A project report on

DYNAMIC SURFACE AR

Submitted in partial fulfilment for the award of degree of

Bachelor's of Technology (B.Tech)

By

Amogh Dubey (16BEC0577)

SCHOOL OF ELECTRONICS ENGINEERING

(SENSE)

JUNE, 2020

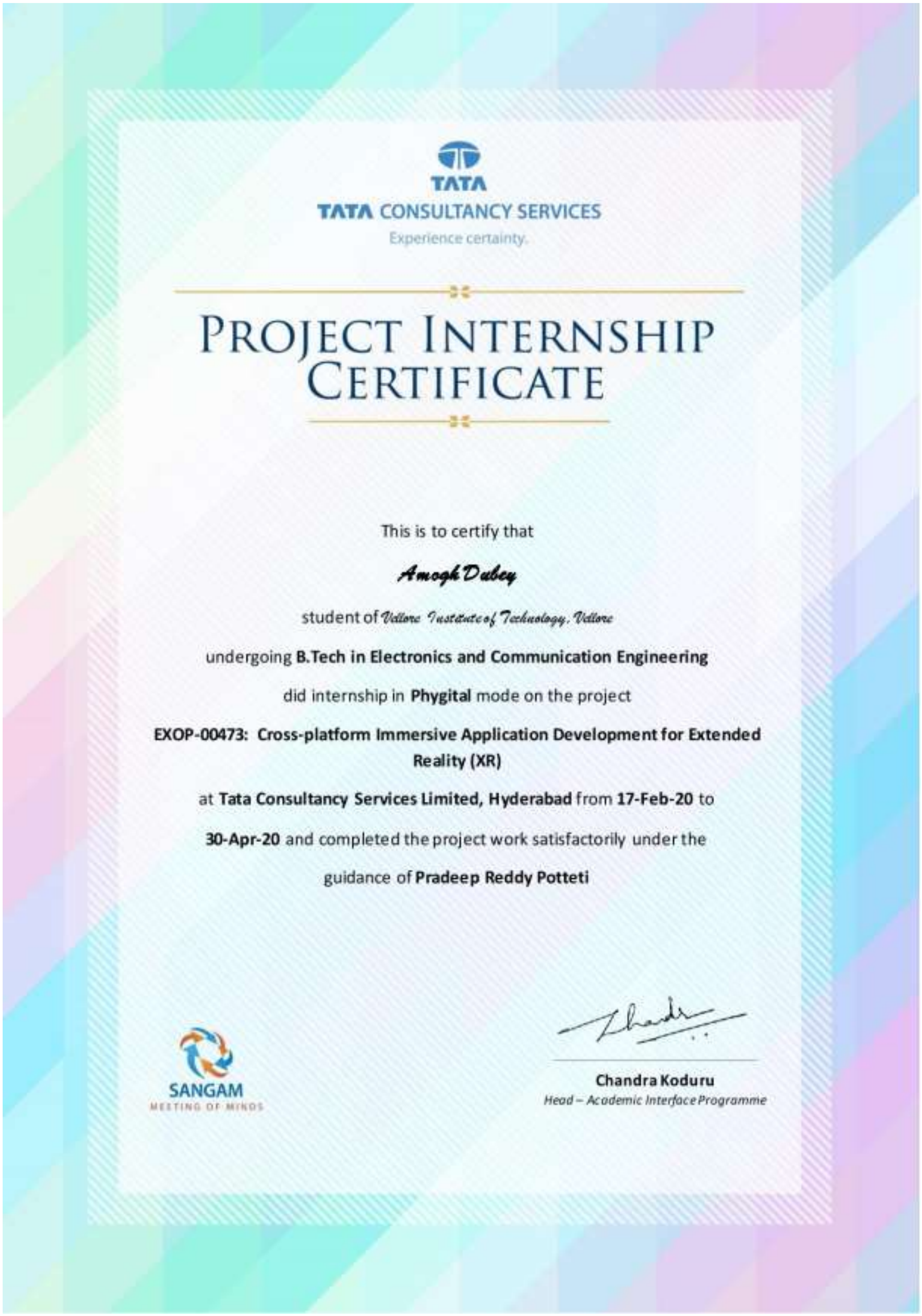
DECLARATION

I hereby declare that the thesis entitled **"DYNAMIC SURFACE AR"** submitted by me, for the award of the degree of Bachelor's of Technology (B.Tech) VIT is a record of bonafide work carried out by me under the supervision of my industry mentor Mr. Pradeep Reddy Potteti.

I further declare that the work submitted in this thesis has not been submitted and will not be submitted, either in part or full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 4th June 2020 Signature of Candidate



TATA

TATA CONSULTANCY SERVICES

Experience certainty.

PROJECT INTERNSHIP CERTIFICATE

This is to certify that

Amogh Dubey

student of *Vellore Institute of Technology, Vellore*

undergoing **B.Tech in Electronics and Communication Engineering**

did internship in **Phygital** mode on the project

EXOP-00473: Cross-platform Immersive Application Development for Extended Reality (XR)

at **Tata Consultancy Services Limited, Hyderabad** from **17-Feb-20** to **30-Apr-20** and completed the project work satisfactorily under the guidance of **Pradeep Reddy Potteti**

SANGAM

MEETING OF MINDS

[illegible]

Chandra Koduru

Head – Academic Interface Programme

ABSTRACT

Augmented reality has created quite a stir and many consumers are wondering how this latest technology will actually have an impact on them as it becomes more widely used. For anyone that uses a mobile device for daily activities, augmented reality is the future that will allow consumers to experience a reality that is based on personal needs and desires. AR will present a completely new way to engage and will expand the abilities of retailers as well. The possibilities of augmented reality are literally endless and when combined with et technology of mobile devices, which is already quite powerful, AR will allow for geo-tracking that will allow amazing experiences for consumers.

Augmented reality will also have a strong impact on society. The marketing and advertisement fields will explode with augmented reality devices. The mobile applications that are being developed will offer facial recognition software that will mainstream quickly. For everyday commuters and drivers, navigation devices will be built into the cars and mobile devices. This is already in the works, with some car manufacturers working to implement augmented reality windshields that will help drivers navigate without taking their eyes off the road.

Considering this, and as per my mentor's provided problem statement, I developed an application for creating augmented reality content on dynamic surfaces i.e., surfaces which are moving and thus provide a means for the implementation of augmented reality in real life scenarios, where virtual objects can be successfully utilised for the day to day activities of any commoner.

ACKNOWLEDGEMENT

I would like to express my gratitude to Dr. G Vishwanathan, Mr. Sankar Vishwanthan, Dr. Sekar Vishwanathan, Mr. G V Selvam, Dr. Anand A Samuel, Dr. S Narayanan and Dr. Kittur Harish Mallikarjun, SENSE, for providing me with an environment to work in and for the inspiration during the tenure of this course.

In jubilant mood I express my whole hearted thanks to Dr. R Sujatha, all teaching staff and members working as limbs of our university for their non-self-centered enthusiasm coupled with timely encouragements showered on me with zeal which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. Last but not the least, I express my gratitude and appreciation to all those who have helped me directly or indirectly towards the successful completion of this project.

Place: Vellore

Date: 4th June 2020 Amogh Dubey

Contents

1. AUGMENTED REALITY (AR) AND EXTENDED REALITY (XR)

Augmented reality overlays digital content and information onto the physical world — as if they're actually there with you, in your own space. AR opens up new ways for your devices to be helpful throughout your day by letting you experience digital content in the same way you experience the world. It lets you search things visually, simply by pointing your camera at them. It can put answers right where your questions are by overlaying visual, immersive content on top of your real world. The primary value of augmented reality is the manner in which components of the digital world blend into a person's perception of the real world, not as a simple display of data, but through the integration of immersive sensations, which are perceived as natural parts of an environment. Commercial augmented reality experiences were first introduced in entertainment and gaming businesses. Subsequently, augmented reality applications have spanned commercial industries such as education, communications, medicine, and entertainment. In education, content may be accessed by scanning or viewing an image with a mobile device or by using marker less AR techniques. The implementation of augmented reality in consumer products requires considering the design of the applications and the related constraints of the technology platform. Since AR systems rely heavily on the immersion of the user and the interaction between the user and the system, design can facilitate the adoption of virtuality.

2. PROBLEM STATEMENT

In current implementations of Augmented Reality and all the applications present, most depend on static surfaces to produce the virtual objects that can be viewed in the extended reality environment. But in real life not every object that we view is static or stationary. So, to tackle this situation and to improve upon existing technologies for the future, we need a more flexible system, one which can operate on dynamic surfaces as well and give a much more real time experience to the users.

Therefore, the problem statement that we were meant to tackle was:

"To produce a solution for providing Augmented Reality content in real time basis, for dynamic surfaces which are mobile and produce an application implementing the same."

3. APPROACH TO THE PROBLEM

Considering the fact that augmented reality is most experienced by handheld devices or more specifically smartphones, we chose to work upon palm detection and hand gestures for our approach to the solution.

Hand detection is a crucial pre-processing procedure for many human hand related computer vision tasks, such as hand pose estimation, hand gesture recognition, human activity analysis, and so on. However, reliably detecting multiple hands from cluttering scenes remains to be a challenging task because of complex appearance diversities of dexterous human hands. To tackle this problem, an accurate hand detection method is proposed to reliably detect multiple hands from a single colour image using a hybrid detection/reconstruction convolutional neural networks (CNN) framework, in which regions of hands are detected and appearances of hands are reconstructed in parallel by sharing features extracted from a region proposal layer, and the proposed model is trained in an end-to-end manner. All this and simultaneous frame by frame strong point detection in real time basis was only possible with the help of OpenCV. OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. OpenCV's application areas include:

1. 2D and 3D feature toolkits
2. Egomotion estimation
3. Facial recognition system
4. Gesture recognition
5. Human–computer interaction (HCI)
6. Mobile robotics
7. Motion understanding
8. Object identification
9. Segmentation and recognition
10. Structure from motion (SFM)
11. Motion tracking
12. Augmented reality

We decided to take the course of developing our application with the help of OpenCV for the desired output. For this purpose, we took the developmental environment of Unity3D to help in creating the said application for our problem.

Unity3D is a powerful cross-platform 3D engine and a user-friendly development environment. Easy enough for the beginner and powerful enough for the expert; Unity should interest anybody who wants to easily create 3D games and applications for mobile, desktop, the web, and consoles.

Furthermore, Unity3D provides us with a brilliant framework to work with multiple augmented reality APIs in the name of AR Foundation. AR Foundation is a set of MonoBehaviours and APIs for dealing with devices that support following concepts:

- World tracking: track the device's position and orientation in physical space.
- Plane detection: detect horizontal and vertical surfaces.
- Point clouds, also known as feature points.
- Reference points: an arbitrary position and orientation that the device tracks.
- Light estimation: estimates for average colour temperature and brightness in physical space.
- Environment probes: a means for generating a cube map to represent a particular area of the physical environment.
- Face tracking: detect and track human faces.
- Image tracking: detect and track 2D images.
- Object tracking: detect 3D objects.

4. SOLUTION AND APPLICATION

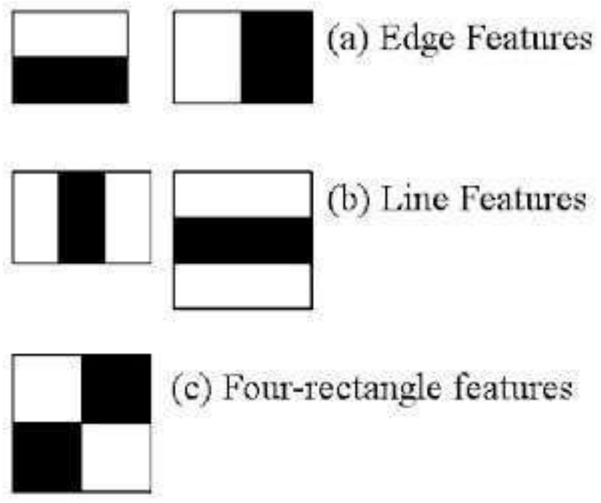
For the development of this particular application we went for Haar Cascade Classifiers. Haar Cascade is a machine learning object detection algorithm used to identify objects in an image or video and based on the concept of features proposed by Paul Viola and Michael Jones in their paper "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.

It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

The algorithm has four stages:

1. Haar Feature Selection
2. Creating Integral Images
3. Adaboost Training
4. Cascading Classifiers

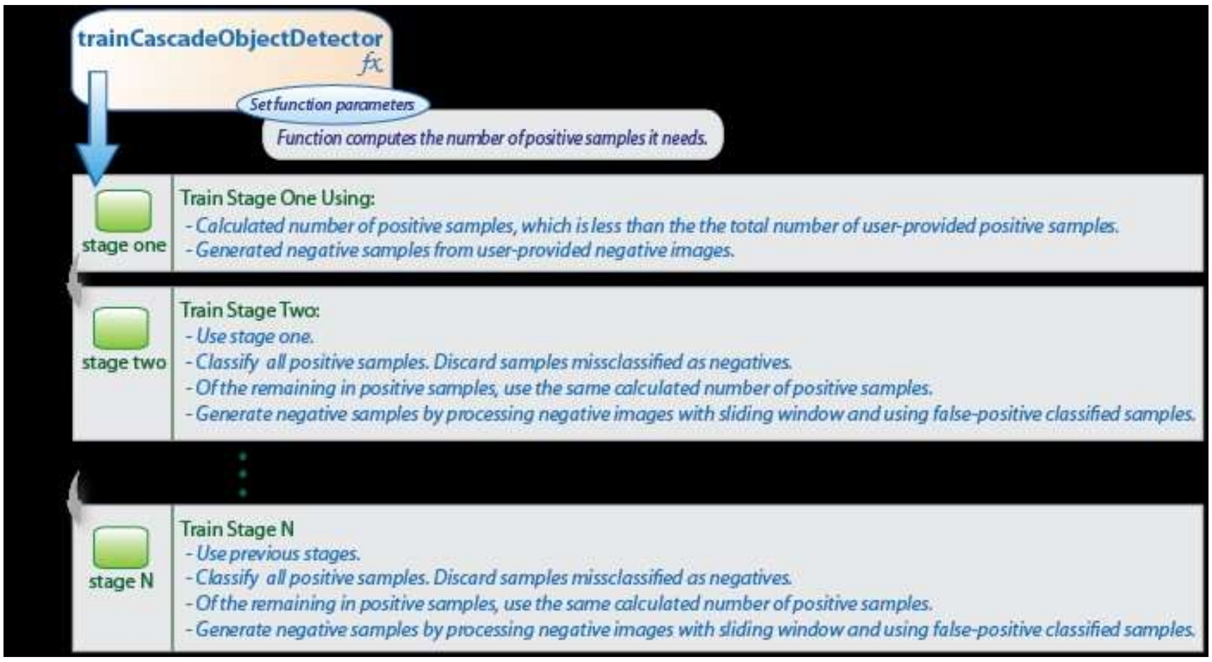
First step is to collect the Haar Features. A Haar feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums.



During the detection phase, a window of the target size is moved over the input image, and for each subsection of the image and Haar features are calculated. This difference is then compared to a learned threshold that separates non-objects from objects. Because each Haar feature is only a "weak classifier" (its detection quality is slightly better than random guessing) a large number of Haar features are necessary to describe an object with sufficient accuracy and are therefore organized into cascade classifiers to form a strong classifier.

The cascade classifier consists of a collection of stages, where each stage is an ensemble of weak learners. The weak learners are simple classifiers called decision stumps. Each stage is trained using a technique called boosting. Boosting provides the ability to train a highly accurate classifier by taking a weighted average of the decisions made by the weak learners.

Each stage of the classifier labels the region defined by the current location of the sliding window as either positive or negative. Positive indicates that an object was found and negative indicates no objects were found. If the label is negative, the classification of this region is complete, and the detector slides the window to the next location. If the label is positive, the classifier passes the region to the next stage. The detector reports an object found at the current window location when the final stage classifies the region as positive.



Function computes the number of positive samples it needs.

Train Stage One Using:
- Calculated number of positive samples, which is less than the the total number of user-provided positive samples.
- Generated negative samples from user-provided negative images.

Train Stage Two:
- Use stage one.
- Classify all positive samples. Discard samples missclassified as negatives.
- Of the remaining in positive samples, use the same calculated number of positive samples.
- Generate negative samples by processing negative images with sliding window and using false-positive classified samples.

Train Stage N
- Use previous stages.
- Classify all positive samples. Discard samples missclassified as negatives.
- Of the remaining in positive samples, use the same calculated number of positive samples.
- Generate negative samples by processing negative images with sliding window and using false-positive classified samples.

Now coming onto our dataset of the hand gesture and palm detection for our application. For this particular reason we need to look into multiple methods for the detection of the said features in our hand.

1. BACKGROUND SUBTRACTION FOR IDENTIFICATION OF HAND OBJECT

   The idea of the background subtraction is to capture the model of the background first and then compare it against the current image. After comparison, the known background parts are subtracted away.

2. GESTURE RECOGNITION

   The gesture recognition is divided into three major parts: first, skin classifier for determine whether the object is hand or not. Second part is the contour finding which is for extracting out the hand region. The last one is using the Haar-like classifier cascades to identify the open palm gesture.

   2.1. Skin Classifier

   Before using the Haar cascade to detect the hand gesture, it is necessary to find out the human hand position. Using the HSV threshold to determine the hand region, and then focus on these regions for the gesture recognition.

