TASK O2/A6

IMPLEMENTATION OF IMPROVEMENTS. TECHNICAL IMPROVEMENT AND 3D DEVELOPMENT

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

"The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein".

Erasmus+

INDEX

INTRODUCTION

NanoSafe consortium produced a 3D Training Tool that included 10 3D risk scenarios based on the application and safe use of nanomaterials in the stone sector that show the main risks derived from their use, as well as the preventive measures needed for their mitigation.

The 3D 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 courses of the participants. In addition, the technical conclusions of the International Seminars were also considered.

Methodology and operation of the application.

The research and development of NanoSafe has 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.

1. DEVELOPMENT OF THE TOOL.

This project is based on the immersion that Virtual Reality allows us. Currently, the most common way of developing applications with this technology is by means of videogame graphic engines. That is why the procedure followed in the creation of the tool is based on the development of a video game, in this case for educational purposes.

This 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:

- **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.
- **Unity:** This is another open source software, but in this case it is a graphics engine and is usually used for the development of video games. It is highly scalable, allowing you to add your own 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 an easy integration with Unity.

2. 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.

Firstly, 10 work situations have been chosen in which the use of nanomaterials in the wrong way can pose a risk to health or the state of the environment. The selected situations are the following:

- **Use of sawing machine with water cooling to reduce dust:** In this situation an area of a house is to be paved. For this purpose, a water-cooled saw is to be used to reduce dust. The player must perform the necessary steps to use this tool safely.
- **Application of different products in a stone factory (part I):** In this situation, different products will be applied to the natural stone in the factory. The correct steps must be taken to avoid endangering the product and our health.
- **Pouring of nanomaterial powder into a liquid matrix to create a mixture:** In this situation, the user will have to make a mixture of powder into a liquid matrix to be applied in a construction environment.
- **Nanomaterial applied with a spray-gun on a stone material surface:** This situation aims to prepare an outdoor working area where the nanomaterials will be applied to the stone surfaces with a spray gun. The application will take place in a swimming pool and the environmental conditions will not be optimal.
- **Nanomaterial fixed in a solid matrix which is being drilled:** In this scene, the work consists of preparing a construction environment for drilling into the coated stone surface to lay the electrical wiring for the outdoor lighting fixtures. Focusing on the prevention of risks from the generation of nanomaterials.
- **Application of different products in a stone factory (part II):** Prepare an indoor work area where the nanomaterials will be applied to the stone surfaces with a spray gun and paint roller. This will be done in the natural stone factory.
- **Dust-air mixtures in a factory:** This scene is set in a natural stone factory. Stones are asked to be cut to restore a church. The objective is to test decision-making to prevent nanomaterials from affecting the workers in the plant.
- **Nano-waste management environmental protection:** This scene will contain a series of missions aimed at air and waste management in a natural stone factory. To achieve a safe environment free of nanomaterials that could harm the workers.
- **Nanomaterial applied as an aerosol:** In this scene the pool will have to be finished using anti-slip spray. This will be done in the scene where the

construction of a house is being finished. We will have to perform the correct steps in order not to damage our health.

- **Waste cleaning or disposal after working hours:** In this scene the user will have to make decisions about the waste generated in a natural stone factory.

Health and safety being the main objective of this project, an analysis of each of these situations and the decision making required for the use of the different nano-products selected for this project has been carried out. Thanks to this analysis, we can summarise each situation in the following diagram:

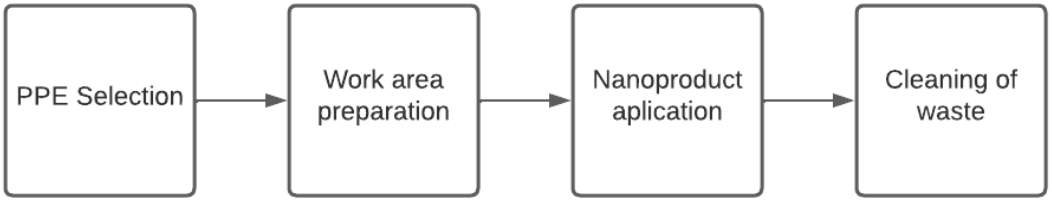

Figure 1: Outline of risk situations.

