AD41700 Computer Games
Prof. Fabian Winkler
Fall 2013

Introduction to Unity3D (vers. 4.2)

Unity3D is a "game development ecosystem" (s. http://unity3d.com/unity/), it includes an environment for the development of interactive 2D and 3D content including a rendering and physics engine, a scripting interface to program interactive content, a content exporter for many platforms (desktop, web, mobile) and a growing knowledge sharing community.

Unity3D comes in two flavors – a free version which we are going to use in this class (download from: http://unity3d.com/unity/download/) and a pro version ($1,500).

Unity3D vs. Unity3D Pro

In contrast to the expensive Unity Pro ($1,500) the free version of Unity does not have: realtime shadows, realtime audio filters, custom splash screen, video playback/streaming, development/deployment possibility for iPhone, Blackberry and Android devices.

But the free version of Unity still has some very powerful features:
Physics engine, positional audio, web browser and standalone deployment, shaders, terrain editor, Mecanim animation system (vers. 4.x), …

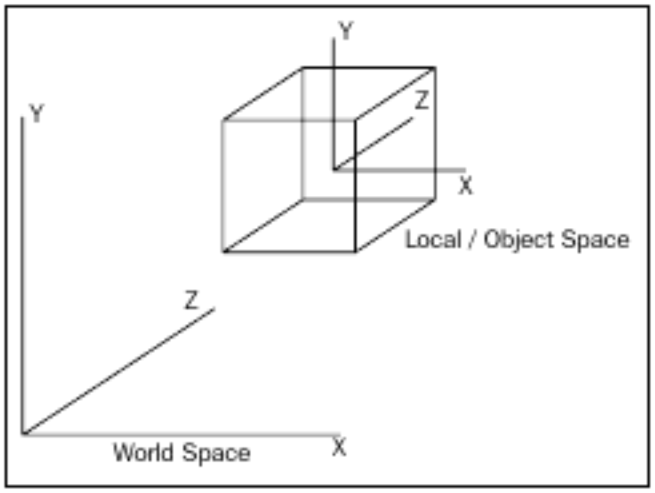
What the Unity tutorial series in AD4170 can and cannot cover

The goal of this workshop series is to teach students how to quickly implement a game design to be tested and tweaked. Due to the limited time we have and the complexity of the field of game design/development, I won't cover the creation of 3D or 2D art assets. This is part of the individual responsibilities of the interdisciplinary teams that are working together in the second half of the class. Resources will be given for good starting points to get started with asset creation outside of Unity3D (3D, 2D sound, etc.). Also, the approach to technical workshops in this class shifts from the beginning to the middle of the semester: from step-by step instructions to using Unity3D (first series of workshops) to a more independent study of all the features students might need for the development of their final projects. Since we are lucky to have quite a few students with expertise in 3D modeling, animation and game development in Unity3D in this class I invite those who are interested to propose specialized workshops in these fields. We can schedule these workshops later in the first of early in the second half of the semester.

Basic concepts

A. Coordinate Space

Three axes X, Y and Z – based on the Cartesian coordinate system (René Descartes, 1637)

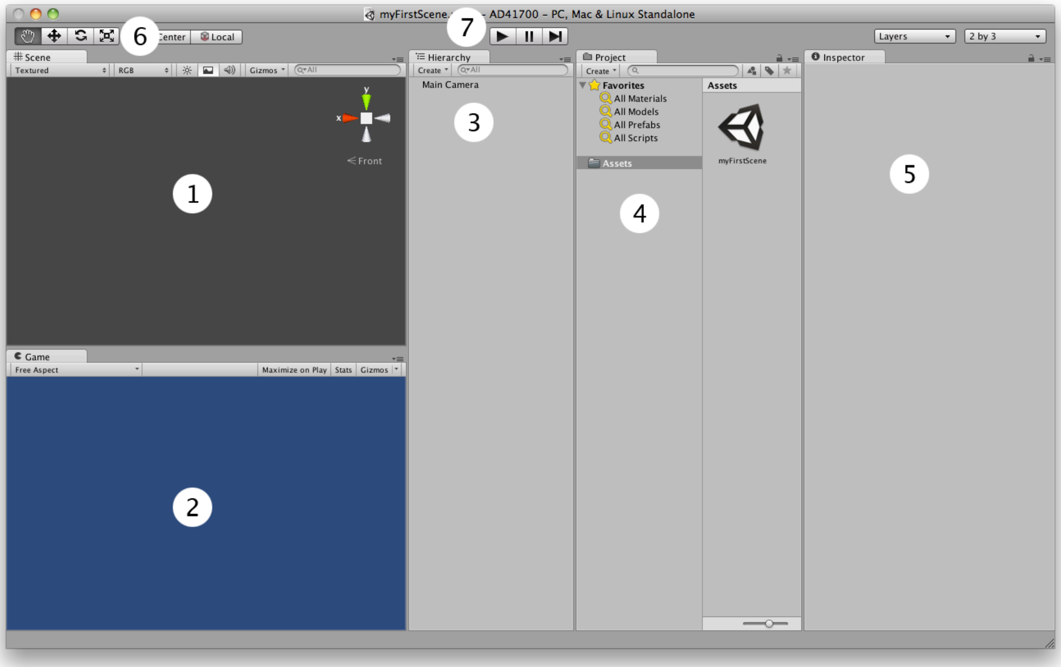


B. User Interface

`File > New Project`

You don't have to import any of the packages at this point, but do specify the save to location in the dialog window. Remember to keep all you project related assets in this location to avoid missing files and broken links later in the game development process.

