R3-A4. Production of an adaptive and immersive virtual reality (VR) training pathway.

Erasmus+

INTRODUCTION

The move towards more inclusive and adaptive training in the natural stone sector is taking shape with the creation of a highly immersive virtual reality (VR) learning path. This project is a vital component of integrating advanced technologies into training, thus facilitating a personalized and effective learning environment.

Designing this educational path requires a detailed development process that ranges from the identification of key scenarios to their implementation in a VR environment. It starts with a research and analysis phase, where the most representative situations and tasks of the sector are carefully selected. Then, we proceed to an instructional design that takes into account both functionality and accessibility, ensuring that each stage of the training is relevant and achievable for all users.

Through this approach, VR scenarios are intended to not only accurately simulate working conditions, but also be tuned to meet the specific educational needs of users. The resulting immersive experience aims not only to instruct, but also to empower users, enabling them to develop practical skills in a safe and controlled environment, ready for real-world application.

In the end, this educational tool not only becomes a bridge to job competence, but also a means to foster greater understanding and acceptance of diversity in the workplace.

This report and all the information about the project are available on the InclusiveStone website: https://inclusivestone.eu/

Content

1. TOOL DEVELOPMENT

The core of this project is based on the immersive capacity offered by Virtual Reality. Currently, the standard for the creation of applications in this sphere is based on the use of graphics engines originating from the world of video games. For this reason, the approach adopted for the development of our educational tool follows a methodology similar to the creation of a video game.

This methodology is structured around three fundamental pillars that will be broken down in detail in the following sections:

- **Scene Design**
- **Development of 3D environments.**
- **Development of functionality and immersive experience.**

To carry out the development efficiently and effectively, it should be noted that different technologies have been used:

- **Blender:** A free, cross-platform software tool, used for a wide range of 3D graphics functions such as modeling, animation, and rendering. Its open-source nature and extensive community make it easy to learn through tutorials and extensive documentation.
- **Unity:** This graphics engine, also open source, is widely used in video game development. It stands out for its scalability and the possibility of customization through scripts, supported by a robust community and complete documentation.
- **Oculus Quest:** An easy-to-access and easy-to-use Virtual Reality device, which has become one of the essential supports of Meta's vision for the future. It offers the advantage of being wireless and has a seamless integration with Unity, thus enhancing accessibility and versatility in VR application development.

2. SCENE DESIGN

This procedure involves the detailed planning of the sequence of steps that the user will perform from the beginning of their interaction with the tool to the moment they reach a full immersive experience. In short, it's about the creation of the script that will guide the comprehensive development of the app.

Initially, 6 work scenarios were selected that adequately represent the various roles identified in the R1 work package, due to their adaptability and the fact that they do not pose a danger to people with different disabilities. The situations chosen are as follows:

- **Forklift Operator - Freight Transport:** In the context of this task, the objective will be to load and unload three pallets of marble slabs, moving them from the processing plant to the designated storage area. This activity will involve manoeuvring in areas with limited space and stacking materials of significant weight.
- **Forklift Operator - Truck Loading:** In this specific scenario, the goal is to load a transport vehicle with three pallets of marble slabs, starting from the storage area. The operation will require the execution of maneuvers in confined space areas and the stowage of heavy cargoes.
- **Overhead crane operator - Board handling:** In this task, the user will aim to move 2 marble slabs to the illuminated storage area, performing the necessary manoeuvres and complying with safety regulations.
- **Overhead crane operator - Block handling:** In the development of this activity, the user's goal will be to transport a block of marble from the truck to the loading area of the loom, executing the maneuvers that are required and rigorously observing the current safety regulations.
- **Cleaning Operator:** The purpose of this activity is to identify and select the appropriate Personal Protective Equipment (PPE) for cleaning and sanitizing the work plant, eliminating debris and debris. This must be carried out responsibly and with the utmost respect for the environment.
- **Waste management:** In the development of this task, the user is responsible for managing chemicals and waste. This includes the appropriate selection of techniques and tools for recycling and the correct disposal of waste, as well as the following of guidelines aimed at ensuring responsible and environmentally sustainable waste management.

In this context, it was decided to design a scenario model composed of sequential stages or missions, in the style of video games; that is, you can only advance to the next step once you have completed the previous one, and in this way progressively. Thus, the need to finish the entire process safely from start to finish is imposed on the user, and any inability to do so is qualified as a failure in the scenario.

Within this set of stages, the initial one will consist of a detailed tutorial about the specific machine or task. In this way, it will be easier for the user to learn the controls corresponding to each situation, allowing a greater fluidity in the operation without the need to check them frequently. To continue with the specific tasks, which will depend on each situation.

2.1. Tutorial

In the course of the design phase of the VR app, we have implemented a fundamental element for the orientation and preparation of the player: the tutorial. This component is essential, as it provides a structured and methodical introduction to the game environment, available tools, and mission mechanics.

The tutorial panel was meticulously developed to ensure a comprehensive learning experience. It started with the creation of a detailed script that breaks down each critical step that the player must understand before embarking on the mission.

The first section of the tutorial covers the handling and functionalities of the machines that players will operate during the game. Subsequently, the tutorial introduces the teleportation tool, a crucial mechanic that expands the strategic and mobility possibilities within the game. Specific exercises have been developed that allow players to experiment with this tool in a controlled environment, ensuring their mastery before facing real-time situations.

In addition, the dashboard includes informational modules on the use of any additional functionality that is relevant to the mission. Each element has been carefully explained and accompanied by practical examples to ensure a holistic understanding of its use and application.

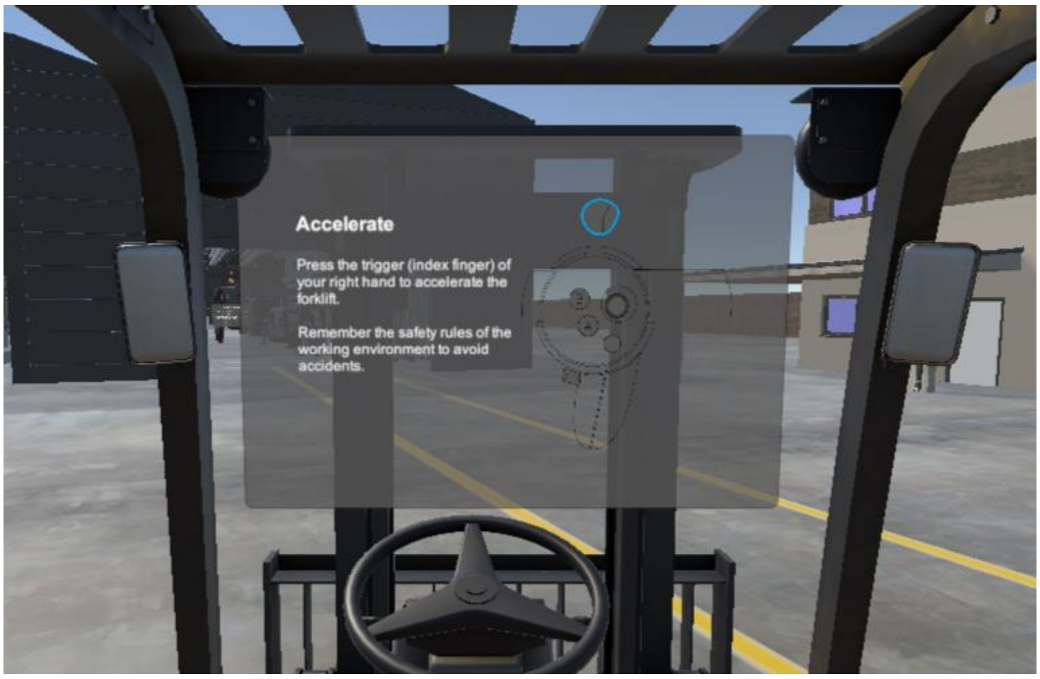

Accelerate
Press the trigger (index finger) of your right hand to accelerate the forklift.
Remember the safety rules of the working environment to avoid accidents.

Figure 1: Panel tutorial

The realization of this tutorial panel has been one of the cornerstones of the development of our video game, ensuring that each user is adequately equipped with the knowledge and experience necessary to fully enjoy the game and succeed in their missions, and, therefore, in the actual work.

2.2. Missions

In the development of our project, we have implemented a mission system that is fundame ntal for the user's progression in learning the job. This system is meticulously designed to s uit different roles and jobs.

Missions fall into two main categories that contribute to the diversity and richness of the to ol:

- **Action Missions**: These missions are at the core of the interactivity within the game. They require the player to be actively involved, completing tasks that can include an ything from operating machinery to using hands for picking up and controlling clea ning. These missions are designed to test the user's motor and strategic skills.
- **Q&A Missions**: In these missions, the player faces situations that require reflection and decision-making based on the information received in the courses that the user must have in order for this app to be really useful. Through Q&A panels, these missi ons assess the user's understanding and acquired knowledge.

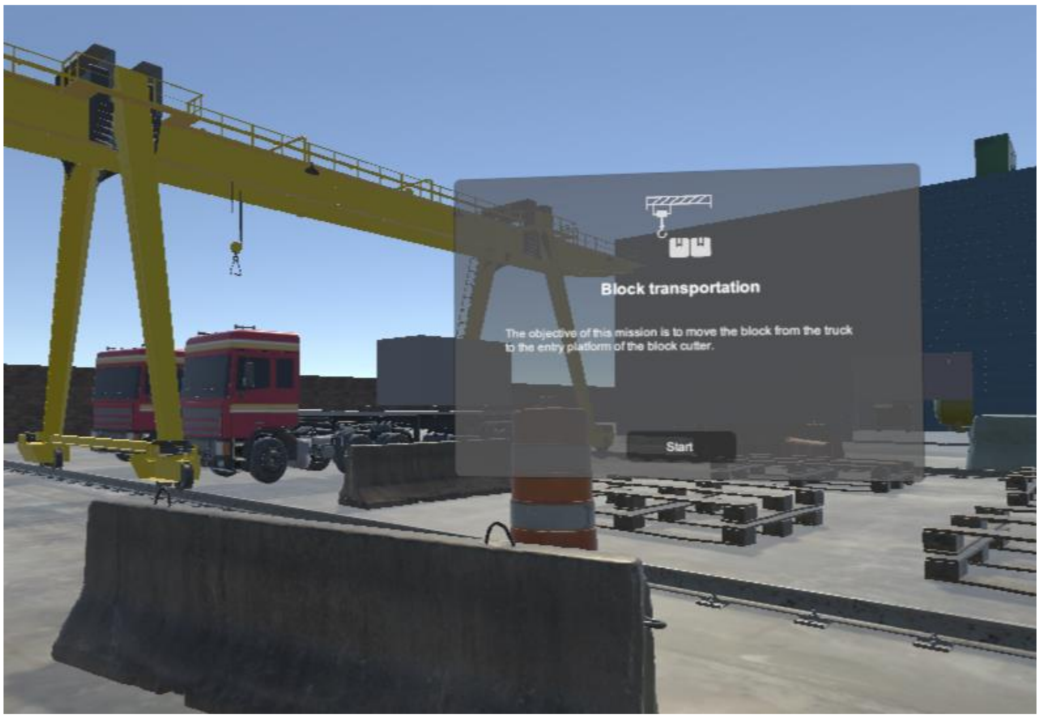

Block transportation
The objective of this mission is to move the block from the truck to the entry platform of the block cutter.
Start

Figure 2:Block Transport Mission

Each mission, regardless of category, is carefully intertwined with the next, ensuring that eac h player's experience is cohesive and deeply immersive. Additionally, this mission design pr omotes a natural learning curve, where players can improve their skills as they progress thro ugh the game.

