

VERSLAG

Realisatie Stage Mobilab

Lander Jacobs 3APP02

Inhoud

INLEIDING	5
1. STAGEOPDRACHT	6
2. ONDERZOEK	7
2.1. Unity	7
2.1.1. VR	7
2.1.2. Algemeen	8
2.2. Omnicept	11
2.2.1. Tools	11
2.2.2. Data	14
3. SETUP PROJECT	19
3.1. VR Project	19
3.1.1. XR Plugin Management	20
3.1.2. XR Interaction Toolkit	20
3.2. Omnicept-specifieke setup	22
3.2.1. Mixed Reality Feature Tool	22
3.2.2. Omnicept SDK pakket	23
3.3. Version control	24
4. BASIS VR-PROJECT	26
4.1. Input management	26
4.1.1. Input Action Map	26
4.1.2. Beweging controllers	27
4.1.3. Interactie	29
4.1.4. Vastgrijpen objecten	33
4.1.5. XR Interaction Manager	35
4.2. Recalibratie	35
4.2.1. XrRig	35
4.2.2. Unity Editor	37
4.2.3. Code	37
4.3. Audio management	40
4.3.1. Unity Editor	40
4.3.2. Code	42
4.4. UI management	46
4.4.1. Camera canvas	46
4.4.2. Handmenu	51
4.4.3. Fade	56

4.5. Scene overgang	60
4.5.1. Selecteren van een omgeving	61
4.5.2. ManagedScene	62
4.5.3. Transition_Manager	62
4.6. Teleportatie	66
4.6.1. Prefab	66
4.6.2. TeleportScript	67
4.6.3. Teleport_Manager	71
4.7. Activiteit management	73
4.7.1. Activiteit starten	73
4.7.2. Activity_Manager	74
4.7.3. Activiteit beëindigen	80
5. OMNICEPT DATA	83
5.1. Data Ophalen	83
5.1.1. Unity Editor	83
5.1.2. Code	84
5.2. Data Manipuleren	85
5.3. Data Gebruiken	90
5.3.1. Vlinder activiteit	91
5.3.2. Beweging van objecten	101
5.3.3. Waypoint manager	104
5.3.4. Route controller	106
5.3.5. Extra route controller	109
6. DOCUMENTATIE	111
7. OBSTAKELS	112
7.1. Poorten van de headset	112
7.2. Interaction Profiles	112
7.3. Omnicept Tray	112
7.4. Camera Canvas afstand	113
7.5. Teleport Circle Rigidbody	113
7.6. Builden applicatie verstoort werking editor	115
8. EVENTUELE UITBREIDINGEN	116
8.1. PPG	116
8.2. Universele manager	116
8.3. Forestmap manager	116
8.4. Kaart Forestmap	116
8.5. Update TeleportManagerScript	117

8.6. Scripts oproepen	117
8.7. Klasse waypoints	117
8.8. Inheritance voor scripts	117
8.9. Code controleren en aanpassen	118
9. CONCLUSIE	119
10. FIGUURLIJST	120
11. VERWIJZINGEN	126

Inleiding

Dit document beschrijft hoe het realisatieproces van mijn stage is verlopen. De realisatiefase begon in de eerste week van de stage en omvatte een tijdsduur van 12 weken.

In het eerste segment van deze tekst, zal in het kort het stagebedrijf besproken worden waar ik de kans kreeg om mijn stage te doen alsook zal de stageopdracht zelf kort toegelicht worden. Meer informatie omtrent beide onderwerpen kan u uitgebreider terug vinden in mijn plan van aanpak.

Hierna doe ik een uiteenzetting van mijn precieze uitvoering en werk tijdens dit project. Hierbij wordt grotendeels de volgorde aangehouden die ik heb beschreven in mijn plan van aanpak, deze is doorheen het project namelijk een aantal keer verandert.

De eerste sectie is zoals in het plan van aanpak: een onderzoek naar wat we gingen gebruiken en wat we nodig hadden. Vervolgens is de sectie "Setup/Speelbaarheid" verandert in twee onderdelen: "Setup" en "Basis VR-project". Dit is doordat we tijdens deze fase doorhadden dat het beter was om dit deel op te splitsen in twee onderdelen om zo de duidelijkheid en overzichtelijkheid te vergroten.

Daarna hadden we verwacht dat we konden zouden werken aan het topic "Datalogebruik" om te testen hoe we de gegevens van de headset konden gebruiken en deze vervolgens zouden kunnen implementeren. Na het onderzoek van de eerste periode was het duidelijk dat er niet veel te testen was en zodoende zijn we sneller doorgegaan met het ophalen van de data, deze te "manipuleren" en dan te gebruiken in enkele scenario's die we moesten uitschrijven.

Vervolgens zal ik het nog hebben over de documentatie die ik heb geschreven om af te leveren aan het einde van de stage. Op het einde geef ik ook nog wat obstakels weer waar ik ben tegenaan gelopen en hoe ik deze heb opgelost.

1. Stageopdracht

Ik heb mijn stage gedaan bij Mobilab & Care, een onderzoeks lab gevestigd op de Thomas More Campus te Geel. Mobilab houdt zich bezig met het inclusiever maken van onze maatschappij, dit doen ze op sociaal, mentaal en fysiek vlak. Zo werken ze ook met VR om mensen tot rust te brengen en te leren op zichzelf tot rust te komen.



Figuur 1: HP Reverb G2 Omnicept Edition

Voor de stageopdracht moest ik samen met een andere stagiair een VR-applicatie maken voor de "HP Reverb G2 Omnicept Edition", een VR-headset die verschillende biologische feedback kan lezen. Deze applicatie moest gebruikmaken van deze biofeedback om de gebruiker te ondersteunen in het tot rust komen. Uiteindelijk werd deze applicatie ook gedocumenteerd zodat de applicatie in de toekomst gemakkelijk kan uitgebreid of aangepast worden.

Voor meer informatie over de stageplek of de stageopdracht verwijs ik u graag door naar het plan van aanpak.

2. Onderzoek

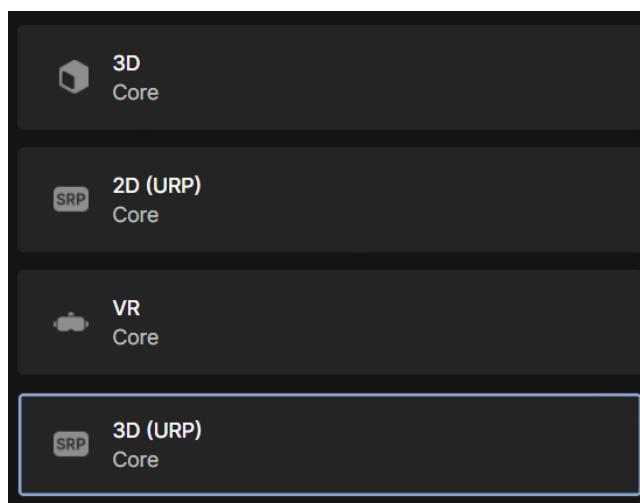
Dit was de eerste fase van de stage en diende om ons onderzoek te laten doen naar wat we nodig hadden om te werken aan de applicatie. Zo kunnen we wat ik heb geleerd opdelen in 2 onderdelen: Unity en Omnicept. Unity omvat alles wat we nodig hebben om een VR-applicatie te maken binnen Unity. Voor Omnicept zal het grootste deel van het onderzoek in dit hoofdstuk worden uitgelegd daarnaast zal kort besproken worden wat we nodig hebben voor de setup met Omnicept.

2.1. Unity

Unity is een game engine, dit betekent dat je hier verschillende games kunt ontwikkelen. Voor deze opdracht is het de bedoeling dat we een VR-game maken. Daarom is het meeste van ons onderzoek dan ook uitgegaan naar hoe we binnen Unity een VR-game kunnen genereren. Daarnaast omvat Unity ook onderdelen die nodig zijn voor een applicatie, ongeacht of dit een VR-game is of gewoon een 3D-game.

2.1.1. VR

Unity zelf heeft meerdere templates om vanuit te starten. Voor een VR-game wordt aangeraden om een 3D-template te gebruiken. Hier kunnen we dan achteraf de onderdelen om te werken met VR aan toevoegen. Hierdoor krijgen we meer controle en wordt ervoor gezorgd dat er niets overbodig wordt aangemaakt, zo houden we de uiteindelijke applicatie ook zo klein en, hopelijk, zo performant mogelijk.



Figuur 2: Unity Templates

Voor onze applicatie kozen we voor de URP-versie [1] van een 3D-app. URP staat voor "Universal Render Pipeline" en zorgt ervoor dat we de graphics van de applicatie zo gemakkelijk en goed mogelijk kunnen aanpassen naar wat we wensen. Ikzelf heb niet veel gewerkt met de grafische kant van de applicatie, dit was een onderdeel van de stage van mijn mede-stagiair.

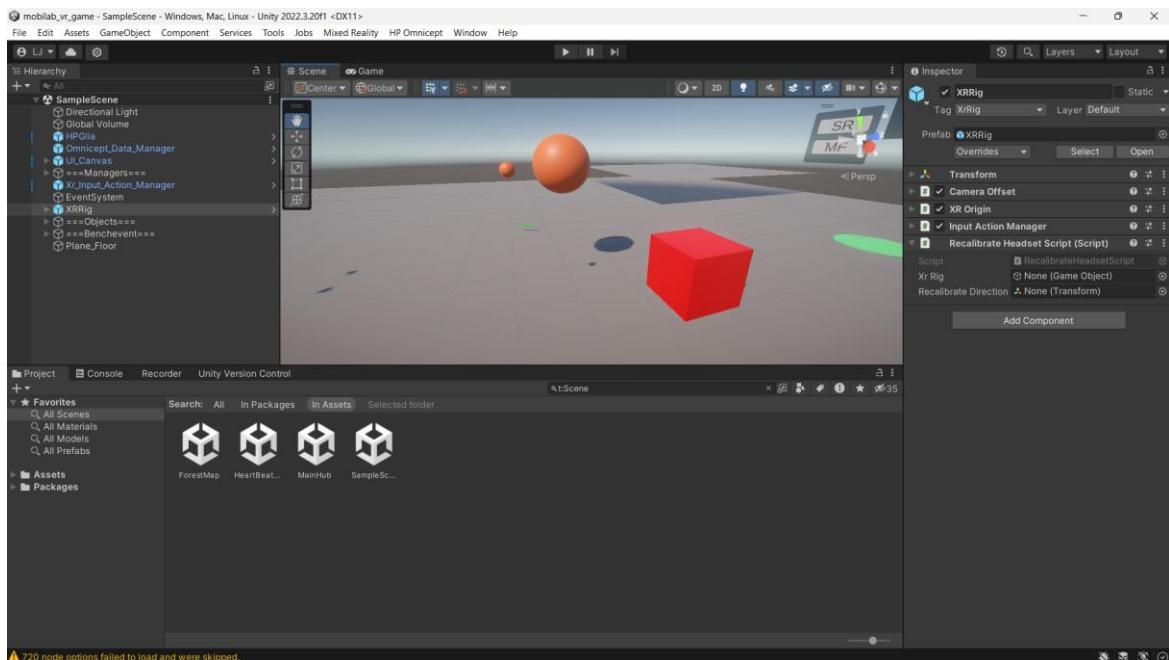
De Extra uitleg over de extra pakketten die we hebben toegevoegd om te werken met VR worden dieper en gedetailleerder uitgelegd bij de Setup van het project.

2.1.2. Algemeen

Om te kunnen werken aan een Unity project, hebben we twee editors nodig. Eén voor het Unity project zelf en één om de code te schrijven die de game zal laten draaien. Het Unity project zelf zullen we maken in de Unity editor en voor de code maken we gebruik van Visual Studio 2022.

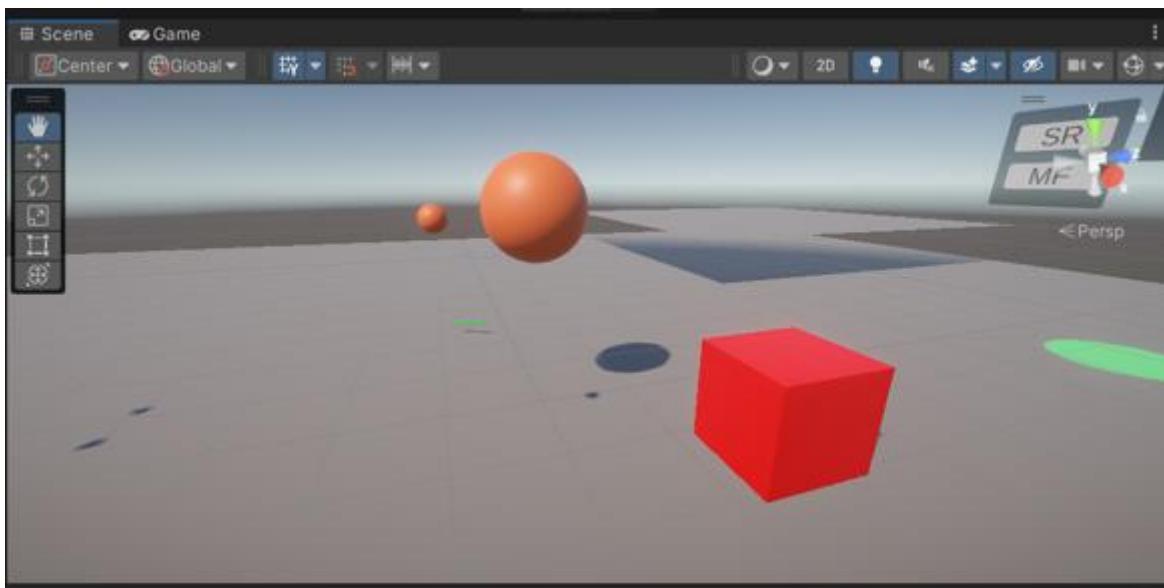
2.1.2.1. Unity editor

Binnen de Unity editor kunnen we het project beheren dat onze game laat draaien. Dit gaat van het toevoegen van objecten tot het uitbreiden van deze objecten met componenten.



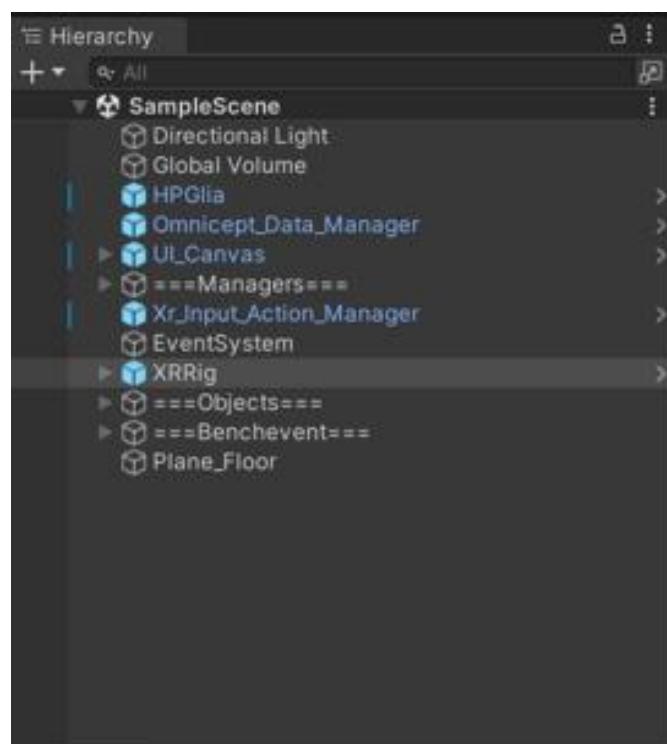
Figuur 3: Unity Editor

Het eerste en meest opvallende onderdeel van de editor is de "Scene View", hier zien we alle gameobjecten [2] die we in onze "Scene" plaatsen. Een "Scene" is een omgeving waar we als gebruiker kunnen rondbewegen. Een game bestaat uit meerdere "Scenes" waartussen we ons kunnen verplaatsen om andere omgevingen te laten zien. Daarnaast zien we boven ook naast het tabblad "Scene" ook het tabblad "Game". Wanneer we willen zien hoe onze game runt, gaan we naar dit tabblad en starten we de game op om te zien hoe dit verloopt.



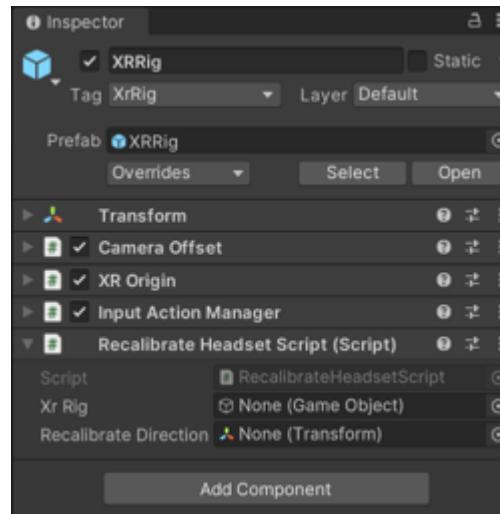
Figuur 4: Unity Scene

Links in figuur 3 zien we de “Hierarchy” staan, dit zijn alle objecten die in onze “Scene” verschijnen. De objecten die we in de “Hierarchy” slepen worden “GameObjects” genoemd. Deze krijgen standaard een positie en transformatie. Hierbij heeft de daadwerkelijke volgorde niet echt een groot effect. Het enige wat er hier invloed op elkaar zou hebben is wanneer een object een “child” is van een ander object. Wanneer objecten “child” zijn van andere objecten heeft dit een invloed op enkele eigenschappen zoals bijvoorbeeld hun positie en rotatie binnen de “scene”.



Figuur 5: Unity Hierarchy

Wanneer we een “Gameobject” selecteren, kunnen we rechts, zoals te zien is in figuur 3, in de “Inspector” alle componenten zien die hieraan hangen. Deze componenten zijn scripts die we zelf schrijven of die Unity aanbiedt. Deze kunnen verschillende effecten hebben die gaan van het verplaatsen in de 3D-omgeving tot interactie met andere “Gameobjects”.



Figuur 6: Unity Gameobject Inspector

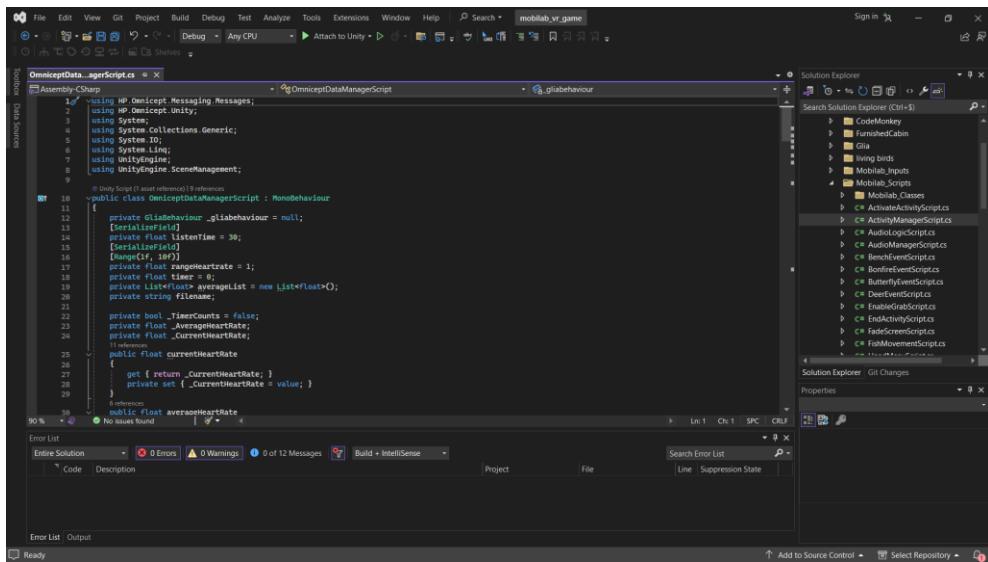
Onderaan de editor vinden we ook de assets die we kunnen gebruiken. Dit zijn alle onderdelen die we kunnen aanpassen in een Unity project. Assets zijn alles dat we kunnen gebruiken als “GameObjects”, de scripts en de “Scenes”. Verder zijn er nog enkele andere tabbladen: “Console”, “Recorder” en “Unity Version Control”. De “Console” gebruiken we tijdens het werken met scripts om te checken waar we zitten en of de code werkt. “Recorder” kunnen we gebruiken om de een video te maken van wat we aan het testen zijn. Deze opnames waren nodig voor de mentoren om te zien hoe ver we stonden in de development van onze game. Het tabblad “Unity Version Control” was nodig omdat we met twee personen aan dit project aan het werken waren. Voor “Unity Version Control” maakten we gebruik van een andere applicatie “Unity Devops Version Control”, deze heeft een duidelijker en uitgebreidere UI om mee te werken.



Figuur 7: Unity Assets

2.1.2.2. Visual Studio 2022

Naast de Unity Editor hebben we ook nood aan een editor voor onze scripts. Omdat Unity C# kan lezen voor het schrijven van scripts viel onze keuze op "Visual Studio 2022" [3]. Unity kan verschillende programmeertalen gebruiken voor de scripts, maar de documentatie die we kunnen vinden is voornamelijk geschreven voor C#. "Visual Studio 2022" is de beste editor die we kunnen vinden om te werken aan C# projecten, deze hebben we in het verleden ook al gebruikt.



Figuur 8: Visual Studio 2022

2.2. Omnicept

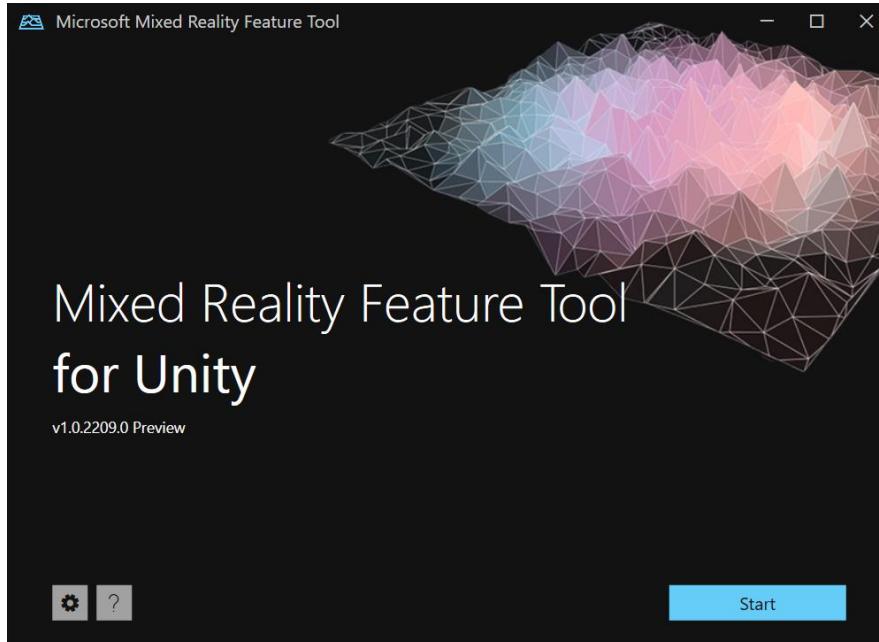
Voor het werken binnen Unity met een HP Reverb G2 Omnicept Edition is er een extra setup nodig dan wanneer we werken met een andere VR-headset. Dit is zo omdat deze headset ook biologische data meet en teruggeeft. Maar daarbovenop heeft HP nog extra vereisten om te werken met de headset en de controllers. Dit heeft betrekking tot het laten werken van een applicatie in de headset en het laten bewegen van de headset en controllers binnen een applicatie. Daarnaast hebben we de verschillende soorten biofeedback ook nagekeken op de vorm waarin ze wordt teruggegeven en of we deze kunnen gebruiken om te zien wanneer een gebruiker rustig is.

2.2.1. Tools

De tools die we nodig hebben zijn opgedeeld in twee pakketten die we online kunnen vinden. Het eerste pakket is de "Mixed Reality Toolkit" [4] voor samenwerking met de headset en controllers. Het tweede pakket is de "Omnicept SDK" [5], deze bevat meerdere applicaties die we kunnen gebruiken om het lezen van data te testen.

2.2.1.1. Mixed Reality Feature Tool

Deze tool helpt ons om de headset te laten kijken in onze app. Ook geeft deze het profiel dat we kunnen gebruiken om de controller input van de fysieke controllers te lezen in onze app.



Figuur 9: Mixed Reality Feature Tool

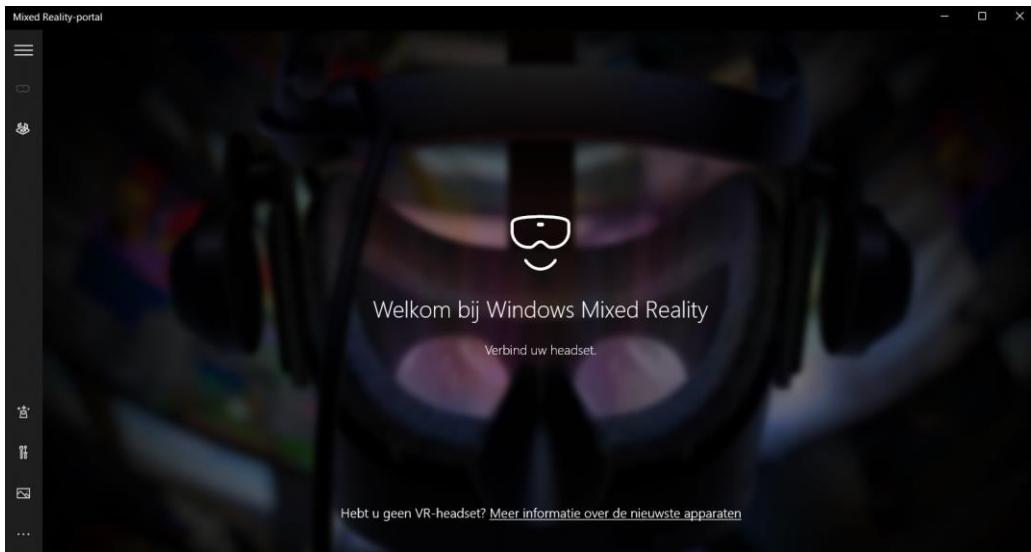
2.2.1.2. Omnicept SDK

Om de data van de headset te lezen moeten we de "Omnicept Developer SDK" downloaden. Daaruit krijgen we enkele programma's om te gebruiken tijdens het werken aan de app.

<p>Omnicept Users:</p> <p>Download latest HP Omnicept Runtime</p> <p>Remember to run Eye Tracking Calibration after installation.</p>	<p>Omnicept Developers:</p> <p>You must be logged in with a registered HP ID to download.</p> <p>Download latest HP Omnicept SDK</p> <p>First time here? Follow the getting started guide to gain access to Omnicept sensor data.</p> <p>Returning users? Remember to re-import your plugin after updating the Omnicept SDK.</p>	<p>Foveated Rendering:</p> <p>Download Eye Tracking</p> <p>Get started here: Developers and Users</p>
--	---	--

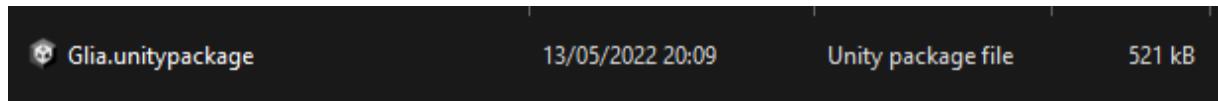
Figuur 10: Omnicept Developer SDK

Een van de eerste apps die wordt toegevoegd is het "Windows Mixed Reality Portal". Deze wordt geactiveerd elke keer dat de headset wordt geconnecteerd met de laptop. Hier kan men de headset testen voordat een game, zoals de app die we maken, wordt gestart. Hier kan men bevestigen of het beeld en het geluid in orde zijn en of de headset comfortabel genoeg zit voordat een game wordt gestart.



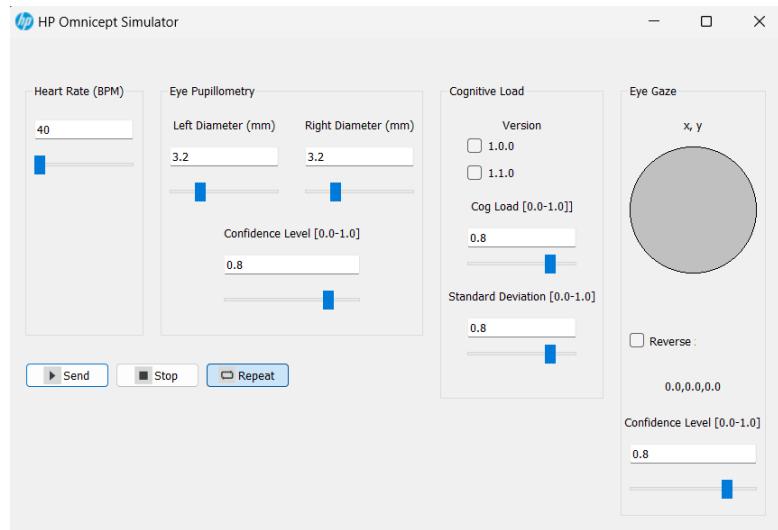
Figuur 11: Mixed Reality Portal

Ook zijn er verschillende pakketten geïnstalleerd voor development met de headset. Voor ons is de belangrijkste het pakket voor Unity, maar het is ook mogelijk om met "Unreal Engine" [6] te werken aan deze headset. Alleen zou er dan betaald moeten worden voor het mogen werken met de editor van "Unreal Engine".



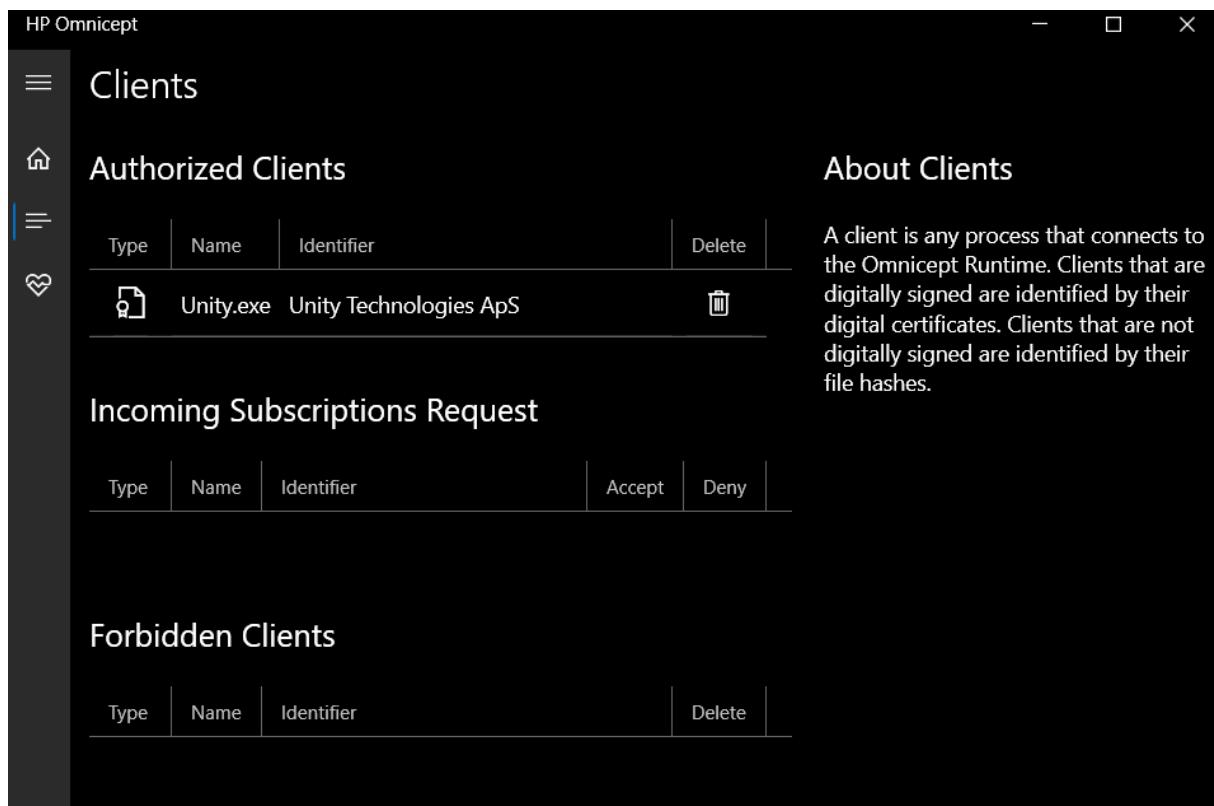
Figuur 12: Unity package voor Glia

De volgende tool die werd geïnstalleerd is de "HP Omnicept Simulator", deze kan de meeste onderdelen die we wensen te testen simuleren. Het enige dat niet gesimuleerd kan worden is de PPG omdat deze niet automatisch wordt vrijgegeven (lees hierover meer onder het titeltje "Data").



Figuur 13: HP Omnicept Simulator

De laatste tool waarvan we gebruikmaakten is de “HP Omnicept Tray”-applicatie. Deze moet slechts eenmalig gebruikt worden om te bevestigen dat onze eigen applicatie, en de Unity editor, toegang hebben tot de data uit de headset.



Figuur 14: HP Omnicept Tray

2.2.2. Data

De Omnicept kan verschillende biologische gegevens teruggeven [7] . Twee hiervan zijn biologische kenmerken, een ander biologisch gegeven wordt berekend met behulp

van deze twee biologische kenmerken en het laatste is een extra onderdeel van de headset dat niet meteen een specifieke functie lijkt te hebben.

2.2.2.1. Hartslag

De hartslag die wordt gegeven door de Omnicept is berekend op basis van PPG. PPG staat voor Photoplethysmography en is een meting van de bloedcirculatie aan de hand van de veranderingen van het volume van het bloed op een specifieke plaats in het lichaam [8] . De Omnicept meet deze met een sensor op het voorhoofd. De hartslag wordt uitgedrukt in slagen per minuut (BPM).

Sensor data [HeartRate]

- **[unit] Rate:** Beats Per Minute (BPM)

Figuur 15: Hartslag Data voor Unity

Van de hartslag kunnen we ook de variabiliteit verkrijgen. De hartslagvariabiliteit is te verkrijgen als SDNN, zijnde de standaarddeviatie tussen twee normale hartslagen. Deze kan ook worden verkregen als RMSSD, zijnde het kwadratisch gemiddelde van opeenvolgende verschillen. Deze twee manieren om dit te meten worden beiden uitgedrukt in milliseconden.

Sensor data [HeartRateVariability]

- **[float] Sdnn:** Standard deviation between two normal heartbeats in milliseconds
- **[float] Rmssd:** Root-mean square of successive differences in milliseconds

Figuur 16: Hartslagvariabiliteit data voor Unity

De ruwe data van de PPG kan gebruikt worden om zelf de hartslag te berekenen en zelf te bepalen hoe we deze interpreteren. Deze kan ook worden gebruikt mits er hier toestemming voor wordt gegeven door HP zelf. Aangezien we deze toestemming niet hadden tijdens de stage heb ik dan ook niet kunnen werken met de PPG.

Special access data

This data can only be accessed with special permission and legal agreement per use case.

PPG

Sensor data [PPGFrame]

- **[List<PPG>]** Data
 - **[List<unit>]** PpgValues
- **[List<unit>]** LEDCurrents

Figuur 17: PPG data voor Unity

Voor deze opdracht zou het interessant geweest zijn om te werken met de data van de PPG. Daarmee hadden we dan zelf kunnen kijken hoe we deze interpreteerden en gebruikten in de applicatie. Zodoende hebben we tijdens deze stage alleen gewerkt met de hartslag-variabele die we konden verkrijgen. Deze wordt niet door ons berekend, maar is wel het beste wat we hadden om te kunnen zien of de gebruiker rustig wordt of niet.

2.2.2.2. Oogbeweging

De Omnicept kan verscheidene variabelen meten voor de ogen. Er is een algemene gaze-variabele die ons vertelt waar beide ogen gezamenlijk naar kijken. Vervolgens worden de overige variabelen specifiek voor elk oog apart berekent of gemeten. Voor elk oog apart krijgen we ook de positie van de pupillen door middel van een x- en y-coördinaat. Als laatste biedt de Omnicept ons ook de “pupildilation”, met name hoe wijd elke pupil openstaat, en “eyeopenness”, hoe open de oogleden zijn per oog.

Sensor data [EyeTracking]

- **[Eye]** LeftEye / RightEye
 - **[EyeGaze]** Gaze: x,y,z eye gaze.
 - **[PupilPosition]** PupilPosition: x,y position.
 - **[float]** PupilDilation
 - **[float]** PupilDilationConfidence
 - **[float]** Openness
 - **[float]** OpennessConfidence
- **[EyeGaze]** CombinedGaze: x,y,z eye gaze.

Figuur 18: Oogbeweging data voor Unity

Echter met deze gegevens is het moeilijk om na te gaan hoe kalm een persoon is. Dit is dan ook de reden waarom we deze gegevens niet meteen hebben gebruikt.

2.2.2.3. Cognitive load

“cognitive load” [9] wordt omschreven als de mentale fortitude die nodig is om een taak uit te oefenen of die gebruikt wordt in een leerproces. Deze wordt berekend aan de hand van de hartslag en de oogbewegingen door middel van berekeningen die zijn gemaakt door het team achter de Omnicept zelf [10]. De waarde die we kunnen terugkrijgen wordt weergegeven met een getal van 0 tot 1.

Sensor data [CognitiveLoad]

- **[float]** CognitiveLoadValue
- **[float]** StandardDeviation
- **[string]** DataState

Figuur 19: Cognitive Load voor Unity

De “cognitive load” is zeer duidelijk in wat ze weergeeft, maar kon binnen ons project niet meteen gebruikt worden.

2.2.2.4. Mondcamera

De mondcamera is een camera die gericht staat op de mond. De camera geeft een zwart-wit beeld terug van de mond die wordt gefilmd. Deze lijkt over het algemeen geen specifieke toepassing te hebben. In het begin heerste bij mij het idee dat de “cognitive load” werd berekend met behulp van, onder andere, de bewegingen van de mond, opgenomen met deze camera. Helaas bleek na het lezen van de documentatie over “cognitive load” dat het gebruiken van gezichtstrekken niet de beste manier is om dit te doen.

Sensor data [CameralImage]

- **[FormatT]** Format: Image format
- **[byte[]]** ImageData: Raw image data.
- **[unit]** Width
- **[unit]** Height
- **[ulong]** FrameNumber
- **[float]** FramesPerSecond

Figuur 20; Mondcamera voor Unity

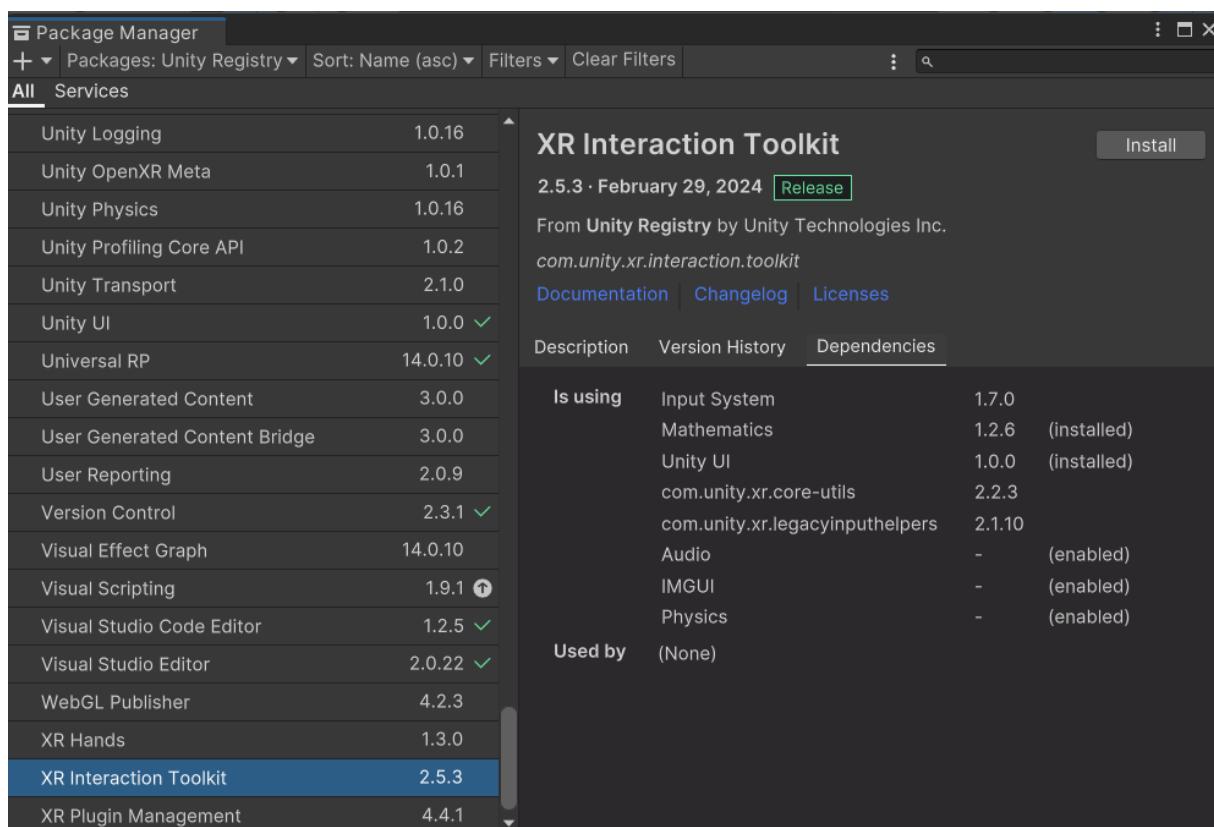
Verder hebben we ook geen praktische toepassing gevonden voor dit onderdeel.

