

Bachelor-Projekt

Felix Nitzsche SS2020

Konzept 1: 18.05.2020	2
Assetliste zu Entwurf 1: 19.05.2020	3
Vergleich: 28.09.2020	5
Implementierte Assets:	6
MainMap	6
Enemy	6
Allgemein	6
EnemyFlyMelee	7
FlyMelee...	7
EnemyFlyOrb	7
FlyOrb...	7
EnemyFlyTetra	7
FlyTetra...	7
KugelProjektil	8
TetraProjektil	8
Floor	8
VRControlls	9
Music	9
MotionControllerPawnCustom	9
UI	10
Hands	10
BaseHand	11
MeleeHand	11
StromHand	12
Strom	12
LaserHand	13
Laser	13

Konzept 1: 18.05.2020

Eine Art simpler VR-Shooter mit eher stationärem Gameplay.

1. Gameplay:
 - a. man steht auf einer weiten Fläche
 - b. aus mittlerer Entfernung fliegen Gegner auf einen zu
 - c. man muss den Angriffen der Gegner ausweichen
 - d. nach zwei Treffern verloren
 - e. man muss die Gegner mit den eigenen Waffen schlagen
2. Umgebungsgestaltung:
 - a. dunkler, nachtaktiger Hintergrund
 - b. Entfernung verschwindet in gräulich schwarzem Dunst/Nebel oder ähnliches
 - c. ein Boden mit Muster aus hexagonalen oder dreieckigen Kacheln
 - d. evtl. einige Abstrakt wirkende Objekte zur Gestaltung (z.B. kahle simple Bäume, Säulen)
 - e. evtl. Partikelsystem für Atmosphäre
3. Gegner:
 - a. simple , aber charakteristische Geometrische Formen (Würfel, Kugel, Tetraeder)
 - b. metallene Oberfläche, evtl. wie flüssig wirkend
 - c. je nach Form unterschiedliche Gegnertypen. Ideen:
 - i. Würfel: Schneller Flug, greift durch Kollision an
 - ii. Kugel: langsamer Flug, verschießt langsame große Projektile
 - iii. Pyramide: sehr langsam, mit Hitscan Waffe
4. Waffen:
 - a. Die virtuellen Hände dienen als Waffen
 - b. unterschiedliche Hände für verschiedene Waffenarten
 - c. es können zwei verschiedene Hände gewählt werden (vor dem Kampf)
 - d. Arten(Ideen):
 - i. Melee: große Faust, Gegner werden im Nahkampf zertrümmert, evtl. Schlag auf Boden löst Welle der Bodenkacheln aus, erstellt also effektiv eine temporäre Barriere um den Spieler
 - ii. Laser: drei Finger gestreckt (Daumen, Zeige- und Mittelfinger), Schüsse werden von den gestreckten Fingern abgegeben, evtl. Schlag auf Boden löst Explosiven Schuss aus
 - iii. Strom: Blitze werden von der Hand auf Gegner näherer Umgebung abgegeben, evtl. Schlag auf Boden Blitzkugel unmittelbar um Spieler
 - iv. Telekinese: greifen einzelner Gegner und werfen dieser, evtl. Schlag auf Boden hebt Bodenkacheln unter nahen Gegnern an und zertrümmert diese damit

Assetliste zu Entwurf 1: 19.05.2020

1. Umgebung:

- a. dunkle Skybox
- b. diffuses Hintergrundlicht
- c. Würfel mit Nebelvolumen oÄ.
- d. Dreiecksprisma für Bodenplatte
 - i. matt raues, metallisches Material in dunkelgrau
 - ii. keine Textur
 - iii. Sound: Knirschen
- e. Bodenplatte zum Füllen der Kachelfugen
 - i. einfarbiges, leuchtendes Material
 - ii. Bonus: Wabern der Leuchtkraft per Noisetextur
- f. Dekosäulen
 - i. raues, nichtmetallisches hellgraues Material
 - ii. Hex.prisma, Quader, Kegel
 - iii. keine Textur
- g. Dekobäume
 - i. raues, nichtmetallisches graubraunes Material
 - ii. mehrere Meshes, mit dem Treegenerator von Blender erstellt
 - iii. keine Textur
- h. Staubpartikel
 - i. leuchtende kleine runde Partikel
 - ii. simple punktförmige Textur