In short, our quest system not only seeks to entertain, but also to engage the player in a pr ocess of continuous learning and discovery, which is essential to the all-round experience w e wish to offer.

3. DEVELOPMENT OF THE 3D ENVIRONMENT

The creation of tools within video game graphics engines often involves the handling of obj ects. Therefore, the second main task is the production of the various objects or assets, whic h are specified both directly and indirectly in the technical scripts previously prepared. This p rocedure is divided into three essential phases:

- **Modeling**: A mathematical representation of an object is constructed in three dime nsions through specialized software. For this project, we have chosen to use Blender .
- **Rigging**: At this stage, the constituent parts of an object are broken down to form a controllable digital skeleton, making it easier to make smooth and precise animatio

ns. This step is crucial for objects that will require animation and has also been carri ed out using Blender.

- **Texturing**: During texturing, colors and details are applied to 3D models to give the m a more realistic and detailed final look, once again using Blender software.

The range of objects made for this project covers a wide and varied spectrum, ranging from elements of extreme simplicity to complex and detailed machinery. The objects produced ha ve been organized into different categories, according to their complexity and function with in the game. We can find:

- Containers, buildings, shelves and decorative elements.

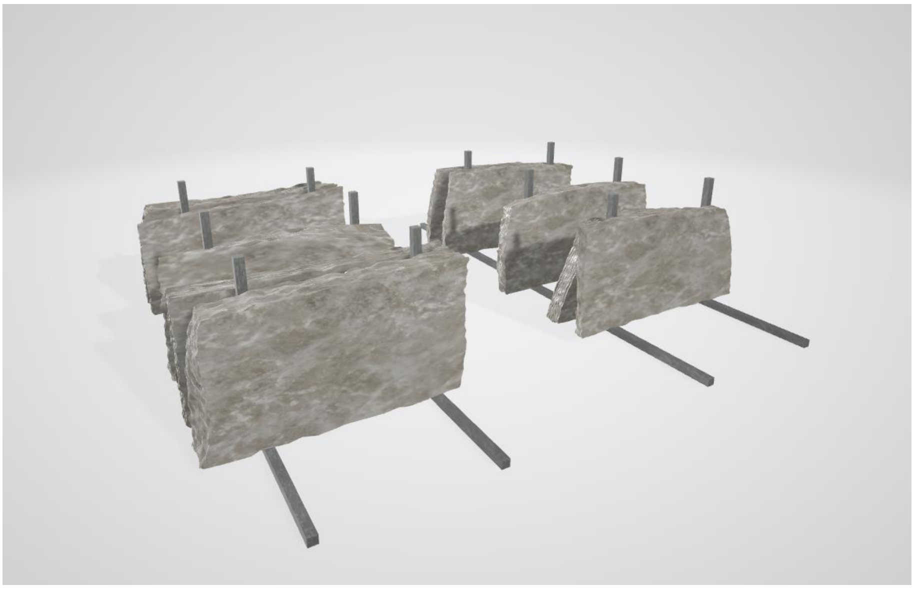


Figure 3: Stone sheets.

- Personal equipment and useful objects, with some functionality.

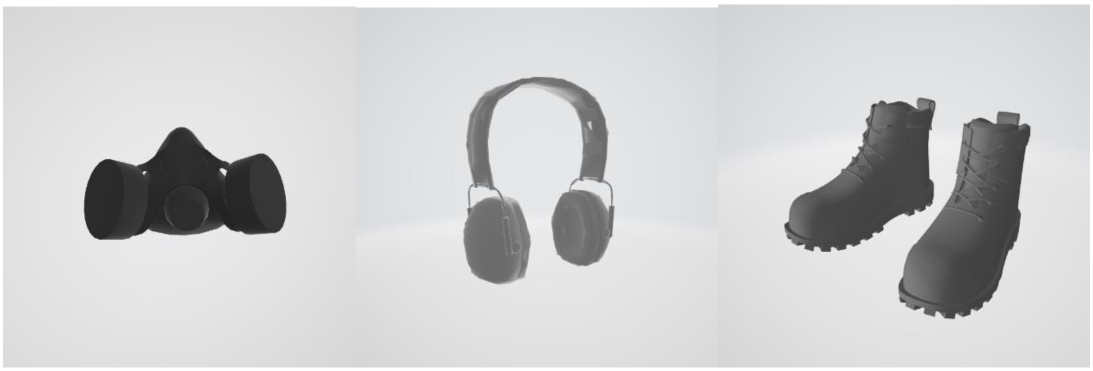


Figure 4: Other elements.

- Specific machines for different situations.

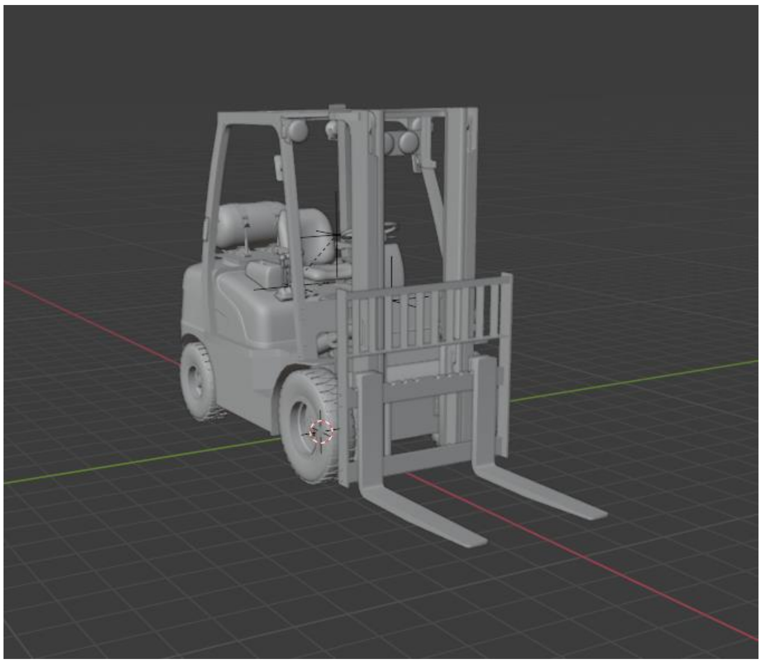


Figure 5: Forklift Model

Once the objects that will be used in the game have been designed and created, the next step is the preparation of the scenarios. This process is crucial for the development of the game's functionality to accurately reflect the environment in which the final action will take place. Importantly, this process is iterative and subject to adjustment, as subsequent testing may reveal the need for modifications. For the creation of the scenario, the Unity graphics engine has been used, which will be the same one that will be used for the production of the final application.

Since the jobs that were chosen in previous tasks are positions that are carried out in the factory and not in the quarry, only one scenario has been developed. The scenario designed for this project is as follows:

- **Natural Stone Factory**: The stage is a virtual representation of a real factory, with cl early differentiated areas including the engine room, the chemical warehouse and t he waste extraction section. This detailed design helps users become familiar with a specific work environment and the various tasks carried out in the natural stone ind ustry.

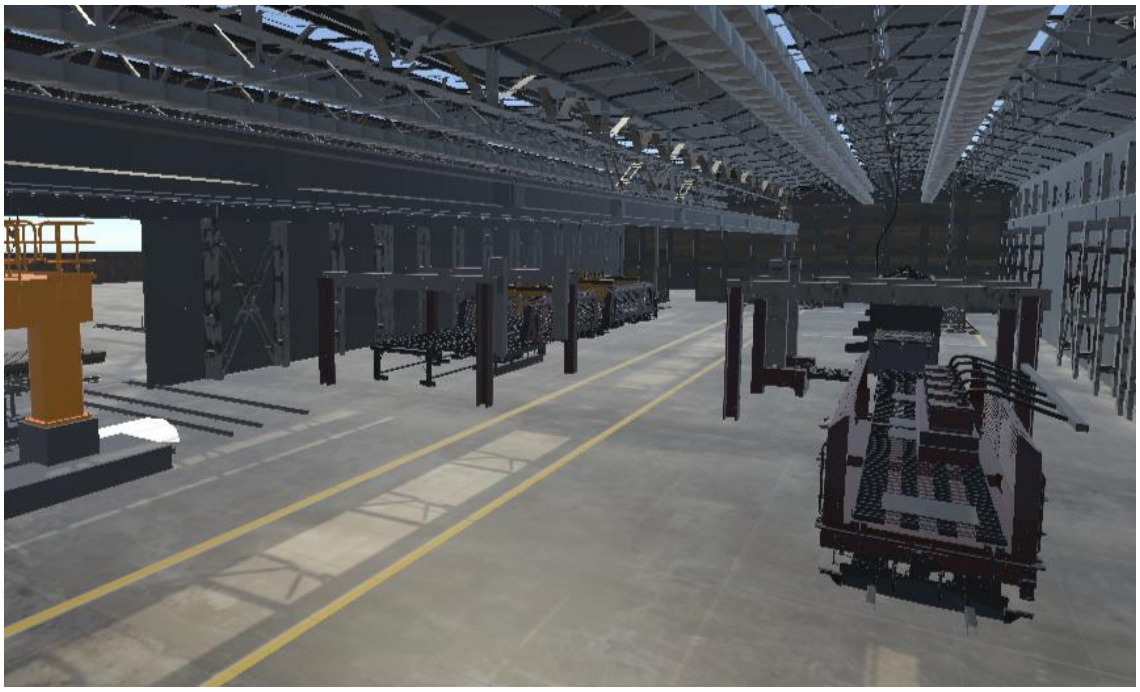


Figure 6: Factory Stage

4. FUNCTIONALITY DEVELOPMENT AND IMMERSIVE EXPERIENCE

Functionality development is the step in which previously collected concepts and information come to life. The purpose of this phase is to turn the theoretical tool into a practical and operational application. To carry out this task, we have selected the Unity graphics engine due to its ease of incorporating Virtual Reality elements, as well as the extensive support offered by its user community.