3. Setup Project

We hebben gebruik gemaakt van een 3D-template om ons project aan te maken. Daarnaast hebben we in de voorgaande stap, ervoor gezorgd dat alle vereisten voor de Omnicept gedownload en beschikbaar zijn. Vervolgens werd het project dan ook uitgebreid met de basis benodigdheden van een VR-project met de Omnicept headset. Omdat ik met een andere stagiaire samenwerkte, hebben we ook gebruik gemaakt van version control binnen Unity.

3.1. VR Project

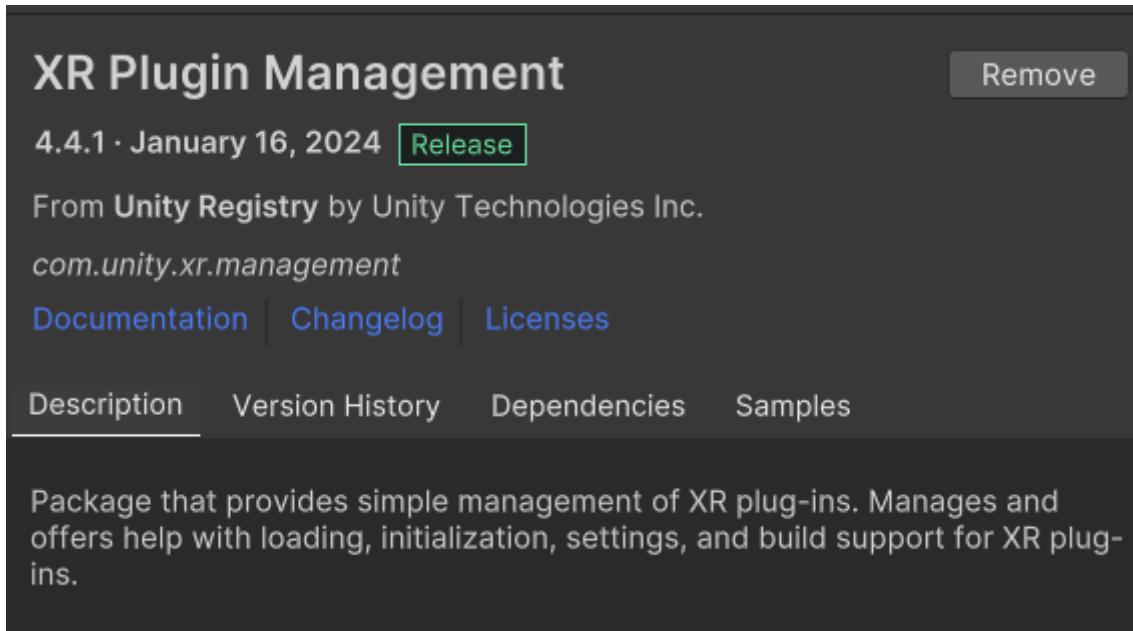
Om ons project uit te breiden voor VR waren er slechts twee pakketten die extra moesten geïnstalleerd worden: "XR Plugin Management" [11] en "XR Interaction Toolkit" [12]. Deze eerste zorgt ervoor dat je alles in verband met XR kunt managen. Het tweede pakket, met name de "XR Interaction Toolkit", liet ons toe enkele mogelijkheden die standaard zijn ingesteld in de VR-template toe te voegen als we deze wensten te gebruiken. De term XR staat voor "Extended Reality" en is de verzamelnaam voor alle technieken die de realiteit "uitbreiden". Enkele voorbeelden hiervan zijn "Augmented Reality" (AR), "Mixed Reality" (MR) en "Virtual Reality" (VR).



Figuur 21: Unity Package Manager

3.1.1. XR Plugin Management

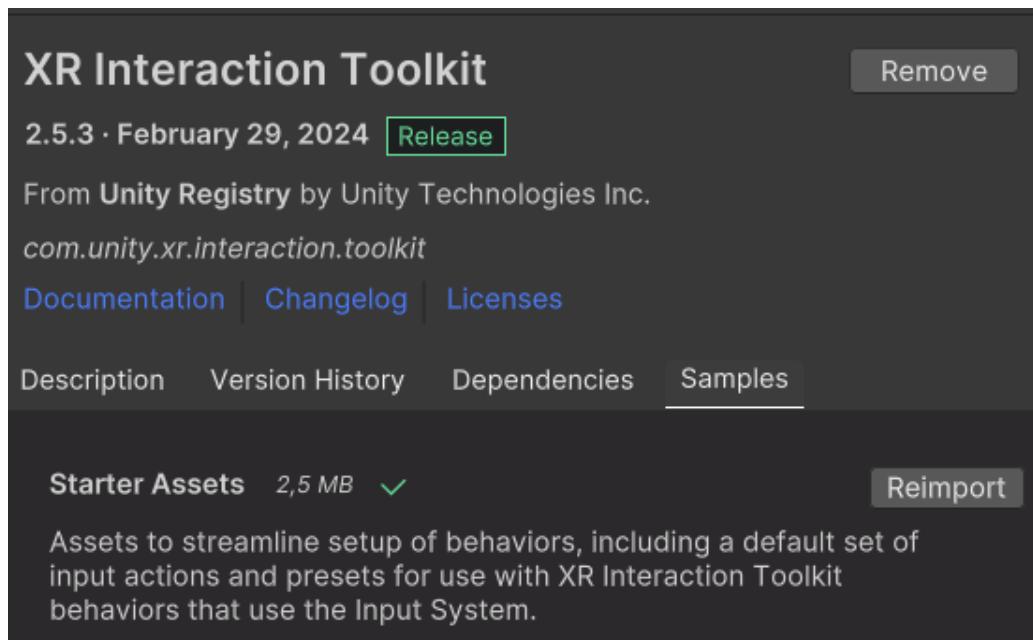
De installatie van dit pakket had niet meteen extra aandacht nodig. Bij de “XR Interaction Toolkit” kwam de installatie hiervan voor het eerst terug. Later wanneer we de “Mixed Reality Feature Tool” gebruikten, zou hier ook weer naar gekeken worden.



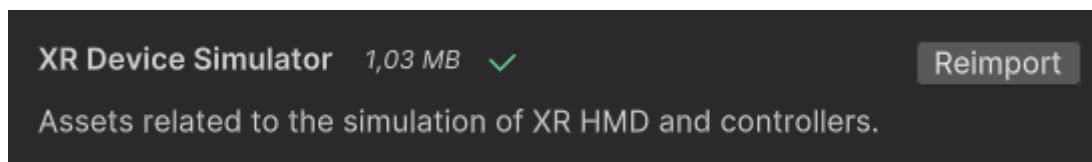
Figuur 22: XR Plugin Management pakket

3.1.2. XR Interaction Toolkit

Het installeren van het “XR Interaction Toolkit”-pakket deed een prompt verschijnen die vroeg om het “New Input System” te installeren. Deze werd later gebruikt om specifiek de input van de controllers en het bewegen van de headset te lezen. Nadat alle pakketten waren geïnstalleerd verscheen er een nieuw tabblad bij het “XR Interaction Toolkit”-pakket met als naam “Samples”. Daaronder vonden we meerdere voorgemaakte onderdelen waarvan we “Starter Assets” [13] en “XR Device Simulator” [14] installeerden. De “XR Device Simulator” simuleert een VR-headset binnen de editor die we gebruiken, zo konden we werken zonder elke keer de fysieke headset te gebruiken voor het testen.

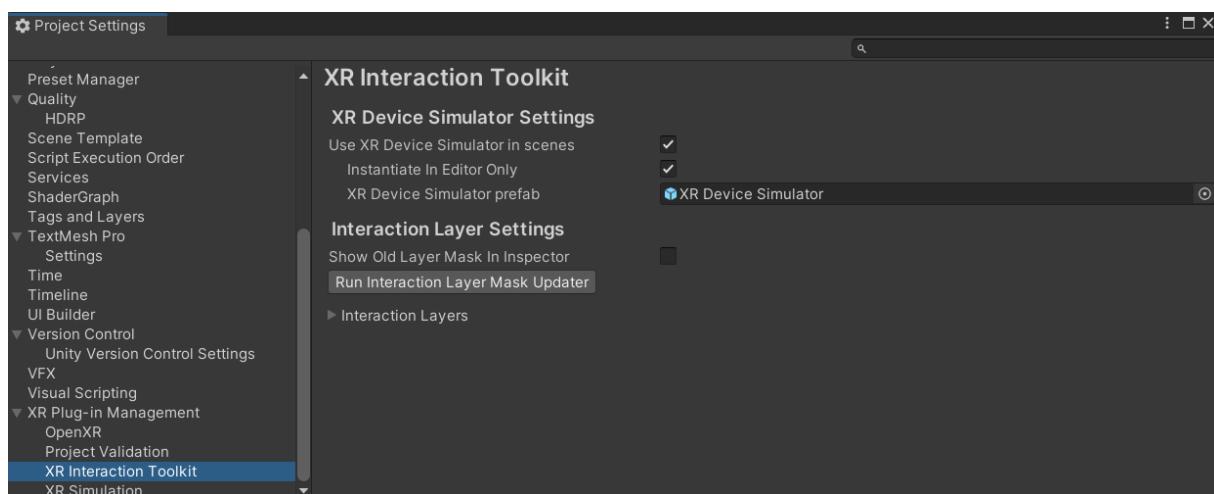


Figuur 23: XR Interaction Toolkit met Starter Assets sample



Figuur 24: Device Simulator sample

Nadat de "XR Device Simulator" was toegevoegd aan ons project konden we deze vanuit de "Project Settings" inschakelen voor elke scene. Wanneer we met de echte headset werkten in de editor moest deze wel weer uitgeschakeld worden.



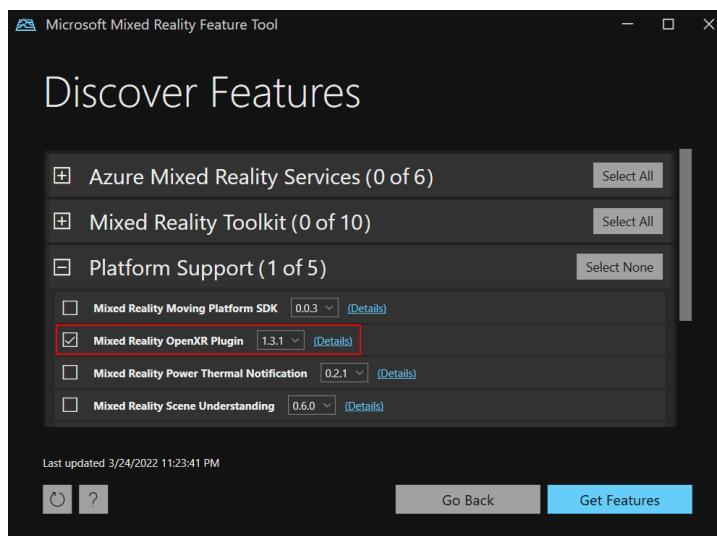
Figuur 25: Device Simulator toevoegen aan scenes

3.2. Omnicept-specifieke setup

De Omnicept zelf vereiste twee grote stappen om gebruikt te kunnen worden. De eerste stap omvatte de “Mixed Reality Feature Tool” voor het ontvangen van input en beweging van de headset en de controllers. De tweede stap was een pakket vanuit de “Omnicept SDK” voor het krijgen van de data die de headset kan aanleveren.

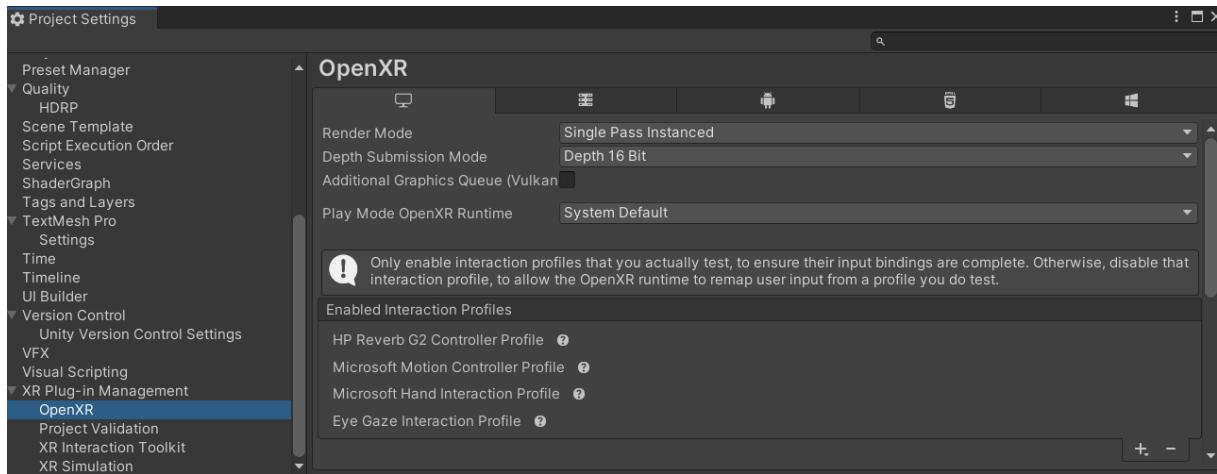
3.2.1. Mixed Reality Feature Tool

Wanneer we ons project hadden aangemaakt en “XR Plugin Management” toegevoegd was, konden we gebruikmaken van de “Mixed Reality Feature Tool”. Deze hielp ons met het installeren van de “Mixed Reality OpenXR Plugin” [15]. Deze konden we dan gebruiken om, onder andere, het juiste “Controller Profile” voor de controllers toe te voegen.



Figuur 26: Mixed Reality OpenXR Plugin

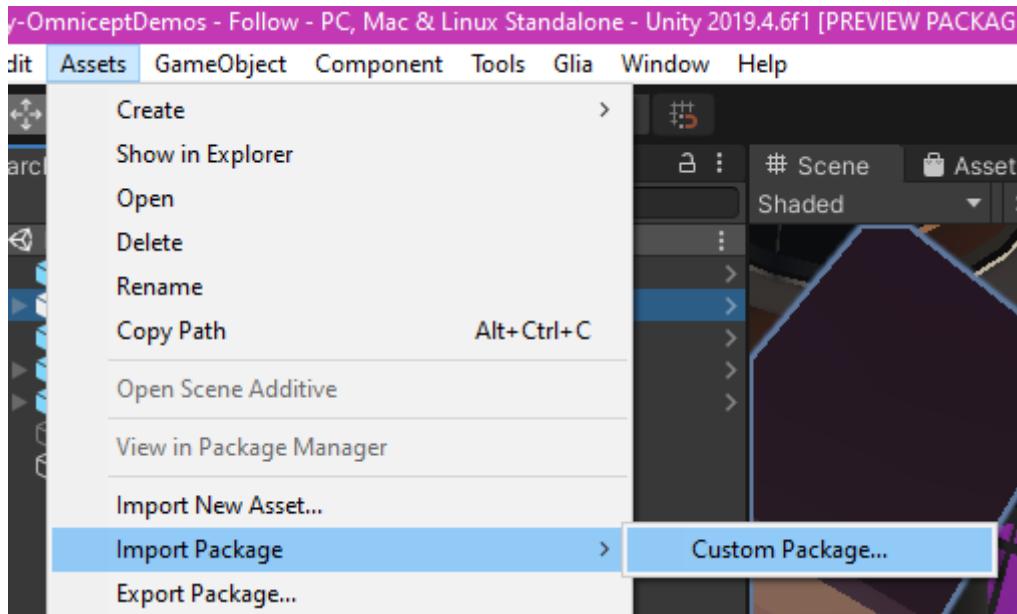
Nadat deze plugin was geïnstalleerd in ons project hebben we onder de “XR Plugin Management”-tabblad in onze Project Settings de “HP Reverb G2 Controller Profile” toegevoegd onder de “Enabled Interaction Profiles”. Deze zorgt ervoor dat de input van de controllers gelezen kan worden.



Figuur 27: Enabled Interaction Profiles

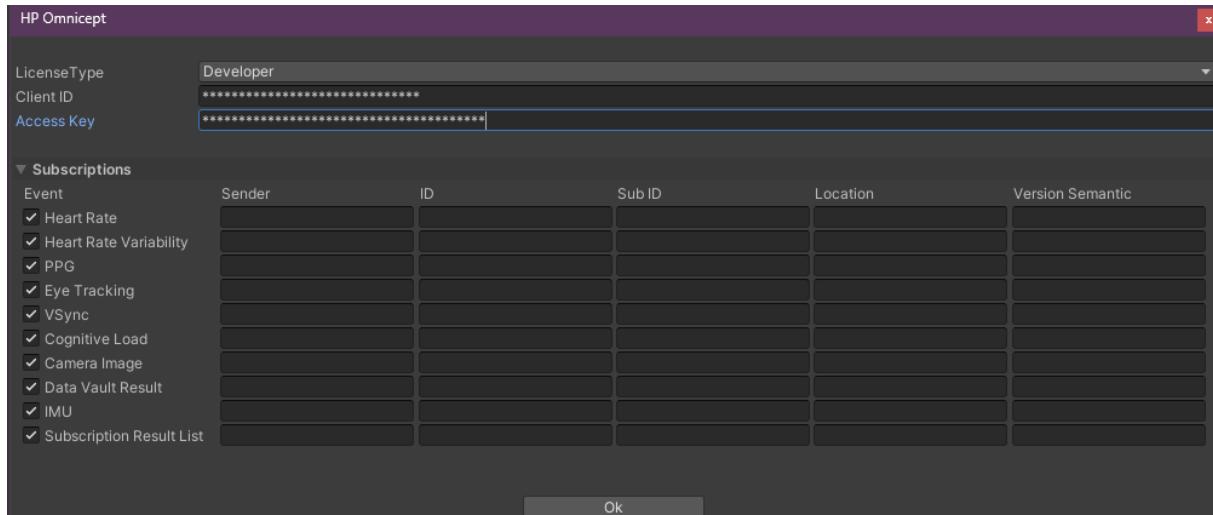
3.2.2. Omnicept SDK pakket

Voor de installatie en het eerste keer opzetten van het Unity-pakket is er een zeer duidelijke handleiding te vinden op het platform van de omnicept zelf [16]. Voor het installeren van dit pakket konden we gebruikmaken van de optie om een custom package toe te voegen. Deze optie opent de verkenner, waarna we kunnen navigeren naar het Unity-pakket van de SDK.



Figuur 28: Custom Package toevoegen

Wanneer dit was geïnstalleerd, moesten we access-keys gebruiken om de toegang tot de headset te doen werken. Dit is een veiligheid die er in zit zodat we zeker zijn dat de data van een product van HP komt. Deze keys kunnen we aanmaken via de Omnicept-console [17].



Figuur 29: HP Omnicept access keys

Wanneer we de Omnicept, of de Omnicept Simulator die bij in de Omnicept SDK zit, voor het eerst gebruiken, moesten we bevestigen dat Unity gebruik mag maken van de Omnicept. Naast het bevestigen dat de data van een HP-product komt, checken deze keys ook of er gebruik mag gemaakt worden van PPG-data. Voor de bevestiging maken we gebruik van de HP Omnicept Tray-app, dit moet slechts eenmaal gebeuren per app.

3.3. Version control

Omdat ik samen heb gewerkt met een andere stagiaire was er ook nood aan online version control zodat we samen aan het project konden werken. Origineel gingen we hiervoor Github gebruiken, maar al snel bleek dat Github de grootte van sommige bestanden niet aan leek te kunnen. Daarom kozen we voor "Unity Devops Version Control" [18], een app die eerst third-party was maar later onder Unity viel.

Het is mogelijk om binnen de Unity editor gebruik te maken van de version control die Unity aanbiedt. Daarentegen is de "Unity Devops Version Control"-app veel duidelijker en uitgebreider in wat het kan. Daarnaast zorgde een onderscheid in werken aan de app en version control voor mij persoonlijk ook voor structuur.



Unity DevOps

Figuur 30: Unity Devops Version Control

4. Basis VR-project

Voor de opdracht moet de app kunnen werken met de Omnicept. Daarnaast is er ook nood aan de basis vereisten van een VR-game. Deze hebben niet direct iets te maken met het gebruik van de Omnicept, maar zijn wel nodig om een werkend product te hebben.

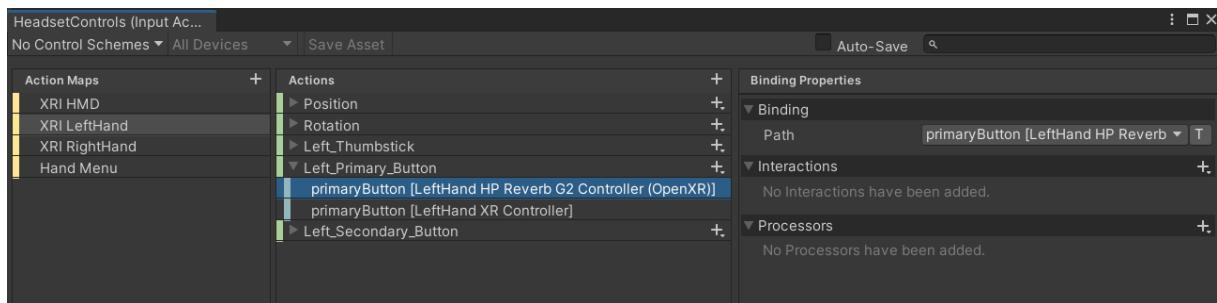
4.1. Input management

Voor het lezen van de input maken we gebruik van het “new” input systeem [19] van Unity. Deze hebben we eerder geïnstalleerd met het toevoegen van de “XR Interaction Toolkit”. Door het gebruiken van dit Input systeem kunnen we ook gebruikmaken van het juiste “Controller Profile”.

Naast het registreren van de input kunnen we dit ook gebruiken om de beweging van de headset en de controllers te registreren. De “XR Interaction Toolkit” heeft enkele van deze “input-maps” al standaard toegevoegd. Door het verschil in controllerprofielen is het wel handig dat we zelf onze eigen “inputmaps” maken.

4.1.1. Input Action Map

Als we in onze Assets-folder rechtsklikken, kunnen we “Input Actions” aanmaken. Zo maakten we een lege “Input Action Map” aan. Aan onze linkerkant creëerden we verschillende “Action Maps”, deze kunnen we instantiëren in de scripts waar moet geluisterd worden naar een van de inputs. Wanneer we hier een van selecteren, kunnen we hier meerdere “Actions” aan toevoegen. Deze “Actions” zijn de termen waarmee we bevestigen dat we luisteren naar een bepaalde waarde. Per “Action” kunnen we dan verschillende “Binding Properties” toevoegen. Deze “Binding Properties” zijn de input van de controllers, deze hebben ook verschillende waarden waar we later op terugkomen. In principe is het toevoegen van verschillende “Binding Properties” om de input van meerdere verschillende controllers op te pakken. Aangezien deze app alleen bedoelt is om te werken met de Omnicept moeten we dan ook alleen maar de “Binding Properties” die zijn vrijgegeven door het controllerprofiel van HP toe voegen. Helaas was het voor het werken met de “XR Device Simulator” ook nodig om bij elke “Action” een tweede “Binding Property” toe te voegen zodat deze werkte.



Figuur 31: Input Action Map – HeadsetControls

Afhankelijk van de “Binding Properties” kunnen er verschillende acties worden uitgevoerd. In het voorbeeld hieronder is het slechts een druk op een knop. Dit is wel een goed voorbeeld om te laten zien hoe een script luistert naar de input van de controllers. Om te beginnen moet er een globale variabele worden aangemaakt met als datatype de klasse van onze “Input Actions Map” (zie linksboven vorige figuur voor naam van de klasse). Vervolgens vullen we deze waarde in de “Awake”-methode [20] , deze methode is de eerste die wordt uitgevoerd in een script, eenmalig, wanneer het script start. In de “OnEnable”-en “OnDisable”-methoden [21] [22] zetten we de specifieke “Action Map” aan of uit. “OnEnable” wordt opgeroepen in de hiërarchie van methoden na de “Awake”-methode, maar wel elke keer wanneer het gameobject waar ons script aan hangt op actief wordt gezet. “OnDisable” wordt als een van de laatste methoden opgeroepen wanneer het gameobject wordt gedeactiveerd. In de “Update”-methode [23] wordt geluisterd naar wanneer de knop wordt ingedrukt, wanneer deze dus “performed” [24] , vervolgens wordt de bijhorende methode uitgevoerd. De “Update”-methode wordt elke frame van een game opgeroepen.

```
HeadsetControls headsetControls;

❸ Unity Message | 0 references
❹ private void Awake()
{
    headsetControls = new HeadsetControls();
}

❸ Unity Message | 0 references
❹ private void OnEnable()
{
    headsetControls.XRILeftHand.Enable();
}

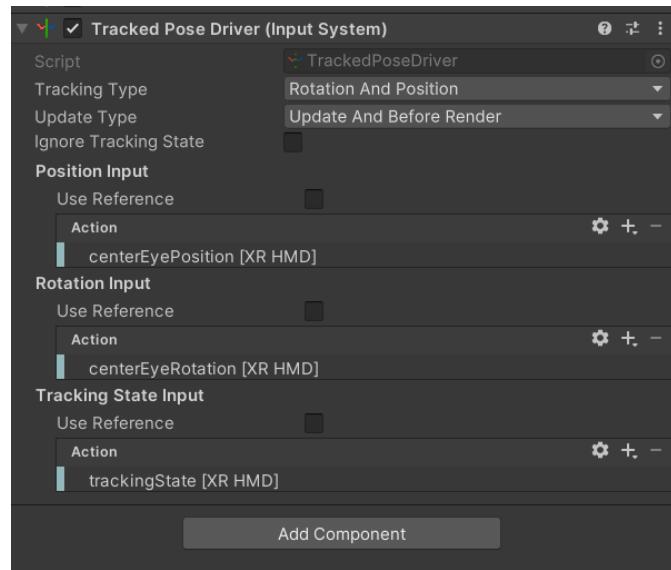
❸ Unity Message | 0 references
❹ private void OnDisable()
{
    headsetControls.XRILeftHand.Disable();
}

❸ Unity Message | 0 references
❹ void Update()
{
    headsetControls.XRILeftHand.Left_Secondary_Button.performed += CreateCircle;
}
```

Figuur 32: Voorbeeld activeren Input Action Map in code

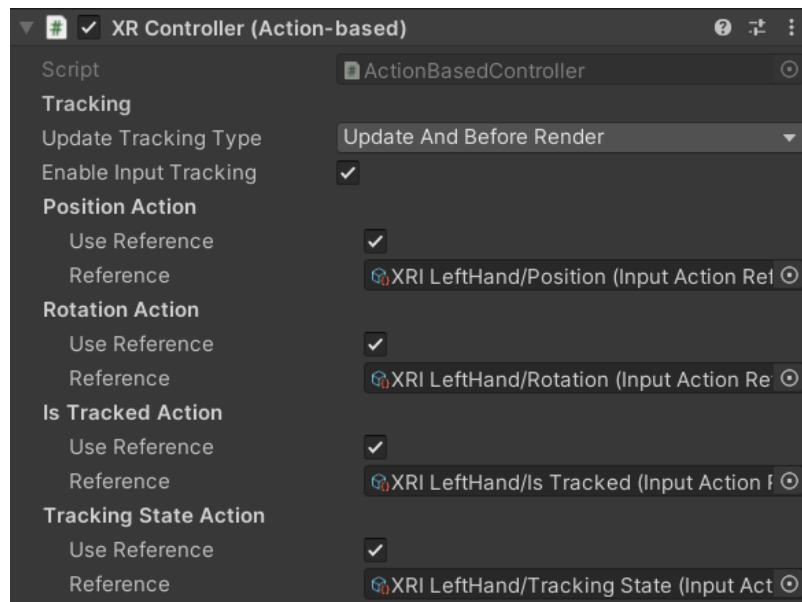
4.1.2. Beweging controllers

Nadat we de input konden lezen van controllers kunnen we ook de positie en rotatie van controllers lezen. De positie en rotatie van de headset geven we dan door aan de camera zodat deze meebeekt met de headset en de gebruiker kan zien wat er gebeurt. De camera is het object dat ons een beeld geeft van wat er gebeurt in een app. Door de positie en rotatie van de headset hier aan te hangen draait de camera ook mee met de gebruiker.



Figuur 33: Camera Component voor tracking headset

Vervolgens doen we dit ook voor de controllers. Zo kunnen de controllers die bij de headset zitten dan ook meebewegen en draaien wanneer de gebruiker deze beweegt of draait.



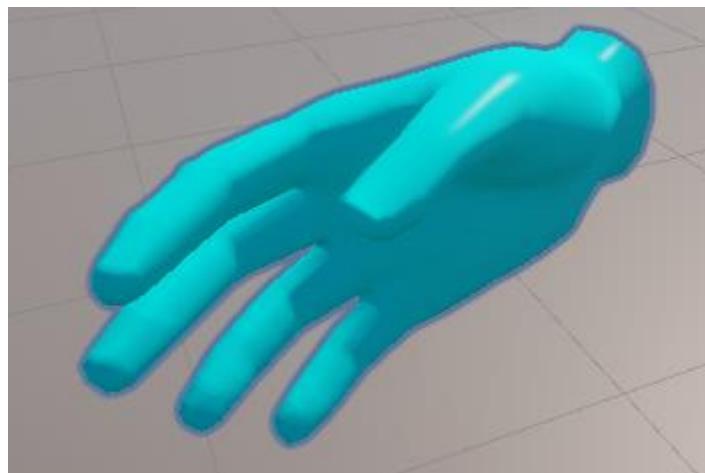
Figuur 34: Linkercontroller Component voor tracking

Om de gebruiker te laten zien waar de controllers zich bevinden in de app voegen we hier ook een model aan toe. Dit model heeft ook animaties die reageren op het indrukken van bepaalde knoppen. De modellen die zijn toegevoegd voor de linker en rechterhand zijn gemaakt en verzorgd door mijn collega. Ik heb deze wel uiteindelijk gelinkt aan de bewegingen van de controllers. Deze modellen worden in de hiërarchie toegevoegd onder de gameobjecten die de input van rotatie en positie volgen. Zo volgen deze modellen dezelfde bewegingen.



Figuur 35: Hiërarchie controllers

In bovenstaande afbeelding ziet u nog enkele andere objecten naast de modellen voor de handen. Het object "Hand_Menu" zal verder worden uitgelegd onder de titel "UI management". De "Attach_Point" wordt uitgelegd onder "Vastgrijpen objecten" en de interactors onder de eerstvolgende titel "Interactie".



Figuur 36: Right Hand Model

4.1.3. Interactie

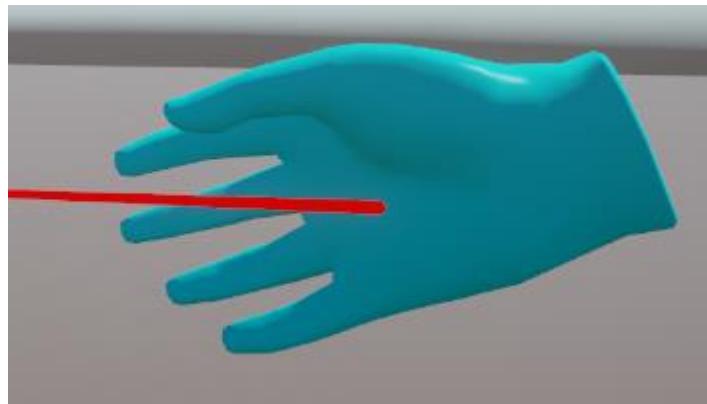
Het moet voor de gebruiker mogelijk zijn om te interageren met de omgeving. Dit wil zeggen dat de gebruiker op knoppen kan drukken om zichzelf te verplaatsen naar andere omgevingen en activiteiten te starten. Hiervoor zijn de benodigde scripts en tools al toegevoegd door de "XR Interaction Toolkit". Zodat de gebruiker niet te veel verwarring zou ervaren, hebben we de mogelijkheden voor interactie onder de rechterhand geplaatst.



Figuur 37: RightHand in Hiërarchie

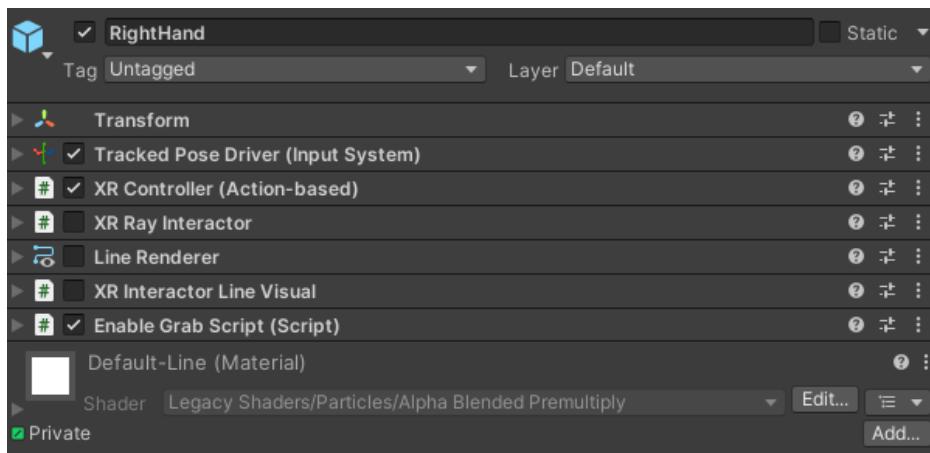
4.1.3.1. Op afstand

Aangezien onze VR-app het best vanuit de zittende positie wordt gespeeld, besloten we om gebruik te maken van de "Ray Interactor". Dit is een lijn die wordt getrokken vanuit de controller. Als deze lijn hovert over een object waarmee interactie mogelijk is, kan er op een knop worden gedrukt om de knop in te drukken.



Figuur 38: Rechterhand met Ray Interactor

Vanuit de "XR Interaction Toolkit" en Unity zelf zijn er 2 scripts en een component die we gebruiken: "XR Ray Interactor" [25] , "Line Renderer" [26] en "XR Interactor Line Visual" [27] . Deze componenten zorgen ervoor dat de lijn verschijnt en kan opmerken wanneer er interactie mogelijk is met een knop of object. Deze zitten direct in het "RightHand"-gameobject.



Figuur 39: RightHand componenten

Zoals te zien in figuur 39 zijn de checkboxen van de nodige componenten afgevinkt, dit betekent dat ze niet by default geenabled zijn als de game begint. Zo is er niet altijd een laser die vanuit de hand verschijnt, een keuze die gemaakt is om de ervaring "natuurlijker" te doen voelen. Uiteraard betekent dit dat we de lijn ook moeten kunnen aanzetten, hiervoor maken we een script dat deze lijn kan activeren en deactiveren: "EnableGrabScript".



Figuur 40: EnableGrabScript in de Unity Editor

De variabelen in dit script zijn er slechts twee. De eerste is een variabele die slechts in dit script mag gebruikt worden, daarom wordt deze ook op “private” gezet. Omdat we de optie willen openlaten dat deze vanuit de Unity editor kan gebruikt worden gebruiken we het [SerializeField]-attribuut [28] . Hierdoor kan men in deze variabele in de Unity Editor bijvoorbeeld een “GameObject” slepen waar de code gebruik van mag maken.

De “headsetControls”-variabele is wat we gebruiken om te luisteren naar de input. Deze moet niet vanuit de Unity Editor gebruikt kunnen worden dus plaatsen we hier geen access-modifier bij. Deze wordt dan default naar “private” gezet.

```
[SerializeField]
private GameObject grabHand;

HeadsetControls headsetControls;
```

Figuur 41: variabelen EnableGrabScript

De setup voor de HeadsetControls is eerder al uitgelegd, onder het titeltje “Input Management”. Voor de “grabHand” variabele vullen we die hier in als dat niet is gedaan in de Unity Editor. Als de waarde leeg is, vullen we deze in met het huidige gameobject waar het script aan vast hangt. Als we het script vasthangen aan de hand met de “Ray Interactor”-scripts en componenten zal deze hand worden gebruikt. Doordat we de “grabHand”-variabele ook in de Unity Editor beschikbaar hebben gemaakt zou dit script ook aan een ander gameobject gehangen kunnen worden. Dan moet de hand met de nodige scripts en componenten enkele in hierin worden gesleept in de Unity Editor. Het belangrijkste voor dit script is dat het gameobject waar het aanhangt altijd actief is, anders zullen de acties in dit script niet uitgevoerd kunnen worden.

```

➊ Unity Message | 0 references
private void Awake()
{
    headsetControls = new HeadsetControls();

    if (grabHand == null)
    {
        grabHand = gameObject;
    }
}

```

Figuur 42: Invullen van default waarden

De check voor het uitvoeren van de gewenste methode staat in de “Update”-methode. Wanneer de overeenkomende knop wordt ingedrukt, zal deze methode worden uitgevoerd.

```

➊ Unity Message | 0 references
void Update()
{
    headsetControls.XRIRightHand.Right_Primary_Button.performed += EnableGrabbing;
}

```

Figuur 43: Update-methode EnableGrabScript

Als parameter maken we gebruik van de “InputAction.CallbackContext”-struct [29] . Deze geeft de informatie van een input terug, deze wordt niet altijd gebruikt, maar zonder zal de code ook niet werken.

We kijken vervolgens eerst naar of de “XRRayInteractor”-component actief en enabled is. Deze true-of false-waarde slaan we op in een extra variabele “newValue” om het gemakkelijker te gebruiken. Vervolgens gebruiken we het uitropteken om de huidige waarde van de onderdelen te veranderen van enabled naar disabled en omgekeerd. Zo zal de lijn verschijnen en verdwijnen als we dat wensen met slechts de druk op een knop.

```

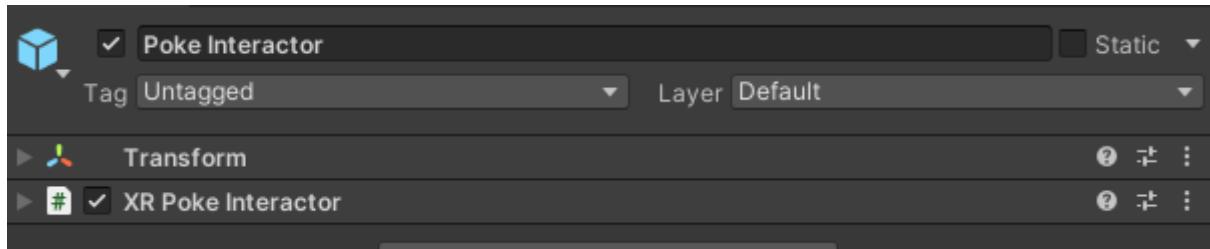
1 reference
private void EnableGrabbing(InputAction.CallbackContext context)
{
    bool newValue = grabHand.GetComponent<XRRayInteractor>().isActiveAndEnabled;
    grabHand.GetComponent<XRRayInteractor>().enabled = !newValue;
    grabHand.GetComponent<LineRenderer>().enabled = !newValue;
    grabHand.GetComponent<XRInteractorLineVisual>().enabled = !newValue;
}

```

Figuur 44: EnableGrabbing-methode

4.1.3.2. Dichtbij

Zoals later te zien, bij het titeltje “UI management”, voorzien we ook een klein menu op onze linkerhand. Omdat het dan onnodig is om een “Ray Interactor” te gebruiken hebben we ook de optie toegevoegd om met de rechterwijsvinger te tikken op knoppen. Hiervoor gebruiken we de “XR Poke Interactor” [30] , dit is een gameobject dat we in de hiërarchie onder het “Righthand”-gameobject zetten. Dit is een apart gameobject omdat een gameobject slechts één vorm van interactor kan bezitten. Door gebruik te maken van child-gameobjects, lossen we dit op.



Figuur 45: Poke Interactor componenten

Om ervoor te zorgen dat het specifiek de wijsvinger is die kan tikken hebben we niet heel veel nodig. We slepen slechts het object dat de wijsvinger is in het “XR Poke Interactor”-script.

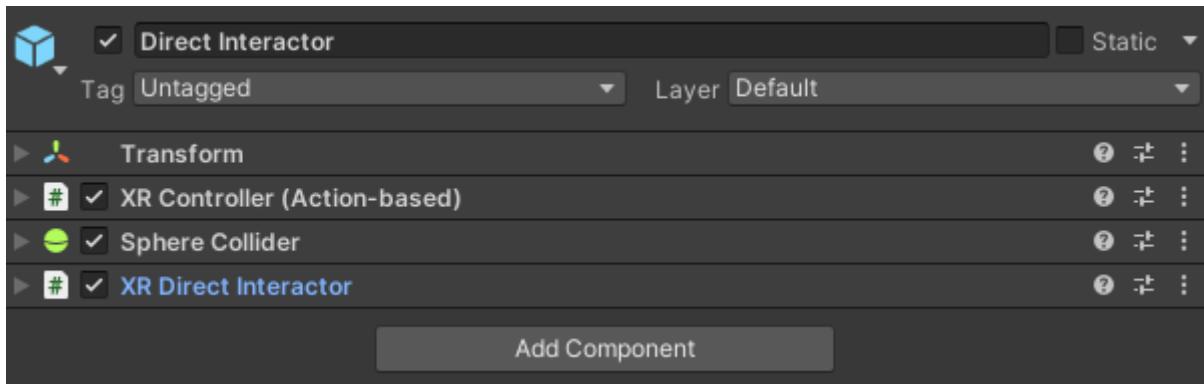


Figuur 46: Juiste vinger In XR Poke Interactor-script

4.1.4. Vastgrijpen objecten

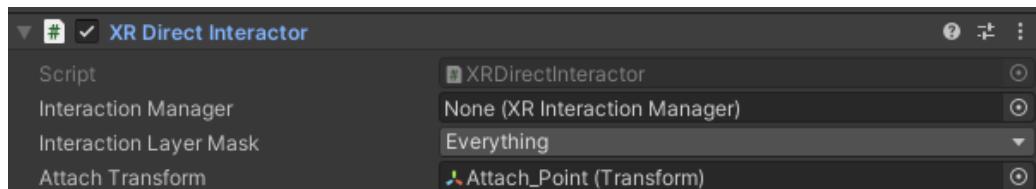
Dit is een functionaliteit die we hebben toegevoegd omdat de optie er was en we later mogelijks een manier zouden vinden om dit te gebruiken. Later merkten we wel dat door de vele zaken die een gebruiker al kan deze functionaliteit niet echt nodig was. Deze heeft op dit moment ook geen toepassing in de daadwerkelijke app, maar ze zit er in, mocht dit alsnog in de toekomst gebruikt worden.

