European Media and Immersion Lab

D3.6 – SDK for integration of physical activities into VR experiences

Work Package 3 – Lighthouse Projects at Laboratory Nodes

Authors:

Christof Lutteroth (UB)
Christopher Clarke (UB)
Crescent Jicol (UB)
Adwait Sharma (UB)

Grant Agreement number **101070533**
Action Acronym **EMIL**
Action Title **European Media and Immersion Lab**
Call **HORIZON-CL4-2021-HUMAN-01**

Co-funded by the European Union

UK Research and Innovation

EMIL project is funded by the European Union and UK Research and innovation.

Version date of the Annex I against which the assessment will be made	*Start date of the project*	*Due date of the deliverable*	*Actual date of submission*	*Lead BEN / AP for the deliverable*	*Dissemination level of the deliverable*
18.3.2022	1.9.2022	28.2.2023	28.2.2023	UB	Public

Document reviewer(s)	
Name	*Beneficiary*
Narcis Pares Burgues	UPF

Abstract

This document, “SDK for integration of physical activities into VR experiences” (D3.6) provides an overview of a Software Development Kit (SDK) that can be used to create VR experiences that integrate physical activities such as walking-in-place, cycling and walking on a crosstrainer. The SDK is able to detect physical activities purely based on headset movements. This document describes the software components of the SDK that support different forms of physical activity, and also gives brief examples of how they can be applied in a VR project.

Contents

1 Introduction

This document, “SDK for integration of physical activities into VR experiences” (D3.6) provides an overview of a Software Development Kit (SDK) that can be used to create VR experiences that integrate physical activities such as walking-in-place, cycling and walking on a crosstrainer. The SDK is able to detect physical activities purely based on headset movements. This document describes the software components of the SDK that support different forms of physical activity, and also gives brief examples of how they can be applied in a VR project. The document can serve as a resource for potential FSTP projects that integrate physical activities into VR. The document is divided into the following main chapters, providing an overview of different forms of locomotion and methods of estimating the direction of intended locomotion:

SDK Overview: An overview of the components that make up the SDK.

Locomotion based on Physical Activity: An overview of the component for locomotion in VR based on physical movements of the user.

Estimating the Direction of Locomotion: An overview of the component that support estimation of the direction the user wants to move towards in the virtual environment.

Application Examples: Examples of how the SDK can be used to implement movement based on physical activity in VR.

2 SDK Overview

This chapter provides an overview of the components that make up the SDK. The SDK is being maintained for several popular game engines including Unity and Godot. All these game engines use scene trees to manage assets in a VR scene; the components contained in the SDK are nodes that can be inserted into the scene tree to add support for various physical activities. Each type of physical activity is implemented as a type of node. The examples given in this document use the Godot game engine, but will be similar for Unity. The SDK can be used with a wide variety of VR hardware including head- and handsets supported by OpenVR (e.g. HTC Vive) and OpenXR (e.g. Oculus Quest). It is designed to work without any additional hardware and does not rely on connections to any exercise equipment. Node types typically support object-oriented features such as inheritance and associations, so can be illustrated using a visual class diagram notation. Image 1 shows a class diagram of the SDK’s node types as well as some of their properties and relationships between them.