In Unity, projects are structured into Scenes, which can be connected through a series of events. Each Scene is made up of several objects, and these are assigned specific components that define their attributes and behaviors within the game. The functionality development process is extensive and can be broken down into several key stages, including **Tool Design and Architecture**, **Data Modeling** to structure information, **SDK Integration** to expand engine capabilities, and **Event Management** for an interactive and consistent game flow.

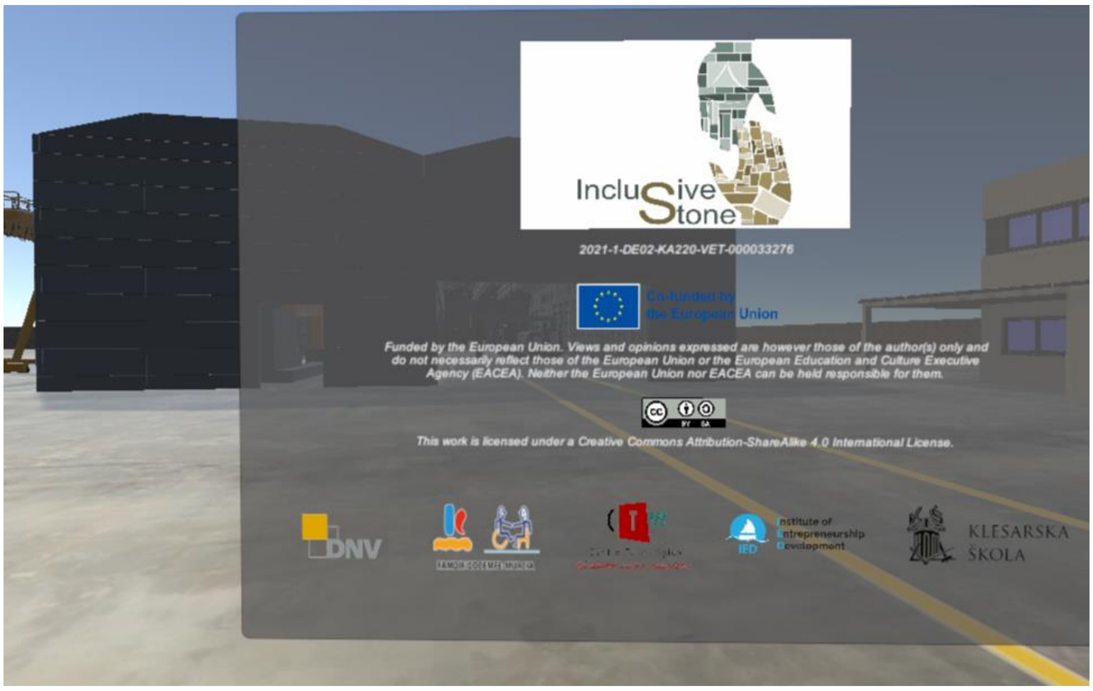
4.1. Design & Architecture

Design and Architecture involve the detailed elaboration of the previously developed script, with the aim of meticulously defining the path that the user must follow and the decisions that will have to be made in each scene or screen of the game. This process is critical, as it establishes the base structure on which the user experience will be built. During this stage, the navigation logic, the arrangement of the interactive elements, and the sequence of events that will guide the user throughout the game are taken into account. Interaction with each object and character within the virtual environment is meticulously planned, as well as transitions between scenes, to ensure an intuitive and immersive experience that keeps the user engaged with the narrative and objectives of the game.

Within the project, users will encounter two types of main Scenes designed in Unity: the Menu Scene and the Specific Scene for each workstation.

4.1.1. Name principal

The Menu screen is presented as an intuitive and accessible introduction to the virtual environment, where users take their first steps in the tool. When starting out, they are faced with a fundamental choice: language settings, ensuring that the experience is personalized and understandable to them. Once this basic preference is established, the screen gently guides the user towards understanding the main purpose of the scene, which is the choice of a specific work situation to explore. Although interactivity at this stage is deliberately kept restricted to keep the focus on the learning objective, the experience is designed to be welcoming and orienting, preparing the user for the transition to the next stage, where the selection made in the menu will determine the specific Work Scene to be accessed next.


InclusiveStone
Co-funded by the European Union
DNV
FAMDIF COCEMFE MURCIA
Institute of Entrepreneurship Development
IED
KLESARSKA ŠKOLA

2021-1-DE02-KA220-VET-000033276

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

4.1.2. *Work Status Scene*

This phase represents the core of the tool, where each proposed work situation comes to life in its own individual Scene. Following the previously established mission scheme, users are immersed in a series of tasks designed to emulate the responsibilities and challenges of a specific job position. The adventure begins with a detailed tutorial, meticulously tuned to the task at hand, providing a solid foundation for learning and executing subsequent missions.

As the user progresses and passes each mission, the next one is unlocked, allowing for linear progress and a sense of constant accomplishment. Upon successfully facing and completing the last mission, you are presented with the option to restart that Scene and hone your skills, or return to the main menu to explore and learn about another job, thus facilitating a continuous learning process and a broader exploration of the various facets of the natural stone industry.

4.2. Data Modeling

The goal of this phase is to create an overarching framework for data storage and retrieval. We have chosen to use PlayerPrefs, a Unity module that facilitates the conservation of information through a system of keys associated with the name of the project. In Oculus applications, these variables are saved in the form of an XML file, located in the /data/data/pkg-name/shared_prefs/pkg-name.v2.playerprefs.xml directory, where "pkg-name" corresponds to the name assigned to the application. Unity makes accessing this data extremely easy through the functions of the PlayerPrefs class.

Because of how this app is structured, there will be a common variable that we'll need to store: the one that determines the language. We'll use the following PlayerPrefs features to manage it:

- **SetString (*name, value*):** With this function we will assign a text value, which will be the ISO code of the chosen language, to the variable identified by "name".
- **GetString (*name*):** This function allows us to retrieve the value assigned to the "name" variable. We will use it in each scene to identify the selected language.

4.3. SDK Integration

The creation of Virtual Reality applications is based on three fundamental pillars: a compatible device that can be connected to a computer, a video game graphics engine and a Software Development Kit (SDK). At the beginning of this chapter, the first two elements were detailed, with Oculus being the device and Unity being the chosen graphics engine. As for the SDK, which is the set of tools that assists software developers in the creation process, there are several options available to suit Unity. Although there is a specific one for Oculus, the XR Interaction Toolkit has been selected, a package developed by Unity that makes it easier to adapt the project to different types of devices. This means that, although the tool is initially developed for Oculus

Quest, it can be installed on any device that is compatible with Unity, thus expanding its accessibility and reach.

To integrate the SDK and enable interaction in Unity, follow the steps below:

1. When you start Unity, you create a new project using the Universal Render Pipeline template, which is key to optimized graphics, a crucial aspect for a smooth and satisfying VR experience.
2. The SDK, in this case, the XR Interaction Toolkit, is installed via the Package Manager window found in the Window menu option.
3. It is necessary to specify the type of device on which the project will be developed. To do this, check the box VR supported within Project Settings/Player.

With these steps, your Unity project is ready for Virtual Reality development. However, to begin interacting your device with the graphical environment, it is essential to follow the instructions in the next section.

4.4. Event Management

Event management is arguably the broadest part in creating our app's functionality. It is based on the use of the components that Unity offers, along with the creation of concise scripts that enable the desired interaction between the user and the system. Since each scene is packed with unique details that affect how you interact with it, this section could be extended considerably. However, in this document we will focus only on highlighting those common and crucial aspects that are essential to achieve our interactivity goals.

4.4.1. Setting the Scene for VR Interaction

Once you have your Unity project and graphically designed scenarios ready, the next step is to prepare the scene for the interaction device to integrate with a camera, making it easier to test during development. This procedure, described in the following stages, must be executed for each of the scenes in the project:

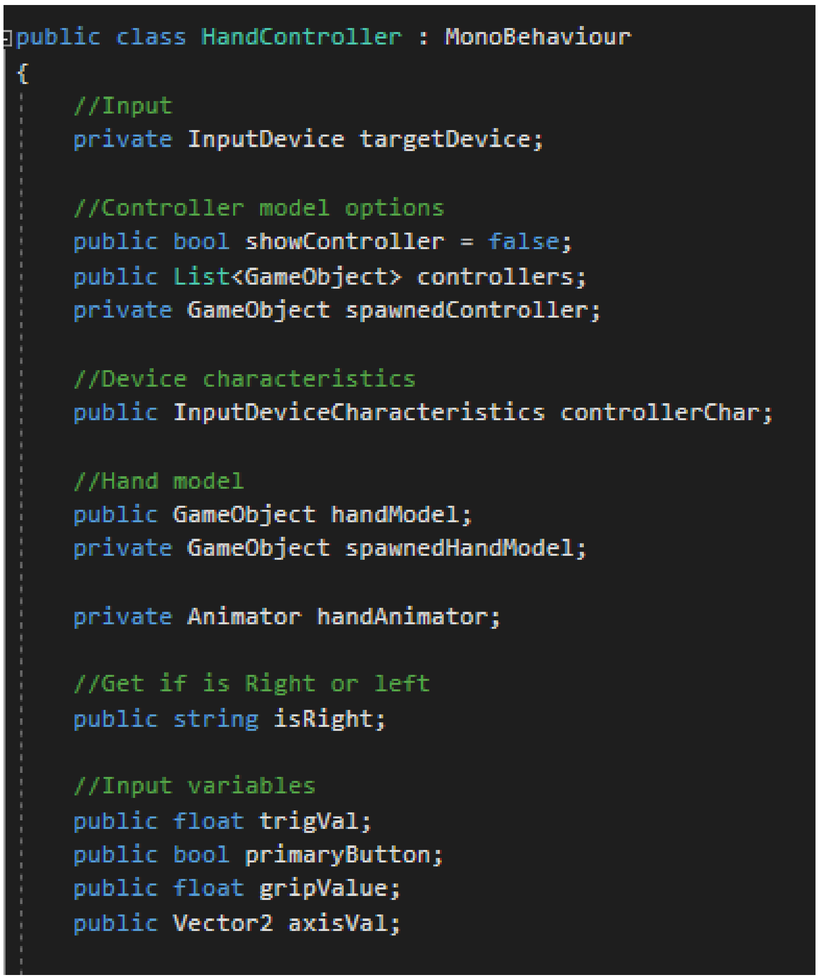
1. Removes the default camera that appears by default in the scene.
2. Create an empty object and add an XR Rig component to it, which will function as the core of the interaction between the user and the machine. We'll name this object VR-Rig.
3. Inside the VR-Rig, it creates an empty child object that will serve as the initial reference point for user interaction. We'll call it Camera Offset.
4. Add a camera as a child of the Camera Offset object and add a Tracked Pose Driver component to this camera. This element will act as our virtual eyes and the component will allow for the rotation of the view.
5. Configure the variables of the XR Rig component (parent, VR-Rig) with the reference position and camera you just created. Set the 'floor' value in the Tracking Origin Mode attribute so that the camera automatically adjusts to the user's height.

