Chapter 5

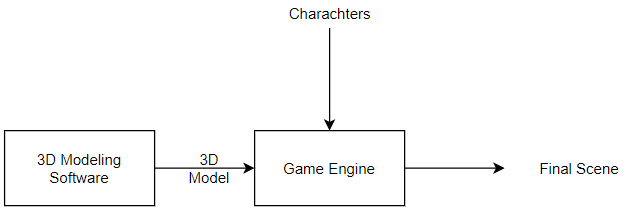
***Graphics***

5.1 Introduction

This chapter focuses on the graphics part of the project. The idea is to build a scene similar to a simple lecture hall to give the user the feeling of authenticity when using the application.

To bring this idea described in Fig. 5.1 to life we have three important tasks the first is to model the scene on some 3D modeling application and then export that scene to a game engine. The second part is to use that game engine to add lighting and further enhancement to the static scene. The third and final part is to add the characters to that scene.

Now that we know the target, we have some problems. What 3D modeling software to use? What game engine is best suitable for our needs and our time plan? Should we design the characters or get some predesigned characters? This chapter mainly answers these questions and gives a walkthrough the development process.



**Figure 5.1** Graphics Phase

5.1.1 Scene Modeling

Designers use 3D modeling programs to make 3D animations, models, games and images with ease. It is used for variety of reasons from architectural design to game design.

3D modeling programs are available to use like 3Ds Max, Maya, Blender and we have to choose one them for our project. We had to decide which one based on ease of use and how long would it take to have it ready and how good is the software actually is.

Our best bet was 3Ds max. Its offers simple interface for both beginners and professionals with some easy to follow tutorials and documentation and we had some previous experience using it so it seemed to be a great choice for our project.

5.1.2 Game Engine

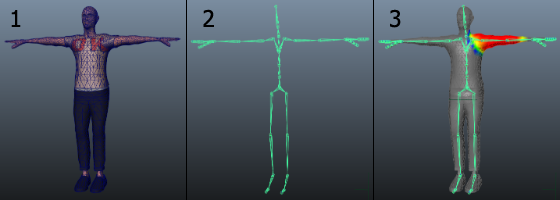
A game engine is a [software-development environment](https://en.wikipedia.org/wiki/Software_development_environment) designed for people to build [video games](https://en.wikipedia.org/wiki/Video_game).

We have to use a game engine to combine the scene - after further enhancing it with lighting, materials and textures - with the characters and their animation and later on to build our application for the android platform. There are many good engines available like Unity or Unreal Game Engine.

Unity has more available resources to learn from and game models and packages to experiment with compared to Unreal Game Engine and the second issue is now resolved.

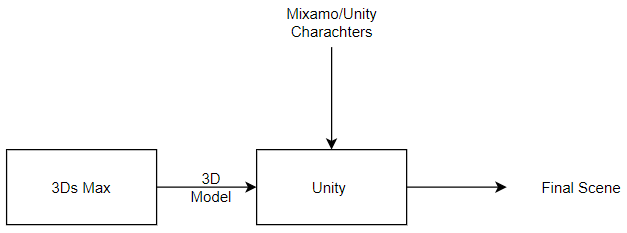
5.1.3 Characters and Animations

After the static scene is built, it is time to add the characters to the scene but that is easier said than done. Character design is hard and tedious process and very time consuming, as we have to use a software to first design the shape, looks of the character, rig it and later export it to Unity. Rigging in Fig. 5.2 means giving the character bones or a skeleton and that is necessary for bringing the characters to life via animations.



**Figure 5.2** Character Design and Rigging

An easy approach to this problem is use free rigged characters that are available on the internet and use them directly rather than designing them from scratch. Mixamo by Adobe characters and Unity’s asset store characters are the best options available. Now that we all questions are answered this is the modified plan in Fig. 5.3.



**Figure 5.3** Graphics Phase (Modified)

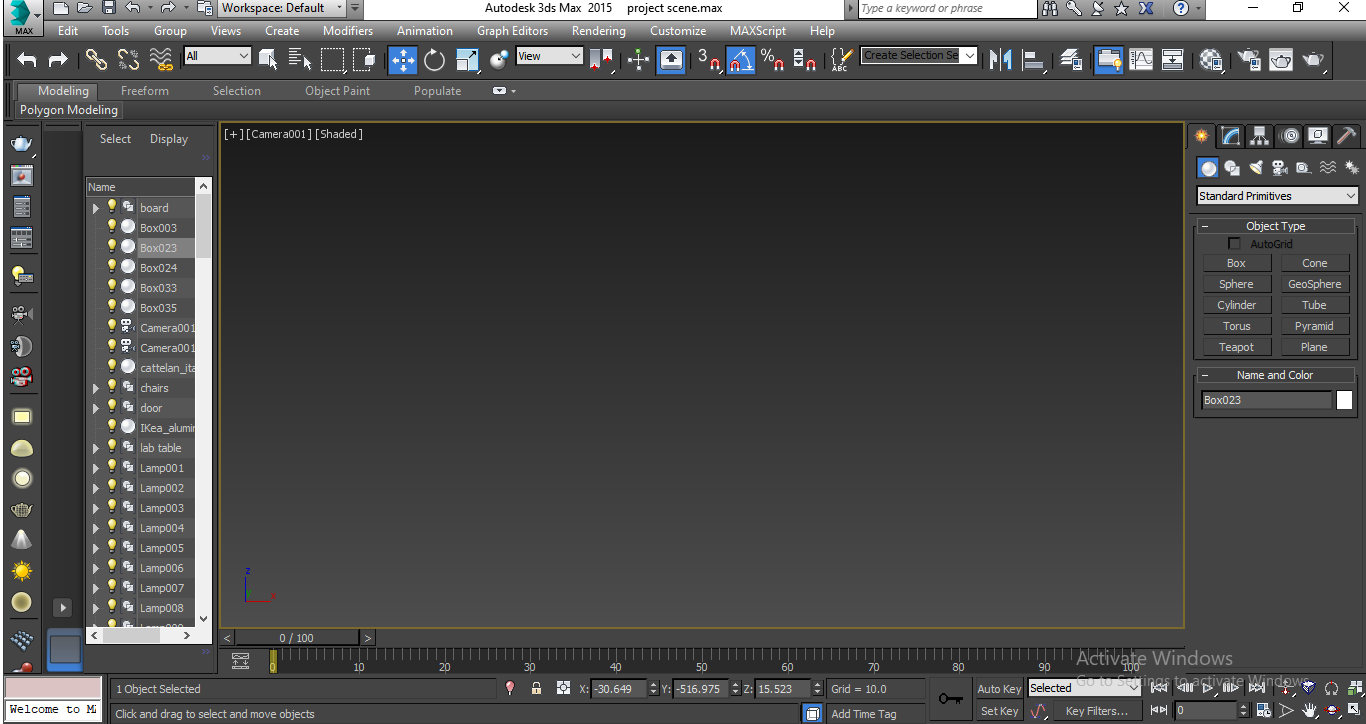
5.2 3Ds Max

A computer graphics program for creating 3D models, animations, and digital images. It is one of the most popular programs in the computer graphics industry and is well known for having a robust tool-set for 3D artists.