   2.2. Contour Finding

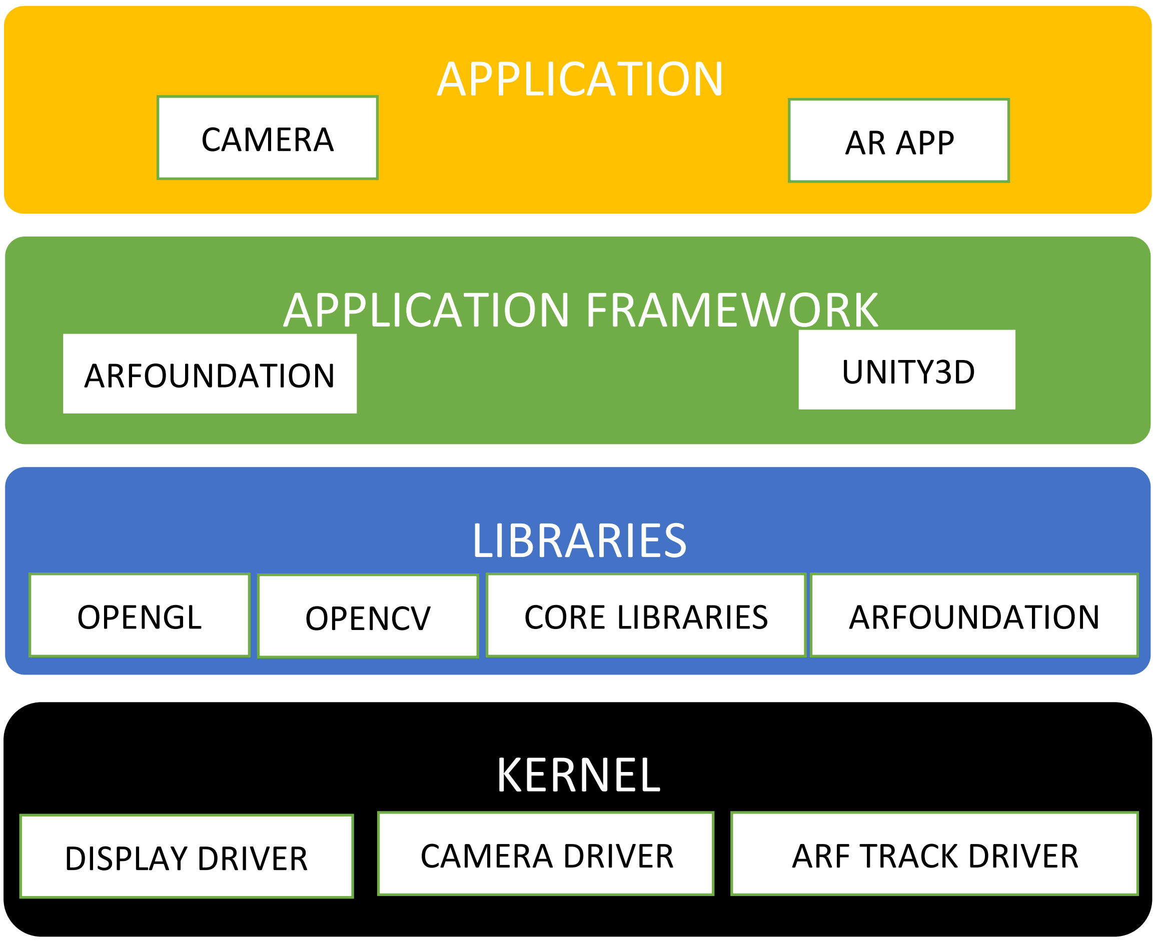
   After extract the region that may be contain the object, we need to use the contour finding methods to find out the boundary of that area. If the area is too small, then it will be disregard during the hand detection. And using the region of interest method that proposed by OpenCV to focus these areas for the recognition and detection.

   2.3. Haar classifier Cascades

   First, in the data preparation, it is necessary to prepare vast original images used for Adaboost training. A large amount of original image is to improve the detection accuracy. It separated into two types, the positive and the negative images. The positive image is the detecting object image, for example, the hand detection needs many hand pictures as the positive image. Another is the negative images which haven't contained any object picture. mage segmentation

   Second, in the image segmentation stage, it is important for reduce the background noise of the original image. Therefore, the pure background colour is required during the image capture progress. Moreover, the lighting condition and background noise also, will influence the quality of the training data such as a hand and the background colour are not separated clearly due to the lighting condition. To tackle the problem, we use HSV model's threshold value to separate the background noise in the training images. Third, in create sample stage; we need to create a description file for those positive images. Fourth, it is time to create the classifier by using the Harr training in OpenCV. When using the Haar training command, we have to set up a lot of negative images, more background image, and the better result. After the process is finish, the Haar training create an .xml file. This classifier is cascaded stronger classifier. At last, the stronger classifier is used to detect the hand gesture.

5. STACK DIAGRAM



6. FRONT END AND DEVELOPMENT

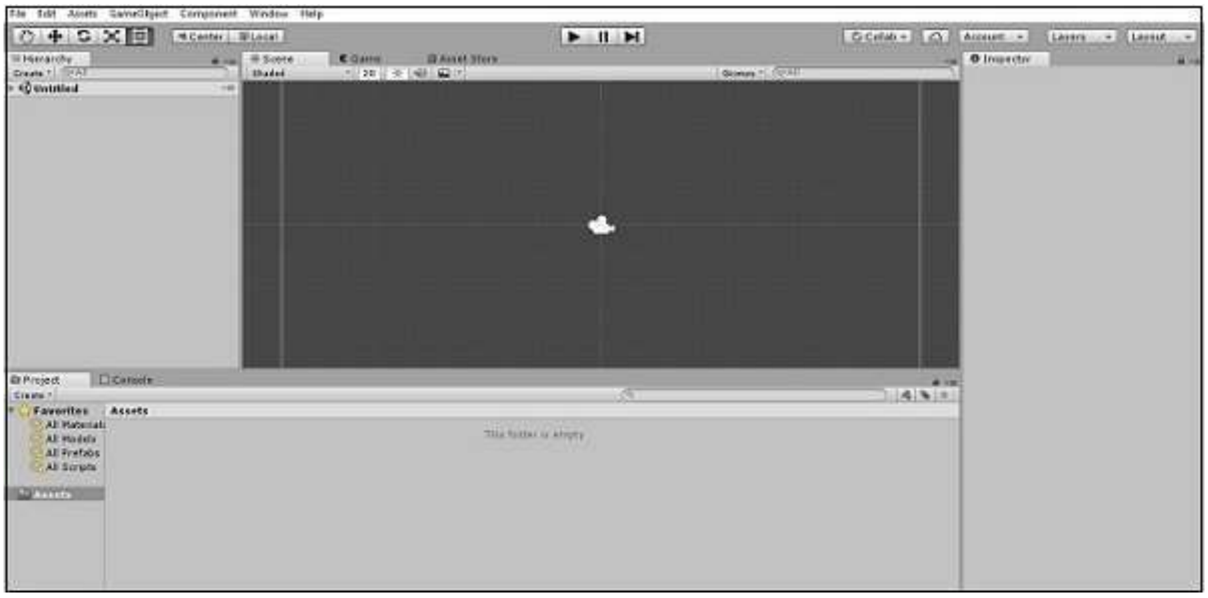
The front end of the application is achieved by using the Unity 3D game engine. The front end consists of the following:

1. Permission access
2. Tutorial
3. Interactive menu
4. Parameter visualization
5. Object rendering

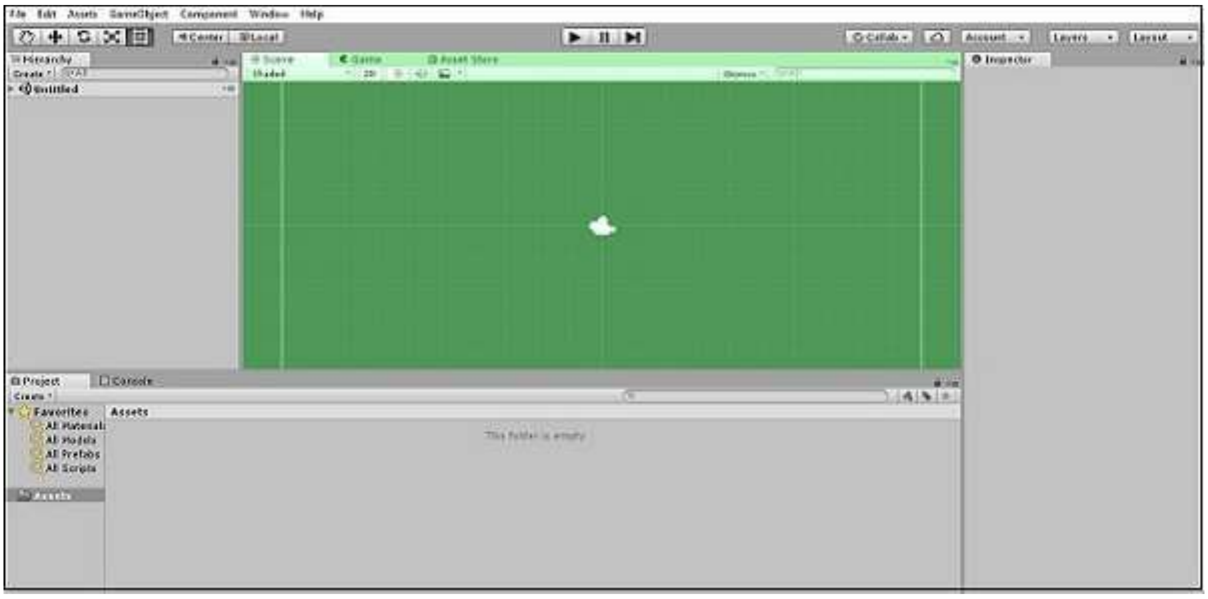
We will discuss about each of the above in detail. It is essential to know about the following keywords in Unity 3D first.

Knowing the Engine

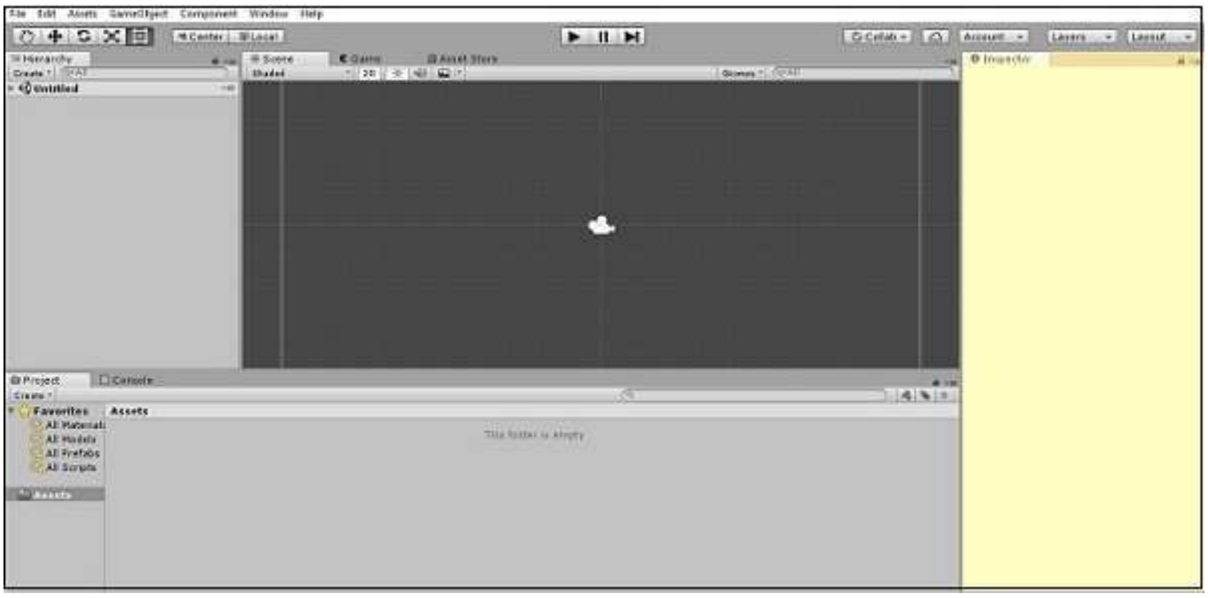
Once your new project is created and Unity opens, the following screen appears –



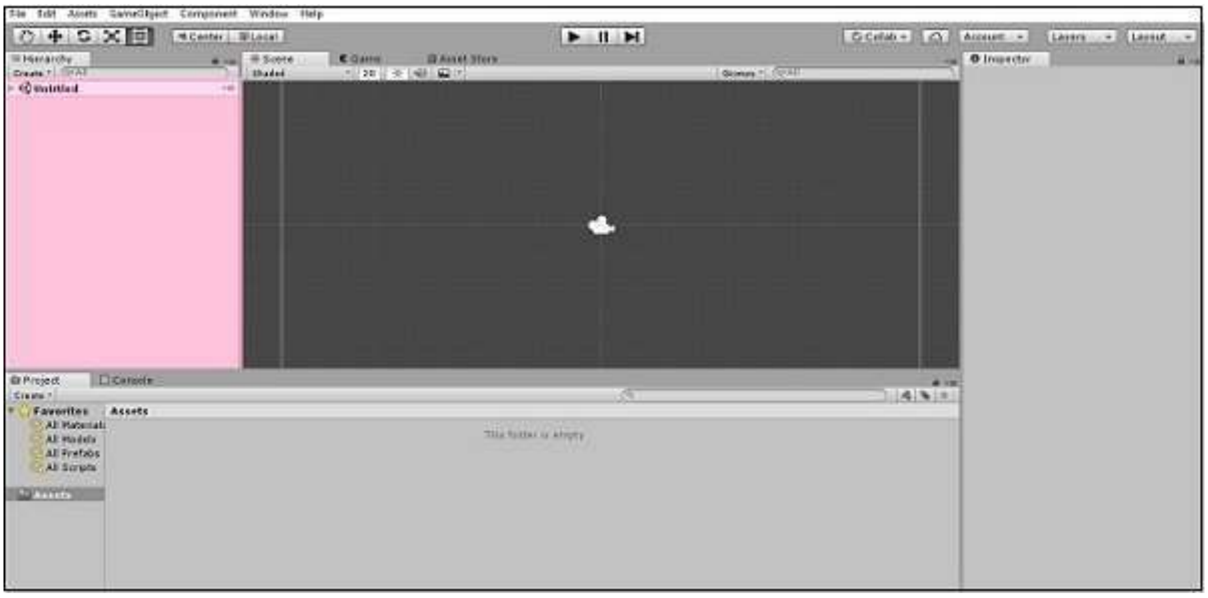
Let us have a quick run-through of what is visible in this window. For the time being, we are concerned with four main regions –



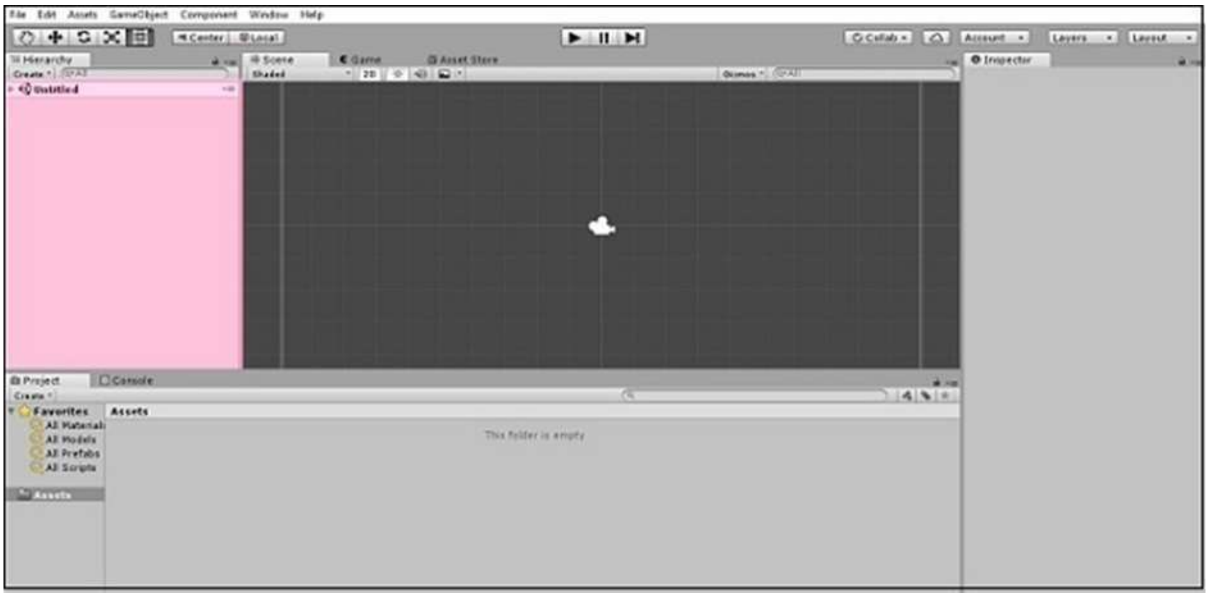
This window is where we will build our Scenes. Scenes are levels in which everything in your game takes place. If you click on the small Game tab, you can see a Preview window of how the game looks like to the player. For now, it should be a simple, blue background.



This region is the Inspector. It is empty for now, because we do not have any objects in our scene. We will see how the Inspector is used later on.



This window is the Scene Hierarchy. It is where all the objects in your currently open scene are listed, along with their parent-child hierarchy. We will add objects to this list only.



Finally, this region is the Project Assets window. All assets in your current project are stored and kept here. All externally imported assets such as textures, fonts and sound files are also kept here before they are used in a scene.

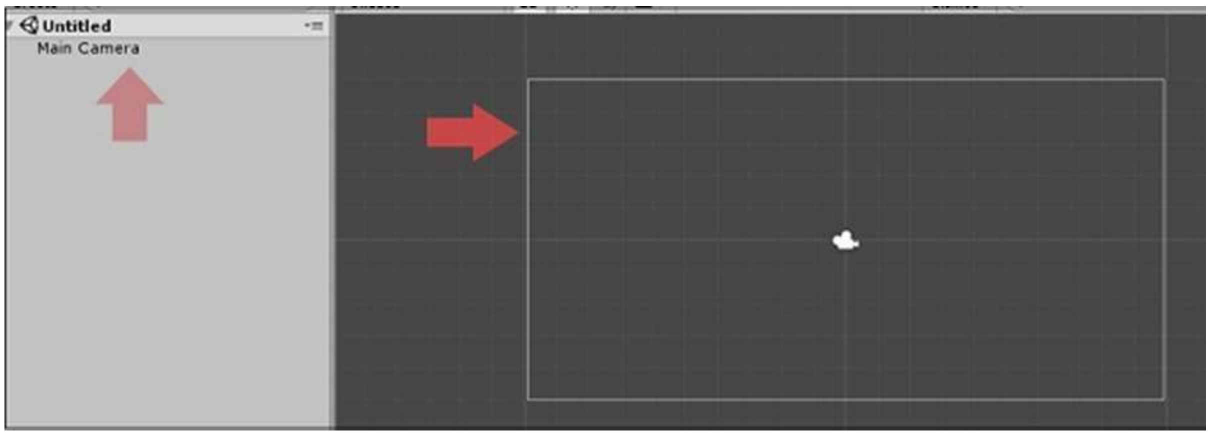
How Unity Works?

