École Polytechnique Fédérale de Lausanne

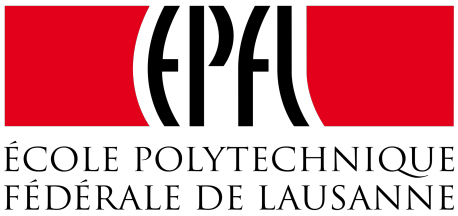
Virtual reality

Project report

Survival VR game

Nathan Quinteiro

Patrik Fraj-Sladoljev



1 Introduction

In our project, we have developed an immersive first-person survival game in a virtual reality environment. The game is designed for HTC-Vive and was developed with the Unity game engine. In the game, the user embodies a warrior lost in an old cemetery, in a lugubrious atmosphere and is given both long and close range weapons: a bow and a sword to defend himself and attempt to survive. The player is free to explore the whole medium sized map using several movement methods:

- Using the tracking pad of the VIVE-controller
- Using teleportation by casting rays from the controller to choose the location
- By running in place, thanks to an Android APP we developed that tracks the steps of the user.

During the game, portals will spawn at random times in several locations in the map, and the undead enemies will return from the Void to attack. The scared player has several methods to try and prolong his survival:

- Draw arrows from his quiver by passing his hand behind neck (virtual quiver metaphor) and shoot with the bow at approaching enemies.
- He can also draw his sword from the scabbard, and use it, either to slash the enemies, or to destroy the invocation totems around portals in order to close them.
- To make the game more dynamic, checkpoints will spawn at random locations and allow the player to gain power-ups by collecting them. The checkpoints include spells (done by gestures) and several weapon improvements.

The goal of the game is for the player to kill as many enemies as possible to slow down the inevitable end (and gain points!).

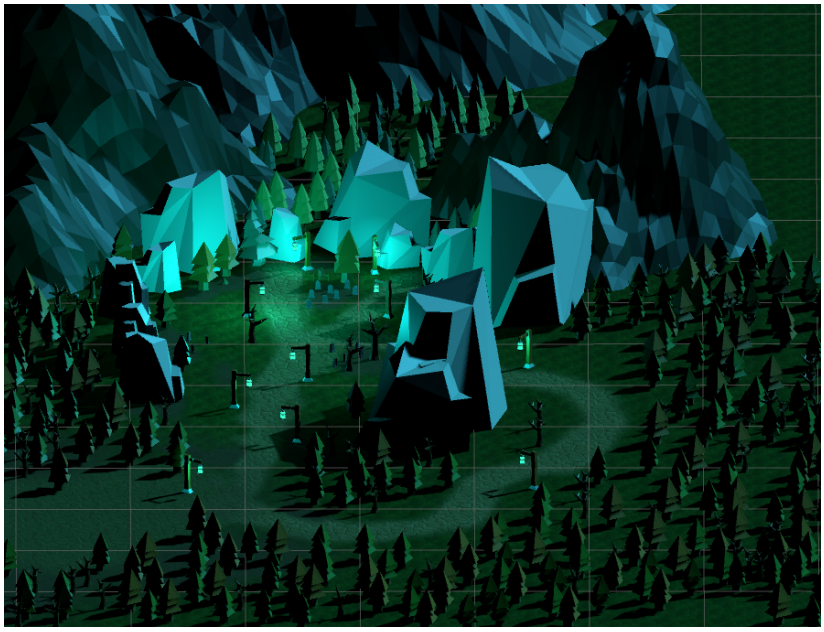


Figure 1: Game map

2 Movement

Since the virtual environment of our game is more spacious than the physical space available in the room, we had to work on providing a movement metaphor allowing the user to explore the whole map without exceeding the physical boundaries. We wanted to keep the advantage of the very accurate and immersive position tracking of the HTC-VIVE system and use additional methods to extend it.

2.1 Step detection

In order to give the user the best immersion, we designed a walking metaphor using a step detection application running on a smartphone in the players pocket. We used Android Studio to develop and build a step detection android application based on the Google pedometer sample [1].
Figure 2 shows the interface of our application. It allows the user to set a threshold on the step amplitude, avoiding to wrongly qualify small movement as steps. It also lets the user set the IP address and port number of the computer running the game in order to transmit step information via UDP (user datagram protocol).