We have chosen this program because it is easier to use and it is one of the best programs in design so that it is easy to convert it to FBX to use in Unity.

5.2.1 Interface

Like any other program, it has a workspace to design the scene and many tools to create it also to modify on the scene to make it as we want as shown in Fig 5.4.



**Figure 5.4** 3Ds Max Interface

5.2.2 Designing

The first stage is to design the scene in a comfortable and realistic way. We use some tools to design it like ( boxes , cubes , cylinders ) and many other shapes to design chairs , stairs , board , laptop , projector, clock and also microphone combine them all together to form the scene.

Fig 5.5 shows how we combine those objects to create our scene.



**Figure 5.5** Scene

5.2.3 Lighting and Materials

We had to make the scene more realistic by adding some lighting and materials to each component in it. V-Ray add-on is used to add lighting easily.

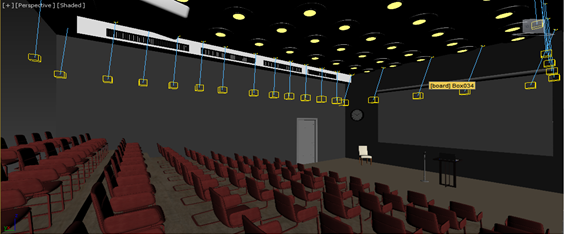
**.** V-Ray

V-Ray for 3ds max offers a number of lights essential for a good render. Whether you are preparing an interior or exterior scene, you can find the appropriate lighting options in the V-Ray toolbar or in the Asset Editor.

**.** Material

We just use an ArchShaders file appending to the program just drag the material and drop on whatever you want.

The combination of light and material is shown in Fig, 5.6.



**Figure 5.6** Materials and Lights

5.2.4 Rendering

The process of generating a photo realistic image from 3D model and the last stage in our process is to create a scene. There are two types of render in 3ds max

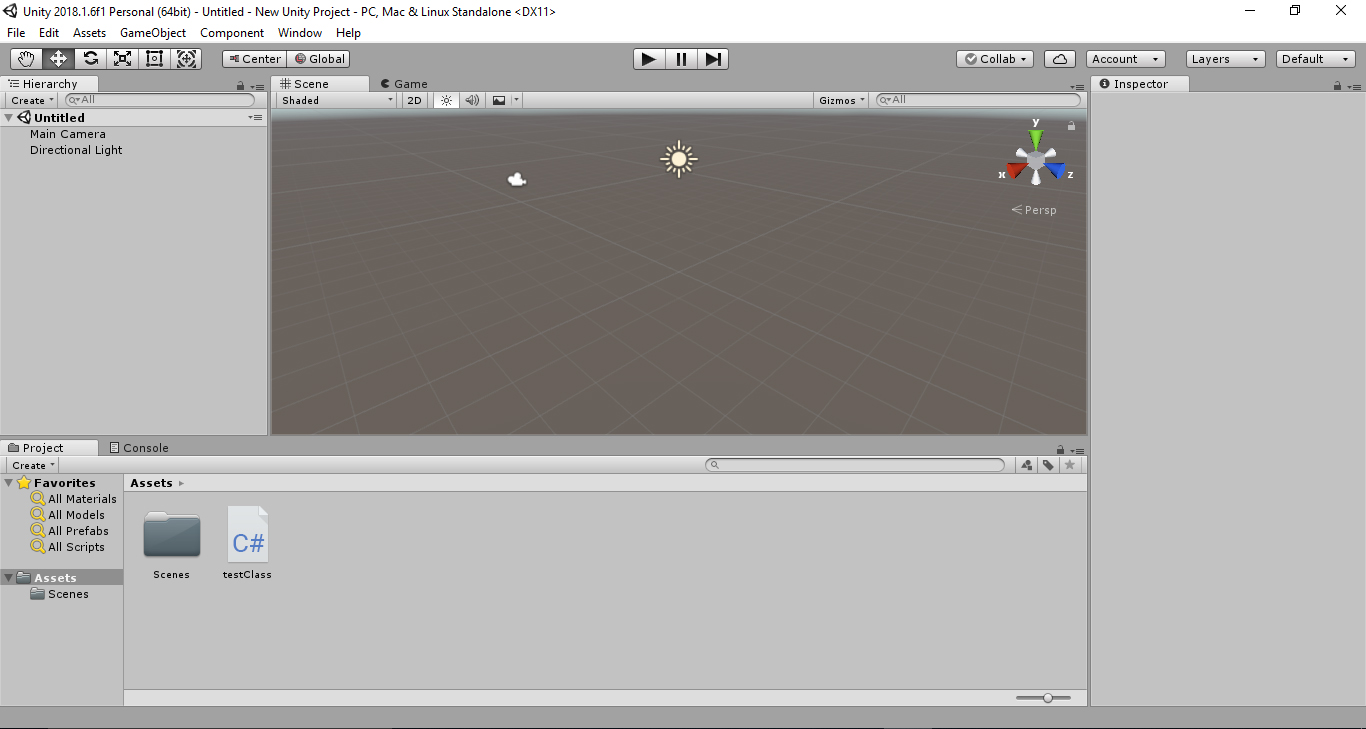
1. Progressive: which generate an image in short time but the resulting image is noisy and not clear.
2. Bucket: which generate a real image without any noises but take too much time to generate it

Here we render the scene using bucket type, using V-Ray render to produce image with lighting and materials.

5.3 Unity

Unity is a 3D/2D game engine and powerful cross-platform IDE for developers. As a game engine, Unity is able to provide many of the most important built-in features that make a game work. That means things like physics, 3D rendering, and collision detection. From a developer’s perspective, this means that there is no need to reinvent the wheel. Rather than starting a new project by creating, a new physics engine from scratch calculating every movement of each material, or the way light should bounce off different surfaces.

Unity is an IDE shown in Fig. 5.7. IDE stands for “integrated development environment,” which describes an interface that gives you access to all the tools you need for development in one place. The Unity software has a visual editor that allows creators to simply drag and drop elements into scenes and then manipulate their properties.



