Task O2/A6

Implementation of improvements-Technical improvement of Virtual Reality (VR) tool

INDEX

1. INTRODUCTION

The SafeCRobot consortium has produced an immersive and interactive virtual reality-based training tool that consists of 10 risk scenarios (each scenario consists of one type of construction robot/autonomous vehicle and risks associated with them) and preventive measures needed for their mitigation. The main aim of the immersive virtual reality-based tool developed for this project will be to use the tool to train construction workers to enable them to work safer alongside construction robots and autonomous equipment on construction sites.

The immersive training tool was improved, so the beta version of the tool was modified with the improvements identified in the evaluation by the collaborators, as well as the testing in the courses of the participants. In addition, the technical conclusions of the International Seminars were also considered.

1.1. Methodology and operation of the application.

The research and development of SafeCRobot have been divided into several distinct parts corresponding to the development of the planned work packages.

The following sections describe the evaluation methodology on which the project has been based, as well as its operation and the improvements implemented.

2. DEVELOPMENT OF THE TOOL.

The SafeCRobot (hereafter called as “tool”) is based on immersive virtual reality. Currently, the most common way of developing applications with this technology is by means of game development engines such as Unity3D.

The development of the tool can be divided into three main tasks that will be covered extensively in the following sections:

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

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

- **Autodesk Revit:** Autodesk Revit is a building information modelling software tool for architects, structural engineers, mechanical, electrical, and plumbing engineers, designers and contractors.

- **Blender:** Multiplatform software dedicated to processing, modelling, rendering and other tasks related to 3D graphics. It is open source and has a large community with videos and documentation, which makes it very easy to learn.
- **Unity3D:** This is another open-source software, a game development engine which aided in the development of an immersive virtual environment. It is highly scalable, allowing users to add modules by means of code scripts, and it also has an interesting community and documentation.
- **Oculus Quest:** This is a very affordable and accessible Virtual Reality device. Developed by Meta as one of the pillars of its new philosophy. They do not require cables for use and have easy integration with Unity.

3. SCENE DESIGN.

This is the process by which each of the steps to be followed from the point of entry into the tool until the immersion is completed are devised. In other words, generating the development script for the application.

The first step was to identify various types of hazards associated with the application of robots/autonomous vehicles in construction sites through a rigorous literature review. Once the hazards are identified and categorised (hazard to self and hazard to others); 10 risk scenarios based on the identified hazards were modelled using the above-mentioned tools. The selected scenarios are the following:

3.1. Scene selection

3.1.1. Drones (Unmanned Aerial Vehicles) – Preparing for flights on construction sites in daylight

During this scenario, the user will take on the role of a pilot of an unmanned aerial vehicle (UAV). The user's task will be to properly prepare for the flight during the day at the construction site. The user is located in the construction office. The container contains the following equipment: a drone, a set of charged batteries, a tablet, and construction documentation. On the wall user will find the development of the construction site. Considering some of the most important safety issues for flying drones in construction sites users are then required to answer questions in the quiz within the scene.

3.1.2. Drones (Unmanned Aerial Vehicles) – Flying on construction sites in favourable weather conditions

During this scenario, the user will take on the role of a pilot of an unmanned aerial vehicle (UAV). The task will be to carry out a mission (construction site raid) using a drone on a bright and sunny day. During the raid, they will have to pay attention to their surroundings and the messages appearing on the controller screen regarding the technical parameters of the flight. Considering some of the most important safety issues for flying drones in construction sites users are then required to answer questions in the quiz within the scene.

3.1.3. Drones (Unmanned Aerial Vehicles) – Flying on construction sites in adverse weather conditions

During this scenario, users will take on the role of a pilot of an unmanned aerial vehicle (BSP). The task will be to carry out a mission (construction site raid) using a drone during adverse weather conditions, including wind, and rain. During the raid, users will have to pay attention to the environment and the messages appearing on the controller's screen regarding the technical parameters of the flight. Considering some of the most important safety issues for flying drones in construction sites users are then required to answer questions in the quiz within the scene.

3.1.4. Drones (Unmanned Aerial Vehicle) – Preparing for flights on construction sites at night

During this scenario, users will take on the role of a pilot of an unmanned aerial vehicle (UAV). The task will be to properly prepare to fly at night on a construction site. The user is located in the construction office. In the container, the user will find the following equipment: a drone, a set of charged batteries, a tablet, construction documentation, and additional lighting. Look around the office and get ready to fly. Considering some of the most important safety issues for flying drones in construction sites users are then required to answer questions in the quiz within the scene.

