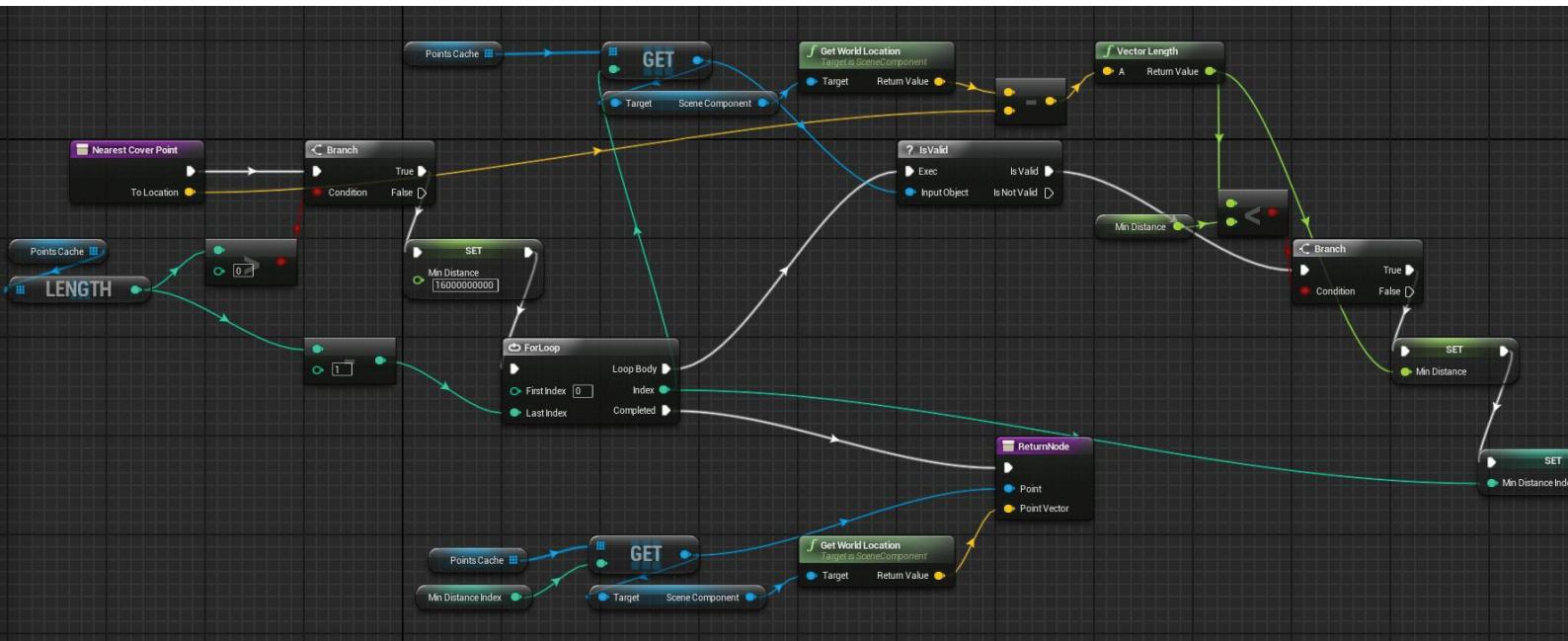


Virtual reality ontwikkeling voor niet-programmeurs

Mark Arts
0851423



Mediatechnologie
Instituut voor Communicatie Media
en Informatietechnologie

“I choose a lazy person to do a hard job. Because a lazy person will find an easy way to do it.”

Bill Gates

HOGESCHOOL ROTTERDAM

Abstract

Mediatechnologie

Instituut voor Communicatie Media en Informatietechnologie

Bachelor of Science

Virtual reality ontwikkeling voor niet-programmeurs

door

Door de groeiende vraag naar virtual reality omgevingen zijn er steeds meer bedrijven gespecialiseerd in 3D visualisatie die met virtual reality aan de slag willen. Omdat deze bedrijven weinig tot geen ervaring met programmeren hebben en het programmeren van interactie in virtual reality momenteel een complexe taak is, wordt in deze scriptie gezocht naar een ontwikkelenomgeving waarin niet-programmeurs die wel ervaring hebben met 3D ontwikkeling interactieve virtual reality omgevingen kunnen maken. Deze ontwikkelenomgeving wordt gerealiseerd door de VRInteractions plugin voor de Unreal Engine 4. De plugin implementeert de virtual reality gerelateerde logica en koppelt deze aan een visuele programmeer taal. De plugin is ontwikkeld met een workflow die zich focust op gebruiksgemak en vrijheid voor de niet-programmeurs. Door middel van workshops en een gebruikstest wordt de conclusie getrokken dat niet-programmeurs zelfstandig omgevingen kunnen opzetten en interactief kunnen maken mits zij hulp kunnen krijgen van een programmeur.

Inhoudsopgave

Abstract	ii
-----------------	-----------

1 Inleiding	1
1.1 Probleemstelling	2
1.2 Doelstelling	2
1.3 Onderzoeksvorag	2
1.3.1 Hoofdvraag	2
1.3.2 Deelvragen	3
1.3.3 Onderzoeksmethoden	3
2 Unreal Engine 4	5
2.1 Vergelijking van game engine's	5
2.1.1 Unity	6
2.1.2 CRYENGINE	7
2.1.3 Unreal Engine 4	7
2.2 Conclusie	8
3 Visual Scripting	10
3.1 Use Case's	11
3.2 Blueprints	11
3.3 Gameplay Logica	12
3.3.1 Het afvuren van een projectiel	12
4 Uiteenzetting Vooronderzoek	16
4.1 Wat is de huidige kennis van computerlogica onder de werknemers van DPI?	17
4.1.1 Conclusie	17
4.2 Is er extra training nodig om met de ontwikkelomgeving aan de slag te gaan?	17
4.2.1 Workshops	17
4.3 Welke componenten zullen gemaakt moeten worden?	18
4.3.1 Wat zijn de huidige taken van een programmeur in het maken van een Virtuele omgeving?	18
4.3.2 Welke tools bestaan er al?	19
4.3.3 Wat zijn de juiste abstracties van de componenten?	19
4.3.4 Conclusie	20
4.4 Is het mogelijk de componenten cross-platform te maken?	21

4.4.1	Welke platforms zijn relevant?	21
4.4.2	Wat zijn de verschillende mogelijkheden van deze platforms?	21
4.4.3	conclusie	22
5	Blueprints en C++	23
5.1	Gameplay	23
5.1.1	Wanneer vraag	24
5.1.2	Wat	26
5.1.3	Hoe vraag	29
5.1.4	Onderhoud en Performance	29
5.2	Labellen en groeperen van functies en opties	30
5.3	Workflow	31
5.4	Versie Controle	32
5.5	Conclusie	33
6	VRInteractions	34
6.1	De plugin	34
6.2	Gamemodes	35
6.2.1	VRInteractionsGameMode	35
6.2.2	VRInteractionsFlyTroughGameMode	36
6.3	Characters	36
6.3.1	VRInteractionBaseCharacter	36
6.3.2	VRInteractionCharacter	36
6.3.3	VRInteractionsFlyCharacter	37
6.4	VRInteractionsPlayerController	37
6.5	VRInteractionsCameraManager	37
6.6	VRInteractionsHud	37
6.7	LookEventsComponent	38
6.7.1	Implementatie	38
6.7.2	Events	40
6.7.3	Instelling	41
6.7.4	Debugging	43
6.8	CircleMenu	44
6.8.1	Instellingen	45
6.8.2	Events	45
6.9	VRMovableMesh	46
6.9.1	Instellingen	47
7	Resultaten, Discussie en Vervolgonderzoek	48
7.1	Resultaten	48
7.2	Discussie	49
7.3	Vervolgonderzoek	50
8	Conclusie	51
A	Workshop 1	53

A.1 Doel	53
A.2 Best practises	53
A.3 Workshop	55
A.4 Reflectie	55
B Workshop 2	57
B.1 Doel	57
B.2 Opdrachten	57
B.3 Resultaten	59
B.4 Reflectie	60
C Workshop 3	62
C.1 Doel	62
C.2 Opdrachten	62
C.3 Reflectie	66
D Usability Test	67
D.1 De Opdracht	67
D.2 Uitvoering	70
D.3 Conclusie	71
E Implementatie voorbeeld speler Unreal Engine 4	72
F Conditional logic van Tick functie van LookEvents in c++	78
G Conditional logic van Tick functie van LookEvents in Blueprints	80
H Oriëntatie Interview	81
H.1 Doel	81
H.2 Interview	82
H.3 Enquête	83
Bibliografie	87

Hoofdstuk 1

Inleiding

In een periode van 18 weken zijn er een reeks van basis componenten voor de Unreal Engine 4 (UE4) ontwikkeld met als doel niet-programmeurs efficiënt interactie aan Virtual Reality (VR) demo's te laten toevoegen.

De UE4 is gebruikt omdat deze een uitgebreid Visual Scripting (VS) systeem heeft die het makkelijk maakt om simpele logica eenvoudig uit te drukken (zie hoofdstuk 3). Het VS systeem maakt het mogelijk voor niet-programmeurs om simpele logica toe te voegen zonder dat een programmeur nodig is. Dit zorgt voor een hogere productiviteit en kwaliteit zoals beschreven door Cutumisu et al. [1].

The traditional approach to implementing interactions between a player character (PC) and objects in computer games is to write scripts in a procedural scripting language. These scripts are usually so complex that they must be written by a computer programmer rather than by the author of the game story. This interruption in the game story authoring process has two distinct disadvantages: it increases the cost of game production and it introduces a disconnect between the author's intentions and the interactions produced from the programmer's written scripts.

1.1 Probleemstelling

Momenteel is het toevoegen van interactieve elementen in VR een intensieve programma taak omdat standaard nog geen interactie aanwezig is, en elke hardware weer andere input gebruikt. Het is daardoor lastig om de geprogrammeerde logica te hergebruiken in een ander project. Hierdoor is er altijd een programmeur nodig om de gewenste functionaliteit toe te voegen of aan te passen.

In gameontwikkeling zijn er de afgelopen jaren steeds meer technieken ontwikkeld om van de afhankelijkheid van programmeurs te verminderen [1, 2]. De oplossing van UE4 is het gebruik van Blueprints om interactie in een spel te programmeren. Maar op het moment van schrijven is er nog geen koppeling tussen Blueprints en VR interactie.

1.2 Doelstelling

De doelstelling van deze scriptie is het creëren van een reeks basis componenten voor UE4 waarmee niet programmeurs interactieve VR demo's kunnen maken die toegevoegde waarde hebben aan de workflow van DPI.

Aan het eind van dit traject zal het mogelijk zijn om zonder de hulp van een programmeur met de gemaakte componenten een bestaande omgeving geschikt te maken voor VR en om een nieuwe VR omgeving op te zetten.

1.3 Onderzoeksvraag

1.3.1 Hoofdvraag

De hoofdvraag van deze scriptie is: Hoe realiseren we een ontwikkelomgeving waarin niet-programmeurs, zoals 3D modellers en level designers, een efficiënte unieke VR ervaring kunnen creëren in UE4.

1.3.2 Deelvragen

- Hoe kan er een koppeling met Virtual Scripting (VS) gemaakt worden die intuïtief is voor niet programmeurs?
 - Wat is de huidige kennis van computer logica onder de werknemers van DPI?
 - Is er extra training nodig om met de ontwikkel omgeving aan de slag te gaan?
- Welke componenten zullen gemaakt moeten worden?
 - Wat zijn de huidige taken van een programmeur in het maken van een virtuele omgeving?
 - Welke tools bestaan er al?
 - Wat zijn de juiste abstracties van de componenten?
- Is het mogelijk de componenten cross-platform te maken?
 - Welke platforms zijn relevant?
 - Wat zijn de verschillende mogelijkheden van deze platforms?

1.3.3 Onderzoeksmethoden

1.3.3.1 Hoe kan er een koppeling met Virtual Scripting gemaakt worden die intuïtief is voor niet programmeurs?

Om een beter beeld te krijgen van het huidige VS landschap word er eerst deskresearch gedaan [1–3].

De vrijheid en mogelijkheden die de niet-programmeur krijgen, oftewel de abstractie van de componenten, zijn afhankelijk van de bestaande intuïtie van programmeren. Als een gebruiker bijvoorbeeld snapt hoe de “als dit dan dat” constructie werkt kan er meer vrijheid in de componenten gecreëerd worden. Maar als de gebruiker deze intuïtie niet heeft zal meer vrijheid, abstractie, alleen voor onduidelijkheid zorgen.

In de eerste plaats zal er onderzocht moeten worden wat de huidige kennis is van de niet-programmeurs. Dit kan door middel van interviews.

Vervolgens zal er onderzocht moeten worden wat een haalbare moeilijkheidsgraad is en of er baat is bij een extra training voor niet-programmeurs.

Als er sprake is van toegevoegde waarde van een extra training dan zal deze parallel aan het project gegeven worden en iteratief ontwikkeld worden.

1.3.3.2 Welke componenten zullen gemaakt moeten worden?

Om een beter beeld te krijgen van het huidige VR landschap word er eerst deskresearch gedaan [4–6, 6, 7]

Het onderzoek naar welke componenten gemaakt moeten worden zal beginnen met een onderzoek naar de huidige taken van de programmeur tijdens het maken van een virtuele reality omgeving. Aan de hand hiervan zal er gekeken worden welke taken geautomatiseerd kunnen worden of versimpeld naar VS.

Als er een duidelijk beeld is van de benodigde componenten zal er een onderzoek gedaan worden naar bestaande tools die ingezet kunnen worden om de workflow te verbeteren.

Als het duidelijk is welke componenten zelf geschreven moeten worden en hoeverre de niet-programmeurs met VS om kunnen gaan, kan er bepaald worden welke abstractie de componenten zo breed inzetbaar mogelijk houdt. Met als randvoornaarde dat het intuïtief blijft voor de niet-programmeurs.

1.3.3.3 Is het mogelijk de componenten cross-platform te maken?

Om te bepalen of het mogelijk is de componenten cross-platform te maken zal er onderzocht worden welke platforms relevant zijn en wat de technische mogelijkheden voor elk platform zijn.

Daarna zal voor elke component een beslissing genomen moeten worden of er een alternatieve versie moet komen of dat het cross-platform in de component zelf meegenomen kan worden.

Hoofdstuk 2

Unreal Engine 4

Als start van het onderzoek wordt eerst onderbouwd waarom er voor de UE4 game engine gekozen is. Een engine is namelijk de ontwikkelomgeving, en bevat de tools die gebruikt word om virtuele werelden te maken. Het is belangrijk dat de gekozen engine ondersteuning heeft voor VS en deze genoeg vrijheid geeft voor dit onderzoek.

2.1 Vergelijking van game engine's

Tijdens het kiezen van de engine waren er drie randvoorwaarden, namelijk:

1. gratis voor educatie, of goedkoop genoeg voor DPI om aan te schaffen
2. Oculus en GearVR ondersteuning
3. Een VS systeem

Er is gekeken naar de volgende engine's die aan de bovenstaande criteria voldoen.

- Unity¹
- CRYENGINE²
- Unreal Engine 4³

¹<https://unity3d.com>

²<https://www.cryengine.com>

³<https://www.unrealengine.com/what-is-unreal-engine-4>

2.1.1 Unity

2.1.1.1 Licentie

Unity heeft een educatie licentie waaronder het grootste gedeelte van de engine gratis gebruikt kan worden⁴, maar rekening houdend met de interesses van DPI, en eventuele vervolg projecten die zij willen ondernemen, zal de licentie minimaal 75 € per maand kosten. Dit is exclusief de kosten van Android builds.

2.1.1.2 Oculus en GearVR ondersteuning

Van de drie engine's heeft Unity de meest uitgebreide platform support.⁵ Daarnaast is Unity vaak een van de eerste keuzes om nieuwe technieken mee te implementeren.⁶ De reden voor deze keuze is het gemak waarop plugins gemaakt kunnen worden en de voordelige / gratis prijsstelling van Unity. Dit maakt Unity een populaire keuze voor experimenteren.

2.1.1.3 Virtual Scripting

Unity heeft zelf niet een ingebouwd VS systeem maar via een aantal plugins kan dit wel worden toegevoegd. Deze plugins zitten wel vast aan de limitaties van een Unity plugin. Daarnaast wordt dit niet officieel gesupport door Unity zelf.

2.1.1.4 Gebruiksgemak

Op gebruiksgemak (in serieuze projecten), fidelity en performance loopt Unity ver achter op de UE4 en CRYENGINE. Onderhoudbaarheid en uitbreidbaarheid van Unity is meestal rampzalig door de manier waarop code voor Unity geschreven moet worden. Het programmeren wordt namelijk in een component style in c-sharp gedaan. Het programmeren in c-sharp is voor veel mensen vertrouwd omdat dit vaak een bekende taal is. Maar het component systeem en de manier waarop Unity relaties en state afhandelt zorgt dat er extreem snel spaghetti code ontstaat.

⁴<https://unity3d.com/get-unity>

⁵<https://unity3d.com/unity/multiplatform>

⁶https://developer.leapmotion.com/documentation/orion/unity/Unity_Overview.html

2.1.2 CRYENGINE

2.1.2.1 Licentie

De CRYENGINE licentie bestaat uit een “Pay what you want” model.⁷ Daarnaast bied Crytek een “CRYENGINE Insider Membership” aan voor 50 € of 150 €.⁸

2.1.2.2 Oculus en GearVR ondersteuning

Op het moment van schrijven ondersteund de CRYENGINE wel de Oculus maar niet officieel de GearVR.⁹ Android ondersteuning is mogelijk maar niet officieel ondersteunt door Crytek. Dit zal het lastig maken om GearVR te ondersteunen.

2.1.2.3 Visual Scripting

De CRYENGINE heeft een ingebouwd VS systeem genaamd Flow.¹⁰ Dit systeem heeft helaas geen ondersteuning voor overerving en de koppeling met C++ bestaat alleen uit het maken van nieuwe nodes en het aanspreken van graphs in C++.

2.1.2.4 Gebruiksgemak

De CRYENGINE staat bekend als een moeilijke engine om in te beginnen maar heeft ondertussen bewezen een goede keuze te zijn voor game ontwikkelaars en grotere teams.¹¹

2.1.3 Unreal Engine 4

2.1.3.1 Licentie

Het gebruik van de UE4 is compleet gratis voor educatie en niet-game applicaties zoals architectuur, visualisatie, films etc.¹² Dit betekend dat zowel tijdens het afstuderen als mogelijke vervolg projecten van DPI er geen licentie kosten betaald hoeven te worden.

⁷<https://www.cryengine.com/get-cryengine>

⁸<https://www.cryengine.com/get-CRYENGINE/service-packages>

⁹<https://www.cryengine.com/features/platforms>

¹⁰<http://docs.cryengine.com/display/SDKDOC2/Flow+Graph+Editor>

¹¹https://en.wikipedia.org/wiki/List_of_CryEngine_games

¹²<https://www.unrealengine.com/what-is-unreal-engine-4>

2.1.3.2 Oculus en GearVR ondersteuning

UE4 ondersteunt zowel de Oculus als Gear VR.¹³ Daarnaast ondersteunt het ook de meeste randapparatuur voor VR zoals Leap Motion en Kinect. Updates hiervoor komen vaak wel later dan voor Unity.

2.1.3.3 Visual Scripting

Unreal heeft een uitgebreid VS systeem wat nauw samenwerkt met C++.¹⁴ Alle functies die in C++ beschikbaar zijn, zijn beschikbaar via Blueprints. Van elke Blueprint kan ook direct de source code ingesprongen worden, zelfs voor de functies uit de UE4 libraries.

Het koppelen van C++ aan Blueprints is vrij simpel en ondersteund ook het creëren van events in C++ en functionaliteit hieraan via Blueprints te koppelen.

2.1.3.4 Gebruiksgemak

De leercurve van UE4 is redelijk steil en binnen een paar weken is het mogelijk het belangrijkste te leren. Een punt om rekening mee te houden is dat het zomaar iets maken in de UE4 vaak verkeerd uitpakt. Daarom is het belangrijk de documentatie, van het onderdeel waar je mee bezig bent, goed te lezen.

Als de basis principes eenmaal onder de knie zijn kan er extreem snel ontwikkeld en geprototyped worden met de UE4 door de implementatie van VS en een strakke en goed uitgedachte UI.

