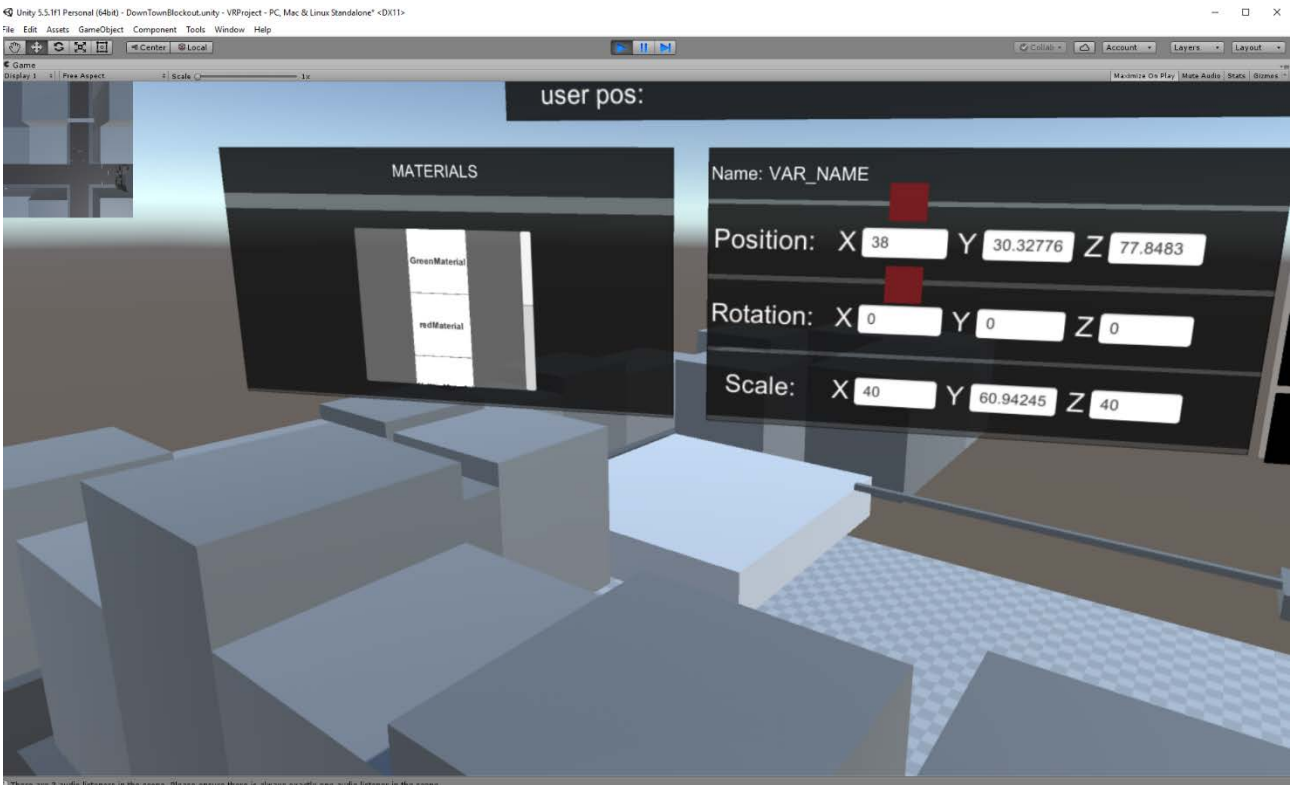
VIRTUAL REALITY EDITOR APLICATION FOR UNITY 5



Fernando Ferrando Terradez

S6098981 Teesside University

Computing Project

Special thanks

Special thanks to Adam Beard to let me the blockout of his final year project in order to make the artefact look more professional and lively and for helping me in UI design for the editor

Index

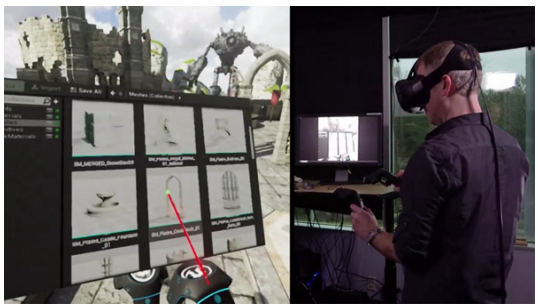
Introduction

This report is going to cover the process of the application building, with my own conclusions and what I learned through all the process.

I will begin with the idea. The application I wanted to work on is a Virtual Reality editor, Virtual Reality is a technology that have had an increase of its popularity last years, in technical terms from the Virtual Reality Society. (2017), *"Virtual reality is the term used to describe a three-dimensional, computer generated environment which can be explored and interacted with by a person. That person becomes part of this virtual world or is immersed within this environment and whilst there, is able to manipulate objects or perform a series of actions."* in order to make this environments, there are some physiology that a virtual reality world/game needs to be considered as Virtual Reality Society. (2017) *states "For example, the human visual field does not look like a video frame. We have (more or less) 180 degrees of vision and although you are not always consciously aware of your peripheral vision, if it were gone you'd notice. Similarly when what your eyes and the vestibular system in your ears tell you are in conflict it can cause motion sickness. Which is what happens to some people on boats or when they read while in a car."* And what is motion sickness? If we look at Nhs.uk. (2017), motion sickness is a mismatch in what the eyes see and what the inner ears, the sense of balance, sense, if this mismatch occurs, the brain will not be able to update the current status and the confusion created can lead to the symptoms of motion sickness, such as severe nausea.

If we want to avoid that, we need tools that help the designers and artists to develop worlds, environments that are realistic enough and with all the care and best practices said in Developer3.oculus.com. (2017). for that, virtual reality editors where created, to help visualizing virtual worlds in the development process. Two of the virtual reality editors that exist in the present are the Unreal Engine virtual reality editor and the Unity3D virtual reality editor.

(Roadtovr.com, 2017)



(Allvirtualreality.com, 2017)

The main characteristic they have, is that they are similar to a normal in-built editor engine, with the only difference is that you are inside the game with the virtual reality headset, being able to know directly how the assets and environmental work and being able to make sure everything in the scene meets the criteria of a virtual reality environment. The problem is that they are bind to the engine itself, you need to use their engine in order to visualize the world, the artefact that is going to be created in this project will be a partially independent application, exempting the designers or artists to handle the engine and its features, providing a much easier tool.

As a map editor, I wanted it to have the main features of a game editor has, which are the following:

- Movement and Virtual Reality headset
- Transform inspector, being able to change the following features
    - Position
    - Rotation
    - Scale
- Material change
    - Change the material of an asset.
- Select objects to edit in the map.
- Being able to move in the map editing.
- Placing assets in the map.
- Load scenes and assets in the application
    - As it is a separate application, I need to be able to load resources in runtime for editing
- Exporting and importing changes
    - Using JSON for exporting the changes made in the map
    - Using the editor in built features to create a button or option to be able to load the changes of the JSON export file.

I was thinking about this since last year, I thought it would suit as a final year project and would aid me to acquire the skills I needed for my ideal future job in the industry. It was going to be a complex project that involved a wide range of fields I needed to study, as user experience, motion sickness caused by virtual reality, that are not really studied in the degree.

Why Unity

The first step was to decide about the engine editor I wanted to use for the application, that was a problem because I had three main ideas to make the application, two of them were current engines in the market, Unity 5 and Unreal Engine 4, and the other was making it by myself using a graphic library like SFML or SDL.

I made fast sketches for each application in paper, trying to follow the path I would need to take and what I would need for each option, here I discarded using a graphic library, because of the scope of the application and the time given, I would not be able to produce a application which met my standards, the pipeline of work and the amount of work to do because of virtual reality would be more that I could afford. Next was trying to get the best of the other two options, I have worked quite a lot in Unreal because previous and current projects in my degree and my career, and my experience told me that Unreal has some problems for short term works:

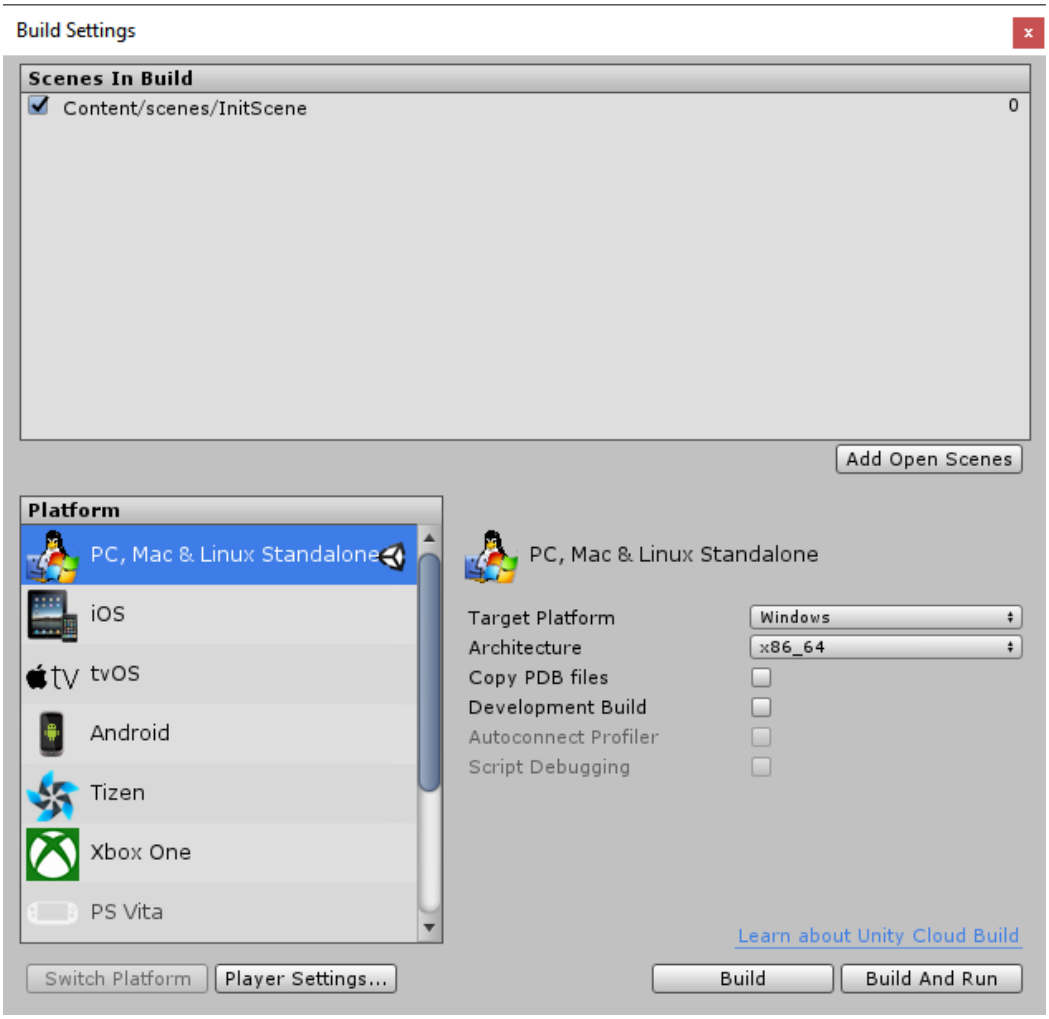
- C++ builds and code are too unstable and tend to break the engine frequently, which means time lost having to compile again the engine and the possibility to lose your work.
- I could make the editor in little time by using blueprints, but it would not suit a computer project, was out of the question because academic purposes.
- Unreal Engine 4 tends to produce bigger applications than Unity 5 when compiled
- The information of Unreal Documentation in C++ is lacking and updating, and finding new information about pipeline and methods is more difficult than for Unity.