It is in this context that it was decided to create a scene model by consecutive steps or missions (as we would say in the videogame world), i.e. only if the first step is completed can the next one be carried out, and so on and so forth. In this way, the user is forced to complete the whole process safely from start to finish, and failure to do so is considered a failure in the scene.

To simulate each of the steps, 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.

2.1. 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:

Missions: This panel will indicate which type of signal will indicate where to find the next mission. This for guiding the user during the scene.

Missions Panel You must complete some missions. To find them look for the following sign:    Continue

Tabla 1. Missions Panel.

Teleport: This panel explains the teleport tool and how to use it. This tool is important because in some situations the scenario will be much bigger than the place that you can use for the virtual reality, so walking is not an option.

Teleport Panel To move around the stage, use the teleport tool. This is activated using the main button (A or X). Continue

Tabla 2. Teleport Panel.

2.2. Fail Panel

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

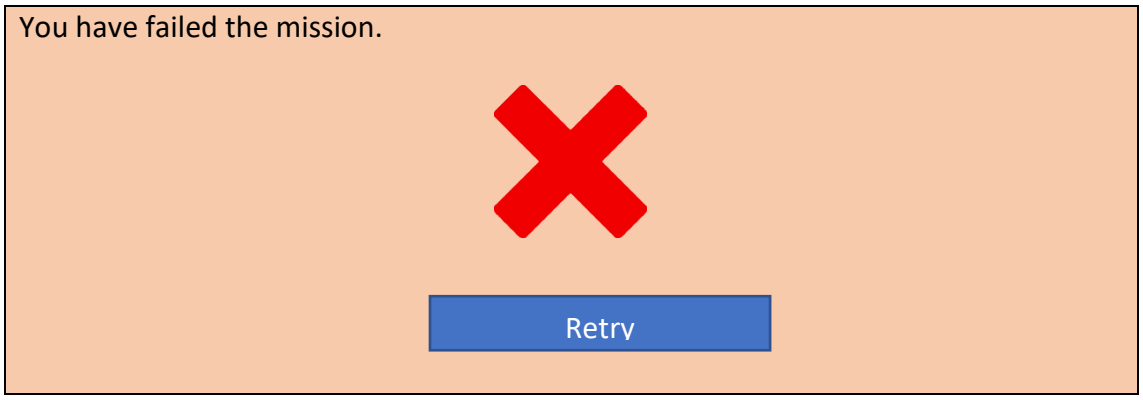

Tabla 3. Fail Panel.

2.3. Success Panel

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

Tabla 4. Success Panel.

2.4. Fail Scene Panel

This panel will appear when you fail 3 times during the scene. This is a measure taken to prevent the user from remaining immersed in virtual reality for too long, as it can cause dizziness. In addition, it is a way to force the user to grasp all the concepts of each scene.

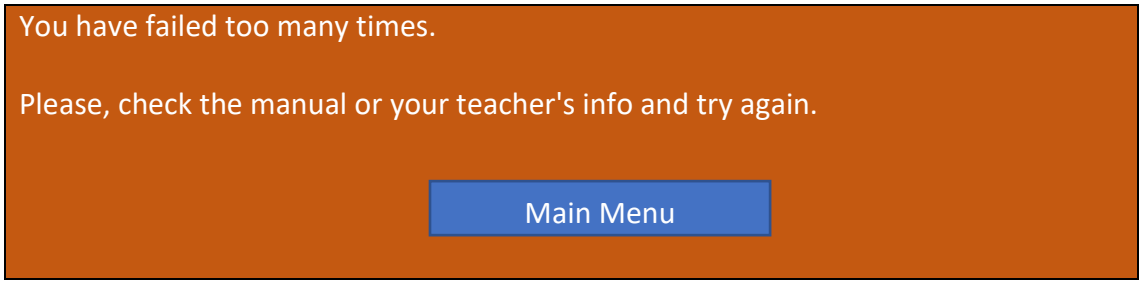

Tabla 5. Fail Scene Panel.

2.5. PPE Mission

This mission shall be common to each and every scene. It shall consist of the selection of the personal protective equipment necessary for the task to be performed.

<table>
<tr><td colspan="2">MISSION 1:

Select the security equipment for the task.

Continue</td></tr>
<tr><td colspan="2">To select an object:

Touch it with your hand.