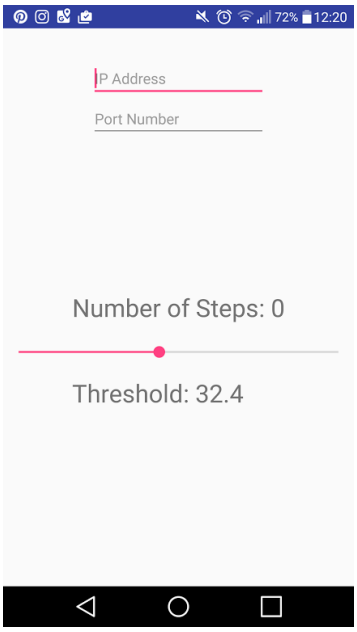


Figure 2: Android Step APP

Figure 3 shows how we implemented the walking metaphor in our game, allowing the user to go further than the physical boundaries in the game. The tracking system remains the same as long as the user stays far enough from the physical boundaries determined by VIVE sensors (blue dashed rectangle).
If the users comes close enough to the boundary (red area), he can start walking or running in place, allowing the application to detect the steps and move the player in the game. This efficiently allows the user to explore the whole map with a familiar movement metaphor.

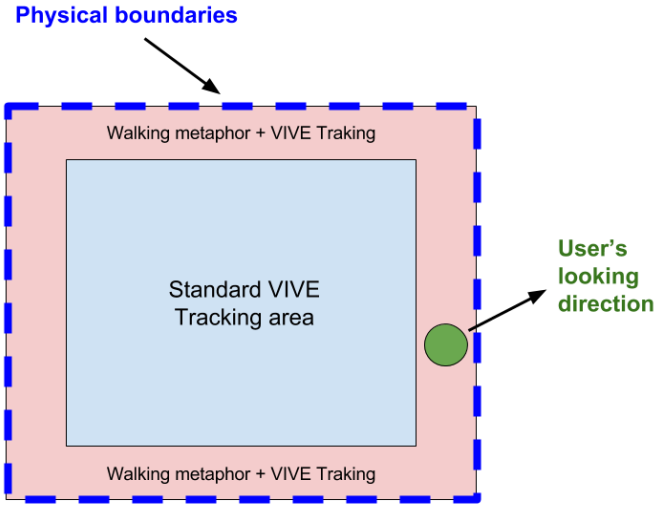


Figure 3: Walking Metaphor area

In order to avoid letting the user pass through obstacles, we assigned a rigidbody to the play area. This way, if the user runs toward an obstacle (a mountain for example), then the area wont move further, and the player won't be able to pass throughe the mountain.

2.2 Trackpad

In the case the player does not want to use the smartphone to provide the walking metaphor, he is still able to explore the whole environment using the trackpad. The X and Y position of his finger on the trackpad function as a movement joystick and allows the player to move.

2.3 Teleportation

A third movement technique available to the user is teleportation. A trigger on the VIVE-Controller casts a ray and let the user point to where he wants to teleport. The teleportation distance is limited, the ray changes colors between green and red, indicating respectably if the teleportation is possible or not. The user validates the teleportation by pressing the trackpad or cancels by releasing the trigger.

3 Object interactions

In the game, the player can interact with several objects and use them as weapons. He is equipped with a bow, a quiver containing arrows and a sword in a scabbard.

3.1 Bow and arrow

The bow is attached to one of the VIVE Controller, held by the non-dominant hand of the user. The position and rotation of the bow follows exactly the ones of the controller, allowing the user to easily place it where he wants in a natural way.

The quiver is placed behind the neck of the user (using the headset position and rotation as reference). This allows easy pick up of arrows by simply passing the dominant hand behind the neck. This is very smooth and feels like a natural behaviour for the player.

Once the user has picked an arrow, he can place it on the nock position of the bow. When the arrow is close enough (few centimeters) to the nock position, it will automatically get clamped to the bow. The player can then start pulling on the string by simply holding the trigger of his dominant hand and pulling his hand, like he would do with a real bow. By releasing the trigger, an arrow is shot with a velocity relative to pull distance.

