

# Unity Overview

---

# Why Unity?

---

Unity is a game engine for 2D and 3D games that allows you to create experiences and games without writing all of the logic and components from scratch.

- Standardizes physics, imports, shading, lighting computation, and much more
- Robust set of build system tools – can export to nearly any platform
- Extensible plug-in system to create add-ons for hardware components
- Visual development interface

# Unity for VR Development

---

Unity 5.1 has support for virtual reality development built-in

Oculus, HTC Vive, Cardboard, and most of the peripheral devices have Unity plug-ins to create *prefabs*, objects that are bundled together to create a more complicated item

Test using VR headsets directly

Some older versions of Unity (generally 4.5+) will support VR SDKs

If you don't have Unity installed, it is highly recommended that you begin the download now, especially if you think that you may want to use Unity for the final project.

# Important Unity Terminology

---

Scene – A collection of different components that make up a specific environment or level

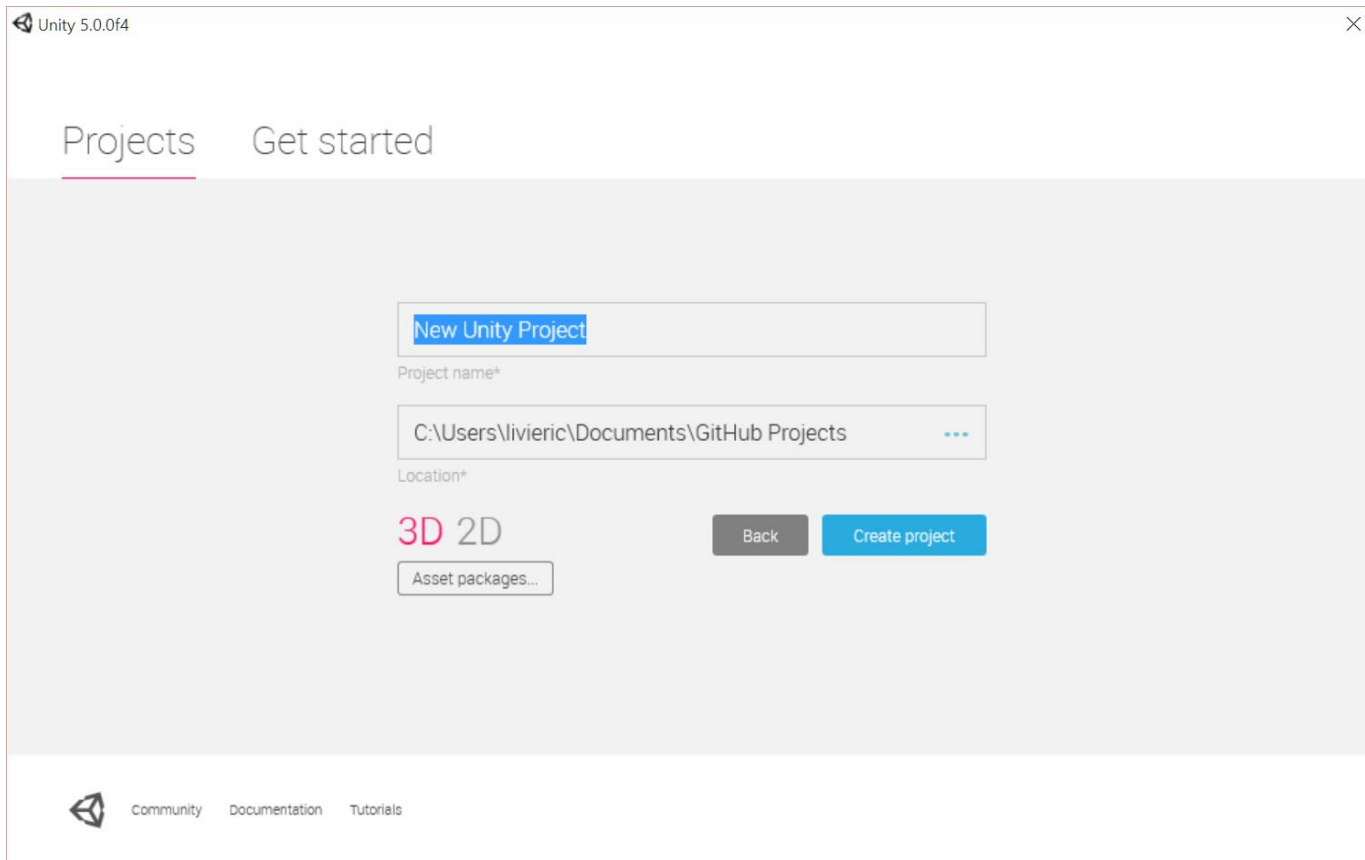
Game Object – An individual object that is within a game scene

Component – An aspect of a game object that gives it certain properties. This can be related to the way the object “behaves”, or its’ appearance

Script- A code file that gives a game object certain behaviors

Asset – an item that is used within your game (materials, textures, meshes, scripts, etc.)

# The Basics of a Unity Project



A Unity Project contains all of the required files for making your game / app

For VR, you should select the 3D option

You can add in various asset packages at the start to avoid entering them manually

# Creating a Project

---

1. Add simple primitive GameObjects to your scene(s)
2. Create scripts and add behaviors to your objects
3. Change the mesh objects to more complex items
4. Add materials and textures to change the appearance of items
5. Test throughout the process!
6. Build your project for your desired platform
7. Play on!

# Editor Modes

---

## CREATE MODE

Modify your scene and view from different angles

View gridlines

Easily modify and move objects around

Non-visible objects are “visible”