**Figure 5.7** Unity IDE

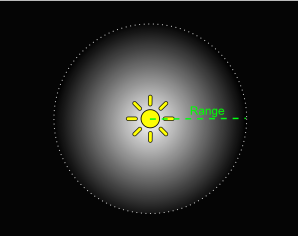
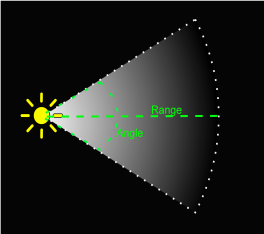
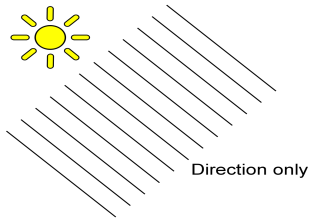
5.3.1 Importing Scene

3Ds Max exports the scene in FBX format, is used in Unity to add some characters, and animate them in some way that characters interact with the presenter to start our project.

We encountered some problems such that the lighting and materials disappeared because 3Ds does not actually export those objects in the FBX format. That meant we have add lighting and materials to the scene once again.

5.3.2 Lighting and Materials

In Unity, we have many tools that help us to add lighting and materials. We use the same ArchShaders files to add materials by a simple drag-and-drop onto any object we want. As for the lights, we use Unity lights like point light, spot light and directional light shown in Fig 5.8.

**Figure 5.8** Unity Lights

Adding materials and lights make the scene more real, comfortable, it simulates real life, and all of that is required to make the presenter believe it is a real scene.

After all is said and done, in Fig 5.9 we see the scene after we fixed it after importing it.



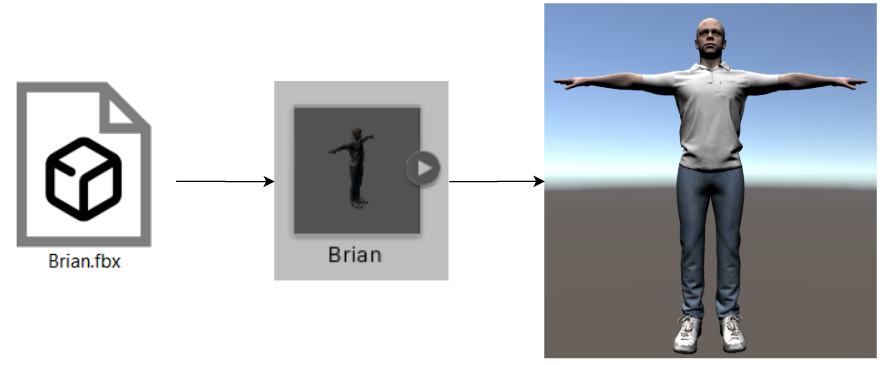
**Figure 5.9** Unity Scene

5.4 Characters

Now that the scene is finally finished in Unity, we now turn our focus to the characters and their facial expressions as that one main aspect of the application and a main indicator of how the user is performing.

5.4.1 Importing Chars. in Unity

As we are using characters from the Unity Assets and Mixamo, we have to import them to the project. This a simply task as we can get those files as .FBX (Filmbox) extension files and that is easily integrated with unity as soon as we import it we get a prefab of our character ready to use which is shown in Fig. 5.10.



**Figure 5.10** Importing Character

5.4.2 Facial Expressions

To control the facial expressions of a character we have to control the muscles of the face and the basic rigging does not offer that option, to overcome this problem there are two options, the first is to enhance the rig to control every bit of the face, but that is a very hard and time consuming process. The second choice is to use a special option that Unity offers which is Blend shapes. Every character with a Blend shape object attached to it can be manipulated in many more ways compared to the normal rig specially the facial expressions and that can be done easily via sliders as shown in Figure 5.11.

That ease of usage comes at the cost of more memory usage, which we shall discuss more in detail later in optimization in this chapter, but for now, we can simply use a few characters with the Blendshape settings and fill the remainder of the scene with normal rigged Mixamo characters



**Figure 5.11** Blendshape Settings

Fig 5.12 shows the scene after we added the characters from both Unity assets and Mixamo.



**Figure 5.12** Scene with Characters

5.5 Animation

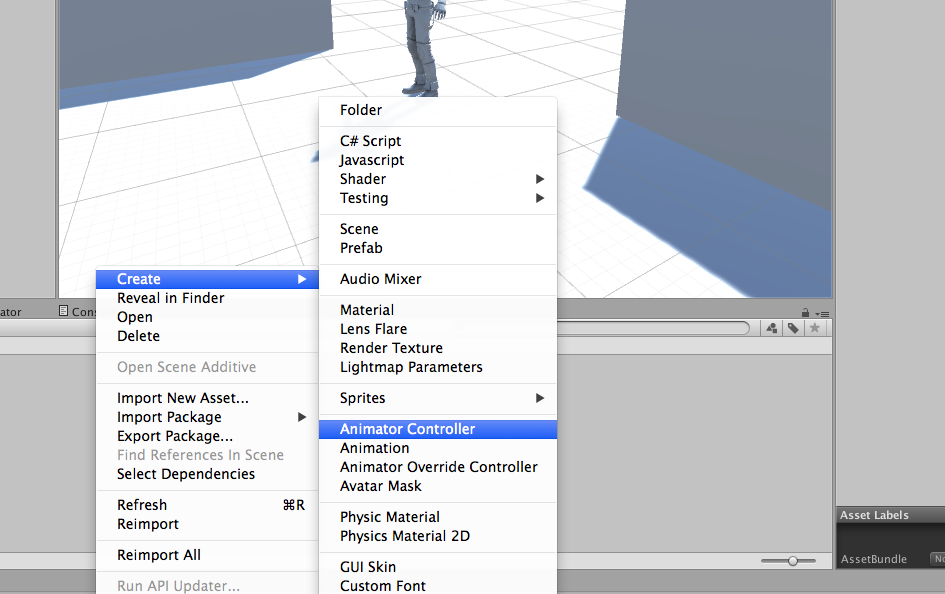
Animating of characters is the technique of figures being manipulated to appear as if moving. In our project, we use animation to make the characters as real and as interactive as we can on a virtual platform. The character can be manipulated to reach the reaction we need to achieve, for example, raising its hand, imitating talking and many more animations that can be applied.

As we discussed in part 5.1.3, characters should be rigged so we can manipulate the whole character and make it move we can make the animation we want, but this is very extensive and time consuming, so there are various standard animations that can be downloaded and imported in Unity and works on some characters, here we used the Mixamo Animations.