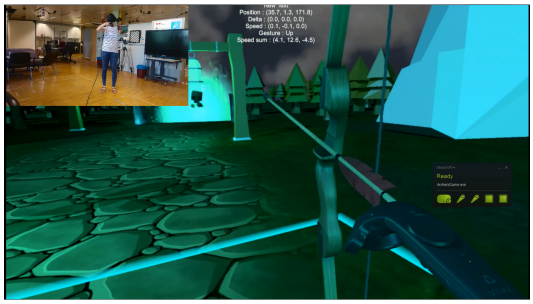


Figure 4: Bow handling

3.2 Sword and scabbard

At the beginning of the game, the sword is located inside the scabbard, and they are both attached to the dominant hand position. The user can then simply position them where he wants the scabbard to be located during the game. Once the desired position is chosen, he can simply press the trigger and the scabbard will stay there for the rest of the game.

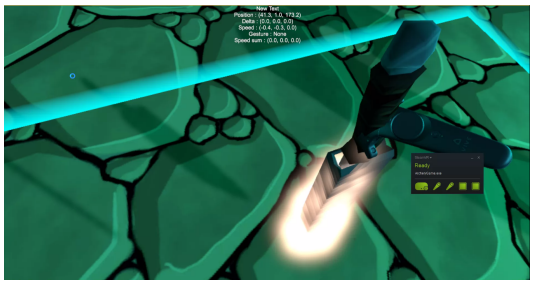


Figure 5: Scabbard holding the sword

From now on, the player can simply pick the sword with his dominant hand by holding the trigger with the controller close enough to the sword. As long as the trigger is pressed, the sword remains in the hand, following the movement of the controller, and can be used to fight the undead enemies. The sword can be put back into the scabbard by releasing the trigger with the hand close enough.

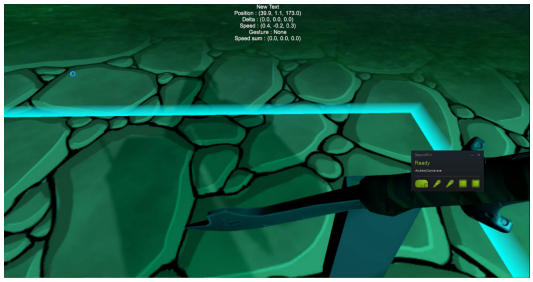


Figure 6: Putting back the sword in the scabbard

If the user releases the sword when it is not close to the scabbard, it will then simply fall on the ground and he will need to pick it up if he wants to use it again. More skilled players can even throw the sword and catch it in the air.

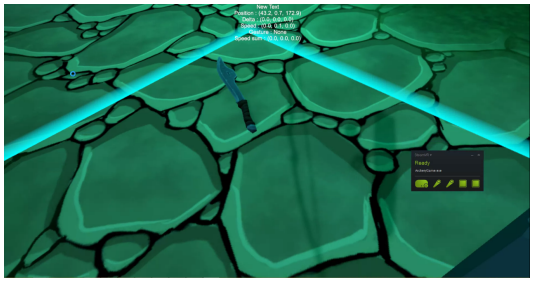


Figure 7: Sword on the floor

4 Enemies

The enemies in the game are the undead, spawning from Void portals appearing randomly across the map. The number of portals that can simultaneously appear on the map varies according to the level of difficulty. In the beginning of the game, there will only be one portal at a time. By killing enemies, the user collects points and advances trough levels and also increasing the difficulty.

There are three types of the undead enemies which are chosen randomly with certain probabilities as they spawn:

- White skeletons: Standard speed and health points. Also most probable to spawn.
- Red skeletons: Run quite fast but have only low health points and are usually killed by one arrow or sword hit.
- Dark skeletons: Walk slowly but have a lot of health points and take a lot of effort to kill.