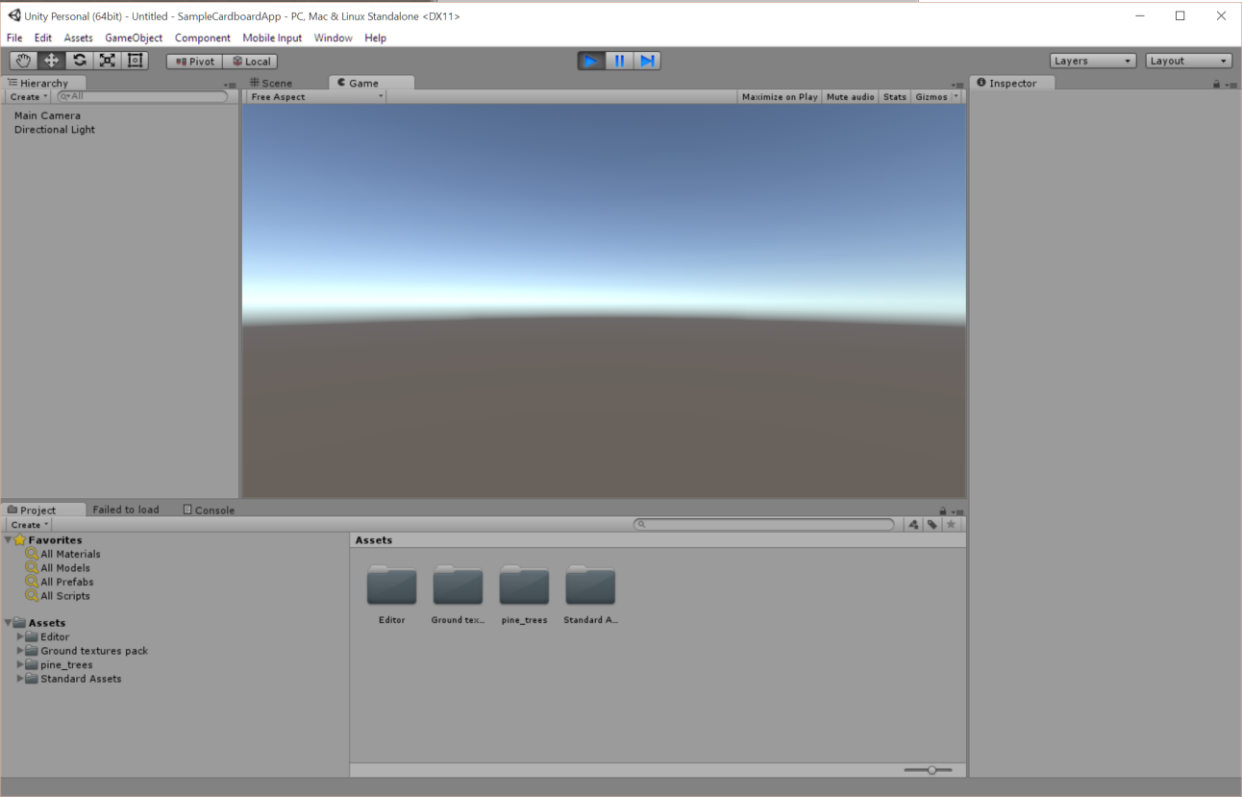
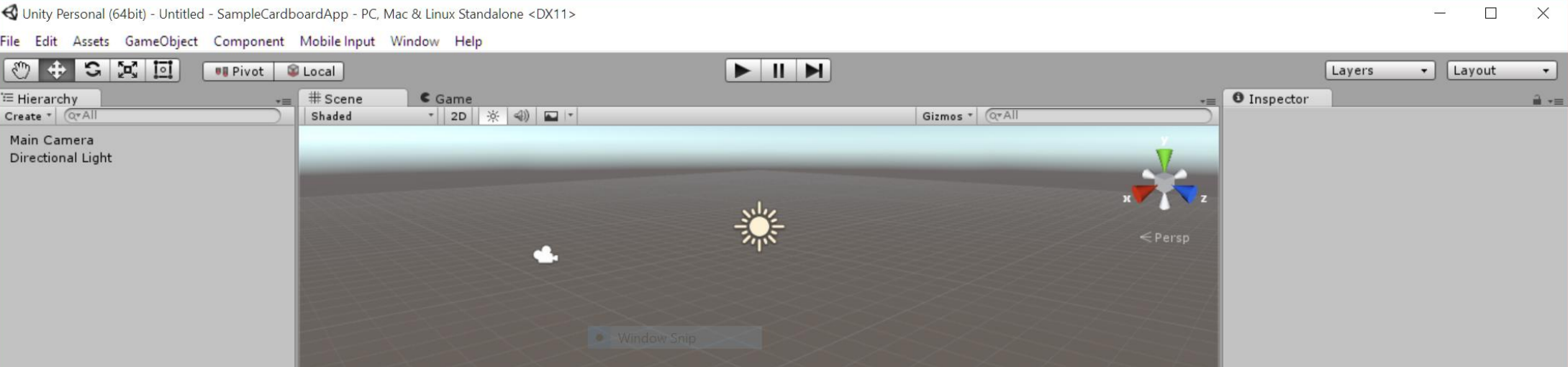
## GAME MODE

Allows you to preview your game without building it to a separate application

You can edit in live-time!

... but changes don't stick

It's strongly encouraged to change your game play color in the Unity settings so you don't make too many changes that don't stay





# GameObjects

---

# Adding Components to a scene

---

The first thing you'll probably want to do when you start your first app is to build a scene!

By default, a camera and simple light source will be included

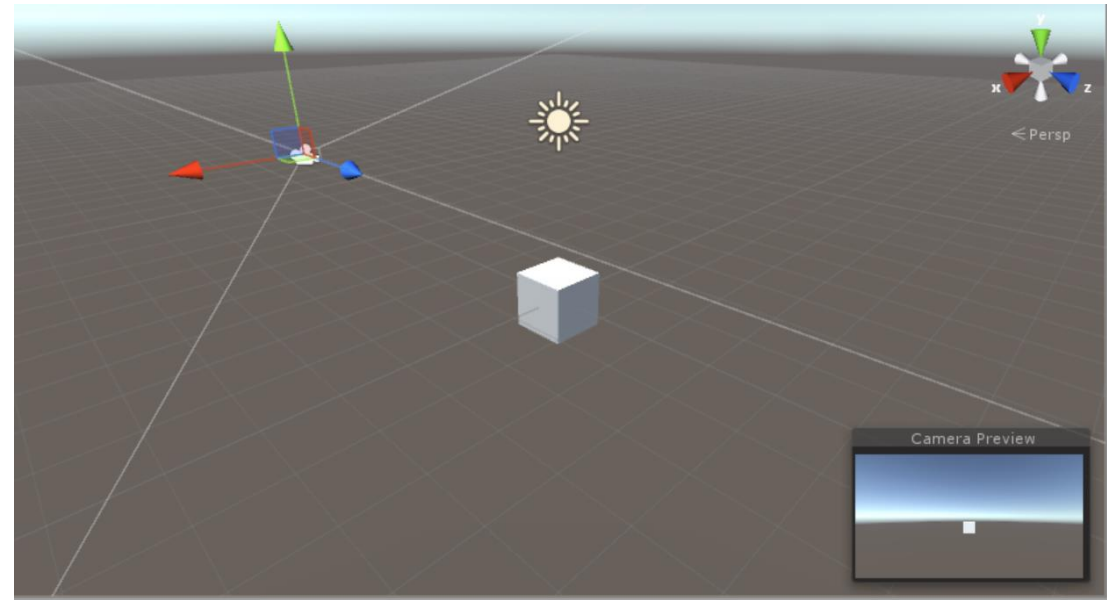
Within the Unity layout, your *hierarchy* will show all of the items in your scene

The **create** drop down menu will let you add new things into your scene

# Create a cube

---

1. On the hierarchy tab, select **Create**
2. Choose **3D Object -> Cube**
3. Click on the Main Camera object to get a small preview of the scene in the lower corner of the game view window. You should see your cube in the distance!



# Changing an Object's Appearance

---

When we have a *mesh* item, such as our cube, we will want to change the way it looks. As an example, we might want our cube to be a wooden box, or a brick wall.