As for Unity, I made a pair of little projects in the past, but nothing too serious, so I started to search informatio, I reviewed the main things I needed to work on, UI, real time asset loading, exporting assets or information in a certain format, like JSON or XML.

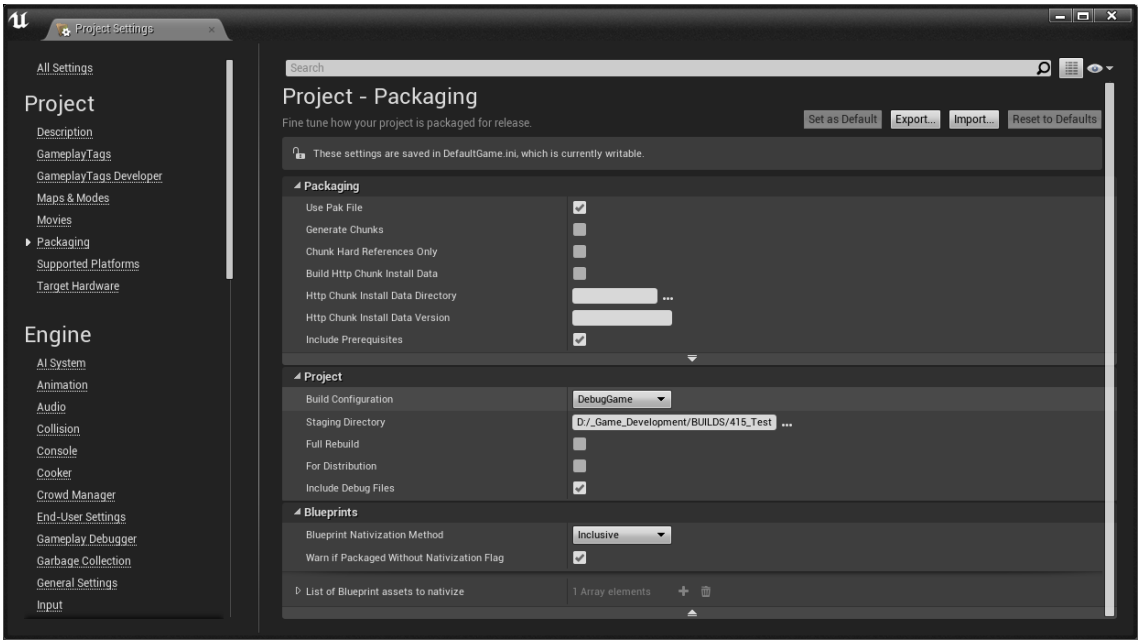
In regards to the code, being a C# script system it will allow me to get my work modularized easy, being able to take care of each module individually. Furthermore, it counted with an extension made by Oculus team, (you can find all the information about the oculus utilities for unity in the following link: Oculus prefab) that will allow me using a prefab to have all the functionalities I needed for the headset. With this information and the urge to learn more about Unity, made me go for Unity 5 in this project, the version used is Unity 5.5.1f1 (64-bit), which is the last version at the moment of the development start.

While working in Unity, I did not have any specific problems we could say they were because of Unity, the pipeline was easy to understand and the modularity that Unity offered my via the scripting system they have and component based engine, proved me I was in the right direction. Nevertheless I found the Unity user interface system, Immediate Mode GUI, and the main GameObject based UI, more difficult, complex, and less usable than the Unreal Engine system UMG, the difference is that adding functionality to the user interface in Unreal Engine is much easier than in Unity, there was no specific problems in the development, but the coding and pipeline for the user graphic interface was way more complex and less intuitive in Unity than in Unreal Engine.

I also learnt that building a project in Unity is also more lightweight than in Unreal and has less options to bake the .exe file, making this process easier as we can see in the images below, Unity offers an easy and configurable way to compile the project in different platforms whereas Unreal has a far complex window for configuring the build in the engine.



Unity build system

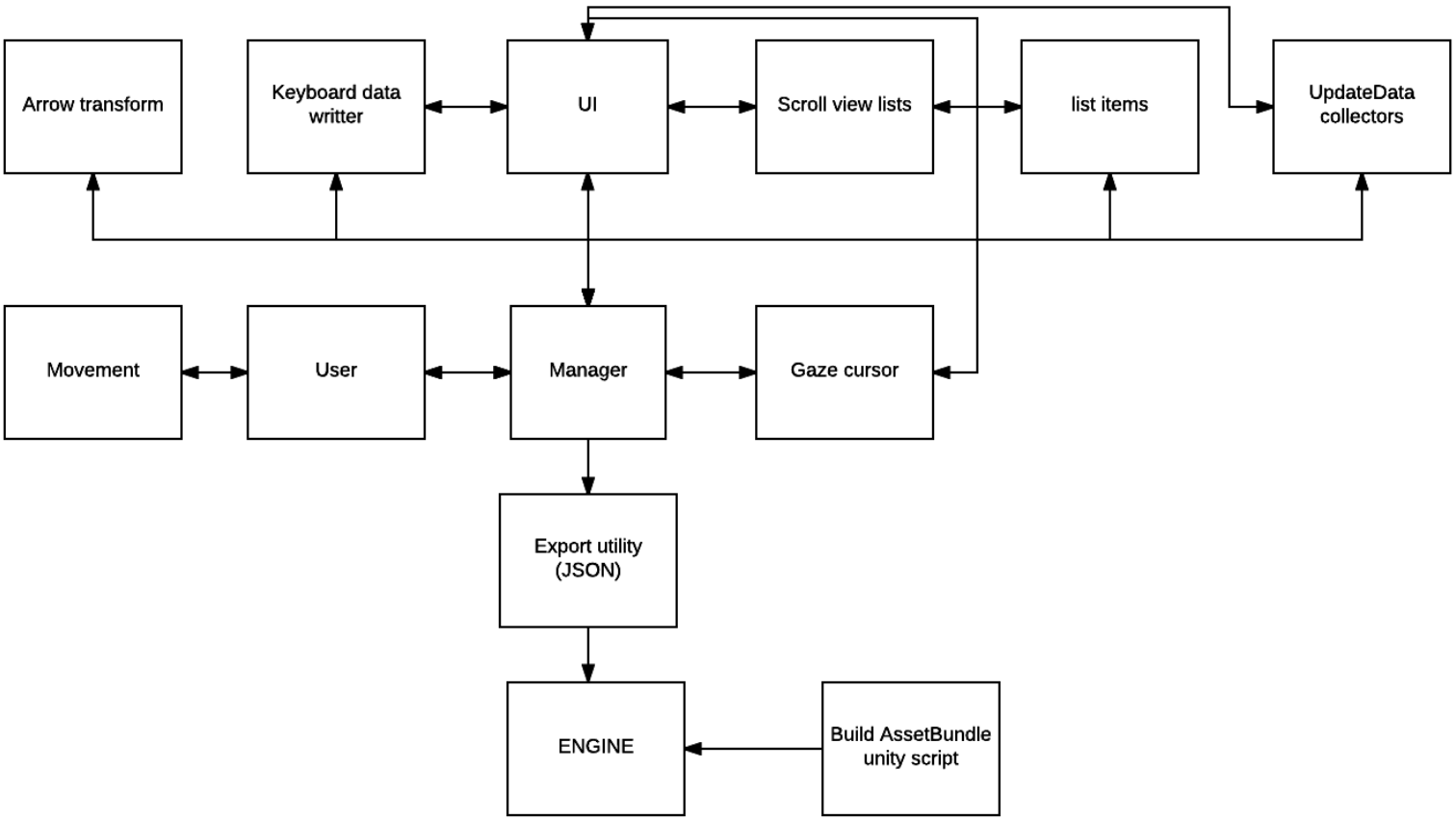


Unreal Packaging system.

Now the project is finished, I think I have made a good choice in using Unity instead of the other two options, it proved to be a powerful engine with a wide range of useful features that allowed me to tackle my project in an effective way and thanks to the modularity it offers, that made me able to solve the problems I had with every single module just coding and fixing things for that specific module without the need to intervene with other parts of the code.

I particularly value all the experience I have gained with this engine which surely will become useful in the near future being one of the most used engines by the videogames studios, coding with the API, Virtual Reality Management in the engine, Graphic User Interface in Unity IMGUI system and overall user experience with Unity.

Project structure



Here is the main the application flow and the relationship between the modules of the application. A centralized design pattern was chosen, in which all the main flow of the virtual reality editor is managed by the manager, made with a singleton pattern, in order to only allow one instance of it in the map application. This pattern was selected because it ensures that the application is going to have only one instance of that object in all the application, regardless of how many objects of it are created, it is done with these lines of code.

```
static Manager instance = null;

void Awake()
{
    //make the manager a singleton
    if (instance == null)
    {
        instance = this;
    }
    else if (instance != this)
    {
        Destroy(gameObject);
    }
}
```

If another instance was going to be created, it gets automatically deleted when Unity Awake method is called, and this method is called by unity automatically at the moment of the script loading, as they say in the API, can be said that is the constructor of the class, as Unity themselves in the API page for the method Awake, Technologies, U. (2017). states the following *"Note for C# and Boo users: use Awake instead of the constructor for initialization, as the serialized state of the component is undefined at construction time. Awake is called once, just like the constructor."*

Thanks to the centralized structure, it helped me to manage all the modules only the dependencies it needed and even removing the dependencies of various modules and functionality, for example, the data collections does not affect the UI in more than having the collector inside, the functionality is managed by the collector script and the manager, and then the manager updates the UI, keeping all controlled by the manager.

Key elements in the application

Input

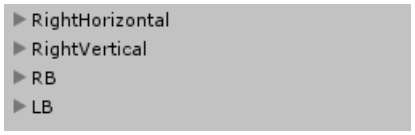
The input consists in two parts, the gaze control, that is the cursor of the game controlled with the Oculus prefab moving your head, which will be reviewed later in this report, and the gamepad buttons, that allows the player to move and interact with the application. Originally, Oculus touch pads were the chosen option, which allow more precision in order to pick objects, because they simulate hands in the application, a motion system tracks the position of your hands and with buttons in the pad, allows you to simulate the object picking of the world.

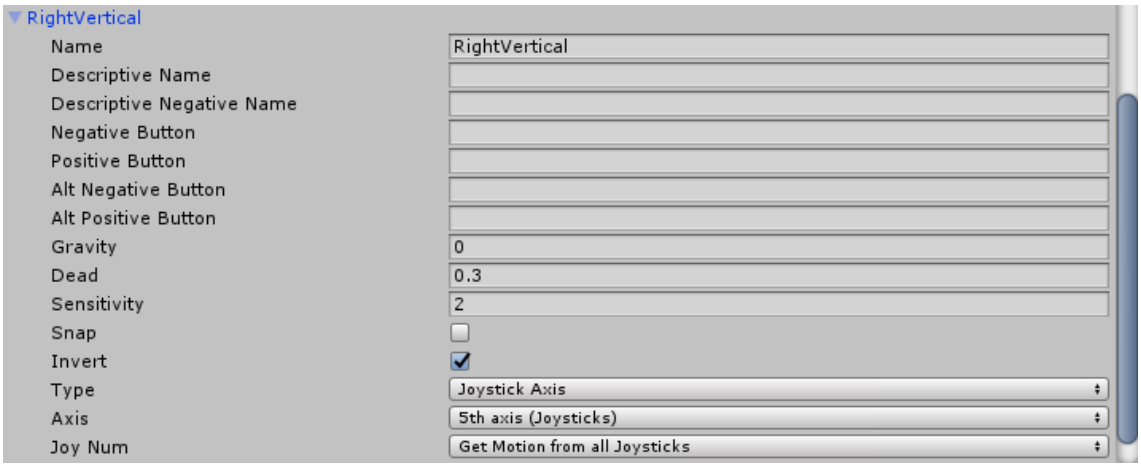
(Oculus touch pads)

Unfortunately, those pads were not available in the developing process, so as those could not be tested in the application and do not knowing if they were going to be available for Expotees fair, there was a need to find a solution. With that, the XBOX360 controller was the option chosen for developing a proper input control, because it is one of the Windows crafted gamepads for Windows environments.