Zoals we al eerder hadden gezien zijn er nog twee gameobjecten die zich onder de “RightHand” bevinden. Het “Attach_Point” en de “Direct Interactor” [31] zijn nodig om objecten vast te grijpen. Daarnaast is dit ook mogelijk met de “Ray Interactor”. De “Ray Interactor” en de “Direct Interactor” maken het mogelijk om op afstand en dichtbij een object vast te pakken.



Figuur 47: Direct Interactor componenten

Het "Attach_Point" is waar het vastgegrepen object zich zal bevinden ten op zichte van de hand, wanneer de vastgrijpactie is uitgevoerd. Dit heeft dan ook geen scripts nodig en hangt aan de hand vast. In de "Direct Interactor" en "Ray Interactor" zijn er variabelen om dit gameobject in te slepen. Als deze variabele leeg wordt gelaten, zullen de coördinaten van de "RightHand" gebruikt worden om het vastgegrepen object naar te "snappen".



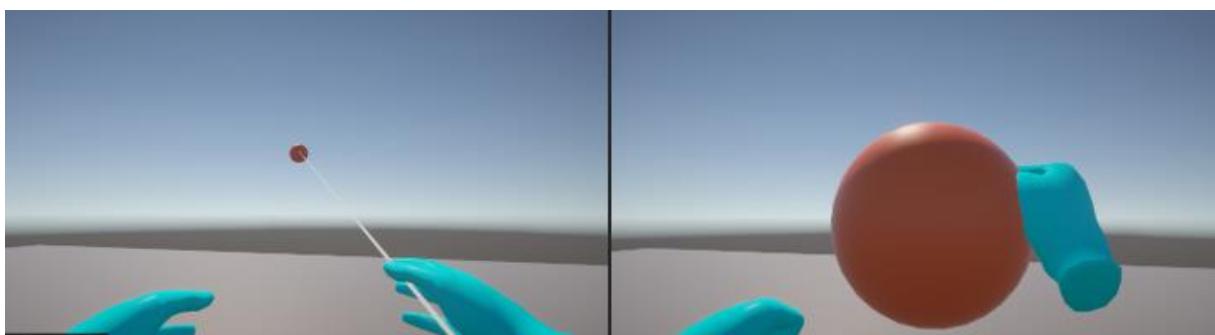
Figuur 48: Attach_Point in XR Direct Interactor-script

Als we willen dat een object kan vastgegrepen worden, moet deze het "XR Grab Interactable"-script [32] hebben. Dit script reageert dan op de actie van de "Ray Interactor" of de "Direct Interactor".



Figuur 49: XR Grab Interactable-script

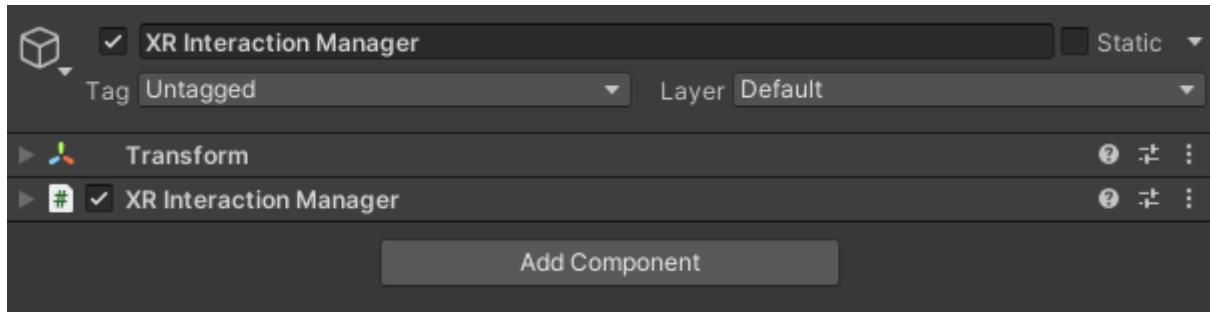
Als we dit allemaal hebben toegevoegd, kunnen we vervolgens een object selecteren en vastgrijpen met onze rechterhand.



Figuur 50: Vastgrijpen van een object

4.1.5. XR Interaction Manager

Omdat sommige van de scripts die we gebruiken gebruikmaken van input die geregeld wordt door "Input Action Maps" die we niet zelf hebben gemaakt hebben, is er nog een extra gameobject nodig in onze hiërarchie. Dit object is de "XR Interaction Manager". Zonder dit object zal het bijvoorbeeld niet mogelijk zijn om op te merken wanneer er op een knop gedrukt wordt. Dit object zit standaard in de pakketten van de "XR Interaction Toolkit" die we hebben geïnstalleerd en kan gewoon in de scene worden gesleept.



Figuur 51: XR Interaction Manager

4.2. Recalibratie

Voor deze applicatie is het de bedoeling dat de gebruiker in een draaistoel zal zitten. Zo zal deze rond kunnen kijken in de applicatie zonder zich zorgen te maken over zijn richting in de echte wereld. Echter wanneer dit niet mogelijk is of de gebruiker zou zich naar één punt willen draaien, dienen we hiervoor een oplossing te voorzien. Daarnaast is het ook mogelijk dat de gebruiker zijn hoofd zou kunnen verplaatsen op een manier die zijn positie in de applicatie kan aantasten. Deze "aantasting" heeft te maken met de structuur van de gameobjecten die de headset representeren en hoe verplaatsing in een 3D-omgeving exact werkt. Hiervoor is ook een oplossing nodig. Beide oplossingen komen er in de vorm van recalibratie.

4.2.1. XrRig

Voordat we verdergaan met onze code-oplossing is het handig om te begrijpen hoe de camera geplaatst is in de wereld en hoe de camera en handen bij elkaar worden gehouden.

Wanneer een VR-app wordt gemaakt, wordt aangeraden om gebruik te maken van de volgorde "XrRig">>"Offset">>"Camera". Hierbij is de "XrRig" het parent-object van de "Offset" en de "Offset" deze van de "Camera". Voor deze sectie verstaan we onder "Camera" niet alleen de headset die kijkt door het oog van de camera, maar ook de handen die hieronder hangen. In ons geval is hier nog een "Narrator"-object, deze wordt verder uitgelegd onder "Audio Management". Door deze structuur van gameobjecten kunnen we alles bij elkaar houden wanneer we ons moeten verplaatsen, door te teleporteren of een activiteit te starten op een andere locatie.

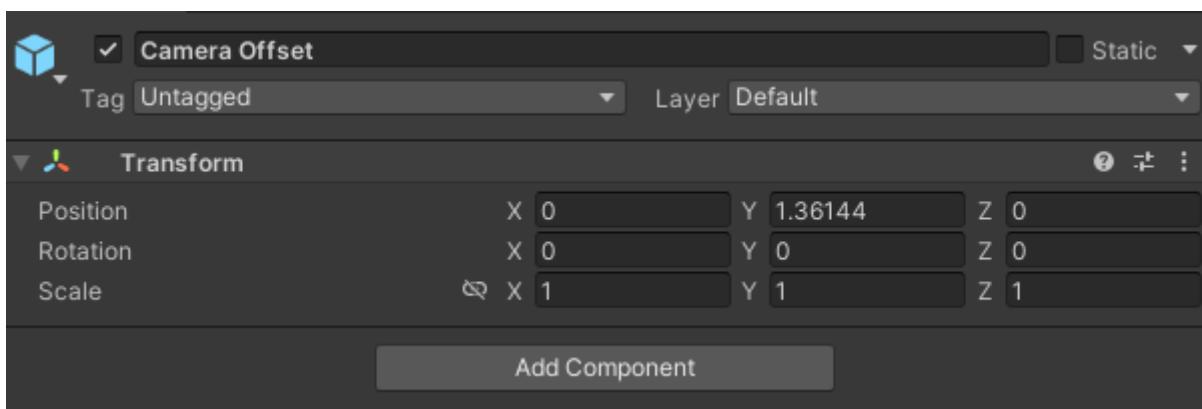


Figuur 52: XrRig in Unity Editor

Wanneer we ons verplaatsen door te teleporteren ,of een andere grote locatie-veranderende activiteit, verplaatsen we de "XrRig". De "Offset", of "Camera Offset" zoals door sommigen wordt verkozen, met alle children-objecten verplaatst zich dan ook mee. De "Offset" is een leeg gameobject waar over het algemeen niets aan wordt verandert. Het enige component dat hier aan hangt is "Transform".

Het Transform-component hangt aan elk gameobject en bepaalt de positie in de 3D-omgeving, maar ook de rotatie en de schaal. Voor de positie komt de waarde 1 overeen met 1 meter in de werkelijkheid. Voor rotatie is de waarde 1 een draaiing van 1 graad. De schaal is relatief voor de grootte van het object waarbij 1 overeenkomt met 100% van de grootte van het object. De schaal is een waarde waar best niet te veel mee gedaan wordt, hier moet eigenlijk rekening mee gehouden worden door mensen die de visuele assets maken.

Bij de "Offset" passen we de Y-waarde aan naar wat we verwachten dat de grootte is van de ogen vanaf de grond. Dit zal dan ook de hoogte zijn waarop de camera van de grond hangt. Tijdens het spelen van de game kan de camera boven en onder deze hoogte komen, afhankelijk van hoe hoog en laag de gebruiker gaat met zijn hoofd. De default voor de y-waarde van een "Offset" is 1.36144, tijdens het testen hadden we ook gemerkt dat er geen probleem is met deze hoogte en hebben we dit zo gelaten.



Figuur 53: Camera Offset in Inspector

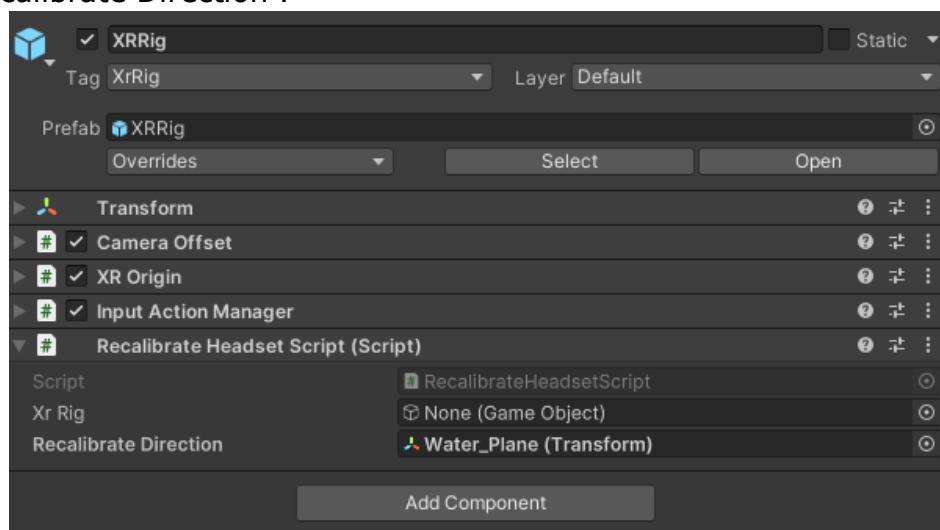
Van nature worden er geen veranderingen gedaan aan de "Offset". Maar na wat werken heb ik besloten deze toch te gebruiken om een probleem met de locatie die niet overeenkomt met de locatie van de "XrRig" te verhelpen. Dit leidde ertoe om een

“RecalibrateScript” te maken. Deze zet de positie van de gebruiker aan het begin van een scene op de originele positie, ongeacht van hoe de gebruiker gedraaid staat in de echte wereld. Wanneer de gebruiker dan een scene begint komt zijn “vooruit” overeen met de “vooruit” in de VR-app.

4.2.2. Unity Editor

We plaatsen ons script in het “XrRig”-gameobject, voornamelijk omdat we voor de rotatie gebruikmaken van de “XrRig”. Als dit niet het geval was, was deze aan de “Camera Offset” gehangen. De eerste variabele is de “XrRig” zelf, voor het geval dit script de “XrRig” zou moeten aanpassen, maar niet in de “XrRig” zou geplaatst zijn. Daarnaast is er nog een variabele “Recalibrate Direction”, deze dient om de draaiing correct te hebben.

Om de werking van het script kort uit te leggen: de positie van de “Offset” wordt aangepast met de positie van de “XrRig” en de rotatie van de “XrRig” met de rotatie van het “Recalibrate Direction”.



Figuur 54: XrRig in Inspector met RecalibrateScript

4.2.3. Code

We hebben alle variabelen al uitgelegd in de vorige sectie. Voor de “Awake”-methode is er wel wat extra uitleg nodig. We vullen hier de “XrRig” met het eerste gameobject dat we kunnen vinden in de huidige scene. Deze keer maken we gebruik van een verkorte manier voor het schrijven van een if-statement.

We maken hier gebruik van een “tag”, dit is een soort van herkenningsvariabele om specifieke objecten te vinden. Een tag kan worden toegekend aan meerdere objecten en kan dan ook worden gebruikt om alle objecten met deze tag te vinden. In dit geval moet er slechts een gevonden worden. Online wordt er aangeraden om gebruik te maken van tags in plaats van de namen van gameobjecten omdat tags beter te controleren zijn. Als je kijkt naar de “XrRig” in de Inspector zie je onmiddellijk onder de naam de tag die hieraan is toegekend.

Nadat de “xrRig”-variabele is opgevuld roepen we de “Recalibrate”-methode die ons zal recalibreren aan het begin van een scene.

```
Unity Message | 0 references
private void Awake()
{
    if (xrRig == null) xrRig = GameObject.FindGameObjectWithTag("XrRig");
    Recalibrate();
}
```

Figuur 55: Awake-methode RecalibrateScript

De “Recalibrate”-methode doet twee grote dingen. Eerst wordt de “Offset” verplaatst zodat de camera perfect in het midden van de “XrRig” staat. Ten tweede draait de “XrRig” rond de as van de camera zodat de camera met zijn voorkant naar de richting van de “recalibrateDirection”-variabele draait.

```
1 reference
public void Recalibrate()
{
    Transform camera = Camera.main.transform;

    //set position using the Camera Offset-object

    GameObject cameraOffset = xrRig.transform.GetChild(0).gameObject;

    Vector3 rigPosition = xrRig.transform.position;

    Vector3 offset = rigPosition - camera.position;
    offset.y = 0;
    cameraOffset.transform.position += offset;

    //set rotation using the XrRig-gameobject

    //this has to be reset after the position otherwise the xrrig will rotate around the previous camera position
    camera = Camera.main.transform;

    Vector3 targetDirection = Vector3.forward;
    if (recalibrateDirection != null) targetDirection = recalibrateDirection.forward;
    targetDirection.y = 0;

    Vector3 cameraDirection = camera.forward;
    cameraDirection.y = 0;

    float angle = Vector3.SignedAngle(cameraDirection, targetDirection, Vector3.up);
    xrRig.transform.RotateAround(camera.position, Vector3.up, angle);
}
```

Figuur 56: Recalibrate-methode

Voordat we verder gaan hebben we de “transform” [33] nodig van de camera. Dit bevat de positie, rotatie en scale van de camera. Wanneer we deze aanpassen wordt deze ook aangepast voor de camera zelf. Dit zou niet mogen lukken aangezien de camera luistert naar de headset voor zijn positie en rotatie. Maar we kunnen wel de waarden lezen die hier in staan.

Vervolgens halen we de “Camera Offset” op. We weten dat dit het eerste, en enige, child-gameobject is van de “XrRig”, dus dan gebruiken we ook de “xrRig”-variabele. Om

het ons wat gemakkelijker te maken halen we ook de positie van de "xrRig"-variabele op in de nieuwe "rigPosition"-variabele.

Om te berekenen wat de afstand is tussen de positie van de "XrRig" en de camera, trekken we de "rigPosition" af van de camera-positie. Dit geeft ons een Vector3-waarde [34] . De waarde van de positie is ook een Vector3 waarde, maar deze "offset"-waarde die we juist hebben gemaakt is nu een afstand tussen deze twee posities.

Bij deze "offset"-waarde zetten we de y-waarde naar 0. Zo kunnen we bij de volgende berekening zeker zijn dat de nieuwe verplaatsing niet ook de y-waarde van de "Camera Offset" aantast.

Op de laatste lijn van dit stuk wordt de positie van de "Camera Offset" verandert met de nieuwe vector die we hebben gemaakt. Zodoende staat de camera perfect op de positie van de "XrRig". Dit segment moet ook worden gedaan voor het draaien van de "XrRig" omdat er anders een probleem voorkomt.

```
Transform camera = Camera.main.transform;  
  
//set position using the Camera Offset-object  
  
GameObject cameraOffset = xrRig.transform.GetChild(0).gameObject;  
  
Vector3 rigPosition = xrRig.transform.position;  
  
Vector3 offset = rigPosition - camera.position;  
offset.y = 0;  
cameraOffset.transform.position += offset;
```

Figuur 57: Verplaatsing

Om te beginnen moeten we de nieuwe positie en rotatie van de camera hebben dus vullen we deze nog eens in.

Vervolgens vinden we de "targetDirection", dit is de richting waarnaar uiteindelijk gedraaid moet worden. We maken hier gebruik van een statische waarde van Vector3: de forward-waarde. Deze kan voor elke Vector3-waarde worden opgehaald en is in essentie de voorkant waar een object naartoe kijkt. Als we gewoon Vector3 gebruiken voor deze methode maakt het gebruik van de richting van onze 3D-omgeving, de worldspace [35] . nadat we dit hebben ingevuld kijken we of de "recalibrateDirection"-waarde was ingevuld in de Unity Editor of niet. Als dit zo is wordt de "targetDirection" die van "recalibrateDirection", anders blijft het de forward van de worldspace.

We zetten de y-waarde weer op 0, zodat de draaiing alleen maar gebeurt over de x- en z-as. Dit is om te voorkomen dat de gebruiker zijn hoofd omhoog zou moeten draaien na een calibratie omdat zijn hoofd naar de grond gericht stond.

Voordat we de daadwerkelijke berekening gaan doen hebben we nog nood aan de voorwaartse richting van de camera. Voor de berekening gaan we dus de hoek tussen deze twee richtingen berekenen.

Met de Vector3 “SignedAngle”-methode [36] berekenen we dan de hoek tussen deze twee voorwaartse richtingen. Hierbij gebruiken we de “up” van de worldspace om een correcte hoek te verkrijgen.

En als laatste draaien we de “XrRig” rond de positie van de camera over de hoek die we juist hebben berekend. Hiervoor gebruiken we de “RotateAround”-methode [37].

```
//set rotation using the XrRig-gameobject  
  
//this has to be reset after the position otherwise the xrriig will rotate around the previous camera position  
camera = Camera.main.transform;  
  
Vector3 targetDirection = Vector3.forward;  
if (recalibrateDirection != null) targetDirection = recalibrateDirection.forward;  
targetDirection.y = 0;  
  
Vector3 cameraDirection = camera.forward;  
cameraDirection.y = 0;  
  
float angle = Vector3.SignedAngle(cameraDirection, targetDirection, Vector3.up);  
xrRig.transform.RotateAround(camera.position, Vector3.up, angle);
```

Figuur 58: Draaiing

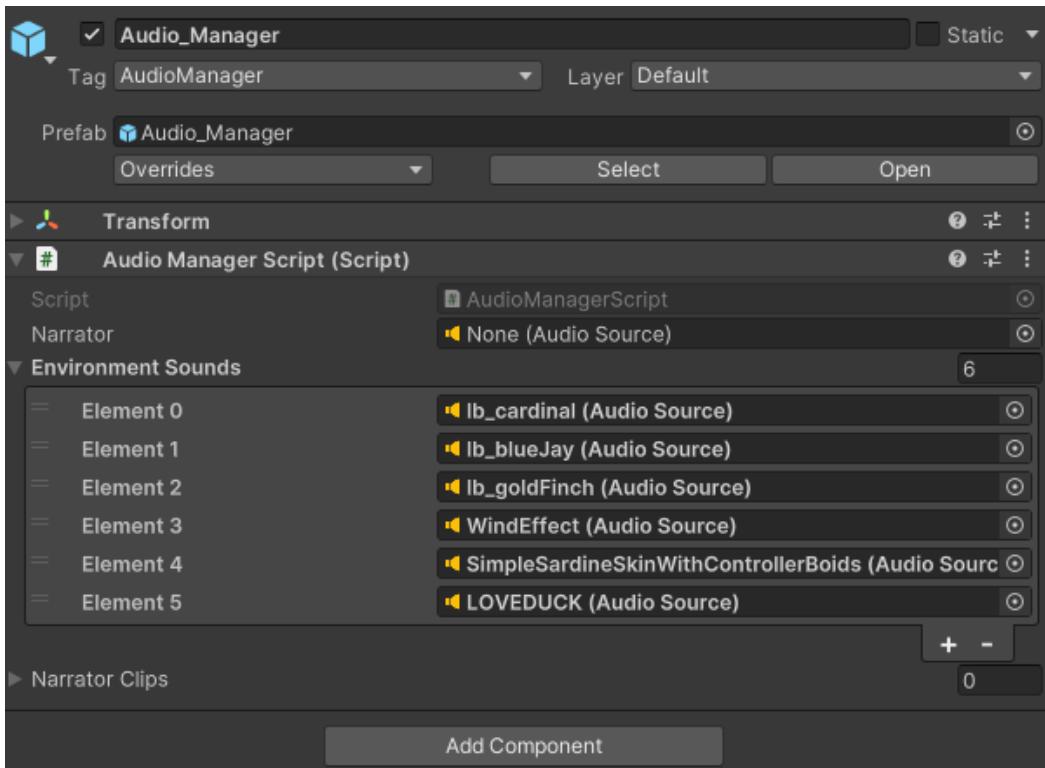
4.3. Audio management

Binnen de applicatie hadden we ook geluid nodig. Dit zijn geluiden van de omgevingen, maar ook de begeleiding die wordt gegeven bij de activiteiten die we aanbieden.

We hadden per omgeving controle nodig over al deze soundeffecten en ingesproken boodschappen. Daarnaast waren er ook methodes nodig om het geluid te doen veranderen of af te zetten. Om deze methoden voor het geluid niet te moeten kopiëren tussen verschillende scripts maakten we één script dat dit allemaal kon managen. Dit script hebben we dan aan een gameobject gehangen dat we de “Audio_Manager” hebben genoemd.

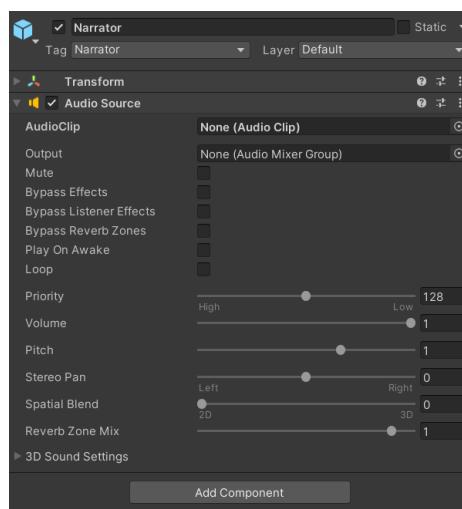
4.3.1. Unity Editor

De “Audio_Manager” heeft slechts een script, het “AudioManagerScript”. Deze heeft slechts 3 variabelen die echt ingevuld kunnen worden. De eerste is een “ AudioSource” [38] met als naam “Narrator”, verdere uitleg hieronder. Een array voor “Environment Sounds”, waar de geluiden uit de omgeving in zitten. Deze kunnen dan gelijktijdig worden gedimd wanneer een activiteit begint die ander geluid voorziet. De “Narrator Clips” is een op het moment ongebruikte array, waar wel code voor is om hem te gebruiken. Dit wordt duidelijker bij de code.



Figuur 59: Audio_Manager met AudioManagerScript

De "Narrator" is een object dat we voorheen onder de "XrRig" hebben geplaatst, zo is de "Narrator" altijd in de scene als de "XrRig" in de scene staat. Deze is leeg en kan worden ingevuld door de "Audio_Manager". Het belangrijkste hierbij is dat de "Spatial Blend" op 2D staat. Dit betekent dat het geluid zich zal afspelen naast de oren van de gebruiker. Wanneer dit op 3D staat zou het geluid van de locatie komen waar het gameobject met de audiosource zich bevindt. Dit creëert dan een effect van afstand. Voor de "Narrator" is dit effect niet nodig en is het gewoon de bedoeling dat de gebruiker de ingesproken boodschappen kan horen.



Figuur 60: Narrator in Unity Editor

4.3.2. Code

Voor het “AudioManagerScript” zijn er buiten de variabelen die we zien in de Unity Editor geen verdere globale variabelen. In de “Awake”-methode vullen we ook de “narrator”-variabele in als dit nog niet gedaan is. We maken hier ook weer gebruik van de werkwijze met tags. Deze keer wordt de code wel wat langer. Wanneer we een “GameObject.Find”-methode gebruiken krijgen we een gameobject terug, maar hier moet het een “ AudioSource” zijn. Daarom pakken we van dit gameobject de component “ AudioSource” met de “ GetComponent”-methode [39] .

```
Unity Message | 0 references
void Awake()
{
    if (narrator == null)
    {
        narrator = GameObject.FindGameObjectWithTag("Narrator").GetComponent<
```

Figuur 61: Awake-methode AudioManagerScript

Verder ga ik de uitleg van de “Audio_Manager” opdelen in kleinere onderdelen. Deze onderdelen beschrijven verschillende functionaliteiten van het “AudioManagerScript” en alle methoden die hierbij horen.

4.3.2.1. Gebruik de verteller

We hebben een algemene methode voor het gebruiken van de verteller. Bij deze eerste versie kunnen we de naam van een “ AudioClip” [40] meegeven. Een “ AudioClip” is een mp3-bestand dat afgespeeld kan worden door het in een “ AudioSource” te zetten en dan af te spelen. Deze methode zoekt de “ AudioClip” in de “ narratorClips”-array als deze gevuld is. Wanneer de clip niet gevonden kan worden wordt deze ook niet afgespeeld. Deze methode kan tijdens het developen gebruikt worden om gemakkelijk een clip op te roepen door deze uit de array te halen. Zoals te zien is, wordt hier ook een methode gebruikt met dezelfde naamgeving, maar deze keer een andere parameter-klasse: “ AudioClip”. Dit is een voorbeeld van overloading.

```
1 reference
public void useNarrator(string clipName = "Mindfulness_Bos_man01_boosted", bool waitTillFinished = false)
{
    AudioClip audioclip = Array.Find(narratorClips, clip => clip.name == clipName);
    if (audioclip != null)
    {
        useNarrator(audioclip, waitTillFinished);
    }
    else
    {
        Debug.Log("Could not find this audiofile");
    }
}
```

Figuur 62: UseNarrator-methode met string parameter

Deze methode accepteert een “ AudioClip” en een “ bool” die bevestigt of we wachten tot de “ Narrator” klaar is met zijn eerste audioclip. Als “ waitTillFinished” true is zal er een “ Coroutine” [41] worden gestart. Deze “ Coroutine” komt later terug aan bod. Als de

bool niet true is, zal de meegegeven "audioclip"-parameter beginnen te spelen vanuit het "Narrator"-gameobject, als deze nog geen "Audioclip" aan het spelen is.

```
3 references
public void useNarrator(AudioClip audioclip, bool waitTillFinished = false)
{
    if (waitTillFinished)
    {
        StartCoroutine(useNarratorRoutine(audioclip));
    }
    else
    {
        if (narrator.isPlaying == false)
        {
            narrator.clip = audioclip;
            narrator.Play();
        }
    }
}
```

Figuur 63: UseNarrator-methode met AudioClip parameter

Dit is een "Coroutine", een coroutine is een methode die in een script blijft werken tot hij wordt gestopt of het script wordt gestopt. Deze loopt op de achtergrond terwijl het script normaal werkt. In deze gebruiken we een while-loop die blijft lopen terwijl de "Narrator" een "Audioclip" aan het spelen is. Door "yield return null" te gebruiken stopt de "Coroutine" tot de volgende frame begint. Wanneer de volgende frame begint gaan we weer verder met de code. Als de "Audioclip" van de "Narrator" klaar is met afspelen of de "Audioclip" wordt gestopt, zal de loop stoppen en wordt de "UseNarrator"-methode gebruikt. Hierbij is de "waitTillfinished" bool niet meegegeven, zoals we eerder hebben gezien zal deze automatisch false zijn en dus de "Audioclip" starten in de "Narrator".

```
1 reference
IEnumerator useNarratorRoutine(AudioClip audioclip)
{
    while (narrator.isPlaying)
    {
        //skips this frame and tries again on the next frame untill narrator stops playing
        yield return null;
    }

    useNarrator(audioclip);
}
```

Figuur 64: Coroutine UseNarratoroutine

4.3.2.2. Volume veranderen

Dit is een "Coroutine" die het volume van een " AudioSource" verandert over een gegeven periode van tijd naar een gegeven volume. Daarnaast zijn er ook nog twee "bools" die voor specifieke aanpassingen zorgen die niet altijd nodig zijn. Door deze twee nog toe te voegen kan deze methode voor verschillende doeleinden gebruikt worden.

Als eerst houden we het originele volume bij, zo zou het volume later terug naar dit volume verandert worden moest dit nodig zijn. Vervolgens hebben we een “timer”-variabele nodig, deze begint bij 0. Omdat we de mogelijkheid voor een tijdsperiode te bepalen geven moeten we ook een tijdsinterval tussen de veranderingen meegeven. Als de meegegeven “seconds” is zetten we de “timeInterval” naar 1. Dit is zodat we later bij het updaten van de “timer” meteen boven de meegegeven tijd komen, ook voorkomt dit een mogelijke error waarbij 0 wordt gedeeld door 100. De reden dat we 100 hebben gekozen is voornamelijk om de verandering van het volume zo geleidelijk mogelijk te laten verlopen.

Bij de loop zien we dat deze loopt terwijl de timer kleiner of gelijk is aan de meegegeven tijd. Als eerste bepalen we het nieuwe volume. Als de “seconds” gelijk is aan 0 moet er geen berekening worden gedaan en zetten we “newVolume” op het eindresultaat dat we zoeken te bereiken. Anders doen we een kleine berekening waarbij “newVolume” het originele volume wordt met daarbij de verandering die nodig is. Als het originele volume groter is dan het eindvolume zal de verandering negatief zijn. Als het originele volume lager is zal de verandering positief zijn.

De verandering wordt berekend door het eindvolume af te trekken van het originele volume. Op die manier hebben we 100% van de verandering die moet gemaakt worden. Dan delen we de “timer” door het totale hoeveelheid seconde die voorbij moeten gaan om te weten op welk percentage in de te verstrijken tijd we zitten. Doordat we de “timer” updaten zal het percentage elke iteratie van de loop omhoog gaan.

Daarna hebben we een if-else-statement met daarbinnen nog een else if-statement om na te kijken of het volume niet onder 0 of boven 1 gaat. Dit zijn de minimale en maximale waarde die dit volume kan bereiken. Is het een van beide opties wordt het volume van de “ AudioSource ” de waarde van “ newVolume ”.

Vervolgens updaten we de “timer” met het “timeInterval” dat we eerder hebben berekend. Dan pauzeren we de loop binnen de “Coroutine” voor de berekende “timeInterval”. Zo klopt onze “timer” ook daadwerkelijk. Na deze tijd beginnen we terug vanboven aan de loop.

Na de loop doen we nog twee checks voor de “bools”. Als “stopPlaying” true is en de “ AudioSource ” heeft een “ AudioClip ” die nog aan het spelen is, wordt deze gestopt. Met de “resetAfterwards” bevestigen we dat het volume na het veranderen onmiddellijk terug wordt gezet naar het originele volume. Het gebruik van deze twee variabelen wordt duidelijker in het volgende segment.

```

3 references
IEnumerator ChangeVolumeToRoutine(float endVolume, float seconds, AudioSource source, bool resetAfterwards, bool stopPlaying = false)
{
    float originalAudioVolume = source.volume;
    float timer = 0;
    float timeInterval = seconds <= 0 ? 1 : seconds / 100;
    while (timer <= seconds)
    {
        float newVolume = seconds == 0 ? endVolume : originalAudioVolume + ((endVolume - originalAudioVolume) * (timer / seconds));

        if (newVolume < 0)
        {
            source.volume = 0;
        }
        else if (newVolume > 1)
        {
            source.volume = 1;
        }
        else
        {
            source.volume = newVolume;
        }

        timer += timeInterval;
        yield return new WaitForSeconds(timeInterval);
    }

    if (stopPlaying && source.isPlaying && source.clip != null) source.Stop();

    //reset volume afterwards
    if (resetAfterwards) source.volume = originalAudioVolume;
}

```

Figuur 65: Coroutine ChangeVolumeToRoutine

4.3.2.3. Specifiek Volume veranderen

Binnen het “AudioManagerScript” zijn er ook drie methodes voor het gebruiken van de “ChangeVolumeRoutine”-methode. De eerste hiervan is voor het veranderen van het volume van de “narrator” die we aan het begin van dit script hebben ingevuld. De methode vraagt de meeste parameters van de “Coroutine” om door te geven aan deze “Coroutine”. Alleen vraagt deze niet om een “ AudioSource” omdat we daarvoor de globale variabele “narrator” voor kunnen gebruiken. We maken er ook de gewoonte van om “stopPlaying” niet altijd te gebruiken. Sommige “AudioSources” staan namelijk op loop, wat betekent dat ze opnieuw spelen wanneer ze klaar zijn. Wanneer ze dan gestopt worden kunnen ze niet gemakkelijk opnieuw worden opgestart met de juiste timing. Om er zeker van te zijn dat iemand die deze methode gebruikt, nadenkt over wat ze willen dat de methode doet.

```

//this method changes the sound's volume of the narrator
3 references
public void ChangeVolumeNarrator(float endVolume, float seconds, bool resetVolumeAfterwards, bool stopPlaying = false)
{
    StartCoroutine(ChangeVolumeToRoutine(endVolume, seconds, narrator, resetVolumeAfterwards, stopPlaying));
}

```

Figuur 66: ChangeVolumeNarrator-methode

We hebben alle geluiden van de omgeving opgeslagen in een array. Deze kunnen we ook aanpassen als we dat willen met een methode. Voordat een activiteit begint zetten we alle geluiden op 0 en na de activiteit zetten we deze terug op 1. Dit is omdat ze de geluiden die we gebruiken in een activiteit kunnen versturen.

Hier geven we enkel het eindvolume en de te verstrijken tijd mee. Vervolgens hebben we een loop die alle “AudioSources” in de “environmentSounds” naar het gewenste volume zet. We zetten de “resetAfterwards” van de “Coroutine” op false omdat deze geluiden niet moeten gereset worden achteraf.

```
//this method changes all sounds in the environmentsound-array
3 references
public void changeVolumeEnvironment(float endVolume, float seconds)
{
    foreach ( AudioSource a in environmentSounds)
    {
        StartCoroutine(changeVolumeToRoutine(endVolume, seconds, a, false));
    }
}
```

Figuur 67: ChangeVolumeEnvironment-methode

We hadden ook de mogelijkheid toegevoegd zelf een “ AudioSource” mee te geven in een methode. Zo kan de gebruiker gebruikmaken van een andere “ AudioSource” die niet vooraf is vastgesteld. Deze methode wordt dan ook vaak binnen de activiteiten gebruikt.

```
2 references
public void changeVolume AudioSource(float endVolume, float seconds, AudioSource source, bool resetVolumeAfterwards, bool stopPlaying = false)
{
    StartCoroutine(changeVolumeToRoutine(endVolume, seconds, source, resetVolumeAfterwards, stopPlaying));
}
```

Figuur 68: ChangeVolume AudioSource-methode

4.4. UI management

Binnen een VR-game kunnen we gebruikmaken van een grafische interface om bepaalde acties te laten gebeuren. Dit zijn acties die niet te makkelijk geactiveerd moeten kunnen worden. Omdat ze een vrij grote verandering veroorzaken of met bedachte rade moeten uitgevoerd worden. Ook kwam de keuze om sommige acties via een interface te doen uit het idee dat de gebruiker niet te veel knoppen moet onthouden om in te drukken.

Ook hebben we de mogelijkheden van Unity’s UI uitgebreid om het mogelijk te maken het scherm donker te laten worden: een fade-to-black mogelijkheid. Dit gebruiken we als we een verplaatsing doen van een scene naar een andere, maar ook vlak voor een activiteit start. Zo wordt de gebruiker niet altijd met een flits verplaatst. Dit was vanuit het idee dat de applicatie rustgevend moet zijn.

4.4.1. Camera canvas

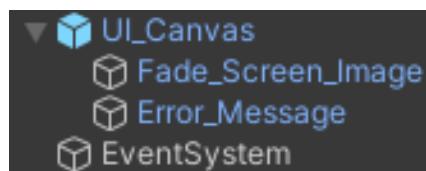
Naast alles wat er in de uiteindelijk applicatie is geplaatst, had ik aan het begin ook wat onderdelen gemaakt die later niet meer nodig waren. Deze zitten nog wel in de applicatie moesten ze later weer gebruikt worden.

4.4.1.1. Unity Editor

Het is mogelijk om een canvas [42] te maken in Unity. Hier kunnen allerlei UI-elementen binnen worden geplaatst, zoals: tekst, afbeeldingen, knoppen, sliders Voor deze sectie kijken we naar de tekst en het canvas zelf.

In het canvas dat wij hadden gebruikt hebben we een "Error_Message" geplaatst van het type "Text". Hiermee kan een tekst worden getoond in het canvas. Daarnaast is er ook een "Fade_Screen_Image" van het type "Image". Hier komt meer uitleg over bij het titeltje "Fade".

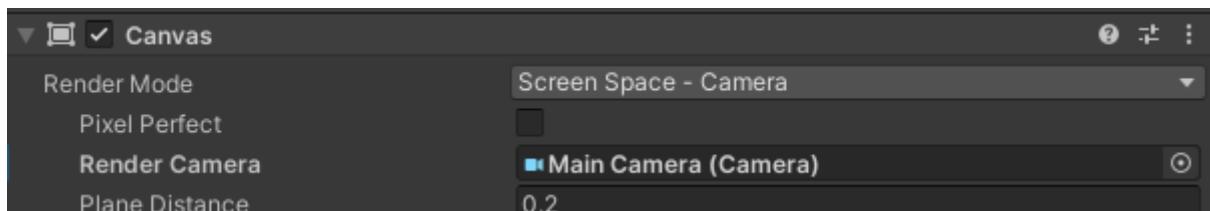
In deze afbeelding zit nog een laatste gameobject dat belangrijk is om eender welk canvas te laten werken: het "EventSystem". Deze wordt automatisch toegevoegd wanneer een canvas wordt aangemaakt. Zonder deze is er geen interactie mogelijk met het canvas.



Figuur 69: Canvas met EventSystem

Dit canvas moest tekst laten zien in het zicht van de gebruiker en later ook de "Fade_Screen_Image". Daarvoor is het mogelijk om in de componenten van het "UI_Canvas" de camera te gebruiken als locatie voor het renderen van het canvas.

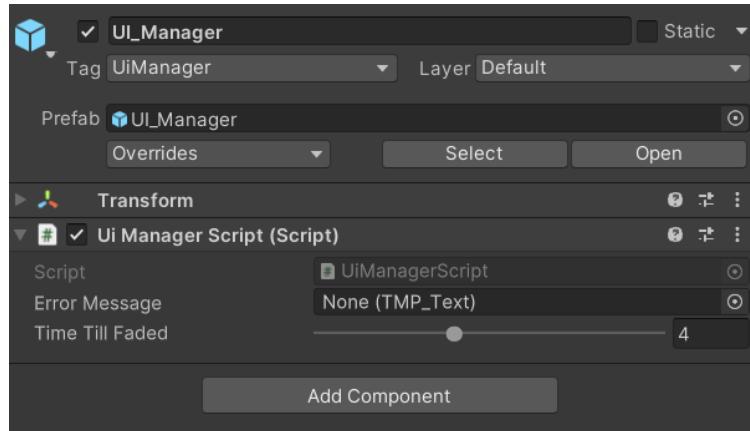
Als de "Render Mode" op "Screen Space – Camera" staat, wordt het zicht van de camera gebruikt om het canvas in te renderen. Bij de "Render Camera" slepen we de "Main Camera", dit is de camera die we gebruiken om door te kijken. Als afstand vanaf de camera, de "Plane Distance", gebruiken we 0.2, dit komt overeen met 20 cm. Oorspronkelijk was dit 0, maar door de vreemde werking van een VR-bril hebben we deze afstand moeten vergroten. Dit wordt wat verder uitgelegd onder het titeltje



Figuur 70; Canvas configuratie

Verder hebben we ook een manager om de UI te managen. Deze heeft ook weer een zelfgeschreven script: het "UiManagerScript". De variabelen die we kunnen invullen in de Unity Editor zijn voor het tonen van een foutmelding in het midden van het scherm. Tijdens het werken met de teleportatie-functionaliteit werd hiermee duidelijk gemaakt aan de gebruiker dat er locaties waren waar niet naar geteleporteerd kon worden. Hiervoor kon het "Error_Message"-gameobject in de "Error Message"-variabele worden

gesleept. De “Time Till Faded” was de tijd dat het achteraf duurde voor de tekst om te verdwijnen. Zo bleef de tekst niet voor altijd op het scherm.