*Textures* and *materials* are two components that we can apply to an object to change the way they look in our game

*Materials* define the properties of an object with how light is reflected off of them

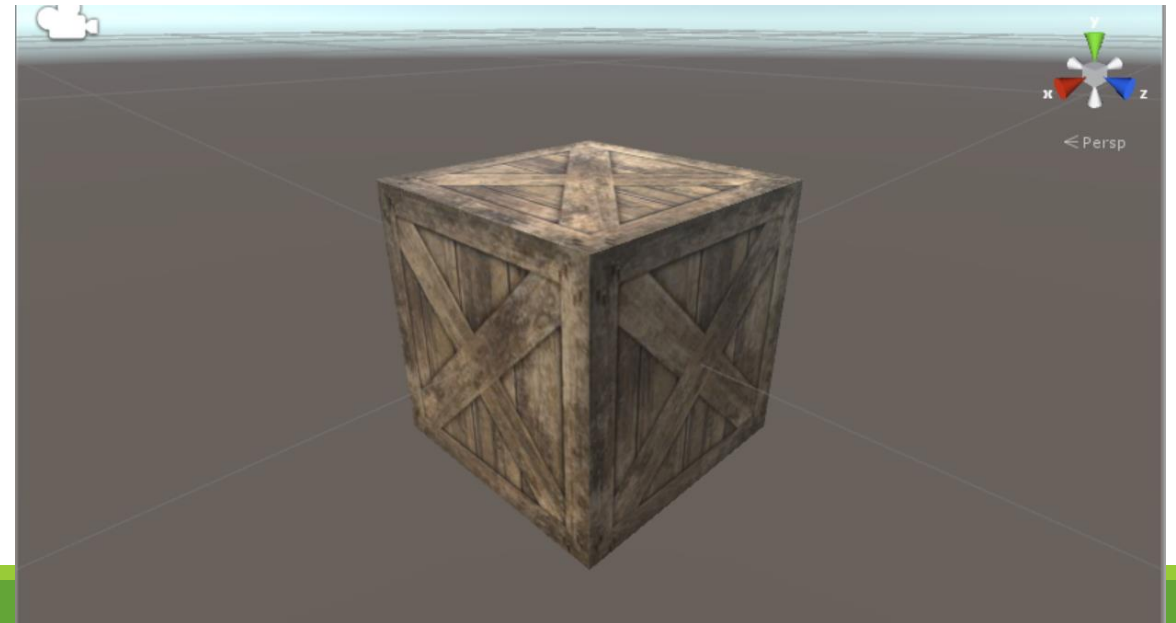
*Textures* are usually image files that create different patterns to make meshes look more complex without adding in new physical elements

# Creating a Texture & Material

---

We're going to make our box more interesting by making a new material and texture

1. We've added an image file for our cube in the sample project
2. In the Assets folder, create a new material for your box
3. Drag the new material onto the cube
4. Drag the image file onto the cube



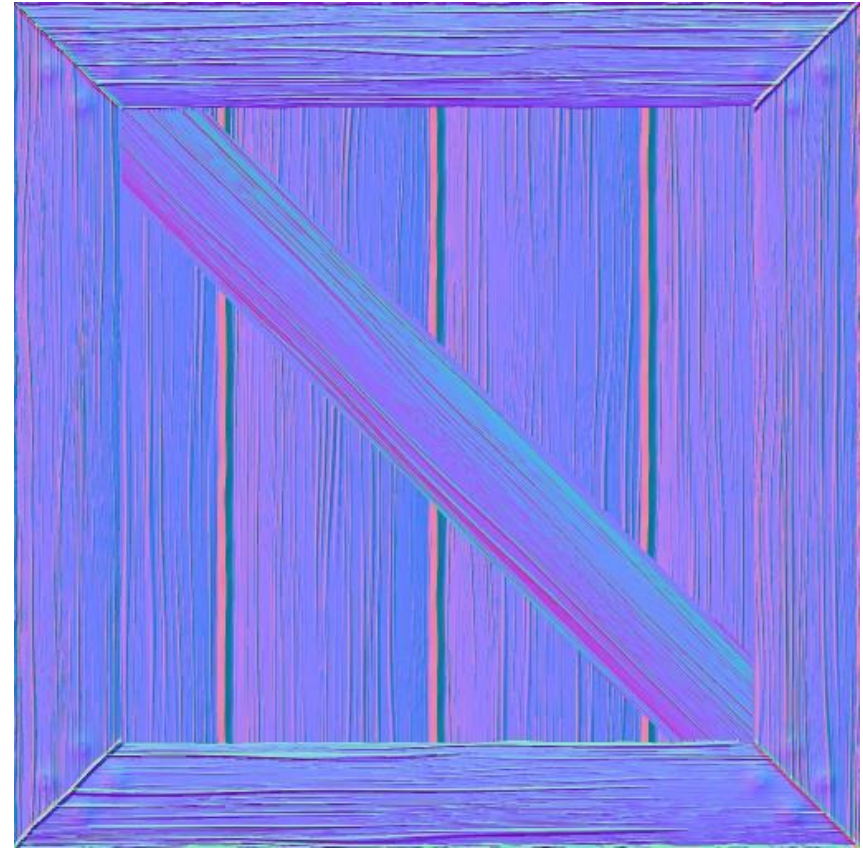
# Give it some depth! Normal Maps

---

A *normal map* is a special type of image file that tells our program how our cube should appear if it were in 3D

Applying a normal map that matches the texture will give it a detailed appearance to seem more realistic

Assets that are downloaded from the asset store and come with a texture may include normal maps



# Adding a normal map to our box

---

1. Select the box and expand the material properties on the inspector
2. Under the material header, find the 'Normal Map' option
3. Drag the desired normal map from the asset folder into the empty spot for the normal map

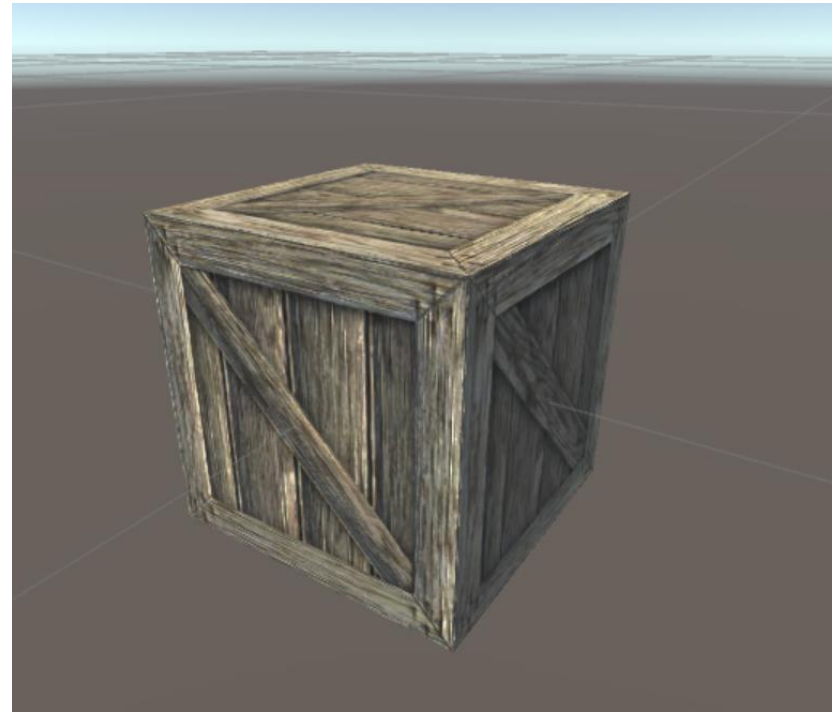
# Crate Normal Map

---

WITHOUT NORMAL MAPPING



WITH NORMAL MAP





# Game Environments

---

# Building an Environment

---

A large part of immersive worlds is the environment that surrounds us when we are playing. There are several approaches for environments in Unity:

1. Open world terrain using the Terrain Editor
2. Small world environments built from simple game objects
3. Enclosed room environments built from simple game objects

You may also find complete scene environments on the Asset store

# Terrains

---



# Unity Terrain Editor

---

Very large worlds that can be modified by painting different textures onto the materials

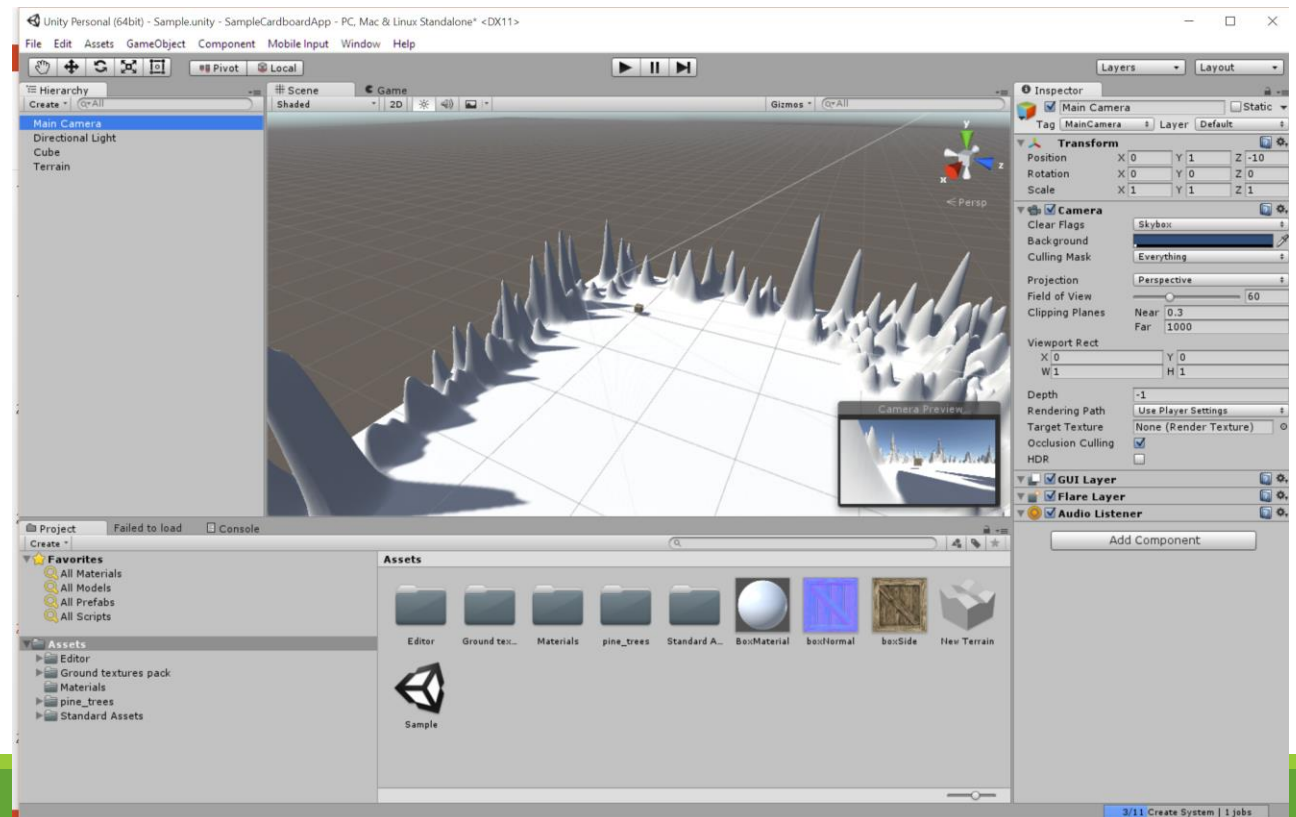
Manipulate the heights of various areas with paint brush

Add trees and other graphics to the terrain via brush strokes

Directly paint texture onto the mesh to create varying looks over the terrain

# Creating a Terrain

1. Under **Create**, choose **3D Object** and pick Terrain
2. Paint height by selecting a brush in the Inspector and clicking + dragging over the terrain mesh in the editor
3. Different brushes = different look!



# Adding a Terrain Texture

---

The Unity Standard Asset package includes some textures to use on terrains. You can also find additional textures in the Asset store.

1. Select the terrain in the editor window
2. Click the small paint brush in the Inspector window
3. Select 'Edit Textures' and pick 'Add Texture'
4. Choose a texture and its normal map
5. Click "Add" when done

The first texture you choose will apply to the entire map. You can then add additional textures and paint specific locations where you want them to be different using the same technique above.



Create ▾ Q All

Terrain



Shaded

2D			
----	--	--	--

Gizmos ▾

☒ Terrain ☒ Static

▼ **Transform**

▼  ☒ Terrain 

## Brushes

## Textures

Edit Textures

### Settings

Opacity  50

1000

Material: None (Physic Material)


