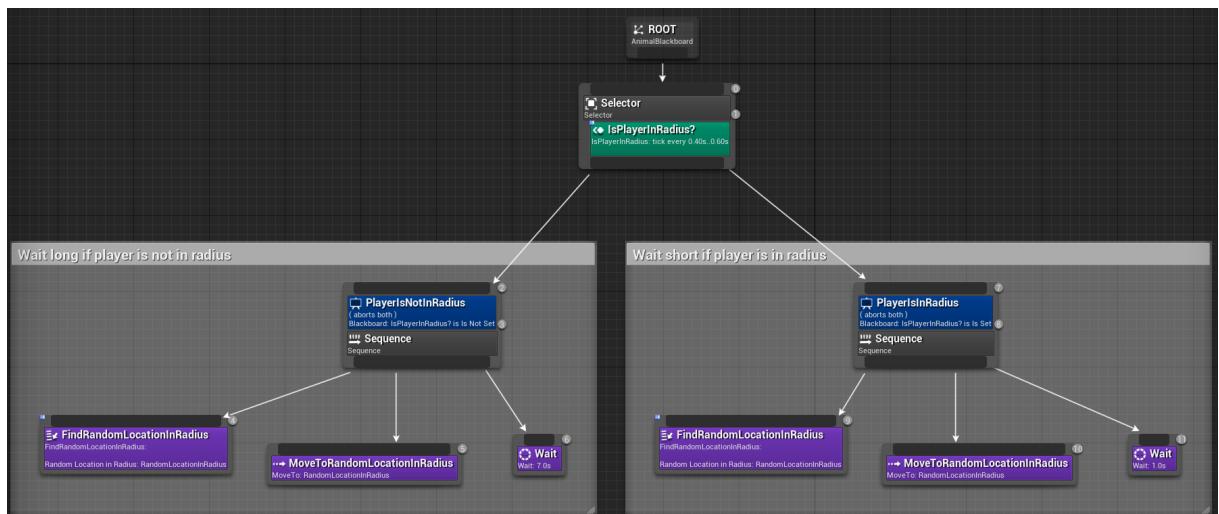
Dadurch, dass sich hinter Blueprints Code verbirgt, funktionieren diese auch nach einer definierten Programmierlogik. Neben Abfragen und Vergleiche ist es auch möglich, Blueprints Funktionen, Variablen und Objekte zuzuweisen. Ereignisse, wie beispielsweise das Drücken eines Buttons im User-Interface, lassen sich ebenfalls verarbeiten. Einzelne Ausführungsblöcke werden als Node beschrieben. Nodes lassen sich miteinander verbinden, wodurch Ausführungsketten gebildet werden können. In der visuellen Darstellung von Blueprints, auf welche im Kapitel 5 verwiesen werden, sind Abzweigungen möglich, falls mehrere Ergebnisse einer Node abgefragt und entsprechend verarbeitet werden müssen (bspw. Tür schließen / Tür öffnen). Visuell werden die Blueprints in der sogenannten „Graph View“ dargestellt. In dieser lassen sich die bereits erwähnten Nodes miteinander verbinden.

Blueprints sind hierbei als Oberbegriff für jede Art eines visuellen Skripts zu verstehen.

Sie können für 3D-Objekte verwendet werden und auch für die Interaktion mit diesen. Ein weiterer Einsatzbereich ist auch die Verwendung sogenannter „Widget Blueprints“. Diese werden bevorzugt für die Interaktion mit einer Benutzeroberfläche, oder auch User Interface genannt, verwendet. Detailliert wird die Verwendung von Blueprints und Widget Blueprints in der Umsetzung beschrieben.

## 2.4 Behavior Trees

Blueprints werden somit für die Implementierung von Interaktionen und Logiken verwendet. Für die Implementierung künstlicher Intelligenzen verwendet die Unreal Engine „Behavior Trees“ (vgl. Unreal Engine Docs 2021a). Hier ist jedoch wichtig zu wissen, in welchem Kontext diese verwendet werden. Künstliche Intelligenzen in einem Spiel können von Flugbahnen eines Vogels am Himmel bis hin zu Interaktionen von Figuren mit dem Spieler reichen. Erwähnt werden Behavior Trees somit aufgrund ihrer Relevanz in der Umsetzung des Programmentwurfs, damit die im Spiel enthaltenen Tieren sich auf der Landschaft eigenständig bewegen können. Die Möglichkeiten, die sie jedoch bieten, reichen weit über den Einsatz in diesem Projekt hinaus. Innerhalb eines Behavior Trees gibt es die sogenannten „Blackboards“. Blackboards werden als das Gehirn der Behavior Trees beschrieben (vgl. Unreal Engine Docs 2021a). In diesen wird die Logik der Behavior Trees festgelegt und wie bei bestimmten Situationen ein Behavior Tree entscheiden soll, beispielsweise wenn ein Wert `true` oder `false` ist.



**Abb. 2.4:** Beispiel eines Behavior Trees, damit Tiere auf der Landschaft laufen können

In diesem Kapitel wurden die wesentlichen Eigenschaften der Unreal Engine beschrieben und die Funktionalitäten, welche für die Umsetzung des Projekts relevant sind. Im folgenden Kapitel werden weitere technische Grundlagen erwähnt und ein Zusammenhang zur Vorlesung geschaffen.

### 3 Technische Grundlagen

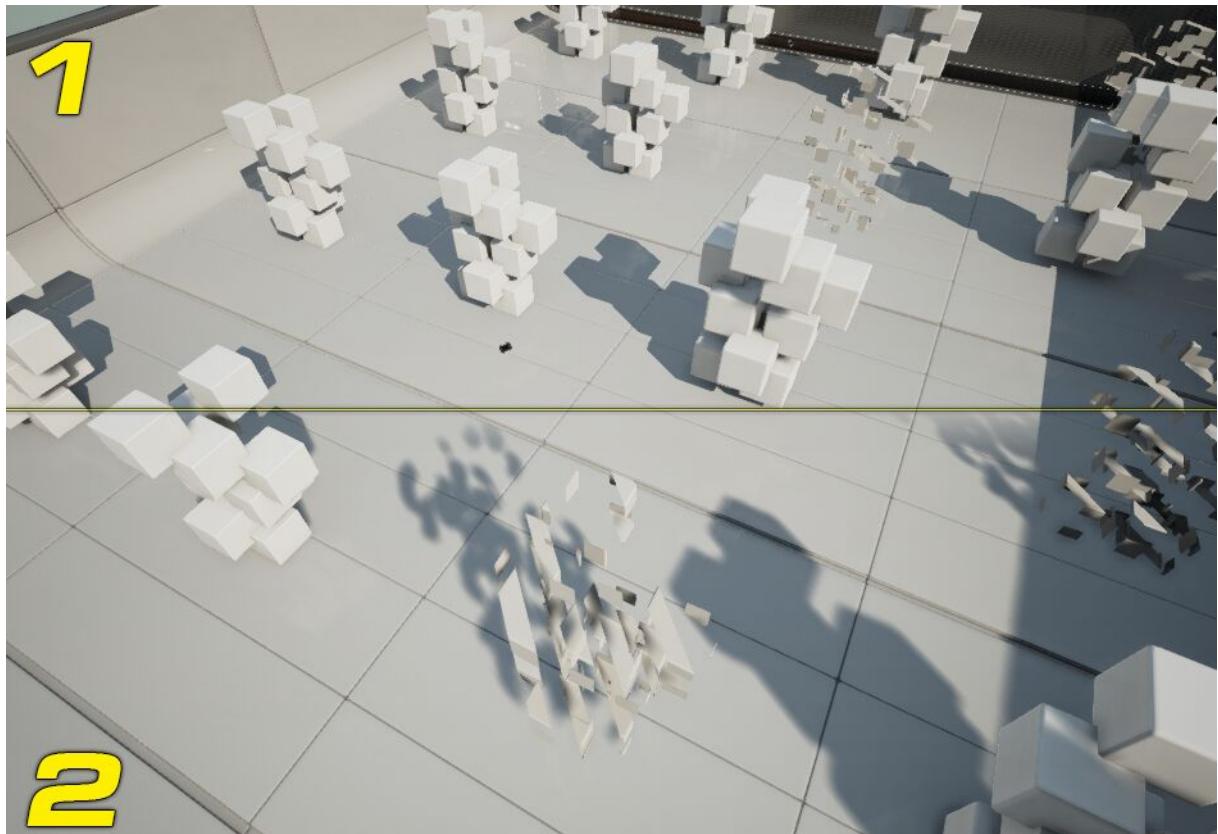
Innerhalb der Unreal Engine können verschiedene Funktionalitäten zur Darstellung von 3D-Szenen verwendet werden, welche auch bereits erwähnt wurden. In diesem Kapitel werden die wesentlichen 3D-Techniken und ihre entsprechende Verbindung zur Vorlesung dargestellt. Besonders wird hierbei der Fokus auf die Verarbeitung von Licht und Schatten gelegt, da dies einen wesentlichen Bereich in der Bearbeitung des Projekts dargestellt hat.

In der Beschreibung der Unreal Engine wurde beschrieben, dass Materials zur äußerlichen visuellen Darstellung von 3D-Objekten benutzt werden. Materials haben die Eigenschaft, je nach gewählter Textur, Licht zu reflektieren und mit diesem zu wechselwirken (vgl. Unreal Engine Docs 2021d). Bei einer Interaktion mit Licht können 3D-Objekte mit dem entsprechenden Material dieses reflektieren. In der Vorlesung wurde dieser Effekt bestätigt und auch die Abhängigkeit von Materialeigenschaften beschrieben (vgl. Kapitel 8 - Reflexion durch Lichtquellen). Hierbei wurde die Reflexion abhängig gemacht von der Farbe und dem Glanz der Oberfläche, der Struktur eines Objekts und der Farbe des einfallenden Lichts.

Was die Materials in der Unreal Engine betrifft, lassen sich die zwei ersten Eigenschaften komplett bearbeiten und konfigurieren, um die Reflexion zu realisieren. Lichtquellen können in der Unreal Engine auch hinzugefügt werden und ihre Eigenschaften in Bezug auf die Farbe des aus ihnen ausgestrahlten Lichts anpassen. Je nach Oberflächenstruktur, können auch diffuse Reflexionen dargestellt werden, wie in Kapitel 8 - Diffuse Reflexionen beschrieben wurde. Besonders wurde in der Vorlesung erwähnt, dass Abstand und Abstrahlwinkel der Lichtquelle ein entscheidender Faktor in der Reflexion darstellen. Wenn jedoch ein Objekt eine matte Oberfläche aufweist, kann dies auch als mattes „Material“ beschreiben werden, dann erscheint die Reflexion als diffus. Die Lichtquellen lassen sich auch entsprechend konfigurieren, um den visuellen Effekt weiter zu verstärken oder abzuschwächen.

Im Gegensatz zur grundlegenden technischen Implementierung und Berechnung von Licht und Reflexionen ist es in der Unreal Engine für den Entwickler nicht erforderlich, für jedes Pixel die Beleuchtungsgleichungen auszuwerten. Dies wird bereits durch die Engine im Hintergrund übernommen. Vielmehr kann der Entwickler, wie bereits im Kapitel 2 beschrieben wurde, einstellen, wie stark oder schwach die Reflexion sein soll. Auch Schattierungen stellen in der Unreal Engine keine technische Hürde dar aufgrund dessen, dass sie, ähnlich wie die meisten anderen Basis-Funktionalitäten moderner 3D-Anwendungen, bereits integriert sind (vgl. Unreal Engine Docs 2021e). Dynami-

sche Schatten werden mithilfe der sogenannten „Cascading Shadow Map“-Methode realisiert (vgl. Unreal Engine Docs 2021b). Diese Methode ermöglicht, „abstandsisierte Schattenkaskaden“ zu generieren, welche eine immer geringere Auflösung haben, je weiter sich der Spieler von einem Objekt mit einem dynamischen Schatten befindet. Falls der Abstand zu groß sein sollte, werden anstelle dynamischer Schatten statische verwendet.



**Abb. 3.1:** Unterschied zwischen statischen Schatten (1) und dynamischen Schatten (2). Aus Unreal Engine Docs 2021b.

In der Vorlesung wurden die Grundlagen des „Schattenwerfens“ erläutert. Wie im Kapitel 8.1 und 8.4 des Vorlesungsskripts beschrieben wurde, ist zwar ein Schatten nur das Fehlen einer Lichtquelle, jedoch ist Schatten ein entscheidender Faktor um Objekte dreidimensional darzustellen auf einem zweidimensionalen Bildschirm. Die Implementierung von Schatten mag durch die Unreal Engine zwar einfach erscheinen, dennoch ist sie ein enormer Faktor für die heutige Darstellung von 3D-Inhalten in beispielsweise Filmen und Videospielen.

Bislang wurde in dieser Dokumentation die Technik hinter dem Programmierwurf betrachtet. Es wurde beschrieben, welche Software für die Realisierung verwendet wird und welche Funktionalitäten dabei von großer Wichtigkeit sind. In diesem Kapitel wur-

de vor allem betrachtet, welche Inhalte der Vorlesung in der Bearbeitung des Projekts wieder gefunden wurden und welche Möglichkeiten diese nicht nur für diesen Programmentwurf, sondern für 3D-Anwendungen generell bieten. Dabei haben wir den Fokus auf den Bereich Licht und Schatten gelegt. Es wurde festgestellt, dass eine grundlegende technische Implementierung, aufgrund der Tools, die zur Verfügung stehen, nicht mehr notwendig ist. Jedoch wird umso mehr der Fokus auf die Konfiguration der visuellen Gestaltung von 3D-Objekten und 3D-Szenen gelegt, sowie auf die Kreativität als solche.

Im weiteren Verlauf der Dokumentation wird ein Konzept zur Gestaltung einer solchen 3D-Landschaft beschrieben und damit einhergehend die praktische Anwendung der in diesem und vorherigen Kapitel beschriebenen theoretischen Betrachtung der Funktionalitäten innerhalb der Unreal Engine.

## 4 Konzeptionierung

Für ein Projekt diesen Ausmaßes, bei welchem zudem noch gewisse Kriterien als Vorgabe gegeben sind, muss zu Beginn eine grundlegende Struktur für die Vorgehensweise festgelegt werden. Als Erstes ist es notwendig sich alle Muss-Kriterien anzuschauen und zu überlegen, wie dies erreicht werden kann. Im nächsten Schritt kann überlegt werden, wie die Szene des Programmentwurfs ungefähr aussehen soll, bevor diese aufwändig erstellt wird.

### 4.1 Vorgaben

Die folgenden Punkte müssen laut Aufgabenstellung in dem Programmentwurf enthalten sein:

1. 3D-Ansichten und 3D-Objekte
2. Betrachter kann sich durch die Szene frei bewegen
3. Animieren von einzelnen Objekten innerhalb der Szene
4. Beleuchtung und Schattierung

Für die einzelnen 3D-Objekte, wie auch die 3D-Ansicht innerhalb der Anwendung kann, wie bereits erwähnt, auf die Unreal Engine zurückgegriffen werden. Über kostenlose Packages, welche Entwicklern der Unreal Engine 4 zur Verfügung stehen, können alle gewünschten Elemente grafisch gesehen bereits importiert werden. Wie genau dies umgesetzt wurde, wird in einem späteren Kapitel noch genauer erläutert.

Weiterhin soll der Betrachter sich in der Grafikumgebung frei bewegen können. Es besteht die Möglichkeit dies in Form einer fliegenden Kamera oder einer Spielfigur, welche in alle Richtungen laufen und nach oben springen kann, umzusetzen. In diesem Fall wurde die Methode einer klassischen Spielfigur gewählt, welche sich frei durch die Welt bewegen kann.

Der nächste Punkt besteht aus dem Animieren von einzelnen Objekten, welche sich innerhalb der Spielwelt befinden. Hier ist geplant, dass die Spielfigur einige Interaktionen mit diesen Objekten hat und den Zustand des Objektes selbst steuern kann. Beispielsweise soll der Benutzer Truhen und Türen öffnen oder schließen können, oder ein Licht anschalten können. Diese Funktion wird dann mit Blueprints, welche im Hintergrund C++ als Quellcode verwendet, innerhalb der Unreal Engine gelöst.

Der letzte Punkt ist die Beleuchtung und die Schattierung, welche in diesem Fall auch

wieder von den Objekten des Packages übernommen werden, da diese unter anderem Licht enthalten und die physikalische Wirkung auf umstehende Objekte bereits programmiert ist.

## 4.2 Struktureller Aufbau

Für die 3D-Szene welche entwickelt werden soll, ist eine Art kleine Spielewelt eines Rollenspiels, kurz RPG, angedacht. Diese soll einen grafischen Stil einer Fantasiewelt bekommen, in der es ein Haus von einem Alchemisten gibt, welches sich in einer Gras/Wald-Landschaft befindet. Diese ist wiederum von Bergen umschlossen, die auch gleichzeitig die betretbaren Grenzen der Szene darstellen. Zudem soll es innerhalb der Welt einen See mit Steg geben. Innerhalb des Hauses sollen einige detaillierte Elemente wie Lampen, Stühle, Schränke, Truhen, Türen, Betten und Bücher integriert werden.

Das Bild 4.1 zeigt einen ersten grafischen Konzept-Entwurf der Spielwelt in der Vogelperspektive.



**Abb. 4.1:** Konzept-Entwurf der Spielewelt

## 5 Umsetzung

Bei der Umsetzung dieses Projektes konnte die bereits grundlegende Erfahrung im Umgang mit der Unreal Engine genutzt werden. Während der Entwicklung wurden die Gedanken der Konzeptionierungsphase eingearbeitet und somit ein planbarer Fortschritt erzielt, welcher uns eine zeitliche Sicherheit ermöglichte. Dabei gehörte die Versionskontrolle über GitHub zum Standard-Werkzeug um Backups zu erstellen, sowie einzelne Entwicklungsstände zu fusionieren. Im Weiteren wird zunächst auf den Aufbau der 3D-Landschaft eingegangen, danach wird der zugehörige Source-Code erläutert und das User Interface anhand von Screenshots beschrieben.

### 5.1 3D-Landschaft

Bei diesem Programmentwurf wollten wir eine idyllische Umgebung schaffen, welche an eine gemütliche Hütte im Wald erinnern soll, in welcher ein Alchemist sich eingerichtet hatte. Die 3D-Landschaft unterteilt sich in drei umfangreiche Teile, dem Haus des Alchemisten, einen See sowie das restliche Waldgebiet.

#### Haus des Alchemisten

Das Haus des Alchemisten<sup>1</sup> bildet die zentrale Szene unserer 3D-Landschaft. Dieses komplexe 3D-Objekt setzt sich aus vielen weiteren 3D-Objekten zusammen und wurde innerhalb eines Blueprints konstruiert. Der Hintergrund dessen ist, dass man den Blueprint als Ganzes im Viewport verschieben kann und nicht jedes Objekt einzeln. Des Weiteren kann direkt Source-Code auf einzelne Objekte angewendet werden, dies vereinfacht beispielsweise die Animation oder generelle Veränderung an den Komponenten. Der Spieler öffnet und schließt automatisch Türen und hat die Möglichkeit eine Kiste im Schlafzimmer mit der Taste „E“ zu öffnen beziehungsweise wieder zu schließen. Die Lichter im Hauptraum des Hauses werden beim Verlassen des Schlafzimmers innerhalb von sieben Sekunden auf die maximale Helligkeit gestellt. Neben dem Essbereich befindet sich ein weiterer Raum, dessen Lichter sich ebenfalls beim Betreten des Raumes langsam aktivieren. Direkt rechts in diesem Raum befindet sich ein Schrank mit Türen, welche vom Spieler mit der Taste „E“ geöffnet sowie geschlossen werden können.

#### See

Der See<sup>2</sup> hinter dem Haus wurde auf Basis eines Static Mesh Cubes erstellt, welcher entsprechend in seiner X Y Z Skalierung modelliert wurde, so dass er einen gewissen

<sup>1</sup>Siehe Abbildungen A.5, A.10, A.11, A.12, A.13

<sup>2</sup>Siehe Abbildungen A.7, A.8

Bereich mit Wasser ausfüllt. Die Kollision des Cubes wurde entfernt und eine Veränderung der Umgebungsfarbe ab einer Tiefe von  $Z = 35$  hinzugefügt. Dadurch entsteht der Eindruck als wäre der Spieler Unterwasser. Desto tiefer der See an einer Stelle ist, umso dunkler wird das Wasser an der Oberfläche. Dies geschieht aufgrund von Konfigurationen des verwendeten Materials, wobei dort der Abstand zum Landscape berechnet und anschließend zur Verdunklung der Wassertexturen verwendet wird.

### **Waldgebiet**

Das Waldgebiet<sup>3</sup> sowie die Gebirge außen herum ermöglichten es uns weitere Details zu implementieren. Darunter verschiedene 3D-Objekte für Gräser, Pflanzen, Pilze, Bäume und vieles mehr. Diese sollen den Grundgedanken unserer idyllischen Landschaft weiter unterstreichen und die Verbindung zu einem realen Wald kreieren. Des Weiteren konnten wir dadurch Tiere integrieren, welche auf Basis von Blueprints sowie dem System eines Behavior Trees aufbauen. Dadurch haben die Tiere entsprechende Bereiche innerhalb eines Volumes, in welchen sie sich frei bewegen können. Dabei interagieren sie aktiv mit dem Spieler, sobald dieser zu nahe kommt, entfernen sich die Tiere vom Spieler an einen beliebigen Standort.

In der Spiewelt sind zusätzlich insgesamt zehn Münzen verteilt, welche der Spieler mit Hilfe einer überlappenden Kollision einsammeln kann. Abschließend gilt es noch zu erwähnen, dass unsichtbare Wände als Abgrenzungen an den Rändern der Spielwelt platziert wurden. Diese verhindern das Herausfallen des Spielers aus der Welt. Hierbei handelt es sich ebenfalls um einen modellierten Cube, welcher ein Material ohne farbliche Parameter zugewiesen bekommen hat.

## **5.2 Source-Code**

Blueprints sind im Allgemeinen eine ziemlich einfache und nachvollziehbare Möglichkeit ein Spiel innerhalb von der Unreal Engine zu programmieren. Aufgrund der Vielzahl an Screenshots wird auf sehr detaillierte Erklärungen verzichtet, da während der Entwicklung sprechende Variablen verwendet wurden, welche den Blueprint-Code für Außenstehende leicht lesbar machen. Dennoch wird im Folgenden ein Überblick der relevanten Blueprints geliefert, welche im Zuge dieses Programmentwurfs entstanden sind.

---

<sup>3</sup>Siehe Abbildungen A.2, A.3, A.4, A.6, A.9

### 5.2.1 MainMenu

Zu Beginn werden im „Construct“<sup>4</sup> die wichtigsten Variablen initialisiert, welche im Laufe des restlichen Blueprints noch benötigt werden. Zusätzlich können erste Voreinstellungen (Execute Console Command) getroffen und komplexerer Code (Save Main Menu<sup>5</sup>) ausgeführt werden. Im Weiteren Verlauf werden die einzelnen Bedienelemente<sup>6, 7</sup> des Widgets mit Code ausgestattet, um ihre eigentliche Funktion zu realisieren. Das Menü wird mit Hilfe einer .sav-Datei gespeichert, welche im SaveGames-Ordner des Projektes hinterlegt wird. Dabei wird der Blueprint „SaveMenuBP“<sup>8</sup> als Basis genutzt, welche Variablen letztendlich gespeichert werden sollen. Wie bereits erwähnt stellen die Grafikeinstellungen<sup>9</sup> den zentralen Punkt des Hauptmenüs dar, folglich wurde hierfür auch der meiste Code verwendet.

Ein Beispiel stellt der Ausschnitt von „AdjustViewDistance“<sup>10</sup> dar, dieser zeigt wie anhand der Veränderung am entsprechenden „ViewDistanceSlider“, sich die Sichtweite über die Konsole anpasst. Der Nutzer bekommt von diesen Konsolen-Befehlen nichts mit, dies geschieht alles im Hintergrund. Parallel wird der entsprechende Text beim Slider angepasst, so dass der Nutzer eine Visualisierung seiner durchgeführten Veränderung zu sehen bekommt.

### 5.2.2 AlchemistsHouse

Dieser Blueprint beinhaltet Blueprints von der „AlchemistsTorch“<sup>11</sup>, welche die Lichtquellen innerhalb des Hauses darstellen.

Im „BeginPlay“<sup>12</sup> des „AlchemistsHouse“ werden ebenfalls die relevanten Variablen initialisiert und weiterer Code ausgeführt. Somit besteht eine starke Ähnlichkeit zum „Construct“ des „MainMenu“. Mit Hilfe von Kollisionsboxen<sup>13</sup> kann eine Interaktion zwischen dem Spieler (FirstPersonCharacter) und dem Blueprint stattfinden<sup>14</sup>. Die Türen öffnen sich beispielsweise automatisch, bei der Kollision mit gewissen Boxen gehen

<sup>4</sup>Siehe Abbildungen B.2, B.3, B.4, B.5

<sup>5</sup>Siehe Abbildungen B.9, B.10, B.11, B.12, B.13, B.14, B.15, B.16, B.17

<sup>6</sup>StartGameButton, ReturnToGameButton, OptionsButton, QuitGameButton, OpenGithubBtn

<sup>7</sup>Siehe Abbildungen B.6, B.7, B.8, B.18

<sup>8</sup>Siehe Abbildung B.21

<sup>9</sup>Siehe Abbildung B.19

<sup>10</sup>Siehe Abbildung B.20

<sup>11</sup>Siehe Abbildungen C.1, C.2

<sup>12</sup>Siehe Abbildungen C.5, C.6

<sup>13</sup>EntryDoorCollisionBox, BedroomDoorCollisionBox, Ambry1CollisionBox, Chest1CollisionBox, AlchemistsRoomEntryCollisionBox

<sup>14</sup>Siehe Abbildungen C.7, C.8, C.9, C.10, C.11, C.12, C.13, C.14, C.15, C.16, C.17, C.18, C.19, C.20, C.21, C.22, C.23

Lichtquellen an oder es verändert sich die Benutzeroberfläche (FirstPersonHUD). Bei der kontinuierlichen Veränderung von Lichtquellen über eine Timeline, muss mit höheren Belastungen der Grafikeinheit gerechnet werden, da unzählige Berechnungen im Hintergrund der Unreal Engine durchgeführt werden müssen.

### 5.2.3 Coin

Die in der Welt platzierten Münzen<sup>15</sup> sollten unserem Programmentwurf eine Art „Gamification“, also einen spielerischen Aspekt, ergänzen. Dabei soll für den Spieler den Anreiz geschaffen werden, alle Münzen zu sammeln und somit die ganze Spielwelt zu erkunden. Eine solche animierte<sup>16</sup> Münze wird bei Kollision mit dem „FirstPerson-Character“ eingesammelt. Dies bedeutet, dass die Instanz des Coin Blueprints zerstört und er entfernt wird. Der Spieler hingegen bekommt im „FirstPersonHUD“ eine Münze gutgeschrieben<sup>17</sup>.

### 5.2.4 Teapot

In Anlehnung an die Vorlesung wollten wir auch das bekannteste 3D-Objekt, den „Utah teapot“<sup>18</sup>, in unseren Programmentwurf implementieren. Dieser befindet sich im Essbereich vom Haus des Alchemisten und ist ebenfalls animiert<sup>19</sup>. Er bewegt sich langsam und zieht somit die Aufmerksamkeit am statischen Esstisch auf sich. Des Weiteren fällt er durch seine hellblaue Färbung auf.

### 5.2.5 FirstPersonCharacter

Dieser Blueprint stellt den Pawn<sup>20</sup> dar, welcher vom Spieler benötigt wird um sich durch die Spielwelt zu navigieren und mit ihr zu interagieren<sup>21</sup>. Dabei werden hier die Bewegung<sup>22</sup> (Movement Input, Jump) sowie Orientierung<sup>23</sup> (Mouse Input) implementiert. Des Weiteren finden sich hier auch die Interaktionen<sup>24</sup> mit Objekten der Spielwelt (Interact With Objects) und den Zugang zum MainMenu<sup>25</sup> (Return To MainMenu) nachdem das Spiel gestartet wurde. Erweitert könnte dieser Blueprint noch mit einem tatsächlichen Spielcharakter und weiteren Aktionen wie beispielsweise rennen oder

<sup>15</sup>Siehe Abbildung D.1

<sup>16</sup>Siehe Abbildungen D.3, D.4, D.5, D.6, D.7

<sup>17</sup>Siehe Abbildungen D.8, D.9

<sup>18</sup>Siehe Abbildung E.1

<sup>19</sup>Siehe Abbildungen E.3, E.4, E.5

<sup>20</sup>Siehe Abbildungen F.1, F.2

<sup>21</sup>Siehe Abbildungen F.3, F.4, F.5

<sup>22</sup>Siehe Abbildungen F.6, F.8

<sup>23</sup>Siehe Abbildung F.7

<sup>24</sup>Siehe Abbildung F.10

<sup>25</sup>Siehe Abbildung F.9

schleichen. Darauf wurde aber aufgrund des bereits vorhandenen Umfangs des Programmierwurfs verzichtet.

### 5.2.6 FirstPersonHUD

Ein HUD, auch Head-up-Display genannt, wird in Spielen oftmals als User Interface während dem Spiel verwendet. Angezeigt werden dort beispielsweise ein Lebensbalken, eine Erfahrungsleiste oder das aktuelle Level des Spielers. In unserem Projekt genügte die Anzeige der bereits gesammelten Münzen sowie eines kurzen Satzes, falls sich der Spieler in der Nähe eines Objektes zur Interaktion befindet<sup>26</sup>.

### 5.2.7 Animals

Wir wollten unserer Spielwelt etwas mehr Leben verleihen, dabei kam uns die Idee Tiere<sup>27</sup> zu implementieren, welche sich frei auf der Landschaft bewegen können. Die verschiedenen Tiere sollten innerhalb einer Datenbank<sup>28</sup> festgehalten werden, da wir ihnen unterschiedliche Werte für ihre gemeinsamen Eigenschaften<sup>29</sup> geben wollten. Ein weiterer Grund für eine Datenbank ist, dass man dadurch die Möglichkeit hat einen sogenannten „MasterBP“<sup>30</sup> zu erstellen, welcher als zentrale Klasse für alle Tiere fungiert. Dadurch muss nicht für jedes Tier ein eigener Blueprint mit dupliziertem Code programmiert werden.

Im Begin Play<sup>31</sup> des „AnimalsMasterBP“ werden die entsprechenden Werte des DataTables<sup>32</sup> mit Hilfe der manuell gesetzten AnimalID aus der Instanz des Blueprints innerhalb der Spielwelt geladen. Zudem wird eine der beiden Variationen der Häute geladen<sup>33</sup>, um automatisch eine gewisse Vielfalt bei den Tieren zu generieren. Dem AnimalsMasterBP wird ein Blueprint (AnimalMasterAiController)<sup>34</sup> hinzugefügt, welcher den entsprechenden Behavior Tree<sup>35</sup> beinhaltet und mit Hilfe von „AiPerception“<sup>36</sup> auf den FirstPersonCharacter reagieren kann. Der Behavior Tree schaut hierbei dauerhaft ob sich der Spieler in der Nähe befindet oder nicht<sup>37</sup>. Befindet sich der Spieler in der Nähe, so bewegt sich das Tier jede Sekunde zu einem neuen zufälligen Standort in

<sup>26</sup>Siehe Abbildungen G.1, F.4, F.5

<sup>27</sup>Siehe Abbildungen H.1, H.2, H.3, H.6

<sup>28</sup>Siehe Abbildungen H.4, H.5

<sup>29</sup>Eigenschaften: AnimalStruct, Werte in Datenbank: AnimalsDataTable

<sup>30</sup>Siehe Abbildung H.7

<sup>31</sup>Siehe Abbildungen H.8, H.9

<sup>32</sup>Siehe Abbildungen H.10, H.11

<sup>33</sup>Siehe Abbildung H.12

<sup>34</sup>Siehe Abbildung H.13

<sup>35</sup>Siehe Abbildung H.14

<sup>36</sup>Siehe Abbildung H.15

<sup>37</sup>Siehe Abbildung H.16

einem Radius von 1500 Einheiten<sup>38</sup>. Ist kein Spieler im Umkreis, so sucht sich das Tier nur alle sieben Sekunden einen neuen Standort.