6. To enable driver mobility and visibility, create two additional objects as children of the Camera Offset and endow them with the XR Controller attribute.
7. Assign the model you want to use as controllers in the Model Prefab attribute of each XR Controller.
8. Finally, it adds the XR Direct Interactor component to these two objects, which will provide the ability to interact with other objects in the scene.

4.4.2. Controller Input

This process refers to the capture of the values emitted by interacting with the buttons on the controls. These values are crucial for performing a variety of actions within the app, such as grabbing objects or selecting different options. Before proceeding with any operation with this data, it is essential to understand the type of information that each input from the controls provides. This information can be viewed in the XR Interaction Debugger window, accessible via Window/Analysis.

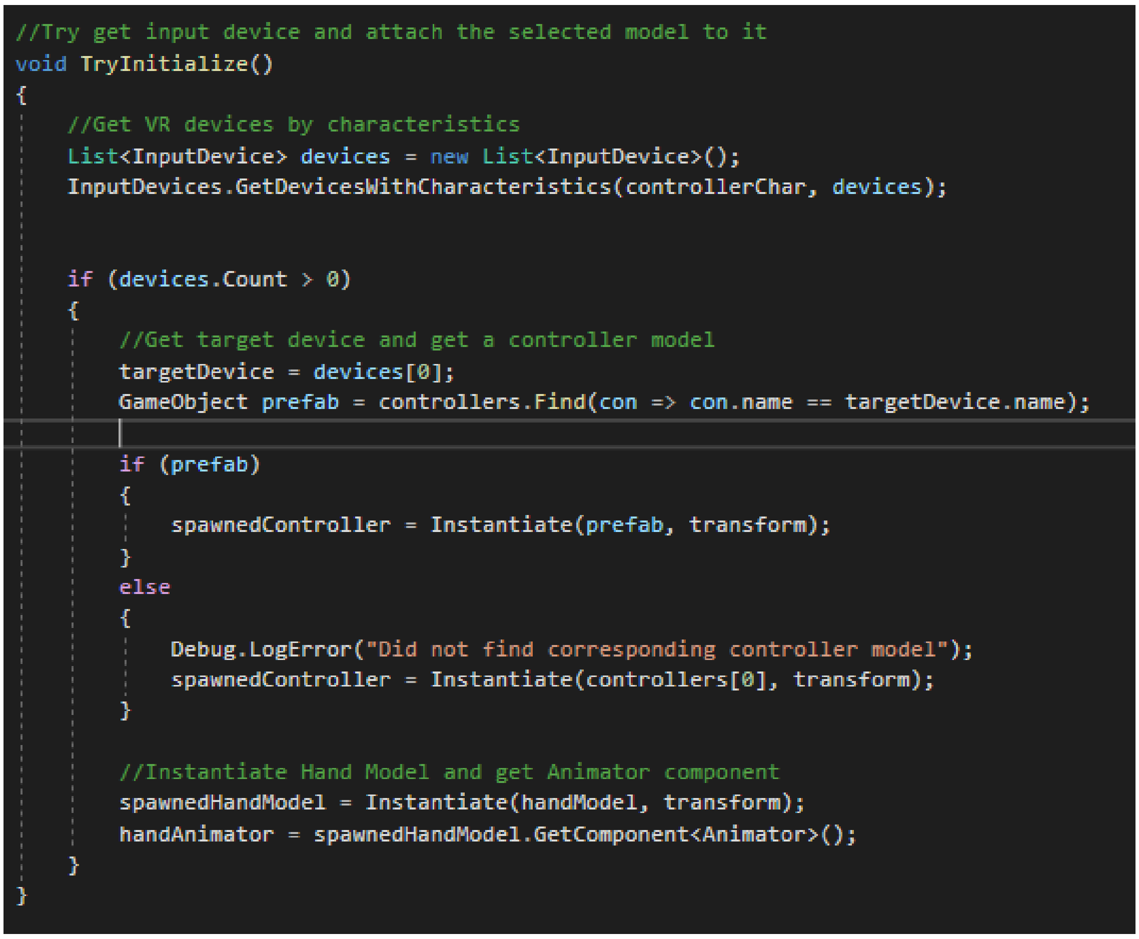
Once you have this understanding, the next step is to develop a script that allows you to collect these values. The class created for this purpose is called `the HandController` and is attached as a component to each of the 3D models specified in the Model Prefab attribute of the controls. Highlights of the code include:



```
public class HandController : MonoBehaviour
{
    //Input
    private InputDevice targetDevice;

    //Controller model options
    public bool showController = false;
    public List<GameObject> controllers;
    private GameObject spawnedController;

    //Device characteristics
    public InputDeviceCharacteristics controllerChar;

    //Hand model
    public GameObject handModel;
    private GameObject spawnedHandModel;

    private Animator handAnimator;

    //Get if is Right or left
    public string isRight;

    //Input variables
    public float trigVal;
    public bool primaryButton;
    public float gripValue;
    public Vector2 axisVal;
```

Figure 7: Clase HandController

1. At the beginning of the script, you can see the declaration of the variables. Those that are public can be modified and will act as attributes of the component.
2. When you create a new script, Unity automatically generates two functions: `Start`, which runs only once at the beginning when the component is instantiated, and `Update`, which is performed repeatedly throughout the component's lifecycle.
3. The first function to be triggered within this component is `TryInitialize`, which identifies the VR device by its specific characteristics, such as whether the controller is right or left, using the `GetDevicesWithCharacteristics` function of the `InputDevices module`.



```
//Try get input device and attach the selected model to it
void TryInitialize()
{
    //Get VR devices by characteristics
    List<InputDevice> devices = new List<InputDevice>();
    InputDevices.GetDevicesWithCharacteristics(controllerChar, devices);


    if (devices.Count > 0)
    {
        //Get target device and get a controller model
        targetDevice = devices[0];
        GameObject prefab = controllers.Find(con => con.name == targetDevice.name);

        if (prefab)
        {
            spawnedController = Instantiate(prefab, transform);
        }
        else
        {
            Debug.LogError("Did not find corresponding controller model");
            spawnedController = Instantiate(controllers[0], transform);
        }

        //Instantiate Hand Model and get Animator component
        spawnedHandModel = Instantiate(handModel, transform);
        handAnimator = spawnedHandModel.GetComponent<Animator>();
    }
}
```

Figure 8: TryInitialize function

4. The `Update` function is then executed, invoking the `UpdateEvents` and `UpdateAnimations functions`.
5. `UpdateEvents` collects the values emitted by the controls using the `TryGetFeatureValue` function and stores them in the pre-declared public variables.

6. `UpdateAnimations` is a function designed to activate animations corresponding to ea ch button, in the case of having a rigged hand model. Although this function is not crucial in the task of acquiring values, it contributes to the interactivity and realism of actions.

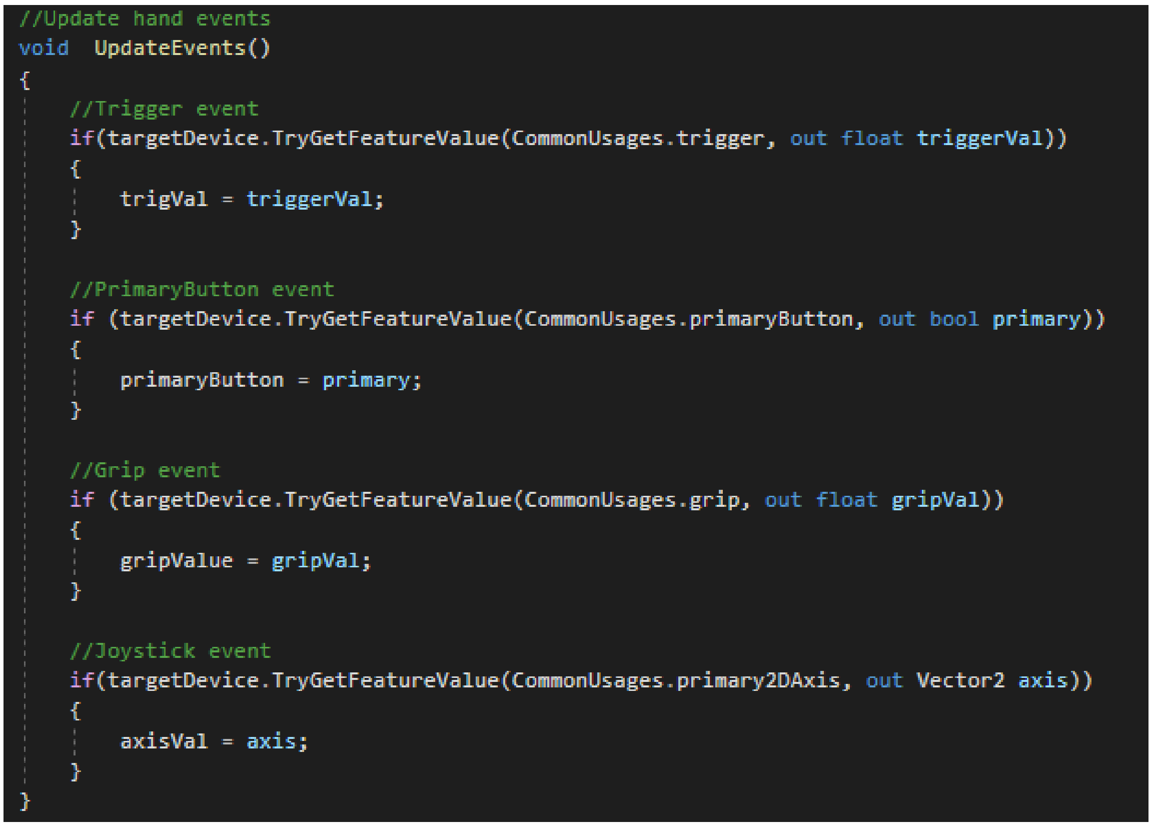

```
//Update hand events
void  UpdateEvents()
{
    //Trigger event
    if(targetDevice.TryGetFeatureValue(CommonUsages.trigger, out float triggerVal))
    {
        trigVal = triggerVal;
    }

    //PrimaryButton event
    if (targetDevice.TryGetFeatureValue(CommonUsages.primaryButton, out bool primary))
    {
        primaryButton = primary;
    }

    //Grip event
    if (targetDevice.TryGetFeatureValue(CommonUsages.grip, out float gripVal))
    {
        gripValue = gripVal;
    }

    //Joystick event
    if(targetDevice.TryGetFeatureValue(CommonUsages.primary2DAxis, out Vector2 axis))
    {
        axisVal = axis;
    }
}
```

