

CS3450 Game Development Lab 1

- [Introduction](#)
 - [Assets](#)
 - [Development in Unity](#)
- [The Unity Interface](#)
 - [Some advice:](#)
 - [The Toolbar](#)
 - [Hierarchy](#)
 - [Scene View](#)
 - [Inspector](#)
 - [Project Window](#)
 - [Game View](#)
 - [Cheat Sheet](#)
 - [GameObjects, Components & Prefabs](#)
 - [GameObjects](#)
 - [Components](#)
 - [Prefabs](#)
 - [First Project](#)
 - [Developing Terrain](#)
 - [Topology](#)
 - [Texturing Terrain](#)
 - [Detailing Terrain](#)
 - [Using the Asset Store](#)
 - [Using the First Person Controller Prefab](#)
 - [Lighting the Scene](#)
 - [Adding a Skybox](#)
 - [Models, Materials, Shaders, and Textures](#)
 - [Collision Detection](#)
 - [Publishing your Game \(Build Options\)](#)

Introduction

Unity is a professional-quality game engine that is used by thousands of seasoned game developers. It is one of the most accessible modern tools for novice game developers and makes it easy to start learning game development skills.

The Unity engine comes with many features such as physics simulation, normal maps, screen space ambient occlusion (SSAO), dynamic shadows. Any game implemented using an engine gets all the features the engine provides.

Unity's key advantages are a productive visual workflow and a high degree of cross-platform support.

Thousands of games are made with Unity [Made With Unity](#).

Here some of the key selling point of Unity:



Workflow

Rapidly assemble your scenes in an intuitive, extensible Editor workspace. Play, test and edit for fast iteration towards your finished game.



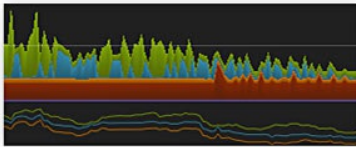
Quality

Create a game with AAA visual fidelity, audio and full-throttle action that performs smooth and clean on any screen.



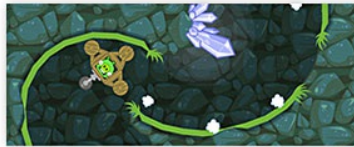
Mecanim

Unity's uniquely powerful and flexible animation system brings any character or object to life with incredibly natural and fluid movement.



Performance

Reliable performance, smooth framerate, and superb game play experiences across target platforms.



Multiplatform

No other game engine gives you the choice of 10 publishing platforms for your game with near-effortless deployment.



Collaboration

Full version control for all game assets; instantly grab changes from other team members, and extend Unity for generic VCS support

Assets

A lot of work within Unity is to do with your Assets, i.e. your Models, Textures, Audio etc. The [Unity Asset Store](#) is the central place to acquire both, free and paid for Assets/Extensions

Development in Unity

The main language to program Unity is C#. Programming is not done within Unity itself, rather code exists as separate files that are available inside Unity. While script files can be created within Unity, a text editor or IDE is used to write the code within those initially empty files. Unity comes bundled with Microsoft Visual Studio and also supports MonoDevelop, an open source, cross-platform IDE for C#.

The Unity Interface

This tutorial should take roughly 3 hours.

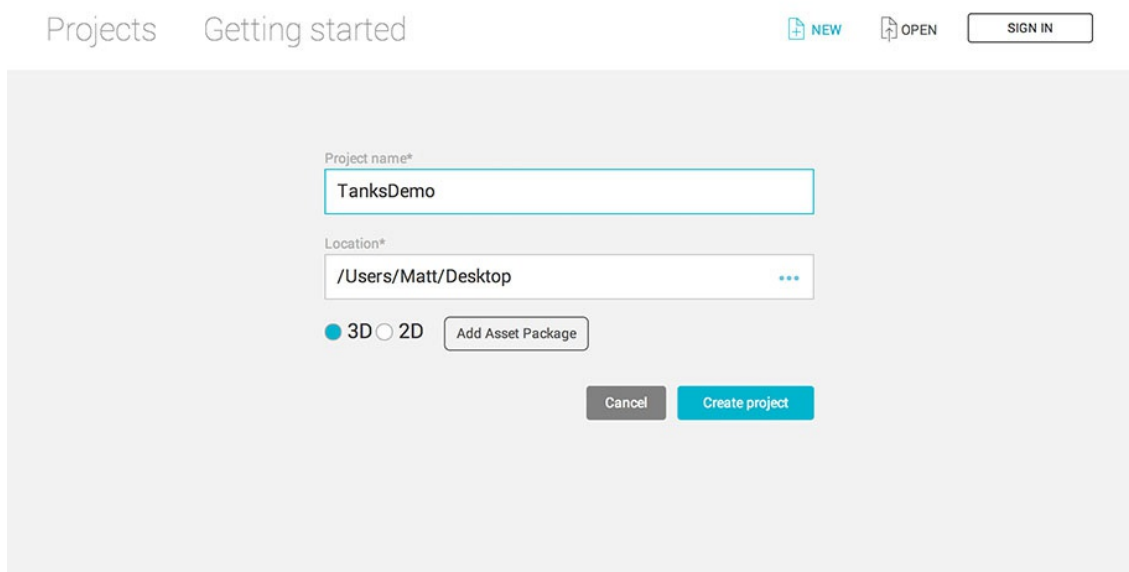
Some advice:

- **Remember to save your work regularly**
- **The use of a free online version control repository such as [BitBucket](#) is highly recommended**
- **Make sure you have plenty of disk space; Unity crashes with no helpful errors and won't let you import anything into the project with not enough disk space.**

To tour you round Unity we are going to explore "[Tanks! Tutorial](#)", one of the many Sample Projects that Unity have uploaded to the Asset Store.

To save time, you can download the unity package `tanks.unitypackage` from Blackboard.

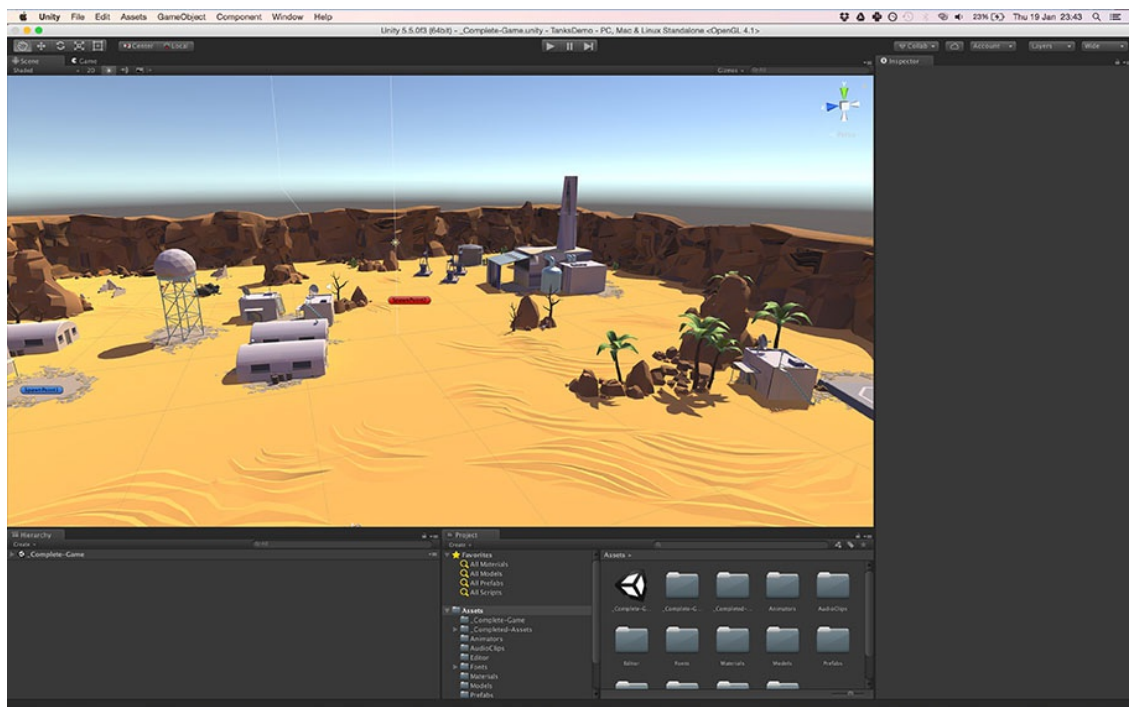
Open Unity, sign up, and create a new Project called 'TanksDemo'.



To download the package you have downloaded, navigate to Assets -> Import Package -> Custom Package..., and locate the package in your file system. Make sure to import the entire contents of the package. The import may take a few minutes.

Within the Project view, double-click on Assets -> _Complete-Game to load a new scene, containing the entire game.

You can always view which project, scene, and build mode you currently have by reading the bar at the top of the screen. In this case the scene is '_CompleteScene.unity', the project 'Tanks Demo' and the Build option is 'PC, Mac & Linux Standalone'. More on build options later.



Unity's interface consists of movable/dockable Views, or Windows, which can be rearranged to suit your particular workflow. You can have as many or as little views as you like. Unity provides several built in layouts, the above being the default 'wide' layout. To switch between layouts, as well as save your own custom layout, you can use the 'Layout' button within the 'Toolbar' - more later.

Lets now take a closer look around the views and their features. I strongly encourage you to keep switching between the tutorial and Unity while reading and explore each window's features to help you become familiar with them quicker. Don't worry about noting down shortcuts either, there's a [Cheat Sheet here](#) ready and waiting to become your new bedtime reading.

The Toolbar

Not quite a View as such, but a very important part of Unity's interface nonetheless. The 'Toolbar' spreads across the top of the Editor and features several key tools for manipulating the 'Scene' and 'Game' windows, as well as the layout of the interface and connections to Unity Services. It also holds the very important 'Play', 'Pause' and 'Step' buttons.



The first component within the Toolbar, located in the top left corner, holds the four 'Transform Tools'.

From left to right they are:

The Hand Tool (Keyboard Shortcut: Q)

By default the Hand Tool will allow you to Click and Drag to pan around the scene. Holding Shift will increase the panning speed; using combinations of Alt, Ctrl, LMB, Middle Mouse Button and RMB will allow you to pan, rotate and zoom around your scene. We will explain these in a little more detail shortly.

The Translate Tool (Keyboard Shortcut: W)

The Translate Tool provides the ability to select GameObjects — more on these later — within your scene (or hierarchy) and then use the arrow-ended axis handles to move that object around the scene by clicking and dragging each axis handle. You can also move the selected GameObjects freely over all three axes by clicking and dragging the small square where the axis handles meet.

The Rotate Tool (Keyboard Shortcut: E)

The Rotate Tool provides axis handles that loop around and surround the selected GameObject which when clicked and dragged allow you to rotate it around either its centre or pivot point.

The Scale Tool (Keyboard Shortcut: R)

Very similar to the Translate Tool, the Scale Tool provides the block ended axis handles which when clicked and dragged allow you to scale the selected GameObject on that particular axis. To scale all 3 axes simultaneously, simply click and drag the grey cube where all 3 axis handles meet.

The Rect Tool (Keyboard Shortcut: T)

This is the tool that is used for manipulating UI elements when designing the UI for your game. More on UI in later labs.

The Transform Tool (Keyboard Shortcut: Y)

This tool combines the translate, rotate, and scale tools.

The Toolbar – Transform Gizmo

The next component within the Toolbar, shown below, is the 'Transform Gizmo', which determines whether the selected object is rotated or scaled relative to its centre point or to its pivot point, and whether local or global rotation is used when scaling and rotating the selected GameObject.



The Toolbar – Control Buttons

The component within the centre of the Toolbar controls the preview of your game within the Game window. Pressing the 'Play' button will play your game. Clicking Play again will stop the game running. Clicking the 'Pause' button will, unsurprisingly, pause your game. You can click 'Pause' before clicking 'Play' to pause the game on the first frame. The 'Step' button along allows you to step through the game frame by frame when it is paused. The buttons will turn *blue* when the game is playing or paused.

Go ahead and hit Play if you already haven't. You control two tanks that can shoot at each other. One is controlled with WASD and the other with the arrow keys. Space and Enter are shoot.



The Toolbar – The Rest

Collab is out of the scope of this course. It is used for teams working together remotely. You can read more [here](#).

The cloud icon is a link to Unity Services. These are features such as Ads and Analytics. Again, not within this course's remit.

