VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Viet Tuan Vu

VR MODELING OF A POWER PLANT FOR THE PURPOSE OF 3D VISUALISATION, MAINTENANCE TRAINING AND DISASTER SIMULATION SCENARIO

Information Technology
2017

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Viet Tuan Vu
Title	VR Modeling of a Power Plant for the Purpose of 3D Visualisation, Maintenance Training and Disaster Simulation Scenario.
Year	2017
Language	English
Pages	72 + 0 Appendices
Name of Supervisor	Timo Kankaanpää

Virtual reality technology has been widely considered as a massive development of 3D computer graphics. Along with modern computer development, it has become surprisingly popular and effective. Despite the common thought that virtual reality is limited to the computer game industry, there are many areas it can be applied to such as arts, tourism, education or psychology. The combination of this technique and modern software systems can guarantee effective industrial application such as designing, manufacturing and maintenance.

The objective of this thesis was to develop an application on a PC platform to simulate a power plant with virtual operations and maintenance. The application was implemented to describe specific maintenance and disaster scenario simulation of actual issues in an industrial facility. Within the scope of the thesis, Unreal Engine was used as the main tool to design and build the application, along with the help of HTC Vive in order to put virtual reality technique into use.

The application was successfully implemented with all required functions and it has been continuously tested. The application's use cases, design and implementations will be mentioned in this document.

Keywords Virtual reality, power plant, 3D visualization, virtual maintenance, Unreal Engine

CONTENTS

ABSTRACT
1 INTRODUCTION 9
2 RELEVANT TECHNOLOGIES 10
2.1 Virtual Reality 10
2.1.1 Definition 10
2.1.2 Operating principle 10
2.2 HTC Vive 10
2.2.1 HTC Vive: Headset 12
2.2.2 HTC Vive: Controllers 12
2.2.3 HTC Vive: Base stations 13
2.3 Unreal Engine 4 14
2.3.1 Blueprints Visual Scripting 14
2.3.2 Blueprints Visual Scripting vs C++ 16
2.4 Blender 17
3 APPLICATION DESCRIPTION 18
3.1 Quality Function Deployment (QFD) 18
3.1.1 Normal requirements (Priority level 1) 18
3.1.2 Expected requirements (Priority level 2) 19
3.1.3 Exciting requirements (Priority level 3) 19
3.2 Application use case diagram 19
4 UNREAL ENGINE 4 TERMINOLOGY 22
4.1 Projects 22
4.2 Classes 22
4.3 Objects, Actors, Pawns and Components 23
4.4 Brushes, Levels 25
5 IMPLEMENTATION 27
5.1 Virtual Reality Template 27
5.1.1 Motion Controller Teleportation 27
5.1.2 Object Interaction 28
5.2 Default Map 32
5.2.1 Exhibition on screen 32

5.2.2 Scene Changing Doors 34
5.3 Control Room Map 36
5.3.1 Return Default Scene 36
5.3.2 Warning Screens 36
5.3.3 IoT Ticket Screen 38
5.3.4 Virtual Keyboard 40
5.4 Engine Room Map 43
5.4.1 Steam Effect 43
5.4.2 Blinking exclamation mark 45
5.4.3 Door interaction 47
5.4.4 Mini-map 50
5.4.5 Engine filter maintenance work 56
5.4.6 Steam leak scenario 60
6 TESTING AND DEBUGGING 62
6.1 Software testing 62
6.2 UE4 Debugging 63
7 CONCLUSION 69
REFERENCES 70

LIST OF FIGURES

Figure 1. HTC Vive bundle essential parts. 11
Figure 2. HTC Vive controller buttons 13
Figure 3. HTC Vive base stations setup position 14
Figure 4. Comment box example 15
Figure 5. Breakpoint debugger example 16
Figure 6. Watching Values debugger example 16
Figure 7. Application use case diagram 20
Figure 8. Content structure example 22
Figure 9. Most common Parent Classes 23
Figure 10. Actors example 24
Figure 11. Component example 25
Figure 12. BSP Brushes example 25
Figure 13. Virtual Reality UE4 template 27
Figure 14. Teleporting Circle Indicator 28
Figure 15. PickupCube interaction 28
Figure 16. Object selected in the scene 29
Figure 17. Object selected in the World Outliner 29
Figure 18. Blueprint Creation 30
Figure 19. Interface Implementation 30
Figure 20. StaticMeshComponent Settings 30
Figure 21. Pickup and Drop events implementation 31
Figure 22. Showcase Screens gameplay 32
Figure 23. Enable Web Browser Widget plugin 32
Figure 24. Widget Blueprint structure 33
Figure 25. Blueprint class structure 33
Figure 26. Two scene changing doors gameplay 34
Figure 27. Door usage 34
Figure 28. Door Blueprint Class components 35
Figure 29. Warning Screen gameplay 37
Figure 30. Warning Screen settings 37
Figure 31. WarningTxt_Mat material creation 38

Figure 32. IoT screen gameplay 38
Figure 33. BrowserMonitorIoT Class Configure 39
Figure 34. WidgetInteraction Component configuration 39
Figure 35. First time user authentication 40
Figure 36. Virtual Keyboard structure 41
Figure 37. Button focusable configuration 41
Figure 38. Widget Component configuration 43
Figure 39. Steam effect gameplay 44
Figure 40. Particle System Modification Interface 44
Figure 41. Steam effect properties configuration 45
Figure 42. Blinking mark and worker model for user attention 45
Figure 43. Blinking mark configuration 46
Figure 44. Timeline_0 creation 47
Figure 45. Opening door procedure gameplay 48
Figure 46. Door structure 48
Figure 47. SlidingDoorTimeline 50
Figure 48. Mini-map gameplay 50
Figure 49. Mini-map required objects 51
Figure 50. MiniCam rotation settings 51
Figure 51. MapRendering Settings 53
Figure 52. MiniCam’s Texture Target 53
Figure 53. Minimap_Mat material 53
Figure 54. Minimap Widget configuration 54
Figure 55. Minimap Widget usage in Minimap Blueprint Class 54
Figure 56. minimapSocket Creation 55
Figure 57. Engine filter maintenance procedure 56
Figure 58. Valve Closing Procedure 60
Figure 59. Door disappearance 63
Figure 60. DoorInteract enable Debugging (1) 64
Figure 61. Door Debugging (1) 64
Figure 62. Door Debugging (2) 64
Figure 63. Door Debugging (3) 65

Figure 64. Next node execution 65
Figure 65. Door Debugging (4)... 66
Figure 66. Correct door workflow.. 68

LIST OF TABLES

Table 1. Minimum system requirements 11
Table 2. Headset specifications 12
Table 3. Functions testing results 62

LIST OF CODE SNIPPETS

Code Snippet 1. Code example of an if statement 15
Code Snippet 2. Input enable 35
Code Snippet 3. Trigger Button implementation.. 36
Code Snippet 4. Return default scene script 36
Code Snippet 5. Trigger button implementation for screen interaction 40
Code Snippet 6. Virtual Keyboard Buttons Implementation.............................. 42
Code Snippet 7. Targeting IoT Screen implementation 43
Code Snippet 8. Blinking mark disappear implementation 46
Code Snippet 9. Blinking function implementation 47
Code Snippet 10. Instruction text toggling implementation 49
Code Snippet 11. Door interaction implementation 49
Code Snippet 12. MiniCam follows user location 52
Code Snippet 13. Outside tick Event called 52
Code Snippet 14. Mini-map actor behaviour implementation............................ 55
Code Snippet 15. Minimap Widget Creation at beginning................................. 56
Code Snippet 16. Top cover implementation (1).. 57
Code Snippet 17. Top cover implementation (2).. 57

Code Snippet 18. Top cover implementation (3) 58
Code Snippet 19. Top cover implementation (4) 59
Code Snippet 20. Dirty filter implementation (1) 59
Code Snippet 21. Clean filter implementation 60
Code Snippet 22. Valve implementation 61

LIST OF ABBREVIATIONS

VR	Virtual Reality
IoT	Internet of Things
UE4	Unreal Engine 4

1 INTRODUCTION

Modern virtual reality technology builds upon the idea that dates back to the 1800s, to the very beginning of practical photography. In 1838, the first stereoscope was invented, using twin mirrors to project a single image. The term “virtual reality”, however, was first put to use in the mid-1980s when Jaron Lanier, founder of VPL Research, began his gear development, including goggles and gloves. After time and efforts of many individuals and companies has been spent, together with massive amount of technological improvements, virtual reality can finally live up to its promise. It is not only a revolution to the gaming industry but it also has so much potential in other fields.

For the virtual operations and maintenance work, the integration of multiple domains is required and it is important to synchronize multiple related technologies so that the application can reach its peak effectiveness. This virtual environment allows the maintenance engineer to virtually maintain the product to make it operable. The virtual maintenance system provides relevant training instructions in an easy and effective way.

This thesis project uses Unreal Engine 4 as the main software to develop an application on Windows platform for virtual environment creation. This environment simulates a power plant with virtual industrial operations and maintenance. The application is characterized by real-time simulation and interaction with a virtual 3D world. HTC Vive is required for testing and running process.

2 RELEVANT TECHNOLOGIES

This section of the document will introduce the technologies, frameworks and tools used to create the application. As mentioned before, Unreal Engine is used as the main tool to develop, design and build the application and Blender is used to edit and split 3D objects. Finally, the HTC Vive is the essential hardware requirement for testing and running.

2.1 Virtual Reality

2.1.1 Definition

Virtual reality (VR) can be described as a three-dimensional, computer generated environment in which the viewer can experience a different reality. In this virtual world, this person can have real-time interaction with objects or execute multiple actions.

2.1.2 Operating principle

In order to deliver the most realistic experience, VR commonly uses a combination of optics, headsets and controllers. A VR headset displays an image and as soon as the user moves his/her head the system modifies that image to make the user have a sense of looking and moving around. /1/ In some cases, the headset includes a headphone to enhance the experience with the aid of 3D audio.

2.2 HTC Vive

HTC Vive is a virtual reality headset developed by HTC and Valve Corporation. The headset uses "room scale" tracking technology, allowing the user to move in 3D space and use motion-tracked handheld controllers to interact with the environment. /2/ HTC Vive bundle consists of one VR headset, two controllers and the Lighthouse base stations as its essential components. This bundle can be bought from the official website with the price of 599$. Figure 1 shows what these essential components look like.

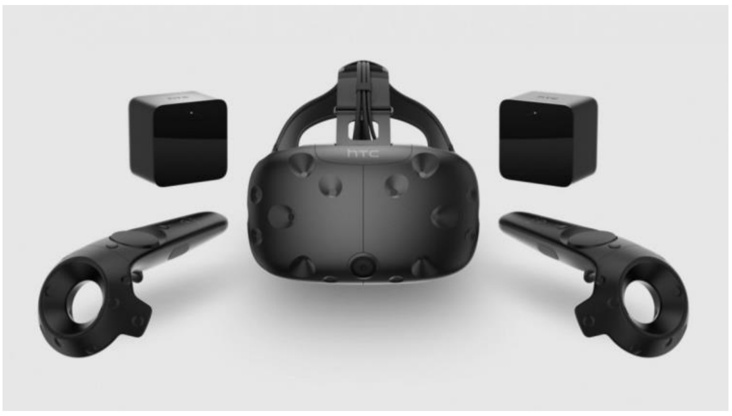


Figure 1. HTC Vive bundle essential parts.

Because of the high resolution display and intensive refresh rate, Vive requires some efficient computer hardware to guarantee a good experience for the user. Table 1 shows the recommended minimum computer specifications /2/.

Table 1. Minimum system requirements

Criteria	**Minimum requirements**
Graphics	NVIDIA® GeForce® GTX 1060 or AMD Radeon™ RX 480, equivalent or better
Processor	Intel® Core™ i5-4590 or AMD FX™ 8350, equivalent or better
Memory	4 GB RAM or more
Video out	HDMI 1.4, DisplayPort 1.2 or newer
USB ports	1x USB 2.0 or better port
Operating system	Windows® 7 SP1, Windows® 8.1 or later, Windows® 10

2.2.1 HTC Vive: Headset

HTC Vive headset contains a gyro sensor, an accelerometer and a laser position sensor, which work together to track the head position. There are also multiple lenses in the headset in order to focus, reshape the picture for each eye and create a stereoscopic 3D image by angling the two 2D images to mimic how a person's eyes view the world /3/. All these heavy graphics will be processed by the computer, which does not only make the headset light and comfortable but also keeps it up to date by simply upgrading the computer.

Table 2 shows the specifications of the HTC Vive headset /2/.

Table 2. Headset specifications

Criteria	**HTC Vive headset**
Screen	Dual AMOLED 3.6'' diagonal
Resolution	1080 x 1200 pixels per eye (2160 x 1200 pixels combined)
Refresh rate	90 Hz
Field of view	110 degrees
Sensors	SteamVR Tracking, G-sensor, gyroscope, proximity
Connections	HDMI, USB 2.0, stereo 3.5 mm headphone jack, Power, Bluetooth
Input	Integrated microphone

2.2.2 HTC Vive: Controllers

HTC Vive controllers let the viewer wirelessly interact with virtual objects by a trackpad, a dual stage trigger button under the forefinger, a pressure-sensitive grip button and a home button for each hand. All these buttons are described in Figure 2.

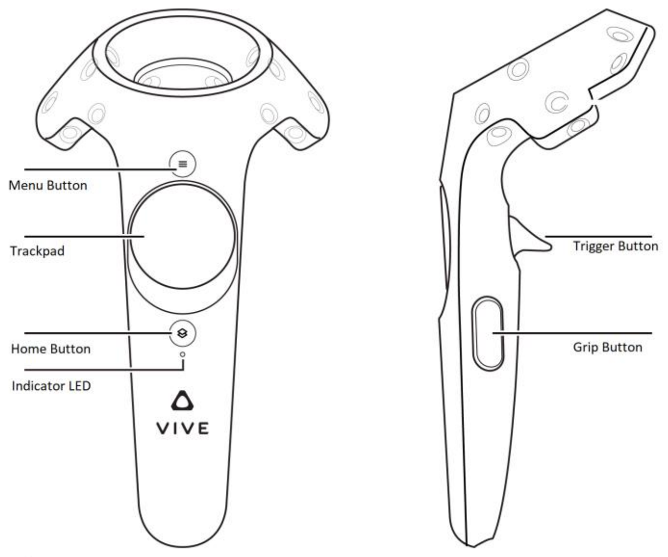


Figure 2. HTC Vive controller buttons

2.2.3 HTC Vive: Base stations

HTC Vive base stations are two wireless infrared "Lighthouse" boxes which track all movements with sub-millimeter precision with the aid of 37 sensors in the headset (more than 70 in total if including the controllers). The "Lighthouse" technique got it name from using non-visible light, which comes from an array of stationary LEDs, plus a pair of active laser emitters that spin rapidly. This array of LEDs flashes, up to 60 times per second, and then one of the two spinning laser sweeps a beam of light across the room. The sensors dotted all over the headset or the controller pick up the light of the LEDs, and the headset begins timing the millisecond it gets hit by this light. Then, it waits until one of its sensors gets hit by the laser beam, and uses the relationship between where the sensor that was hit is, and when the beam hit it to mathematically calculate the exact position relative to the base stations.

In order to have the best experience, Room Scale VR requires a minimum of 2m x 1.5m free space. The maximum distance between the base stations is 5m. In order to see the base stations from anywhere in the play area, it is recommended to place the base stations at opposite corners of the free space, near the edges of the play area as in Figure 3.

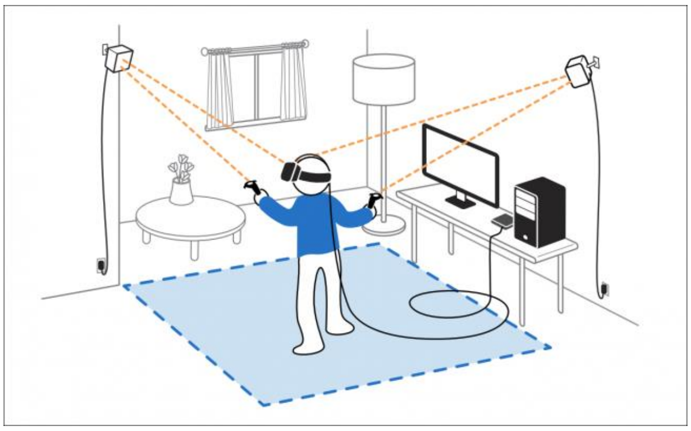


Figure 3. HTC Vive base stations setup position

2.3 Unreal Engine 4

The Unreal Engine is a game engine developed by Epic Games. Although primarily developed for first-person shooters, it has been successfully used in various other genres. With its code written in C++, the Unreal Engine features a high degree of portability and is a tool used by many game developers today. It has been awarded by Guinness World Records as "the most successful video game engine". /4/ Unreal Engine can be downloaded free of charge from the official website.

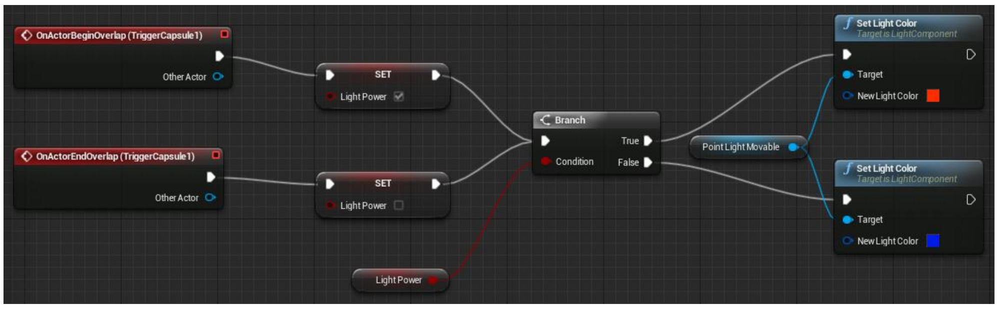
Unreal Engine is made with the C++ programming language. It enables developers to deploy applications to multiple platforms such as Windows, Linux, Mac OS, Xbox One and HTML5.

In March 2014, Epic Games announced that the Unreal Engine 4 would no longer be supporting UnrealScript, but instead support game scripting in C++. Visual scripting would be supported by the Blueprints Visual Scripting system, a replacement for the earlier Kismet visual scripting system. /4/

2.3.1 Blueprints Visual Scripting

Visual programming language (VPL) is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually. A VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols, used either as elements of syntax or secondary notation. /5/

Unreal Engine 4 introduces a node-based visual programming language called Blueprints. Code snippet 1 is an example of an **if statement** by using the **branch** node.



Code Snippet 1. Code example of an if statement

Figure 4 describes how Blueprints code can be organized by creating comment boxes.

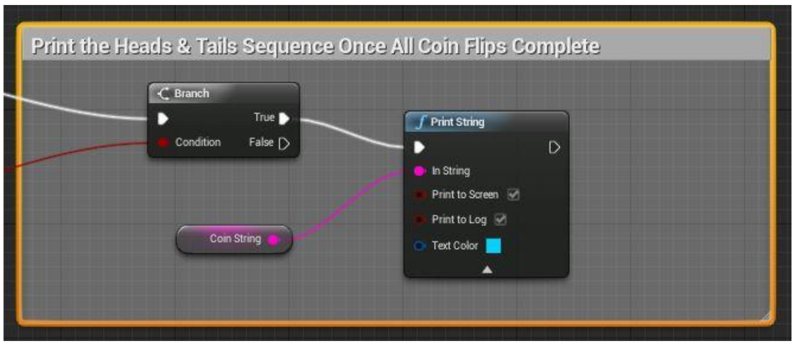


Figure 4. Comment box example

When working with Blueprints, to diagnose any problems, the Blueprint Visual Scripting has its own debugger and one of the most powerful aspects of it is the ability to add Breakpoints to any node. By adding Breakpoints to nodes, when reaching the node with the Breakpoint in runtime, the application will pause and jump to that node so that the scripts will be checked to see where issues are occurring.

Another useful debugging feature is Watch Values, which allows developers to choose the variables in their Blueprints to observe. This allows them to see these variables values while the application is running. Figure 5 and figure 6 describe an example of each of these features.

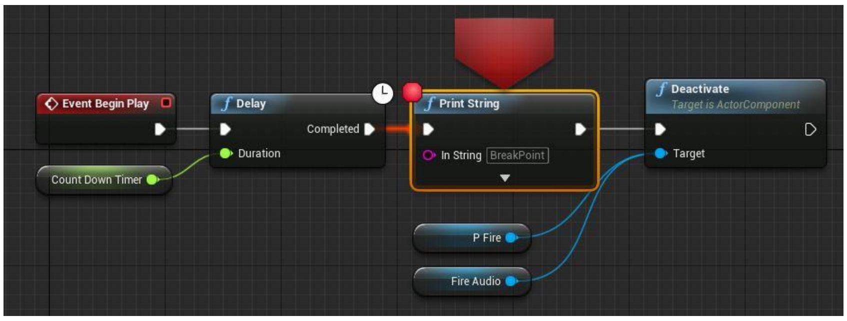


Figure 5. Breakpoint debugger example

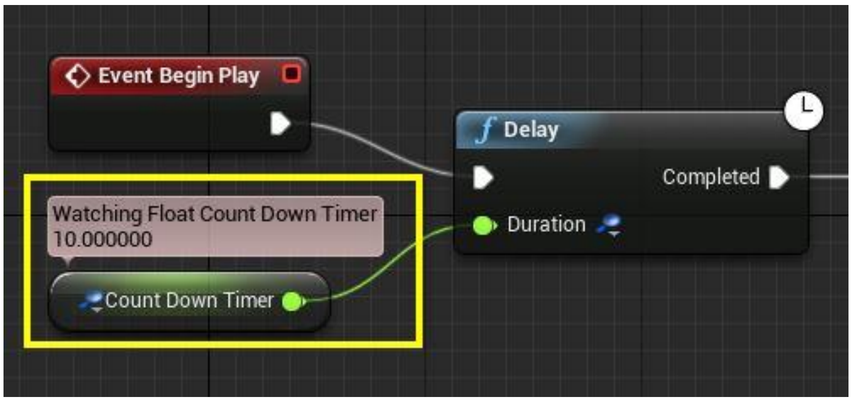


Figure 6. Watching Values debugger example

2.3.2 Blueprints Visual Scripting vs C++

There are hardcore coders who consider C++, C and/or Java to be very good, and there are those who dislike looking at any code. There are also people in the middle who can code, but do not want to spend much time to learn and master a programming language. This is where Unreal Engine 4 shines by providing two toolsets for programmers with the possibility of using them in tandem to accelerate development workflows. New gameplay classes, Slate and Canvas user interface elements, and editor functionality can be written with C++, and all changes will be reflected in Unreal Editor after compiling with either Visual Studio or XCode. The Blueprints visual scripting system is a robust tool which enables classes to be created in-editor through wiring together function blocks and property references. /6/

Whether the application was made with C++, Blueprints, or a combination of the two, the underlying Unreal architecture is the same. Blueprints method can be easily approached and time-saving but when the algorithm becomes complicated, traditional coding looks much cleaner and better maintainable. Duplicating nodes, dragging lines can be too much for a 27 inch monitor to be visible even partially.