`File > Save Scene as…`



`Window > Layouts > 2 by 3`

User Interface Components (see screenshot on previous page):

1. **Scene:** This is where you will place any visual assets in your Unity environment. It will update in real-time when you are previewing the game. Note the manipulator on the top right; this allows you to switch between a number of standard views. We are currently in the perspective view (toggle between isometric (2D) and perspective (3D)). Although this doesn't matter too much, it allows us to view our scene with a vanishing point, which is the standard way Unity games will display.

2. **Game:** When you're not actively running the game, it will show a rendering of how the game will look, ignoring graphical effects that need to be computed at run-time, from the point of view of the main camera. When you're previewing the game, you'll be playing through this window. Since our scene is currently empty, all this window is showing is the background color.

3. **Hierarchy:** This lists all the objects in the currently loaded scene, and any children they may have. **Children** are objects that can be thought of as subordinate to the parent object; wherever the top object moves, they'll follow, keeping the current offset they have to this object. This is an important concept for Unity beginners to understand; we'll cover it more in detail later and in the workshops.

4. **Project/Assets view:** This is a list of all custom assets for our game, including graphical assets, sound, scripts (more on these later), prefabs (pre- assembled game objects), and much more. Our current game is currently using only one empty scene (titled "myFirstScene").

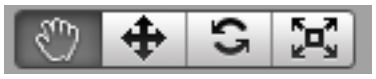
5. **Inspector:** Since we currently don't have any objects selected in the Hierarchy or the Project/Assets view, it's completely blank. The inspector allows us to look at and tweak individual settings of various game objects and assets, as well as adjust some global settings. The Inspector is content-sensitive and changes its parameters based on which game object/asset is selected. This is also a place to show you your project settings and preferences by choosing them from the Edit menu.

6. **Graphical icons for moving the scene and its contents**. The hand allows us to pan around the scene; when combined with other scene camera controls, Unity becomes very easy to navigate (see below). The icon on its right, which looks like four arrows, allows you to move a selected object around. We call this transforming the object. The next icon allows for rotation of the object, and the final one allows for uniform scaling of the object.

7. **Playback bar**. This allows us to play, pause, and stop running our game in the Unity editor. This is the quickest and easiest way to test and tweak the game.

Navigating the Scene Window

The scene view is what allows you to look around and move the visual assets you import into Unity. It's how you'll assemble your levels and place important things like lighting, trigger zones, audio, and much more. Being able to control the camera is important if you want to do anything at all with it.



Hand Tool (shortcut Q): drag around in the scene to pan your view. Holding down alt+drag will rotate the view, Ctrl.+drag will allow you to zoom. It is important to remember that this doesn't move anything in the scene, just your point of view.

Translate Tool (shortcut X): active selection tool, enables to drag an object's axis handles in order to reposition it.

Rotate Tool (shortcut E): using handles to allow you to rotate an object around either of its axes.

Scale Tool (shortcut R): works the same as the previous two tools, allows scaling of an object.

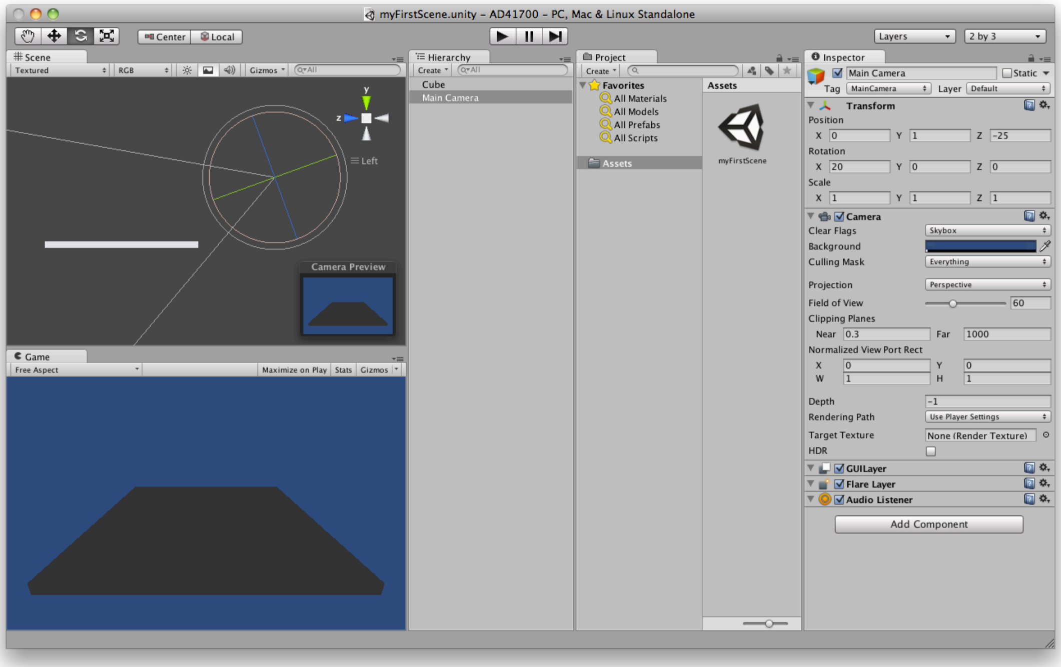
Your First Unity3D Scene

Now that you can look around the scene, let's learn a few ways we can place things in it. First we'll take a look at the basic game objects Unity can create without importing external assets. Unity 3 has geometric primitives (cubes, spheres, planes, capsule, etc.), lights, particle systems, cameras, and more that it can create without needing external assets. To access these, go to the top menu bar, select Game Object ->Create Other and make a choice. To begin with, try making a simple scene with a cube (functioning as a floor), a sphere, and a light. There's three different types of light – for now, a directional should work just fine, as light travels in rays with the direction of the arrows of the light, a good way to simulate the sun in Unity.