Figuur 71: UI_Manager in Inspector

Daarnaast wordt dit script ook gebruikt om het “Handmenu” in en uit te schakelen. Dit is een functionaliteit dat we niet in een script kunnen plaatsen dat vastzit aan het “Handmenu”. Als het “Handmenu” wordt uitgeschakeld kan er namelijk geen gebruik worden gemaakt van scripts die aan dit gameobject vasthangen.

4.4.1.2. Code

In het “UiManagerScript” zijn er de initiële variabelen die we ook zien in de Unity Editor. “errorMessage” is van de klasse TMP_Text [43], dit is het type gameobject dat we gebruiken om tekst te tonen in een 3D-omgeving. Bij “TimeTillFaded” voegen we een [Range]-attribuut toe dat ervoor zorgt dat we in de Unity Editor een slider krijgen die niet onder of boven de voorziene waarden kan gaan.

Naast het gebruik van de “headsetControls” die we gebruiken om te luisteren naar de input hebben we nog een globale variabele. Het “handMenu”-variabele laten we niet vanuit de Unity Editor ingevuld worden omdat dit er per scene toch maar een kan zijn. Deze zullen we dan ook zelf ophalen aan het begin van het script.

```
[SerializeField]
private TMP_Text errorMessage;
[SerializeField]
[Range(0.1f, 10f)]
private float timeTillFaded = 4;

HeadsetControls headsetControls;

GameObject handMenu = null;
```

Figuur 72: globale variabelen van het UiManagerScript

De “Start”-methode [44] komt ook van de Unity-library en staat in de hiërarchie onder de “OnEnable” en “Awake” methodes. Net zoals de “Awake”methode wordt deze slechts één keer uitgevoerd aan het begin van het script. De “Start”-methode wordt over het algemeen meer gebruikt dan de “Awake”-methode. Als de “errorMessage”-variabele is opgevuld zetten we de alpha-waarde op 0, zo wordt de tekst onzichtbaar en staat deze niet altijd in het midden van ons scherm. Aangezien het “handMenu” altijd leeg is vullen we deze hier op met het gameobject met de juiste tag. Daarna maken we het “handMenu” inactief, zo is deze niet constant in de weg.

```
Unity Message | 0 references
void Start()
{
    if (errorMessage != null)
    {
        errorMessage.alpha = 0;
    }

    handMenu = GameObject.FindGameObjectWithTag("HandMenu");
    handMenu.SetActive(false);
}
```

Figuur 73: Start-methode van het UIManagerScript

Deze methode pakt de tekst-waarde van de “errorMessage”-variabele. Deze wordt dan ingevuld met de tekst die is meegegeven in de “message”-parameter. Deze methode zou oorspronkelijk vanuit andere scripts opgeroepen moeten worden. Daarna beginnen we een “Coroutine” om de tekst weg te faden, om de tekst uit het beeld te doen verdwijnen.

```
1 reference
public void ShowMessage(string message)
{
    errorMessage.text = message;
    StartCoroutine(FadeText());
}
```

Figuur 74: ShowMessage-methode in het UIManagerScript

In deze “Coroutine” zetten we eerst de alpha-waarde op 1. Dan gaan we deze weer met een while-loop kleiner maken met een voorberekende “waitTime”-variabele en een hard-coded verkleinende waarde van 0.05. In deze methode is er niet veel rekening gehouden met het voorkomen van errors. Dit is omdat de methode maar voor kort werd gebruikt en daarna nooit meer moest worden nagekeken.

```

1 reference
IEnumerator FadeText()
{
    errorMessage.alpha = 1;
    float waitTime = timeTillFaded / (1 / 0.05f);
    while (errorMessage.alpha > 0)
    {
        errorMessage.alpha -= 0.05f;
        yield return new WaitForSeconds(waitTime);
    }
}

```

Figuur 75: *FadeText-Coroutine in UIManagerScript*

Binnen de “Update”-methode van het “UiManagerScript” doen we elke frame een check of het “HandMenu” mag opgeroepen worden of niet. Om te voorkomen dat er problemen zijn tijdens met het teleporteren, wat later wordt uitgelegd, kan deze alleen worden opgeroepen als er niet wordt geteleporteerd. Voor het teleporteren maken we gebruik van een object met de tag “TeleportCircle”. Als deze niet aanwezig is in de scene kan er ook niet geteleporteerd worden en mogen we het “HandMenu” op actief zetten.

Als we het “HandMenu” mogen oproepen of niet doen we eerst een check of de controls voor deze actie zijn enabled of niet. Is dit niet wanneer het wel nodig is zullen we deze activeren. Zijn deze wel enabled wanneer het niet mag worden deze gedisabled. Wanneer het wel mag doen we ook een check of de knop voor het activeren wordt ingedrukt of niet.

```

1 reference
void Update()
{
    GameObject go = GameObject.FindGameObjectWithTag("TeleportCircle");
    if (go == null)
    {
        if (!headsetControls.HandMenu.enabled) headsetControls.HandMenu.Enable();
        headsetControls.HandMenu.Left_Primary_Button.performed += hideHandMenu;
    }
    else
    {
        if (headsetControls.HandMenu.enabled) headsetControls.HandMenu.Disable();
    }
}

```

Figuur 76: *Update-methode in het UIManagerScript*

De methode voor het activeren is vrij simpel. Als het menu actief is in de huidige scene wordt deze op inactief gezet, anders op actief. Dit kan door simpelweg de omgekeerde waarde te pakken van de huidige toestand met de “!”-operator. “activeInHierarchy” [45] checkt of een object actief is in de hiërarchie van de huidige scene.

```

1 reference
public void hideHandMenu(InputAction.CallbackContext context)
{
    //if active becomes inactive and vice-versa
    handMenu.SetActive(!handMenu.activeInHierarchy);
}

```

Figuur 77: *HideHandMenu-methode in het UIManagerScript*

4.4.2. Handmenu

Het “HandMenu”-gameobject heeft 3 knoppen. De knop linksboven doet een herberekening van de gemiddelde hartslag, meet uitleg over deze berekening komt bij “Omniccept Data”. De knop rechtsboven berekent zorgt voor de recalibratie, zoals eerder al was uitgelegd. De knop in het midden zorgt ervoor dat men terug kan gaan naar de “MainHub”-omgeving van waaruit men omgevingen kan selecteren. De uitleg over het switchen tussen scenes komt bij het titeltje “Scene Overgang”



Figuur 78: Handmenu in een omgeving

Wanneer we ons in de “MainHub” bevinden zal deze knop veranderen van uiterlijk en functie. Daar zal hij ervoor zorgen dat men de game kan afsluiten.



Figuur 79: HandMenu in MainHub

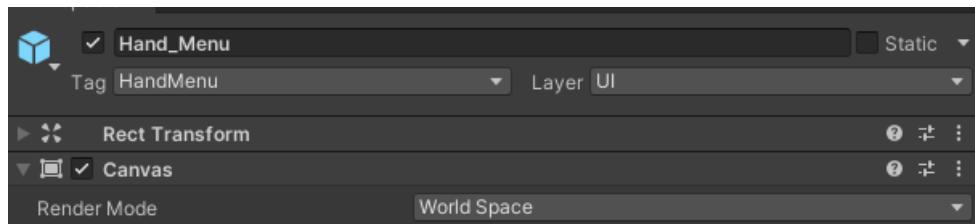
4.4.2.1. Unity Editor

In de hiërarchie zien we inderdaad de 3 knoppen die we eerder hebben gebruikt en een “Panel” [46]. Een panel is een object dat we kunnen gebruiken om visueel een achtergrond te voorzien voor een canvas. Het “HandMenu” is gewoon een canvas dat we in de “world space” hebben geplaatst als een child-object van de “LeftHand”. Deze staat boven de knoppen in de hiërarchie omdat deze anders voor de knoppen zou staan tijdens het spelen van de VR-game. Dan kan men dus niet op de knoppen duwen.



Figuur 80: HandMenu in hiërarchie

Zoals eerder vermeld staat dit canvas in de “Render Mode” “World Space”. Zo wordt deze niet vlak voor de camera gehangen, maar op de positie waar deze zich daadwerkelijk bevindt in de 3D-omgeving.

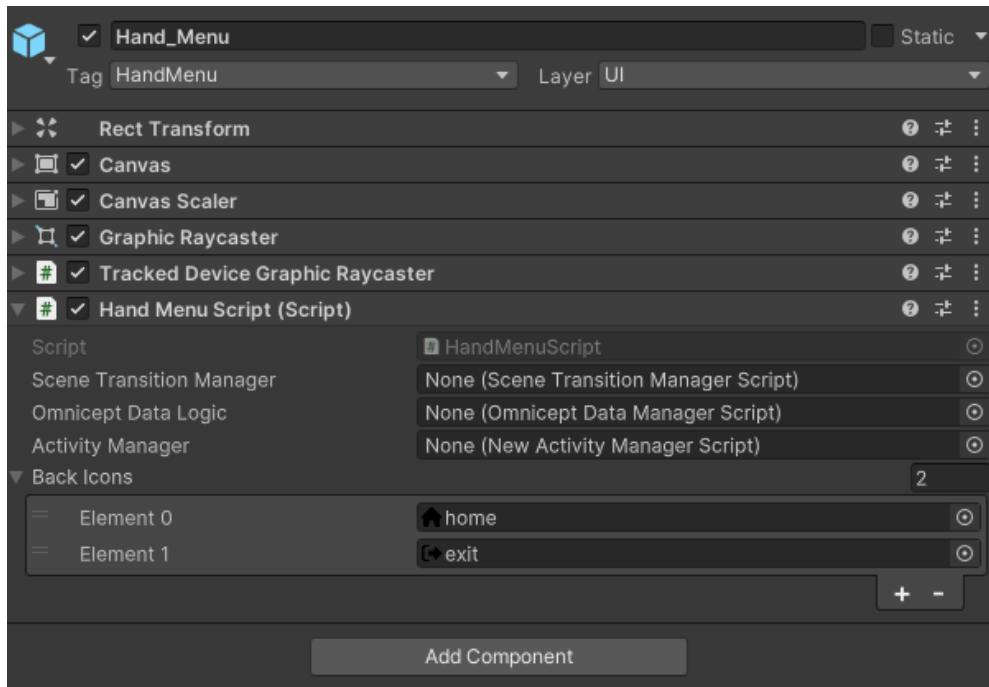


Figuur 81: HandMenu in World Space

Het “HandMenu” heeft ook een eigen “HandMenuScript”. Dit script kan alleen door het “HandMenu” worden gebruikt omdat het specifiek voor dit gameobject gemaakt werd.

De eerste drie variabelen zijn managers die later nog worden uitgelegd. Kort dient de “Scene Transition Manager” om de overgang tussen scenes te managen en naar welke scene er moet gegaan worden. De “Omniccept Data Logic” regelt alle data van de Omniccept die we gebruiken. De “Activity Manager” start en stopt de activiteit-scripts die we later gaan gebruiken.

De “Back Icons” is een array met de afbeeldingen die we gebruiken. Dit zijn specifiek de symbolen om terug te gaan naar de “MainHub” of om de game af te sluiten.



Figuur 82: HandMenu in Inspector

4.4.2.2. Code

De eerste drie globale variabelen kunnen worden ingevuld vanuit de Unity Editor, of we vullen ze op in onze code.

De array met als naam “backIcons” moet vanuit de Unity Editor ingevuld worden. We maken gebruik van de klasse “Texture2D” [47] zodat we de afbeeldingen als PNG’s zonder achtergrond kunnen gebruiken. Het is niet mogelijk om deze in een map te steken en automatisch op te halen. Bij het builden van de applicatie zou dit problemen geven.

```
[SerializeField]
private SceneTransitionManagerScript sceneTransitionManager;
[SerializeField]
private OmniceptDataManagerScript OmniceptDataLogic;
[SerializeField]
private NewActivityManagerScript activityManager;
[SerializeField]
private Texture2D[] backIcons;
```

Figuur 83: globale variabelen HandMenuScript

We maken ook gebruik van een listener uit de “omniceptDataLogic”. Deze kijkt of de gemiddelde hartslag wordt berekend of niet. Wanneer dit wordt berekend zal de knop niet indrukbaar zijn. En als de berekening klaar is of niet bezig is zal deze knop wel in te drukken zijn. Deze “OnTimerCountsChange” wordt later nog uitgelegd. Ditzelfde gebeurt voor de recalibratieknop. Wanneer we in een activiteit zitten mogen we niet recalibreren omdat dit problemen kan veroorzaken.

```

    Unity Message | 0 references
private void OnEnable()
{
    if(OmniceptDataLogic != null)
    {
        DisableResetButton(OmniceptDataLogic.timerCounts);
        OmniceptDataLogic.OnTimerCountsChange += DisableResetButton;
    }

    if (activityManager != null)
    {
        DisableRecalibrateButton(activityManager.activityIsActive);
        activityManager.OnActivityIsActiveChange += DisableRecalibrateButton;
    }
}

```

Figuur 84: *OnEnable-methode van het HandMenuScript*

Wanneer het “HandMenu” wordt gedisabled of vernietigt met “OnDestroy” [48] zullen we de listeners ook verwijderen. Zo blijven deze niet werken als dit niet nodig is.

```

    Unity Message | 0 references
private void OnDisable()
{
    OmniceptDataLogic.OnTimerCountsChange -= DisableResetButton;
    activityManager.OnActivityIsActiveChange -= DisableRecalibrateButton;
}

    Unity Message | 0 references
private void OnDestroy()
{
    OmniceptDataLogic.OnTimerCountsChange -= DisableResetButton;
    activityManager.OnActivityIsActiveChange -= DisableRecalibrateButton;
}

```

Figuur 85: *verwijderen listener in HandMenuScript*

In de “Start”-methode gebeurt er deze keer vrij veel. Dit is voornamelijk omdat we niet meteen een korte makkelijke manier vonden om de knoppen op te halen. Deze code zet eerst een default waarde voor de “buttonImageName” en de “buttonMethod”. Deze default-waarden zijn om terug te gaan naar de “MainHub”. De “buttonMethod” maakt gebruik van de “UnityAction”-klasse [49]. Hiermee kunnen we een methode meegeven om toe te voegen aan een knop.

Als we de naam van de huidige scene “MainHub” is zullen de naam van de “buttonImageName” en “buttonMethod” veranderen naar de waarden om de app af te sluiten.

Dan halen we alle knoppen op die children zijn van het huidige object. Het huidige object is het “HandMenu”, dus de knoppen die we eerder getoond hebben zijn de children. Als de naam gelijk is aan de waarde “Back_Button” zal deze knop als methode de “buttonMethod” meekrijgen.

Elke knop heeft een “Image”-gameobject om afbeeldingen te tonen, voor de “Back_Button” noemt deze “Back_Button_Image”. Deze vullen we dan ook in met de afbeelding uit onze “backButtons”-array waarbij de naam overeenkomt met de “buttonImageName”.

Deze wijze van werken is vrij omslachtig. Maar ze werd bedacht op een moment dat ik probeerde te werken rond het probleem dat kon ontstaan wanneer iemand een verkeerde knop of button toevoegde via de Unity Editor invul-wijze.

Achteraf vullen we ook nog enkele variabelen voor managers in als deze nog niet waren ingevuld. Dit werkt hetzelfde als we al eerder hadden gezien.

```
Unity Message | 0 references
void Start()
{
    //change the variables for the button accordingly, default is mainhub/home
    string buttonImageName = "home";
    //methods that have been declared this way can't be called from another script, so a method in this script calls them
    UnityAction buttonMethod = new UnityAction(GoToMain);

    if (SceneManager.GetActiveScene().name == "MainHub")
    {
        buttonImageName = "exit";
        buttonMethod = new UnityAction(CloseGame);
    }

    //change the values for the button
    Button[] buttons = gameObject.GetComponentsInChildren<Button>();
    Image[] images = gameObject.GetComponentsInChildren<Image>();

    foreach(Button button in buttons)
    {
        if(button.gameObject.name == "Back_Button") button.onClick.AddListener(buttonMethod);
    }

    foreach (Image image in images)
    {
        if(image.gameObject.name == "Back_Button_Image")
        {
            Texture2D texture = Array.Find(backIcons, x => x.name == buttonImageName);
            image.sprite = Sprite.Create(texture, new Rect(0, 0, texture.width, texture.height), new Vector2(0.5f, 0.5f));
        }
    }

    //enable necessary scripts
}
```

Figuur 86: Start-methode in het HandMenuScript

Deze methoden hebben we eerder al gezien bij het gebruiken van de listeners. Omdat de code zo goed als hetzelfde was voor beide heb ik een methode gemaakt waar ze beide gebruik van kunnen maken. We gebruiken hier weer de “!”-operator. Dit is omdat, wanneer de “activityisActive” of “reset” true zijn, de knoppen niets mogen doen. Verder moeten deze methodes beiden uniek zijn om de listener te laten werken.

```
//two methods that are used as listeners, they use a method to avoid rewriting code
4 references
private void DisableRecalibrateButton(bool activityIsActive)
{
    SetButton("Recalibrate_Button", !activityIsActive);
}

4 references
private void DisableResetButton(bool reset)
{
    SetButton("Reset_Button", !reset);
}
```

Figuur 87: methodes voor listeners in het HandMenuScript

In deze algemene methode halen we alle knoppen op die een child zijn van het huidige gameobject. Dan kijken we of de naam overeenkomt met de “buttonName”-parameter zal er in deze knop een actie worden uitgevoerd. De “interactable” waarde van de knop zal worden gezet naar de “enable” waarde. Wanneer deze false is zal de knop niet meer kunnen gebruikt worden om op te klikken tot deze weer wordt geactiveerd.

```
//this method is to avoid rewriting code
2 references
private void SetButton(string buttonName, bool enable)
{
    Button[] buttons = gameObject.GetComponentsInChildren<Button>();
    foreach (Button button in buttons)
    {
        if (button.gameObject.name == buttonName)
        {
            button.interactable = enable;
            break;
        }
    }
}
```

Figuur 88: Algemene SetButton-methode in het HandMenuScript

Door de manier waarop het “omniceptDataLogic”-script werkt kunnen we deze gewoon disablén en dan weer enablen om de berekening opnieuw te laten werken. Dit is de methode die achter de resetknop linksboven aan het “HandMenu” zit.

```
// simple restartmethod by disabling and enabling the script, this does also restart the writing of the script
0 references
public void ResetOmniceptData()
{
    OmniceptDataLogic.enabled = false;
    OmniceptDataLogic.enabled = true;
}
```

Figuur 89: ResetOmniceptData-methode in het HandMenuScript

Deze methoden zijn nodig om de “UnityAction” te laten werken. Wanneer een methode niet in hetzelfde script staat kan “UnityAction” hier geen gebruik van maken. De logica zit in de respectievelijke scripts.

```
//helpermethods to access methods in another script
1 reference
public void GoToMain()
{
    sceneTransitionManager?.GoToSceneByName();
}

1 reference
public void CloseGame()
{
    sceneTransitionManager?.CloseGame();
}
```

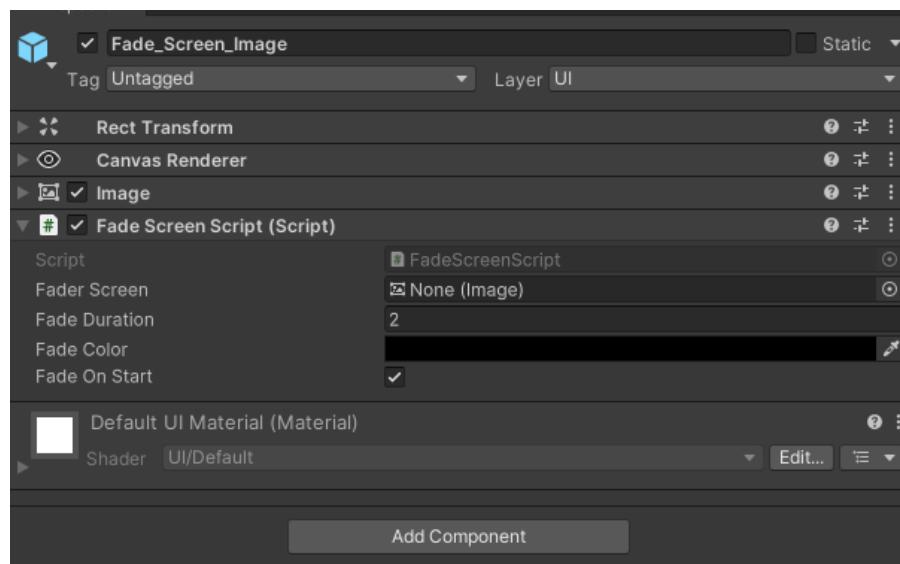
Figuur 90: helpermethoden voor UnityAction

4.4.3. Fade

Het concept voor de fade kwam oorspronkelijk van mijn collega. Nadat we hier meermalen problemen mee ondervonden, had ik dit, met toestemming, meegenomen in mijn takenpakket. Na wat opzoekwerk kwam ik er achter dat de beste manier om hiermee te werken was om een zwart vlak in een canvas te plaatsen. Van dit zwart vlak kunnen we dan de alpha-waarden aanpassen zodat het zichtbaar en onzichtbaar wordt. Dit vlak hebben we eerder gezien in de vorm van het “Fade_Screen_Image”.

4.4.3.1. Unity Editor

Het script dat we gebruiken is het “FadeScreenScript”. Ik heb gewerkt met wat mijn collega oorspronkelijk gebruikte. Ik heb er alleen voor gezorgd dat het werkt. De “Fade Screen” is de “Image” van het huidige gameobject. Deze kan er in worden gesleept of in de code opgehaald worden. De “Fade Duration” is hoe lang het duurt voor het faden gedaan is. De “Fade Color” hoeft niet echt gebruikt te worden, maar zou in de toekomst voor fades naar andere kleuren kunnen gebruikt worden. “FadeOnStart” bepaalt of er een fade moet gebeuren wanneer de scene start.



Figuur 91: *Fade_Screen_Image* in de inspector

4.4.3.2. Code

De meeste variabelen zijn hier niet anders, behalve de “_FadeDuration”. Dit was de waarde die we in de Unity Editor hebben aangepast. Omdat deze gebruikt wordt in andere scripts zou deze normaal op “public” moeten staan. Maar dan zou deze ook verandert kunnen worden in andere scripts. Dit is niet de bedoeling, daarom maken we gebruik van een getter en setter. Doordat we een “private set” gebruiken kan deze waarde alleen maar aangepast worden in dit scripts als dit nodig is.

```
[SerializeField]
private Image faderScreen;

[SerializeField]
private float _FadeDuration = 2f;

[SerializeField]
private Color fadeColor;

[SerializeField]
private bool fadeOnStart = true;

10 references
public float fadeDuration
{
    get { return _FadeDuration; }
    private set { _FadeDuration = value; }
}
```

Figuur 92: globale variabelen in het FadeScreenScript

Hier zien we dat wanneer de scene wordt ingeladen en het “Fade_Screen_Image” ook, we onmiddellijk een “FadeIn” doen. Deze methode doet een fade waarbij het scherm eerst zwart is en geleidelijk aan de omgeving zichtbaar wordt.

```
@ Unity Message | 0 references
void Start()
{
    if(faderScreen == null) faderScreen = gameObject.GetComponent<Image>();

    if (fadeOnStart) FadeIn();
}
```

Figuur 93: Start-methode in het FadeScreenScript

De “FadeIn” start bij een zwart scherm, alpha van “Fade_Screen_Image” is 1, en gaat naar een zichtbare omgeving, alpha van “Fade_Screen_Image” is 0. Bij de “FadeOut” is dit omgekeerd. Daarnaast wordt de “faderScreen”-variabele ook aangezet voor de “Fade”-methode bij “FadeOut”. Dit is om een probleem te voorkomen, wanneer de “FadeIn” klaar is zetten we de “faderScreen” af. Zo voorkomen we visuele problemen tijdens het spelen omdat het “Fade_Screen_Image” nog in de weg zou hangen en het beeld doet “wobbelen”.

```
3 references
public void FadeIn()
{
    Fade(1, 0);
}

4 references
public void FadeOut()
{
    faderScreen.enabled = true;
    Fade(0, 1);
}
```

Figuur 94: FadeIn en FadeOut methodes in het FaderScreenScript

De algemene “Fade”-methode start de “Coroutine” op. Tijdens deze “Coroutine” gebeurt de daadwerkelijke fade.

```
2 references
private void Fade(float alphaIn, float alphaOut)
{
    StartCoroutine(FadeRoutine(alphaIn, alphaOut));
}
```

Figuur 95: algemene Fade-methode in het FadeScreenScript

In deze “Coroutine” gebruiken we ook weer een “timer” om de fade te laten duren tot de “fadeDuration” bereikt is. Volgens mijn collega moet de “newColor” blijven staan om dit te laten werken. Daarna veranderen we de “a”-waarde [50] van de “Color” [51] naar het punt dat we zitten op onze weg naar de “fadeDuration”. We maken hier gebruik van “Lerp” [52], een methode om het procentuele punt tussen 2 waarden te berekenen. In dit geval zijn dit de parameters “alphaIn” en “alphaOut”. Vervolgens wordt de “Color” van onze “fadeScreen” op deze nieuwe waarde gezet.

Aan het einde vermeerderen we de “timer” met “Time.deltaTime” [53]. “Time.deltaTime” geeft ons de tijd, in de vorm van een float, sinds de laatste frame. Daarna doen we een “yield return null” zodat we wachten tot de volgende frame er is voor we verder gaan met de loop.

Na de loop zorgen we ervoor dat de “fadeScreen” zeker op de “alphaOut” is beland. Een probleem met “Lerp” is dat het een procentueel punt tussen twee waarden berekend. Dan kan deze waarde ook juist boven of onder de eindwaarde zitten. Als de eindwaarde van de “newColor.a” 0 is, zetten we de “fadeScreen” af. Dit is om het eerder vermelde visuele probleem te voorkomen.

```
1 reference
private IEnumerator FadeRoutine(float alphaIn, float alphaOut)
{
    float timer = 0;
    Color newColor = fadeColor;
    while (timer <= fadeDuration)
    {
        //use of Lerp makes it so we can smoothly go to alphaOut using a percentage
        newColor.a = Mathf.Lerp(alphaIn, alphaOut, timer / fadeDuration);
        faderScreen.color = newColor;
        timer += Time.deltaTime;
        yield return null;
    }
    //to ensure our alphaOut is reached since Lerp could over- or undershoot alphaOut
    newColor.a = alphaOut;
    faderScreen.color = newColor;
    if (newColor.a == 0) faderScreen.enabled = false;
}
```

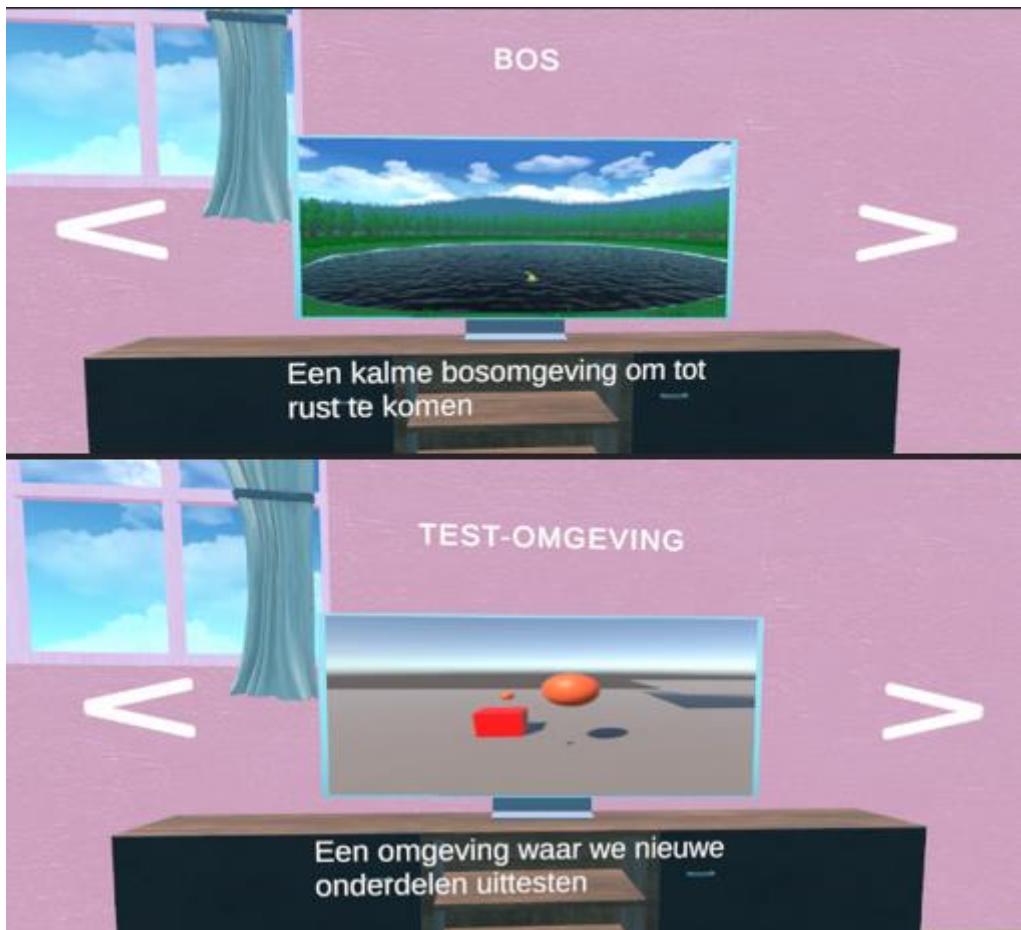
Figuur 96: FadeRoutine-Coroutine in het FadeScreenScript

4.5. Scene overgang

We moesten ons van scene naar scene kunnen verplaatsen en rekening houden met het feit dat er ooit nog nieuwe scenes zouden kunnen worden toegevoegd. Verder moeten we vanuit deze scenes ons ook terug kunnen verplaatsen naar de "MainHub" om naar een nieuwe scene te kunnen gaan.

Voor dit probleem heb ik het "SceneTransitionManagerScript" gemaakt. Dit zit in een gameobject met als naam "Transition_Manager", dit object slepen we in elke scene. Daarnaast maken we deze keer voor het eerste gebruik van een eigen klasse. Deze klasse wordt voornamelijk gebruikt in de "MainHub" om visueel duidelijk te maken welke opties er waren voor de verplaatsing.

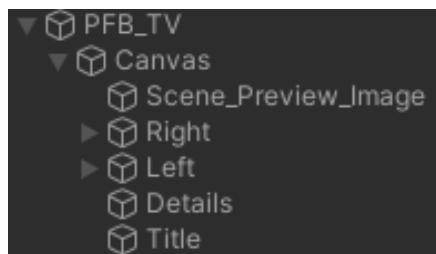
Verder hebben we in de "MainHub" ook een canvas toegevoegd waar de gebruiker een omgeving naar wens kan selecteren. Om alle informatie van een omgeving bij elkaar te tonen op het canvas maken we dan gebruik van de nieuwe klasse.



Figuur 97: Tv voor selectie omgeving

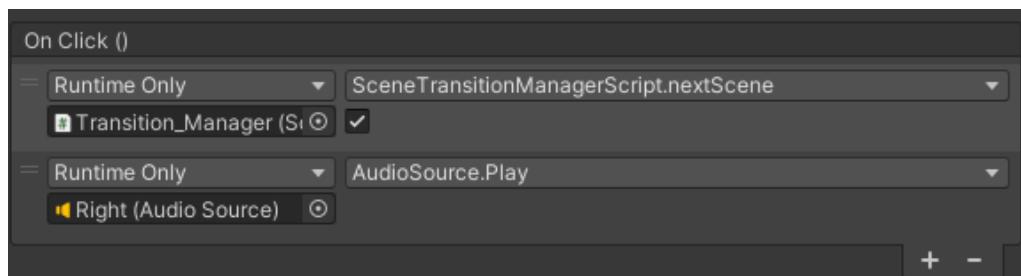
4.5.1. Selecteren van een omgeving

Om het canvas voor de selectie binnen het thema van de “MainHub” te laten passen hebben we deze voor een tv-gameobject geplaatst. Dit is gewoon een gameobject met het model van een tv en niet echt iets speciaal. Verder heeft dit canvas een afbeelding als preview van de omgeving. Knoppen om naar de volgende en vorige in de lijst te gaan. Ook een “Text”-gameobject voor de titel van de omgeving en een voor een korte omschrijving van de omgeving.



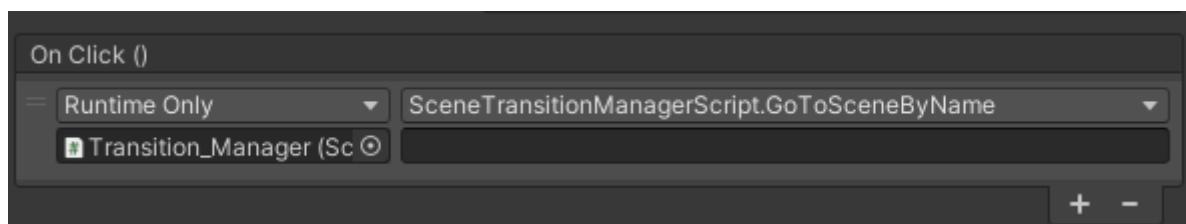
Figuur 98: Tv met canvas in Hiërarchie

In de “OnClick”-array van een knop, een onderdeel van Unity, kunnen we ook methodes plaatsen. In dit geval zitten daar twee methodes in. De eerste is een methode van ons eigen “SceneTransitionManagerScript” waarmee de volgende scene in de lijst wordt geselecteerd. De tweede methode speelt een soundeffect af dat door mijn collega is toegevoegd.



Figuur 99: OnClick van de Right-knop

Er zit ook een knop achter de “Scene_Preview_Image”, deze heeft een methode die de gebruiker verplaatst naar de huidig geselecteerde scene. Er is hier ook een leeg tekstvak te, maar dat wordt later duidelijk.

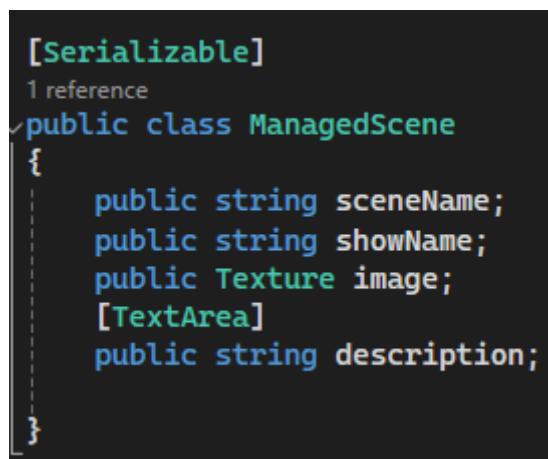


Figuur 100: OnClick van de Scene_Preview_Image

4.5.2. ManagedScene

Deze keer is er geen overerving van de MonoBehaviour-klasse. Dit is omdat deze klasse niet moet getoond worden in de Unity Editor als een component dat kan toegevoegd worden aan een gameobject. Wel hebben we hier het [Serializable]-attribuut [54]. Met dit attribuut kan de klasse ook worden ingevuld in de Unity Editor als variabele van een zelfgeschreven script.

Zoals te zien is, zijn de "sceneName" en "showName" gewoon tekst dat ingegeven kan worden. Bij "image" is er wel een "Texture"-klasse [55], deze moet gebruikt worden volgens de code die mijn collega oorspronkelijk had geschreven. Er is niet veel verschil met de "Texture2D"-klasse. Bij de "description" staat het [TextArea]-attribuut [56], zo kan deze tekst in de Unity Editor gemakkelijk worden ingevuld in een groter tekstvak.



```
[Serializable]
public class ManagedScene
{
    public string sceneName;
    public string showName;
    public Texture image;
    [TextArea]
    public string description;
}
```

Figuur 101: ManagedScene-klasse

De "sceneName" is de naam van de daadwerkelijke scene waar naartoe moet genavigeerd worden. De "showName" is de naam die we in de app gebruiken om aan de gebruikers te laten zien.



Figuur 102: ManagedScene voorbeeld in de Unity Editor

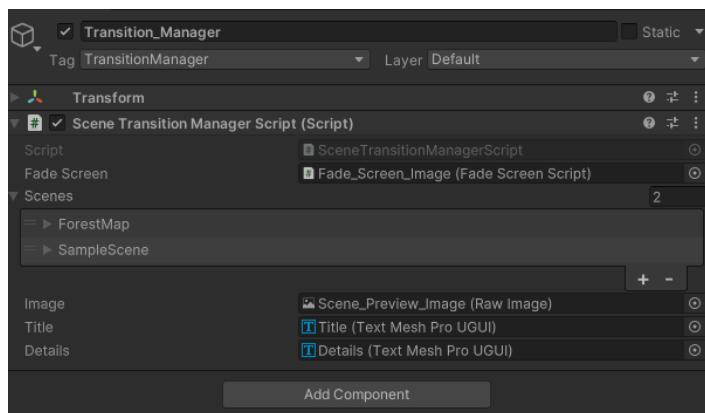
4.5.3. Transition_Manager

De "Transition_Manager" is voornamelijk bedoelt om te kiezen naar welke omgeving we kunnen gaan in de "MainHub". Verder zorgt deze ook voor de daadwerkelijke verplaatsing. Omdat er in deze manager ook methoden staan om naar een andere omgeving te gaan kunnen we deze dusdanig ook buiten de "MainHub" gebruiken.

4.5.3.1. Unity Editor

De eerste variabele is de "Fade Screen", dit is het script dat we eerder aanmaakten. In de "MainHub" wordt de "Scenes"-array ook opgevuld met de scenes waar naartoe gegaan kan worden. We hebben tijdens deze stageperiode slechts de tijd gehad om één omgeving te maken waar de gebruiker echt iets kan ervaren om tot rust te komen. De andere omgeving, de "SampleScene", was de scene waar ik voornamelijk op heb gewerkt. Hier heb ik onderdelen getest voordat ik deze toevoegde aan de "ForestMap", waar mijn collega het meest tijd heeft gespendeerd. Wanneer het project wordt gebuild, zou de "SampleScene" uit de array mogen verwijderd worden.

Verder vinden we de variabelen waar we de onderdelen van het canvas van de tv in kunnen plaatsen. Dit zijn de "Scene_Preview_Image", de "Title" en de "Details"-gameobjecten.



Figuur 103: Transition_Manager in de inspector

4.5.3.2. Code

De variabelen die we in de Unity Editor zien, zijn exact hetzelfde als we al een aantal keer hebben gezien. Het zijn globale variabelen die niet vanuit andere scripts bereikt mogen worden, maar wel vanuit de Unity Editor met behulp van [SerializeField].

Wel is er een globale variabele om bij te houden welke scene er op het moment te zien is op het tv-canvas. Dit is een getal om de index binnen de array bij te houden. Deze begint op 0, omdat een array ook vanaf 0 begint te tellen.

```
private int currentSelectedScene = 0;
```

Figuur 104: currentSelectedScene variabele in het ScenTransitionManagerScript

Binnen de "Start"-methode maken we gebruik van de "SetCurrentScene"-methode. Maar alleen als de variabelen van het tv-canvas niet leeg zijn. Deze methode toont dus de juiste tekst en afbeelding op het tv-canvas als er een canvas is om dit op te tonen. Verder vullen we het "fadeScreen" deze keer met een andere methode. "FindObjectOfType" [57] vindt in de huidige hiërarchie het eerste object met het overeenkomende datatype.

```
Unity Message | 0 references
private void Start()
{
    if (scenes.Length > 0 && image != null && title != null && details != null)
    {
        setCurrentScene();
    }

    if(fadeScreen == null)
    {
        fadeScreen = GameObject.FindObjectOfType<FadeScreenScript>();
    }
}
```

Figuur 105: Start-methode in het SceneTransitionManagerScript

De “SetCurrentScene”-methode vervangt de huidige waarden van het tv-canvas door de waarden die in de “ManagedScene”-array op de index van de “currentSelectedScene”-variabele staan.

```
2 references
public void setCurrentScene()
{
    image.texture = scenes[currentSelectedScene].image;
    title.text = scenes[currentSelectedScene].showName;
    details.text = scenes[currentSelectedScene].description;
}
```

Figuur 106: SetcurrentScene-methode in het ScenTransitionManagerScript

De “NextScene”-methode zal, als de array is opgevuld, de index verhogen. Als de “next”-parameter op false staat, wordt de index verlaagt met 1. Op deze manier kan de methode gebruikt worden om voorwaarts en achterwaarts door de methode te gaan. Er is ook een check om te zien of de nieuwe waarde kleiner is dan 0, of boven de limiet van de array gaat. Wanneer dit het geval is zal de waarde naar het einde of het begin van de array gaan.