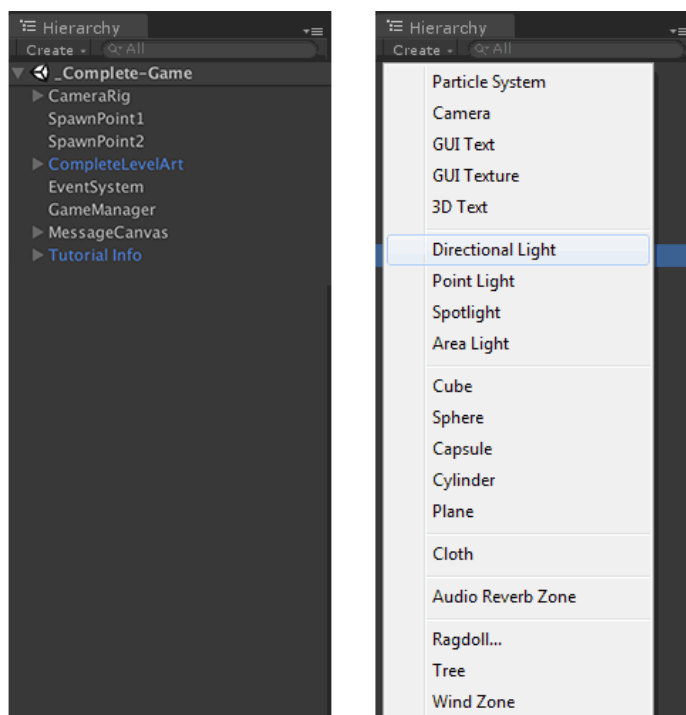
'Account' gives you access to your Unity account, profile, etc.

The 'Layers' drop down menu, located to the right hand side of the screen within the Toolbar, allows you to toggle the visibility of particular layers on and off. The 'Layout' drop down menu provides options regarding the Layout of the windows or views on screen. If you create a custom layout and are particularly fond of it, you can use this menu to save your layout, so you don't have to recreate it every time you restart Unity.



Hierarchy

The 'Hierarchy' window provides a list of all the GameObjects within the Scene view, discussed next. When a GameObject is selected within the Hierarchy, it is also selected within Scene view and vice versa. You can add new GameObjects to the Hierarchy via the 'Create' menu shown below, via the GameObjects menu at the top of the screen, or by dragging from the Project View straight into the Hierarchy or Scene View.



Within the Hierarchy at the very top you will see the name of the scene. Unity supports

loading and editing of multiple scenes at the same time. You can right-click on the scene name to save, unload, or remove that scene. To load an additional scene, you select it from the Project View and drag it into the Hierarchy View.

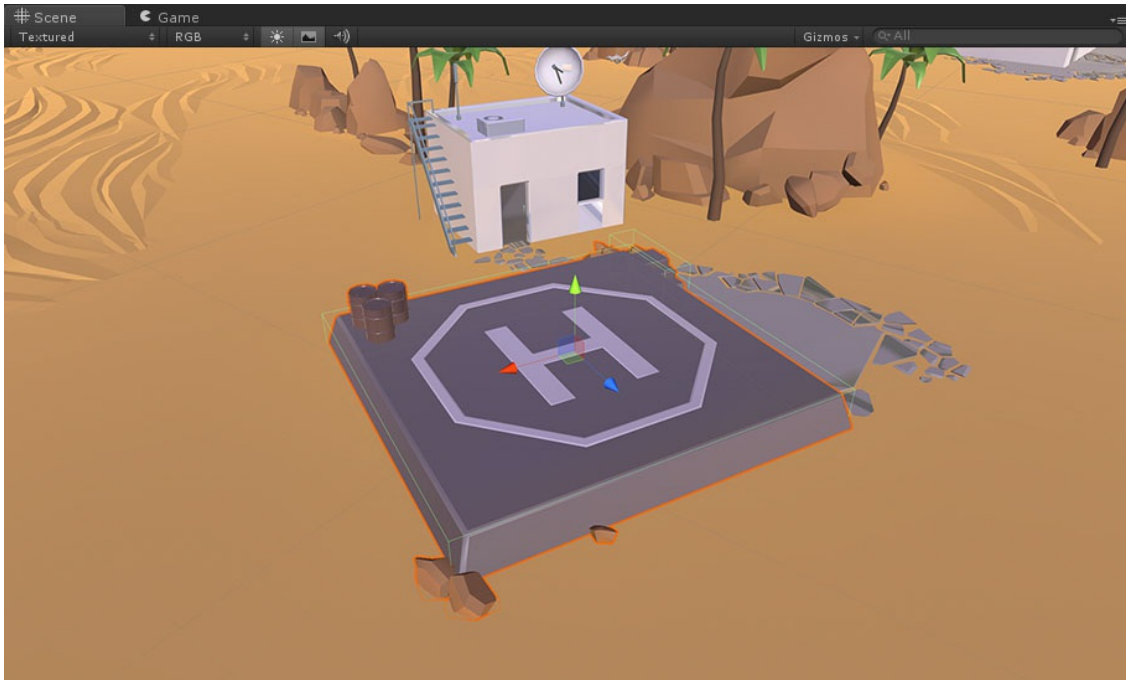
Objects that have children, like the scene, have a grey triangle to the left of their name, which toggles the display of their children. You can rename GameObjects by selecting them in the Hierarchy and pressing F2 or using the Inspector discussed below.

You can also search the Hierarchy (and therefore also the scene) by using the search field at the top of the View - more on this shortly.

GameObjects shown in blue text within the Hierarchy are Prefabs which will be explained shortly.

Scene View

The 'Scene' view is where you will spend a lot of your time building and editing the Assets that make up your game within Unity. A fully rendered 3D 'preview' of your game is displayed inside the window where you can add, edit, and remove GameObjects. All objects within this window are considered GameObjects and are listed within the 'Hierarchy', which we just discussed.



Navigating the Scene View

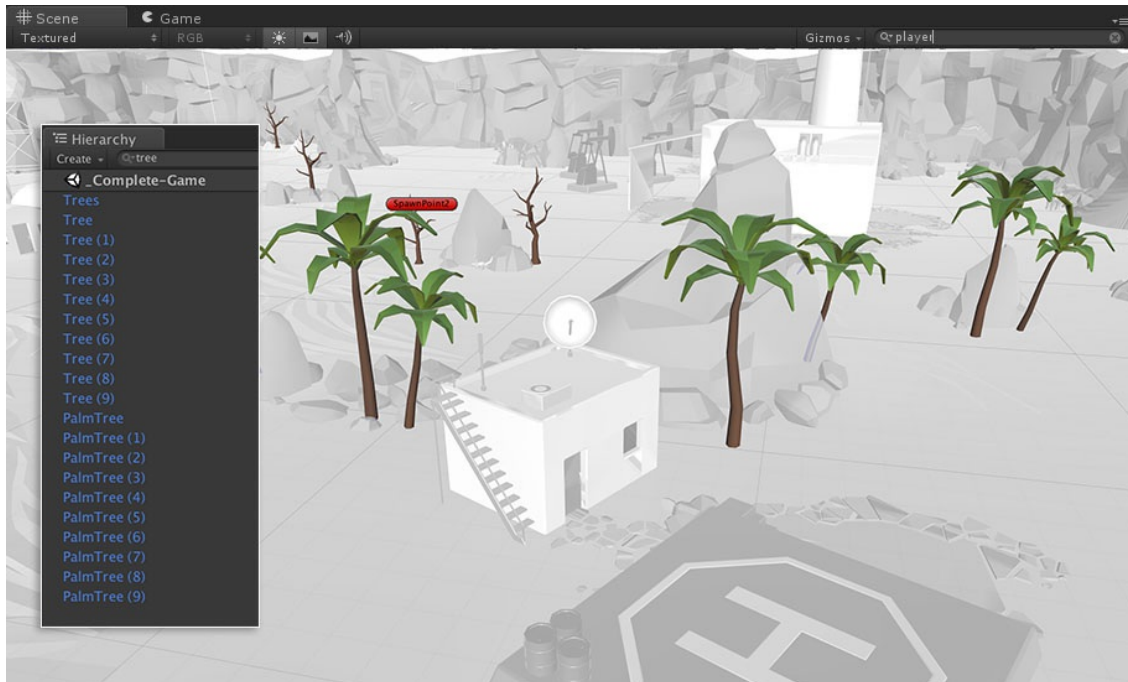
There are several methods for navigating the Scene view, some more suited to particular tasks than others. Try to explore all of them, so you can navigate quickly and efficiently around your scene.

One of the most useful features is the ability to focus on any GameObject simply by double clicking on it in the 'Hierarchy'. Try this to focus on the 'Helipad', similar to above (found within CompleteLevelArt).

You can also use the Search Bar to search for specific GameObjects, which can be useful and far quicker when you have a full scene. Simply type the name of the GameObject into the Scene or Hierarchy Search Bar and all of your scene will be greyed out except for the GameObject(s) with the searched name as shown below. The Hierarchy will also return all those GameObjects containing the search string. Search for 'tree' within both Scene and Hierarchy views and you should get something similar to the below.

As well as searching the Scene/Hierarchy by name you can also search by Type by clicking the magnifying glass and changing the search criteria. Try searching for AudioSource for example and you will be shown a list of all the GameObjects that have an AudioSource

Component attached. An AudioSource is a Component which allows the object that it is attached to produce sound.



You can fly through the scene by holding down the right mouse button in the Scene view, using the WASD keys to move. If you wish to fly-through faster hold the SHIFT key. To raise and lower the camera use the Q and E Keys.

There are quite a few keyboard & mouse commands for navigating the Scene view within Unity. The below are for a 3 button mouse and can be used when any 'Transform' tool is selected. These are crucial to a fast work-flow.

To **rotate** around current pivot point within the scene: **ALT + Left Mouse Button**

To **rotate** on the spot within the scene: **CTRL + Right Mouse Button**

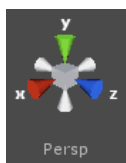
To **pan** around the scene: **CTRL + ALT + Left Mouse Button / ALT + Middle Mouse Button**

To **zoom** in or out of the scene: **ALT + Right Mouse Button / Mouse Wheel**

With all the above, hold **Shift** to speed them up.

Unity's Scene view also features a 'Gizmo' in the top right hand corner which shows the current viewing angle and allows quick navigation to **Top** (Y) / **Side** (X) / **Front** (Z) and **Perspective/Orthographic** views.

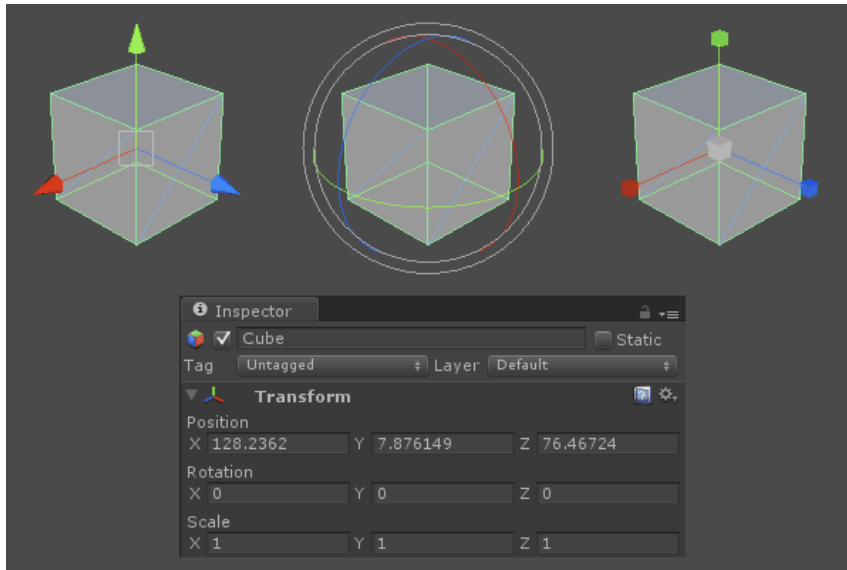
Clicking the centre of the Gizmo will toggle orthographic and perspective views. The [orthographic view](#) can be used for [2D games](#) or for views where depth of field (DOF) is not required. The perspective view is what you are likely used to seeing within a 3D game.



Spend a few minutes navigating around the scene using these methods, the quicker your fingers learn them the better.

Translating, Rotating, and Scaling Game Objects within Scene View

When building your games you will likely place many different objects within your scene. To do this, it is easiest to use the 'Transform' tools in the 'Toolbar' to Translate, Rotate, and Scale individual GameObjects using the appearing axis handles.



You can also type values directly into the Transform Component shown in the Inspector, as shown above.

Search the scene for the 'Radar' and move it to a different position, anywhere. Rotate it by 90 degrees on the Y axis and then Scale it to twice the size. You can do all this by using the axis handles or explicitly adjusting the values of the Transform Component within the Inspector.