Pull the trigger of your middle finger.

Continue</td></tr>
<tr><td colspan="2">Select the objects and check if your selection is correct:

Check</td></tr>
<tr><td colspan="2">Checking your security equipment...</td></tr>
<tr><td>(Success Panel)</td><td>(Fail Panel)

This is not the right equipment.</td></tr>
</table>

Tabla 6. PPE Mission.

2.6. Mission (Example)

These panels will vary according to the task to be performed, containing information, questions, dialogues or orders to be completed. There will be between 3 and 6 missions for each situation.

MISSION 5: The tank of water is full and it must be emptied. Continue	
Quiz: Where the water should be emptied? 1. The floor. 2. Into a waste container.	
Take the bucket and put it in the right place to pour the water in it.	(Fail Panel)
Quiz What do you do with the waste container? 1. Run it through a filter system and dispose the grinding scrap. 2. Use it to water the lawn. 3. Dump the water into the next canal.	
(Success Panel)	(Fail Panel)

Tabla 7. Ejemplo de una misión.

3. DEVELOPMENT OF 3D ENVIRONMENT.

In the development of tools in videogame graphic engines, we usually work with objects. This is why 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 has been used.
- **Rigging:** This is the process by which the different parts of an object are separated in order to create a controllable system that allows effective and efficient animations. Blender has been used as the software. This will be necessary for those objects that are going to be animated.
- **Texturizado:** This is the process by which 3D objects are given colour and detail. Blender has been used as software.

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 and many more decoration elements.

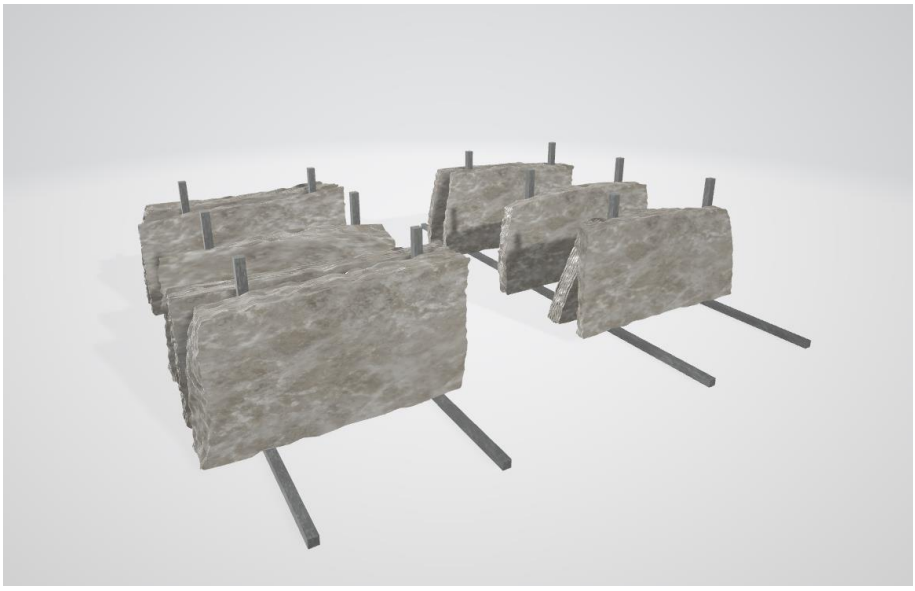

Figure 2: Stone slabs.