### 5.3 User Interface

Die Benutzeroberfläche eines Computerprogrammes ermöglicht es dem Nutzer mit dem Programm zu interagieren. Bei einem Spiel beinhaltet es oftmals Bereiche für Grafikeinstellungen sowie Bedienelemente oder zur Darstellung relevanter Informationen. Bei unserem Programmentwurf entschlossen wir uns zur Implementierung eines umfangreichen Hauptmenüs<sup>39</sup> (MainMenu), welches es dem Nutzer erlauben soll, gezielte Änderungen an der Grafikqualität<sup>40</sup> innerhalb der Spielwelt vorzunehmen. Des Weiteren sind im Bereich der Credits<sup>41</sup> unsere Quellen für die verwendeten Texturen, Materials und Objekte hinterlegt. Unten rechts im Hauptmenü befindet sich ein GitHub-Button, welcher auf unser GitHub Repository referenziert. Die Buttons „Start Game“ sowie „Quit Game“ sind selbsterklärend.

Innerhalb der Spielwelt hat der Spieler eine dauerhafte Anzeige der bereits gesammelten Münzen. Des Weiteren wird dem Spieler angezeigt, sobald er sich in der Nähe eines Objektes befindet, mit welchem dieser interagieren kann. Dabei haben wir uns für die Taste E entschieden, da diese sich direkt neben den Tasten zur Steuerung (W, A, S, D) befindet.

---

<sup>38</sup>Siehe Abbildungen H.17, H.18, H.19

<sup>39</sup>Siehe Abbildung I.1

<sup>40</sup>Siehe Abbildung I.2

<sup>41</sup>Siehe Abbildung I.3

## 6 Fazit

Das Projekt ist im Allgemeinen mehr als erfolgreich gelaufen. Alle gestellten Muss-Anforderungen wurden erfüllt. Von der Idee und dem Aufbau der 3D-Szene, der Anzahl der verwendeten Objekte, sowie der Steuerung des Spielercharakters und dem Menü zur Verwaltung der Applikation sind alle Punkte vorhanden. Weiterhin wurde die Landschaft sehr kreativ durch völlig unterschiedliche 3D-Objekte ausgeschmückt und enthält zudem ein Ziel, nämlich das Einsammeln von zehn Münzen die sich verteilt in der Spielwelt befinden.

Abschließend möchten wir festhalten, dass uns das tiefere Einarbeiten in die Unreal Engine vor neue Herausforderungen gestellt hat, wir diese jedoch meistern konnten. Des Weiteren konnte uns die Computergrafik die komplexen Hintergründe einer solchen Engine näher bringen, was unser Verständnis nochmals erweiterte.

## Literatur

- Funes, Antonio (24. Mai 2020). *Unreal Engine 5: Was kann die Next-Gen-Engine und was sind Engines überhaupt*. PC GAMES. URL: <https://www.pcgames.de/Unreal-Engine - Software - 239301 / Specials / spiele - games - grafik - 3d - historie - allgemein-epic-vorschau-1350751/> (besucht am 17.11.2021).
- Grubb, Jeff (19. Nov. 2019). *Why 'The Mandalorian' Cites Fortnite Dev Epic Games in Its Credits*. VentureBeat. URL: <https://venturebeat.com/2019/11/19/why-the-mandalorian-cites-fortnite-dev-epic-games-in-its-credits/> (besucht am 21.11.2021).
- Pv, Satheesh (2021). "Introduction to Blueprints". In: *Beginning Unreal Engine 4 Blueprints Visual Scripting: Using C++: From Beginner to Pro*. Hrsg. von Satheesh Pv. Berkeley, CA: Apress, S. 21–31. ISBN: 978-1-4842-6396-9. DOI: 10.1007/978-1-4842-6396-9\_2. URL: [https://doi.org/10.1007/978-1-4842-6396-9\\_2](https://doi.org/10.1007/978-1-4842-6396-9_2) (besucht am 21.11.2021).
- Unreal Engine (2021). *Unreal Engine | The Most Powerful Real-Time 3D Creation Tool*. Unreal Engine. URL: <https://www.unrealengine.com/en-US/> (besucht am 16.11.2021).
- Unreal Engine Docs (2021a). *Behavior Trees*. URL: <https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/BehaviorTrees/> (besucht am 22.11.2021).
- (2021b). *Dynamic Scene Shadows*. URL: <https://docs.unrealengine.com/4.27/en-US/Resources/ContentExamples/DynamicSceneShadows/> (besucht am 22.11.2021).
- (2021c). *Level Editor*. URL: <https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LevelEditor/> (besucht am 22.11.2021).
- (2021d). *Materials*. URL: <https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphing/Materials/> (besucht am 22.11.2021).
- (2021e). *Shadow Casting*. URL: <https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/Shadows/> (besucht am 22.11.2021).
- (2021f). *Blueprint Visual Scripting*. URL: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/> (besucht am 21.11.2021).
- (2021g). *Opening Doors*. URL: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Timelines/Examples/OpeningDoors/> (besucht am 22.11.2021).
- Valencia-García, Rafael u. a. (7. Nov. 2016). *Technologies and Innovation: Second International Conference, CITI 2016, Guayaquil, Ecuador, November 23-25, 2016, Proceedings*. Springer. 289 S. ISBN: 978-3-319-48024-4. Google Books: bZZyDQAAQBAJ.

## A Anhang: Map



**Abb. A.1:** Aufnahme der Landschaft 1



**Abb. A.2:** Aufnahme der Landschaft 2



**Abb. A.3:** Aufnahme der Landschaft 3



**Abb. A.4:** Aufnahme der Landschaft 4



**Abb. A.5:** Aufnahme der Landschaft 5



**Abb. A.6:** Aufnahme der Landschaft 6



**Abb. A.7:** Aufnahme der Landschaft 7



**Abb. A.8:** Aufnahme der Landschaft 8



**Abb. A.9:** Aufnahme der Landschaft 9



**Abb. A.10:** Aufnahme der Landschaft 10



**Abb. A.11:** Aufnahme der Landschaft 11



**Abb. A.12:** Aufnahme der Landschaft 12



**Abb. A.13:** Aufnahme der Landschaft 13

## B Anhang: MainMenu Widget

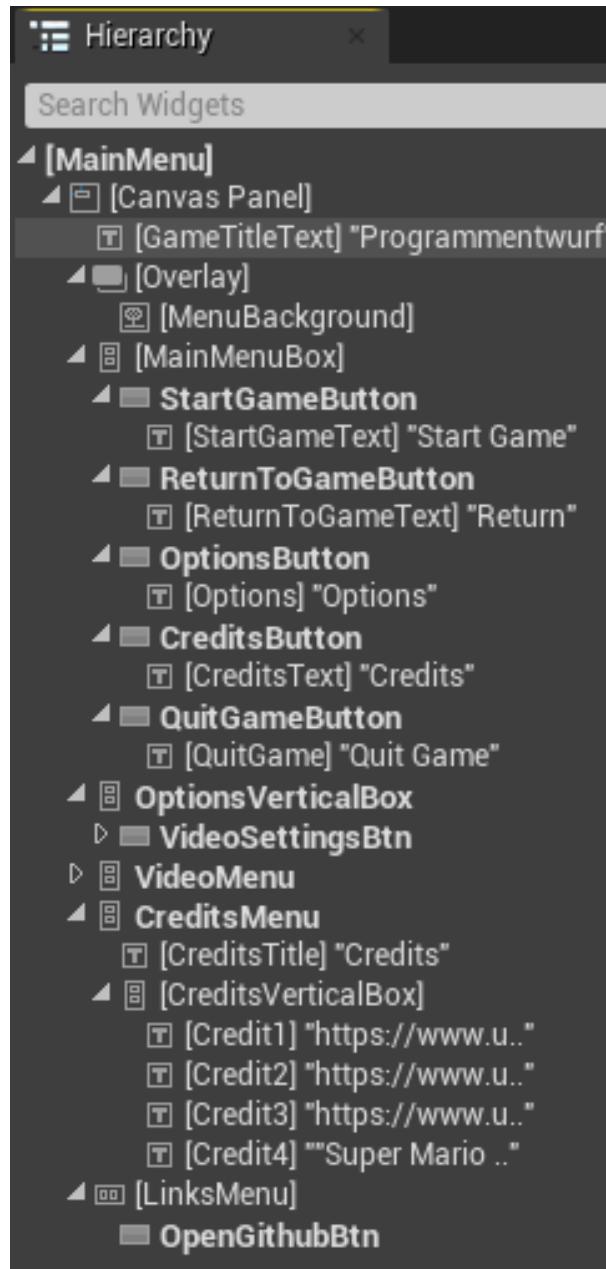


Abb. B.1: MainMenu Widget: Hierarchy

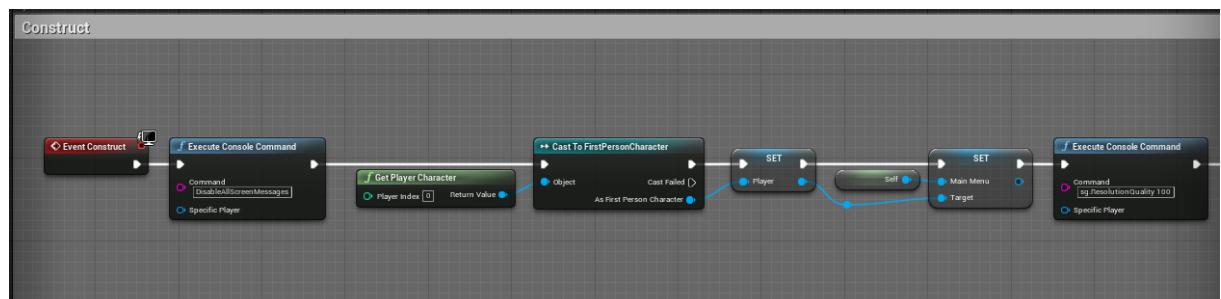
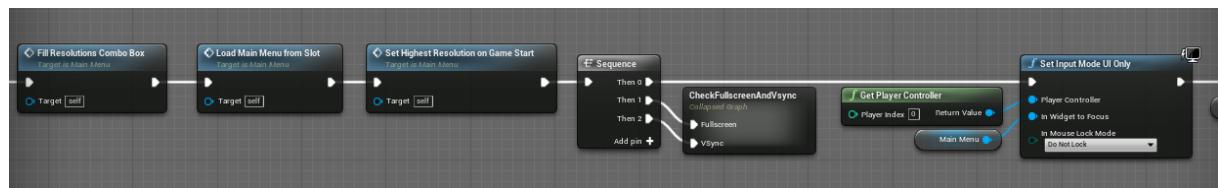
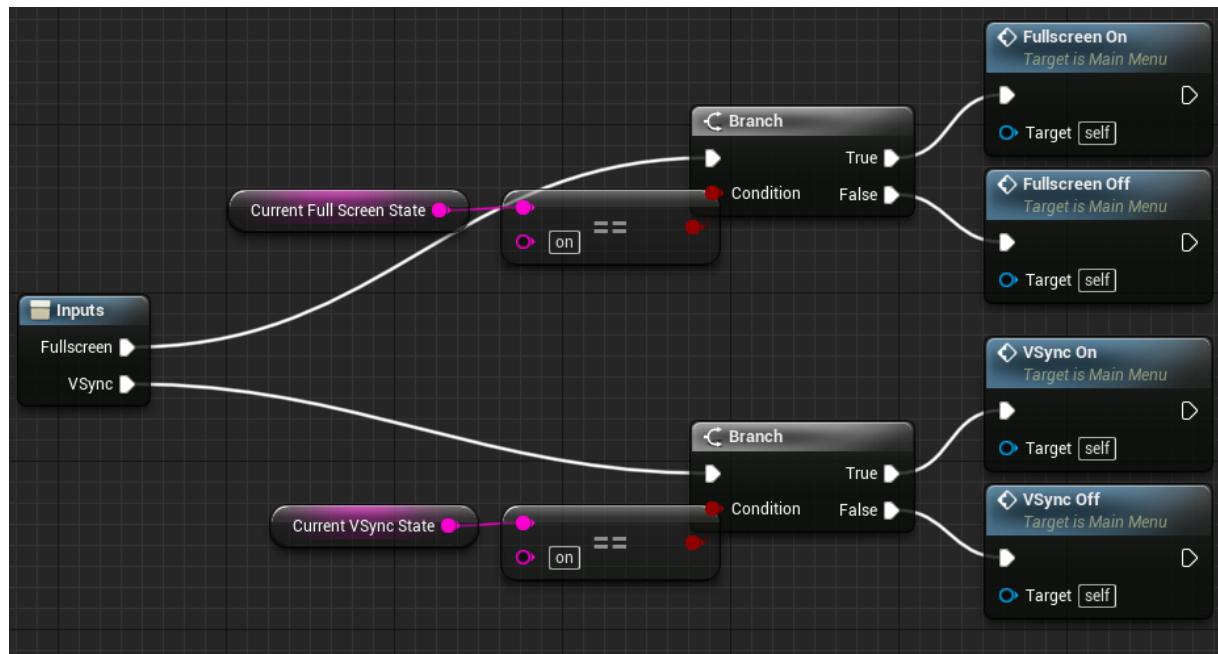
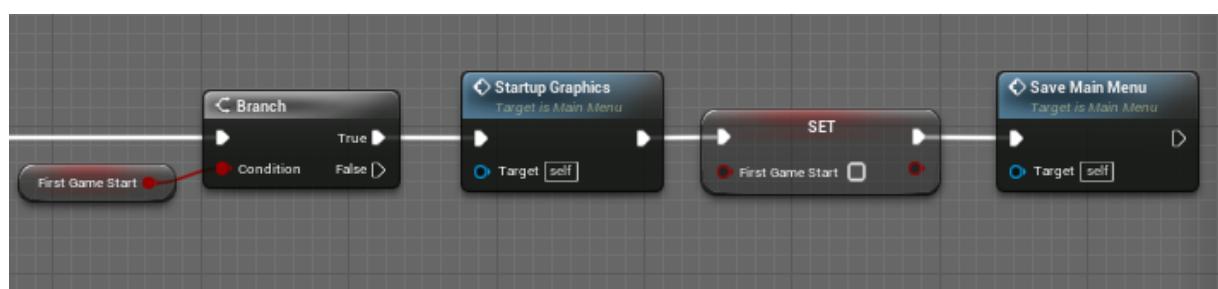


Abb. B.2: MainMenu Widget: Construct Part 1

**Abb. B.3:** MainMenu Widget: Construct Part 2**Abb. B.4:** MainMenu Widget: Construct Part 2 - Check Fullscreen And Vsync**Abb. B.5:** MainMenu Widget: Construct Part 3

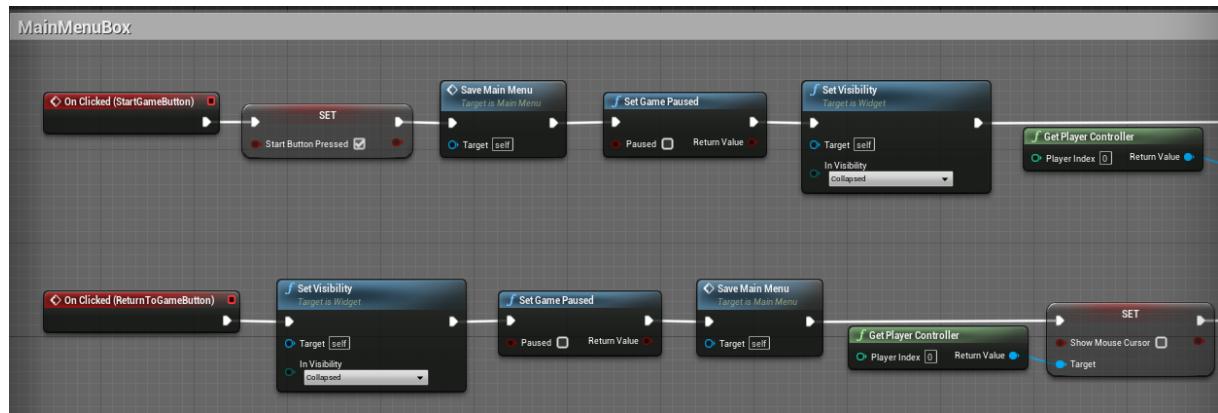


Abb. B.6: MainMenu Widget: MainMenuBox Part 1

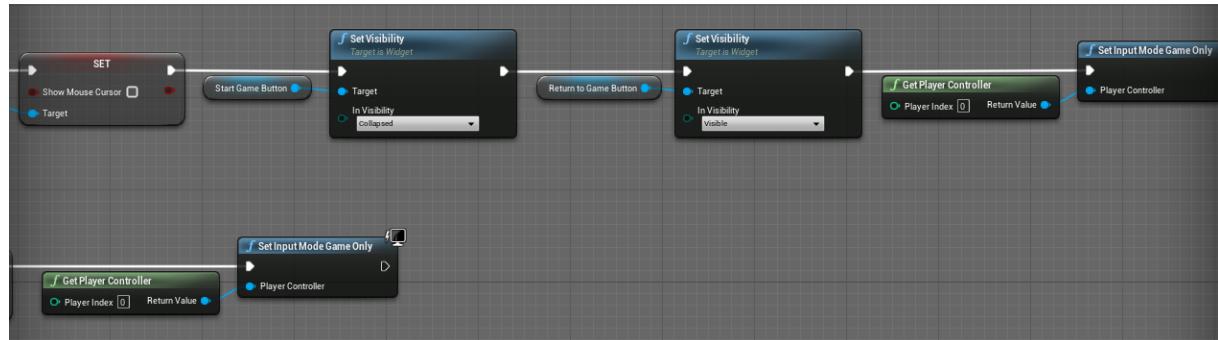


Abb. B.7: MainMenu Widget: MainMenuBox Part 2

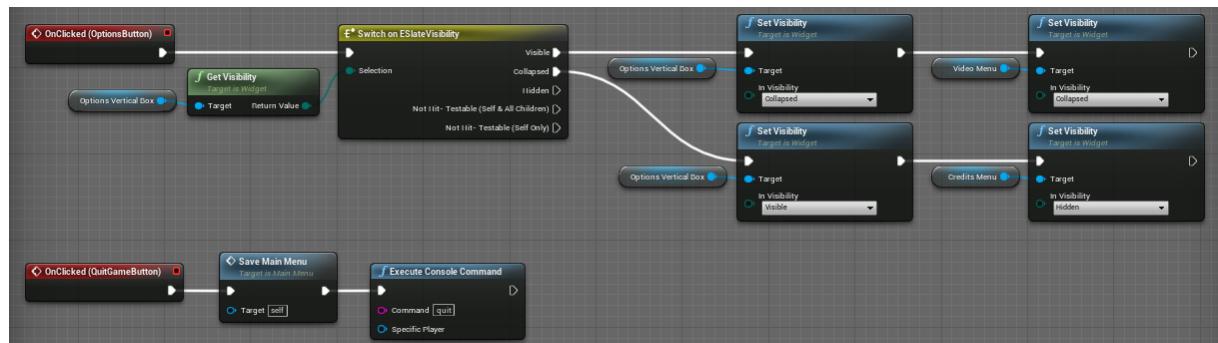
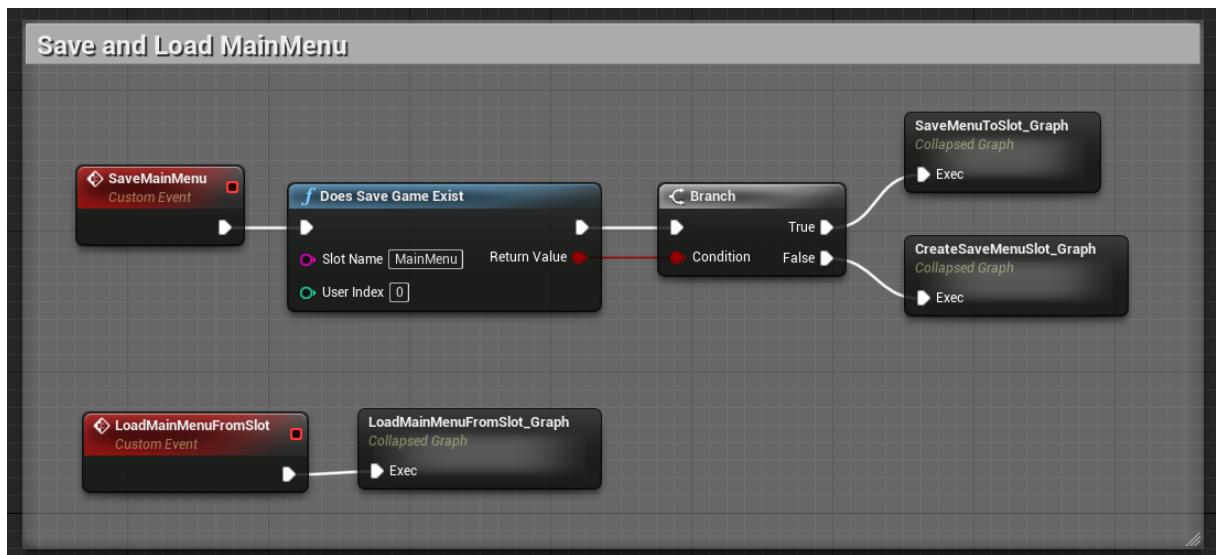
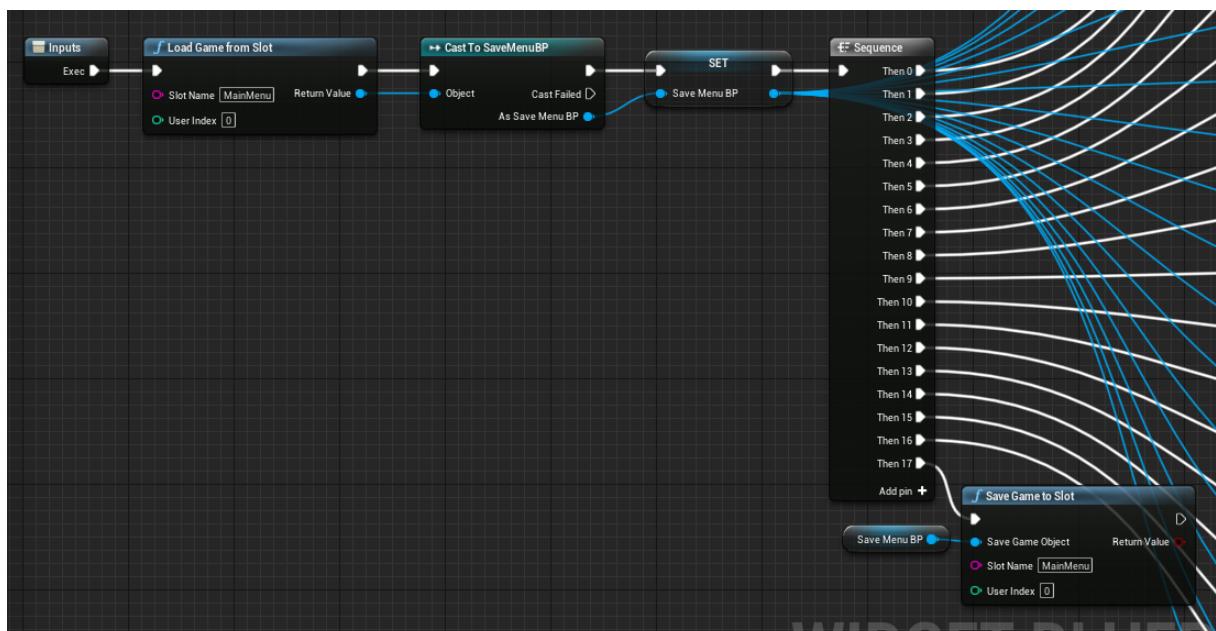


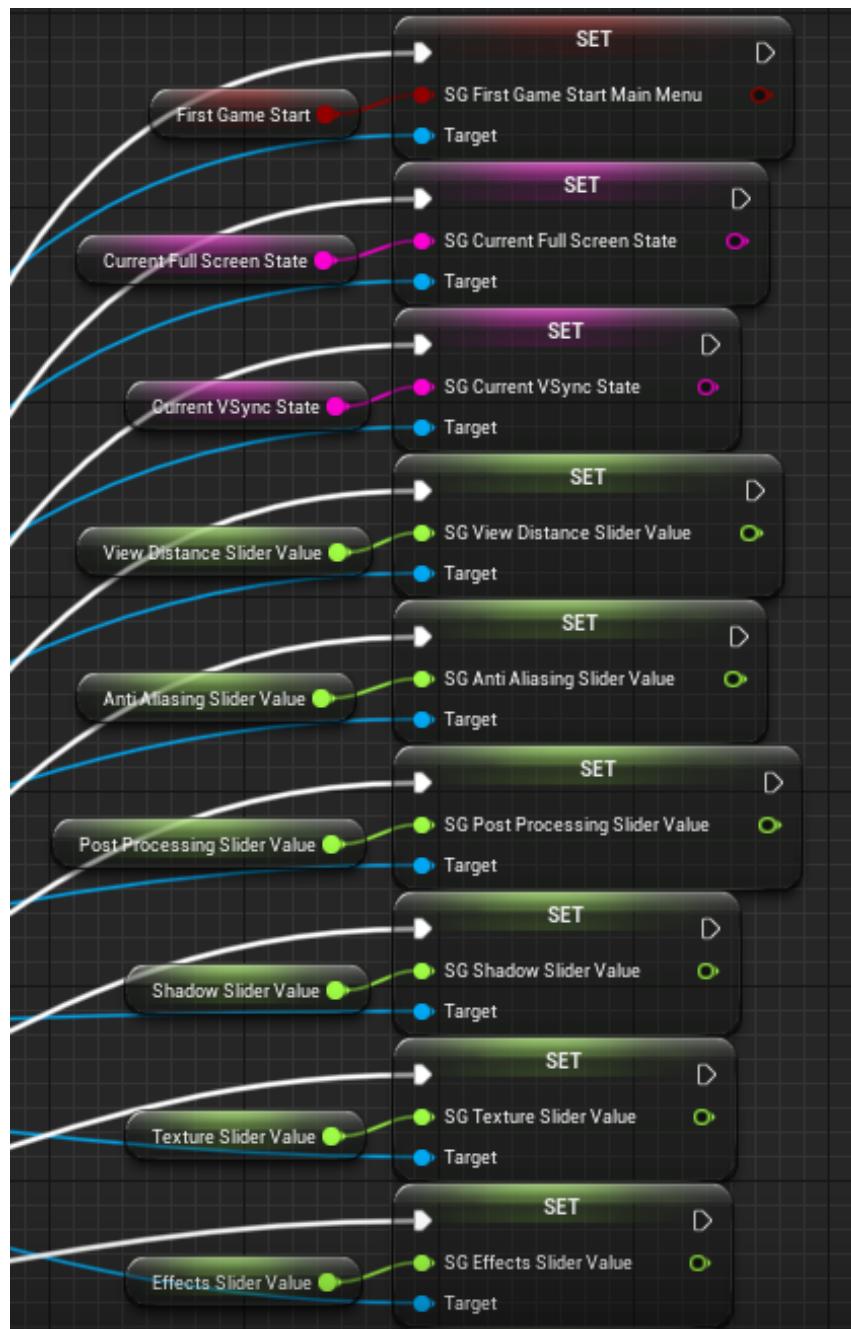
Abb. B.8: MainMenu Widget: MainMenuBox Part 3



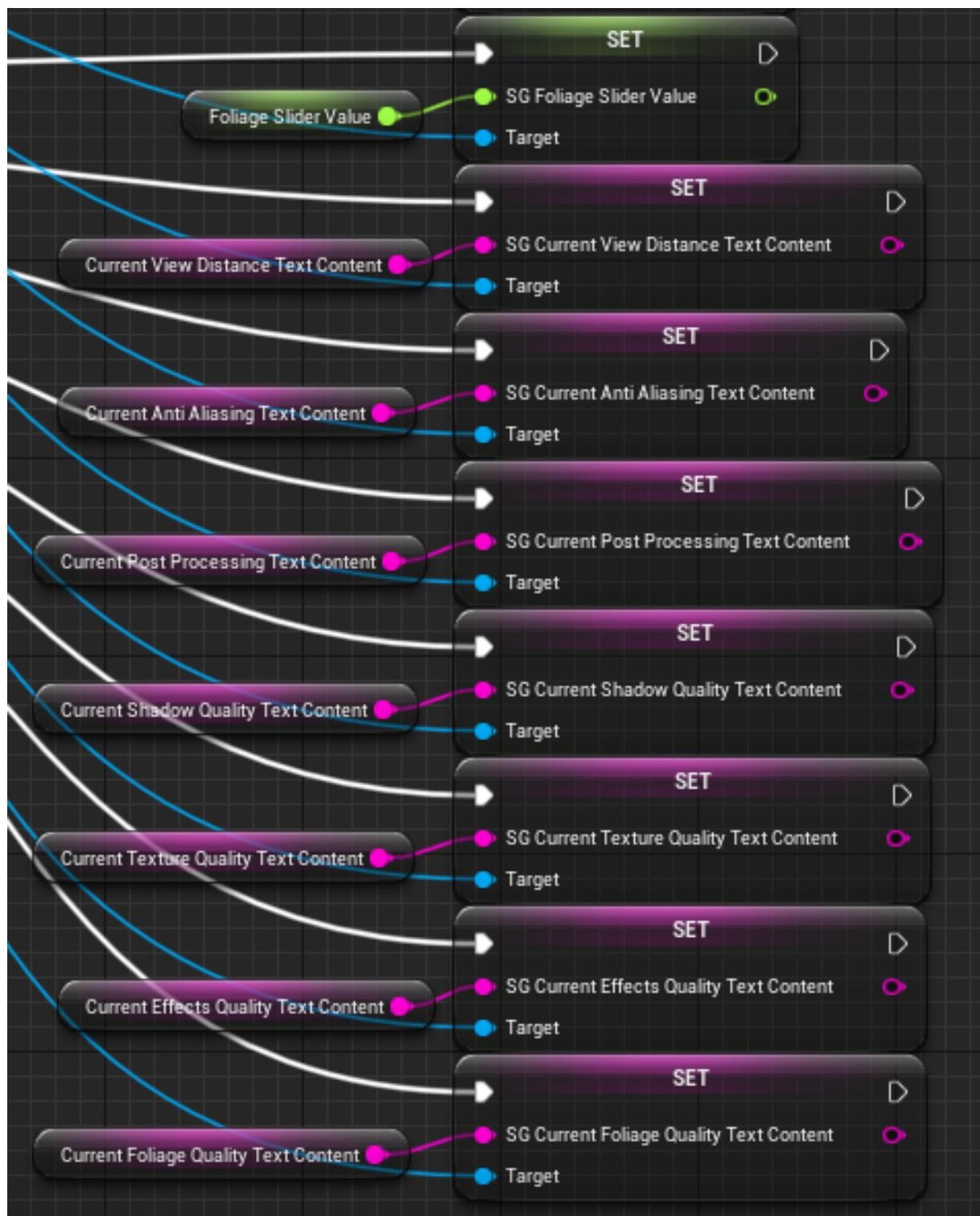
**Abb. B.9:** MainMenu Widget: Save And Load MainMenu Part 1