Enable Tree Collider ☒

Add Component

Add Component

Failed to load

Create ▾


 All Materials

 All Models


 All Prefabs


 All Scripts

## Assets

▶  Editor

►  Ground textures pack

 Materials

▶  pine\_trees

►  Standard Assets

Downloaded from <http://ajph.org/> on November 10, 2015

Editor

Ground tex

## Materials

nine trees

### Standard A

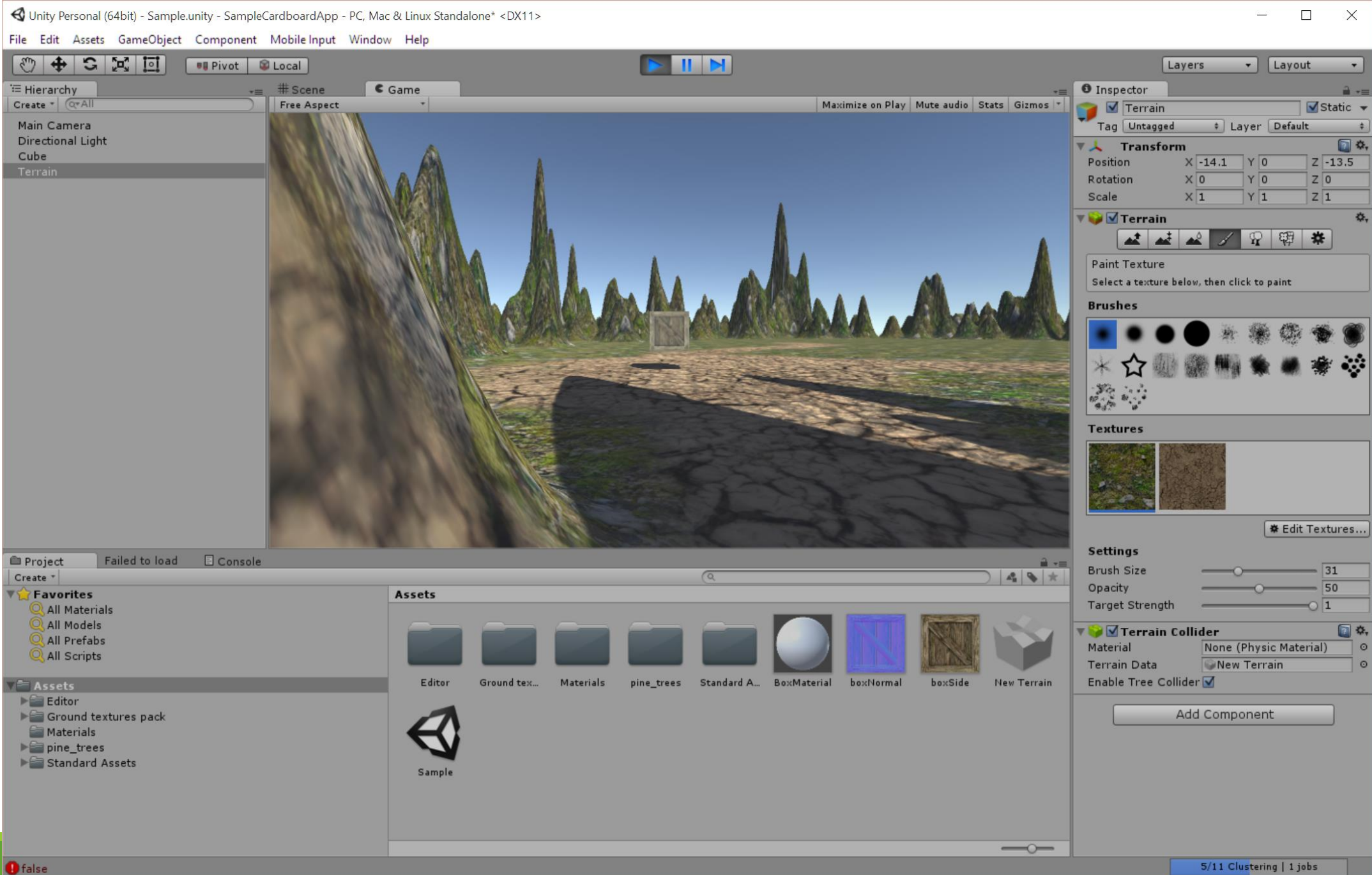
BoxMaterial

### boxNormal

### boxSide

New Terrain

Sample





# Adding Trees

---

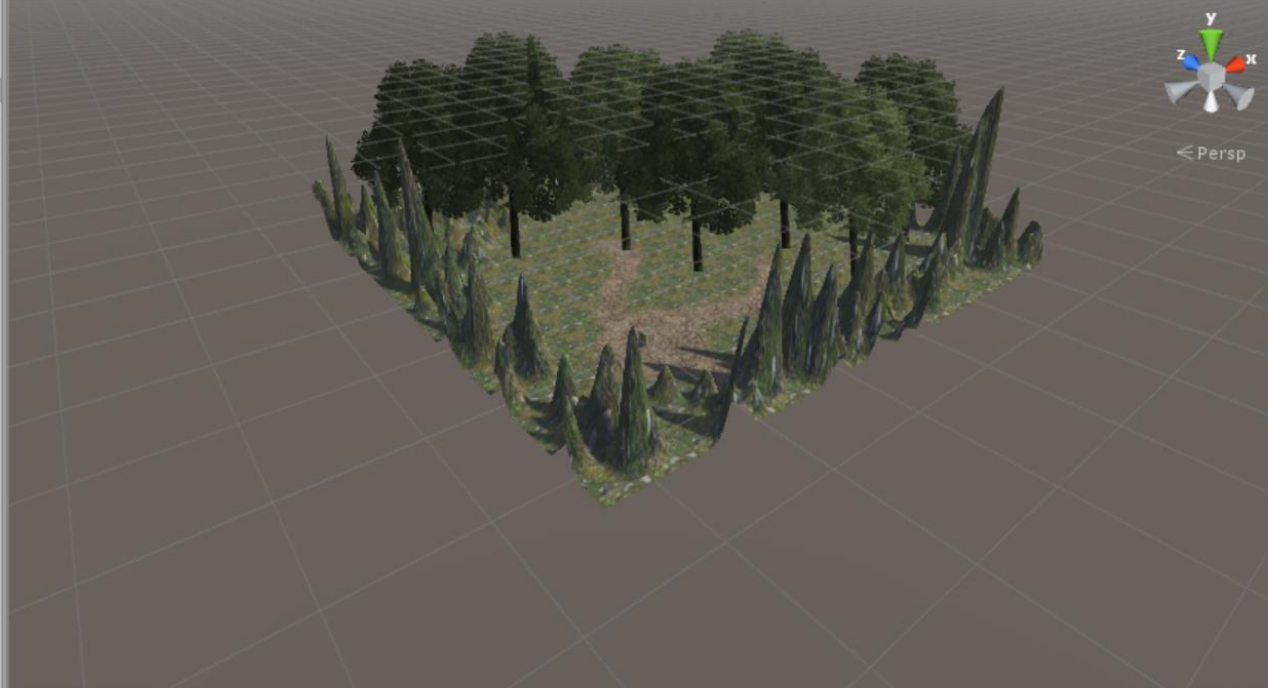
Trees can be actual trees or other objects that you wish to add in large quantities to your terrain.

1. Select the button shaped like trees on the Inspector window
2. Choose 'Edit Trees' then 'Add Tree'
3. Click the small circle by the empty box and choose your desired tree
4. Click 'Add' to save them
5. Paint the trees onto your terrain by dragging the brush, or choose "Mass Place Trees" to fill up trees over your terrain



Game

Gizmos



Create ▾

Assets ▶ Standard Assets

Cameras CrossPlatform...