2. Gegner:

- a. Würfel
 - i. Mesh: Körper
 - 1. Würfel
 - ii. Material: Körper
 - 1. silber, metallenes Material
 - 2. flüssig wirkende Oberfläche durch Noisetextur für Normal
- b. Kugel
 - i. Mesh: Körper
 - 1. Kugel
 - ii. Material: Körper
 - 1. silber, metallenes Material
 - 2. flüssig wirkende Oberfläche durch Noisetextur für Normal
 - iii. Mesh: Projektil
 - 1. Doppelte Kugel
 - iv. Material: Projektil
 - 1. Innen: transparent, rot orange leuchtend, keine Textur
 - 2. Außen: transparent, blau leuchtend, keine Textur
- c. Pyramide
 - i. Mesh: Körper

- 1. Kugel
 - ii. Material: Körper
 - 1. silber, metallenes Material
 - 2. flüssig wirkende Oberfläche durch Noisetextur für Normal
 - iii. Mesh: Projektil
 - 1. Zylinder
 - iv. Material: Projektil
 - 1. matt, nichtmetallisch weißes Material, keine Textur
 - d. Spawn:
 - i. Materialize Material
 - ii. heller werdendes Summen
 - e. Hit:
 - i. Partikel,
 - ii. prozedurale Textur, metallisch,
 - iii. Dumpfer Schlag
 - f. Death:
 - i. Materialize Material
 - ii. abnehmendes Summen
3. Waffen:
- a. Mesh: Hände
 - i. Hände links und rechts, erstellt mit MBLab in Blender
 - b. Variante 1: Meele
 - i. matt, nichtmetallisches, grünes Material
 - ii. Mesh zu Faust gebogen
 - iii. Sound: Dumpfer Schlag
 - c. Variante 2: Laser
 - i. glänzend, nichtmetallisch, weißes Material
 - ii. Mesh mit 3 Fingern gestreckt
 - iii. Projektil:
 - 1. Zylinder
 - 2. Material: matt, blau leuchtend
 - iv. Sound: Blaster
 - d. Variante 3: Strom
 - i. glänzend, metallisch, silbernes Material
 - ii. Handfläche nach oben, Finger leicht gekrümmt
 - iii. Beam Partikel, Lightningmaterial
 - iv. Sound: knisterndes Summen
 - e. Variante 4: Telekinese
 - i. matt, rot leuchtendes Material
 - ii. Handfläche nach vorne unten, Finger leicht gerümmt
 - iii. SelektionsKugel
 - 1. Mesh: Kugel
 - 2. Material: transparent, rotleuchtend
 - iv. dumpfes Summen

Vergleich: 28.09.2020

Vergleich der Implementierung mit dem ursprünglichen Konzept.

1. Gameplay:
 - i. man steht auf einer weiten Fläche **Implementiert**
 - ii. aus mittlerer Entfernung fliegen Gegner auf einen zu **Implementiert**
, aber mit einer hohen Distanz
 - iii. man muss den Angriffen der Gegner ausweichen **Implementiert**
, die Hitbox ist aber auf den Kopf beschränkt
 - iv. nach zwei Treffern verloren **Implementiert**
, da zwei Trefferpunkte zu wenig waren verliert man nach 6 Treffern
 - v. man muss die Gegner mit den eigenen Waffen schlagen **Implementiert**
2. Umgebungsgestaltung:
 - i. dunkler, nachtartiger Hintergrund **Implementiert**
es ist jedoch zur Orientierung eine Sonne am Horizont zu sehen
 - ii. Entfernung verschwindet in gräulich schwarzem Dunst/Nebel oder ähnliches **Nicht Implementiert**
 - iii. ein Boden mit Muster aus hexagonalen oder dreieckigen Kacheln **Implementiert, es sind Dreieckige Kacheln geworden**
 - iv. evtl. einige abstrakt wirkende Objekte zur Gestaltung (z.B. kahle simple Bäume, Säulen) **Implementiert, werden in mittlerer Entfernung zum Spieler erstellt, und lassen eine kreisförmige Fläche frei**
 - v. evtl. Partikelsystem für Atmosphäre **Nicht Implementiert**
3. Gegner:
 - i. simple, aber charakteristische Geometrische Formen (Würfel, Kugel, Tetraeder) **Implementiert**
 - ii. metallene Oberfläche, evtl. wie flüssig wirkend **Implementiert**
 - iii. je nach Form unterschiedliche Gegnertypen. Ideen:
 1. Würfel: Schneller Flug, greift durch Kollision an **Implementiert**
 2. Kugel: langsamer Flug, verschießt langsame große Projektile **Implementiert**
 3. Pyramide: sehr langsam, mit Hitscan Waffe **Implementiert, aber mit schnellem Flug**
4. Waffen:
 - i. Die virtuellen Hände dienen als Waffen **Implementiert**
 - ii. unterschiedliche Hände für verschiedene Waffenarten **Implementiert**
 - iii. es können zwei verschiedene Hände gewählt werden (vor dem Kampf) **Implementiert, aber Hände können jederzeit im Spiel gewechselt werden**
 - iv. Arten(Ideen):