3.1.5. Autonomous Site Transport Vehicle – Indoor site conditions

This module introduces users to health and safety requirements for operating **Autonomous Transport Vehicles (ATV)** in construction indoor site conditions. ATV is a vehicle capable of operating on its own. They are usually wheeled or tracked and vary in size. For indoor transport, ATVs are normally small-medium sized. Assume the ATV used in this module is electric and has the following dimensions: L-1200mm, H-600mm and W-700mm and can carry loads up to 500kg.

Users are required to P observe the environment and the equipment to identify any issues they consider health and safety risks. Considering some of the most important safety issues for managing an ATV on a construction site users are then required to answer questions in the quiz within the scene.

3.1.6. Autonomous Site Transport Vehicle – External and outdoor site conditions

This module introduces the user to health and safety requirements for operating **Autonomous Transport Vehicles (ATV)** in construction external or outdoor site conditions. For external or outdoor site activities ATVs vary from small to very large equipment. The ATV used in this scene is a large dumper used for transporting material on site. Typical materials include aggregate, excavated or demolished material. Please observe the environment and the equipment to identify any issues you consider as health and safety risks. Considering some of the most important safety issues for managing an ATV on a construction site users are then required to answer questions in the quiz within the scene.

3.1.7. Remote Controlled Equipment (Demolition Robots) – General Requirements

This module introduces users to health and safety requirements for operating a remote-controlled demolition robot in construction site conditions (indoor and outdoor). Contractor Smith mandates two of his employees, Marc and Gordon, to demolish several walls inside/outside a large industrial factory. A remote-controlled demolition robot is to be used. The hydraulic hammer has already been mounted. Marc and Gordon have never worked with a demolition robot before but are looking forward to it. Please join Marc and Gordon on their job and find out the health and safety risks when working with the demolition robot. Then the users will check and consolidate their knowledge by working through the quiz**.**

3.1.8. Remote Controlled Equipment (Demolition Robots) – Indoor site conditions

This module introduces users to health and safety requirements for operating a remote-controlled demolition robot in construction site conditions (indoor and outdoor). Contractor Smith mandates two of his employees, Marc and Gordon, to demolish several walls inside/outside a large industrial factory. A remote-controlled demolition robot is to be used. The hydraulic hammer has already been mounted. Marc and Gordon have never worked with a demolition robot before but are looking forward to it. Please join Marc and Gordon on their job and find out the health and safety risks when working with the demolition robot. Then the users will check and consolidate their knowledge by working through the quiz**.**

3.1.9. Remote Controlled Equipment (Demolition Robots) – External and outdoor site conditions

This module introduces users to health and safety requirements for operating a remote-controlled demolition robot in construction site conditions (indoor and outdoor). Contractor Smith mandates two of his employees, Marc and Gordon, to demolish several walls inside/outside a large industrial factory. A remote-controlled demolition robot is to be used. The hydraulic hammer has already been mounted. Marc and Gordon have never worked with a demolition robot before but are looking forward to it. Please join Marc and Gordon on their job and find out the health and safety risks when working with the demolition robot. Then the users will check and consolidate their knowledge by working through the quiz**.**

3.1.10. Remote Controlled Equipment (Diggers/excavators) - External and outdoor site conditions

This module introduces users to health and safety requirements for operating remote-controlled equipment using Diggers/Excavators as an example. The scenario depicts construction in external or outdoor site conditions. An excavator or digger is a piece of equipment consisting of a boom, dipper, bucket and a cab on a rotating platform. They are primarily used for digging and excavation of material. They are usually wheeled or tracked and vary in size. The excavator used in this scene is medium-sized. Users are required to observe the environment and the equipment to identify any issues they consider health and safety risks. Considering some of the most important safety issues for managing a remote-controlled digger/excavator on a construction site user are then required to answer questions in the quiz for the scene.

Based on the above-detailed scene script, 3D modelling of the virtual construction environment was carried out using previously detailed modelling tools.

To simulate each of the steps whilst the users are in the scene, different panels have been generated with information that the user will have to read to act. The panels found in each scene can be classified into the following types.

3.2. General instructions

These are panels that appear at the beginning of each scene. They explain the procedure to follow to reach the missions, as well as general information about certain tools. Each mission will start with a welcome panel and two panels with basic instructions