As mention earlier, the thesis application is made by Unreal Engine version 4.13 and all scripts are made with Blueprints method.

2.4 Blender

Blender is a free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. /7/

Blender is used in this project to edit the pivot point, split parts of 3D objects and export them into correct format for the Unreal Engine.

3 APPLICATION DESCRIPTION

The application simulates a power plant in a virtual world in which the users can freely move and interact with 3D objects. The engine model in the power plant was taken from Wärtsilä Finland Oy, a global leader in advanced technologies and complete lifecycle solutions for the marine and energy markets /8/. The application also guides the users to do the maintenance task and take the necessary action in a disaster scenario.

3.1 Quality Function Deployment (QFD)

QFD is a quality management technique that defines the needs or requirements of the customer and converts them into technical requirements for a software product. QFD is a part of software requirements analysis which concentrates on achieving customer satisfaction to the fullest from the software engineering process.

The application requirements are categorized into three types based on their priorities, which are normal requirements, expected requirements and exciting requirements.

3.1.1 Normal requirements (Priority level 1)

Normal requirements are the fundamental requirements that the project must have. The customer is satisfied if these specifications are fulfilled, therefore, they are given the highest priority. The normal requirements for this application are listed below.

- The user can freely move around by walking a short distance or teleporting for long distance movement.
- The user can interact with multiple objects by using the trigger buttons.
- The interactable objects need to follow physics such as colliding and gravity falling.
- The user is guided for the maintenance work of changing the engine filter.

- The user is guided for urgent action when a disaster scenario happens. This means closing the valve for steam leak scenario.

3.1.2 Expected requirements (Priority level 2)

Expected requirement are essentially features not explicitly declared by the client for the software or system but their absence will be a significant dissatisfaction. The expected requirements for this application are:

- The application simulates a realistic environment for the user to blend in.
- Visual effects and animations are added to the scene.
- The scene should be lightweight in order to be rapidly processed in run time so that users can have the best possible experience when running.

3.1.3 Exciting requirements (Priority level 3)

Exciting requirement are needs beyond the scope of the project but they result in high satisfaction. These application requirements include:

- There are blinking exclamation marks to indicate the tasks the user needs to do.
- Returning to default scene after pressing left hand controller menu button.
- Engine room scene: A minimap in top view style attached to the right hand by pressing the right hand controller menu button.
- Control room scene: A screen to view and control the interface of Wapice's IoT ticket system.
- Control room scene: A virtual keyboard to fill in the username and password for the IoT screen.

3.2 Application use case diagram

Use case diagram describes a set of user actions with the system. Figure 7 shows the relationship between the external users with the functions of the system.

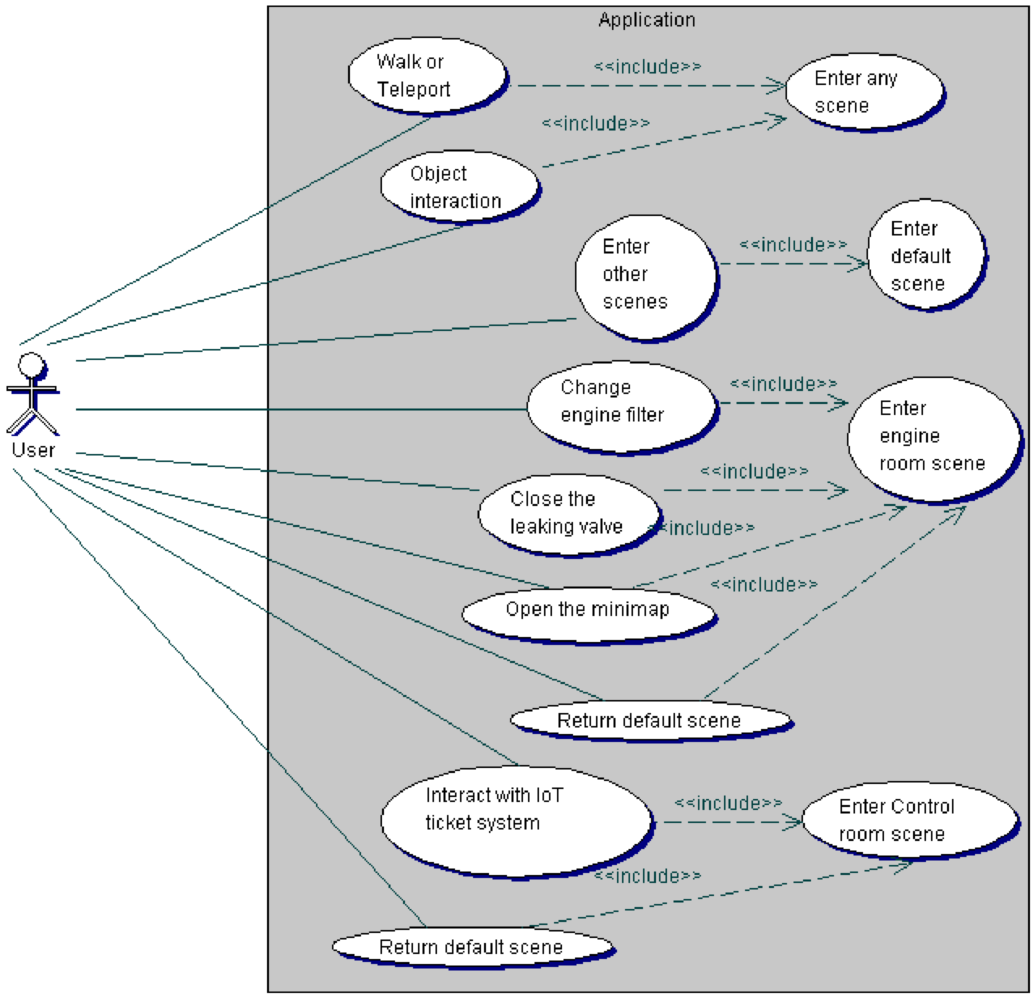


Figure 7. Application use case diagram

Although some application functions are available in every scene such as user movement and user interaction with objects, the others are categorized into three scenes: the default scene, the control room scene and the engine room scene.

The default scene is where the user will be located in after starting the application. The main function of this scene is to let the users become familiar with the environment and to know how the controller works, how they can move by walking or teleporting, and how they can interact with 3D objects. The scene also includes three big screens with running video for exhibition purposes and a power plant miniature. From this default scene, the user can choose to enter another scene.

The control room scene imitates an operation control center where the whole power plant facility can be monitored or controlled. In this room, there is a warning screen with notifications about maintenance works or actions to be done in case of an emergency. The user can control the IoT ticket system of Wapice by using the right controller to interact with the touch screen. The menu button of the left controller is used to relocate the user to the default scene.

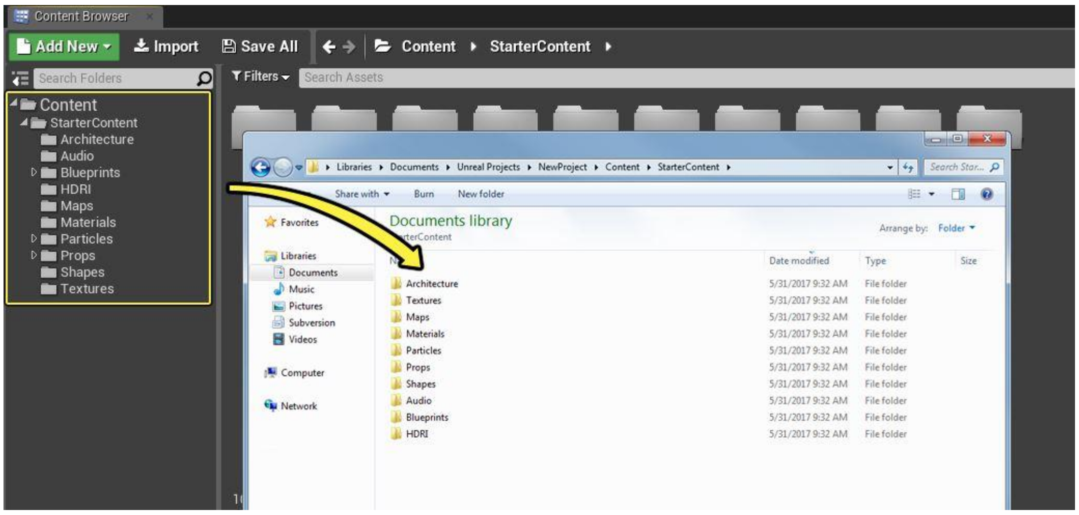
The engine room scene simulates the power plant establishment. This scene includes all the fundamental features of the applications such as engine models, maintenance work and emergency case. A mini map attached to the user’s right hand can be toggled on/off by the menu button of the right controller. The menu button of the left controller is still used to switch to the default scene.

4 UNREAL ENGINE 4 TERMINOLOGY

There are many terms in Unreal Engine 4, but this section of the document is dedicated to describing the commonly used ones in the application.

4.1 Projects

A **Project** is a self-contained unit that stores all the content and code that make up an individual game. The Hierarchy Tree of the Content Browser has the same directory structure as is found inside the Project folder on a disk. Figure 8 demonstrates this characteristic. /9/



Name	Date modified	Type	Size
Architecture	5/31/2017 9:32 AM	File folder	
Textures	5/31/2017 9:32 AM	File folder	
Maps	5/31/2017 9:32 AM	File folder	
Materials	5/31/2017 9:32 AM	File folder	
Particles	5/31/2017 9:32 AM	File folder	
Props	5/31/2017 9:32 AM	File folder	
Shapes	5/31/2017 9:32 AM	File folder	
Audio	5/31/2017 9:32 AM	File folder	
Blueprints	5/31/2017 9:32 AM	File folder	
HDRI	5/31/2017 9:32 AM	File folder	

Figure 8. Content structure example

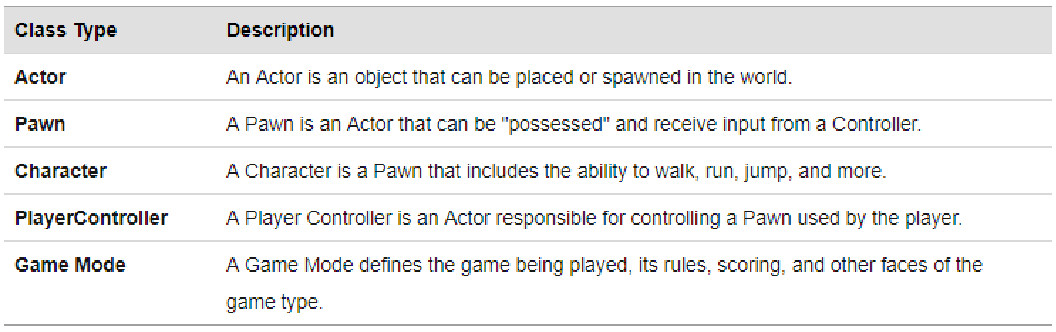
A project is referenced by a file with .uproject extension. This file has multiple functionalities such as to create, open or save a project. Multiple projects can be created, maintained and developed in parallel without any problem.

4.2 Classes

A **Class** defines the behaviors and properties of a particular Actor or Object used in the creation of an Unreal Engine game. Classes are hierarchical, meaning a Class inherits information from its parent Classes and passes that information to its children. Classes can be created in C++ code or in Blueprints. /10/

All of the classes in this thesis application are created in Blueprints. A Blueprint Class, often shortened as Blueprint, is an asset that allows content creators to easily add functionality on top of the existing gameplay classes. Blueprints are created inside of Unreal Editor visually, instead of by typing code, and saved as assets in a content package. /11/

In general, A Blueprints Class allows the developer to set up new classes using the Blueprints Visual Scripting system. After a Blueprint Class is created, components, functions, gameplay behavior with visual scripting can be added, and class variables default values can be set. There are several different types of Blueprints that can be created, however, before doing that a Parent Class must be specified. Figure 9 describes the most common Parent Classes used when creating a new Blueprint.



Class Type	Description
Actor	An Actor is an object that can be placed or spawned in the world.
Pawn	A Pawn is an Actor that can be "possessed" and receive input from a Controller.
Character	A Character is a Pawn that includes the ability to walk, run, jump, and more.
PlayerController	A Player Controller is an Actor responsible for controlling a Pawn used by the player.
Game Mode	A Game Mode defines the game being played, its rules, scoring, and other faces of the game type.

Figure 9. Most common Parent Classes

Most of all classes in the application are created with Actor Parent Class and the VR Main Camera is created with Pawn Parent Class. For the IoT screen and the minimap, the Widget Blueprint is used with the Parent Class UMG.UserWidget.

4.3 Objects, Actors, Pawns and Components

The base building blocks in Unreal Engine are called **Objects** and they contain a lot of the essential "under the hood" functionality. Just about everything in Unreal Engine 4 inherits (or gets some functionality) from an Object. In C++, UObject is the base class of all objects. /10/

An **Actor** is any object that can be placed into a level (a scene). Actors will be placed into a map, positioned around to generate an environment and their properties will be modified to achieve appropriate appearances or behaviors. Actors are a generic Class that support 3D transformations such as translation, rotation, and scale. Actors can be created or destroyed through gameplay code (C++ or Blueprints). In C++, AActor is the base class of all Actors. /12/ Figure 10 demonstrates multiple actors in one scene.

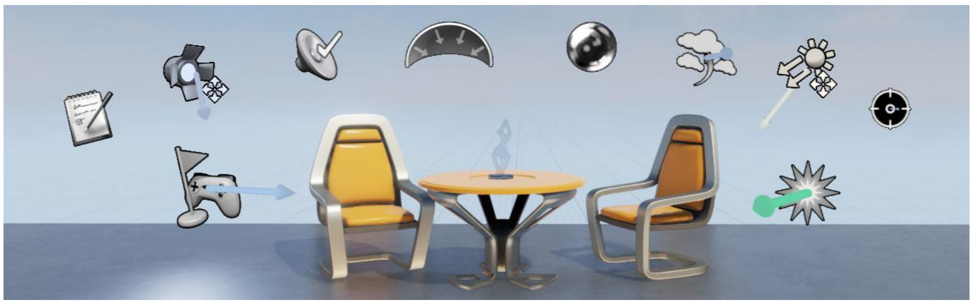


Figure 10. Actors example

The **Pawn** class is the base class of all Actors that can be controlled by players or AI. A Pawn is the physical representation of a player or AI entity within the world. This not only means that the Pawn determines what the player or AI entity looks like visually, but also how it interacts with the world in terms of collisions and other physical interactions. /13/

A **Component** is a piece of functionality that can be added to an Actor. Components cannot exist by themselves, however, when added to an Actor, the Actor will have access to and can use functionality provided by the Component. /14/ Figure 11 is an example of a component called Point Light. By adding this component, this actor can emit light like a spot light.

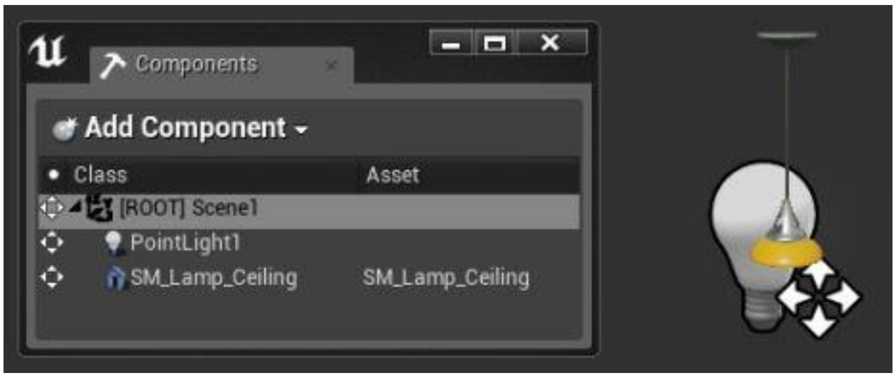


Figure 11. Component example

In general, **Actors** are instances of classes that derive from the AActor class; the base class of all gameplay objects that can be placed into the world. **Objects** are instances of classes that inherit from the UObject class; the base class of all objects in Unreal Engine, including **Actors**. This makes all instances in Unreal Engine **Objects**. However, **Actors** can be thought of as whole items or entities, while Objects are more specialized parts. **Actors** often make use of **Components**, which are specialized **Objects**, to define certain aspects of their functionality or hold values for a collection of properties. /15/

4.4 Brushes, Levels

A **Brush** is an Actor that describes a 3D volume that is placed in a level in order sketch out geometry of the environment (Binary Space Partitioning) and gameplay volumes. BSP Brushes are used to prototype or block-out the level for gameplay testing. Figure 12 is an example of BSP Brushes before and after polishing.

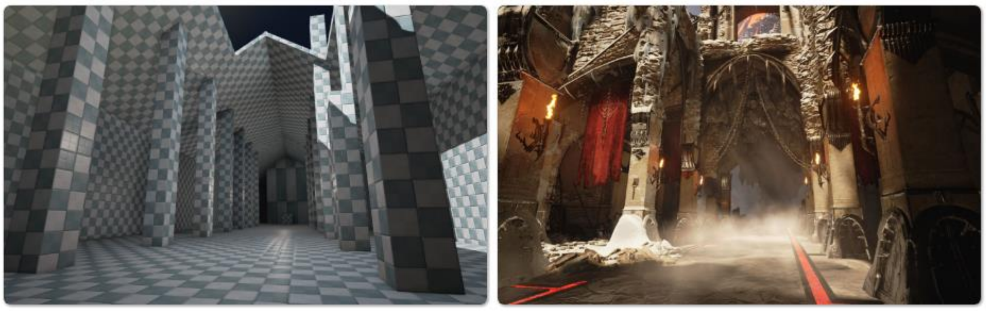


Figure 12. BSP Brushes example

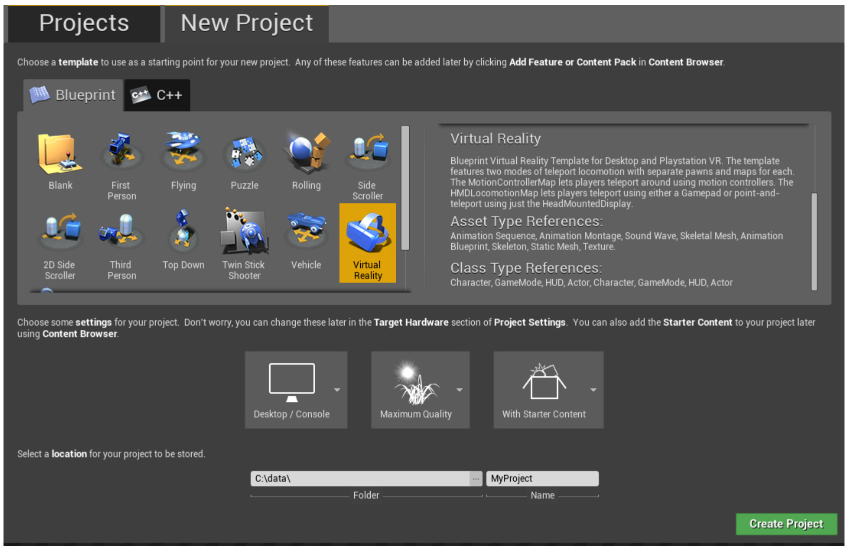
Volumes Brushes on the other hand have several uses depending upon the effects attached to them such as: Blocking Volumes (which are invisible and used to prevent Actors from passing through them), Pain Causing Volumes (which causes damage over time to any Actor that overlaps it) or Trigger Volumes (which are used as a way to cause events when an Actor enters or exits them). /10/

A **Level** is the gameplay area that is defined by the user. By locating and altering the properties of the Actors it contains, level can be created and modified. In the Unreal Editor, each Level is saved as a separate .umap file. A Level can also be called a map or a scene.

5 IMPLEMENTATION

5.1 Virtual Reality Template

Unreal Engine 4.13 introduced the official Virtual Reality template made entirely in Blueprint. This VR template targets desktop and console only and it supports Oculus Rift, HTC Vive and PlayStation VR out of the box. This template is considered as the fastest method to start a VR project. This can be done by opening an UE4 Editor and choosing Virtual Reality template in New Project tab (Figure 13).



Choose a **template** to use as a starting point for your new project. Any of these features can be added later by clicking **Add Feature or Content Pack** in **Content Browser**.

Virtual Reality

Blueprint Virtual Reality Template for Desktop and Playstation VR. The template features two modes of teleport locomotion with separate pawns and maps for each. The MotionControllerMap lets players teleport around using motion controllers. The HMDLocomotionMap lets players teleport using either a Gamepad or point-and-teleport using just the HeadMountedDisplay.

Asset Type References:
Animation Sequence, Animation Montage, Sound Wave, Skeletal Mesh, Animation Blueprint, Skeleton, Static Mesh, Texture.

Class Type References:
Character, GameMode, HUD, Actor, Character, GameMode, HUD, Actor

Choose some **settings** for your project. Don't worry, you can change these later in the **Target Hardware** section of **Project Settings**. You can also add the **Starter Content** to your project later using **Content Browser**.