Start by creating a cube:
`Game Object > Create Other > Cube`

You can already use the Inspector window (after selecting the cube in the hierarchy) to modify its scale properties: Scale: X: 25, Y: 1, Z: 25 and to translate it Position: X: 0, Y: -10, Z: 0

Now we just need to move our camera a little back and point it downward to see the newly created box (it looks more like a plane now, see screenshot on the following page – I changed the view in the Scene window to left-isometric)
Position: X: 0, Y: 0, Z: -25; Rotation: X: 20, Y: 0, Z: 0;



Main Camera
Static
Tag MainCamera
Layer Default

Transform
Position X 0 Y 1 Z -25
Rotation X 20 Y 0 Z 0
Scale X 1 Y 1 Z 1

Camera
Clear Flags Skybox
Background
Culling Mask Everything
Projection Perspective
Field of View 60
Clipping Planes
Near 0.3 Far 1000
Normalized View Port Rect
X 0 Y 0
W 1 H 1
Depth -1
Rendering Path Use Player Settings
Target Texture None (Render Texture)
HDR

GUILayer
Flare Layer
Audio Listener
Add Component

Tip: If you would like to remove a game objects from the hierarchy/scene – select them press command+delete or go to Edit>Delete.

Basic lights

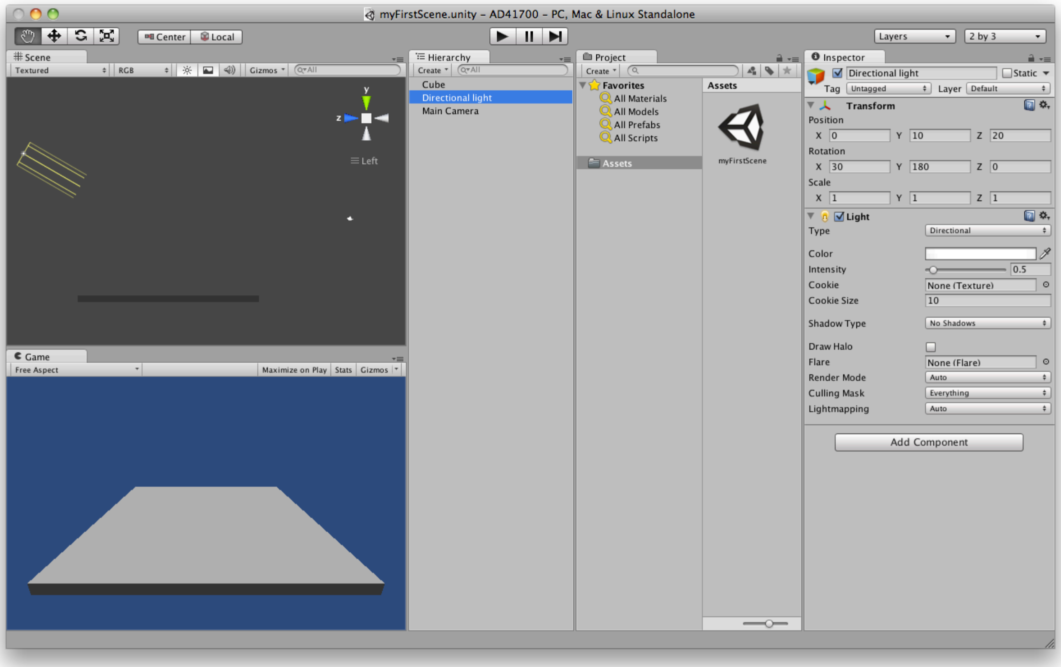
See: http://docs.unity3d.com/Documentation/Components/class-Light.html

- **Directional lights** are placed infinitely far away and affect everything in the scene, like the sun.
- **Point lights** shine from a location equally in all directions, like a light bulb.
- **Spot lights** shine from a point in a direction and only illuminate objects within a cone – like the headlights of a car.
- **Area lights** (only available for lightmap baking) shine in all directions to one side of a rectangular section of a plane.

We create a directional light source to illuminate the box in the Game window:

```
Game Object > Create Other > Directional Light
```

We should also move the light source a little back, up and tilt it downward to see its effect on the box object:
Position: X: 0, Y: 10, Z: 20; Rotation: X: 30, Y:180, Z: 0;