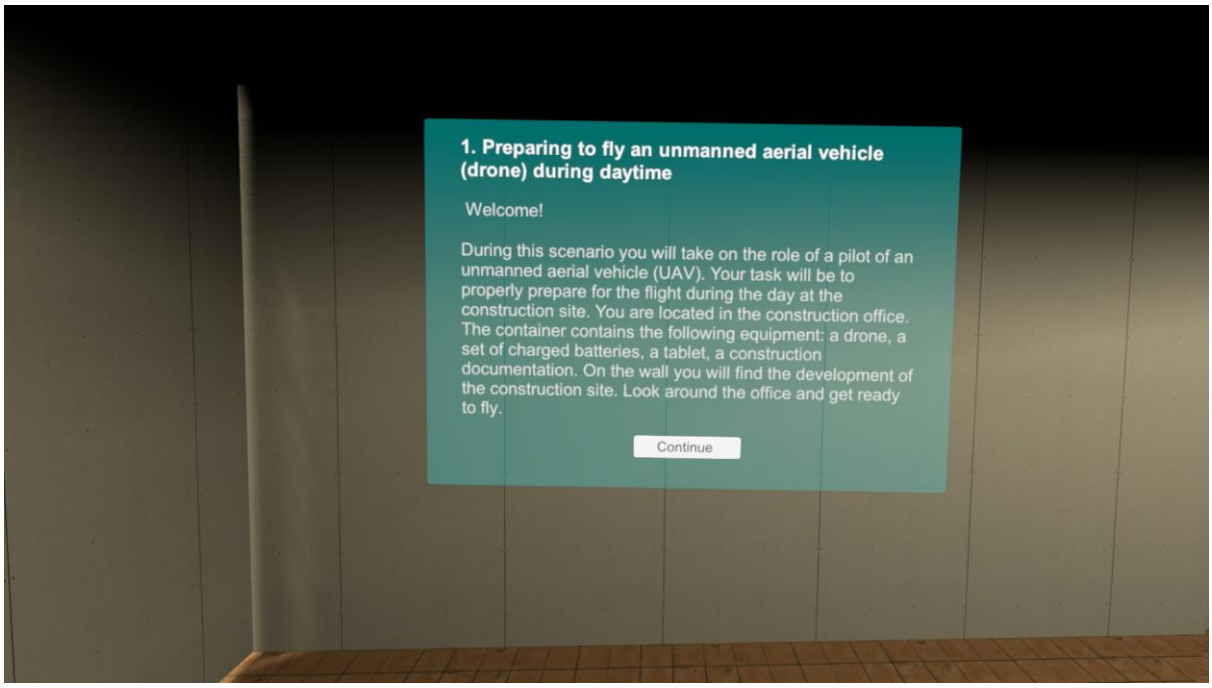


Figure 1. Welcome Screen with Scene Information and Instructions

Quiz Panel: This panel will present a set of hazard-related questions associated with the relevant scene and users can use the laser pointer and trigger button on the controller to answer the questions which will be recorded and results will be displayed at the end of the quiz.

3.3. Fail Panel

Each time a mission is failed, the following panel will appear:

Figure 2. Fail Panel.

3.4. Success Panel

Each time a mission is completed, the following panel will appear:

Figure 3. Success Panel.

3.5. Task (Example)

These panels will vary according to the task to be performed, containing information, questions, dialogues or orders to be completed.

Task 5: Autonomous Site Transport Vehicle – Indoor site conditions (please refer to details in section 3): Continue
Quiz: Which of the following health risks exists on the site you have just reviewed? 1. Collision (V) 2. Musculoskeletal Disorders 3. Trapping (V) 4. Crushing (V) 5. Bruising
Quiz Which of the following statements about the scene is true? 1. The route planning for the vehicle was poor (V) 2. The vehicle warning systems were very good 3. The warning and safety signs are inadequate (V) 4. There's a risk of overturning 5. The operator or banksman is not identifiable (V) 6. Workers maintained an appropriate distance between themselves and the vehicle 7. The vehicle needs to be separated from the workers

8. Poor demarcation of pedestrian crossing points (V)
Quiz Which of the following can contribute to the risk of a machine falling? 1. Vehicle route too close to openings (V) 2. The site is too small for the vehicle 3. Imbalanced load and overloading (V) 4. Lack of use of barricades 5. Noise
(Success Panel) | (Fail Panel)

Table 1. Example Task Quiz

4. Development of the 3D Virtual Environment

Unity 3D was used as the game development engine and the 3D models developed for the scene creation are called game objects. Thus the second major task consists of generating each of the objects or assets that are directly and indirectly defined in the scripts generated in the previous section.

This process is in turn divided into three tasks:

- **Modelling:** A mathematical representation of a three-dimensional object is made using specific software. In this case, Blender and Autodesk Revit have been used. Some of the game objects were procured from the asset store and are ready to use.
- **Rigging:** This is the process by which the different parts of an object are separated to create a controllable system that allows effective and efficient animations. Unity3D animation functionality alone was utilised for this.
- **Texturing:** This is the process by which 3D objects are given colour and detail. Blender and unities inbuilt material editor has been used to achieve this.