For using the input in unity, I needed to configure the two sticks click action and the right and left shoulders of the gamepad. For that, the creation of a new input configuration in the input manager was needed; this is where you define all the different input axes and game actions for the project, as they say in their documentation at Technologies, U. (2017).

Four instances in the input manager were created, one for each pad I need to configure the data with the necessary values





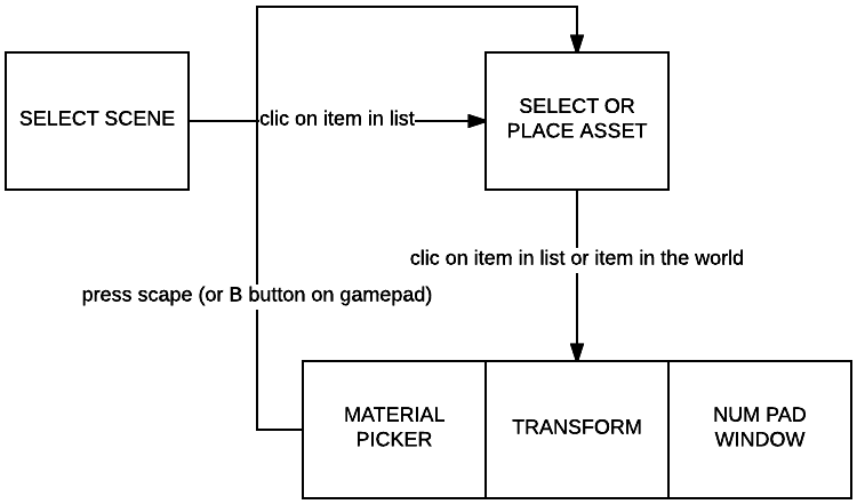
The data needed to configure each button was provided from the Wiki.unity3d.com. (2017)., the official wiki for Unity with useful information for various topics, as they state.

Input	Platform			Notes
	Windows	Mac Os X	Linux	
Button Name	**Mapped Button Number**			
A Button	0	16	0	

B Button	1	17	1	
X Button	2	18	2	
Y Button	3	19	3	
Left Bumper	4	13	4	
Right Bumper	5	14	5	
Back Button	6	10	6	
Start Button	7	9	7	
Left Stick Click	8	11	9	
Right Stick Click	9	12	10	
D-Pad Up		5	13	On Linux, only wireless controllers support using the D-Pad as buttons
D-Pad Down		6	14	
D-Pad Left		7	11	
D-Pad Right		8	12	
Xbox Button		15		

With the gamepad controller mapped, the controller was mapped for moving and interacting for the game, it does not give the desired experience when planned the project was planned, but there was a need to change the specifications and the planning of the project because having hardware planned at the beginning of the development was not possible. Nevertheless, this developed in a fine movement for a product, preserving the classic movement for First Person Shooter movements, helping to make it comfortable for people which are used to play games.

User Interface



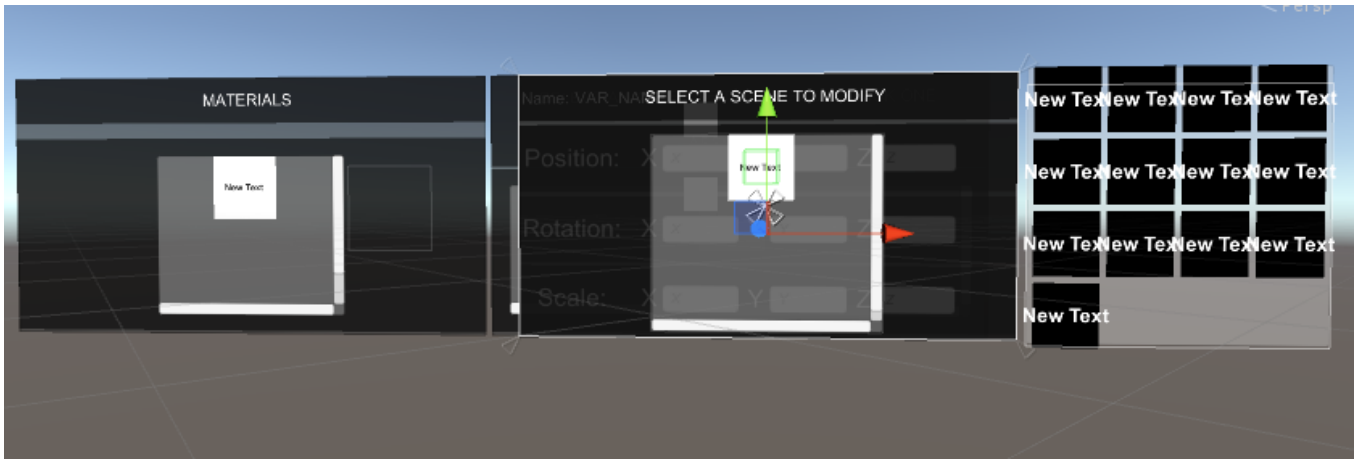
Working in the UI was an easy design choice, but a complex developing process, the design choice, show in the graph in the top of this paragraph and shows a simple but effective design for an UI that iterates and changes due certain flags.

For UI, I chose to go with the User Interface unity has for its engine, the UI in Unity is separated in components, it allows the developer to create graphically a UI and give it some functionality through scripting. The most important part in an UI in unity is the Canvas, as they state in (Technologies, U. 2017) "*The Canvas is the area that all UI elements should be inside. The Canvas is a Game Object with a Canvas component on it, and all UI elements must be children of such a Canvas.",* meaning that if you want to build an UI, you will need to start placing a Canvas in the world, they are three types of canvas, depending on the render mode.

- Screen Space – Overlay
  - This render mode places UI elements on the screen rendered on top of the scene. If the screen is resized or changes resolution, the Canvas will automatically change size to match this.
- Screen Space – Camera
  - This is similar to Screen Space - Overlay, but in this render mode the Canvas is placed a given distance in front of a specified Camera. The UI elements are rendered by this camera, which means that the Camera settings affect the appearance of the UI.
- World Space
  - In this render mode, the Canvas will behave as any other object in the scene. The size of the Canvas can be set manually using its "Rect Transform" (the same as a normal transformation component, but for UI elements) and UI elements will render in front of or behind other objects in the scene based on 3D placement. This is useful for UIs that are meant to be a part of the world.

As we need to build a VR application, we need the UI to be another element of the game, that has its own spatial position and representation in the world, because fixed UI in the front of the player, will lead to more chances of causing motion sickness to the user.

With that, adding panels, images for backgrounds and elements we will review in the features of the project, I came with this UI



As it is shown in the image, the UI elements are overlapped, but follows the flowchart in the beginning of this paragraph thanks to the following code (managed by the Manager class)

Each intractable element in the UI has a box collider that allows the gaze control to be able to interact with them and recognize each element of the UI, as it can be seen in the image attached above of the game UI.

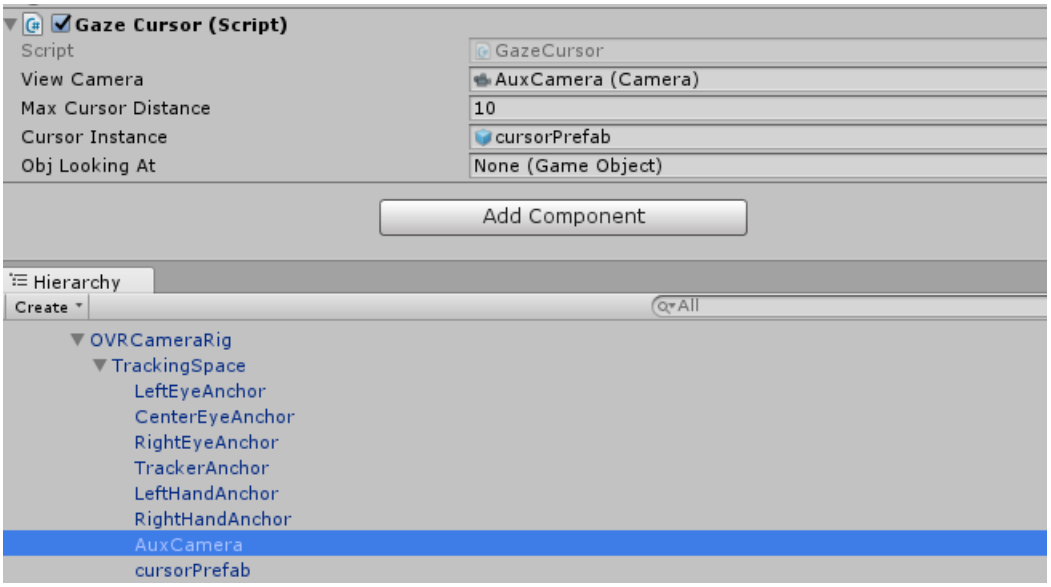
As the UI is a World Space canvas, is meant to be in a place on the world, for the comfort of the player, I have chosen to attach it to the physical representation of the player, meaning that the UI will follow the player in the world, staying in the same position always relative to the player, being able to be accessed by just turning the headset to the position of the UI.

With the project finished, this UI flow helped me to keep the development simple and the UI screens I needed for the application enclosed and designed in early stages. Aside from learning about UI systems in Unity, it proved a valuable experience in how a good early design can help to clarify the development in late stages and make the core code design clearly.