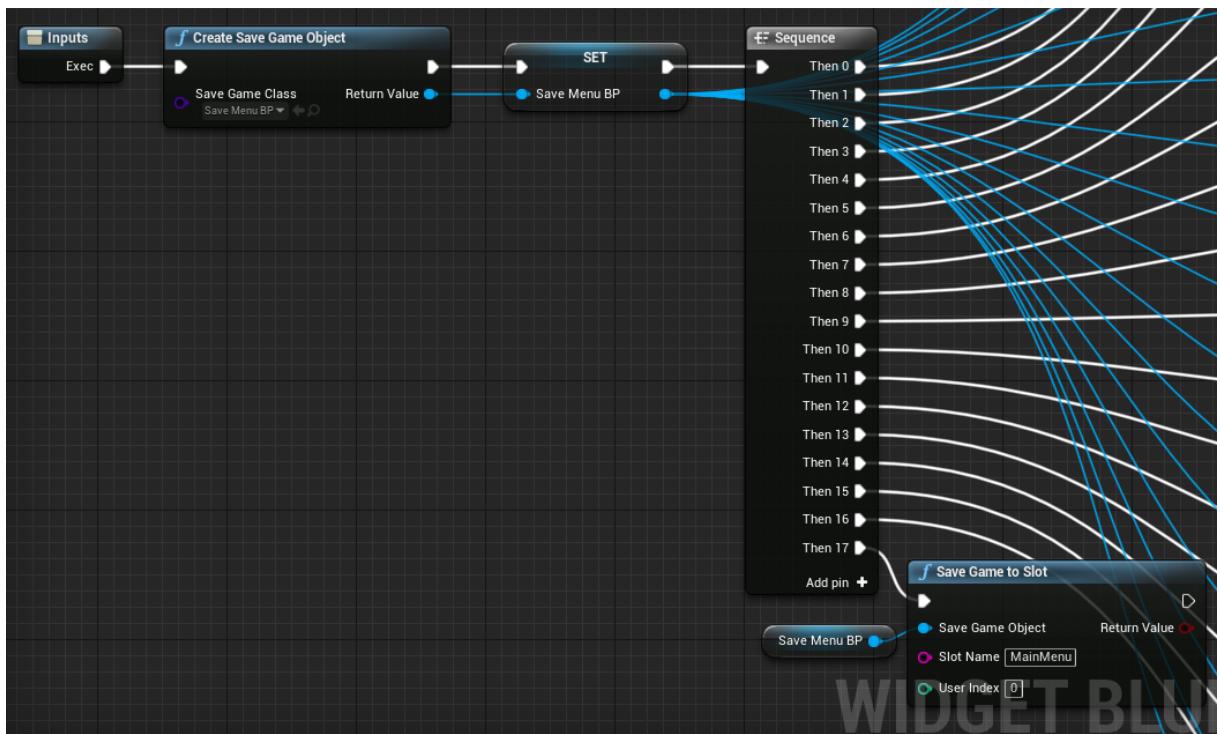
**Abb. B.10:** MainMenu Widget: Save And Load MainMenu Part 2 - Save Menu To Slot Part 1



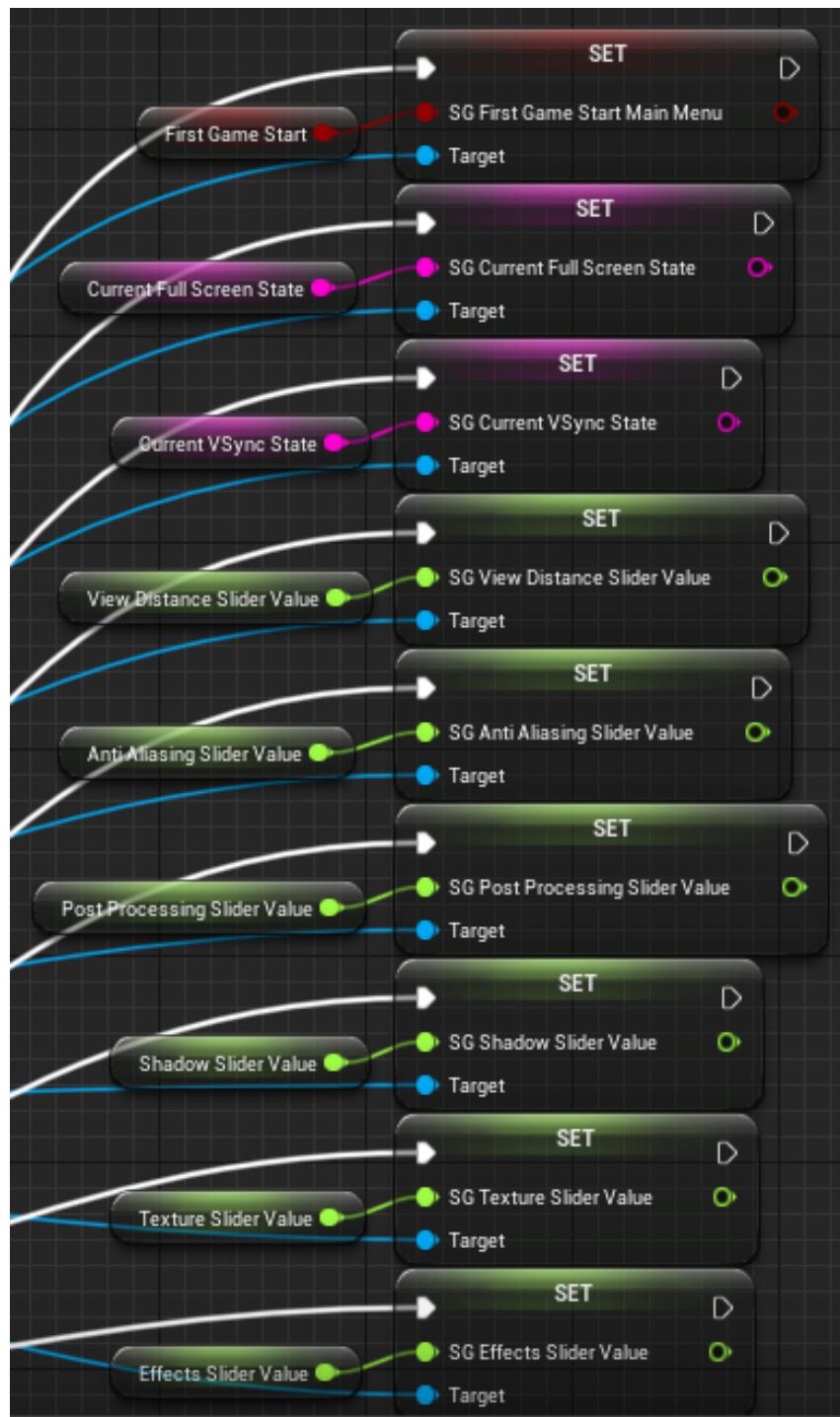
**Abb. B.11:** MainMenu Widget: Save And Load MainMenu Part 3 - Save Menu To Slot Part 2



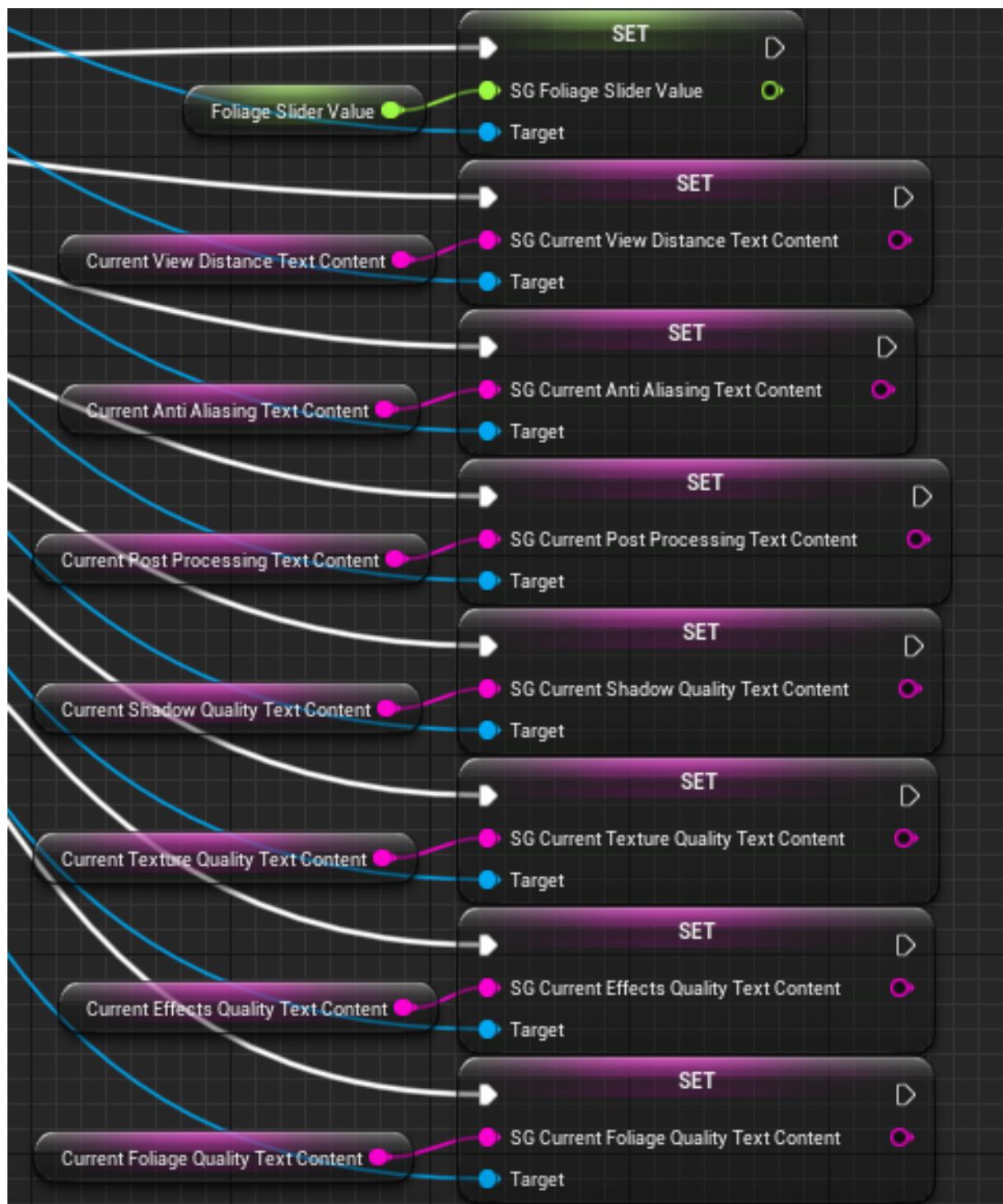
**Abb. B.12:** MainMenu Widget: Save And Load MainMenu Part 4 - Save Menu To Slot Part 3



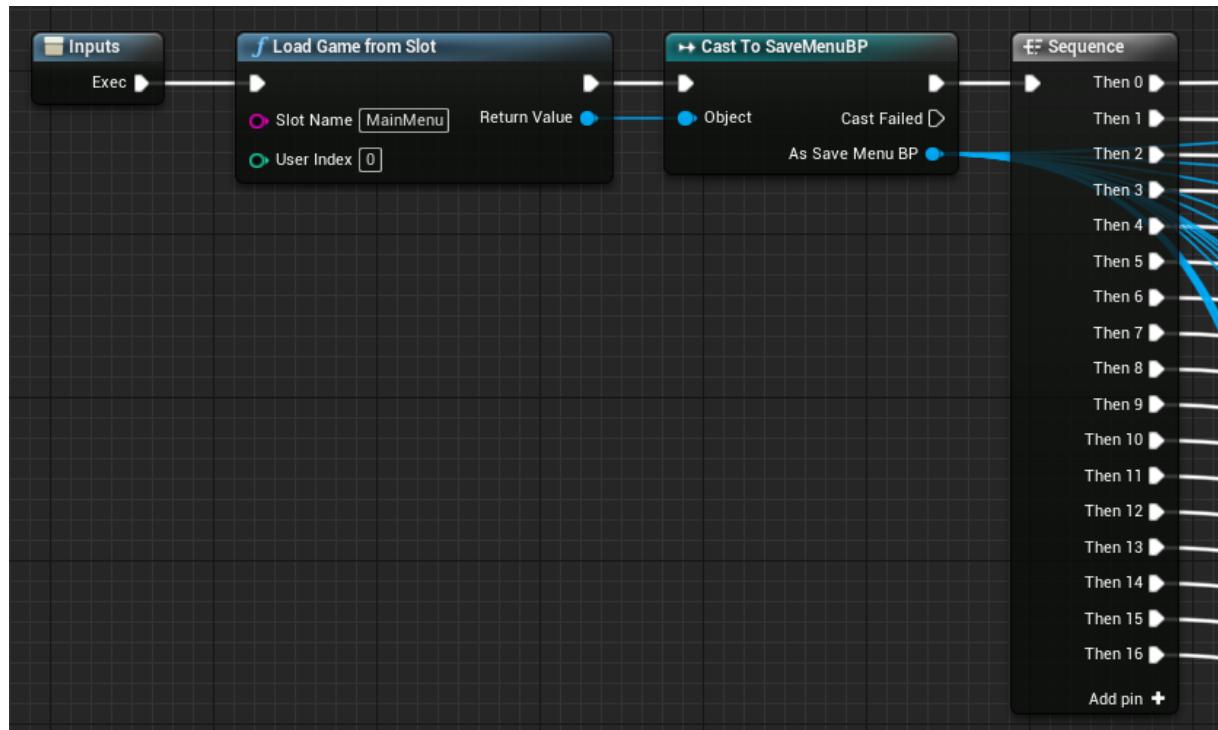
**Abb. B.13:** MainMenu Widget: Save And Load MainMenu Part 5 - Create Save Menu Slot Part 1



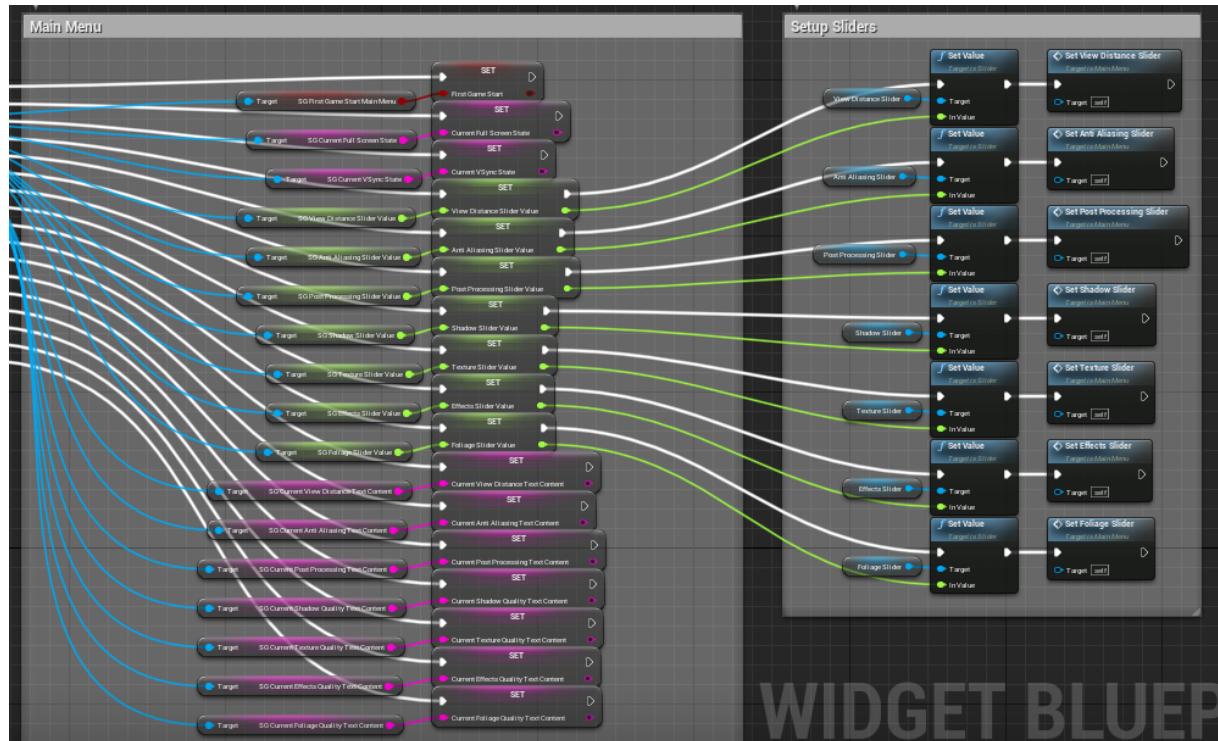
**Abb. B.14:** MainMenu Widget: Save And Load MainMenu Part 6 - Create Save Menu Slot Part 2



**Abb. B.15:** MainMenu Widget: Save And Load MainMenu Part 7 - Create Save Menu Slot Part 3



**Abb. B.16:** MainMenu Widget: Save And Load MainMenu Part 8 - Load MainMenu From Slot Part 1



**Abb. B.17:** MainMenu Widget: Save And Load MainMenu Part 9 - Load MainMenu From Slot Part 2

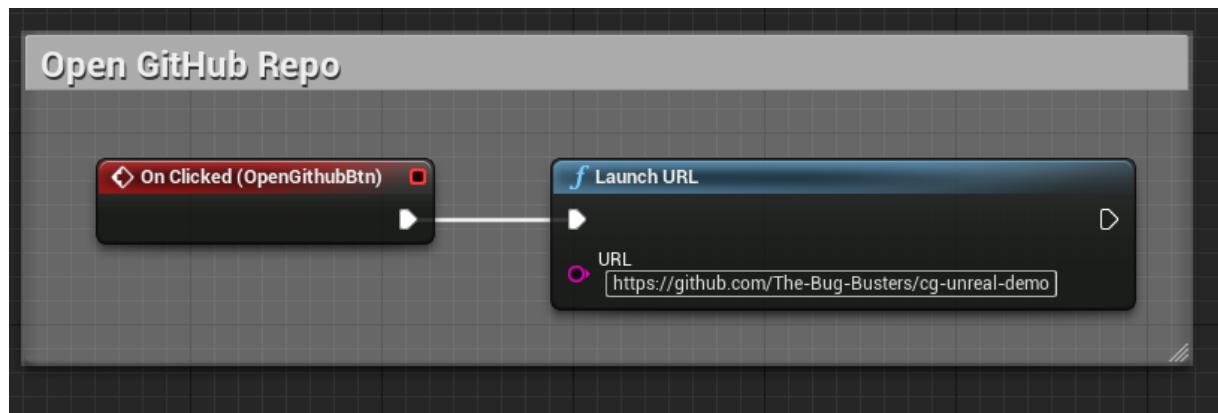


Abb. B.18: MainMenu Widget: Open GitHub Repo

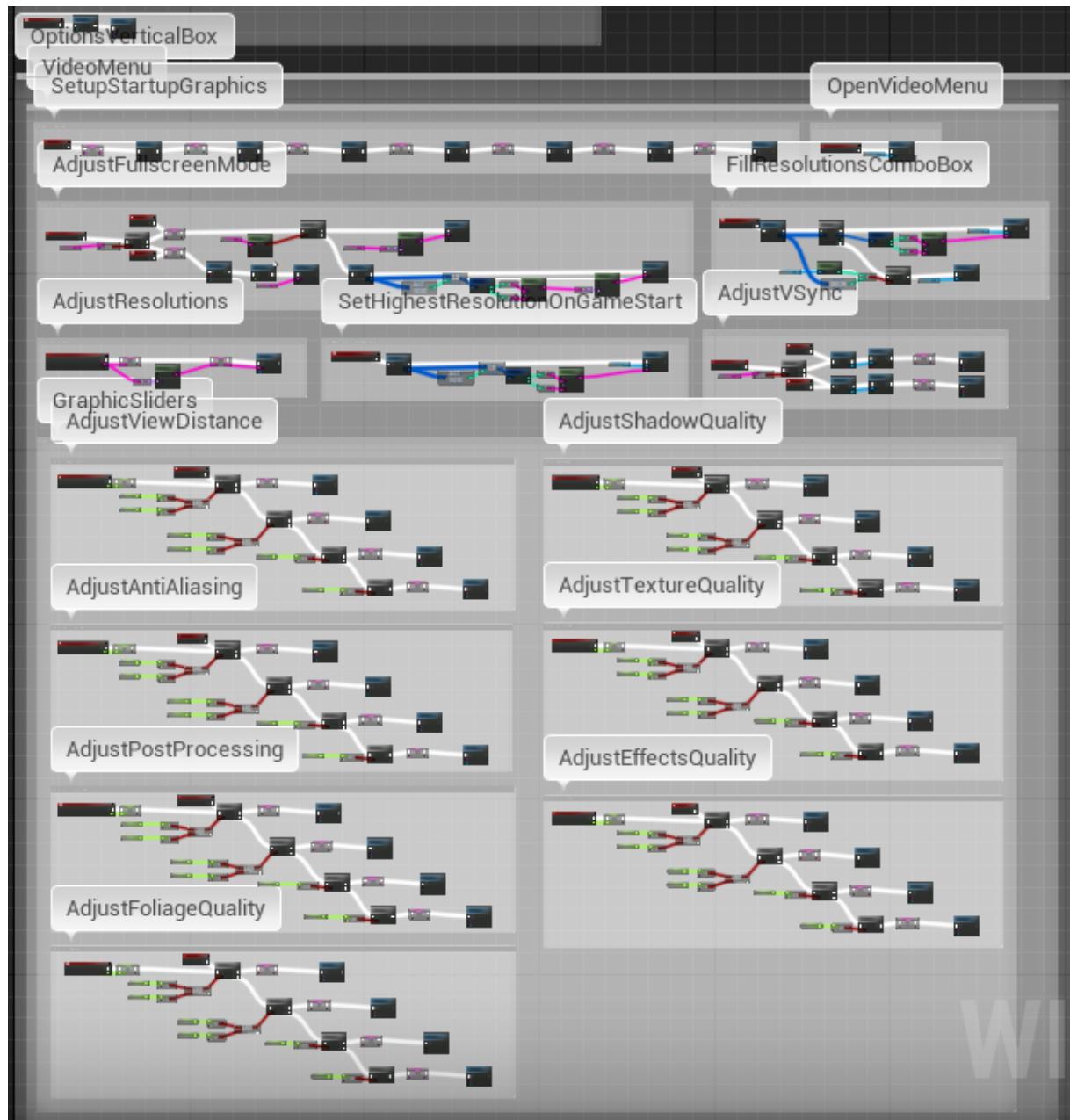


Abb. B.19: MainMenu Widget: Options

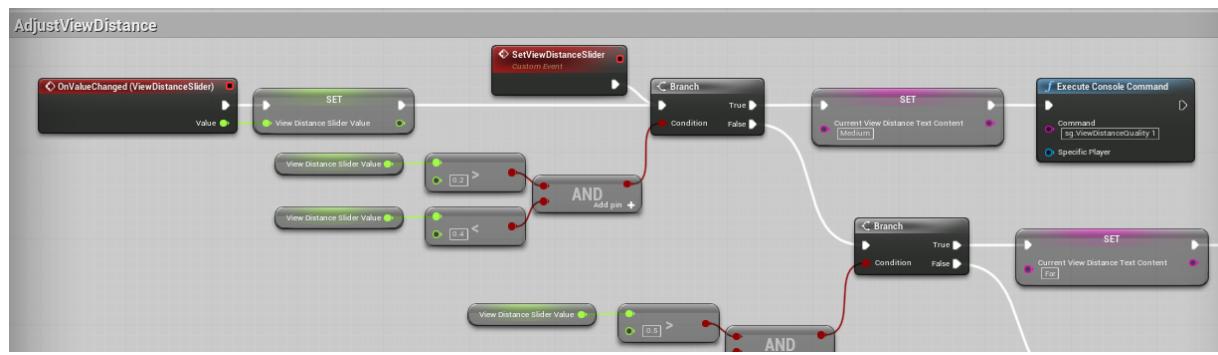


Abb. B.20: MainMenu Widget: Options Example

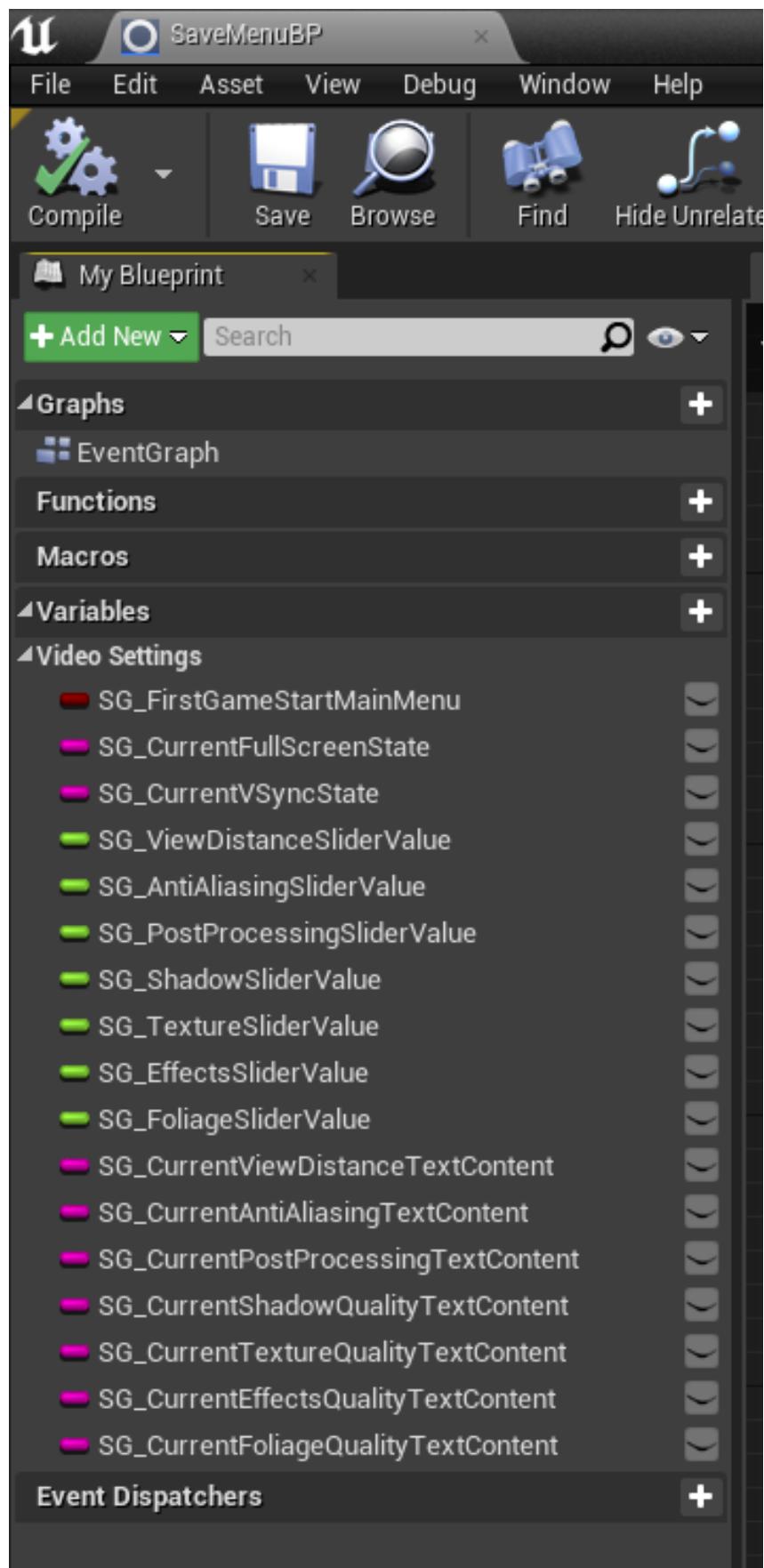
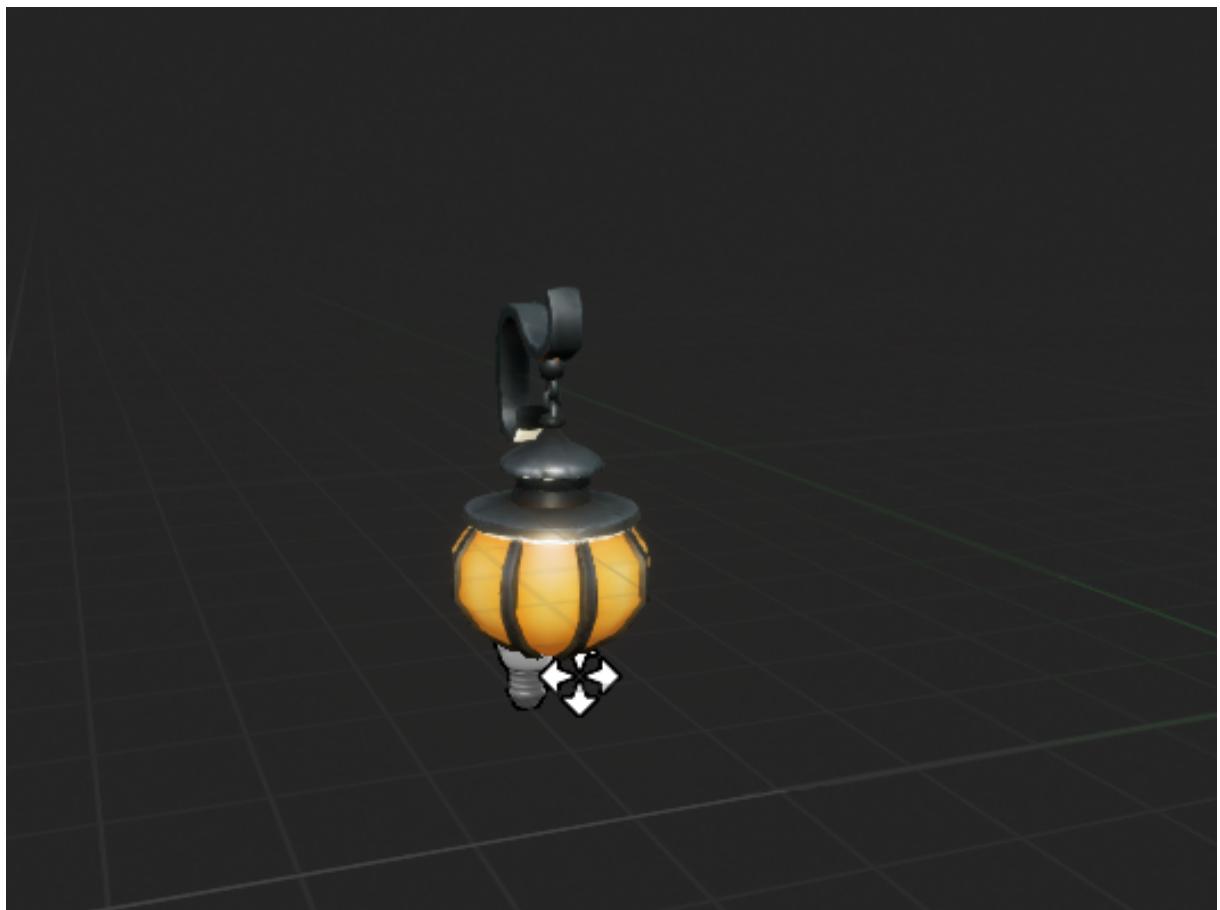