1. Melee: große Faust, Gegner werden im Nahkampf zertrümmert, **Implementiert**
evtl. Schlag auf Boden löst Welle der Bodenkacheln aus, erstellt also effektiv eine temporäre Barriere um den Spieler **nicht Implementiert**
2. Laser: drei Finger gestreckt (Daumen, Zeige- und Mittelfinger), Schüsse werden von den gestreckten Fingern abgegeben, **Implementiert**
evtl. Schlag auf Boden löst Explosiven Schuss aus **nicht Implementiert**
3. Strom: Blitze werden von der Hand auf Gegner näherer Umgebung abgegeben, **Implementiert**
evtl. Schlag auf Boden Blitzkugel unmittelbar um Spieler **nicht Implementiert**
4. Telekinese: greifen einzelner Gegner und werfen dieser, **Implementiert**
evtl. Schlag auf Boden hebt Bodenkacheln unter nahen Gegnern an und zertrümmert diese damit **nicht Implementiert**

Implementierte Assets:

MainMap

Die MainMap stellt die Spielwelt, die während des Spiels geladen ist. In dieser befinden sich die für den Spielbeginn benötigten Komponenten. Das sind

- AtmosphereFog
- DirectionalLight
- EnemySpawner
- ExponentialHeightFog
- FloorTiles
- MotionControllerPawnCustom

Die Datei findet sich in Content/Assets/MainMap.umap

Enemy

Allgemein

Gegner besitzen vier Methoden.

1. In Event BeginPlay wird der AIController erstellt. Dieser ist für die Bewegungssteuerung des Gegners zuständig.
2. In gotHit wird ein Treffer verarbeitet, wenn der Gegner vom Spieler getroffen wurde. Es werden die Lebenspunkte neu berechnet, und wenn die Lebenspunkte auf 0

fallen wird ein Sound (Content/Assets/Sounds/Enemys/EnemyKilled.uasset) abgespielt und der Gegner zerstört.

3. In pull wird eine durch die TeleHand ausgeübte Kraft an den AIController weitergegeben.
4. In Event Hit wird geprüft ob der Gegner durch eine Bewegung der TeleHand mit einem Objekt kollidiert, wenn ja wird der Kollisionsgeschwindigkeit entsprechend Lebenspunkte abgezogen.

EnemyFlyMelee

Dieser Gegner greift mit einem Sturzflug auf den Spieler an und verursacht Schaden wenn er mit dem Spieler kollidiert.

Der Gegner besteht aus zwei Objekten. Einer kollisions Box, und einem Static Mesh Content/Assets/Enemy/Components/EnemyW.uasset.

Die verwendeten Materialien sind EnemyEdges_M und EnemyFaces_M.

FlyMelee...

EnemyFlyOrb

Dieser Gegner bewegt sich langsam im Zickzack auf den Spieler zu. Er verschießt große langsame Projektile (Content/Assets/Enemy/KugelProjektil.uasset). Der Gegner zielt auf die Position, an der sich der Spieler befand als der Gegner gespawnt wurde.

Er besteht aus zwei Komponenten. Einer kollisions Sphere und einem Static Mesh Content/Assets/Enemy/Components/EnemyK.uasset.

Das verwendete Material ist EnemySphere_M.

FlyOrb...

EnemyFlyTetra

Dieser Gegner fliegt einen kleinen Kreis um den Spieler und teleportiert sich dann zu drei verschiedenen zufälligen Positionen. Von diesen Positionen verschießt der Gegner jeweils ein schnelles Projektil Content/Assets/Enemy/TetraProjektil.uasset. Der Gegner zielt auf die Position, an der sich der Spieler befand als der Gegner gespawnt wurde.