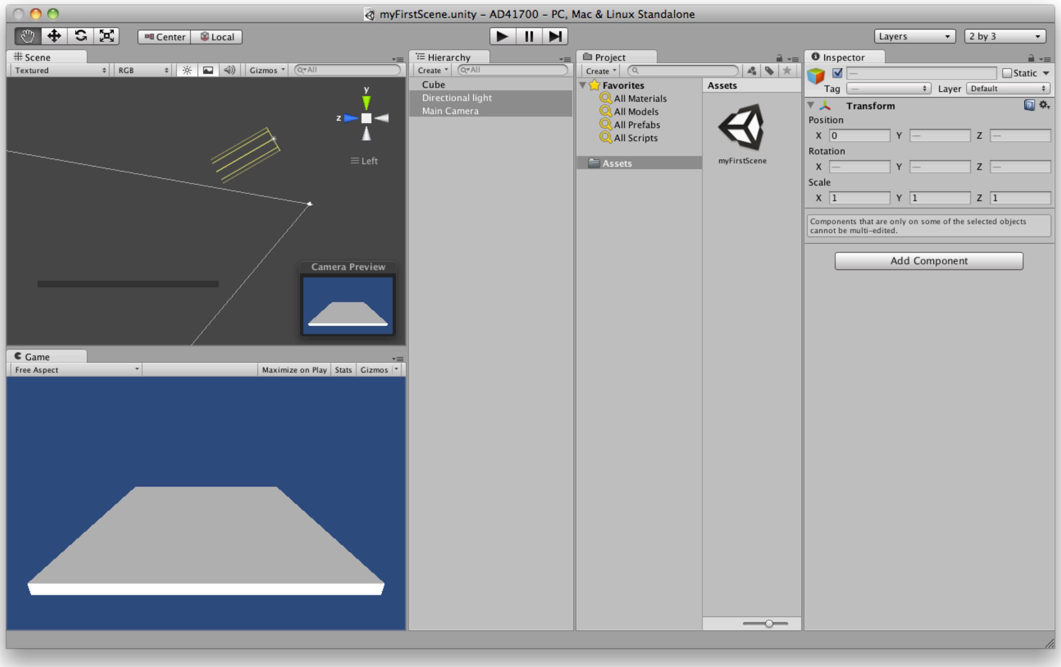


Directional light ☐ Static
Tag Untagged Layer Default

Transform
Position X 0 Y 10 Z 20
Rotation X 30 Y 180 Z 0
Scale X 1 Y 1 Z 1

☑ Light
Type: Directional
Color
Intensity: 0.5
Cookie: None (Texture)
Cookie Size: 10
Shadow Type: No Shadows
Draw Halo: ☐
Flare: None (Flare)
Render Mode: Auto
Culling Mask: Everything
Lightmapping: Auto

See the change in how the light affects the visual appearance of the box by experimenting with different angles and directions of your light source:
For example:
Position: X: 0, Y: 10, Z: -20; Rotation: X: 30, Y:0, Z: 0;



— ☐ Static
Tag — Layer Default

Transform
Position X 0 Y — Z —
Rotation X — Y — Z —
Scale X 1 Y 1 Z 1

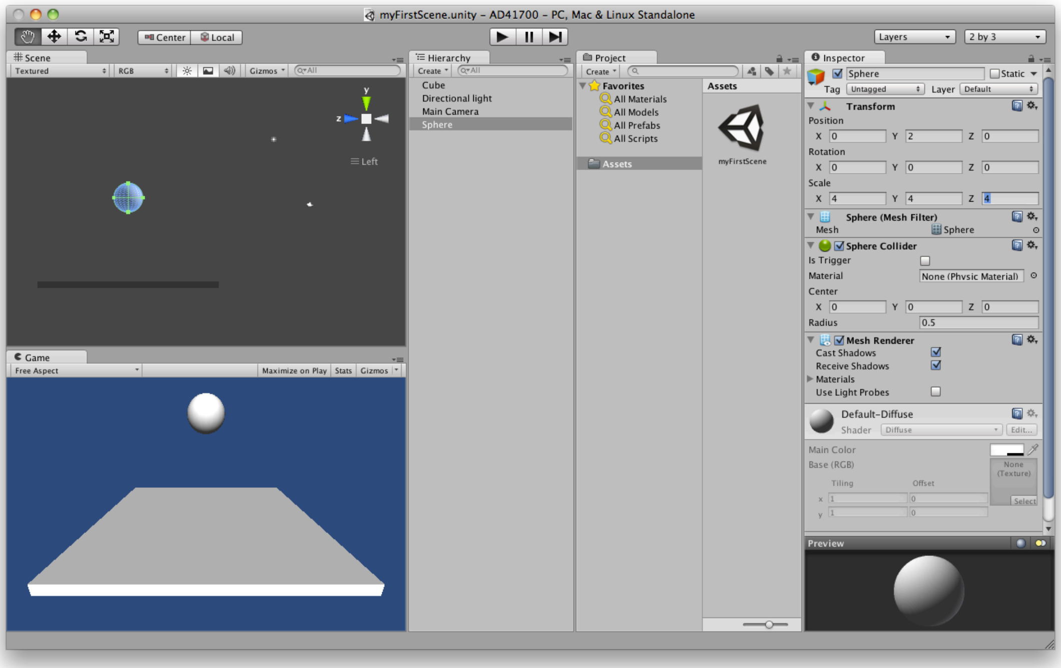
Components that are only on some of the selected objects cannot be multi-edited.

Basic Physics and Materials

First, create a sphere and place it above the ground plane:

`Game Object > Create Other > Sphere`

Position: X: 0, Y: 2, Z: 0; Scale: X: 4, Y: 4, Z: 4



Inspector
Sphere ☐ Static
Tag Untagged Layer Default

Transform
Position X 0 Y 2 Z 0
Rotation X 0 Y 0 Z 0
Scale X 4 Y 4 Z 4

Sphere (Mesh Filter)
Mesh Sphere

Sphere Collider
Is Trigger ☐
Material None (Physic Material)
Center X 0 Y 0 Z 0
Radius 0.5

Mesh Renderer
Cast Shadows ✓
Receive Shadows ✓
Materials
Use Light Probes ☐

Default-Diffuse
Shader Diffuse
Main Color
Base (RGB) None (Texture)
Tiling x 1 y 1
Offset x 0 y 0

Since there is a main camera that comes with every scene, you could hit the play button now and view your scene. Unfortunately, it will be entirely static. You can't control the movement of the camera and none of the scene is moving itself.

It would be nice if the sphere would behave as we expect it to from our observations in the real world – it would fall down and bounce (to a certain extend) when hitting the ground plane. We can simulate this behavior thanks to Unity3D's built-in physics engine.