2.2 Conclusie

De CRYENGINE valt af vanwege zijn gebrek aan officiële GearVR support.

Untiy heeft de minst steile learning curve maar geavanceerde 3D technieken zullen uiteindelijk toch geleerd moeten worden om de kwaliteit te behalen die nodig is voor VR. Het visuele scripting systeem van Unity is beperkt en mist de kracht en flexibiliteit die nodig is om deze scriptie succesvol af te ronden.

¹³<https://www.unrealengine.com/vr-page>

¹⁴<https://docs.unrealengine.com/latest/INT/Engine/Blueprints/GettingStarted/index.html>

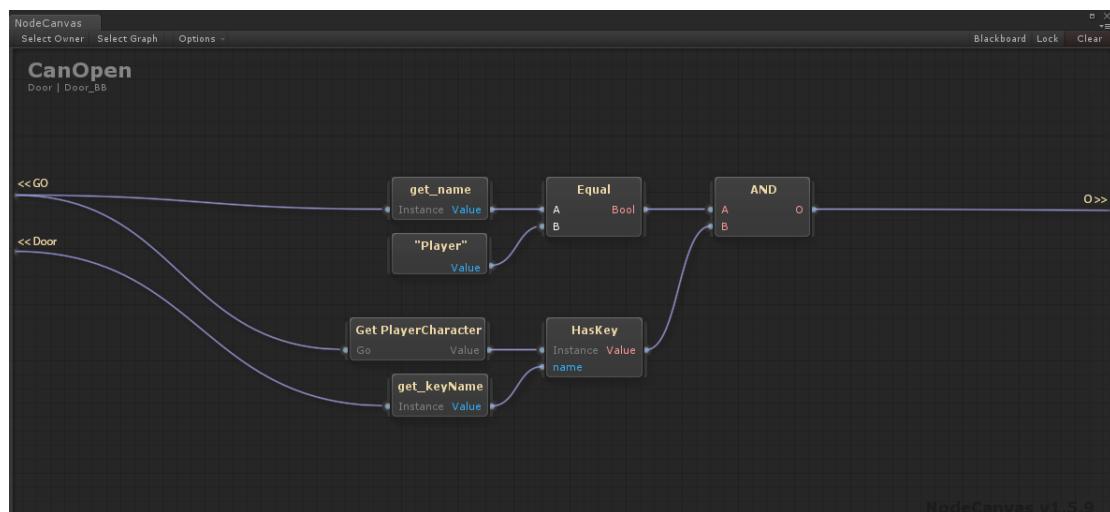
Het VS systeem van UE4 is het meest geschikt. Het integreert naadloos met de engine en word door Epic zelf ondersteund en ontwikkeld. Het VS systeem is bijna net zo flexibel als het schrijven van code en geeft genoeg vrijheid voor dit onderzoek. Daarnaast ondersteunt UE4 alle grote platformen en de laatste ontwikkelingen in render technieken.

Hoofdstuk 3

Visual Scripting

In dit hoofdstuk wordt een introductie gegeven van een visuele programmeer taal. Daarna wordt als voorbeeld een vergelijking gemaakt tussen C++ en een visuele taal.

VS maakt het mogelijk om logica op een visuele manier weer te geven in plaats van tekstueel te schrijven. Er is minder kennis van de onderliggende werking van computersystemen nodig dan voor tekstueel programmeren. Hierdoor kan er zonder kennis van computers en hun werking al snel mee gewerkt worden door niet programmeurs.



FIGUUR 3.1: Voorbeeld van een visuele script taal.

Daarnaast worden de mogelijkheden van de visuele programmeer taal ook vaak gelimiteerd om het de gebruiker makkelijker te maken en het programma te beschermen tegen fouten van de gebruiker. Als de limitaties van VS een probleem worden was het vaak beter geweest om toch te kiezen voor een tekstuele taal.

3.1 Use Case's

Door de jaren heen zijn er voor verschillende doeleinden visuele programmeertalen gemaakt. Een aantal voorbeelden zijn:

- Data flow in applicaties
- State flow voor onder andere animaties
- Geluids effecten programmeren
- Gameplay Logica
- Programmeren leren aan beginners

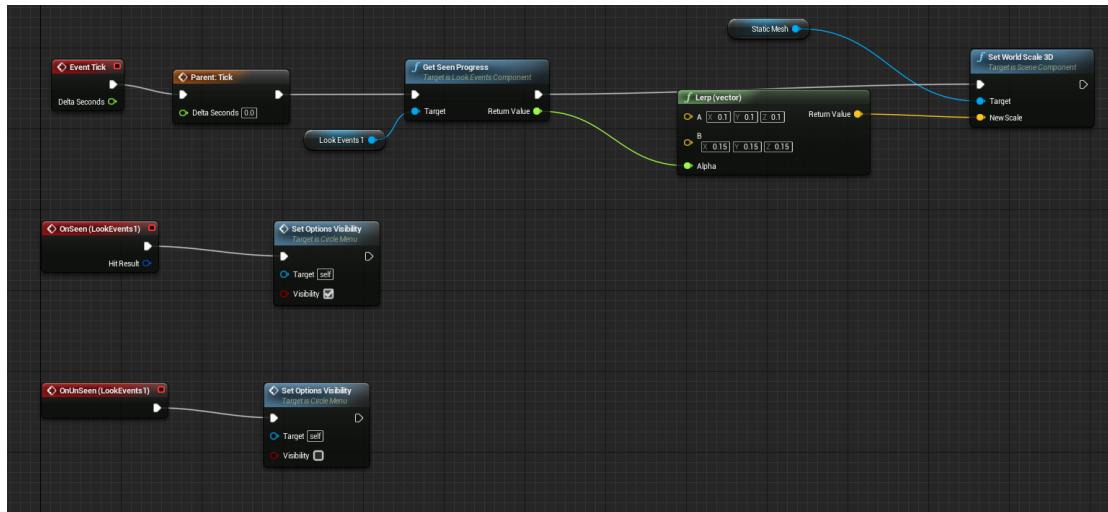
In deze scriptie wordt er gefocust op het gebruik van VS voor gameplay logica.

3.2 Blueprints

Blueprints is het visuele scripting systeem van UE4 en heeft de doelstelling van deze scriptie mogelijk gemaakt. De beschrijving van Unreal zelf is als volgt:

“Blueprints are special assets that provide an intuitive, node-based interface that can be used to create new types of Actors and script level events; giving designers and gameplay programmers the tools to quickly create and iterate gameplay from within Unreal Editor without ever needing to write a line of code.”

Blueprints ziet er als volgt uit:



FIGUUR 3.2: Voorbeeld van een Blueprint.

Kort samengevat is dat de rode nodes Events zijn en dus aangeven wanneer iets gebeurd. De witte lijnen bepalen de volgorde waarin de nodes worden uitgevoerd. De blauwe nodes zijn functies terwijl de rest variabelen zijn of pure transformatie van data.

3.3 Gameplay Logica

Het programmeren van gameplay in een spel bestaat voor een groot gedeelte uit de vragen “Wanneer moet iets gebeuren”, “Wat moet er gebeuren” en “Hoe moet dit gebeuren”. De vragen “Waarom moet iets gebeuren” en “Waar moet dit gebeuren” komen het meeste voor in de gameplay logica van een spel, vaak hebben deze vragen ook een makkelijk antwoord. Maar deze vragen zijn lastig om te beantwoorden in tekstuele code.

3.3.1 Het afvuren van een projectiel

In deze paragraaf wordt een voorbeeld van deze drie vragen uitgewerkt. Het voorbeeld laat de logica zien van het schieten van een projectiel in C++. Dit is gebaseerd op het standaard voorbeeld van een speler uit de UE4.

3.3.1.1 Tekstuele Implementatie

In een C++ header bestand registreren wij de volgende functie in onze speler classe.

```
1 void OnFire();
```

LISTING 3.1: Registratie van de OnFire functie

Dan tijdens het opzetten van de input voor ons karakter laten we weten wanneer we een projectiel willen schieten.

```
1 if( EnableTouchscreenMovement( InputComponent ) == false )
{
3   InputComponent->BindAction(
4     "Fire",
5     IE_Pressed,
6     this,
7     &Adpi_unreal_colosseumCharacter::OnFire
8   );
9 }
```

LISTING 3.2: Koppelen van de Fire actie aan de OnFire functie

Maar omdat de “fire” actie niet werkt op een touch interface moeten we de onFire zelf afvuren als iemand het scherm aanraakt (De logica achter het registeren van touch events wordt niet getoond maar is wel aanwezig in de speler classe).

```
1 void Adpi_unreal_colosseumCharacter::EndTouch( const ETouchIndex::Type
2   FingerIndex, const FVector Location)
3 {
4   if( TouchItem.bIsPressed == false )
5   {
6     return;
7   }
8   if( ( FingerIndex == TouchItem.FingerIndex ) && ( TouchItem.bMoved ==
9     false ) )
10  {
11    OnFire();
12  }
13  TouchItem.bIsPressed = false;
14 }
```

LISTING 3.3: Aanroepen van de OnFire functie tijdens het EndTouch event

Vervolgens definiëren we OnFire als volgt

```

1 void Adpi_unreal_colosseumCharacter::OnFire()
2 {
3     // try and fire a projectile
4     if (ProjectileClass != NULL)
5     {
6         const FRotator SpawnRotation = GetControlRotation();
7         // MuzzleOffset is in camera space, so transform it to world space
8         // before offsetting from the character location to find the final muzzle
9         // position
10        const FVector SpawnLocation = GetActorLocation() + SpawnRotation.
11        RotateVector(GunOffset);
12
13        UWorld* const World = GetWorld();
14        if (World != NULL)
15        {
16            // spawn the projectile at the muzzle
17            World->SpawnActor<Adpi_unreal_colosseumProjectile>(ProjectileClass,
18            SpawnLocation, SpawnRotation);
19        }
20    }
21
22    // try and play the sound and fire animation if specified
23    ...
24}

```

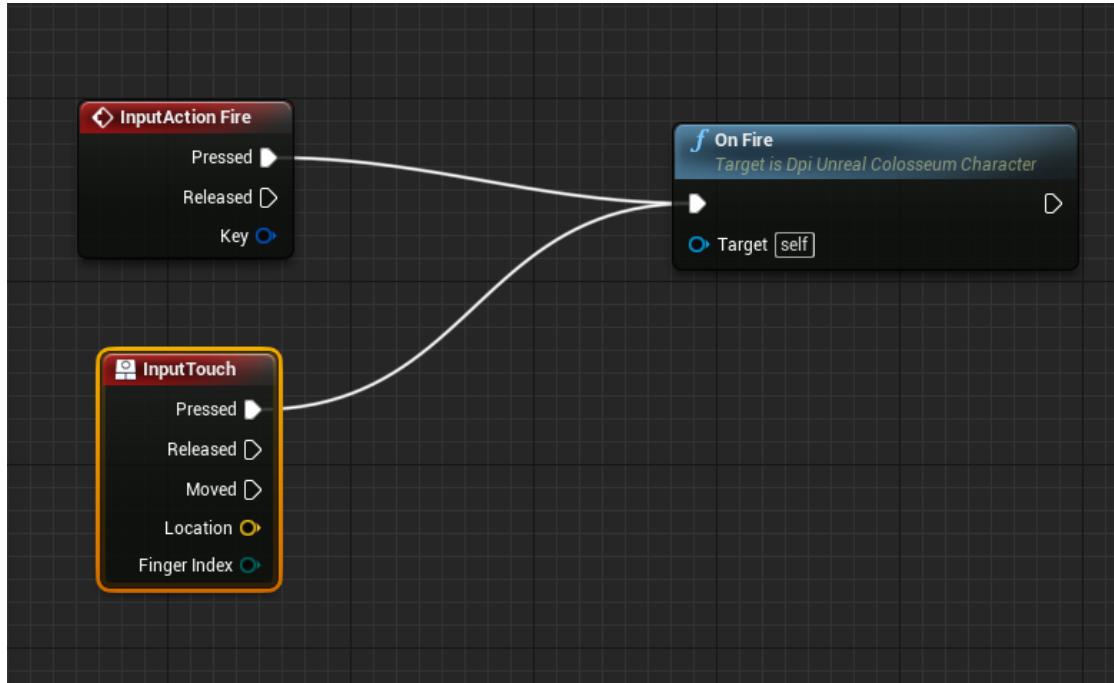
LISTING 3.4: Implementatie van de OnFire functie

Om de logica te implementeren voor het vuren van de kogel is op vier verschillende plekken de logica gebruikt. De implementatie is verspreid in een bestand van 218 regels C++.

3.3.1.2 Visuele Implementatie

De “Hoe moet dit gebeuren” is een complexe vraag die prima beantwoord wordt in code. Vooral omdat er verschillende logica achter elkaar plaatsvindt zoals geluid afspelen / animatie starten, en een aantal wiskundige berekeningen.

De “Wanneer” vraag is veel makkelijker te beantwoorden. Namelijk als de fire actie uitgevoerd wordt of als er op het scherm gedrukt wordt. De logica in blueprints ziet er als volgt uit (in de event graph van een playerCharacter).



FIGUUR 3.3: Afvuren van een projectiel in Blueprints.

In tegenstelling tot de C++ code is de logica van de “Wanneer” vraag dit maal niet verspreid en is het in een oogopslag duidelijk wat er voor zorgt dat er een projectiel geschoten wordt.

Hoofdstuk 4

Uiteenzetting Vooronderzoek

In dit hoofdstuk wordt antwoord gegeven op de volgende vragen door middel van de onderzoeksmethoden beschreven in 1.3.3.

- Wat is de huidige kennis van computerlogica onder de werknemers van DPI?
- Is er extra training nodig om met de ontwikkelomgeving aan de slag te gaan?
- Welke componenten zullen gemaakt moeten worden?
 - Wat zijn de huidige taken van een programmeur bij het maken van een virtuele omgeving?
 - Welke tools bestaan er al?
 - Wat zijn de juiste abstracties van de componenten?
- Is het mogelijk de componenten cross-platform te maken?
 - Welke platformen zijn relevant?
 - Wat zijn de verschillende mogelijkheden van deze platforms?

4.1 Wat is de huidige kennis van computerlogica onder de werknemers van DPI?

Om een idee te krijgen van de huidige kennis van de niet-programmeurs is er een enquête uitgezet over programmeur terminologie H.3 in combinatie met een interview over vorige werk ervaringen. H.2.

De enquête schept een goed beeld over hoe groot de vorige programmeur-ervaring van de niet-programmeurs is. De correcte terminologie begrijpen van een onderwerp toont namelijk aan dat de er naast het in aanraking met het onderwerp ook bewust kennis is gezocht of gedeeld. En het interview geeft context aan de enquête.

4.1.1 Conclusie

Uit de enquête en het interview blijkt dat er nauwelijks kennis is van programmeren. Wel is er ervaring met 3D software wat het werken in het begin met UE4 makkelijker zal maken en er sneller gefocust kan worden op het logica aspect.

4.2 Is er extra training nodig om met de ontwikkelomgeving aan de slag te gaan?

Omdat er geen kennis is over het Blueprint systeem van UE4 of eerder contact is geweest met programmeren, is er besloten om een aantal workshops te geven waarin de niet-programmeurs de basis van Blueprints leren.

4.2.1 Workshops

In de workshop A is er een introductie gemaakt in de UE4 en Blueprints. Omdat de niet-programmeurs van een 3D modelling achtergrond komen zijn de verschillen in best practises tussen een game engine en 3d software benadrukt.

In de workshop B krijgen de niet-programmeurs uitleg over Blueprints en de VRInteractions plugin en maken zij zelfstandig een opdracht.

In de workshop C wordt er een complexe opdracht gemaakt en wordt er verwacht dat de niet-programmeurs zelfstandig problemen kunnen oplossen.

4.3 Welke componenten zullen gemaakt moeten worden?

De ontwikkelde componenten zijn gebaseerd op de problemen die de niet-programmeurs tegen komen tijdens het maken en opzetten van VR omgevingen. DPI heeft eerder een volledige VR ervaring gemaakt in Unity, maar deze was niet interactief en gaf weinig input voor het kiezen van componenten.

Tijdens het schrijven van de scriptie zijn de volgende VR omgevingen gemaakt met de VRInteractions plugin:

- Een fly-trough door een menselijk lichaam
- Een appartement waarvan o.a. het meubilair en de vloer dynamisch aangepast kon worden
- Een demo van een machine uit een fabriek
- Een virtuele omgeving van een tentoonstelling
- Een leeromgeving rond het colosseum
- Een demo omgeving van de VRInteractions plugin

Op basis van de functionaliteit en problemen van deze projecten is de VRInteractions plugin ontwikkeld.

4.3.1 Wat zijn de huidige taken van een programmeur in het maken van een Virtuele omgeving?

DPI focust zich voornamelijk op het maken van exposities in VR. Een voorbeeld van een use-case is een opstelling op een beurs vervangen door een VR omgeving.

Om de omgeving in te stellen, te starten of tussen omgevingen te navigeren wordt een menu gebruikt waarin de gebruiker opties kan kiezen. Daarnaast wordt er tijdens de demo zelf interactie gebruikt om op een betrokken manier informatie te tonen.

De taken van een programmeur komen in de meeste exposities neer op:

- Opzetten van het project
- Programmeren van menu en transitie van levels
- Programmeren van de triggers voor animaties

4.3.2 Welke tools bestaan er al?

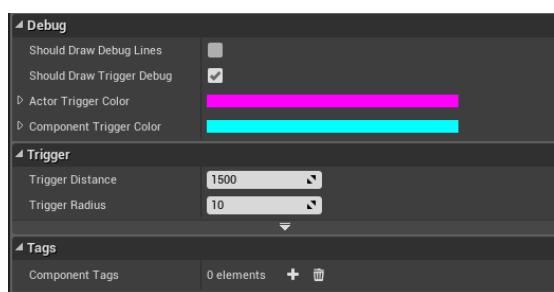
Op het moment van schrijven zijn er een aantal VR interactieve producties gemaakt in UE4. De producties die het meest lijken op de expositie projecten van DPI zijn:

- The Rose and I ¹
- Henry ²
- The Body VR ³

Deze producties gebruiken voornamelijk het kijken naar elementen als interactie vorm. Geen van de projecten heeft echter een toolkit gebruikt of gesproken over implementatie van de interactie.

4.3.3 Wat zijn de juiste abstracties van de componenten?

Blueprints maakt het mogelijk om bepaalde opties te groeperen en als geavanceerd te labelen.

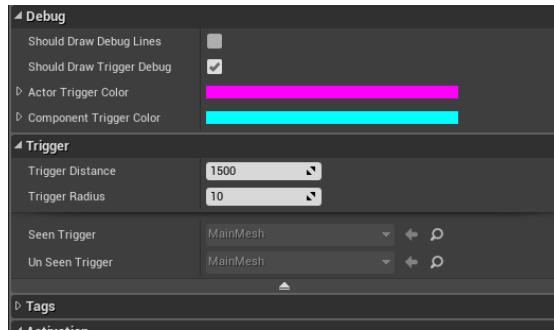


FIGUUR 4.1: Voorbeeld van de geavanceerde opties die verborgen zijn.

¹<http://store.steampowered.com/app/396060/>

²<https://storystudio.oculus.com/en-us/henry/>

³<https://www2.oculus.com/experiences/rift/967071646715932/>



FIGUUR 4.2: Voorbeeld van de geavanceerde opties die zichtbaar zijn.

Dit zorgt ervoor dat de componenten zo abstract mogelijk gecodeerd kunnen worden en de opties die voor onduidelijkheid zorgen verborgen kunnen worden. Ook is het mogelijk om versimpelde versies van functies te maken die aanspreekbaar zijn in Blueprints.

4.3.4 Conclusie

Op basis van de gemaakte projecten en de workshops zijn de volgende componenten ontwikkeld voor de VRInteractions plugin:

- Een game mode voor een demo waarin de speler kan rondlopen en een voor een fly-through demo
- Een speler klasse per game mode
- Een component die verantwoordelijk is voor de input van de speler
- Een component die verantwoordelijk is voor de camera van de speler
- Een component die events afvuurt als er naar een object gekeken wordt
- Een HUD die het mogelijk maakt om op basis van de look events informatie te tonen
- Een 3D menu die zijn opties in een cirkel toont
- Een object wat verplaats kan worden door middel van ernaar te kijken.

Deze zijn in code zo abstract mogelijk opgezet en vervolgens door groepering van de instellingen en het toevoegen van hulp functies versimpelt in Blueprints.