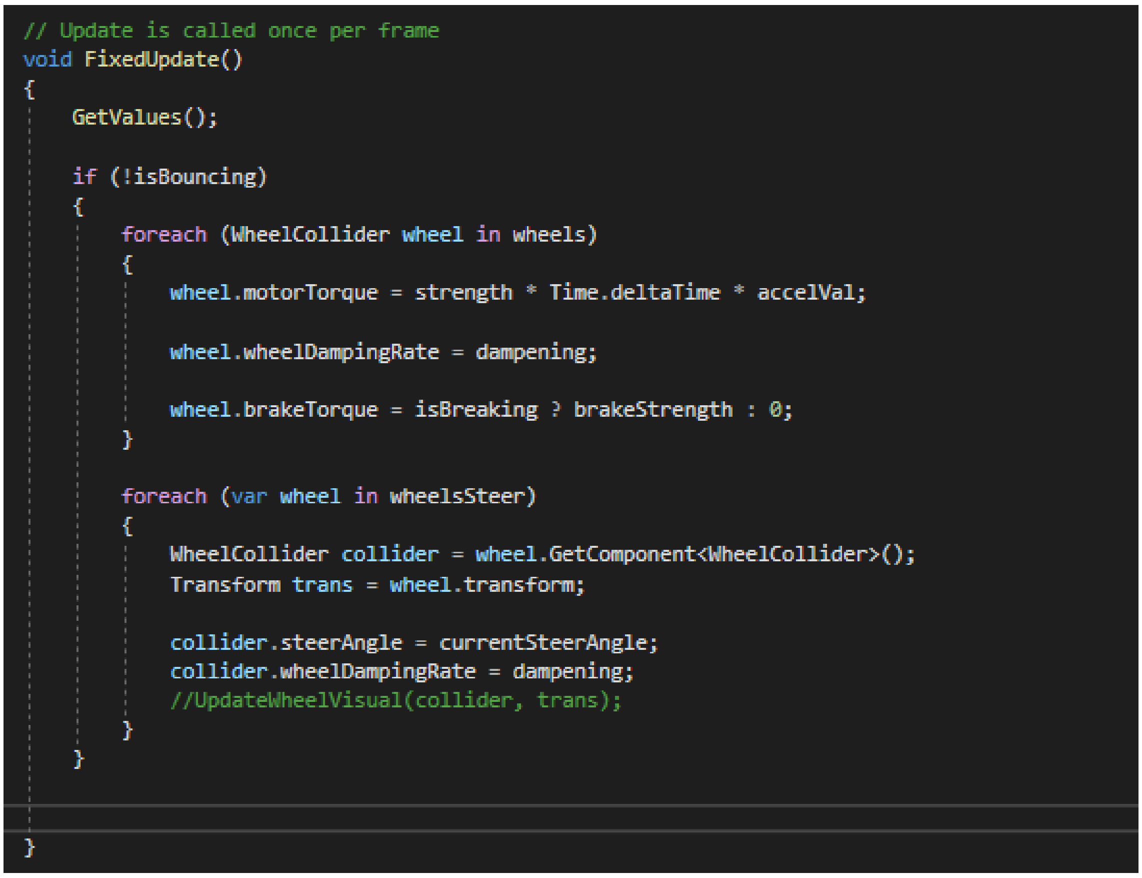
Figure 9: Función UpdateEvents

4.4.3. Forklift control

To achieve a convincing and functional virtual reality experience when operating a forklift, it is essential to have a series of detailed and specialized scripts. These scripts are responsible for providing the precise interactivity and control that are required to effectively simulate the operation of this type of machinery. Within the VR tool, a primary script dedicated to driving will be required, which will allow the user to maneuver the forklift within the virtual environment. In addition, several additional scripts will be required focused on the control of the levers and other mechanisms of the truck, ensuring that all possible actions in the actual operation can be replicated virtually.

Next, we will proceed to explain each of these scripts in detail, exploring how they work, the interaction with the elements of the forklift and how they contribute to an immersive and realistic user experience in the handling of this machinery in the context of virtual reality.

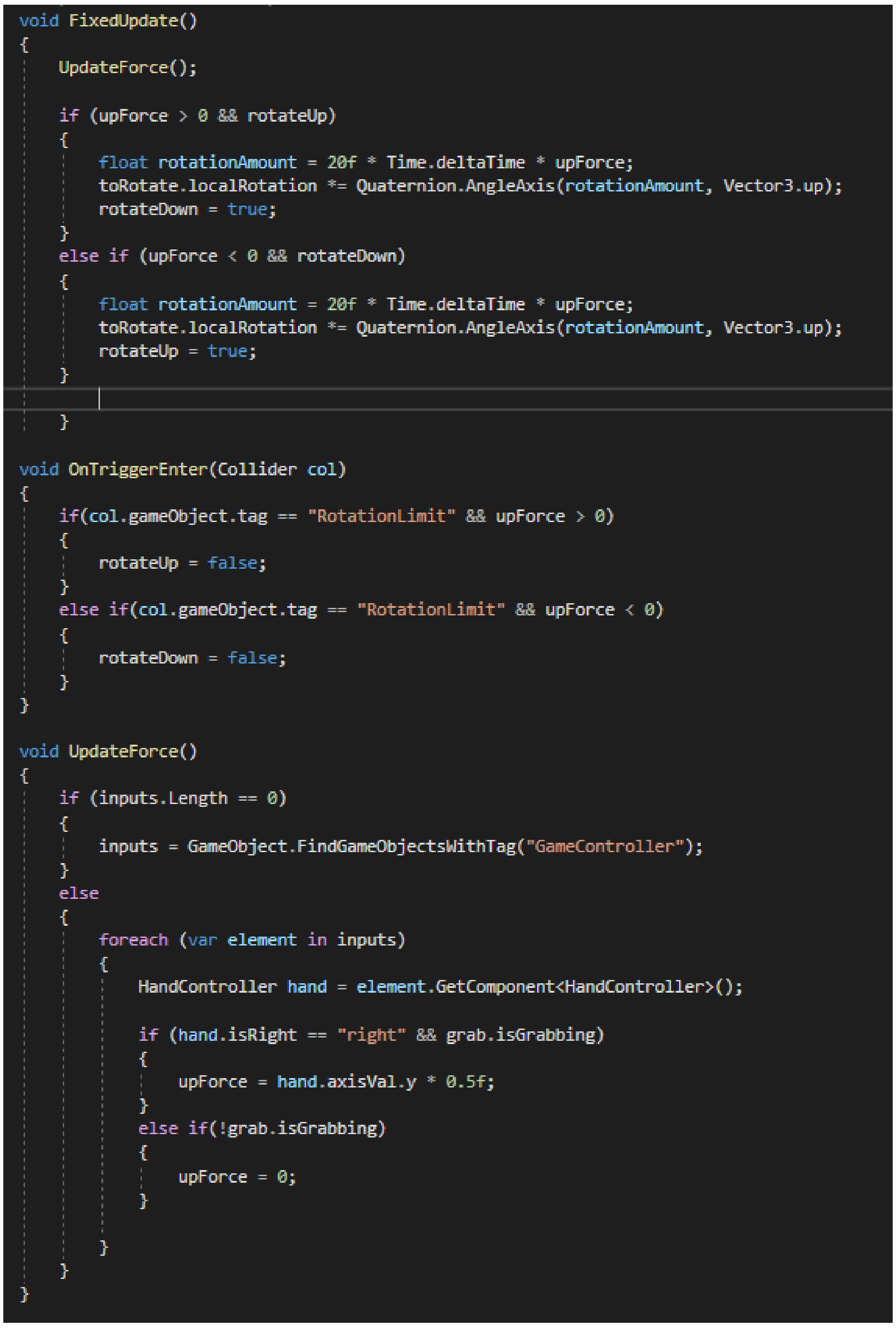
- The `CarController` script provides an interactive driving experience in virtual reality, where the user can directly influence the acceleration of the vehicle through the trigger of the controls, which is reflected in the `accelVal` variable. In addition, the script adjusts the direction of the vehicle by modifying the angle of the wheels based on user input, which is handled with the `currentSteerAngle` variable. The brakes are activated by a button, which impacts the `isBreaking` variable, and the script manages collision physics to simulate a bounce when needed. In short, this code translates user interactions into mechanical vehicle behaviors within the virtual environment.



```
// Update is called once per frame
void FixedUpdate()
{
    GetValues();

    if (!isBouncing)
    {
        foreach (WheelCollider wheel in wheels)
        {
            wheel.motorTorque = strength * Time.deltaTime * accelVal;

            wheel.wheelDampingRate = dampening;

            wheel.brakeTorque = isBreaking ? brakeStrength : 0;
        }

        foreach (var wheel in wheelsSteer)
        {
            WheelCollider collider = wheel.GetComponent<WheelCollider>();
            Transform trans = wheel.transform;

            collider.steerAngle = currentSteerAngle;
            collider.wheelDampingRate = dampening;
            //UpdateWheelVisual(collider, trans);
        }
    }


}
```