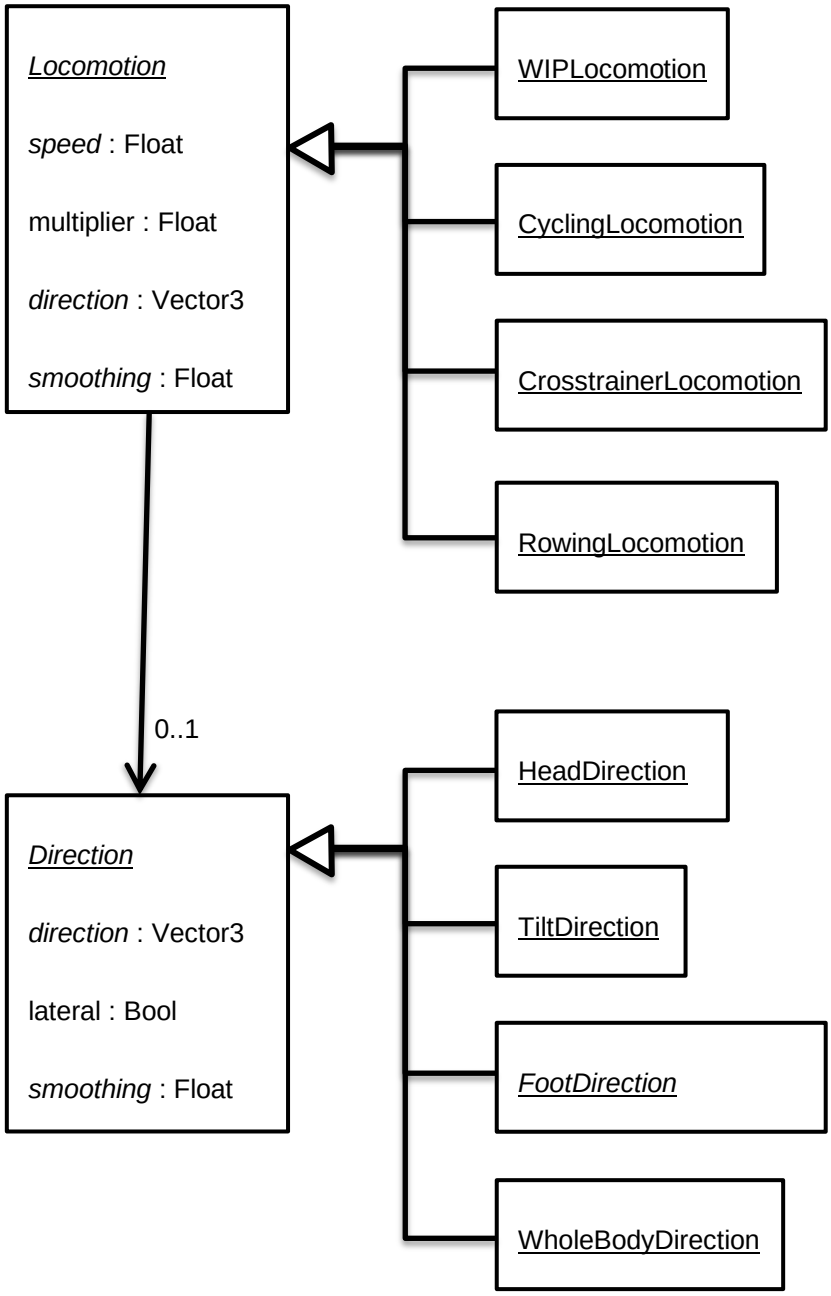

Image 1 Overview of node types in the SDK.

Boxes represent node types. The identifier at the top of a box is the name of the node type. If it is written in italics (such as *Locomotion* and *Direction*) this means it is an abstract type, i.e. it is not used directly but only though its subtypes. Below the node type name, some of the most important properties of node types are listed, together with their type. For example, *Locomotion* has a property called *speed* which is a floating-point number and a property *direction* which is a 3D vector. The properties of node types can be accessed in instances of the node types through the graphical user interface (GUI) of the game engine. This makes it easy to add an instance of a node type to the scene tree of a VR scene, adjust its properties to fine-tune the behaviour of the node to match the requirements of the designer, and see property values during testing.

The arrows with hollow triangular arrowheads represent inheritance. For example, *WIPLocomotion* is a subtype of *Locomotion*. This means that all properties of *Locomotion* are inherited by *WIPLocomotion*, *CyclingLocomotion*, *CrosstrainerLocomotion* and *RowingLocomotion*, respectively.

The arrow with the solid arrowhead represents an association between *Locomotion* nodes and *Direction* nodes. A *Locomotion* node can have at most one associated *Direction* node. This association is typically done by simply placing the *Direction* node as a child under the *Locomotion* node in the scene tree. Examples of how nodes can be used are provided in Section 5.

2.1 Locomotion

The abstract node type *Locomotion* defines an interface shared by all physical VR locomotion methods implemented by the SDK. That is, it defines properties and functionality shared by all methods that move the user in VR based on their physical movements. The different subtypes of *Locomotion* implement locomotion for specific types of physical user movements:

WIPLocomotion supports locomotion through walking-in-place (WIP) movements. That is, the user lifts one foot, as if taking a step, and then puts it down again onto the floor, and then repeating the movement with the other foot and so on. This is similar to the typical human walk cycle, but without the forward movement. The higher the stepping frequency, the faster is the movement in VR. This node can also be used for locomotion in VR through running-on-the spot.