4.4 Is het mogelijk de componenten cross-platform te maken?

De huidige componenten zijn geschreven met het kijken naar objecten als interactie vorm. Dit is een interactie vorm die op elk VR device mogelijk is.

Voor de GearVR, Vive, en de Oculus kan er gebruik gemaakt worden van de camera positie van de speler. Voor VR brillen zoals de Cardboard word de positie van de camera niet automatisch gecorrigeerd en moet de headtracking zelf geïmplementeerd worden.

Voor elke bril die de camera van de speler automatisch verwerkt werken de componenten out of the box.

4.4.1 Welke platforms zijn relevant?

De huidige VR headsets die in productie zijn, dus niet in een beta fase, zijn de GearVR, Oculus, Vive en de Cardboard. De Oculus en Vive zijn geschikt voor uitgebreide producties van hoge kwaliteit maar de GearVR is gelimiteerd aan de grafische kracht van de gebruikte mobiel, maar werken alleen op een telefoon uit de Samsung Galaxy serie.

De Cardboard is ook gelimiteerd aan de kracht van de mobiel die gebruikt word maar kan met Android telefoon gebruikt worden. Dit limiteert de grafische mogelijkheden zover dat er geen comfortabele framerate en responsiveness bereikt kan worden.

4.4.2 Wat zijn de verschillende mogelijkheden van deze platforms?

Omdat de controllers van de GearVR, Oculus en de Vive hardwarematig van elkaar verschillen is het erg lastig om deze alle drie te ondersteunen.

De GearVR maakt namelijk gebruik van een touchpad op de bril of een aangesloten gamepad. De Oculus en Vive hebben beiden eigen controllers ontwikkeld maar deze bevatten verschillende knoppen en door het hardwarematige verschil moet hier dubbele code voor geschreven worden.

4.4.3 conclusie

Door de interactie te beperken tot het kijken naar objecten is het mogelijk voor de plugin om alle huidige headsets in productie te ondersteunen op de Cardboard gebaseerde headsets na.

Voor dit onderzoek worden alle brillen ondersteund op de Cardboard na. De GearVR heeft lage instap kosten voor consumenten en is makkelijk portable. De Oculus en Vive hebben hogere grafische mogelijkheden.

Hoofdstuk 5

Blueprints en C++

In dit hoofdstuk wordt gameplay logica in drie aspecten verdeeld en wordt voor elk aspect naar de voor en nadelen van C++ en Blueprints gekeken.

Aan de hand van deze vergelijkingen wordt een workflow gekozen waarin het beste uit C++ en Blueprints wordt gecombineerd. In dit hoofdstuk wordt antwoord gegeven op de vraag "Hoe kan er een koppeling met VS gemaakt worden die intuïtief is voor niet programmeurs?".

5.1 Gameplay

Om de scheiding tussen C++ en Blueprints concreet te maken is de gameplay logica verdeeld in de volgende vragen:

- Wanneer moet iets gebeuren?
- Wat moet er gebeuren?
- Hoe moet dit gebeuren?

Door deze scheiding wordt het makkelijker om de keuze tussen een C++ en een Blueprint implementatie te maken en kunnen er een aantal richtlijnen opgezet worden.

5.1.1 Wanneer vraag

De wanneer vragen zijn vaak makkelijk te beantwoorden. Bijvoorbeeld als de speler geraakt wordt door een projectiel moet geluid x afgespeeld worden, maar dit is moeilijk te coderen door hun asynchrone natuur. Een van de krachtigste voordeelen van VS is dat de flow van een programma uitgedrukt kan worden door middel van de lijnen tussen nodes.

Om het verschil tussen asynchrone logica in C++ en Blueprints duidelijk te maken wordt het afspeLEN van een geluid geïmplementeerd nadat een speler dood gaat.

5.1.1.1 C++

We registeren eerst een functie die aangesproken kan worden door de timeout en het geluid wat afgespeeld wordt in de header van de character.

```

1 // Plays a sound x seconds after the death of the player
2 void AfterDeathSoundTimeOut();
3
4 // Sound to play each time we fire
5 UPROPERTY(EditAnywhere, BlueprintReadWrite, Category=Gameplay)
6 class USoundBase* DeathSound;

```

LISTING 5.1: Registratie van de timeout functie en het geluid

Vervolgens wordt deze functie geïmplementeerd in de character.cpp

```

void Adpi_unreal_colosseumCharacter :: AfterDeathSoundTimeOut()
{
    UGameplayStatics :: PlaySoundAtLocation( this , FireSound , GetActorLocation()
);
}

```

LISTING 5.2: Registratie van de timeout functie en het geluid

En wordt de timeout voor het geluid gezet tijdens het dood gaan van de speler.

```

void Adpi_unreal_colosseumCharacter :: OnDeath( const FDeathReason Reason )
{
    // death logic
    ...
}

```

```

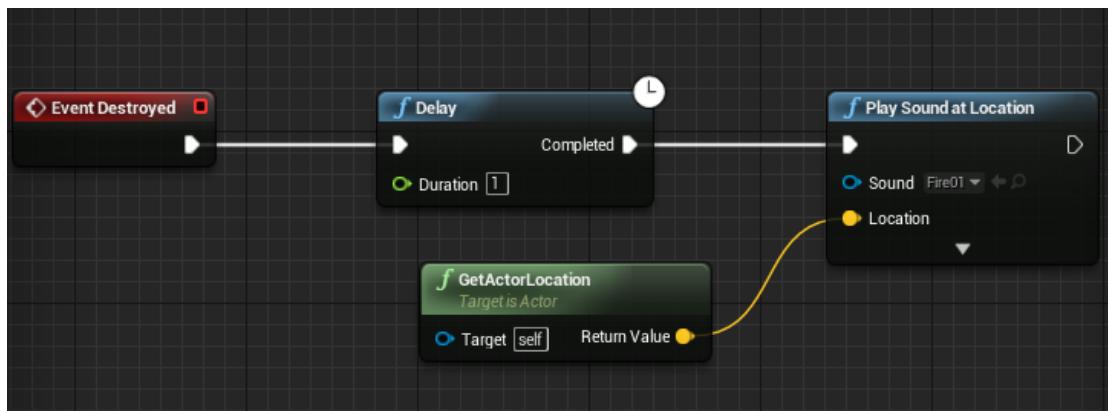
6   FTimerHandle UnusedHandler= FTimerHandle();
7   GetWorld()->GetTimerManager().SetTimer(
8     UnusedHandler,
9     this,
10    &Adpi_unreal_colosseumCharacter::AfterDeathSoundTimeOut,
11    1.0 f
12  );
13 }
```

LISTING 5.3: Implementatie van de OnDeath functie

We zien hier dat de gerelateerde code op drie verschillende plekken komt te staan, tussen niet relevante code in. Pas na het lezen van de code op deze drie plekken wordt het duidelijk wat de complete functionaliteit is. Er zijn hier natuurlijk hulpmiddelen voor zoals opmerkingen boven code plaatsen maar bij elke extra taak die, asynchroon, uitgevoerd moet worden wordt de code complexer en moeilijker te begrijpen. De lezer moet namelijk alle gerelateerde functionaliteit in zijn geheugen hebben.

5.1.1.2 Blueprints

In Blueprints zou deze logica er als volgt uit zien:



FIGUUR 5.1: Afspelen van een geluid na het doodgaan van de speler in Blueprints.

In de Blueprint implementatie is het in een oogopslag duidelijk dat er een geluid afgespeeld wordt op de locatie van de speler een seconde nadat deze dood gaat. De logica bevindt zich op de dezelfde plek en de witte lijnen geven de flow van de logica aan.

Een ander groot verschil dat we hier zien is, dat er voor iets simpels als een vertraging in tekstuele code naast de standaard kennis van de C++ syntax ook kennis nodig is van de volgende concepten:

- Pointers
- Pass by reference
- Function references
- Namespaces
- Out parameters (de UnusedHandler)
- Floats
- Types
- Macros

Terwijl in Blueprints al deze concepten verborgen zijn in de nodes. Het verbergen van deze logica heeft als gevolg dat er zonder programmeerkennis de wanneer-logica geïmplementeerd kan worden.

5.1.2 Wat

Het plaatsen van de wat-logica in Blueprints of C++ is een lastigere keuze. Voor logica die de niet-programmeurs schrijven is C++ geen optie en moet dit wel in Blueprints, maar programmeurs kunnen wel de keuze maken tussen C++ en Blueprints.

Het probleem ontstaat voornamelijk bij complexe conditionele logica.

5.1.2.1 Conditionele Logica

Als we de conditionele logica van het afvuren van de events van een LookEventsComponent in C++ en Blueprints met elkaar vergelijken zijn beide varianten moeilijk te lezen.

- Bijlage 1: Tick functie van de LookEventsComponent F
- Bijlage 2: Conditional logic van Tick functie van LookEvents in Blueprints G

Voor iemand die niet codeert ziet de Blueprints variant er waarschijnlijk begrijpbaarder uit maar de complexiteit komt voornamelijk door de logica zelf en in tekstuele code zijn er een aantal manieren om dit soort constructies kleiner te maken zoals:

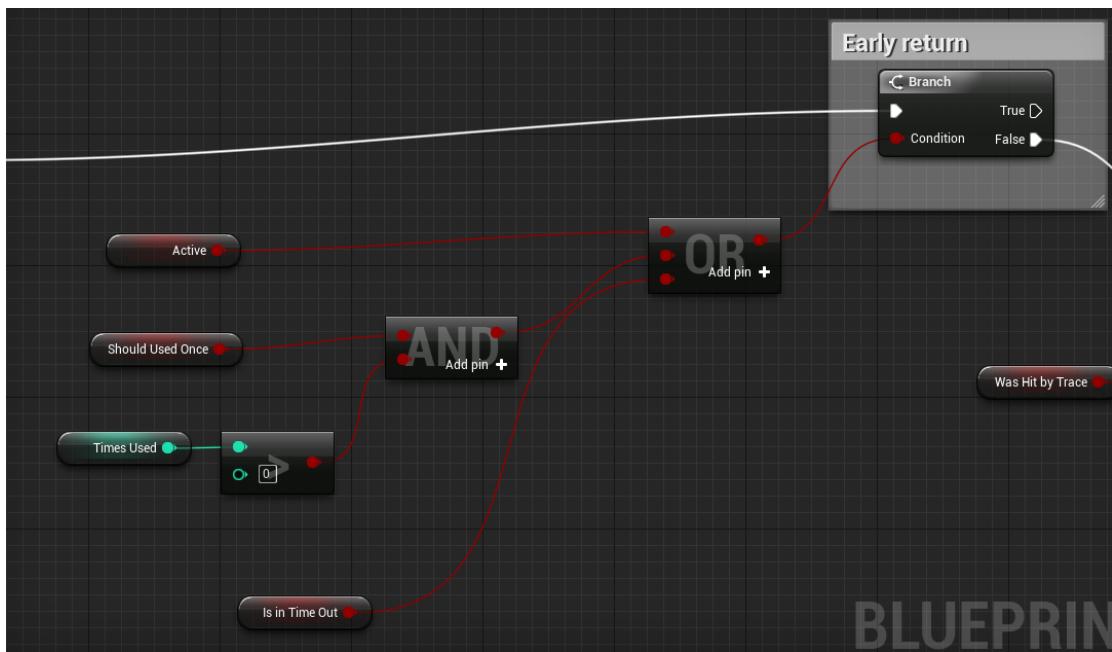
```

1 if ( bActive != true || ( bShouldUsedOnce && TimesUsed > 0 ) || bIsInTimeOut
     == true )
{
3   return;
}

```

LISTING 5.4: Early return in een complexere if statement in C++

In vergelijking met



FIGUUR 5.2: Early return in een complexere if statement in Blueprints.

Een ander voorbeeld is de volgende functie die bepaald of de trigger van een LookEventsComponent onderdeel was van een trace.

```

FHitResult ULookEventsComponent :: WasHitByTrace( const TArray<FHitResult>
                                                HitResults )
2 {
  UObject* Trigger = ( bIsBeingWatched ) ? UnSeenTrigger : SeenTrigger ;

```

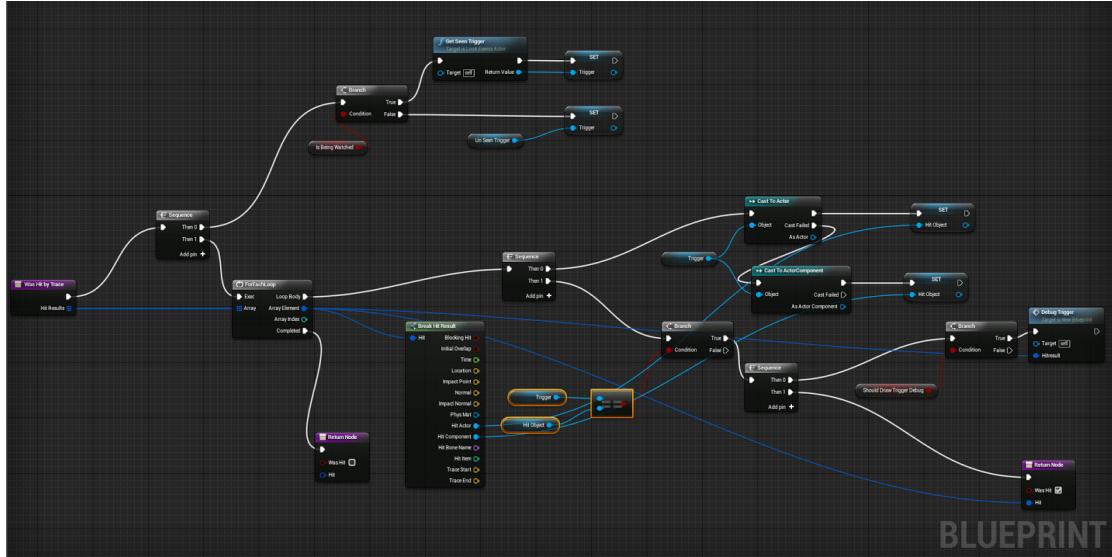
```

4
5     for (auto& HitResult : HitResults)
6     {
7         UObject* HitObject = (Trigger->IsA(AActor::StaticClass())) ? Cast<
8             UObject>(HitResult.GetActor()) : Cast<UObject>(HitResult.GetComponent());
9     }
10
11    if (HitObject == Trigger)
12    {
13        if (bShouldDrawTriggerDebug) {
14            DebugTrigger(HitObject, 0.1f);
15        }
16    }
17
18    return FHitResult();
}

```

LISTING 5.5: Een functie die kijkt of de huidige component geraakt is door een trace

In vergelijking met:



FIGUUR 5.3: Een Blueprint die kijkt of het huidige component geraakt is door een trace.

Voor complexe conditionele logica is C++ sneller te begrijpen en heeft C++ meer mogelijkheden om dit te versimpelen.

5.1.3 Hoe vraag

Met de hoe-vraag wordt de interne werking van een functionaliteit bedoelt waaronder ook de integratie met de rest van het programma, denk aan integratie met de rest van de engine, lifecycles, overerving en performance.

Complexe algoritmes zijn altijd makkelijker in C++ voor zowel leesbaarheid als onderhoudbaarheid. Daarnaast is de scheiding van algoritmes met de rest van de code belangrijk voor het scheiden van complexiteit.

5.1.4 Onderhoud en Performance

Naast het schrijven van de logica zijn performance en onderhoud twee onderwerpen die in elk aspect van een spel voorkomen en daarom ook onderdeel moeten zijn voor de keuze van workflow.

5.1.4.1 Onderhoud

Naast een kleine verbetering in leesbaarheid heeft het schrijven van logica in C++ een ander belangrijk voordeel. Namelijk uitgebreide debug-, zoek- en meta-informatie.

De error's die de UE4 genereerd zijn niet altijd even duidelijk. Vooral error's die pas tijdens het packagen van een project ontstaan. Het vinden van een foutieve Blueprint kan namelijk erg moeilijk zijn. Bijna elk element in de UE4 kan Blueprints code bevatten en elk element op zich kan die Blueprints weer in kleinere elementen verdelen. Deze elementen bevinden zich tussen alle andere soorten assets zoals Meshes, Animaties, Deciscion Trees, Materials, Textures etc. Het zoeken van een onbekende fout hierin is extreem lastig.

Als de foutieve code in C++ had gestaan kan er door middel van een stack trace gekeken worden in welke functie het fout gaat.

Daarnaast versimpelt het automatisch aanvullen van functienamen en informatie tonen over functies, door middel van de JavaDoc opmerking notatie, het schrijven van complexe code.

5.1.4.2 Performance

Een ander belangrijk onderdeel van de hoe-logica is performance. De performance van Blueprints is namelijk lager dan dat van C++. In de hoe- en wat-vraag is dit verschil niet meetbaar, maar in de wat-vraag kan het verschil merkbaar worden. In C++ is er ook veel meer controle over de manier waarop de computer logica interpreteert. In Blueprints is dit een stuk minder duidelijk. Dit zorgt ervoor dat op een lager niveau optimalisaties mogelijk zijn.

Ook is het makkelijker om complexe optimalisaties te schrijven op een onderhoudbare manier. Als er bijvoorbeeld een performance probleem ontstaat in het aantal raytraces dat de LookEventsComponents nodig heeft, is het mogelijk om een cache te schrijven voor de ray traces waar de LookEventsComponent een trace uit vraagt. Als deze trace dan al eerder door een andere LookEventsComponent berekend is krijgt hij de resultaten van die trace in plaats van een nieuwe trace te maken.

5.2 Labellen en groeperen van functies en opties

Om de opties duidelijker te maken worden de volgende UE4 flags gebruikt op variabelen:

- VisibleDefaultsOnly
- VisibleAnywhere
- EditAnywhere
- BlueprintReadOnly
- BlueprintReadWrite
- Category

Door gebruik te maken van deze flags word het mogelijk om variabelen af te schermen voor de niet-programmeurs. De BlueprintReadOnly flag word bijvoorbeeld gebruikt op het aantal keer dat een event getriggerd is. Dit is een variabele die in de code ophehoogd wordt, maar niet handmatig aanpasbaar moet zijn. Zie figuur 4.1 en figuur 4.2 voor een ander voorbeeld.

Voor functies gebruiken wij de volgende UE4 flags:

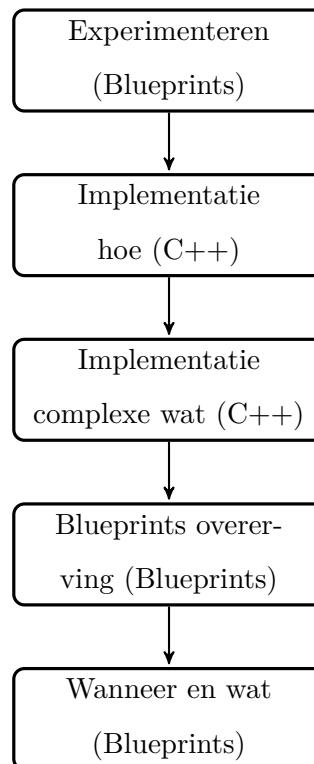
- BlueprintCallable
- BlueprintProtected
- Category

Door aan te geven welke functies wel en niet aangesproken kunnen worden vanuit Blueprints is het makkelijker om te zien wat er met een element mogelijk is.

5.3 Workflow

Voor onze workflow willen wij de complexiteit voor de niet-programmeurs zo laag mogelijk houden met zo veel mogelijk vrijheid. Daarnaast is flexibiliteit belangrijk voor het makkelijk en veilig experimenteren en itereren.

Gebaseerd op het verdelen van de gameplay logica in de drie vragen hoe, wat en wanneer wordt er voor de volgende workflow gekozen.



FIGUUR 5.4: Workflow voor het programmeren voor de Unreal Engine 4.

Elk component begint als een imperfecte Blueprint logica. In dit stadium is performance en onderhoud niet belangrijk, en hoeft het niet eens compleet functioneel te zijn. Er kan direct geëxperimenteerd en getest worden. Dit kan niet alleen waardevolle informatie opleveren voor het design aspect van een project, maar plaats ook de logica in de context van het project. Vaak komt hier gewenste functionaliteit uit die tijdens het bedenken over het hoofd gezien is.

Nadat het duidelijk is dat het component gebruikt gaat worden in het project en de gewenste functionaliteiten duidelijk zijn kan er een fatsoenlijke en efficiënte C++ implementatie geschreven worden.