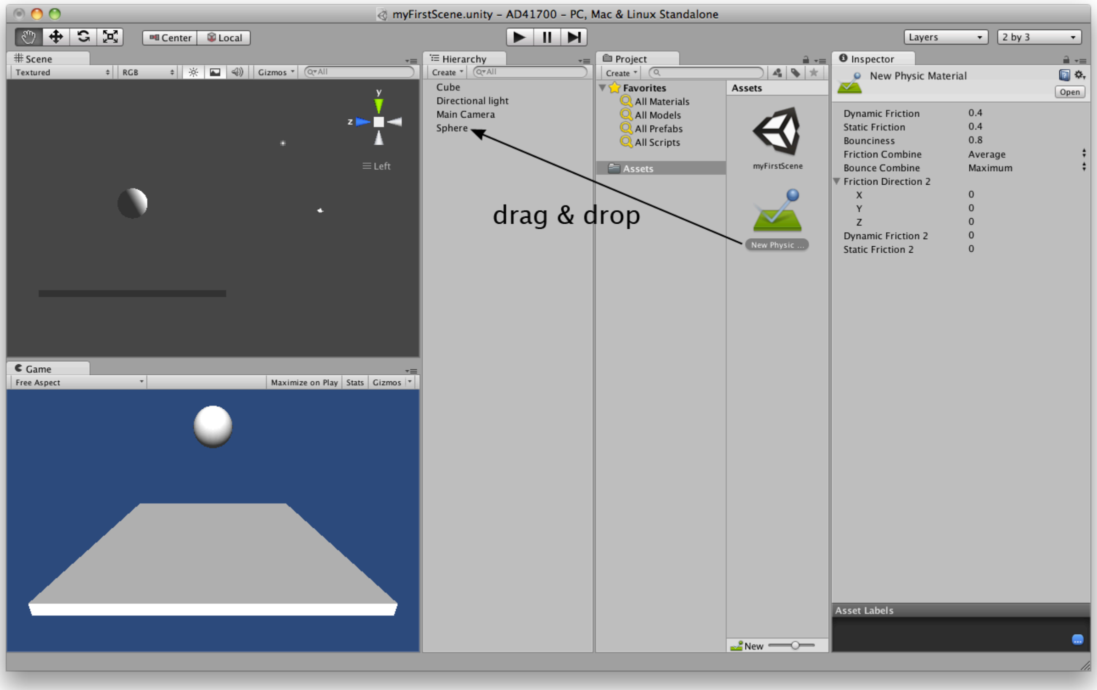
The first thing we need to do is to give the sphere a rigid body component. Select the sphere and go to: `Component > Physics > Rigid Body`.

Hit the play button and you see how the sphere is falling, but not bouncing.

In the next step we create a physic material, which will provide the material properties to make the game object bouncy:

`Asset > Create > Physics Material`

Then drag the newly created physics material from the Asset window onto the "Sphere" game object in the Hierarchy window (or directly onto the sphere in the Scene window).



New Physic Material

Dynamic Friction	0.4
Static Friction	0.4
Bounciness	0.8
Friction Combine	Average
Bounce Combine	Maximum
Friction Direction 2	
X	0
Y	0
Z	0
Dynamic Friction 2	0
Static Friction 2	0

I experimented with the properties of the physics material to make it behave like a rubber ball – from the default settings, I changed *Bounciness* to 0.8 and *Bounce Combine* to *Maximum* (for more information on the properties of physic materials see: http://docs.unity3d.com/Documentation/Components/class-PhysicMaterial.html).

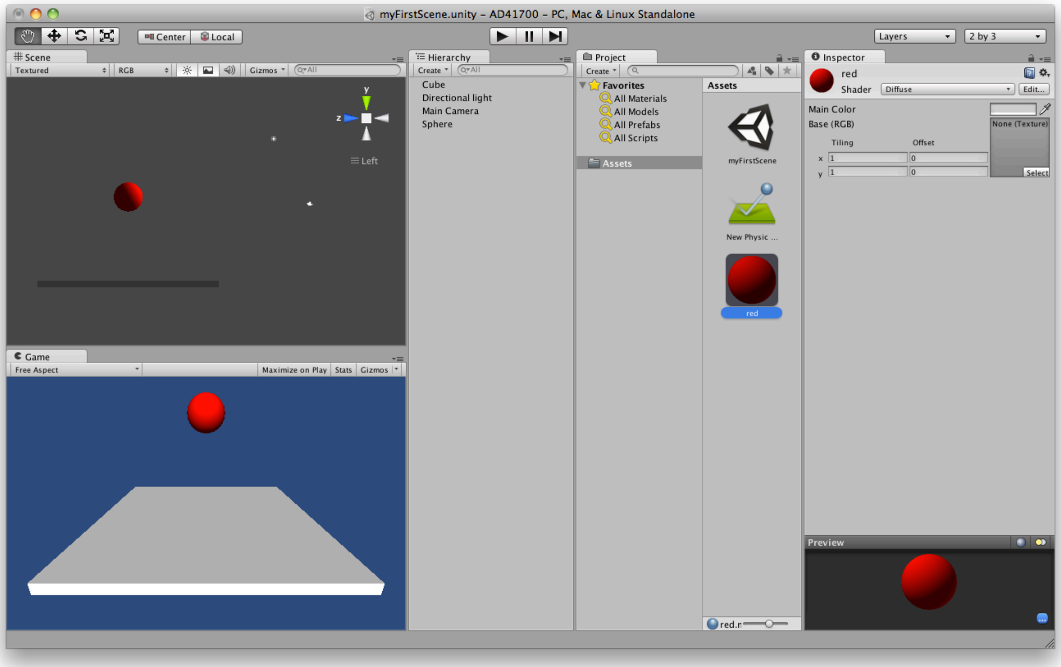
Hit the play button and see how the sphere is bouncing similar to a rubber ball.