Figure 10: FixedUpdate function

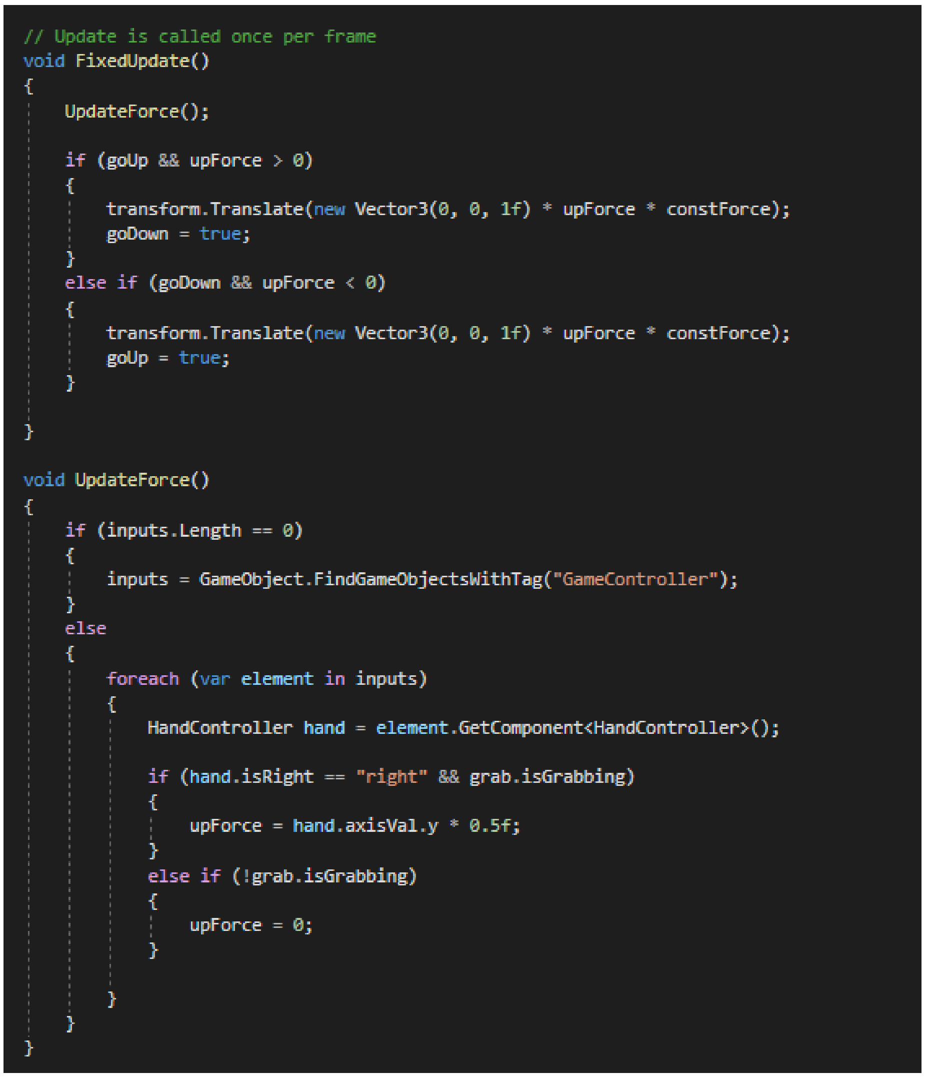
- The `RotateElement` script allows the user to control the rotation of the elevator in the virtual reality environment by using manual controls. Through interaction with the controller's vertical input axis, the `upForce variable` adjusts and determines the intensity and direction of the toRotate object's rotation. With built-in constraints that stop rotation when predefined limits are reached, this script ensures smooth and controlled operation of the elevator's rotational mechanism.



```
void FixedUpdate()
{
    UpdateForce();

    if (upForce > 0 && rotateUp)
    {
        float rotationAmount = 20f * Time.deltaTime * upForce;
        toRotate.localRotation *= Quaternion.AngleAxis(rotationAmount, Vector3.up);
        rotateDown = true;
    }
    else if (upForce < 0 && rotateDown)
    {
        float rotationAmount = 20f * Time.deltaTime * upForce;
        toRotate.localRotation *= Quaternion.AngleAxis(rotationAmount, Vector3.up);
        rotateUp = true;
    }

    }

void OnTriggerEnter(Collider col)
{
    if(col.gameObject.tag == "RotationLimit" && upForce > 0)
    {
        rotateUp = false;
    }
    else if(col.gameObject.tag == "RotationLimit" && upForce < 0)
    {
        rotateDown = false;
    }
}

void UpdateForce()
{
    if (inputs.Length == 0)
    {
        inputs = GameObject.FindGameObjectsWithTag("GameController");
    }
    else
    {
        foreach (var element in inputs)
        {
            HandController hand = element.GetComponent<HandController>();

            if (hand.isRight == "right" && grab.isGrabbing)
            {
                upForce = hand.axisVal.y * 0.5f;
            }
            else if(!grab.isGrabbing)
            {
                upForce = 0;
            }

        }
    }
}
```

Figure 11: Clase RotateElement

- The MoveUp and MoveRetractil scripts work together to manage the vertical movement of an object in a virtual reality environment. While `MoveUp` takes care of the direct vertical movement of the object, `MoveRetractil` controls the vertical movement of a related component, such as a retractable mast. Both use `upForce` vertical force input, which is influenced by the user's interaction with the controller, to determine the direction and intensity of movement. The coordination between these two scripts ensures smooth and restricted vertical movement, where `MoveRetractil` also considers collisions and spatial constraints to prevent unwanted movements, such as reaching the upper or lower stop of the allowed travel.



```
// Update is called once per frame
void FixedUpdate()
{
    UpdateForce();

    if (goUp && upForce > 0)
    {
        transform.Translate(new Vector3(0, 0, 1f) * upForce * constForce);
        goDown = true;
    }
    else if (goDown && upForce < 0)
    {
        transform.Translate(new Vector3(0, 0, 1f) * upForce * constForce);
        goUp = true;
    }

}

void UpdateForce()
{
    if (inputs.Length == 0)
    {
        inputs = GameObject.FindGameObjectsWithTag("GameController");
    }
    else
    {
        foreach (var element in inputs)
        {
            HandController hand = element.GetComponent<HandController>();

            if (hand.isRight == "right" && grab.isGrabbing)
            {
                upForce = hand.axisVal.y * 0.5f;
            }
            else if (!grab.isGrabbing)
            {
                upForce = 0;
            }

        }
    }
}
```

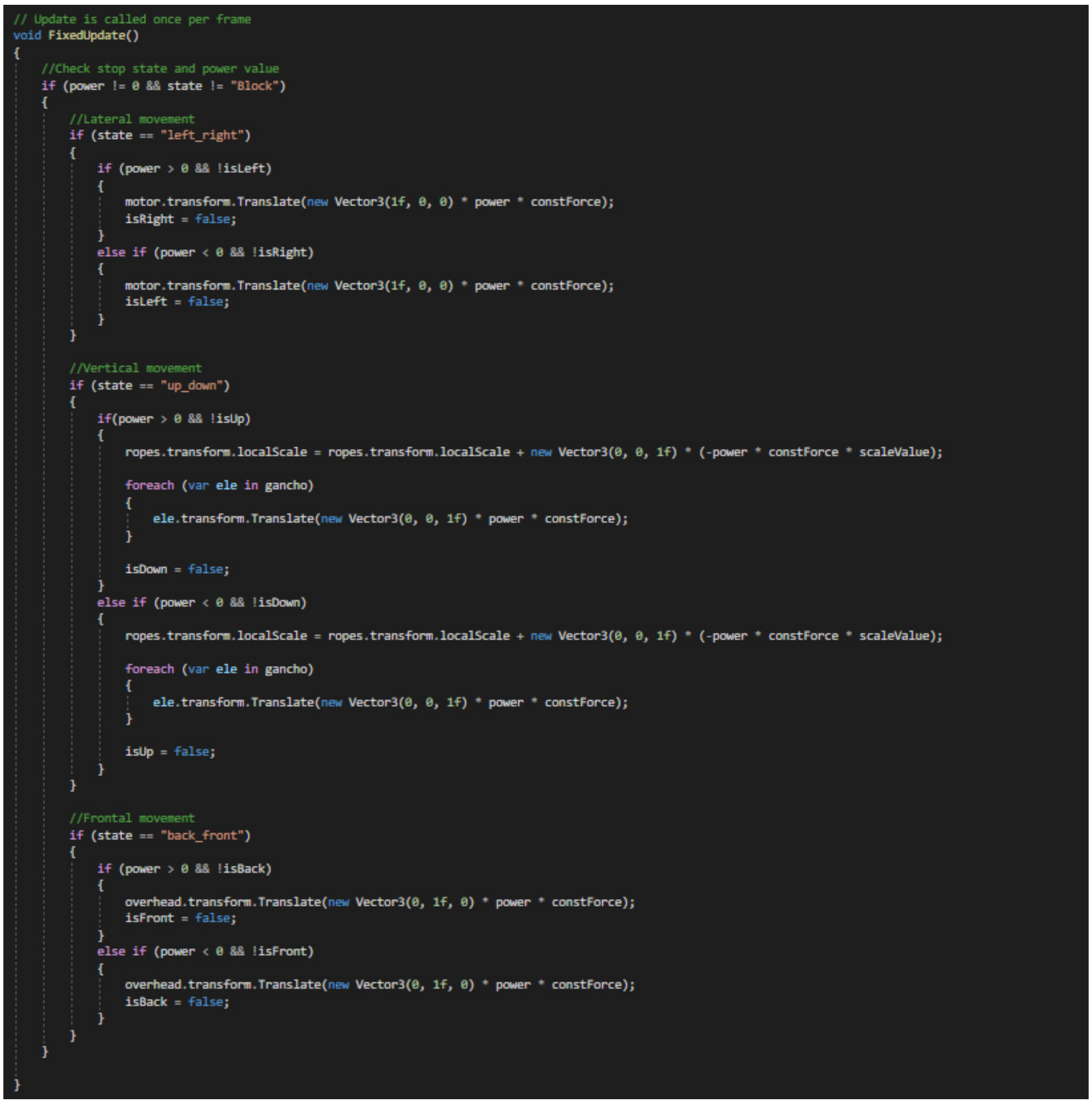
Figure 12: Clase MoveUp

4.4.4. Overhead Crane Control

The control of the overhead crane in the virtual reality environment is an integral part of the interactive experience, allowing users to manipulate objects within a three-dimensional space with precision and realism. For bridge and hook movement, two essential scripts are used:

`CraneController` and `HookController`. The former manages the operations of the overhead crane, such as translation in different directions, while the latter deals with the specific movements of the hook, facilitating the descent, ascent and precise positioning of the hook. Additionally, for the grasping and handling of boards and other objects, the `AttachTarget` script is used, which allows effective coupling and uncoupling, ensuring that the objects remain fixed during transport and are released smoothly at the desired destination. Together, these scripts create a cohesive and efficient control system for overhead crane operation within the simulation.