Scene View Control Bar

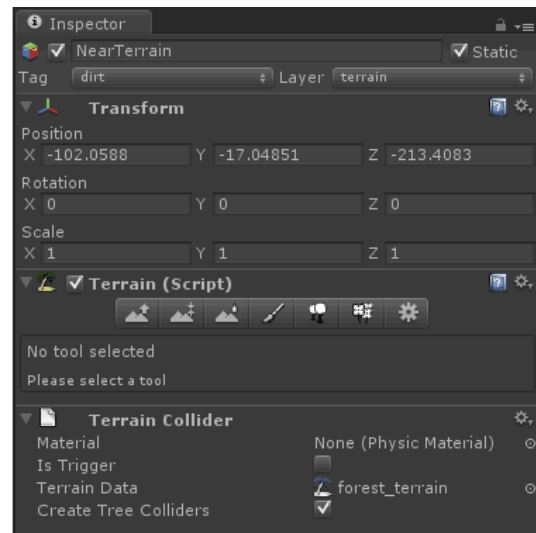
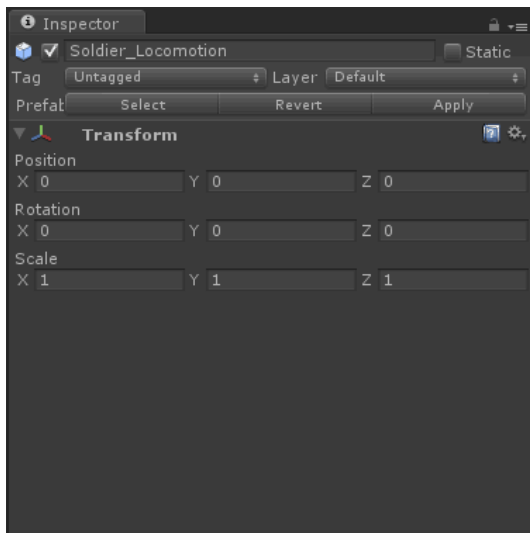
The control bar along the top of the Scene View provides various view modes such as textured, wireframe and textured wireframe. It also provides toggles for 'in-game' Lighting, Audio and Elements such as Skyboxes within the Scene View. The above image of the scene and hierarchy for example has Lighting and Elements toggled on and Audio toggled off. Click around to understand what each does.

It also provides access to a list of Gizmos. Gizmos are visual representations of entities in the scene view which do not have a visual appearance in-game, such as light or audio sources. Using this menu you can alter the size of the Gizmo icons and toggle each Gizmo on and off.

Inspector

The 'Inspector' displays information about the selected GameObject, including all of its attached components and their adjustable properties. Properties can be assigned or modified by dragging and dropping GameObjects or Prefabs from the Hierarchy or Project views or by simply typing values, checking boxes, etc. Numerical values can be adjusted by hovering your mouse over the name of the property and dragging sideways. Try this now for the Radar.

Right clicking on the name of a Component (e.g. Transform) provides several convenience functions such as Copy Component, Paste Component values, and Reset which can be used to quickly copy component settings between GameObjects.



The very top of the Inspector displays the GameObject's name and, to the left, whether it is enabled. On the far right you can define whether the GameObject is static, for lighting and navigational purposes. Beneath the GameObject's name is its tag, if you have assigned one, and the layer it is part of.

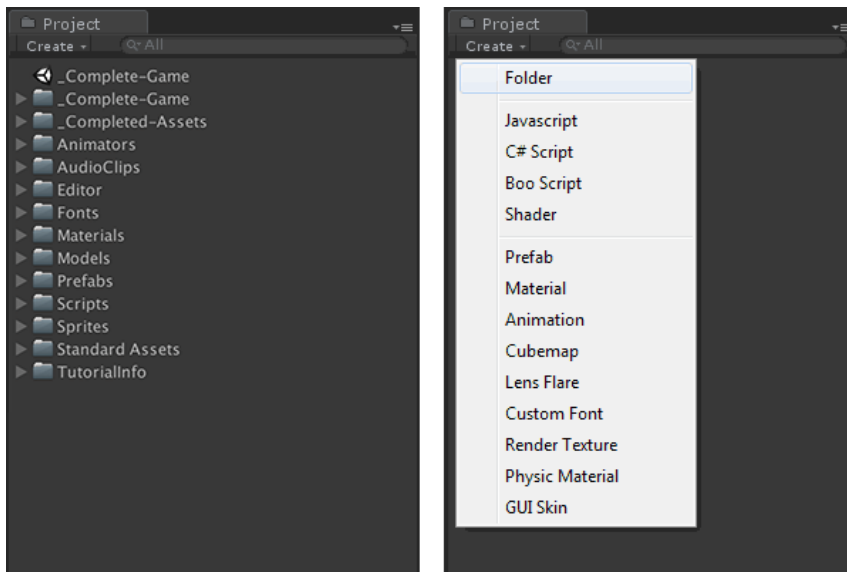
A GameObject's Components are listed below and are collapsible via the triangle left of their name. The check-box directly left of the name defines whether the Component is enabled, which comes in very useful when debugging. To the right of the name is a link to the Components Reference as well as a menu for resetting or removing the Component. You can also right-click a Component's name to remove it.

At the very bottom of the Inspector is the Add Component button. This will list *all* available Components, even user-defined ones.

Project Window

Every Unity project has an 'Assets' directory for all the models, textures, scripts, prefabs etc. used by or made available to the project. You can navigate and explore this directory within Unity's Project window, as well as create new sub-directories and Assets (e.g. scripts) within it, by right-clicking or selecting 'Create' as shown below. Assets can also be searched for by using the Search field located in the top-right corner of the window.

NB: The default layout for the Project view in Unity is two columns. You may prefer to use the one-column layout. You can change the layout of the Project View by left-clicking the top right corner of the view and selecting One/Two Column Layout.



To add assets such as models and textures to your project, from the top menu select *Assets -> Import New Asset*. Alternatively you can simply drag them from your operating system's file system to the Project View. When assets are altered — for instance when you edit a texture in Photoshop — they are automatically updated within Unity.

You should *never* move project assets around using your operating system's file system, since this will break any meta-data associated with the asset, e.g. textures and materials applied to a model, or scripts applied to a GameObject. You should always use the Project View to organize your assets.

If you wish to locate a Unity asset within your file system you can right-click the asset and select 'Open Containing Folder'.

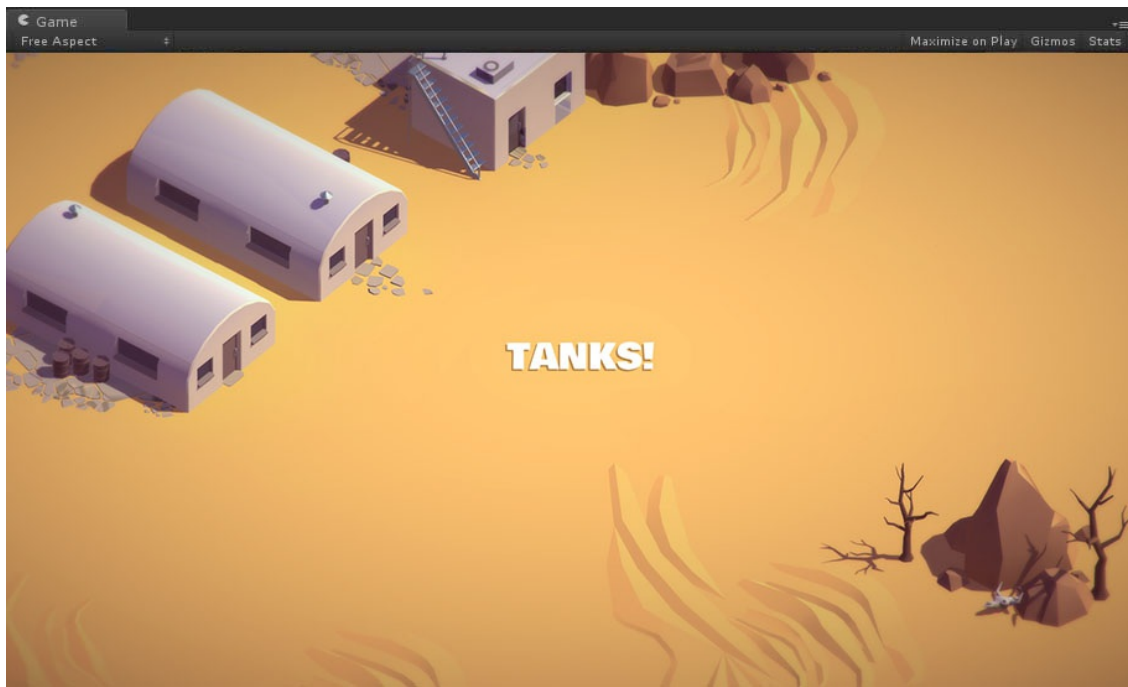
Game View

The 'Game' view is where you play and test your game as if it is published. When you run your game, using the Play button within the Toolbar discussed earlier, you will notice that changes within the Game View are reflected within both Scene View, Hierarchy and the Inspector and that it's still all editable. This gives you the advantage of being able to edit and tweak your game to get the best result while its running. However, as soon as you stop running the game all changes will be lost.

ALL changes made while the game is RUNNING (blue play button) WILL BE LOST once the game stops.

Try it: Run the game and then delete a building or something from the scene. Stop the game running and voila, it's back.

FYI: You can adjust the colour of the editor while in play mode which helps.... Top Menu -> Edit -> Preferences -> Colors -> Playmode Tint.



Game View Control Bar

The control bar at the top of the Game view allows you to force an aspect ratio on your game within the Game view, toggle Gizmos, toggle Stats and toggle whether the Game view goes fullscreen when you click Play.

Cheat Sheet

In order to work as efficiently as possible within Unity you seriously need to commit all the keyboard shortcuts to memory. To help you learn them quicker, there is a Unity Cheat Sheet that includes all the shortcuts detailed above, all on one page - [Unity Cheatsheet Download](#).

GameObjects, Components & Prefabs

You may have noticed these three terms in the above sections. GameObjects, Components & Prefabs are key concepts to building games within Unity and it's essential you understand what they are and how you can use them. Embedded in the explanations below are links to the corresponding Unity Documentation. Have a quick read in the documentation so that you are familiar with the language for how GameObjects, Components and Prefabs are all related when scripting.

Your Turn:

Create a new scene, File -> New Scene.

GameObjects

Every object within your game is a technically a [GameObject](#). A GameObject is essentially a container that has Components added to determine its behaviour and appearance, i.e. AudioSource, BoxCollider, MeshRenderer to name a few Components.

Every GameObject must have a [Transform Component](#) which determines its location, scale and rotation within the game world.

Your Turn:

Add a Cube GameObject using the menu, GameObject -> 3D Object -> Cube. Double click on it in the Hierarchy to focus on the cube.

Take a look in the Inspector and you will see it has the aforementioned Transform Component as well as a couple of other Components...

Components

[Components](#) determine the behaviour and appearance of GameObjects they are added to and are the real workings of any game within Unity.

Unity comes complete with a handful of built in Components such as Transform, BoxCollider, AudioSource, Rigidbody to name just a few - literally all built-in behaviour is controlled by Components. Most components can be switched on and off (enabled/disabled) as well as added/destroyed at runtime.

In our case with our Cube we have a MeshFilter which is the geometry data for the cube, a MeshRenderer which renders the MeshFilter geometry to the screen and finally we have a BoxCollider which allows collisions to be detected with our cube.

Scripts that you develop can be added to GameObjects in exactly the same way as the built in Components because that's essentially what they are, Components. They are Components that **you** developed which all extend the base MonoBehaviour class.

Your Turn:

1. Try adding a Particle System Component to the Cube using `Component -> Effects -> Particle System`. You should now see some white particles flying out of your cube while the cube is selected.

If the particles appear pink it's because the Material is missing from the Particle Renderer. At the bottom of the Particle System Component, expand the Renderer section, click the icon to the right of 'Material' and select 'Default-Particle' from the list that appears.

2. Now remove the Particles Component from the Cube by RMB (right mouse button) on the Particles Component title in the Inspector or on the little Gears/Settings Icon to the right of the title and selecting 'Remove Component'. The particles should now stop.

Prefabs

[Prefabs](#) are 'prefabricated' or re-usable GameObjects stored within the Project window alongside all your other Assets.

Within the Hierarchy Prefabs are blue. If they lose the connection to their original (i.e. it's deleted) then they will appear a reddish colour.