You can now start experimenting with different heights from which the ball is falling as well as different angles of the box to see what response they create in the behavior of the sphere.

Creating Materials:

```
Assets->Create->Material
```

You will see the new material in you Project/Assets window and its properties in the Inspector window. In order to change its color double click the color swatch next to Main color and choose a different color. You apply the material to a game object by simply dragging it from the Project window onto the game object in the Hierarchy window (like you did when you applied the physics material above). You can then start tweaking the material's properties by experimenting with different shaders from the shader drop-down menu in the Inspector window.



Shader: Diffuse
Main Color
Base (RGB): None (Texture)
Tiling x 1, y 1; Offset x 0, y 0

Another way to change the visual appearance of game objects is to use 2D textures. A good overview of 2D texture features in Unity is in this chapter of the software's documentation: http://unity3d.com/support/documentation/Components/class-Texture2D.html. I hope we can take a closer look at textures in one of the following workshops.

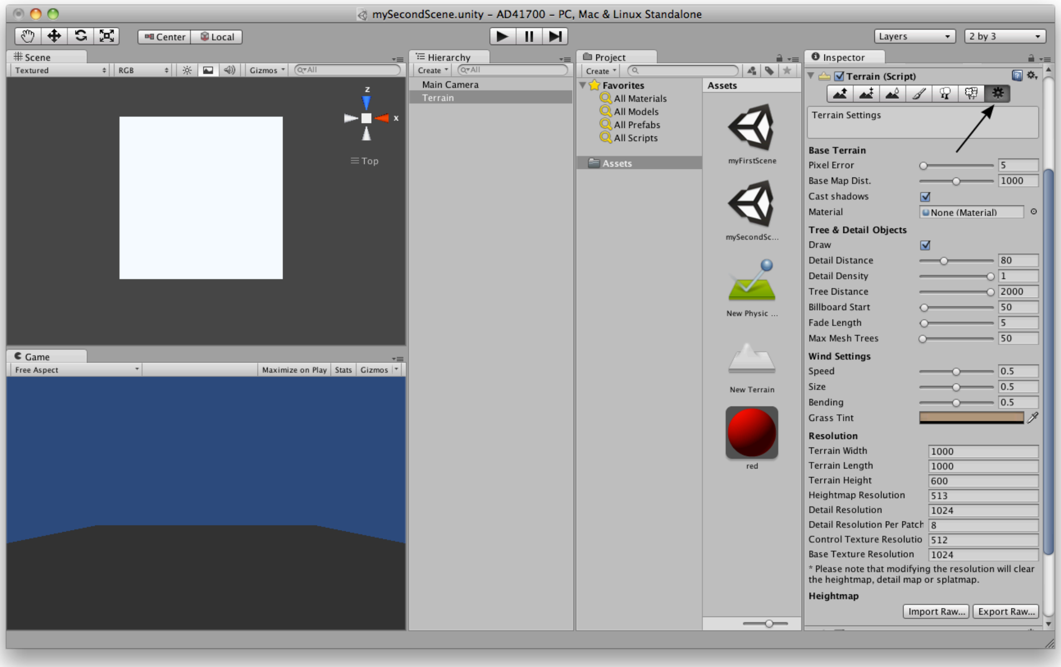
Using the Terrain Editor and Prefabs (First Person Controller)

After we have learned how to create simple primitive geometries in Unity3D and attach physics properties to them, let's explore how we can use prefabs to move around a virtual environment interactively, using a first person point of view.

We also create a terrain, so we have a very basic environment to explore. We can use Unity's Terrain editor to do this – we start with a simple plane, which can be easily turned into a topographical landscape.

Note: In 2004 the art ensemble Futurefarmers created *Fingerprint Maze* a very charming interactive artwork by automatically generating a terrain from audience member's fingerprint scans. Visitors were then able to navigate through their own fingerprint terrain in a first person perspective, see http://www.futurefarmers.com/survey/fingerprint2.php

First create a new Scene and save it. Then create a terrain: `GameObject > Create Other > Create Terrain`. Access its heightmap resolution by selecting the Terrain in the Hierarchy window and then working with its resolution properties in the Inspector window.



Terrain Settings

Base Terrain

Pixel Error	5
Base Map Dist.	1000
Cast shadows	✓
Material	None (Material)

Tree & Detail Objects

Draw	✓
Detail Distance	80
Detail Density	1
Tree Distance	2000
Billboard Start	50
Fade Length	5
Max Mesh Trees	50

Wind Settings

Speed	0.5
Size	0.5
Bending	0.5
Grass Tint	🟫

Resolution

Terrain Width	1000
Terrain Length	1000
Terrain Height	600
Heightmap Resolution	513
Detail Resolution	1024
Detail Resolution Per Patch	8
Control Texture Resolutio	512
Base Texture Resolution	1024

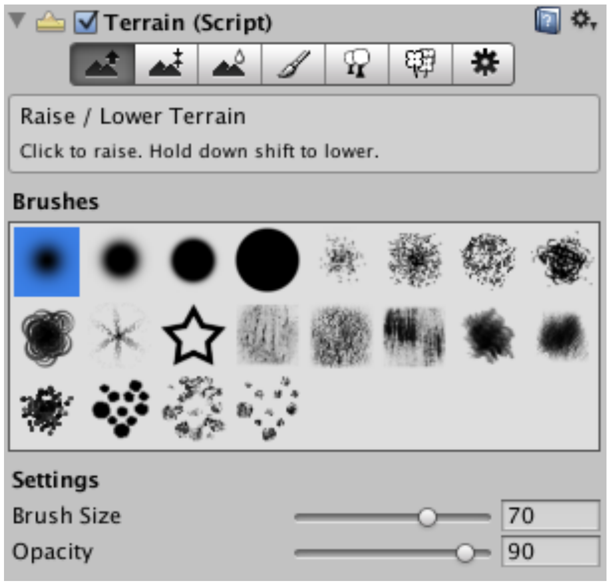
* Please note that modifying the resolution will clear the heightmap, detail map or splatmap.