Select a **location** for your project to be stored.

Folder: C:\data\
Name: MyProject

Figure 13. Virtual Reality UE4 template

5.1.1 Motion Controller Teleportation

Motion controller teleportation feature is included in Virtual Reality template. After pressing the touchpad of the controllers, a blue circle will appear to indicate the position that the user will be moved to. The moment the touchpad is released, this circle will disappear and the user will be located at a new location. Figure 14 demonstrates what the blue circle looks like.

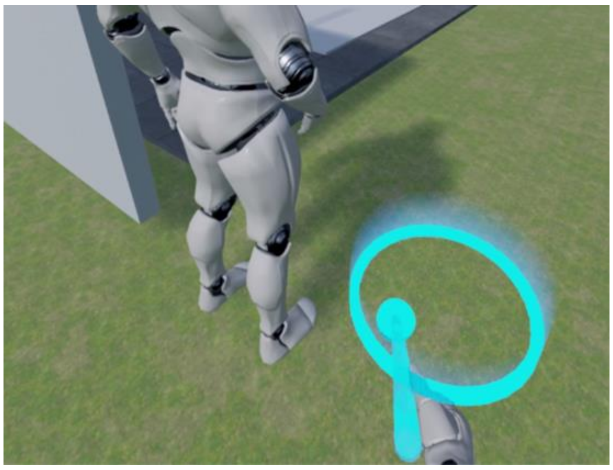


Figure 14. Teleporting Circle Indicator

5.1.2 Object Interaction

In the template, there are multiple objects which are instances of Blueprint class BP-PickupCube. By using the trigger button of the controllers, a user can grab and move these cubes (Figure 15). These objects were made to help the developers understand how to implement interactable objects.

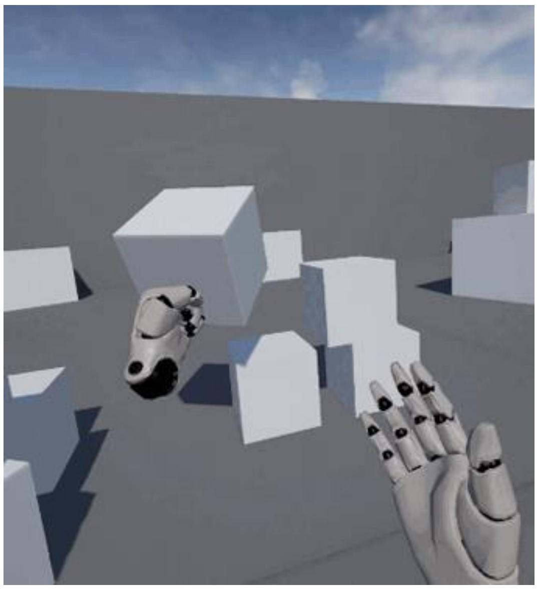


Figure 15. PickupCube interaction

In order to take an object in the scene and allow it to be interactable with the motion controller, there are multiple steps that need to be followed. Firstly, the object or actor needs to be selected. This can be done either by clicking it in the scene (Figure 16) or choosing it from the World Outliner (Figure 17). The World Outliner tab can be enabled by clicking Windows/World Outliner.

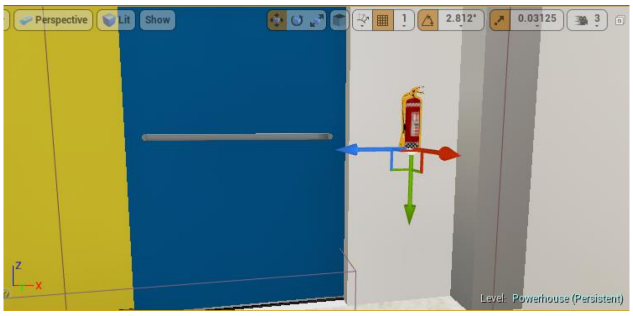
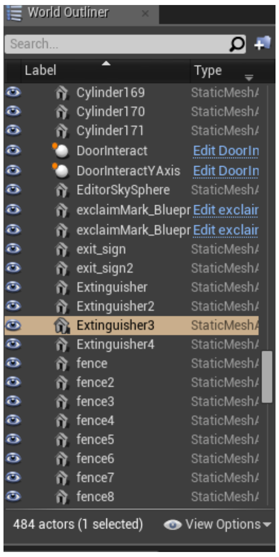


Figure 16. Object selected in the scene


World Outliner

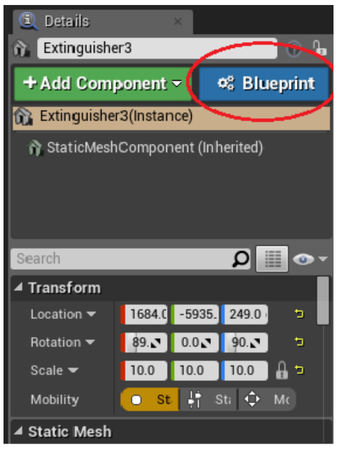
Search...

Label	Type
Cylinder169	StaticMeshA
Cylinder170	StaticMeshA
Cylinder171	StaticMeshA
DoorInteract	Edit DoorIn
DoorInteractYAxis	Edit DoorIn
EditorSkySphere	StaticMeshA
exclaimMark_Bluepr	Edit exclair
exclaimMark_Bluepr	Edit exclair
exit_sign	StaticMeshA
exit_sign2	StaticMeshA
Extinguisher	StaticMeshA
Extinguisher2	StaticMeshA
Extinguisher3	StaticMeshA
Extinguisher4	StaticMeshA
fence	StaticMeshA
fence2	StaticMeshA
fence3	StaticMeshA
fence4	StaticMeshA
fence5	StaticMeshA
fence6	StaticMeshA
fence7	StaticMeshA
fence8	StaticMeshA

484 actors (1 selected)   View Options

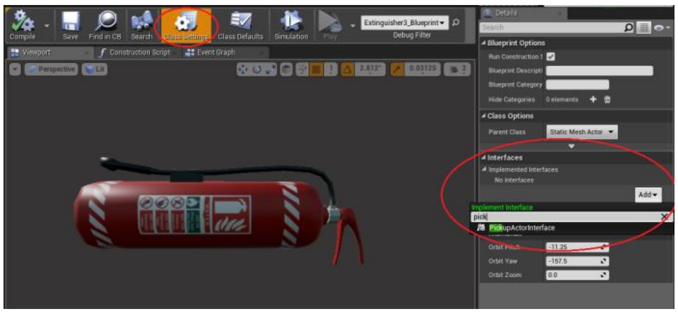
Figure 17. Object selected in the World Outliner

Secondly, a Blueprint is created for this StaticMeshActor by clicking Blueprint button in Details tab (Figure 18). After the creation, this Blueprint will be opened, and then in order to add the PickUpActorInterface to it, the Class Settings must be chosen first (Figure 19).



Location 1684.0 -5935. 249.0
Rotation 89. 0.0 90.
Scale 10.0 10.0 10.0
Mobility St Sta Mo

Figure 18. Blueprint Creation



Run Construction S ✓
Blueprint Descripti
Blueprint Category
Hide Categories 0 elements

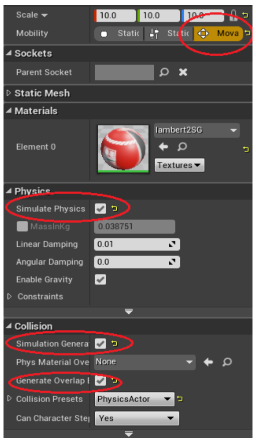
Parent Class Static Mesh Actor

pick
PickupActorInterface

Orbit Pitch -11.25
Orbit Yaw -157.5
Orbit Zoom 0.0

Figure 19. Interface Implementation

Next, the StaticMeshComponent in Component tab needs to be chosen in order to change its properties. In the details tab, Mobility needs to be changed into Movable, Simulate Physics in Physics field, Simulation Generates Hit Events and Generate Overlap Events in Collision field must be checked as in figure 20.


Scale 10.0 10.0 10.0
Mobility Static Static Mova

Parent Socket

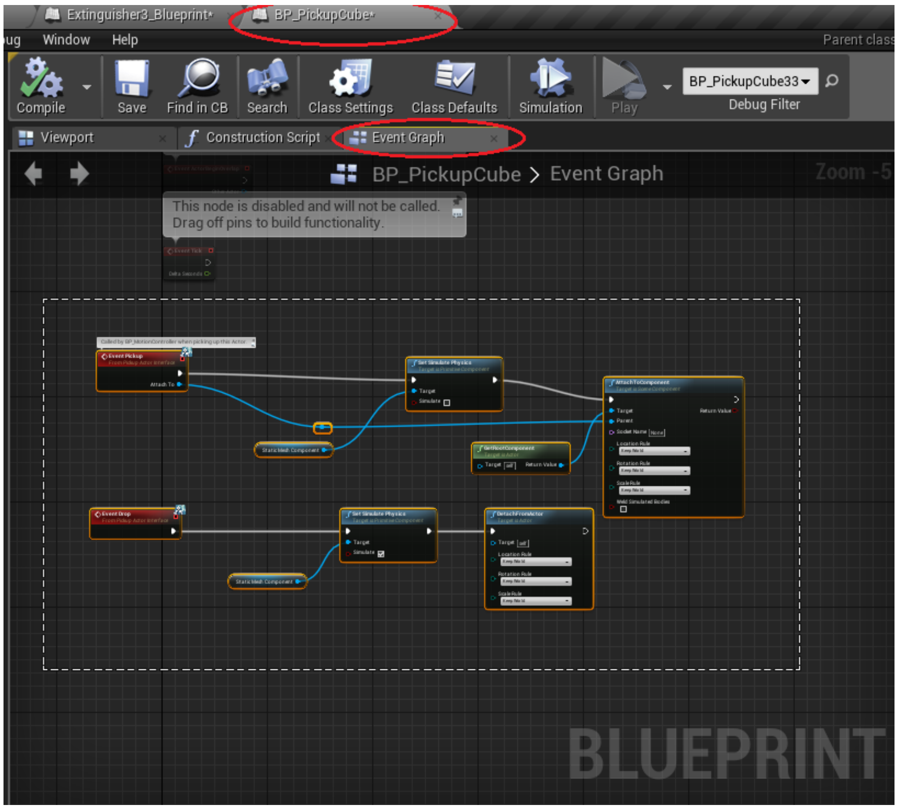
Element 0 lambert2SG
Textures

Simulate Physics ✓
MassInKg 0.038751
Linear Damping 0.01
Angular Damping 0.0
Enable Gravity ✓
Constraints

Simulation Genera ✓
Phys Material Ove None
Generate Overlap I ✓
Collision Presets PhysicsActor
Can Character Ste Yes

Figure 20. StaticMeshComponent Settings

Finally, this Blueprint needs to implement Pickup event and Drop event so that the object can be interacted with. This can be done by duplicating the script of the PickupCube existing in the VR template. The script for events can be found in Blueprint Class BP_PickupCube at Event Graph tab. After copying this script to the Blueprint of the targeted object, the controller can interact with it.



This node is disabled and will not be called.
Drag off pins to build functionality.

Figure 21. Pickup and Drop events implementation

5.2 Default Map

5.2.1 Exhibition on screen

This section of the document is used to demonstrate the implementation of one example display screen in the default scene. Figure 22 demonstrates what it looks like in gameplay.

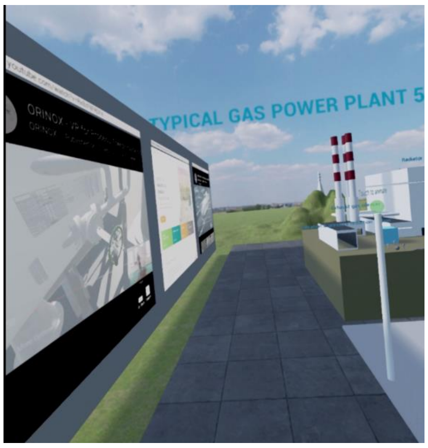
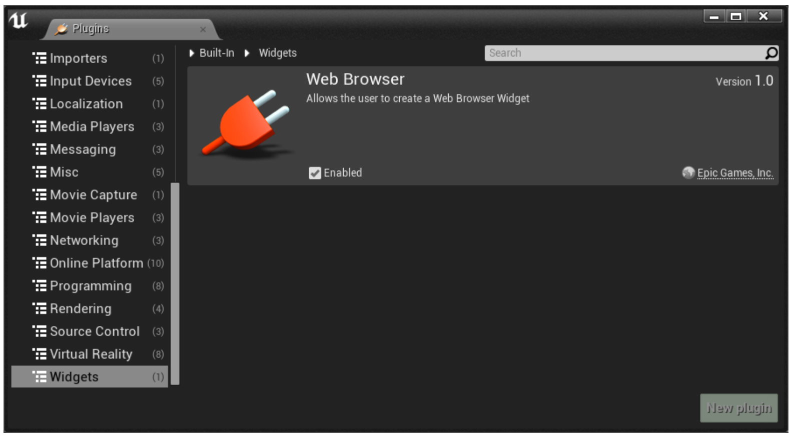


Figure 22. Showcase Screens gameplay

A display is created by using a Blueprint class and a Widget Blueprint. In order to use the experimental Web Browser Widget, this plugin must be enabled in the project by clicking Edit/Plugins (Figure 23).



Web Browser
Allows the user to create a Web Browser Widget
Version 1.0
☑ Enabled
Epic Games, Inc.

Figure 23. Enable Web Browser Widget plugin

The Widget Blueprint is created with a Web Browser – the experimental component enabled before and a Text Box. The Web Browser component displays the contents of the webpage which the Initial URL leads to. (Figure 24)

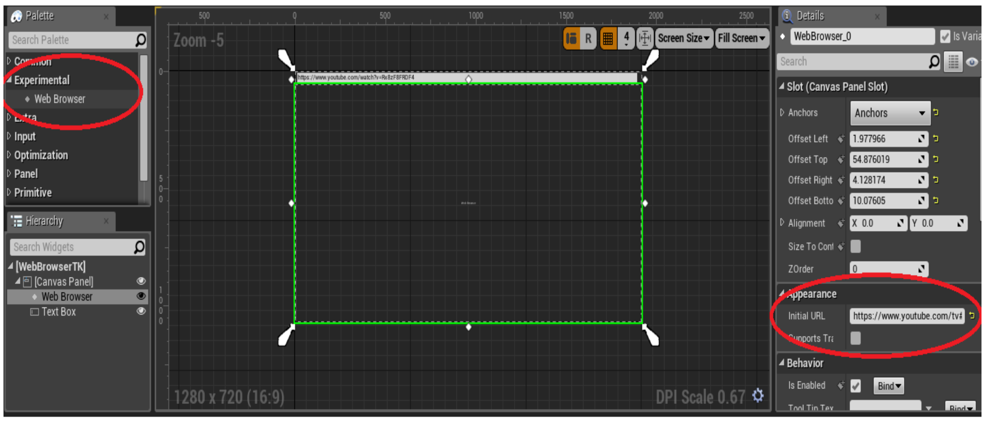


Figure 24. Widget Blueprint structure

For the screen in the scene, a Blueprint class must be made. The most important component of this class is the Widget Component. The Widget Class of this component must be chosen as the Widget Blueprint mentioned above. Figure 25 illustrates how the Blueprint class was implemented.

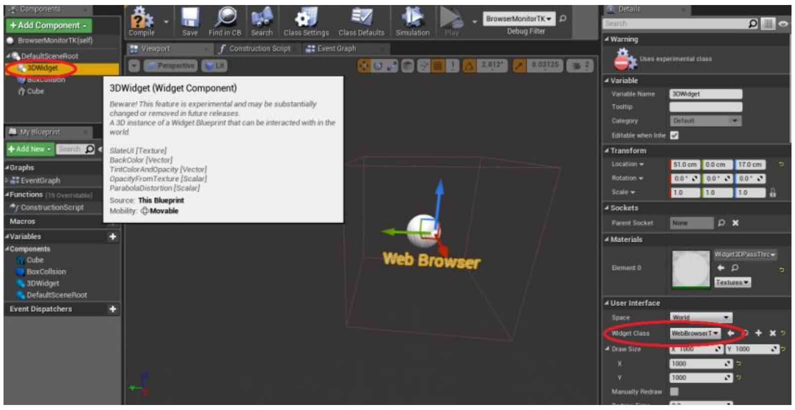


Figure 25. Blueprint class structure

5.2.2 Scene Changing Doors

The default scene provides two scene changing doors to allow the user to enter other scenes. The powerhouse door is for the engine room scene and the other door is for the control room scene (Figure 26).

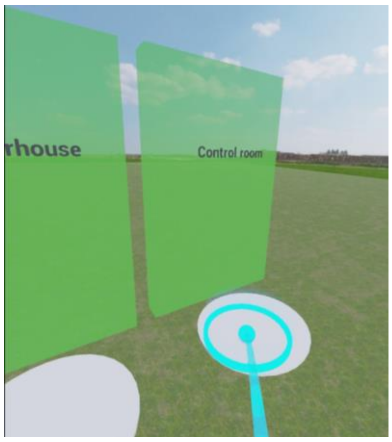


Figure 26. Two scene changing doors gameplay

Each of these doors was implemented so that the user can enter another scene by pressing the trigger buttons of the controllers while touching the corresponding door (Figure 27).

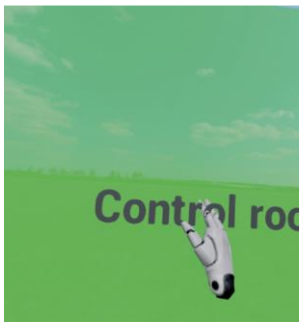
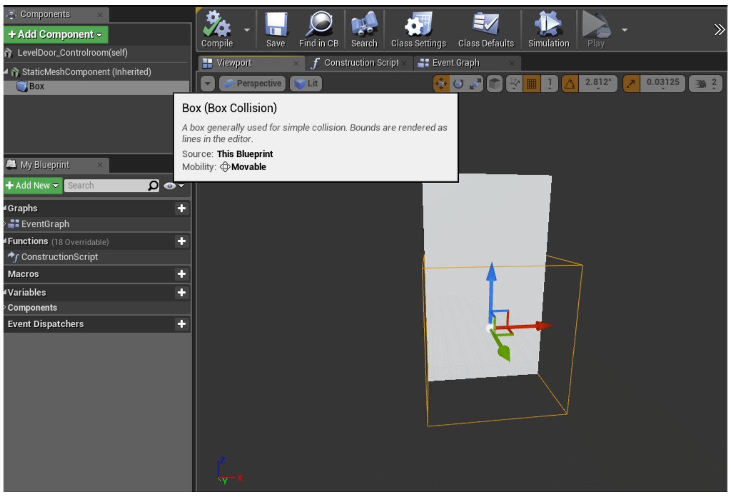


Figure 27. Door usage

The Door Blueprint Class consists of a StaticMesh for the door physical appearance and a Box Collision. (Figure 28) The Box Collision is used to let the program detect when the user comes near the door.



Box (Box Collision)

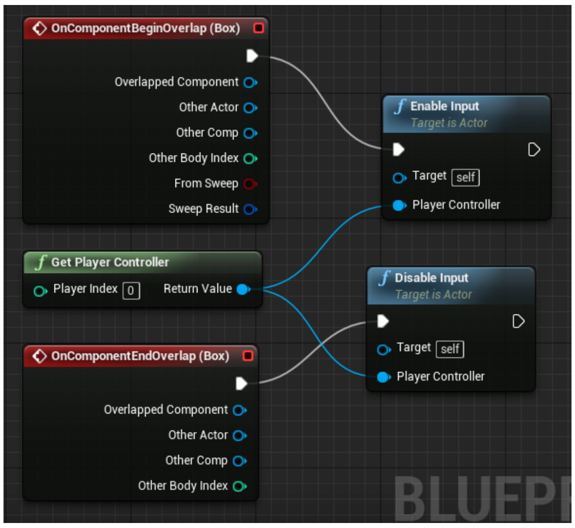
A box generally used for simple collision. Bounds are rendered as lines in the editor.

Source: **This Blueprint**

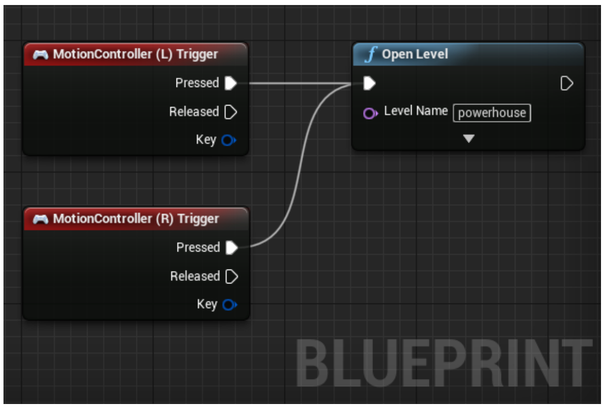
Mobility: **Movable**

Figure 28. Door Blueprint Class components

For the scripting of this Blueprint Class, the controller needs to be enable input whenever the user is overlapping the collision Box (Code Snippet 2) and the trigger button needs to be implemented to open the new scene.



Code Snippet 2. Input enable

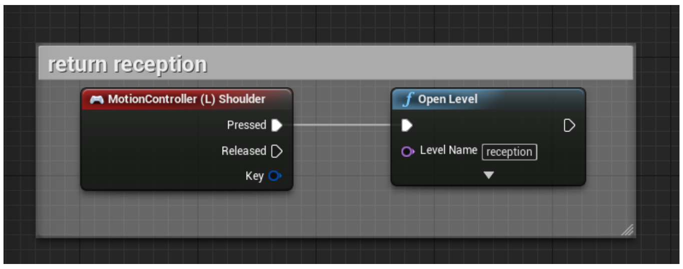


Code Snippet 3. Trigger Button implementation

5.3 Control Room Map

5.3.1 Return Default Scene

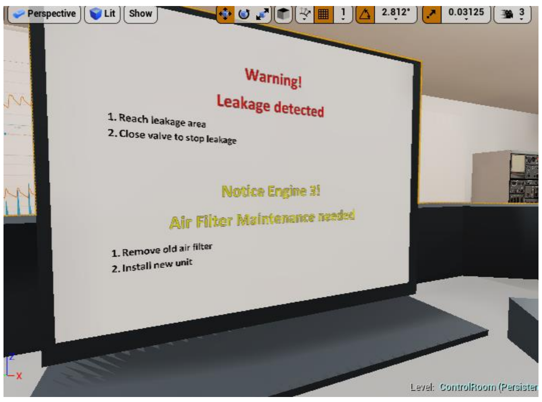
The user can return to the default scene by pressing the menu button of the left controller. This is done by defining a simple script in the Level Blueprint (Code Snippet 4). This piece of script is also included in the Engine room Scene.