In Unity, all gameplay takes place in scenes. Scenes are levels in which all aspects of your game such as game levels, the title screen, menus and cut scenes take place.

By default, a new Scene in Unity will have a Camera object in the scene called the Main Camera. It is possible to add multiple cameras to the scene, but we will only deal with the main camera for now.

The main camera renders everything that it sees or "captures" in a specific region called the viewport. Everything that comes into this region becomes visible for the player.

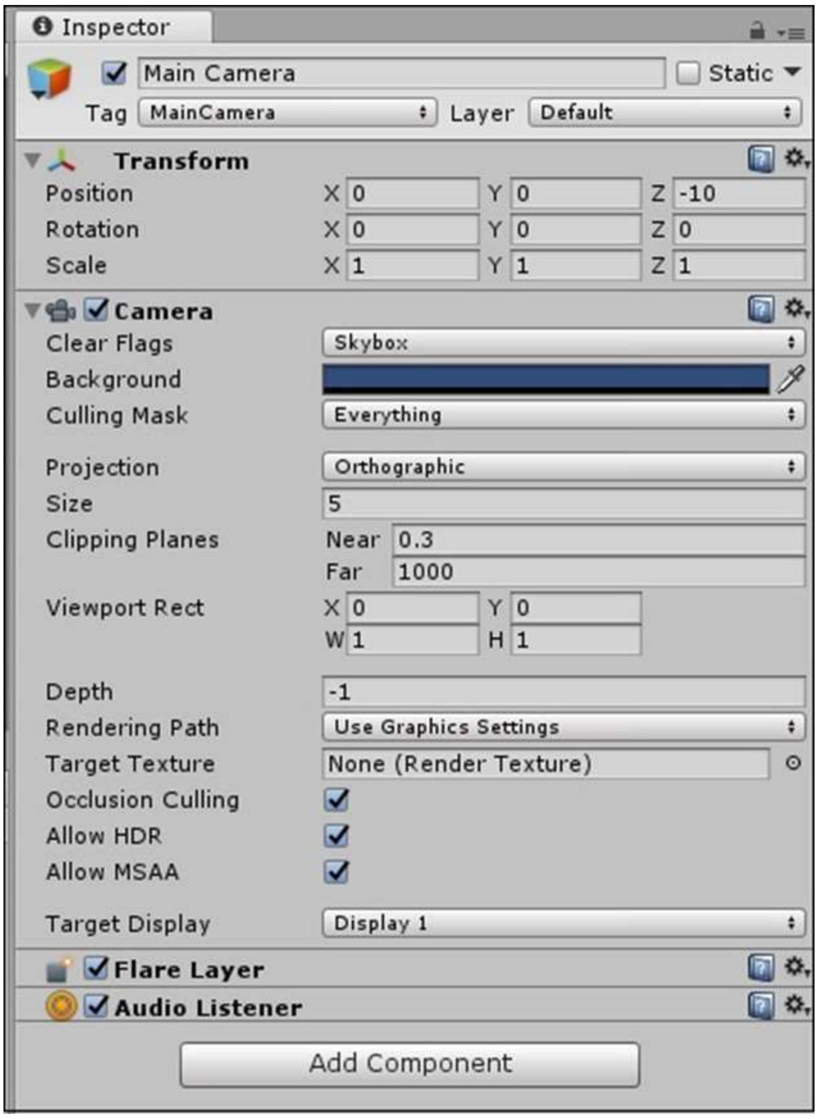
You can see this viewport as a grey rectangle by placing your mouse inside the scene view and scrolling down to zoom out the scene view. (You can also do so by holding Alt and dragging Right-click).



A scene itself is made out of objects, called Game Objects. Game Objects can be anything from the player's model to the GUI on the screen, from buttons and enemies to invisible "managers" like sources of sound.

Game Objects have a set of components attached to them, which describe how they behave in the scene, as well as how they react to others in the scene.

In fact, we can explore that right now. Click on the Main Camera in the Scene Hierarchy and look at the Inspector. It will not be empty now; instead, it will have a series of "modules" in it.



Main Camera ☑   Static ☐
Tag: MainCamera   Layer: Default

Transform
Position X 0 Y 0 Z -10
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

☑ Camera
Clear Flags: Skybox
Background: 🟦
Culling Mask: Everything
Projection: Orthographic
Size: 5
Clipping Planes: Near 0.3, Far 1000
Viewport Rect: X 0 Y 0 W 1 H 1
Depth: -1
Rendering Path: Use Graphics Settings
Target Texture: None (Render Texture)
Occlusion Culling: ☑
Allow HDR: ☑
Allow MSAA: ☑
Target Display: Display 1

☑ Flare Layer
☑ Audio Listener

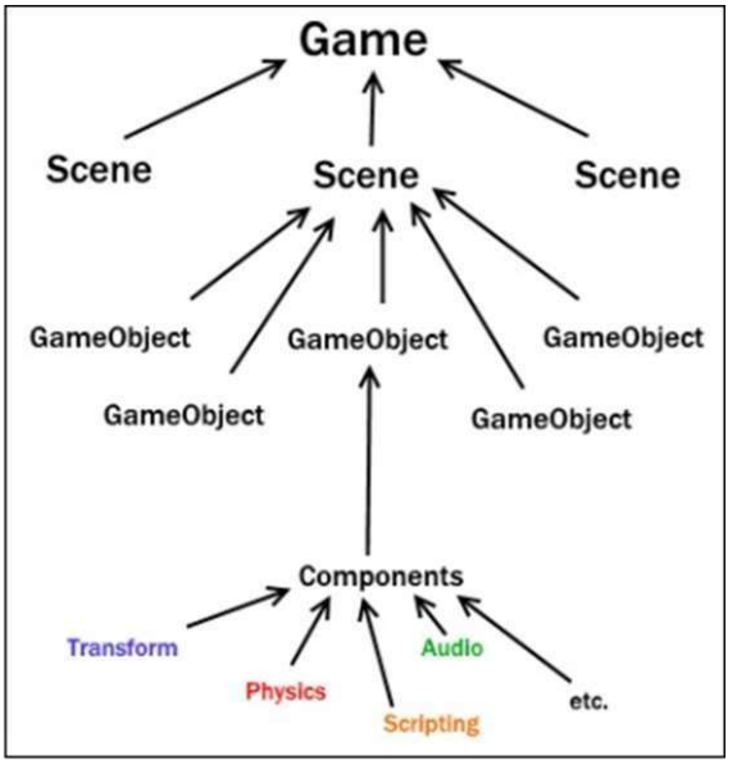
The most important component for any Game Object is its Transform component. Any object that exists in a scene will have a transform, which defines its position, rotation and scale with respect to the game world, or its parent if any.

The additional components can be attached to an object by clicking on Add Component and selecting the desired component. In our subsequent lessons, we will also be attaching Scripts to Game Objects so that we can give them programmed behaviour.

Let us now consider a few examples of components −

- Renderer − Responsible for rendering and making objects visible.
- Collider − Define the physical collision boundaries for objects.
- Rigid body − Gives an object real-time physics properties such as weight and gravity.

- Audio Source − Gives object properties to play and store sound.
- Audio Listener − The component that actually “hears” audio and outputs it to the player’s speakers. By default, one exists in the main camera.
- Animator − Gives an object access to the animation system.
- Light − Makes the object behave as a light source, with a variety of different effects.

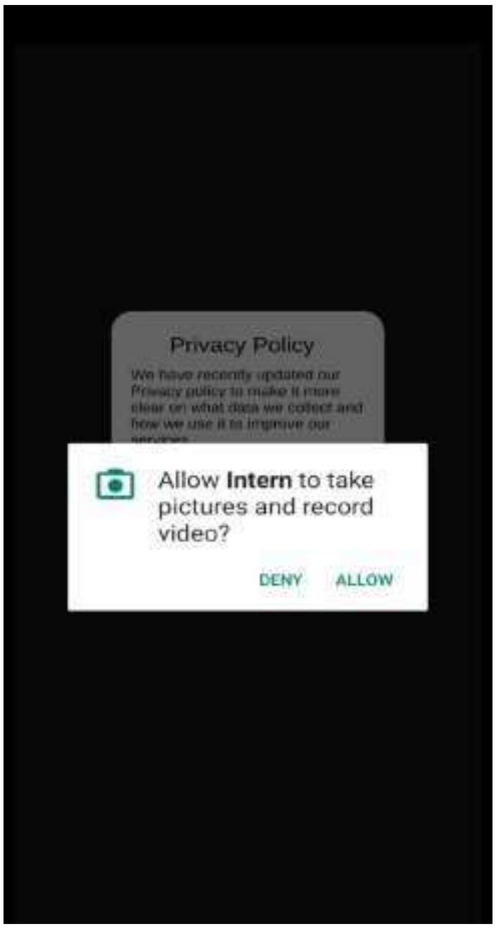


In this chart, we can see how Unity composes itself through Game Objects into scenes.

Now, since we know about the basics of the unity 3D we can look into how we implemented the front end functionalities.

5.1 Permission Access:

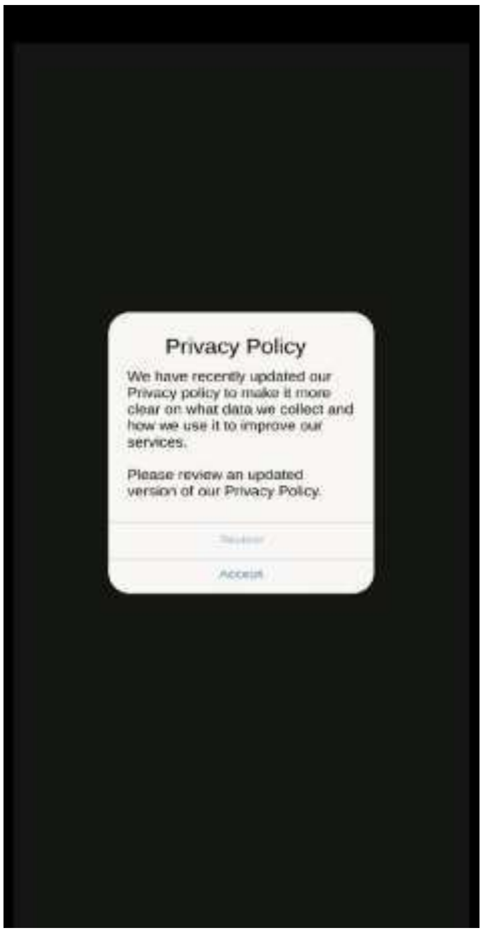
The application to be developed requires access to storage and camera to perform the required tasks. Hence, we are required to make a dialogue box pop up in front of the user and ask for his permission. Luckily, Unity provides with built in functions to trigger the permissions. Using these functions, we can auto access the camera and storage, but it is not advisable. Hence, an interactive pop up is necessary. This is achieved by instantiating the canvas object and upon button trigger. This is discussed in detail in the next topics.



Privacy Policy

We have recently updated our Privacy policy to make it more clear on what data we collect and how we use it to improve our services.

Allow **Intern** to take pictures and record video?



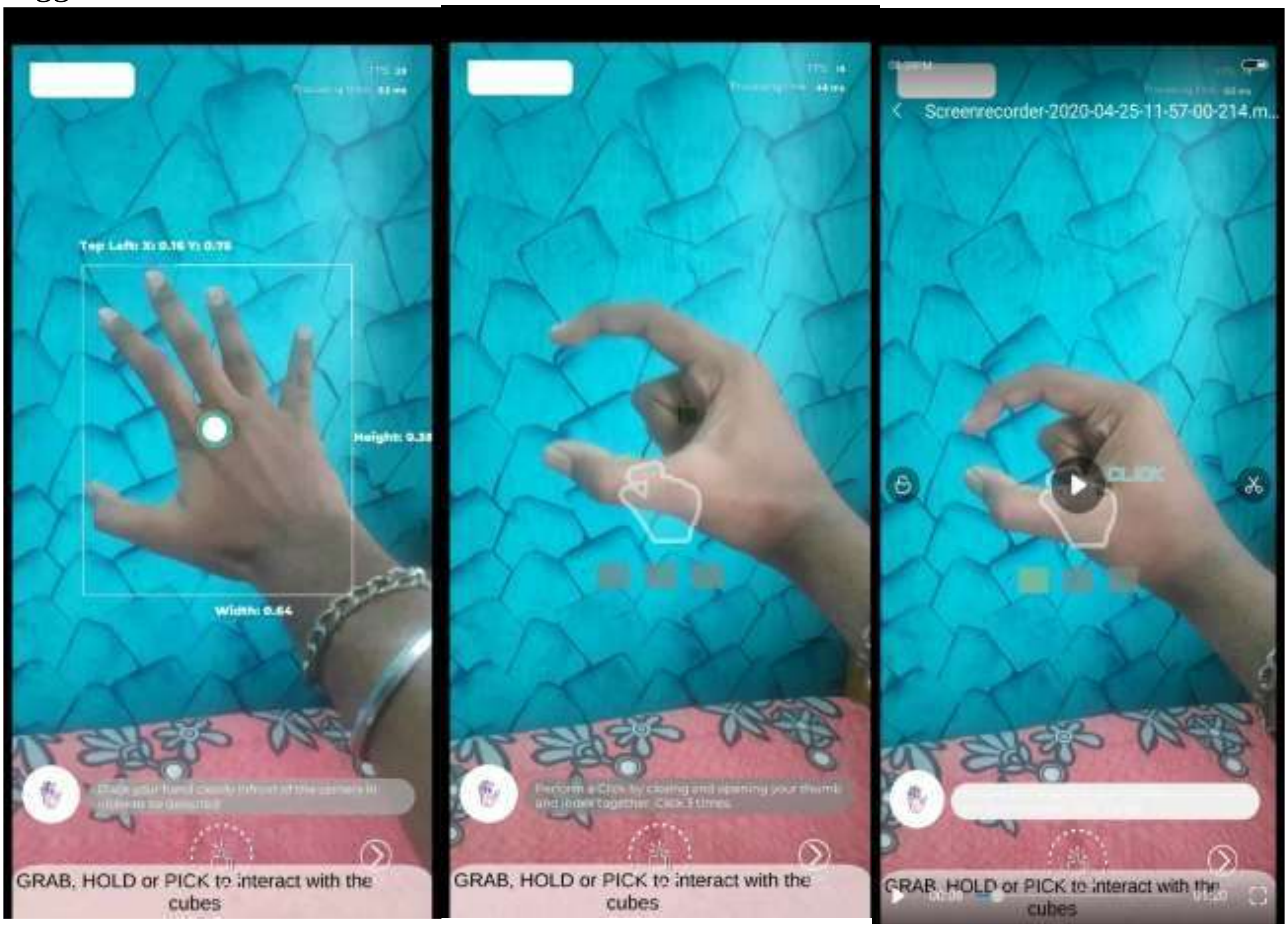
Privacy Policy

We have recently updated our Privacy policy to make it more clear on what data we collect and how we use it to improve our services.

Please review an updated version of our Privacy Policy.

5.2 Tutorials:

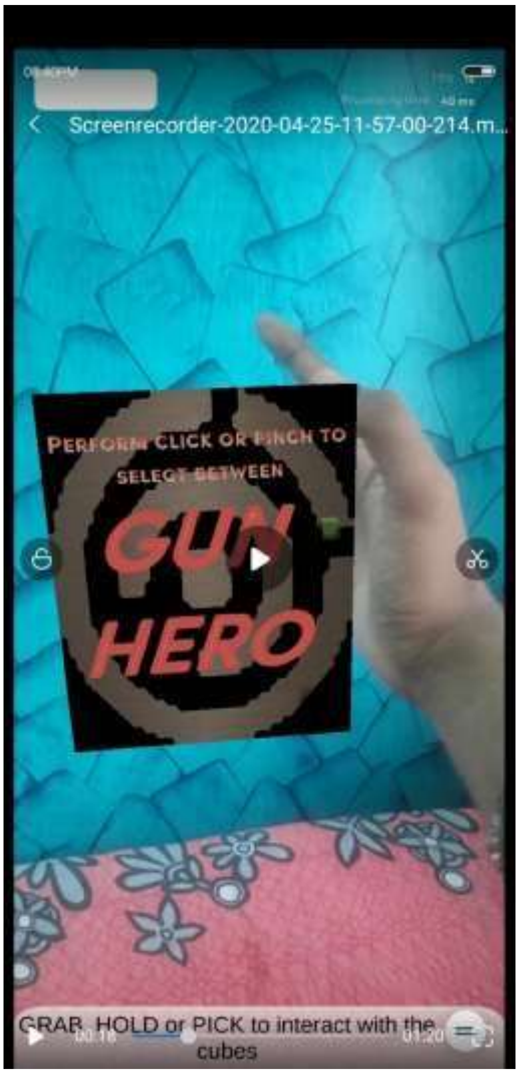
While using the application for the first time, the user is unaware of how to control through the application. Hence, an interactive tutorial is put in to improve his understanding of the application. This is also achieved through gesture-based triggers and canvas instantiation.