Your Turn:

1. Create a new folder in your Project View, using the 'Create' menu at the top of the view, called 'LabPrefabs'.

2. The Cube in the Hierarchy is currently a GameObject, not a Prefab. Lets now make it a prefab.

3. Drag the Cube from the Hierarchy onto the LabPrefabs folder and release. It should now appear in the folder and turn blue in the inspector.

4. If you now make changes to the prefab in the Project View it will change within the scene... with the Cube in the Project View LabPrefabs selected change the scale X value in the Transform Component from 1 to 5. The cube in the scene will change size to reflect the prefab changes, this is because the cube in the scene is now an **instance** or copy of the prefab.

5. Now select the cube that's within the scene and change the scale X value back to 1. Notice that it's now **bold**? Any values of the instance of the prefab that are **different** to those of the original prefab are shown in bold.

6. You may have noticed that an extra line appears at the top of the inspector when you have selected a prefab instance in the scene/hierarchy with several prefab options... Select, Revert, Apply. Lets have a go...

The **Select** option will select the original prefab within the Project view, showing you where it's located and its default values. Click it and the cube within the Project View should be selected/highlighted for you...

The **Revert** option will revert the selected prefab instance back to its original settings should you have changed any values. Click it and you should see that the cube in the scene now has its scale value set back to 5.

The **Apply** option will apply ALL the Component values of the selected instance to the original prefab and therefore to **every** other instance! Be wary of using this option since you have to want the applied changes **everywhere** in your game, else you could undo a lot of hard work.

A good example of the application of prefabs is imagine you are creating a racing car game with four players. You want each player to have the exact same car with the same behaviours etc., yet them be different colours. The solution? Prefabs. You'd create a GameObject which will be your car and add Components that provided the behaviour of a car. This GameObject would then be converted to a Prefab and four instances created within your scene. You then edit each instance to apply a different colour.

Hopefully this has given you an idea for how GameObjects, Components and Prefabs are all related.

First Project

Now that you have explored the Tanks Tutorial it is time to create your own project for the rest of the lab. This is done via `File -> New Project`. Don't worry about saving anything from previous scenes. Select the directory where you wish to save your project and name it Lab1. You can delete the Tanks project from your drive to save space if you wish.

As your project grows it is important to keep it organised. E.g. always save your scenes in the 'Scenes' folder that Unity created in your project. A Unity Project can have as many scenes as you wish. Scenes can be loaded and unloaded at runtime. Scenes can be as complex or as simple as you wish, and can feature whole game levels or just tiny sections, or even just menus.

Developing Terrain

We are going to start by making an outside game world, which we will start building using Unity's Terrain Tools. Thankfully, the Terrain Editor features a host of options so the creation of sweeping landscapes and mountainous terrain becomes quick and easy.

Create a new folder and rename it 'Terrain'. From the top menu select `GameObject -> 3DObject -> Terrain`. Within the Project window rename the terrain 'New Terrain' to 'Terrain' and drag it into the Terrain folder to keep the project neatly organised.

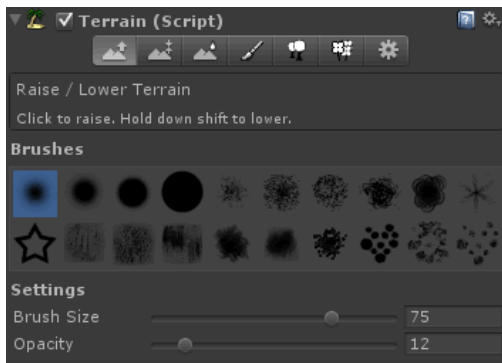
If you cannot see your terrain (large grey square) once you have created it, double click to focus on it. When the terrain is selected, the Inspector window provides several options for editing it. Let us have a look at them.

Topology

The Raise/Lower Tool (Keyboard Shortcut: F1)

To build up and develop your terrain select the `Raise/Lower Terrain Tool` circled below and click and drag across the terrain to start developing it. You can select from a range of brushes, brush sizes, opacity (softness) while raising your terrain.

If you wish to lower the terrain, simply hold the Shift key while you click and drag.

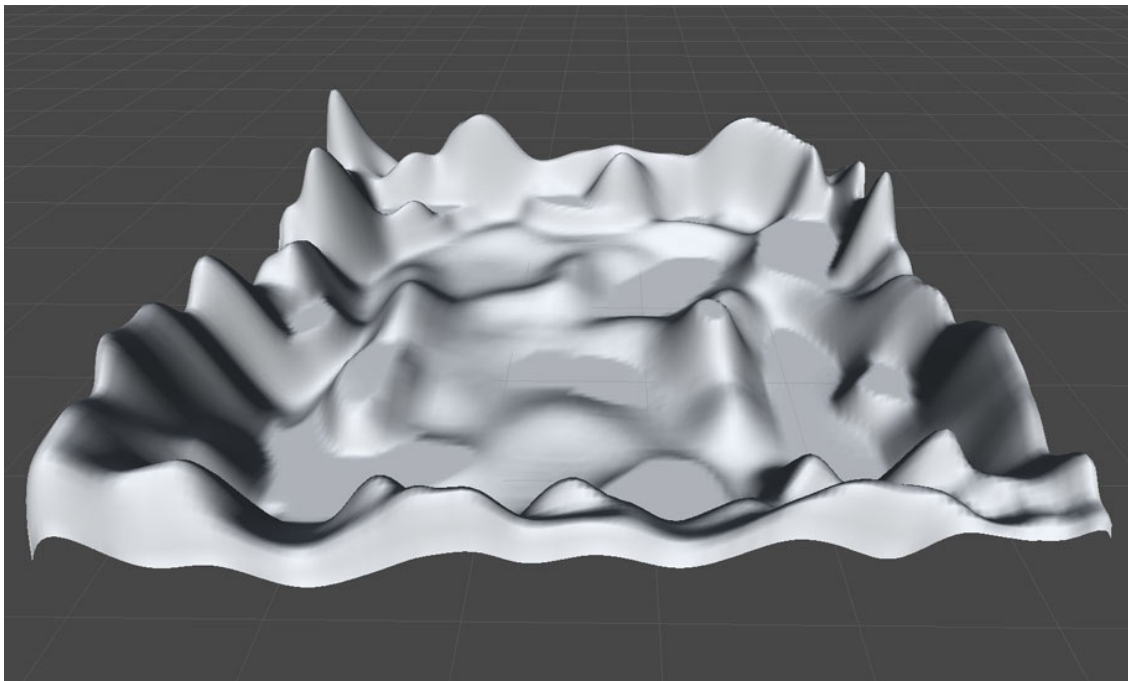


Try using different brushes and opacities to generate some mountainous terrain similar to the one below.

Just keep in mind you do not want to make the terrain too steep, since your player will then struggle to navigate it, unless you are purposefully making a section of your terrain that is inaccessible to the player.

It is crucial that the player is never able to see the edge of the map or walk off it, so ensure that the edges of your map are high enough.

Another thing to take into account with regard to the last point is the size, or resolution, of your terrain. By default it is 500 x 500. Make sure that your map is not too large and empty, as this makes for a boring experience.



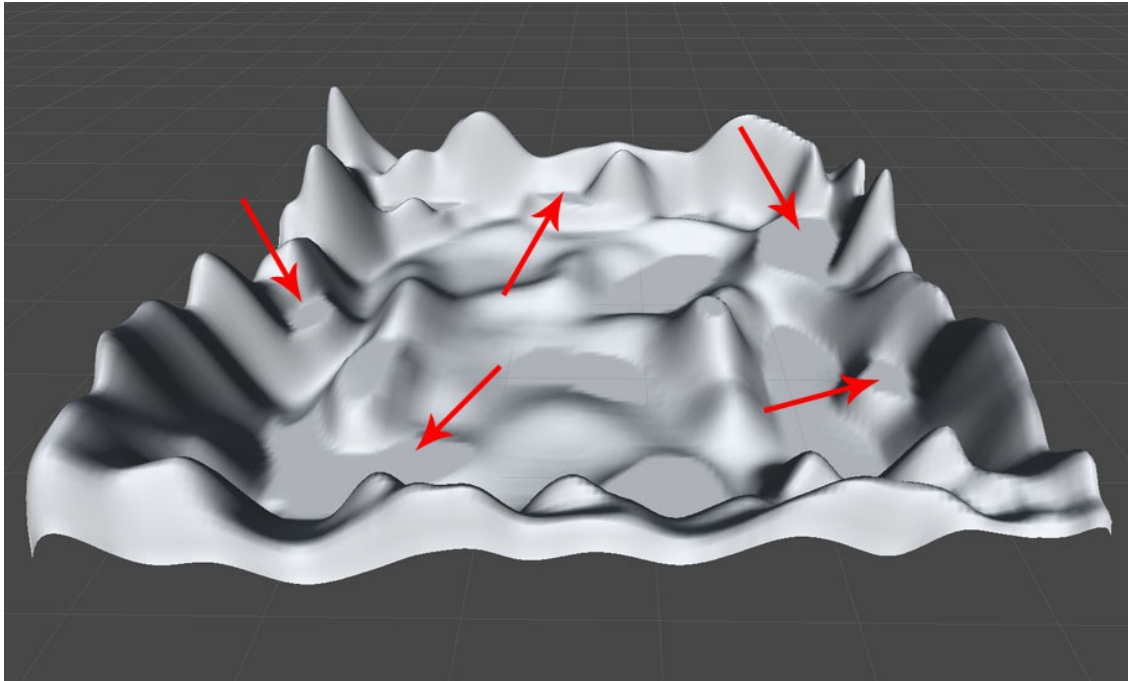
The above was created by starting with a large brush with a soft opacity to generate some smooth hill areas, before then reducing the brush size and increasing the opacity *slightly* to steepen particular areas.

If your terrain has sharp edges and is very steep try using a larger brush and softer opacity.

The Paint Height Tool (Keyboard Shortcut: F2)

Now select the 'Paint Height' tool, one along to the right from the 'Raise / Lower' tool. This is identical to the previous tool except that there is a maximum height the terrain can be raised to. You can Shift + Click to set the height to that of the terrain where your mouse is currently located.

Below, this tool was used to cut paths into and through the 'Mountains'. Use it to similar effect, or to create a plateau somewhere on your terrain, or a pathway through a mountain.



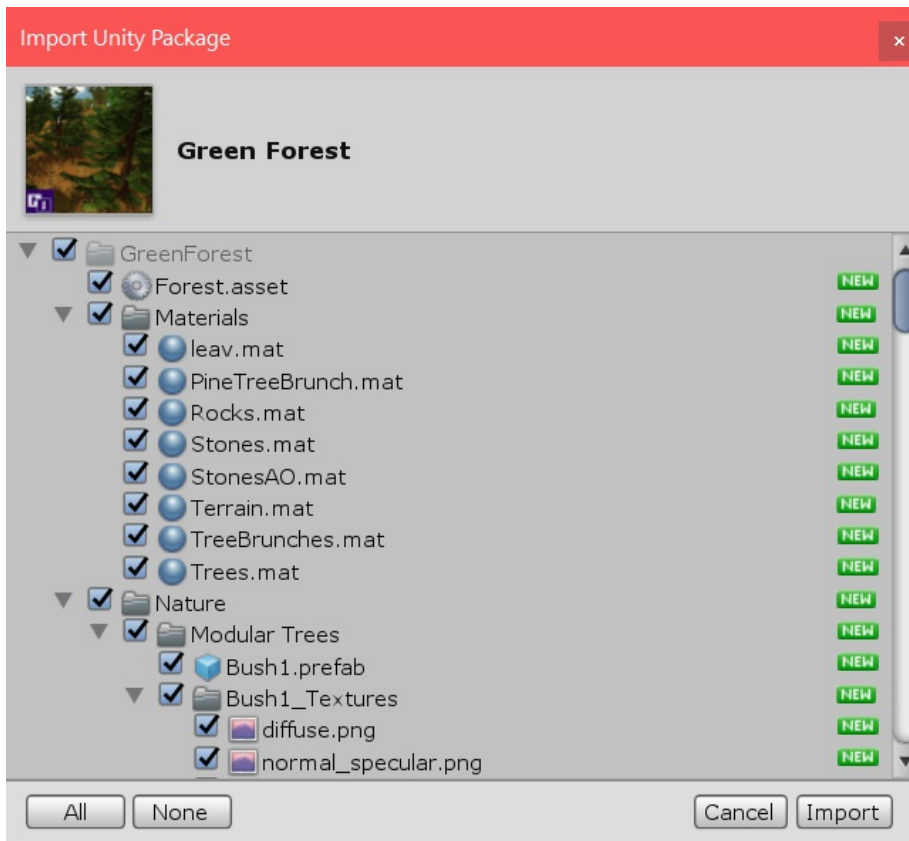
The Smooth Height Tool (Keyboard Shortcut: F3)

Next, select the 'Smooth Height' tool, next one along from the 'Paint Height' tool. This smoothens the terrain, making much smoother surfaces for characters or vehicles to navigate on. Use the Smooth Height tool on your terrain to smoothen out any jagged areas. If you find areas later on that you cannot access when playing your game, you can always come back and use this tool to smoothen those areas a little more.