Heightmap

I also moved the camera up and over a little, this helps you seeing the original plane for the terrain a little better: Position: X: 500, Y: 100, Z: 0

I changed the view in the Scene window to top view by clicking on the view manipulator icon in the Scene window's top right corner (top-isometric).

Now, using the Raise/Lower Terrain tool in the Inspector window (make sure the Terrain is selected in the Hierarchy window), I can simply "draw" on the top view of the ground plane in the Scene window to create certain terrain features.



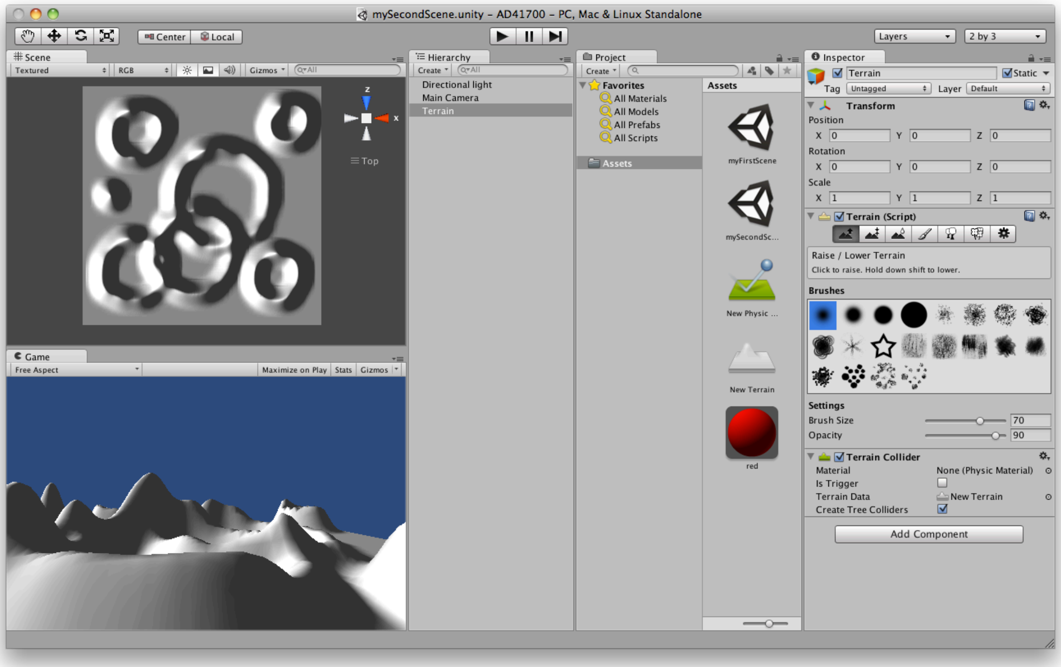
Raise / Lower Terrain

Click to raise. Hold down shift to lower.

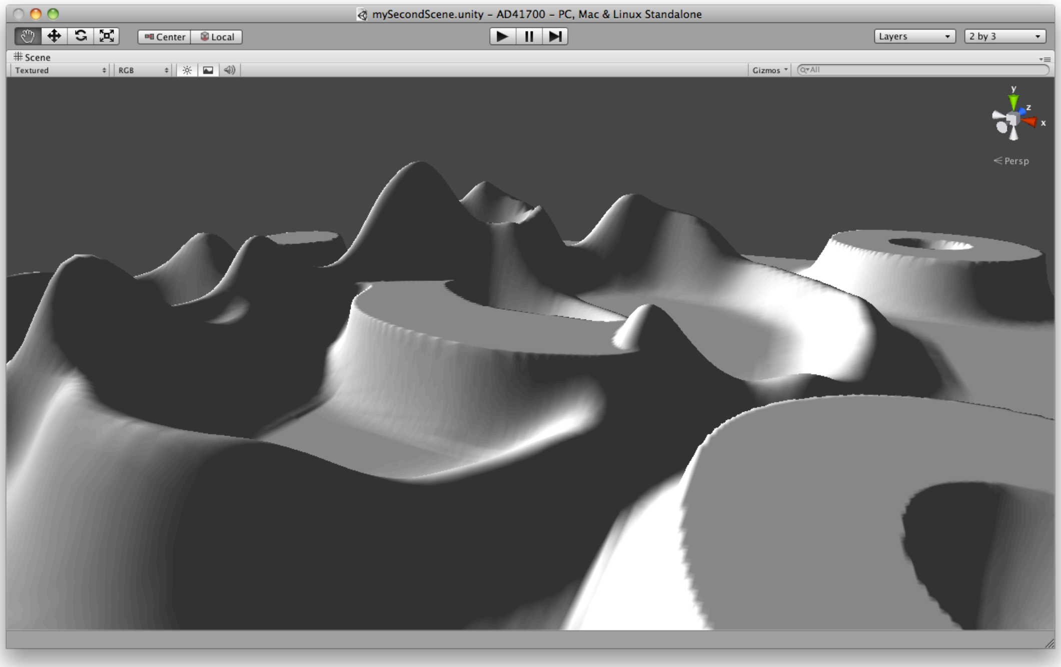
Brush Size	70
Opacity	90

I used Brush Size: 70 and Opacity: 90 to create shapes that are immediately visible, but you can fine-tune these settings based on what you would like to create.