Code Snippet 4. Return default scene script

5.3.2 Warning Screens

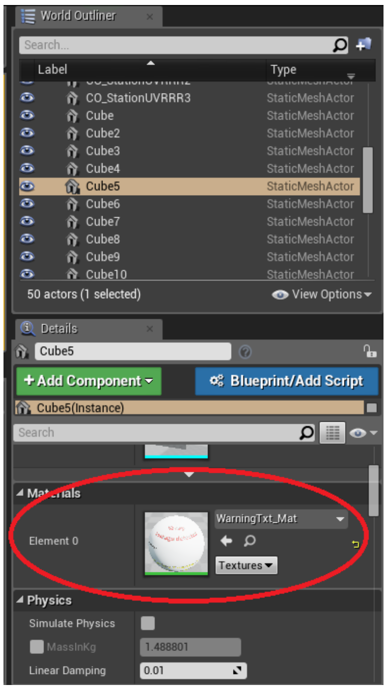
There are multiple warning screens in the control room which notify the user on what needs to be done. An example is shown in Figure 29.



Warning!
Leakage detected
1. Reach leakage area
2. Close valve to stop leakage
Notice Engine #!
Air Filter Maintenance needed
1. Remove old air filter
2. Install new unit

Figure 29. Warning Screen gameplay

These warning screens are created by changing the material of a basic Cube object into WarningTxt_Mat. This is shown in figure 30.



Label	Type
CO_StationUVRRR3	StaticMeshActor
Cube	StaticMeshActor
Cube2	StaticMeshActor
Cube3	StaticMeshActor
Cube4	StaticMeshActor
Cube5	StaticMeshActor
Cube6	StaticMeshActor
Cube7	StaticMeshActor
Cube8	StaticMeshActor
Cube9	StaticMeshActor
Cube10	StaticMeshActor

50 actors (1 selected)

Cube5
Cube5(Instance)

Materials
Element 0: WarningTxt_Mat

Physics
Simulate Physics
MassInKg: 1.488801
Linear Damping: 0.01

Figure 30. Warning Screen settings

The custom material (WarningTxt_Mat) is generated by the texture of a jpeg file.

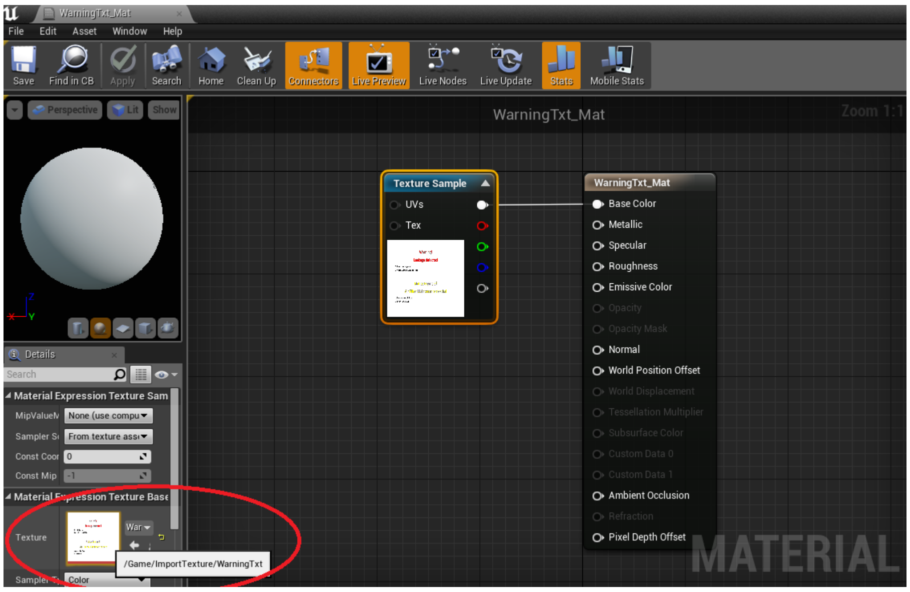


Figure 31. WarningTxt_Mat material creation

5.3.3 IoT Ticket Screen

The IoT Ticket Screen serves as a touch screen which enables the user to interact with this system. The right controller emits a laser light to notify the interaction point (Figure 32). The trigger button can be used to click, drag or drop as the mouse pointer works on any browser.

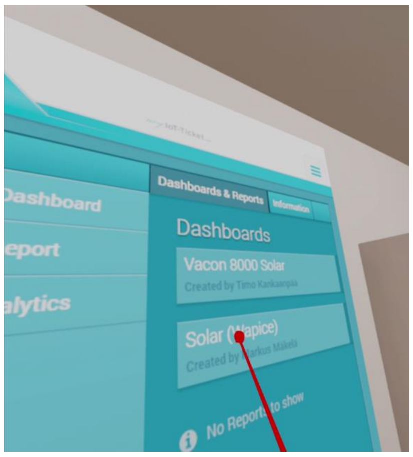
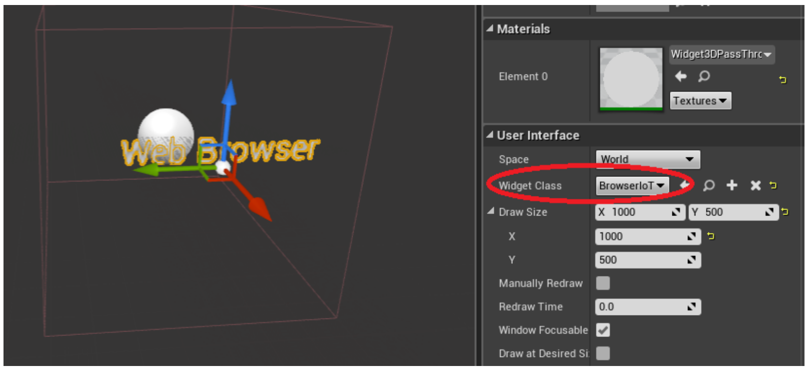


Figure 32. IoT screen gameplay

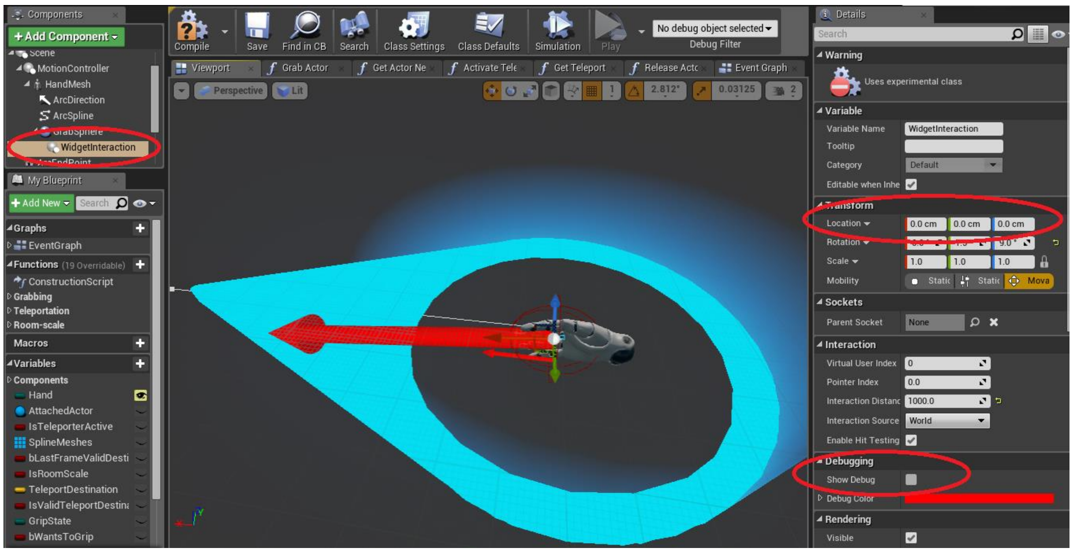
This screen was implemented the same way as the exhibition screens in the default map. In other words, the Blueprint Class **BrowserMonitorIoT** was created with the **BrowserIoT** Widget blueprint as its Widget Class (Figure 33).



Space: World
Widget Class: BrowserIoT
Draw Size: X 1000, Y 500
X: 1000
Y: 500
Manually Redraw: ☐
Redraw Time: 0.0
Window Focusable: ✓
Draw at Desired Si: ☐

Figure 33. BrowserMonitorIoT Class Configure

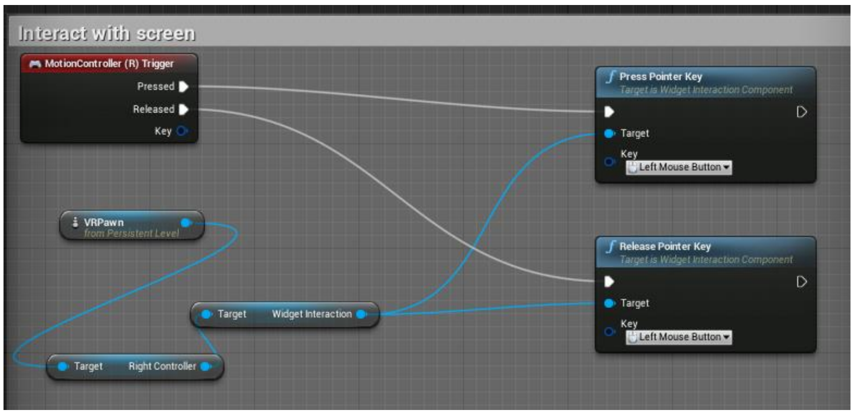
However, this screen only shows the content without any user interaction. To enable the user interaction, firstly, the widget interaction component needs to be added to the Blueprint class BP_MotionController found in Content/VirtualRealityBP/Blueprints folder. This WidgetInteraction component has to be attached to the Sphere then zero out the Location. The Show Debug option in the Details panel will make this component emit a line with a sphere at the end of the line to help interaction. (Figure 34) The Debug line will be toggled on/off by script.



Variable Name: WidgetInteraction
Tooltip:
Category: Default
Editable when Inhe: ✓
Location: 0.0 cm, 0.0 cm, 0.0 cm
Rotation: [illegible], [illegible], 9.0°
Scale: 1.0, 1.0, 1.0
Mobility: Static, Static, Mova
Parent Socket: None
Virtual User Index: 0
Pointer Index: 0.0
Interaction Distanc: 1000.0
Interaction Source: World
Enable Hit Testing: ✓
Show Debug: ☐
Debug Color: 🟥
Visible: ✓

Figure 34. WidgetInteraction Component configuration

Next, the script for the screen interaction of the motion controller needs to be implemented in the Level Blueprint. The script below will enable the trigger button of the controller to imitate how the left mouse button works. This will allow the user to click, drag and drop in the IoT screen.



Code Snippet 5. Trigger button implementation for screen interaction

5.3.4 Virtual Keyboard

The IoT ticket system requires authentication for the first time user so a virtual keyboard is needed for the user to set his/her username and password.

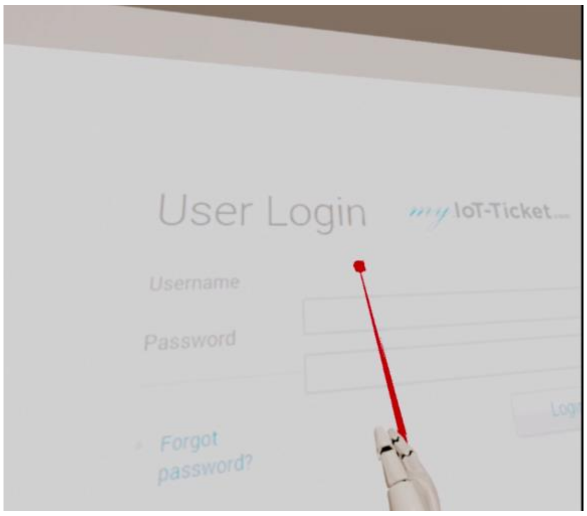


Figure 35. First time user authentication

The virtual keyboard is created with the Blueprint Class **KeyboardBP** and the WidgetBlueprint **VirtualKeyboard**. The WidgetBlueprint is used to design the interface and function of each button in the virtual keyboard. This keyboard can be created with the same layout as in reality, however, this will take a lot of time because each button of the keyboard needs to be implemented. For this reason, the virtual keyboard in this application only contains three buttons as in Figure 36.

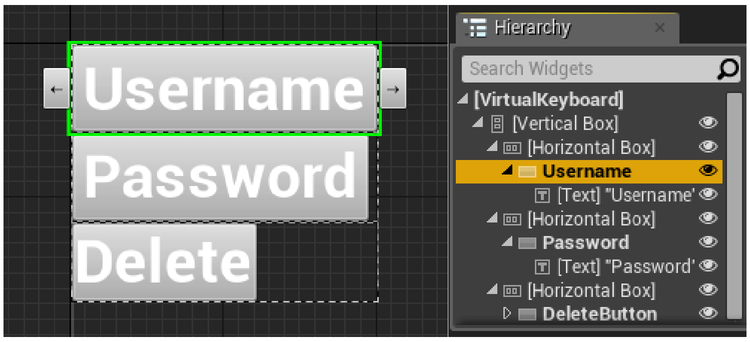


Figure 36. Virtual Keyboard structure

Each of these buttons needs to be set in “Is Focusable” option to False (Figure 37). This action is to ensure that no other widgets the user interacts with will take the focus away from the IoT Ticket Screen widget. If any of these buttons has this set to true, when this button is clicked, the focus will be taken away from the IoT Screen and shifted to that button preventing the program from passing the key char to the IoT Screen.

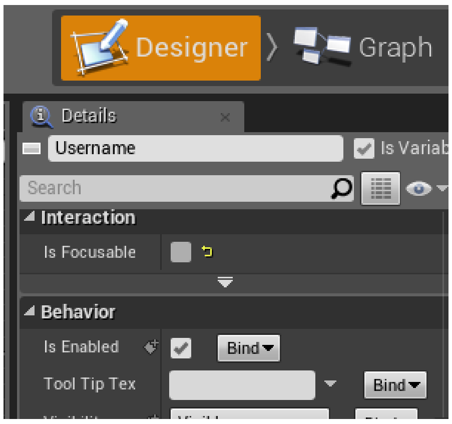
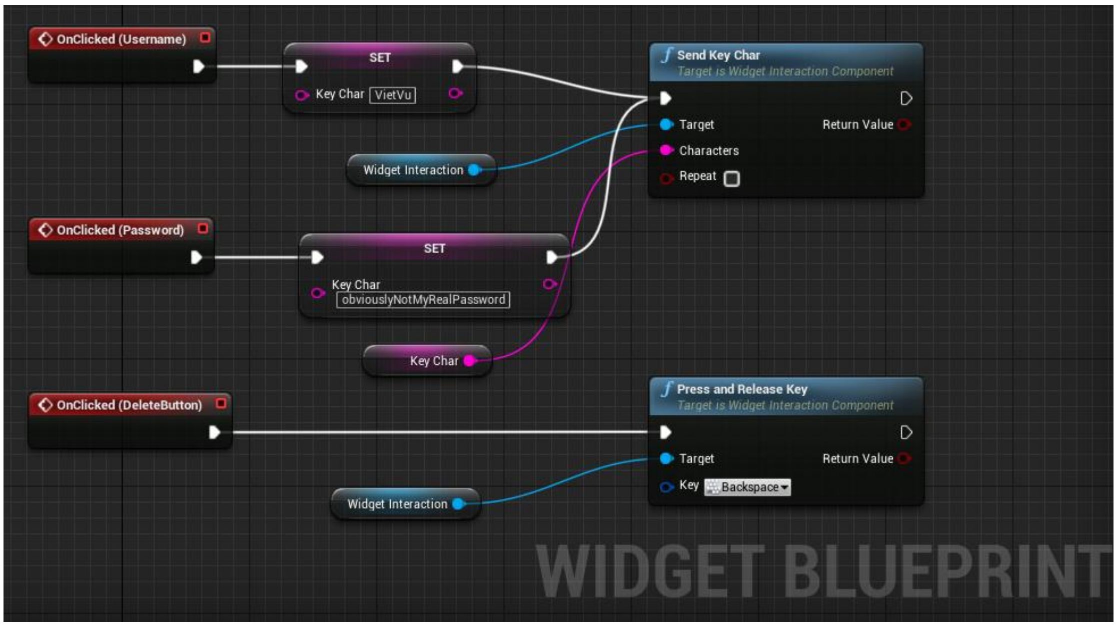


Figure 37. Button focusable configuration

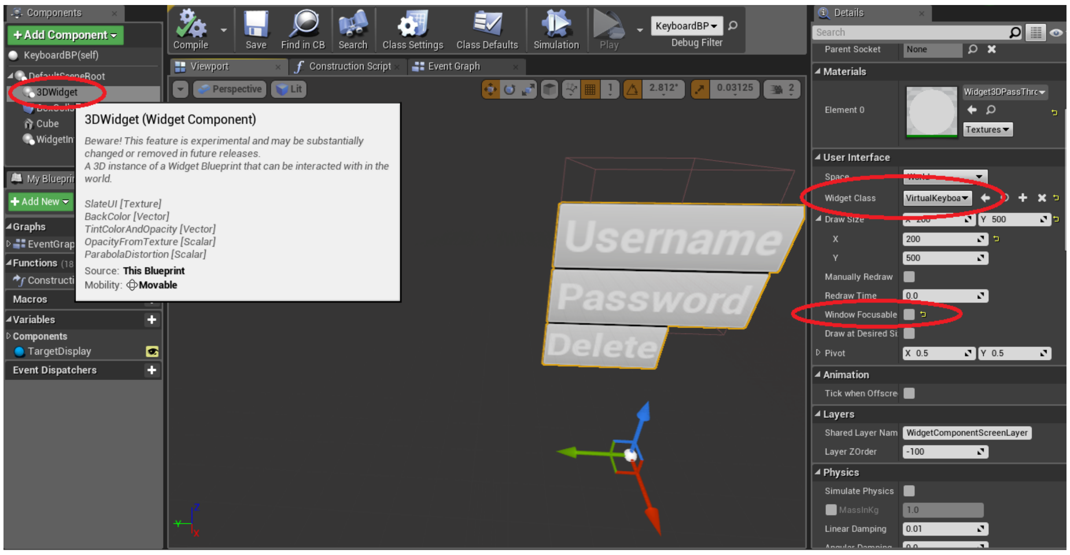
Next, the events OnClick for each of these buttons needs to be implemented (CodeSnippet 6). Two variables, a String type called KeyChar and a WidgetInteractionComponent called widgetInteraction, are also created. Each of the OnClick event will set the KeyChar value and then send this string to Widget Interaction.

In this case, the string will be sent to the IoT screen to fill in the username and password. The DeleteButton functions as the Backspace key of the real keyboard.



Code Snippet 6. Virtual Keyboard Buttons Implementation

After the implementation of the Widget Blueprint, the Blueprint class can add it as Widget Component. This can be done by choosing that Widget Blueprint in Widget Class. The Window Focusable option also needs to be unchecked so that this component will not take focus away from the IoT screen. All of these actions are demonstrated in Figure 38. A WidgetInteraction component was also added to this Blueprint so that a reference to the WidgetInteraction component in the player character do not need to be given.



3DWidget (Widget Component)

Beware! This feature is experimental and may be substantially changed or removed in future releases.
A 3D instance of a Widget Blueprint that can be interacted with in the world.

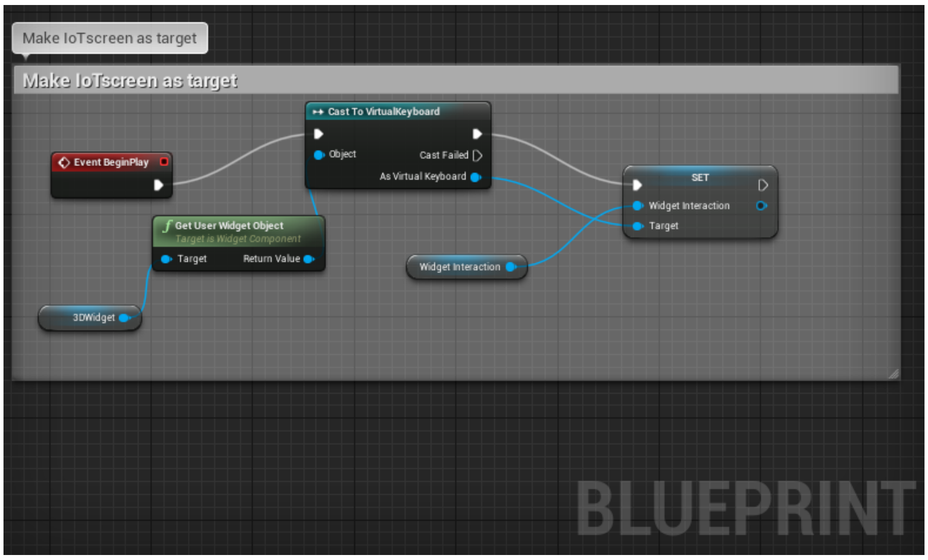
SlateUI [Texture]
BackColor [Vector]
TintColorAndOpacity [Vector]
OpacityFromTexture [Scalar]
ParabolaDistortion [Scalar]

Source: **This Blueprint**
Mobility: **Movable**

Parent Socket	None
Materials	
Element 0	Widget3DPassThro
User Interface	
Space	World
Widget Class	VirtualKeyboa
Draw Size	X 200 Y 500
X	200
Y	500
Manually Redraw	☐
Redraw Time	0.0
Window Focusable	☐
Draw at Desired Si	☐
Pivot	X 0.5 Y 0.5
Animation	
Tick when Offscre	☐
Layers	
Shared Layer Nam	WidgetComponentScreenLayer
Layer ZOrder	-100
Physics	
Simulate Physics	☐
MassInKg	1.0
Linear Damping	0.01

Figure 38. Widget Component configuration

Code Snippet 7 illustrates how the Blueprint will set the IoT screen as the target to send the strings of username or password to.



