NOTE: Please be sure you've watched the following videos before you get started to familiarize yourself with the basics of getting around in the UE4 engine:

- VR Project Setup
- The View Port [Video Guide]
- Placing Objects [Video Guide]

In this tutorial, we'll briefly go over how to use (and abuse) the UE4 virtual reality templates. Which let us rapidly create a VR experience that is playable on a variety of hardware. It's almost too easy.

WARNING: Recall what we spoke about earlier. These templates are essentially a game engine within a game engine, so they can limit you drastically in terms of creativity. That said, they are extremely useful if you just want to accomplish the following:

1. Build a VR-ready 3D environment.
2. Move around within that environment while wearing a headset.
3. Pick up objects and have them respond naturally when dropped.

They are great starting points while you're prototyping something, but just be aware they can limit you. That is until we fully dive in to how they are made (which we'll do in a subsequent tutorial).

First, create a new project. Choose the Virtual Reality Template, name it whatever you want and include Starter Content.
After the engine loads up, you'll see a sample tutorial map that details the two templates that are included. These differ in a variety of ways. One template makes use of motion controllers (hand controllers) to provide users with teleportation around the environment, and the other template includes gaze-based motion.

In our case, we’re working with motion controllers, go ahead and open the motion controller map located in /VirtualRealityBP/Maps/.
STOP! Before you do anything else, make sure your VR headset is ready, and select **VR Preview** from the dropdown next to the **Play** icon in the toolbar.

To teleport, you can press the **Trackpad** (9). To pick up the cubes, use the **Trigger** (7).

After you've explored the template in VR, we can briefly go over all of the goodies here and their purpose.

**View Port, Hierarchy and Geometry**

The best way to understand the template is to examine what’s included in the level hierarchy. In this case we have the following types of objects:
What we have is basically the following:
- **Lighting**: A direction light source (e.g., the sun). We can see shadows being cast by this. The Sky Light captures the distant parts of your level and applies that to the scene as a light.

- **Geometry**: The floor, walls and gray/white boxes, the blue cubes (they are special)

- **A Camera**: This is our VRPawn (the thing that we control as the player in the VR world)

- **Interactable Objects**: The blue cubes

- **Volumes**: The NavModifierVolume stops the player from teleporting into those locations. The NavMeshBounds. The PostProcessVolume controls what types of rendering effects are applied to the scene or the volume, and the LightmassImportanceVolume controls how much of the complex (and beautiful) light interactions that you can create in UE4 are rendered at a high resolution.

- **Sky**: The SkySphereBlueprint controls the look and feel of our sky and clouds.

- Some text.

**The VR Pawn**

We’ll cover Pawns in more detail later, but just know that the **Pawn** can be controlled by players or AI. A Pawn is the physical representation of a player or AI entity within the world. This not only means that the Pawn determines what the player or AI entity looks like visually, but also how it interacts with the world in terms of collisions and other physical interactions. The Pawn represents the player’s or entity’s physical location, rotation, etc. within the world. This is a **critical thing to understand**.

The reason the Pawn’s camera is in the middle of the floor is that the actual height of the camera will be the physical height of the player (provided SteamVR or the Oculus software has been configured correctly).
If you are feeling particularly adventurous, go ahead and click on **Edit MotionController Pawn** in the **World Outliner**

Okay, close that window for now, we’ll be spending quite a bit of time in it soon, especially when we are building interactions and other types of interactable objects.

**Editor Cube (Actor)**

The 'Editor Cube' is an Actor in the Unreal Engine. Actors are extremely important too.

**What is an Actor?**

An **Actor** is any object that can be placed into a level. Actors are generally anything that is needed for gameplay, they can be dynamic, static, invisible (e.g., just a sound or a trigger). Actors can be created (spawned) and destroyed through gameplay code. Almost everything that is not a simple piece of geometry or a light in your level is an actor.
Actors can be thought of as components that can hold many different types of objects (e.g., a car could be an actor, but within the car you’d have:

- an engine (the blueprints or code that control how it responds to input)
- 4 wheels (these may spin at certain speeds depending on the current velocity of the car actor)
- a sound (which changes depending on the speed of the car and the type of terrain it’s driving over)

etc…

All of these components (code, the 4 wheels, the sound sources) would be a part of the Car Actor.

Volumes

The template has a variety of volumes (navigation, lightmass and post-processing) that are already set up. In general, you don’t need the post-processing or lightmass importance volume (well, you do when you’re working on a larger level). You absolutely need the navigation volume to use the teleport locomotion in the scene.

Building for VR
Lastly, let’s make sure we can actually run the project. Go to File > Package Project > Windows > Windows (64-bit)

NOTE: Make sure you’ve installed Visual Studio 2017 and included C++ tools before you try and build the template. It will crash and burn if you don’t :(

Oftentimes, you’ll want to package your VR project and share it, or test it before you publish it. In order to do this, you’ll need to build your project (compile it) for a platform (in this case, Windows 64-bit).

With your project open, got to Project Settings

![Project Settings](image)

Give your project a description and name

![Project Settings](image)

Scroll to Settings and check start in VR

![Project Settings](image)

Select Maps and Modes
The **MotionControllerMap** is already set by default (but in the future, you may want to change the starting map to another level).

Scroll down until you find the **Platforms** heading and select **Windows**.

Optional: Add a **game splash screen image** (600px x 200px) and an **icon** (256px x 256px)
Now, we are ready to begin the **Build Process**. Go to **File > Package Project > Windows > Windows (64-bit)**

Create a new folder called **Build** and **Select** it.
UE4 will now start to build. This can take anywhere from 10 seconds to 10 minutes depending on the size of your project.

Once that is all done, you’ll find the standalone executable in your **Build Directory**. Browse to your project directory and find the Build folder, or click the link to the compiled build in the output log popup in UE4.

Your compiled game is inside the Build/WindowsNoEditor directory.
Select all of the files and make a .zip file, this file can be shared, and the resulting Application (in this case, MyProject3) can be run by anyone with a VR headset.

That’s it, you are now a VR developer!