CyclingLocomotion supports locomotion by riding a stationary bike such as an exercycle. When a user start pedalling, then the user will move in VR as if the bike is a normal bike and not a stationary one. The higher the cycling cadence, the faster is the user moved in VR.

CrosstrainerLocomotion supports locomotion by walking on a crosstrainer or elliptical trainer. When a user starts walking or running on the crosstrainer, then the user will also move in VR. The higher the cadence of the crosstrainer, the faster is the user moved in VR.

RowingLocomotion supports locomotion by rowing on a stationary rowing machine. When a user starts rowing, then the user will move in VR. The higher the rowing strokes per minute, the faster is the user moved in VR.

Every *Locomotion* node has the following properties:

speed is a read-only floating-point value that shows the locomotion speed in meters per second currently estimated by the locomotion method. This can be applied directly to the VR representation of the user to create an illusion of real movement.

multiplier is a floating-point coefficient that can be used to adjust the speed of movement produced by the locomotion method. A *multiplier* of 1.0 will produce a speed that is plausible for the respective physical activity, e.g. for the measured cycling cadence assuming an average gear. A *multiplier* of 2.0 will produce movement that is twice as fast, and 0.5 will produce movement half as fast etc.

direction is a 3D vector describing the direction the locomotion will move the VR user towards in the 3D scene if no *Direction* node is used to estimate the direction dynamically (see below). For example, a *direction* vector of (0, 0, -1) would move the user forward in a right-handed coordinate system. Such a constant direction vector is useful if the player is moving only into one direction based on physical activity such as cycling, e.g. if the player is moving along a straight road.

smoothing is a floating-point parameter describing how easily the estimated locomotion speed can fluctuate from moment to moment. This is based on an exponential average between the current and previous speed values, resulting in a low-pass filter. A *smoothing* of 0 will not apply any smoothing, i.e. estimated speed values can change quickly and potentially erratically. A *smoothing* greater than 0 and smaller than 1 will average between current and previous locomotion estimates. The higher the *smoothing*, the more gradual are the changes in speed produced by a locomotion method.

2.2 Direction

The abstract node type *Direction* defines an interface shared by different methods for estimating the direction of locomotion. It defines properties and functionality shared by all methods that estimate the direction the user wants to travel into based on their physical movements. The subtypes of *Direction* implement different ways of estimating this direction, based on different types of user movements:

HeadDirection allows users to move into the direction they are currently facing. For example, if *HeadDirection* is used with *WIPLocomotion* (i.e. a *HeadDirection* node is child of *WIPLocomotion*), then when a user starts walking on the spot, she is moved into the direction their headset is facing at a speed congruent with their walking.

TiltDirection allows users to move into the direction their head is tilted towards. There is a natural tendency of users to tilt their head forward when walking forward, i.e. to have their head angled slightly towards the ground. *TiltDirection* also can be used to move sideways, e.g. for "strafing" in a first-person game, by tilting the head sideways towards the shoulder. It can also be used to move backwards, by tilting the head back.

FootDirection estimates the direction of movement based on estimated movements of the feet. It is used only with the *WIPLocomotion* method, i.e. when the user is walking on the spot. The direction of locomotion is estimated based on the movement of the headset, which correlates with the movement of the feet during walking-in-place movements.

WholeBodyDirection estimates the direction of movement based on a variety of movement measures, including head direction, head tilt and foot direction. It is used only with the *WIPLocomotion* method. It aims to balance the different indicators of direction in order to provide a natural yet powerful experience and reduce estimation noise.

Every *Direction* node has the following properties:

direction is a read-only 3D vector describing the direction of locomotion estimated by a locomotion method.

lateral is a Boolean parameter indicating whether the user should be able to move only laterally (true), i.e. only left and right relative to the direction given in the *Locomotion* parent node, or into any direction (false). Restricting the direction to lateral movements can reduce VR sickness.

smoothing is a floating-point parameter describing how easily the estimated direction can fluctuate from moment to moment. This is based on an exponential average between the current and previous direction vectors. A *smoothing* of 0 will not apply any smoothing, i.e. estimated direction vectors can change quickly and potentially erratically. A *smoothing* greater than 0 and smaller than 1 will average between current and previous direction estimates. The higher the *smoothing*, the more gradual are the changes in direction produced by a direction estimation method.