Texturing Terrain

Now that we have developed the topology for our terrain, let us add for some textures and details.

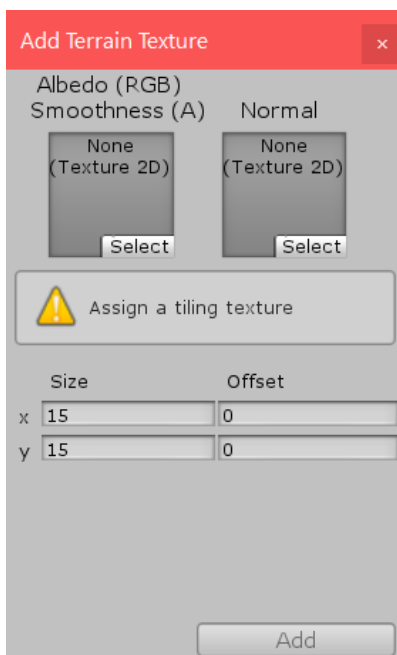
All the assets you need for this lab are on Blackboard (we will see later how to get new assets from the Asset Store). Get the files `Rocky Hills Environment - Light Pack.unitypackage` and `Green Forest.unitypackage` from Blackboard. Import the packages by drag them from the filebrowser and dropping them onto the `Assets` folder in the Project view. Select 'Import All' to have all assets available.



The Paint Texture Tool (Keyboard Shortcut: F4)

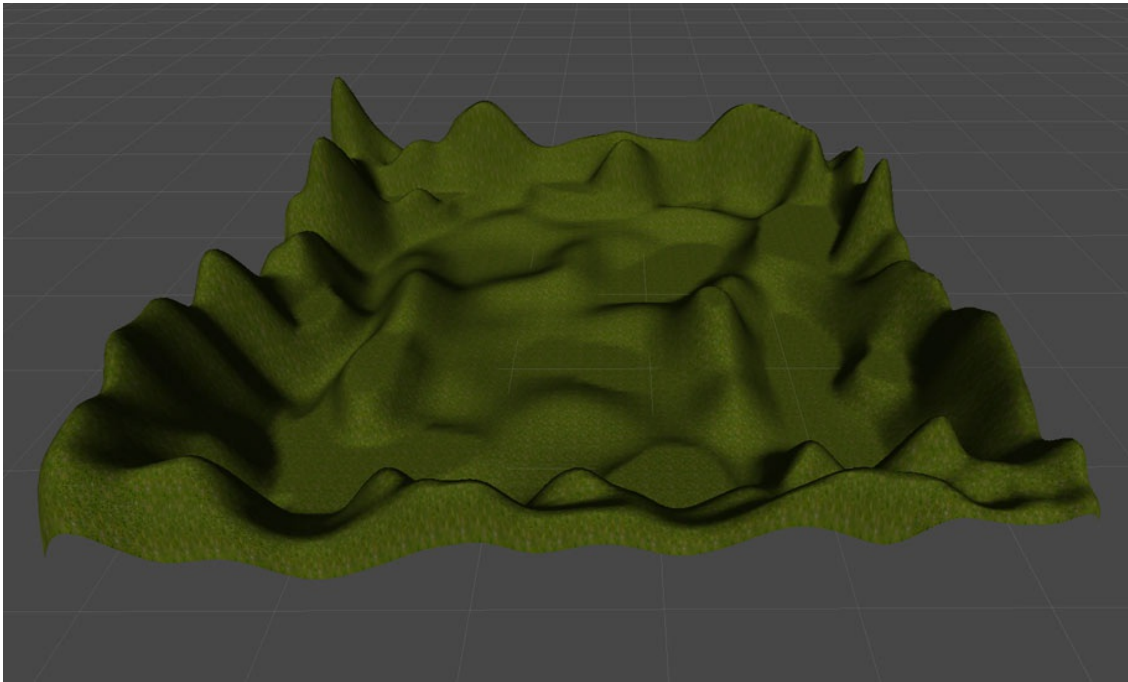
Reselect your terrain and select the Paint Texture Tool, next tool along. Beneath the brushes on the right select **Edit Textures** -> **Add Texture**.

Set the Albedo (RGB) texture by clicking **Select** in the 'None (Texture 2D)' box. Browse the available textures and select a texture such as 'Forest Floor'; click 'Apply'.



As this is the first texture applied to this terrain, the entire terrain is textured, since this is the base layer texture for the terrain. Do not worry too much that when zoomed out you can see an obvious repeat in the texture. When navigating the scene as a player you will

not notice it.

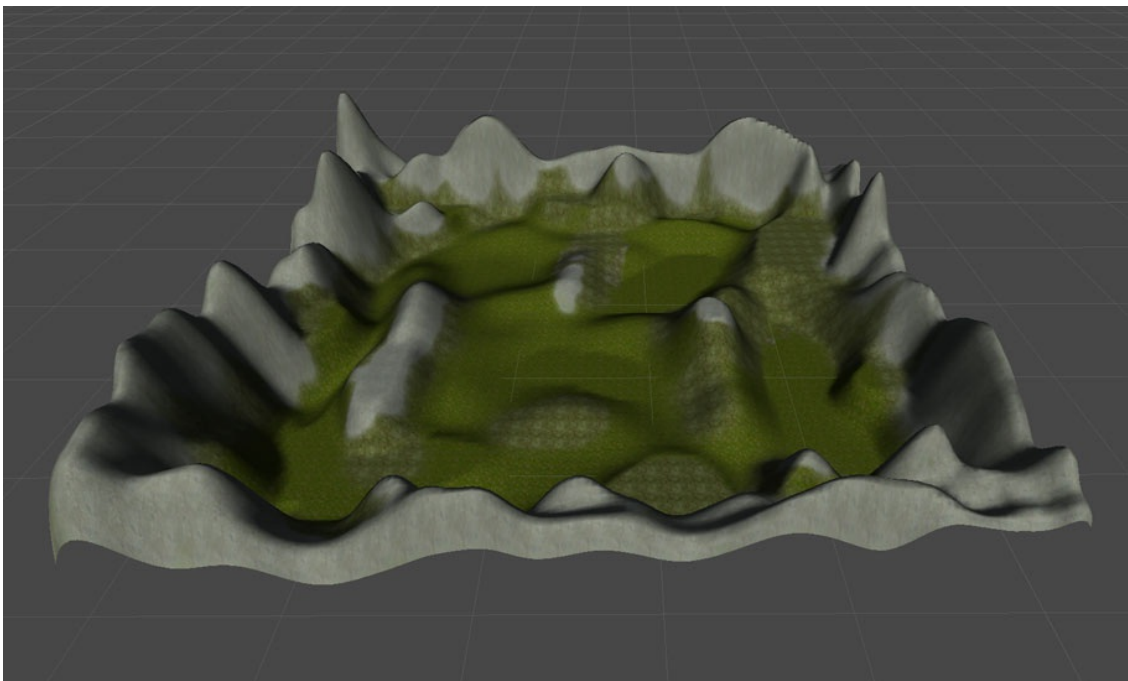


You can now overlay more textures by adding them and painting them over the terrain with the mouse, using various brush sizes and opacities. The texture that is currently selected will be the one that is painted. If you wish to change a texture you can do so simply by double clicking it.

To delete a texture, simply select it and click `Edit Textures -> Remove Texture`.

Add as many textures as you wish, experimenting with different brush and opacity settings to make your scene appear however you would like.

The below image uses the Forest Floor texture as the base texture, followed by Cliff (Grassy), then Cliff (Layered Rock) and then finally Snow for the peaks of the mountains.



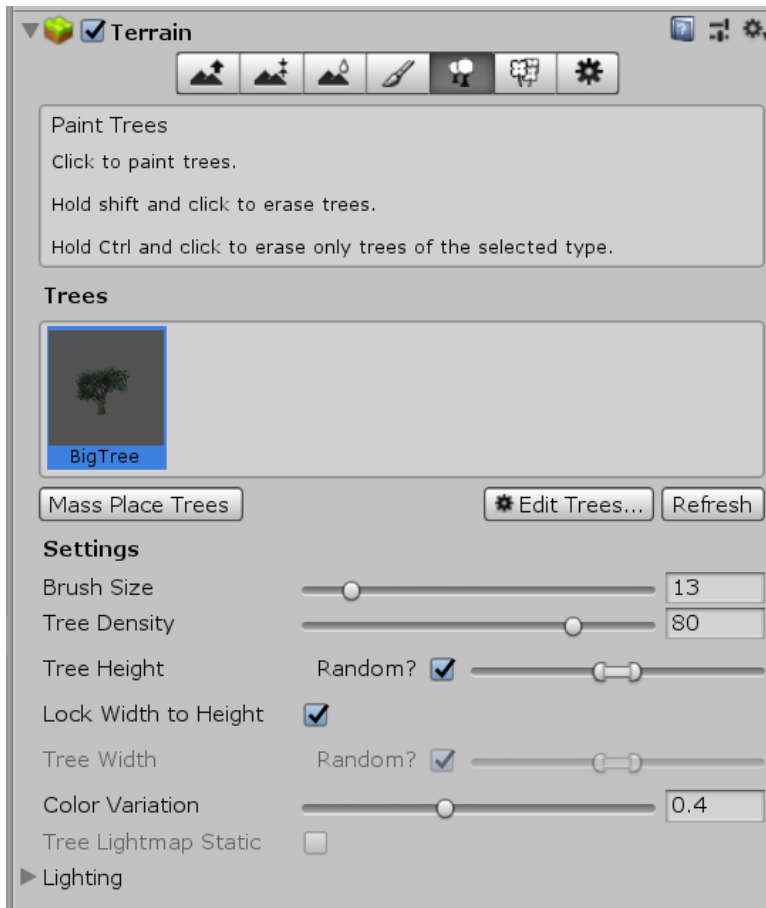
Detailing Terrain

Unity has a two built-in tools for adding details to a terrain.

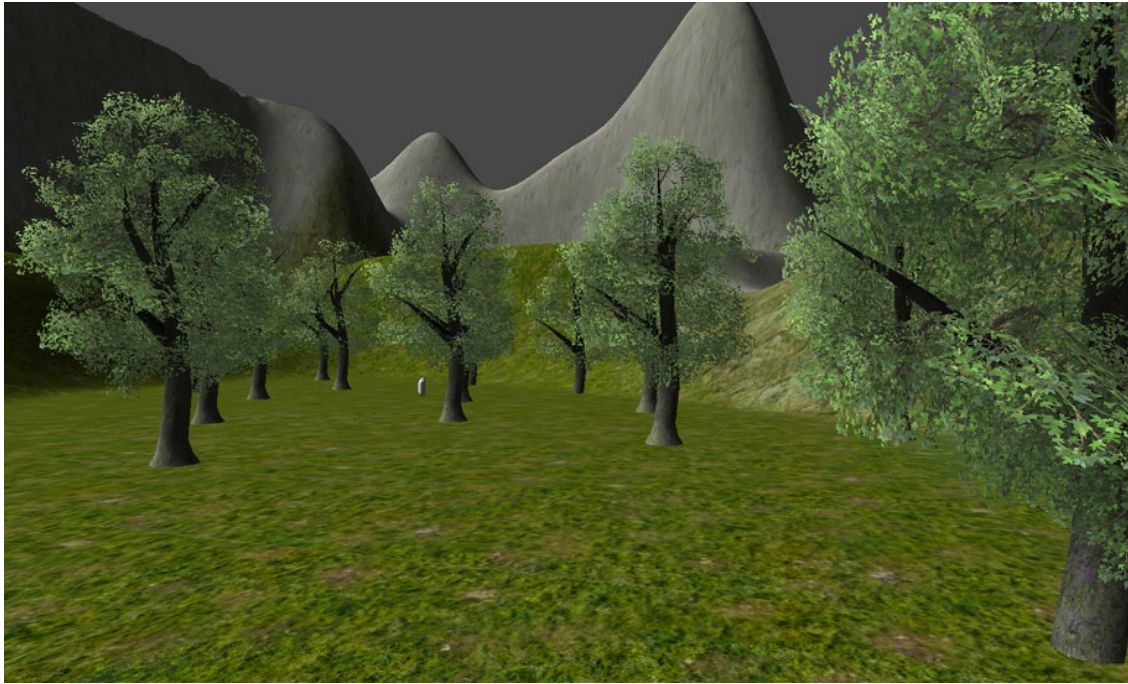
The Paint Trees Tool (Keyboard Shortcut: F5)

The 'Paint Trees' tool, the next tool along from the Paint Tool is used for placing trees. Click `Edit Tree` to bring up a dialogue to define a new tree. On the right side of the `Tree Prefab` field click the small circle to bring up the browser to select a prefab of a tree.