Gaze control

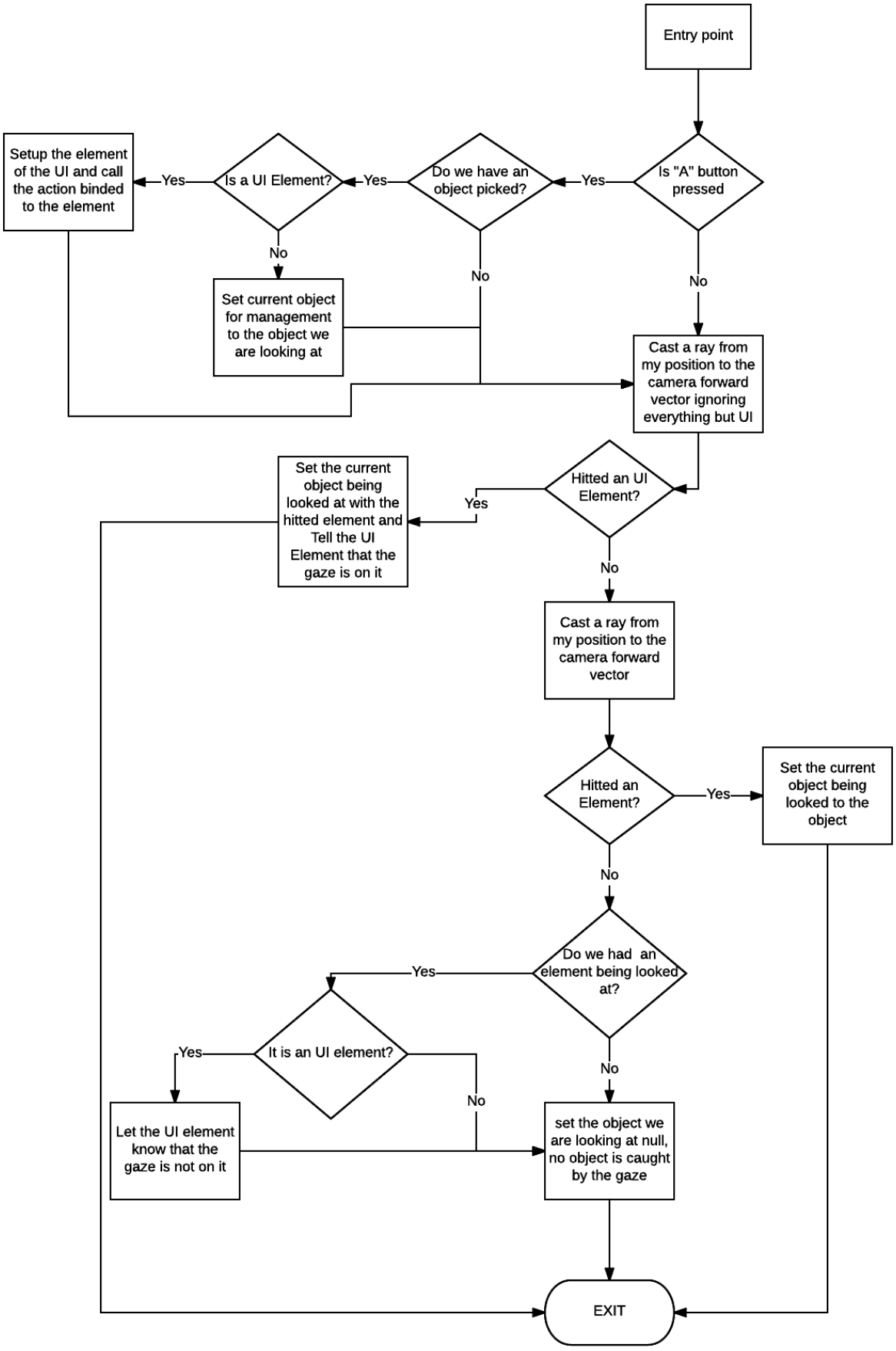
Gaze control is the cursor of the game, is the module that will allow the user interact with the elements in the game, for example, picking an element in the world or spawning a new asset from the item list.

Gaze cursor is an script that is attached to a special new camera attached in the Oculus VR prefab element from the game.

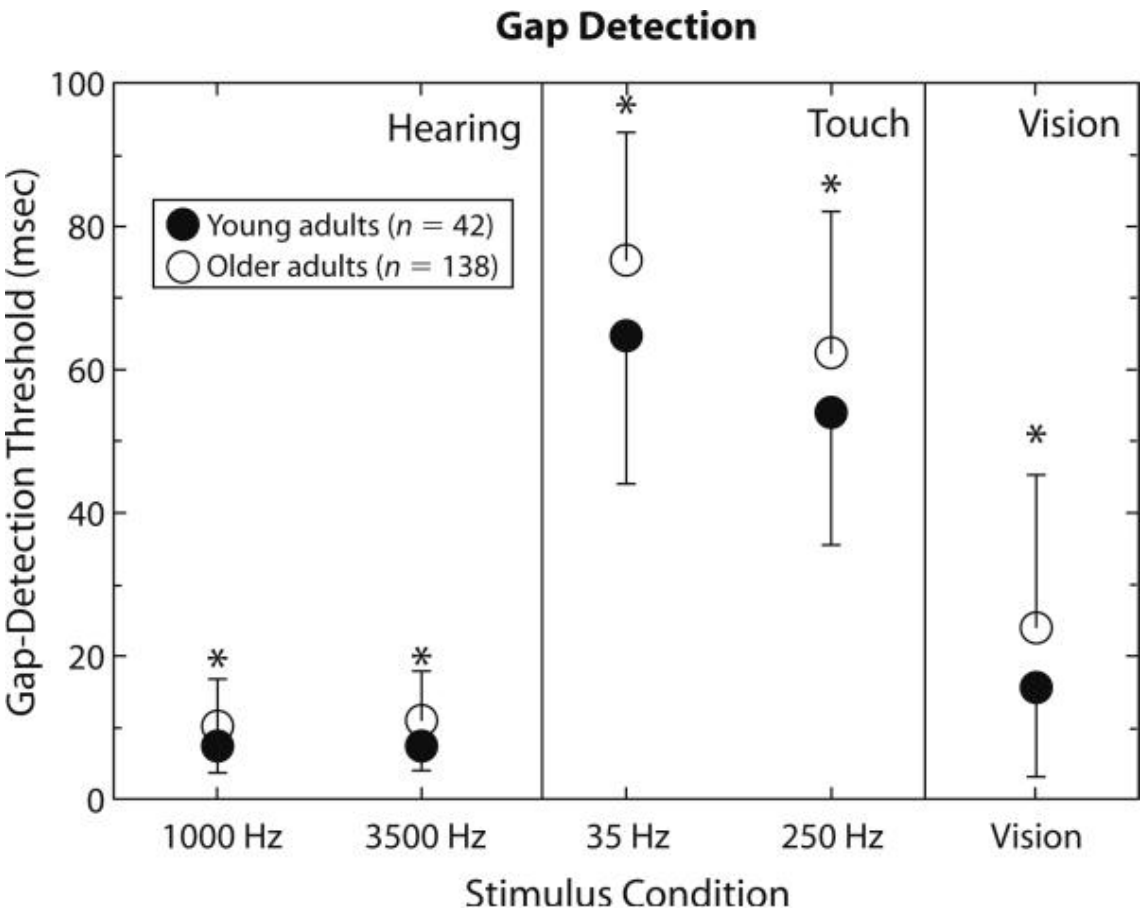


The camera is centred in the same position as the Oculus prefab and in the same direction, being child of the main anchor point for the cameras; it means that when the camera rotates with the headset, the same will do this camera. The camera is used to cast a raycast, and if it collides with an item in the world, we will know that that element has entered into the user's gaze.

The Gaze control flow chart will be the following



The Gaze control, as it was shown before in the application flow chart, is in continous communication with the manager, feeding the information of what the user is currently looking in a moment in the time. Throwing two raycast each frame is costly, but trying to reduce the frequency will turn in a loose of the answer time for the application, if we run an Update that runs in 60FPS constant, each 16.67 milliseconds we are going to be querying to the world and getting the information, but if we delay it to having less queries per millisecond, it will reduce the answer time, and when we are talking about time, we are only allowed to be slightly down 13,71ms (45 FPS), that is, as the results on (Humes et al., 2017) Figure 2 illustrates, the average population perception regarding to FPS, I wanted to have queries running all those frame rates and changing values in order to achieve a realistic sense of sight, reducing the chances of causing motion sickness due to the despairing of information and realism between the application sight and real sight



(Humes et al., 2017) Figure 2

When an object is picked through the gaze control (Pressing the A button of the gamepad), a reference to that Unity GameObject is given to the manager, allowing to make public to all the modules the information of it and changing the UI to the next flow step.

This is done by the function setObject, called in the GazeCursor class

```
public class Manager : MonoBehaviour
{
    public GameObject obj_in_use;
    public GameObject inspector_ui_;
    public GameObject mat_selection_ui_;
    public GameObject scene_selection_ui_;
    public GameObject prefab_selection_ui_;
    public GameObject keyboard_ui_;

    public UpdateCollector pos;
    public UpdateCollector rot;
    public UpdateCollector sca;

    public void setObject(GameObject obj)
    {
        obj_in_use = obj;
        pos.newValues(obj_in_use.transform.localPosition);
        rot.newValues(obj_in_use.transform.localRotation.eulerAngles);
        sca.newValues(obj_in_use.transform.localScale);
        keyboard_ui_.SetActive(true);
        inspector_ui_.SetActive(true);
        mat_selection_ui_.SetActive(true);
        inspector_ui_.SetActive(true);
        prefab_selection_ui_.SetActive(false);    }

}
```

This, was working perfectly with the elements in the world, but not the UI, the UI was not colliding with the UI elements. That was because the raycast in Unity is meant to be colliding with a component call “Colliders”, this component is a collision shape, used for collision detection in computer physics, when something enters into their bounding, is detect as a collision and depending on the pre-sets of the collider, can act as a trigger zone, in order to detect when a object is within a particular space in the game world, or a physic collision wall, making it stop, bounce, or all the possible physic behaviour that it can have scripted on it. Some research was made, and Oculus developer canter had an entry that explained the problem, the problem was not the UI and the raycasting, the problem was about the event system Unity has to notify the changes as the UI call-backs are called, short version, is as they state in their blog entry in (Borell, 2017), *“The short answer is that there’s no screen in VR, and therefore no visible surface for a mouse to move across.”.* They provided with a solution, but was late enough to remove all the UI work and modify the relationships between the modules, so a workaround was needed. The solution was converting each UI element into a game element, so the UI elements that were intractable thought the gaze control, had to be converted with a special format. They had to implement a “Collider” component, in order to be detected, and an interface, in order to standardize all the UI elements that were intractable, having the same functions implemented for the gaze to call. This is made by inheriting from the following interface.

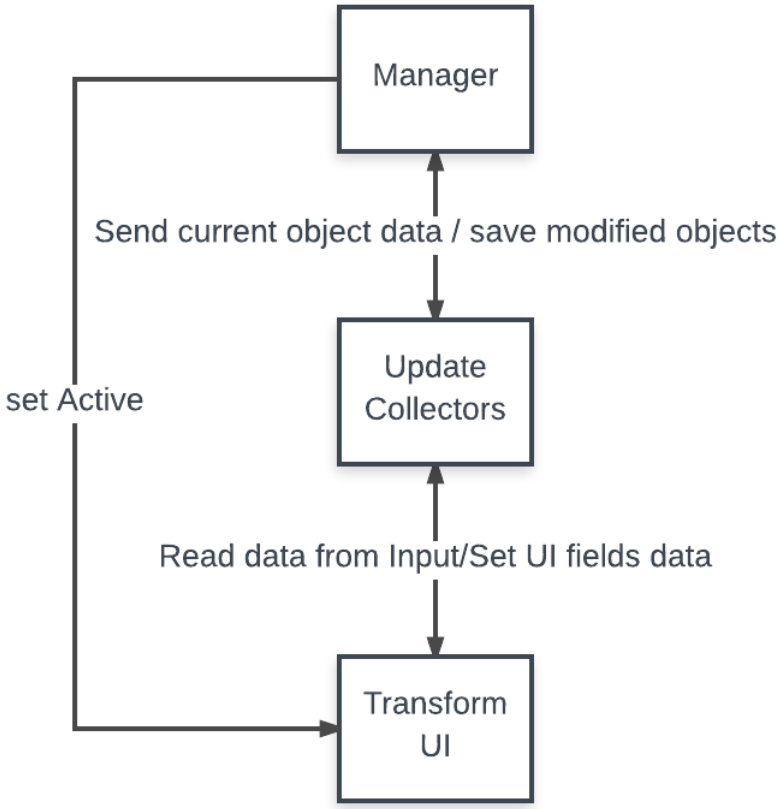
```
public interface IVRInteractuable {

    void onGazeIn();
    void onGazeOver();
    void onGazeOut();

    void action();
}
```

All the classes that inherit this, need to implement those methods, and if the user press “A” button of the XBOX360 controller when the object is in picked, or gazes in, into, or out the object, will call the pertinent function of those above, so functionality can be implemented.