De C++ implementatie wordt vervolgens overerft door een Blueprint die de aanvullende logica toevoegt.

Ook in gevallen waarin de Blueprint niet iets toe te voegen heeft, is het belangrijk om toch een Blueprint te maken van de C++ code. Er kan hierdoor namelijk makkelijk mee geëxperimenteerd worden en niet-programmeurs hebben hierdoor ook de mogelijkheid om iets toe te voegen of een standaard waarde te veranderen.

5.4 Versie Controle

Versie controle is een belangrijk onderdeel van elk ICT project en alhoewel dit buiten de scope van deze scriptie valt, wordt er hier toch een korte vermelding van gemaakt.

Versie controle is een belangrijk onderdeel in samen werken aan code en speelt een belangrijke rol in het onderhoud van code. Er zijn tools in UE4 om versie controle tools zoals git en svn op Blueprints, en assets toe te passen. Voor deze scriptie is de plugin ontwikkeld met behulp van git maar is er niet gekeken naar een manier waarop niet-programmeurs hier ook mee kunnen werken.

Tijdens de interviews, zie bijlage H, werd het duidelijk dat er geen bestaande kennis van versie controle aanwezig was in de visuele teams van DPI. Het kiezen, opzetten en leren van versie controle en de integratie hiervan met de UE4 valt buiten de scope van deze scriptie.

5.5 Conclusie

Door de gameplay logica te verdelen in Hoe, Wat en Waar zijn richtlijnen opgesteld voor het implementeren van gameplay. Voor de wanneer-vraag is Blueprints voor zowel de programmeur als de niet-programmeur altijd de beste optie. Niet alleen is het leesbaarder maar ook flexibeler doordat de logica op een logische manier achter elkaar staat. Voor de hoe-vraag is C++ altijd de beste optie. Onderhoudbaarheid en performance spelen hier een belangrijke rol. Voor de wat-vraag heeft de niet-programmeur geen optie maar de programmeur wel. Voor triviale code kan er gekozen worden voor Blueprints maar voor complexe conditionele logica heeft C++ de voorkeur.

Door deze workflow aan te houden en bewust gebruik te maken van de mogelijke metadata flags op functies en variabelen wordt een zo intuïtief mogelijke koppeling gemaakt tussen C++ en Blueprints.

Hoofdstuk 6

VRInteractions

In dit hoofdstuk worden de keuzes voor en de werking van de VRInteractions UE4 plugin besproken. Voor elk component in de plugin wordt de werking en mogelijkheden getoond. Daarnaast wordt voor elk component, waar relevant, besproken hoe de workshops de design keuzes beïnvloed heeft.

6.1 De plugin

De VRInteractions plugin is een combinatie van C++ code, Blueprints en overschrijfbare game modes. De plugin heeft als doel om de VR specifieke logica te verbergen voor gameplay logica zodat niet-programmeurs interactieve VR omgevingen kunnen maken.

De plugin is ontwikkeld met de workflow die beschreven is in paragraaf 5.3.

De plugin bevat de volgende componenten:

- Gamemodes
 - VRInteractionsFlyTroughGameMode
 - VRInteractionsGameMode
- Characters
 - VRInteractionBaseCharacter
 - VRInteractionCharacter

- VRInteractionsFlyCharacter
- VRInteractionsPlayerController
- VRInteractionsCameraManager
- VRInteractionsHud
- LookEventsComponent
- CircleMenu
- VRMovableMesh

6.2 Gamemodes

De plugin bevat de gamemodes VRInteractionsFlyTroughGameMode en VRInteractionsGameMode. Deze modes zijn beide een uitbreiding van de voorbeeld UE4 characters. Beide gamemodes zijn geschikt voor zowel de Oculus als de GearVR.

De gamemodes zijn bedoelt om de basis instellingen voor VR in te stellen en zijn voor de meeste applicaties voldoende. In het geval dat er iets in de gamemode aangepast moet worden is het mogelijk de instelling te overschrijven of de gehele gamemode over te erven.

Omdat er gebruik wordt gemaakt van een aantal input events is het aan te raden om een project altijd te beginnen als First Person Template.

6.2.1 VRInteractionsGameMode

De VRInteractionsGameMode maakt gebruik van de VRInteractionCharacter, VRInteractionsHud, VRInteractionsPlayerController en de VRInteractionsCameraManager.

De gamemode is bedoeld voor spellen waarin de speler kan rondlopen en zorgt ervoor dat de headset juist wordt geïnstalleerd en de juiste controllers ingesteld staan.

6.2.2 VRInteractionsFlyTroughGameMode

De VRInteractionsFlyTroughGameMode is een gestripte versie van de VRInteractionsGameMode en maakt gebruik van de VRInteractionsFlyCharacter, VRInteractionsHud, VRInteractionsPlayerController en VRInteractionsCameraManager.

Deze gamemode is bedoelt voor VR projecten waarin de speler met behulp van Matinee rondvliegt.

6.3 Characters

De VRInteractionsCharacters bevatten alle beweging- en camera instellingen zoals ooghoogte. De instellingen zijn gebaseerd op de VR guidelines van Epic¹.

6.3.1 VRInteractionBaseCharacter

De VRInteractionBaseCharacter is de basis voor de VRInteractionCharacter en de VRInteractionsFlyCharacter. Deze characters bevatten de instellingen voor de camera hoogte en enkele VR gerelateerde logica zoals het roteren van de speler op basis van zijn kijkrichting.

Ook wordt hierin de GearVR touch pad input afgehandeld.

Het is niet de bedoeling deze classe direct te implementeren. In plaats daarvan kan er gekozen worden voor de VRInteractionCharacter of de VRInteractionsFlyCharacter. Als er een nieuw soort character gemaakt moet worden kan die natuurlijk overerven van de VRInteractionBaseCharacter.

6.3.2 VRInteractionCharacter

De VRInteractionCharacter is bedoelt voor spellen waarin de speler rond kan lopen. De instellingen en functies zijn op basis van de UE4 VR guidelines en de testen van de VR demo gemaakt.

¹<https://docs.unrealengine.com/latest/INT/Platforms/VR/ContentSetup/index.html>

6.3.3 VRInteractionsFlyCharacter

De VRInteractionCharacter is bedoelt voor spellen waarin de speler op een rails loopt. Het is mogelijk deze character makkelijk in Matinee te gebruiken. Verschil met de VRInteractionCharacter, naast het niet registeren van controle gerelateerde input, is dat de camera niet op ooghoogte maar in het midden van de character geplaatst wordt. Dit maakt het maken van de Matinee sequences logischer.

6.4 VRInteractionsPlayerController

De VRInteractionsPlayerController is gebaseerd op de standaard PlayerController van Epic. De controller is verantwoordelijk voor het correct doorgeven van de input naar de ingestelde Character.

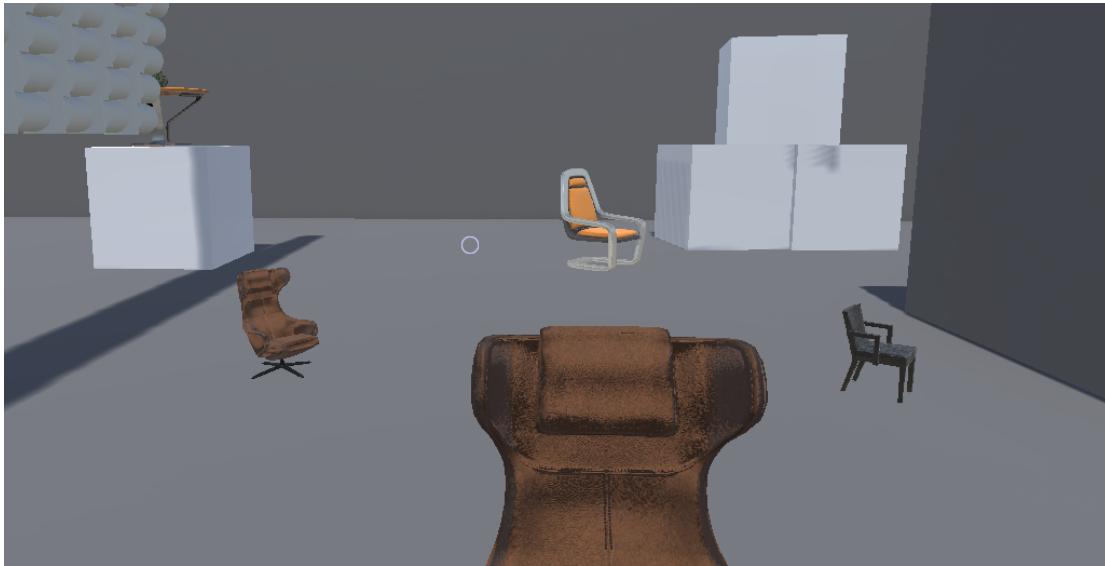
In de controller wordt er ook voor gezorgd dat de rotatie van de controller klopt met de rotatie van de headset op het moment dat de headset aangezet wordt. Zo kan de headset tijdens het spelen aan en uit gezet worden.

6.5 VRInteractionsCameraManager

De VRInteractionsCameraManager is gebaseerd op de standaard VRInteractionsCameraManager van Epic. Deze Manager is verantwoordelijk voor het instellen van camera instellingen zoals de FOV.

6.6 VRInteractionsHud

De VRInteractionsHud is de basis Head-up Display (HUD) die een cirkel tekent in het midden van het scherm om aan te geven dat er naar een actor met een LookEvent gekeken wordt.



FIGUUR 6.1: De cirkel die getekend wordt op het moment dat er naar een Actor met een LookEventsComponent gekeken wordt.

De HUD is afhankelijk van de VRInteractionsDataSingleton. Om te zorgen dat deze altijd correct wordt ingeladen kan de singleton als Game Singleton Class ingesteld worden. De singleton wordt namelijk gebruikt om informatie van de LookEventsComponents te verzamelen.

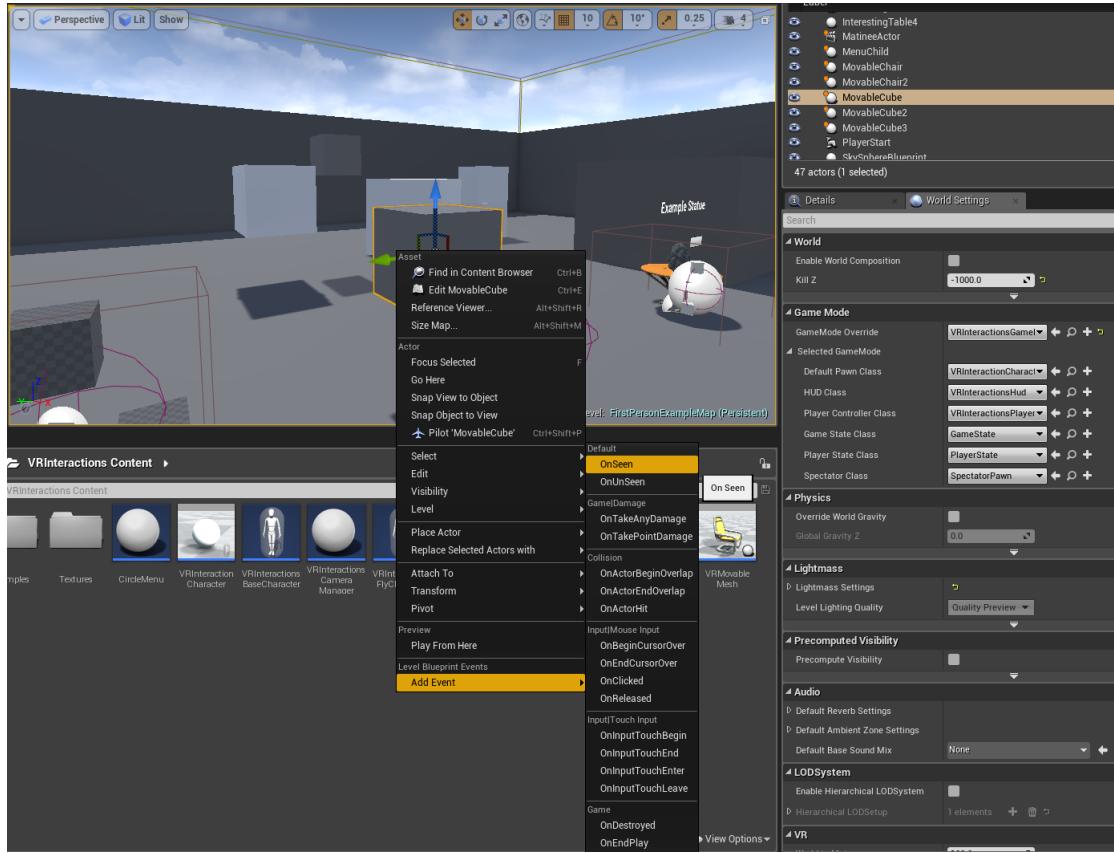
6.7 LookEventsComponent

De LookEventsComponent vuurt events af op het moment dat er naar de opgegeven trigger gekeken word. Het is mogelijk om meerdere LookEventsComponents aan dezelfde actor toe te voegen waardoor onder andere menu's op basis van kijken gemaakt kunnen worden.

De LookEventsComponent is de basis van de plugin en bijna alle nadere componenten zijn hierop gebaseerd.

6.7.1 Implementatie

Voor een aantal weken zijn de LookEvents als zowel een Actor als een Component geïmplementeerd. Het voordeel van een Actor implementatie is dat er vanuit world editor in het rechter muisknop menu event's gekoppeld kunnen worden.



FIGUUR 6.2: Voorbeeld van het kopelen van een event aan een actor.

Dit heeft als nadeel dat het event direct in het level Blueprint gekoppeld wordt wat aanmoedigt om hier de logica te plaatsen, terwijl dit vaak in een Blueprint Actor zou moeten gebeuren.

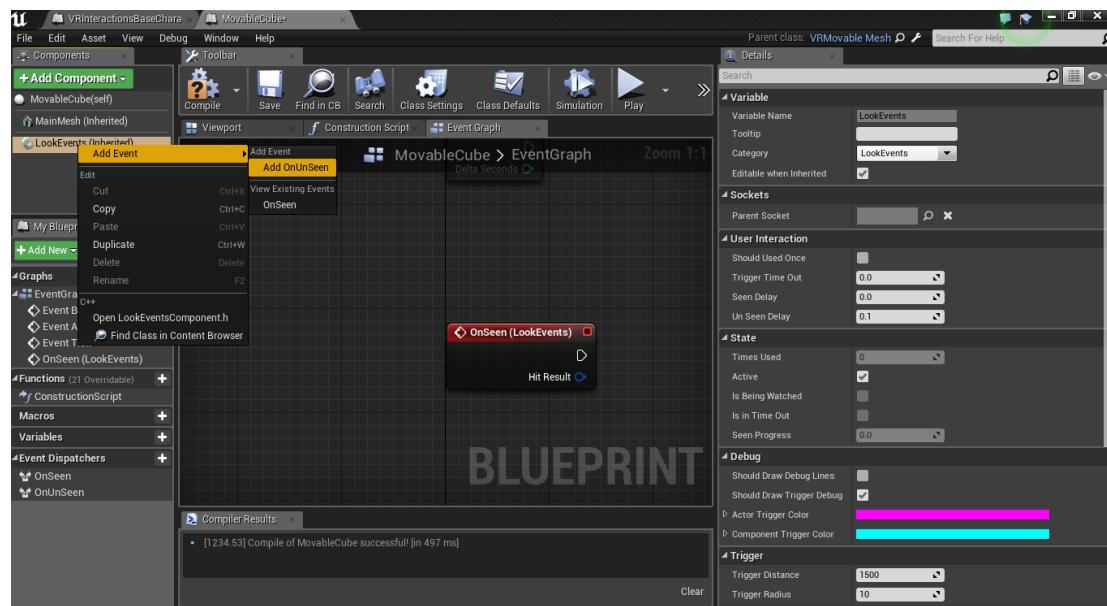
Ook is het lastig de Actor implementatie binnen een Blueprint Actor te gebruiken. Dit kan via een ActorComponent. Het ActorComponent maakt het niet mogelijk om instellingen via de editor te veranderen en verplicht daardoor om alle settings via Blueprints te zetten, wat omslachtig en fout gevoelig is.

Daarom is er voor de Component implementatie gekozen. Dit maakt het makkelijk om via de Blueprint editor events toe te voegen en kunnen er meerdere LookEvents gebruikt worden. Het is nog steeds mogelijk om een level Blueprint event binnen de actor te maken die gekoppeld is aan het event van de component.

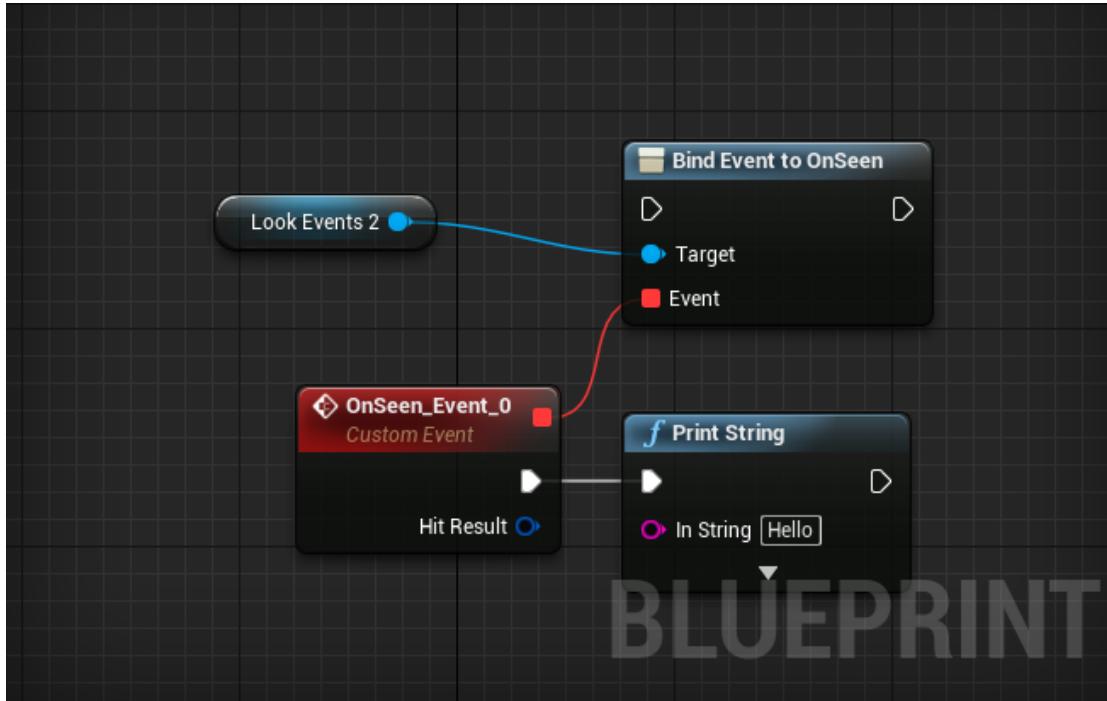
6.7.2 Events

De koppeling met Blueprints zit voornamelijk in de events die afgevuurd worden door de LookEventsComponent. Dit is ook de interface waarmee de niet-programmeurs de interactie kunnen programmeren.

Er kan op twee manieren logica aan een event gekoppeld worden. De eerste is via het component menu zoals getoond in figuur 6.3 en de tweede zoals getoond in figuur 6.4.



FIGUUR 6.3: Voorbeeld van het koppelen van een event via het component menu.



FIGUUR 6.4: Voorbeeld van het koppelen van een event via een reference naar een LookEventsComponent

Vooral de eerste manier werd als duidelijk ervaren voor de niet-programmeurs, terwijl voor manier twee wel uitleg nodig was in de tweede workshop (bijlage B).