Code Snippet 7. Targeting IoT Screen implementation

5.4 Engine Room Map

5.4.1 Steam Effect

In order to make the scene more realistic, the steam effect is added by using the Particle System Component. This effect appears at the exhale valve of the engine, the power plant silos and the container area. Figure 39 demonstrates what the steam effect looks like in gameplay. Figure 40 illustrates the interface to modify the Particle System Component.

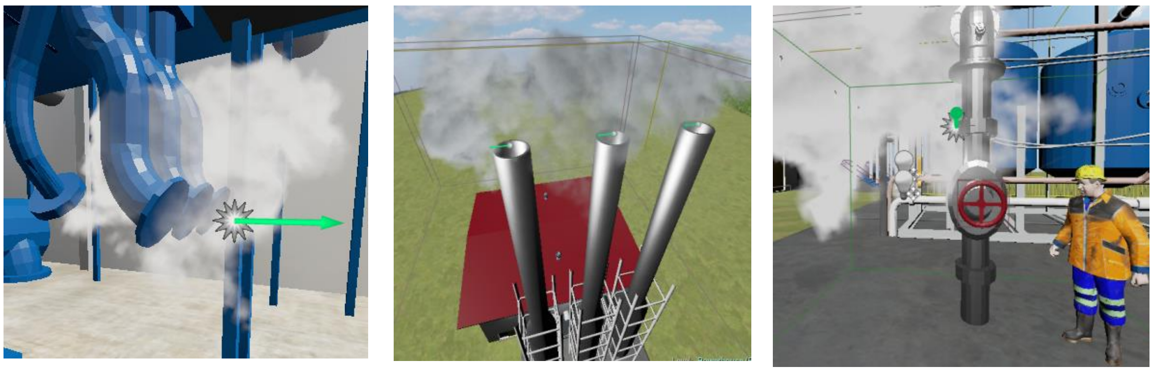
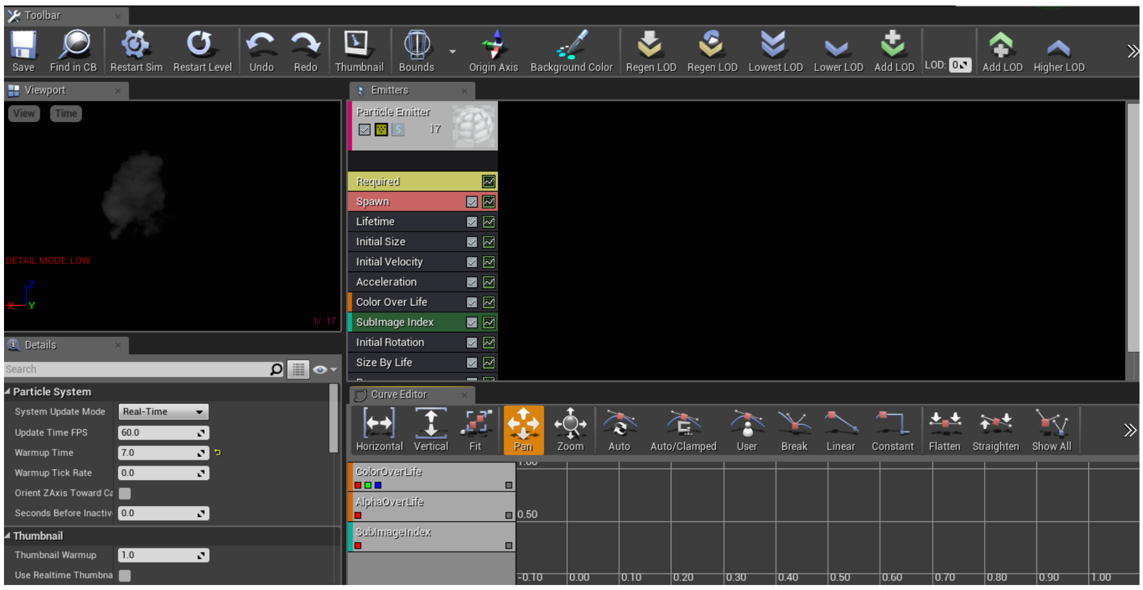


Figure 39. Steam effect gameplay



Details

Particle System

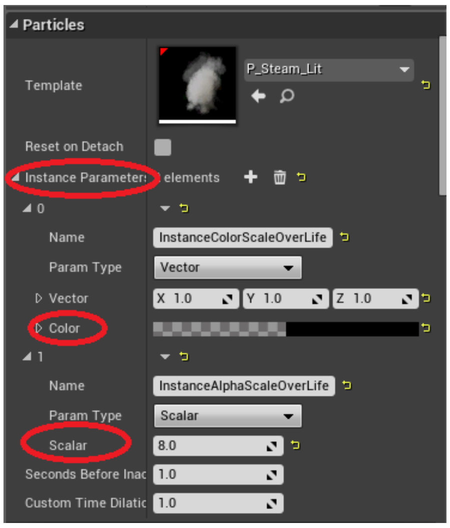
System Update Mode	Real-Time
Update Time FPS	60.0
Warmup Time	7.0
Warmup Tick Rate	0.0
Orient ZAxis Toward Ca	☐
Seconds Before Inactiv	0.0

Thumbnail

Thumbnail Warmup	1.0
Use Realtime Thumbna	☐

Figure 40. Particle System Modification Interface

An instance of the particle system can be modified by changing value in the Instance Parameters field of Properties tab. The parameter InstanceColorScaleOverLife is used to change the color of the steam effect and the InstanceAlphaScaleOverLife is used to change its opacity. These modifications are illustrated in Figure 41.



```
Particles
Template                P_Steam_Lit
Reset on Detach
Instance Parameters     elements
  0
    Name                InstanceColorScaleOverLife
    Param Type          Vector
    Vector              X 1.0   Y 1.0   Z 1.0
    Color
  1
    Name                InstanceAlphaScaleOverLife
    Param Type          Scalar
    Scalar              8.0
Seconds Before Inac     1.0
Custom Time Dilatic     1.0
```

Figure 41. Steam effect properties configuration

5.4.2 Blinking exclamation mark

There are two blinking exclamation marks along with the worker model in the scene to get the user attention for the maintenance work and the emergency case as in Figure 42. These blinking marks will disappear if the user comes near to the place because their intention has been achieved.

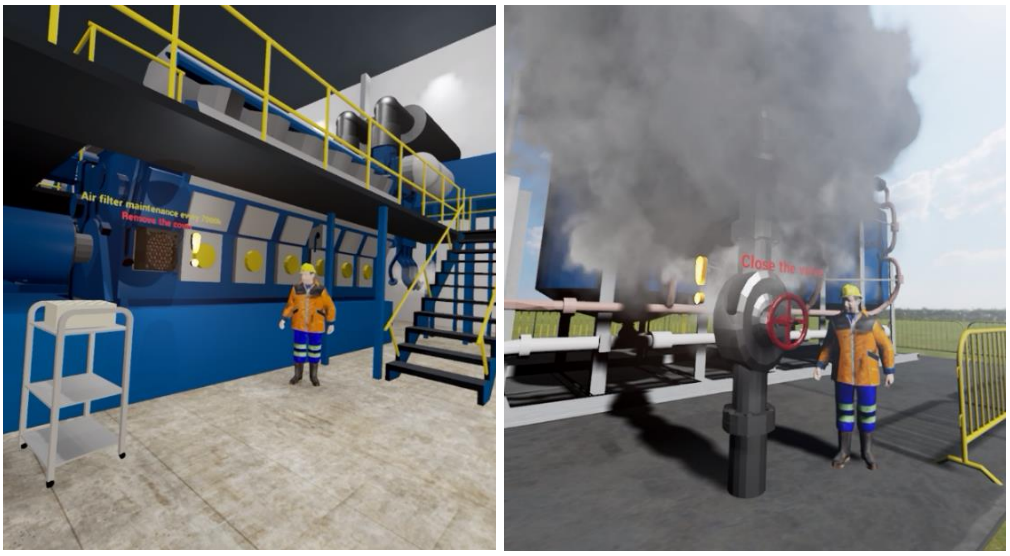
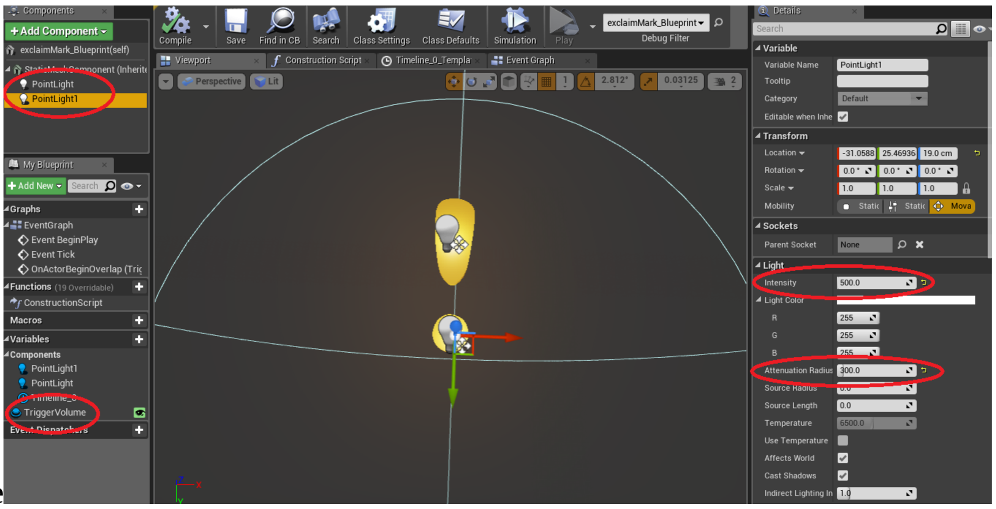


Figure 42. Blinking mark and worker model for user attention

The blinking marks were made up of two point light components. Each of the point light components is for one part of the static mesh. A trigger volume is created to detect if the user comes near the actor. If the user overlaps this volume, the blink mark will be destroyed. The intensity value indicates how bright the point light is and the attenuation radius value demonstrates how broadly the light affect. All the settings are illustrated in Figure 43.

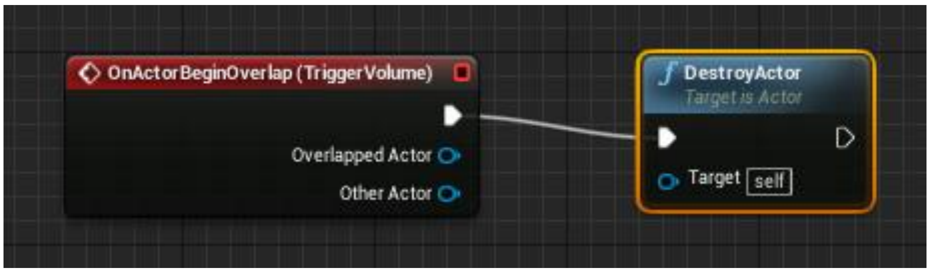
e



Variable	
Variable Name	PointLight1
Tooltip	
Category	Default
Editable when Inhe	✓
Transform	
Location	-31.0588 / 25.46936 / 19.0 cm
Rotation	0.0° / 0.0° / 0.0°
Scale	1.0 / 1.0 / 1.0
Mobility	Static / Static / Mova
Sockets	
Parent Socket	None
Light	
Intensity	500.0
Light Color	
R	255
G	255
B	255
Attenuation Radius	300.0
Source Radius	0.0
Source Length	0.0
Temperature	6500.0
Use Temperature	
Affects World	✓
Cast Shadows	✓
Indirect Lighting In	1.0

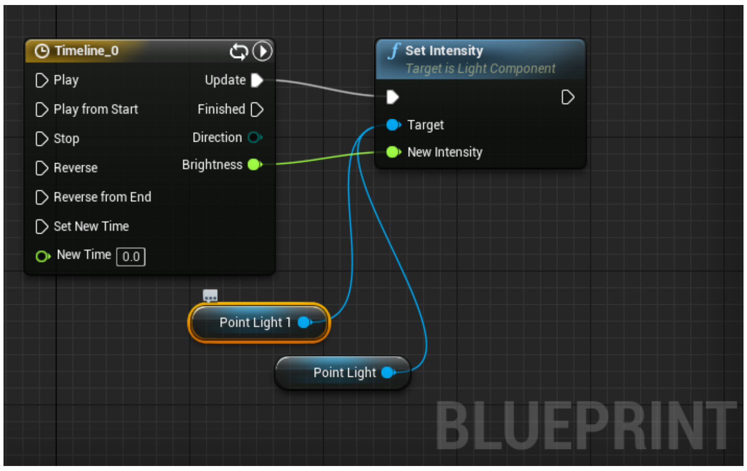
Figure 43. Blinking mark configuration

Code Snippet 8 is used to make the blinking mark disappear when the user comes near to it.



Code Snippet 8. Blinking mark disappear implementation

For now, the mark has the same intensity all the time. With Code Snippet 9, the mark will be able to blink by updating the intensity every second. The Timeline used in the script is illustrated in Figure 44. The timeline defines values for every moment. These values will be set as the intensity to make the mark blink.



Code Snippet 9. Blinking function implementation

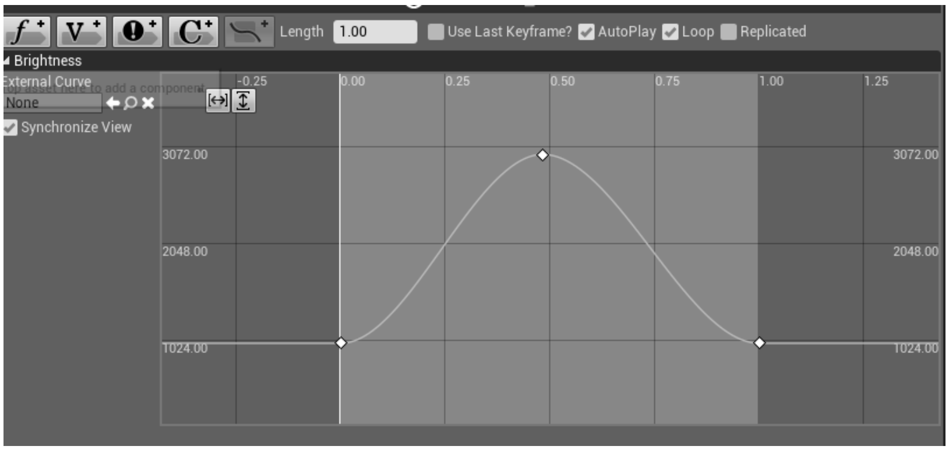


Figure 44. Timeline_0 creation

5.4.3 Door interaction

There are two doors in the scene which let the user use the motion controllers to open them. Figure 45 describes the door opening procedure for going outside of the engine building. The door is closed by default when the user enters the scene. An instruction will be displayed when the user comes near to the door. By using the trigger buttons of the controllers, the user can make the door open in order to go outside.

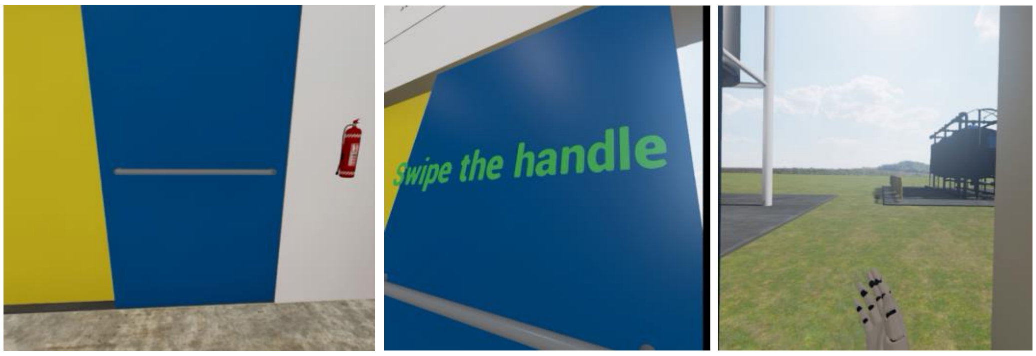


Figure 45. Opening door procedure gameplay

The door is created by multiple meshes, a Text Renderer and a Trigger Volume. The Text Renderer is used to display the instruction text. The Trigger Volume is used to detect if the user comes near to the door. Two variables were also added, a Vector type for the initial location and a Boolean type to determine if the door is opened or closed. All these components are illustrated in Figure 46.

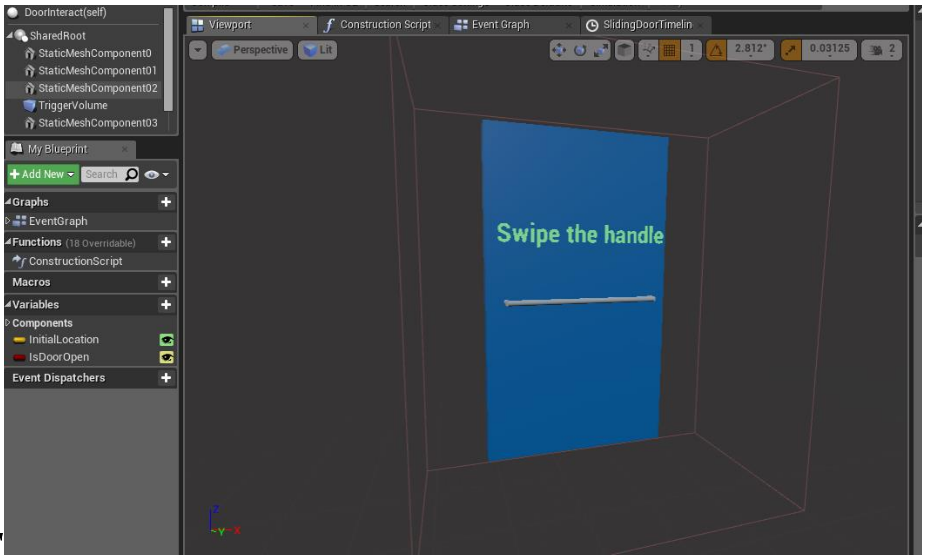
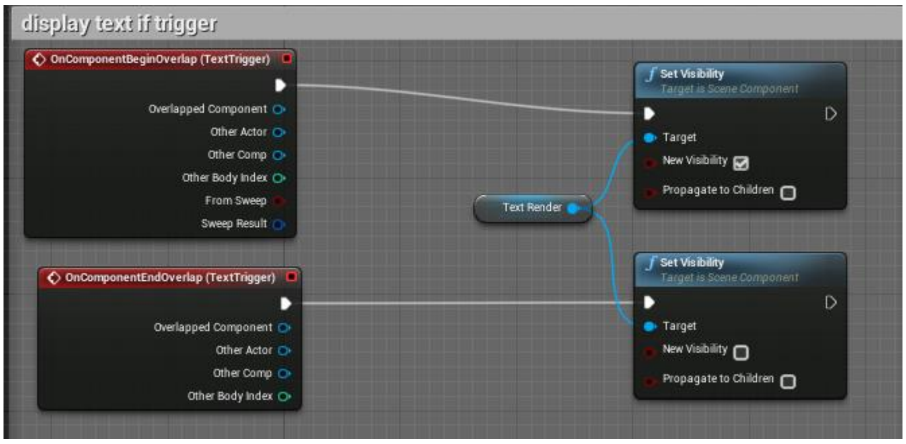


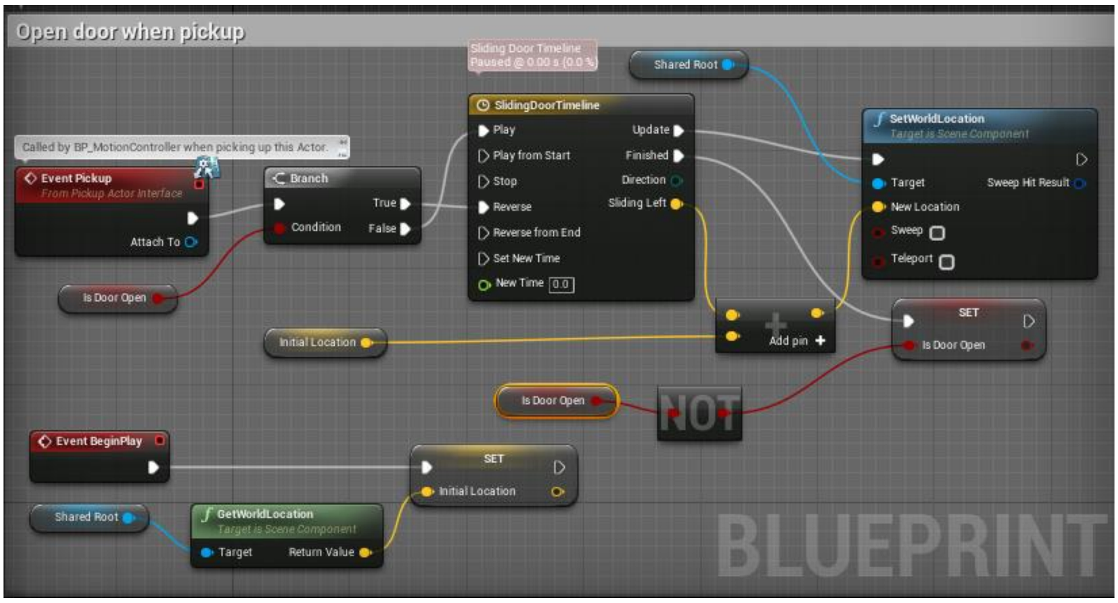
Figure 46. Door structure

Code Snippet 10 is used to toggle on/off the instruction text appearance. If the user comes near to the door the text visibility will be set to true so the user can see it. If the user goes outside of the trigger volume, this visibility value will be set to false, therefore, the text will disappear.