The list of objects developed for this project is very wide and diverse, from very simple objects to complex machines with a lot of detail. The objects generated can be classified into the following categories.

- Containers, Buildings, Stores, humanoids and other elements such as (not limited to) dust, rain etc to recreate a realistic construction environment.

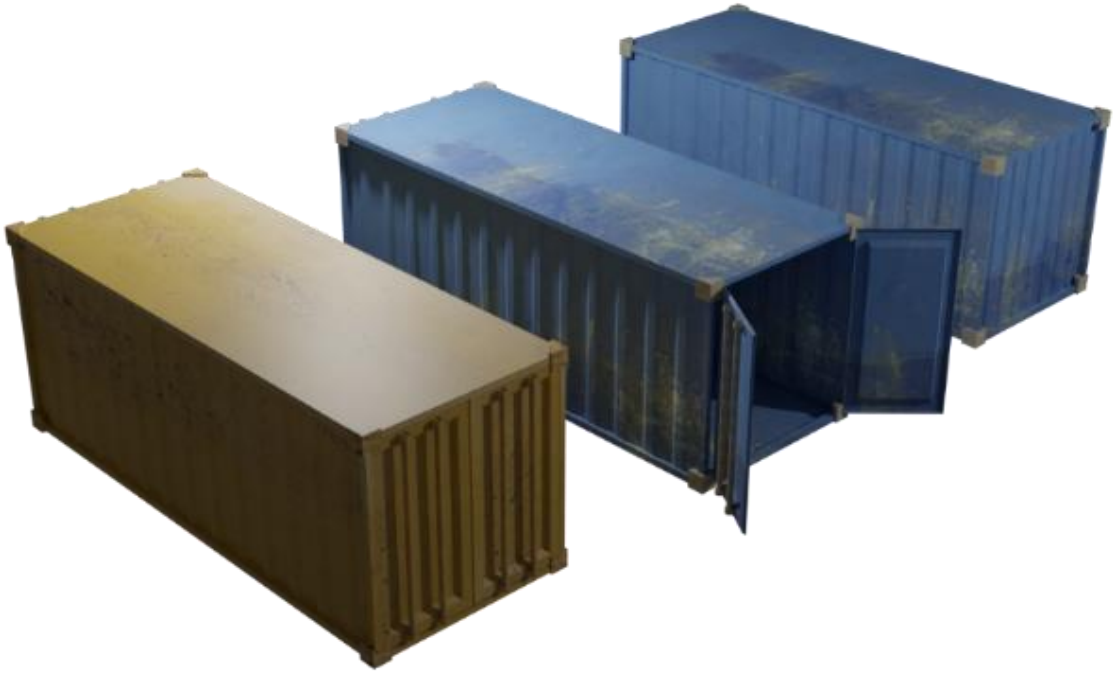


Figure 4. Containers (Example game Objects used for scene development)

Figure 5. Demolisher (Example game Objects used for scene development)

Figure 6. Construction Workers (Example game Objects used for scene development)

Figure 7. Demolisher in action

Figure 8. Active Construction Site (Developed using the game objects)

After the development of each of the objects to be used, the scenarios are prepared so that the development of the functionality is as accurate and as close to the final situation as possible. This process is not usually definitive, as the testing process can lead to modifications. This process has been carried out with the Unity graphics engine, as it is the one that will be used to generate the application.

If we analyse the descriptions of each of the situations described in the previous section, we can see that there are three different scenarios:

- Container (site office): In the container, the user will find the following equipment: a drone, a set of charged batteries, a tablet, construction documentation, and additional lighting.

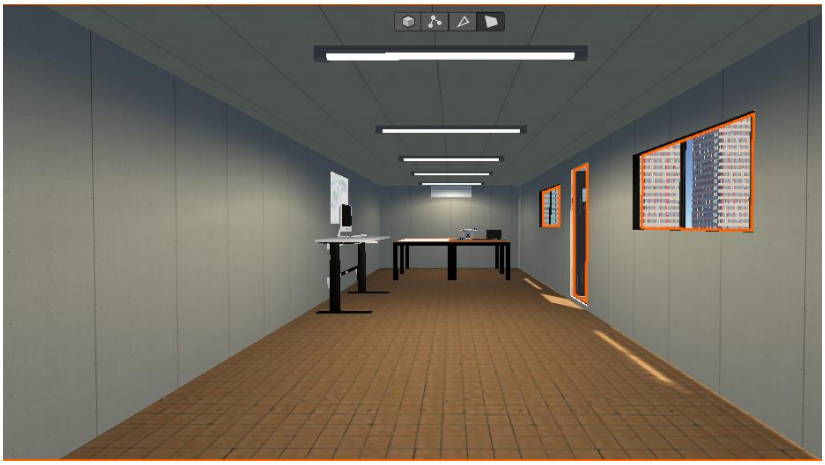


Figure 9. Container office

Figure 10. Drone and Accessories in Container Office

- **Construction site:** This model shows a construction site with, scaffolding, the skeleton of a building and relevant autonomous vehicles.