Der Gegner besteht aus zwei Objekten. Einer kollisions Capsule, und einem Static Mesh Content/Assets/Enemy/Components/EnemyT.uasset

Die verwendeten Materialien sind EnemyEdges_M und EnemyFaces_M.

FlyTetra...

KugelProjektil

Dieses Objekt ist das Projektil, dass vom EnemyFlyOrb verschossen wird. Das Projektil hat zwei Methoden. In Event BeginPlay wird das Projektil, mit einer TimeLine über eine Dauer von 5 Sekunden, von der Spawnposition zum Ziel bewegt.

In Event ActorBeginOverlap wird das Projektil bei Kollision zerstört. Von der Kollision ausgenommen sind

1. TeleHand
2. StromHand
3. EnemyFlyOrb

Das Projektil besteht aus einem Static Mesh

Content/Assets/Enemy/Components/EnemyTProjektil.uasset mit dem Material ProjektilT_M.

TetraProjektil

Dieses Objekt ist das Projektil, dass vom EnemyFlyTetra verschossen wird. Das Projektil hat zwei Methoden. In Event BeginPlay wird das Projektil, mit einer TimeLine über eine Dauer von 1,5 Sekunden, von der Spawnposition zum Ziel bewegt.

In Event ActorBeginOverlap wird das Projektil bei Kollision zerstört. Von der Kollision ausgenommen sind

1. TeleHand
2. StromHand
3. EnemyFlyTetra

Das Projektil besteht aus einem Static Mesh

Content/Assets/Enemy/Components/EnemyKProjektil.uasset mit den Materialien ProjektilKIn_M und ProjektilKOut_M.

Floor

Der Boden wird über den Actor "FloorTiles" erzeugt.

FloorTiles beinhaltet die zwei Komponenten Tile und FloorPlane.

- Tile ist ein "Instanced Static Mesh Component" und instanziiert das Static Mesh "Content/Assets/Floor/Components/Dreiecksprisma.uasset" in einem 120 mal 120 Gitter. Der Code dafür ist im ConstuctionScript zu finden. Bei der Generierung können die Gittergröße ("count" * 2), die Distanz der Prismen zueinander in X und Y Richtung ("DistanceX" und "DistanceY") angepasst werden.
- FloorPlane (Content/Assets/Floor/Components/FloorPlane.uasset) ist eine einfache Ebene mit dem Material Fill, welches den Tiles ähnlich sieht. Dieses Objekt ist dazu gedacht die Ebene von Tile, auf der der Spieler steht, auch in der ferne fortzusetzen, jedoch die Rechenlast zu reduzieren. Im Material Fill gibt es zwei Konstanten ("radius"), mit denen das verlöschen des leuchtenden Füllmaterials zwischen den Kacheln über die Distanz zur Mitte des Feldes eingestellt werden kann.

FloorTiles hat eine Methode. In der Methode Event BeginPlay werden alle Dekogegenstände auf dem Feld platziert. Dazu wird in einer Schleife, deren Iterationsanzahl mit der Variable "DekoCountMax" vorgegeben werden kann, jeweils ein Dekoobjekt (aus

“Content/Assets/Floor/Deko/”) zufällig ausgewählt und mit zufälligen X und Y Koordinaten, in einem durch “distMin” und “distMax” vorgegebenen Radius, zufälliger Rotation um die Z-Achse und zufälliger Skalierung initialisiert (s. “Random Transform”).

Bei den Dekoobjekten handelt es sich um

1. Content/Assets/Floor/Deko/BaumGrossLight.uasset
2. Content/Assets/Floor/Deko/BaumKleinLight.uasset
3. Content/Assets/Floor/Deko/BeetWithLight.uasset
4. Content/Assets/Floor/Deko/DoppelPSauleLight.uasset
5. Content/Assets/Floor/Deko/HeckeLight.uasset

Alle diese Objekte sind vom Typ Actor, haben jedoch keinerlei Methoden oder Funktion. Sie bestehen aus einem in Blender erstellten Mesh mit einigen PointLights.

VRControlls

Dieser Ordner enthält alle zum VRRig gehörenden Komponenten.