5.5.1 How animations work on characters

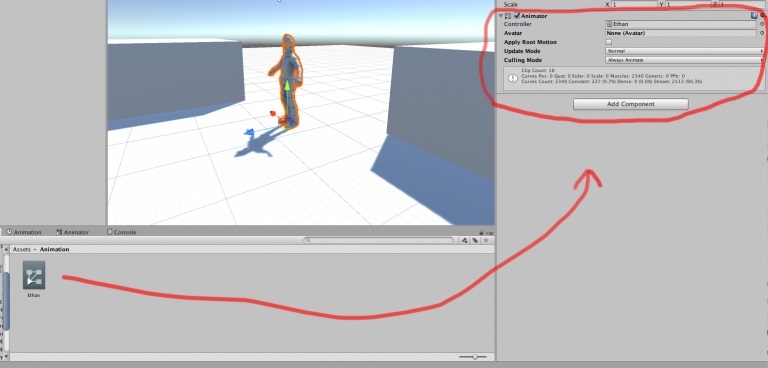
First characters should be rigged, and in our project, we have already rigged characters from Mixamo and from Blendershapes, so we start by downloading the characters and importing them to Unity.

For animations, we use something called the Animator. To start the animator, we create a folder in the Assets folder called Animation, right click and go to Create -> “Animator Controller”. The process is shown in Fig 5.13.



**Figure 5.13** Animator Controller

We name the character controller, preferably the character’s name and drag our new Animator Controller on the character’s model as shown in Fig 5.14.

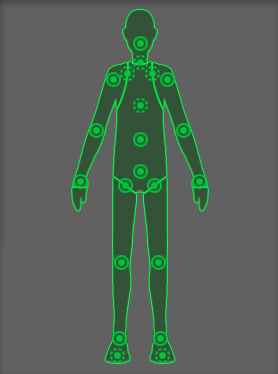


**Figure 5.14** Animator on Character

The Animator Controller allows you to assign multiple animations to a model. For humanoid characters, such as ours, it requires something called an “Avatar” seen in Fig 5.15.

[The Avatar system](https://docs.unity3d.com/Manual/ConfiguringtheAvatar.html) is how Unity identifies that a particular animated model is humanoid in layout, and which parts of the model correspond to the legs, arms, head and body.

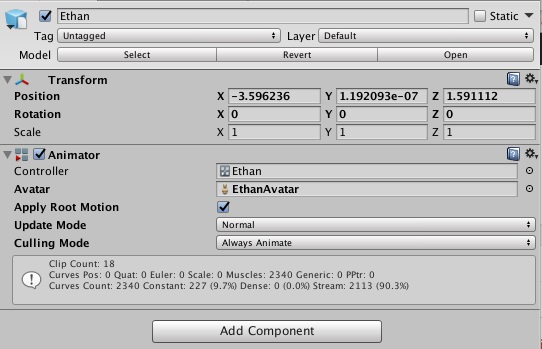
Because of the similarity in bone structure between different humanoid characters, it is possible to map animations from one humanoid character to another, allowing **retargeting** and **inverse**kinematics (IK).



**Figure 5.15** Avatar

The characters we have in our project, already have their avatars.

In the character’s model, Fig 5.16 shows some of the settings in the Animator component.



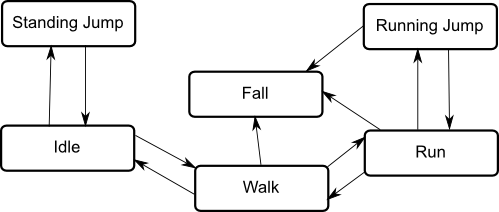
**Figure 5.16** Animator Settings

The “Apply Root Motion” Boolean determines if Unity will use the movement in the animation or a script. For example, say I have a script that makes my character move forward. I have an animation attached of the character running in place. Since the script is moving the character not the animation then I would want Root Motion set to false.

 With “Update Mode” you can determine how you want your animations to play. Such as with physics, with a fixed speed, or just normal. Culling mode allows you to determine if you want to keep animating even off screen.

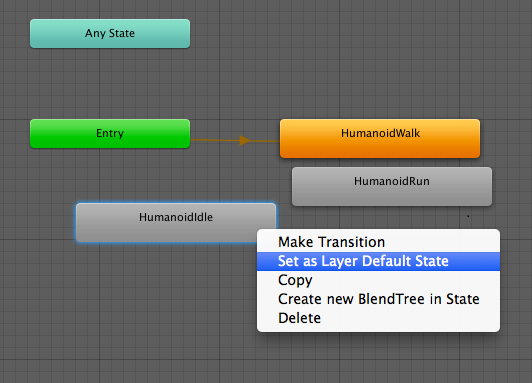
The character can move from one animation to another and change its state using State Machine Diagram an example is shown in Fig 5.17.

The basic idea is that a character is engaged in some particular kind of action at any given time. The actions available will depend on the type of gameplay but typical actions include things like idling, walking, running, jumping, etc. These actions are referred to as **states**, in the sense that the character is in a “state” where it is walking, idling or whatever. In general, the character will have restrictions on the next state it can go to rather than being able to switch immediately from any state to any other. For example, a running jump can only be taken when the character is already running and not when it is at a standstill, so it should never switch straight from the idle state to the running jump state. The options for the next state that a character can enter from its current state are referred to as [state transitions](https://docs.unity3d.com/Manual/class-Transition.html). Taken together, the set of states, the set of transitions and the variable to remember the current state form a **state machine**.



**Figure 5.17** State Machine Diagram

To make our state machine we drag our states into the animator tab shown in Fig 5.18, the orange one is the first one you selected in your Project tab. That is known a “Default State”. We can also change this default state by right clicking the desires state and choosing “Set as Layer Default State”



**Figure 5.18** Animator State Diagram

The character is by default in that state and if we want to make a transition to another state, right click on the Default State and select “Make transition”.  Then click on the desired state.

So now, if we play our animation we will find that the character is idle for some time and then starts the next state for example “Walking”.

We have our model animating but it lacks control. It idles and runs without us doing anything. The way we add control is through Parameters. Parameters are variables that are defined within the animator that scripts can access and assign values to; this is how a script can interact with the Animator. There are three types: Int (or Integer), Float, Boolean, and Trigger. All of them are self-explanatory except for Trigger. A trigger is like a Boolean but as soon as it is set to true it immediately goes back to false. Parameters can be assigned values from a script using functions in the Animator class: [SetFloat](https://docs.unity3d.com/ScriptReference/Animator.SetFloat.html), [SetInteger](https://docs.unity3d.com/ScriptReference/Animator.SetInteger.html), [SetBool](https://docs.unity3d.com/ScriptReference/Animator.SetBool.html), [SetTrigger](https://docs.unity3d.com/ScriptReference/Animator.SetTrigger.html) and [ResetTrigger](https://docs.unity3d.com/ScriptReference/Animator.ResetTrigger.html).