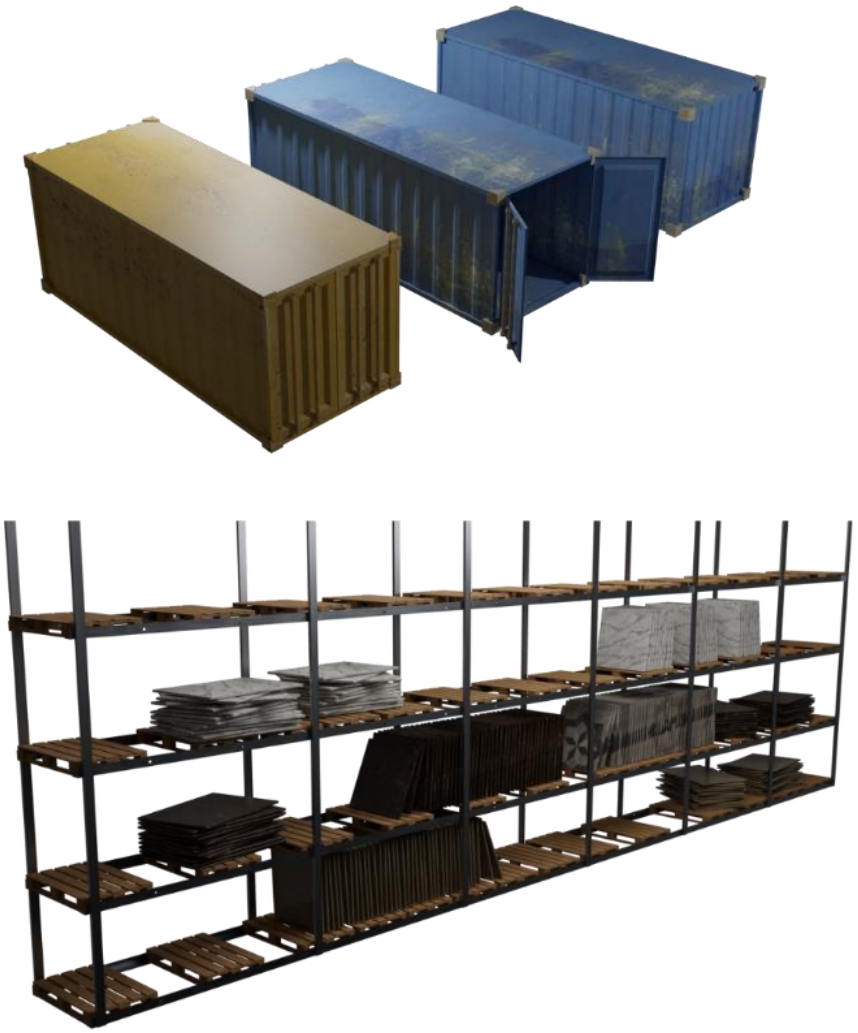

Figure 3: Containers and stores.

- Personal equipment and many handy objects.

Figure 4: Other elements.

- Specific machines for the different situations.