Op het einde gebruiken we nog eens de “SetCurrentScene”-methode om het tv-canvas mee te updaten.

```

0 references
public void nextScene(bool next)
{
    if (scenes.Length > 0)
    {
        currentSelectedScene += next ? 1 : -1;

        if (currentSelectedScene >= scenes.Length)
        {
            currentSelectedScene = 0;
        }
        else if (currentSelectedScene < 0)
        {
            currentSelectedScene = scenes.Length - 1;
        }

        //To also change the scene in the environment
        if (image != null && title != null && details != null)
        {
            setCurrentScene();
        }
    }
}

```

Figuur 107: NextScene-methode in het SceneTransitionManagerScript

“CloseGame” zal de game afsluiten als we de methode gebruiken. Wanneer we builden moeten we de lijn met “UnityEditor” in comment zetten. Dit werkt niet bij een build en zal het build-process ook stoppen en een error gooien. “Application.Quit” sluit de applicatie wel af als de applicatie is gebuild. Deze methode staat hier om de navigatiemogelijkheden in slechts een script te houden. Dan moeten we deze niet op meerdere plaatsen zoeken.

```

1 reference
public void CloseGame()
{
    //because of the difference between the editor and a build product we comment out the next lines for the build
    UnityEditor.EditorApplication.isPlaying = false;
    //Application.Quit();
}

```

Figuur 108: CloseGame-methode in het SceneTransitionManagerScript

Deze methode zal de huidige scene afsluiten en naar de gewenste gaan met behulp van een “Coroutine”. De methode maakt gebruik van de naam van de scene. De “name”-parameter is leeg, maar dit wordt duidelijker in de “Coroutine”.

```

1 reference
public void GoToSceneByName(string name = "")
{
    StartCoroutine(GoToSceneRoutineByName(name));
}

```

Figuur 109: GoToSceneByName-methode in het SceneTransitionManagerScript

De “Coroutine” begint met een fade en een “WaitForSeconds”. De tijd van de “WaitForSeconds” duurt zo lang als de fade, doordat we de “fadeDuration” gebruiken als tijd. Daarna komt er een check voor te zien naar welke scene er wordt gegaan. Als de “name”-parameter is ingevuld gebruiken we deze scene. Anders kijken we of de

“scenes”-array leeg is of niet. Wanneer deze gevuld is gebruiken we de huidig geselecteerde scene. Als niets van de voorgaande opties mogelijk was gaan we default naar de “MainHub”. Door de methode in de knop een lege string mee te geven gaan we dus naar de geselecteerde scene. Wanneer we in een andere scene zitten en de array niet opvullen kunnen we met deze methode automatisch naar de “MainHub”.

```
1 reference
IEnumerator GoToSceneRoutineByName(string name)
{
    fadeScreen.FadeOut();
    yield return new WaitForSeconds(fadeScreen.fadeDuration);

    if (!string.IsNullOrEmpty(name)) //when the name is filled in
    {
        SceneManager.LoadScene(name);
    }
    else if (scenes.Length > 0) //when the name isn't filled in and the array of scenes isn't empty
    {
        SceneManager.LoadScene(scenes[currentSelectedScene].sceneName);
    }
    else //default to mainhub
    {
        SceneManager.LoadScene("MainHub");
    }
}
```

Figuur 110: GoToSceneRoutineByName-Coroutine in het SceneTransitionManagerScript

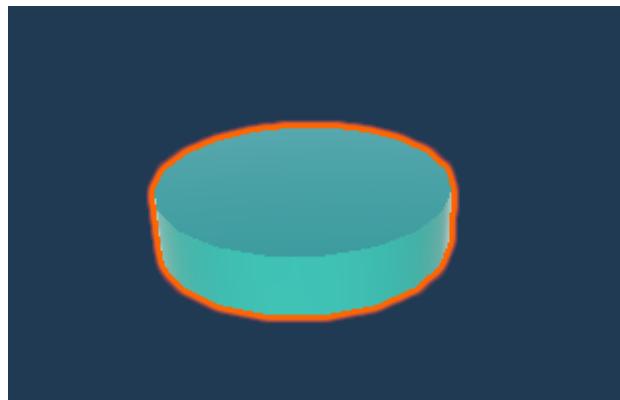
4.6. Teleportatie

Om te kunnen teleporteren kozen we voor de mogelijkheid om een vlak op te roepen dat we kunnen rond bewegen met een stick op onze controller. Dit was om niet met de andere optie die de “XR Interaction Toolkit” ons aanbied te moeten werken. Deze vereist namelijk dat we met onze arm moeten mikken. Wij hadden het vermoeden dat dit niet echt zou helpen bij het tot rust komen.

4.6.1. Prefab

Voor het werken met teleporteren moeten we een vlak oproepen en dan weer laten verdwijnen. Hiervoor maken we gebruik van een “prefab” [58]. Dit is een object dat we vooraf hebben gemaakt en ingesteld met alles wat we nodig hebben en die we dan via een script kunnen oproepen in een scene. Prefabs zitten in de assets-folder en kunnen van hieruit in een scene worden gesleept. Wanneer we aanpassingen maken in een scene aan een prefab kunnen we deze toepassen op de prefab in de assets-folder. Dan zullen deze veranderingen worden doorgevoerd in elke scene voor elke instantie van die prefab die als is gebruikt. Als een gameobject een blauwe tekst heeft in de hiërarchie is dit een prefab.

Voor het teleporteren maakten we gebruik van een cilinder waarvan de hoogte is ingesteld op 10 cm. Deze heeft ook een “rigidbody”-component [59] om ervoor te zorgen dat ze niet door alle objecten heen kan gaan.

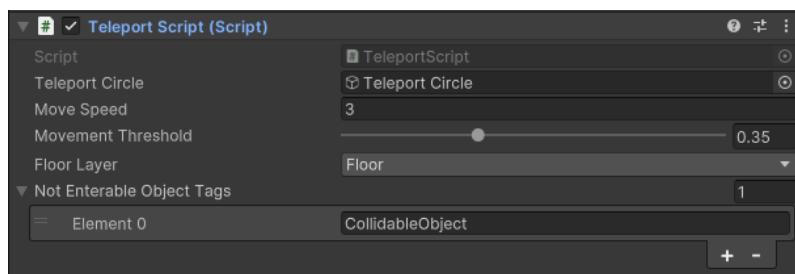


Figuur 111: Prefab voor Teleport Circle

Voor deze “Teleport Circle” is er ook een script. Dit “TeleportScript” zorgt voor de beweging van het gameobject en ook het vervolgens teleporteren naar de locatie van de cirkel.

Als eerste is er de plaats om het gameobject zelf in te plaatsen, deze wordt ook weer ingevuld in de code als dit niet hier gebeurt. Vervolgens zien we de snelheid waarmee de cirkel kan bewegen. Daarna volgt een “Movement Threshold”, deze is toegevoegd omdat er “drift” zat op de sticks van de controllers. “drift” betekent dat de controllers een input zullen doorgeven zonder dat de gebruiker dit zelf doet. De waarde die op deze slider wordt gebruikt is de minimale waarde die van de controllers moet komen voor de beweging zal worden geregistreerd.

“Floor Layer” is de layer [60] waarop de cirkel zal bewegen. Dit is een waarde die net zoals de tags kan worden ingegeven om orde te creëren. De array “Not Enterable Object Tags” is een lijst met tags van objecten waar de cirkel niet door mag gaan. Deze was langer, maar door aanpassingen van mijn collega is dit nu nog maar een tag dat moet gebruikt worden.



Figuur 112: TeleportScript in de Teleport Circle-prefab

4.6.2. TeleportScript

De prefab die we gebruiken moet een “GameObject” zijn, anders zal dit niet werken. De “moveSpeed” mag eender welke waarde hebben, zolang het maar een getal is. De “movementThreshold” mag van 0 tot 1 gaan, met een waarde van 1 zal de cirkel wel niet bewegen. Voor “floorLayer” gebruiken we “LayerMask” [61] als klasse, zo kunnen we in de Unity Editor namelijk gebruikmaken van een lijst met alle aanwezige layers. Voor tags is dit namelijk niet mogelijk zoals te zien is bij “notEnterableObjectTags”. Dit

is een array van het datatype string. Tags zijn niet echt een klasse die gemakkelijk bereikt kunnen worden.

Dan komen enkele variabelen die niet van buiten het script bereikt mogen worden. De “xrOrigin” is een andere naam voor de “XrRig”, deze kan niet worden ingegeven in de Unity Editor. Dit komt omdat een prefab niet kan weten wat de huidige gameobjecten zijn die in een scene zitten, voordat hij in deze scene zit. Deze moet daarom door de code worden opgehaald. “headsetControls” wordt altijd hetzelfde ingevuld. De “move”-variabele is een algemene variabele die zal lezen wat de input van de controller-stick is. De waarde van deze stick op onze controller is een Vector2 [62], dit is een 2-dimensionale vector. We kunnen bij deze stick immers alleen maar de x-en y-waarde lezen: horizontale en verticale beweging. Deze is leeg bij het initialiseren. Daarna is er nog een variabele die bepaald of we mogen bewegen of niet. Deze wordt aangepast wanneer we in contact komen met een object, zie het titeltje “Obstakels”.

```
[SerializeField]
private GameObject teleportCircle;
[SerializeField]
private int moveSpeed = 1;
[SerializeField]
[Range(0f, 1f)]
private float movementThreshold = 0.1f;
[SerializeField]
private LayerMask floorLayer;
[SerializeField]
private string[] notEnterableObjectTags;

private GameObject xrOrigin;
HeadsetControls headsetControls;
private Vector2 move = Vector2.zero;

private bool mayMove = true;
```

Figuur 113: globale variabelen in het TeleportScript

Omdat het invullen van sommige variabelen in de “Awake”-methode er exact hetzelfde uitziet als de vorige keren ga ik verder met de “OnEnable”-methode. Hier maken we gebruik van de methode “keepHeight”, deze methode zorgt ervoor dat de cirkel op de vloer blijft als hij wordt aangemaakt in de scene.

```
Unity Message | 0 references
private void OnEnable()
{
    headsetControls.XRILeftHand.Enable();

    //sets height when circle is activated
    KeepHeight();
}
```

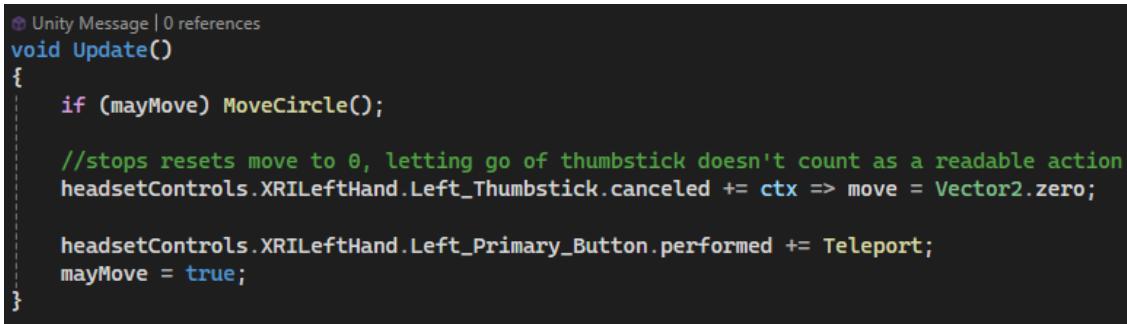
Figuur 114: OnEnable-methode van het TeleportScript

Binnen de “Update”-methode zien we vier zaken waar de “Teleport Circle” rekening mee moet houden. De eerste is de “MoveCircle”-methode. Deze leest de input van de

controller-stick en beweegt dan de cirkel. Omdat de "move"-variabele globaal is moeten we deze achteraf ook weer op een lege Vector2 zetten, anders beweegt de cirkel verder als we hebben losgelaten. De "mayMove"-variabele kan false worden in een methode die hoger staat in de hiërarchie, zie het titeltje "Obstakels" voor meer informatie.

Het cancelen of loslaten van een stick telt niet als een input. Dit is ook de reden dat de "move"-variabele globaal is en elke frame wordt gelezen. Als we de "MoveCircle"-methode enkel zouden uitvoeren bij een "performed" actie van de controller zou ik de stick niet ingedrukt kunnen houden om een richting op te blijven bewegen. Daarbij blijft de waarde namelijk gelijk en verandert er dus niets.

Voor te teleporteren gebruiken we slechts de druk op een knop. Deze mag dus wel uitgevoerd worden wanneer de knop wordt "performed". Voor de volgende frame zetten we "mayMove" ook terug op true. Deze zal opnieuw op false worden gezet als we toch niet mogen bewegen.



```
Unity Message | 0 references
void Update()
{
    if (mayMove) MoveCircle();

    //stops resets move to 0, letting go of thumbstick doesn't count as a readable action
    headsetControls.XRILeftHand.Left_Thumbstick.canceled += ctx => move = Vector2.zero;

    headsetControls.XRILeftHand.Left_Primary_Button.performed += Teleport;
    mayMove = true;
}
```

Figuur 115: Update-methode van het TeleportScript

We lezen eerst in wat de waarde van de stick is. Als deze beweging kleiner is voor de x-waarden en y-waarden in de positieve of negatieve zin, zal de cirkel niet bewegen. moest dit wel zo zijn begint onze berekening. In het kort beweegt de cirkel voorwaarts of zijwaarts afhankelijk van hoe de camera gedraaid staat.

Eerst zoeken we de voorwaartse richting van de camera en normaliseren we deze waarde [63]. Als de waarde genormaliseerd is, telt deze slechts als een richting. Onze "movement"-variabele is de richting die daadwerkelijk zal uitgevoerd moeten worden. De "move"-variabele heeft een x-waarde die we vermenigvuldigen met de "transform.right" van de camera. Dit betekent dat deze x-waarde nu naar links of rechts gaat ten opzichte van de camera. Voor de y-waarde van de "move"-variabele gebruiken we de "cameraForward"-variabele. Deze wijst naar wat de voorwaartse richting is van de camera. Door de y-waarde eerder op 0 te zetten betekent dit ook dat we ons geen zorgen hoeven te maken dat de cirkel opeens de lucht in gaat of door de grond zal zakken. "movement" wordt genormaliseerd [64] zodat dit slechts een richting aanduid.

Aan het einde gebruiken we ook weer de methode "KeepHeight". Deze zorgt ervoor dat we bij een oneven grondvlak toch nog op de juiste hoogte zitten en de cirkel dan ook zichtbaar is.

```

1 reference
private void MoveCircle()
{
    move = headsetControls.XRILeftHand.Left_Thumbstick.ReadValue<Vector2>();
    if (move.x < -movementThreshold || move.x > movementThreshold || move.y < -movementThreshold || move.y > movementThreshold)
    {
        //we take the main camera
        Camera mainCamera = Camera.main;

        //we use the direction the camera is looking to
        Vector3 cameraForward = mainCamera.transform.forward;
        //we change the height to 0, we rectify this later in the code in KeepHeight()
        cameraForward.y = 0f;
        //we make it into a normalized vector3 to just point out a direction
        cameraForward.Normalize();

        //the x from the move-variable decides how much we move to the right, the y from the move-variable decides how much we move forward
        //negative results for x and y result in moving left and backwards
        Vector3 movement = (move.x * mainCamera.transform.right + move.y * cameraForward).normalized;
        // the resulted vector3 is the way we move the circle
        //teleportCircle.transform.Translate(movement * moveSpeed * Time.deltaTime, Space.World);
        teleportCircle.transform.position += movement * moveSpeed * Time.deltaTime;

        KeepHeight();
    }
}

```

Figuur 116: MoveCircle-methode in het TeleportScript

De “KeepHeight”-methode zorgt ervoor dat onze cirkel altijd over de vloer schuift. Eerste halen we de huidige positie van de “teleportCircle” op. Van hieruit doen we een “RayCast” [65]. Dit is een lijn die kan teruggeven wat er allemaal op zijn pad ligt. De methode geeft een true terug wanneer hij iets tegenkomt. Als eerste parameter gebruikt hij een Vector3 als positie van waar hij begint. We gebruiken hiervoor de positie van de “teleportCircle” met nog 10 meter extra in de y-waarde. Deze extra 10 is om ook op te merken wanneer de teleportCircle onder het grondvlak zit. Vervolgens maken gebruiken we een Vector3 als richting. Hier gebruiken we “Vector3.down” [66] om naar beneden te mikken met de neerwaartse richting van de worldspace. Hier volgt een variabele van de klasse “RaycastHit” [67] om bij te houden wat de eerste gevonden waarde is. Dan komt de lengte van de “RayCast”, hierbij koos ik voor 20, zo kunnen we 10 meter boven en onder de “teleportCircle”. Dit zou zeker voldoende moeten zijn. De laatste waarde zijn de layers die we in de Unity Editor hebben bevestigd. Zo zal de “RayCast” enkel de objecten checken die op deze layers zitten.

Moest er een hit worden geregistreerd zal deze worden opgeslagen in de “firstHit”-waarde. Als de hit niet overeenkomt met de huidige positie van de “teleportCircle” zal deze daarnaartoe worden verplaatst.

```

2 references
private void KeepHeight()
{
    Vector3 positionTeleport = teleportCircle.transform.position;

    RaycastHit firstHit;
    if(Physics.Raycast(positionTeleport + new Vector3(0, 10, 0), Vector3.down, out firstHit, 20, floorLayer))
    {
        if(teleportCircle.transform.position.y != firstHit.point.y)
        {
            teleportCircle.transform.position = firstHit.point;
        }
    }
}

```

Figuur 117: KeepHeight-methode in het TeleportScript

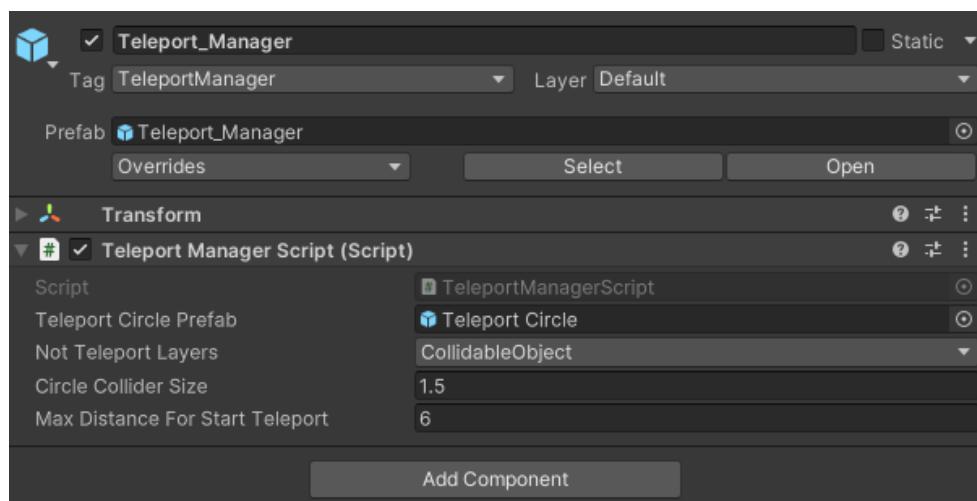
De "Teleport"-methode verplaatst de "XrRig" naar de huidige positie van de "teleportCircle". Vervolgens wordt het huidige object, de "teleportCircle" vernietigd. Hiervoor hebben we de "Destroy"-methode [68] . Op deze manier blijft de scène proper en kunnen we een nieuwe cirkel oproepen met behulp van het "TeleportManagerScript".

```
1 reference
private void Teleport(InputAction.CallbackContext context)
{
    xrOrigin.transform.position = teleportCircle.transform.position;
    Destroy(teleportCircle);
}
```

Figuur 118: Teleport-method in het TeleportScript

4.6.3. Teleport_Manager

Dit is gewoon een gameobject met daarin het "TeleportManagerScript". In de Unity Editor zijn hier ook weer wat variabelen die ingevuld kunnen worden. "Teleport Circle Prefab" is de prefab die we in onze assets-folder hebben staan, zie ook het titeltje hierboven. De "Not Teleport Layers" zijn de layers waar niet naar mag geteleporteerd worden. De manager mag hier dus ook geen "Teleport Circle" op instantiëren. Voor te bepalen waar de prefab wordt geïnstantieerd hebben we ook een radius nodig waarin niets mag staan dat in de weg staat, dit is de "Circle Collider Size". "Max Distance For Start Teleport" is vrij duidelijk, dit is de afstand vanaf de camera waar de prefab zal worden aangemaakt als dit mogelijk is.



Figuur 119: Teleport_Manager in de inspector

De globale variabelen hebben deze keer niets nieuws. Ook wordt er deze keer gebruik gemaakt van de "HeadsetControls"-klasse. Deze zullen dan ook worden geenabled en gedisabled met de bijhorende methoden die Unity aanbied.

```

[Serializable]
private GameObject teleportCirclePrefab;
[Serializable]
private LayerMask notTeleportLayers;
[Serializable]
private float circleColliderSize = 1.5f;
[Serializable]
private int maxDistanceForStartTeleport = 6;

HeadsetControls headsetControls;

```

Figuur 120: globale variabelen in het TeleportManagerScript

Deze keer staat er niet veel in de “Update”-methode. Er wordt alleen gecheckt of de knop voor het activeren van de “Teleport Circle” wordt ingedrukt.

```

0 references
void Update()
{
    headsetControls.XRILeftHand.Left_Secondary_Button.performed += CreateCircle;
}

```

Figuur 121: Update-methode in het TeleportManagerScript

Het eerste dat we checken is of het “HandMenu” actief is of niet. Wanneer dit niet actief is in de hiërarchie zullen “Find”-methodes van de “GameObject”-klasse null teruggeven. De reden dat we dit doen is omdat de knop om het “HandMenu” op te roepen dezelfde is als deze om de “Teleport”-methode uit te voeren in het “TeleportScript”. Om te voorkomen dat deze tegelijk kunnen uitgevoerd worden creëren we enkel een “Teleport Circle” als er geen “HandMenu” is. Ook wanneer er al een “Teleport Circle” aanwezig is maken we geen nieuwe aan in de huidige scene. Dan vernietigen we de “Teleport Circle” die al in de scene aanwezig is. Zo kun je de “Teleport Circle” oproepen en verwijderen met slechts één knop.

Om de onze prefab toch aan te maken, moeten we eerst checken of er de ruimte voor is. We gebruiken hiervoor een loop die begint bij de “maxDistanceForStartTeleport” en doorgaat tot de afstand tot de camera 0 is. Dat zou willen zeggen dat wanneer de gebruiker kijkt in een richting waar op geen enkele afstand kan geteleporteerd worden, de locatie vlak onder de gebruiker is waar de cirkel zal verschijnen.

Om de locatie op die afstand te checken gebruiken we een “OverlapSphere” [69]. Het idee hierachter is ongeveer hetzelfde als de “RayCast”. Behalve dat deze geen lijn maakt vanuit de camera, maar op één locatie een bol, een “sphere”, maakt die checkt naar alles dat met deze bol in contact komt. Voor de positie van de bol kiezen we een punt op de huidige afstand in de loop, vermenigvuldigt met de voorwaartse richting van de camera. We geven de “OverlapSphere” de layers mee waar we niet mogen belanden met behulp van de “notTeleportLayers”-variabele. Zo krijgen we een lijst die alleen objecten zal bevatten als deze er zijn.

Als deze lijst leeg is maken we een nieuwe “Teleport Circle” aan op de gewenste positie met de “Instantiate”-methode [70] . We voeren ook een break aan om uit de loop te gaan. Zo maken we niet per ongeluk meerdere instanties van de “Teleport Circle”.

```
1 reference
private void CreateCircle(InputAction.CallbackContext context)
{
    GameObject handMenu = GameObject.FindGameObjectWithTag("HandMenu");
    if (handMenu == null)
    {
        GameObject currentTeleport = GameObject.FindWithTag("TeleportCircle");
        if (currentTeleport != null)
        {
            Destroy(currentTeleport);
        }
        else
        {
            int maxDistanceTeleport = maxDistanceForStartTeleport;

            for (int i = maxDistanceTeleport; i >= 0; i--)
            {
                Vector3 cameraPosition = Camera.main.transform.TransformPoint(Vector3.forward * i);
                Collider[] allColliders = Physics.OverlapSphere(cameraPosition, circleColliderSize, notTeleportLayers);
                if (allColliders.Length <= 0)
                {
                    Instantiate(teleportCirclePrefab, new Vector3(cameraPosition.x, 0, cameraPosition.z), new Quaternion());
                    break;
                }
            }
        }
    }
}
```

Figuur 122: CreateCircle-methode in het TeleportManagerScript

4.7. Activiteit management

Tijdens het spelen van de game zijn er bepaalde activiteiten waarbij de onderdelen reageren op de hartslag van de gebruiker. Om deze te kunnen managen, staren en beëindigen heb ik een “Activity_Manager” gemaakt. Hiermee is het mogelijk om een activiteit te starten door op een knop te duwen.

Doorheen dit segment zult u vaak het woord “New” zien staan in de naamgeving van scripts en objecten. Dit is omdat er een aanpassing nodig was voor hoe de activiteiten werden gemanaged, terwijl mijn collega nog verder werkte met de oude variant. Uiteindelijk is alles uit de oude variant terug gebruikt in de nieuwe versie van het product. Helaas is er door tijdsgebrek nooit ruimte geweest om de naamgeving hiervan aan te passen.

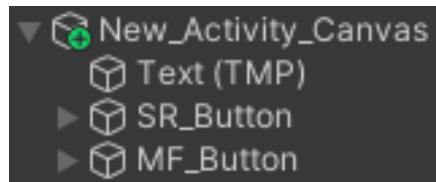
4.7.1. Activiteit starten

Een activiteit kan gestart worden door op een knop te duwen kunnen we een activiteit starten. Hierbij zijn er twee opties: “SR” en “MF”. Deze staan voor “SpierRelaxatie” en “MindFulness” en verwijzen naar het type ingesproken begeleiding er bij de activiteit zal plaatsvinden om de gebruiker te ondersteunen om rustig te worden. Deze zijn elks ongeveer 15 minuten lang. Ik heb het geluid van de geluidsfragmenten ook nog zelf moeten boosten met Audacity [71] om ze hoorbaar te maken in de app.



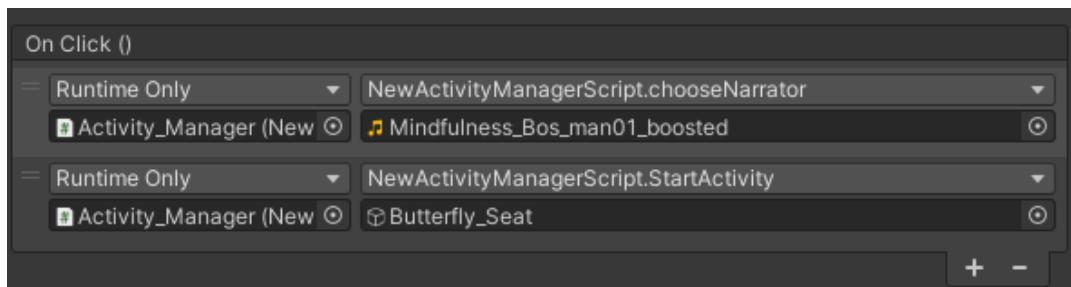
Figuur 123: bord om vlinder-activiteit te starten

Het activiteit canvas is niet bijzonder gevuld, slechts een "Text"-object en twee knoppen. Elke knop maakt gebruik van twee methoden. Dit is zo omdat de "Onclick"-array van een knop alleen maar methoden met maximaal één parameter accepteert.



Figuur 124: activiteit canvas

De eerste methode zorgt ervoor dat de juiste "AudioClip" wordt doorgegeven. De tweede methode heeft een gameobject nodig dat gebruikt wordt om te herkennen welke activiteit gestart werd. Hierbij is het de "Butterfly_seat", de plaats waar de gebruiker naar zal geteleporteerd worden als de vlinder-activiteit begint.



Figuur 125: methoden in de OnClick-array van de MF_Button

4.7.2. Activity_Manager

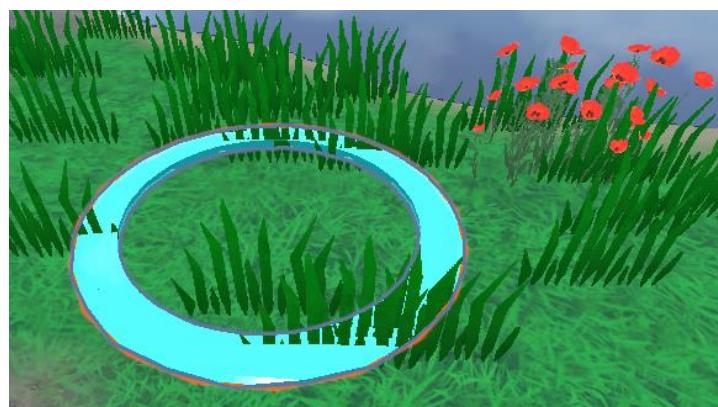
De "Activity_Manager" is een gameobject met slechts één script: het "NewActivityManagerScript". Deze manager hoeft alleen maar gebruikt te worden in een omgeving waar activiteiten worden uitgevoerd. Er is hier ook een eigen klasse om de activiteiten te managen: de "NewManagedActivity"-script.

4.7.2.1. NewManagedActivity

De eerste variabele is "visibleCircle". Om aan te tonen waar de een activiteit plaatsvindt hebben we een zichtbare cirkel geplaatst. Tijdens de activiteit zal deze worden weggehaald zodat die de gebruiker niet uit de belevening kan halen. De "Seat" is op de plaats van de zichtbare cirkel, alleen is dit object onzichtbaar. Wanneer een gebruiker een activiteit verlaat moet hij gewoon met de camera weggaan van de activiteit. Wanneer de "seat" niet meer wordt gevonden onder de camera, vermoeden we dat de gebruiker weggaat en eindigt de activiteit. "activityScript" is het script dat de daadwerkelijke activiteit bevat. Net zoals de meeste code die ik eerder in dit document beschreef is dit "activityScript" een klasse die overerft van "MonoBehaviour" [72]. Wanneer we het script aanzetten begint de activiteit. Wanneer het script wordt afgezet zal deze eindigen. De "heightFromSeat" bepaalt hoe hoog we staan vanaf de "seat". Op deze manier kunnen we simuleren dat iemand ergens zit of gehurkt staat bij een bosje bloemen.

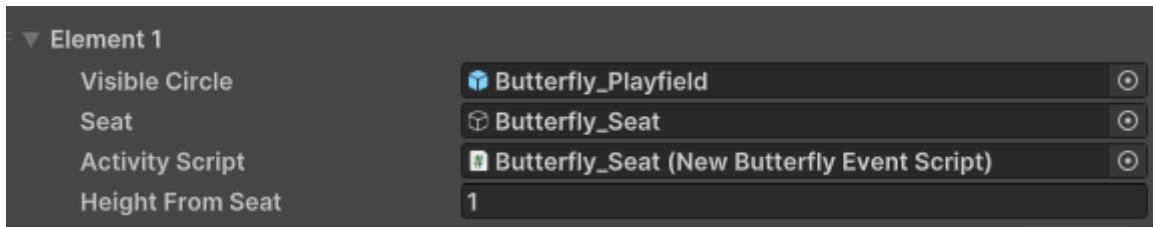
```
[Serializable]
6 references
public class NewManagedActivity
{
    public GameObject visibleCircle;
    public GameObject seat;
    public MonoBehaviour activityScript;
    public float heightFromSeat;
}
```

Figuur 126: NewManagedActivity-klasse



Figuur 127: zichtbare cirkel bij de vlinder-activiteit

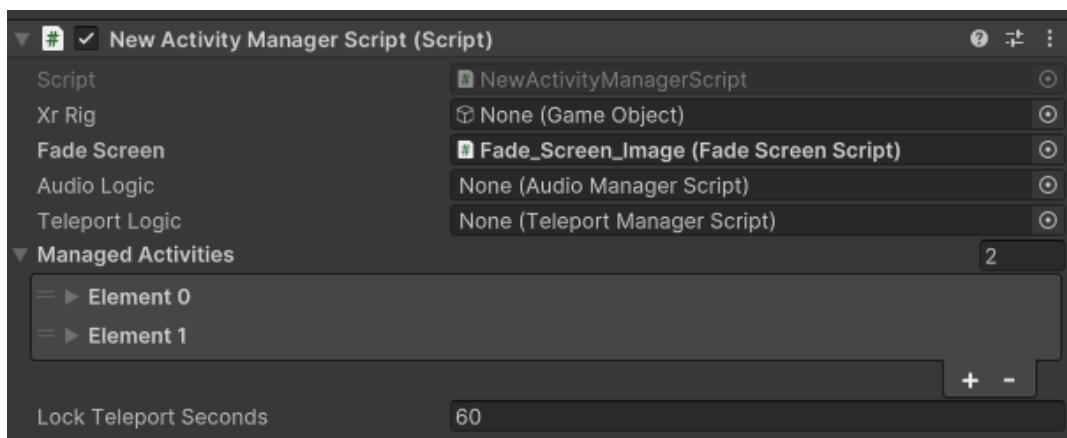
In de Unity editor kunnen we bij in een "NewActivityManagerScript" de klasse gebruiken om meerdere activiteiten toe te voegen die we dan kunnen starten met een druk op de knop.



Figuur 128: voorbeeld NewManagedActivity-klasse in de Unity Editor

4.7.2.2. Unity Editor

In de editor kunnen we in het “NewActivityManagerScript” meerdere zaken invullen. Om te beginnen is er de “Xr Rig”, deze wordt ook opgehaald in de code. De “Xr Rig” is nodig om de gebruiker te verplaatsen voor een activiteit begint. Het “Fade Screen” en de andere twee managers worden bij de code uitgelegd. De “Managed Activities” is een lijst van alle activiteiten die binnen deze scene kunnen worden uitgevoerd. Op het einde staat er nog een “Lock Teleport Seconds”, deze zorgt ervoor dat de gebruiker niet meteen kan vertrekken als ze een activiteit starten. Zo kunnen ze enigszins nog de activiteit ervaren voordat ze kunnen vertrekken.



Figuur 129: NewActivityManagerScript in de Activity_Manager in de inspector

4.7.2.3. Code

De variabelen in de Unity Editor hebben geen verdere uitleg nodig. Daarnaast zijn er nog wel enkele anderen. De “narratorClip” is de clip die we zullen doorgeven aan de “Narrator” om de gebruiker te begeleiden tijdens de activiteit.

“currentManagedActivity” is de activiteit die bezig is op het huidige moment. Wanneer er geen activiteit bezig is zal deze dan ook “null” zijn. De “_ActivityIsActive” mag alleen in dit script worden aangepast, zodoende heeft deze dan ook een getter en setter. Als extra hebben we hier ook de setup voor een listener. Wanneer de waarde in de setter wordt aangepast naar een nieuwe waarde en het gekoppelde event is niet leeg geven we dit door naar alle methodes die luisteren. In dit geval is dit er één in het “HandMenuScript”. Het idee hiervoor vond ik in een antwoord op een vraag op het Unity Forum [73] . Het gebruik van de listener maakt het mogelijk om de verandering door te voeren tijdens dat het “HandMenu” openstaat of wanneer dit juist wordt opengedaan.

```

private AudioClip narratorClip;
private NewManagedActivity currentManagedActivity;

private bool _ActivityIsActive = false;

5 references
public bool ...activityIsActive
{
    get { return _ActivityIsActive; }
    private set
    {
        if (_ActivityIsActive == value) return;
        _ActivityIsActive = value;
        if (OnActivityIsActiveChange != null) OnActivityIsActiveChange(_ActivityIsActive);
    }
}

public delegate void OnActivityIsActiveDelegate(bool value);
public event OnActivityIsActiveDelegate OnActivityIsActiveChange;

```

Figuur 130: globale variabelen in het NewActivityManagerScript

In de "Awake"-methode zetten we alle scripts op inactief. Omdat dit gebeurt voor de "OnEnable"-methode in de hiërarchie van methoden, worden er dus nog geen activiteiten gestart. De activiteiten beginnen namelijk in hun "OnEnable"-methode zodat ze afgezet kunnen worden en daarna terug opgestart moet dit gewenst zijn. Om een activiteit af te ronden zal de code voor het afronden van een activiteit in de "OnDisable"-methode van het bijhorende script zitten.

```

➊ Unity Message | 0 references
private void Awake()
{
    //to ensure all scripts are disabled before they can do something
    foreach (NewManagedActivity ma in managedActivities)
    {
        ma.activityScript.enabled = false;
    }
}

```

Figuur 131: Awake-methode in het NewActivityManagerScript

In de "Start"-methode worden alleen alle managers en de scripts ingevuld als deze nog leeg zijn, met behulp van hun tags.

De "ChooseNarrator"-methode zorgt ervoor dat de "narratorClip" wordt ingevuld. Zoals eerder vermeld is dit een workaround voor het feit dat de "OnClick"-methoden van een knop slechts een variabele kunnen meekrijgen.

```

0 references
public void chooseNarrator(AudioClip audioClip)
{
    //setting the clip globally so we can end it when the activity ends
    //separate method as workaround for 1 parameter we can use in the unity editor in an OnClick-method
    narratorClip = audioClip;
}

```

Figuur 132: ChooseNarrator-methode van het NewActivityManagerScript

Wanneer we de activiteit starten met een knop geven we de "seat" van de "ManagedActivity" mee. Dan vinden we in de "managedActivities"-array de geselecteerde activiteit en starten deze op met de "StartActivityRoutine". Maar daarvoor zetten we eerst het geluid van de omgeving af over de tijd dat een fade gebeurt. Zo kan tijdens een activiteit ander geluid naar de voorgrond worden geduwd. Ook is er de "LockTeleportRoutine" om het teleporteren te vermijden voor de voorziene tijd.

```
0 references
public void StartActivity(GameObject seat)
{
    if (!activityIsActive)
    {
        foreach (NewManagedActivity managed in managedActivities)
        {
            if (managed.seat == seat)
            {
                currentManagedActivity = managed;
                break;
            }
        }
        //silence environment for certain activities
        audioLogic.changeVolumeNarrator(0, fadeScreen.fadeDuration, true, false);
        //make sure users can first enjoy the experience for a chosen time
        StartCoroutine(lockTeleportRoutine(lockTeleportSeconds));
        //start the activity
        StartCoroutine(StartActivityRoutine(currentManagedActivity));
    }
}
```

Figuur 133: StartActivity-methode in het NewActivityManagerScript

Deze "Coroutine" loopt terwijl een activiteit bezig is. Eerst wordt de "Teleport Circle" verwijderd als deze per ongeluk nog in de scene zou staan. Daarna checken we dat de manager voor het teleporteren zeker bestaat. Wanneer deze bestaat, disablen we hem. Na de aangewezen tijd te wachten, wordt de manager weer geenableed.

```
1 reference
IEnumerator lockTeleportRoutine(int secondsLocked)
{
    GameObject currentTeleport = GameObject.FindWithTag("TeleportCircle");
    if (currentTeleport != null)
    {
        Destroy(currentTeleport);
    }

    if (teleportLogic != null)
    {
        teleportLogic.enabled = false;
        yield return new WaitForSeconds(secondsLocked);
        teleportLogic.enabled = true;
    }
}
```

Figuur 134: LockTeleportRoutine-Coroutine in het ?-NewActivityManagerScript

We beginnen met een fade om de gebruiker niet meteen te doen verschieten wanneer de activiteit begint en deze zich op een andere locatie bevindt. Wanneer de fade het donker heeft gemaakt verplaatsen we de gebruiker met de "PlacePlayer"-methode. We maken de ring ook onzichtbaar door de "MeshRenderer" [74] af te zetten. Dan wordt is het zichtbare van de cirkel verborgen. De "activityIsActive" waarde wordt true en

waarschuwt zo ook het “HandMenu”. Dan wordt het script van de activiteit ook aangezet. Met “UseNarrator” geven we de “narratorClip” aan de “Narrator” die deze dan zal starten als er niets anders meer speelt. Normaalgezien zou er niets mogen zijn dat speelt buiten de activiteiten, maar in de toekomst zou dit kunnen toegevoegd worden. in dat geval staat dit al klaar. We wachten nog 1 seconde om de fade niet te snel voorbij te laten gaan en zodat alles zeker klaar staat. Als laatste voeren we dan de fadein uit zodat we weer alles kunnen zien.

```
1 reference
IEnumerator StartActivityRoutine(NewManagedActivity activity)
{
    fadeScreen.FadeOut();
    yield return new WaitForSeconds(fadeScreen.fadeDuration);

    PlacePlayer(activity);
    currentManagedActivity.visibleCircle.GetComponent<MeshRenderer>().enabled = false;
    activityIsActive = true;
    activity.activityScript.enabled = true;
    audioLogic.useNarrator(narratorClip, true);

    //wait a bit in the dark to avoid to allow audio to change
    yield return new WaitForSeconds(1f);

    fadeScreen.FadeIn();
}
```

Figuur 135: StartActivityRoutine-Coroutine in het NewActivityManagerScript

Voordat we de gebruiker kunnen verplaatsen verzamelen we alle “Transforms” die we nodig hebben. Deze zijn voor de “XrRig”, de camera en de “seat” van de activiteit. Omdat het mogelijk is dat de camera niet perfect in het midden staat van de “XrRig”, omdat de gebruiker bijvoorbeeld niet calibreert. Moeten we de verandering berekenen. We bekijken eerst wat de “offset” is tussen de positie van de “camera” en de “XrRig”. We verminderen dan de y-waarde met de “heightFromSeat”, zo zal de hoogte zijn zoals gewenst. We verplaatsen ons dan naar de positie van onze target met daarvan de “offset” die we juist hebben berekend afgetrokken. Zo staat de “XrRig” niet exact op de locatie van de “seat”, maar de camera wel. En dat is de bedoeling.