6.7.3 Instelling

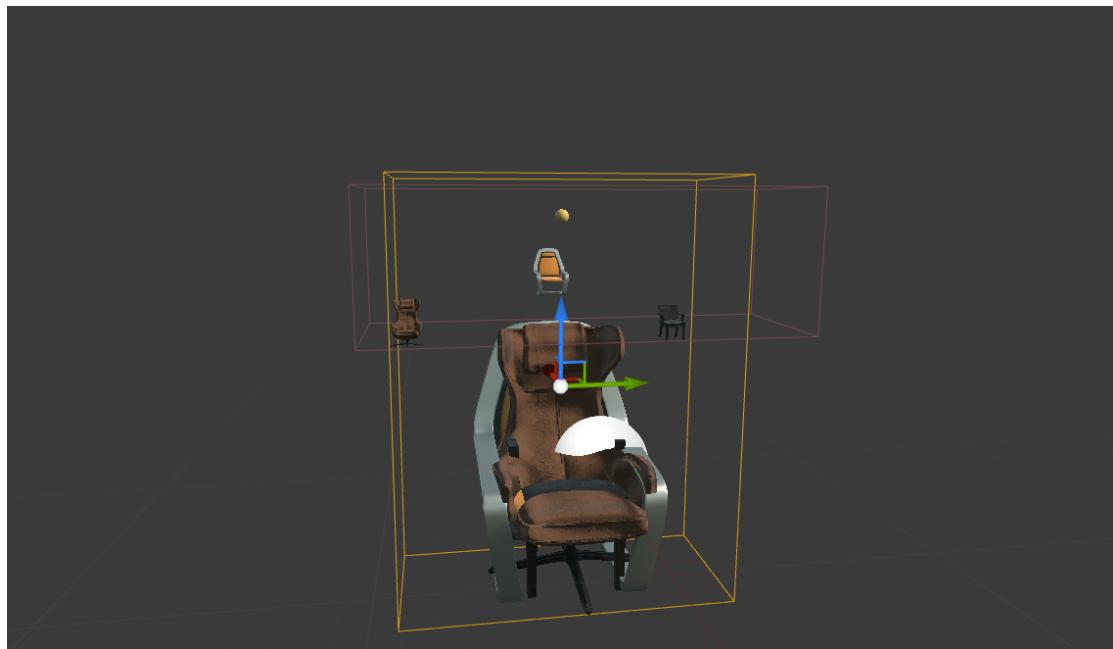
De volgende instellingen zijn mogelijk voor de LookEventsComponent:

- bShouldUsedOnce
- TimesUsed
- TriggerTimeOut
- SeenDelay
- UnSeenDelay
- bShouldDrawDebugLines
- bShouldDrawTriggerDebug
- ActorTriggerColor

- ComponentTriggerColor
- TriggerDistance
- TriggerRadius
- bActive
- SeenTrigger
- UnSeenTrigger

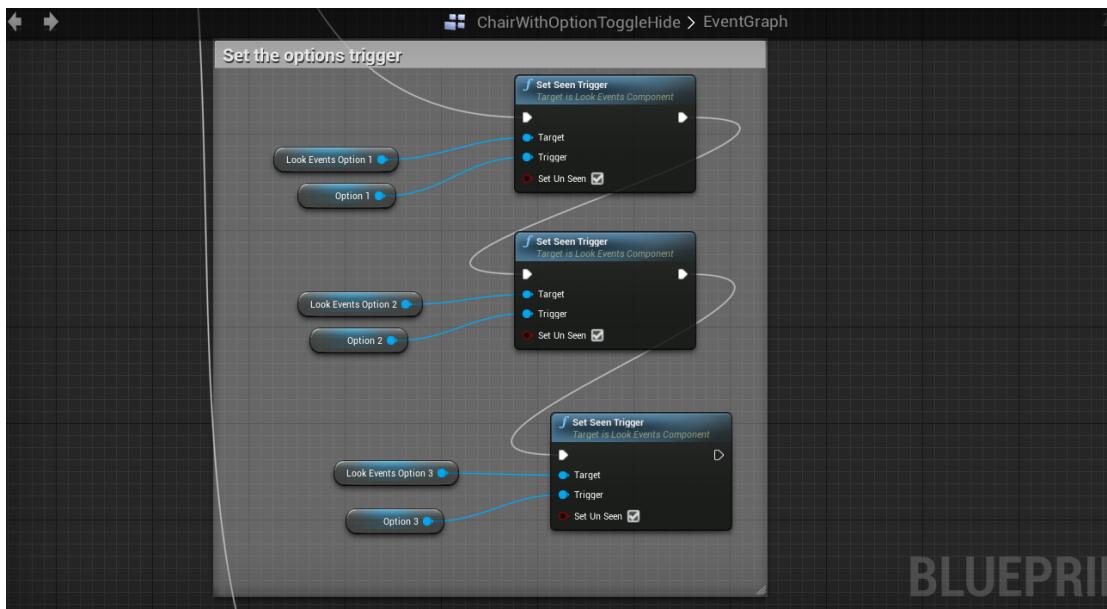
6.7.3.1 Seen en UnSeen trigger

De trigger is het object waar naar gekeken kan worden. Er kan voor zowel het triggeren van het Seen event als voor het triggeren van het UnSeen event een ander object gekozen worden. Deze keus is gemaakt nadat het duidelijk werd dat er vaak een animatie plaatsvindt, of er extra objecten (zoals een menu) tevoorschijn komen, die niet in de originele trigger passen.



FIGUUR 6.5: Een voorbeeld waarin de geselecteerde trigger box het menu toont en het menu getoond blijft zolang er naar de tweede trigger box gekeken word

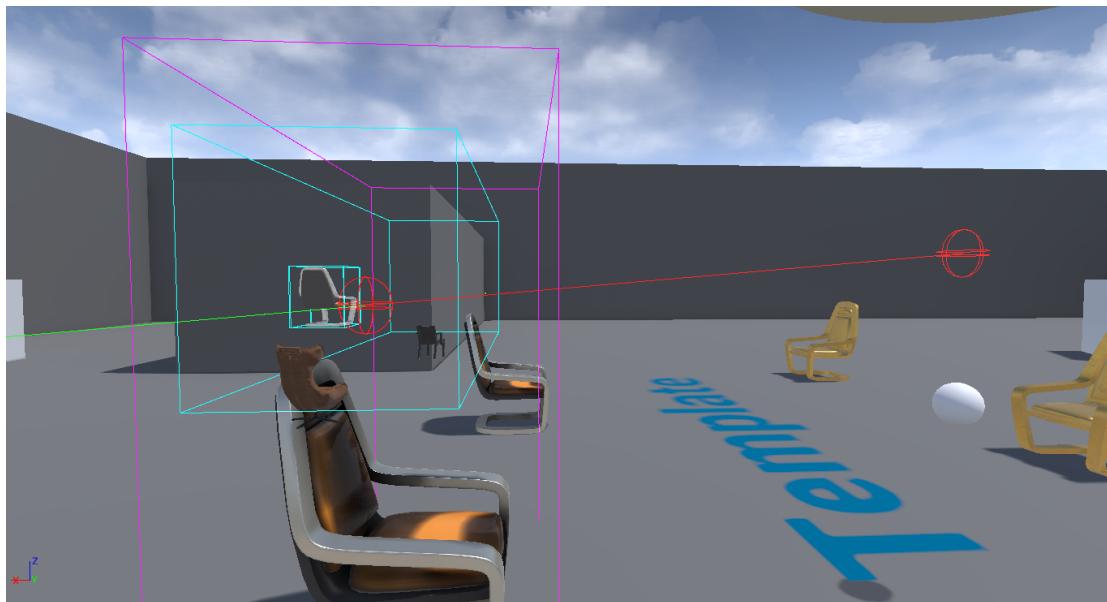
Als er niet specifiek een trigger opgegeven word word de Actor die het component bevat als Seen en Unseen trigger gezien. De triggers kunnen via blueprint gezet worden.



FIGUUR 6.6: Een voorbeeld van het instellen van de triggers voor drie verschillende LookEventComponents

6.7.4 Debugging

Tijdens de workshops werd er aangegeven dat het lastig was om de LookEvents te debuggen omdat de triggers niet te zien zijn. Er is voor gekozen om dezelfde debugging voor raytraces te implementeren voor de LookEvents.



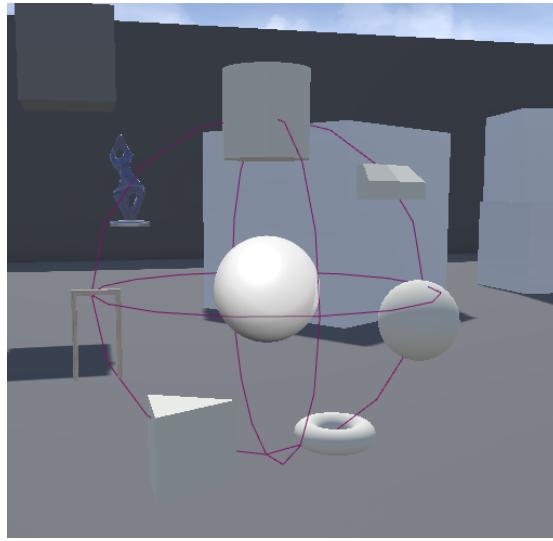
FIGUUR 6.7: Voorbeeld van debug informatie die een LookEvent kan tekenen.

Er wordt een lijn getekend vanaf de speler ter lengte van de TriggerDistance. Vervolgens word er een vierkant getekend rond de active triggers, er kan een kleur voor Actors en

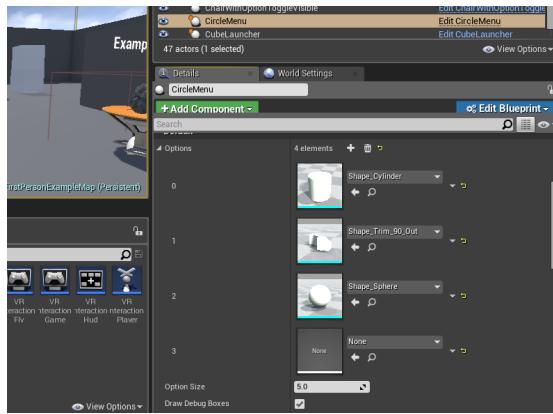
voor Components opgegeven worden. Als er een blocking hit is, door een niet doorzichtig object, wordt er een cirkel getekend met als diameter de TriggerRadius op de plek waar de blocking hit plaats vond.

6.8 CircleMenu

Het CircleMenu genereert een menu van LookEventsComponents op basis van een lijst van meshes.



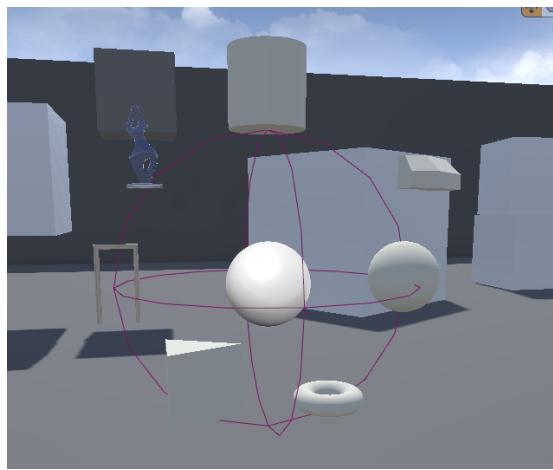
FIGUUR 6.8: Voorbeeld van het CircleMenu.



FIGUUR 6.9: Het instellen van de opties in een CircleMenu.

Om te zorgen dat de meshes correct gepositioneerd worden in een cirkel kan er een grootte van de meshes opgegeven worden en worden de meshes op basis hiervan individueel gescaled. Ook wordt er rekening mee gehouden dat het pivot point van een mesh zich niet altijd in het midden bevindt en wordt er berekend wat het midden van de mesh

is om vervolgens de mesh correct te positioneren. Er is ook een optie om het menu altijd richting de speler te tonen.



FIGUUR 6.10: Het CIrcleMenu zonder dat de pivot gecorrigeerd word.

De combinatie van het genereren en volgen van de speler zorgt ervoor dat de niet-programmeurs uitgebreide menu's in VR kunnen creëren.

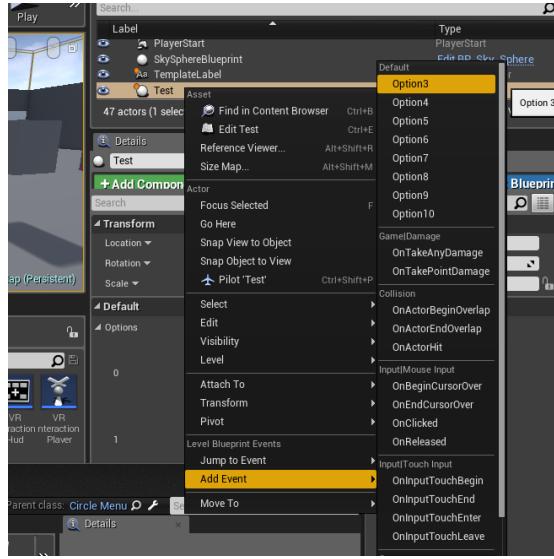
6.8.1 Instellingen

- Options
- OptionsSize
- DrawDebugBoxes
- TriggerDistance
- FollowPlayer

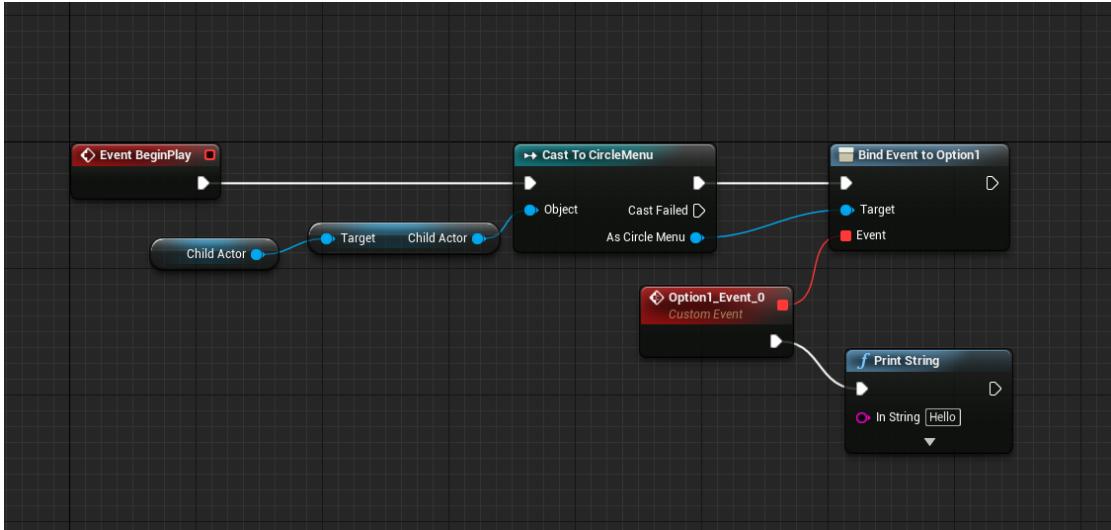
6.8.2 Events

Voor het circlemenu moest een Actor gemaakt worden, omdat components geen sub-components kunnen hebben. Dit maakt het mogelijk om via het rechtermuisknop menu events te koppelen in het level Blueprint.

Helaas maakt dit het gebruik van het CircleMenu in een Actor omslachtig. Er moet namelijk gebruik gemaakt worden van een ChildActor component. De ChildActor component maakt het niet mogelijk om instellingen via het detail menu in te stellen. Hierdoor zal het daadwerkelijk CircleMenu via Blueprints benaderd moeten worden.



FIGUUR 6.11: Het koppelen van een event via het level Blueprint.



FIGUUR 6.12: Het koppelen van het instellen van een event in een CircleMenu als ChildActor component.

Deze omslachtigere manier van event koppeling vergt beter kennis van Blueprints en werd als onduidelijk ervaren door de niet-programmeurs.

6.9 VRMovableMesh

De VRMovableMesh is een Actor die verplaatsbaar is door middel van een interactie knop en het kijken naar de bestemming. Het object wordt versleept naar de locatie waar naar gekeken wordt zolang men een knop indrukt. Deze Actor is voornamelijk

bedoelt als voorbeeld en kan als referentie gebruikt worden tijdens het implementeren van eigen Actors.



FIGUUR 6.13: Voorbeeld van het verplaatsen van een MovableMesh.

6.9.1 Instellingen

- MainMesh
- MaxDistance
- MinDistance
- Throwable
- bSetRotation
- bIsInteracting

Hoofdstuk 7

Resultaten, Discussie en Vervolgonderzoek

In dit hoofdstuk word er antwoord gegeven op de deelvragen en worden de resultaten in context geplaatst en besproken. Als laatste word er ingegaan op mogelijke vervolgstappen.

7.1 Resultaten

In Hoofdstuk 5 is antwoord te geven op de vraag "Hoe kan er een koppeling met VS gemaakt worden die intuïtief is voor niet programmeurs?". Met de workflow die in paragraaf 5.3 uiteen is gezet, zijn de projecten uit paragraaf 4.3 ontwikkeld en is antwoord gegeven op de vraag "Welke componenten zullen gemaakt moeten worden?". Namelijk de componenten uit de VRInteractions plugin die besproken is in hoofdstuk 6. De componenten werken op basis van kijken naar objecten, een interactie vorm die werkt op de platforms die gekozen zijn in paragraaf 4.4.1, waarmee antwoord gegeven is op de vraag "Is het mogelijk de componenten cross-platform te maken?".

Het gebruiksgemak van de VRInteractions plugin is getest, bijlage D, na het geven van drie workshops, bijlagen A, B en C. Uit de test bleek dat het niet mogelijk was voor de niet-programmeurs om zelfstandig een complex interactief object te maken met behulp van de VRInteractions plugin. Uit de workshops bleek wel dat de niet-programmeurs na een korte uitleg zelfstandig aan de slag konden.

7.2 Discussie

Alhoewel tijdens de gebruikstest van de VRInteractions plugin een onacceptabele resultaat behaald werd, zijn de opdrachten uit workshop 3 wel succesvol zelfstandig gemaakt. Ook zijn er parallel aan het schrijven van deze scriptie twee projecten met de VRInteractions plugin ontwikkeld door de niet-programmeurs waarbij geen hulp nodig was. Het gaat om een demo in het Colosseum en het menu van een fly-through door het menselijk lichaam. Beide testers gaven aan dat ze een tijd (2 a 3 weken) niet met Blueprints gewerkt hadden en dat ze daarom moeite hadden met de test.

Tijdens het afstudeer traject zijn de volgende projecten succesvol gemaakt, waarvan twee zoals eerder aangegeven zonder hulp:

- Een fly-trough door een menselijk lichaam
- Een appartement waarvan o.a. het meubilair en de vloer dynamisch aangepast kon worden
- Een demo van een machine uit een fabriek
- Een virtuele omgeving van een tentoonstelling
- Een leeromgeving rond het Colosseum
- Een demo omgeving van de VRInteractions plugin

Naast het gebruik van de VRInteractions plugin waren er ook een aantal performance problemen in de demo's die niet opgelost konden worden door de niet-programmeurs. Het is wel mogelijk voor niet-programmeurs om zelfstandig interactie toe te voegen aan een 3D omgeving zijn er toch problemen die alleen door een programmeur opgelost kunnen worden.

Uit de behoefte aan hulp voor de niet-programmeurs en het niet kunnen vermijden van problemen die de niet-programmeurs niet zelf kunnen oplossen word er geadviseerd om in een ontwikkelteam minstens een programmeur met 3D ervaring beschikbaar te hebben die complexere problemen kan oplossen en ondersteuning kan geven aan de niet-programmeurs.

7.3 Vervolgonderzoek

De VRInteractions is ontwikkeld op de tweede dev kit van de Oculus en focust zich voornamelijk op demo's met als einddoel het overdragen van informatie. Er zijn waarschijnlijk cases die niet tijdens het afstudeer traject langs zijn gekomen waar de VRInteractions plugin ook voor ingezet kan worden.

Het aanbod van VR headsets is nog steeds aan het groeien en er zijn all een aantal nieuwe technieken die momenteel ontwikkeld worden om de ervaring te verbeteren. Er kan onderzocht worden of de VRInteractions plugin meerwaarde kan hebben op deze headsets.

In deze scriptie is er gefocust op interactie op basis van kijken. Er kan onderzocht worden of er naast deze vorm van interactie ook andere vormen ondersteund kunnen worden door de VRInteractions plugin.

Hoofdstuk 8

Conclusie

Om het ontwikkelen van interactieve 3D omgevingen mogelijk te maken voor niet-programmeurs is de VRInteractions plugin ontwikkeld. De plugin maakt het mogelijk om met een aantal simpele stappen VR omgevingen op te zetten en om bestaande omgevingen werkend te krijgen in VR.