This was the major problem while making the application, this taught that UI in virtual reality was not an easy task, and was way more different for a 2D based UI for a computer game, needs to be configured in different ways and also developed and coded, because the standards of VR and the way Unity is developed. Nevertheless, my experience in programming helped me to find a solution for this issue and keep the development in schedule and the project released by finding a workaround to this problem, relying in techniques used for 3D physic objects and convert them for UI elements.

Features check

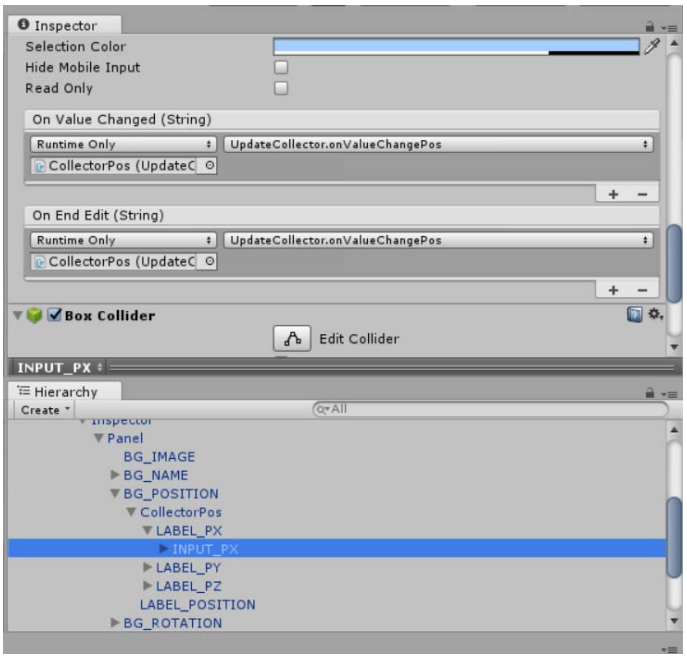
Transform inspector



The transform inspector is one of the key features of the most used game engine editors, it allows the user to modify and set the position, rotation and scale of an object in the world.

This is how it looks inside the app

It is composed by the three components that define the spatial representation for an object in an engine, the position, the rotation and the scale, each one of them can be presented as a 3 dimensional vector, the white boxes are an input element that allows the user to put values into each of the vectors that are transformed in code to values Unity can read and apply to the objects in the game. This task is done by the data collectors in the game, a script that is attached to each transform component in the inspector, allowing recording each input made by the keyboard UI. The functions in the script are bind to the function they need because Unity5 provides a call-back when the value has changed and when the user made focus into the input box and added value. The function binding is done in the editor,



```
public class UpdateCollector : MonoBehaviour
{
    //X, Y, Z values representing each coordinate of the vector
    public InputField x;
    public InputField y;
    public InputField z;

    //change position
    public void onValueChangePos(string temp)
    {
        float new_x = GetFloat(x.text, 0);
        float new_y = GetFloat(y.text, 0);
        float new_z = GetFloat(z.text, 0);
        if (Manager.getInstance().getObject() != null)
            Manager.getInstance().updateObjInUsePos(new Vector3(new_x, new_y,
new_z));
    }
    //cgabge rotation
    public void onValueChangeRot(string temp)
    {
        float new_x = GetFloat(x.text, 0);
        float new_y = GetFloat(y.text, 0);
        float new_z = GetFloat(z.text, 0);
        if (Manager.getInstance().getObject() != null)
            Manager.getInstance().updateObjInUseRot(new Vector3(new_x, new_y,
new_z));
    }
    //change scale
    public void onValueChangeScale(string temp)
    {
        float new_x = GetFloat(x.text, 0);
        float new_y = GetFloat(y.text, 0);
        float new_z = GetFloat(z.text, 0);
        if (Manager.getInstance().getObject() != null)
            Manager.getInstance().updateObjInUseSca(new Vector3(new_x, new_y,
new_z));
    }
    //put new values to the UI elements
    public void newValues(Vector3 v)
    {
        x.text = v.x.ToString();
        y.text = v.y.ToString();
        z.text = v.z.ToString();
    }

}
```

Each OnValueChange function is binded in one of the transform UI components, binding each of the input fields to their corresponding variable in the collector, they collect the data, update

the variables and then a new vector with the new status is created and given to the manager,

```
public class Manager : MonoBehaviour
{

    public UpdateCollector pos;

    public void updateObjInUsePos(Vector3 newpos)
    {
        if (!changelog.ContainsKey(obj_in_use.GetInstanceID()))
        {
            createEntryInChangelog(obj_in_use);
        }
        obj_in_use.transform.localPosition = newpos;
        changelog[obj_in_use.GetInstanceID()].t_x = newpos.x;
        changelog[obj_in_use.GetInstanceID()].t_y = newpos.y;
        changelog[obj_in_use.GetInstanceID()].t_z = newpos.z;
    }

}
```

There is one function per component in the transform, the change log is a feature of data exporting, this feature will be reviewed it in further sections.

With the editor working, this has proven to be a fine solution mixing and putting together UI and core logic, being able to accomplish one of the intended features of the editor.

Material change

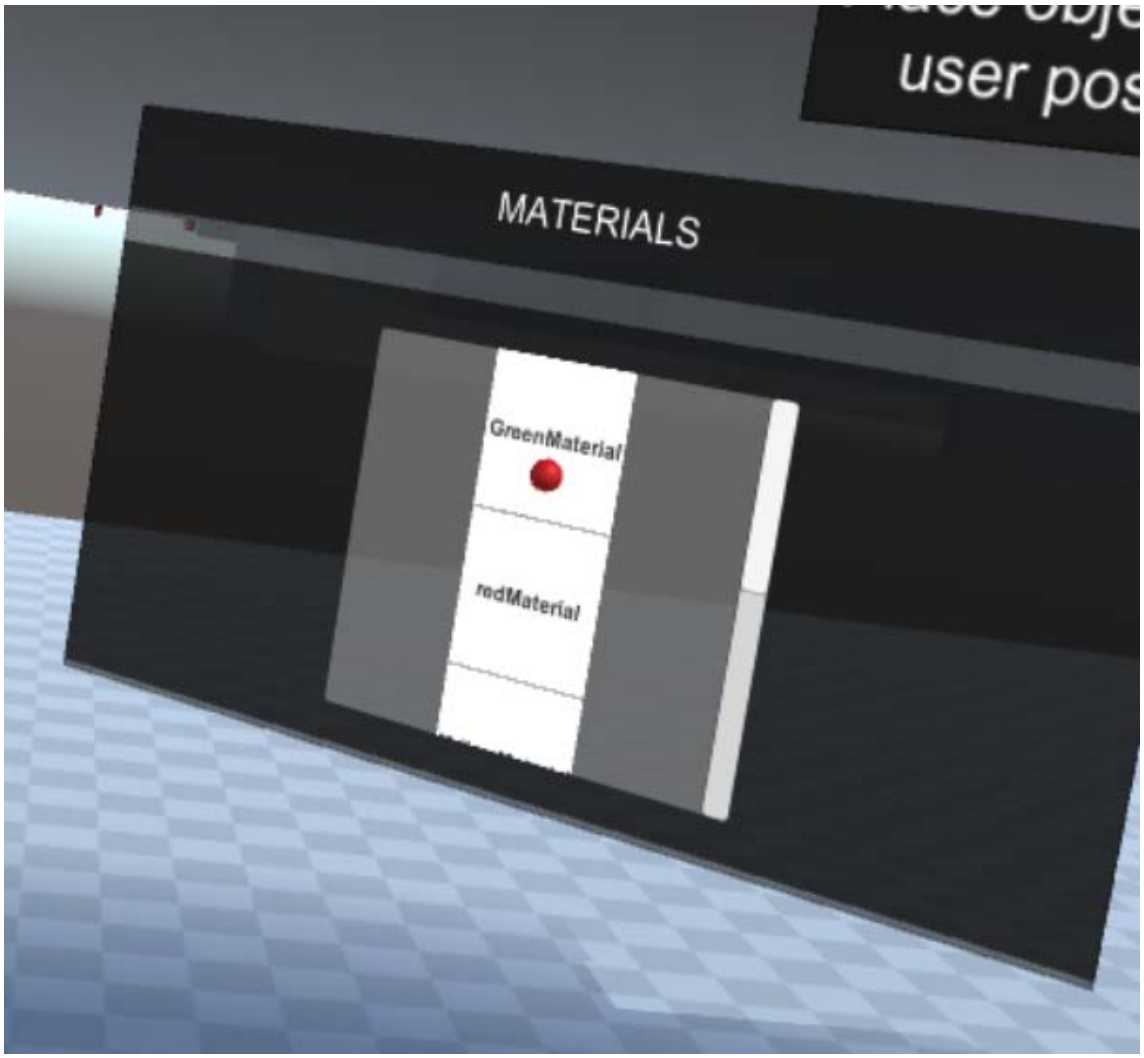
Rendering in Unity is done with Materials, Shaders and Textures. Materials are definitions of how a surface should be rendered, including references to textures used, tiling information, colour tints and more. The available options for a material depend on which shader the material is using. This created application editor is allowed to change materials of the elements in the world for one that has been loaded. The UI for selecting a material is a list with items. Unity 5 does not provide logic for a scroll list that hides items, one had to be created. We will explain the script that handles the items for the list now. The script is attached to a ScrollView Unity UI element, and has a reference to itself, a grid layout reference, in order to position the objects in a single row and for removing the items.

```
public class ScrollView : MonoBehaviour {
    public GameObject item;
    public GridLayoutGroup glgroup;
    public RectTransform scrollContent;
    public ScrollRect scroll;
    // if is a prefabs, material or scene list.
    public string listType;

    private void ClearOldElements()
    {
        for(int i = 0; i< glgroup.transform.childCount; i++) {
            Destroy(glgroup.transform.GetChild(i).gameObject);
        }
    }
    public void setContentHeight()
    {
        float scrollContentHeight = (glgroup.transform.childCount * glgroup.cellSize.y) +
((glgroup.transform.childCount - 1) * glgroup.spacing.y);
        scrollContent.sizeDelta = new Vector2(scrollContent.sizeDelta.x, scrollContentHeight);
    }
    public void InitializeList(string type)
    {
        ClearOldElements();
        if (type == "prefab"){
            foreach (KeyValuePair<string, GameObject> entry in
Manager.getInstance().prefab_dict)
            {
                InitializeNewItem(entry.Key, type);
            }
        }
        else if (type == "material")
        {
            foreach (KeyValuePair<string, Material> entry in Manager.getInstance().mat_dict)
            {
                InitializeNewItem(entry.Key, type);
            }
        }else if(type == "scenes")
        {
            foreach (string n in Manager.getInstance().scenes_dict)            {
                InitializeNewItem(n, type);
            }
        }
        setContentHeight();
    }
    private void InitializeNewItem(string name, string type)
    {
        GameObject newItem = Instantiate(item, glgroup.transform);
        newItem.name = name;
        newItem.transform.localPosition = Vector3.zero;
        newItem.transform.localRotation = Quaternion.Euler(Vector3.zero);
        newItem.transform.localScale = Vector3.one;
        newItem.GetComponent<Image>().enabled = true;
        string[] n = name.Split('/');
        newItem.GetComponentInChildren<Text>(true).text = n[n.Length - 1];
        newItem.GetComponentInChildren<Text>().enabled = true;
        newItem.GetComponentInChildren<ListItem>().scview = this;
        newItem.GetComponentInChildren<ListItem>().assetName = name;
        newItem.GetComponentInChildren<ListItem>().type = type;

        newItem.SetActive(true);
    }
    private IEnumerator MoveTowardsTarget(float time, float from, float target)
    {
        float i = 0;
        float rate = 1 / time;
        while (i < 1)
        {
            i += rate * Time.deltaTime;
            scroll.verticalNormalizedPosition = Mathf.Lerp(from, target, i);
            yield return 0;
        }
    }
}
```