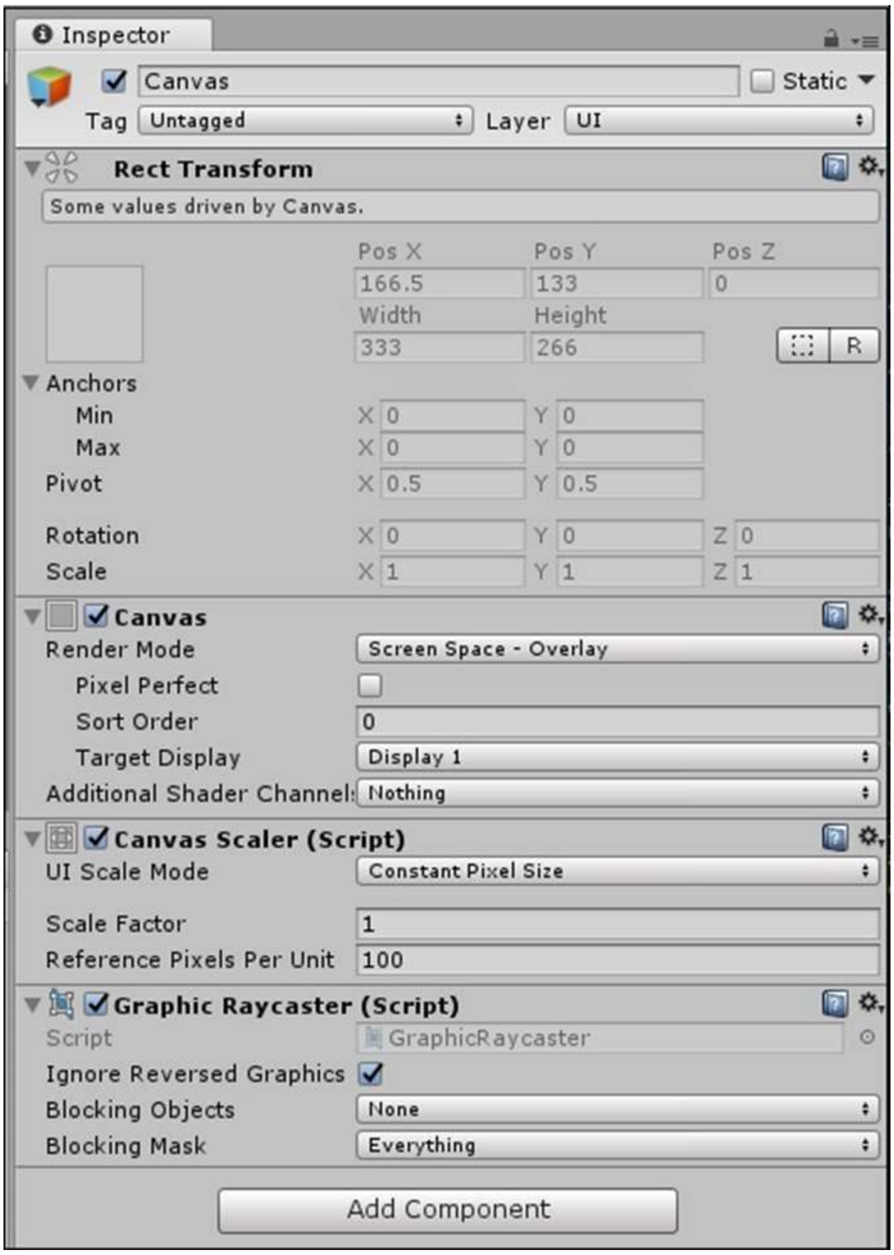

Top Left: X: 0.16 Y: 0.76
Height: 0.38
Width: 0.64
GRAB, HOLD or PICK to interact with the cubes
GRAB, HOLD or PICK to interact with the cubes
Screenrecorder-2020-04-25-11-57-00-214.m...
CLICK
GRAB, HOLD or PICK to interact with the cubes

5.3 Interactive Menu:

The workflow for designing UI in Unity follows a slightly different path than the one we have been going through so far UI elements are not standard Game Objects and cannot be used as such. UI elements are designed differently; a menu button which looks correct in a 4:3 resolution may look stretched or distorted in a 16:9 resolution if not set up right.

UI elements in Unity are not placed directly onto the scene. They are always placed as children of a special Game Object called the Canvas. The Canvas is like a “drawing sheet” for UI on the scene, where all UI elements will render. Creating a UI element from the Create context menu without an existing Canvas will automatically generate one.


Screenrecorder-2020-04-25-11-57-00-214.m...
PERFORM CLICK OR PINCH TO SELECT BETWEEN
GUN HERO
GRAB, HOLD or PICK to interact with the cubes


Inspector

Canvas ✓ — Static ☐
Tag: Untagged — Layer: UI

Rect Transform
Some values driven by Canvas.

	Pos X	Pos Y	Pos Z
	166.5	133	0
	Width	Height	
	333	266	

	X	Y	Z
Anchors Min	0	0	
Anchors Max	0	0	
Pivot	0.5	0.5	
Rotation	0	0	0
Scale	1	1	1

Canvas ✓

Render Mode	Screen Space - Overlay
Pixel Perfect	☐
Sort Order	0
Target Display	Display 1
Additional Shader Channel:	Nothing

Canvas Scaler (Script) ✓

UI Scale Mode	Constant Pixel Size
Scale Factor	1
Reference Pixels Per Unit	100

Graphic Raycaster (Script) ✓

Script	GraphicRaycaster
Ignore Reversed Graphics	✓
Blocking Objects	None
Blocking Mask	Everything

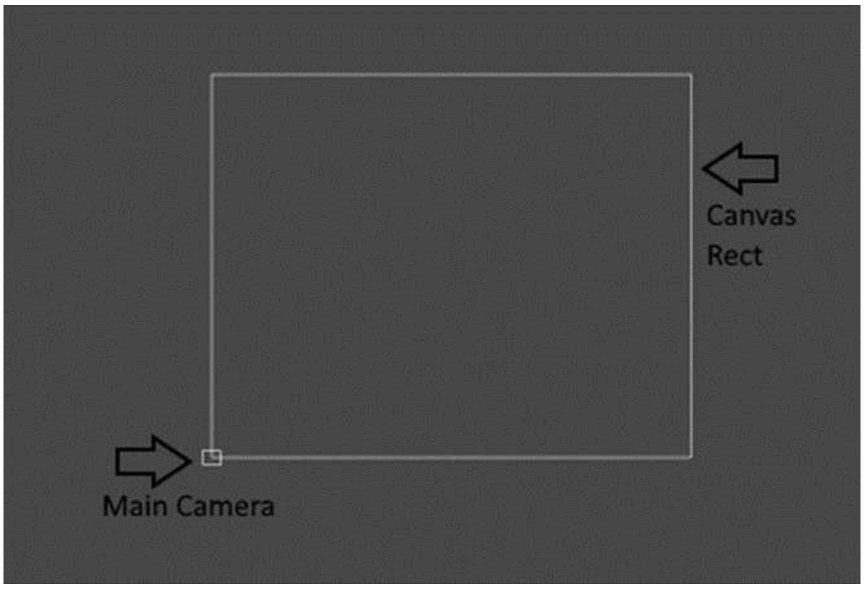
Add Component

The Rect Transform at the top appears to have many new properties that a standard game Object's Transform does not have.

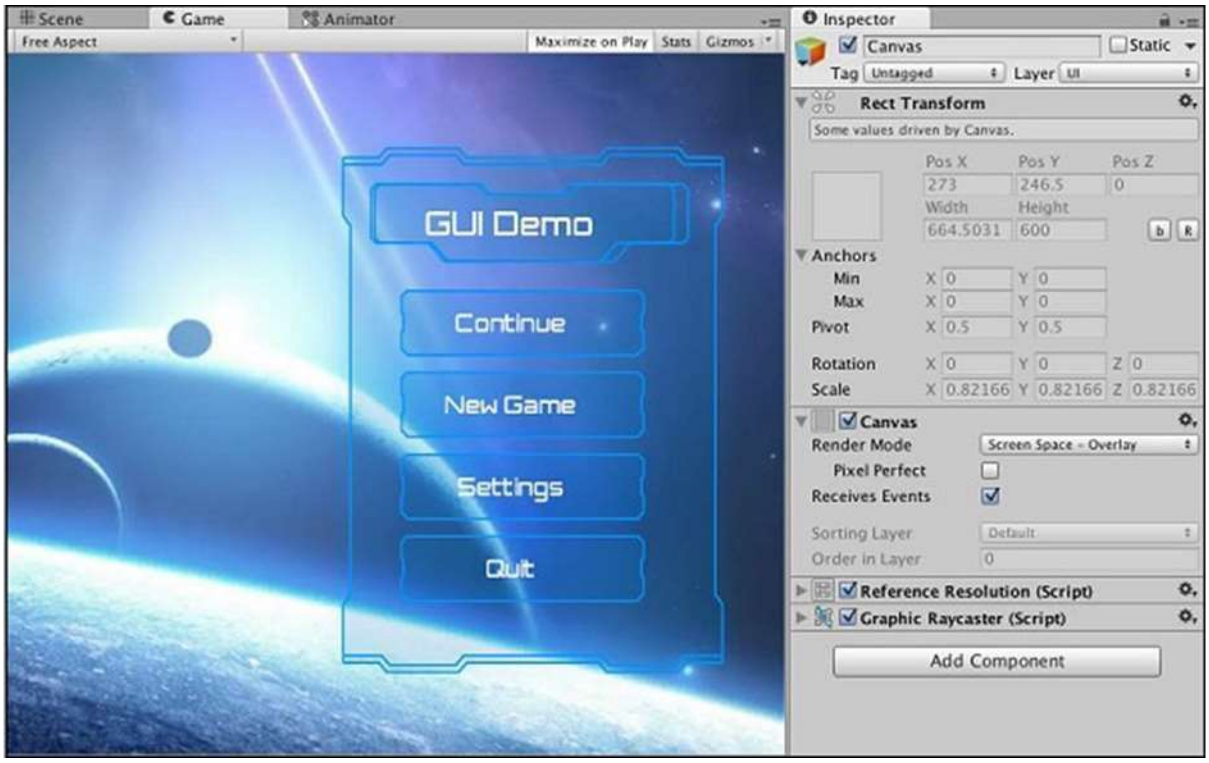
This is because while a normal game Object's Transform describes an imaginary point in 3D space, a RectTransform defines an imaginary rectangle. This means we need additional properties for defining exactly where the rectangle is, how big it is and how it is oriented.

We can see some standard properties of a rectangle like the Height and Width, as well as two new properties called Anchors. Anchors are pointing that other entities can "lock" onto in the Canvas. This means that if a UI element (say, a button) is anchored to the Canvas on the right, resizing the Canvas will ensure that the Button is always on the relative right of the Canvas.

By default, you will not be able to modify the shape of the canvas area, and it will be a comparatively gigantic rectangle around your scene.



Next is the Canvas Component. This is the master component that holds a couple of universal options as to how the UI is drawn.



Canvas ✓ · Static ☐
Tag: Untagged · Layer: UI

Rect Transform
Some values driven by Canvas.

	Pos X	Pos Y	Pos Z
	273	246.5	0
	Width	Height	
	664.5031	600	
Anchors Min	X 0	Y 0	
Anchors Max	X 0	Y 0	
Pivot	X 0.5	Y 0.5	
Rotation	X 0	Y 0	Z 0
Scale	X 0.82166	Y 0.82166	Z 0.82166

Canvas ✓
Render Mode: Screen Space - Overlay
Pixel Perfect: ☐
Receives Events: ✓
Sorting Layer: Default
Order in Layer: 0

Reference Resolution (Script) ✓
Graphic Raycaster (Script) ✓
Add Component

The first option we see is the Render Mode. This property defines the method that is used to draw the Canvas onto the game’s view.

We have three options in the dropdown list. Let us learn about the options in our subsequent sections.

Screen Space - Overlay

This mode is the most standard for menus, HUDs and so on. It renders UI on top of everything else in the scene, exactly how it is arranged and without exception. It also scales the UI nicely when the screen or game window size changes. This is the default Render Mode in the Canvas.

Screen Space - Camera

Screen Space - Camera creates an imaginary projection plane, a set distance from the camera, and projects all UI onto it. This means that the appearance of the UI in the scene depends heavily on the settings used by the camera; this includes perspective, field of view, and so on.

World Space

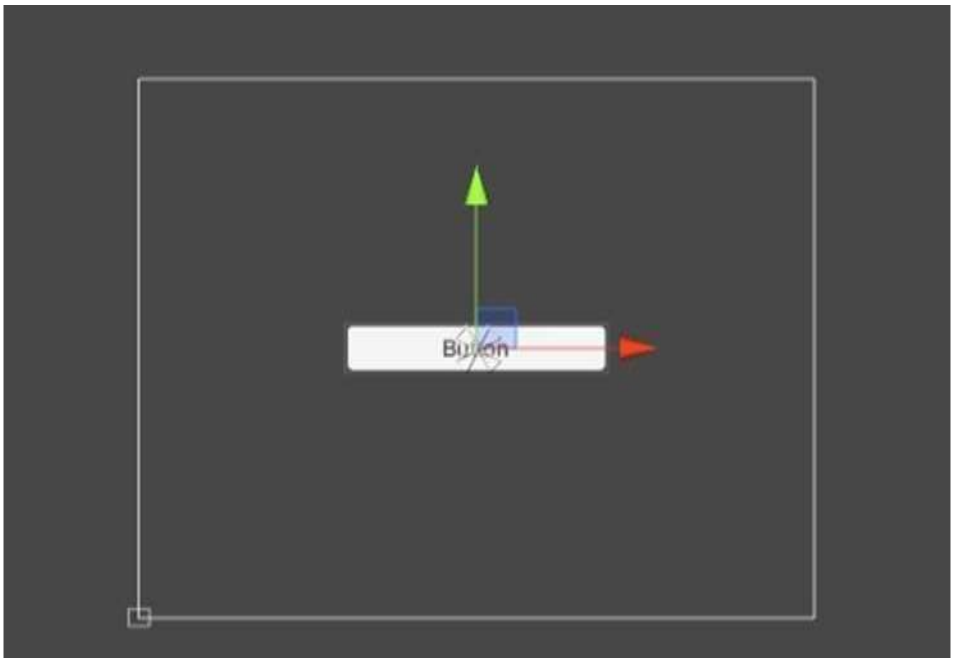
In World Space mode, UI elements behave as if they were normal Game Objects placed into the world. They are similar to sprites, however, so they are typically used as part of the game world instead of for the player, like ingame monitors and displays. Because of this nature, you can directly modify the values of the Canvas RectTransform in this mode.

The Canvas Scaler is a set of options that lets you adjust the scale and appearance of the UI elements in a more definitive way; it allows you to define how UI elements resize themselves when the size of the screen changes. For example, UI elements can remain the same size regardless of as well as in ratio to the screen size, or they can scale according to a Reference Resolution.

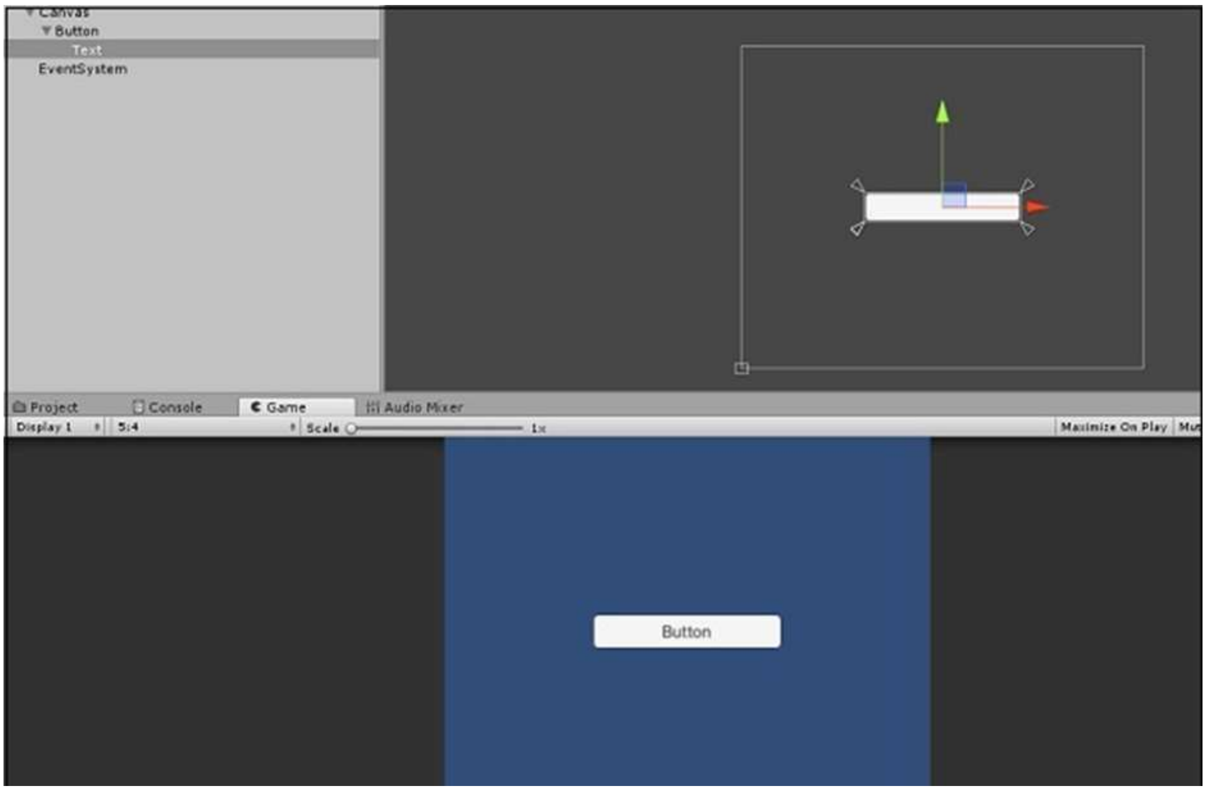
The Graphics Ray caster deals primarily with ray casting (link to Unity Documentation for Ray casting) the UI elements and ensuring user-initiated events like clicks and drags work correctly.

BUTTONS:

To insert a button, right click in the Scene Hierarchy and go to Create → UI → Button. If you do not have an existing Canvas and an Event System, Unity will automatically create one for you, and place the button inside the Canvas as well.



Remember that in Overlay rendering mode, which is the default mode, the size of the Canvas is independent of the size of the camera. You can test this by clicking on the Game tab.

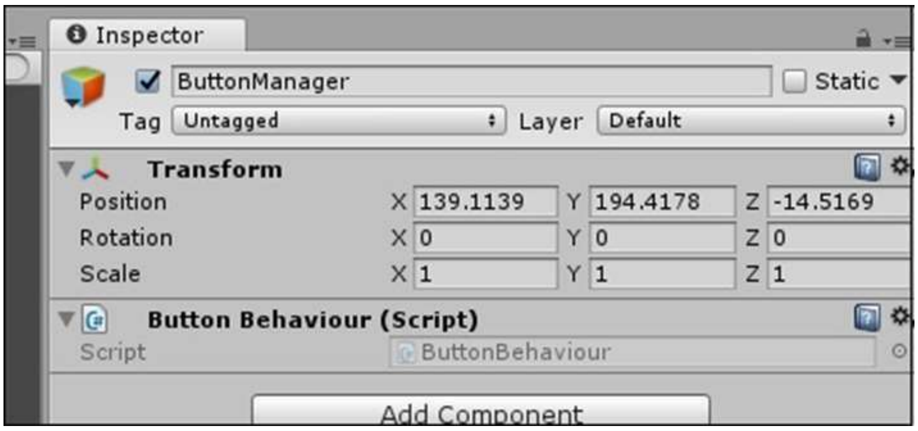


If you play the scene, you will notice the button already has some standard functionality such as detecting when the mouse is hovering over it, and changing colour when pressed.