Daarna moet de camera nog de juiste kant uitgedraaid staan. We gebruiken de forward van de camera en deze van de “seat”. Daartussen berekenen we dan weer de hoek met de “SignedAngle”-methode. Vervolgens draaien we de “XrRig” dan weer om de positie van de camera.

Het idee voor deze manier van verplaatsing komt van een tutorial [75] die ik online had gevonden toen ik bezig was met teleportatie. Ik heb de code hiervoor gebruikt en dan aangepast naar wat er voor dit project nodig was.

```

1 reference
private void PlacePlayer(NewManagedActivity activity)
{
    Transform origin = xrRig.transform;
    Transform camera = Camera.main.transform;
    Transform target = currentManagedActivity.seat.transform;

    Vector3 offset = camera.position - origin.position;
    //this makes it so we ourselves can arrange the height we are placed on our seat
    offset.y -= currentManagedActivity.heightFromSeat;
    origin.position = target.position - offset;

    Vector3 targetDirection = target.forward;
    targetDirection.y = 0;
    Vector3 cameraDirection = camera.forward;
    cameraDirection.y = 0;

    float angle = Vector3.SignedAngle(cameraDirection, targetDirection, Vector3.up);
    origin.RotateAround(camera.position, Vector3.up, angle);
}

```

Figuur 136: PlacePlayer-methode in het NewActivityManagerScript

Deze methode beëindigt een activiteit. Deze zal opgeroepen worden vanuit een script dat wordt uitgelegd onder het volgende titeltje.

Wanneer de activiteit beëindigt is zetten we de “activityIsActive” op false, dit waarschuwt ook het “HandMenu”. Daarna wordt het “activityScript” gedisabled, dit start dan de code in de “OnDisable”-methode van dit script. We schakelen ook de “Narrator” uit met de methode uit de “Audio_Manager”. De cirkel is ook weer te zien, zo weten we ook weer waar de activiteit zich zal afspelen. Om alles af te ronden wordt de “currentManagedActivity” null, zodat die later weer kan ingevuld worden met een nieuwe activiteit.

```

1 reference
public void EndActivity()
{
    activityIsActive = false;
    currentManagedActivity.activityScript.enabled = false;
    audioLogic.changeVolumeNarrator(0, 4, true, true);
    currentManagedActivity.visibleCircle.GetComponent<MeshRenderer>().enabled = true;
    currentManagedActivity = null;
}

```

Figuur 137: EndActivity-methode in het NewActivityManagerScript

4.7.3. Activiteit beëindigen

We maken gebruik van een onzichtbaar gameobject dat onder de camera hangt om te zien of een activiteit bezig is of niet. Wanneer deze bezig is kijken we of we boven de “seat” hangen, wanneer dit niet meer het geval is beëindigen we de activiteit.



Figuur 138: Seat van de vlinder-activiteit zonder zichtbare cirkel

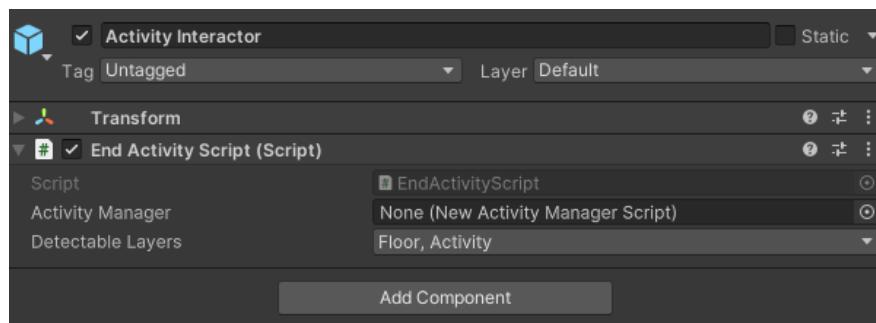
4.7.3.1. Unity Editor

In de hiërarchie zit er onder de camera die we gebruiken een "Activity Interactor". Deze heeft een script dat de activiteit kan stoppen wanneer we de "seat" verlaten.



Figuur 139: Activity Interactor onder de camera

Het script dat we gebruiken in de "Activity Interactor" heeft als naam "EndActivityScript". Deze heeft slechts een doel en dat is de activiteit stoppen wanneer de "seat" wordt verlaten. We kunnen hier de "Activity Manager" in slepen, ook deze wordt in de code opgehaald. Daarnaast kunnen we laten weten welke layers er mogen gedetecteerd worden. Deze layers zijn nodig voor de "Raycast" die zal gebeuren.



Figuur 140: Activity Interactor in de inspector

4.7.3.2. Code

De variabelen die we zien in de Unity Editor zijn de enige globale variabelen die aanwezig zijn. Deze zijn ook weer private met een [SerializeField]-attribuut. In de "Start"-methode vullen we enkel de "activityManager"-variabele op als deze nog leeg is.

Alles in dit script gebeurt in de "Update"-methode. Hier kijken we elke frame of er een activiteit bezig is of niet. Hier is geen nood aan een listener omdat we toch elke frame de check doen. Wanneer de activiteit bezig is gaan we een "Raycast" uitvoeren. Deze start van de positie van het huidige object, dat altijd onder de camera hangt. De richting is naar beneden door de Vector3(0, -1, 0). Dit is gelijk aan een negatieve y-waarde in de worldspace en wijst dus naar beneden, daardoor gaat de ray ook naar beneden. We slaan het eerste object dat we vinden op in een "RaycastHit"-variabele. We laten de ray doorgaan voor 20 meter en gebruiken alleen de layers die hebben ingegeven. Wanneer het object in kwestie als tag "Seat" heeft weten we dat we boven de "seat" hangen. Wanneer dit dan een andere tag heeft, zoals bijvoorbeeld alles op de "floor"-layer dan zal dit geregistreerd worden en de "EndActivity"-methode uitvoeren.

Als we alle layers zouden laten gebruiken zou de performance van de app hieronder kunnen leiden. Maar als er geen layer is die kan gedetecteerd kan worden zal de "RayCast" niks teruggeven en kunnen we ook niet de "EndActivity"-methode laten afgaan. Dan zou een activiteit ook niet kunnen stoppen. Daarom detecteren we de "activity"-layer en de "floor"-layer

```
// Update is called once per frame
@Unity Message | 0 references
void Update()
{
    if(activityManager.activityIsActive)
    {
        RaycastHit hit;
        if (Physics.Raycast(transform.position, new Vector3(0, -1), out hit, 20, detectableLayers))
        {
            if (hit.collider.tag != "Seat")
            {
                activityManager.EndActivity();
            }
        }
    }
}
```

Figuur 141: Update-methode in het EndActivityScript

5. Omnicept Data

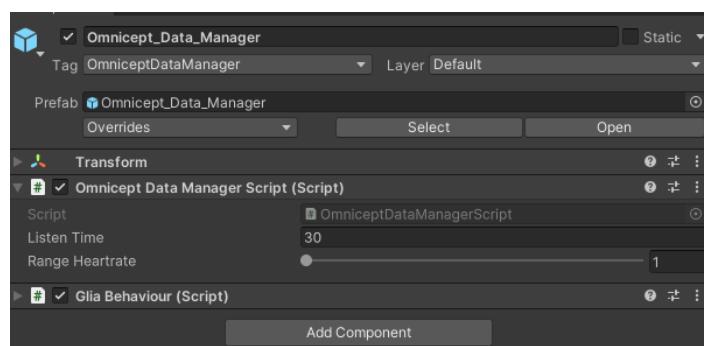
Dit onderdeel onderscheidde deze opdracht van een normale vr-app, met name de biofeedback die de Omnicept ons kan leveren. Tijdens de onderzoekfase hadden we de verschillende data bekeken. Hierbij was er eerst aandacht voor welke data we konden krijgen en dan vervolgens welke we het best konden gebruiken. Uit dit onderzoek is gekomen dat we best de hartslag gebruiken. Als ooit de toegang tot de PPG-data beschikbaar is, kan de code worden geüpdatet om deze te gebruiken. In dit onderdeel bespreken we hoe we de data hebben opgehaald, bruikbaar hebben gemaakt en uiteindelijk gebruikt.

5.1. Data Ophalen

Voor het ophalen van de data maken we gebruik van twee scripts: "GliaBehaviour" en het OmniceptDataManagerScript. Deze zijn verzameld onder de "Omnicept_Data_Manager". "GliaBehaviour" is een script dat in het Unity Package zat in de Omnicept SDK. Dit script zorgt voor het directe contact tussen de headset en een Unity-applicatie. We hebben dit dus nodig om de data op te halen. in het "OmniceptDataManagerScript" roepen we dit script op. Het "OmniceptDataManagerScript" is een tussenstap voor onze app om de data weg te schrijven, te bekijken en in de toekomst nog andere bewerkingen mee te doen. Het is mogelijk om de data direct uit een "GliaBehaviour"-script op te halen, maar dan zouden bepaalde berekeningen opnieuw moeten gemaakt worden in elk script dat deze data wil gebruiken.

5.1.1. Unity Editor

In de inspector kunnen we zien dat de "Omnicept_Data_Manager" het script voor "GliaBehaviour" en het "OmniceptDataManagerScript" als componenten heeft. In "GliaBehaviour" moet niets worden aangepast. Voor ons eigen script zijn er wel enkele variabelen die we kunnen aanpassen. Deze hebben geen invloed op het ophalen van de data, maar zijn later nodig voor het manipuleren van de data. "Listen Time" is een tijd dat ons script luistert naar de hartslag om een gemiddelde te berekenen. "Range Heartrate" is een range rond de berekende gemiddelde hartslag die nodig is voor de activiteiten die we later uitleggen.



Figuur 142: Omnicept_Data_Manager in de inspector

5.1.2. Code

De code om “GliaBehaviour” te gebruiken in een ander script wordt uitgelegd op de site van de Omnicept zelf [76] . Deze code is toegevoegd aan het “OmniceptDataManagerScript” en zou normaalgezien nergens anders moeten gebruikt worden.

Door de manier waarop deze “gliaBehaviour” wordt opgehaald, zal deze worden ingevuld de eerste keer dat deze wordt gebruikt. HP gebruikt de “FindObjectOfType”-methode. Doordat bij ons het “OmniceptDataManagerScript” en “GliaBehaviour” beiden componenten zijn van hetzelfde gameobject kunnen we ook “GetComponent” gebruiken. In werking is hier geen verschil tussen, maar het is wel een interessante manier om te denken over de initiële intentie voor het gebruik en hoe wij het uiteindelijk hebben gebruikt.



```
private GliaBehaviour _gliabehaviour = null;

11 references
public GliaBehaviour gliaBehaviour
{
    get
    {
        if (_gliabehaviour == null)
        {
            _gliabehaviour = FindObjectOfType<GliaBehaviour>();
        }
        return _gliabehaviour;
    }
}
```

Figuur 143: gliaBehaviour toevoegen aan een script

Om de data vervolgens op te halen zijn er twee manieren. De eerste is door de “GetLast”-methoden, deze halen de laatst gemeten data op. De tweede is een stuk interessanter, deze werkt met een listener. Zo kunnen we bepaalde data pas wegschrijven of aanpassen als de Omnicept deze echt teruggeeft.

Het gebruik van listeners zal ook een stuk interessanter zijn wanneer er gewerkt kan worden met PPG-data. PPG-data wordt heel snel gemeten en geeft dan ook meerdere resultaten en schnellere veranderingen weer. De hartslag wordt berekend tijdens de eerste 20 seconden en geeft dan ook niets terug. Daardoor is er aan het begin van de applicatie elke keer een leegte van 20 seconden waarin we de data niet kunnen omschrijven of gebruiken. Verder wordt de hartslag van de headset na de eerste 20 seconden elke 5 seconden pas teruggegeven.

```
float currentHeartRate = gliaBehaviour.GetLastHeartRate().Rate;
gliaBehaviour.OnHeartRate.AddListener(SetHeartRate);
```

Figuur 144: wijzen om data van gliaBehaviour op te halen

5.2. Data Manipuleren

De variabelen die we zagen in de Unity Editor zijn van het format: private met [SerializeField]-attribuut. Verder hebben we ook een "timer", deze keer zal de timer worden gebruikt in de "Update"-methode en daarom is deze globaal. Daarbij hebben we ook een lijst voor het bijhouden van de hartslag-metingen die plaatsvinden tijdens de "listenTime"-periode die we eerder invulden. De "fileName" is de naam van het CSV-bestand waarin we de hartslag wegschrijven. Dit wegschrijven van data was op verzoek van onze mentoren. Zo kan men achteraf bekijken hoe een "speelsessie" verliep voor een gebruiker.

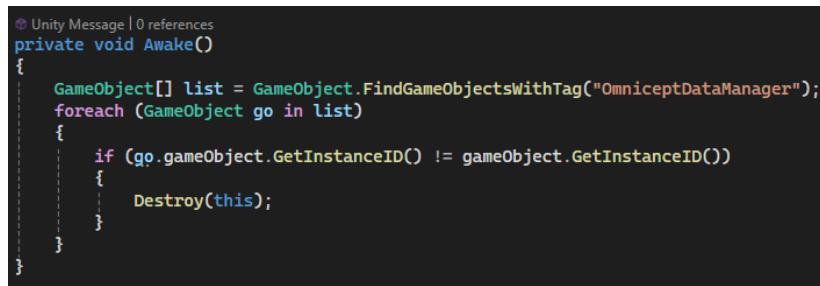
Vervolgens zijn er enkele variabelen die getters en setters hebben. "timerCounts" is true zolang de gemiddelde hartslag nog niet berekend is en de "timer"-variabele nog optelt. Deze heeft dan ook een "OnChangeEvent" hier aan hangen, op dezelfde manier als we eerder hebben gezien. Deze is ook voor het "HandMenu" zodat men niet per ongeluk op de "reset"-knop zou duwen als deze de gemiddelde hartslag nog aan het berekenen is. Het herstarten van de berekening heeft geen effect op de berekening, maar maakt wel elke keer een nieuw CSV-bestand aan. "currentHeartRate" en "averageHeartRate" hebben getters en setters omdat ze in andere scripts moeten worden opgeroepen, maar daar niet van waarde mogen veranderen. Daarvoor werken we weer met een "private" setter.

```
private float timer = 0;
private List<float> averageList = new List<float>();
private string filename;

private bool _TimerCounts = false;
private float _AverageHeartRate;
private float _CurrentHeartRate;
18 references
public float currentHeartRate...
6 references
public float averageHeartRate...
6 references
public bool timerCounts
{
    get { return _TimerCounts; }
    private set
    {
        if (_TimerCounts == value) return;
        _TimerCounts = value;
        if (OnTimerCountsChange != null) OnTimerCountsChange(_TimerCounts);
    }
}
public delegate void OnTimerCountsChangeDelegate(bool value);
public event OnTimerCountsChangeDelegate OnTimerCountsChange;
```

Figuur 145: globale variabelen in het OmniceptDataManagerScript

In deze "Awake"-methode zorgen we ervoor dat er slechts een instantie bestaat van de "Omnicpt_Data_Manager". Deze moet namelijk blijven bestaan tussen scenes door, zodat de data blijft gemeten worden doorheen de hele applicatie. Later leggen we uit hoe deze blijft bestaan tussen meerdere scenes. De code in de "Awake"-methode vernietigt namelijk het huidige gameobject als er al een "Omnicpt_Data_Manager" is. Hiervoor gebruiken we de "instanceID" [77] van een object. Als een "Omnicpt_Data_Manager" wordt aangemaakt in een scene zoekt deze in de hiërarchie voor alle gameobjecten met de tag "OmnicptDataManager". Hiervoor gebruiken we de method "FindGameObjectsWithTag" [78] om een lijst op te halen in plaats van slechts één gameobject. Bij deze kijkt hij dan of er een inzit die niet dezelfde "instanceID" heeft als het huidige object. Wanneer dit zo is wordt het huidige gameobject vernietigd, omdat deze pas later kwam. Door deze code in de "Awake"-methode te plaatsen zal die slechts eenmaal worden uitgevoerd. Daarbovenop zal de rest van de code niet worden uitgevoerd, omdat deze later in de hiërarchie van Unity-methoden zit.



Figuur 146: Awake-methode in het OmnicptDataManagerScript

Als het script wordt geenabled. Zetten we eerst de "timerCounts" op false. Zo zal de "timer" pas beginnen te tellen wanneer we de eerste hartslag terugkrijgen. Vervolgens doen we een check of de folder waar we alle CSV-bestanden opslaan bestaat, deze wordt aangemaakt als dit niet zo is. Daarna bepalen we de naam van het bestand. Deze moet staan in de folder die we juist aanmaakten. We maken gebruik van het tijdstip zodat op een dag meerdere bestanden kunnen geschreven worden. We eindigen de naam met "biofeedback" voor verduidelijking. De extensie is ".csv" zodat het een CSV-bestand wordt.

Aan het einde zetten we ook drie listeners. De eerste zal de "currentHeartrate" aanpassen naar de laatst gemeten hartslag. De tweede schrijft de hartslag weg in het bestand. En de derde start de berekening van de gemiddelde hartslag.

```

    @ Unity Message | 0 references
private void OnEnable()
{
    timerCounts = false;

    //this creates a directory if the directory doesn't exist
    string directory = @"C:\MobilabAppResults\";
    Directory.CreateDirectory(directory);

    filename = directory + DateTime.Now.ToString("dd_MM_yyyy-HH-mm-ss") + "_biofeedback.csv";

    gliaBehaviour.OnHeartRate.AddListener(SetHeartRate);
    gliaBehaviour.OnHeartRate.AddListener(WriteHeartRate);
    gliaBehaviour.OnHeartRate.AddListener(StartCalculatingAverage);
}

```

Figuur 147: *OnEnable*-methode in het *OmniceptDataManagerScript*

In de “OnDisable”-methode verwijderen we alle listeners, zo zouden we errors kunnen voorkomen. De “AddDataToAverageList” staat laatst omdat deze stopt als de “timer” klaar is met tellen. Deze kan dus al verwijderd zijn en dan ontstaat er een error. Deze error vangen we ook op met de try-catch.

```

    @ Unity Message | 0 references
private void OnDisable()
{
    try
    {
        gliaBehaviour.OnHeartRate.RemoveListener(SetHeartRate);
        gliaBehaviour.OnHeartRate.RemoveListener(WriteHeartRate);
        gliaBehaviour.OnHeartRate.RemoveListener(AddDataToAverageList);
    }
    catch (System.Exception e)
    {
        Debug.Log($"The listeners couldn't be found: {e}");
    }
}

```

Figuur 148: *OnDisable*-methode van het *OmniceptDataManagerScript*

We gebruiken de “DontDestroyOnLoad”-methode [78] om te voorkomen dat het huidige gameobject wordt verwijderd wanneer een nieuwe scene wordt ingeladen. Omdat “Start” na “Awake” komt zal de notie dat het huidige object niet mag vernietigd worden niet bestaan voordat we checken of ze moet vernietigd worden.

```

    @ Unity Message | 0 references
private void Start()
{
    DontDestroyOnLoad(this);
}

```

Figuur 149: *Start*-methode in het *OmniceptDataManagerScript*

In de “Update”-methode stijgt de “timer”. Maar alleen als er nog moet geteld worden volgens “timerCounts”. “timerCounts” stopt dus ook onnodige bewerkingen, naast het feit dat deze het “HandMenu” op de hoogte houdt.

```

Unity Message | 0 references
void Update()
{
    if (timerCounts)
    {
        TimerCounting();
    }
}

```

Figuur 150: Update-methode in het OmniceptDataManagerScript

De methode om “currentHeartRate” aan te passen past deze alleen aan wanneer deze een aanpassing merkt vanuit de “gliaBehaviour”. De parameter die we hier gebruiken is wat het “OnHeartRate”-event ons meegeeft.

```

2 references
private void SetHeartRate(HeartRate heart)
{
    currentHeartRate = heart.Rate;
}

```

Figuur 151: SetHeartRate-methode voor het OnHearrate-event in het OmniceptDataManagerScript

Met de volgende code schrijven we de hartslag weg. Als het bestand nog niet bestaat, maken we deze aan een voegen we kolomnamen toe. Deze kolomnamen zijn ter verduidelijking van wat er wordt weggeschreven. “Scene” is de naam van de scene, “Time” is het tijdstip dat het werd gemeten en “Hearrate” is de waarde in BPM.

De methode die we hier gebruiken hebben we ook zelf geschreven. De eerste parameter bepaald of de tekst bovenaan de CSV moet geschreven worden. Bij een false-waarde wordt de meting onderaan geschreven. Vervolgens gebruiken we een array van het datatype string. Hier zetten we de tekst in die moet geschreven worden, deze wordt dan, gescheiden door ‘;’, weggeschreven.

```

2 references
private void WriteHeartRate(HeartRate heartRate)
{
    if (!File.Exists(filename))
    {
        WriteRowToCsv(true, new string[] { "Scene", "Time", "Hearrate" });
    }
    WriteRowToCsv(false, new string[] { SceneManager.GetActiveScene().name, DateTime.Now.ToString("HH:mm:ss.FFFF"), ((float)heartRate.Rate).ToString() });
}

```

Figuur 152: WriteHeartRate-methode voor het OnHearrate-event in het OmniceptDataManagerScript

Deze methode schrijft de tekst die is ingegeven weg in het document dat we hebben bepaald in de “OnEnable”-methode. Als we tekst bovenaan het document willen schrijven moeten we eerst alle tekst in het huidige document lezen en dan opnieuw wegschrijven. Dit wordt gedaan in de eerste if-statement. We schrijven de tekst weg nadat we de array in een string hebben gezet met een “join”-methode [79]. Als de data vanboven moet staan gebruiken we een tweede if-statement om deze er terug onder te schrijven. Voor meer uitleg over “File.ReadAllText” [80] en het gebruik van “StreamWriter” [81] zou ik aanraden de documentatie te lezen.

```

3 references
private void WriteRowToCsv(bool prependDocument, string[] textPerCell)
{
    string originalText = "";
    if (prependDocument && File.Exists(filename))
    {
        originalText = File.ReadAllText(filename);
    }
    StreamWriter writer = new StreamWriter(filename, !prependDocument);
    string text = string.Join(';', textPerCell);
    writer.WriteLine(text);
    if (prependDocument && File.Exists(filename))
    {
        writer.WriteLine(originalText);
    }
    writer.Close();
}

```

Figuur 153: *WriteRowToCsv*-methode in het *OmniceptDataManagerScript*

Deze methode leegt eerst de huidige lijst die we mogelijks hebben gebruikt om eerder een gemiddelde te berekenen. We resetten de “timer” naar 0 en laten het tellen beginnen met een true-waarde in de “timerCounts”. We voegen de eerst gemeten hartslag toe aan de lijst met een extra methode. Deze methode zal vervolgens als een nieuwe listener worden gebruikt. We verwijderen deze methode ook onmiddellijk als listener.

```

3 references
private void StartCalculatingAverage(HeartRate heart)
{
    averageList.Clear();
    timer = 0;
    timerCounts = true;
    AddDataToAverageList(heart);
    gliaBehaviour.OnHeartRate.AddListener(AddDataToAverageList);

    gliaBehaviour.OnHeartRate.RemoveListener(StartCalculatingAverage);
}

```

Figuur 154: *StartCalculatingAverage*-methode voor het *OnHeartrate*-event in het *OmniceptDataManagerScript*

Deze methode voegt de hartslag elke keer toe aan de lijst. Wanneer de “timer” klaar is met tellen zal deze listener ook worden verwijderd. En het gemiddelde wordt dan berekend.

```

5 references
private void AddDataToAverageList(HeartRate heart)
{
    averageList.Add(heart.Rate);
}

```

Figuur 155: *AddDataToAverageList*-methode in het *OmniceptDataManagerScript*

Deze methode hebben we gezien in de "Update"-methode. Als de "timer" kleiner is dan de "listenTime" zullen we gewoon de deltaTime toevoegen. Van zodra dat deze groter is hebben we de "listenTime" bereikt en mogen we stoppen met de "timer" te verhogen. We zetten "timerCounts" op false zodat we hierna niet meer verder tellen. De "averageHeartRate" wordt het gemiddelde van de "averageList" die elke keer werd aangevuld door de listener "AddDataToAverageList". We doen ook een check of er hier geen 0-waarden tussen zitten. Deze komen voor wanneer de sensor in de headset niets heeft kunnen meten omdat deze niet dicht genoeg bij het voorhoofd zat. We schrijven het gemiddelde ook bovenaan in het document. Zo kan dit later ook teruggevonden worden. Als laatste verwijderen we de listener voor "AddDataToAverageList". Zo zal de lijst niet meer onnodig worden opgevuld.

```
1 reference
private void TimerCounting()
{
    if (timer < listenTime)
    {
        timer += Time.deltaTime;
    }
    else
    {
        timerCounts = false;
        averageHeartRate = averageList.Where(x => x != 0).ToList().Average();

        WriteRowToCsv(true, new string[] { "Average heartrate:", averageHeartRate.ToString() });

        gliaBehaviour.OnHeartRate.RemoveListener(AddDataToAverageList);
    }
}
```

Figuur 156: TimerCounting-methode in het OmniceptDataManagerScript

Op het einde van het script vinden we nog drie methoden. Deze lezen of de gebruiker "kalm", "normaal" of "onrustig" is. Deze gebruiken we in de activiteiten om de status van de gebruiker te zien. Als het PPG ooit gebruikt kan worden kunnen deze methoden ook worden aangepast. Dan hoeven de huidige activiteiten niet te hard worden aangepast.

```
7 references
public bool UserIsCalm()
{
    return currentHeartRate < averageHeartRate - rangeHeartRate;
}

3 references
public bool UserIsNormal()
{
    return currentHeartRate >= averageHeartRate - rangeHeartRate && currentHeartRate <= averageHeartRate + rangeHeartRate;
}

13 references
public bool UserIsUneasy()
{
    return currentHeartRate > averageHeartRate + rangeHeartRate;
}
```

Figuur 157:check status hartslag methoden in het OmniceptDataManagerScript

5.3. Data Gebruiken

We maken gebruik van de data door middel van activiteiten. Deze activiteiten zullen altijd wel iets tonen, maar door te lezen of een gebruiker rustiger is kunnen er andere dingen gebeuren. In principe zijn dit gewoon scripts die we activeren met de "Activity_Manager" en de knoppen die hierbij horen.

Ikzelf heb een activiteit gemaakt: de vlinder-activiteit. Dit was eerst een taak van mijn collega. Omdat Deze er te lang mee bezig was en er geen progressie te zien was heb ik dit mogen overnemen, met zijn toestemming.

Voor deze activiteit was er wel nood aan wat extra scripts. Deze scripts zorgen voor de beweging van de vlinders. Zo kan er in het script van de activiteit worden gefocust op de activiteit zelf en hoe er moet gereageerd worden op de veranderingen in de gemoedsstatus van de gebruiker. Deze extra scripts zijn zo gemaakt dat ze in de toekomst ook voor andere zaken gebruikt kunnen worden. deze scripts zullen uigelegd worden nadat de activiteit is uitgelegd.

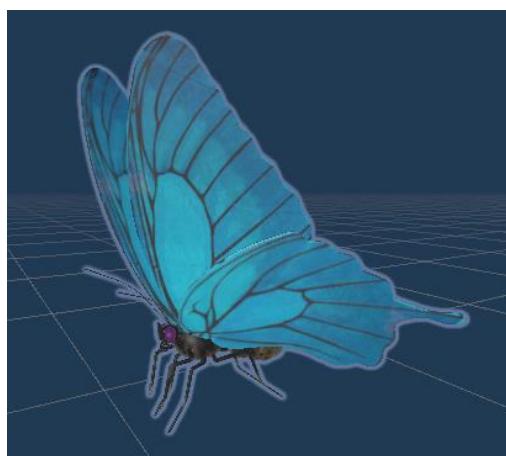
5.3.1. Vlinder activiteit

Bij deze activiteit zal de gebruiker naast een bosje bloemen worden gezet. Bij dit bosje zullen vervolgens vlinders verschijnen. Als de gebruiker rustig is zullen er 4 vlinders aanwezig zijn. Moest de hartslag van de gebruiker binnen de range rond de gemiddelde hartslag schommelen zullen er slechts 3 vlinders zijn. Overbodige vlinders vliegen dan weer weg. Als de gebruiker onrustig is zullen er slechts 2 vlinders zijn.

Daarnaast is er ook een vlinder die fladdert naar de hand van de gebruiker. In deze hand plaatsen we een bloem zodat de vlinder hierop kan landen. Wanneer de hartslag onrustig is zal de vlinder weer wegfladderen. Als extra functionaliteit heb ik hier aan toegevoegd dat als de hand te hard beweegt de vlinder terug komt fladderen naar de bloem.

5.3.1.1. Vlinders

Het model van de vlinders is gevonden door mijn collega. Dit is een gratis asset van een vlinder, met daarbij ook nog een animatie [82] met behulp van een "Animation"-component [83]. Deze animatie heeft mijn collega-stagiaire dan ook bewerkt zodat deze past bij wat we willen.



Figuur 158: Model van de vlinder

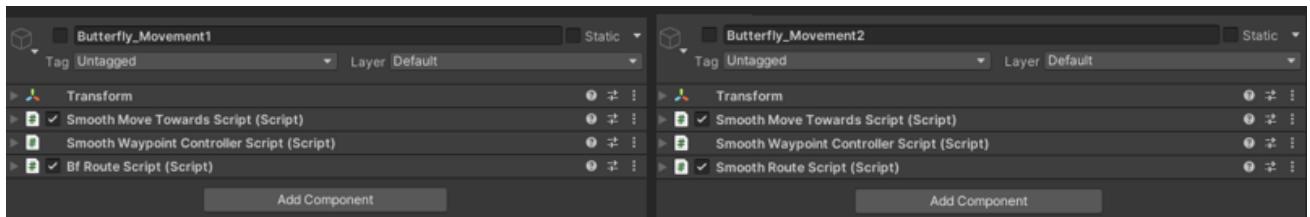
Het probleem met het model is dat deze verkeerd gedraaid staat. Zo staat deze zijwaarts gekanteld en 90 graden naar rechts. Om ervoor te zorgen dat deze kan fladderen met zijn gezicht voorwaarts gericht zet ik deze in een gameobject. Dit gameobject "Butterfly_MovementX" is hetgene dat de beweging in de 3D-omgeving uitvoert. Het model dat er in staat voert alleen maar de animatie uit als dit gevraagd wordt.



Figuur 159: Vlinder in de hiërarchie

Er is daarnaast ook nog een verschil in de "Butterfly_MovementX"-gamobjecten. De eerste vlinder vliegt naar de hand van de gebruiker en legt daarom een andere route af. De overige vinders maken gebruik van een algemaan routescrypt. Zoals hier te zien is hebben de meeste scripts de prefix "Smooth". Er zijn oudere versies van deze scripts die niet zo soepel bewogen. Maar aangezien de code niet heel anders is leg ik deze scripts niet echt uit. Door tijdsgebrek konden we ook de namen niet meer aanpassen.

Het "BfRouteScript" heef slechts een kleine aanpassing in één methode. Deze zal later ook nog uitgelegd worden.

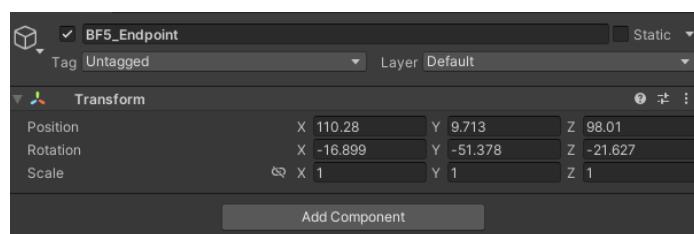


Figuur 160: verschil in vinders

5.3.1.2. Waypoints

Om de vinders van positie naar positie te laten bewegen maken we gebruik van "waypoints". Dit zijn lege gameobjecten waarvan we de positie en rotatie gebruiken voor de verplaatsing van de vinders.

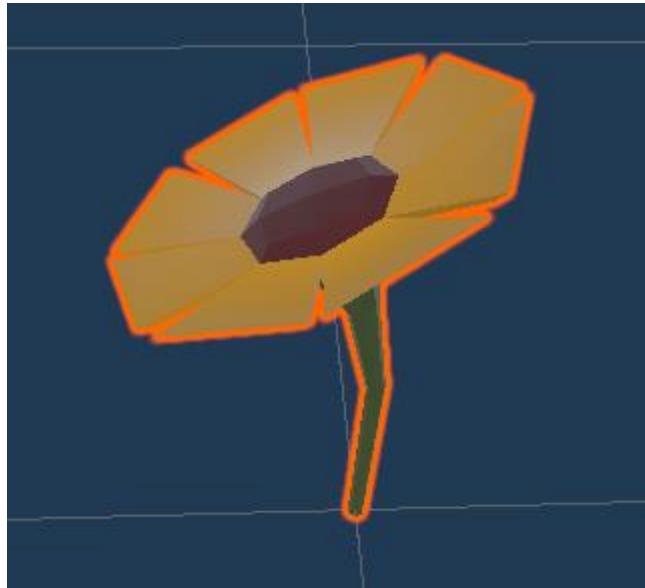
Zoals te zien is op deze afbeelding zit er alleen een transform-component in dit gameobject. Deze heeft ook een aangepaste rotatie. Wanneer een vlinder op deze positie aankomt zal deze naar deze rotatie draaien.



Figuur 161: waypoint voor vlinder 5

5.3.1.3. Bloem in hand gebruiker

De bloem in de hand van de gebruiker is het eindpunt van de eerste vlinder die beweegt. Deze wordt geenabled wanneer de gebruiker de activiteit start. Na de activiteit verdwijnt deze ook.



Figuur 162: model bloem

In de hiërarchie zien we dat er nog een gameobject onder de bloem staat. Dit is de daadwerkelijke plaats waar de vlinder zal eindigen. Deze is gepositioneerd in de kelk van de bloem. Anders zou de vlinder halverwege de bloem zijn "eindpositie" vinden.



Figuur 163: bloem in de hiërarchie

5.3.1.4. Activiteitscript in de Unity Editor

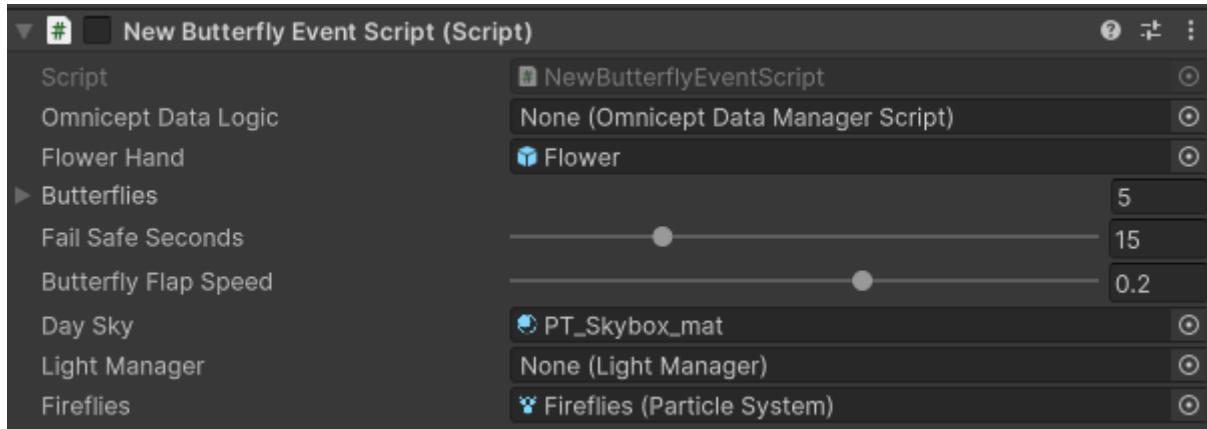
Het script dat we gebruiken noemt "NewButterflyEventScript", de "New" is er gekomen omdat ik deze heb gemaakt terwijl mijn collega nog verder werken aan het oude script. Met zijn toestemming heb ik dit dan "New" als naam gegeven.

In het begin hebben we de manager voor de omniceptdata nodig. Dat is dan ook de grootste reden voor dit project. Daarna komt de bloem die we moeten activeren aan het begin van dit script, en achteraf ook deactiveren. De array "Butterflies" zijn alle "Butterfly_MovementX"-gameobjecten. Zo kan dit script gebruikmaken van de routescripts om te bepalen wanneer ze welke richting op moeten vliegen.

Er zijn vervolgens twee sliders. De eerste is omdat het kan zijn dat een vlinder zijn eindpositie niet kan bereiken en er perfect blijft rond cirkelen. Daarvoor voorzien we een failsafe. Na een aantal seconden zal de volgende vlinder verdergaan, ongeacht of

de vorige klaar is met zijn trip of niet. De “Butterfly Flap Speed” bepaalt hoe snel de animatie van de vlinder is. Deze mag niet te snel zijn anders zou deze epilepsie kunnen veroorzaken bij mensen die hier gevoelig voor zijn. Dit is op aanraden van onze mentor gedaan.

De laatste 3 variabelen moesten van mijn collega toegevoegd worden. In de bosomgeving kan het zijn dat het nacht is en er vuurvliegjes rondvliegen. Deze zijn niet nodig tijdens deze activiteit en mogen dan ook gedeactiveerd worden.



Figuur 164: NewButterflyEventScript in de inspector

5.3.1.5. Code van het Activiteitsscript

Aangezien de variabelen die we zien in de Unity Editor hetzelfde zijn als in al de vorige scripts is hier geen verdere uitweiding over. Wel hebben we hier een “coroutineList”. Wanneer we een nieuwe “Coroutine” maken slaan we die hier in op. op het einde kunnen we dan alle “Coroutines” stoppen om de activiteit af te sluiten. De “failSafeCheck” is een bool die later duidelijker wordt.

```
private List<IEnumerator> coroutineList = new List<IEnumerator>();
private bool failSafeCheck = false;
```

Figuur 165: globale variabelen in het NewButterflyEventScript

In de “Awake”-methode vullen we de nodige managers in met behulp van de tags. “OnEnable” is het begin van onze activiteit, daarom zal alles wat moet beginnen hier ook in worden gezet. Als eerste zien we een try-catch met code om de omgeving om te zetten naar dag. Dit is code die mijn collega heeft gevonden en ik er in moest plaatsen zodat de activiteit werkt in de bosomgeving. Aangezien ik dit testte in de “SampleScene” was dit niet altijd nodig en plaatste ik dit in een try-catch zodat de rest van de code verder kon worden uitgevoerd ongeacht of dit vastliep.

Om alle vlinders zeker op de juiste snelheid te laten fladderen, zette ik de snelheid van de “Animation”-component op de vooraf ingestelde snelheid. We maken hier gebruik van “GetComponentInChildren” [84] omdat het component niet in het “Butterfly_MovementX”-gamobject zit, maar in het gameobject er onder. De [“Take

001_Edited"] is nodig omdat een “Animation”-component meerdere animatie-clips kan hebben. Dat is de naam die mijn collega aan de juiste clip heeft gegeven. Vervolgens zetten we alle vlinders op actief. Als ze inactief zijn buiten de activiteit zijn ze ook verborgen buiten de activiteit.

Vervolgens starten we drie “Coroutines” met onze eigen methode “CreateCoroutine”. De eerste laat de eerste vlinder in de array vliegen als vlinder die naar de bloem fladdert. De tweede bepaalt hoe snel de vleugels fladderen. Omdat de vlinder zo dicht bij de gebruiker komt zal deze stoppen met fladderen als hij op de bloem zit. Deze “Coroutine” houdt in het oog wanneer de vlinder is geland. De laatste bepaalt de volgorde van de overige vlinders die fladderen.

```
Unity Message | 0 references
private void OnEnable()
{
    //code necessary for forestmap
    try
    {
        flowerHand.SetActive(true);

        RenderSettings.skybox.SetFloat("_Exposure", 1);

        lightManager.TimeOfDay = 720;

        if (fireflies.gameObject.activeSelf) fireflies.gameObject.SetActive(false);
    }
    catch (Exception e)
    {
        Debug.Log("error outside of forestmap: " + e);
    }

    //slow down animation
    foreach (GameObject bf in butterflies)
    {
        bf.GetComponentInChildren<Animation>()["Take 001_Edited"].speed = butterflyFlapSpeed;
    }

    //enable butterflies
    foreach (GameObject bf in butterflies)
    {
        bf.SetActive(true);
    }

    createCoroutine(StartInitialBfRoutine(butterflies[0]));
    createCoroutine(FlappingWingsRoutine(butterflies[0]));
    createCoroutine(BfsFlyRoutine());
}
```

Figuur 166: OnEnable-methode in het NewButterflyEventScript

Wanneer het script wordt gedisabled door de “Activity_Manager” zal de “OnDisable”-methode gebruikt worden. Daarom voeren we het afsluiten van de activiteit hier uit. Als eerste stoppen we alle “Coroutines” die bezig zijn. Zo zullen deze niet storen met de afrondende “Coroutine”. Ik had een methode gevonden [86] die dit in één lijn kon doen, maar mijn collega raadde dit af. Later ben ik er achter gekomen dat deze methode wel zou werken. door tijdsgebrek heb ik dit nooit kunnen testen.