- ▶ Editor
- ▶ Ground textures pack
- ▶ Materials
- ▶ pine\_trees
- ▶ **Standard Assets**
- ▶ Terrain Assets

5/11 Clustering | 1 jobs

# Adding Grass & Plants

---

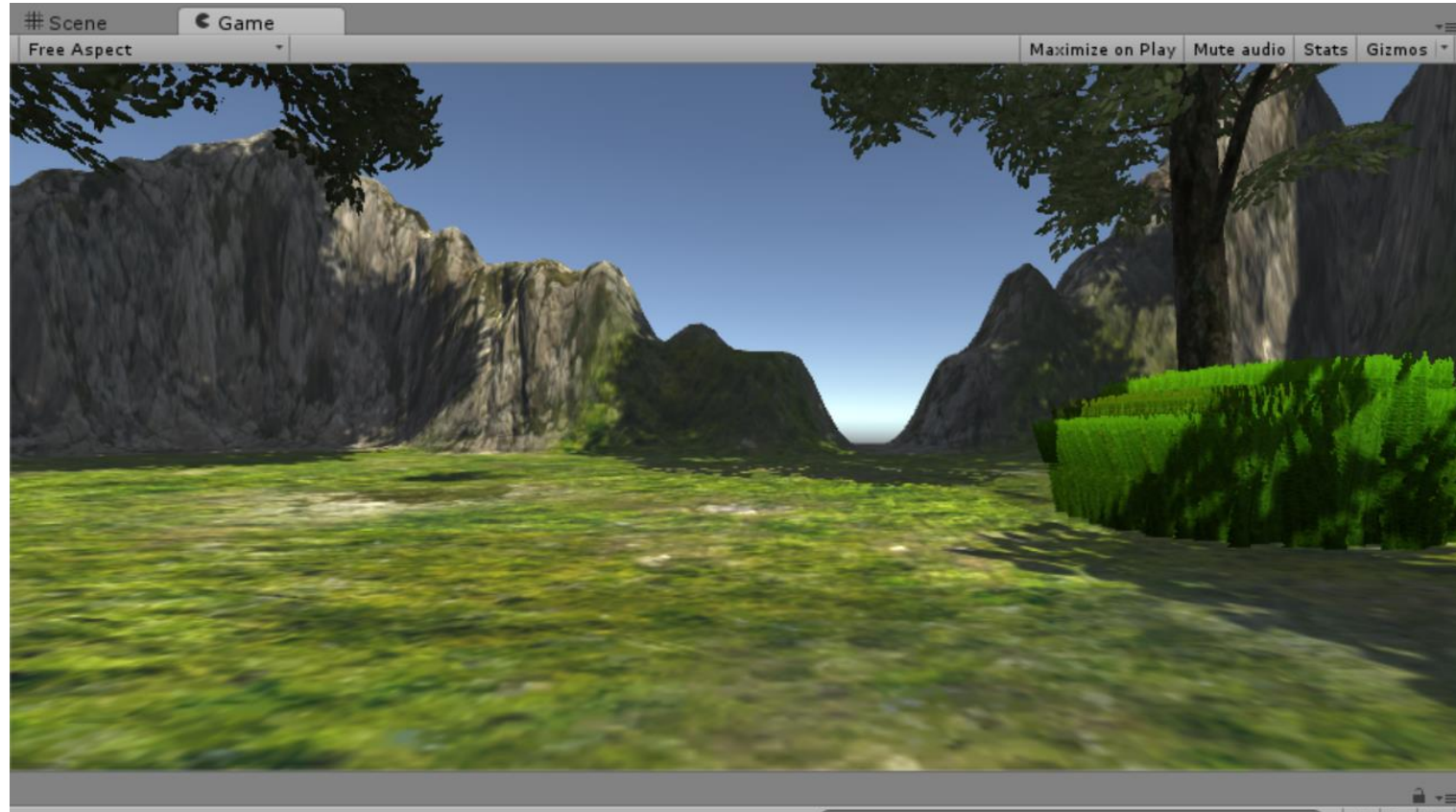
Grass objects can be painted onto the terrain similar to how trees are placed, but with some different properties

1. Select the button in the terrain Inspector panel with flowers on it
2. Click 'Add Detail'
3. Create your detail and grass objects
4. Paint them onto the terrain like the trees

Use sparingly! Grass is rendered individually and takes up a lot of processing power to manipulate and show effects

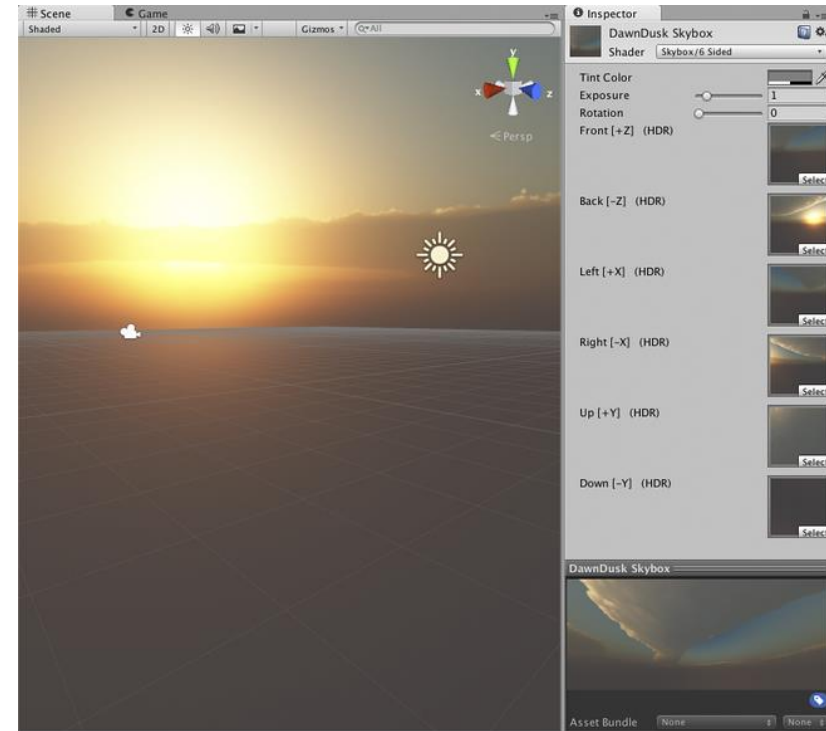
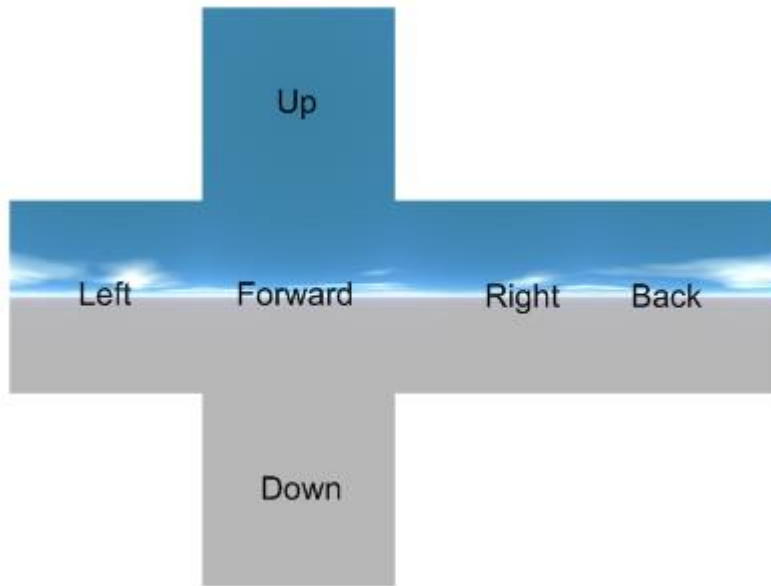
# Fern Rendering Example