Abb. B.21: SaveMenuBP

## C Anhang: AlchemistsHouse



**Abb. C.1:** AlchemistsTorchBP: Viewport

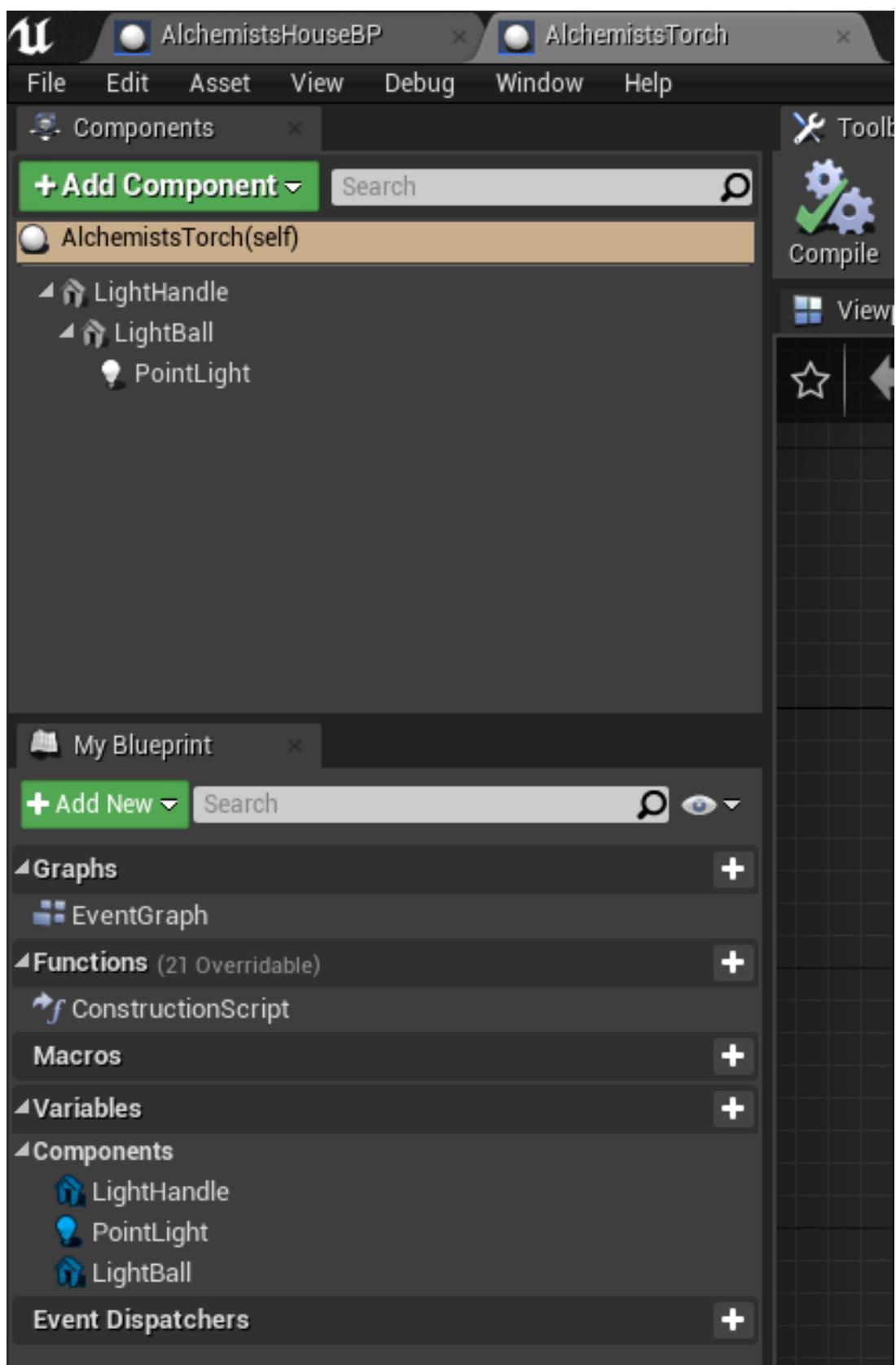
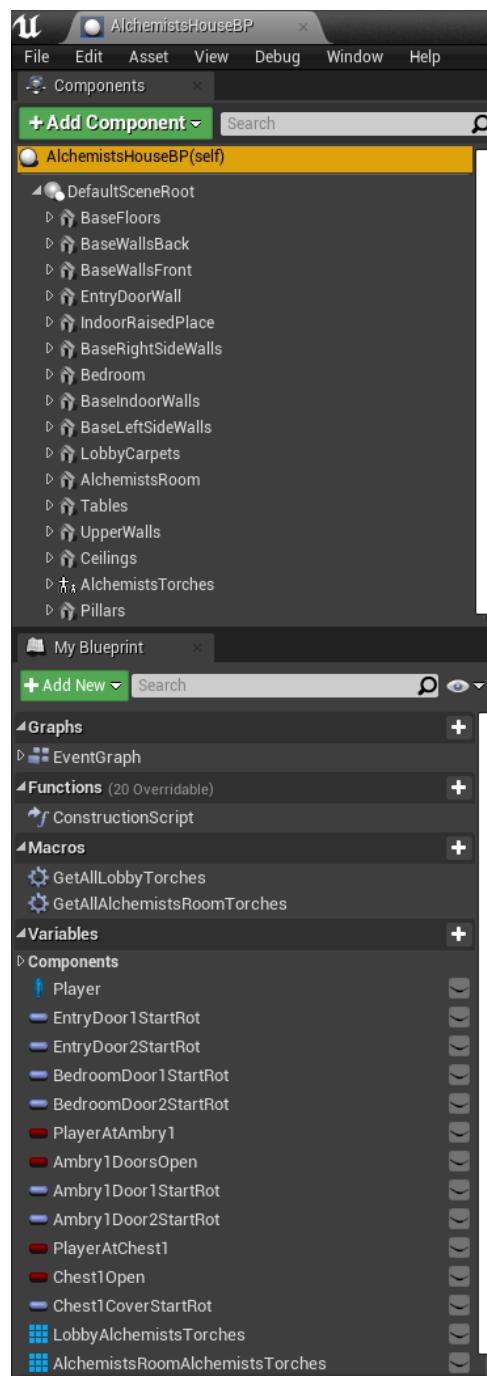


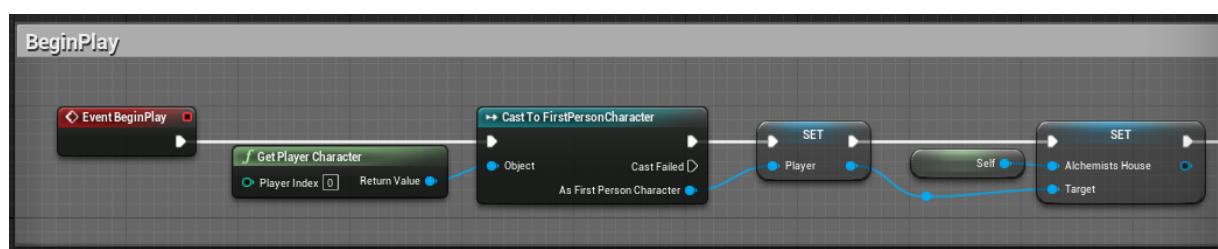
Abb. C.2: AlchemistsTorchBP: Components



**Abb. C.3:** AlchemistsHouseBP: Viewport



**Abb. C.4:** AlchemistsHouseBP: Components And Variables



**Abb. C.5:** AlchemistsHouseBP: Begin Play Part 1

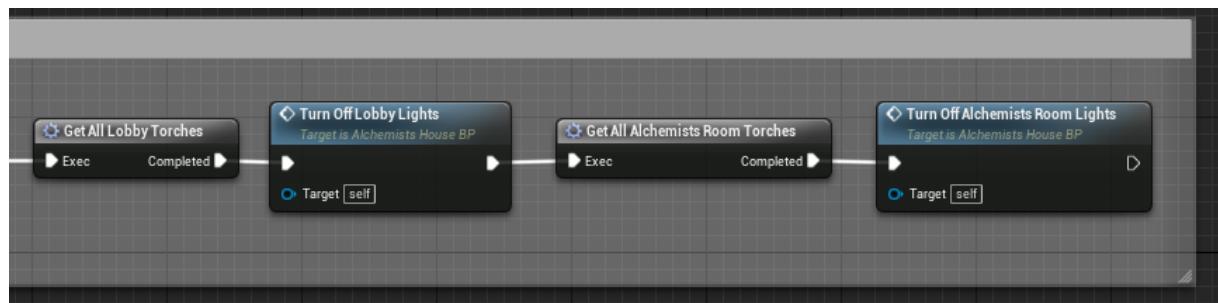


Abb. C.6: AlchemistsHouseBP: Begin Play Part 2

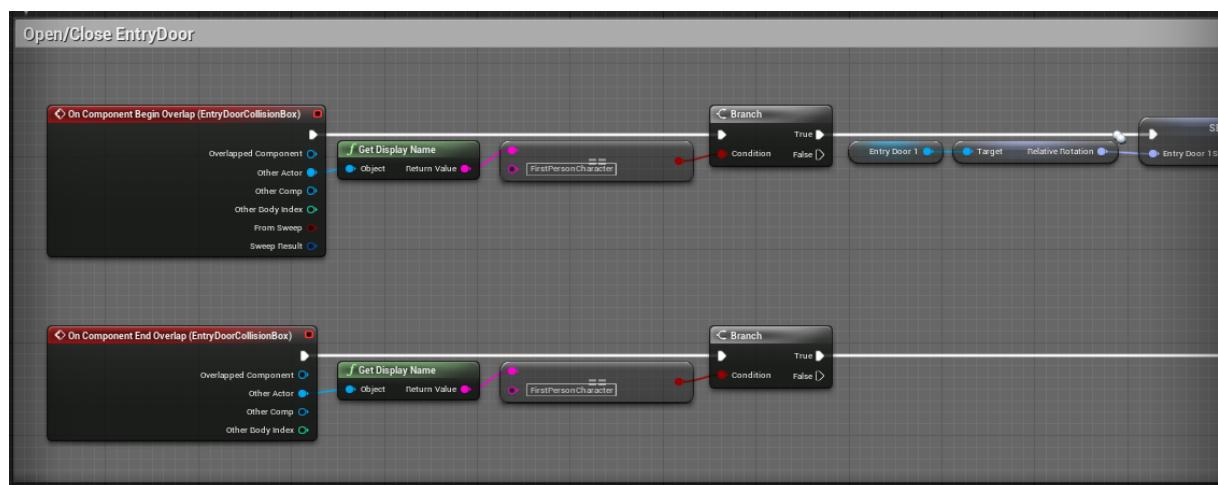


Abb. C.7: AlchemistsHouseBP: Open/Close Entry Door Part 1

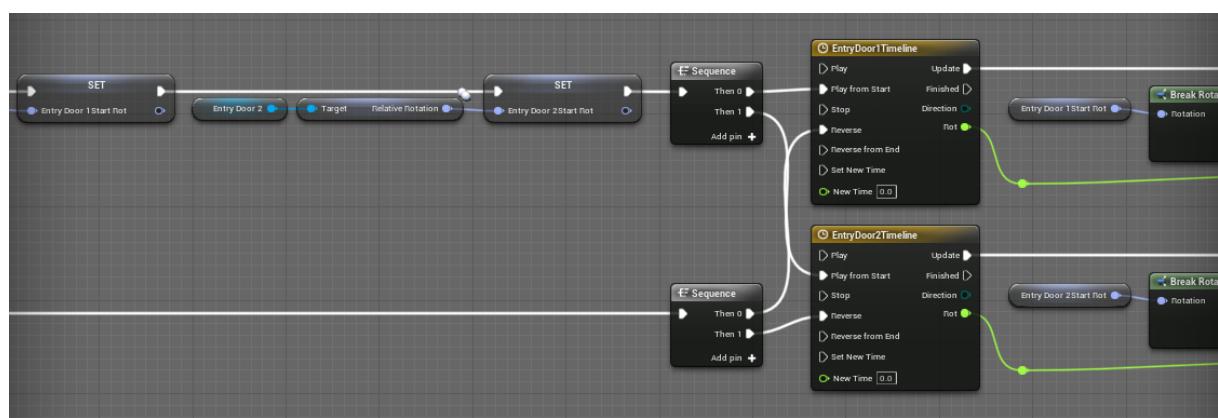
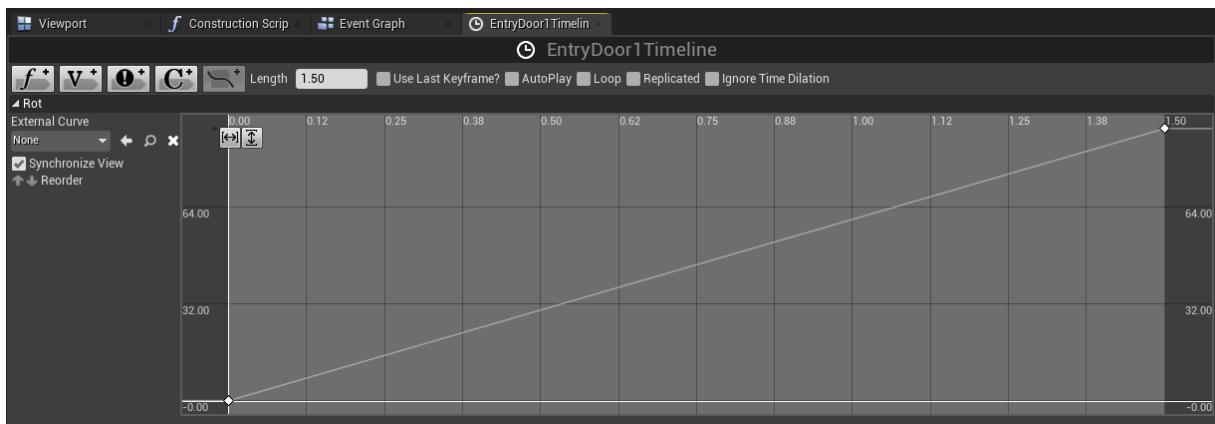
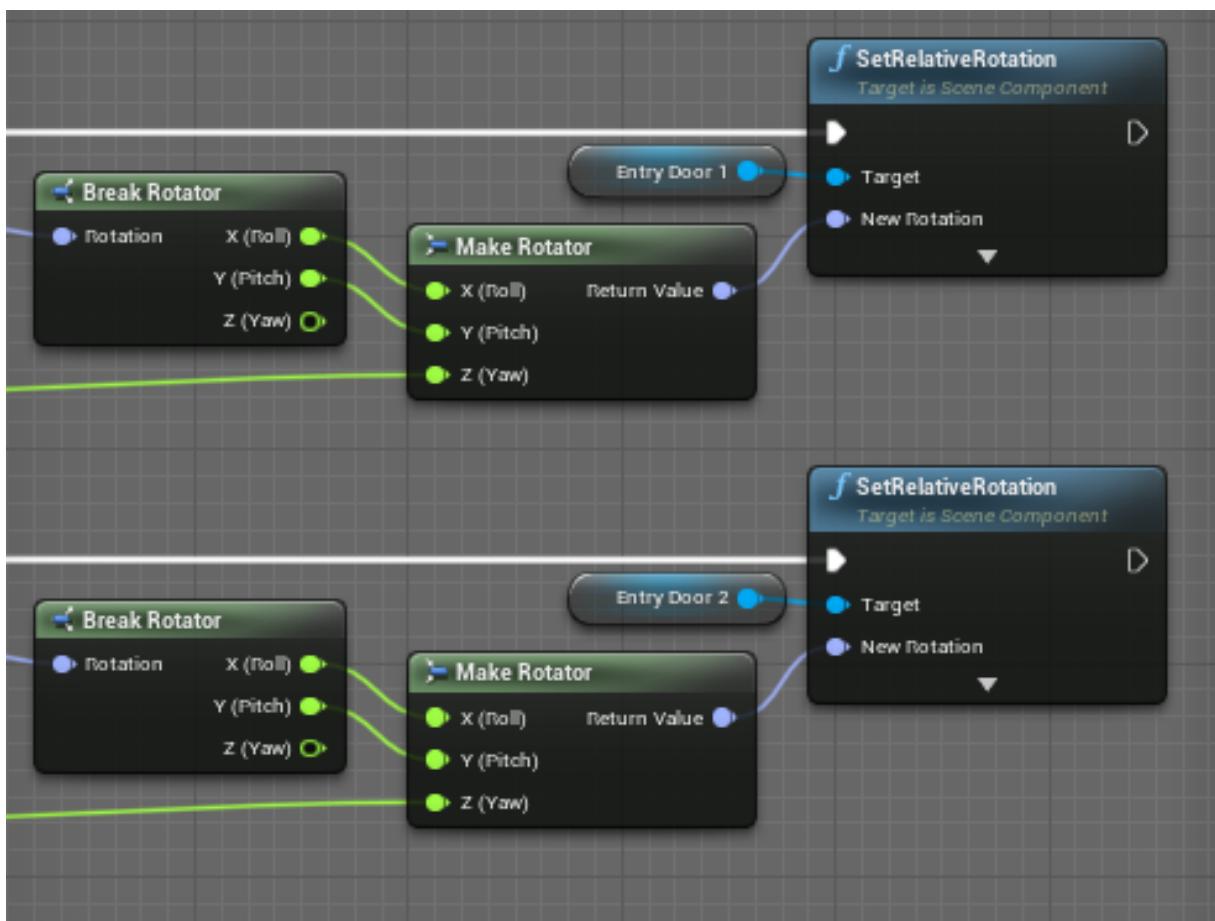


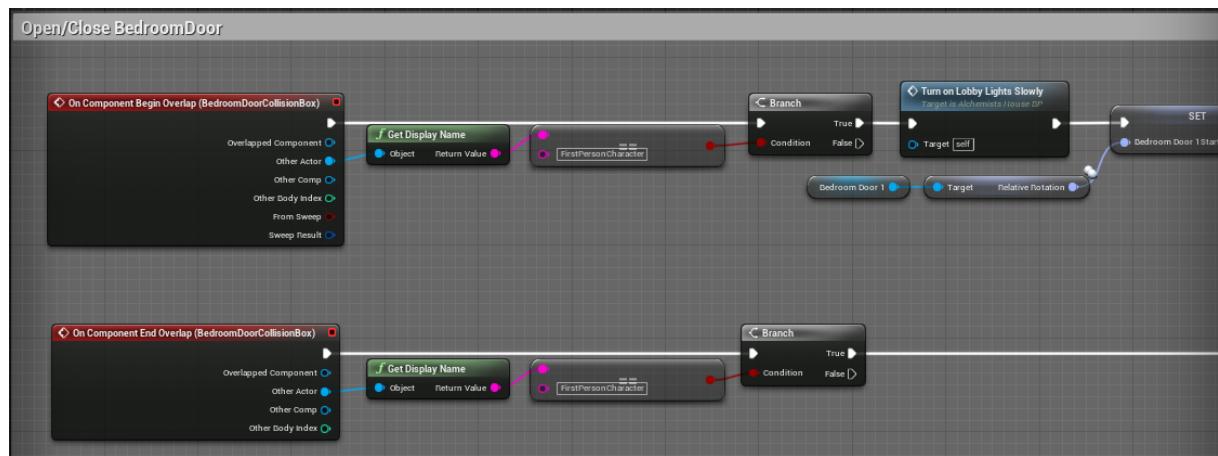
Abb. C.8: AlchemistsHouseBP: Open/Close Entry Door Part 2



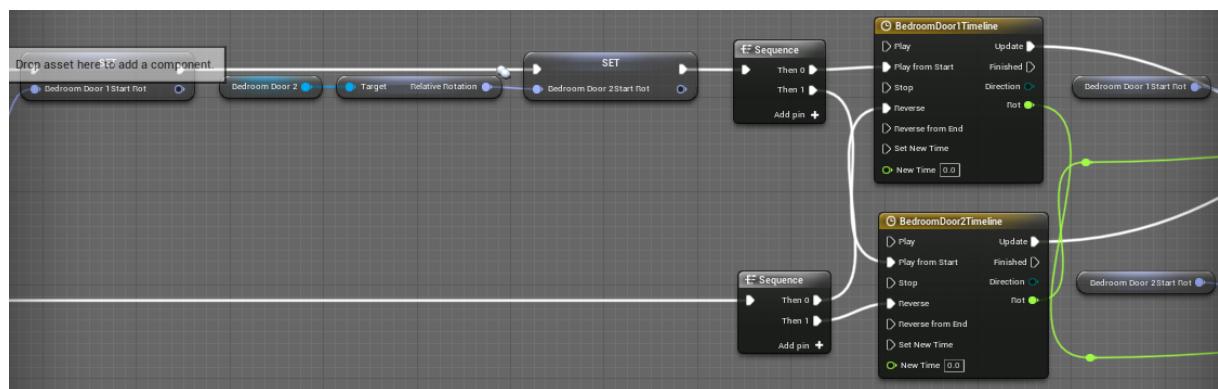
**Abb. C.9:** AlchemistsHouseBP: Open/Close Entry Door Part 2 - EntryDoor1Timeline



**Abb. C.10:** AlchemistsHouseBP: Open/Close Entry Door Part 3



**Abb. C.11:** AlchemistsHouseBP: Open/Close Bedroom Door Part 1



**Abb. C.12:** AlchemistsHouseBP: Open/Close Bedroom Door Part 2

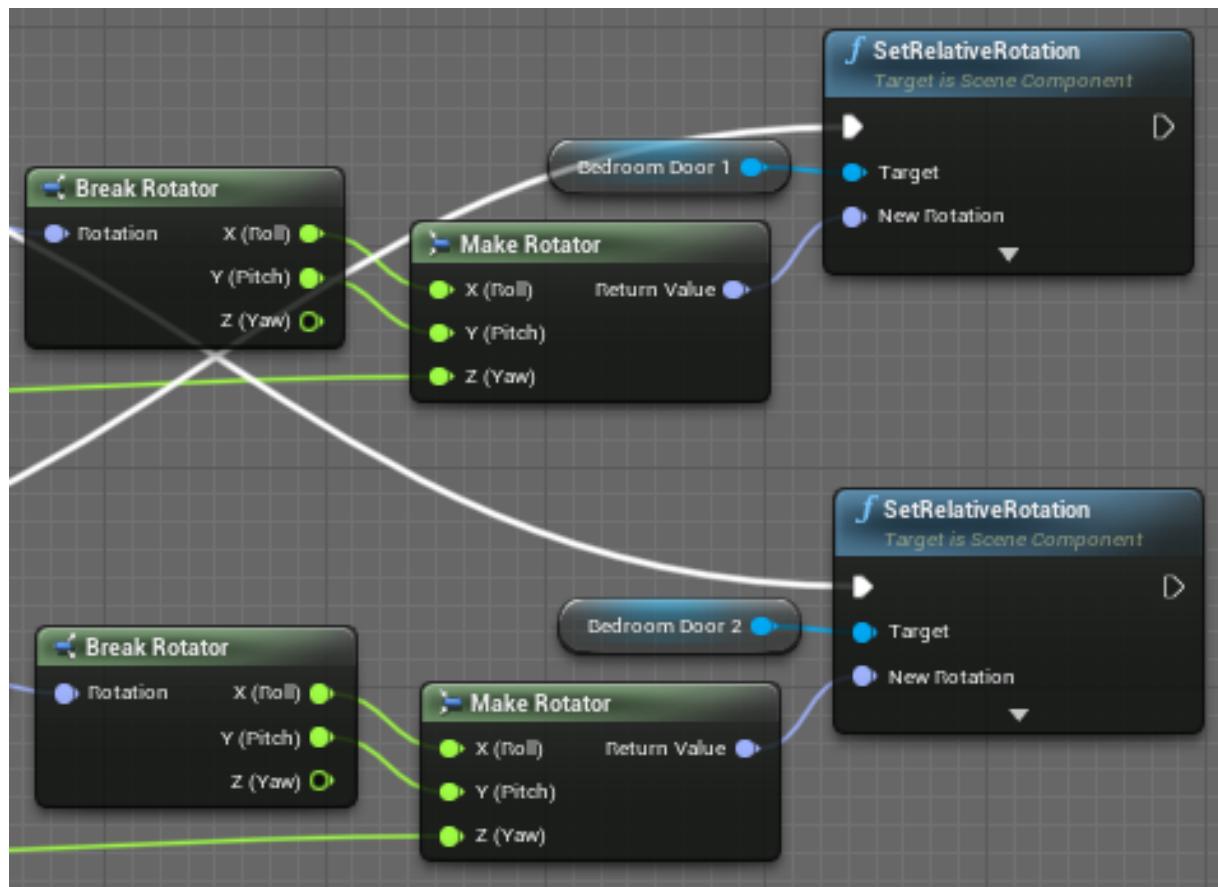


Abb. C.13: AlchemistsHouseBP: Open/Close Bedroom Door Part 3

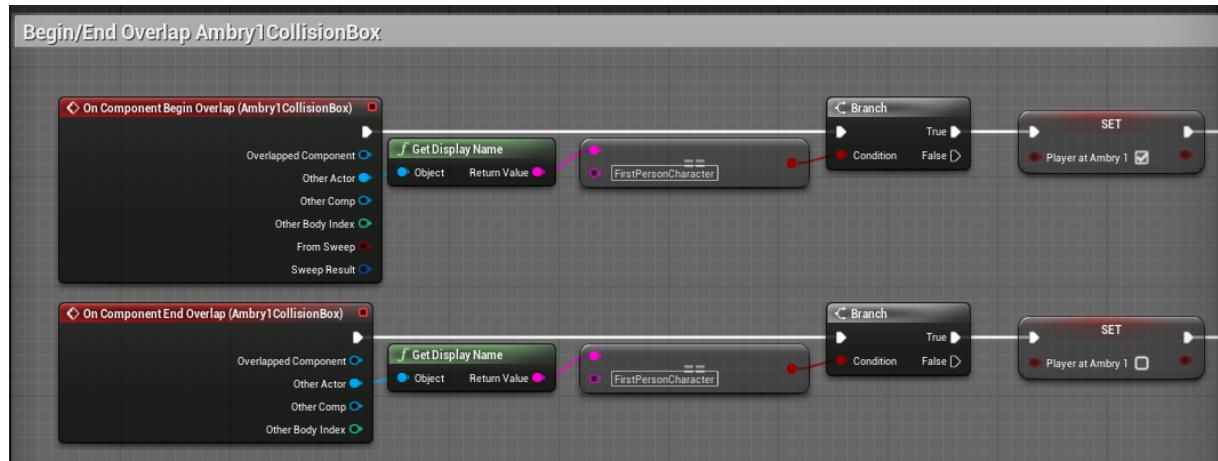
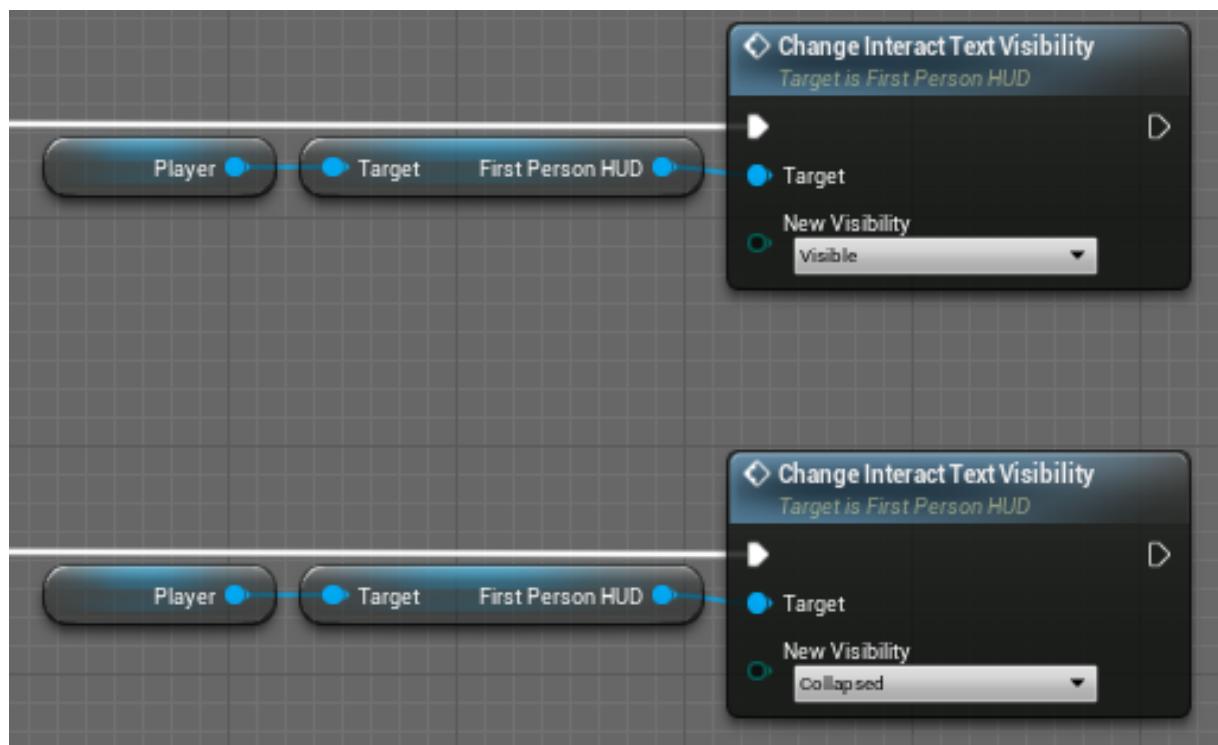
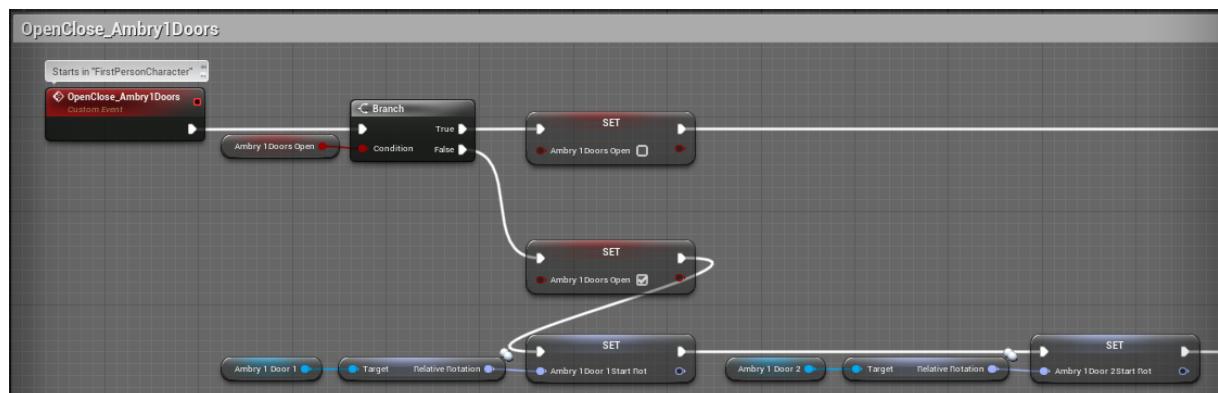