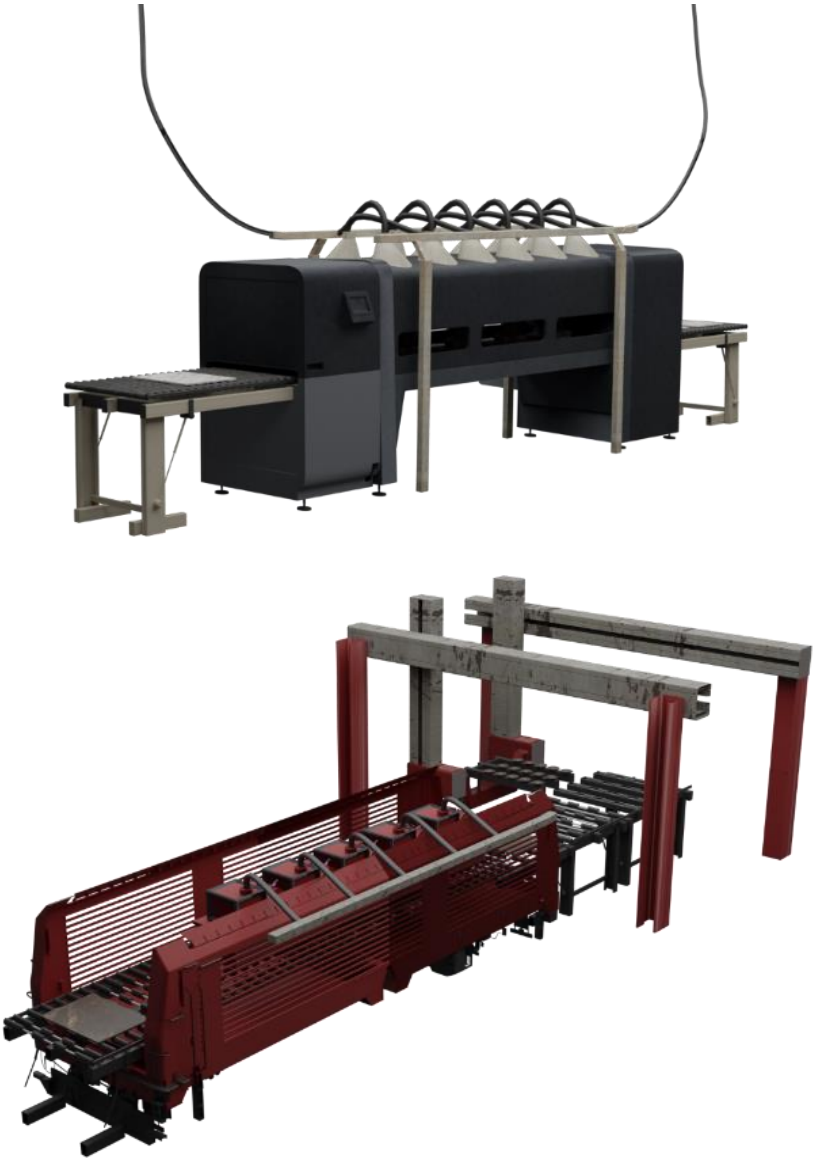

Figure 5: Specific machines.

Figure 6: Saw machine.

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:

- **Stone factory:** It is a factory based on a natural stone plant. It is divided into several rooms, including the machine room, the chemical store and the waste extraction room.

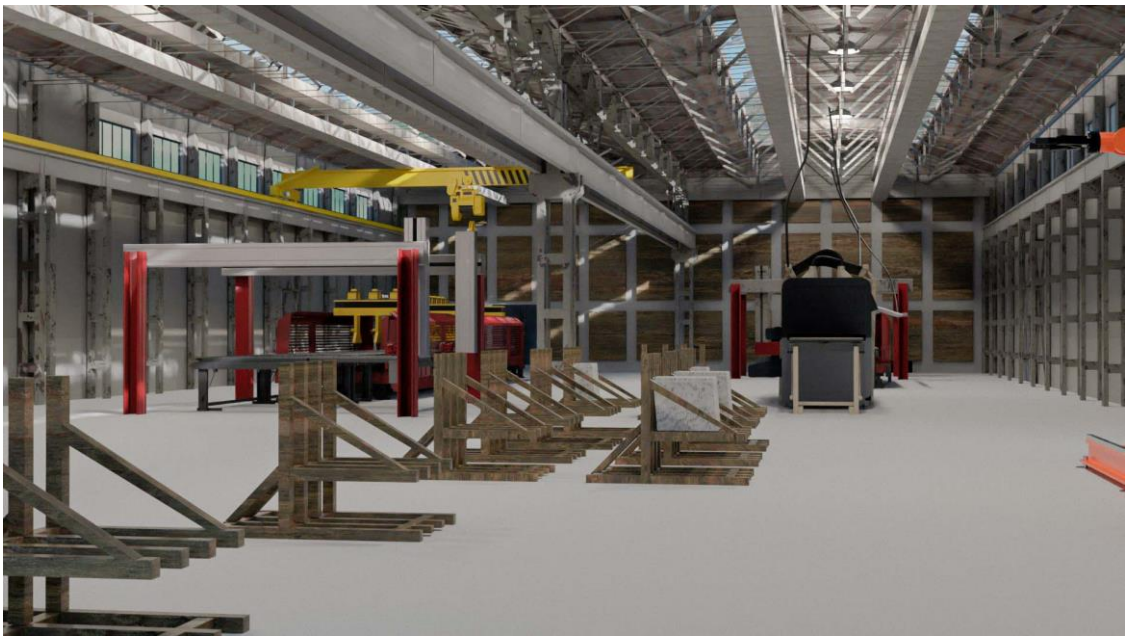

Figure 7: Stone factory scenario.

- **Construction site:** This model shows a construction site with various huts, scaffolding, the skeleton of a building and some working machines.

Figure 8: Construction site scenario.

- **House to be finished:** This is a villa with a swimming pool, with some unfinished finishes. It has a management office and some decorative objects.

Figure 9: House scenario.

4. 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.**

4.1. 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 it will present is shown in the flowchart in Figure 9.

4.1.1. 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. Previously, it is necessary to select the language in which we want the application to run.