---



# Skyboxes

Skyboxes are wrappers that surround your entire scene outside of your terrain with a visual representation of the sky. You can find these on the Unity Asset store or create your own from a series of 6 images



# Interior Environments

---

When building an environment that is set indoors, you will build out the scene with primitive shapes (cubes, etc.) instead of the terrain builder

You can save combinations of materials and shapes in prefabs to prevent duplication of work

# Adding VR

---

THE BASICS OF VR DEVELOPMENT IN UNITY

# VR Development Kits

---

Oculus SDK (Desktop, Mobile): Build for the Oculus Rift. Windows only. Unity / Unreal plugins

SteamVR (Desktop): Build for HTC Vive, but support for Oculus too

Cardboard SDK (Mobile): Build for mobile phones

We will focus on the Cardboard SDK, but development principles for each platform are generally very similar



# Cardboard SDK for Unity

---

Download the Unity plugin from Google [here](#)

Double click the Unity plugin to open Unity and install the assets

You should see a 'Cardboard' folder under the Assets tree



# The Cardboard SDK Components

---

Cardboard Adapter: Prefabs to add to an existing model that you may have already created for your character. Contains left and right cameras to render stereoscopically

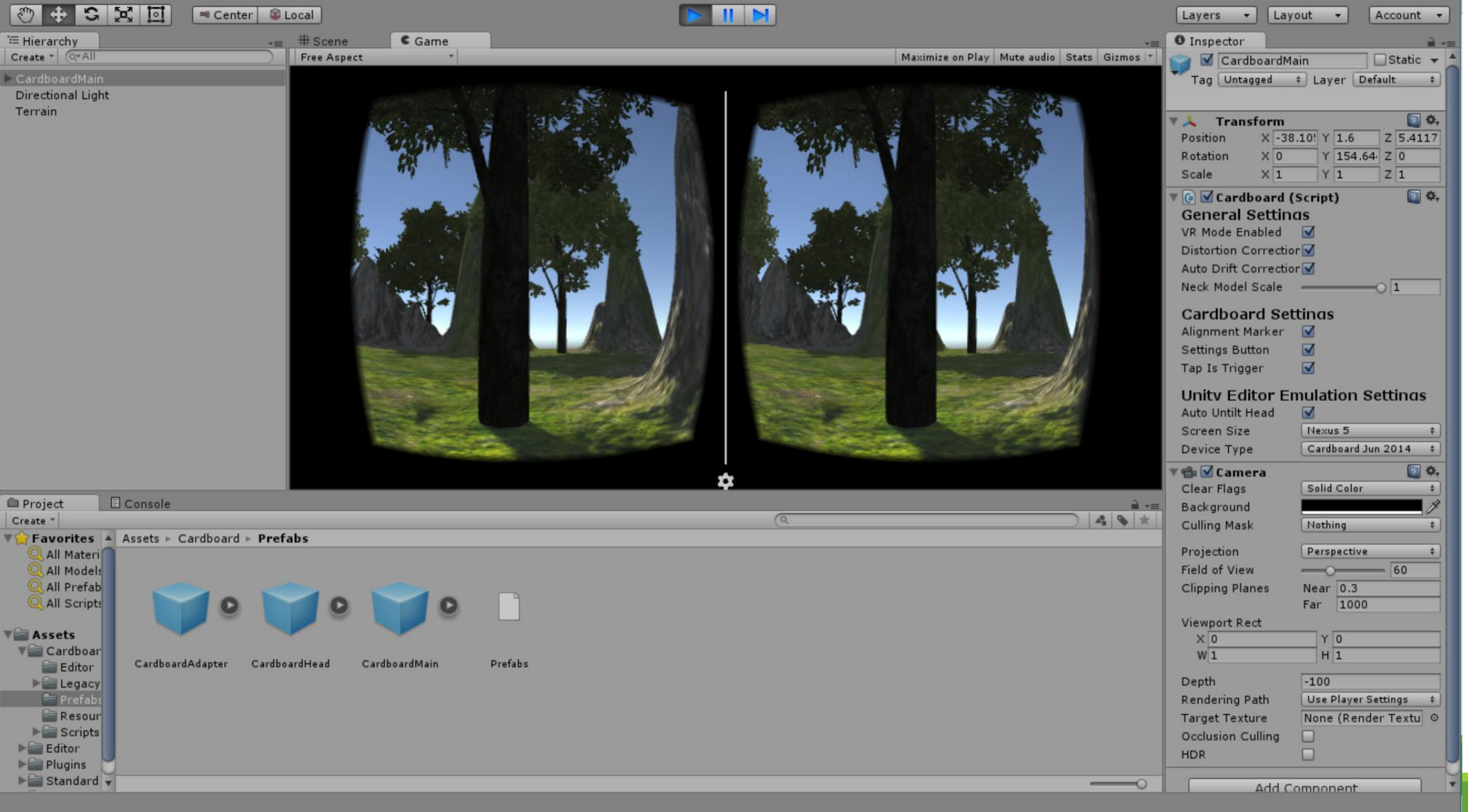
Cardboard Head: The prefab used to contain the adapter with multiple cameras. This separates out the logic of the rendering (done by the cameras) and the look/update behavior (head)

Cardboard Main: The prefab that contains the setup for creating a Cardboard viewer. You can begin with this if you are building for Cardboard from scratch and do not have an existing character with scripts to re-create

# Creating the Cardboard Viewer

---

1. Delete the Main Camera from your scene
2. Drag the Cardboard Main prefab from the Assets folder into the scene
3. Press play to see your stereoscopic rendering!



# Testing in Unity

---

You can mimic some of the behaviors of your phone with the following:

**Look Around:** Alt + Click + Drag

**Tilt Phone:** Ctrl + Click + Drag

# Movement

---

To include movement in your scene (be careful with how this is done in VR!) that can be controlled by the player, you will need to attach a 'Character' element to your camera.

1. Add a First Person Controller to your scene
2. Delete the camera included in the character controller
3. Replace with a Cardboard Main prefab to serve as the player controller

# User Interaction

---

# Refresher: Mobile VR Input

---

Designing VR apps with user input means considering how to approach the variety of devices that players may have

Some users may have peripheral devices that enhance an experience

Decide if your application needs input:

- Meditation apps
- Simple 'movies'
- Storytelling environment

Get creative!