You will notice that this tool has quite a few more settings, such as 'Tree Density', which controls the percentage of trees within the brush area, and 'Tree Height / Width Variation', which will make the trees look a little less identical by randomizing the width and height of the trees placed.



There is also an option for mass placing of trees within the top menu, `Terrain -> Mass Place Trees`, where you can specify a number and Unity will place that many trees over your terrain, choosing randomly from those you have added within the Place Trees Tool.



In the above image, an alder tree was used. It was painted with a medium brush size and high density. The above was produced by clicking once at either end of the level.

The Paint Details Tool (Keyboard Shortcut: F6)

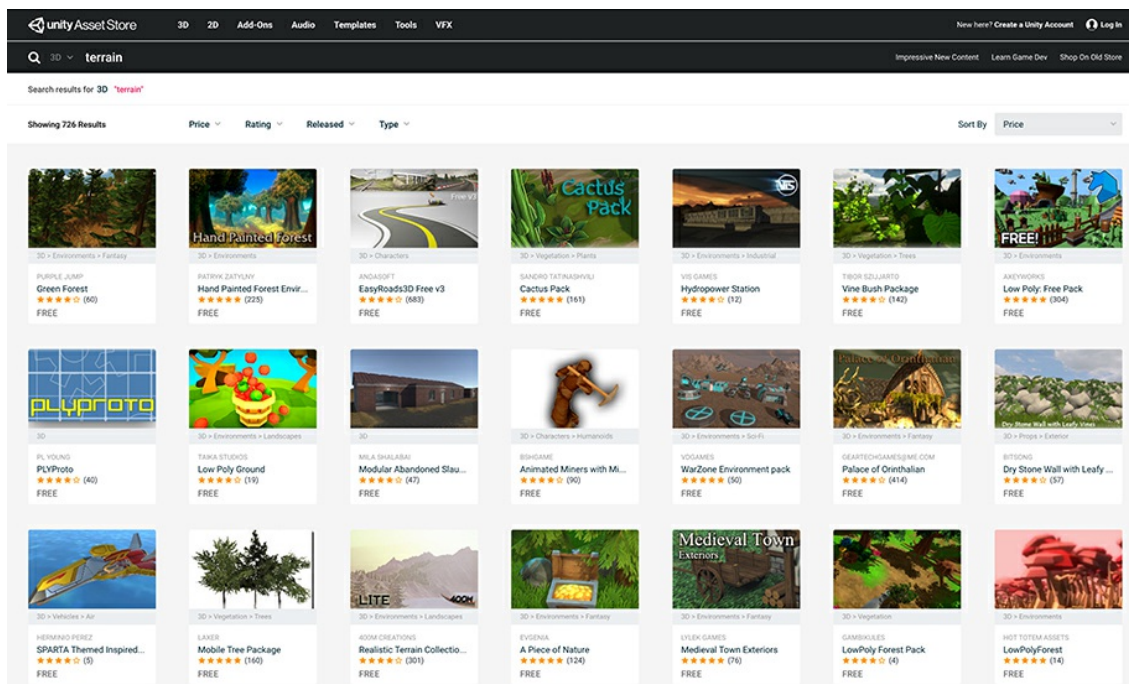
It is also possible to place smaller details, using the 'Place Details' tool in an identical way to the Place Trees tool. Use it sparingly, as placing a large amount of details can adversely affect the game's performance.

Terrain Settings The very last button within the Terrain Component within the Inspector allows you to control factors such as how far from the camera detail is drawn, from and at what distance billboarding (the process of replacing meshes/models with images) occurs. If you are walking around and you suddenly see trees appear and bend etc. then increase these values slightly.

You can also determine the speed of the wind within the scene, and how much the grass will bend in the wind. You can set how much the trees bend in the wind when you add or edit a tree using the Place trees Tool.

Using the Asset Store

You can also get assets from the Unity store. To download our Assets, we are going to use the Asset Store under Window -> Asset Store. Alternatively, navigate to the [Asset Store](#) in your browser (it is faster than in the Editor) and sign in with your Unity account. Now find some trees and terrain textures for your level. I would search for terrain within the 3D category, sort by price as below and see what comes up... ideally you can find a package with several ground textures and a couple of variations of tree...



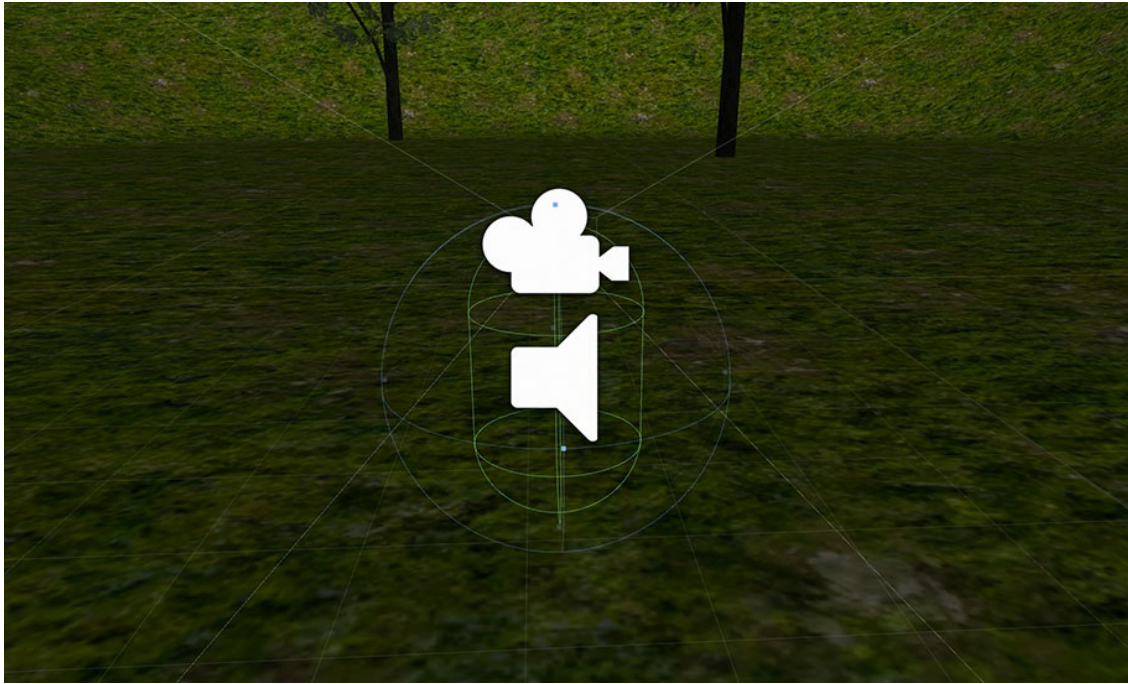
When downloading something from the Asset Store, click 'Download' and then 'Import' to open and download in Unity. When the download from the Asset Store within Unity is complete it will list the contents of the package, allowing you to select what you wish to import.

Using the First Person Controller Prefab

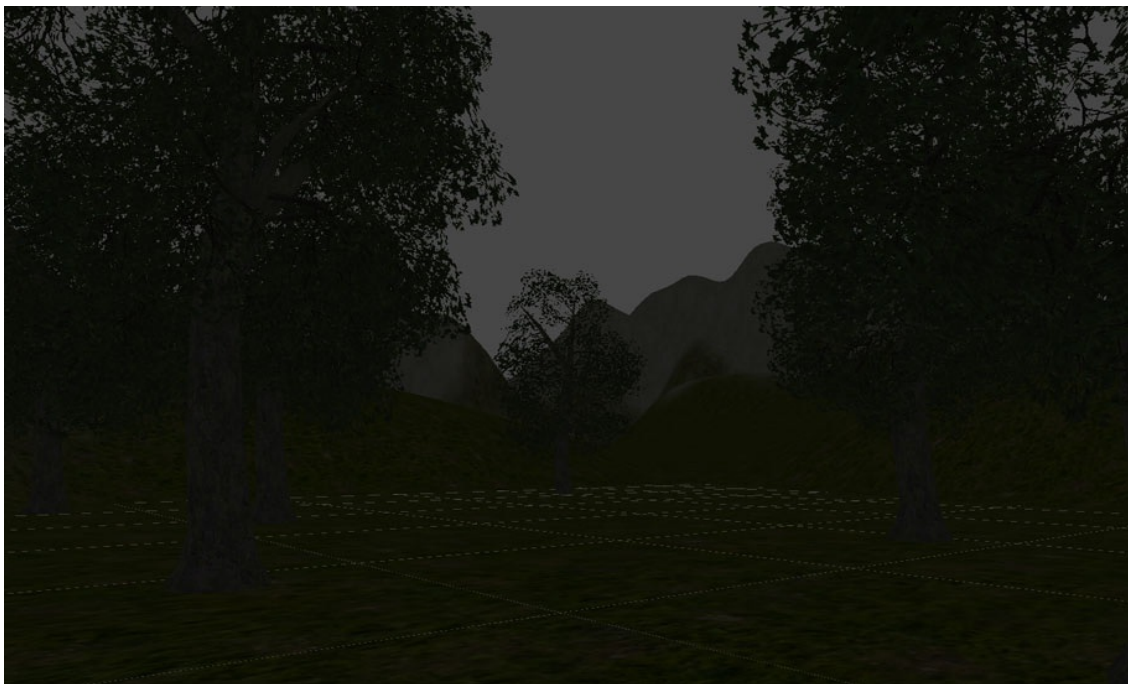
So now that you've crafted your first level, you now will want to explore it. If you hit play though you are going to be left disappointed with a static camera rendering the scene from some random angle... Our scene has a camera, but we currently have no way of controlling it and turning it into a Player. For this we need some form of player controller.

To begin, download the `Standard Assets.unitypackage` from Blackboard and import it into your project. This is a reduced version of the original [Unity Standard Assets package](#) which is full of really cool stuff.

1. Drag the FPSController prefab (blue square icon) from `Standard Assets -> Characters -> FirstPersonCharacter -> Prefabs` into the Scene view where you would like to begin playing from, ensuring it is placed above the terrain using the Transform tools. If you start the game and you fall through the terrain, then your first person controller is not above the terrain. In that case, try rotating your scene view and check that the green capsule outline is above the floor and not intersecting or beneath it.
2. Since our FirstPersonController contains a camera we do not need the default one within the scene, so go ahead and delete it.
3. Within the Toolbar click the **Play** button.

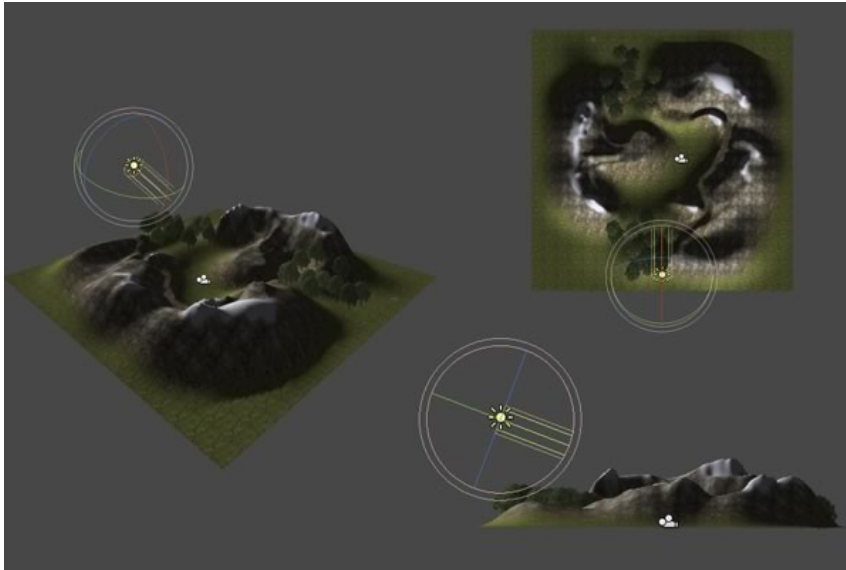


You should be able to run around your terrain using WASD or the Arrow Keys and turn around using the Mouse. You can jump with the space bar. You will also notice that, depending on which tree you selected, you can walk through the trees. This will be addressed shortly.