Figure 11. Active Construction Site

5. FUNCTIONALITY DEVELOPMENT AND IMMERSIVE EXPERIENCE.

Functionality development is the process by which all the collected theory takes shape. In other words, the aim of this task is to make the tool usable. The Unity graphics engine has been used to develop it, as it provides a simple integration of Virtual Reality and has a large community.

Projects in Unity are divided into Scenes, which can be intertwined with events. Scenes are composed of different objects that will contain certain components, which give different characteristics or behaviours to the objects.

The development of functionality is a broad task but can be subdivided into several processes: **Design and Architecture, Data Modelling, SDK Integration and Event Management.**

5.1. Design and architecture

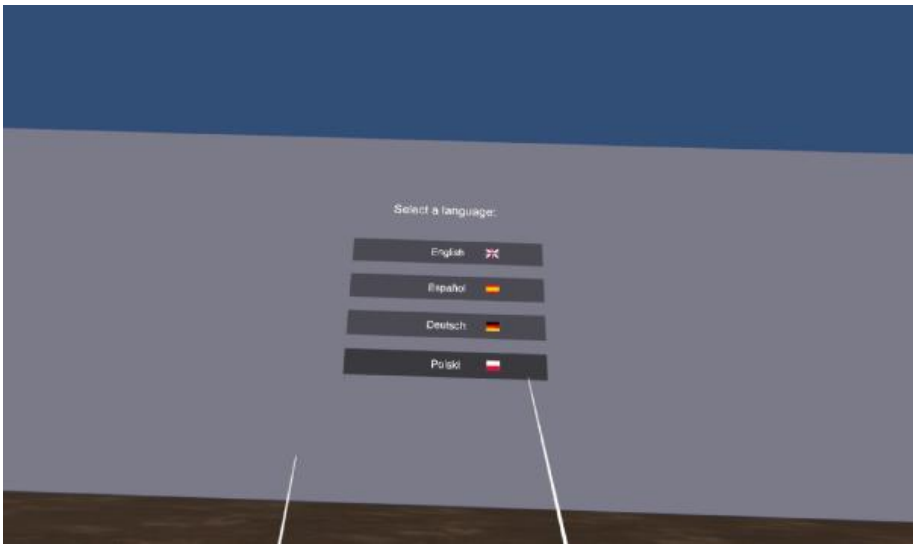
This is the process of shaping the script developed in previous sections, that is, devising the path and decision-making that each scene or screen will entail.

The experience of the tool has two distinct parts. The structure will present is shown in the flowchart in Figure 9.

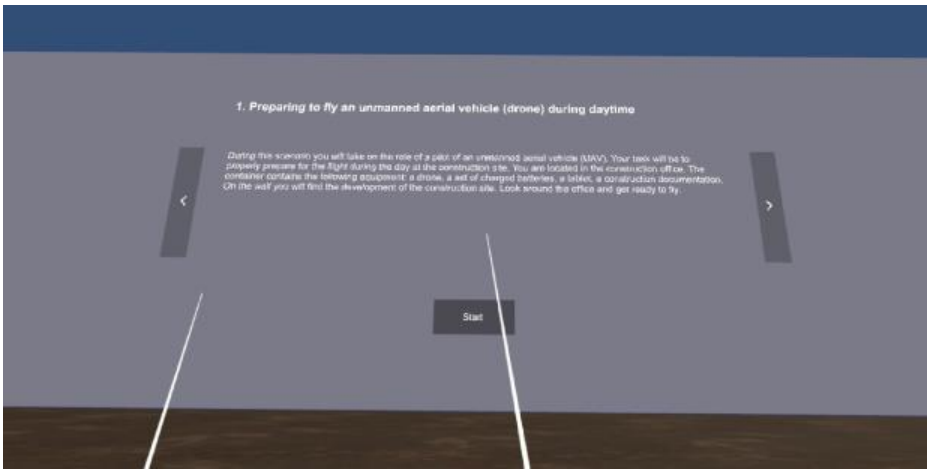
5.1.1. Main Menu

The first screen allows the user to adapt to the virtual environment in which he/she will be involved during the use of the tool. Although the movements are limited in order to focus the user's attention on the real objective of the scene, which is to select one of the situations proposed for the correct use of nano products. It is also necessary to select the language in which the user wants the application to run.

The next screen will depend on the selection made on this screen.