The list starts on the method InitializeList, which is called when the assets are loaded in the game, depending on the type of the asset (Scenes, Materials or prefabs) it will start populating a list, with a button

```
public class ListItem : MonoBehaviour, IVRInteractuable
{

    public string assetName;
    public ScrollView scview;
    public GameObject ob_to_look_at;
    public string type;

    public void LateUpdate()
    {
        Vector2 vec2ScreenPoint = new Vector2(transform.position.x,
transform.position.y);
        if (scview)
        {
            if
(RectTransformUtility.RectangleContainsScreenPoint(scview.GetComponent<RectTransform
>(), vec2ScreenPoint)){
                GetComponent<BoxCollider>().enabled = true;
            }
            else{
                GetComponent<BoxCollider>().enabled = false;
            }
        }
    }
    //VRINTERACTUABLE FUNCTIONS
    public void onGazeIn(){}
    public void onGazeOver(){}
    public void onGazeOut(){}

    public void action()
    {
        if (type == "prefab"){
            Manager.getInstance().spawnObject(assetName);

        }
        else if (type == "material"){
            Manager.getInstance().changeMaterialToCurrentObject(assetName);

        }
        else if (type == "scenes"){
            Manager.getInstance().loadScene(assetName);
        }
    }
}
```

List items have this script attached, this allows this item to be intractable with the gaze cursor and it will call the action method when user presses “A” button if the item is being looked. When an item is created and pushed into the list, the ScrollView script sets up the name of the of the asset that it represents, which will be used by the action function in the ListItem script for either spawning a prefab, change a material or loading a scene.




```
public void changeMaterialToCurrentObject(string mat_name)
{
    obj_in_use.GetComponent<Renderer>().material = mat_dict[mat_name];
    if (!changelog.ContainsKey(obj_in_use.GetInstanceID()))
    {
        createEntryInChangelog(obj_in_use);
    }
    changelog[obj_in_use.GetInstanceID()].mat_name = mat_name;
}
```

Again, the manager manages the editing of the element in the map, as the transform, it searches for a material in the material asset map that has been loaded at the start of the application. If the object does not exist in the changelog, it adds the object to it, for the exporting script to know that the object has been modified, if it exists, just changes the material.

The material selector is a feature that was planned for the editor as is a basic element for an editor and proves useful to change materials in the map in an effective and easy way.

Being able to move in the map editing.

The movement is done by a script that modifies the position of the user GameObject and ensures that is only one movement script (for the player) in the world at a time by using a Singleton pattern.

In the Update method, the axes that are being pressed are monitored, the application has two types of movement and a looking feature.

Fly function makes the user able to fly onwards and backwards. Strafe function allows moving right and left. Look uses the mouse input to turn the object. When using rotations, we want to avoid what is called gimbal lock, explained in Hoag, D. (1963) as "*Gimbal lock occurs when the outer gimbal axis is carried around by vehicle motion to be parallel to the inner gimbal axis. At this trivial point the three gimbal axes lie in a single plane. No gimbal freedom now exists to "unwind" base motion about an axis normal to this plane. Even though any vehicle orientation with respect to the stable member can be accommodated by particular sets of the three gimbal angles the condition at gimbal lock prevents accommodation of a particular orientation change from the locked condition.",* this will lock as said the rotation of the object, to prevent this, we want to clamp the position before it gets into the case that the outer gimbal (World rotation) is parallel to the inner gimbal (Local rotation).

```
       // Update is called once per frame
    void Update () {
        if (Input.GetAxis("Horizontal") < 0){
            Strafe(-keySpeed * Time.deltaTime);
        }else if (Input.GetAxis("Horizontal") > 0){
            Strafe(keySpeed * Time.deltaTime);
        }
        if (Input.GetAxis("Vertical") > 0){
            Fly(keySpeed * Time.deltaTime);
        }
        else if (Input.GetAxis("Vertical") < 0){
            Fly(-keySpeed * Time.deltaTime);
        }
        float dx = Input.GetAxis("RightHorizontal");
        float dy = Input.GetAxis("RightVertical");
        Look(new Vector2(dx, dy) * mouseSpeed);
    }
    void Strafe(float dist){
        transform.position += eye.transform.right * dist;
    }
    void Fly(float dist){
        transform.position += eye.transform.forward * dist;
    }
    void Look(Vector2 dist){
        angle += dist;
        angle.x = ClampAngle(angle.x, minAngle.x, maxAngle.x);
        angle.y = ClampAngle(angle.y, minAngle.y, maxAngle.y);
        Quaternion quatX = Quaternion.AngleAxis(angle.x, Vector3.up);
        Quaternion quatY = Quaternion.AngleAxis(angle.y, -Vector3.right);
        transform.localRotation = originalRotation * quatX * quatY;
    }
    float ClampAngle(float angle, float min, float max)
    {
        if (angle < -limit){
            angle += limit;
        }
        else if (angle > limit){
            angle -= limit;
        }
        return Mathf.Clamp(angle, min, max);
    }
}
```

Placing assets in the map.

Placing assets is done by the manager. This allows the application to have full control over the asset creation in the map. The function that allows spawning is the following.

```
public void spawnObject(string obj)
{
    GameObject go = Instantiate(prefab_dict[obj]);
    removeObject();
    setObject(go);
    createEntryInChangelog(go);

    // name =! string.empty == new object
    changelog[go.GetInstanceID()].obj_name = obj;
}
```

Again, as all the other configuration that needs to be exported, it uses the changelog. When an object is created, is instantly added to the changelog.

It uses the same list system as the scenes and the materials, meaning you can have a high number of elements loaded in your application in the list.

The gaze cursor will handle all the action of activating the spawning, the user needs to put the gaze cursor in the list item it want to spawn and then press the action button, it will spawn in the world, therefore adding it to the new items in world for exporting.

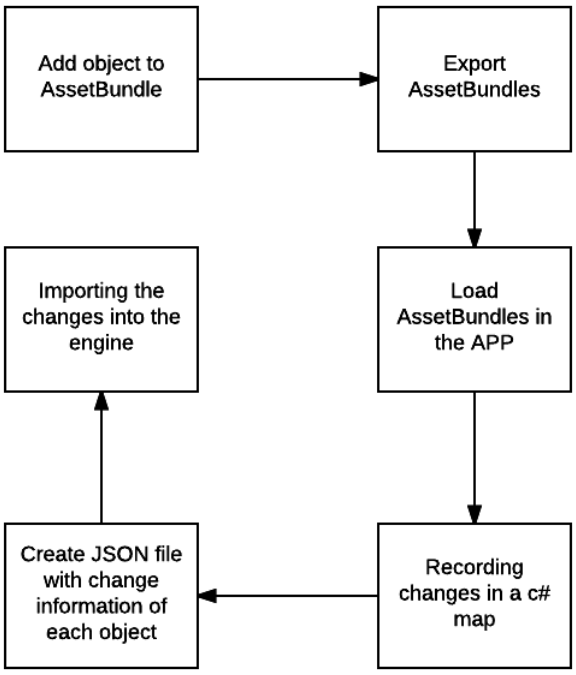
Load scenes and assets in the application, exporting and importing changes

Structure

This is the module that will allow the user to use the assets, scenes and materials of their own into the virtual reality editor, exporting the changes made into the scene into a JSON file and importing that file into the unity editor.

This module makes use of Unity Assetbundles. As they state in Technologies, U. (2017), *"Asset Bundles are a collection of assets, packaged for loading at runtime. With Asset Bundles, you can dynamically load and unload new content into your application."* This is used in the

application for loading the data we need, this is the flow chart this module follows.



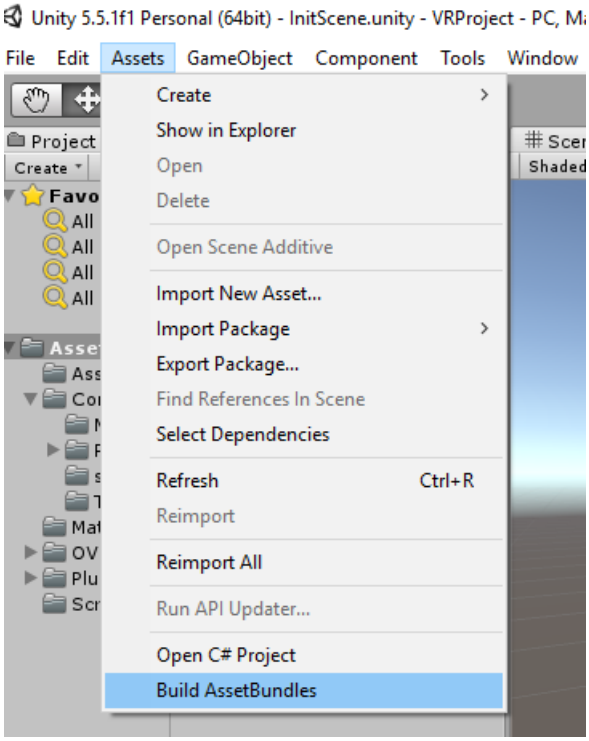
The process is going to be review in the next paragraphs.

Assetbundle creation, exporting and loading.