By using parameters, we can check if a Boolean flag is true or false and transit to another state according to the value of flag.

5.5.2 Animation Layers

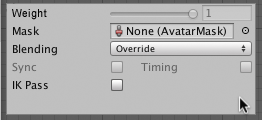
Unity uses Animation Layers for managing complex state machines for different body parts. An example of this is if you have a lower-body layer for walking-jumping, and an upper-body layer for throwing objects / shooting.

You can manage animation layers from the Layers Widget shown in Fig 5.19 in the top-left corner of the Animator Controller



**Figure 5.19** Layers Tab  
.

Clicking the gear wheel on the right of the window shows you the settings as seen in Fig 5.20 for this layer.

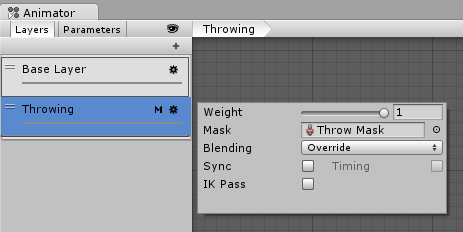


**Figure 5.20** Layer Settings

On each layer, you can specify the mask (the part of the animated model on which the animation would be applied), and the Blending type. Override means information from other layers will be ignored, while Additive means that the animation will be added on top of previous layers.

You can add a new layer by pressing the + above the widget.

The Mask property is there to [specify the mask used on this layer](https://docs.unity3d.com/Manual/class-AvatarMask.html). For example if you wanted to play a throwing animation on just the upper body of your model, while having your character also able to walk, run or stand still at the same time, you would use a mask on the layer which plays the throwing animation where the upper body sections are defined an example is shown Fig 5.21.

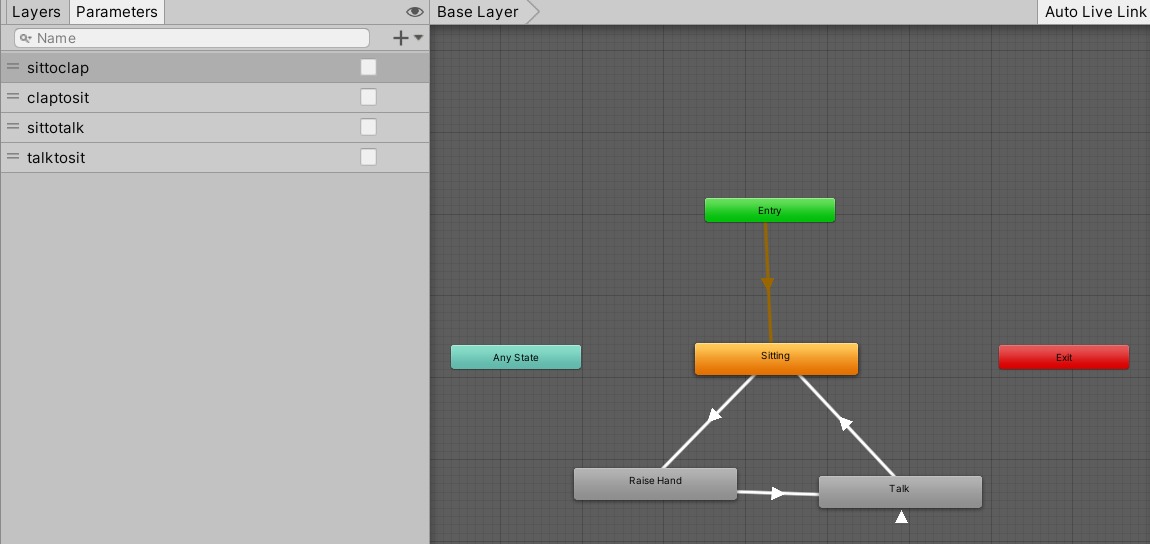


**Figure 5.21** Controlling Layers

An ‘M’ symbol is visible in the Layers sidebar to indicate the layer has a mask applied.

5.5.3 Animations used in this Project

We have two layers in the project, the Base Layer and the Face Layer. The **Base layer** shown in Fig 5.22 is responsible for the body movements of the characters it consists of three states: Idle Sitting, Raise Hand, and Talk.



**Figure 5.22** Base Layer

* Idle Sitting

Idle sitting is the animation that works for most of the application, where characters are basically sitting in an idle position, the Mixamo animation makes them appear as it they are breathing, and this animation works on all characters unless it is invoked by any other flag that causes it to enter into another state of the state machine.

* Raise Hand

This animation in Fig. 5.23 is invoked if the sit-to-talk flag, which is a Boolean parameter, here the character raises its hand to ask a question.