A Button requires functionality to be actually useful in the UI. This functionality can be added through its properties.

Button Behaviour

Let us create an empty Game Object and attach this script to it. We do this because a button will not do anything on its own; it only calls the specified method in its scripting.



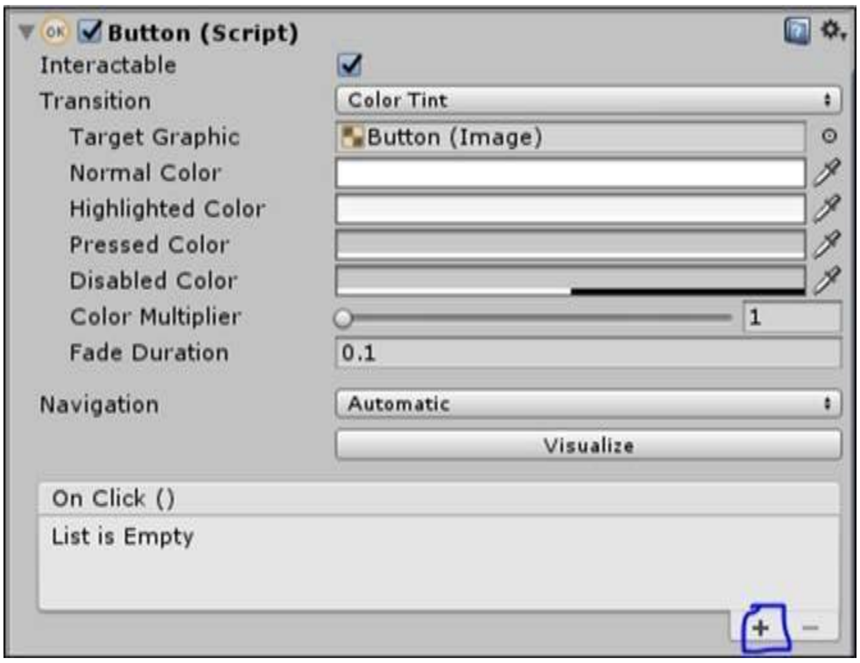
Inspector
ButtonManager   Static
Tag Untagged   Layer Default

Transform
Position X 139.1139 Y 194.4178 Z -14.5169
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Button Behaviour (Script)
Script ButtonBehaviour

Add Component

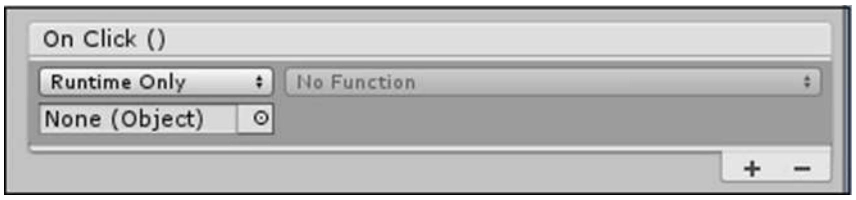
Now, go into the Button's properties, and find the On Click() property.



Button (Script)
Interactable ✓
Transition Color Tint
Target Graphic Button (Image)
Normal Color
Highlighted Color
Pressed Color
Disabled Color
Color Multiplier 1
Fade Duration 0.1
Navigation Automatic
Visualize

On Click ()
List is Empty

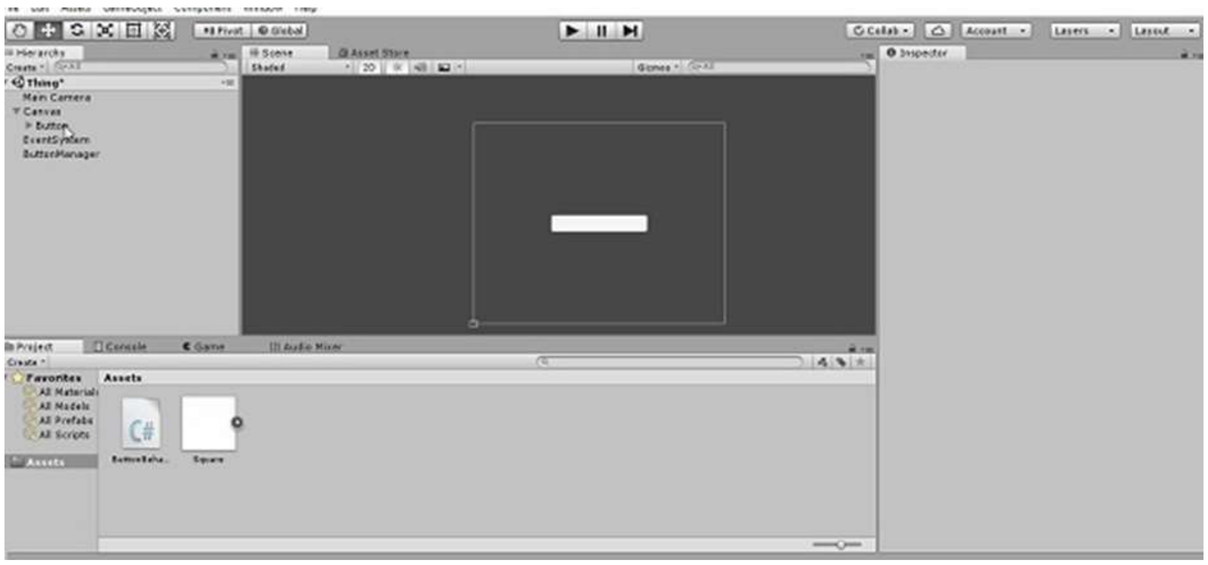
Hit the + icon on the bottom tab, and a new entry should show up in the list.



On Click ()
Runtime Only   No Function
None (Object)

This entry defines what object the button presses acts on, and what function of that object's script is called. Because of the event system used in the button press, you can trigger multiple functions simply by adding them to the list.

Drag and drop the empty Game Object, which contains the Button Manager script we created, onto the None (Object) slot.

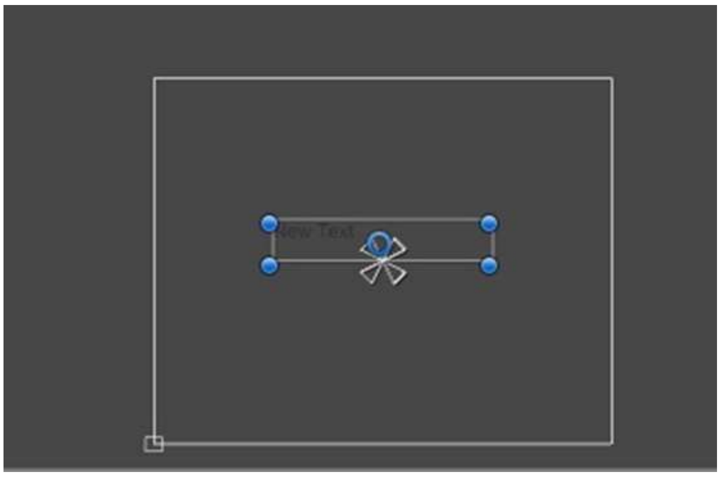


Navigate the No Function dropdown list, and look for our On Button Press method. You should find it in the Button Behaviour section.

TEXT:

Text being a distinct UI element of its own is primarily due to the dynamism of that element. For example, printing the player's current score to the screen requires the numeric value of the score to be converted to a string, generally through the .to String() method, before it is displayed.

To insert a Text UI element, go to the Scene Hierarchy, Create → UI → Text.



A new Text element should show up in your Canvas region. If we have a look at its properties, we will see some very useful options.

What is most significant of all, however, is the **Text field**. You can type out what you want the text box to say in that field, but we want to go a step further than that.

To change the font of the text, you must first import the **font file** from your computer into Unity, as an Asset. A font does not need to be actively attached to anything in the scene, and it can be directly referenced from the Assets.

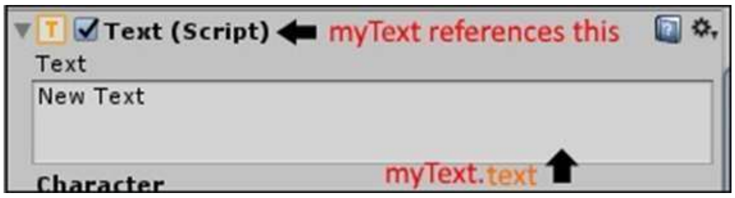
The Text element can be accessed through scripting as well; this is where the importance of **dynamic** UI comes in.

Instead of the console, outputting how many times the button has been pressed, as in the previous chapter; let us actually print it out on the game screen. To do so, we will open up our Button Behaviour script from the previous lesson, and make a few changes to it.

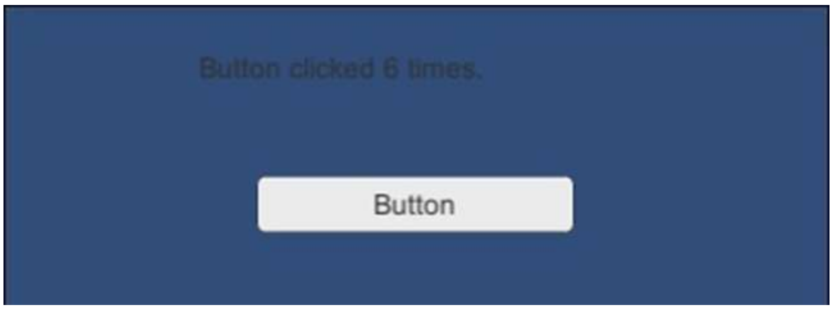
The first change we did was to add a new namespace reference. This reference is used to work with Unity's UI components, and so we add the using UnityEngine.UI line.

Next, we create a public Text variable, where we can drag and drop our Text UI element onto.

Finally, we access the actual text this UI element contains using myText.text.

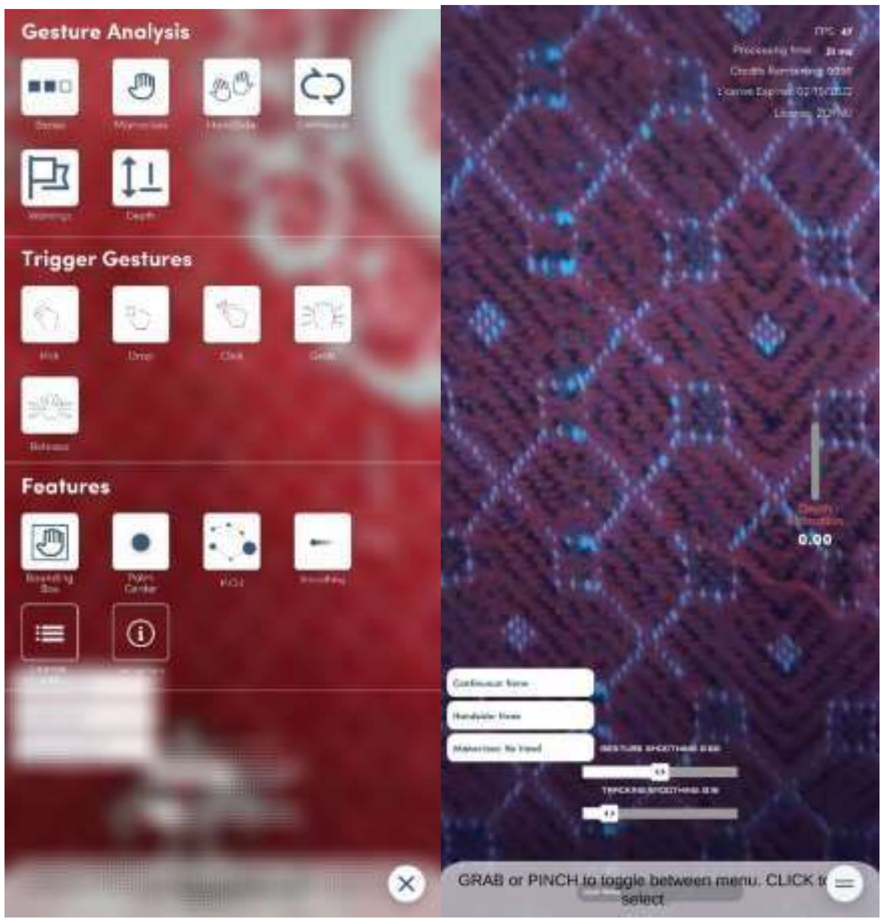


If we save our script, we will now see a new slot for the Text UI element in our Button Manager. Simply drag and drop the game Object containing that Text element onto the slot, and hit the Play button.



5.4 Parameter Visualization:

By making use of the buttons, as discussed above, we are able to show/hide the various parameters such as FPS which are calculated and attached to the game objects as scripts. On toggling these buttons ON, the prefabs are instantiated and the information is viewed on screen. The instantiation and destruction are discussed later.



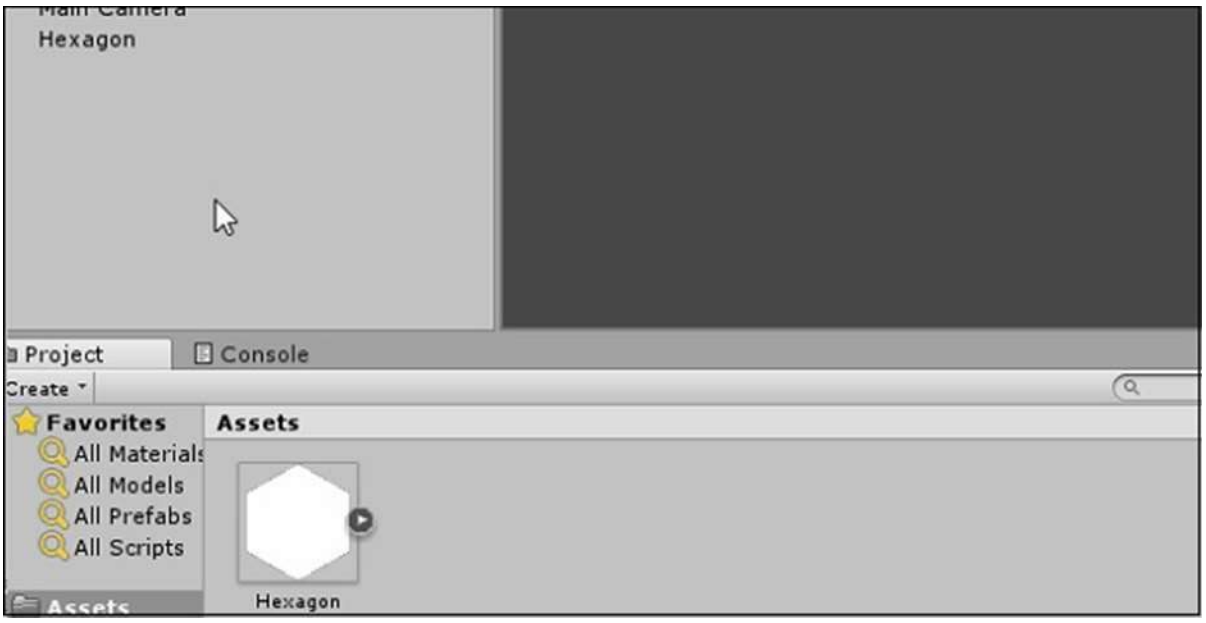
5.5 Object Rendering:

The virtual objects are used to appear and disappear while performing gestures. This is achieved through object instantiation and deletion.

Instantiating and destroying objects is considered very important during gameplay. Instantiating simply means bringing into existence. Items appear or "spawn" in the game, enemies die, GUI elements vanish and scenes are loaded all the time in-game. Knowing how to properly get rid of unneeded objects and how to bring in those you do then becomes even more essential. An example of how to instantiate is described below.

Prefabs are like blueprints of a Game Object. Prefabs are, in a way, a copy of a Game Object that can be duplicated and put into a scene, even if it did not exist when the scene was being made; in other words, prefabs can be used to dynamically generate Game Objects.

To create a prefab, you simply have to drag the desired Game Object from your scene hierarchy into the project Assets.



Now, to instantiate a Game Object, we call the Instantiate() method in our script. This method, defined in Mono Behaviour, takes in a Game Object as a parameter, so it knows which Game Object to create/duplicate. It also has various overrides for changing the newly instantiated object's transform, as well as parenting.

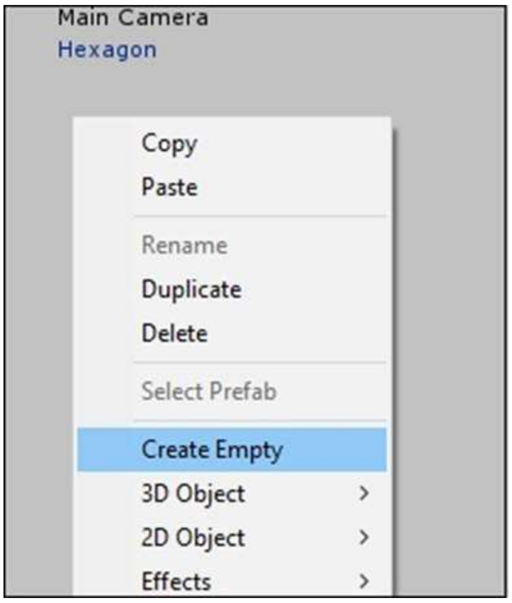
Let us try instantiating a new hexagon whenever the Space key is pressed.

Create a new script called Instantiator and open it up. In the Update method, type in the code given below.

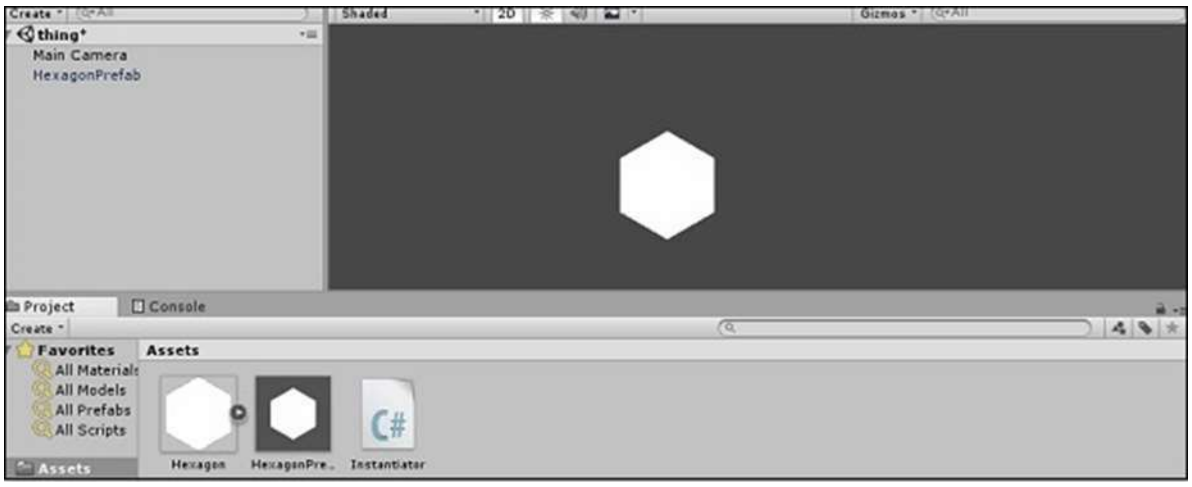
Here, we are using the GetKeyDown method of the Input class to check if the player pressed a specific button during the last frame. Since we want it to keep checking, we put it in Update, which runs 60 times per second. The GetKeyDown method returns true if the key specified by the KeyCode enum (which lists all possible keys on a standard keyboard) is pressed in that frame.