Abb. C.14: AlchemistsHouseBP: Begin/End Overlap Ambry1 CollisionBox Part 1



**Abb. C.15:** AlchemistsHouseBP: Begin/End Overlap Ambry1 CollisionBox Part 2



**Abb. C.16:** AlchemistsHouseBP: Open/Close Ambry1 Doors Part 1

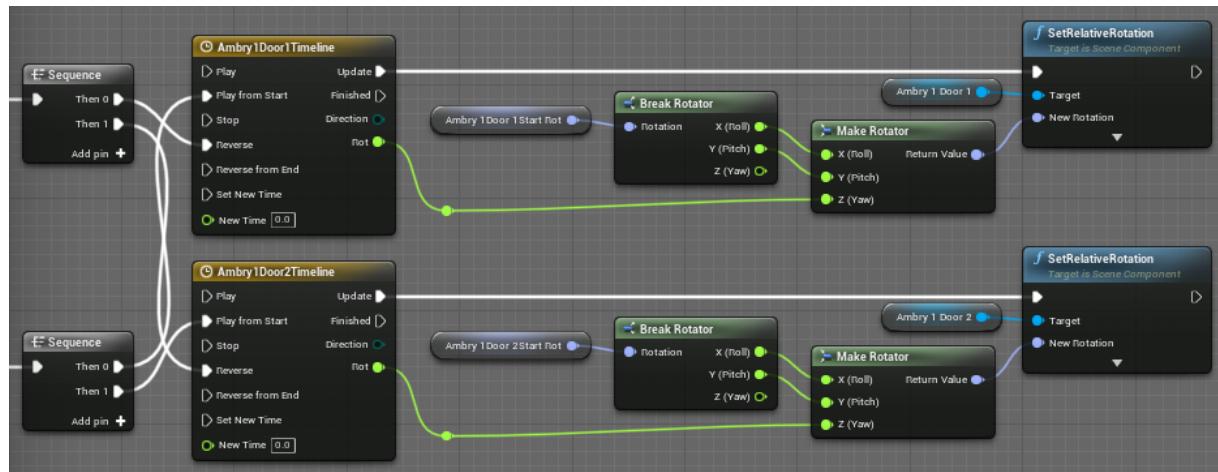


Abb. C.17: AlchemistsHouseBP: Open/Close Ambry1 Doors Part 2

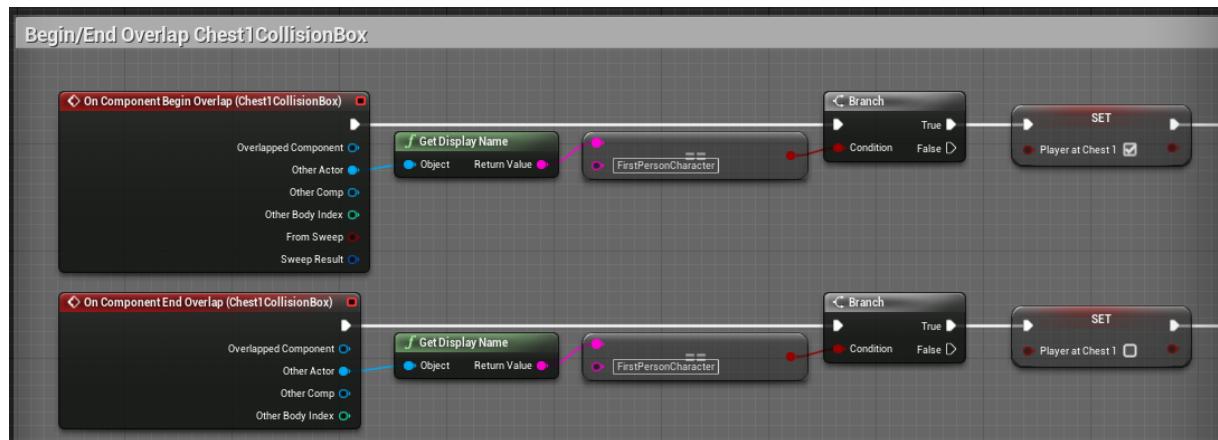
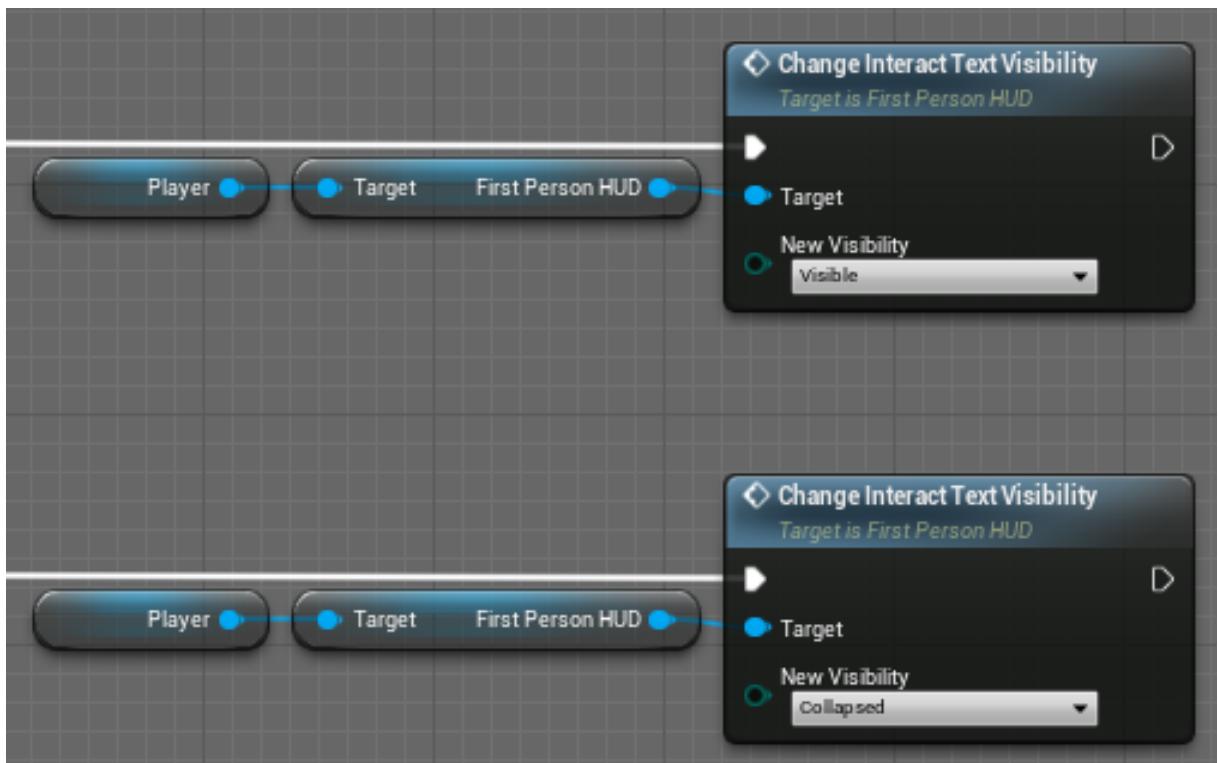
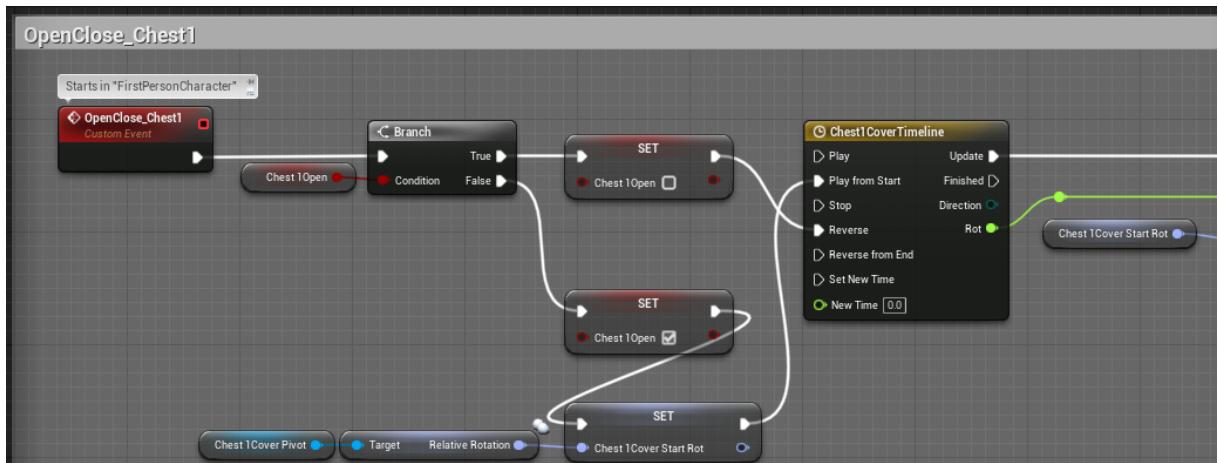


Abb. C.18: AlchemistsHouseBP: Begin/End Overlap Chest1 CollisionBox Part 1



**Abb. C.19:** AlchemistsHouseBP: Begin/End Overlap Chest1 CollisionBox Part 2



**Abb. C.20:** AlchemistsHouseBP: Open/Close Chest1 Part 1

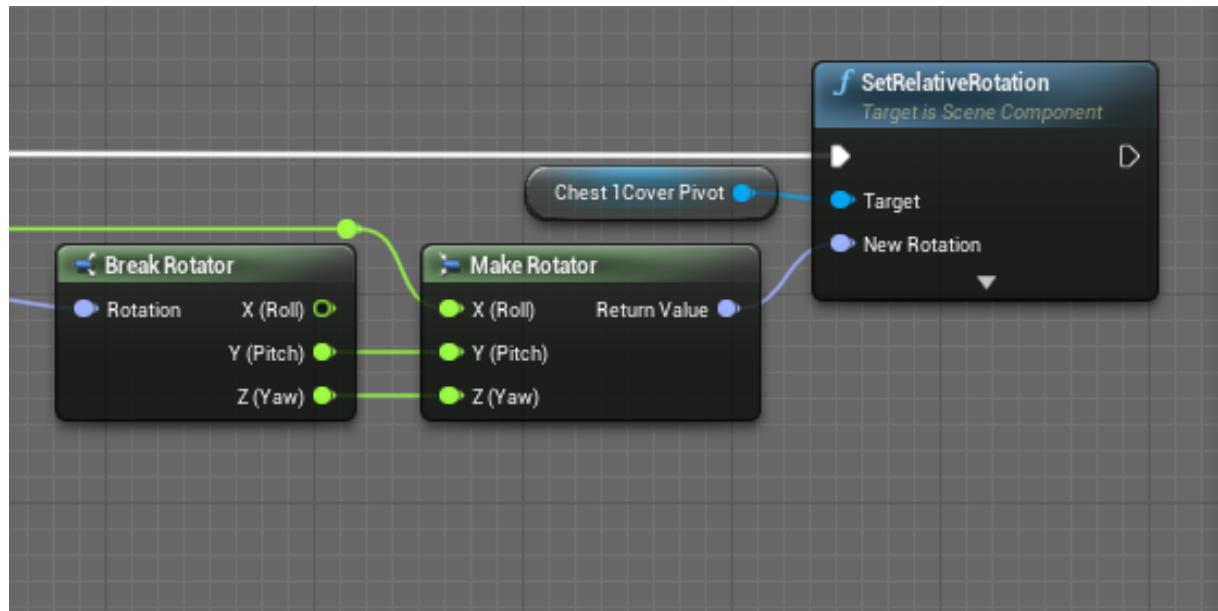


Abb. C.21: AlchemistsHouseBP: Open/Close Chest1 Part 2

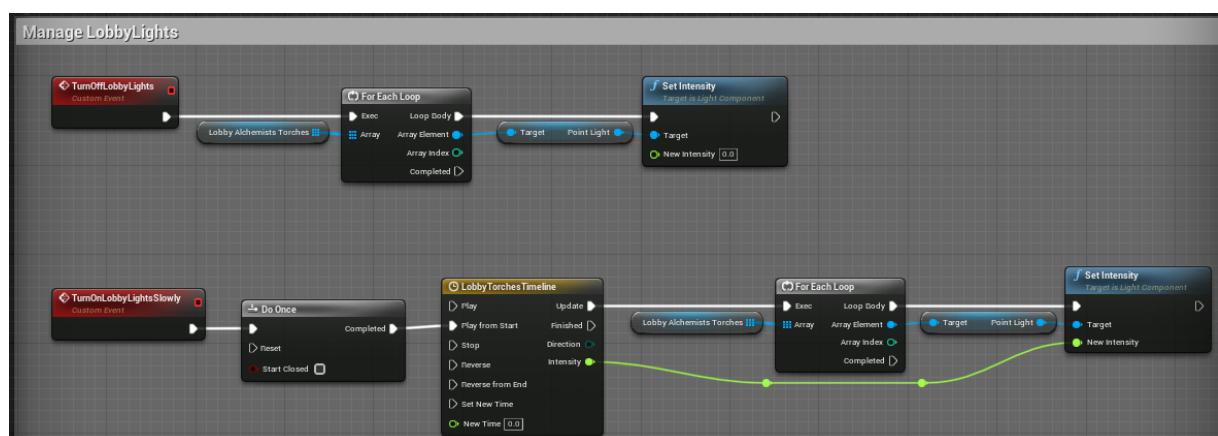


Abb. C.22: AlchemistsHouseBP: Manage Lobby Lights

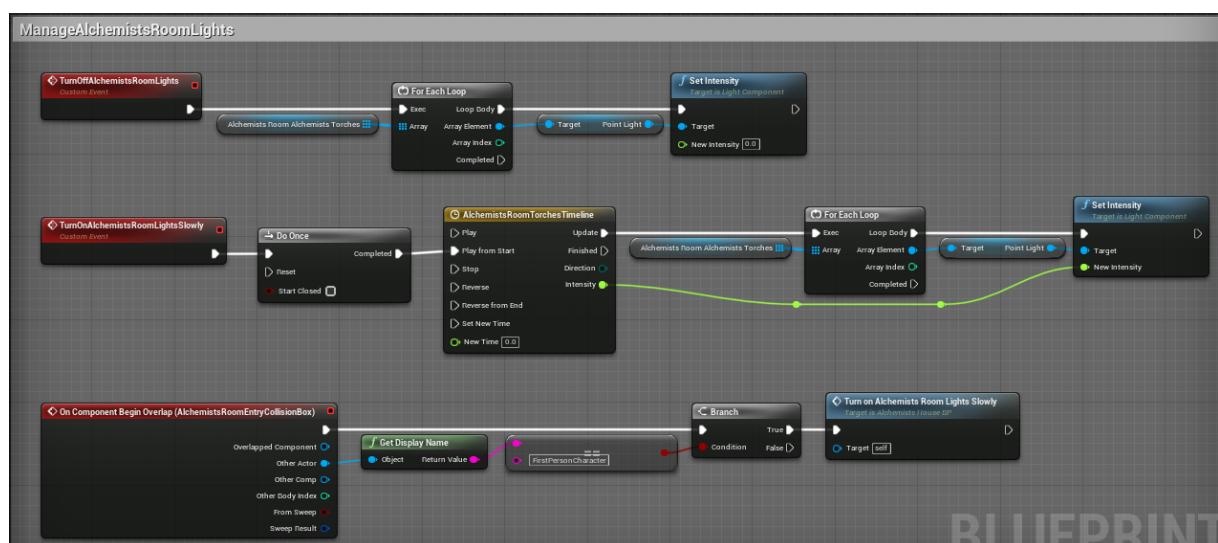


Abb. C.23: AlchemistsHouseBP: Manage Alchemists Room Lights

## D Anhang: CoinBP



**Abb. D.1:** CoinBP: Viewport

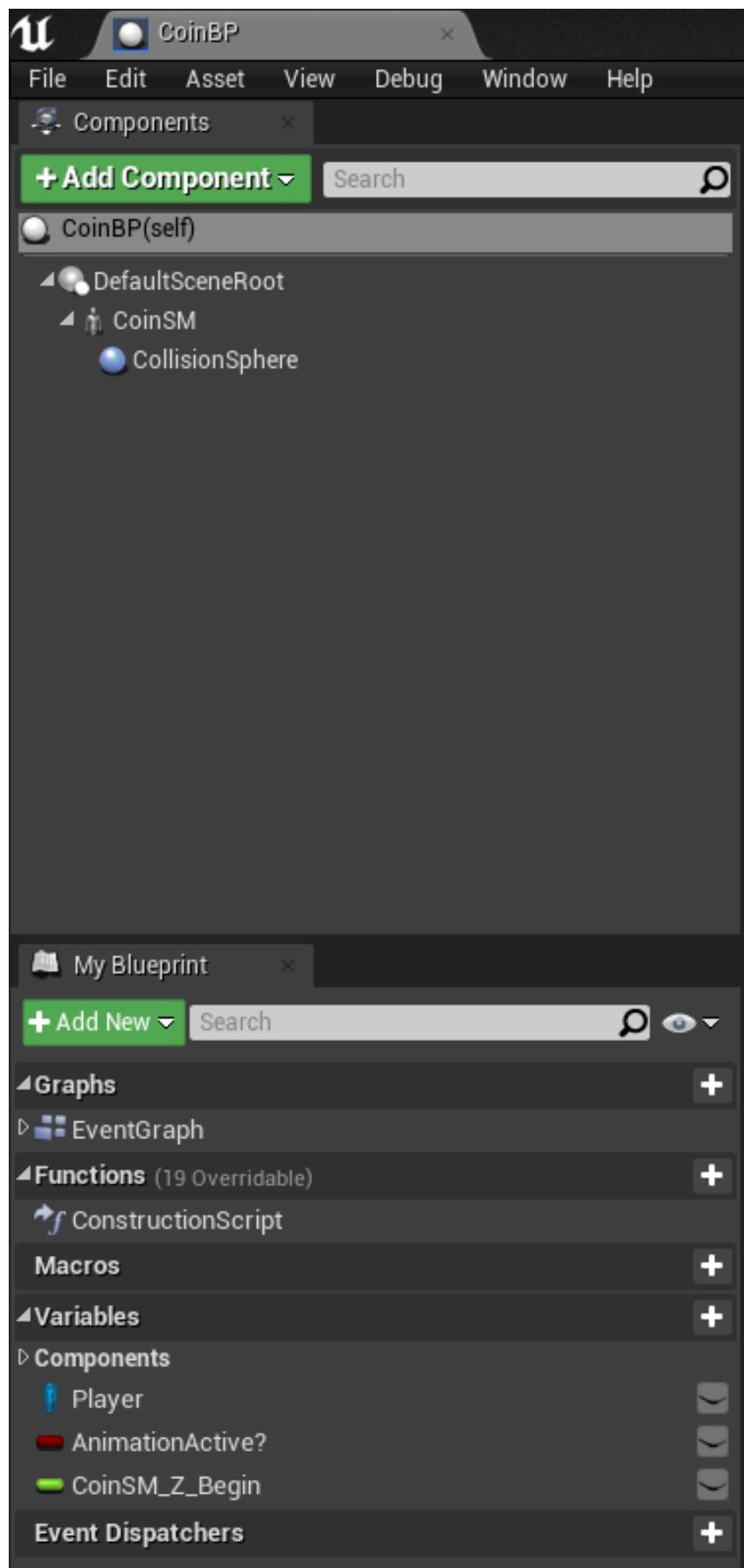


Abb. D.2: CoinBP: Components And Variables

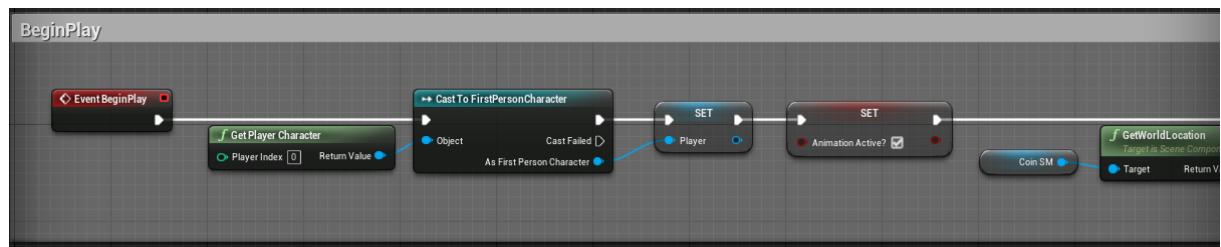


Abb. D.3: CoinBP: Begin Play Part 1



Abb. D.4: CoinBP: Begin Play Part 2

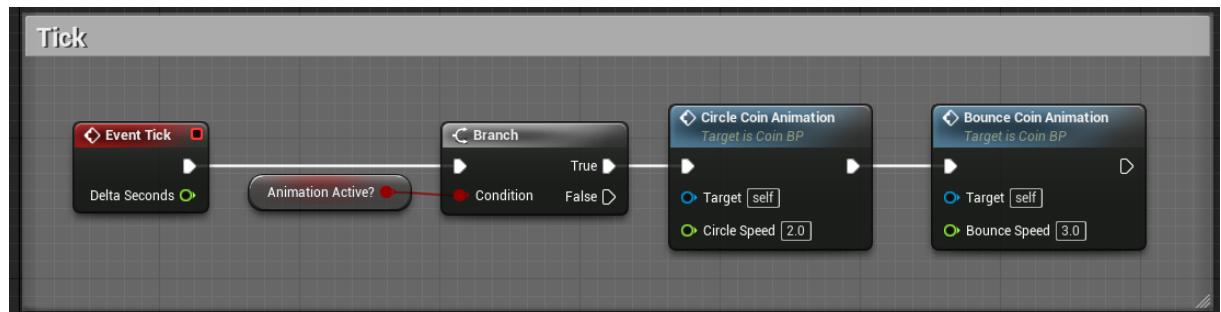


Abb. D.5: CoinBP: Tick

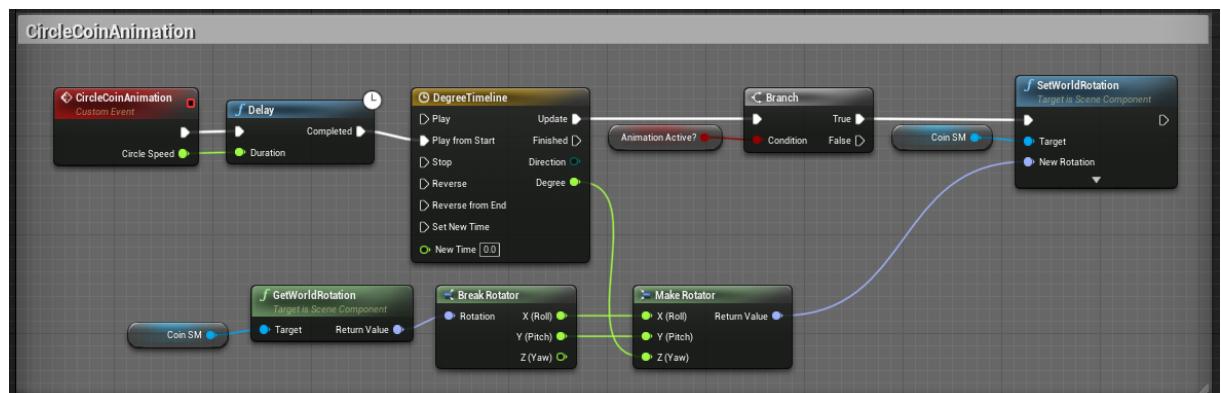


Abb. D.6: CoinBP: Circle Coin Animation Part 1

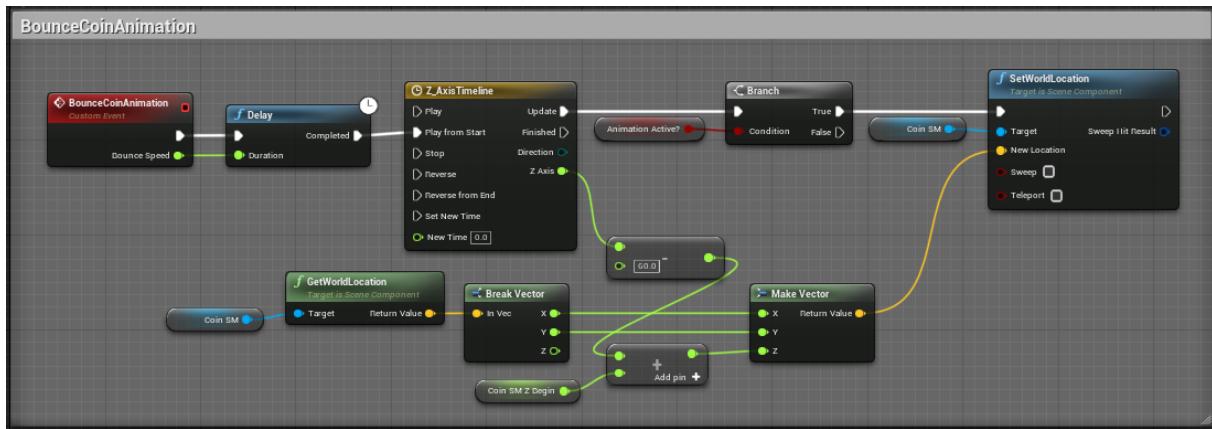


Abb. D.7: CoinBP: Circle Coin Animation Part 2

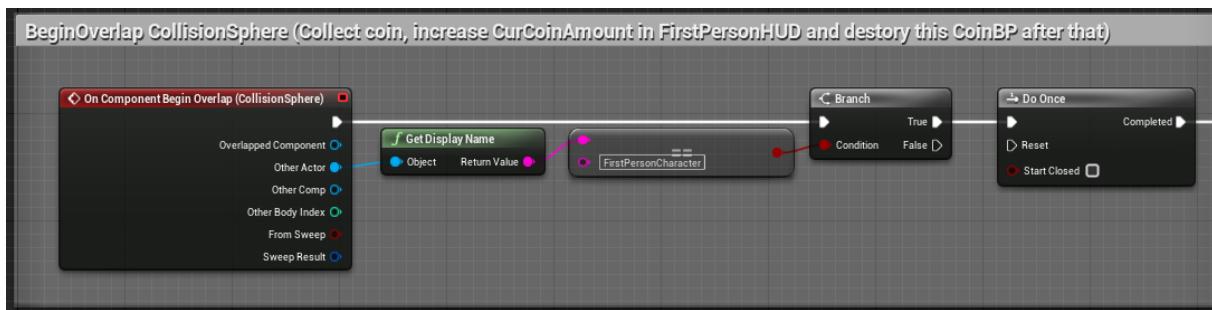


Abb. D.8: CoinBP: Begin Overlap Collision Sphere Part 1

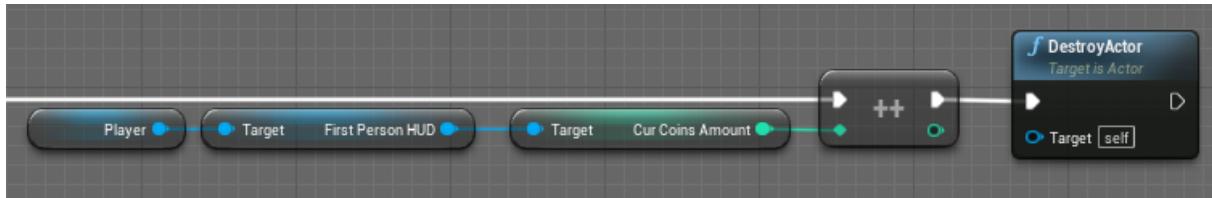
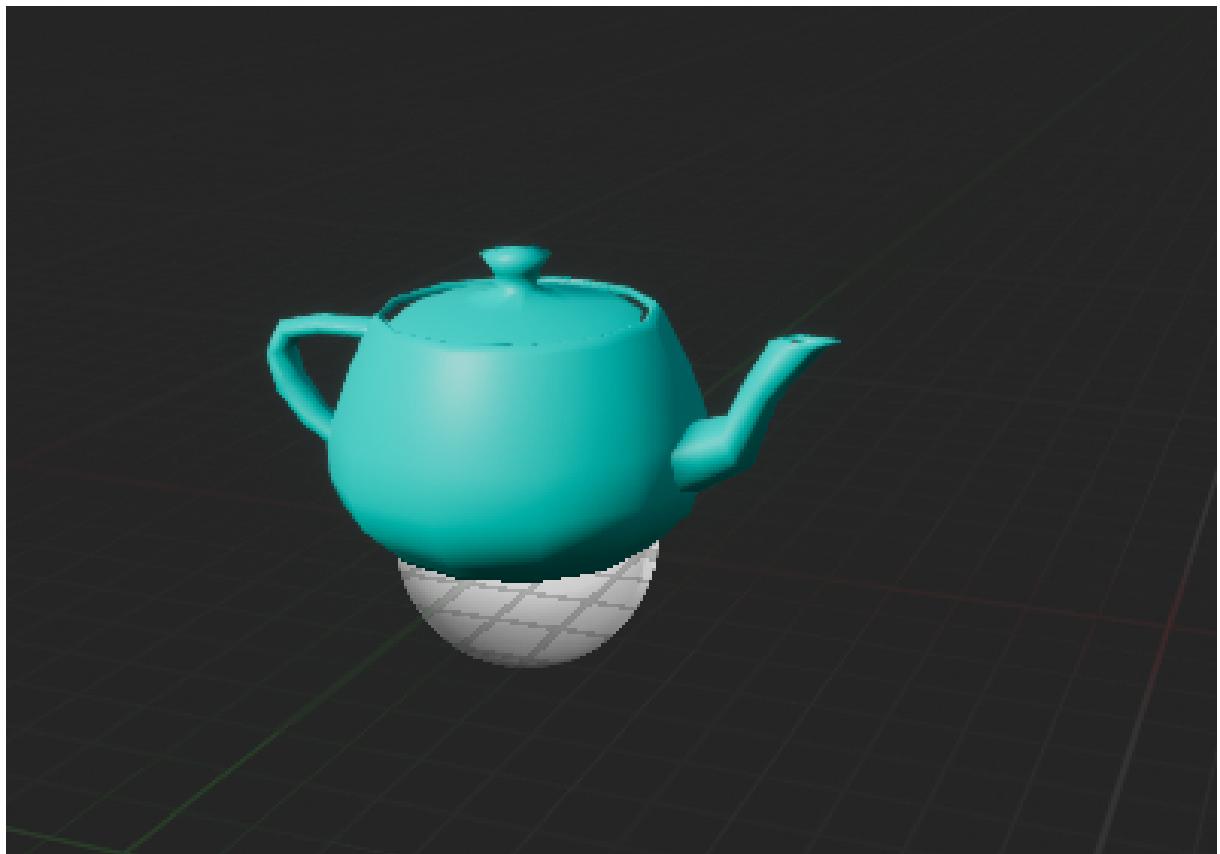
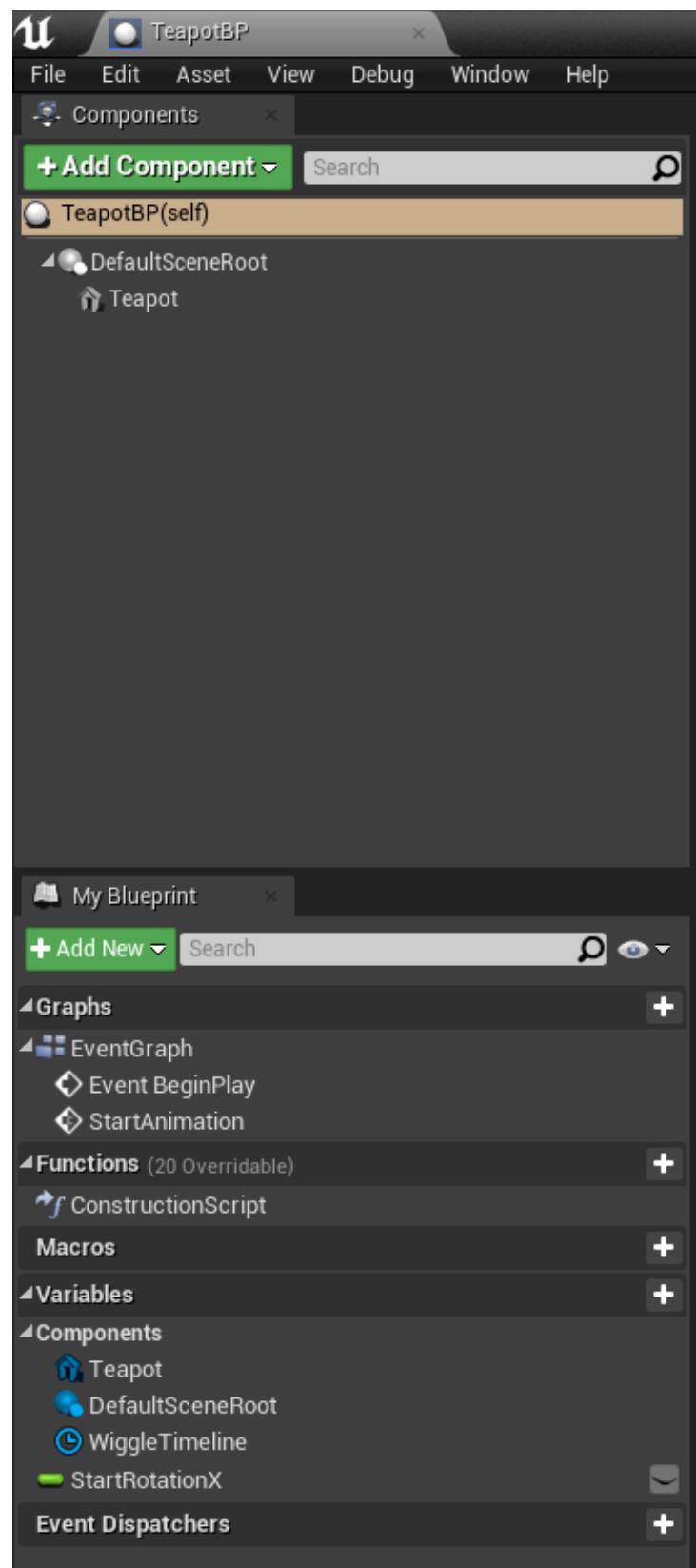