**Figure 5.23** Raise Hand

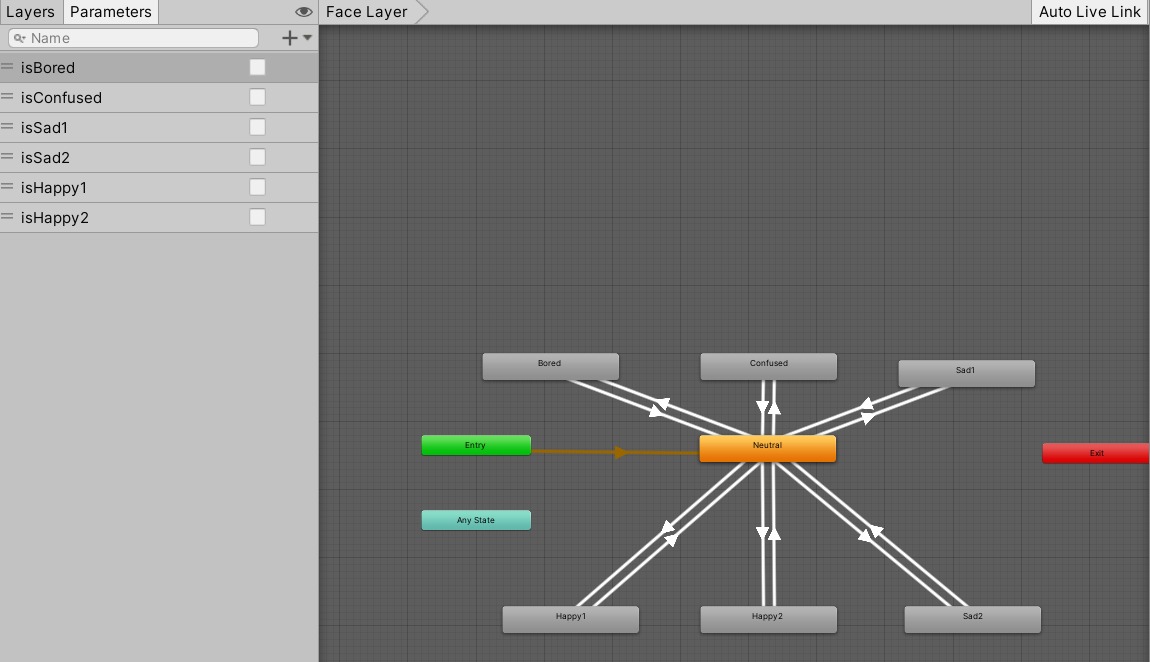
* Talk

This animation in Fig 5.24 is invoked after the Raise Hand animation is exited, here the character starts imitating talking and asks the question we got from the model.



**Figure 5.24** Talking

The **Face layer** shown in Fig 5.25 is responsible for the Facial Expressions of the characters it consists of three states: Neutral, Confused, Sad, Happy, and Bored.



**Figure 5.25** Face Layer

We transit from these states according to the speech rate, if it is a good rate the is-Happy Boolean is set to true and the character makes a happy face in Fig 5.26 to reflect satisfaction.

**Figure 5.26** Happy Faces

But if the speech rate is too slow or too fast the is-Sad Boolean is set to true and the facial expressions reflect not being satisfied as seen in Fig. 5.27.

**Figure 5.27** Unsatisfied Faces

5.6 Optimization

The hall scene is finally complete and filled with characters but it is not optimized at all and it is even laggy with low frame late even when run on PC, Which means it will only run slower on android devices especially older models.

The reason for this laggy performance because we are making a lot of calls to the GPU card to draw every single object in our scene every single frame, which is consuming and we can reduce those calls greatly with some techniques, which we will discuss in the part of the chapter.

5.6.1 Draw Calls

We have to first analyze what are draw calls and how many of them are too bad or more than what the GPU can handle, To draw a GameObject on the screen, the engine has to issue a draw call to the graphics API (such as OpenGL or Direct3D). Draw calls are often resource-intensive, with the graphics API doing significant work for every draw call, causing performance overhead on the CPU side. This is mostly caused by the state changes done between the draw calls (such as switching to a different Material), which causes resource-intensive validation and translation steps in the graphics driver.

Now that we now what is a draw call, it is obvious that we have to reduce their number as much as possible to get a smoother feeling to our application with more than 50-60 fps. On computer devices we can get that frame rate with as many as 1000-2000 draw calls made from Unity, but that is not the case for mobile devices as we are limited to about 120-160 draw calls and that is only available on the newer models of mobile devices which we are targeting currently.

The lecture hall now uses about 2200 draw calls, we have to reduce that to be in the range of 120-160 draw calls, and we are going to discuss how to achieve that in the following bits of this section.

5.6.2 Draw Calls Reduction Techniques

Fig. 5.28 shows the current state of the application as seen from stats window in Unity. We can see we have 2216 draw calls and the objects drawn consist of 6.6 million vertices. We have to reduce both the draw calls and number of vertices. The FPS is currently in the range 60-80, but on android, it does not reach 15.

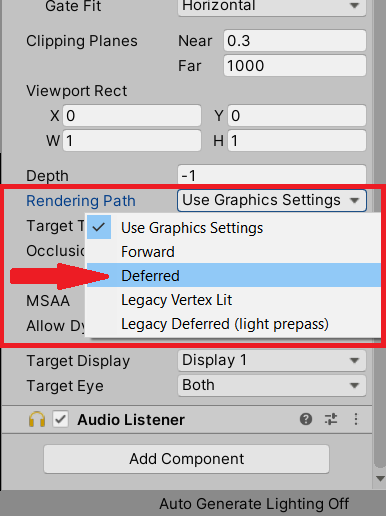


**Figure 5.28** Scene Stats

Now we begin to explore how to make our application suitable for mobile devices via some simple methods that can reduce the draw calls greatly without affecting the quality too much.

* **Camera**

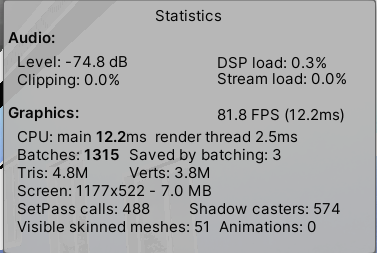
The camera control the quality of the drawn images so we can reduce that quality a bit since the mobile devices are usually so small so it won’t affect the quality that much. Fig 5.29 shows the camera settings and how we can choose a different rendering path.

**Figure 5.29** Camera Rendering Path **Figure 5.30** Camera Sensor Size

The camera angle controls how much we see so if we can reduce the angle and cover some object this means we can remove those objects from our draw calls. We can modify this setting in the camera settings shown in Fig. 5.30.

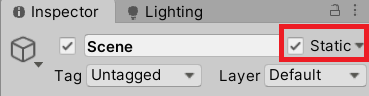
Fig. 5.31 shows the new stats we achieved after modifying the previous settings and we are already off to a great start.



**Figure 5.31** Modified Stats

* **Static Objects**

Each object is responsible for a draw call but the scene is static meaning that no object in it changes it’s position as time passes. We can take advantage of that by marking the scene objects as static – except characters – in the object settings shown in Fig 5.32 to minimize draw calls. If an object is marked as static it’s properties get calculated and drawn only once as we always know it’s position.

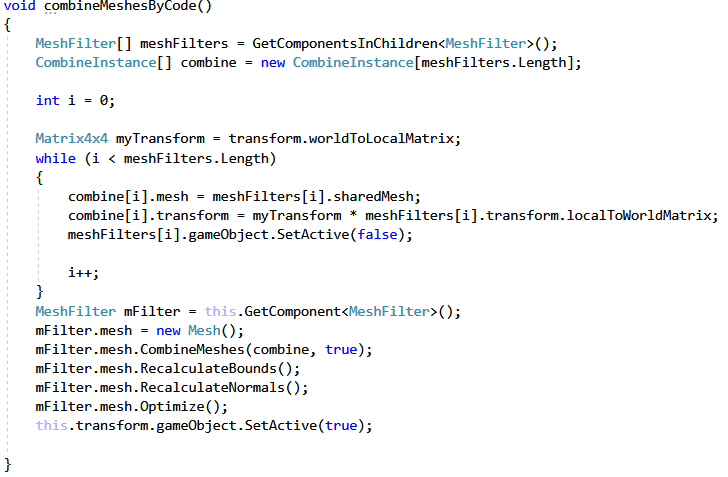


**Figure 5.32** Static Object

* **Mesh Combining**

As we mentioned before every object get a draw call but some objects are identical to each other like the chairs, walls and others. We can take advantage of that and combine those objects or combine their meshes to have them be drawn as a single object at run time.

