# Introduction to Unity

# Unity



Unity is a cross-platform game development system

Consists of a game engine and an IDE

Can be used to develop games and applications for many different AR/VR platforms



#### Installation

Unity is already installed on the lab computers in LWSN B131

If you wish to install your own computer:

- Download the 'Personal Edition': <u>https://unity3d.com/get-unity</u>
  - Make sure to get Unity version 2019.1.11f1
  - Make sure to add Android Build Support during installation.

Add Modules			)
Add modules to Unity 2019.1.11f1 : total space avai Platforms	lable 262.5 GB -	total space required 0 B	
> 🗸 Android Build Support	Installed	2.0 GB	
iOS Build Support	897.4 MB	3.6 GB	
tvOS Build Support	328.8 MB	1.4 GB	
Linux Build Support	176.9 MB	777.9 MB	
Mac Build Support (Mono)	81.2 MB	436.7 MB	
Universal Windows Platform Build Support	274.0 MB	2.0 GB	
Vuforia Augmented Reality Support	107.8 MB	306.7 MB	
WebGL Build Support	236.5 MB	857.8 MB	
Windows Ruild Support (IL 2000)	50 3 MR	301 0 MR	
CANCEL			DONE

#### Documentation

- Unity User Manual: <u>https://docs.unity3d.com/Manual/index.html</u>
- Scripting API: <u>http://docs.unity3d.com/ScriptReference/index.html</u>
- These pages should become your best friends.
- Also documentation on the OVR Utilities Plugin: <u>https://developer.oculus.com/documentation/unity/unity-utilities-overview/</u>

Unity Official Scripting Videos: These also serve as a good introduction to C#.

- Beginner Scripting Playlist: <u>https://www.youtube.com/watch?v=Z0Z7xc18CcA&list=PLX2vGYjWbI0S9-X2Q021GUt</u> <u>olTqbUBB9B</u>
- Intermediate Scripting Playlist: <u>https://www.youtube.com/watch?v=HzlqrlSbjjU&list=PLX2vGYjWbl0S8YpPPKKvXZayCjkKj4bUP</u>

# Unity Basic Concepts

Project - The project contains all the elements that makes up the game, including models, assets, scripts, scenes, and so on.

Scenes - A scene contains a collection of game objects that constitute the world that the player sees at any time.

Packages: A package is an aggregation of game objects and their associated metadata

# Unity Basic Concepts (continued)

Prefabs: A prefab is a template for grouping various assets under a single header.

- Prefabs are used for creating multiple instances of a common object.
- For example, you may have a large number of copies of a single element (e.g., street lights, trees)
- Prefabs can be instantiated during runtime



# Overview of the Unity IDE:



# **Editor Camera Controls**

- Controls:
  - Alt + Left Click & Move: Rotate Camera
  - Alt + Right Click & Move (Or Scroll Up/Down): Zoom in and out
  - Alt + Middle Click & Move: Move camera up/down or left right
- Flythrough Mode:
  - Click and hold right mouse button and now you can use FPS-like controls to move around through the scene (WASD, Q/E to move up down).
- Unity Documentation:

http://docs.unity3d.com/Manual/SceneViewNavigation.html

# Creating Geometry via the Unity Editor

GameObject Component M	lobile Input Windov	v Help					
Create Empty	Ctrl+Shift+N						
Create Empty Child	Alt+Shift+N	#Scane C.Ga					
3D Object	>	Cube					
2D Object	>	Sphere	# Scene	C Game			······································
Light	>	Capsule	Theded	(*)(20)) * (40) 🖬 (*)		General (2014)	Tag Untaged 1 Layer (Defect 1
Audio	>	Cylinder			- A-	>1	Position X 0.13 Y -0.04 2 -0.15
UI	>	Plane	_		-0-	<td>Scale X 1 V 1 Z 1</td>	Scale X 1 V 1 Z 1
Particle System		Quad					Mesh Cube C
Camera		Ragdoll					Is Trigger
Center On Children		Terrain					Material         None (Physic Raterial)         C           Center         X (0)         Y (0)         Z (0)           Size         X         X         X
Make Parent		Tree					x 1 Y 1 Z 1 T Kesh Renderer G 0
Clear Parent		Wind Zone					Cast Shadows Co
Apply Changes To Prefab		2D Test					Use Light Probes 🖬 Reflection Probes Bland Probes 1
Break Prefab Instance		SD TEXT					Ancher Override None (Transform) 0 Default-Material
Set as first sibling	Ctrl+=						Add Component
Set as last sibling	Ctrl+-						
Move To View	Ctrl+Alt+F						
Align With View	Ctrl+Shift+F						
Align View to Selected							
Toggle Active State	Alt+Shift+A						