Code Snippet 10. Instruction text toggling implementation

Code Snippet 11 is added for the interaction of the door. As soon as the user enters the scene, the location of the door will be saved to the variable Initial Location. When the user uses the controller to interact with the door, the script will consider the variable IsDoorOpen to decide if the user is trying to open or close the door. If the door needs to be opened, the script will run the SlidingDoorTimeline to move the door to the left. If the user wants to close the door, the script will run the SlidingDoorTimeline in reverse which makes the door moves to the right.



Code Snippet 11. Door interaction implementation

The SlidingDoorTimeline for the time-based door opening animation is described in Figure 47. Because the door is only sliding to the left or right (open/close), the only value which changes occasionally is x-value of the xyz coordinates.

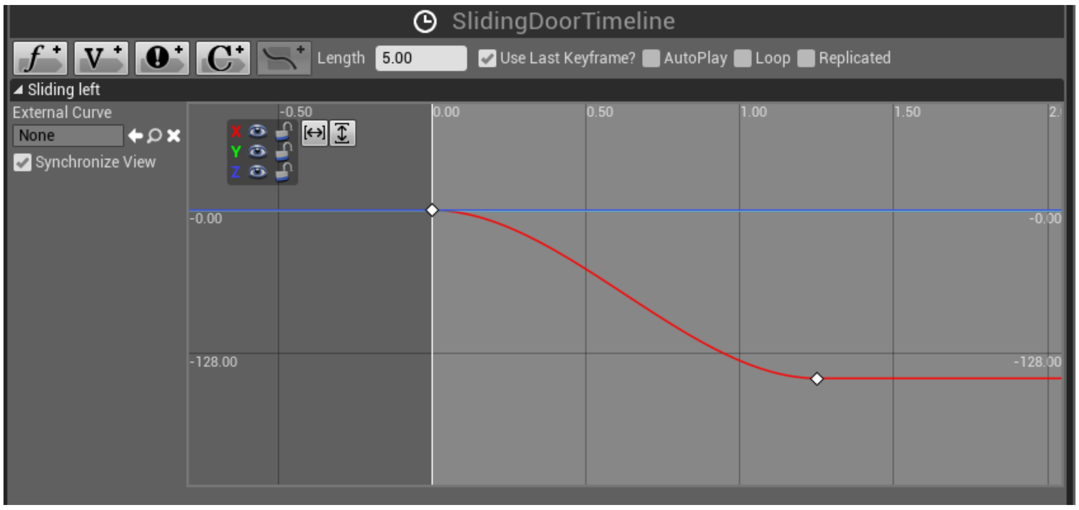


Figure 47. SlidingDoorTimeline

5.4.4 Mini-map

The engine room scene has a feature called a mini-map, which enables the user to track his/her real-time character location. The mini-map shows the top view of the scene and mark user location with a green blinking dot. Figure 48 describes what the mini-map looks like when running the application.

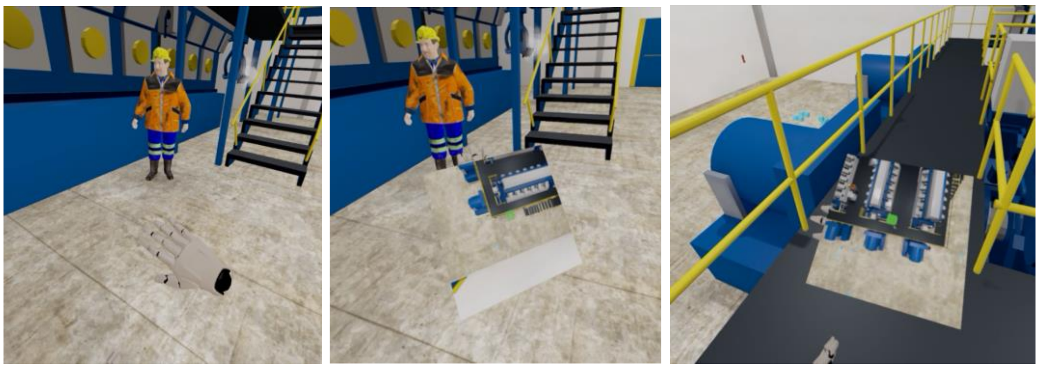


Figure 48. Mini-map gameplay

This mini-map was created by defining five objects: a Blueprint Class named Minicam which has the Parent Class SceneCapture2D, a Render Target named

MapRendering to render the texture, a Widget Blueprint named Minimap Widget, a Material named Minimap_Mat and a Blueprint Class named Minimap.

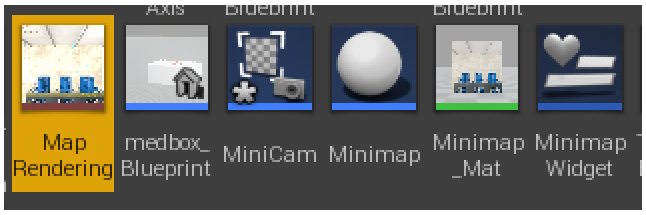
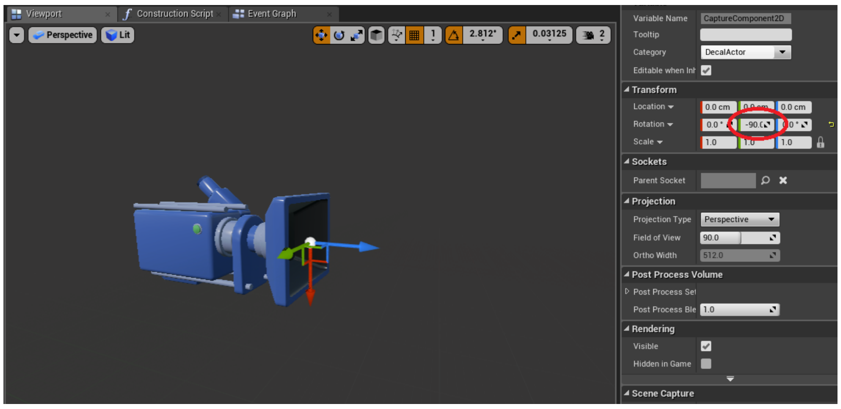


Figure 49. Mini-map required objects

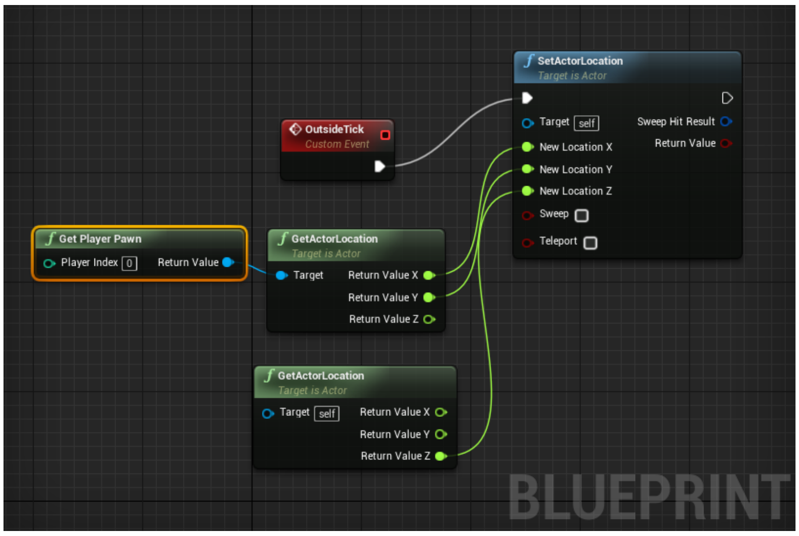
The MiniCam is actually another camera with a rotation y-value -90 degree to capture the view of the scene in top view.



Variable Name	CaptureComponent2D		
Tooltip			
Category	DecalActor		
Editable when Inh	✓		
Transform			
Location	0.0 cm	0.0 cm	0.0 cm
Rotation	0.0 °	-90.0	0.0 °
Scale	1.0	1.0	1.0
Sockets			
Parent Socket			
Projection			
Projection Type	Perspective		
Field of View	90.0		
Ortho Width	512.0		
Post Process Volume			
Post Process Set			
Post Process Ble	1.0		
Rendering			
Visible	✓		
Hidden in Game			
Scene Capture			

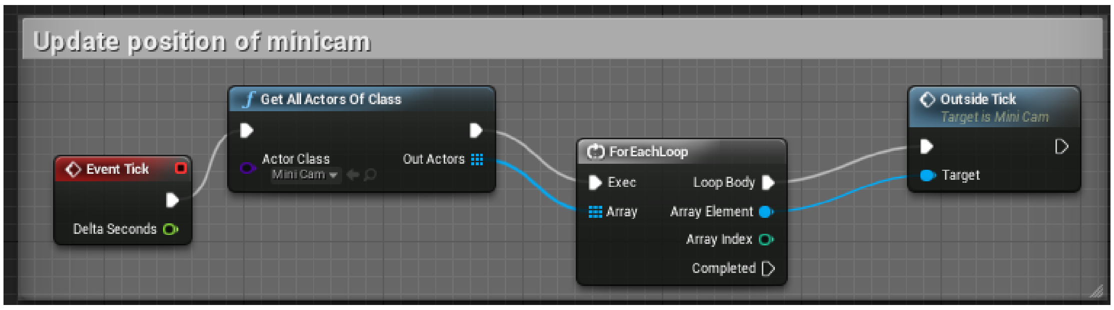
Figure 50. MiniCam rotation settings

Code Snippet 12 was used to create a custom event for the MiniCam camera to follow the user. The camera will get only the Value X and Value Y of user location. It always has the same height so Value Z will be taken from itself.



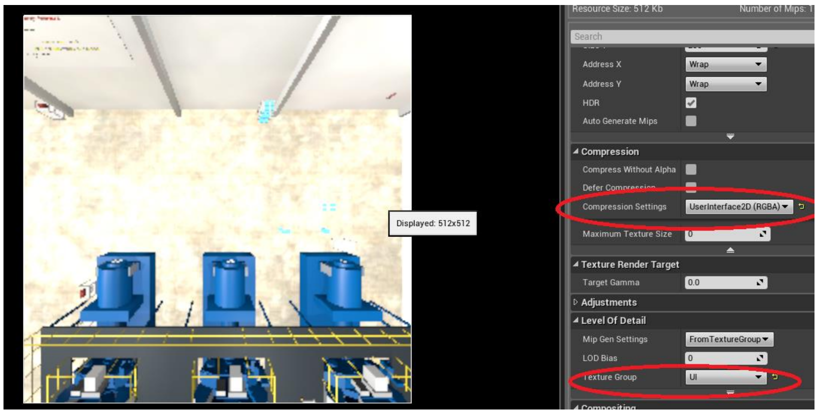
Code Snippet 12. MiniCam follows user location

Code Snippet 13 also needs to be added in the Level Blueprint. For every frame, the event of the MiniCam will be called.



Code Snippet 13. Outside tick Event called

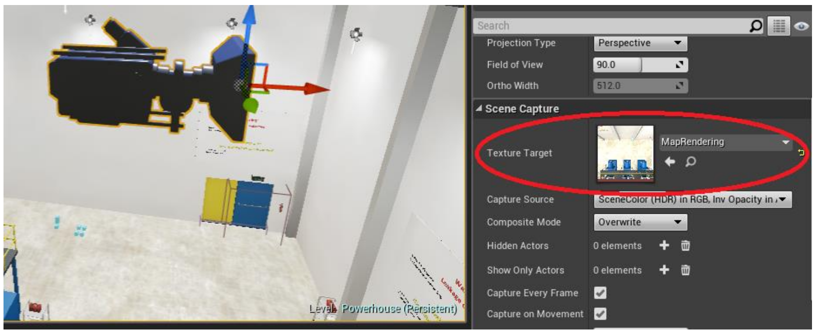
The MapRendering needs to change the Compression Settings to UserInterface2D and TextureGroup to UI (Figure 51). This MapRendering will be used as Render Target of the MiniCam (Figure 52).



Resource Size: 512 Kb   Number of Mips: 1

Address X	Wrap
Address Y	Wrap
HDR	✓
Auto Generate Mips	
Compression	
Compress Without Alpha	
Defer Compression	
Compression Settings	UserInterface2D (RGBA)
Maximum Texture Size	0
Texture Render Target	
Target Gamma	0.0
Adjustments	
Level Of Detail	
Mip Gen Settings	FromTextureGroup
LOD Bias	0
Texture Group	UI

Figure 51. MapRendering Settings



Projection Type	Perspective
Field of View	90.0
Ortho Width	512.0
Scene Capture	
Texture Target	MapRendering
Capture Source	SceneColor (HDR) in RGB, Inv Opacity in A
Composite Mode	Overwrite
Hidden Actors	0 elements
Show Only Actors	0 elements
Capture Every Frame	✓
Capture on Movement	✓

Figure 52. MiniCam's Texture Target

The material Minimap_Mat was created by the MapRendering (Figure 53).

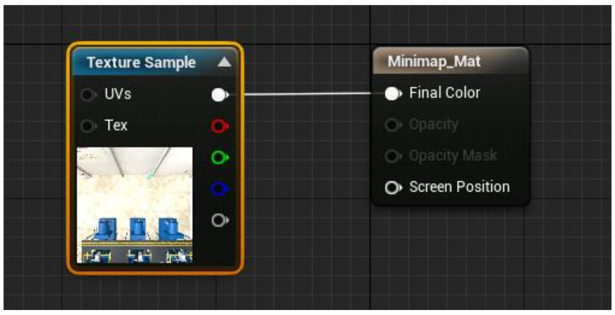


Figure 53. Minimap_Mat material

The Widget Blueprint called MinimapWidget has two components: a Throbber component to illustrate the user's current location and an Image component to display the scene capture. The Image component uses Minimap_Mat material mentioned earlier. These settings are described in Figure 54.

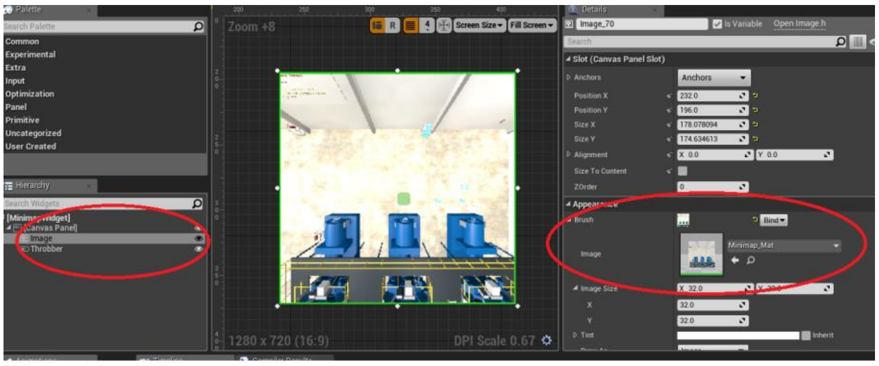
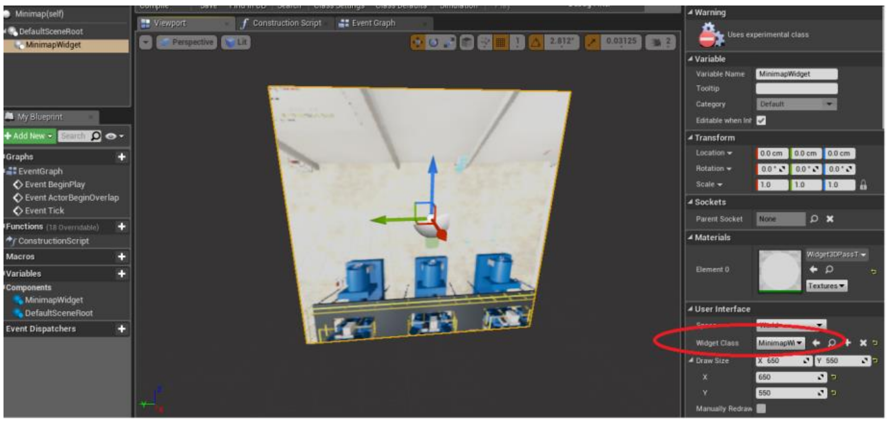


Image_70
Anchors: Anchors
Position X: 232.0
Position Y: 196.0
Size X: 178.078094
Size Y: 174.634613
Alignment: X 0.0 Y 0.0
ZOrder: 0
Image: Minimap_Mat
Image Size: X 32.0 Y 32.0
X: 32.0
Y: 32.0

Figure 54. Minimap Widget configuration

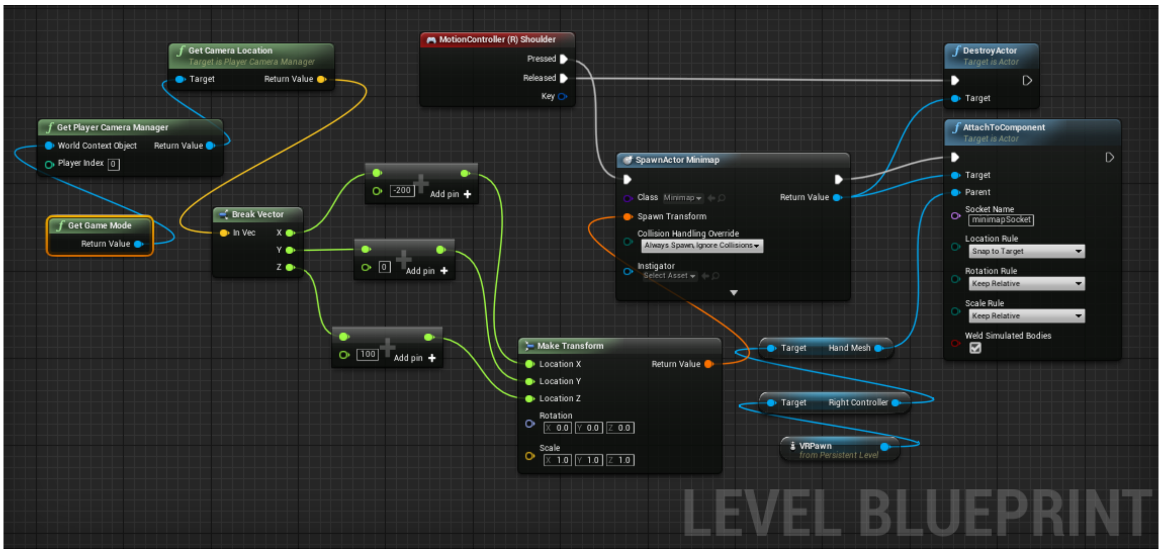
After the configuration, the Minimap Widget is now ready to be added to the Blueprint Class Minimap as a Widget Component (Figure 55).



Variable Name: MinimapWidget
Tooltip:
Category: Default
Editable when Inherited: ✓
Location: 0.0 cm 0.0 cm 0.0 cm
Rotation: 0.0° 0.0° 0.0°
Scale: 1.0 1.0 1.0
Parent Socket: None
Element 0: Widget3DPassT
Widget Class: MinimapWi
Draw Size: X 650 Y 550
X: 650
Y: 550
Manually Redraw

Figure 55. Minimap Widget usage in Minimap Blueprint Class

Although this Blueprint Class is ready to be used, the mini-map actor does not need to be spawn at the beginning of the gameplay. It should only be created when the user presses the menu button of the right controller. This behaviour was done by using the Code Snippet 14.



Code Snippet 14. Mini-map actor behaviour implementation

When the user presses the menu button of the right controller, the Mini-map actor will be created and be attached to the right hand VR. This actor will be destroyed when the user releases the button. In the AttachToComponent function, the socket name is defined to let the program know which part of the hand that the mini-map should attach to. The minimapSocket was created in the MannequinHand_Right_Skeleton of the Blueprint Class BP_MotionController (Figure 56).

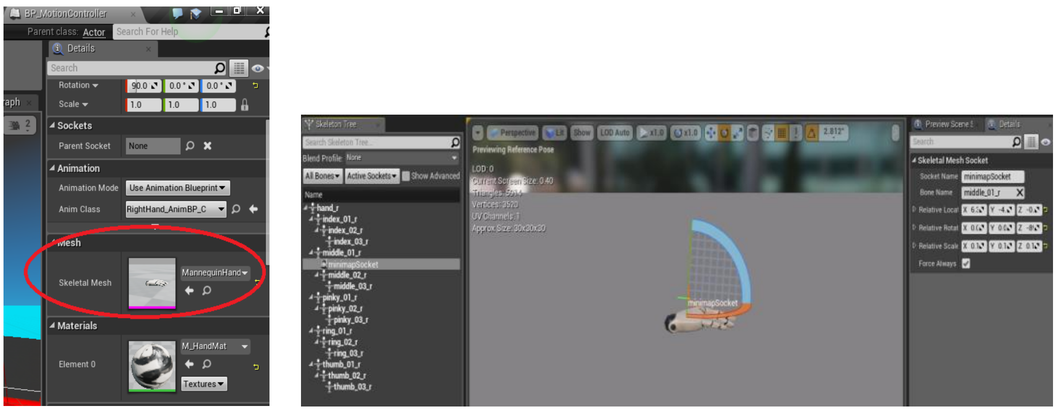
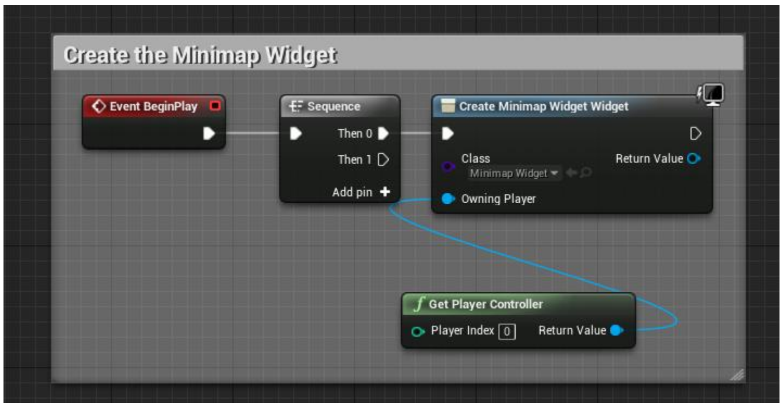


Figure 56. minimapSocket Creation

Because the mini-map Actor does not exist until the controller button is pressed, the Minimap Widget that the Actor used needs to be ready for the Blueprint class to use. This is why it needs to be created at the starting point of the scene. This was done by adding the Code Snippet 14 to the Level Blueprint.