The idea discussed above can be achieved in two ways. The first is to design the object as a single item but that is not always possible and hard to achieve. The second method is easier and achieved through a simple script shown in Fig. 5.33.

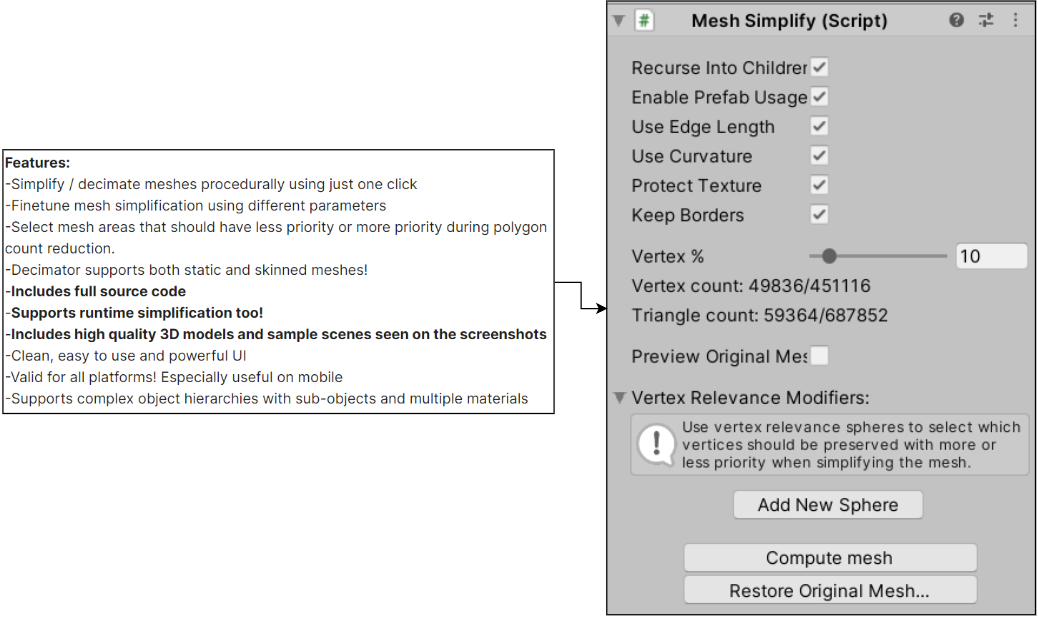
5

**Figure 5.33** Mesh Combining Script

The main advantage of this script is that if we 1000 identical objects they would cause 1000 draw calls but if we combine them they would all require only a single draw call.

* Mesh Simplification

Surface mesh simplification is the process of reducing the number of faces used in a surface mesh while keeping the overall shape, volume and boundaries preserved as much as possible. Although it does not decrease draw calls count but it does has two features it simplifies the amount of work done in a draw call since the mesh is simpler than it was. The second and more important feature is that Unity cannot combine meshes with more than 64,000 Vertices so sometimes mesh simplification is a necessary step before combining meshes. Fig. 5.34 shows a package called “Mesh Simplify” and the settings as it reduces a mesh and the percentage of vertices we want to keep of the original mesh.



**Figure 5.34** Mesh Simplify Features

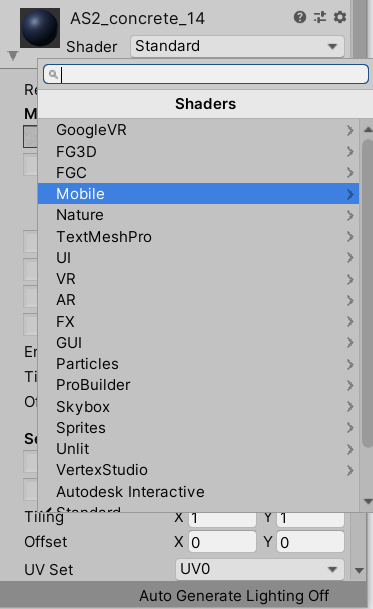
* Materials

Similar to static objects we can allow GPU to make an instance of that material and use it one time to draw all the objects in the scene that use this material all in a single swoop. This is done by enabling GPU instancing in materials settings shown in figure 5.35.



**Figure 5.35** GPU Instancing

Another enhancement that we can do is choose a suitable shader type. A shader is a type of computer program originally used for shading in 3D scenes. Depending on the target platform, we can change the shader to a less memory consuming option, in our case we can choose Mobile/Diffuse shader. Different options for shaders are shown in figure 5.36.



**Figure 5.36** Shader Types

* Lighting

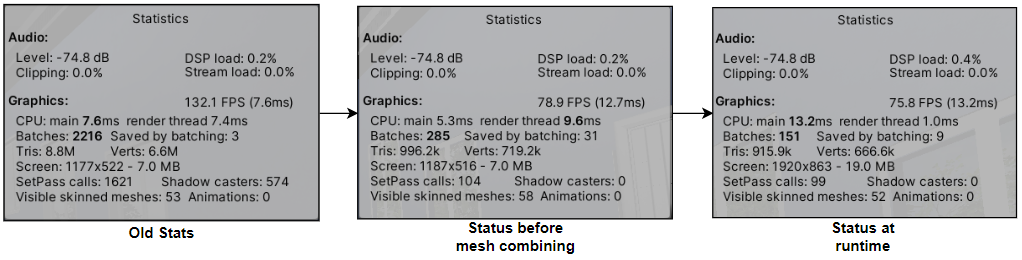
The last thing we can enhance is the light settings. Light directions and shadows are calculated during runtime and that adds an overhead to the draw calls but our scene is mostly static so we always know where is the light hitting the objects and because of that we can set the light mode to Baked which means the light is calculated only once as we always know it’s settings. We can also turn of the shadows as they add an overhead to the calculations as well. Fig 5.37 shows the modified light settings we are using.