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

Figure 10: Main Menu

4.1.2. Scene N

This is the important part of the tool. In this section you will find each of the missions developed, the first one being the selection of personal protective equipment (PPE).

If a mission is passed, we will move on to the next one. If it is the last mission, we will return to the main menu. In case of failure of a mission, we will have the option to try again, unless the number of failures during the scene is greater than 2.

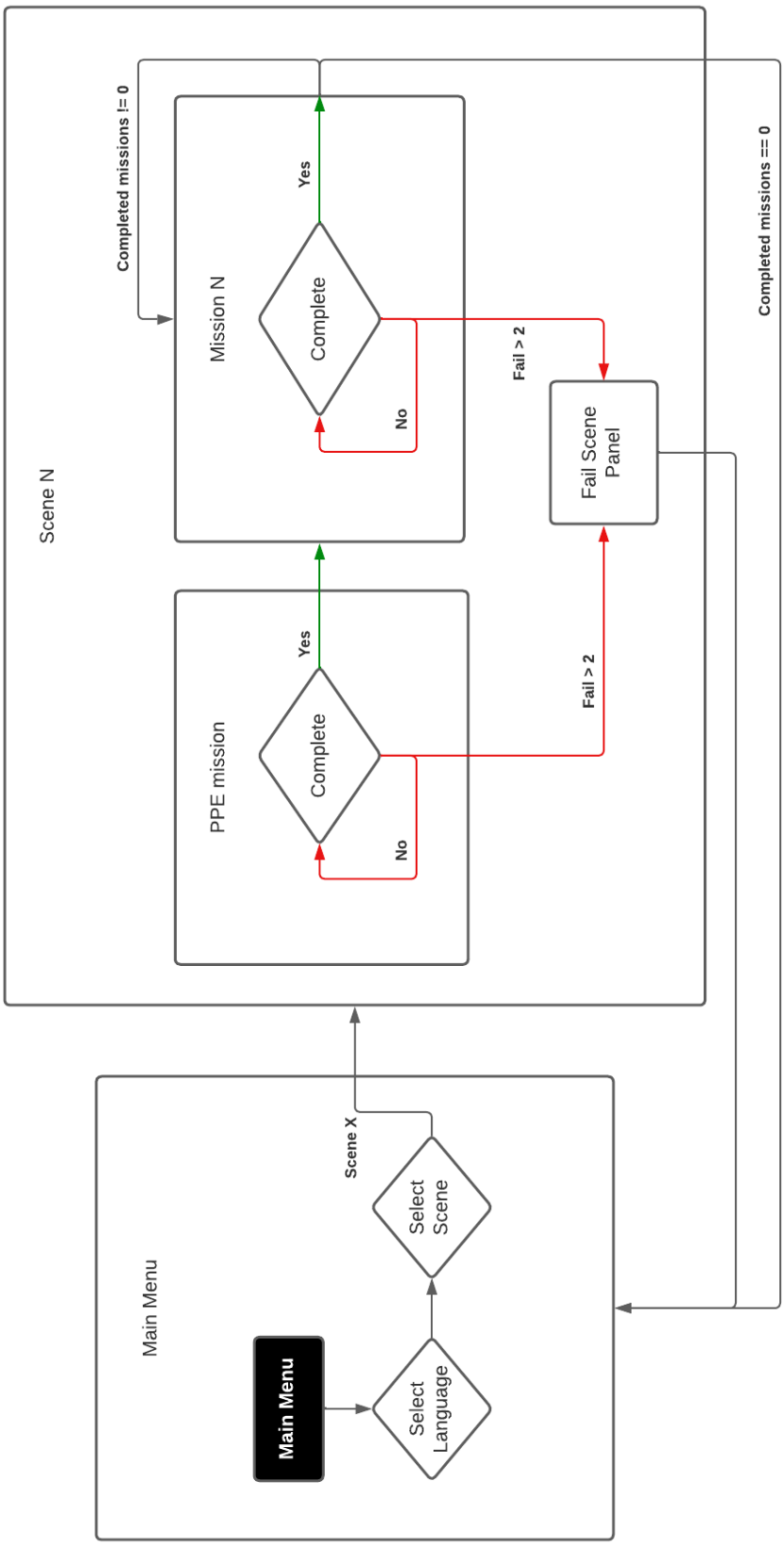

Figure 11: Tool flow diagram.