# Setting Up The Scene Camera

- Do not confuse the scene camera with the editor camera.
- Unity scenes by default come with a "Main Camera." Notice the tag of "MainCamera" in the inspector, this will be useful for accessing the camera from your scripts.
- "Camera Preview" box is useful to see what your camera can see.
- "Camera Preview" is what you will see when you hit Play.



# Setting Up The Scene Camera

- Moving the scene camera can be done manually by changing the position/rotation/scale in the Inspector.
- Or you can move the editor camera around as mentioned earlier, select the camera and align the camera to the view. Note that your camera should be selected before doing this.



### Import External Objects

- Create and export an object from Maya/Blender/3ds Max as either an \*.OBJ or a \*.FBX. You can save this anywhere.
- Then import this asset into Unity. Unity will take care of everything for you.
- Alternatively, you can just save your \*.OBJ or \*.FBX inside the "Assets" folder.
  - You will need to right click on the folder it is in and click "Refresh" to get it to show up.
- Click and drag the object from the assets library into your scene hierarchy and it should now show up!



# Game Objects

Game Objects: The game objects are all the "things" that constitute your scene.

- Light sources
- Audio sources
- Cameras
- Gameplay Logic
- User Interface
- Etc.

GameObject: http://docs.unity3d.com/ScriptReference/GameObject.html

#### Everything is a "GameObject"

- A *Game Object* does nothing on its own.
- *Game Objects* always have a *Transform* component which has a position/rotation/scale.
- Must add *Components* to the *Game Object* to give it some behavior.

O Inspector	🔀 Navigation	Services				â +m
Sphere						🗌 🗔 Static 🔻
Tag Untagged	i .	t Lay	er	Default		1
▼↓ Transform	n					0
Position	×	-0.0212254	Y	1.529108	Z	-0.7281733
Rotation	x	0	Y	0	Z	0
Scale	x	1	Y	1	Z	1
V Sphere (M	esh Filter)					0
Mesh		Sphere				0
ls Trigger		♪ Edit Coll	ide	r		
Material	N	lone (Physic M	later	rial)	-	0
Center	x	0	Y	0	Z	0
Radius	0	.5				
<ul> <li>▼ → Lighting</li> <li>▶ Materials</li> <li>Dynamic Occlude</li> </ul>	derer ed 💽	1				0
Default-	Material		_		_	۵ 🗋
Shader	Standard					•)
	A	dd Compone	nt			

🔰 🗹 Directional Ligh	t			Static	8
Tag Untagged	1	Layer Default		_	4
↓ Transform				6	X
Position	X 0	Y 3	Z 0		
Rotation	X 50	Y -30	Z 0		
Scale	X 1	Y 1	Z 1		
📀 🗹 Light				0	ī
Туре	Directional				1
Color					1
Mode	Realtime				
Intensity	1				-
Indirect Multiplier	1				
Shadow Type	Soft Shadow	s			
Realtime Shadows					
Strength				1	
Resolution	Use Quality S	Settings			1
Bias	-0	223010-0-223		0.05	
Normal Bias				0.4	_
Near Plane	0			0.2	
Cookie	None (Text	ure)			4
Cookie Size	10				-
Draw Halo					
Flare	None (Flare	.)		1	(
Render Mode	Auto				
Culling Mask	Everything			_	