(a)



(b)

Figure 12: Main Menu (a) Language Selection (b) Scenario Selection

Scene: Once the user has picked the specific scenario he/she wants to be tested, the user will be taken into that scene with a floating welcome screen where the details of the scene and the task requirement will be detailed. If the user fails the task, then they can redo it 3 times in a row; further failure will result in redirecting them to get better theoretical knowledge before attempting again. If they pass, they can go to the next scenario and so on.

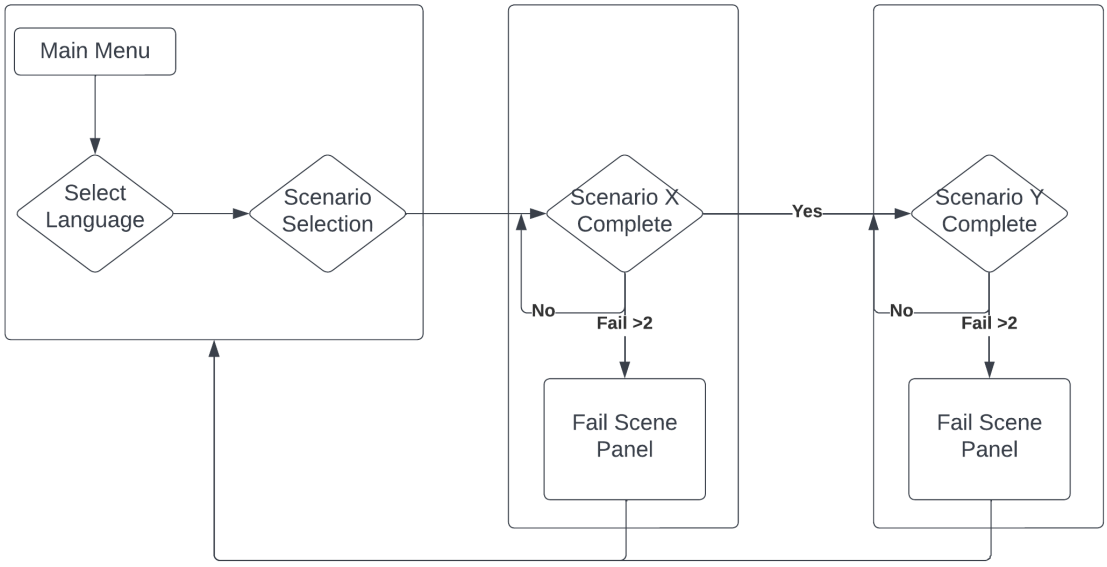


Figure 13: Tool flow diagram.

5.2. Data modelling

In this task, the objective is to generate a general schema on which the information is stored and retrieved. To achieve this, the PlayerPrefs module has been used, which is a class provided by Unity that allows storing data as a key with the name of the project.

These variables in an Oculus application are stored as an XML file inside the directory /data/data/pkg-name/shared_prefs/pkg-name.v2.playerprefs.xml, where pkg-name will be the name of the application.

In addition, Unity provides very easy accessibility to this dataset thanks to the functions provided by the PlayerPrefs class.

The only variable that will be common and that we will have to store, due to the structure of this application, will be the one that refers to the language. To do so, we will use the following functions:

- **SetString (*name*, *value*):** This function will allow us to write a string value on the name variable. In this case, the reference ISO code of the language selected in the menu will be entered.
- **GetString (*name*):** This function returns the value of the variable name. It will be used in each of the scenes to know which language is selected.

5.3. SDK Integration

The development of any type of Virtual Reality application requires three components. A compatible device that can be connected to the computer, a videogame graphics engine and a

Software Development Kit (SDK). The first two requirements were explained at the beginning of the chapter and will be Oculus and Unity, respectively.

An SDK is any set of tools that enable a software developer in the creation process. In the case of Virtual Reality, and in particular the packages adaptable to Unity, there are several, including one specific to Oculus. But the XR Interaction Toolkit has been chosen. A package developed by Unity that allows the project or application to be adapted to any type of device. In other words, even if the tool is developed with the Oculus Quest, it can be installed on any device compatible with Unity. So, you get a wider reach.

To build a project that introduces the SDK and allows interaction in Unity, the following steps must be followed:

1. When opening the engine generate a project with the *Universal Render Pipeline* template. It is important to do it with this template as it provides optimised graphics, and this is a requirement of Virtual Reality if we want to have a correct experience.
2. Install the SDK, i.e. the XR Interaction Toolkit package. This can be done from the Package Manager window, accessed from the Window menu option.
3. We will have to indicate the type of device on which it is going to be built. To do this, activate the VR-supported option in Project Settings/Player.