BP_MotionController, GripEnum, MotionControllerHaptics und MotionControllerPawnCustom sind aus “Content/VirtualRealityBP/Blueprints/” kopiert. Nur letzteres wurde modifiziert.

Music

Music ist eine SoundCue, die an die Kamera des VRRig geheftet ist und die Hintergrundmusik in zufälliger Reihenfolge abspielt. Die Musikdateien sind in “Content/Assets/Sounds/Background/” zu finden.

MotionControllerPawnCustom

Im folgenden werde ich die Änderungen erläutern, die ich an der Vorlage vorgenommen habe.

Ich habe an die Kamera eine kollisions Capsule und Music gehängt. Die Capsule ist dafür da, dass der Spieler von den Gegnern getroffen werden kann. Die Music ist die Hintergrundmusik.

Des Weiteren habe ich drei Textobjekte erstellt. Diese sind dem VROrigin unterstellt.

Das Objekt “Lifes” zeigt die noch verfügbaren Reserveleben an, GO wird angezeigt wenn der Spieler verloren hat und Runde zeigt die aktuelle Gegnerwelle/Runde an in die sich der Spieler vorgearbeitet hat.

Ich habe die beiden Variablen LeftController und RightController durch LCon und RCon ausgetauscht. LCon und RCon sind vom Typ “BaseHand”. Den Bereich “Spawn and attach both motion controllers” der Methode “Event BeginPlay” habe ich entsprechend angepasst und auch die gespawnten Handobjekte ausgetauscht.

In der Methode "Event BeginOverlap" wird geprüft ob das überlappende Objekt vom Typ "EnemyFlyTetra", "KugelProjektil" oder "TetraProjektil" ist. Falls dies nicht der Fall ist, wird die Kollision ignoriert. Sonst wird ein Lebenspunkt des Spielers abgezogen und überprüft ob der Spieler damit alle Leben aufgebraucht hat. Wenn ja, so wird das Textobjekt "GO" eingeblendet, der Sound "GameOver" abgespielt und alle Gegner im Spiel vernichtet.

In der nächste Gruppe an Methoden "Handle Controller Input" werden die Spielereingaben der Controller verarbeitet. In der Gruppe "UI" werden die Interaktionen mit der UI ermöglicht. Dazu wird mit der "Grab"-Taste der Controller ein Linksklick auf das anvisierte Element ausgeführt. Damit die UI-Pointer während des Spieles nicht sichtbar sind, werden diese nur angezeigt, wenn der Zeigefinger vom Trigger genommen wird. Die folgenden beiden Inputs sind das Drücken und Loslassen des Triggers (Zeigefinger). Diese Ereignisse werden an L- und RCon weitergegeben.

In der folgenden Gruppe "Hände ersetzen" werden bei bewegen der Thumbsticks in eine der vier Richtungen die in L-oder RCon gespeicherte Hand gelöscht und durch die gewählte neu gespawnte Hand ersetzt (mittels "ReplaceHand").

Die Methode "SetLives" wird durch "Event BeginOverlap" aufgerufen und aktualisiert die Anzeige der Reserveleben ("Lives").

Die Methode "UpdateRunde" wird durch den EnemySpawner aufgerufen und aktualisiert die Anzeige der aktuellen Gegnerwelle/Runde ("Runde").

In der Methode "ReplaceHand" wird eine der beiden Hände ersetzt, ob links oder rechts wird mit dem Parameter "left" übergeben. Daraufhin werden alle aktionen der geladenen Hand mit "ReleaseTrigger" beendet und die Hand wird gelöscht. Dann wird eine neue Hand des übergebenen Typs erzeugt und in L-oder RCon gespeichert.

UI

In dem Unterordner "UI" befinden sich die beiden Dateien "VR_UI" und Interface. Interface ist ein Actor der nur als "Hülle" für "VR_UI" dient. In "VR_UI" sind zwei Buttons angeordnet. Der linke, mit der Aufschrift "Restart", lädt das Level neu. *Dies kann eine länger Wartezeit zur folge haben.* Der rechte Knopf beendet das Programm.

Hands

Dieser Ordner enthält alle Komponenten der vier Hände, bis auf die Soundfiles.

Dieser Ordner ist ein Unterordner von VRControlls. Es befinden sich fünf Dateien und vier Unterordner in "Hands". Die fünf Dateien sind die Basishand mit den vier ausgestalteten Händen. In den Unterordnern befinden sich die einzelnen Komponenten der Hände.