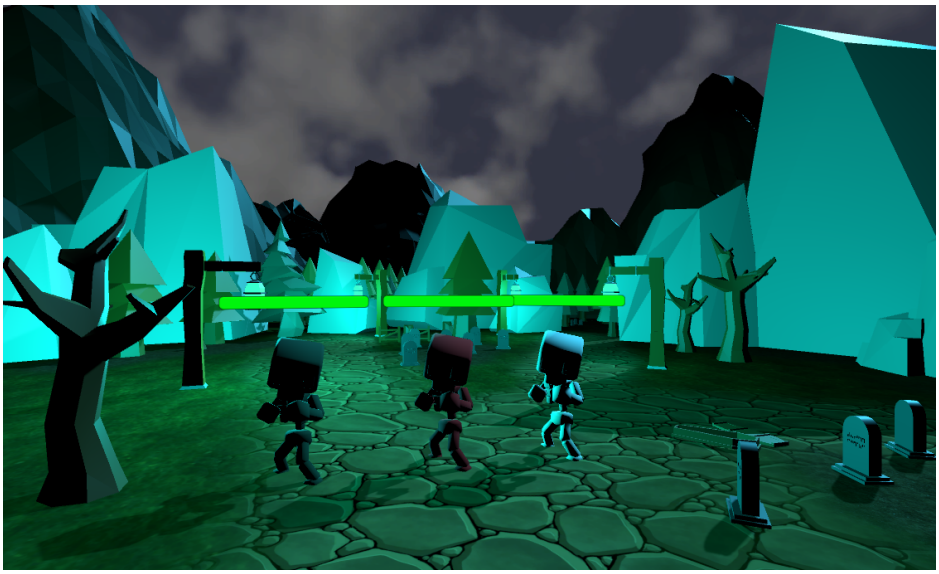


Figure 8: Different types of enemies

5 Power-ups

To make the game more dynamic, checkpoints in the form of glowing green columns (visible on figure 9). When collected by walking trough it, a player is granted a random power-up.

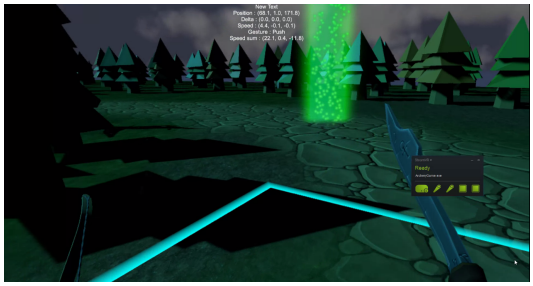


Figure 9: Power-Up Checkpoint

With each different power-up, a voice clip plays and inform the user of the granted power-up. The six existing power-ups are:

- Legendary sword: The sword is in flames for a short time and will kill enemies in one hit.
- Piercing arrows: During a short time, the arrows will kill the enemies in one shot.
- Frozen arrow: The player gets a certain amount of ice arrow which slow enemies on hit for a limited amount of time.
- Double and Triple damage: This multiplies the damage inflicted by the sword and arrows for a short time.
- Spell: This gives the player the ability to cast a certain amount of spells, activated by gestures. (see chapter 6: Gestures)

6 Gesture detection

For the purpose of the game, we have developed a simple gesture detection algorithm. It is able to detect 6 different gestures which are used to cast spells in the game. The gestures correspond to swiping in a certain direction in a 3D space:

- push / pull
- up / down
- left / right

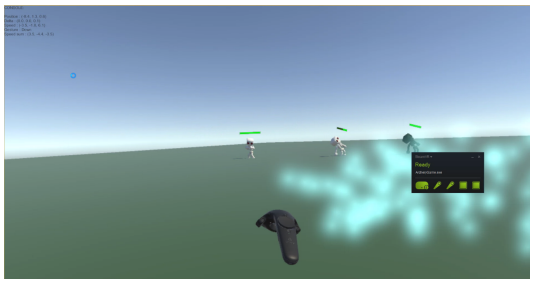


Figure 10: Casting a spell

6.1 Basic idea

The principle is quite simple: measure the speed of the tracked object (VIVE controller) and use the magnitudes to determine the general direction of the movement. The easiest way to do is is to sum several measured speed vectors and determine the direction of the result vector compared to the 6 possible gesture directions. This is efficiently done by finding the vector component with the biggest absolute value to determine the axis and then observing the sign for the direction along the axis.

Example:

$$\text{result speed vector:} \quad v = (5.4, -0.2, -8.9) \tag{1}$$

$$\text{axis} = \mathbf{z} \quad v = (5.4, -0.2, \mathbf{-8.9}) \tag{2}$$

$$\text{direction} = \mathbf{left} \quad \text{sign}(-8.9) = \mathbf{negative} \tag{3}$$

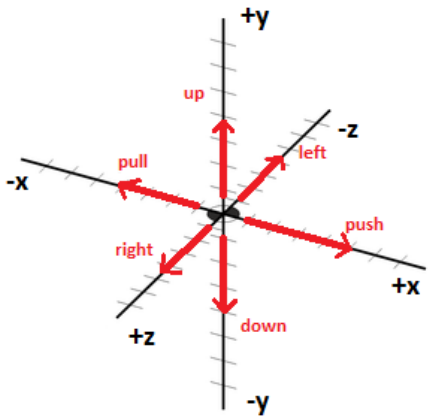


Figure 11: Gestures in 3D space

6.2 Robustness

In order to make the algorithm more robust and noise resistant we use some basic filtering methods and sampling modifications.