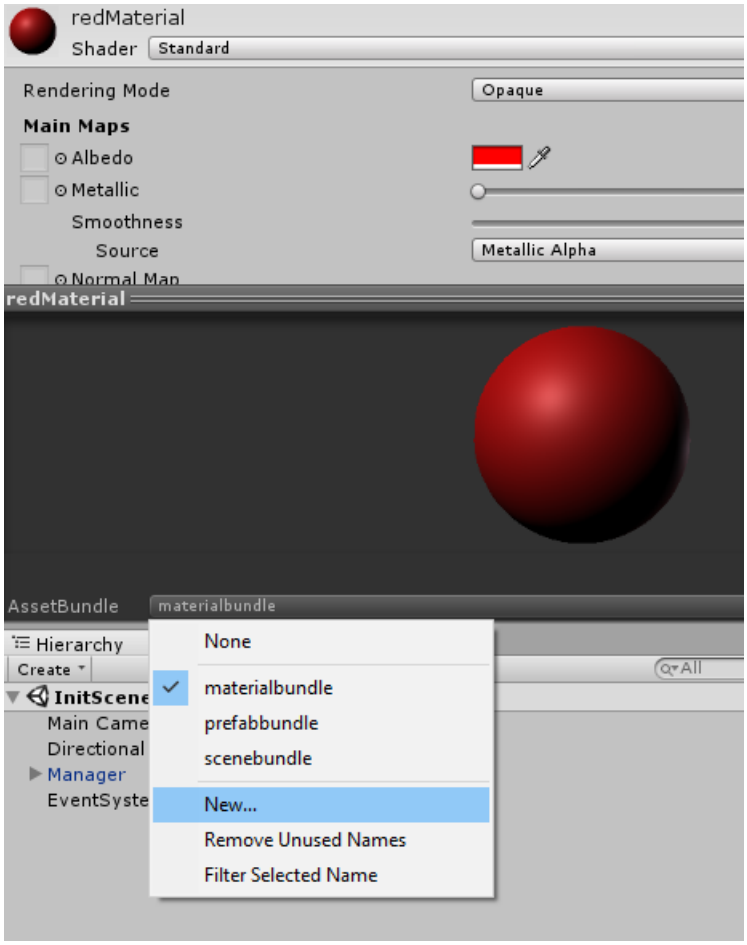
First the creation of an editor script is needed, this scripts differs from the normal object scripts because they affect the editor pipeline and functions, and are able to create new functions, in this script, a new menu item is created.

```
#if UNITY_EDITOR
using UnityEditor;

public class CreateAssetBundles
{
    [MenuItem("Assets/Build AssetBundles")]
    static void BuildAllAssetBundles()
    {
        BuildPipeline.BuildAssetBundles("Assets/AssetBundles",
BuildAssetBundleOptions.None, BuildTarget.StandaloneWindows64);
    }
}

#endif
```

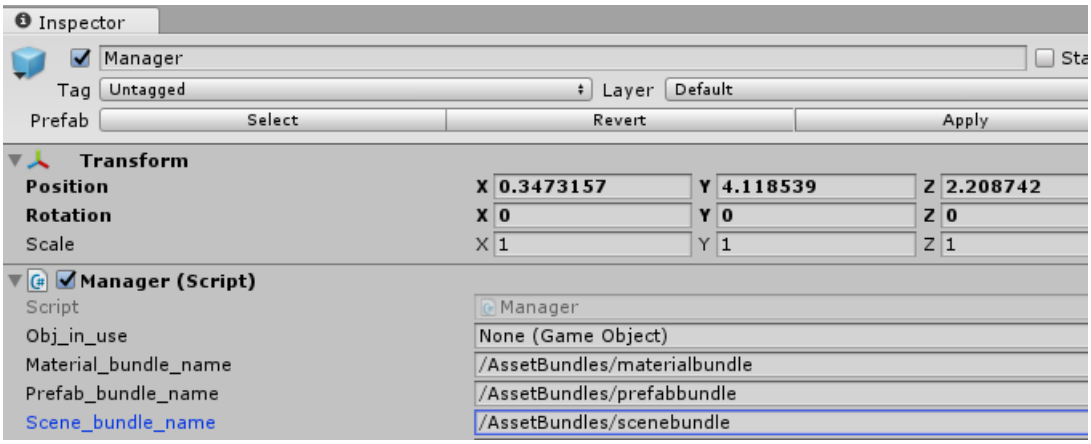
This script creates the following menu item.



When this menu item is pressed, Unity will start creating the Assetbundle files and storing them into the designated position by the script, this will allow also starting adding assets into an Assetbundle



In this application, the number of Assetbundles are specified by the application and have their own variables public in the inspector for filling, this is because a very specific and simple way to exporting the items was intended.



The following bundle uses are the following:

- Material_Bundle_Name: As the name states, the path here must lead to an Assetbundle with materials on it.
- Prefab_Bundle_Name: As the name states, the path here must lead to an Assetbundle with prefabs on it.
- Scene_Bundle_Name: As the name states, the path here must lead to an Assetbundle with scenes on it.

Once the Assetbundles are compiled and packaged and the variables in the manager are initialized, the bundles are loaded in the start process of the application, this is done in the manager and is made in Unity Coroutines.

```
    void Awake()
    {
        if (instance == null)
        {
            Caching.CleanCache();
            DontDestroyOnLoad(this);
            prefab_dict = new Dictionary<string, GameObject>();
            mat_dict = new Dictionary<string, Material>();
            changelog = new Dictionary<int, JsonData>();
            StartCoroutine(LoadAssetBundleOnApp(material_bundle_name,
"material"));
            StartCoroutine(LoadAssetBundleOnApp(prefab_bundle_name, "prefab"));
            StartCoroutine(LoadAssetBundleOnApp(scene_bundle_name, "scenes"));
            instance = this;
            obj_in_use = null;
        }
        else if (instance != this)
        {
            Destroy(gameObject);
        }
    }
```

A Unity Coroutine is a function that has the ability to pause execution and return control to Unity but then to continue where it left off on the following frame. Is similar to a fake multithreading function, it starts the function, then stops when it is out of time (Decided by Unity internal clock), and then resuming the state in the same place the function stopped the execution.

This is the function in the manager in charge of loading an Assetbundle in the application.

```
IEnumerator LoadAssetBundleOnApp(string file, string type)
    {
        string path = "File://" + Application.dataPath + file;
        WWW www = WWW.LoadFromCacheOrDownload(path, 2);

        print("Loading");

        //waiting for completion
        yield return www;
        if (!string.IsNullOrEmpty(www.error))
        {
            Debug.LogError(www.error);
            yield return null;
        }

        myLoadedBundle = www.assetBundle;

        if (type == "prefab")
        {
            GameObject[] loadedObjs_obj =
myLoadedBundle.LoadAllAssets<GameObject>();
            for (int i = 0; i < loadedObjs_obj.Length; i++)
            {
                prefab_dict.Add(loadedObjs_obj[i].name, loadedObjs_obj[i]);
            }
            prefabDataShowing.InitializeList(type);

        }
        else if (type == "material")
        {
            Material[] loadedObjs_obj = myLoadedBundle.LoadAllAssets<Material>();
            for (int i = 0; i < loadedObjs_obj.Length; i++)
            {
                mat_dict.Add(loadedObjs_obj[i].name, loadedObjs_obj[i]);
            }
            materialDataShowing.InitializeList(type);

        }
        else if (type == "scenes")
        {
            scenes_dict = myLoadedBundle.GetAllScenePaths();
            sceneDataShowing.InitializeList(type);
        }

        www.Dispose();
    }
```

this function searches for an Assetbundle in the path provided, and stores the data retrieved from the retrieval into different C# Dictionaries, each map stores all the data from one Assetbundle, for example, the prefab dictionary stores the objects by asset name and then, the

prefab object for instancing a copy. Once the Dictionaries are filled, they are ready to be used by the UI elements or the spawning actions.

Modified objects, changelog

Once an object is modified, we add it into the changelog. The changelog is a C# dictionary storing all the needed information for an object in order to save its current status as the data, while the unique object ID from Unity is the key. The data structure is the following.

```
[System.Serializable]
public class JsonData
{
    public int objectID;

    //old transform position information, for deleting
    //the object when importing
    public float old_x;
    public float old_y;
    public float old_z;

    //current position
    public float t_x;
    public float t_y;
    public float t_z;

    //Current rotation
    public float r_x;
    public float r_y;
    public float r_z;

    //current scale
    public float s_x;
    public float s_y;
    public float s_z;

    //material name and object name
    public string mat_name;
    public string obj_name;
}
```

Each time an object is created, a new entry in the changelog is created, storing basic information of the object, this function is in the manager as well, having all the loading and exporting logic in the same script, allowing an easier control. A new entry will be created too if we modify an existing object in the world.

```
private void createEntryInChangelog(GameObject obj)
{
    JsonData temp = new JsonData();
    changelog[obj.GetInstanceID()] = temp;

    temp.old_x = obj.transform.position.x;
    temp.old_y = obj.transform.position.y;
    temp.old_z = obj.transform.position.z;

    temp.t_x = obj.transform.position.x;
    temp.t_y = obj.transform.position.y;
    temp.t_z = obj.transform.position.z;

    temp.r_x = obj.transform.rotation.x;
    temp.r_y = obj.transform.rotation.y;
    temp.r_z = obj.transform.rotation.z;

    temp.s_x = obj.transform.localScale.x;
    temp.s_y = obj.transform.localScale.y;
    temp.s_z = obj.transform.localScale.z;

    temp.mat_name = string.Empty;
    temp.obj_name = string.Empty;
}
```

Apart from creating entries, we need to modify those entries in the changelog, when the transform editor was reviewed, there are functions which are in charge of updating the transform status of an object.

```
public void updateObjInUsePos(Vector3 newpos)
{
    obj_in_use.transform.position = newpos;
    if (!changelog.ContainsKey(obj_in_use.GetInstanceID()))
    {
        createEntryInChangelog(obj_in_use);
    }
    changelog[obj_in_use.GetInstanceID()].t_x = newpos.x;
    changelog[obj_in_use.GetInstanceID()].t_y = newpos.y;
    changelog[obj_in_use.GetInstanceID()].t_z = newpos.z;
}
```

There is one function of this type for scale, rotation, position and material, and their job is, apart from updating the object status, creating or updating the object entry in the changelog, ensuring all the data for exporting is correct.

Data to JSON

In the first stages of the application, JSON and XML were considered. JSON (JavaScript Object Notation) is a lightweight format used for data interchanging and the format is specified in (Rockford, 2017). XML is, as stated in (W3.org, 2017), *"Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere."* One of those had to be selected, and after doing some research, JSON was the selected format, in (Haq, Khan and Hussain, n.d: 5), they say that XML seemed to use less user CPU using than JSON but on the

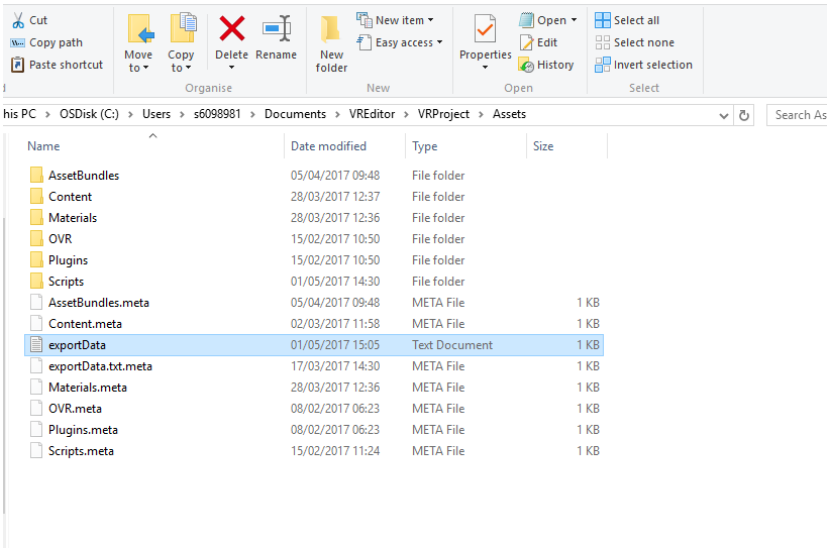
other hand, it had more system CPU, whereas the memory remains very similar, in conclusion, the efficacy is similar. The difference was made by the point (Haq, Khan and Hussain, n.d: 5) stated: *"Data format of JSON is very simple that can be transmitted with a single array, variable of number or Boolean type or also string type"* . The key point was simplicity, JSON standard, compared to XML is much readable and comprehensive.