In this way, the Unity project is ready for Virtual Reality development. However, to start the interaction of the device with the graphical environment, the steps described in the following section must be followed.

5.4. Event management

Event management could be considered the most extensive task within the development of the functionality. It consists of using the components provided by Unity and generating small scripts to achieve the expected user-machine interaction.

This section could be too long, as each scene has a lot of specific details at the interaction level. But here only the common and important points to achieve the goal will be detailed.

5.4.1. Prepare the scene for VR interaction

Once the Unity project has been prepared and the scenarios have been set up or graphically prepared, the next step will be to prepare the scene so that the interaction device can be linked to a camera, and we can use it for testing during development. The process explained in the following steps will be repeated for each of the scenes to be used in the project:

1. The first thing is to remove the default camera.
2. Generate an empty object with an XR Rig component, which will be the container for all the human-machine interaction. We call it VR-Rig.
3. Generate a child object, also empty, which will be the reference position, where the user will start the interaction. This will be called Camera Offset.
4. Generate a camera as a child of Camera Offset, with a Tracked Pose Driver component. This is the object that will act as our eyes and the added component that will allow us to rotate.

5. After this, fill the variables of the XR Rig component (parent object, VR-Rig) with the reference position and the created camera. And place floor as the value of the Tracking Origin Mode attribute, this will make the camera adapt to our height automatically. This way, we get the eyes of the system fully functional.
6. To achieve mobility and visibility of the controllers, two new children are generated inside the Camera Offset object and the XR Controller attribute is added.
7. Add the model we want to use as controllers in the Model Prefab attribute.
8. Finally, to these two objects, we add the XR Direct Interactor component, which will give the controllers interactivity with the rest of the objects in the scene.

5.4.2. Controller input

This is the process by which the values generated by pressing each of the buttons on the knobs are obtained. This will be used for a huge number of tasks, such as picking objects, selecting options, etc.

Before generating anything with these values, it is important to know what type of data each input handles. This can be seen in the *XR Interaction Debugger* window, which you can access from Window/Analysis.

With this information, the next step is to generate a script to obtain these values. The class that has been generated with this is called *HandController* and is used as a component of each of the 3D models that were added in the Model Prefab attribute of the controls. The highlights of the code are:

```
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 14: HandController class

1. The first thing you see in the image above is the declaration of variables. Those that are public will correspond to alterable attributes of the component.
2. When generating a new script, two functions are always automatically created: *Start()*, which is executed once at the start of instantiation, and *Update()*, which is executed iteratively for the life of the component.
3. The first thing that happens inside this component is the *TryInitialize()* function, in this, the VR device is obtained according to its characteristics. For each controller, the characteristic will be whether it is right or left. This is achieved by calling the *GetDevicesWithCharacteristics()* function inside 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 15: TryInitialize() function

4. The *Update()* function is then executed, which calls the *UpdateEvents()* and *UpdateAnimations()* functions.
5. *UpdateEvents()* is responsible for obtaining each of the values generated by the controller through the *TryGetFeatureValue()* function and storing them in the public variables that were generated in the first instance.

6. UpdateAnimations() is a function created to activate the animations that each of the buttons would suppose in case of having a rigged hand model. In any case, it is not an important function in the task of obtaining the values.

```
//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 16: UpdateEvents() function.

5.4.3. Quiz Manager

To make the development easier and faster, a class called *QuizManager* was generated to standardise the process of creating panels. This element consists of the following steps:

1. Variables are generated for all the questions, the current question, the correct answer, and the UI elements.
2. The *InitQuiz()* function is generated, which is responsible for initialising each of the variables that will be needed. It will then call the *SetNextQuestion()* functions, to initialise the quiz.
3. *SetNextQuestion()* and *SetAnswerOptions()*: These functions update the variables and panel elements such as texts and buttons, setting the value of the next queued question, removing the previous one if there was one
4. The function that is activated when an answer button is pressed is *SelectButton(Button but)*. In the case of a question with only one answer, this will call the *CheckResponse()* function.
5. *Checkresponse ():* It is the function with the logic that is in charge of checking if the selected answer is the correct one. If it is, it will pass to the next panel, if it is not, it will

add a failure to the scene and will give rise either to the failure panel or to the scene failure panel.

Figure 17: Quiz Interface

5.4.4. Translator

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

First, a translation of the texts into each of the languages of the participating partners has been carried out. These documents will have a format in which each word or phrase will have a key, which will allow us to access this phrase in any of the languages.

For this purpose, two classes have been created:

- *MainTranslate:* This class oversees creating the link between the translation scripts and the working environment, in this case, the code. Each time a search for this key generated in the previous script is performed, this class will return the word or phrase in the correct language.
- *SceneTranslate:* This class oversees indicating to MainTranslate at the beginning of each new situation, which is the search language selected in the menu.