I also already create a directional light (Position: X: 500 Y: 150 Z: 500 and Rotation: X: 20 Y: 80 Z: 0) so the terrain's features will look a little more dramatic.



Experiment also with some of the other terrain paint tools, such as "lower terrain height", "set terrain height" and "smooth terrain height."



To see a larger version of your terrain for working on it select the Scene window and press the "space" bar – this will maximize the window (hitting the space bar again allows you to return to the regular 2 by 3 layout).

This time we created a terrain manually by “drawing” on the ground plane. However, you can also import heightmaps to create terrains from actual images/maps/drawings. Here is a good series of introductory videos on how to do this: http://vimeo.com/album/150503 (going back to the Futurefarmer’s project, this is how you would get your fingerprint scan into unity!)

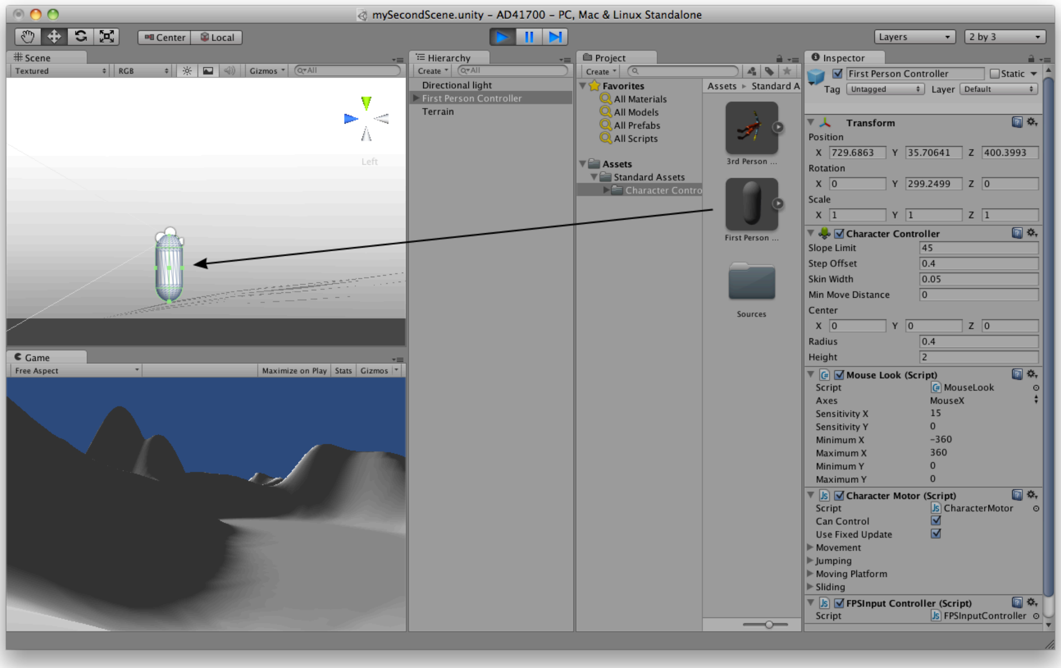
In the interest of time we will stop our experimentations with Unity’s Terrain Editor here and move on to creating a first person perspective that allows you to navigate the terrain interactively. (For a more in depth discussion of terrains and Unity’s Terrain Editor see chapter 2 in Will Goldstone’s book *Unity Game Development Essentials*).

First Person Controller

Unity makes it easy to move around a world interactively (either in a first person or third person perspective) using prefabs. Prefabs store a game object together with its components (transforms, appearances, scripts, etc.) and configurations for easy duplication/reuse.

If you haven’t included the Standard Assets Package while you created the new project you should import it now, specifically the Character Controller package: `Assets > Import Package > Character Controller`.

In the Project Window open the Standard Assets folder, open the Character Controller folder and drag the First Person Controller onto your scene. Delete the main camera, hit the play button and you are ready to explore the terrain you have just created. You can look around with your mouse and move with WASD or the arrow keys and jump with the space bar. Observe also how you can “slide” down some mountain slopes, if they are too steep for walking.



First Person Controller ☐ Static
Tag Untagged Layer Default

Transform
Position X 729.6863 Y 35.70641 Z 400.3993
Rotation X 0 Y 299.2499 Z 0
Scale X 1 Y 1 Z 1

Character Controller
Slope Limit 45
Step Offset 0.4
Skin Width 0.05
Min Move Distance 0
Center X 0 Y 0 Z 0
Radius 0.4
Height 2

Mouse Look (Script)
Script MouseLook
Axes MouseX
Sensitivity X 15
Sensitivity Y 0
Minimum X -360
Maximum X 360
Minimum Y 0
Maximum Y 0

Character Motor (Script)
Script CharacterMotor
Can Control ✓
Use Fixed Update ✓
Movement
Jumping
Moving Platform
Sliding

FPSInput Controller (Script)
Script FPSInputController

Make sure that the first person character controller has a Y position slightly greater than 1.0 – otherwise it won't stay on your ground plane and "fall through" the ground. The default scale in Unity is in meters – our 1000x1000 terrain is fairly large (a little more than half a mile in each direction) and it will take some time to explore it. Of course you can setup your own scale by adjusting Unity's scale-related settings in the project settings.