Figure 5.37 Light Settings

5.6.3 Conclusion

Now it is time to reflect back on what we discussed in optimization and how it affected the performance of our application. First, we need to understand that it is a tradeoff between quality and performance and we need to figure the perfect balance to reach best performance with best quality. Fig 5.38 shows a comparison between the numbers we started with as shown previously in Fig 5.28. These results are achieved by combining all of the above techniques.



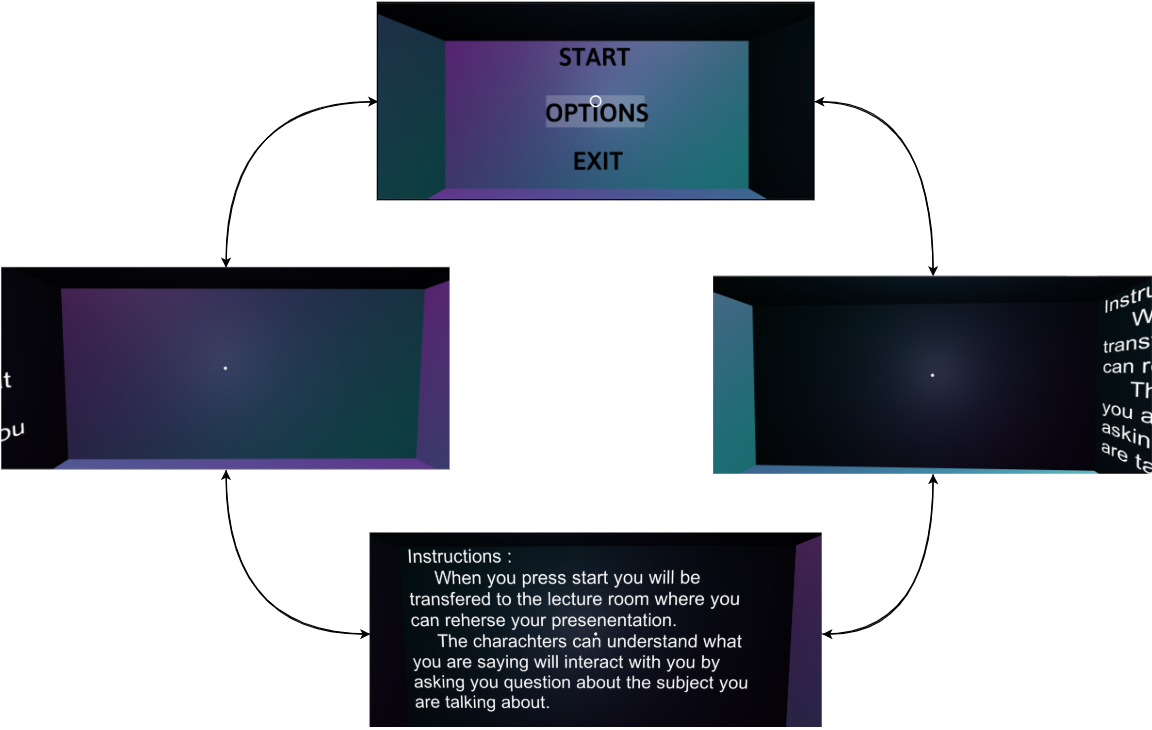
**Figure 5.38** Modified Stats

5.7 Final Presentation

The scene is now all finished and functional but we need to polish it for a better user experience. The first thing we notice that we launch directly into the hall scene with no start screen. We can easily fix this by adding a simply starting screen or room in our case since we are in VR and add a start button and some simple options. The second thing we are going to do is add tree and a sky to our lecture scene to give more of real life feeling.

5.7.1 Starting Room

Fig 5.39 shows a simple hollow cube with our buttons and instructions all laid out on the walls of sides of that cube.



**Figure 5.39** 360° Rotation of Starting Room

As for the options menu, Fig 5.40 shows another 360 view of the starting menu after we choose the click the menu button.



**Figure 5.40** 360° Rotation of Options Screen

The server IP is currently static so we want to change it during run time to be able to receive question from the UNILM model.

The camera distance refers to the distance between the distance between the camera and the first line of audience and the user is able to change based on their liking and level of comfort.

5.7.2 Main Scene Polishing

To make the user make more at a real life scene we added some simple items to make it more realistic below we explore those items:

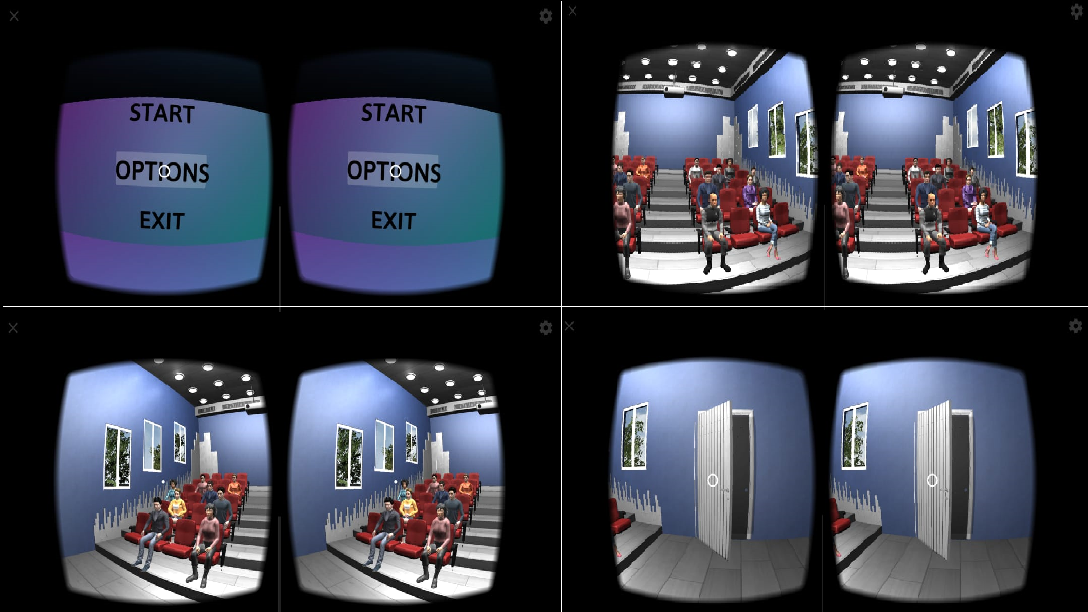
* Tree and a Skybox
* Laptop image
* Textbox for feedback on speech’s speed

Fig. 5.41 shows the final scene image during runtime.



**Fig 5.41** Final Scene

Fig 5.42 shows the application on an android device.



**Figure 5.42** Mobile Application