Abb. D.9: CoinBP: Begin Overlap Collision Sphere Part 2

## E Anhang: TeapotBP



**Abb. E.1:** TeapotBP: Viewport



**Abb. E.2:** TeapotBP: Components And Variables

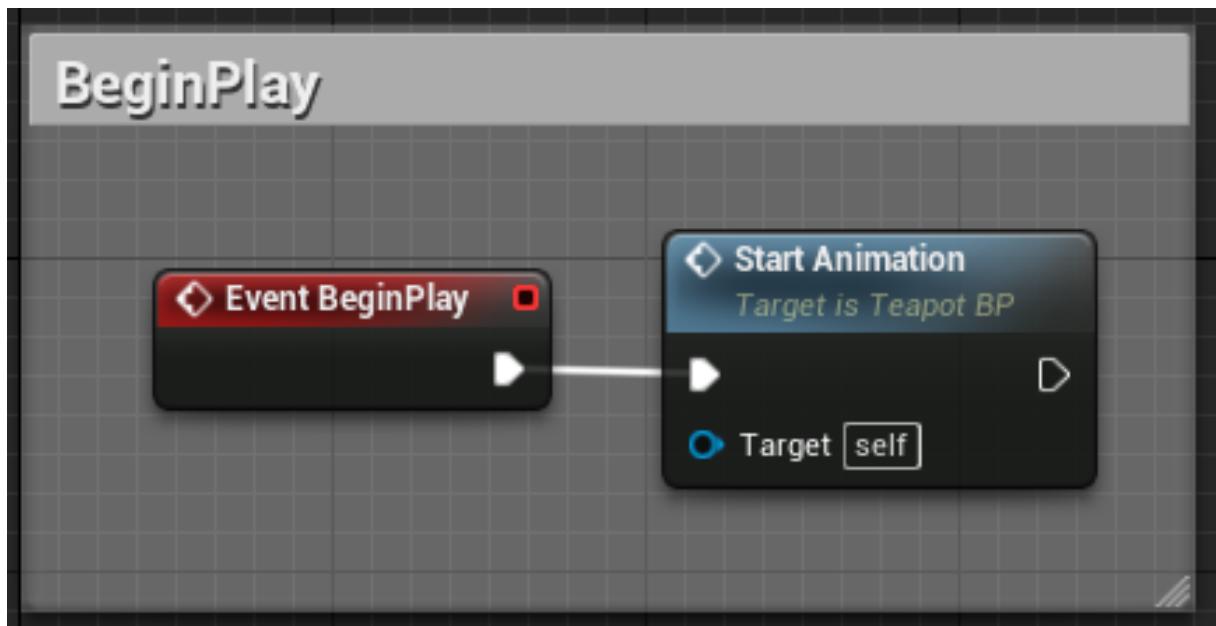


Abb. E.3: TeapotBP: Begin Play

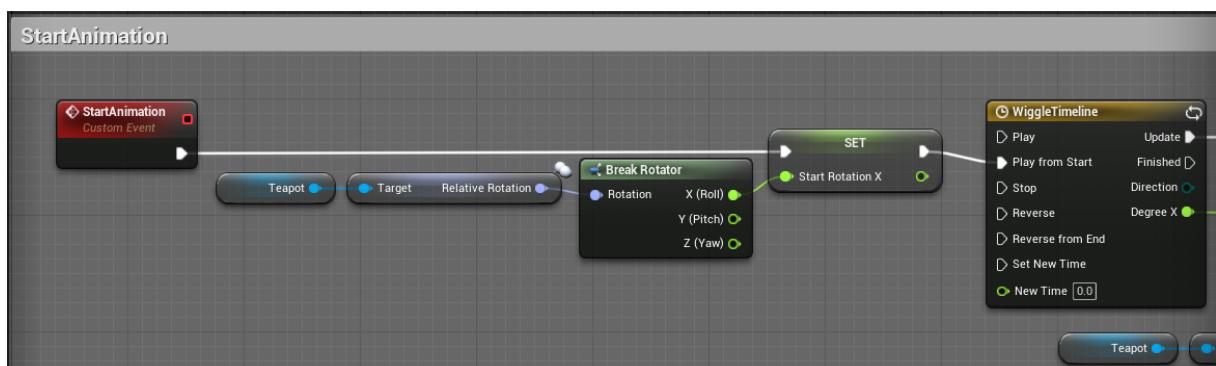


Abb. E.4: TeapotBP: Start Animation Part 1

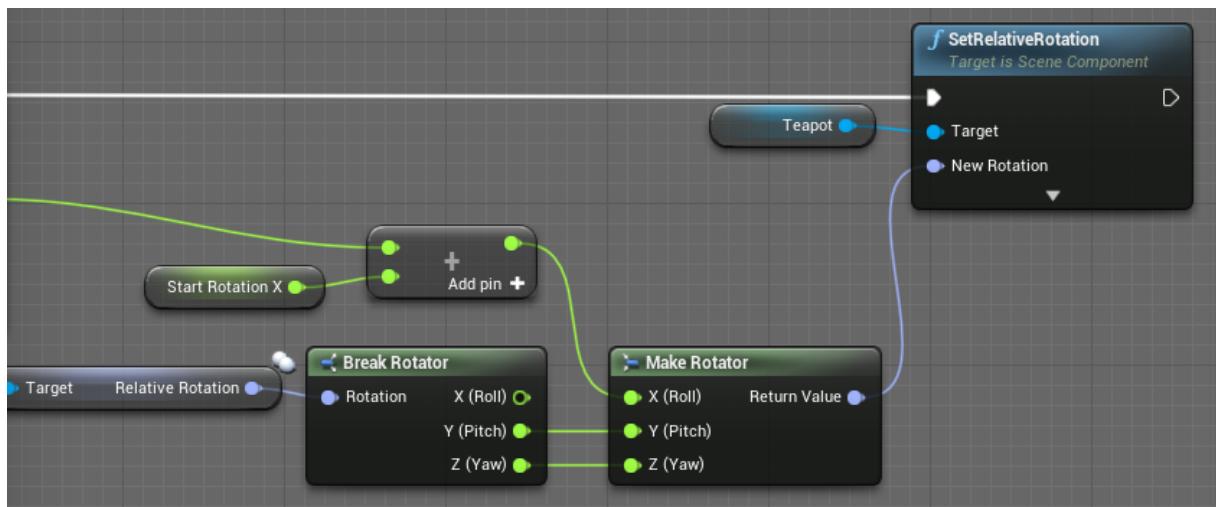


Abb. E.5: TeapotBP: Start Animation Part 2

## F Anhang: FirstPersonCharacter

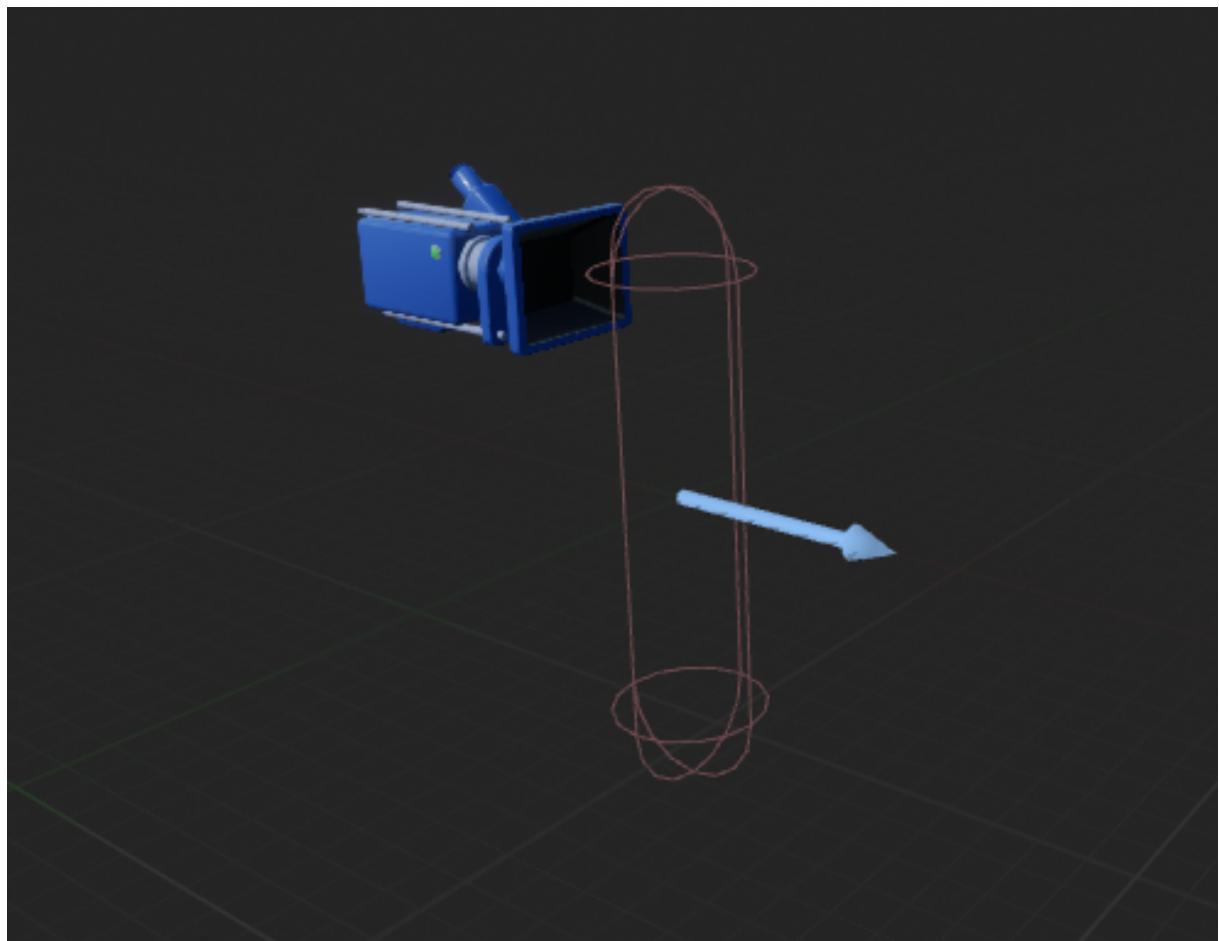


Abb. F.1: FirstPersonCharacter: Viewport

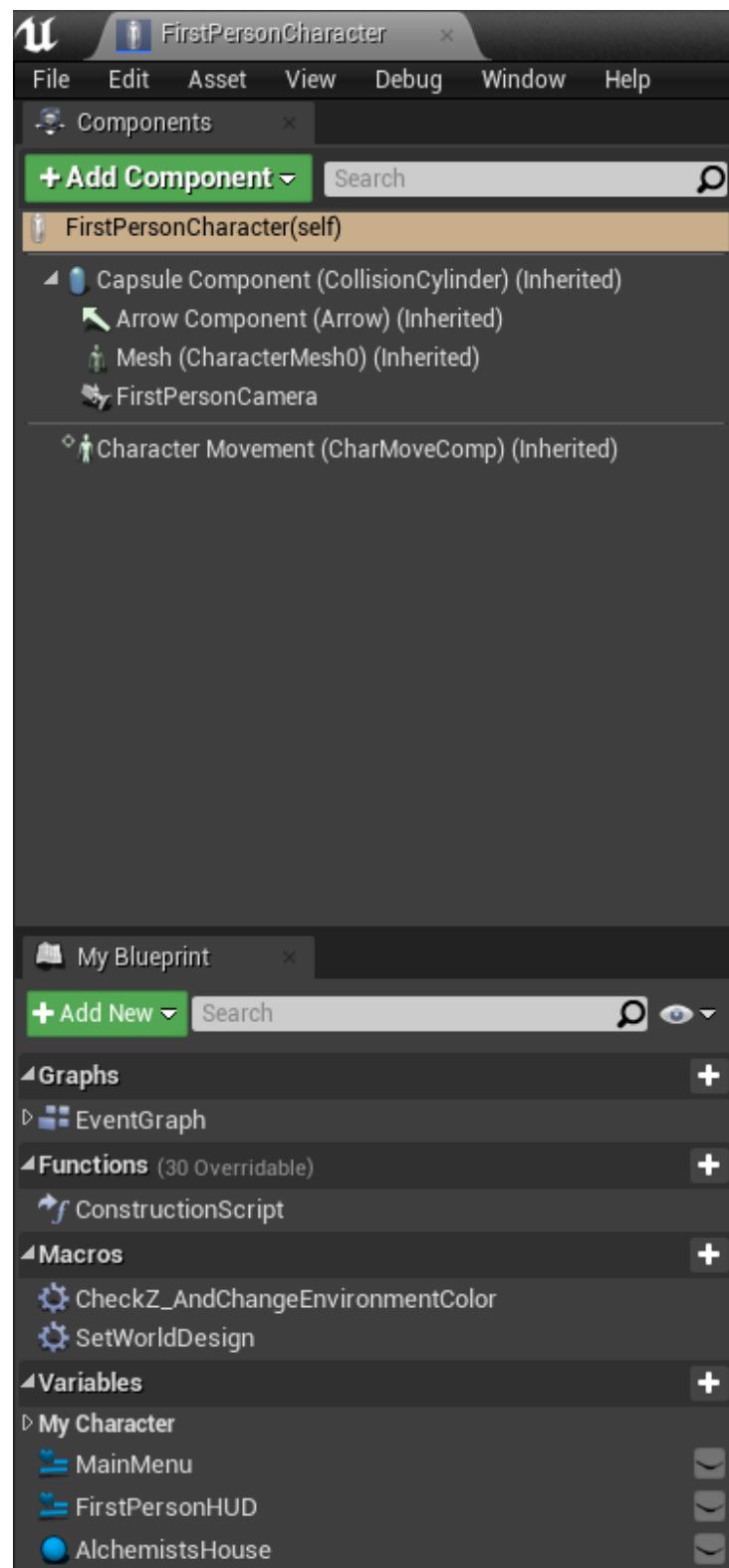


Abb. F.2: FirstPersonCharacter: Components And Variables

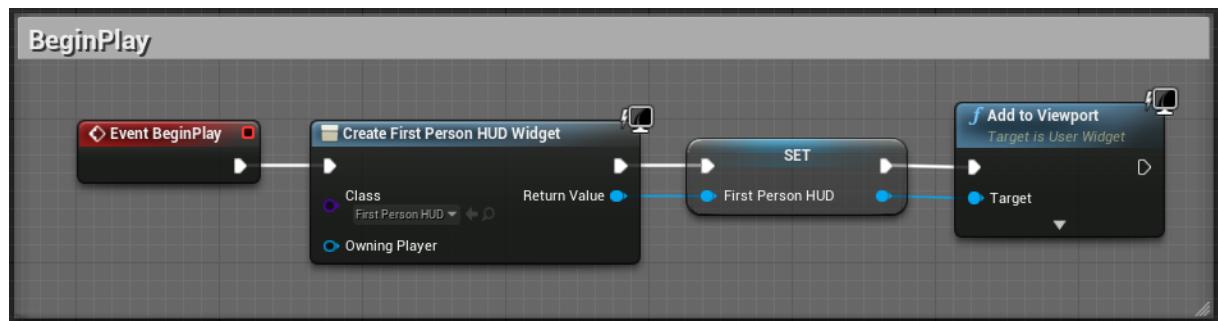


Abb. F.3: FirstPersonCharacter: Begin Play

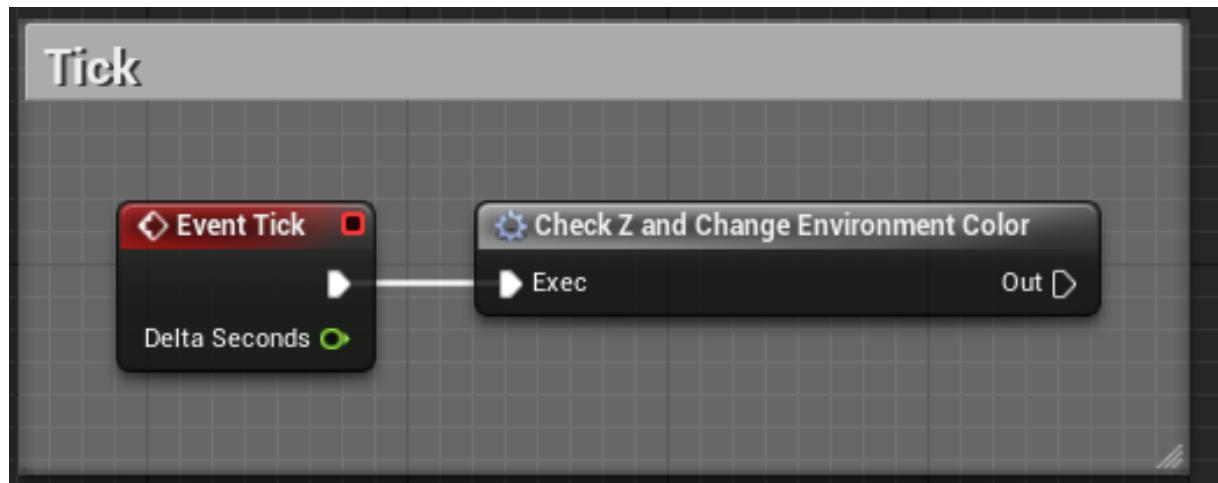


Abb. F.4: FirstPersonCharacter: Tick Part 1

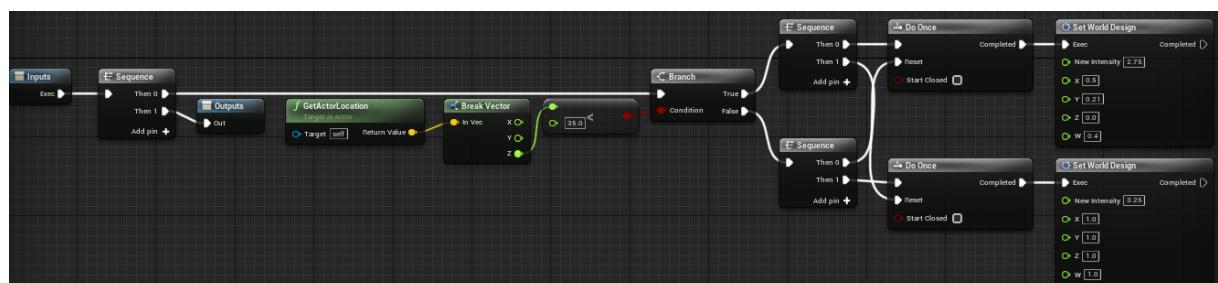


Abb. F.5: FirstPersonCharacter: Tick Part 2 - Check Z And Change Environment Color