3 Locomotion based on Physical Activity

The following sections describe the four methods of locomotion currently supported by the SDK: *WIPLocomotion*, *CyclingLocomotion*, *CrosstrainerLocomotion* and *RowingLocomotion*. *Locomotion* nodes provide vection by moving their children according to the speed estimated based on the respective physical activity, and the direction set either in the *direction* property of the node or a *Direction* child node (see chapter 4).

3.1 *WIPLocomotion*: Walking-In-Place

Although the physical activity of walking-in-place does not include forward movement, it closely resembles the normal walking cycle. It has been shown that walking-in-place feels natural and is a suitable method for locomotion that can create a fairly realistic experience of walking and running in VR[1]. It can be used in a fairly small VR play area, while exerting the player physically similar to real walking.

When using walking-in-place, some users accidentally step forward in the real world. Therefore, it is important to keep a sufficient distance to physical objects in the real world and to set up a virtual boundary that warns users before they step outside the play area. Also tactile cues on the floor can be used to remind players when they are leaving the play area, e.g. a carpet tile to walk on with edges that can be felt on the feet.

The *WIPLocomotion* node supports both walking and running on the spot. In both cases it estimates speed based on steps per minute. The *WIPLocomotion* node uses movements of the headset to estimate footfall, which is not as accurate as foot trackers but feasible as footfalls cause systematic movements of the head. This makes it possible to use the node with any VR headset, regardless of whether the feet are tracked directly.

3.2 *CyclingLocomotion*: Cycling on a Stationary Bike

Cycling on a stationary bike or exercycle is a suitable method for locomotion in VR[2]. Cycling can be performed from very low to very high levels of exertion, offering a wide range of physical health benefits. It is fairly safe in VR as the user remains stationary on the bike and can be seated. Furthermore, it is a good fit for VR experiences that are based on experiences of driving a vehicle, or riding a bike or motorcycle. *CyclingLocomotion* preserves the momentum of the bike when the user stops pedalling. Without pedalling, the speed decreases gradually until locomotion stops.

---

[1] Ioannou, C., Archard, P., O'Neill, E. & Lutteroth, C. (2019). Virtual Performance Augmentation in an Immersive Jump & Run Exergame. Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. ACM Press, 15 p.

[2] Barathi, S. C.; Finnegan, D. J.; Farrow, M.; Whaley, A.; Heath, P.; Buckley, J.; Dowrick, P. W.; Wünsche, B. C.; Bilzon, J. L. J.; O'Neill, E. & Lutteroth, C. (2018) Interactive Feedforward for Improving Performance and Maintaining Intrinsic Motivation in VR Exergaming. Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, ACM. 14 p.

Using this node, a pedalling user will move in VR as if the bike is a normal bike and not a stationary one. The estimated *speed* is proportional to the cycling cadence. The cycling cadence is detected based on movements of the headset, so it is not necessary for the bike to be instrumented. A disadvantage of this is that the *CyclingLocomotion* node is not aware of the current breaking force applied in the bike. This means that changing the resistance of the bike will not automatically change the speed of VR locomotion, given a constant cadence. The *multiplier* property can be used to adjust the speed according to the resistance level.

3.3 CrosstrainerLocomotion: Walking on a Crosstrainer or Elliptical Trainer

Crosstrainers are fairly safe for VR use because the user stays fairly stationary and can hold on to handlebars for stability. The *CrosstrainerLocomotion* node supports users walking on a crosstrainer based on the movements of the headset, which are correlated with the user's steps. Similar to *CyclingLocomotion*, this has the advantage that any crosstrainer can be used with this node and no instrumentation is required. It also shares the disadvantage that the node is not aware of the current resistance of the crosstrainer, so cannot take this into account when calculating locomotion speed. The *multiplier* property can be used to adjust the speed according to the resistance level.

3.4 RowingLocomotion: Using a Stationary Rowing Machine

Rowing on a stationary rowing ergometer is a physical activity suitable for locomotion as the user is seated and the feet are secured, providing some stability. Similar to cycling, rowing can cater for a wide range of exertion, and similar to a crosstrainer, it engages both the upper and the lower body.