Code Snippet 15. Minimap Widget Creation at beginning

5.4.5 Engine filter maintenance work

The engine filter maintenance procedure requires the user to do a series of actions in order to achieve success. Figure 57 illustrates the series of actions need to be done in the gameplay.

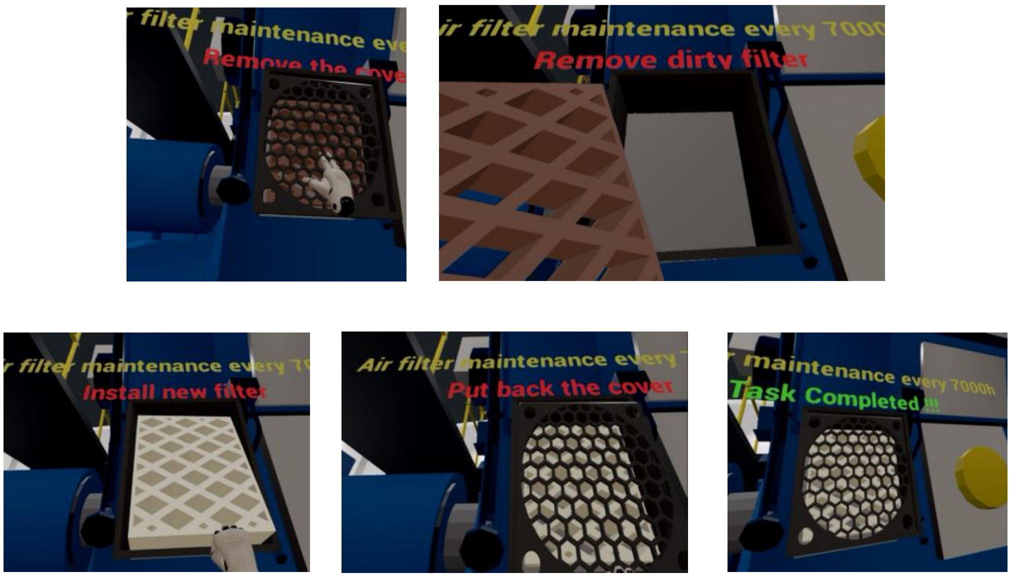
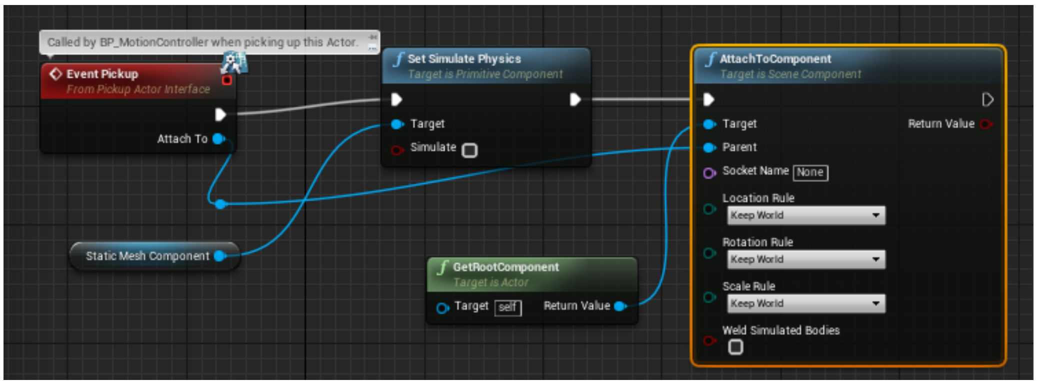


Figure 57. Engine filter maintenance procedure

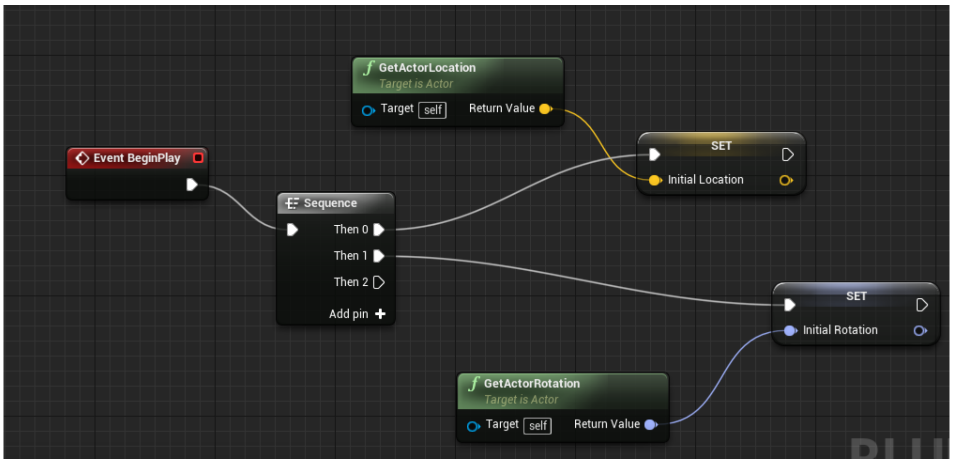
Three crucial Blueprint class was created for this maintenance job: the top filter, the dirty filter and the clean filter. These three actors in the scene communicates with each other to decide their behaviors. The Text Render Actor is used to guide the user what to do next. The TriggerVolume Actor is invisible to the user but it is used to determine if the user puts correct objects (filter, cover) to right place.

For the top cover of the filter box, at the beginning, it needs to be fixed in the box, so its Simulation Physics value needs to be set to False as a default value. When this Actor is being picked up by the controller, it will become attached to the controller and stay there to simulate grabbing action (Code Snippet 16).



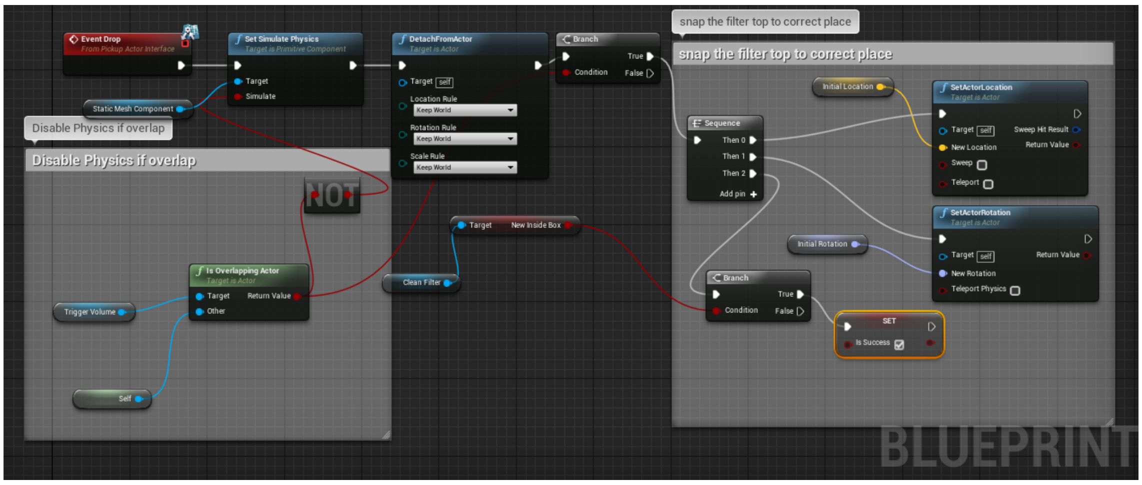
Code Snippet 16. Top cover implementation (1)

At the beginning, the Initial Location and the Initial Rotation variables will be set. These variables will be used to help the top cover snap back to correct position and rotation. (Code Snippet 17)



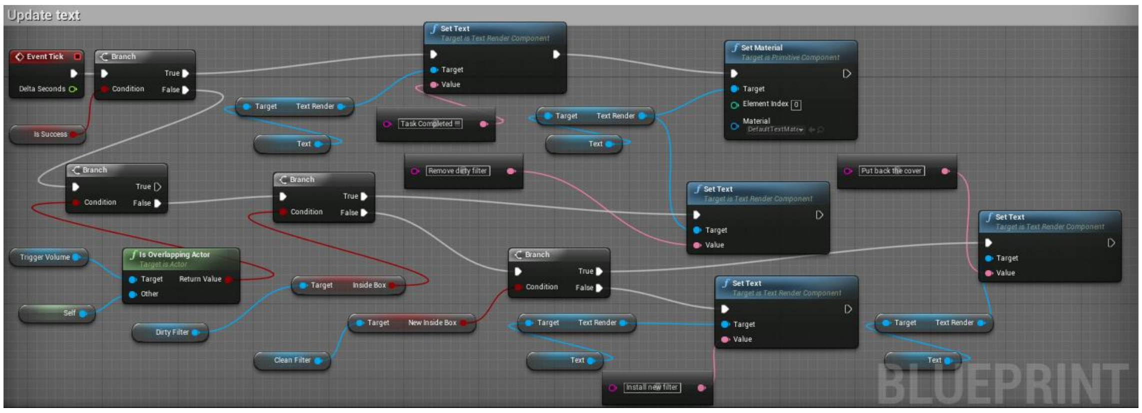
Code Snippet 17. Top cover implementation (2)

As mentioned earlier, the top cover has the default value of the Simulation Physics, which is false. This value will be set to True if the cover is not overlapping the TriggerVolume. After it has been dropped, the top cover will be detach from the controller. If the cover is overlapping the TriggerVolume, it will be snapped back to the old position and rotation. And then if the new filter has already been installed, the variable IsSuccess will be set to true, which confirms the task has been completed. The Drop event implementation is illustrated in Code Snippet 18.



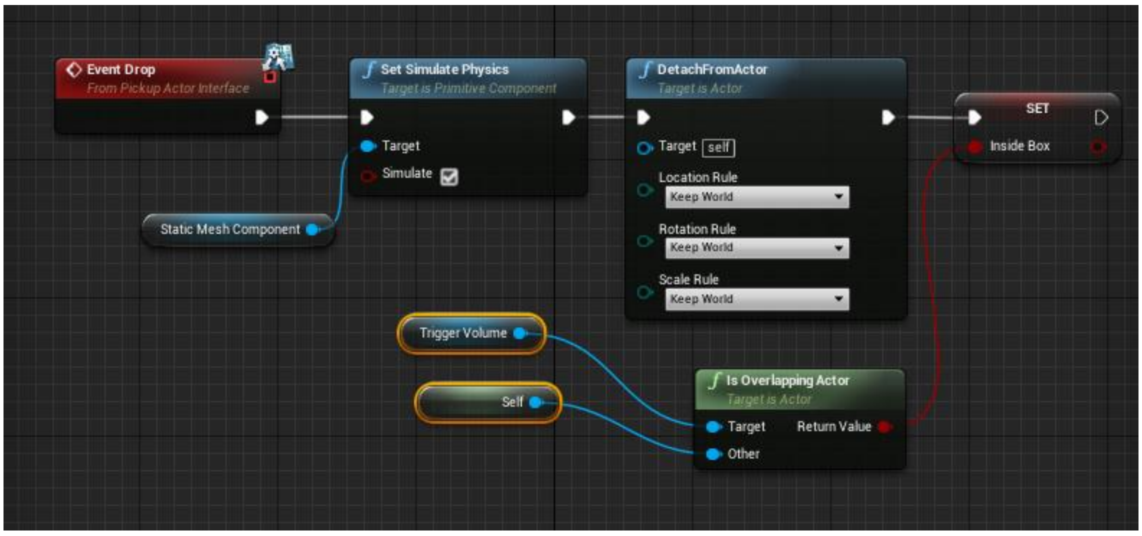
Code Snippet 18. Top cover implementation (3)

The Code Snippet 19 was also added to the top cover blueprint. This script is used to change the instruction text. By using the Tick event, multiple variables will be checked in every frame to determine what will be the text value. For the gameplay, at first, the text has the default value as “Remove the cover”. After the cover has been removed, it will change to “Remove dirty filter”. When the dirty filter has been thrown out, it will change to “Install new filter”. After this has been done, the text will change to “Put back the cover”. Finally, the text “Task completed !!!” with a different colour will appear when the cover has been put back.



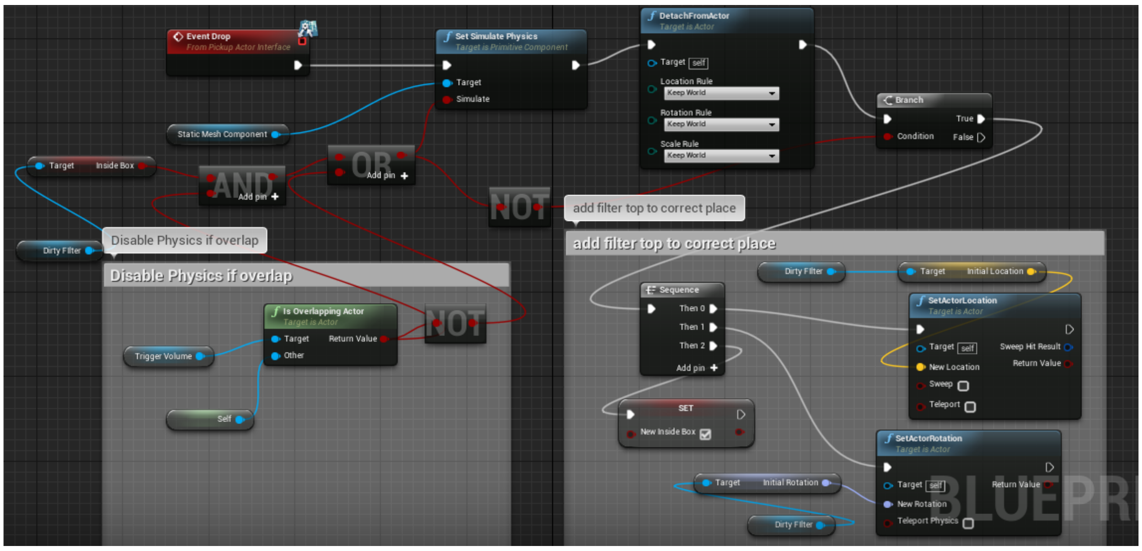
Code Snippet 19. Top cover implementation (4)

For the dirty filter of the filter box, the event PickUp has the same implementation as for the top cover. And for the event Drop, there is one difference that after the dirty filter is detached from the controller, the variable InsideBox will be set to true if this filter is overlapping the Trigger Volume and vice versa (Code Snippet 19). This filter also needs to add Code Snippet 17 into its Blueprint. The initial location and the initial rotation of the dirty filter will be used for the clean filter so that the clean one will be in the same position as the dirty one.



Code Snippet 20. Dirty filter implementation (1)

For the clean filter to be installed in the filter box, the event PickUp has the same implementation as for the dirty filter. The event Drop, however, has a more complicated logic for the implementation because many circumstances can happen with this Actor. The clean filter will have the same behaviour as normal interactable objects when it is outside the box or when it is inside the box but the dirty filter is still inside the box. The clean filter will become fixed to the correct position when it is inside the box and the dirty filter is outside the box. This logic was implemented as in Code Snippet 22.



Code Snippet 21. Clean filter implementation

5.4.6 Steam leak scenario

The second task that the user needs to do in this scene is to close a leaking valve. The user needs to press the trigger button of the controller to start the action. The procedure of closing the valve is demonstrated in Figure 58.

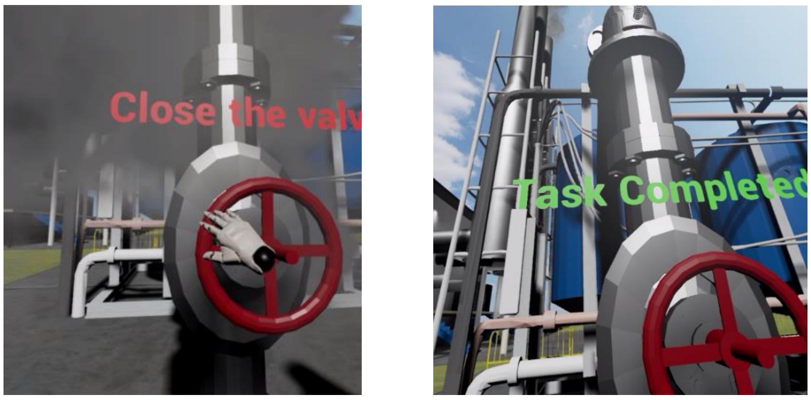
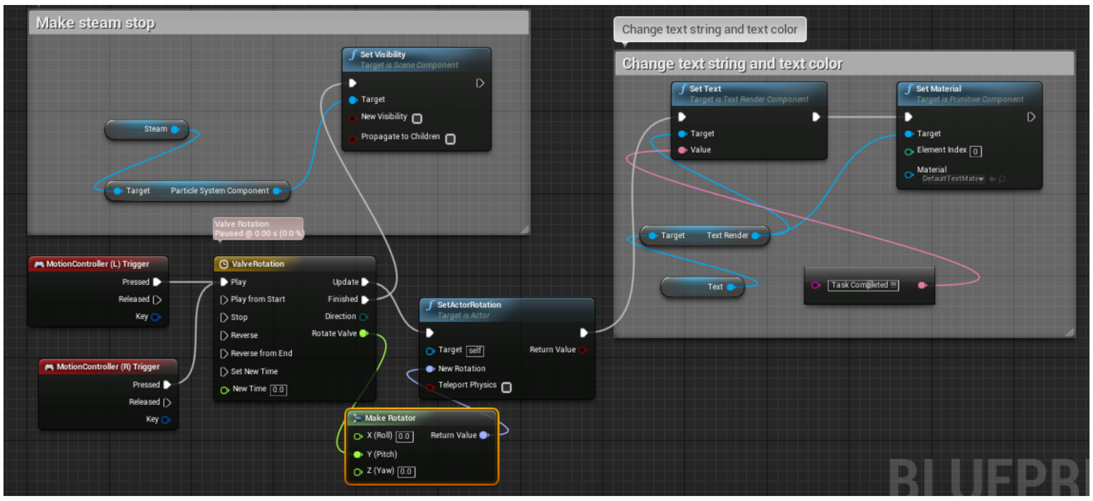


Figure 58. Valve Closing Procedure

The valve uses Code Snippet 23 to enable the interaction. The script uses timeline to display the animation and uses SetVisibility function to make the steam disappear. The script also changes the instruction text's content and colour to indicate task success by using SetText function and SetMaterial function.



Code Snippet 22. Valve implementation

6 TESTING AND DEBUGGING

6.1 Software testing

Software testing is a phase of software development in which the testers execute various tests with the intent of finding software bugs. It takes an important role in the development process to ensure the software quality and prove that all software functions work as expected.

During the development process, tests were carried out after every requirement of the QFD had been achieved. Additionally, after the application had finished, there was a final test for all functions by numerous users. Table 3 shows the testing results for each of application requirements.

Table 3. Functions testing results

No	Function to test	Expected result	Actual result
1.	User movement	The user is able to walk around or teleport using the controllers	OK
2.	Object interaction	The user is able to interact with objects by using the trigger button.	OK
3.	Object physics simulating	The interactable objects simulate physics such as colliding each other or gravity falling	OK
4.	Scene switching	The user is able to change the current scene.	OK
5.	IoT ticket screen	The user can use IoT ticket system with the screen and the controller while he/she is in control room scene	OK

6.	Mini-map display-ing	The user can open the mini-map by pressing menu button of right controller while he/she is in the engine room scene	OK
7.	Maintenance task (Changing engine filter)	The user can perform multiple actions in order to finish the task by following the given instructions.	OK
8.	Emergency task (Closing leaking valve)	The user can close the leaking valve by following the given instructions.	OK

6.2 UE4 Debugging

As mentioned earlier, when working with Blueprints of UE4, the Blueprint Visual Scripting provides its own unique debugger. This section of the document shows how this debugger can be used with a simple example of the interactable door in the engine room scene.

After the user interaction, the door should open (move to the left), but at the first test, it disappears without any trace. Without the debugger, this bug can be difficult to be solved.

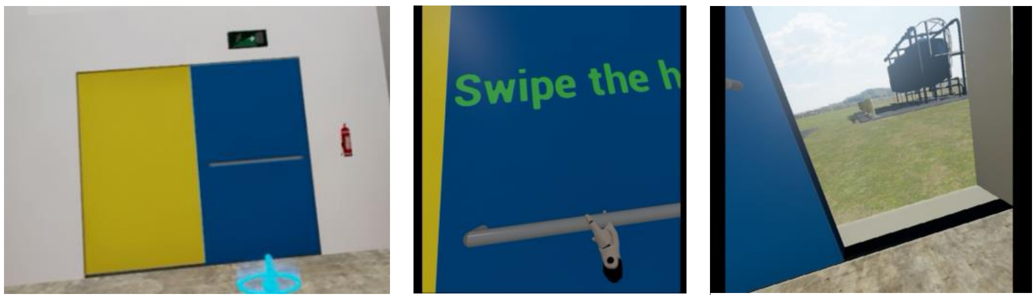


Figure 59. Door disappearance

To enable Debugging of a Blueprint, an instance of the Blueprint in the scene must be chosen. Inside the Blueprint, this instance can be chosen in in the Debug Object drop-down menu (Figure 60).

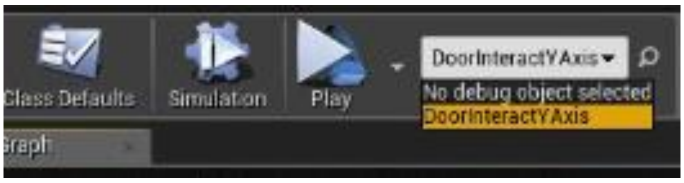


Figure 60. DoorInteract enable Debugging (1)

A breakpoint is added to SlidingDoor Timeline node by right click this node and choose “Add breakpoint” (Figure 61).