With these processes in place, all you must do is change each of the lines of text you intended to enter to the next line of code:

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

Figure 18: Translation line

5.4.5. Interaction with rays

This task allows the user to interact with objects or panels at a certain distance. It involves generating a beam that comes out of the hand and allows the user to perform actions on certain objects. For this project, it will be of great importance to answer the questions that have been posed.

To generate this type of interaction, the following steps will be necessary:

1. Create a Ray Interactor object that appears in the GameObject/XR menu for each of the hands.
2. Next, select the objects to be acted upon. To do this, change the Raycast Mask parameter of the XR Ray Interactor component. This attribute will give us the option to choose on which Layer we want the ray to act so that the objects on which we want the ray to act will have to have that Layer.
3. In this case, we have selected the UI layer of the panels provided by Unity.
4. Finally, so that the ray doesn't appear all the time, we will have to change the Invalid Color Gradient attribute of the XR Interactor Line Visual component to a transparent value. So that only when the hand points at a panel will the lightning bolt appear.
5. These two objects will be introduced as children of Camera Offset, an object created in the scene preparation so that it will be included in the object referring to the VR device.

With the implementation of this section, we get the functionality that allows us to generate each of the different panels described and interact with them.

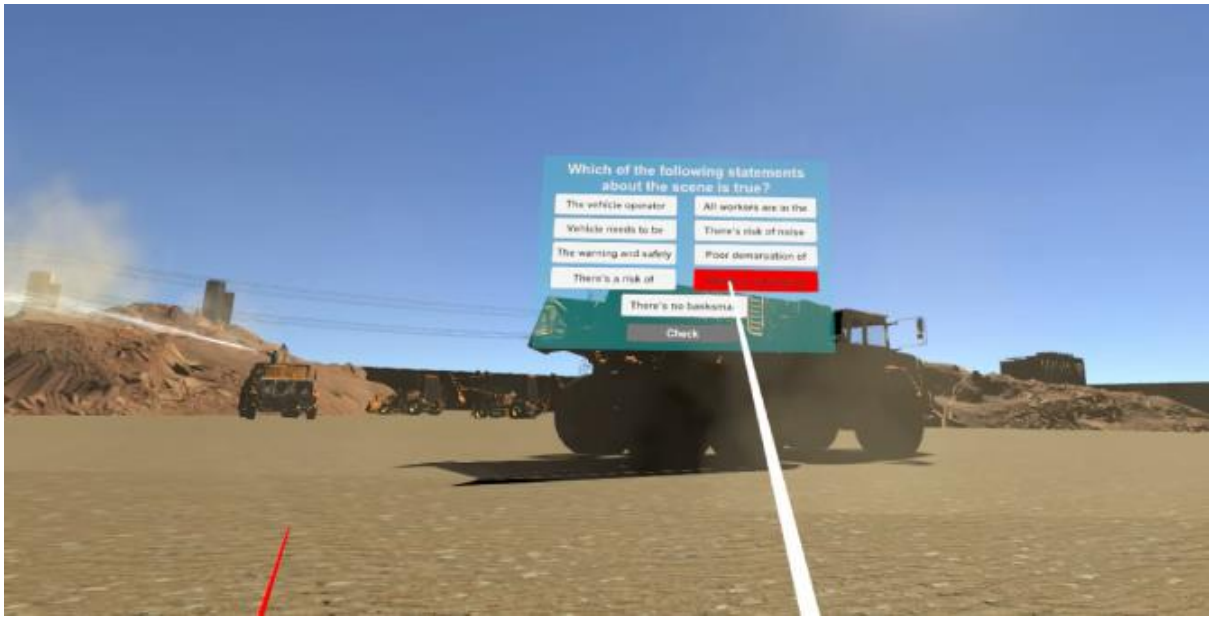


Figure 19: Ray Interaction Image

6. Conclusion

The main aim of this project is the creation of a very innovative immersive and interactive training environment based on Virtual Reality (VR) technology to impart construction workers essential skills and education to interact with machinery and materials which are achived through the development of the SafeCRobot tool presented in this report. This report presents the methodology used for the development of the SafeCrobot tool which is an immersive virtual safety training tool. Awareness of the risks and safety needs of automation is gradually becoming an essential skill in current and future workplaces as a result of the complex interactions imposed by the associated machines and materials in the same work environment. This tool will allow different stakeholders from the construction sector (professional associations, unions, administration) to achieve the desired awareness and upskilling about the impact of robotics and automation on health, safety and the environment in the construction industry across Europe.