- The `CraneController` script is the core of the overhead crane control system, allowing lateral, vertical, and frontal movement of the overhead crane engine, cab, and ropes. It uses the `power` variable to determine the intensity and direction of the movement, while `state` indicates the type of movement being executed. Boolean variables such as `isBack`, `isFront`, `isLeft`, `isRight`, `isDown`, and `isUp` act as limit switches to prevent the crane from moving beyond its physical limits. On the other hand, the `HookController` works in tandem with the `CraneController`, specifically monitoring collisions with specific tags to manage and restrict the vertical movement of the hook, preventing it from moving beyond the upper and lower limits. Both scripts ensure smooth and safe operation of the overhead crane, allowing the user to handle loads accurately within the virtual environment.
- The `AttachTarget script` takes care of the mechanics of gripping and holding objects, such as boards, in a virtual reality environment. It uses a `HingeJoint` to simulate a flexible connection point between the object and a tie-down point, allowing for controlled pendulum movement. Detecting the object's proximity to a target and user input are crucial to activating the grip. When the user presses the grip button and the object is in the correct position, the script activates the tether texture and configures the properties of the `HingeJoint`, setting limits and speed to simulate realistic movement. If the user releases the grab button or the object moves too far away, the script disables the connection, allowing the object to be released in a controlled manner. In addition, the script interacts with an instruction panel to guide the user through the process of handling the objects.



```
// Update is called once per frame
void FixedUpdate()
{
    //Check stop state and power value
    if (power != 0 && state != "Block")
    {
        //Lateral movement
        if (state == "left_right")
        {
            if (power > 0 && !isLeft)
            {
                motor.transform.Translate(new Vector3(1f, 0, 0) * power * constForce);
                isRight = false;
            }
            else if (power < 0 && !isRight)
            {
                motor.transform.Translate(new Vector3(1f, 0, 0) * power * constForce);
                isLeft = false;
            }
        }

        //Vertical movement
        if (state == "up_down")
        {
            if(power > 0 && !isUp)
            {
                ropes.transform.localScale = ropes.transform.localScale + new Vector3(0, 0, 1f) * (-power * constForce * scaleValue);

                foreach (var ele in gancho)
                {
                    ele.transform.Translate(new Vector3(0, 0, 1f) * power * constForce);
                }

                isDown = false;
            }
            else if (power < 0 && !isDown)
            {
                ropes.transform.localScale = ropes.transform.localScale + new Vector3(0, 0, 1f) * (-power * constForce * scaleValue);

                foreach (var ele in gancho)
                {
                    ele.transform.Translate(new Vector3(0, 0, 1f) * power * constForce);
                }

                isUp = false;
            }
        }

        //Frontal movement
        if (state == "back_front")
        {
            if (power > 0 && !isBack)
            {
                overhead.transform.Translate(new Vector3(0, 1f, 0) * power * constForce);
                isFront = false;
            }
            else if (power < 0 && !isFront)
            {
                overhead.transform.Translate(new Vector3(0, 1f, 0) * power * constForce);
                isBack = false;
            }
        }
    }

}
```

Figure 13: FixedUpdate function of the CraneController class

4.4.5. Questionnaire Manager

In order to optimize and simplify the process of creating questionnaires, a class called `QuizManager` was designed that standardizes the construction of question panels. The operation of this class is broken down into the following steps:

1. Variables are defined to store all questions, the current question, the correct answer, and user interface (UI) elements.
2. The InitQuiz `function is implemented`, the purpose of which is to initialize all the necessary variables and then invoke `SetNextQuestion` to start the quiz.

3. The `SetNextQuestion` and `SetAnswerOptions functions` are responsible for updating the variables and components of the panel, such as texts and buttons, assigning the data of the new question and discarding the previous one if necessary.
4. The `SelectButton(Button but)` function is activated when an answer button is pressed. For single-answer questions, this function will lead to the execution of `CheckResponse`.
5. The `CheckResponse` function contains the logic needed to verify whether the selected response is correct. If yes, the user will advance to the next panel; otherwise, you'll have a chance to try answering again.

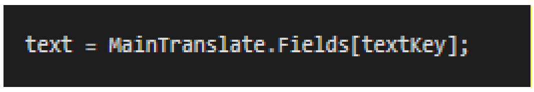
4.4.6. Translator

This process acts on each of the scenes in the background to provide each element of the tool's user interface with texts translated into the selected language.

The first step was to translate the texts into the languages of the participating partners. These translations are structured in a format in which each word or phrase is associated with a unique key, making it easy to retrieve the text in the desired language. To manage this system, two specific classes have been developed:

1. `MainTranslate`: This class has the function of linking translation scripts to the operating environment, i.e., the code. Whenever a specific key generated in the mentioned script needs to be localized, `MainTranslate` will provide the corresponding word or phrase in the correct language.
2. `SceneTranslate`: The responsibility of this class is to communicate to `MainTranslate`, at the beginning of each scene, which language has been selected in the options menu.

With these processes active, the only requirement for the user is to replace each of the lines of text they want to insert with the following line of code:



```
text = MainTranslate.Fields[textKey];
```

Figure 14: Line to translate.

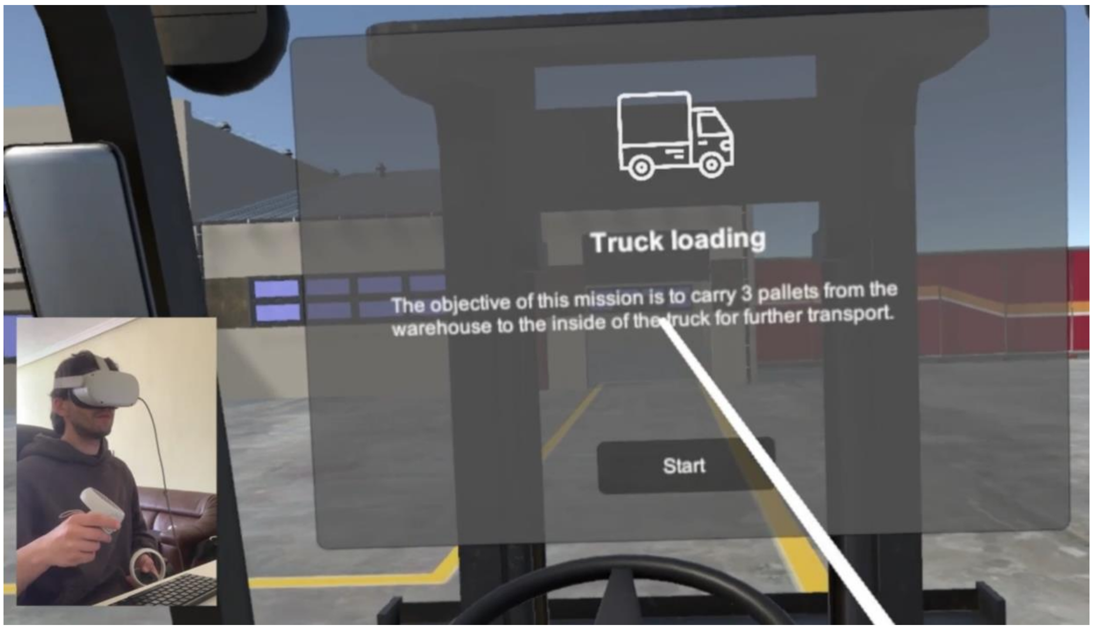
4.4.7. Interaction with lightning

This functionality enables the user to interact with objects or panels that are located at a certain distance. It is achieved by creating a virtual beam that departs from the user's hand, allowing them to perform specific actions on certain objects. In the context of this project, it is crucial that the user is able to answer the questions posed using this technique. To implement this type of interaction, the following steps must be followed:

1. Generate a Ray Interactor object for each hand, accessible via the GameObject/XR menu.

2. Choose the objects to interact with by adjusting the Raycast Mask parameter of the XR Ray Interactive component. This setting allows you to select the Layer on which the beam will act, therefore the objects intended to interact with the beam must be assigned to the corresponding Layer.
3. In this scenario, we have chosen to select the UI layer for the panels provided by Unity.
4. To prevent the lightning from being constantly visible, it is necessary to modify the Invalid Color Gradient attribute of the XR Interactor Line Visual component to a transparent value. This way, the beam will only become visible when the hand is pointing at a panel.
5. Both objects will be incorporated as children of the Camera Offset object, which was previously created in the scene preparation phase, ensuring that they are part of the object linked to the VR device.

With the implementation of these steps, the functionality required to generate and interact with the various dashboards described above is achieved.


Truck loading

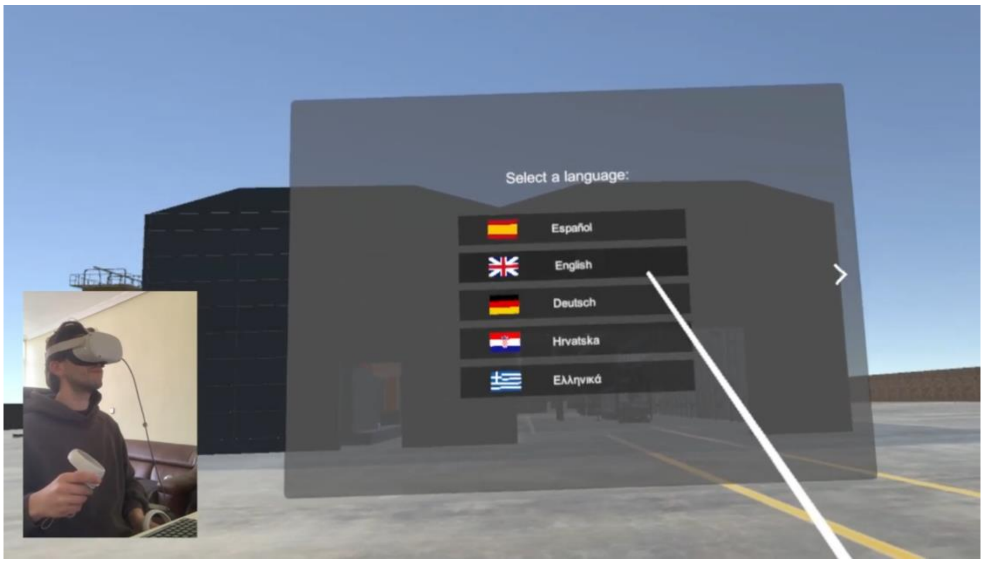
The objective of this mission is to carry 3 pallets from the warehouse to the inside of the truck for further transport.

Figure 15: Interaction with Lightning

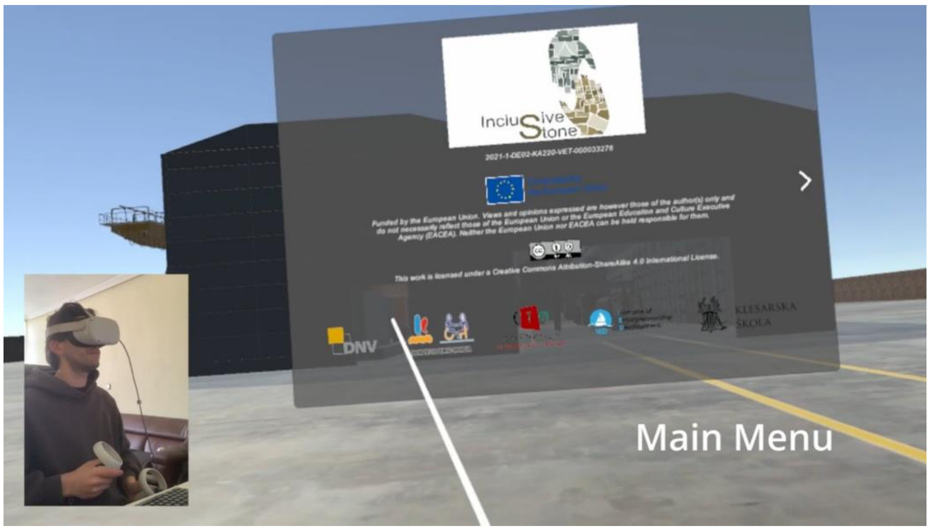
5. YOUTUBE VIDEOS

To complement the white paper and offer a more dynamic perspective on our VR tool, we have selected a series of YouTube videos that demonstrate its functionality. Below, we present these audiovisual materials that provide a clear and direct sample of the performance and possibilities that our VR solution offers.