Once the researched was concluded, the development started. Unity does not provide a serialization for JSON items, so one had to be made. With some research, a template that could be used was found on this thread of StackOverflow , as an entry point, it provided all the needed that to start developing.

```
public class JSonHelper : MonoBehaviour {

    public static T[] FromJson<T>(string json)
    {
        Wrapper<T> wrapper = JsonUtility.FromJson<Wrapper<T>>(json);
        return wrapper.Items;
    }
    public static string ToJson<T>(T[] array)
    {
        Wrapper<T> wrapper = new Wrapper<T>();
        wrapper.Items = array;
        return JsonUtility.ToJson(wrapper);
    }
    public static string ToJson<T>(T[] array, bool prettyPrint)
    {
        Wrapper<T> wrapper = new Wrapper<T>();
        wrapper.Items = array;
        return JsonUtility.ToJson(wrapper, prettyPrint);
    }
    [System.Serializable]
    private class Wrapper<T>
    {
        public T[] Items;
    }

}
```

This class, not attached to any script, allows the application to convert from JSON to an specified object and vice versa, this is thanks the using of C# templates, C# templates are, as (Msdn.microsoft.com, 2017) states in their webpage: "*type-safe data structures, without committing to actual data types. This results in a significant performance boost and higher quality code, because you get to reuse data processing algorithms without duplicating type-specific code*".

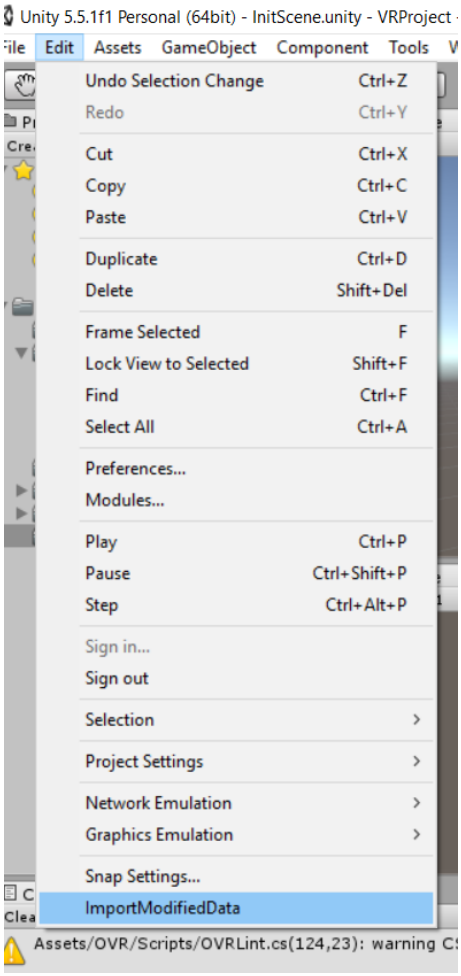
The changelog is converted through the "ToJson" function, providing a new string with all the Json items, this string is written into disk



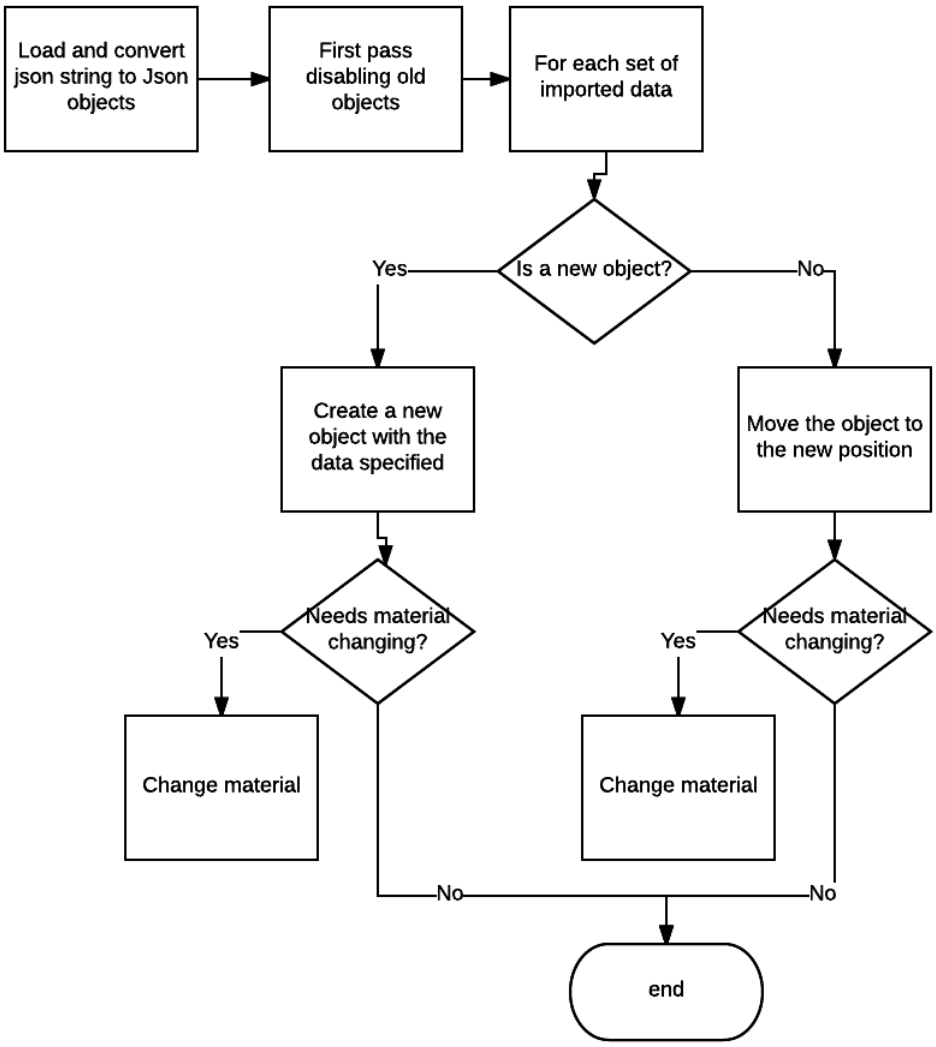
The data will be created under “APPFolder/VREditorApp_Data” and

Importing data to the engine

After the information is exported, in order to continue development, there is a need to import those changes back into Unity 5, for that, I decided to add an import option for my tool into unity as a new menu option, with an editor script, then adding the functionality to read the exported file and implement the changes in the map.



When this menu script is executed, it follows the next flow chart.



When the script ends, the changes in the map made in the application will be in the scene opened in Unity. The export file needs to be placed inside the “Assets” folder of the project, the editor is going to search for it in that path.

This proved to be a long succession of functionality one linked to another, was far long that the expectations, but the functionality was created successfully, therefore completing a mandatory feature needed for the application. The big number of relationships and complex structure taught me things I will need in my future as a Unity developer, as well as programming code structure for applications.

Conclusion

Now the project is finished, it proved to be a valuable experience for learning more about virtual reality and Unity engine, though the conditions were not the best as they could be and the planned idea with the input was not achievable, new ideas needed to be considered and that taught me about having to make quick choices in the middle of a development process, which for sure will prove to be useful in my professional career. Also, the feeling with the UI is not the expected, it sometimes feel very uncomfortable, that`s due the poor testing time I had to test the look and feel of the game and the little number of people I had to test the application. A virtual reality application needs to be tested both the integrity part and the comfort part, because a poor design in UI and character feel can lead to an application that tends to produce motion sickness into the user, making the application unusable, if I needed to make this application again, I would gather a number of testers for the application and instead of trying to test the application in the end, I would have them test each movement and UI module separated each time one is developed, first the module alone to get the problems with the module, and then with all the other modules gathered together, to get integration problems and how well integrated are the movement modules with the UI work. Nevertheless, this project provided me with the experience of having to develop from scratch a high level modular application, having to plan and develop a big structure, making me to learn about data structures, project design and how to structure a project with a centralized structure.

The features of the project were achieved in an artefact that has no errors in the execution and accomplishes and covers all the needs for a simple world editor for handing it to artists and designers, it allows perfectly to exempt the user from the engine pipeline and provide a much simpler tool for developing virtual reality worlds.

References

Allvirtualreality.com. (2017). *Citar un sitio web - Cite This For Me*. [online] Available at: http://allvirtualreality.com/Image/NewsImg/20160208-UE4/Unreal-Engine-4-VR.jpg [Accessed 25 Apr. 2017].

Msdn.microsoft.com. (2017). *An Introduction to C# Generics*. [online] Available at: https://msdn.microsoft.com/en-us/library/ms379564(v=vs.80).aspx [Accessed 1 May 2017].

Borell, A. (2017). *Unity's UI System in VR*. [online] Developer3.oculus.com. Available at: https://developer3.oculus.com/blog/unitys-ui-system-in-vr/ [Accessed 29 Apr. 2017].

W3.org. (2017). *Extensible Markup Language (XML)*. [online] Available at: https://www.w3.org/XML/ [Accessed 1 May 2017].

Developer3.oculus.com. (2017). *Developer Center — Documentation and SDKs | Oculus*. [online] Available at: https://developer3.oculus.com/documentation/intro-vr/latest/concepts/bp_intro/ [Accessed 25 Apr. 2017].

Haq, Z., Khan, G. and Hussain, T. (n.d.). *A Comprehensive analysis of XML and JSON web technologies*. [online] www.inase.org. Available at: http://www.inase.org/library/2015/vienna/bypaper/CSSCC/CSSCC-14.pdf [Accessed 1 May 2017].

Hoag, D. (1963). *Apollo IMU Gimbal Lock*. [online] Hq.nasa.gov. Available at: https://www.hq.nasa.gov/alsj/e-1344.htm [Accessed 30 Apr. 2017].

Humes, L., Busey, T., Craig, J. and Kewley-Port, D. (2017). *The effects of age on sensory thresholds and temporal gap detection in hearing, vision, and touch*.

Nhs.uk. (2017). *Motion sickness - NHS Choices*. [online] Available at: http://www.nhs.uk/conditions/Motion-sickness/Pages/Introduction.aspx [Accessed 25 Apr. 2017].

Roadtovr.com. (2017). *Citar un sitio web - Cite This For Me*. [online] Available at: http://www.roadtovr.com/wp-content/uploads/2016/12/unity-editorvr.jpg [Accessed 25 Apr. 2017].

Rockford, D. (2017). *RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)*. [online] Tools.ietf.org. Available at: https://tools.ietf.org/html/rfc4627 [Accessed 1 May 2017].

Technologies, U. (2017). *Unity - Manual: Asset Bundles*. [online] Docs.unity3d.com. Available at: https://docs.unity3d.com/Manual/AssetBundlesIntro.html [Accessed 30 Apr. 2017].

Technologies, U. (2017). *Unity - Manual: Input Manager*. [online] Docs.unity3d.com. Available at: https://docs.unity3d.com/Manual/class-InputManager.html [Accessed 26 Apr. 2017].

Technologies, U. (2017). *Unity - Manual: UI*. [online] Docs.unity3d.com. Available at: https://docs.unity3d.com/Manual/UISystem.html [Accessed 26 Apr. 2017].

Technologies, U. (2017). *Unity - Scripting API:*. [online] Docs.unity3d.com. Available at: https://docs.unity3d.com/ScriptReference/ [Accessed 22 Apr. 2017].

Technologies, U. (2017). *Unity - Scripting API: MonoBehaviour.Awake()*. [online] Docs.unity3d.com. Available at: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html [Accessed 25 Apr. 2017].

Virtual Reality Society. (2017). *What is Virtual Reality? - Virtual Reality*. [online] Available at: https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html [Accessed 25 Apr. 2017].

Wiki.unity3d.com. (2017). *Xbox360Controller - Unify Community Wiki*. [online] Available at: http://wiki.unity3d.com/index.php?title=Xbox360Controller [Accessed 26 Apr. 2017].