Similar to *CyclingLocomotion* and *CrosstrainerLocomotion*, the node estimates rowing *speed* as strokes per minute based on movements of the headset. This means that any rowing machine can be used and no instrumentation is necessary. It also means that the node is not aware of the resistance of the rowing machine, so cannot factor that into the locomotion speed calculations. The *multiplier* property can be used to adjust the speed according to the resistance level.

4 Estimating the Direction of Locomotion

By default, *Locomotion* nodes produce vection in the direction given by their *direction* property. This is useful for VR experiences where the player moves primarily in a straight line. If a user should be able to move freely in any direction, such as in typical first person experiences, then a *Direction* node needs to be added as a child of the *Locomotion* node. This SDK is designed to support experiences where the user stays stationary in a fairly small play area, i.e. where the user does not really move in any particular direction but stays mainly in one place. This means that the direction of locomotion must be estimated from cues such as head and body movements. The following subtypes of node *Direction* implement different methods of estimating a locomotion direction based on such cues.

4.1 *HeadDirection*: Walking in the Direction of the Headset

HeadDirection is a simple and effective method for controlling the direction of movement by rotating the headset into the direction of travel. It is intuitive as it is direct and accurate as the headset direction can be measured by the VR tracking system. A user can look around when standing still, and they start moving as soon as the locomotion method is used that is parented to the *HeadDirection* node.

However, a drawback of *HeadDirection* is that users cannot move in one direction and simultaneous look into another. For example, it is not possible to move backwards or “strafe” sideways, or even look around you while walking in a certain direction. The other *Direction* nodes are designed to address these shortcomings, at the cost of accuracy.

4.2 *TiltDirection*: Walking in the Direction of Head Tilt

TiltDirection allows users to move into the direction their head is tilted towards. There is a natural tendency of users to tilt their head forward when walking forward, i.e. to have their head angled slightly towards the ground in the direction of travel. This means that controlling the direction by tilting is at least somewhat intuitive, although not as easy to use as head direction.

TiltDirection also can be used to move sideways, e.g. for “strafing” in a first-person game, by tilting the head sideways towards the shoulder. It can also be used to move backwards, by tilting the head back. Unlike *HeadDirection*, *TiltDirection* decouples the direction of movement somewhat from the direction the user is facing towards: the user can, for example, look forward but move sideways. However, it is less predictable and intuitive compared to *HeadDirection*.

4.3 *FootDirection*: Walking in the Direction the Feet are Pointing at

FootDirection estimates the direction of movement based on estimated movements of the feet. It is used only with the *WIPLocomotion* method, i.e. when the user is walking or running on the spot, as the feet cannot easily be turned in the other supported physical activities. The direction of locomotion is estimated based on the movement of the headset, which correlates with the movement of the feet during walking-in-place movements.

FootDirection is fairly intuitive for normal walking, as our feet are naturally pointing in our direction of travel (when taking the average direction of the two feet). It allows users to look around while walking in a certain direction. However, it does not support special walking movements such as walking sideways (“strafing”) or walking backwards. Also, as the direction of the feet is only edtimated indirectly from the movement of the headset, it is not as accurate as, for example, *HeadDirection*.

4.4 *WholeBodyDirection*: Using Multiple Measures to Estimate Direction

WholeBodyDirection estimates the direction of movement based on a variety of movement measures, including head direction, head tilt and foot direction. It can only be used with *WIPLocomotion*. It aims to balance the different indicators of direction in order to provide a natural yet powerful experience. It also aims to reduce estimation noise by combining different measures.

WholeBodyDirection supports walking in a certain direction while looking into another direction. To achieve this, the direction of the feet are estimated similar to *FootDirection* and combined with other indicators such as head tilt. It also supports walking sideways (“strafing”) and backwards based on head tilting movements, as in *TiltDirection*. While *WholeBodyDirection* supports a variety of movements, it is less predictable and less accurate than simpler direction estimators such as *HeadDirection*. We are continuously improving *WholeBodyDirection* to deliver a rich and accurate experience of free walking in place in first-person virtual environments.

5 Application Examples

In the following, we present examples of how the SDK for integration of physical activities into VR experiences can be used. We briefly describe the respective VR experience, and then detail how *Locomotion* and *Direction* nodes can be used to integrate physical activities accordingly.