- Experiment into voice controls, 'On Gaze' interactions



# Basic VR Input

---

## CARDBOARD MAGNET TRIGGER

Provides a simple “On Click” mechanism similar to a mouse

Great for trigger-type interactions

Follow gaze

Can broaden for different input behaviors depending on things like where the user is looking

## BLUETOOTH CONTROLLERS

Provide a wider range of different behaviors

Allow for better motion control and character control for movement

Support more complex game mechanics

Downside: May need to customize for different types of controllers

# Cardboard Trigger Interactions

---

Events that happen within your scene require an “Event System” item – this is a component that will handle clicks and other interactions

The objects that cause the various events will need an “Event Trigger” component

This approach can be used for UI elements, such as button clicks, or for objects within the scene

# Add an Event System for Cardboard

---

1. Under your scene hierarchy, click 'Create' -> 'UI' -> 'Event System'
2. Select 'Add Component' -> 'Gaze Input Module'
3. On your Main Camera (under Cardboard Main object), select 'Add Component' -> Physics Raycaster
4. For game objects that you want to add a behavior to, add an 'Event Trigger' by going into the Inspector for the item
5. Choose 'Add Component' -> 'Event Trigger'

# Vanishing Act!

---

We are going to create a very short script that we can use with our Event System. This script will cause the object “clicked” on with the Cardboard trigger to disappear.

1. Under Assets, right click and choose ‘Create’ -> C# Script
2. You can use any editor you’d like with Unity – by default, Unity comes with an editor called ‘Mono Develop’, but I like to use Visual Studio since it has a few more debugging capabilities

# Script Basics

---

Each Unity script will have two methods by default:

- Start() -> Called the first time the scene is loaded
- Update() -> Called once every frame

Scripts can have public variables, which can be modified by other game objects, or be self-contained

Add scripts to objects through the Inspector panel

There are additional methods that Unity will recognize (example: `OnClick()` )

# Our Script

---

```
// Delete is called on trigger pull  
public void DeleteOnTrigger()  
{  
    GameObject.Destroy(gameObject);  
}
```

# Set an action for Event Triggers

---

1. On your Event Trigger for your specific object, click 'Add New Event Type'
2. Choose the desired event type (such as "Pointer Click" for a click / trigger pull)
3. Under 'List is Empty', click the plus sign
4. Click the circle next to the empty box and select the object (make sure you attached your script with your function!)
5. On the drop down, find the desired script and the function you want to call

# Adding a Gaze Indicator

---

You can add an object to be used by the Event System as a 'Gaze Tracker' that shows where the current pointer location is.

1. Under your 'Cardboard Head' object in the hierarchy, add a capsule or sphere
2. Set the object properties to have a Z-index value that is ~5 units away from the head and .1 scaled on X, Y, and Z coordinates



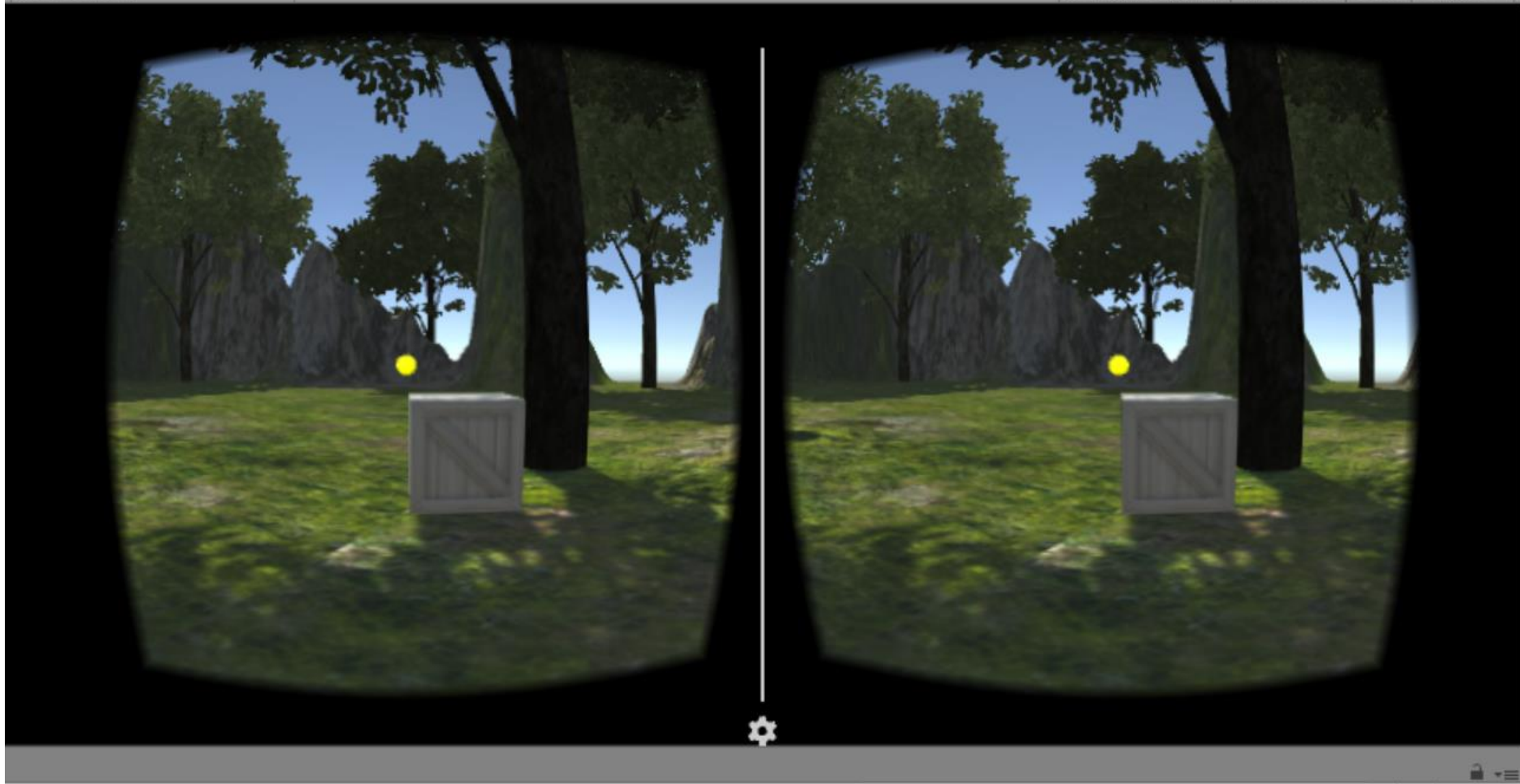
Free Aspect

Maximize on Play

Mute audio

Stats

Gizmos



# Final Notes

---

# Gameplay Considerations

---

**Character Movement:** How do you want your user to move throughout your scene? Do you even want to allow movement in your game? Keep in mind that with VR, not all headsets will have an easy way to support moving around

**Challenges:** Are you building a game with challenges for a player to figure out? Or is your goal more experiential in nature?

**Environment Design:** Do you want your game to feel as though it takes place in a realistic environment, or do you want to create a surreal environment that feels totally different than the real world?