We zetten voor de zekerheid de snelheid van de initiële vlinder op wat het hoort te zijn. Als deze dan wegvliegt vanuit zijn eindpositie, de bloem, zal deze dus normaal

fladderen. Ook gebruiken we de "GoBackWards"-methode van ons eigen "BfRouteScript". Daarmee vliegt de eerste vlinder naar zijn begin-positie. Verder vliegen de overige vlinders allemaal weg, hiervoor is een speciale "Coroutine" met als naam "BfsFlyAwayRoutine".

```
Unity Message | 0 references
private void OnDisable()
{
    foreach(IEnumerator c in coroutineList)
    {
        StopCoroutine(c);
    }

    butterflies[0].GetComponentInChildren<Animation>()["Take_001_Edited"].speed = butterflyFlapSpeed;
    butterflies[0].GetComponent<BfRouteScript>().GoBackwards();

    StartCoroutine(BfsFlyAwayRoutine());
}
```

Figuur 167: OnDisable-methode in het NewButterflyEventScript

De methode "CreateCoroutine" vermijd voornamelijk dat de we de hele tijd 3 lijnen code moeten schrijven. Deze maakt de "Coroutine" naar keuze, voegt deze toe aan de "coroutineList" en start deze dan ook.

```
3 references
private void CreateCoroutine(IEnumerator coroutine)
{
    coroutineList.Add(coroutine);
    StartCoroutine(coroutine);
}
```

Figuur 168: CreateCoroutine-methode in het NewButterflyEventScript

Deze "Coroutine" laat de eerste vlinder vliegen naar de bloem. Deze heeft het "BfRouteScript" dat we gebruiken om de vlinder te laten vliegen. Eerst zetten we de "forward" op true, zo weten we dat de vlinder zal beginnen met voorwaarts te vliegen door de lijst van "waypoints" die hij heeft. Dit wil zeggen richting zijn eindpunt. Daarvoor gebruiken we de "StartMovement" en "ChangeDirection" methodes uit het "BfScript". Dit wordt later nog duidelijk hoe deze precies werken.

Vervolgens hebben we een while-loop die oneindig zal doorgaan, of toch tot de "Coroutine" wordt gestopt. Deze loop wordt elke keer voor 2 seconden gestopt zodat de vlinder niet te snel in rondjes zou kunnen vliegen bij een hard afwisselende hartslag. Elke iteratie van de loop doen we ook een check om te zien of we van richting moeten veranderen. Wanneer de gebruiker niet "onrustig" is, zal de vlinder naar zijn eindpunt gaan. Als de gebruiker plots "onrustig" wordt verandert de waarde van de methode uit de "Omnicept_Data_Manager" en gebruiken we de "ChangeDirection"-methode. Verder veranderen we de "forward"-waarde naar het omgekeerde van wat de gebruiker zijn "onrust"-status. Zo zal de verandering van richting gebeuren, slechts wanneer de "forward" en "UserIsUneasy"-methode overeenkomen.

```

1 reference
IEnumerator StartInitialBfRoutine(GameObject butterfly)
{
    BfRouteScript bfScript = butterfly.GetComponent<BfRouteScript>();
    bool forward = true;

    yield return null;
    bfScript.StartMovement();
    bfScript.ChangeDirection();

    while (true)
    {
        if (forward == omniceptDataLogic.UserIsUneasy())
        {
            bfScript.ChangeDirection();
            forward = !omniceptDataLogic.UserIsUneasy();
        }
        //2 seconds to make sure the butterfly doesn't constantly turn with a rapid changing heartrate
        yield return new WaitForSeconds(2);
    }
}

```

Figuur 169: StartInitialBfRoutine-Coroutine in het NewButterflyEventScript

Om te bepalen wanneer de eerste vlinder moet stoppen met fladderen gebruiken we het “routeScript” en “waypointScript” van de vlinder.

Aangezien we in de “OnEnable” alle animatiesnelheden naar de gewenste snelheid hebben gebracht moeten we hier wachten tot de vlinder aankomt op de bloem, het laatste eindpunt uit de “waypointScript”. Om wat ruimte te geven gebruiken we hier de afstand tot dit eindpunt. Wanneer dit kleiner is dan 20 cm zal de vlinderen stoppen met fladderen. Om dit geleidelijk te laten gebeuren gebruiken we een loop. Deze kleine loop voert een actie uit elke tiende van een seconde. De snelheid wordt elke keer verminderd met een twintigste van de originele snelheid. Dit was een getal dat tijdens het testen het soepelst leek te werken. We blijven door de loop gaan tot de snelheid 0 is. Als de nieuwe snelheid onder 0 zou gaan zetten we deze gewoon op 0 om errors te voorkomen.

Daarna wachten we tot de afstand tot het eindpunt groter is dan 20 cm. Daarna zetten we de snelheid terug naar wat het hoort te zijn.

```

1 reference
IEnumerator FlappingWingsRoutine(GameObject butterfly)
{
    yield return null;

    Animation flapping = butterfly.GetComponentInChildren<Animation>();
    BfRouteScript routeScript = butterfly.GetComponent<BfRouteScript>();
    SmoothWaypointControllerScript waypointScript = butterfly.GetComponent<SmoothWaypointControllerScript>();

    while (true)
    {
        yield return new WaitUntil(() => routeScript.MeasureDistanceToCurrentPosition(waypointScript.GetLastWaypoint()) < 0.02f);
        while (flapping["Take 001_Edited"].speed > 0)
        {
            float newSpeed = flapping["Take 001_Edited"].speed - butterflyFlapSpeed / 20;
            flapping["Take 001_Edited"].speed = newSpeed > 0 ? newSpeed : 0;
            yield return new WaitForSeconds(0.1f);
        }
        yield return new WaitUntil(() => routeScript.MeasureDistanceToCurrentPosition(waypointScript.GetLastWaypoint()) > 0.02f);
        flapping["Take 001_Edited"].speed = butterflyFlapSpeed;
    }
}

```

Figuur 170: FlappingWingsRoutine-Coroutine in het NewButterflyEventScript

Omdat de "Coroutine" voor de overige vlinders vrij lang is in code zal ik de uitleg in meerder onderdelen verdelen. De vlinders moeten namelijk in een volgorde wegvliegen, van tweede vlinder tot vijfde vlinder. Als er een te grote verandering is in de hartslag zou het anders kunnen dat twee vlinders tegelijk wegvliegen. Dat is niet de bedoeling en kunnen we met de huidige denkwijze voorkomen.

In het begin voeren we een "yield return null" uit. Zo wachten we tot de volgende frame voor we de rest van de code uitvoeren. Anders ontstaan er errors omdat de vlinder in dezelfde frame op actief zijn gezet. Zie de code in de "OnEnable"-methode. Vervolgens starten we weer de while-loop die zal blijven doorgaan. Nu gebruiken we in deze while-loop een for-loop die door de overige vlinders gaat. Zo voeren we elke vlieg-actie van een vlinder uit voordat de volgende vlinder mag vertrekken.

```
1 reference
IEnumerator BfsFlyRoutine()
{
    //wait for objects to be set to active
    yield return null;
    while(true)
    {
        for (int i = 1; i < butterflies.Length; i++)
    }
```

Figuur 171: BfsFlyRoutine-Coroutine in het NewButterflyEventScript begin

Aan het begin van de for-loop halen we het "routeScript" en "waypointScript" op van de vlinder die we nodig hebben. Als de index van de huidige vlinder kleiner is dan twee voeren we de "GoForward"-methode uit. Deze vlinders moeten niet wegvliegen dus vliegen ze alleen voorwaarts. De "GoForward"-methode blijft gaan tot de vlinder zijn eindpunt bereikt. Wanneer de vlinder al op zijn eindpunt zit zal deze dan ook meteen stoppen.

Als de vlinder in kwestie de derde vlinder is checken we of de gebruiker "onrustig" is. Wanneer de gebruiker "onrustig" is, vliegt deze vlinder namelijk weg. "GoBackWards" werkt op dezelfde manier als "GoForward" en blijft gaan tot de vlinder het beginpunt van zijn tocht bereikt. Als deze daar al is stopt deze methode meteen.

Voor de vierde vlinder doen we hetzelfde als de derde vlinder. Alleen komt deze slechts wanneer de gebruiker "kalm" is. Anders vliegt ze weer weg. Doordat we hier met een else-statement werken kunnen er meerdere vlinders worden toegevoegd aan de "butterflies"-array. Deze zullen dan ook dezelfde actie uitvoeren als de vierde vlinder in de volgorde dat gewenst is.

```

for (int i = 1; i < butterflies.Length; i++)
{
    SmoothRouteScript routeScript = butterflies[i].GetComponent<SmoothRouteScript>();
    SmoothWaypointControllerScript waypointScript = butterflies[i].GetComponent<SmoothWaypointControllerScript>();
    if (i <= 2)
    {
        routeScript.GoForward();
    }
    else if (i <= 3)
    {
        if (!omnieceptDataLogic.UserIsUneasy())
        {
            routeScript.GoForward();
        }
        else
        {
            routeScript.GoBackwards();
        }
    }
    else
    {
        if (omnieceptDataLogic.UserIsCalm())
        {
            routeScript.GoForward();
        }
        else
        {
            routeScript.GoBackwards();
        }
    }
}

//start timer in case butterfly gets stuck in a loop

```

Figuur 172: BfsFlyRoutine-Coroutine in het NewButterflyEventScript midden

Omdat we hier werken met een loop in een “Coroutine” moeten we ook een wait uitvoeren, anders creëren we een oneindige loop die de game crasht. Daarnaast moeten de overige vlinders wachten tot de vorige klaar is met vliegen. Deze wait gaat dus wachten tot de huidige vlinder op een eindpunt beland. Maar om ervoor te zorgen dat de afstand tot een eindpunt niet 0 is, wachten we voor 0.1 seconde zodat de vlinder al begonnen is met vliegen. Het kan namelijk dat de vlinder al op een eindpunt zit en naar een ander eindpunt moet vliegen.

We maken hier een “Coroutine” aan voor de “CheckFailSafe”-routine. Deze zal wachten tot de door “failSafeSeconds” gespecificeerde tijd voorbij is. Ondertussen vliegt de vlinder naar zijn eindbestemming, dit kan het eerste of de laatste waypoint in zijn lijst zijn. Als de afstand tot eender welke van deze twee 0 is gaan we terug naar boven in de loop. Maar als de vlinder vastzit in een loop rond zijn eindpunt zal de “Coroutine” met als variabele naam “check” de globale variabele “failSafeCheck” op true zetten. Als dit gebeurt gaan we ook gewoon verder met de loop. Achteraf moeten we de “check”-routine ook stoppen en voor de zekerheid “failSafeCheck” op false zetten.

Daarmee komen we aan het einde van de for-loop en gaan we terug naar boven tot deze loop klaar is. Wanneer de for-loop klaar is gaan we terug naar de while-loop die opnieuw begint met de for-loop te checken.

```

//start timer in case butterfly gets stuck in a loop
IEnumerator check = CheckFailSafe();
StartCoroutine(check);

// we wait to ensure we aren't accidentally at the start or end of the flight
yield return new WaitForSeconds(0.1f);
yield return new WaitUntil(() 
    => (routeScript.MeasureDistanceToCurrentPosition(waypointScript.GetLastWaypoint()) == 0
    || routeScript.MeasureDistanceToCurrentPosition(waypointScript.GetFirstWaypoint()) == 0)
    || failSafeCheck);

//stop check in case the end was reached before the timer was over
StopCoroutine(check);
failSafeCheck = false;

```

Figuur 173: *BfsFlyRoutine-Coroutine* in het *NewButterflyEventScript* einde

Aan het begin zetten we de “failSafeCheck” voor de zekerheid op false. Daarna checken we of de methode niet toevallig een tijd heeft meegekregen. Als dit niet zo is gebruiken we de globale variabele hiervoor. Met een loop verhogen we de “timer” tot de verwachte tijd bereikt is. Daarna zetten we “failSafeCheck” op true. De reden dat we hier werken met een loop voor de timer en niet gewoon een “WaitForSeconds” is zodat we de “Coroutine” kunnen stoppen. Anders blijft hij verdergaan ongeacht of we hem stoppen.

```

2 references
IEnumerator CheckFailSafe(float seconds = 0)
{
    failSafeCheck = false;
    float maxWaitTime = seconds <= 0 ? failSafeSeconds : seconds;
    float timer = 0;
    while (timer < maxWaitTime)
    {
        timer += Time.deltaTime;
        yield return null;
    }
    failSafeCheck = true;
}

```

Figuur 174: *CheckFailSafe-Coroutine* in het *NewButterflyEventScript*

Deze methode laat de vlinders allemaal specifiek terugvliegen, met ook de notie dat ze moeten wachten tot de vorige vlinder klaar is met zijn tocht. Eerst zetten we de “flowerHand” op inactief. Zo kan deze niet worden meegenomen buiten de activiteit. Daarna zien we een for-loop zoals voorheen bij de “BfsFlyRoutine”, maar deze keer niet in een while-loop. We gebruiken deze keer de “GoBackWards”methode van het “routeScript”. We gebruiken ook weer de “failSafeCheck” voor het geval de vlinder komt vast te zitten op zijn terugtocht. Omdat we deze keer specifiek naar het begin gaan hoeven we ook alleen maar de afstand tot het eerste punt te checken. Wanneer de vlinder op de eindpositie is aangekomen wordt deze inactief, zo is deze niet zichtbaar buiten de activiteit. Na de loop waarmee alle vlinders zijn weggevlogen laten we de eerste vlinder ook verdwijnen. Voor deze gebruikten we de “GoBackWards”-methode in de “OnDisable”-methode. Deze zal al zijn aangekomen tegen de tijd dat de laatste vlinder op zijn positie is. Er is geen andere moment in de code om dit te doen zonder dat deze vlinder in het midden van de vlucht zou verdwijnen.

Verder bevat deze code nog enkele “yield return null” en “WaitForSeconds”. Deze zijn voornamelijk om de timing wat meer verspreid te maken tussen de vlinders. Zo beginnen ze niet te vliegen meteen als een vlinder is aangekomen, maar met een kleine pauze.

```
// reference
IEnumerator BfsFlyAwayRoutine()
{
    flowerHand.SetActive(false);
    //this short piece of code is here to make sure it disappears after the initial butterfly has left
    yield return null;

    for (int i = 1; i < butterflies.Length; i++)
    {
        yield return new WaitForSeconds(1);
        SmoothRouteScript routeScript = butterflies[i].GetComponent<SmoothRouteScript>();
        SmoothWaypointControllerScript waypointScript = butterflies[i].GetComponent<SmoothWaypointControllerScript>();
        routeScript.GoBackwards();

        //start timer in case butterfly gets stuck in a loop
        IEnumerator check = CheckFailSafe();
        StartCoroutine(check);

        yield return new WaitForSeconds(0.1f);
        yield return new WaitUntil(() => routeScript.MeasureDistanceToCurrentPosition(waypointScript.GetFirstWaypoint()) == 0 || failSafeCheck);
        butterflies[i].SetActive(false);

        //stop check in case the end was reached before the timer was over
        StopCoroutine(check);
        failSafeCheck = false;
    }
    //also set initial butterfly to inactive
    yield return null;
    butterflies[0].SetActive(false);
}
```

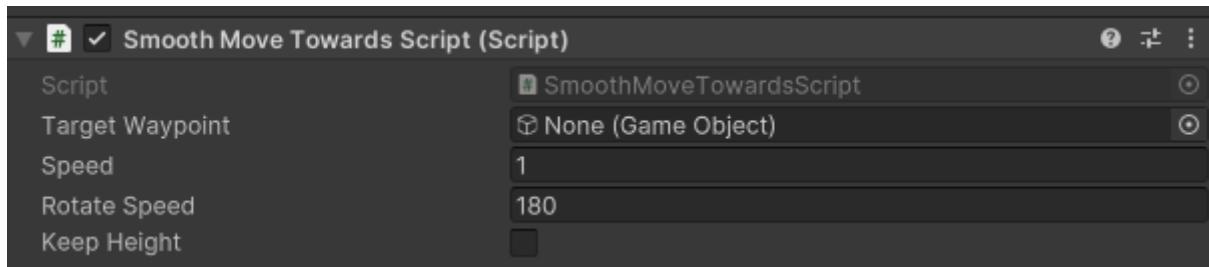
Figuur 175: BfsFlyAwayRoutine-Coroutine in het NewButterflyEventScript

5.3.2. Beweging van objecten

Voor de beweging heb ik een script gemaakt dat een object laat “stappen” in zekere zin. Je kunt in dit script een gameobject ingeven, vervolgens zal dit script het “bewegende” object voorwaarts laten stappen. Tijdens het stappen zal het object zich ook draaien naar zijn einddoel. Dit script is het “SmoothMoveTowardsScript”.

5.3.2.1. Inspector

De eerste variabele is de “Target Waypoint”. Dit is het gameobject waarnaartoe wordt bewogen. Wanneer het bewegende object op de locatie aankomt zal deze waarde null worden. zo blijft deze niet constant hiernaartoe willen bewegen. de “Speed” is de snelheid van het voorwaarts stappen. “Rotate Speed” bepaald hoe snel het bewegend object zich kan draaien. Omdat dit in graden is, zal dit vaak een redelijk hoge waarde zijn. De laatste waarde “Keep Height” was met het vooruitzicht naar een toekomstige activiteit. Bij deze activiteit zouden er eendjes moeten zwemmen richting naar een bepaald punt. Om er voor te zorgen dat deze eendjes niet boven het water zouden zweven of onderduiken zonder reden is “Keep Height” toegevoegd. Als deze aangevinkt is zal het bewegend object altijd dezelfde hoogte behouden.



Figuur 176: SmoothMoveTowardsScript in de inspector

5.3.2.2. Code

De meeste globale variabelen hebben dezelfde structuur als we al vaker hebben gezien. “targetWaypoint” is daarentegen public. Deze wordt gebruikt in het “routescript” dat we al een paar keer hebben gezien. Daar mag het ook worden aangepast. De laatste globale variabele “mayTurn” bepaald of het bewegend object op de positie van het “targetWaypoint” mag draaien richting de rotatie van dit gameobject. We gebruiken hier [HideInInspector] [87] omdat deze mag gebruikt worden in andere scripts, maar niet in de Unity Editor.

```
public GameObject targetWaypoint;
[SerializeField]
private float speed = 1f;
[SerializeField]
private float rotateSpeed = 1f;
[SerializeField]
private bool keepHeight = false;
[HideInInspector]
public bool mayTurn = false;
```

Figuur 177: globale variabelen in het SmoothMoveTowardsScript

De “Update”-methode bekijkt elke frame of er een “targetWaypoint” is. Wanneer deze er is en de positie van het bewegend object en het “targetWaypoint” komen niet overeen moeten we hiernaartoe bewegen.

We bepalen eerst de richting waar het “targetWaypoint” staat ten opzichte van het huidige object. Dit is een Vector3 die als richting werkt. Met deze maken we een “Quaternion” [88] aan door middel van de “LookRotation”-methode [89].

“LookRotation” berekent wat een Quaternion is op basis van een Vector3 die een richting aanduid. Een Quaternion is de klasse van de rotatie van een object. Bij het gebruiken van Quaternions moet men voorzichtig zijn omdat deze vrij onduidelijk kunnen zijn. Vervolgens met de “RotateTowards”-methode [90] van de Quaternion kunnen we de rotatie van ons bewegend object naar het “targetWaypoint” draaien. Deze werkt net zoals de “RotatAround” die we eerder zagen met een hoek. Deze hoek is de “rotateSpeed” vermenigvuldigt met de deltaTime. DeltaTime wordt ook hier weer gebruikt om rekening te houden met de snelheid van frames en het niet te snel laten bewegen van objecten.

Voor het stappen zelf berekenen we eerst de afstand tot ons doel. Vervolgens berekenen we de stap die we normaalgezien nemen. Als we merken dat de normale stap over ons doel zou gaan gebruiken we de "distanceToTarget" voor onze stap. Anders zou het kunnen dat ons bewegend object voor altijd over zijn doel blijft stappen omdat de "normalStep" te groot is. De normale stap is de voorwaartse richting van ons huidig object vermenigvuldigt met de "speed" die we hebben meegegeven. De "transform.forward" is een richting en verandert dus niet hoe groot de stap is, maar alleen de richting. Vervolgens zetten we de "step".

Als onze positie overeenkomt met deze van onze "targetWaypoint" gaan we ons afvragen of we ons mogen draaien. Als dit het geval is kunnen we "RotateTowards" weer gebruiken, maar deze keer met de rotatie van het "targetWaypoint".

Wanneer het stappen en draaien allebei vervolledigt is, maken we het "targetWaypoint" leeg. Als we mogen draaien moet dit leegmaken wel in die else if-clause gebeuren.

Deze code werkt voor stilstaande doelen, maar ook voor bewegende. Doordat de "endRotation" elke frame wordt berekend wordt dan ook een mogelijke nieuwe positie van "targetWaypoint" gebruikt.

```
Unity Message | 0 references
void Update()
{
    if (targetWaypoint != null)
    {
        if (transform.position != GetTargetCoordinates())
        {
            Vector3 directionOfWaypoint = (GetTargetCoordinates() - transform.position);
            Quaternion endRotation = Quaternion.LookRotation(directionOfWaypoint);
            //turn towards target
            transform.rotation = Quaternion.RotateTowards(transform.rotation, endRotation, rotateSpeed * Time.deltaTime);
            //move forward
            float distanceToTarget = Vector3.Distance(GetTargetCoordinates(), transform.position);
            float normalStep = Vector3.Distance(transform.position, transform.position + (transform.forward * speed * Time.deltaTime));
            //if the step overshoots the target we could spin forever, so this keeps that from happening
            Vector3 step = distanceToTarget < normalStep ? GetTargetCoordinates() - transform.position : transform.forward * speed * Time.deltaTime;
            transform.position += step;
        }
        else if(mayTurn)
        {
            transform.rotation = Quaternion.RotateTowards(transform.rotation, targetWaypoint.transform.rotation, rotateSpeed * Time.deltaTime);
            if (transform.rotation == targetWaypoint.transform.rotation) targetWaypoint = null;
        }
        else
        {
            targetWaypoint = null;
        }
    }
}
```

Figuur 178: Update-methode in het SmoothMoveTowardsScript

Deze methode zorgt ervoor dat we de juiste eindpositie terugkrijgen. Als "keepHeight" true is veranderen we de y-waarde naar die van het huidige bewegende object.

```

public Vector3 GetTargetCoordinates()
{
    Vector3 endPoint = targetWaypoint.transform.position;
    if (keepHeight) endPoint.y = transform.position.y;
    return endPoint;
}

```

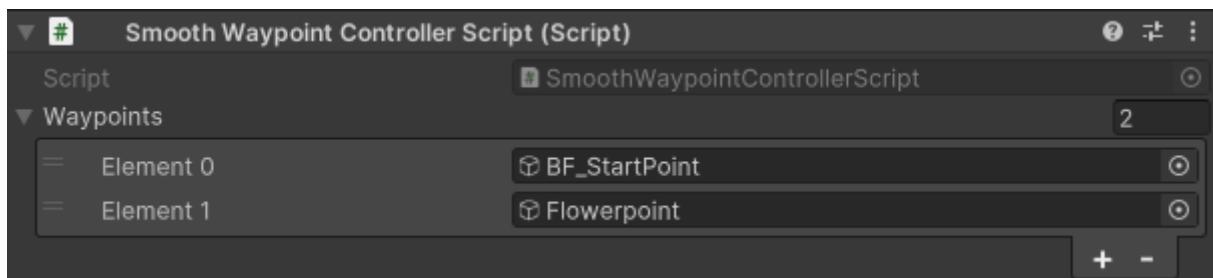
Figuur 179: *GetTargetCoordinates*-methode in het *SmoothMoveTowardsScript*

5.3.3. Waypoint manager

Dit is een script dat alle “waypoints” bijhoudt waarlangs een vlieger kan vliegen. Dit script zou in de toekomst hergebruikt kunnen worden door andere scripts.

5.3.3.1. Inspector

Het “SmoothWaypointControllerScript” heeft slechts de nood aan een array waar de nodige waypoints in kunnen gesleept worden. Dit script maakt verder ook geen gebruik van methoden die Unity aanlevert. Dus geen “Update”, “Awake”, “OnEnable”, enzovoorts. Daarom zien we ook geen vinkje linksboven om deze te enablen of disablen. In principe is dit gewoon een klasse met enkele methoden die we in de Unity Editor kunnen invullen voor het gemak. Om deze zichtbaar te maken in de inspector moet dit script wel overerven van “MonoBehaviour”, net zoals alle scripts die we als component aan een gameobject hangen.



Figuur 180: *SmoothWaypointControllerScript* in de inspector

5.3.3.2. Code

De “waypoints”-array is niet bereikbaar in andere scripts. Maar om gebruik te maken van deze array gebruiken we een index. Deze index heeft ook een veiligheid zodat hij niet groter kan worden als de lengte van de array. Ook zal hij niet lager kunnen worden dan 0, de eerste waarde in de array.

```

[SerializeField]
private GameObject[] waypoints;
private int _currentWaypoint = 0;

10 references
public int CurrentWaypoint
{
    get { return _currentWaypoint; }
    set
    {
        if(value <= 0)
        {
            _currentWaypoint = 0;
        }
        else if(value > waypoints.Length - 1)
        {
            _currentWaypoint = waypoints.Length - 1;
        }
        else
        {
            _currentWaypoint = value;
        }
    }
}

```

Figuur 181: globale variabelen in het SmoothWaypointControllerScript

Omdat de code voor de methoden van dit script niet echt interessant zijn en alleen maar dienen voor het managen van de "currentWaypoint" zal ik hier snel over gaan.

De eerste methode kan weergeven welke waypoint er op een index staat. De twee daaropvolgende methoden zijn bedoelt om op te halen wat het huidige waypoint is volgens onze index. Als we het huidige waypoint zouden willen zetten kunnen we dat met "SetCurrentWaypoint". Beide geven de huidig geselecteerde waypoint terug als return-waarde, ook als deze juist is veranderd door de "Set"-methode.

"GetNextWaypoint" haalt op wat de volgende waypoint zal zijn. Met "SetNextWaypoint" wordt de "currentWaypoint" vermeerderd met 1. Zo zal de huidige waypoint dus de volgende worden. De return-waarde hiervan is dan ook de juist-gesette waypoint.

De methoden voor "PreviousWaypoint", "LastWaypoint" en "FirstWaypoint" werken op dezelfde wijze. Alleen doen deze dit door de index te verminderen met 1, de index naar 0 te zetten of de index naar de laatste waypoint in de lijst te zetten.

Sommige van deze methoden worden op dit moment nog niet gebruikt, maar kunnen in de toekomst gebruikt worden als dit nodig is. Vervolgens heb je dan alleen nog maar een script nodig dat bepaald hoe er van waypoint naar waypoint wordt gegaan. In ons geval is dit het "SmoothRouteScript" of de speciale "BfRouteScript"-variant.

```

0 references
public GameObject GetWaypoint(int index)...

9 references
public GameObject GetCurrentWaypoint()...

0 references
public GameObject SetCurrentWaypoint(int newWaypoint)...

0 references
public GameObject GetNextWaypoint()...

4 references
public GameObject SetNextWaypoint()...

4 references
public GameObject SetPreviousWaypoint()...

0 references
public GameObject GetPreviousWaypoint()...

11 references
public GameObject GetLastWaypoint()...

0 references
public GameObject SetLastWaypoint()...

6 references
public GameObject GetFirstWaypoint()...

2 references
public GameObject SetFirstWaypoint()...

```

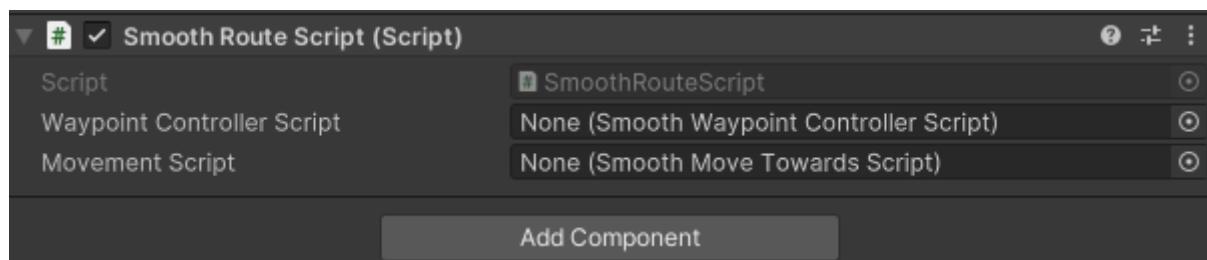
Figuur 182: methodes in het SmoothWaypointControllerScript

5.3.4. Route controller

Dit is een script om door de lijst die het "SmoothWaypointControllerScript" ons aanbied te reizen. Zo kunnen we "voorwaarts" gaan door de lijst, van index 0 tot het einde, of "achterwaarts", van het einde naar index 0.

5.3.4.1. Inspector

In de inspector zien we dat we de twee eerdere scripts hier gebruiken. Deze worden uiteraard opgehaald uit hetzelfde component als ze hier niet worden opgevuld.



Figuur 183: SmoothRouteScript in de inspector

5.3.4.2. Code

De eerste twee variabelen hebben we al vaker gezien. De andere zijn vrij specifiek voor dit script. De "currentDirection" houdt bij welke richting we uitgaan. Er zijn twee

“Coroutines”, waarvan een voorwaarts gaat en de andere achterwaarts. Als er een van de twee bezig is en we veranderen van richting moeten we deze ook kunnen stoppen. Daarnaast is er ook een “forward” bool. Deze dient als check zodat we niet de voorwaartse “Coroutine” stoppen als we bijvoorbeeld nog eens voorwaarts willen gaan.

```
[SerializeField]
SmoothWaypointControllerScript waypointControllerScript;
[SerializeField]
SmoothMoveTowardsScript movementScript;

private IEnumerator currentDirection = null;
private bool forward = true;
```

Figuur 184: globale variabelen in het SmoothRouteScript

In de “Start”-methode vullen we alleen de scripts in die we nog niet hebben ingevuld in de Unity Editor. De eerste methode die we vinden is de “StartMovement”-methode. Hiermee zal het “targetWaypoint” van het “movementScript” worden ingevuld met de eerste in de lijst van de “WaypointController”. Omdat we naar de eerste waypoint gaan, zijn we achterwaarts aan het gaan en wordt “forward” false.

```
1 reference
public void StartMovement()
{
    movementScript.targetWaypoint = waypointControllerScript.SetFirstWaypoint();
    forward = false;
}
```

Figuur 185: StartMovement-methode in het SmoothRouteScript

De “GoForward”-methode stopt de huidige richting en start de voorwaartse “Coroutine” als we nog niet voorwaarts aan het gaan zijn. De “forward” wordt vervolgens ook true, omdat we nu echt voorwaarts aan het gaan zijn.

```
4 references
public void GoForward()
{
    if (currentDirection != null) StopCoroutine(currentDirection);
    currentDirection = ForwardCoroutine();
    forward = true;
    StartCoroutine(currentDirection);
}
```

Figuur 186: GoForward-methode in het SmoothRouteScript

“GoBackWards” doet exact hetzelfde als “GoForward”, maar deze keer in de omgekeerde richting.

```

5 references
public void GoBackwards()
{
    if (currentDirection != null) StopCoroutine(currentDirection);
    currentDirection = BackwardsCoroutine();
    forward = false;
    StartCoroutine(currentDirection);
}

```

Figuur 187: GoBackwards-methode in het SmoothRouteScript

De “ChangeDirection”-methode verandert de richting waar we op dit moment naartoe gaan van voorwaarts naar achterwaarts en omgekeerd. Dit hangt dus af van de “forward”-variabele.

```

2 references
public void ChangeDirection()
{
    if (!forward)
    {
        GoForward();
    }
    else
    {
        GoBackwards();
    }
}

```

Figuur 188: ChangeDirection-methode in het SmoothRouteScript

Deze hulpmethode geeft terug of het bewegend object op dit moment aan het bewegen is naar het meegegeven waypoint. Deze methode wordt meestal gebruikt als we willen weten of we naar het begin of eindwaypoint aan het gaan zijn.

```

4 references
private bool IsGoingToEnd(GameObject waypoint)
{
    return movementScript.targetWaypoint == waypoint;
}

```

Figuur 189: IsGoingToEnd-methode in het SmoothRouteScript

Dit is ook een hulpmethode en wordt gebruikt om te weten wat de afstand is tot aan het meegegeven waypoint. Deze werd eerst in de voorwaartse en achterwaartse routine van dit script gebruikt. Na wat veranderingen en updates wordt deze alleen nog maar gebruikt in de “NewButterflyEventScript” gebruikt.

```

3 references
public float MeasureDistanceToCurrentPosition(GameObject waypoint)
{
    return Vector3.Distance(gameObject.transform.position, waypoint.transform.position);
}

```

Figuur 190: MeasureDistanceToCurrentPosition-methode in het SmoothRouteScript

Aan het begin van de “ForwardRoutine” bewegen we voorwaarts door de “targetWaypoint” in te vullen met de “SetNextWaypoint”-methode. Vervolgens maken we een while-loop. Deze gaat door tot het huidige waypoint waar we naartoe gaan

gelijk hetzelfde is als de laatste waypoint in de lijst. Tijdens deze loop checken we elke frame of het “targetWaypoint” gelijk staat aan null. Wanneer dit zo is, is het waypoint bereikt en kunnen we de volgende setten als het nieuwe target. Als het huidige “targetWaypoint” gelijk staat aan de laatste in de lijst mogen we op deze positie draaien naar de rotatie van dit waypoint.

```
1 reference
IEnumerator ForwardRoutine()
{
    movementScript.targetWaypoint = waypointControllerScript.SetNextWaypoint();
    movementScript.mayTurn = IsGoingToEnd(waypointControllerScript.GetLastWaypoint());
    while (waypointControllerScript.GetCurrentWaypoint() != waypointControllerScript.GetLastWaypoint())
    {
        if (movementScript.targetWaypoint == null)
        {
            movementScript.targetWaypoint = waypointControllerScript.SetNextWaypoint();
            movementScript.mayTurn = IsGoingToEnd(waypointControllerScript.GetLastWaypoint());
        }
        yield return null;
    }
}
```

Figuur 191: ForwardRoutine-Coroutine in het SmoothRouteScript

De “BackwardsRoutine” is qua structuur hetzelfde als de “ForwardRoutine”. Alleen gebruiken we hier “SetPreviousWaypoint” en mogen we pas draaien als het huidige “targetWaypoint” gelijk staat aan het eerste waypoint in de lijst. De if-statement wacht ook nog steeds tot het “targetWaypoint” leeg is, als bevestiging dat een waypoint is bereikt, voordat we de volgende setten.

```
1 reference
IEnumerator BackwardsRoutine()
{
    movementScript.targetWaypoint = waypointControllerScript.SetPreviousWaypoint();
    movementScript.mayTurn = IsGoingToEnd(waypointControllerScript.GetFirstWaypoint());
    while (waypointControllerScript.GetCurrentWaypoint() != waypointControllerScript.GetLastWaypoint())
    {
        if (movementScript.targetWaypoint == null)
        {
            movementScript.targetWaypoint = waypointControllerScript.SetPreviousWaypoint();
            movementScript.mayTurn = IsGoingToEnd(waypointControllerScript.GetFirstWaypoint());
        }
        yield return null;
    }
}
```

Figuur 192: BackwardsRoutine-Coroutine in het SmoothRouteScript

5.3.5. Extra route controller

Voor de initiële vlinder die naar de bloem vliegt hebben we een extra script aangemaakt. Dit script ziet er over het algemeen hetzelfde uit als de “RouteControllerScript” in de inspector. Maar in de code is er een klein verschil in dit “BfRouteScript”.

Aan het begin zien we al dat de mayTurn altijd op false zal staan. Voor het draaien op een bewegend object kunnen we best niet het “SmoothMoveTowardsScript” gebruiken. Dit wordt deze keer geregeld in dit script. De “targetWaypoint” zal uiteraards de volgende worden met de “SetNextWaypoint”.

Doordat we een while-loop gebruiken met een true-waarde zal deze blijven doorgaan tot de "Coroutine" wordt gestopt. De eerste check maakt de "targetWaypoint" de volgende in de lijst als deze leeg is.

Dan komt doen we een check of de afstand van het huidige object tot het laatste waypoint kleiner is dan 10 cm. Als dit zo is maken we de "targetWaypoint" onmiddellijk leeg. Doordat deze verandering in dezelfde frame gebeurt probeert de vlinder niet hiernaartoe te bewegen. Daarna zal de positie van het huidige object gelijk worden gesteld aan deze van de target. Dit gebeurt ook voor de rotatie. Door dit zo te doen kunnen we zachtjes met onze hand bewegen terwijl de vlinder op de bloem zit en hij zal blijven zitten. Als er te hard wordt bewogen zal hij vliegen naar zijn volgende target, wat nog steeds de bloem is.

```
1 reference
IEnumerator ForwardRoutine()
{
    //we handle turning different here
    movementScript.mayTurn = false;

    movementScript.targetWaypoint = waypointControllerScript.SetNextWaypoint();
    while (true)
    {
        if (movementScript.targetWaypoint == null)
        {
            movementScript.targetWaypoint = waypointControllerScript.SetNextWaypoint();
        }
        if(MeasureDistanceToCurrentPosition(waypointControllerScript.GetLastWaypoint()) < 0.01)
        {
            movementScript.targetWaypoint = null;
            transform.position = waypointControllerScript.GetLastWaypoint().transform.position;
            transform.rotation = waypointControllerScript.GetLastWaypoint().transform.rotation;
        }
        yield return null;
    }
}
```

Figuur 193: ForwardRoutine-Coroutine in het BfRouteScript

6. Documentatie

Tegen het einde van het project moesten we aan Mobilab documentatie afleveren van ons project alsook van de Omnicept. Dit diende een technische documentatie te zijn en een gebruikershandleiding voor mensen die de applicatie gebruiken.

Gedurende het project heb ik mijn bevindingen en vooruitgang opgeschreven. Mijn collega deed hetzelfde voor zijn kant van de applicatie. In de laatste week hebben we deze dan gebundeld tot een document dat door Mobilab gebruikt kan worden. In de toekomst kan Mobilab dit uitbreiden en veranderen naargelang ze de applicatie aanpassen.

De laatste weken hebben we ook opnames gemaakt van de speelervaring van de app. Dit wil zeggen dat we zelf de game hebben gespeeld en ondertussen de belevening hebben opgenomen. Met deze opname hebben we vervolgens een "tutorial"-video gemaakt. Deze video legt uit op welke knoppen je kunt duwen om de app te gebruiken. Daarnaast geeft deze video ook uitleg over wat er in de applicatie allemaal gebeurt op het vlak van activiteiten.

7. Obstakels

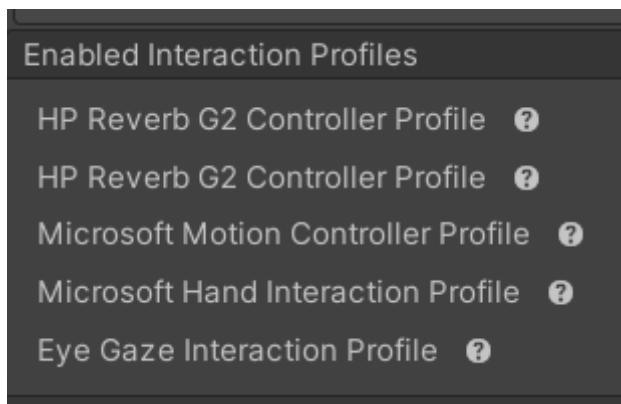
Onder deze sectie wil ik nog enkele obstakels uitleggen die ik ben tegengekomen tijdens het project met hun oplossing. In sommige gevallen zijn dit uitweidingen op iets wat eerder in dit document al is voorgekomen.

7.1. Poorten van de headset

Om de headset te gebruiken moet een computer of laptop minstens een USB-C poort en een displayport hebben. Aangezien mijn laptop en die van de andere stagiaire beiden geen displayport hadden, hebben we gebruik gemaakt van een gaming-laptop van Mobilab. Deze had ook een betere grafische kaart en was over het algemeen sterker dan die van ons. Zo was het visuele aspect van de applicatie tijdens het testen met de headset ook van een hogere kwaliteit en liep de game soepeler.

7.2. Interaction Profiles

Nadat we de "XR Interaction Toolkit" hadden geïnstalleerd was er ook een "Controller Profile" voor de Omnicept-controllers [82] . Deze kwam niet uit de "Mixed Reality Feature Toolkit" en werkt dus ook niet met de controllers van de headset. De enige manier om in de editor te zien welke van Unity is en welke van HP is door op de link naar de documentatie te klikken (het vraagteken achter de naam).



Figuur 194: Twee identiek lijkende Controller Profiles

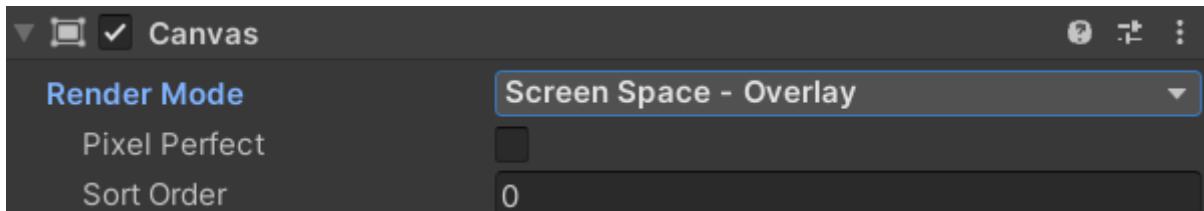
7.3. Omnicept Tray

Bij de initiële bevestiging voor het gebruik van de data van de HP Omnicept is er de kans dat er iets misloopt. Dit is een probleem dat specifiek is naar landen buiten de Verenigde Staten. Met name de pop-up om de toegang te bevestigen, verschijnt niet. De oplossing hiervoor is om de Omnicept-Tray app te gebruiken en dit hierin te bevestigen. Maar door een verschil in regio-gebonden instellingen, konden we de Tray-app niet openen. Dit losten we op door de Tray-app af te sluiten in taakbeheer en de regionale instellingen over te schakelen naar "Verenigde Staten" om zo vervolgens de Tray-app opnieuw op te zetten [83] . Na het eenmalig bevestigen van de toegang,

hadden we verder geen nood aan de Tray-app en konden onze regionale instellingen eventueel weer teruggezet worden.