Door het ontwikkelen met een workflow die prioriteit geeft aan vrijheid, experimenten en gebruiksgemak zijn de componenten iteratief ontwikkeld in samenwerking met de niet-programmeurs. Hierdoor is een intuïtieve koppeling gemaakt tussen Blueprints en VS.

Op basis van de gemaakte projecten en de taken van een programmeur hierin zijn de volgende componenten ontwikkeld:

- Een game mode voor een demo waarin de speler kan rondlopen en een voor een fly-through demo
- Een speler klasse per game mode
- Een component die verantwoordelijk is voor de input van de speler
- Een component die verantwoordelijk is voor de camera van de speler
- Een component die events afvuurt als er naar een object gekeken wordt
- Een HUD die het mogelijk maakt om op basis van de look events informatie te tonen

- Een 3D menu die zijn opties in een cirkel toont
- Een object wat verplaats kan worden door middel van ernaar te kijken.

Door de interactie te beperken tot kijken naar objecten is er voor gezorgd dat alle componenten in de VRInteractions plugin werken op de Oculus, Vive en de GearVR.

Met behulp van de plugin zijn meerdere projecten ontwikkeld door niet-programmeurs wat aantoont dat er succesvol een ontwikkelomgeving gerealiseerd is waarin niet-programmeurs in UE4 unieke virtual reality ervaring kunnen creëren.

Bijlage A

Workshop 1

In de eerste workshop werd een introductie gegeven van Blueprints en UE4. Daarnaast werden de ‘best practises’ besproken en een aantal valkuilen tijdens het modelleren voor een game engine.

De workshop is opgenomen en het audio fragment is beschikbaar op aanvraag.

A.1 Doel

- Het doornemen van best practises voor UE4 gefocused op verschillen tussen 3D moddeling en 3D games.
- Een begin maken en kennis laten maken met Blueprints.

A.2 Best practises

De volgende problemen waren de niet-programmeurs tegenkomen en werden kort uitgelicht. Er werd ook verteld dat in tegenstelling tot het maken van statische 3D scene het maken van een 3D wereld veel fout gevoeliger is en dat er bewuster met de verschillende elementen van de engine omgegaan moet worden.

De volgende best practises werden toegelicht.

- Probeer altijd de documentatie te lezen van een onderdeel voordat je het in productie wilt gebruiken.
- Geen scenes in Max maken maar objecten los exporteren en de scene bouwen in UE4.
- Textures altijd in dimensies in de “Power of Two” maken (4, 8, 16, 32, 64, 128, 256 etc.).
- Texture grootte en vertices count zijn belangrijk voor performance in tegenstelling tot 3D moddeling.
- Maximale texture grote 4096 * 4096 (2048*2048 voor mobiel).
- 65,000 vertices max.
- Gebruik altijd het tweede UV channel.

Ook werd er gevraagd om de volgende link goed door te lezen als voorbeeld van de UE4 documentatie <https://docs.unrealengine.com/latest/INT/Engine/Content/FBX/BestPractices/index.html>

Ontwikkeling voor mobiel

De volgende best practises met betrekking tot mobile ontwikkeling werden toegelicht:

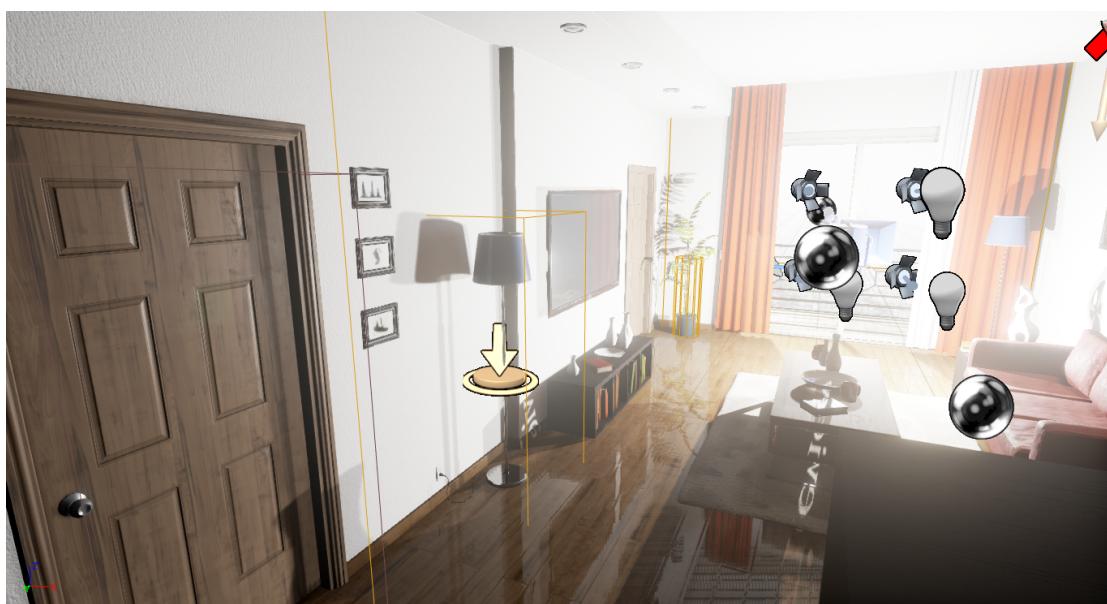
- Tijdens het opzetten van een project de target hardware setting op mobiel en de quality setting op scalable.
- Het is makkelijker om de kwaliteit te upscalen dan te downscalen.
- Voor het mobile platform komen alle assets in de package terecht. Daarom is het belangrijk om niet gebruikte assets te verwijderen.
- Maximale texture van 2048*2048 en maximaal 65,000 per object.
- De belichting is minder geavanceerd op een mobiel. Daardoor moet er handmatig meer tijd gestopt worden in het zelf belichten van scenes.

Ook werd er gevraagd om de volgende link goed door te lezen als voorbeeld van de UE4 documentatie <https://docs.unrealengine.com/latest/INT/Engine/Content/FBX/BestPractices/index.html>

A.3 Workshop

Na het bespreken van de best practises voor UE4 werd er voor gedaan hoe een lamp aan en uit gezet kan worden als de speler in de buurt komt door middel van een Blueprint.

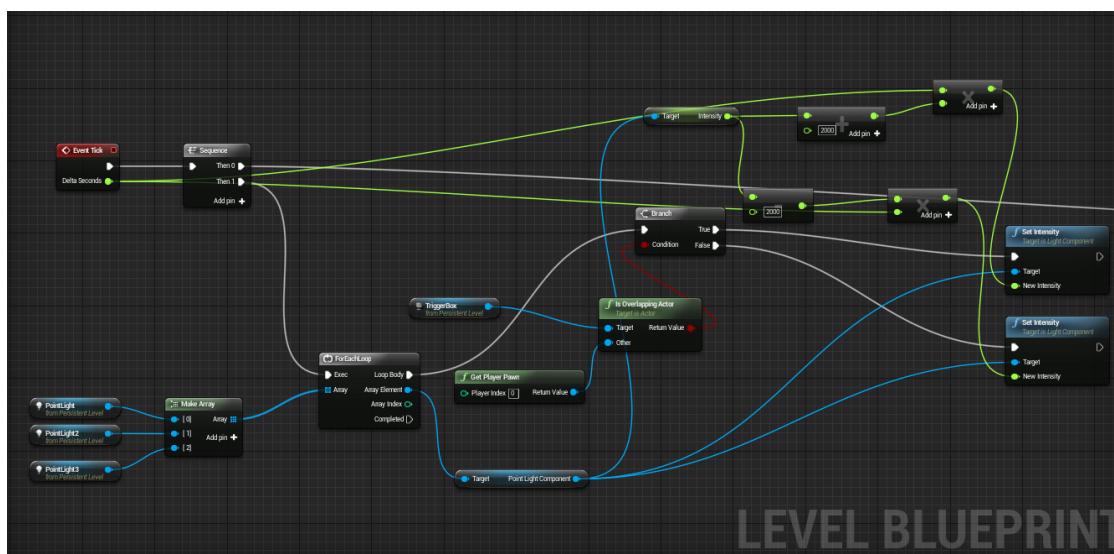
Er werd gevraagd of de lamp ook geleidelijk in sterkte kon toenemen en dit werd ook gedemonstreerd.



FIGUUR A.1: De lamp die aan en uit gezet werd.

A.4 Reflectie

Na de workshop is er met de deelnemers besproken welke van de behandelde onderwerpen onduidelijk waren of interessant om dieper op in te gaan. Op basis hiervan is besloten om dieper op Blueprints, en algemene programmeer logica, in te gaan. Ook is er besloten om in de volgende workshop zelfstandig een opdracht te maken omdat vooral de onderwerpen die het best geleerd kunnen worden door middel van ervaring onduidelijk waren.



FIGUUR A.2: De Blueprint van het langzaam laten toenemen van de licht sterke van de lamp.

Bijlage B

Workshop 2

De workshop begon met een uitleg van de concepten die behandeld gaan worden waarna de niet-programmeurs zelfstandig de opdrachten gingen maken.

Er is voor elke deelnemer een test omgevingen aangemaakt waar de VRInteractions plugin geïnstalleerd is en een aantal voorbeelden bevat die als hulp gebruikt kunnen worden.

De workshop is opgenomen en het audio fragment is beschikbaar op aanvraag.

B.1 Doel

- Kennis laten maken met de VRInteractions plugin en een tv aan en uit zetten door er naar te kijken.
- Door de deelnemers zelf te laten programmeren en daardoor het doel van de komende workshops te laten bepalen.

B.2 Opdrachten

Opdracht 1

Zet de TV aan door er naar te kijken en pauzeer hem als er niet meer naar gekeken wordt.

In het oefenproject staat een TV met een promo video van DPI. Voeg de LookEvents component aan de TV blueprint toe. Koppel vervolgens de play en pause van de media texture, het filmpje, aan de Lookevents door middel van de LookEvents component.

Opdracht 2

Zorg dat de LookEvents makkelijker getriggerd worden door een Collisie box te laten bepalen wanneer er naar de TV gekeken wordt.

Voeg een collision box toe aan de tv blueprint en zet zijn collisie profile op custom en zet alle collisie op ignore behalve de visibility. De visibility moet namelijk op overlap.

Creëer een begin play event en sleep de LookEvents Component in de blueprint. Als je op van de exit node een lijn sleept en los laat kan je in het pop up window zoeken naar “Set Seen Trigger”. Hiermee geef je aan dat je de events wil laten afvuren als naar een specifiek component van de blueprint gekeken wordt. Zet vervolgens de gemaakte Box Collision als de trigger parameter.

Opdracht 3

Stel de LookEvents in zodat de TV op een kortere (of langere) afstand getriggerd wordt.

Als je de look events component in het components venster (links boven) selecteert verschijnt er aan de rechter kant in het details venster de instelling hiervan. Verander de waardes onder de kopjes “User Interaction” en “Trigger” totdat jij tevreden bent met het resultaat.

Opdracht 4

Maak een loader die het kijken naar het object duidelijk maakt.

Voeg een statische mesh component toe en zet de statische mesh op de standaard materialshpere. In de tick functie van de blueprint vraag je de “Seen Progress” van de lookevents door de functie “Get Seen Progress” Op de lookevents component aan te spreken. Creëer nu een “Lerp(Vector)” door met een rechte muisknop in de blueprints het zoek venster te opennen en hierin te zoeken naar de functie. Zet als waarde A de

begin locatie van de Loader en als waarde B zijn eind waarde (waar hij naartoe gaat bewegen) Vervolgens pak zet je relative locatie van de loader op de return value van de lerp.

B.3 Resultaten

Uiteindelijk zijn de deelnemers samen de opdracht gaan maken en hebben het volgende ingeleverd:

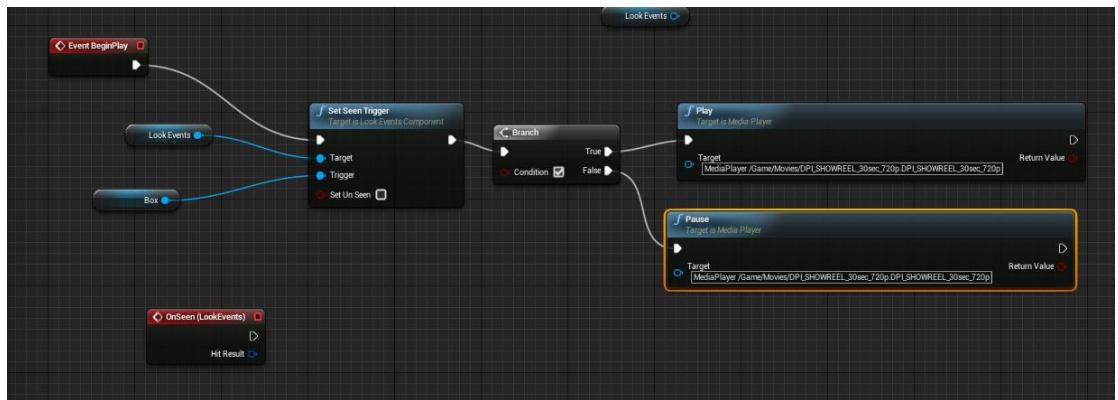
Yo Mark,

We hebben dit even gezamenlijk gedaan.

Hier wat onduidelijk heden.

Opdracht 1: we konden in eerste instantie de play node niet vinden. Het was wat onduidelijk dat we de movie texture moesten verslepen naar media player target.

Opdracht 2 en 3: Event begin play was niet duidelijk. Deze triggerde het filmpje vanaf het begin. zoals in de afbeelding te zien.

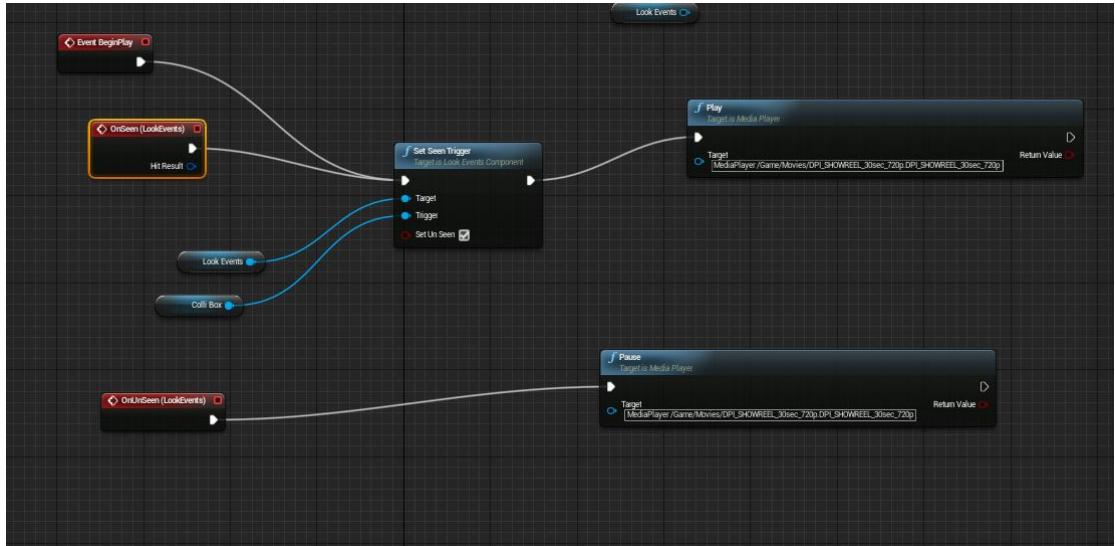


FIGUUR B.1: Niet werkende Blueprint van opdracht twee en drie.

We probeerde ook wat uit met Branch, maar helaas.

Daarna hebben we weer alles vastgekoppeld aan de OnSeen look event en aan de Event beginPlay. Dit bleek wel te werken.

Opdracht4: Uitleg was beter, plaatje werkte ook mee, maar op zich allemaal logisch.



FIGUUR B.2: Functionele Blueprint van opdracht twee en drie.



FIGUUR B.3: Eind resultaat van de opdrachten.

B.4 Reflectie

Het probleem van de play node vinden was inderdaad lastig omdat dit een uitzondering is op hoe de rest van UE4 werkt. De enige manier om dit soort problemen op te lossen is door ervaring krijgen met hoe de ue4 werkt.

De uitwerking van opdracht twee en drie was werkend maar het zetten van de LookEvents optie werd in het Tick event gedaan wat voor onverwachte functionaliteit kan zorgen. Dit had eigenlijk tijdens de Begin Play moeten gebeuren.

Na het bespreken van de opdrachten is er gezamenlijk gekozen om de volgende workshops op dezelfde manier aan te pakken; uitleg en dan zelfstandige opdrachten. Ook is er besloten om langzaam dieper in Blueprint scripting te duiken.

Bijlage C

Workshop 3

De workshop begon met een uitleg over de implementatie van de LookEvents uit de VRInteractions plugin en de verschillende configuratie mogelijkheden. Daarnaast is gezamenlijk een vergelijkbare functionaliteit gemaakt zodat het probleem van een node naam niet weten, uit Workshop 2, niet meer voor zou komen.

De workshop is opgenomen en het audio fragment is beschikbaar op aanvraag.

C.1 Doel

Een ingewikkelder versie van de LookEvents implementeren met hierin:

- Meerdere look events in een Actor
- Veranderden van een material tijdens runtime
- Het gebruik en maken van een functie
- Het gebruik en maken van variabel

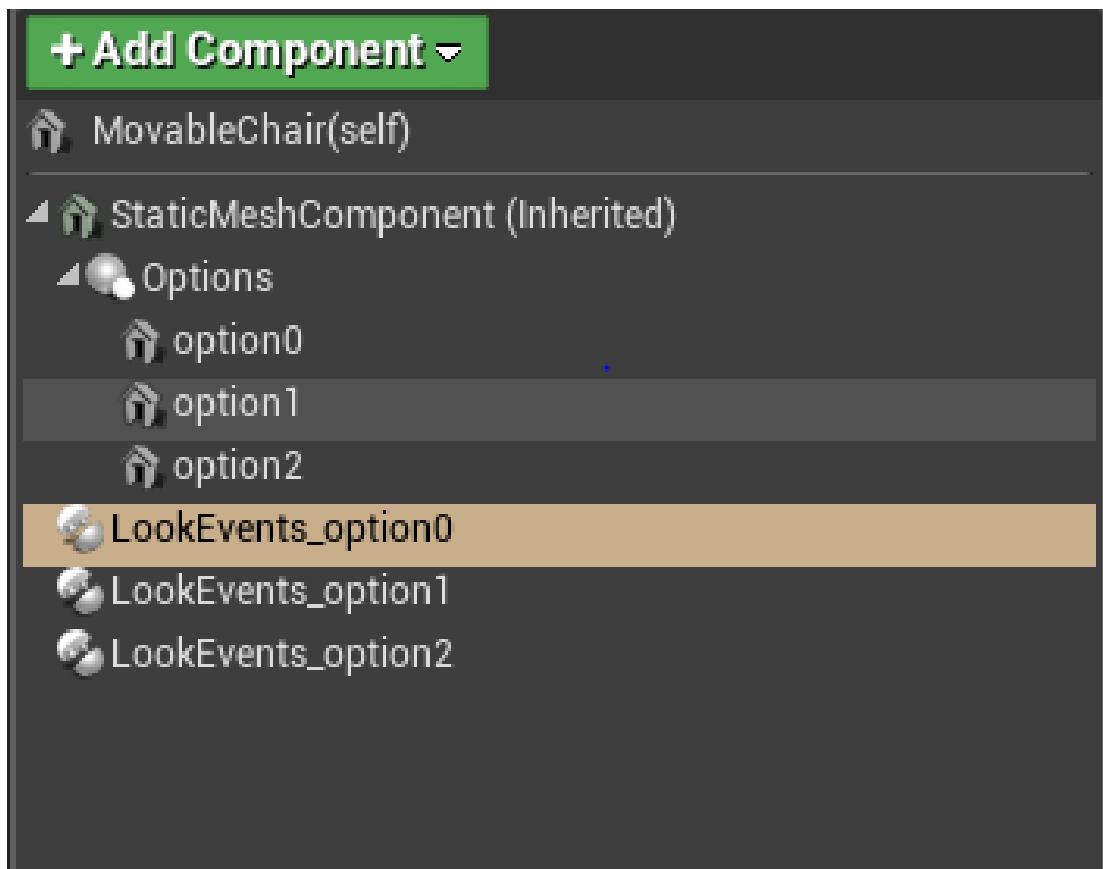
C.2 Opdrachten

We gaan een bank maken waar een menu boven zweeft waarmee wij het materiaal van de bank kunnen kiezen. Het menu wordt zichtbaar op het moment dat er naar de bank gekeken wordt.