BaseHand

Um alle Hände über einheitliche Methoden ansprechen zu können, habe ich das Blueprint "BaseHand" erstellt. Von diesen erben alle anderen Hände.

Die BaseHand hat, neben dem kopierten MotionController, zwei Komponenten. Das Static Mesh beherbergt alle Komponenten der Hand, die für die Unterschiede zwischen linker und rechter Hand gespiegelt werden müssen. Die WidgetInteraction Komponente stellt den Pointer um mit der UI interagieren zu können.

In der BaseHand sind sechs Methoden implementiert. In der "Event BeginPlay" Methode wird überprüft ob es sich um eine linke oder rechte Hand **handelt**. Wenn es eine rechte Hand ist wird das Static Mesh an der Y-Achse gespiegelt.

In der "tryHit" Methode werden Treffer an den betreffenden Gegner weitergegeben. Dazu bekommt die Methode den getroffenen Actor übergeben und castet diesen dann in den passende Gegnertypen. Dem wird dann der Schaden und die den Treffer landende Hand übergeben.

Die letzten vier implementierten Methoden sind die Methoden zur Interaktion mit dem UI. In "PointerPress" und "PointerRelease" wird mit dem "Widget Interaction" ein klick der linken Maustaste simuliert. In "Beam Press" und "BeamRelease" wird der Pointer des "Widget Interaction" eingeblendet.

Die fünf Methoden "Event ActorBeginOverlap", "EventTick", "EnemyKill", "Triggered" und "TriggerReleased" stellen Interfaces dar, die in den einzelnen Händen implementiert werden können.

MeleeHand

Die MeleeHand erbt von der BaseHand und hat drei eigene Komponenten. Die Faust (/Content/Assets/VRCOnrolls/Hands/Hand/Poses/Faust.uasset) ist das im Spiel angezeigte Mesh der Hand. Die Komponente "Sphere1" stellt die Hitbox der Hand dar. Und die C++-Klasse "MeleeHandController".

MeleeHand hat drei Methoden, die ersten beiden sind jedoch sehr ähnlich. "Event Hit" oder "Event ActorBeginOverlap" werden bei treffen eines Objektes mit der Hand ausgelöst.

Daraufhin wird in der Methodew "onHit" des MeleeHandControllers der Schaden für den Gegner berechnet (es wird immer 110 zurückgegeben). Dieser Schaden wird dann mit der "Try Hit" Methode der BaseHand an den Gegner weitergegeben. Zuletzt wird noch der Sound "MeleeSound" (Content/Assets/Sounds/Hand/MeleeSound.uasset) abgespielt.

In der Methode "Event Tick" soll anhand der Position und gerichteten Geschwindigkeit der Hand überprüft werden ob auf den Boden geschlagen wurden um die Spezialfähigkeit auszulösen. Diese ist jedoch nicht implementiert und daher ist diese Methode ungetestet und ein Platzhalter.

StromHand

Die StromHand erbt von der BaseHand. Sie fügt sechs Komponenten hinzu. Die optischen Komponenten sind "Hold" und "Sphere", Hold ist das Mesh der Hand und Sphere ist die darüber schwebende Kugel mit einem stromartigen Material. An der Sphere hängt noch eine SoundCue. Diese erzeugt ein blitzartiges Geräusch, während der Trigger gedrückt wird. Neben dem C++-Skript "StromHandController" gibt es noch zwei Boxen. Die Boxen dienen als Collide um zu ermitteln, welche Gegner sich in Reichweite befinden.

Die StromHand hat fünf Methoden "Event BeginPlay", "Event Tick", "Event Enemy Kill", "Event Triggered" und "Event Trigger Released". In letzteren beiden wird die Variable "active" auf true/false gesetzt und der die SoundCue gestartet/gestoppt.

"Event Enemy Kill" wird von einem Gegner aufgerufen, der zerstört wurde. In der Methode wird der Gegner aus der Liste aller Gegner in Reichweite entfernt.

Die Methode "Event Tick" ist mit Abstand die größte Methode der StromHand.

Als erstes wird auch bei dieser Hand auf die (nicht implementierte) Spezialfähigkeit getestet, also ob auf den Boden geschlagen wurde. Dieser Teil der Methode hat jedoch keine Funktion.