The public Game Object declaration at the top creates a slot similar to the one we made for the Rigidbody2D in our previous lessons. This slot only accepts prefabs (in editor time) and game Objects (in runtime), however.

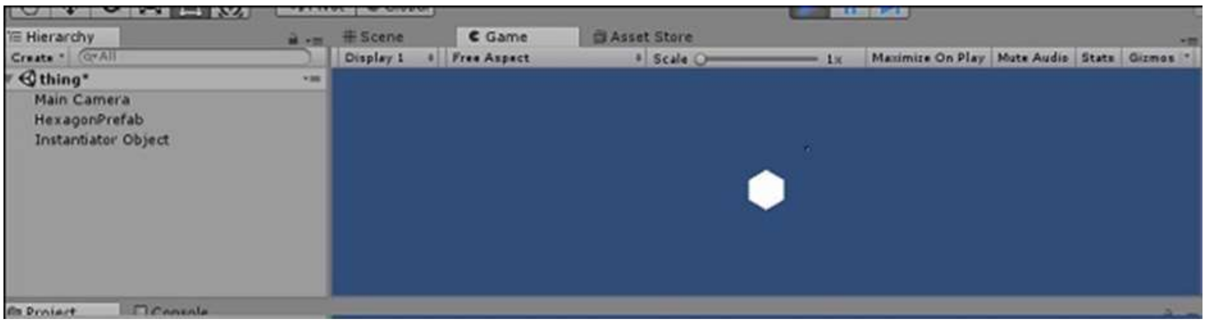
Save the script, and let it compile. Once it is done, create a new, empty Game Object by going to your object hierarchy right-click menu, and selecting Create Empty.



Name this Object something recognizable such as Instantiator Object and attach our newly created script to it. In the slot that shows up for the Game Object, drag in the prefab we created.



If we run the game now, pressing the Spacebar will create a new Hexagon object identical to the one we used to create the prefab. You can see each hexagon being created in the object hierarchy. The reason you cannot see them show up in the game is because for the time being, they are all being created exactly one over the other.



This concept is applied to the objects being instantiated in the application.

The destruction of Game Objects is as important as the instantiation. Fortunately, destroying Game Objects is as easy as it is creating them. You simply need a reference to the object to be destroyed, and call the Destroy() method with this reference as a parameter.

An example of how the concept of destruction works is given below.

let us try to make 5 hexagons which will destroy themselves when an assigned key is pressed.

Let us make a new script called Hexagon Destroyer and open it in Visual Studio. We will start by making a public KeyCode variable. A KeyCode is used to specify a key on a standard keyboard, and the Input class in its methods uses it. By making this variable public, as we did with Rigid body and Prefabs previously, we can make it accessible through the editor. When the variable is made public, we need not hardcode values such as "KeyCode.A" into the code. The code can be made flexible with as many objects as we want.

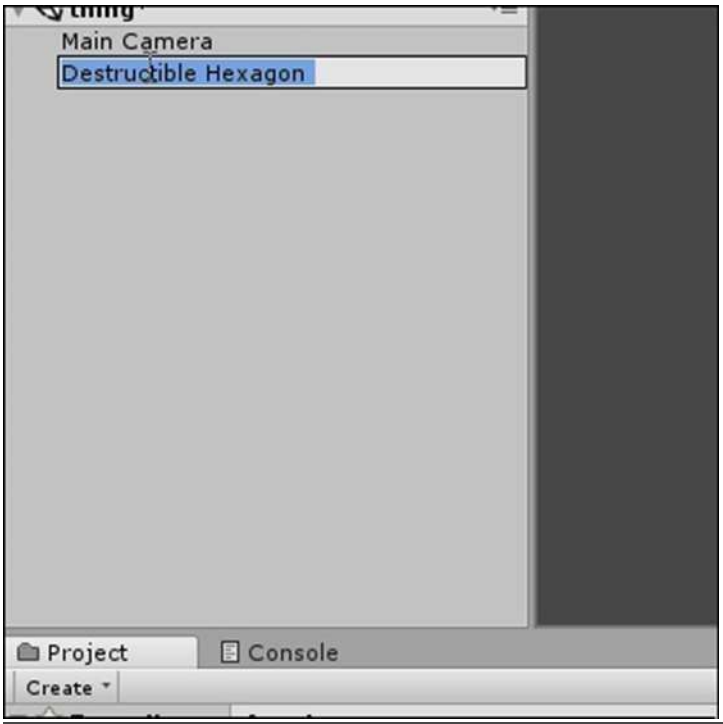
Observe how we used the variable named "game Object" (small g, capital O) in the method. This new game Object variable (of type Game Object) is used to refer to the game Object this script is attached to. If you attach this script on multiple objects, they will all react the same way whenever this variable is involved.

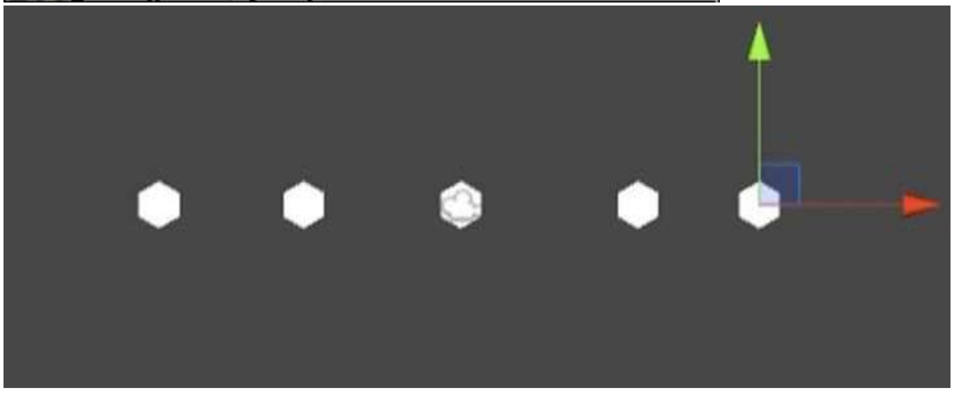
Do not get confused between the two, however.

- Game Object with a capital G and O is the class that encompasses all Game Objects and provides standard methods like Instantiate, Destroy and methods to fetch components.
- game Object with a small g and capital O is the specific instance of a Game Object, used to refer to the game Object this script is currently attached to.

Let us now compile our code, and head back to Unity.

Now, we will create a new hexagon sprite, and attach our script to it. Next, right-click the game Object in the hierarchy and select Duplicate. A new sprite is created in the hierarchy; you should use the Move tool to reposition it. Repeat the steps to create similar hexagons.



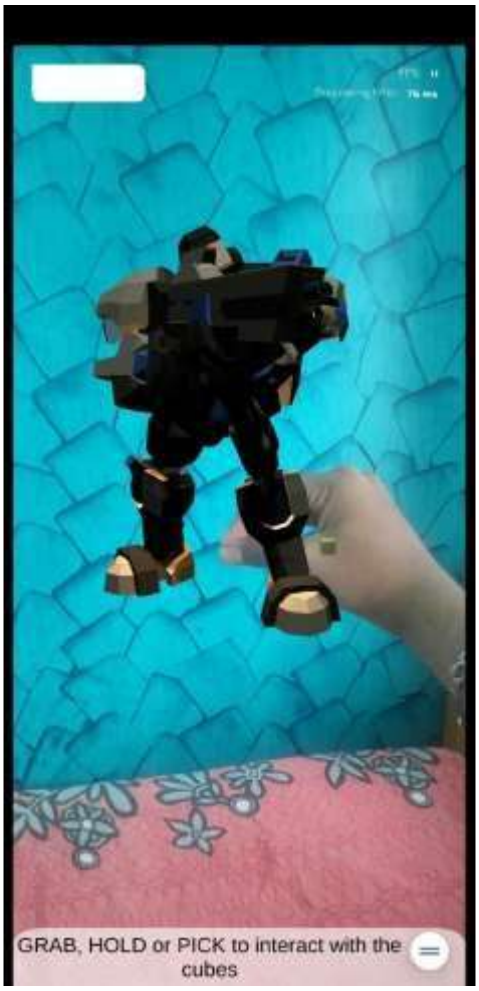


Click on each of the hexagons and look at their script components. You can now set the individual keys so that a Game Object destroys itself when that key is pressed. For example, let us create 5 hexagons, and set them to destroy when the A, S, D, F and G keys are pressed.

You can set the same key on multiple hexagons, and they will all destroy themselves simultaneously when the key is pressed; this is an example of the use of the game Object reference, which you can use to refer to individual objects using the script without having to set them individually.

The same key can be set on multiple hexagons, and they will all destroy themselves simultaneously when the key is pressed; this is an example of the use of the game Object reference, which you can use to refer to individual objects using the script without having to set them individually.

It is important to understand that destroying a Game Object does not mean an object will shatter or explode. Destroying an object will simply (and immediately) cease its existence as far as the game (and its code) is concerned. The links to this object and its references are now broken, and trying to access or use either of them will usually result in errors and crashes. This concept is used to destroy the unnecessary instances of a game object.



7. SOURCE CODES OF THE APPLICATION

AR cube Interaction code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ARCubeInteraction : MonoBehaviour
{
    private ManoGestureContinuous grab;
    private ManoGestureContinuous pinch;
    private ManoGestureTrigger click;

    [SerializeField]
    private Material[] arCubeMaterial;
    [SerializeField]
    private GameObject smallCube;

    private string handTag = "Player";
    private Renderer cubeRenderer;
    private TrackingInfo tracking;
    public Vector3 currentPosition;


    void Start()
    {
        Initialize();
    }
    void Update()
    {
        tracking =
ManomotionManager.Instance.Hand_infos[0].hand_info.tracking_info;
        currentPosition = Camera.main.ViewportToWorldPoint(new
Vector3(tracking.palm_center.x - 0.25f, tracking.palm_center.y - 0.05f ,
tracking.depth_estimation+0.15f));
        transform.position = currentPosition;
        if
(ManomotionManager.Instance.Hand_infos[0].hand_info.gesture_info.mano_gestur
e_continuous == pinch)
        {
            transform.Rotate(Vector3.up * Time.deltaTime * 50, Space.World);
        }
    }



    private void Initialize()
    {
        grab = ManoGestureContinuous.CLOSED_HAND_GESTURE;
        pinch = ManoGestureContinuous.HOLD_GESTURE;
        click = ManoGestureTrigger.CLICK;
        cubeRenderer = GetComponent<Renderer>();
        cubeRenderer.sharedMaterial = arCubeMaterial[0];
        cubeRenderer.material = arCubeMaterial[0];
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="other">The collider that stays</param>
    private void OnTriggerStay(Collider other)
    {
        //MoveWhenGrab(other);
        //RotateWhenHolding(other);
        //SpawnWhenClicking(other);
    }

    /// <summary>
    /// If grab is performed while hand collider is in the cube.
    /// The cube will follow the hand.
    /// </summary>
    private void MoveWhenGrab(Collider other)
    {
        if
(ManomotionManager.Instance.Hand_infos[0].hand_info.gesture_info.mano_gestur
e_continuous == grab)
        {
            transform.parent = other.gameObject.transform;
        }

        else
        {
            transform.parent = null;
        }
    }

    /// <summary>
    /// If pinch is performed while hand collider is in the cube.
    /// The cube will start rotate.
    /// </summary>
    private void RotateWhenHolding(Collider other)
    {
        if
(ManomotionManager.Instance.Hand_infos[0].hand_info.gesture_info.mano_gestur
e_continuous == pinch)
        {
            transform.Rotate(Vector3.up * Time.deltaTime * 50, Space.World);
        }
    }

    /// <summary>
    /// If pick is performed while hand collider is in the cube.
    /// The cube will follow the hand.
    /// </summary>
    private void SpawnWhenClicking(Collider other)
    {
        if
(ManomotionManager.Instance.Hand_infos[0].hand_info.gesture_info.mano_gestur
e_trigger == click)
        {
            Instantiate(smallCube, new Vector3(transform.position.x,
transform.position.y + transform.localScale.y / 1.5f, transform.position.z),
Quaternion.identity);
        }
    }

    /// <summary>
    /// Vibrate when hand collider enters the cube.
    /// </summary>
    /// <param name="other">The collider that enters</param>
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == handTag)
        {
            cubeRenderer.sharedMaterial = arCubeMaterial[1];
            Handheld.Vibrate();
        }
    }

    /// <summary>
    /// Change material when exit the cube
    /// </summary>
    /// <param name="other">The collider that exits</param>
    private void OnTriggerExit(Collider other)
    {
        cubeRenderer.sharedMaterial = arCubeMaterial[0];
    }
}
```

Bounding Box code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public struct BoundingBox
{
   public Vector3 top_left;
   public float width;
   public float height;
}
```

Bounding Box UI code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;

public class BoundingBoxUI : MonoBehaviour
{

   [SerializeField]
   TextMesh top_left, width, height;
   public LineRenderer bound_line_renderer;
   private ManoUtils mano_utils;
   float textDepthModifier = 4;
   float textAdjustment = 0.01f;

   float backgroundDepth = 8;
```

```
    private void Start()
    {
        mano_utils = ManoUtils.Instance;
        bound_line_renderer.positionCount = 4;
    }

    float normalizedTopLeftX;
    float normalizedTopLeftY;
    float normalizedBBWidth;
    float normalizedHeight;

    Vector3 normalizedTopLeft;
    Vector3 normalizedTopRight;
    Vector3 normalizedBotRight;
    Vector3 normalizedBotLeft;
    Vector3 normalizedTextHeightPosition;
    Vector3 normalizedTextWidth;

    /// <summary>
    /// Positions the parts of the BoundingBox Points and updates the 3D Text
information
    /// </summary>
    /// <param name="bounding_box"></param>
    public void UpdateInfo(BoundingBox bounding_box)
    {

        if (!mano_utils)
            mano_utils = ManoUtils.Instance;
        if (!bound_line_renderer)
        {
           Debug.Log("bound_line_renderer: null");
           return;

        }
        normalizedTopLeftX = bounding_box.top_left.x;
        normalizedTopLeftY = bounding_box.top_left.y;
        normalizedBBWidth = bounding_box.width;
        normalizedHeight = bounding_box.height;

        normalizedTopLeft = new Vector3(normalizedTopLeftX,
normalizedTopLeftY);
        normalizedTopRight = new Vector3(normalizedTopLeftX +
normalizedBBWidth, normalizedTopLeftY);
        normalizedBotRight = new Vector3(normalizedTopLeftX +
normalizedBBWidth, normalizedTopLeftY - normalizedHeight);
        normalizedBotLeft = new Vector3(normalizedTopLeftX, normalizedTopLeftY
- normalizedHeight);

        bound_line_renderer.SetPosition(0,
ManoUtils.Instance.CalculateNewPosition(normalizedTopLeft,
backgroundDepth));
        bound_line_renderer.SetPosition(1,
ManoUtils.Instance.CalculateNewPosition(normalizedTopRight,
backgroundDepth));
        bound_line_renderer.SetPosition(2,
ManoUtils.Instance.CalculateNewPosition(normalizedBotRight,
backgroundDepth));
        bound_line_renderer.SetPosition(3,
ManoUtils.Instance.CalculateNewPosition(normalizedBotLeft, backgroundDepth));
```

```
        normalizedTopLeft.y += textAdjustment * 3;

        top_left.gameObject.transform.position =
ManoUtils.Instance.CalculateNewPosition(normalizedTopLeft, backgroundDepth /
textDepthModifier);

        top_left.text = "Top Left: " + "X: " + normalizedTopLeftX.ToString("F2") + "
Y: " + normalizedTopLeftY.ToString("F2");


        normalizedTextHeightPosition = new Vector3(normalizedTopLeftX +
normalizedBBWidth + textAdjustment, (normalizedTopLeftY - normalizedHeight /
2f));

        height.transform.position =
ManoUtils.Instance.CalculateNewPosition(normalizedTextHeightPosition,
backgroundDepth / textDepthModifier);

        height.GetComponent<TextMesh>().text = "Height: " +
normalizedHeight.ToString("F2");


        normalizedTextWidth = new Vector3(normalizedTopLeftX +
normalizedBBWidth / 2f, (normalizedTopLeftY - normalizedHeight) -
textAdjustment);

        width.transform.position =
ManoUtils.Instance.CalculateNewPosition(normalizedTextWidth,
backgroundDepth / textDepthModifier);

        width.GetComponent<TextMesh>().text = "Width: " +
normalizedBBWidth.ToString("F2");