Lighting the Scene

There are various types of Lighting within Unity depending on what you are needing. For our scene to be in daylight we just need some global illumination so we are going to use a Directional Light. Go ahead and add one, `GameObject -> Light -> Directional Light`.



You may notice a difference as soon as the light is added depending on its direction but reposition it using the Transform and Rotation tools to light your scene properly. It may help to use the Orthographic views, particularly TOP to position and SIDE to rotate as shown above.

Hopefully now when you play the scene is lit a bit better than before as visible in the above image.

Adding a Skybox

To make the scene, we can change the default Skybox. A Skybox is rendered around the whole scene in order to give the impression of complex scenery at the horizon.

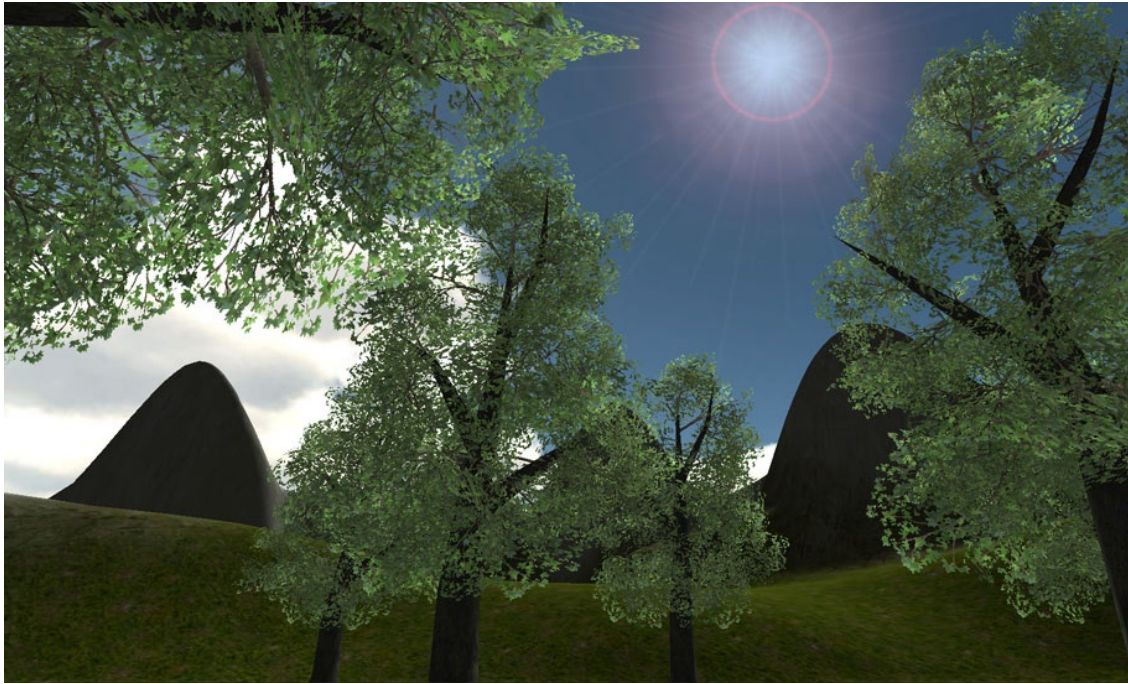
We have already imported a Skybox earlier with the Standard Assets. To apply a Skybox open the Lighting Window Settings from the top menu `Window -> Rendering -> Lighting Settings`.

Here, under the Scene tab and under Environmental Lighting you are able to edit several lighting settings for the scene such as fog, ambient light and the Skybox Material.

The Standard Assets package we imported when we created our project contains 'Sunny1 Skybox' which you can directly use.

If you have chosen a skybox with a sun you may have to alter the position of your directional light so that the shadows roughly align with the Skybox's sun.

If you wish to change the skybox, simply drag a different skybox onto the Skybox Material property to overwrite the currently assigned skybox. You can find more Skyboxes in the Asset Store.



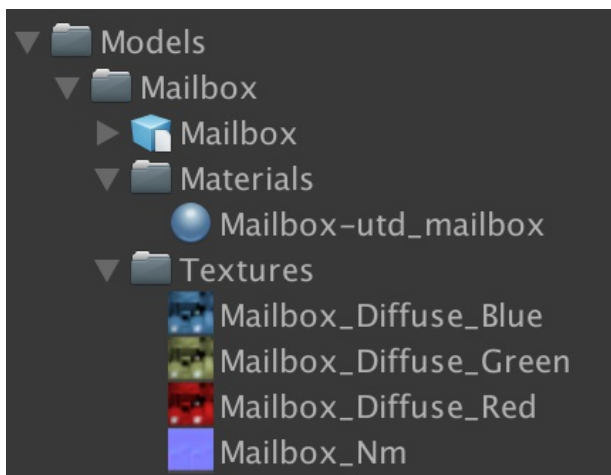
Models, Materials, Shaders, and Textures

Your map should look pretty good now considering you first opened Unity only a couple of hours ago.

So that's how you can develop a terrain which could serve as the basic ground for your game. What if you want buildings though? This is where we start to import 3d models into our scene.

Import the `mailbox.unitypackage` package from Blackboard (this is a slightly adjusted version [of this package](#) from the Asset Store.)

Within your Project View, expand `Models` -> `Mailbox`. Let us quickly look over what we have actually imported for each model.



We have a model file 'Mailbox' which has a blue cube icon, similar to the prefab icon we looked at earlier, but this has a white file icon to indicate that it is a model file. Model files hold the actual geometry data and can be in various formats such as `.obj` and `.fbx`.

Next, we have some texture files. Texturing determines what the model looks like, how it is 'textured' in the game. The Diffuse texture is the base texture for how the model is going to appear and you may see people describing '2k' or '4k' textures on the Asset Store, this is the quality, the higher the better but higher quality costs more in performance. Some

models might come with several Diffuse textures so you can skin the model in different colours for example. Models may also come with a Normal/Bump map texture which tricks you into thinking there are details in the model that look like 3d geometry but are actually just the texture - great for keeping a low poly count and saving performance. Some might also come with an emissive texture which controls how the light bounces off the model.

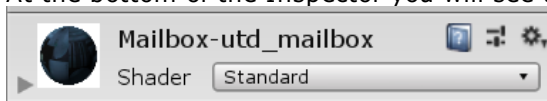
Finally, we have at least one Material for our model (some models may have several). The material(s) hold the information about which shader to render the model with and which textures to actually use. Most of the models we will work with will use the Unity Standard Shader. This shader allows us to customize how the model's surface reacts to light, so we can make something appear as if it's made out of metal or out of rubber by simply changing a slider!

Drag the mailbox model into the scene somewhere near the player. Alternatively, place it somewhere else and then move the player nearby.



By default the mailbox is blue and has no normal map. Select it within the Hierarchy and look at the MeshRenderer Component. Within the Materials property you can see that it is using the material 'Mailbox-utd_mailbox'.

At the bottom of the Inspector you will see the settings being used by this material:



The default rendering mode is Opaque, which means solid, i.e., not see-through. If you want to make an object transparent, you need to change the rendering mode to Transparent and then adjust the colour, Metallic and Smoothness.

The 'Albedo' setting is the base diffuse texture. In the right hand column you can find the RGBA (A=alpha/transparency) values if you wish to tint or make the object transparent. Further down is the Normal Map texture. When a Normal Map is set you can decide how strongly it is applied using the multiplier that appears in the right hand column. The little squares to the left of the names are where you can drag and drop to set those textures. Metallic and smoothness let you change the appearance of the object to make it appear as if it's made of different materials.

Try setting a different diffuse texture to change the colour and apply the normal map onto the relevant texture slot. Notice the difference that the normal map makes: suddenly the details like the rivets and ridges look like they are actual geometry!



Add another mailbox to the scene, not too far from the first one. It will look identical because it's using the same material so how can we have one red mailbox and one blue?

Your Turn:

1. Duplicate the mailbox material (Ctrl + D) and append it with '_red'. Rename the original to append it with '_blue'.
2. Select the red material and set the red mailbox texture as the Albedo texture of the material.
3. Select one of the mailboxes in the scene and drag the mailbox_red material to replace the current MeshRender material within the Materials array.



Hopefully you now have a base understanding of how models, materials, shaders and textures all come together in your game.

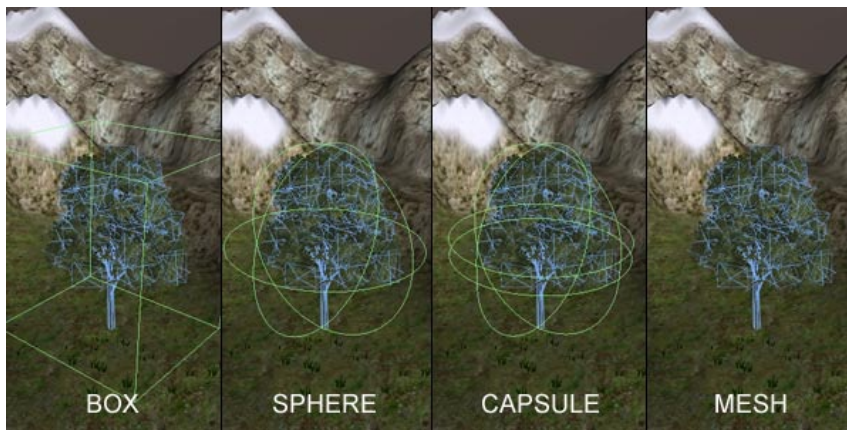
Your map should look pretty good now considering you first opened Unity only a couple of hours ago although you can still walk through trees and if you've tried you can also walk through the mailbox...

Collision Detection

Detecting collisions within your game is a crucial element of both realism and functionality; currently walking around your level you will be able to walk through the trees and the mailbox which isn't very realistic. Let's get that fixed with the use of *Colliders*.

Colliders are Components built in to Unity that provide collision detection using their various 'Bounding Boxes', the green lines shown surrounding the tree in the below image. The Mesh Collider however doesn't have any green lines surrounding in the image below it since it uses the Tree Mesh, outlined in blue. .

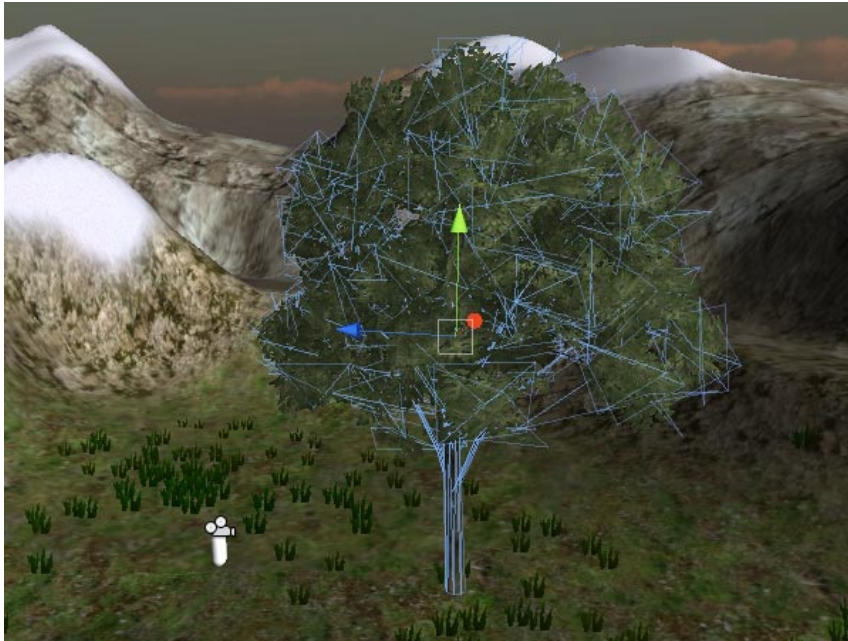
NB: If you are using a MeshCollider and wish to see the Collider's bounding box then just temporarily disable the MeshRenderer via Inspector.