5.1 *Race Yourselves*: An Exercycle Racing Game

Image 2 *Screenshot of the "Race Yourselves" VR exercycle racing game. Left: The user is riding a stationary exercycle. Right: While racing along a straight road, previous racing attempts are visualised as avatars to compete against.*

Image 2 shows the game "Race Yourselves"[3], which is played by riding a stationary exercycle. The user races along a straight road and competes against VR avatars. These avatars are in fact recordings of previous racing attempts of the user, so the user can use them to measure her exercise progress. With regular gameplay, the user will get faster and overtake her own avatars.

Cycling always moves the user forward in the game, along the straight road. Furthermore, the user can move laterally to the left and right by tilting their head left and right, respectively. This is achieved in the following way: A CyclingLocomotion node is set up to move the player according to the pedalling speed of the exercycle. To achieve this, the node representing the player with her bike in VR is added as a child of the CyclingLocomotion node. The direction vector of CyclingLocomotion is set to (0, 0, -1), i.e. straight ahead along the road. To enable players to move sideways by tilting their head, a TiltDirection node is added as a child of CyclingLocomotion. The lateral property of TiltDirection is set to true, making sure that the tilt of the head results only in lateral left-and-right

[3] Michael, A. & Lutteroth, C. (2020). Race Yourselves: A Longitudinal Exploration of Self-Competition Between Past, Present, and Future Performances in a VR Exergame. Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, ACM, 17 p.

movements,. This means the bike will always keep pointing forward along the road, which helps to avoid VR sickness.

5.2 *Virtual Performance Augmentation*: Walking, Running and Jumping

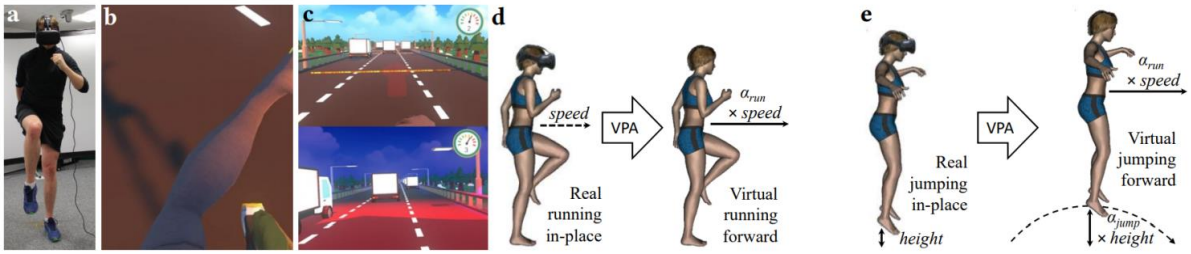

Image 3 Screenshot of the "Virtual Performance Augmentation" prototype. (a) The user is walking, running and jumping on the spot. (b) User movements are reflected in VR. (c) Day and night scenes prompt the user to either walk/run slowly or at a high intensity. (d) Running in place movements are translated into virtual forward movements. (e) Jumping on the spot is translated into forward jumping, based on the speed of running-on-the-spot before a jump.

Image 3 shows a VR prototype for virtual performance augmentation[4], i.e. a system that augments performance virtually: the system conveys users the experience of being able to run faster and jump higher that they actually can. Users walk or run along a straight road by walking or running on the spot. They move left and right in their play area to move left and right in the game, avoiding obstacles. And they jump on the spot to jump over "lava gaps" in the road. The game amplifies their running speed as well as their jumping height.

This is achieved as follows: Walking and running on the spot are detected with a *WIPLocomotion* node. The *direction* property of WIPLocomotion is set to vector (0, 0, -1), i.e. straight ahead along the road. The node representing the player is a child node of *WIPLocomotion*, so gets moved accordingly. The multiplier property of *WIPLocomotion* is used to amplify walking and running speed with values greater than 1, so the user's performance is augmented. No *Direction* node is used; instead, real sideway movements in the play area result in respective sideway movements in the game, so the player can evade obstacles. Jumping is supported with custom code, which reads the estimated walking/running speed from *WIPLocomotion* and, when the user jumps, applies the speed right before the jump as a forward vector to the jump itself.