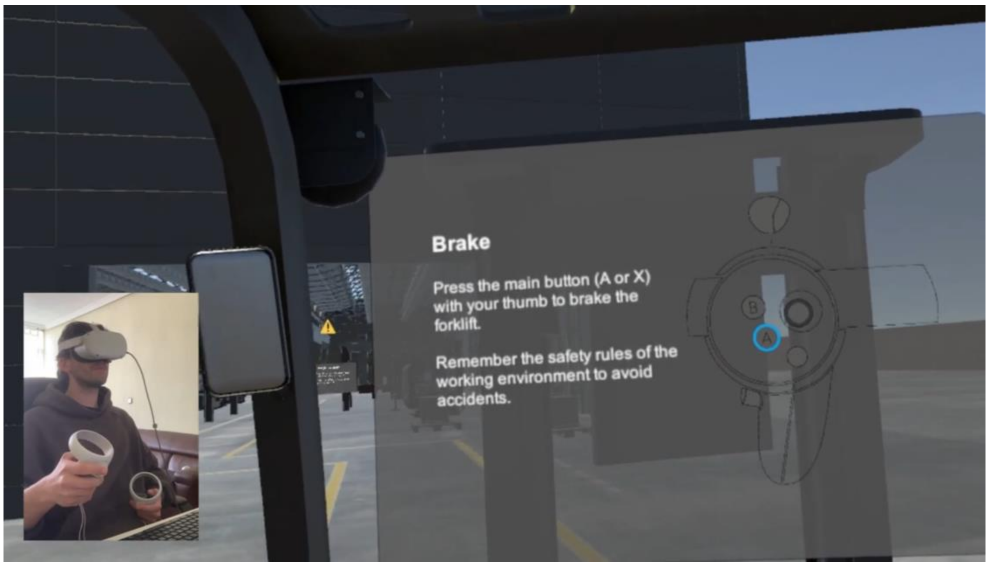
- Promotion: https://www.youtube.com/watch?v=Ogs2WzzCRe0&ab_channel=AEIPiedraNatural



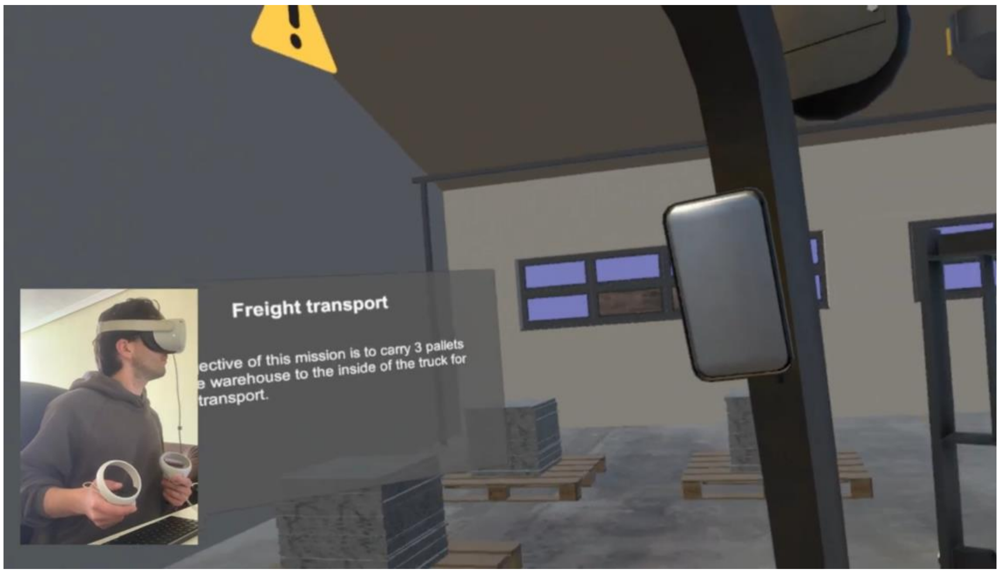
- Main Menu: https://www.youtube.com/watch?v=I7j9rTMeWmo&ab_channel=AEIPiedraNatural



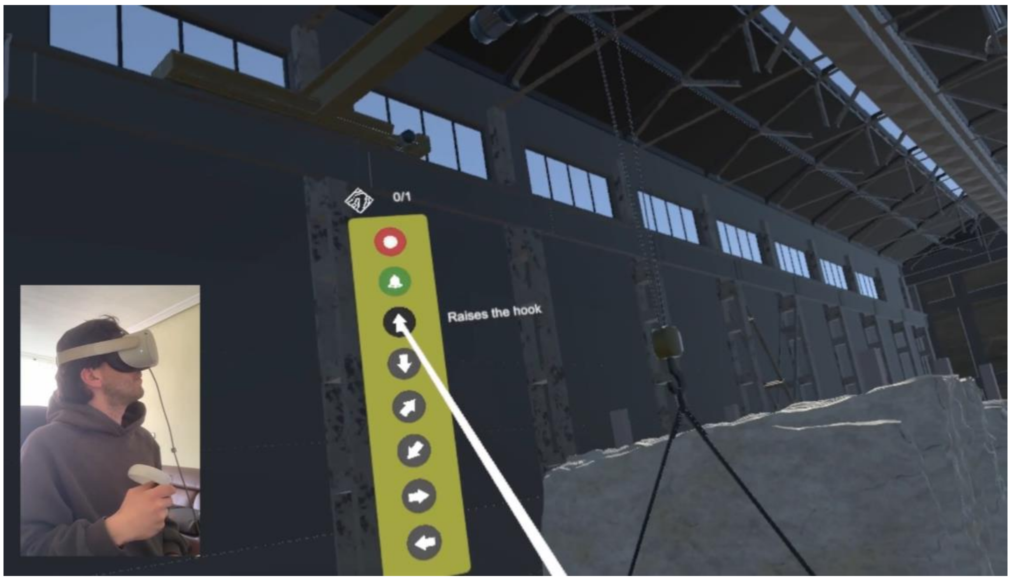
- Forklift – Storage: https://www.youtube.com/watch?v=hCkCl9ihiLU&t=14s&ab_channel=AEIPiedraNatural


Brake
Press the main button (A or X) with your thumb to brake the forklift.
Remember the safety rules of the working environment to avoid accidents.

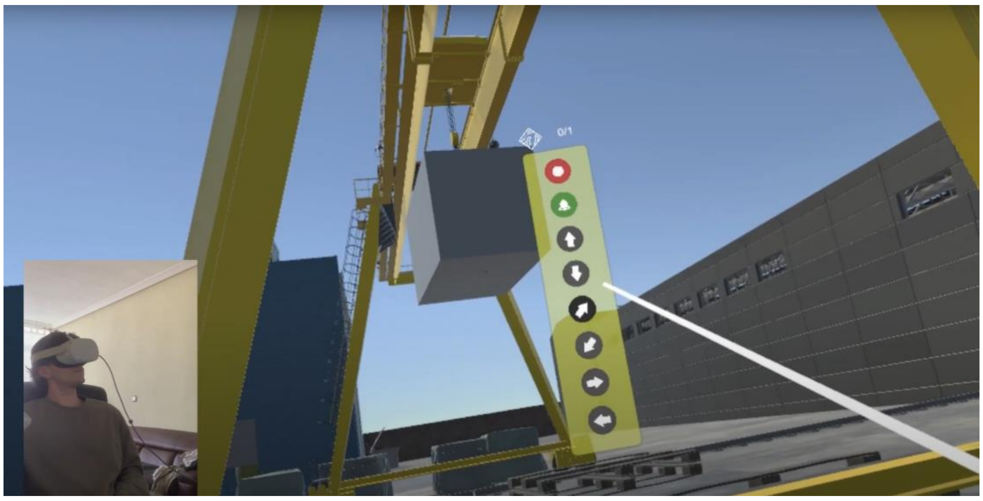
- Forklift – Truck loading: https://www.youtube.com/watch?v=NnpA44V6DMY&t=13s&ab_channel=AEIPiedraNatural


Freight transport
ective of this mission is to carry 3 pallets
e warehouse to the inside of the truck for
transport.

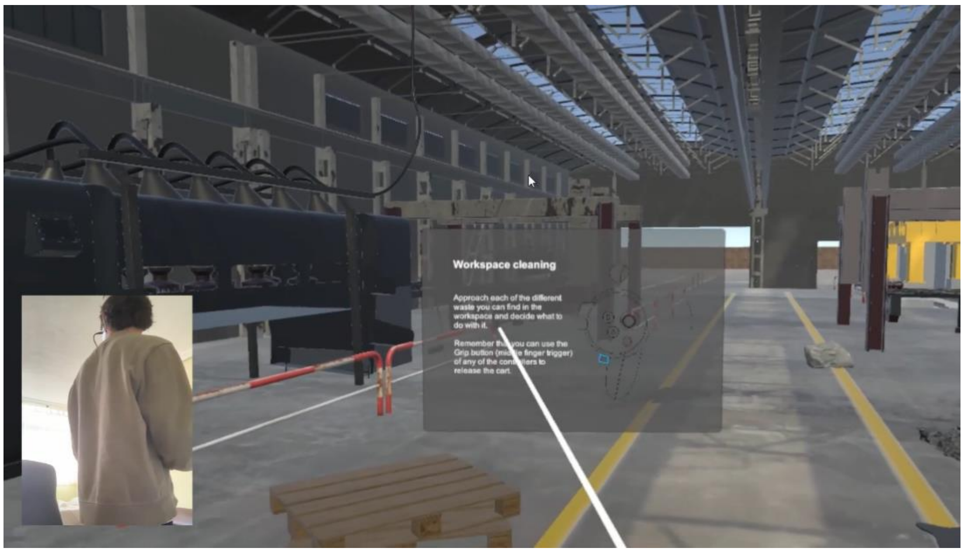
- Overhead Crane – Slabs: https://www.youtube.com/watch?v=LK9pSCPLMrw&ab_channel=AEIPiedraNatural



- Overhead Crane – Blocks: https://www.youtube.com/watch?v=tVyFFRJyQ4U&ab_channel=AEIPiedraNatural



- cleaning: hattops://www.youtube.com/watch?v=twiffarjyak4u&ab_channel=apedarentural



Workspace cleaning

Approach each of the different waste you can find in the workspace and decide what to do with it.

Remember that you can use the Grip button (middle finger trigger) of any of the controllers to release the cart.

- Waste management: https://www.youtube.com/watch?v=mojMZ2G6Huc&ab_channel=AEIPiedraNatural