Opdracht 1

Maak van de bank een Blueprint en voeg hier een scene (dit is een component) aan toe genaamd options met daarin drie static meshes genaamd option0 t/m option2.

Voeg voor elk element een Lookevent component toe en noem die LookEventOption0 t/m LookEventOption2.



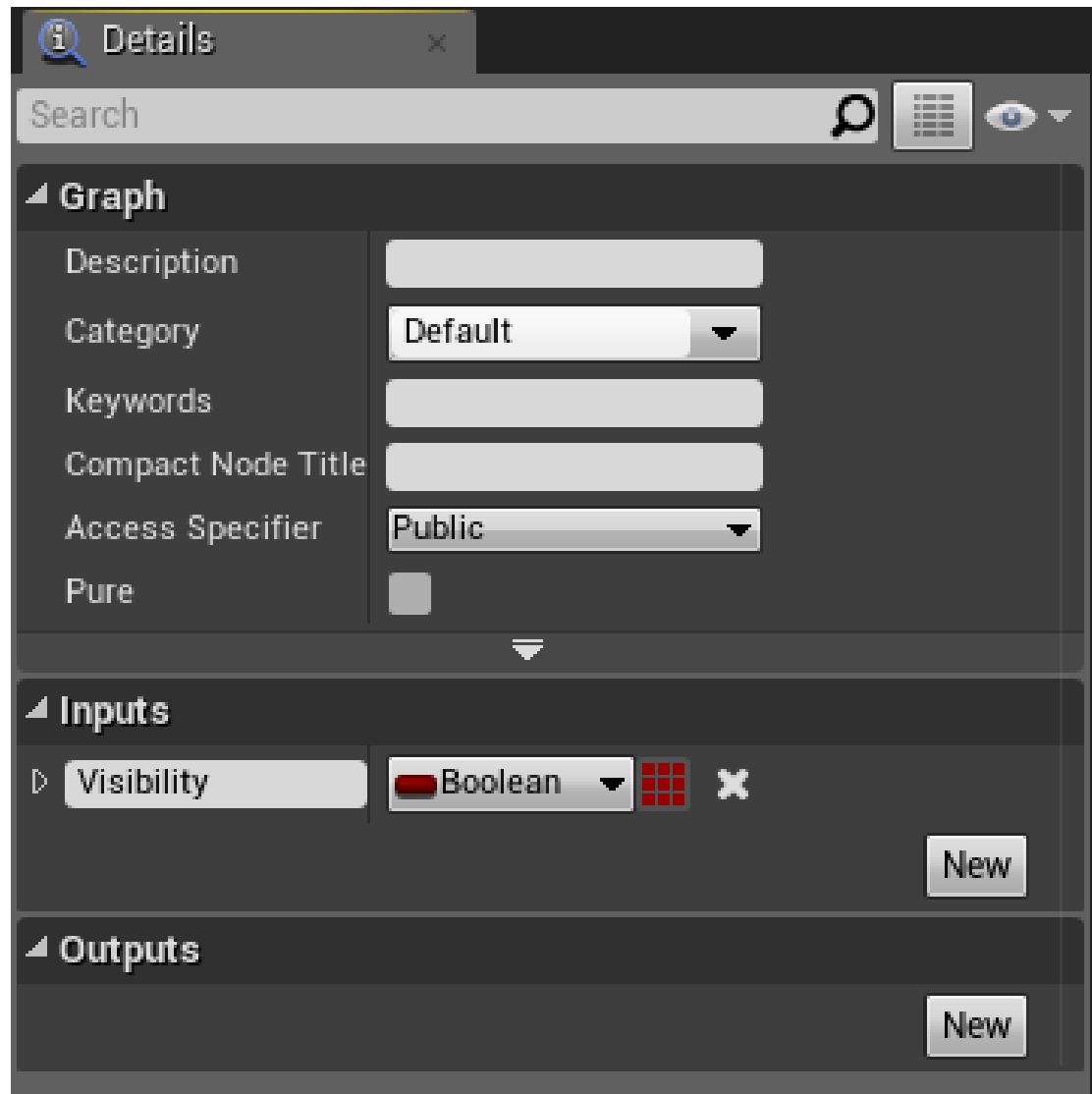
FIGUUR C.1: Voorbeeld van het toevoegen van de LookEvents componenten aan een Actor.

Koppel de look events aan de juiste options, kijk hiervoor naar de vorige opdracht, en zet de optie “Draw debug trigger” aan zodat je makkelijk kan zien of alles werkt.

Opdracht 2

Om te zorgen dat de options alleen zichtbaar worden als er naar de bank gekeken wordt voegen we nog een LookEvents component toe genaamd “LookEventsMain”.

Om de code schoner te houden maken we een functie genaamd “SetOptionsVisibility” met boolean als input genaamd “Visibility”. In deze functie haal je alle kinderen van de Options scene component op en zet je de visibility hiervan op de input.



FIGUUR C.2: Voorbeeld van een functie in Blueprints.

Koppel nu de OnSeen en de OnUnseen events van de “LookEventsMain” aan de “SetOptionsVisibility” functie.

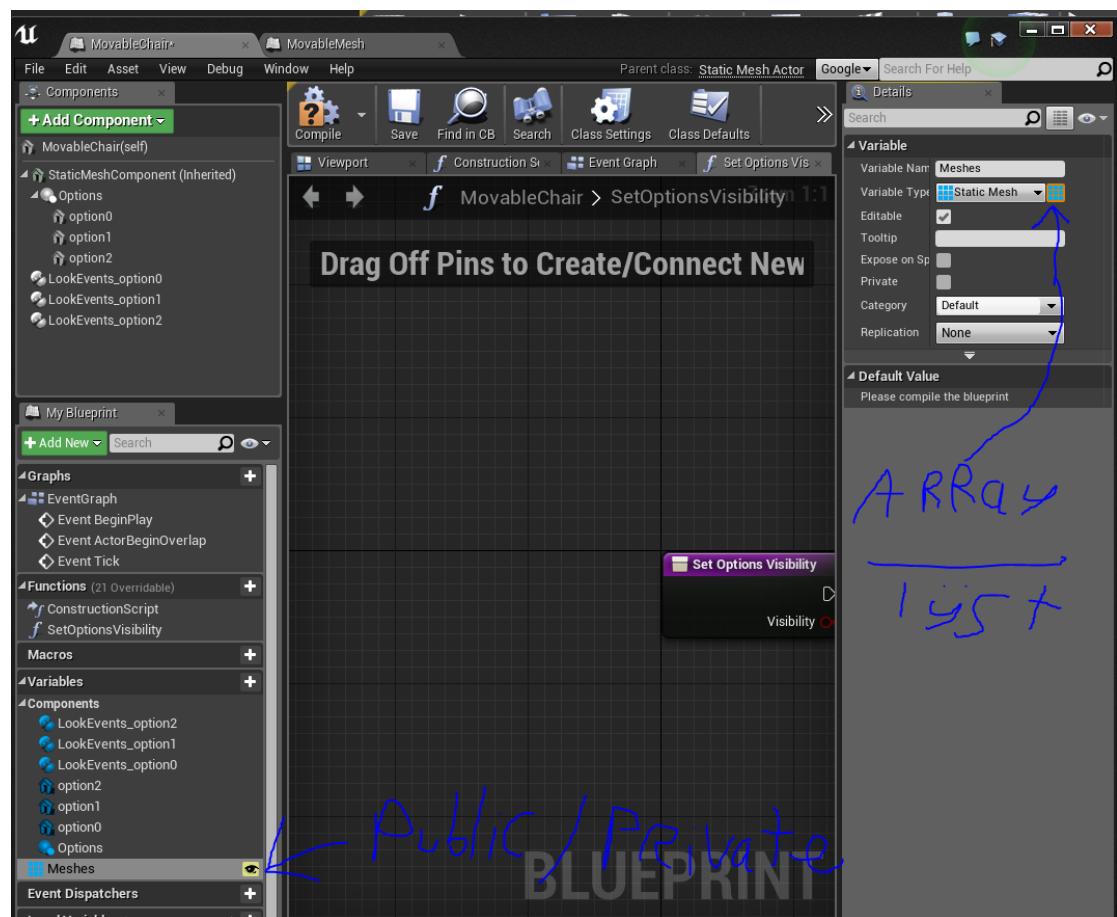
Opdracht 3

We gebruiken de materials van de options static meshes om de material van de bank te bepalen. Op het moment dat er naar een option gekeken wordt (dus het OnSeen event)

halen we de material uit de bijhorende option en zetten we material van de hoofd static mesh hierop.

Opdracht 4

Probeer niet de material maar de mesh van de bank te veranderen. Je kan dit doen via de static meshes van de options (zelfde manier als wij de kleuren maken) maar als je een uitdaging wil probeer het via een array variable.



FIGUUR C.3: Voorbeeld van een variable in Blueprints.

Als je voor een variabel gaat vergeet niet dat dit een lijst van Static Mesh References moet zijn.

C.3 Reflectie

Na het bespreken van de opdracht werd er aangegeven dat er minder tijd kwijt was aan het zoeken naar functienamen omdat een gelijke opdracht van te voren samen gemaakt was.

Het viel op dat concepten waar in de vorige workshop problemen mee waren dit keer wel goed gingen, bijvoorbeeld het correct zetten van de LookEvents trigger, zonder hier specifieke instructies voor te geven.

Bijlage D

Usability Test

Om het gebruiksgemak te meten van de VRInteractions plugin werden de niet-programmeurs gevraagd een probleem op te lossen, waar een optimale oplossing voor is vastgelegd, met behulp van de plugin.

De gebruikers worden gefilmd tijdens het maken van de opdracht en mogen vragen stellen als ze vast lopen of iets niet begrijpen. De begeleider mag antwoord geven op vragen in een uitleggende vorm maar mag alleen ingrijpen in de opdracht als de niet-programmeer vast zit. De begeleider mag ook ingrijpen als het duidelijk wordt dat de niet-programmeer de opdracht niet begrepen heeft.

Aan de hand van afwijkende stappen en vragen die gesteld moesten worden werd een conclusie getrokken.

De test is opgenomen en het video fragment is beschikbaar op aanvraag.

D.1 De Opdracht

Voor de opdracht zal er een Mesh generator gemaakt worden. De generator zal bestaan uit een vierkant met hierin een aantal Meshes die als er lang genoeg naar gekeken wordt een MovableMesh versie van zichzelf zullen spawnen.

Omschrijving

Maak een Actor met daarin een triggerbox die zichtbaar is tijdens het spelen. In de triggerbox plaats je een aantal meshes. Als er naar een mesh gekeken wordt zal deze beginnen met ronddraaien en uiteindelijk zal een MovableMesh versie van de mesh ge-spawnd worden.

Optimale oplossing

De volgende stappen zijn de benodigde stappen om tot een optimaal resultaat te komen. Onder het optimaal resultaat valt ook best-practices zoals naamgeving, gebruik van functies en geen vragen.

Maken van de Actor

- Maak een nieuwe Actor.
- Voeg een triggerbox component toe.
 - Verander de "Hidden in Game" setting naar false.
- Voeg een sceneComponent toe met een relevante naam (bijvoorbeeld opties).
- Voeg een StaticMeshComponent toe voor de gewenste opties.
- Voeg een LookEventsComponent toe met een relevante naam (bijvoorbeeld optiesLookEvents).
- Maak een functie die alle opties verbergt of toont met een relevante naam (bijvoorbeeld SetOptionsVisibility).
- Koppel de OnSeen en OnUnSeen van de optiesLookEvents aan de SetOptionsVisibility.

Maken van opties

- Voeg een StaticMeshComponent toe aan de opties SceneComponent.
- Voeg een LookEventsComponent toe met een relevante naam (bijvoorbeeld ChairLookEvents of Optie1).

- Koppel aan de Tick functie de volgende nodes.
 - LooKEvents reference
 - Get Seen Progress
 - Seen Progress * rotation speed
 - Add Relative Rotation to the static mesh
- Koppel het OnSeen event aan de volgende nodes
 - SpawnActor
 - SetOptionsVisibility
 - Delay
 - SetOptionsVisibility

Maken van een MovableMesh variant

- Maak een nieuwe Blueprint class.
- Kies voor de VRMovableMesh class.
- verander de static mesh van de DefaultMesh naar de gewenste Mesh

Goede oplossing

Een acceptabel resultaat is een werkende versie van de opdracht zonder bugs of lange termijn performance problemen. Tijdens het maken mogen er geen vragen gesteld worden die een uitleg nodig hebben. Vragen zoals waar kan ik node x vinden zijn toegestaan omdat dit ook in documentatie te vinden is en meer ervaring dan begrip is.

Acceptable oplossing

Een acceptabel resultaat is een werkende versie van de opdracht zonder bugs of lange termijn performance problemen. Tijdens het maken mogen er inhoudelijke vragen gesteld worden over werking van de plugin of naar mathematische problemen, zoals het gebruik van lineaire interpolatie.

Onacceptabel oplossing

Als de opdracht niet werkend gekregen wordt zonder het ingrijpen van de begeleider, op onduidelijkheden over de opdracht na.

D.2 Uitvoering

Beide testers waren 2 uur bezig met de opdracht en bij beide testen werd er meerdere malen ingegrepen door de begeleider.

Test A

Tijdens het begin van test A was er onduidelijkheid over de manier waarop een Actor gemaakt kan worden. Er waren wat hints nodig om duidelijk te maken dat er in de content browser een Actor aangemaakt moest worden.

Tester A wist de correcte structuur van de Actor te maken en begreep zonder uitleg hoe de LookEvents, en de events, geïmplementeerd moesten worden.

Het eerste momenten dat er ingegrepen moest worden was over onduidelijkheid over de werking van de setVisibility node. Er werd namelijk vanuit gegaan dat de functie altijd een actor zichtbaar zou maken, in plaats van de zichtbaarheid van de actor baseren op de opgegeven Boolean. Het tweede moment van ingrijpen was tijdens het gebruiken van meerdere LookEvents en hoe een LookEvent gekoppeld kon worden aan een component.

Wat opviel was dat de tester slim gebruik wist te maken van een functie voor herhalende logica, namelijk het verbergen van de menu opties.

Tijdens het laatste gedeelte was er ook veel onduidelijkheid over de manier waarop een Actor gespawnd kon worden op de juiste locatie. Zelfs na zoeken op het internet kwam de tester er niet zelf uit hoe hij een Actor dynamisch in de wereld kon plaatsen.

Tester A gaf aan het eind aan dat de meeste problemen die hij met Blueprints ervaart komen omdat hij desondanks de lage instap nog steeds moet nadenken als een programmeur. Ook gaf hij aan dat hij het erg lastig vond om met Blueprints te werken als hij dit een tijd niet had gedaan, in dit geval twee weken.

Test B

Tester B wist sneller op te starten maar kwam al snel vast op de manier waarop events werken in Blueprints. Zo probeerde hij het instellen van de LookEvents component te laten gebeuren tijdens een van de events hiervan, wat zou zorgen dat de LookEvents pas correct ingesteld zouden worden nadat er naar gekeken werd.

Ook was er veel onduidelijkheid over hoe meerdere LookEvents in dezelfde Actor gebruikt konden worden.

In tegenstelling tot tester A wist tester B wel het spawnen van een Actor werkend te krijgen nadat de werking van de Blueprints events hem werd uitgelegd.

Tester B gaf aan het eind aan dat hij het erg lastig vond om met Blueprints te werken nadat hij dit een tijd niet gedaan had, namelijk drie weken i.v.m. vakantie.

D.3 Conclusie

Omdat er bij beide testers ingegrepen moest worden hebben beide testers een onacceptabel resultaat behaald. Er was bij beide testers onduidelijkheid over de manier waarop Blueprints werkt en er werden foutieve conclusies getrokken die de testers niet zelfstandig wisten op te lossen.

Beide testers gaven aan dat ze het lastig vonden om met Blueprints aan de slag te gaan nadat zij dit een periode niet gedaan hadden. Na uitleg over de correcte werking van een element wisten zie dit element wel succesvol te hergebruiken.

Uit deze test trekken wij de conclusie dat de VRInteractions plugin, en Blueprints zelf, niet intuïtief genoeg werkt voor niet-programmeurs om zonder regelmatige begeleiding.

Bijlage E

Implementatie voorbeeld speler Unreal Engine 4

```
1 #pragma once
2 #include "GameFramework/Character.h"
3 #include "dipi_unreal_colosseumCharacter.generated.h"
4
5 class UInputComponent;
6
7 UCLASS(config=Game)
8 class Adpi_unreal_colosseumCharacter : public ACharacter
9 {
10     GENERATED_BODY()
11
12     // Pawn mesh: 1st person view (arms; seen only by self)
13     UPROPERTY(VisibleDefaultsOnly, Category=Mesh)
14     class USkeletalMeshComponent* Mesh1P;
15
16     // Gun mesh: 1st person view (seen only by self)
17     UPROPERTY(VisibleDefaultsOnly, Category = Mesh)
18     class USkeletalMeshComponent* FP_Gun;
19
20     // First person camera
21     UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess
22         = "true"))
23     class UCameraComponent* FirstPersonCameraComponent;
24
25     Adpi_unreal_colosseumCharacter();
26
27     // Base turn rate, in deg/sec. Other scaling may affect final turn rate.
28     UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category=Camera)
29     float BaseTurnRate;
30
31     // Base look up/down rate, in deg/sec. Other scaling may affect final rate.
32     UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category=Camera)
33     float BaseLookUpRate;
34
35     // Gun muzzle's offset from the characters location
36     UPROPERTY(EditAnywhere, BlueprintReadWrite, Category=Gameplay)
37     FVector GunOffset;
38
39     // Projectile class to spawn
```

```

1 UPROPERTY(EditDefaultsOnly, Category=Projectile)
2 TSubclassOf<class Adpi_unreal_colosseumProjectile> ProjectileClass;
3
4 // Sound to play each time we fire
5 UPROPERTY(EditAnywhere, BlueprintReadWrite, Category=Gameplay)
6 class USoundBase* FireSound;
7
8 // AnimMontage to play each time we fire
9 UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Gameplay)
10 class UAnimMontage* FireAnimation;
11
12 protected:
13
14 // Fires a projectile.
15 void OnFire();
16
17 // Handles moving forward/backward
18 void MoveForward(float Val);
19
20 // Handles strafing movement, left and right
21 void MoveRight(float Val);
22
23 //
24 // Called via input to turn at a given rate.
25 // @param Rate This is a normalized rate, i.e. 1.0 means 100% of desired turn rate
26
27 void TurnAtRate(float Rate);
28
29 //
30 // Called via input to turn look up/down at a given rate.
31 // @param Rate This is a normalized rate, i.e. 1.0 means 100% of desired turn rate
32
33 void LookUpAtRate(float Rate);
34
35 struct TouchData
36 {
37     TouchData() { bIsPressed = false; Location=FVector::ZeroVector; }
38     bool bIsPressed;
39     ETouchIndex::Type FingerIndex;
40     FVector Location;
41     bool bMoved;
42 };
43 void BeginTouch(const ETouchIndex::Type FingerIndex, const FVector Location);
44 void EndTouch(const ETouchIndex::Type FingerIndex, const FVector Location);
45 void TouchUpdate(const ETouchIndex::Type FingerIndex, const FVector Location);
46 TouchData TouchItem;
47
48 protected:
49 // APawn interface
50 virtual void SetupPlayerInputComponent(UInputComponent* InputComponent) override;
51 // End of APawn interface
52
53 //
54 // Configures input for touchscreen devices if there is a valid touch interface for doing so
55 //
56 // @param InputComponent The input component pointer to bind controls to
57 // @returns true if touch controls were enabled.
58
59 bool EnableTouchscreenMovement(UInputComponent* InputComponent);
60
61 public:
62 // Returns Mesh1P subobject *
63 FORCEINLINE class USkeletalMeshComponent* GetMesh1P() const { return Mesh1P; }
64 // Returns FirstPersonCameraComponent subobject *
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102

```

```

104     FORCEINLINE class UCameraComponent* GetFirstPersonCameraComponent() const { return
105         FirstPersonCameraComponent; }
106     };