- To prevent noise or temporal loss of tracking from interfering with the measurements, we construct a low pass filter to eliminate big translations and speeds. This is done with a simple adjustable threshold.
- The algorithm samples a certain amount of speeds with a certain period e.g. sample the speed every 3 frames and acquire a total of 10 speeds. The parameters can be adjusted for best performance.
- One adjustment that we found convenient for the gameplay is to set a minimum length of the performed gesture.

7 Code details

The two scenes used in our final project are IntroScene.unity and map1.unity and are located in the Assets/Scenes folder of the project

All the scripts we developed or modified are located in the Assets/Scripts folder.

7.1 Player scripts

The Player Game object is called VR_Player and is present in both scenes. It encapsulates the whole player in the game - the "play area" (box determined by physical VIVE area bounds), the headset camera and the controllers and also contains all the SteamVR plugin scripts handling the VIVE hardware.

7.1.1 SteamVR_ ControllerManager.cs

The script SteamVR_ ControllerManager.cs, used in child CameraRig of VR_Player is based on the SteamVR assets. It already handles the interaction with the vive and the play area. The cameraRig has now a Rigidbody, this lets the Movement method put a velocity on this rigidBody in order to make the whole player area move. We added several methods allowing to handle weapons and movement.

Movement: The script now implement the abstract class StepObserver, which makes it notified when a step is detected by the phone by calling the method NotifyStep(). When a step is detected, the script checks if the user is close to the physical border of the scenes and if the user is directed to go outside of it. The method computeBorderDistance() encapsulates the operation. If it is the case, the whole play area will move in the scene according to step velocity. There are also methods Teleport() and SetSpeed() that lets other scripts the possibility to make the user move using respectively the Teleport or the Trackpad.

Weapons: The script is also responsible for instantiating the weapons for the player. Methods InstantiateKnife() and InstantiateArrow() will handle the instantiating of the weapons and place them in the appropriate hand.

7.1.2 ControllerScript.cs

In each controller of the CameraRig, the script ControllerScript.cs handles all user's interactions. Controllers can be set as either the *Dominant* or the *NonDominant* hand and will then run differently.

HandleDominant() method will take care of the Sword (called knife in the script), the arrows and the gestures. The controller can be in several "modes" according to the current state: empty (not holding an object), arrow and sword. Depending on the mode, the scripts runs a certain method.

In arrow mode, the dominant controller will take care of setting the arrow on the bow when it is close enough to the nock position, animate the bow when the string is pulled, and launch the arrow when the trigger is released. In knife mode (sword), the controller just makes the sword follow the user's hand as long as the trigger is held. When the trigger is released, the sword is detached from the hand, and a proper velocity is assigned to the sword to let it follow its path in a natural physic way (also allowing the user sword throwing). To compute the velocity, we simply take the distance between the sword position in the last frame and the current frame, and divide it by the time of a frame. Thanks to the high accuracy of the Vive tracking, the computed velocity is really accurate and the result in the game feels quite natural. In spell mode, the script handles the gestures and allows the user to cast spells.

HandleNonDominant() method is responsible for handling the teleportation and the trackpad movement. It casts a ray in the scene when the user press the trigger to show the teleport destination, and calls the Teleport() method of SteamVR_ControllerManager if the user presses the pad while the ray is cast.
To let the user move using the trackpad, the XY position of the finger on the trackpad is read

and the method SetSpeed() of the SteamVR_ControllerManager is called with the appropriate speed corresponding to the command of the user.

7.1.3 PlayerForceController.cs

method handles the spells and includes reading the gesture form the gesture manager script and applying the correct spells (and effect on the enemies). It also implements several methods of selecting the affected enemies:

- ray cast - affect only enemies allowing a ray cast from the camera
- cone cast - affect enemies in a cone of a certain angle, cast from the camera
- sphere cast - affect enemies in a sphere around the player

7.1.4 KnifePocketController.cs

This script take cares of the positioning the sword scabbard. It makes sure the scabbard follows the user at a constant height, using the head as reference.