Im folgenden wird dann getestet ob die Hand aktiv ist, ob die Variable "activ" gesetzt ist oder nicht.

Wenn sie nicht aktiv ist, werden alle in "Arcs" gespeicherten Partikelsysteme gelöscht, die Listen "Arcs", "Enemies in Reach" und "Strom" geleert und Midpoint und FirstArc auf hidden gesetzt.

Ist die Hand jedoch aktiv werden Midpoint und FirstArc angezeigt. Folgend werden dann alle Objekte, die mit den beiden Boxen kollidieren, in der Methode "get Targets" des StromhandControllers, ermittelt und alle Objekte die keine Gegner sind aussortiert. Die ermittelten Gegner werden in der Liste "Enemies in Reach" gespeichert. In der Gruppe "Mittelpunkt berechnen" wird die Position des Midpoint berechnet. Die Position entspricht dem Mittelpunkt zwischen allen Gegnern in Reichweite und der Hand.

In "Destroy Left" werden alle Partikelsysteme zerstört, deren Gegner sich nicht mehr in Reichweite befinden. In "Add new" werden für jeden Gegner, der neu in Reichweite gekommen ist, ein Partikelsystem erzeugt, welches den Midpoint mit dem Gegner verbindet. In der Gruppe "Schaden errechnen" wird mit der Methode "Get Damage" des StromHandControllers der Schaden für jeden Gegner errechnet. Dieser wird den Gegnern dann in "Schaden verursachen" mittels "Try Hit" zugefügt.

In der Methode "Event BeginPlay" werden zuerst zwei Objekte erzeugt. Das sind der Midpoint und ein Partikelsystem, welches den Midpoint mit der Hand verbindet. Beide werden direkt auf hidden gesetzt. Zuletzt werden noch die beiden Kollisionsboxen in eine Liste eingefügt.

Strom

In dem Unterordner "Strom" befinden sich wichtige Komponenten der StromHand. Diese sind:

1. Lightning_M: Das Material für die Strom-Partikelsysteme
2. Midpoint: Das ist der Actor, der als Midpoint der Hand erstellt wird

3. MidpointArcs: Das Material des Midpoint.
4. Strompart: Das Partikelsystem, das Gegner mit dem Midpoint verbindet.
5. StromPart Location: Das Partikelsystem, das die Hand mit dem Midpoint verbindet.
6. StromSoundCue: Die SoundCue für die StromHand.

LaserHand

Die LaserHand erbt von der BaseHand. Neben den geerbten Komponenten hat die LaserHand noch fünf weitere Komponenten. Das sind Pointing, als visuelle Handkomponente, r, r1, r2 als Spawnpunkte der Projektile und das C++-Skript LaserHandController.

Die LaserHand hat zwei Methoden "Event Tick" und "Event Triggered". In "Event Tick" wird, wie bei den anderen Händen auch die Aktivierung der, nicht implementierten, Sonderfunktion abgefragt. Dies sollte über die Distanz zu Boden und der Geschwindigkeit in Z-Richtung geschehen, aber die Methode "Check Floor Hit" des LaserHandControllers ist leer.

Die Methode "Event Triggered" ist für das abfeuern der Projektile zuständig. Dafür werden drei Projektile an den Positionen von r, r1, r2 in deren Z-Richtung gespawnt und der Sound "LaserSound" abgespielt.

Laser

Im Unterordner "Laser" befinden sich die Komponenten und das Projektil. "projektilLaser" ist das Mesh und "laser_M" ist das Material des Projektils. Das Blueprint des Projektils ist "laserProjektil". Das Projektil besteht aus dem Static Mesh und einem, an der Spitze positionierten, Point Light. Des Weiteren hat das Projektil drei Methoden. In "Event BeginPlay" wird das Projektil mit einer TimeLine über einen Zeitraum von 0,5 Sekunden von dem Spawnpunkt aus 5000 Einheiten vorwärts bewegt. Sobald das Projektil die ganze Strecke zurückgelegt hat wird es zerstört.

Die beiden anderen Methoden sind fast identisch. In "Event ActorBeginOverlap" und "Event Hit" wird versucht dem getroffenen Objekt, sofern es keine Hand oder ein Laserprojektil ist, mittels der "TryHit" Methode, der schießenden Hand, Schaden zuzufügen.