**Software design document**

**Proof of concept**

**vanHaren**

**Author**: Stela Trencheva

**Student number**: 4520095

**Date**: 09.01.2023

Contents

[**1.** **Introduction** 3](#_Toc124160065)

[**2.** **Technical overview** 4](#_Toc124160066)

# **Introduction**

This POC is an VR shopping experience. It is visualizing what most of the companies nowadays are doing in the metaverse – bringing a physical store into VR and make it a commerce experience in the metaverse. It was developed for the shoe-selling clients vanHaren.

In the experience, the user is placed into a VR 3D world of a vanHaren shop. They are able to look around and move. Two types of movement are implemented – continuous movement and teleportation. The user is also able to grab shoes from the shelves and try them on.

The primary objective of this proof-of-concept from a development standpoint was to learn how to establish the environment for virtual reality development and construct fundamental interactions.

The application was build using Unity and its XR Interaction Toolkit.

# **Technical overview**

The assets in this POC were downloaded from the Unity asset store as well as another 3D asset store – Sketchfab. The shelves and look of the room/shop were reordered so that it looks more like a store. The shoes from vanHaren were placed on store shelves and each shoe has a box collider, which allows them to interact with other objects by colliding when they come into contact with them.

The avatar implementation was done using XR Rig. An XR rig in Unity is a setup of cameras and other components that are used to create a VR or AR experience. The rig typically includes at least one camera for each eye as well as other components such as head tracking and controller tracking. The XR rig is used to create the illusion of being inside the virtual environment and to provide the user with a sense of immersion.

An avatar downloaded from Unity Assets store was implemented inside the object of the XR Rig. By creating a class *AvatarInputConverter.cs*, the avatar model was programmed to move following the movement of the user. That creates the feeling that the avatar is a representation of the user in the VR world. In this case the body movement was stimulated from the head movement of the avatar. The *AvatarInputConverter.cs* script is used to translate input from the user's XR controllers into animations and actions on an avatar, such as hand or body movements. It works by mapping the input from the controllers to the bones and joints of the avatar. For example, when the user moves their hand controller, the script can be used to move the corresponding hand on the avatar in the same way. Additionally, the script can also be used to map input from the controllers to animations on the avatar. For example, when the user pulls the trigger on the controller, the script can be used to play an animation of the avatar grabbing an object. The script can be added to an avatar object in Unity and configured through its properties. It requires an avatar and XR controller objects to be associated with it before it can function properly. The script needs to be integrated with other parts of the system such as the XR rig, XR controllers and XR Interaction manager in order to create a fully functional and interactive experience.

The script has several public variables declared at the top, which include:

* Transforms for the avatar object such as MainAvatarTransform, AvatarHead, AvatarBody, AvatarHand\_Left, AvatarHand\_Right. These are used to store references to the corresponding bones or joints on the avatar that the script will manipulate.
* Transforms for the XR rig such as XRHead, XRHand\_Left, XRHand\_Right. These are used to store references to the corresponding bones or joints on the XR rig that the script will use as input.
* A *Vector3* variable named headPositionOffset, which is used to offset the position of the avatar's head relative to the XR head.
* A *Vector3* variable named handRotationOffset, which is used to offset the rotation of the avatar's hands relative to the XR hand controllers.

The Update() function is present in the script and is invoked once per frame. The transform properties of the avatar and XR rig objects are used by the script to synchronize the position and rotation of the avatar with the input from the user's XR controllers. Interpolation between the current position and rotation of the avatar and the position and rotation of the XR rig is achieved by the script using Vector3.Lerp and Quaternion.Lerp functions, which allows for a smooth transition between the two positions and rotations over time, rather than an immediate jump. A rotation offset to the avatar's hand is also applied by the script using Quaternion.Euler(handRotationOffset) based on the handRotationOffset variable.

Movement in VR is called Locomotion. In VR there are two types – continuous movement and teleportation.

Continuous movement means that when the joystick is moved or rotated, the avatar will be doing the same. Continuous movement in Unity VR was implemented by the *AvatarInputConverter.cs* in combination with a script using Input.GetAxis() method to get the input from the VR controllers and it was used to move the player object. The Input.GetButton() method was also used to check if a button on the VR controller was pressed and this was used to control the movement of the player. Finally, the implementation was tested in the Unity editor and on a VR device to ensure that the movement was smooth and responsive.

Teleportation is done by allowing the player to aim at a specific location and then teleport to that location. This is useful as it allows the player to move around the virtual environment without experiencing motion sickness caused by traditional movement methods such as joystick controls. The teleportation method is used as an addition to the traditional movement controls. It is implemented by using a *raycasting* method where the player aims with a reticle and when they press the teleport button the player is moved to the location where the ray hit. *Unity's Physics Raycast* was used to detect if the player was aiming at a valid teleportation location. The Physics.Raycast() method was used to shoot a ray from the camera or player object and it was checked if it hit a collider with a specific tag. Once a valid teleportation location was detected, the Transform.position property was used to move the player to the hit point of the *raycast*. Visual effects were added to indicate the teleportation process to the player.

Grabbing was also implemented in this POC. In order to make an object interactable and grabbable in Unity VR, a few additional components were added to the objects. One component that was added was a *Box collider* which was used to detect when the object was hit by a raycast. Another component that was added was a *Rigidbody*, which was used to simulate physics interactions when the object was grabbed. A script was also added to the interactable object to handle the specific interactions when it was being grabbed. This script used the OnCollisionEnter() methods to detect when the object was being grabbed, and then used the *Transform.parent* property to make the object a child of the grabbing object. In this way, the interactable object moved and rotated with the grabbing object. Additionally, a script was added to the controller to handle the grabbing action. It checked for the input and fired the raycast, if it hit an interactable object it executed the interaction that was needed.

In order to allow the user to try the shoes that they are grabbing, a transparent sphere was added on the position of the feet of the avatar. In case a shoe object is being dropped near or on the sphere, the object becomes child of the avatar parent object and gets a specific fixed position so that is looks like the avatar is wearing shoes. The object of the shoes stays there until another pair of shoes is dropped around/on the sphere.