    }


}
```

Category creator code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ManoMotion.UI.Buttons;
public class Category : MonoBehaviour
{
    public string categoryName;
    public int categoryOrder;
    public GameObject[] icons;
    public List<UIIconBehavior> iconBehaviors = new List<UIIconBehavior>();
    public GameObject[,] iconArray;

    #region UIValues

    public int maxIconsForRow;
    public float categoryPositionY;
    public int categoryHeight;
    public int numberOfRows;

    private int defaultHeight = 160;
    private int extraHeightForRow = 100;
    private int iconWidth = 60;
    private int iconLeftMargin;
    private int iconRightMargin;
    private int iconSpaceTaken;
    private int categoryWidth;
    private RectTransform rt;
```

```
#endregion

private void Awake()
{
    InitializeIconBehaviors();
}

void Start()
{
    InitializeUIValues();
    InitializeIconBehaviors();
    IconMainMenu.OnOrientationChanged += CalculateDimensions;
}

/// <summary>
/// Initializes the values needed for the UI in order to be responsive.
/// </summary>
void InitializeUIValues()
{
    defaultHeight = 160;
    extraHeightForRow = 100;
    iconWidth = 60;
    iconLeftMargin = iconWidth / 3;
    iconRightMargin = iconWidth / 3;
    iconSpaceTaken = iconWidth + iconLeftMargin + iconRightMargin;

    rt = this.GetComponent<RectTransform>();
    }

    /// <summary>
    /// Calculates the dimensions.
    /// </summary>
    public void CalculateDimensions()
    {
        StartCoroutine(Calculate());
    }

    /// <summary>
    /// Calculate the dimensions of each category in order to align the icons.
    /// </summary>
    /// <returns>The calculate.</returns>
    IEnumerator Calculate()
    {
        yield return new WaitForSeconds(0.05f);
        categoryWidth = Mathf.RoundToInt(rt.rect.width);
        numberOfRows = Mathf.CeilToInt((float)iconSpaceTaken * icons.Length /
categoryWidth);
        categoryHeight = (Mathf.Max(numberOfRows, 1) * 100) + 50;

        //Fix the size of the category
        rt.sizeDelta = new Vector2(0, categoryHeight);
        maxIconsForRow = categoryWidth / iconSpaceTaken;
        rt = this.GetComponent<RectTransform>();
        categoryPositionY = rt.anchoredPosition.y;
        //Allign the icons inside the space of the category
        StartCoroutine(AlignIcons());
    }

    /// <summary>
    /// Aligns the icons.
    /// </summary>
    /// <returns>The icons.</returns>
    IEnumerator AlignIcons()
    {
        yield return new WaitForSeconds(0.01f);
        iconArray = new GameObject[numberOfRows, maxIconsForRow];

        int totalNumber = numberOfRows * maxIconsForRow;

        for (int index = 0; index < totalNumber; index++)
        {
            int column = index % maxIconsForRow;
            int row = index / maxIconsForRow;
            if (index < icons.Length)
            {
                iconArray[row, column] = icons[index];
            }
            if (iconArray[row, column])
            {
                iconArray[row, column].transform.localPosition = new
Vector2(iconLeftMargin + column * iconSpaceTaken, -55 + (-100 * row));
            }
        }
    }

    void InitializeIconBehaviors()
    {
        for (int i = 0; i < icons.Length; i++)
        {
            iconBehaviors.Add(icons[i].GetComponent<UIIconBehavior>());
        }
    }
}
```

Category manager code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ManoMotion.UI.Buttons;
public class CategoryManager : MonoBehaviour
{
	private static CategoryManager instance;
	public static CategoryManager Instance
	{
		get
		{
			return instance;
		}

		set
		{
```

```
                instance = value;
            }
        }

        public List<UIIconBehavior> uIIconBehaviors = new
List<UIIconBehavior>();

        [SerializeField]
        private GameObject[] categories;

        private Vector2 categoryPosition;
        private Category previousCategory;

        private void Awake()
        {
            if (instance == null)
            {
                instance = this;
            }
        }

        void Start()
        {
            PositionCategories();
            IconMainMenu.OnOrientationChanged += PositionCategories;
        }

        public void SetupMenu(List<UIIconBehavior.IconFunctionality>
listOfFeatures)
```

```
        {
            GetAllIconBehaviors();

            for (int i = 0; i < uIIconBehaviors.Count; i++)
            {
                uIIconBehaviors[i].SetIsActive(false);
            }

            for (int j = 0; j < uIIconBehaviors.Count; j++)
            {
                for (int i = 0; i < listOfFeatures.Count; i++)
                {

                    UIIconBehavior currentIcon = uIIconBehaviors[j];
                    UIIconBehavior.IconFunctionality currentFunctionality
= listOfFeatures[i];
                    if (currentIcon.myIconFunctionality ==
currentFunctionality)
                    {
                        currentIcon.SetIsActive(true);
                    }
                }
            }
        }

        /// <summary>
        /// Positions the categories.
        /// </summary>
        void PositionCategories()
        {
                StartCoroutine(PositionCategoriesAfter(0.1f));
        }

        /// <summary>
        /// Positions the categories after a delay.
        /// </summary>
        /// <returns>The categories after.</returns>
        /// <param name="time">Requires a float value of delay.</param>
        IEnumerator PositionCategoriesAfter(float time)
        {
                yield return new WaitForSeconds(time);
                for (int i = 1; i < categories.Length; i++)
                {
                        if (categories[i - 1].GetComponent<Category>())
                        {
                                previousCategory = categories[i - 
1].GetComponent<Category>();
                                categoryPosition = 
categories[i].GetComponent<RectTransform>().anchoredPosition;

        categories[i].GetComponent<RectTransform>().anchoredPosition = new 
Vector2(0, (-previousCategory.categoryHeight + categories[i - 
1].GetComponent<RectTransform>().anchoredPosition.y));
                        }
                }
        }

        /// <summary>
        /// Retrieves all the Icon Behaviors from their Categories
        /// </summary>
        void GetAllIconBehaviors()
        {
                for (int i = 0; i < categories.Length; i++)
                {
                        Category currentCategory = 
categories[i].GetComponent<Category>();

                        for (int j = 0; j < currentCategory.iconBehaviors.Count; j++)
                        {
                                UIIconBehavior currentBehavior = 
currentCategory.iconBehaviors[j];
                                uIIconBehaviors.Add(currentBehavior);
                        }
                }
        }
}
```

Example detection canvas

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using UnityEngine.Android;
using UnityEngine.XR.ARFoundation;
using TMPro;

public class ExampleDetectionCanvas : MonoBehaviour
{
    [SerializeField]
    private TextMeshProUGUI statusText;

    //This square can be used to indicate that a plane has been found by overlaying a
2D sprite on the Raycasting of the screen onto a detected plane.
    [SerializeField]
    private GameObject Square;
    [SerializeField]
    private GameObject textDisplay;

    [SerializeField]
    private GameObject GizmoCanvas;

    [SerializeField]
    private ManoVisualization manoVisualization;

    private bool showBBStoredValue;

    void Start()
    {
        ARSession.stateChanged += HandleStateChanged;
        ToggleVisualizationValues.OnShowBoundingBoxValueChanged +=
HandleShowBoundingBoxValueChanged;
        showBBStoredValue = manoVisualization.Show_bounding_box;
    }

    void HandleShowBoundingBoxValueChanged(bool state)
    {
        showBBStoredValue = state;
```

```
}

void HandleStateChanged(ARSessionStateChangedEventArgs eventArg)
{
   switch (eventArg.state)
   {
      case ARSessionState.None:
         statusText.text = "session status none";

         break;
      case ARSessionState.Unsupported:
         statusText.text = "ARFoundation not supported";

         break;
      case ARSessionState.CheckingAvailability:
         statusText.text = "Checking availability";

         break;
      case ARSessionState.NeedsInstall:
         statusText.text = "Needs Install";

         break;
      case ARSessionState.Installing:
         statusText.text = "Installing";

         break;
      case ARSessionState.Ready:
         statusText.text = "Ready";
```

```
            break;
          case ARSessionState.SessionInitializing:
            statusText.text = "Poor SLAM Quality";
            break;

          case ARSessionState.SessionTracking:
            statusText.text = "Tracking quality is Good";

            break;
          default:
            break;
        }

        //Advice
        //Maybe combine this with the plane detection being 1. See if its needed after
the QA.
        //Maybe use the box in the Raycast scenario
        textDisplay.SetActive(eventArg.state != ARSessionState.SessionTracking);
        Square.SetActive(eventArg.state != ARSessionState.SessionTracking);
        GizmoCanvas.SetActive(eventArg.state == ARSessionState.SessionTracking);
    }
}
```

}

Gesture Info

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/// <summary>
/// Manoclass is the core block of the Gesture Classification. This value will be continuously updated based on the hand detection on a give frame.
/// </summary>
public enum ManoClass
{
   NO_HAND = -1,
   GRAB_GESTURE_FAMILY = 0,
   PINCH_GESTURE_FAMILY = 1,
   POINTER_GESTURE_FAMILY = 2
}
;

/// <summary>
/// The HandSide gives the information of which side of the hand is being detected with respect to the camera.
/// </summary>
public enum HandSide
{
   None = -1,
   Backside = 0,
   Palmside = 1
}
;
```

```
/// <summary>

/// Trigger Gestures are a type of Gesture Information retrieved for a given frame
when the user perfoms the correct sequence of hand movements that matches to
their action.

/// </summary>

public enum ManoGestureTrigger

{

   NO_GESTURE = -1,

   CLICK = 1,

   GRAB_GESTURE = 4,

   DROP = 8,

   PICK = 7,

   RELEASE_GESTURE = 3

}

;


/// <summary>

/// Similar to Manoclass Continuous Gestures are Gesture Information that is being
updated on every frame according to the detection of the hand pose.

/// </summary>

public enum ManoGestureContinuous

{

   NO_GESTURE = -1,

   HOLD_GESTURE = 1,

   OPEN_HAND_GESTURE = 2,

   OPEN_PINCH_GESTURE = 3,

   CLOSED_HAND_GESTURE = 4,

   POINTER_GESTURE = 5,
}
;

/// <summary>
///  Information about the gesture performed by the user.
/// </summary>
public struct GestureInfo
{
   /// <summary>
   /// Class or gesture family.
   /// </summary>
   public ManoClass mano_class;

   /// <summary>
   /// Continuous gestures are those that are mantained throug multiple frames.
   /// </summary>
   public ManoGestureContinuous mano_gesture_continuous;

   /// <summary>
   /// Trigger gestures are those that happen in one frame.
   /// </summary>
   public ManoGestureTrigger mano_gesture_trigger;

   /// <summary>
   /// State is the information of the pose of the hand depending on each class
   /// The values go from 0 (most open) to 13 (most closed)
   /// </summary>
   public int state;
```