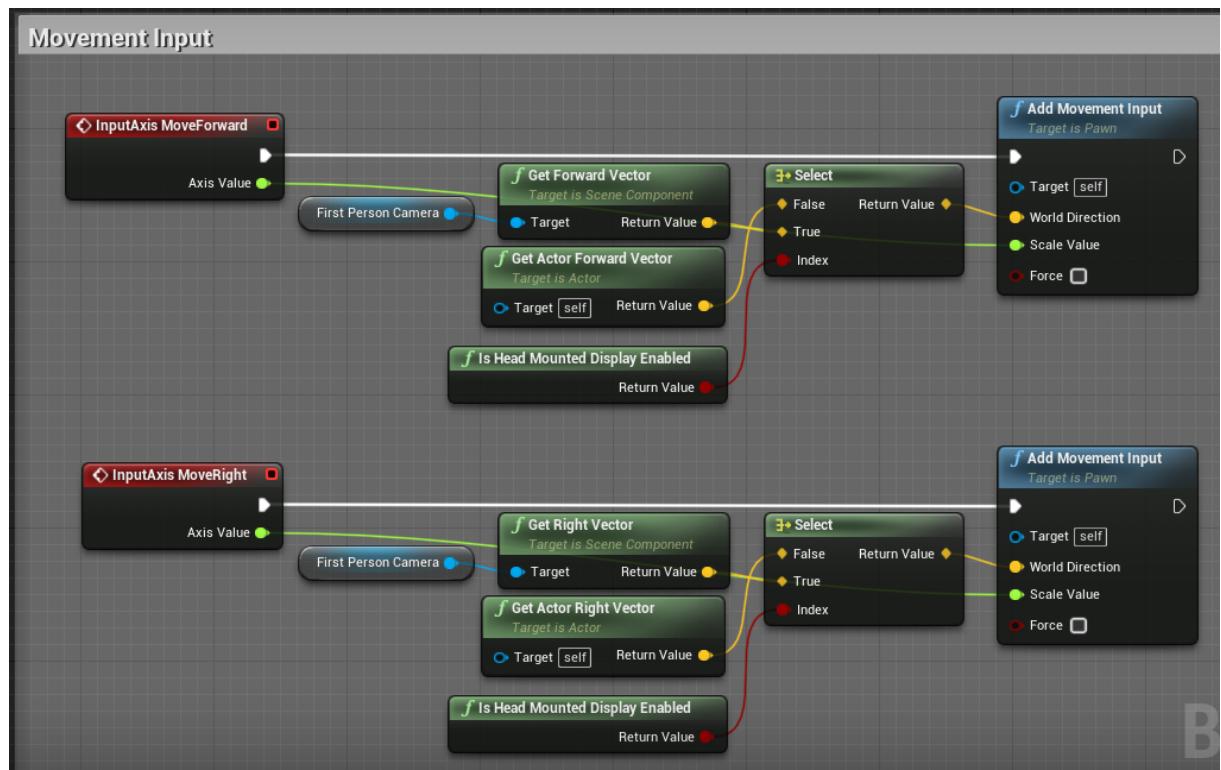


Abb. F.6: FirstPersonCharacter: Movement Input

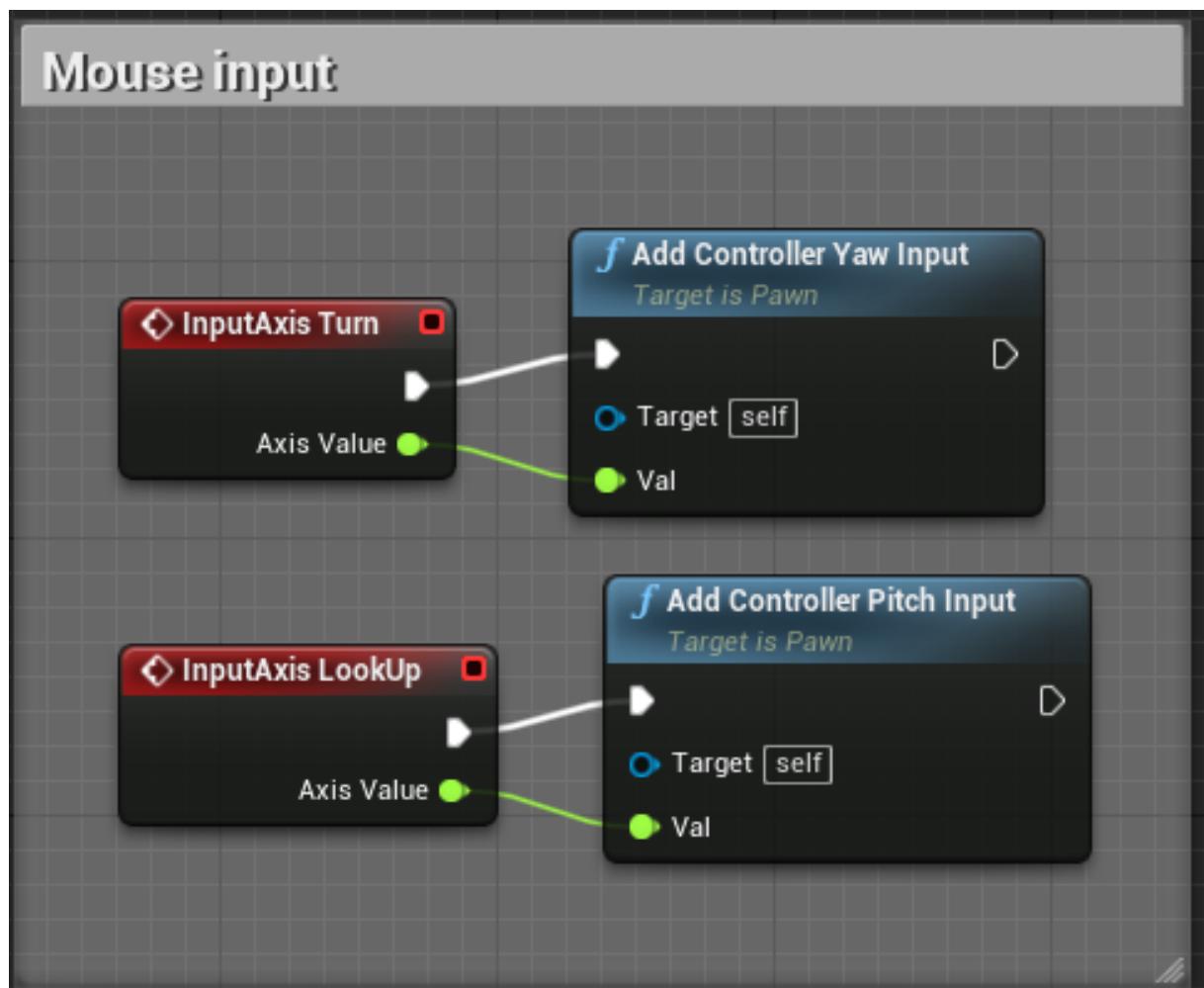


Abb. F.7: FirstPersonCharacter: Mouse Input

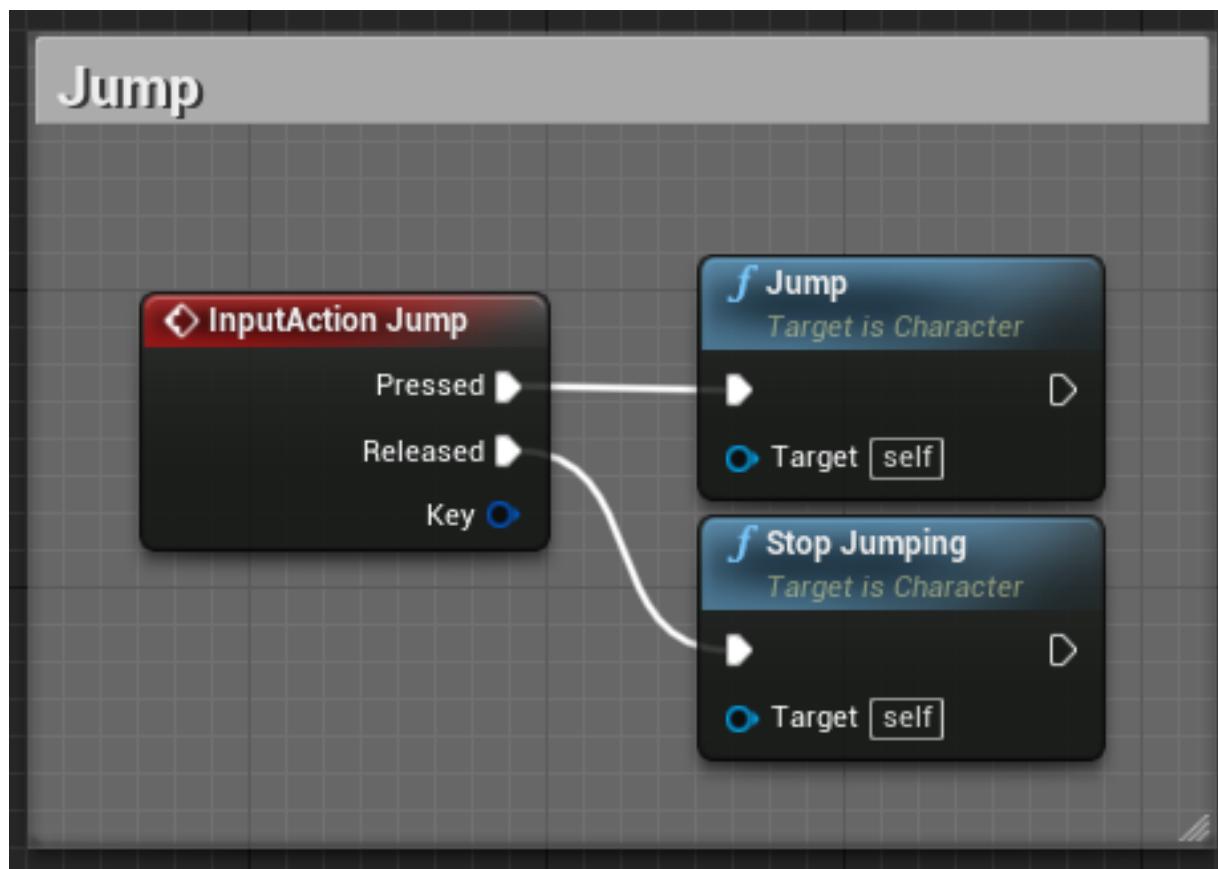


Abb. F.8: FirstPersonCharacter: Jump

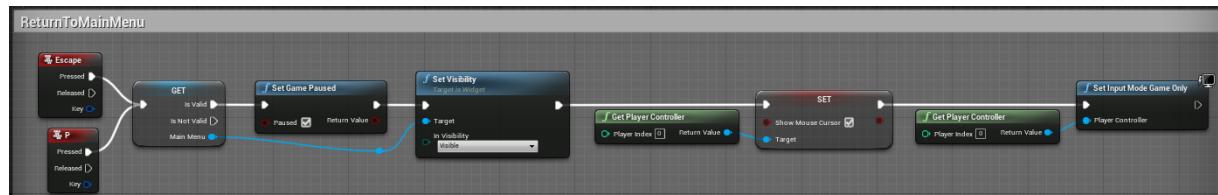


Abb. F.9: FirstPersonCharacter: Return To MainMenu

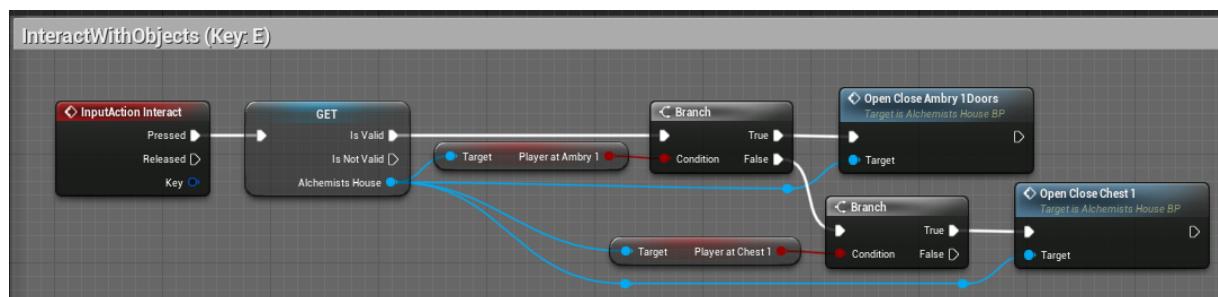


Abb. F.10: FirstPersonCharacter: Interact With Objects

## G Anhang: FirstPersonHUD

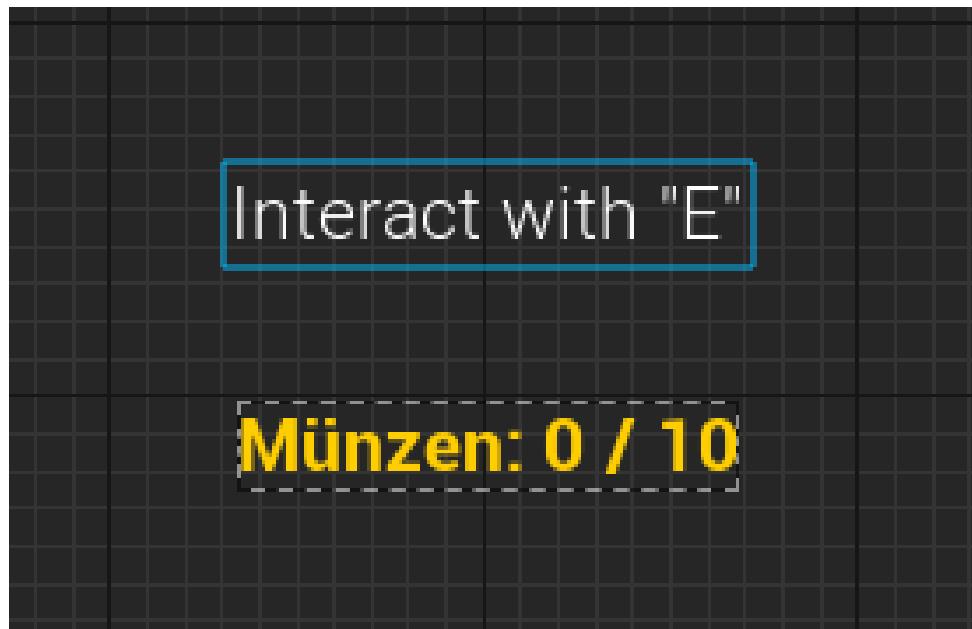


Abb. G.1: FirstPersonHUD Widget: Designer

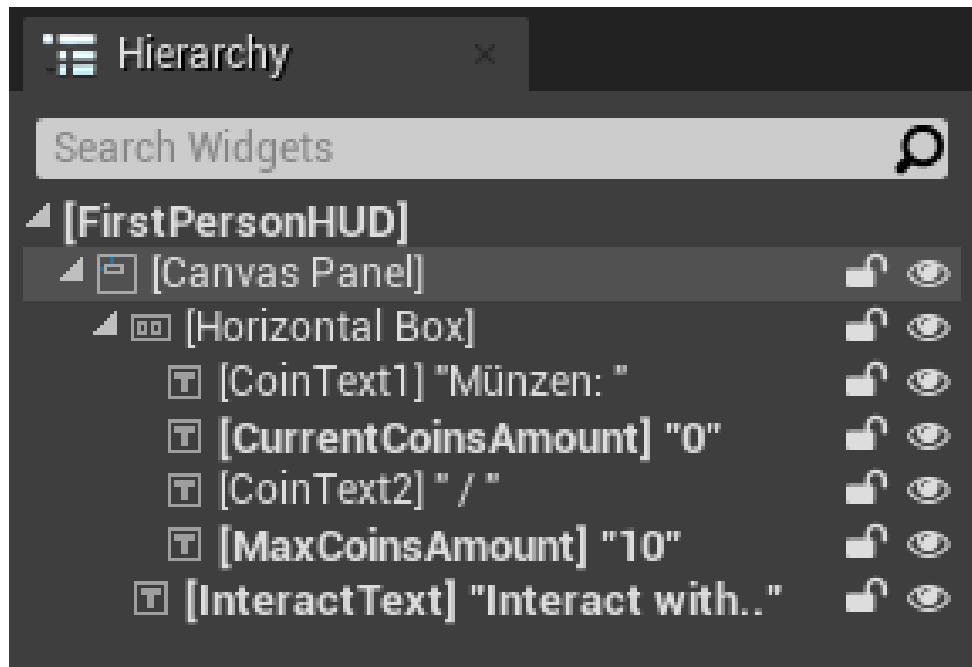


Abb. G.2: FirstPersonHUD Widget: Hierarchy

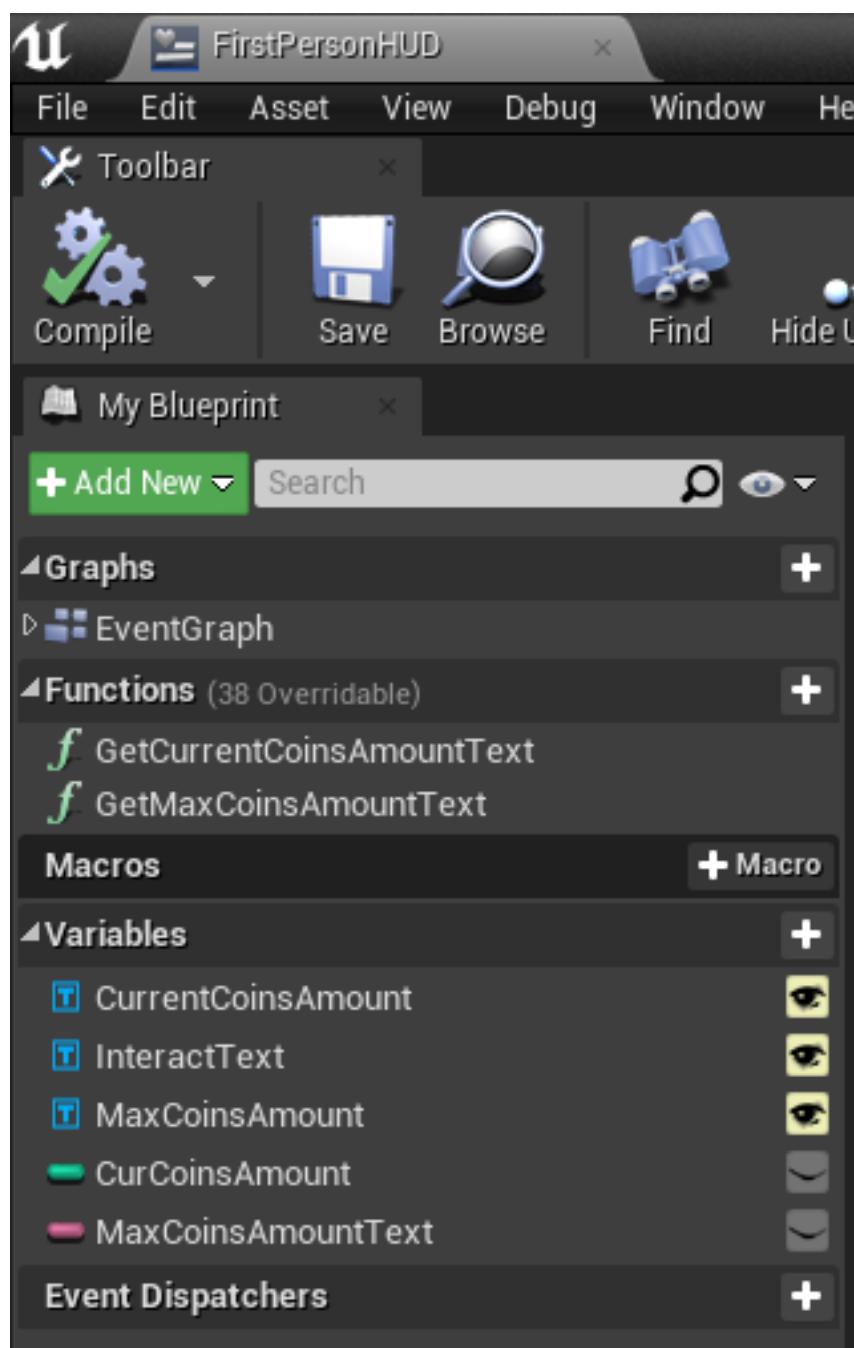


Abb. G.3: FirstPersonHUD Widget: Functions And Variables

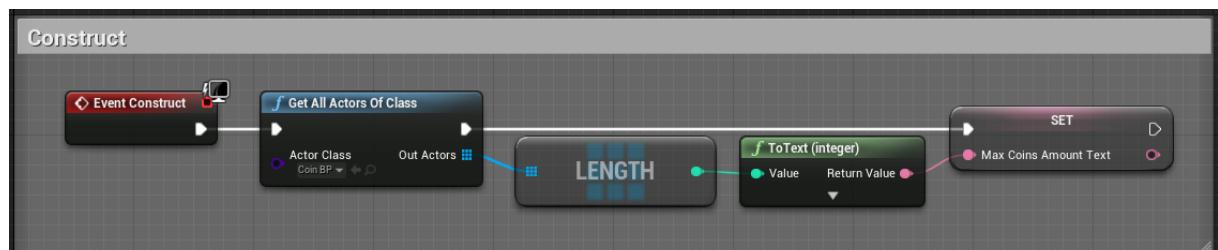


Abb. G.4: FirstPersonHUD Widget: Construct

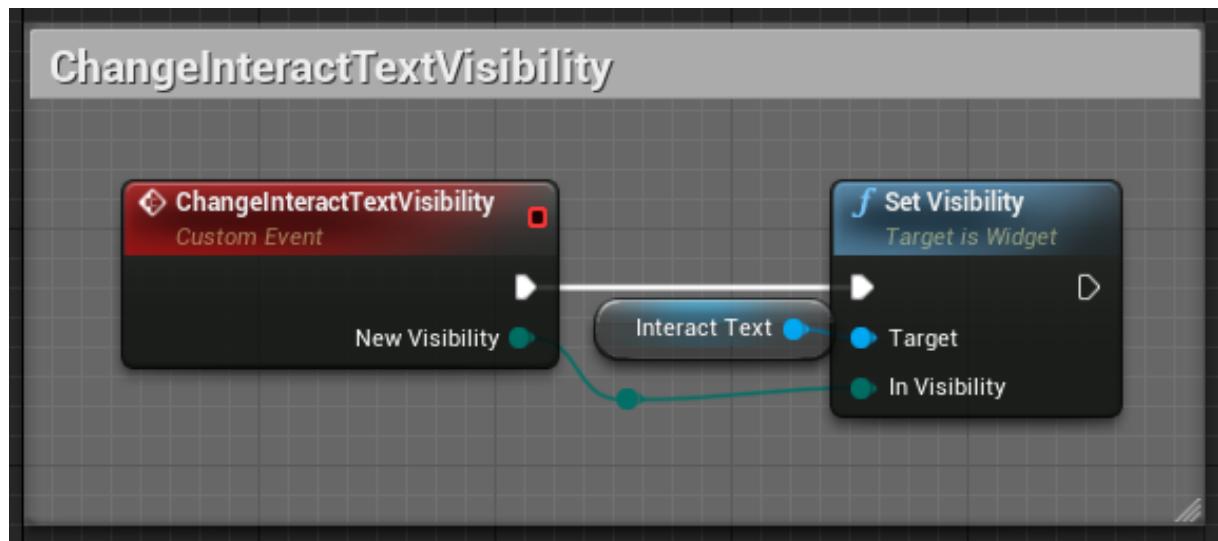


Abb. G.5: FirstPersonHUD Widget: Change Interact Text Visibility

## H Anhang: Animals

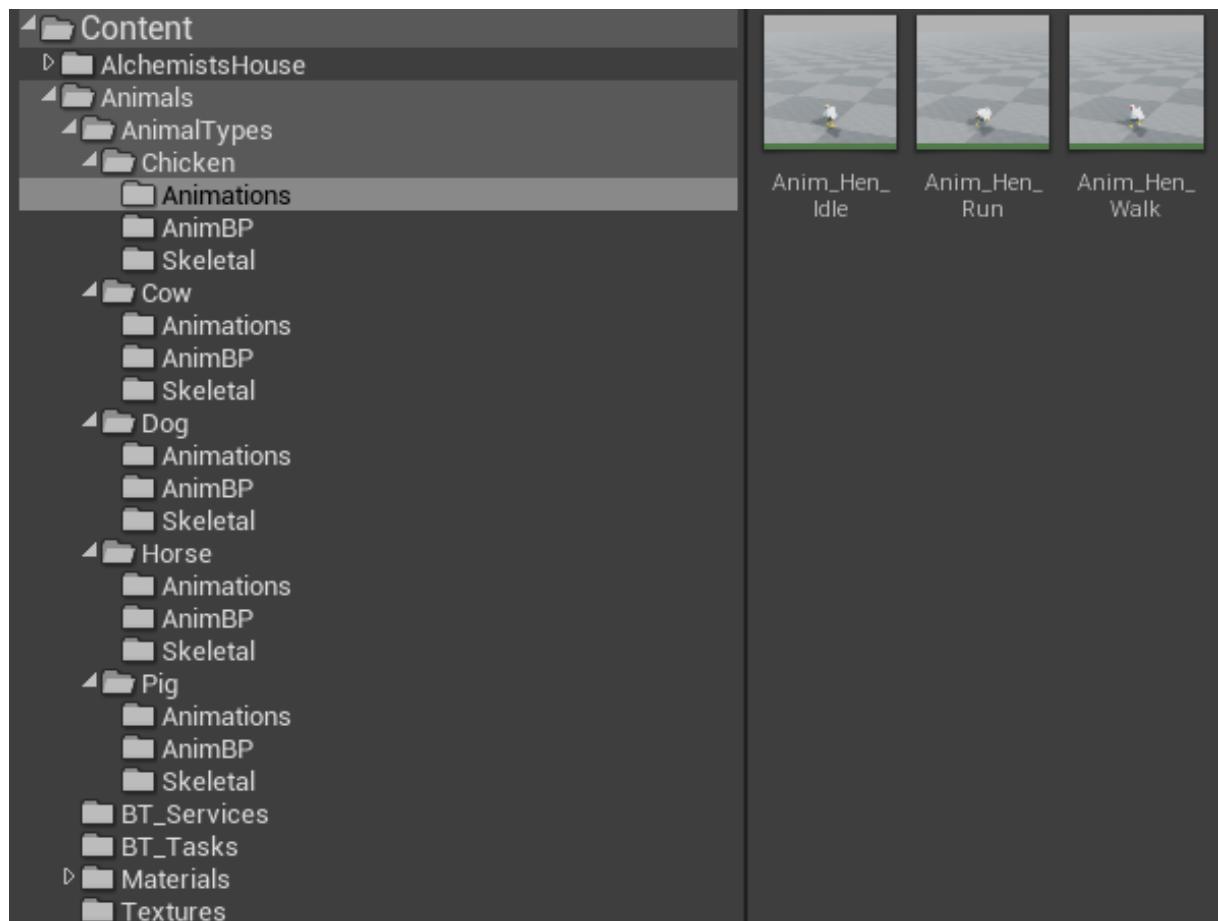


Abb. H.1: Animals: Chicken Folder Animations



Abb. H.2: Animals: Chicken Folder AnimBP

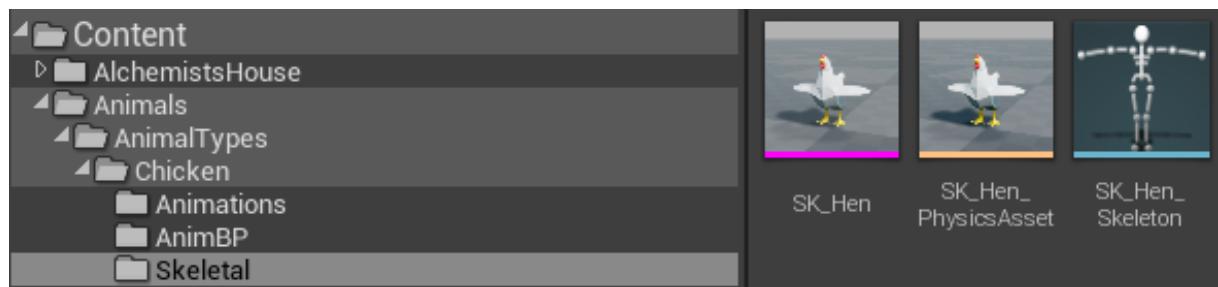


Abb. H.3: Animals: Chicken Folder Skeletal

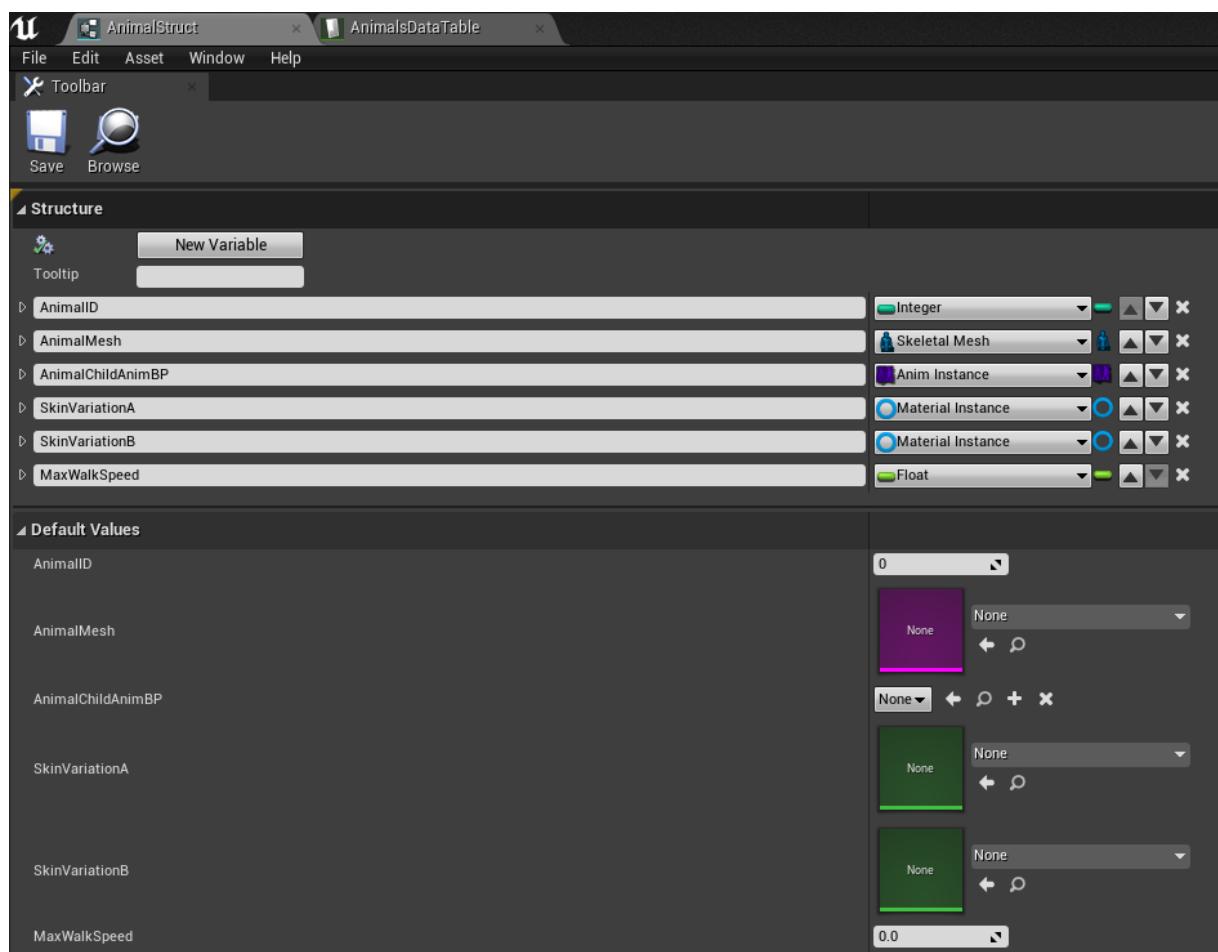
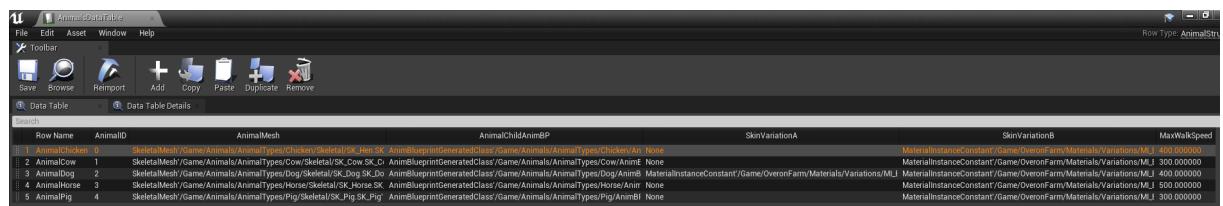
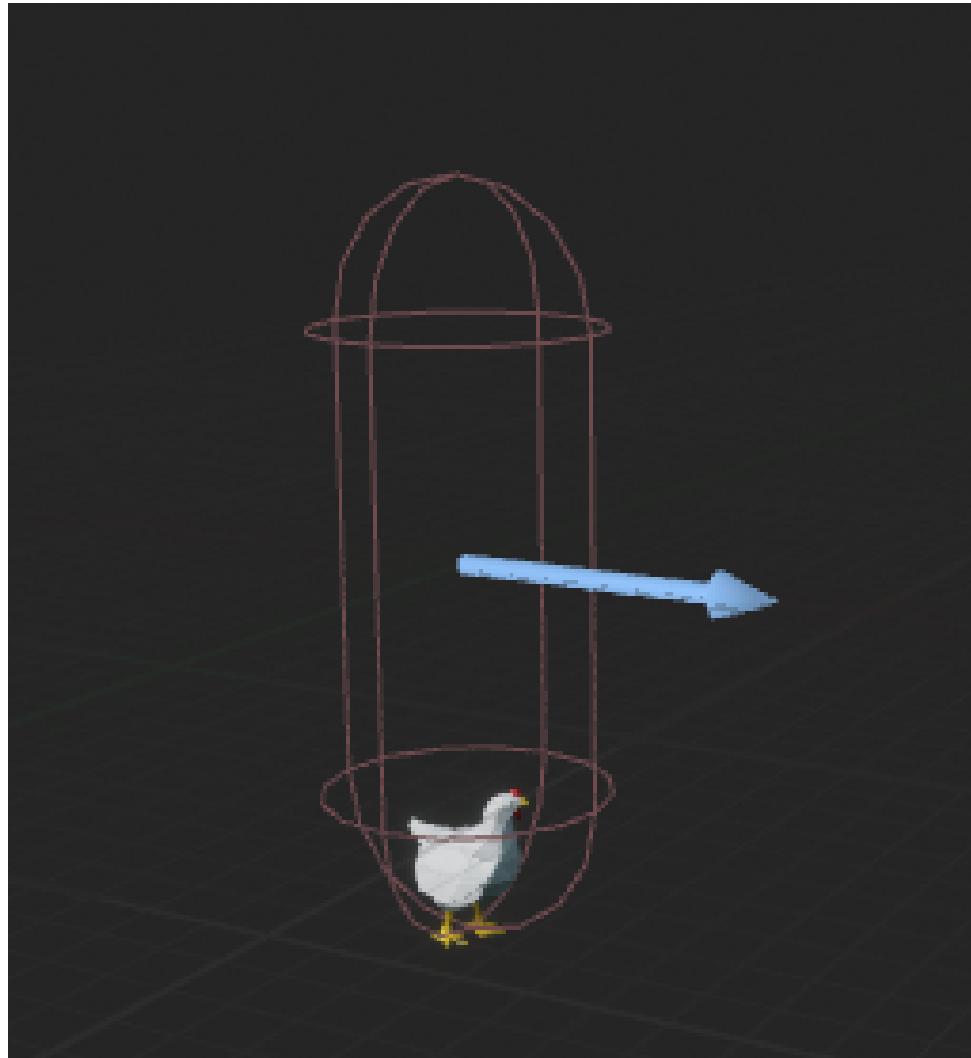


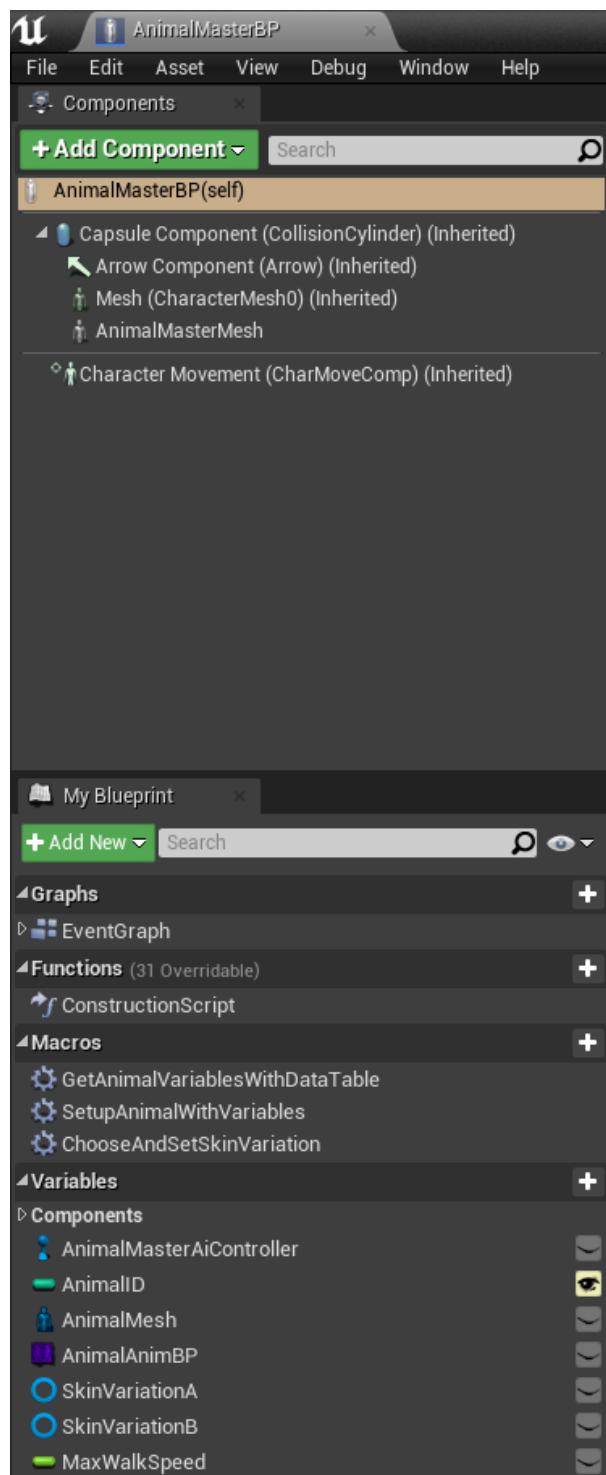
Abb. H.4: Animals: AnimalStruct



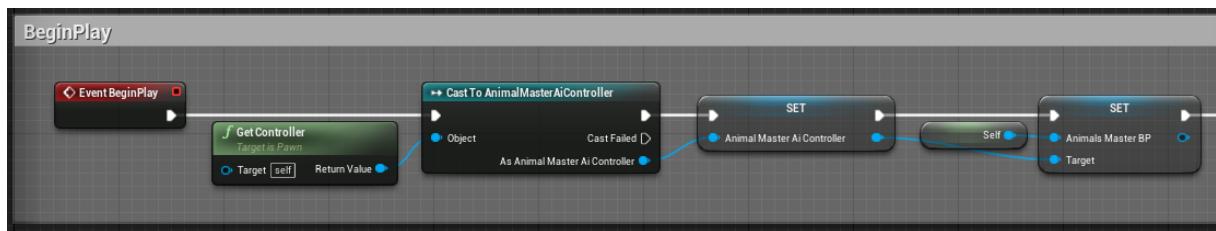
The screenshot shows the Unreal Engine's Data Table Editor window. The table has columns for Row Name, AnimalID, AnimalMesh, AnimalChildAnimBP, SkinVariationA, SkinVariationB, and MaxWalkSpeed. The data rows are:

Row Name	AnimalID	AnimalMesh	AnimalChildAnimBP	SkinVariationA	SkinVariationB	MaxWalkSpeed
1	AnimalChicken	0	SkinnedMesh'/Game/Animals/AnimalTypes/Chicken/Skeletal/SC_Cow_SK_0_AnimBlueprintGeneratedClass'/Game/Animals/AnimalTypes/Chicken/Mesh/None	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_300.000000	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_400.000000	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_500.000000
2	AnimalCow	1	SkinnedMesh'/Game/Animals/AnimalTypes/Cow/Skeletal/SC_Cow_SK_0_AnimBlueprintGeneratedClass'/Game/Animals/AnimalTypes/Cow/Mesh/None	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_300.000000	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_400.000000	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_500.000000
3	AnimalDog	2	SkinnedMesh'/Game/Animals/AnimalTypes/Dog/Skeletal/SC_Dog_SK_0_AnimBlueprintGeneratedClass'/Game/Animals/AnimalTypes/Dog/Mesh/None	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_300.000000	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_400.000000	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_500.000000
4	AnimalHorse	3	SkinnedMesh'/Game/Animals/AnimalTypes/Horse/Skeletal/SC_Horse_SK_Pig_AnimBlueprintGeneratedClass'/Game/Animals/AnimalTypes/Horse/Mesh/None	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_300.000000	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_400.000000	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_500.000000
5	AnimalPig	4	SkinnedMesh'/Game/Animals/AnimalTypes/Pig/Skeletal/SC_Pig_SK_Pig_AnimBlueprintGeneratedClass'/Game/Animals/AnimalTypes/Pig/Mesh/None	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_300.000000	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_400.000000	MaterialInstanceConstant'/Game/OvenFarm/Materials/Variations/MI_J_500.000000

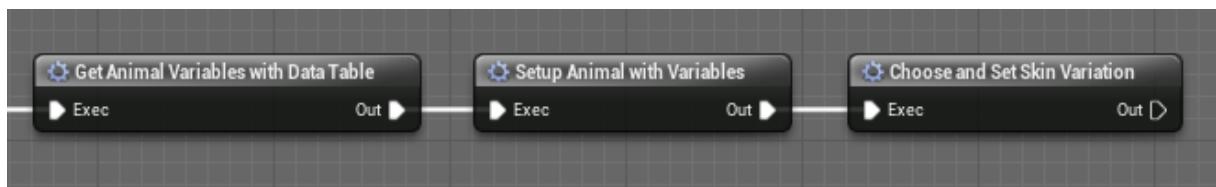
**Abb. H.5:** Animals: DataTable**Abb. H.6:** Animals: AnimalsMasterBP - Viewport



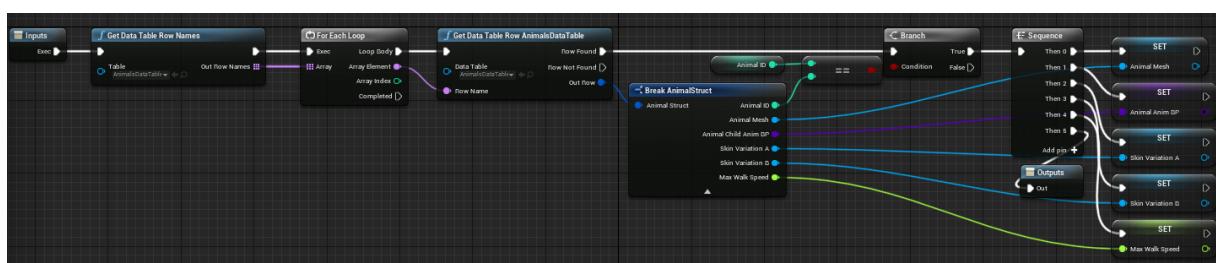
**Abb. H.7:** Animals: AnimalsMasterBP - Components And Variables



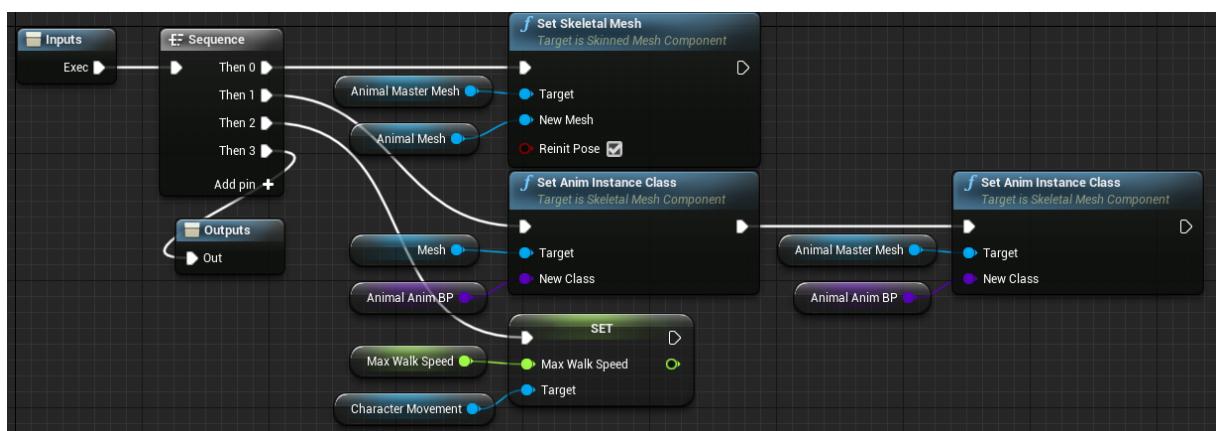
**Abb. H.8:** Animals: AnimalsMasterBP - Begin Play Part 1



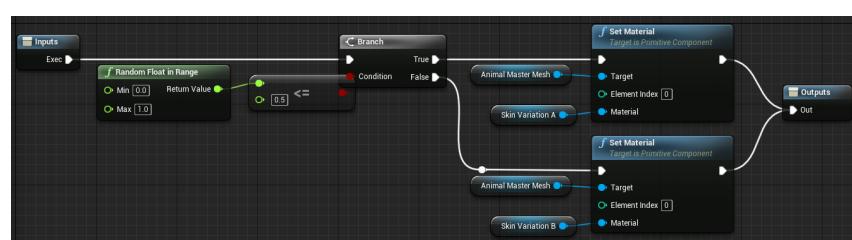
**Abb. H.9:** Animals: AnimalsMasterBP - Begin Play Part 2



**Abb. H.10:** Animals: AnimalsMasterBP - Get Animal Variables With DataTable



**Abb. H.11:** Animals: AnimalsMasterBP - Setup Animal With Variables



**Abb. H.12:** Animals: AnimalsMasterBP - Choose And Set Skin Variation

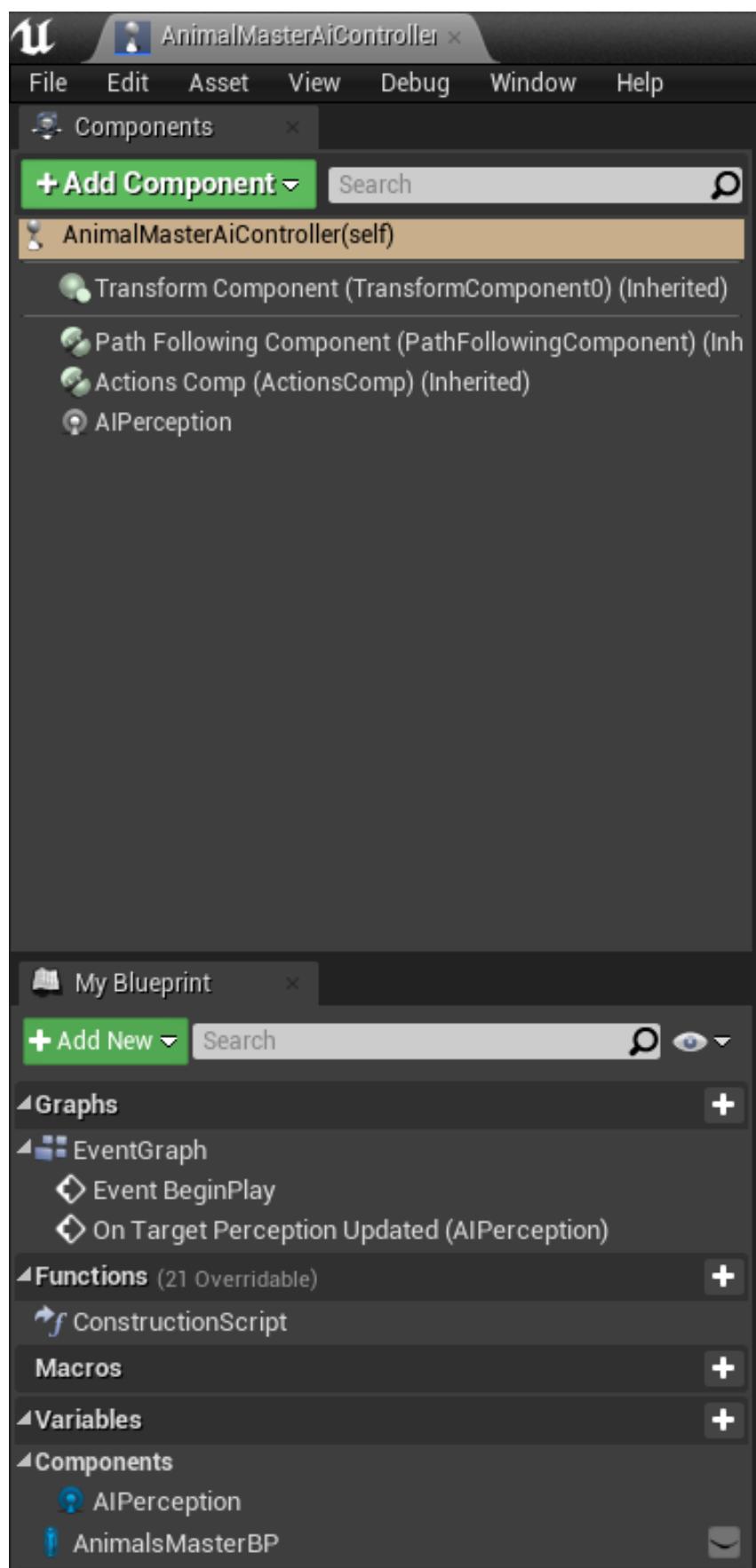
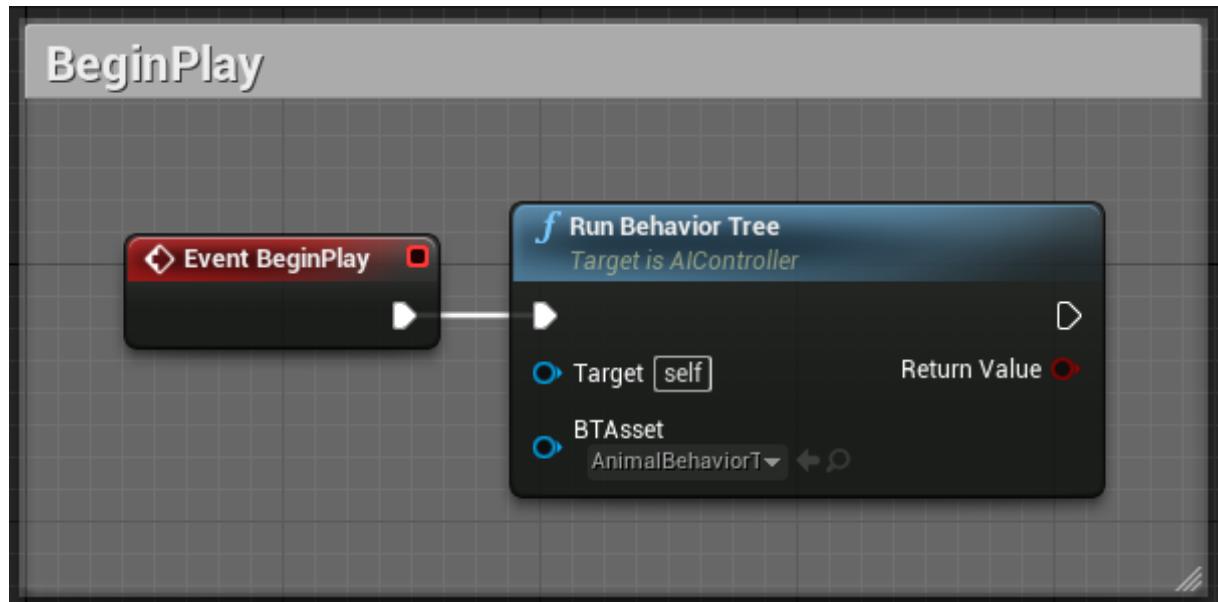
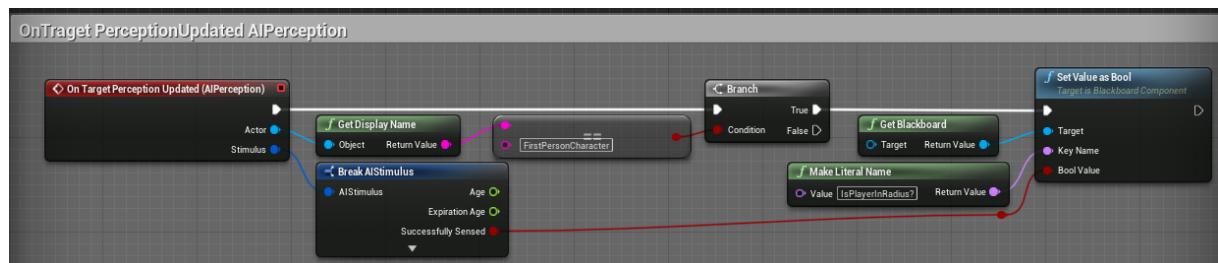


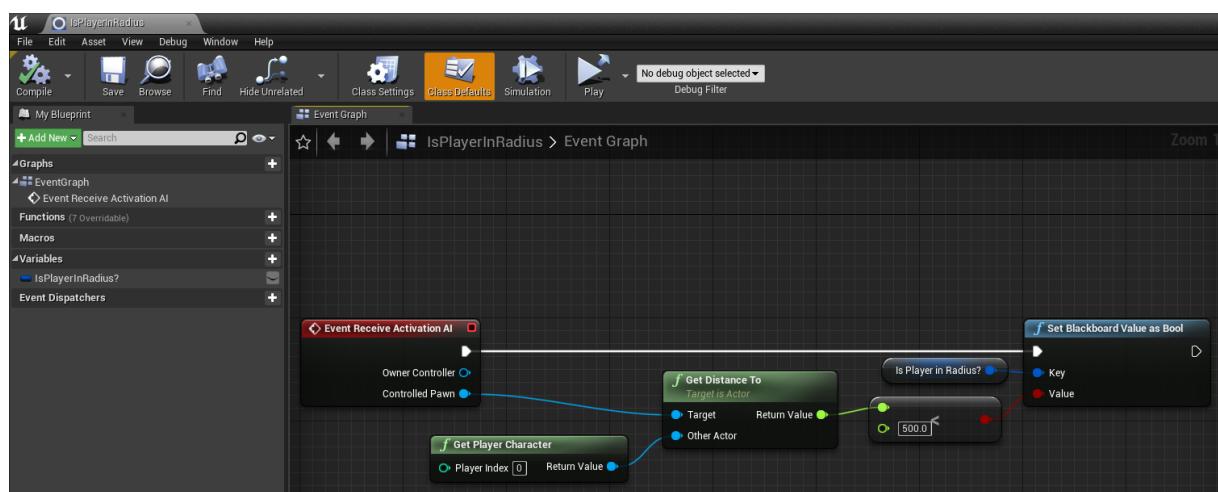
Abb. H.13: Animals: AnimalsMasterAiController - Components And Variables



**Abb. H.14:** Animals: AnimalsMasterAiController - Begin Play



**Abb. H.15:** Animals: AnimalsMasterAiController - On Target Perception Updated Ai Perception



**Abb. H.16:** Animals: Behavior Tree Services - Is Player In Radius

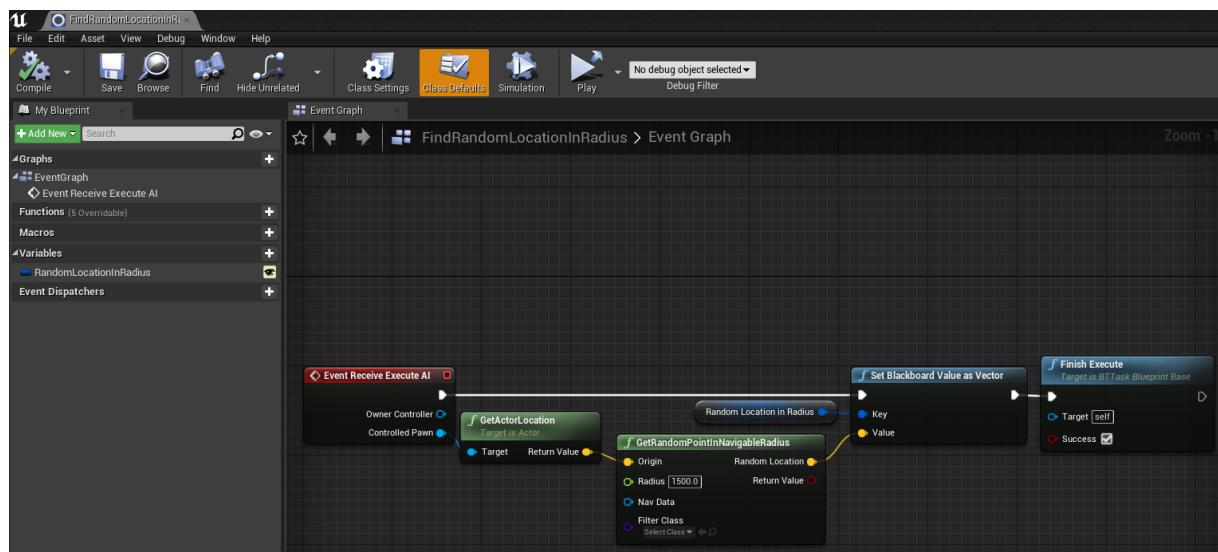


Abb. H.17: Animals: Behavior Tree Tasks - Find Random Location In Radius

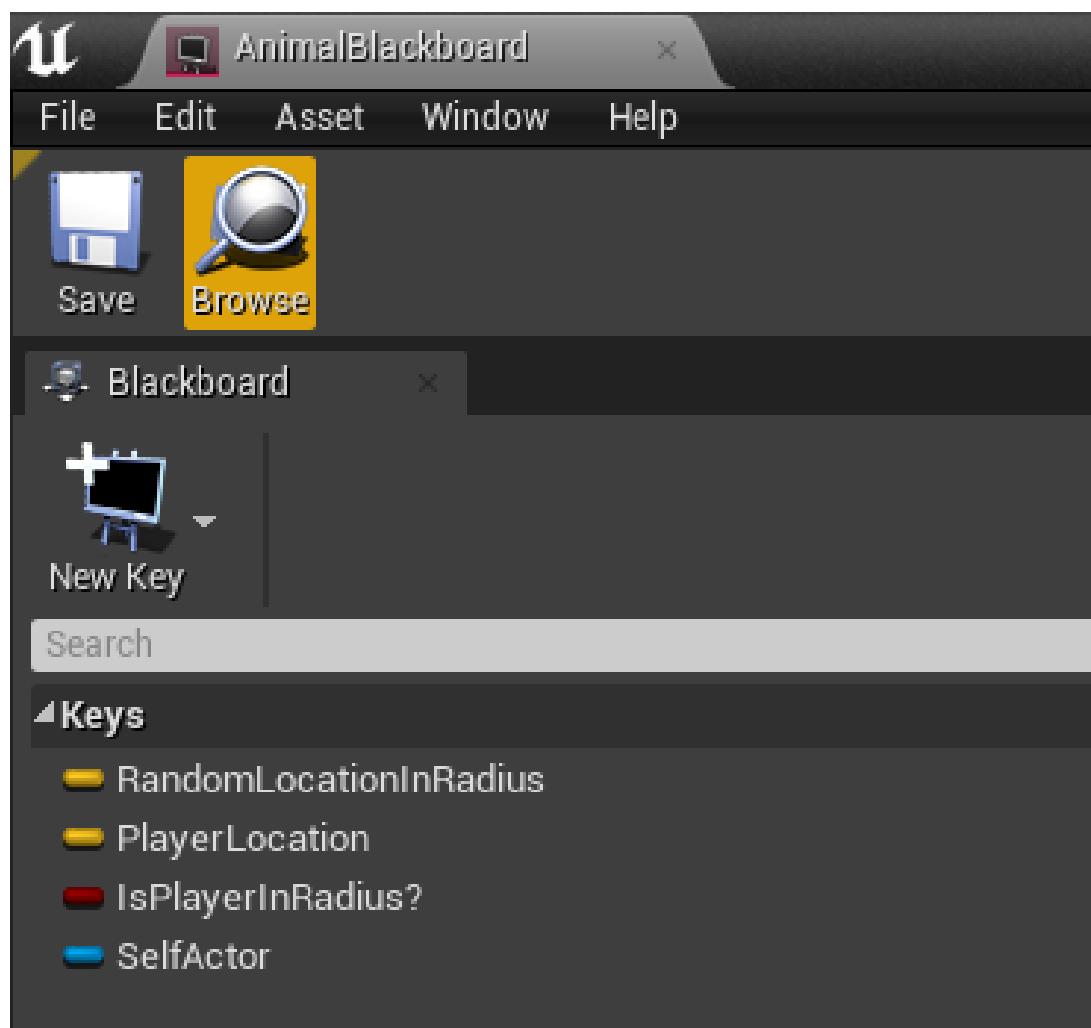
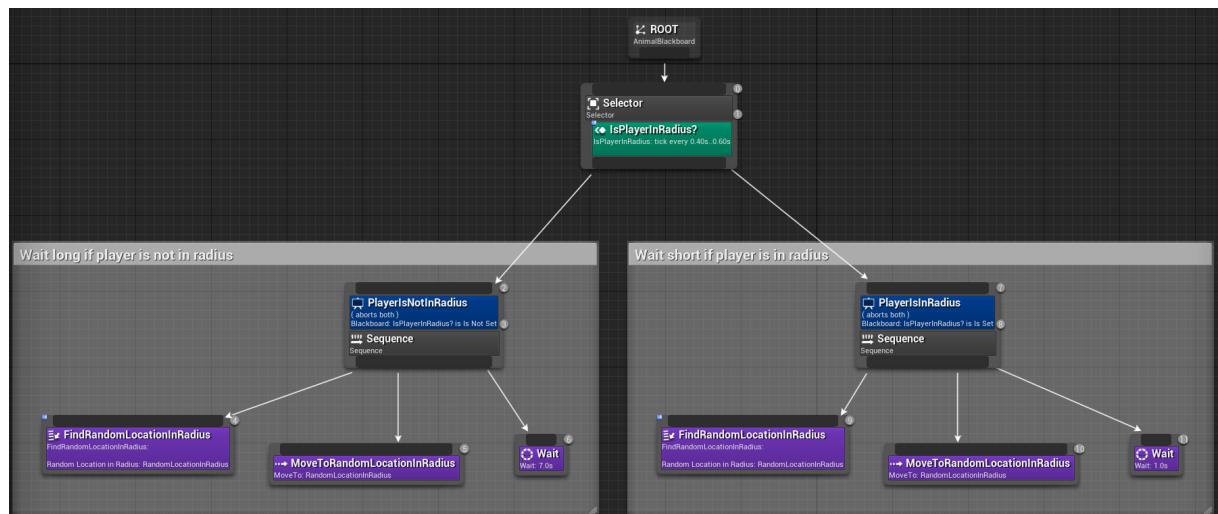
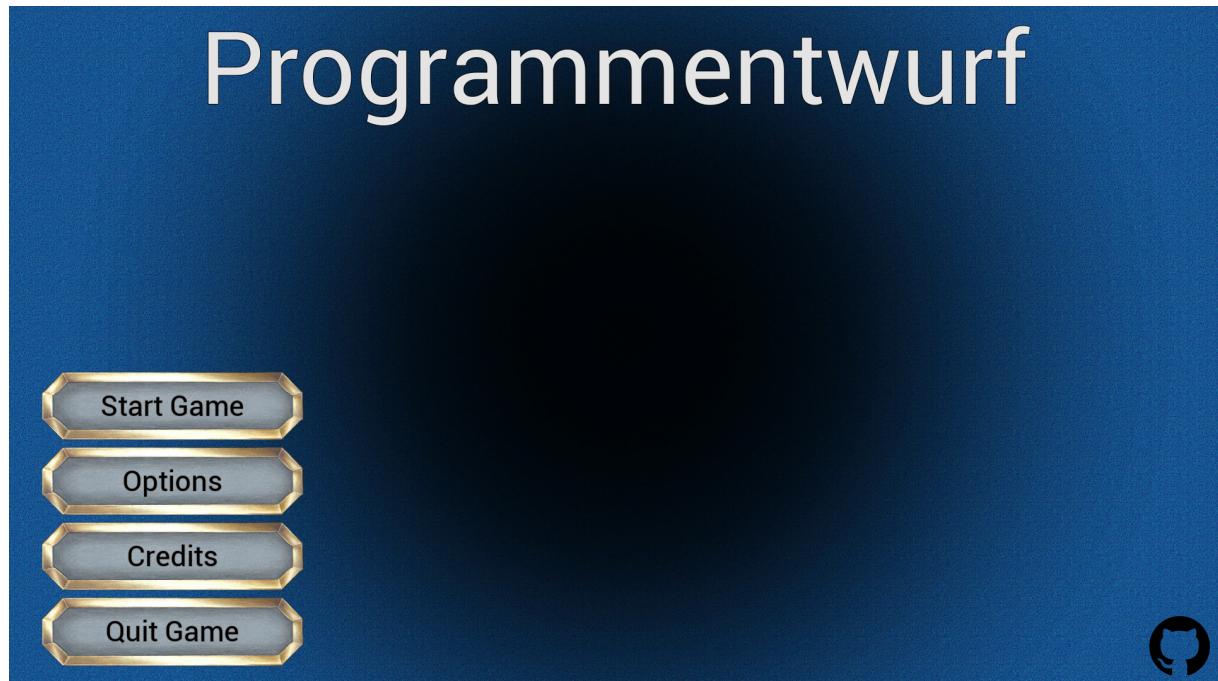


Abb. H.18: Animals: Behavior Tree Blackboard

**Abb. H.19:** Animals: Behavior Tree Overview

## I Anhang: MainMenu Interface

**Abb. I.1:** MainMenu Interface: Startup

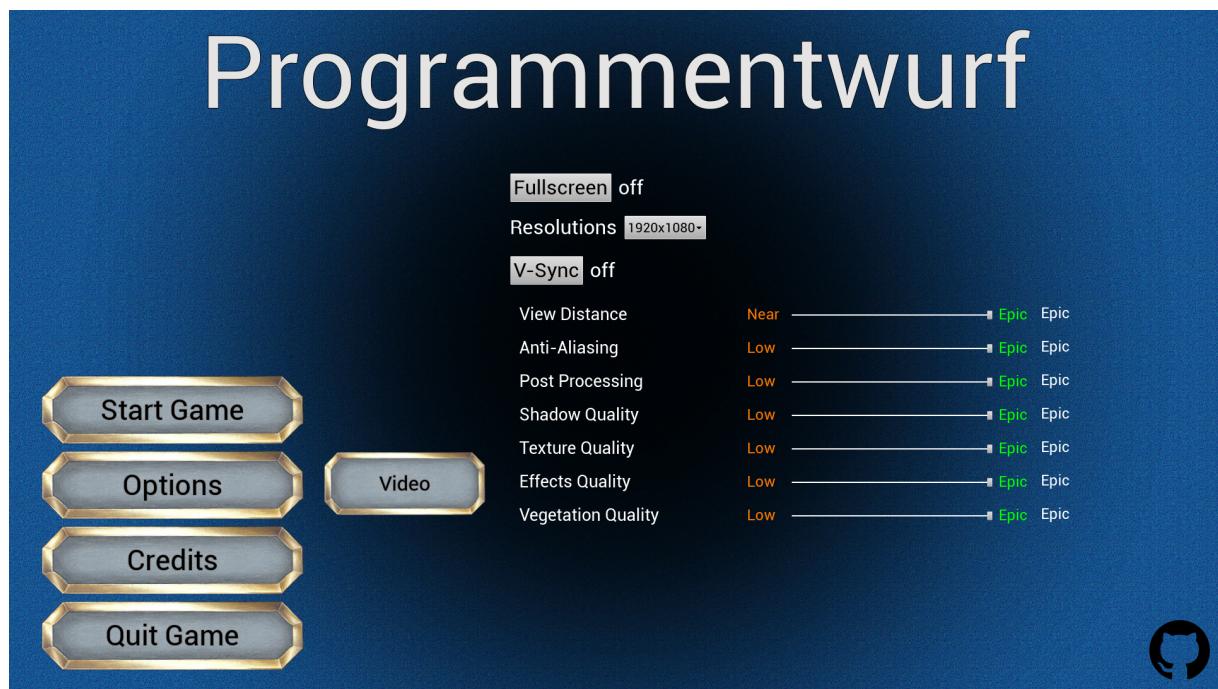


Abb. I.2: MainMenu Interface: Options - Video

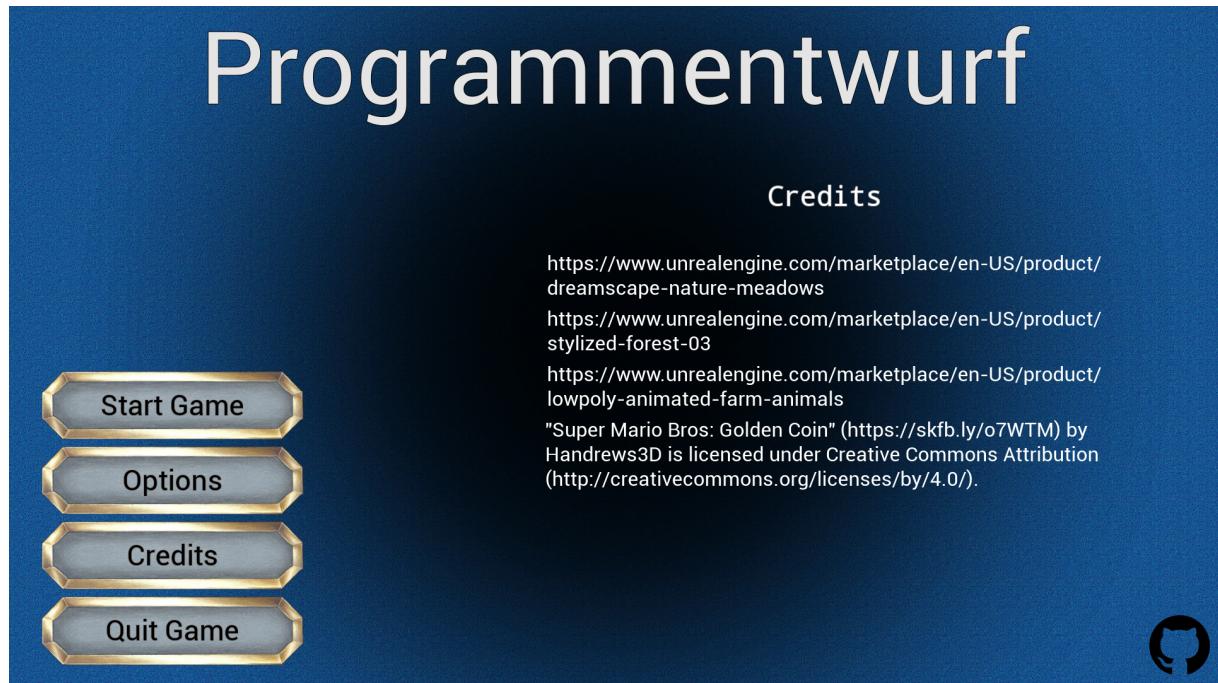


Abb. I.3: MainMenu Interface: Credits