7.4. Camera Canvas afstand

Voor de camera zou normaal de optie “Screen Space – Overlay” moeten werken om deze voor het zicht van de camera te hangen. Maar tijdens het werken met de “Fade_Screen_Image” hebben we gemerkt dat deze op de laptop wel wordt vertoond, maar niet in de bril.

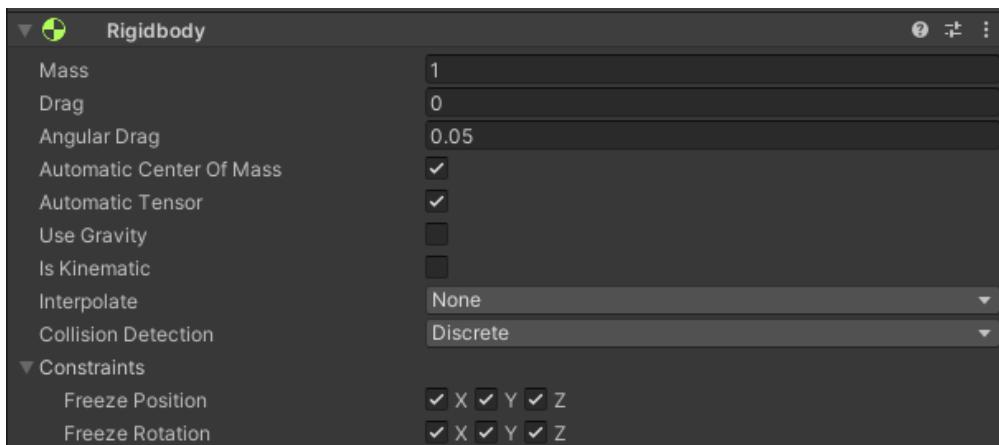


Figuur 195: Screen Space - Overlay

Het zicht van de bril is een combinatie van het linker-en rechteroog. Zodoende wordt hier ook niet exact getoond wat we op een laptop zien met dezelfde bewegingen. Vandaar dat we de “Render Mode” hebben verandert naar “Screen Space – Camera”, zo is het mogelijk om er een afstand aan toe te voegen. Met de afstand op 0 zien we nog steeds niks in de VR-bril. Met een afstand van 0.1, 10 cm, zien we alleen iets in het linkeroog. En met 0.2, 20 cm was het wel mogelijk om het canvas te zien in de headset.

7.5. Teleport Circle Rigidbody

Omdat het object dat we hadden na contact met andere objecten begon te schuiven tegen de wil van de gebruiker in, dienden we enkele aanpassingen te doen. Zo hebben we eerst de constraints van de “Rigidbody” allemaal gefreezed. Wanneer deze in contact kwam met een object, kon ze niet meer wegschuiven buiten de controle van de gebruiker.



Figuur 196: Constraints Rigidbody Teleport Circle

Dit betekende wel dat de “Teleport Circle” gewoon door objecten kon gaan. Daarom hebben we zelf bepaald dat ze alleen zal weggeduwd worden met onze code wanneer ze in contact komt met een object dat niet mag. Dit vormde wel wat extra werk voor mij, maar het was nodig om het schuiven te voorkomen.

Binnen de methoden “OnCollisionEnter” en “OnCollisionStay”, die activeren bij het binnengaan bij een ander object en het aanwezig zijn in een ander object, voegden we onze eigen code toe. Deze code duwt de cirkel weg van dit object.

```
Unity Message | 0 references
private void OnCollisionEnter(Collision collision)
{
    StopEnter(collision);
}

Unity Message | 0 references
private void OnCollisionStay(Collision collision)
{
    StopEnter(collision);
}
```

Figuur 197: *OnCollision*-methoden in het *TeleportScript*

Deze “StopEnter”-methode zal, zoals te zien is op figuur 198, eerst checken of het object waarmee we in contact zijn gekomen een tag heeft waar we niet in mogen komen. Als dit zo is, zal de code verder gaan. We pakken het oorsprongspunt van dit object en vervolgens zetten we de hoogte op deze van de “teleportCircle”. Zo zal het duwen niet naar beneden of boven worden uitgevoerd als het oorsprongspunt op een andere hoogte zit. Hierna wordt de “directionAway” berekend, dit is een Vector3 die een richting aanduid. “normalized” maakt de Vector3 tot een richting in plaats van een afstand. Vervolgens verplaatsen we de “teleportCircle” in de richting van “directionAway”, rekening houden met de “moveSpeed” en een waarde van 0.02. Deze laatste is de waarde waarmee het wegschuiven het minst lastig was visueel en heeft verder geen grote betekenis.

“mayMove” wordt false zodat we de “teleportCircle” niet kunnen bewegen als we in een object zitten. De “OnCollision”-methoden komen in de hiërarchie ook voor de “Start”-methode. Door de “mayMove” hier op false te zetten, wordt de “MoveCircle”-methode in de “Start”-methode dus niet uitgevoerd. Zo bewegen we niet verder in de cirkel terwijl we de andere kant op worden geduwd.

```

2 references
private void StopEnter(Collision collision)
{
    if (Array.Find(notEnterableObjectTags, x => x == collision.gameObject.tag) != null)
    {
        //manual code to stop from entering object we aren't supposed to enter
        //normal unity-way has the added effect of sliding along the floor without control
        Vector3 objectPoint = collision.gameObject.transform.position;
        objectPoint.y = teleportCircle.transform.position.y;
        Vector3 directionAway = (teleportCircle.transform.position - objectPoint).normalized;
        teleportCircle.transform.Translate(directionAway * 0.02f * moveSpeed);
        mayMove = false;
    }
}

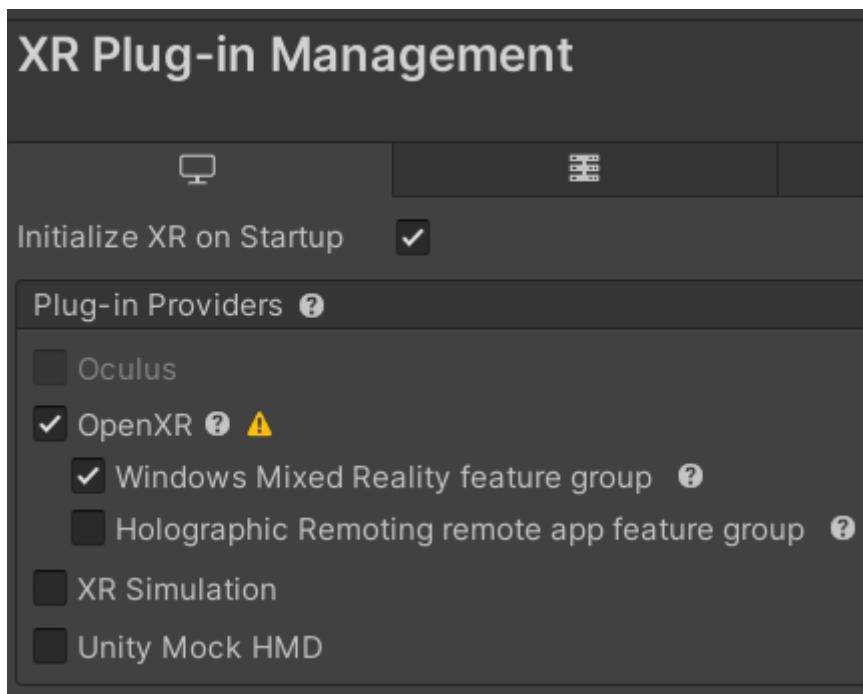
```

Figuur 198: StopEnter-methode in het TeleportScript

7.6. Builden applicatie verstoort werking editor

Nadat we de applicatie hadden gebuild, werkte de headset niet meer met de editor. Tijdens het builden van de applicatie was er namelijk een setting aangevinkt, die nodig was om aan de headset toegang te verlenen tot de applicatie. Deze kan gemakkelijk terug worden aangevinkt, maar dan moet men wel op de hoogte zijn van deze setting.

Deze setting is terug te vinden in de “Project Settings” onder “XR Plug-in Management”. De setting in kwestie is de “OpenXR” setting. Vervolgens zal de “Windows Mixed Reality feature group” setting automatisch worden aangevinkt, het is deze setting die ons toegang verleent.



Figuur 199: Windows Mixed Reality feature group setting

8. Eventuele Uitbreidingen

Onder dit topic volgt een beschrijving van welke uitbreidingen ik nog zou willen toevoegen alsook welke wijzigingen ik nog zou doorvoeren, mocht ik hiervoor nog de tijd hebben. Echter doordat de stage slechts 13 weken in beslag neemt, ben ik hiertoe jammer genoeg niet meer gekomen.

8.1. PPG

Momenteel hebben we nog geen toegang tot de PPG-data. Ik zou graag hiermee hebben gewerkt omdat dit het “OmnicceptDataManagerScript” zeker had kunnen verbeteren. Daarnaast zouden we dan waarschijnlijk meer input hebben kunnen krijgen van onze mentoren met betrekking tot de berekeningen die daarin worden uitgevoerd.

8.2. Universele manager

Op dit moment hebben we meerdere managers voor verschillende onderdelen. Als het mogelijk was zou ik willen kijken hoe we deze misschien onder één gameobject zouden kunnen plaatsen. Zo zouden we de performance van de game eventueel kunnen bevorderen. Ook al hebben de zo goed als lege gameobjecten niet zo'n heel grote invloed op de performance.

Het probleem met een algemene manager is dat verschillende van onze managers ook specifiek zijn voor de omgeving waar ze in geplaatst zijn. Daar zou dus rekening mee gehouden moeten worden. Ook worden enkele managers van tijd tot tijd uitgeschakeld. Een voorbeeld hiervan is de “Teleport_Manager” die wordt uitgeschakeld als een activiteit begint. Bij deze wordt wel het script uitgeschakeld en niet het gameobject zelf. Dat geeft mij het idee dat we de managers op een manier onder een gemeenschappelijk gameobject kunnen plaatsen.

8.3. Forestmap manager

Op dit moment heb ik in mijn “NewButterflyEventScript” code moeten kopiëren van mijn collega. Dit is code om de dag-nacht-cyclus om te zetten naar dag en de vuurvliegjes af te zetten als het dag is. Ik heb gevraagd of het mogelijk was om een manager te maken die dit kon regelen. Zo zouden we dit gemakkelijk aan toekomstige activiteiten kunnen toevoegen. Als er dan nog iets zou bijkomen dat de omgeving drastisch aanpast kan dit ook hier aan worden toegevoegd. Helaas heeft mijn collega dit idee laten liggen om te werken aan andere zaken. ik zou dan ook graag hier nog een manager voor maken.

8.4. Kaart Forestmap

Onze mentoren hebben ons ook de suggestie gedaan om een soort van kaart te maken zodat gebruikers kunnen zien waar ze zijn in een omgeving. Dit zou goed werken als begeleiding. Dit was een extra feature die we graag hadden willen toevoegen aan de

app, maar waar geen tijd meer voor was. Ik heb hier wel kort eventjes onderzoek naar gedaan, maar dan laten liggen om andere zaken af te ronden.

8.5. Update TeleportManagerScript

In het “TeleportManagerScript” maakte ik gebruik van een “Overlapsphere”. Dit was om te zien of er plaats was op een positie een aantal meter voor de gebruiker om een “Teleport Circle” op te roepen. Het probleem is dat hiermee niet wordt gekeken of er tussen de camera en dit punt iets in de weg staat. Over het algemeen lijkt dit tijdens het testen geen problemen te veroorzaken.

Ik heb na wat verder onderzoek gevonden dat er een “SphereCast” [93] bestaat. Deze werkt net zoals de “RayCast”, maar over een breder oppervlak. Zo zou het niet mogelijk moeten zijn om door objecten te kijken waar je niet door mag en daar een “Teleport Circle” aan te maken. Maar dat is slechts een kleine toevoeging die ik zou maken.

8.6. Scripts oproepen

Bij het werken met “gliaBehaviour” vond ik dat ze dit script invullen met behulp van de getter. Dit lijkt me een interessante manier om dit te doen. Online zie ik deze methode niet vaak gebruikt worden. Maar ik zou deze manier willen checken om te zien of dit er ook rekening mee houdt wanneer een script dat er inzit niet meer gevonden kan worden. Wanneer deze bijvoorbeeld vernietigd is of er naar een andere scene is overgegaan. Dan zou het idee van de universele manager, misschien wel een stuk mogelijker lijken.

8.7. Klasse waypoints

In de scripts die ik gebruik voor te bewegen maak ik gebruik van de “GameObject”-klasse. Dit was met de intentie om misschien iets meer hiermee te doen. Later merkte ik dat ik ook de “Transform”-klasse zou kunnen gebruiken. Dit is een component dat standaard in elk gameobject zit en vermoedelijk minder zwaar is om mee te werken. Al de code die ik op dit moment heb geschreven zou ook een stuk korter kunnen worden als ik gebruik maak van de “Transform”-klasse op een directere wijze. Dan zou ik ze niet meer met extra code uit de gameobjecten moeten halen zoals ik nu doe. Dit is ook weer een kleine verandering die niet meteen effect heeft op de werking van de app.

8.8. Inheritance voor scripts

Op dit moment is het “BfRouteScript” een zo goed als exacte kopie van het “SmoothRouteScript”. Dit zou mogelijks verbeterd kunnen worden door gebruik te maken van inheritance. Dat is een piste die ik al op een gegeven moment heb onderzocht voor de “Activity_Manager”. Daarbij heeft de klasse “NewManagedActivity” een variabele van het type “MonoBehaviour”. Hier erven alle scripts van over en daarom kan je daar ook eender welk script in plaatsen. Even heb ik getwijfeld om een eigen “ActivityScript” te maken met enkele altijd nodige methoden. Dit idee werd uiteindelijk het “NewActivityManagerScript” dat gewoon eender welk script kan

behandelen als een activiteit-script. Dit zou het maken van nieuwe "RouteScript"-varianten ook weer kunnen versimpelen.

8.9. Code controleren en aanpassen

Over het algemeen zou ik mijn code nog eens goed willen bekijken. Ik zou graag de namen van enkele scripts willen vervangen. Hierbij denk ik voornamelijk aan de scripts die beginnen met "New" of "Smooth" omdat deze niet meer extra zijn, maar de nieuwe norm. Daarnaast zijn er ook enkele methoden die best wel van naam mogen veranderen. In het "SmoothRouteScript" en "BfRouteScript" is er een methode die "IsGoingToEnd" noemt, deze is van functionaliteit verandert en zou zodoende ook van naam mogen veranderen. Een betere naam zou "isGoingTowards" kunnen zijn aangezien je zelf een waarde kunt meegeven en zien of deze hiernaartoe gaat. Ook zou deze methode waarschijnlijk in het "SmoothMoveTowardsScript" beter op zijn plek staan qua functionaliteit. Ook heb ik gemerkt dat ik in enkele scripts voor de naamgeving van methoden per ongeluk camel-case heb gebruikt in plaats van Pascal-case.

Vele van deze zaken moeten voorzichtig worden aangepast omdat ze op meerdere plekken worden gebruikt. Daarom zouden deze goed getest moeten worden voordat deze simpele aanpassingen echt kunnen worden doorgevoerd.

Dit zijn voornamelijk zaken die tijdens de laatste week of weken van de stage zijn toegevoegd en dus niet grondig genoeg zijn nagekeken wegens drukte.

9. Conclusie

Als laatste wil ik aan dit document toevoegen dat de applicatie zeker goed in elkaar zit. We hebben de applicatie vaak getest en hebben nooit echt grote problemen ontdekt. Daarnaast heb ik altijd rekening proberen te houden met de personen die in de toekomst aan dit project zouden werken.

Ondanks het feit dat er nog uitbreidingen mogelijk zijn, denk ik dat ik mag zeggen dat het project zeker klaar is voor hen die het gaan gebruiken. Met name zowel voor de feitelijke gebruikers als voor de developers die er verder aan zullen werken.

10. Figuurlijst

Figuur 1: HP Reverb G2 Omnicept Edition	6
Figuur 2: Unity Templates	7
Figuur 3: Unity Editor	8
Figuur 4: Unity Scene	9
Figuur 5: Unity Hierarchy	9
Figuur 6: Unity Gameobject Inspector	10
Figuur 7: Unity Assets	10
Figuur 8: Visual Studio 2022	11
Figuur 9: Mixed Reality Feature Tool	12
Figuur 10: Omnicept Developer SDK	12
Figuur 11: Mixed Reality Portal	13
Figuur 12: Unity package voor Glia	13
Figuur 13: HP Omnicept Simulator	14
Figuur 14: HP Omnicept Tray	14
Figuur 15: Hartslag Data voor Unity	15
Figuur 16: Hartslagvariabiliteit data voor Unity	15
Figuur 17: PPG data voor Unity	16
Figuur 18: Oogbeweging data voor Unity	16
Figuur 19: Cognitive Load voor Unity	17
Figuur 20: Mondcamera voor Unity	18
Figuur 21: Unity Package Manager	19
Figuur 22: XR Plugin Management pakket	20
Figuur 23: XR Interaction Toolkit met Starter Assets sample	21
Figuur 24: Device Simulator sample	21
Figuur 25: Device Simulator toevoegen aan scenes	21
Figuur 26: Mixed Reality OpenXR Plugin	22
Figuur 27: Enabled Interaction Profiles	23
Figuur 28: Custom Package toevoegen	23
Figuur 29: HP Omnicept access keys	24
Figuur 30: Unity Devops Version Control	25
Figuur 31: Input Action Map – HeadsetControls	26
Figuur 32: Voorbeeld activeren Input Action Map in code	27
Figuur 33: Camera Component voor tracking headset	28
Figuur 34: Linkercontroller Component voor tracking	28

Figuur 35: Hiërarchie controllers	29
Figuur 36: Right Hand Model	29
Figuur 37: RightHand in Hiërarchie	29
Figuur 38: Rechterhand met Ray Interactor	30
Figuur 39: RightHand componenten	30
Figuur 40: EnableGrabScript in de Unity Editor	31
Figuur 41: variabelen EnableGrabScript	31
Figuur 42: Invullen van default waarden	32
Figuur 43: Update-methode EnableGrabScript	32
Figuur 44: EnableGrabbing-methode	32
Figuur 45: Poke Interactor componenten	33
Figuur 46: Juiste vinger In XR Poke Interactor-script	33
Figuur 47: Direct Interactor componenten	34
Figuur 48: Attach_Point in XR Direct Interactor-script	34
Figuur 49: XR Grab Interactable-script	34
Figuur 50: Vastgrijpen van een object	34
Figuur 51: XR Interaction Manager	35
Figuur 52: XrRig in Unity Editor	36
Figuur 53: Camera Offset in Inspector	36
Figuur 54: XrRig in Inspector met RecalibrateScript	37
Figuur 55: Awake-methode RecalibrateScript	38
Figuur 56: Recalibrate-methode	38
Figuur 57: Verplaatsing	39
Figuur 58: Draaiing	40
Figuur 59: Audio_Manager met AudioManagerScript	41
Figuur 60: Narrator in Unity Editor	41
Figuur 61: Awake-methode AudioManagerScript	42
Figuur 62: UseNarrator-methode met string parameter	42
Figuur 63: UseNarrator-methode met AudioClip parameter	43
Figuur 64: Coroutine UseNarratoroutine	43
Figuur 65: Coroutine ChangeVolumeToRoutine	45
Figuur 66: ChangeVolumeNarrator-methode	45
Figuur 67: ChangeVolumeEnvironment-methode	46
Figuur 68: ChangeVolume AudioSource-methode	46
Figuur 69: Canvas met EventSystem	47
Figuur 70; Canvas configuratie	47
Figuur 71: UI_Manager in Inspector	48

Figuur 72: globale variabelen van het UIManagerScript	48
Figuur 73: Start-methode van het UIManagerScript	49
Figuur 74: ShowMessage-methode in het UIManagerScript	49
Figuur 75: FadeText-Coroutine in UIManagerScript	50
Figuur 76: Update-methode in het UIManagerScript	50
Figuur 77: HideHandMenu-methode in het UIManagerScript	50
Figuur 78: Handmenu in een omgeving	51
Figuur 79: HandMenu in MainHub	51
Figuur 80: HandMenu in hiërarchie	52
Figuur 81: HandMenu in World Space	52
Figuur 82: HandMenu in Inspector	53
Figuur 83: globale variabelen HandMenuScript	53
Figuur 84: OnEnable-methode van het HandMenuScript	54
Figuur 85: verwijderen listener in HandMenuScript	54
Figuur 86: Start-methode in het HandMenuScript	55
Figuur 87: methodes voor listeners in het HandMenuScript	55
Figuur 88: Algemene SetButton-methode in het HandMenuScript	56
Figuur 89: ResetOmniceptData-methode in het HandMenuScript	56
Figuur 90: helpermethoden voor UnityAction	56
Figuur 91: Fade_Screen_Image in de inspector	57
Figuur 92: globale variabelen in het FadeScreenScript	58
Figuur 93: Start-methode in het FadeScreenScript	58
Figuur 94: Fadeln en FadeOut methodes in het FaderScreenScript	58
Figuur 95: algemene Fade-methode in het FadeScreenScript	59
Figuur 96: FadeRoutine-Coroutine in het FadeScreenScript	59
Figuur 97: Tv voor selectie omgeving	60
Figuur 98: Tv met canvas in Hiërarchie	61
Figuur 99: OnClick van de Right-knop	61
Figuur 100: OnClick van de Scene_Preview_Image	61
Figuur 101: ManagedScene-klasse	62
Figuur 102: ManagedScene voorbeeld in de Unity Editor	62
Figuur 103: Transition_Manager in de inspector	63
Figuur 104: currentSelectedScene variabele in het ScenTransitionManagerScript	63
Figuur 105: Start-methode in het SceneTransitionManagerScript	64
Figuur 106: SetcurrentScene-methode in het ScenTransitionManagerScript	64
Figuur 107: NextScene-methode in het SceneTransitionManagerScript	65
Figuur 108: CloseGame-methode in het SceneTransitionManagerScript	65

Figuur 109: GoToSceneByName-methode in het SceneTransitionManagerScript	65
Figuur 110: GoToSceneRoutineByName-Coroutine in het SceneTransitionManagerScript	66
Figuur 111: Prefab voor Teleport Circle	67
Figuur 112: TeleportScript in de Teleport Circle-prefab	67
Figuur 113: globale variabelen in het TeleportScript	68
Figuur 114: OnEnable-methode van het TeleportScript	68
Figuur 115: Update-methode van het TeleportScript	69
Figuur 116: MoveCircle-methode in het TeleportScript	70
Figuur 117: KeepHeight-methode in het TeleportScript	70
Figuur 118: Teleport-method in het TeleportScript	71
Figuur 119: Teleport_Manager in de inspector	71
Figuur 120: globale variabelen in het TeleportManagerScript	72
Figuur 121: Update-methode in het TeleportManagerScript	72
Figuur 122: CreateCircle-methode in het TeleportManagerScript	73
Figuur 123: bord om vlinder-activiteit te starten	74
Figuur 124: activiteit canvas	74
Figuur 125: methoden in de OnClick-array van de MF_Button	74
Figuur 126: NewManagedActivity-klasse	75
Figuur 127: zichtbare cirkel bij de vlinder-activiteit	75
Figuur 128: voorbeeld NewManagedActivity-klasse in de Unity Editor	76
Figuur 129: NewActivityManagerScript in de Activity_Manager in de inspector	76
Figuur 130: globale variabelen in het NewActivityManagerScript	77
Figuur 131: Awake-methode in het NewActivityManagerScript	77
Figuur 132: ChooseNarrator-methode van het NewActivityManagerScript	77
Figuur 133: StartActivity-methode in het NewActivityManagerScript	78
Figuur 134: LockTeleportRoutine-Coroutine in het ?-NewActivityManagerScript	78
Figuur 135: StartActivityRoutine-Coroutine in het NewActivityManagerScript	79
Figuur 136: PlacePlayer-methode in het NewActivityManagerScript	80
Figuur 137: EndActivity-methode in het NewActivityManagerScript	80
Figuur 138: Seat van de vlinder-activiteit zonder zichtbare cirkel	81
Figuur 139: Activity Interactor onder de camera	81
Figuur 140: Activity Interactor in de inspector	81
Figuur 141: Update-methode in het EndActivityScript	82
Figuur 142: Omnicept_Data_Manager in de inspector	83
Figuur 143: gliaBehaviour toevoegen aan een script	84
Figuur 144: wijzen om data van gliaBehaviour op te halen	85
Figuur 145: globale variabelen in het OmniceptDataManagerScript	85

Figuur 146: Awake-methode in het OmniceptDataManagerScript	86
Figuur 147: OnEnable-methode in het OmniceptDataManagerScript	87
Figuur 148: OnDisable-methode van het OmniceptDataManagerScript	87
Figuur 149: Start-methode in het OmniceptDataManagerScript	87
Figuur 150: Update-methode in het OmniceptDataManagerScript	88
Figuur 151: SetHeartRate-methode voor het OnHeartrate-event in het OmniceptDataManagerScript	88
Figuur 152: WriteHeartRate-methode voor het OnHeartrate-event in het OmniceptDataManagerScript	88
Figuur 153: WriteRowToCsv-methode in het OmniceptDataManagerScript	89
Figuur 154: StartCalculatingAverage-methode voor het OnHeartrate-event in het OmniceptDataManagerScript	89
Figuur 155: AddDataToAverageList-methode in het OmniceptDataManagerScript	89
Figuur 156: TimerCounting-methode in het OmniceptDataManagerScript	90
Figuur 157:check status hartslag methoden in het OmniceptDataManagerScript	90
Figuur 158: Model van de vlinder	91
Figuur 159: Vlinder in de hiërarchie	92
Figuur 160: verschil in vlinders	92
Figuur 161: waypoint voor vlinder 5	92
Figuur 162: model bloem	93
Figuur 163: bloem in de hiërarchie	93
Figuur 164: NewButterflyEventScript in de inspector	94
Figuur 165: globale variabelen in het NewButterflyEventScript	94
Figuur 166: OnEnable-methode in het NewButterflyEventScript	95
Figuur 167: OnDisable-methode in het NewButterflyEventScript	96
Figuur 168: CreateCoroutine-methode in het NewButterflyEventScript	96
Figuur 169: StartInitialBfRoutine-Coroutine in het NewButterflyEventScript	97
Figuur 170: FlappingWingsRoutine-Coroutine in het NewButterflyEventScript	97
Figuur 171: BfsFlyRoutine-Coroutine in het NewButterflyEventScript begin	98
Figuur 172: BfsFlyRoutine-Coroutine in het NewButterflyEventScript midden	99
Figuur 173: BfsFlyRoutine-Coroutine in het NewButterflyEventScript einde	100
Figuur 174: CheckFailSafe-Coroutine in het NewButterflyEventScript	100
Figuur 175: BfsFlyAwayRoutine-Coroutine in het NewButterflyEventScript	101
Figuur 176: SmoothMoveTowardsScript in de inspector	102
Figuur 177: globale variabelen in het SmoothMoveTowardsScript	102
Figuur 178: Update-methode in het SmoothMoveTowardsScript	103
Figuur 179: GetTargetCoordinates-methode in het SmoothMoveTowardsScript	104
Figuur 180: SmoothWaypointControllerScript in de inspector	104

Figuur 181: globale variabelen in het SmoothWaypointControllerScript	105
Figuur 182: methodes in het SmoothWaypointControllerScript	106
Figuur 183: SmoothRouteScript in de inspector	106
Figuur 184: globale variabelen in het SmoothRouteScript	107
Figuur 185: StartMovement-methode in het SmoothRouteScript	107
Figuur 186: GoForward-methode in het SmoothRouteScript	107
Figuur 187: GoBackWards-methode in het SmoothRouteScript	108
Figuur 188: ChangeDirection-methode in het SmoothRouteScript	108
Figuur 189: IsGoingToEnd-methode in het SmoothRouteScript	108
Figuur 190: MeasureDistanceToCurrentPosition-methode in het SmoothRouteScript	108
Figuur 191: ForwardRoutine-Coroutine in het SmoothRouteScript	109
Figuur 192: BackwardsRoutine-Coroutine in het SmoothRouteScript	109
Figuur 193: ForwardRoutine-Coroutine in het BfRouteScript	110
Figuur 194: Twee identiek lijkende Controller Profiles	112
Figuur 195: Screen Space - Overlay	113
Figuur 196: Constraints Rigidbody Teleport Circle	113
Figuur 197: OnCollision-methoden in het TeleportScript	114
Figuur 198: StopEnter-methode in het TeleportScript	115
Figuur 199: Windows Mixed Reality feature group setting	115

11. Verwijzingen

- [1] Unity, „Universal Render Pipeline overview | Universal RP | 17.0.3,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@17.0/manual/>.
- [2] Unity, „Unity - Scripting API: GameObject,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/GameObject.html>.
- [3] Microsoft, „Visual Studio 2022 IDE - Programming Tool for Software Developers,” [Online]. Available: <https://visualstudio.microsoft.com/vs/>.
- [4] Microsoft, „Welcome to the Mixed Reality Feature Tool - Mixed Reality | Microsoft Learn,” [Online]. Available: <https://learn.microsoft.com/en-us/windows/mixed-reality/develop/unity/welcome-to-mr-feature-tool>.
- [5] HP, „HP Developers Portal | Downloads,” [Online]. Available: <https://developers.hp.com/omnicipt/downloads>.
- [6] Unreal Engine, „The most powerful real-time 3D creation tool - Unreal Engine,” [Online]. Available: <https://www.unrealengine.com/en-US>.
- [7] HP, „HP Developers Portal | Fundamentals,” [Online]. Available: <https://developers.hp.com/omnicipt/docs/fundamentals>.
- [8] S. Cheriyedath, „Photoplethysmography (PPG),” 27 February 2019. [Online]. Available: [https://www.news-medical.net/health/Photoplethysmography-\(PPG\).aspx#:~:text=Pharm.,the%20surface%20of%20the%20skin..](https://www.news-medical.net/health/Photoplethysmography-(PPG).aspx#:~:text=Pharm.,the%20surface%20of%20the%20skin..)
- [9] HP, „HPO-CLD Technical Report 4.30.21.pdf,” [Online]. Available: <https://developers.hp.com/system/files/attachments/HPO-CLD%20Technical%20Report%204.30.21.pdf>.
- [10] HP, „HP Developers Portal | [Technical Whitepaper] HP Labs: Omnicept Cognitive Load Inference Dataset,” [Online]. Available: <https://developers.hp.com/omnicipt/hp-labs-omnicipt-cognitive-load-inference-dataset>.
- [11] Unity, „Unity - Manual: XR Plugin Management,” [Online]. Available: <https://docs.unity3d.com/Manual/com.unity.xr.management.html>.
- [12] Unity, „XR Interaction Toolkit | XR Interaction Toolkit | 3.0.1,” 21 Februari 2024. [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/index.html>.
- [13] Unity, „Starter Assets | XR Interaction Toolkit | 3.0.1,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/samples-starter-assets.html>.
- [14] Unity, „XR Device Simulator | XR Interaction Toolkit | 3.0.1,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/samples-xr-device-simulator.html>.

- [15] Microsoft, „Set up a new OpenXR project without MRTK - Mixed Reality | Microsoft Learn,” 02 November 2022. [Online]. Available: <https://learn.microsoft.com/en-ca/windows/mixed-reality/develop/unity/new-openxr-project-without-mrtk>.
- [16] HP, „HP Developers Portal | Getting Started with Omnicept,” 2021. [Online]. Available: <https://developers.hp.com/omnicept/docs>.
- [17] HP, „Hp Developers Portal | Getting Started,” [Online]. Available: <https://developers.hp.com/omnicept/docs/console/getting-started>.
- [18] Unity, „Unity Version Control (Previously Plastic SCM) - Fast VCS | Unity,” [Online]. Available: <https://unity.com/solutions/version-control>.
- [19] Unity, „Input System | Input System | 1.8.1,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/manual/index.html>.
- [20] Unity, „Unity - Scripting API: MonoBehaviour.Awake(),” [Online]. Available: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html>.
- [21] Unity, „Unity - Scripting API: Monobehaviour.OnEnable(),” [Online]. Available: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnEnable.html>.
- [22] Unity, „Unity - Scripting API: Monobehaviour.OnDisable(),” [Online]. Available: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnDisable.html>.
- [23] Unity, „Unity - Scripting API: MonoBehaviour.Update(),” [Online]. Available: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html>.
- [24] Unity, „Responding to Actions | Input System | 1.8.2,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/manual/RespondingToActions.html#responding-to-actions-using-callbacks>.
- [25] Unity, „XR Ray Interactor | XR Interaction Toolkit | 2.0.4,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-ray-interactor.html>.
- [26] Unity, „Unity - Manual: Line Renderer component,” [Online]. Available: <https://docs.unity3d.com/Manual/class-LineRenderer.html>.
- [27] Unity, „XR Interactor Line Visual | XR Interaction Toolkit | 2.0.4,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-interactor-line-visual.html>.
- [28] Unity, „Unity - Scripting API: SerializeField,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/SerializeField.html>.
- [29] Unity, „Struct InputAction.CallbackContext | Input System | 1.0.2,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/api/UnityEngine.InputSystem.InputAction.CallbackContext.html>.
- [30] Unity, „XR Poke Interactor | XR Interaction Toolkit | 2.3.2,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.3/manual/xr-poke-interactor.html>.

- [31] Unity, „XR Direct Interactor | XR Interaction Toolkit | 2.0.4,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-direct-interactor.html>.
- [32] Unity, „XR Grab Interactable | XR Interaction Toolkit | 2.0.4,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-grab-interactable.html>.
- [33] Unity, „Unity - Scripting API: Transform,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Transform.html>.
- [34] Unity, „Unity - Scripting API: Vector3,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Vector3.html>.
- [35] Unity, „Unity - Scripting API: Space.World,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Space.World.html>.
- [36] Unity, „Unity - Scripting API: Vector3.SignedAngle,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Vector3.SignedAngle.html>.
- [37] Unity, „Unity - Scripting API: Transform.RotateAround,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Transform.RotateAround.html>.
- [38] Unity, „Unity - Scripting API: AudioSource,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/AudioSource.html>.
- [39] Unity, „Unity - Scripting API: GameObject.GetComponent,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/GameObject.GetComponent.html>.
- [40] Unity, „Unity - Scripting API: AudioClip,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Clipboard.html>.
- [41] Unity, „Unity - Manual: Coroutines,” [Online]. Available: <https://docs.unity3d.com/Manual/Coroutines.html>.
- [42] Unity, „Unity - Manual: Canvas,” [Online]. Available: <https://docs.unity3d.com/2020.1/Documentation/Manual/UICanvas.html>.
- [43] Unity, „Class TMP_Text,” [Online]. Available: https://docs.unity3d.com/Packages/com.unity.textmeshpro@1.0/api/TMPro.TMP_Text.html.
- [44] Unity, „Unity - Scripting API: MonoBehaviour.Start(),” [Online]. Available: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html#:~:text=Start%20is%20called%20on%20the,not%20the%20script%20is%20enabled..>
- [45] Unity, „Unity - Scripting API: GameObject.activeInHierarchy,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/GameObject-activeInHierarchy.html>.
- [46] Unity, „Unity - Manual: Panels,” [Online]. Available: <https://docs.unity3d.com/Manual/UIE-panels.html>.
- [47] Unity, „Unity - Scripting API: Texture2D,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Texture2D.html>.
- [48] Unity, „Unity - Scripting API: MonoBehaviour.OnDestroy(),” [Online]. Available: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnDestroy.html>.
- [49] Unity, „Unity - scripting API: .UnityAction,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Events.UnityAction.html>.

- [50] Unity, „Unity - Scripting API: Color.a,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Color-a.html>.
- [51] Unity, „Unity - Scripting API: Color,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Color.html>.
- [52] Unity, „Unity - Scripting API: Mathf.Lerp,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Mathf.Lerp.html>.
- [53] Unity, „Unity - Scripting API: Time.deltaTime,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Time-deltaTime.html>.
- [54] Unity, „Unity - Scripting API: Serializable,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Serializable.html>.
- [55] Unity, „Unity - Scripting API: Texture,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Texture.html>.
- [56] Unity, „Unity - Scripting API: TextAreaAttribute,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/TextAreaAttribute.html>.
- [57] Unity, „Unity - Scripting API: Object.FindObjectOfType,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Object.FindObjectOfType.html>.
- [58] Unity, „Unity - Manual: Prefabs,” [Online]. Available: <https://docs.unity3d.com/Manual/Prefabs.html>.
- [59] Unity, „Unity - Manual: Rigidbody component reference,” [Online]. Available: <https://docs.unity3d.com/Manual/class-Rigidbody.html>.
- [60] Unity, „Unity - Manual: Layers,” [Online]. Available: <https://docs.unity3d.com/Manual/Layers.html>.
- [61] Unity, „Unity - Scripting API: LayerMask,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/LayerMask.html>.
- [62] Unity, „Unity - Scripting API: Vector2,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Vector2.html>.
- [63] Unity, „Unity - Scripting API: Vector3.Normalize,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Vector3.Normalize.html>.
- [64] Unity, „Unity - Scripting API: Vector3.normalized,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Vector3-normalized.html>.
- [65] Unity, „Unity - Scripting API: Physics.Raycast,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Physics.Raycast.html>.
- [66] Unity, „Unity - Scripting API: Vector3.down,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Vector3-down.html>.
- [67] Unity, „Unity - Scripting API: RaycastHit,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Physics.Raycast.html>.
- [68] Unity, „Unity - Scripting API: Object.Destroy,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Object.Destroy.html>.
- [69] Unity, „Unity - Scripting API: Physics.OverlapSphere,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Physics.OverlapSphere.html>.
- [70] Unity, „Unity - Scripting API: Object.Instantiate,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Object.Instantiate.html>.

- [71] Audacity, „Audacity | Free Audio editor, recorder, music making and more!,” [Online]. Available: <https://www.audacityteam.org/>.
- [72] Unity, „Unity - Scripting API: MonoBehaviour,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>.
- [73] Zwusel, „Trigger event on variable change - Questions and Answers - Unity Forum,” [Online]. Available: <https://discussions.unity.com/t/trigger-event-on-variable-change/167849/2>.
- [74] Unity, „Unity - Scripting API: MeshRenderer,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/MeshRenderer.html>.
- [75] V. Tutorials, „How To Recenter in VR - Unity Tutorial,” [Online]. Available: https://www.youtube.com/watch?v=NOCXB_ETKrM.
- [76] HP, „HP Developers Portal | Usage Examples,” [Online]. Available: <https://developers.hp.com/omnicipt/docs/unity/usage-examples>.
- [77] Unity, „Unity - Scripting API: Object.GetInstanceID,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Object.GetInstanceID.html>.
- [78] Unity, „Unity - Scripting API: GameObject.FindGameObjectsWithTag,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/GameObject.FindGameObjectsWithTag.html>.
- [79] Unity, „Unity - Scripting API: Object.DontDestroyOnLoad,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html>.
- [80] Microsoft, „String.Join Method (System.IO) | Microsoft Learn,” [Online]. Available: <https://learn.microsoft.com/en-us/dotnet/api/system.string.join?view=net-8.0>.
- [81] Microsoft, „File.ReadAllText Method (System.IO) | Microsoft Learn,” [Online]. Available: <https://learn.microsoft.com/en-us/dotnet/api/system.io.file.readalltext?view=net-8.0>.
- [82] Microsoft, „StreamWriter Class (System.IO) | Microsoft Learn,” [Online]. Available: <https://learn.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=net-8.0>.
- [83] Unity, „Unity - Manual: Animation,” [Online]. Available: <https://docs.unity3d.com/Manual/AnimationSection.html>.
- [84] Unity, „Unity - Scripting API: Animation,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Animation.html>.
- [85] Unity, „Unity - Scripting API: Component.GetComponentInChildren,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Component.GetComponentInChildren.html>.
- [86] Unity, „Unity - Scripting API: MonoBehaviour.StopAllCoroutines,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.StopAllCoroutines.html>
- .

- [87] Unity, „Unity - Scripting API: HideInInspector,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/HideInInspector.html>.
- [88] Unity, „Unity - Scripting API: Quaternion,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Quaternion.html>.
- [89] Unity, „Unity - Scripting API: Quaternion.LookRotation,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Quaternion.LookRotation.html>.
- [90] Unity, „Unity - Scripting API: Quaternion.RotateTowards,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Quaternion.RotateTowards.html>.
- [91] Unity, „HP Reverb G2 Controller Profile | OpenXR Plugin | 1.10.0,” [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.10/manual/features/hpreverb2controllerprofile.html>.
- [92] S. A. Omlor, „Having Trouble with Omnicept Tray,” 2023. [Online]. Available: <https://hpomnicept.zendesk.com/hc/en-us/community/posts/12202894774935-Having-trouble-with-Omnicept-Tray-Check-windows-region-settings>.
- [93] Unity, „Unity - Scripting API: Physics.SphereCast,” [Online]. Available: <https://docs.unity3d.com/ScriptReference/Physics.SphereCast.html>.