```
    /// <summary>
    /// Represents which side of the hand is seen by the camera.
    /// </summary>
    public HandSide hand_side;
}
```

Gizmo manager code

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections.Generic;
using TMPro;

public class GizmoManager : MonoBehaviour
{
    #region Singleton

    private static GizmoManager _instance;
    public static GizmoManager Instance
    {
        get
        {
            return _instance;
        }

        set
        {
            _instance = value;
        }
    }

    #endregion

    public Color disabledStateColor;

    [SerializeField]
    private Image[] stateImages;

    [SerializeField]
    private GameObject handStatesGizmo;

    [SerializeField]
    private GameObject manoClassGizmo;

    [SerializeField]
    private GameObject handSideGizmo;

    [SerializeField]
    private GameObject continuousGestureGizmo;

    [SerializeField]
    private GameObject triggerTextPrefab;

    [SerializeField]
    private GameObject palmCenterGizmo, POIGizmo;

    [SerializeField]
```

```
    private GameObject flagHolderGizmo;

    [SerializeField]
    private GameObject smoothingSliderControler;

    [SerializeField]
    private Text currentSmoothingValue;

    [SerializeField]
    private GameObject gestureSmoothingSliderControler;

    [SerializeField]
    private Text currentGestureSmoothingValue;

    [SerializeField]
    private GameObject depthEstimationGizmo;

    [SerializeField]
    private bool _showHandStates, _showManoClass, _showPalmCenter, _showPOI,
_showHandSide, _showContinuousGestures, _showWarnings,
_showPickTriggerGesture, _showDropTriggerGesture, _showClickTriggerGesture,
_showGrabTriggerGesture, _showReleaseTriggerGesture, _showSmoothingSlider,
_showDepthEstimation;

    private GameObject topFlag, leftFlag, rightFlag;
    private RectTransform palmCenterRectTransform, palmCenterFillAmmount,
poiRectTransform, poiFillAmmount;
    private Text manoClassText, handSideText, continuousGestureText;
    private TextMeshProUGUI depthEstimationValue;
    private Image depthFillAmmount;
```

```
#region Properties

public bool ShowContinuousGestures
{
    get
    {
        return _showContinuousGestures;
    }

    set
    {
        _showContinuousGestures = value;
    }
}

public bool ShowManoClass
{
    get
    {
        return _showManoClass;
    }

    set
    {
        _showManoClass = value;
    }
}
```

```
public bool ShowPalmCenter
{
    get
    {
        return _showPalmCenter;
    }

    set
    {
        _showPalmCenter = value;
    }
}

public bool ShowPOI
{
    get
    {
        return _showPOI;
    }

    set
    {
        _showPOI = value;
    }
}

public bool ShowHandSide
```

```
{
   get
   {
      return _showHandSide;
   }

   set
   {
      _showHandSide = value;
   }
}

public bool ShowHandStates
{
   get
   {
      return _showHandStates;
   }

   set
   {
      _showHandStates = value;
   }
}

public bool ShowWarnings
{
   get
```

```
    {
        return _showWarnings;
    }

    set
    {
        _showWarnings = value;
    }
}

public bool ShowPickTriggerGesture
{
    get
    {
        return _showPickTriggerGesture;
    }

    set
    {
        _showPickTriggerGesture = value;
    }
}

public bool ShowDropTriggerGesture
{
    get
    {
        return _showDropTriggerGesture;
    }

    set
    {
        _showDropTriggerGesture = value;
    }
}

public bool ShowClickTriggerGesture
{
    get
    {
        return _showClickTriggerGesture;
    }

    set
    {
        _showClickTriggerGesture = value;
    }
}

public bool ShowGrabTriggerGesture
{
    get
    {
        return _showGrabTriggerGesture;
    }
    set
    {
        _showGrabTriggerGesture = value;
    }
}

public bool ShowReleaseTriggerGesture
{
    get
    {
        return _showReleaseTriggerGesture;
    }

    set
    {
        _showReleaseTriggerGesture = value;
    }
}

public bool ShowSmoothingSlider
{
    get
    {
        return _showSmoothingSlider;
    }

    set
    {
        _showSmoothingSlider = value;
    }
}

public bool ShowDepthEstimation
{
    get
    {
        return _showDepthEstimation;
    }
    set
    {
        _showDepthEstimation = value;
    }
}

#endregion

void Start()
{
    if (_instance == null)
    {
        _instance = this;
    }
    else
    {
        Destroy(this.gameObject);
    }
```

```
        Initialize();
    }

    private void Initialize()
    {
        SetGestureDescriptionParts();
        HighlightStatesToStateDetection(0);
        InitializeFlagParts();
        InitializeTriggerPool();
        ManomotionManager.OnManoMotionFrameProcessed +=
DisplayInformationAfterManoMotionProcessFrame;
    }

    /// <summary>
    /// Visualizes information from the ManoMotion Manager after the frame has
been processed.
    /// </summary>
    void DisplayInformationAfterManoMotionProcessFrame()
    {
        GestureInfo gestureInfo =
ManomotionManager.Instance.Hand_infos[0].hand_info.gesture_info;
        TrackingInfo trackingInfo =
ManomotionManager.Instance.Hand_infos[0].hand_info.tracking_info;
        Warning warning =
ManomotionManager.Instance.Hand_infos[0].hand_info.warning;
        Session session = ManomotionManager.Instance.Manomotion_Session;

        DisplayContinuousGestures(gestureInfo.mano_gesture_continuous);
        DisplayManoclass(gestureInfo.mano_class);
        DisplayTriggerGesture(gestureInfo.mano_gesture_trigger, trackingInfo);
        DisplayHandState(gestureInfo.state);
        DisplayPalmCenter(trackingInfo.palm_center, gestureInfo, warning);
        DisplayPOI(gestureInfo, warning, trackingInfo);
        DisplayHandSide(gestureInfo.hand_side);
        DisplayApproachingToEdgeFlags(warning);
        DisplayCurrentsmoothingValue(session);
        DisplayCurrentGestureSmoothingValue(session);
        DisplaySmoothingSlider();
        DisplayDepthEstimation(trackingInfo.depth_estimation);
    }

    #region Display Methods

    /// <summary>
    /// Displays the depth estimation of the detected hand.
    /// </summary>
    /// <param name="depthEstimation">Requires the float value of depth
estimation.</param>
    void DisplayDepthEstimation(float depthEstimation)
    {
        depthEstimationGizmo.SetActive(ShowDepthEstimation);

        if (!depthEstimationValue)
        {
            depthEstimationValue =
depthEstimationGizmo.transform.Find("DepthValue").gameObject.GetComponent
<TextMeshProUGUI>();
        }
        if (!depthFillAmmount)
        {
            depthFillAmmount = 
depthEstimationGizmo.transform.Find("CurrentLevel").gameObject.GetComponen
t<Image>();
        }
        if (ShowDepthEstimation)
        {
            depthEstimationValue.text = depthEstimation.ToString("F2");
            depthFillAmmount.fillAmount = depthEstimation;
        }
    }

    /// <summary>
    /// Displays in text value the current smoothing value of the session
    /// </summary>
    /// <param name="session">Session.</param>
    void DisplayCurrentsmoothingValue(Session session)
    {
        if (smoothingSliderControler.activeInHierarchy)
        {
            currentSmoothingValue.text = "Tracking Smoothing: " + 
session.smoothing_controller.ToString("F2");
        }
    }

    /// <summary>
    /// Displays in text value the current smoothing value of the session
    /// </summary>
    /// <param name="session">Session.</param>
    void DisplayCurrentGestureSmoothingValue(Session session)
    {
        if (smoothingSliderControler.activeInHierarchy)
        {
            currentGestureSmoothingValue.text = "Gesture Smoothing: " +
session.gesture_smoothing_controller.ToString("F2");
        }
    }


    /// <summary>
    /// Displayes palm center cursor
    /// </summary>
    /// <param name="palmCenter">Requires the estimated position of the bounding
box center.</param>
    void DisplayPalmCenter(Vector3 palmCenter, GestureInfo gesture, Warning
warning)
    {
        if (ShowPalmCenter)
        {
            if (warning != Warning.WARNING_HAND_NOT_FOUND)
            {
                if (!palmCenterGizmo.activeInHierarchy)
                {
                    palmCenterGizmo.SetActive(true);
                }
                float smoothing = 1 -
ManomotionManager.Instance.Manomotion_Session.smoothing_controller;
```

```
                palmCenterRectTransform.position =
Camera.main.ViewportToScreenPoint(palmCenter);
                float newFillAmmount = 1 - ((int)(gesture.state / 6) * 0.25f);
                palmCenterFillAmmount.localScale =
Vector3.Lerp(palmCenterFillAmmount.localScale, Vector3.one *
newFillAmmount, 0.9f);
            }
            else
            {
                if (palmCenterGizmo.activeInHierarchy)
                {
                    palmCenterGizmo.SetActive(false);
                }
            }
        }
        else
        {
            if (palmCenterGizmo.activeInHierarchy)
            {
                palmCenterGizmo.SetActive(false);
            }
        }
    }

    int maxThumbCounter = 10;
    int minThumbCounter = 0;
    int currentThumbCounter = 0;

    /// <summary>
    /// Display the POI cursor
    /// </summary>
    /// <param name="gesture">Gesture.</param>
    /// <param name="warning">Warning.</param>
    /// <param name="trackingInfo">Tracking info.</param>
    void DisplayPOI(GestureInfo gesture, Warning warning, TrackingInfo
trackingInfo)
    {
        bool isPinchWellDetected = currentThumbCounter > maxThumbCounter / 2;
        if (ShowPOI)
        {
            if (gesture.mano_class == ManoClass.PINCH_GESTURE_FAMILY)
            {
                if (currentThumbCounter < maxThumbCounter)
                {
                    currentThumbCounter++;
                }
            }
            else
            {
                if (currentThumbCounter > minThumbCounter)
                {
                    currentThumbCounter--;
                }
            }

            if (warning != Warning.WARNING_HAND_NOT_FOUND &&
isPinchWellDetected)
            {
```

```
            if (!POIGizmo.activeInHierarchy)
            {
                POIGizmo.SetActive(true);
            }
            float smoothing = 1 -
ManomotionManager.Instance.Manomotion_Session.smoothing_controller;

            poiRectTransform.position =
Camera.main.ViewportToScreenPoint(trackingInfo.poi);

            float newFillAmmount = 1 - ((int)(gesture.state / 6) * 0.25f);
            poiFillAmmount.localScale =
Vector3.Lerp(palmCenterFillAmmount.localScale, Vector3.one *
newFillAmmount, 0.9f);
        }
        else
        {
            if (POIGizmo.activeInHierarchy)
            {
                POIGizmo.SetActive(false);
            }
        }
    }
    else
    {
        if (POIGizmo.activeInHierarchy)
        {
            POIGizmo.SetActive(false);
        }
    }
}

/// <summary>
/// Displays information regarding the detected manoclass
/// </summary>
/// <param name="manoclass">Manoclass.</param>
void DisplayManoclass(ManoClass manoclass)
{
    manoClassGizmo.SetActive(ShowManoClass);
    if (ShowManoClass)
    {
        switch (manoclass)
        {
            case ManoClass.NO_HAND:
                manoClassText.text = "Manoclass: No Hand";
                break;
            case ManoClass.GRAB_GESTURE_FAMILY:
                manoClassText.text = "Manoclass: Grab Class";
                break;
            case ManoClass.PINCH_GESTURE_FAMILY:
                manoClassText.text = "Manoclass: Pinch Class";
                break;
            case ManoClass.POINTER_GESTURE_FAMILY:
                manoClassText.text = "Manoclass: Pointer Class";
                break;
            default:
                manoClassText.text = "Manoclass: No Hand";
                break;
            }
        }
    }


    /// <summary>
    /// Displays information regarding the detected manoclass
    /// </summary>
    /// <param name="manoGestureContinuous">Requires a continuous
Gesture.</param>
    void DisplayContinuousGestures(ManoGestureContinuous
manoGestureContinuous)
    {
        continuousGestureGizmo.SetActive(ShowContinuousGestures);
        if (ShowContinuousGestures)
        {
            switch (manoGestureContinuous)
            {
                case ManoGestureContinuous.CLOSED_HAND_GESTURE:
                    continuousGestureText.text = "Continuous: Closed Hand";
                    break;
                case ManoGestureContinuous.OPEN_HAND_GESTURE:
                    continuousGestureText.text = "Continuous: Open Hand";
                    break;
                case ManoGestureContinuous.HOLD_GESTURE:
                    continuousGestureText.text = "Continuous: Hold";
                    break;
                case ManoGestureContinuous.OPEN_PINCH_GESTURE:
                    continuousGestureText.text = "Continuous: Open Pinch";
                break;
            case ManoGestureContinuous.POINTER_GESTURE:
                continuousGestureText.text = "Continuous: Pointing";
                break;
            case ManoGestureContinuous.NO_GESTURE:
                continuousGestureText.text = "Continuous: None";
                break;
            default:
                continuousGestureText.text = "Continuous: None";
                break;
        }
    }
}

/// <summary>
/// Displays the hand side.
/// </summary>
/// <param name="handside">Requires a ManoMotion Handside.</param>
void DisplayHandSide(HandSide handside)
{
    handSideGizmo.SetActive(ShowHandSide);
    if (ShowHandSide)
    {
        switch (handside)
        {
            case HandSide.Palmside:
                handSideText.text = "Handside: Palm Side";
                break;
                case HandSide.Backside:
                    handSideText.text = "Handside: Back Side";
                    break;
                case HandSide.None:
                    handSideText.text = "Handside: None";
                    break;
                default:
                    handSideText.text = "Handside: None";
                    break;
            }
        }
    }

    ///// <summary>
    ///// Updates the visual information that showcases the hand state (how
open/closed) it is
    ///// </summary>
    ///// <param name="gesture_info"></param>
    void DisplayHandState(int handstate)
    {
        handStatesGizmo.SetActive(ShowHandStates);
        if (ShowHandStates)
        {
            HighlightStatesToStateDetection(handstate);
        }
    }

    ManoGestureTrigger previousTrigger;
    /// <summary>
    /// Display Visual information of the detected trigger gesture.
    /// In the case where a click is intended (Open pinch, Closed Pinch, Open Pinch)
we are clearing out the visual information that are generated from the pick/drop
    /// </summary>
    /// <param name="triggerGesture">Requires an input of trigger gesture.</param>
    void DisplayTriggerGesture(ManoGestureTrigger triggerGesture, TrackingInfo
trackingInfo)
    {
        if (triggerGesture != ManoGestureTrigger.NO_GESTURE)
        {
            if (_showPickTriggerGesture)
            {
                if (triggerGesture == ManoGestureTrigger.PICK)
                {
                    TriggerDisplay(trackingInfo, ManoGestureTrigger.PICK);
                }
            }
            if (_showDropTriggerGesture)
            {
                if (triggerGesture == ManoGestureTrigger.DROP)
                {
                    if (previousTrigger != ManoGestureTrigger.CLICK)
                    {
                        TriggerDisplay(trackingInfo, ManoGestureTrigger.DROP);
                    }
                }
            }
            if (_showClickTriggerGesture)
            {
                if (triggerGesture == ManoGestureTrigger.CLICK)
                {
                    TriggerDisplay(trackingInfo, ManoGestureTrigger.CLICK);
                    if (GameObject.Find("PICK"))
                    {
                        GameObject.Find("PICK").SetActive(false);
                    }
                }
            }
            if (_showGrabTriggerGesture)
            {
                if (triggerGesture == ManoGestureTrigger.GRAB_GESTURE)
                    TriggerDisplay(trackingInfo,
ManoGestureTrigger.GRAB_GESTURE);
            }
            if (_showReleaseTriggerGesture)
            {
                if (triggerGesture == ManoGestureTrigger.RELEASE_GESTURE)
                    TriggerDisplay(trackingInfo,
ManoGestureTrigger.RELEASE_GESTURE);
            }
        }
        previousTrigger = triggerGesture;
    }

    public List<GameObject> triggerObjectPool = new List<GameObject>();
    public int amountToPool = 20;
```

```
/// <summary>
/// Initializes the object pool for trigger gestures.
/// </summary>
private void InitializeTriggerPool()
{
    for (int i = 0; i < amountToPool; i++)
    {
        GameObject newTriggerObject = Instantiate(triggerTextPrefab);
        newTriggerObject.transform.SetParent(transform);
        newTriggerObject.SetActive(false);
        triggerObjectPool.Add(newTriggerObject);
    }
}

/// <summary>
/// Gets the current pooled trigger object.
/// </summary>
/// <returns>The current pooled trigger.</returns>
private GameObject GetCurrentPooledTrigger()
{
    for (int i = 0; i < triggerObjectPool.Count; i++)
    {
        if (!triggerObjectPool[i].activeInHierarchy)
        {
            return triggerObjectPool[i];
        }
    }
    return null;
    }


    /// <summary>
    /// Displays the visual information of the performed trigger gesture.
    /// </summary>
    /// <param name="triggerGesture">Trigger gesture.</param>
    void TriggerDisplay(TrackingInfo trackingInfo, ManoGestureTrigger
triggerGesture)
    {
        if (GetCurrentPooledTrigger())
        {
            GameObject triggerVisualInformation = GetCurrentPooledTrigger();


            triggerVisualInformation.SetActive(true);
            triggerVisualInformation.name = triggerGesture.ToString();

triggerVisualInformation.GetComponent<TriggerGizmo>().InitializeTriggerGizmo
(triggerGesture);
            triggerVisualInformation.GetComponent<RectTransform>().position =
Camera.main.ViewportToScreenPoint(trackingInfo.palm_center);
        }
    }


    /// <summary>
    /// Visualizes the current hand state by coloring white the images up to that value
and turning grey the rest
    /// </summary>
    /// <param name="stateValue">Requires a hand state value to assign the colors
accordingly </param>
    void HighlightStatesToStateDetection(int stateValue)
    {
        for (int i = 0; i < stateImages.Length; i++)
        {
            if (i > stateValue)
            {
                stateImages[i].color = disabledStateColor;
            }
            else
            {
                stateImages[i].color = Color.white;
            }
        }
    }

    /// <summary>
    /// Highlights the edges of the screen according to the warning given by the
ManoMotion Manager
    /// </summary>
    /// <param name="warning">Requires a warning.</param>
    void DisplayApproachingToEdgeFlags(Warning warning)
    {
        if (_showWarnings)
        {
            if (!flagHolderGizmo.activeInHierarchy)
            {
                flagHolderGizmo.SetActive(true);
            }
            rightFlag.SetActive(warning ==
Warning.WARNING_APPROACHING_RIGHT_EDGE);
            topFlag.SetActive(warning ==
Warning.WARNING_APPROACHING_UPPER_EDGE);
            leftFlag.SetActive(warning ==
Warning.WARNING_APPROACHING_LEFT_EDGE);
        }
        else
        {
            if (flagHolderGizmo.activeInHierarchy)
            {
                flagHolderGizmo.SetActive(false);
            }
        }
    }

    /// <summary>
    /// Displayes the smoothing slider.
    /// </summary>
    /// <param name="display">If set to <c>true</c> display.</param>
    public void ShouldDisplaySmoothingSlider(bool display)
    {
        smoothingSliderControler.SetActive(display);
    }

    /// <summary>
    /// Displays the smoothing slider that controls the level of delay applied to the
calculations for Tracking Information.
    /// </summary>
    public void DisplaySmoothingSlider()
    {
        smoothingSliderControler.SetActive(_showSmoothingSlider);
        gestureSmoothingSliderControler.SetActive(_showSmoothingSlider);
    }

    /// <summary>
    /// Initializes the components of the Manoclass,Continuous Gesture and Trigger
Gesture Gizmos
    /// </summary>
    void SetGestureDescriptionParts()
    {
        manoClassText =
manoClassGizmo.transform.Find("Description").GetComponent<Text>();
        handSideText =
handSideGizmo.transform.Find("Description").GetComponent<Text>();
        continuousGestureText =
continuousGestureGizmo.transform.Find("Description").GetComponent<Text>();
        palmCenterRectTransform =
palmCenterGizmo.GetComponent<RectTransform>();
        palmCenterFillAmmount =
palmCenterGizmo.transform.GetChild(0).GetComponent<RectTransform>();

        poiRectTransform = POIGizmo.GetComponent<RectTransform>();
        poiFillAmmount =
POIGizmo.transform.GetChild(0).GetComponent<RectTransform>();
    }

    /// <summary>
    /// Initializes the components for the visual illustration of warnings related to
approaching edges flags.
```

```
    /// </summary>
    void InitializeFlagParts()
    {
        topFlag = flagHolderGizmo.transform.Find("Top").gameObject;
        rightFlag = flagHolderGizmo.transform.Find("Right").gameObject;
        leftFlag = flagHolderGizmo.transform.Find("Left").gameObject;
    }

    #endregion
}
```

Hand collider code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HandCollider : MonoBehaviour
{
    #region Singleton
    private static HandCollider _instance;
    public static HandCollider Instance
    {
        get
        {
            return _instance;
        }

        set
        {
            _instance = value;
        }
    }
    #endregion

    private TrackingInfo tracking;
    public Vector3 currentPosition;

    /// <summary>
    /// Set the hand collider tag.
    /// </summary>
    private void Start()
    {
        gameObject.tag = "Player";
    }
    /// <summary>
    /// Get the tracking information from the ManoMotionManager and set the
position of the hand Collider according to that.
    /// </summary>
    void Update()
    {
        tracking =
ManomotionManager.Instance.Hand_infos[0].hand_info.tracking_info;
        currentPosition = Camera.main.ViewportToWorldPoint(new
Vector3(tracking.palm_center.x, tracking.palm_center.y,
tracking.depth_estimation));
        transform.position = currentPosition;
    }
}
```

Hand information saving code

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System.Runtime.InteropServices;


/// <summary>

/// Warnings are a list of messages that the SDK is providing in order to prevent a
situation where the hand will be not clearly detected.

/// </summary>

public enum Warning

{

   NO_WARNING = 0,

   WARNING_HAND_NOT_FOUND = 1,

   WARNING_APPROACHING_UPPER_EDGE = 4,

   WARNING_APPROACHING_LEFT_EDGE = 5,

   WARNING_APPROACHING_RIGHT_EDGE = 6,

};


/// <summary>

/// The Hand value provides additional information regarding the classification of
the hand regarding if its a right or a left hand

/// </summary>

public enum Hand

{

   LEFT = 0,

   RIGHT = 1,

};
```

```
/// <summary>
/// Contrains information about the hand
/// </summary>
public struct HandInfo
{
   /// <summary>
   /// Information about position
   /// </summary>
   public TrackingInfo tracking_info;

   /// <summary>
   /// Information about gesture
   /// </summary>
   public GestureInfo gesture_info;

   /// <summary>
   /// Warnings of conditions that could mean problems on detection
   /// </summary>
   public Warning warning;
}
```

Hand information unity storage

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Runtime.InteropServices;

/// <summary>
/// Contrains information about the hand
/// </summary>
public struct HandInfoUnity
{
   /// <summary>
   /// Information about position
   /// </summary>
        public HandInfo hand_info;
}
```

The remaining codes for the object instantiation and the procedures as stated above are TCS confidential and thus cannot be shared. But for the sake of recreation of such an application many free open source libraries are available in Unity which can serve as an alternative to the ones provided to me by the industry.

8. FUTURE PROSPECTS OF THE APPLICATION

1. FOREIGN LOCATION GLOBAL MAPPING AND WAYFINDING

Digital image processing has progressed a lot in the recent years and one such application of digital image processing is License plate recognition (LPR). By integration of such technologies, we could create a global/local system to maintain a record of public transports and their routes of operation. This when used in a particular customer's handheld device would provide him with relevant information regarding any public transport by just pointing towards any stationary or moving vehicle. Such an application would be very useful for people going to foreign places with languages not recognized by the customer, cutting the costs of local guides and/or translators and provide an easy wayfinding method even in foreign soil.

2. HAPTIC XR

Any gamer – or anyone with a mobile phone for that matter – is well versed in the art of vibration feedback. Whether you're getting a text or phone call, experiencing the rumble of the machine gun firing in a First-Person Shooter or colliding with another vehicle in a racing game, that sense of vibration and its intensity is very familiar nowadays. But the problem with traditional rumble packs is that the vibration is all on one level, with little variation in its intensity regardless of its cause. That's where haptic feedback comes in. Haptic feedback is all about better simulating what it would feel like to touch or interact with something in real life, allowing precision vibrations that help better represent what's happening in-game. Rather than the blasting vibrations of current controllers, haptic feedback allows there to be a lot more subtlety, ranging from the delicate splashing of raindrops to something more severe like an explosion. Therefore, the development we did for dynamic surfaces to be interactable could be used in an extended reality where virtual objects are present in our real time environment which are mobile and yet give a haptic feedback as if they are real.

3. FIRE CREW AID IN REAL TIME BASIS AR

We all know that the response time and efficiency are the most crucial aspects of a fire crew and its reliability to be successful in its operations. And given that in today's world where we are living in a forest of buildings, each unique and with its own layout plans, the fire crew cannot have 100% efficiency in mapping new environments in an emergency. Plus learning the way around, every possible escape route and al the layouts is nigh impossible for the members of the fire crew. Therefore, an AR based dynamic system which stores all the layouts, all possible escape routes and all the internal plans of the houses in their jurisdictional area could potentially help them. An advanced HUD (Heads Up Display) constantly feeding each and every crew with the relevant information of their surroundings could help them with their operations in the near future.

4. AR BASED SHOPPING

As feature detection becomes more advanced, and more of the human body can be recognized efficiently in the future; our project would be able to help the online garments and accessories shopping industries by being able to map clothes and watches or other accessories to the human body. This would help in giving the customers a trial of their products virtually thus helping in making online shopping for such products much more easier and efficient.

9. REFERENCES AND BIBLIOGRAPHY

- https://en.wikipedia.org/wiki/Augmented_reality
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6982909/
- https://opencv.org/about/
- https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@2.2/manual/index.html
- https://towardsdatascience.com/computer-vision-detecting-objectsusing-haar-cascade-classifier-4585472829a9?gi=6e837b186058
- http://www.willberger.org/cascade-haar-explained/
- https://www.ultraleap.com/company/news/blog/what-is-hapticfeedback/
- https://www.gamesradar.com/haptic-feedback-explained/
- https://en.wikipedia.org/wiki/Haptic_technology
- https://blogs.unity3d.com/2018/12/18/unitys-handheld-ar-ecosystemar-foundation-arcore-and-arkit/
- https://blogs.unity3d.com/2020/02/28/best-practices-for-bringing-arapplications-to-the-field/
- AUGMENTED HUMAN: HOW TECHNOLOGY IS SHAPING THE NEW REALITY BY HELEN PAPAGIANNIS
- THE FOURTH TRANSFORMATION BY ROBERT SCOBLE AND SHEL ISRAEL
- AUGMENTED REALITY: INNOVATIVE PERSPECTIVES ACROSS ART, INDUSTRY, AND ACADEMIA BY JOHN TINNELL AND SEAN MOREY