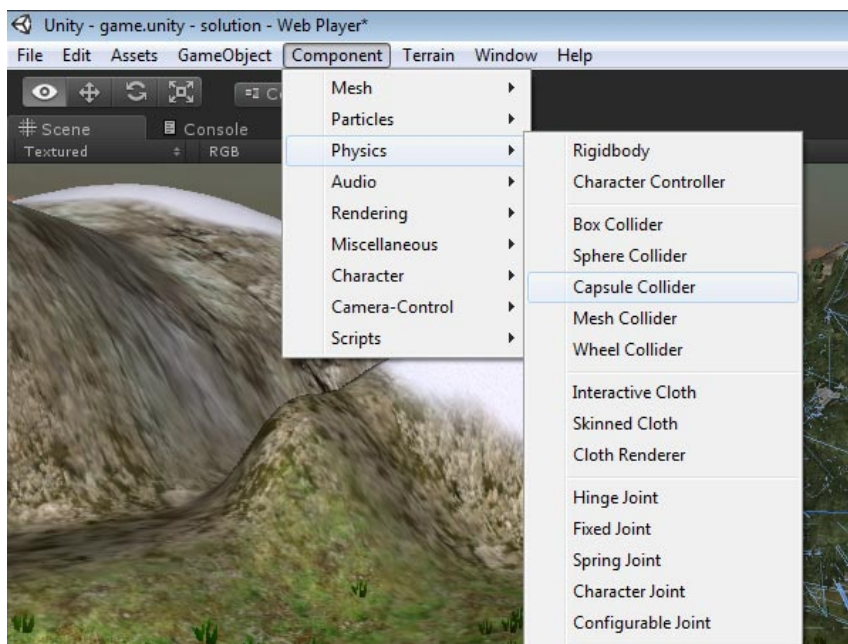
We are currently using the 'Place Tree' tool to place copies of the same 'template' tree model across the terrain. To make collision detection available for the trees in our scene, we will make this template into a prefab and attach a Collider to it that matches the tree trunk.

Within the Project Window, select the tree model that you have used and drag it into the scene. Position it close to your First Person Controller so you can easily and quickly navigate to it when the game is running.

Add '(Collision)' to the tree's name within the Hierarchy. For instance, 'Alder' would become 'Alder (Collision)'.




With the tree selected within the Hierarchy, add a Capsule Collider as a Component via the top menu: Component -> Physics -> Capsule Collider.

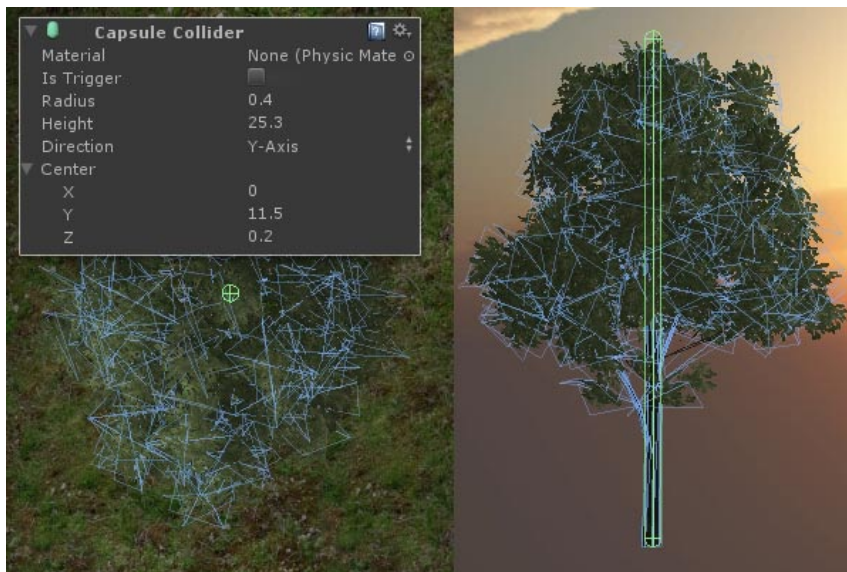


Click play and try and run through this particular tree. You will find that you cannot even get close to it because of the size of the capsule collider's 'Bounding Box'. Depending how close to the tree you are, you may even find yourself stuck within the bounding box.

Now, make sure to **stop** the game. We will resize the collider the fit the tree model.

With the tree selected, in the Inspector go to the Capsule Collider component and use the

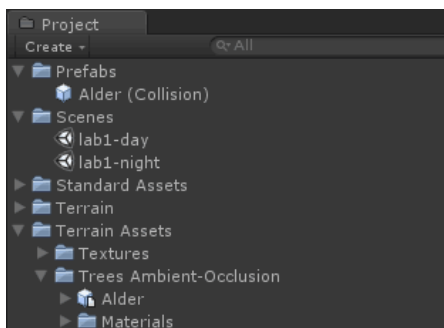
Edit Collider button  Edit Collider to adjust the collider's radius and centre point values, so that it only occupies the trunk of the tree as shown below. Depending which tree you have used these values may be different, but you want the collider to follow the trunk up the tree.



Run the game again and you should be able to walk right up to the tree but not through its trunk. We could currently still walk through the branches if we were tall enough. However, since we are not, it is not a problem. If we needed a more accurate collider that includes the branches we would use a Mesh Collider instead of a Capsule Collider.

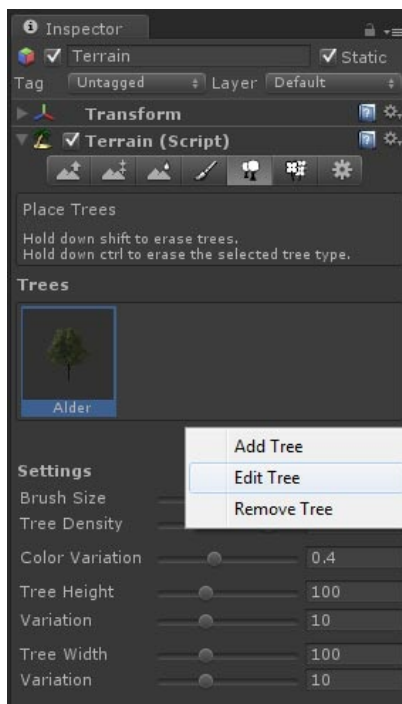
We now have a Tree that our user cannot walk through. However, we can still walk through all the *other* trees that were placed with the Terrain's Place Trees Tool. Let us replace those with the tree we have just added the Collider to. This will be achieved using Prefabs.

Within the Project Window create a new folder and rename it 'Prefabs'. Drag the tree from the scene onto the Prefabs folder to create it as a prefab within that folder. You can now delete the tree from the Hierarchy to remove it from the scene.

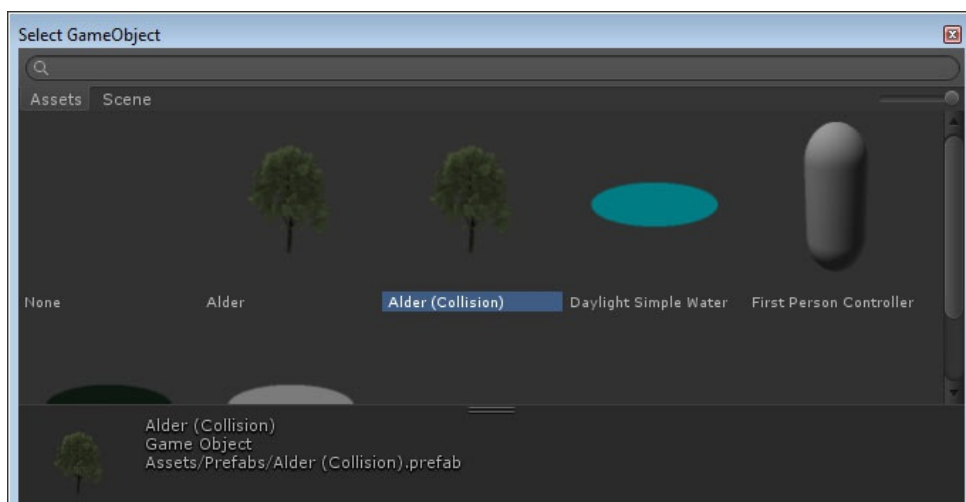


We now have a tree with a collider attached that can serve as a template for the rest of the trees in the terrain. All that is left to do now is to update the tree that the Terrain Tool is using to place the trees.

Select the Terrain in the Hierarchy and within the Inspector select the 'Place Trees' tool.



The current tree being placed over the map is the 'Alder' tree. Select it and Click 'Edit Trees...' to bring up the Edit Tree Dialog Box. Click the circle to the right of the selected tree, in this case Alder, and select the prefab we just created from the Select GameObject dialogue that appears.



All the trees will now be replaced with our tree that has the collider attached to it. To finish, select the terrain and at the bottom of the Inspector ensure 'Create Tree Collisions' is toggled on.



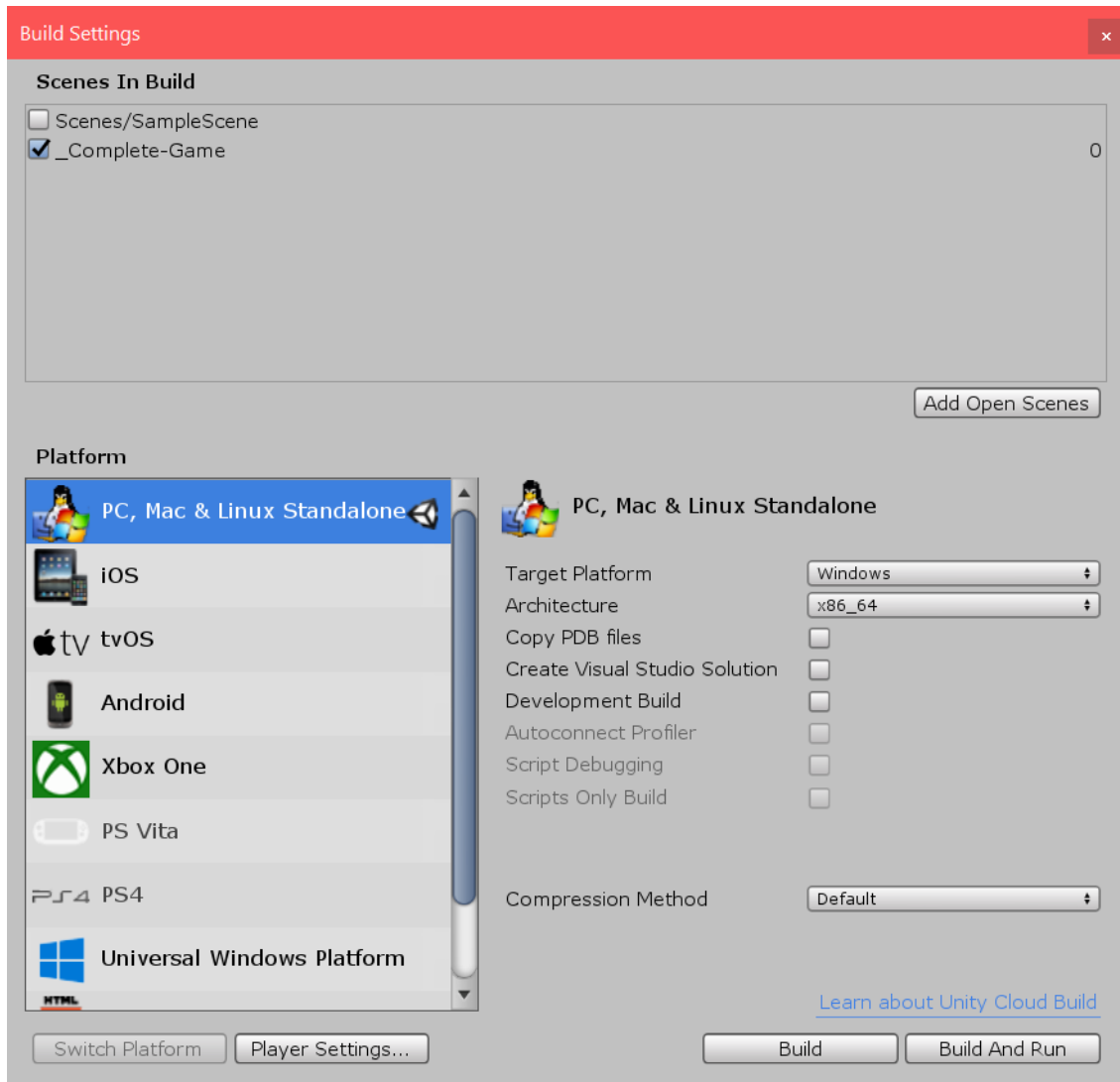
Now run the game again and you should collide with all of the trees that you have placed. If you have placed more than one type of tree, repeat the process for the other trees you have.

You can still walk through that mailbox though... What type of collider should we use for the mailbox? Try different Colliders and see which works best.

Publishing your Game (Build Options)

So far, we have only run the game inside the Unity IDE; to be able to run the game independently of Unity and publish it, we need to build it.

From the top menu select File -> Build Settings or Ctrl + Shift + B



Select the scene you want to build from 'Scenes In Build' or add the open scene with Add Open Scenes. Choose the target platform you want to build for (Windows/OSX) and select 'Build And Run'. You will then be prompted for a folder for the build files to be exported to. Unity will now export your build files and assuming no errors will launch your game as a standalone executable.