7.1.5 PlayerHealth.cs

This script is used by the Camera (eye) GameObject of the VR_Player. It handles the health points of the player, by detecting hits from the enemies and decrementing the health accordingly. When the health decreases, the vision of the player is blurred in red to give feedback and feel of health amount. When The life falls to zero, all lights of the scene turn off, and a "You Died" indicator appears, showing the player that the game is over.

7.2 Enemy scripts

7.2.1 AIController.cs

The script handles all logic for the enemies including initializing the type, calculating the movement direction and handling the animator (playing certain animations) and attacking the player. It provides public methods for stun, slow and damage and also for doing certain actions such as applying a spell.

7.3 Weapon scripts

The abstract class DamagingObject from DamagingObject.cs is implemented by all weapons to provide the interface between the enemy script and the weapons and inflict damage or a slow effect accordingly.

7.3.1 ArrowController.cs

This script is used by the arrow prefabs and implements the DamagingObject interface. It contains all the parameters of an arrow. The GetDamage() method is used by the enemy script to know how much damage the arrow inflict, which depends on the type of arrow. The GetSlow() method will return nothing if the arrow is not a frozen arrow, and will return a frost percentage and frost duration if the arrow is a frozen arrow.

7.3.2 SwordScript.cs

This script is used by the Sword prefab, it also implements the DamagingObject interface in order to inflict damage to enemies. The GetSlow() method always returns nothing, since there are no power-ups that give a freezing ability to the sword. The GetDamage() method will inflict damage only if the sword velocity is higher than a certain threshold, in order to avoid killing undead without even swiping the sword. The velocity of the sword is computed in each frame by dividing the distance difference by the frame duration.

7.4 Portal scripts

Several Portal GameObjects are randomly activated on the scene by the PortalManager GameObject. Each Portal spawn enemies and can be destroyed by hitting the totems with the sword.

7.4.1 PortalManager.cs

This script is responsible for the activation and destruction of portals in the scene. It has parameters allowing to decide the location where the portals may appear, the period and the amount of portals.

7.4.2 PortalScript.cs

This script allows the portal to create totems around him, that will then allow the user to destroy the portal. It spawns enemies with a certain frequency, which can be set by the PortalManager according to the level of difficulty.
If the Portal is destroyed, its size will decreases for two second until it disappears completely. The PortalManager is notified when the portal is destroyed so that a count of active portals is correct.

7.4.3 Totemcript.cs

This simple script is just responsible for informing its parent Portal that the totem was hit by the sword and the portal should be destroyed.

7.5 Power-up scripts

During the game, a Power-Up checkpoint is always present in the scene, allowing the user to gain bonus by passing through it. Once the bonus is picked up, the checkpoint moves somewhere else in the scene.

7.5.1 GameManager.cs

This script is responsible for spawning checkpoints at random locations. As the user passes trough a checkpoint, the script is notified and proceeds to grant a power-up by notifying the PowerupManager and also spawns a new checkpoint.

7.5.2 PowerupManager.cs

This script handles the power-ups (bonuses). It is notified with the method CheckpointCollected() that the user has collected a checkpoint. A random power-up is then given to the user. The class contains static variables holding the arrow and sword damage multipliers, and can be used by the arrows and sword scripts to inflict more damage.

8 Conclusion

Our game offers a complete VR experience, with a tutorial explaining how to interact with the weapons and the enemies, and a complete scene containing 4 levels. After letting novices test our game, we observed that they felt totally immersed very quickly and could naturally understand the interaction with the weapons. The game has a nice and complete environment and allows to interaction with different weapons. Picking the arrow and using the bowis done in a very natural way, making the whole interaction easier and more fun.

The sword interaction is quite intuitive and simple for the player. Fighting the enemies with the sword is easy and feels natural, apart for the lack of a really good haptic feeling when enemy are hit as the vibrations in the controller don't provide a good enough feeling. The enemies are very nice to fight, they follow the player, and their speed/life points are designed in such a way that it is interesting enough to play. The increasing difficulty and power-ups make the game dynamic and keep the user from giving up or feeling bored.

Finally, the background music and sound effect make the atmosphere more real and the game more enjoyable.

9 References

[1] Google Simple Pedometer sample: used for the Android app detecting the steps - `https://github.com/google/simple-pedometer`

[2] SteamVR Plugin for Unity: used as base for the VRPlayer - `https://www.assetstore.unity3d.com/en/#!/content/32647`