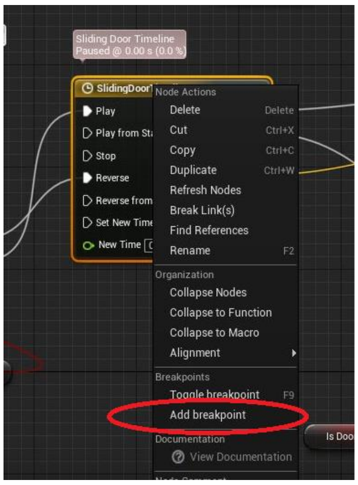


Figure 61. Door Debugging (1)

The Sliding Left value is watched by right clicking the pin and then by choosing “Watch this value”. This value is the same as the new location value of the door (Figure 62). The IsDoorOpen variable is also watched by using the same action.

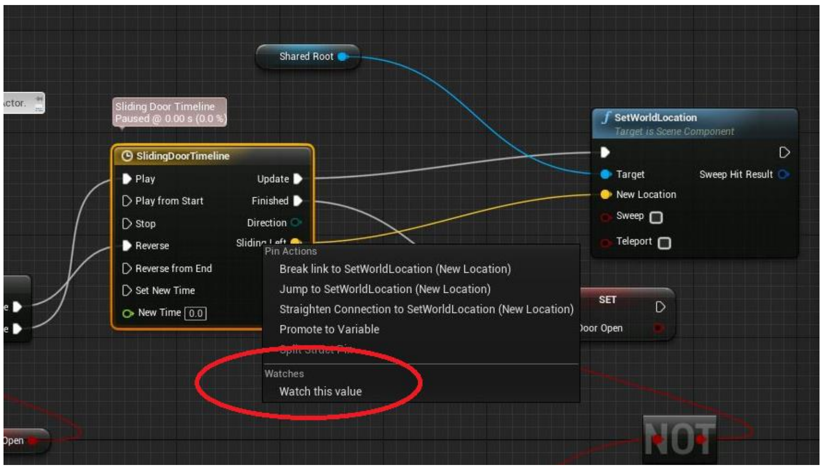


Figure 62. Door Debugging (2)

After the breakpoint is added, when the user interacts with the door, the application will pause. The Editor will show in which node the application is running. At this moment, the IsDoorOpen variable value is False and the value of the new location is still (0,0,0) as in Figure 63.

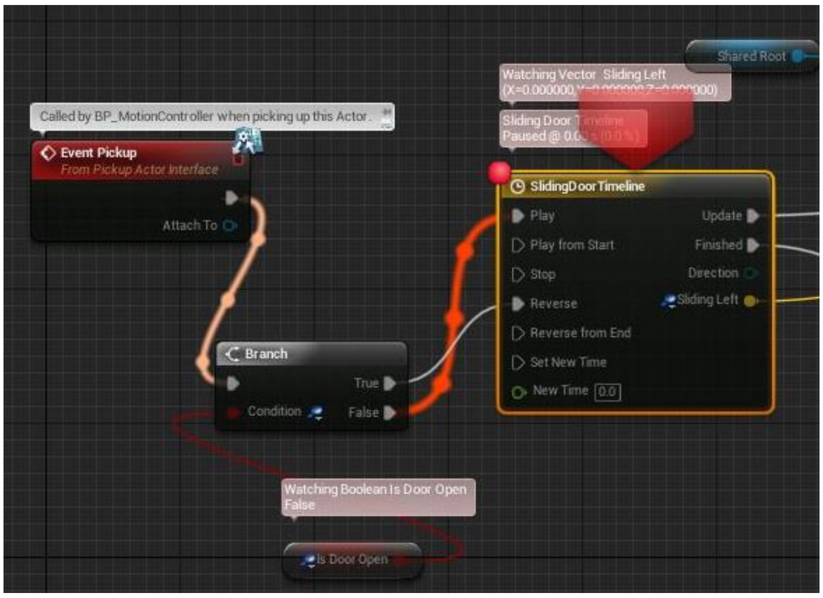

Shared Root
Watching Vector Sliding Left (X=0.000000,Y=0.000000,Z=0.000000)
Sliding Door Timeline Paused @ 0.00 s (0.0 %)
Called by BP_MotionController when picking up this Actor.
Event Pickup
From Pickup Actor Interface
Attach To
SlidingDoorTimeline
Play
Play from Start
Stop
Reverse
Reverse from End
Set New Time
New Time 0.0
Update
Finished
Direction
Sliding Left
Branch
True
Condition
False
Watching Boolean Is Door Open False
Is Door Open

Figure 63. Door Debugging (3)

By using the Step Into button, the next node can be executed. (Figure 64)

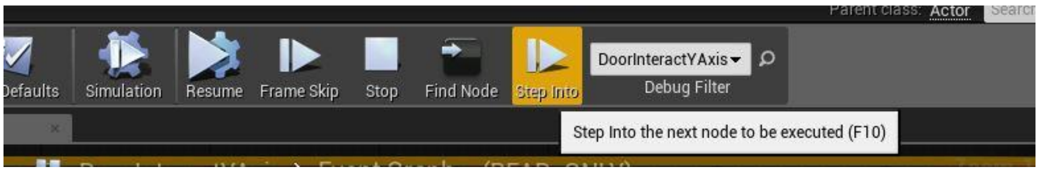


Figure 64. Next node execution

After pressing this button multiple times, the whole procedure of opening the door is observed and it works normally. After the user interaction, the code will run SlidingDoorTimeline node. The door position is updated according to the timeline and when the timeline finishes, the final position of the door is (0,150,0) and then the IsDoorOpen variable is set to True so that the next time the user interacts with the door, the timeline will run in reverse (Figure 65).

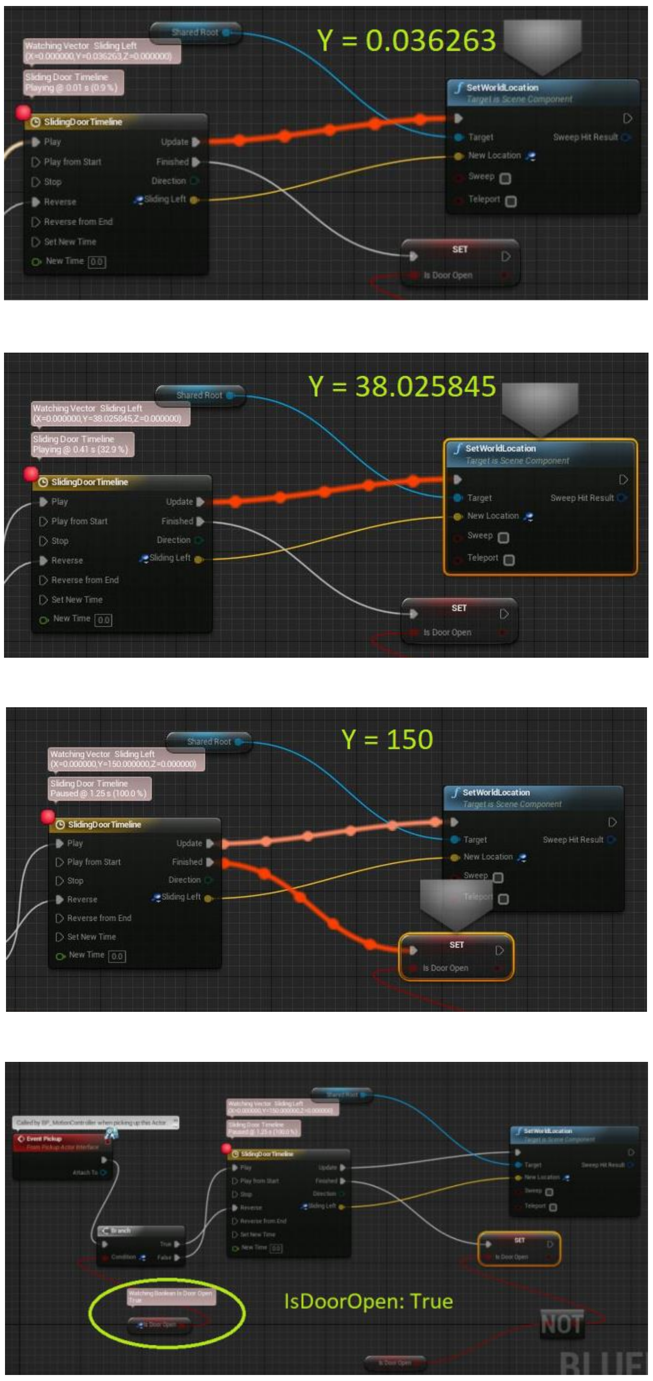
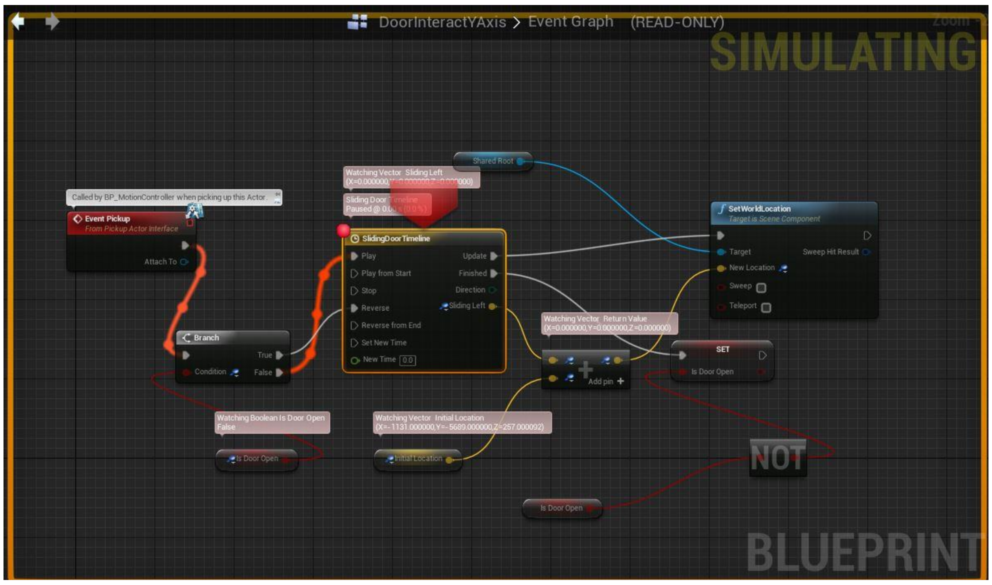
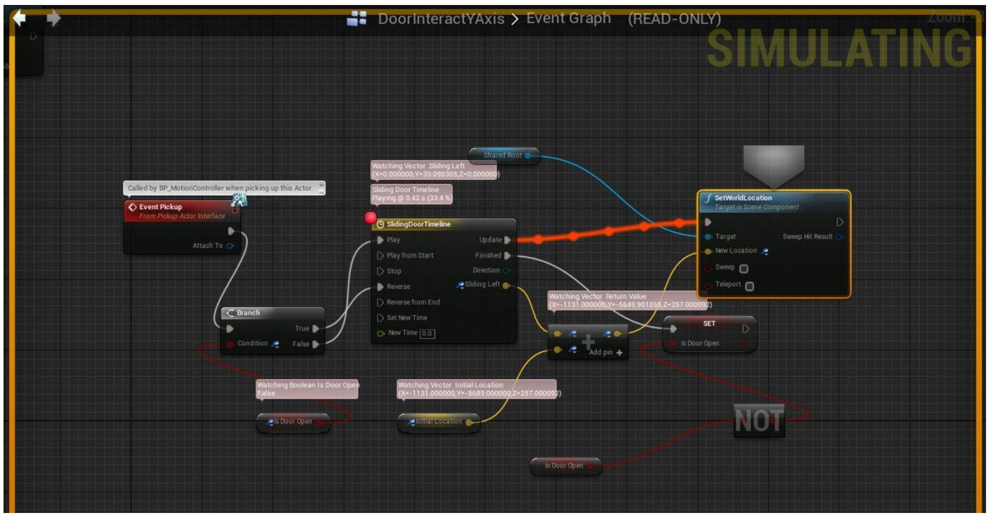
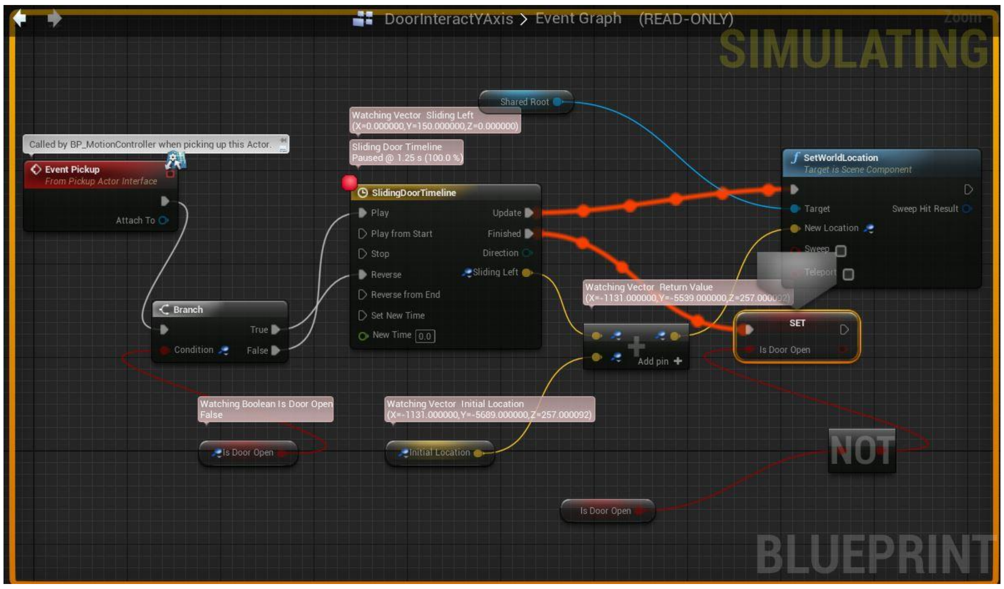
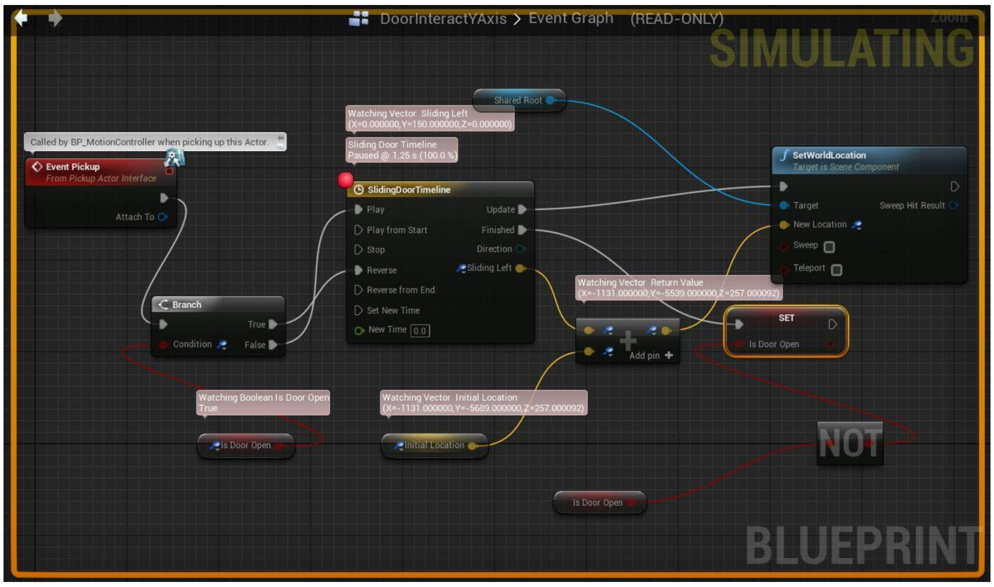

Y = 0.036263
Watching Vector Sliding Left (X=0.000000,Y=0.036263,Z=0.000000)
Sliding Door Timeline Playing @ 0.01 s (0.9 %)
Y = 38.025845
Watching Vector Sliding Left (X=0.000000,Y=38.025845,Z=0.000000)
Sliding Door Timeline Playing @ 0.41 s (32.9 %)
Y = 150
Watching Vector Sliding Left (X=0.000000,Y=150.000000,Z=0.000000)
Sliding Door Timeline Paused @ 1.25 s (100.0 %)
SlidingDoorTimeline
Play
Play from Start
Stop
Reverse
Reverse from End
Set New Time
New Time
Update
Finished
Direction
Sliding Left
Shared Root
SetWorldLocation
Target is Scene Component
Target
New Location
Sweep
Teleport
Sweep Hit Result
SET
Is Door Open
Event Pickup
Branch
IsDoorOpen: True
NOT

Figure 65. Door Debugging (4)

The reason the door disappears is simply because it moves to the wrong position that the user cannot see. After the InitialLocation is implemented in the Blueprint, the door works as it should. (Figure 66)


DoorInteractYAxis > Event Graph (READ-ONLY)
SIMULATING
Called by BP_MotionController when picking up this Actor
Event Pickup
From Pickup Actor Interface
Attach To
Branch
Condition
True
False
Shared Root
Watching Vector Sliding Left
SlidingDoorTimeline
Play
Play from Start
Stop
Reverse
Reverse from End
Set New Time
New Time
Update
Finished
Direction
Sliding Left
SetWorldLocation
Target is Scene Component
Target
New Location
Sweep
Teleport
Sweep Hit Result
Watching Vector Return Value
Add pin
SET
Is Door Open
Watching Boolean Is Door Open
False
Watching Vector Initial Location
Initial Location
NOT
BLUEPRINT

DoorInteractYAxis > Event Graph (READ-ONLY)
SIMULATING
Called by BP_MotionController when picking up this Actor
Event Pickup
Sliding Door Timeline
SlidingDoorTimeline
SetWorldLocation
Branch
SET
Is Door Open
NOT

DoorInteractYAxis > Event Graph (READ-ONLY)
SIMULATING
Called by BP_MotionController when picking up this Actor
Event Pickup
From Pickup Actor Interface
Attach To
Shared Root
Watching Vector Sliding Left
(X=0.000000,Y=150.000000,Z=0.000000)
Sliding Door Timeline
Paused @ 1.25 s (100.0 %)
SlidingDoorTimeline
Play
Play from Start
Stop
Reverse
Reverse from End
Set New Time
New Time
Update
Finished
Direction
Sliding Left
SetWorldLocation
Target is Scene Component
Target
New Location
Sweep
Teleport
Sweep Hit Result
Watching Vector Return Value
(X=-1131.000000,Y=-5539.000000,Z=257.000092)
Add pin
SET
Is Door Open
Branch
Condition
True
False
Watching Boolean Is Door Open
False
Watching Vector Initial Location
(X=-1131.000000,Y=-5689.000000,Z=257.000092)
Initial Location
NOT
BLUEPRINT



Called by BP_MotionController when picking up this Actor.

Watching Vector Sliding Left
(X=0.000000,Y=150.000000,Z=0.000000)

Sliding Door Timeline
Paused @ 1.25 s (100.0 %)

Watching Vector Return Value
(X=-1131.000000,Y=-5539.000000,Z=257.000092)

Watching Boolean Is Door Open
True

Watching Vector Initial Location
(X=-1131.000000,Y=-5689.000000,Z=257.000092)

Figure 66. Correct door workflow

7 CONCLUSION

The main objective of the project was to develop a virtual environment simulating a power plant. This application runs on Windows platform and guides the user to perform virtual industrial operations and maintenance work. The project achieved its main objectives. During the application development process, many implementation approaches were considered and tested in order to choose the most suitable one for optimization and performance purposes. The application was tested multiple times and it run smoothly and stably.

The main challenges in the development process were to become familiar with Unreal Engine and to acquire knowledge of 3D computer graphics. It also took a large amount of time to find and put the 3D object models into use. The assets in the application either have a Creative Common licence or were created from scratch.

Developing the application has provided an opportunity to learn about new technologies. There were many problems and difficulties during the implementation but all of them have been overcame in the end. The application has much potential for further development. Many other industrial operations can be added to the scene, the engine can be updated to a new model and the virtual environment can still be modified in order to appear more realistic.

REFERENCES

/1/ How does Virtual Reality work? Accessed 15.10.2017

http://time.com/3987716/how-does-virtual-reality-work/

/2/ HTC Vive Accessed 15.10.2017

https://en.wikipedia.org/wiki/HTC_Vive

/3/ Explained: How does VR actually works Accessed 15.10.2017

https://www.wareable.com/vr/how-does-vr-work-explained

/4/ Unreal Engine Accessed 15.10.2017

https://en.wikipedia.org/wiki/Unreal_Engine

/5/ Visual programming language Accessed 15.10.2017

https://en.wikipedia.org/wiki/Visual_programming_language

/6/ Programming Guide Accessed 15.10.2017

https://docs.unrealengine.com/latest/INT/Programming/index.html

/7/ About Blender Accessed 15.10.2017

https://www.blender.org/

/8/ About Wärtsilä Accessed 15.10.2017

https://www.wartsila.com/about

/9/ Unreal Engine 4 Projects Accessed 15.10.2017

https://docs.unrealengine.com/latest/INT/Engine/Basics/Projects/index.html

/10/ Unreal Engine 4 Terminology Accessed 15.10.2017

https://docs.unrealengine.com/latest/INT/GettingStarted/Terminology/index.html

/11/ Blueprint Class Accessed 15.10.2017

https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Types/ClassBlueprint/index.html

/12/ Actors and Geometry Accessed 15.10.2017

https://docs.unrealengine.com/latest/INT/Engine/Actors/index.html

/13/ Pawn Accessed 15.10.2017

https://docs.unrealengine.com/latest/INT/Gameplay/Framework/Pawn/index.html

/14/ Components Accessed 15.10.2017

https://docs.unrealengine.com/latest/INT/Engine/Components/index.html

/15/ Gameplay Programming Accessed 15.10.2017

https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/index.html

/16/ VR Template Guide for Unreal Engine 4 Accessed 15.10.2017

http://www.tomlooman.com/vrtemplate/