4.2. Data modeling

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 a 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.

4.3. 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.

4.4. 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 interaction level. But here only the common and important points to achieve the goal will be captured.

4.4.1. 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.

4.4.2. 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 of the input handles. This can be seen in the *XR Interaction Debugger* window, which you can access from Window/Analysis.

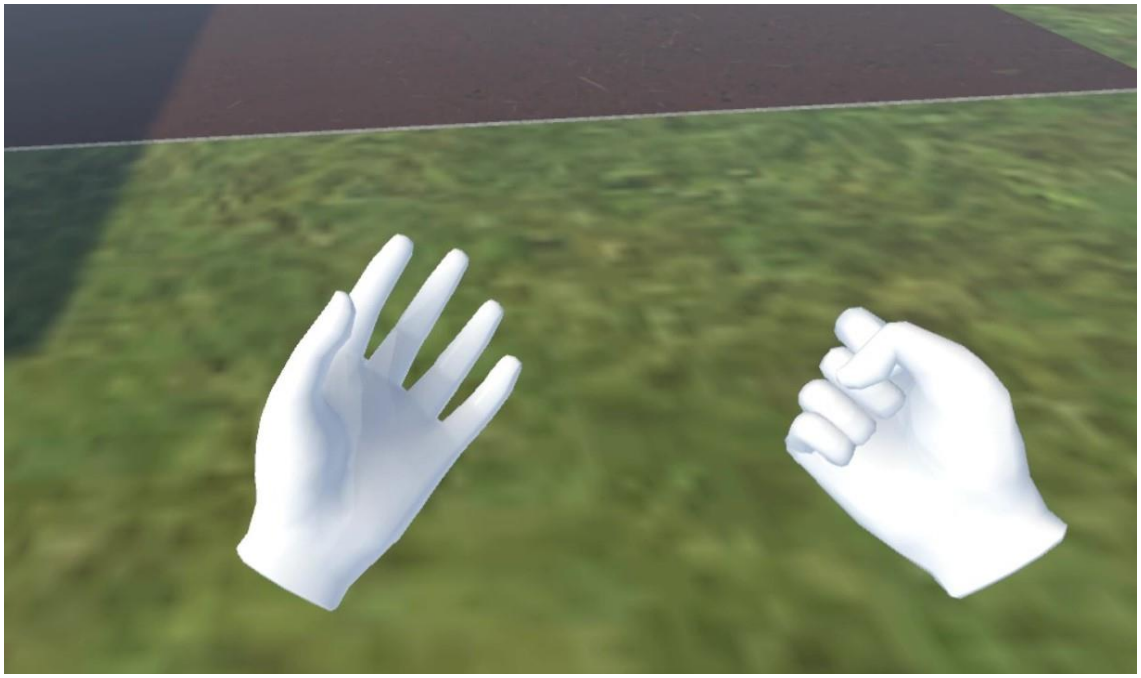

Figure 12: Hand Input.

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 13: 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 14: 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 stores 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 15: UpdateEvents() function.

4.4.3. 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. La función que se activa al pulsar un botón de respuesta es *SelectButton(Button but)*.
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 16: Quiz

4.4.4. 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 17: Translation line

4.4.5. 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 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 18: Ray Interactor.

4.4.6. NPC

The world of Artificial Intelligence is very broad and encompasses many different branches. In the world of video games, it is usually understood as an NPC (Non Playable Character) that makes decisions in a semi-autonomous way. In this case, it is a character that will tell the user what the next mission is and will interact with him on certain occasions.

To make the character go to the different missions, the *NPCTargetPath* class has been created, which takes advantage of the Nav-mesh functionality provided by Unity. This allows you to create an agent that moves through a space with obstacles. To generate the code fragment of this, the following steps must be followed and understood:

1. Generate the variables that are considered important. In this case, all of them will be public variables that can be modified. Mainly the destinations to which the character must move in order.
2. The *SetDestination()* function is created, which is activated from the application and is responsible for activating the next destination of the character, making it automatically go to this destination, also activating the walking animations.
3. The *Update()* function will check if the character is moving or not, to activate the static animations.

Figure 19: NPC interaction.