```

LISTING E.1: dpi unreal colosseumCharacter header

```

1 #include "dpi_unreal_colosseum.h"
2 #include "dpi_unreal_colosseumCharacter.h"
3 #include "dpi_unreal_colosseumProjectile.h"
4 #include "Animation/AnimInstance.h"
5 #include "GameFramework/InputSettings.h"
6
7 DEFINE_LOG_CATEGORY_STATIC(LogFPChar, Warning, All);
8
9 //////////////////////////////////////////////////////////////////
10 // Adpi_unreal_colosseumCharacter
11
12 Adpi_unreal_colosseumCharacter::Adpi_unreal_colosseumCharacter()
13 {
14     // Set size for collision capsule
15     GetCapsuleComponent()->InitCapsuleSize(42.f, 96.0f);
16
17     // set our turn rates for input
18     BaseTurnRate = 45.f;
19     BaseLookUpRate = 45.f;
20
21     // Create a CameraComponent
22     FirstPersonCameraComponent = CreateDefaultSubobject<UCameraComponent>(TEXT("FirstPersonCamera"));
23     FirstPersonCameraComponent->AttachParent = GetCapsuleComponent();
24     FirstPersonCameraComponent->RelativeLocation = FVector(0, 0, 64.f); // Position the camera
25     FirstPersonCameraComponent->bUsePawnControlRotation = true;
26
27     // Create a mesh component that will be used when being viewed from a '1st person' view (
28     // when controlling this pawn)
29     Mesh1P = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("CharacterMesh1P"));
30     Mesh1P->SetOnlyOwnerSee(true);
31     Mesh1P->AttachParent = FirstPersonCameraComponent;
32     Mesh1P->bCastDynamicShadow = false;
33     Mesh1P->CastShadow = false;
34
35     // Create a gun mesh component
36     FP_Gun = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("FP_Gun"));
37     FP_Gun->SetOnlyOwnerSee(true); // only the owning player will see this mesh
38     FP_Gun->bCastDynamicShadow = false;
39     FP_Gun->CastShadow = false;
40     FP_Gun->AttachTo(Mesh1P, TEXT("GripPoint"), EAttachLocation::SnapToTargetIncludingScale,
41                       true);
42
43     // Default offset from the character location for projectiles to spawn
44     GunOffset = FVector(100.0f, 30.0f, 10.0f);
45
46     // Note: The ProjectileClass and the skeletal mesh/anim blueprints for Mesh1P are set in the
47     // derived blueprint asset named MyCharacter (to avoid direct content references in C++)
48 }
49
50 // Input
51
52 void Adpi_unreal_colosseumCharacter::SetupPlayerInputComponent(class UInputComponent*
53     InputComponent)
54 {

```

```

54 // set up gameplay key bindings
55 check(InputComponent);
56
57 InputComponent->BindAction("Jump", IE_Pressed, this, &ACharacter::Jump);
58 InputComponent->BindAction("Jump", IE_Released, this, &ACharacter::StopJumping);
59
60 //InputComponent->BindTouch(EInputEvent::IE_Pressed, this, &Adpi_unreal_colosseumCharacter::
61 // TouchStarted);
62 if( EnableTouchscreenMovement(InputComponent) == false )
63 {
64     InputComponent->BindAction("Fire", IE_Pressed, this, &Adpi_unreal_colosseumCharacter::
65     OnFire);
66 }
67
68 // We have 2 versions of the rotation bindings to handle different kinds of devices
69 // differently
70 // "turn" handles devices that provide an absolute delta, such as a mouse.
71 // "turnrate" is for devices that we choose to treat as a rate of change, such as an analog
72 // joystick
73 InputComponent->BindAxis("Turn", this, &APawn::AddControllerYawInput);
74 InputComponent->BindAxis("TurnRate", this, &Adpi_unreal_colosseumCharacter::TurnAtRate);
75 InputComponent->BindAxis("LookUp", this, &APawn::AddControllerPitchInput);
76 InputComponent->BindAxis("LookUpRate", this, &Adpi_unreal_colosseumCharacter::LookUpAtRate);
77 }
78
79 void Adpi_unreal_colosseumCharacter::OnFire()
80 {
81     // try and fire a projectile
82     if (ProjectileClass != NULL)
83     {
84         const FRotator SpawnRotation = GetControlRotation();
85         // MuzzleOffset is in camera space, so transform it to world space before offsetting from
86         // the character location to find the final muzzle position
87         const FVector SpawnLocation = GetActorLocation() + SpawnRotation.RotateVector(GunOffset);
88
89         UWorld* const World = GetWorld();
90         if (World != NULL)
91         {
92             // spawn the projectile at the muzzle
93             World->SpawnActor<Adpi_unreal_colosseumProjectile>(ProjectileClass, SpawnLocation,
94             SpawnRotation);
95         }
96
97     // try and play the sound if specified
98     if (FireSound != NULL)
99     {
100         UGameplayStatics::PlaySoundAtLocation(this, FireSound, GetActorLocation());
101     }
102
103     // try and play a firing animation if specified
104     if(FireAnimation != NULL)
105     {
106         // Get the animation object for the arms mesh
107         UAnimInstance* AnimInstance = Mesh1P->GetAnimInstance();
108         if(AnimInstance != NULL)
109         {
110             AnimInstance->Montage_Play(FireAnimation, 1.f);
111         }
112     }
113 }
```

```

114 void Adpi_unreal_colosseumCharacter::BeginTouch(const ETouchIndex::Type FingerIndex, const
115 FVector Location)
116 {
117     if( TouchItem.bIsPressed == true )
118     {
119         return;
120     }
121     TouchItem.bIsPressed = true;
122     TouchItem.FingerIndex = FingerIndex;
123     TouchItem.Location = Location;
124     TouchItem.bMoved = false;
125 }
126
127 void Adpi_unreal_colosseumCharacter::EndTouch(const ETouchIndex::Type FingerIndex, const
128 FVector Location)
129 {
130     if( TouchItem.bIsPressed == false )
131     {
132         return;
133     }
134     if( ( FingerIndex == TouchItem.FingerIndex ) && ( TouchItem.bMoved == false ) )
135     {
136         OnFire();
137     }
138     TouchItem.bIsPressed = false;
139 }
140
141 void Adpi_unreal_colosseumCharacter::TouchUpdate(const ETouchIndex::Type FingerIndex, const
142 FVector Location)
143 {
144     if( ((TouchItem.bIsPressed == true) && ( TouchItem.FingerIndex==FingerIndex)) )
145     {
146         if( TouchItem.bIsPressed )
147         {
148             if( GetWorld() != nullptr )
149             {
150                 FVector MoveDelta = Location - TouchItem.Location;
151                 FVector2D ScreenSize;
152                 ViewportClient->GetViewportSize(ScreenSize);
153                 FVector2D ScaledDelta = FVector2D( MoveDelta.X, MoveDelta.Y ) / ScreenSize;
154                 if( ScaledDelta.X != 0.0f )
155                 {
156                     TouchItem.bMoved = true;
157                     float Value = ScaledDelta.X * BaseTurnRate;
158                     AddControllerYawInput(Value);
159                 }
160                 if( ScaledDelta.Y != 0.0f )
161                 {
162                     TouchItem.bMoved = true;
163                     float Value = ScaledDelta.Y* BaseTurnRate;
164                     AddControllerPitchInput(Value);
165                 }
166                 TouchItem.Location = Location;
167             }
168             TouchItem.Location = Location;
169         }
170     }
171 }
172
173 void Adpi_unreal_colosseumCharacter::MoveForward(float Value)

```

```

176 {
177     if (Value != 0.0f)
178     {
179         // add movement in that direction
180         AddMovementInput(GetActorForwardVector(), Value);
181     }
182 }
183 void Adpi_unreal_colosseumCharacter::MoveRight(float Value)
184 {
185     if (Value != 0.0f)
186     {
187         // add movement in that direction
188         AddMovementInput(GetActorRightVector(), Value);
189     }
190 }
191 void Adpi_unreal_colosseumCharacter::TurnAtRate(float Rate)
192 {
193     // calculate delta for this frame from the rate information
194     AddControllerYawInput(Rate * BaseTurnRate * GetWorld()>GetDeltaSeconds());
195 }
196
197 void Adpi_unreal_colosseumCharacter::LookUpAtRate(float Rate)
198 {
199     // calculate delta for this frame from the rate information
200     AddControllerPitchInput(Rate * BaseLookUpRate * GetWorld()>GetDeltaSeconds());
201 }
202
203 bool Adpi_unreal_colosseumCharacter::EnableTouchscreenMovement(class UInputComponent*
204     InputComponent)
205 {
206     bool bResult = false;
207     if(FPlatformMisc::GetUseVirtualJoysticks() || GetDefault<UInputSettings>()>
208         bUseMouseForTouch )
209     {
210         bResult = true;
211         InputComponent->BindTouch(EInputEvent::IE_Pressed, this, &Adpi_unreal_colosseumCharacter::BeginTouch);
212         InputComponent->BindTouch(EInputEvent::IE_Released, this, &Adpi_unreal_colosseumCharacter::EndTouch);
213         InputComponent->BindTouch(EInputEvent::IE_Repeat, this, &Adpi_unreal_colosseumCharacter::TouchUpdate);
214     }
215     return bResult;
216 }

```

LISTING E.2: dpi unreal colosseumCharacter.cpp

Bijlage F

Conditional logic van Tick functie van LookEvents in c++

```
void ULookEventsComponent::TickComponent( float DeltaTime , enum ELevelTick TickType ,
FActorComponentTickFunction *ThisTickFunction )
{
    if ( bActive != true || ( bShouldUsedOnce && TimesUsed > 0 ) || bIsInTimeOut == true )
    {
        return ;
    }

    FHitResult HitResult = WasHitByTrace( Trace() );

    if ( HitResult.Actor.IsValid() )
    {

        if ( bIsInUnSeenDelay )
        {

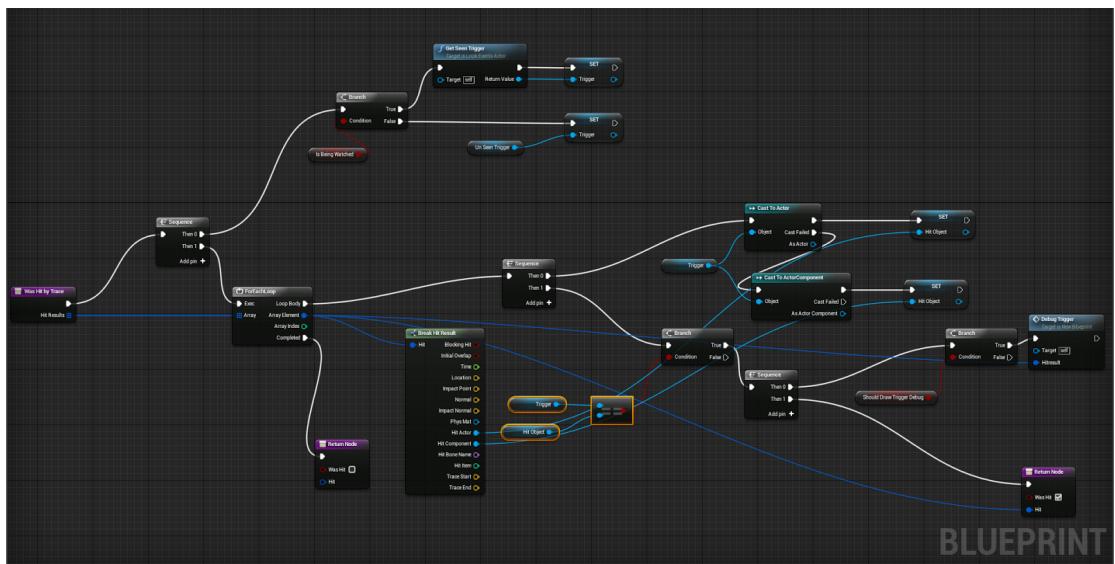
            if ( ! bIsInSeenDelay )
            {
                if ( SeenDelay > 0 && ! bIsSeenDelayFinished )
                {
                    } else if ( ! bIsBeingWatched )
                    {
                    }
                }
            }
        else if ( bIsInSeenDelay )
        {
        }
        else if ( bIsBeingWatched )
        {
            if ( UnSeenDelay > 0 )
            {
                if ( bIsUnSeenDelayFinished )
                {
                }
                else if ( ! bIsInUnSeenDelay )
                {

```

```
    }
40 } else
41 {
42 }
43 }
44 }
```

Bijlage G

Conditional logic van Tick functie van LookEvents in Blueprints



FIGUUR G.1: De conditionele logica van de Tick functie van de LookEvents in Blueprints.

Bijlage H

Oriëntatie Interview

Interviewer: Mark Arts. Deelnemers: Huib, Danny

H.1 Doel

Het doel van dit interview is om de huidige kennis over gameplay programmering, visuele programmeer talen en logica constructies vast te leggen bij de werknemers van DPI.

Daarnaast wordt er ook een eerst introductie gemaakt naar Blueprints, de visuele programmeer taal die in de Unreal Engine 4 gebruikt wordt.

Er wordt geprobeerd antwoord te krijgen op de volgende vragen:

- Welke ervaring met programmeren hebben de deelnemers
- Welke ervaring met UE4 hebben de deelnemers
- Welke ervaring met visueel programmeren hebben de deelnemers
- Tot hoe verre begrijpen de deelnemers programmeer terminologie (enquête)

H.2 Interview

Wat voor 3D projecten hebben jullie tot nu toe gemaakt en wat was jullie taak hierin

Danny

Voornamelijk het modellen / maken van 3D visualisaties in Max / Maya voor architectuur. Daarnaast een aantal 3D omgevingen in de Unreal Engine 4.

Huib

Voornamelijk het modellen / maken van 3D visualisaties in Max / Maya voor architectuur. Binnen DPI nog niet aan een UE4 project gewerkt.

Wat is jullie ervaring met Unreal Engine 4

Danny

Een aantal hobby projecten en binnen DPI het opzetten van een aantal kleine omgevingen in UE4. Lastig om modellen die voor maya / max gebruikt werden te importeren en rekening te houden met performance. De interface van de Engine is wel duidelijk en door ervaring met maya / max was het makkelijk om te beginnen met het bouwen van een omgeving.

Huib

Heeft alleen nog aan hobby projecten gewerkt binnen Unreal maar kon door ervaring met Maya en Max makkelijk aan de slag. Hij had wel moeite met het gebruik van Blueprints in demo's en vond het vaak overweldigend.

Wat weten jullie over programmeren

Danny

Geen ervaring met programmeren en weet er weinig over.

Huib

Heeft CMD gestudeerd en heeft tijdens zijn studie ervaring opgedaan met website's programmeren. Hij snapt hoe code werkt en wat je er mee kan doen maar zou niet C++ voor een game kunnen programmeren.

Wat voor ervaring hebben jullie met visuele programmeer talen

Danny

Geen ervaring.

Huib

Heeft Blueprints geprobeerd maar daarnaast geen ervaring.

H.3 Tot hoeverre begrijpen de deelnemers programmeer terminologie (enquête)

Er is door zowel Huib en Danny een vragenlijst ingevuld met de volgende introductie:

Omschrijf in eigen woorden wat jij denkt dat de volgende begrippen betekenen. Als het begrip onbekend is omschrijf wat jij denkt dat het betekent.

Danny

Branch (conditional statement)

Gok: ik denk een vertakking in verschillende nodes?

For loop

Gok: ik denk een loop in een script, bijvoorbeeld een walkcycle

Switch statement

Gok: True of false switch?

Integer

geen idee

Float

Gok: gravity? geen idee.

Boolean

objecten die met elkaar worden gecombineerd en kan worden bepaald of er gesubstract wordt.

Vectors

De punten van een object (uiteindes)

Transform

Geometry aanpassen in onderandere scalen

Struct (structur)

Gok: hoe een object is opgebouwd? quad en tris?

Array

Op deze manier wordt er iets geduplicate

Class

verschillende blueprints, zoals firstpersonmode of playercontroller.

Object Type

wat voor object, static mesh of een BSP

Propertie

instellingen van bepaalde actors.

Reference (pointer)

Je kan volgens mij objecten converteren naar triggerboxen.

Cast

Gok: ik heb het weleens langs zien komen, maar volgens mij zijn het nodes in het script van Blueprints

Actor

eigenlijk alles wat in de scene wordt gezet.

Component

Gok: heeft iets met de actors te maken, ik denk bepaalde instellingen daarin.

Huib

Branch (conditional statement)

Keus uit meerdere mogelijkheden / inputs

For loop

Iets doen wanneer iets een bepaalde waarde heeft...

Switch statement

Schakelaar met meer dan 2 mogelijkheden.

Integer

Heel getal

Float

Getal met decimalen

Boolean

True / False

Vectors

Punt in 3D space (x,y,z coördinaat)

Transform

Wat dit in de context van Unreal is weet ik niet, ik ken het als iets dat te maken heeft met de positie, rotatie en schaal van objecten.

Struct (structur)

Constructie van een element.

Array

Een reeks

Class

Een groep objecten met dezelfde eigenschappen.

Object Type

Een soort object, mesh, light, etc.

Propertie

Eigenschap van een object.

Reference (pointer)

Gok: Verwijzing naar iets

Cast

Gok: Iets versturen

Actor

Een object dat ergens op kan reageren.

Component

Een bouwblöök

Bibliografie

- [1] Maria Cutumisu, Curtis Onuczko, Matthew McNaughton, Thomas Roy, Jonathan Schaeffer, Allan Schumacher, Jeff Siegel, Duane Szafron, Kevin Waugh, Mike Carbonaro, Harvey Duff, and Stephanie Gillis. Scriptease: A generative/adaptive programming paradigm for game scripting. *Science of Computer Programming*, 67(1):32 – 58, 2007. ISSN 0167-6423. doi: <http://dx.doi.org/10.1016/j.scico.2007.01.005>. URL <http://www.sciencedirect.com/science/article/pii/S0167642307000536>. Special Issue on Aspects of Game Programming.
- [2] M. Cutumisu, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, C. Onuczko, and M. Carbonaro. Generating ambient behaviors in computer role-playing games. *IEEE Intelligent Systems*, 21(5):19–27, Sept 2006. ISSN 1541-1672. doi: 10.1109/MIS.2006.92.
- [3] JOHN F. PANE, CHOTIRAT “ANN” RATANAMAHATANA, and BRAD A. MYERS. Studying the language and structure in non-programmers’ solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237 – 264, 2001. ISSN 1071-5819. doi: <http://dx.doi.org/10.1006/ijhc.2000.0410>. URL <http://www.sciencedirect.com/science/article/pii/S1071581900904105>.
- [4] Jens Bauer and Achim Ebert. *Virtual Realities: International Dagstuhl Seminar, Dagstuhl Castle, Germany, June 9-14, 2013, Revised Selected Papers*, chapter Mobile Devices for Virtual Reality Interaction. A Survey of Techniques and Metaphors, pages 91–107. Springer International Publishing, Cham, 2015. ISBN 978-3-319-17043-5. doi: 10.1007/978-3-319-17043-5_6. URL http://dx.doi.org/10.1007/978-3-319-17043-5_6.
- [5] A. Martini, L. Colizzi, F. Chionna, F. Argese, M. Bellone, P. Cirillo, and V. Palmieri. A novel 3d user interface for the immersive design review. In *3D User Interfaces*

- (3DUI), 2015 IEEE Symposium on, pages 175–176, March 2015. doi: 10.1109/3DUI.2015.7131757.
- [6] Mark Mine. Towards virtual reality for the masses: 10 years of research at disney’s vr studio. In *Proceedings of the Workshop on Virtual Environments 2003*, EGVE ’03, pages 11–17, New York, NY, USA, 2003. ACM. ISBN 1-58113-686-2. doi: 10.1145/769953.769955. URL <http://doi.acm.org/10.1145/769953.769955>.
- [7] Randy Pausch, Jon Snoddy, Robert Taylor, Scott Watson, and Eric Haseltine. Disney’s aladdin: First steps toward storytelling in virtual reality. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’96, pages 193–203, New York, NY, USA, 1996. ACM. ISBN 0-89791-746-4. doi: 10.1145/237170.237257. URL <http://doi.acm.org/10.1145/237170.237257>.
- [8] Steffi Beckhaus, Kristopher J Blom, and Matthias Haringer. Intuitive, hands-free travel interfaces for virtual environments. In *New Directions in 3D User Interfaces Workshop of IEEE VR*, pages 57–60, 2005.