[4] Ioannou, C., Archard, P., O'Neill, E. & Lutteroth, C. (2019). Virtual Performance Augmentation in an Immersive Jump & Run Exergame. Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. ACM Press, 15 p.

5.3 *Savannah Walk*: Free Walking

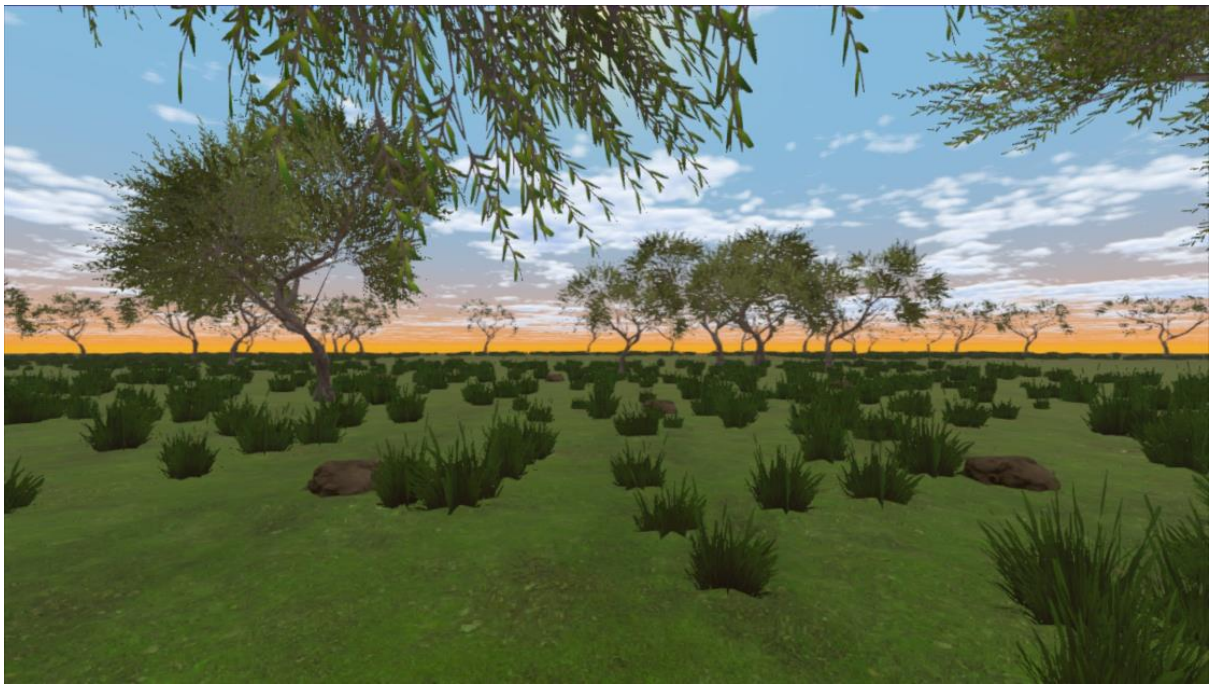

Image 4 *Screenshot of the "Savannah Walk" prototype. The user can travel freely through the virtual environment by walking in place.*

Image 4 shows the "Savannah Walk" prototype, which is a virtual nature walk experience. The user can travel through the virtual landscape by walking in place and turning in the play area. This is achieved with a *WIPLocomotion* node, which is the parent of the node representing the player and also the parent of a *WholeBodyDirection* node. The direction property of *WIPLocomotion* is not used. The *lateral* property of *WholeBodyDirection* is set to false, allowing for movements in all directions.

6 Conclusions

We have presented an SDK for the integration of physical activities into VR experiences. This SDK is maintained for the Unity and Godot game engines, supporting a wide variety of VR hardware through the OpenVR and OpenXR standards. The SDK allows developers to control locomotion in a VR experience based on a physical activity such as walking, running, cycling, using a crosstrainer, and rowing. The SDK is able to detect physical activities purely based on headset movements. Furthermore, it enables users to control the direction of locomotion based on different natural input methods (i.e. methods based on body movements). The core of the SDK is formed by *Locomotion* and *Direction* node types, which can be inserted into the scene tree of a VR scene. The way physical activity is integrated into a VR experience can be configured by combining different *Locomotion* and

Direction nodes and adjusting their properties. As a result, physical activity can often be integrated into many VR experiences without much design effort.