Main Camera							tatic *
Tag MainCamera		+ Laye	er (	Default			
▼↓ Transform							
Position	X 0		Y	1.56	Z	-5.04	
Rotation	X 0		Y	0	Z	0	
Scale	X 1		Y	1	Z	1	
🔻 🍩 🗹 Camera							0
Clear Flags	Solid	Color					
Background			_				- 1
Culling Mask	Every	thing	_				
Projection	Persp	ective			_		
Field of View			_		_	6	0
Clipping Planes	Near	0.3	_				
	Far	1000					
Viewport Rect	X 0		Y	0			
	W 1		н	1			
Depth	-1		_				-
Rendering Path	Use G	raphics Setti	ng	\$			
Target Texture	None	(Render T	ext	ture)			0
Occlusion Culling							
Allow HDR							
Allow MSAA							
Allow Dynamic Resolution							
Target Display	Displ	ay 1					
Flare Laver							
O Audio Listener							0
	Add	Componen			7		
		componen	-		4		

# Scene graph

A scene graph is a collection of nodes in a graph or tree structure.

In Unity all tree nodes have only a single parent but may have many children.

Operations applied to a parent are applied to all its child nodes.



# Components

*Game Objects* have *Components* to give it some behavior.

Many components already exist within Unity:

- Mesh Filter
- Mesh Renderer
- Rigidbody
- Collidiers
- VideoPlayer

But you will also need create your own. These are your scripts that inherit from *MonoBehaviour*.

rer 🔟 🗟 🌣	O Inspector	Services	<b>≧</b> •≡
	Stylize_L	ava	🔟 🕂 🔅
Blend Probes =	Shader 🛛	Standard (Specular s	etup) •
None (Transform)	Rendering Mod	e Opaque	+
None (Transform) 0	Main Maps		
, j	O Albedo	- 1	
	© Specular	<b>—</b> <i>I</i>	
Per Object Motion ;	Smoothne	ss —0——	0.258
	Source	Specular Alp	ha 🕴
ration of lightmaps for	📔 🖸 🛛 Normal Ma	ар	1
erer, please enable the	🛛 🖉 OHeight Ma	p	0.08
c property.	© Occlusion	-	01
	©Detail Mas	k	
ider 🔯 🗐 🖈	Emission		
A Edit Collider	Tiling	X 1	Y 1
	Oliset	X U	
	Secondary Ma	aps	
None (Physic Mate	©Detail Alb	edo x2	
	©Normal Ma	ар	1
20	Tiling	X 1	Y 1
0.5	Offset	X 0	YO
💽 류 🌣	UV Set		
1	Forward Rend	lering Options	
0	Specular Highlin	ghts 🗹	
0.05	Reflections		
	Advanced Op	tions	
	Enable GPU Ins	tancin	
None ‡	Double Sided G	ilobal 🗌	
Discrete ‡			
	Stylize Lava		
			۲
	rer IIII IIIIIIIIIIIIIIIIIIIIIIIIIIIIII	rer Inspector Stylize_L Blend Probes : Blend Probes : None (Transform) © On : Per Object Motion : Com : Per Object Motion : Com	rer lightmaps for Per Object Motion f Don time Per Object Motion f Con time Per Object M

# Scripts

- Many components already exist! But you will also need create your own. These are your scripts that inherit from *MonoBehaviour*.
- Public variables will show up in the Inspector. A variable that is a Component can also be modified by the inspector



🔻 📾 🗹 Rotate Y (Scr	ipt)	1	\$,
Script	RotateY		0
Degrees Per Second	15		

# Adding Components to Game Objects

- GameObject: <a href="http://docs.unity3d.com/ScriptReference/GameObject.html">http://docs.unity3d.com/ScriptReference/GameObject.html</a>
- MonoBehaviour: <u>http://docs.unity3d.com/ScriptReference/MonoBehaviour.html</u>
- Drag and Drop Script onto the GameObject in the "Inspector" or manually add it by going to:
  - Add Component > Scripts > YOUR\_SCRIPT\_NAME\_HERE

⊡ us	ing UnityEngine;
Lus	ing system.corrections;
Ξpu	<pre>blic class test_script : MonoBehaviour {</pre>
	<pre>// Use this for initialization</pre>
É	<pre>void Start () {</pre>
-	}
	<pre>// Update is called once per frame</pre>
É	<pre>void Update () {</pre>
	}
}	

O Insp	ector	2					<b>a</b> •≡
	GameO	bj	ect				Static 👻
Tag	Untagge	d	÷ L	.ay	er Defa	ılt	\$
	Missing						
¥1.	Transfo	rı	n			_	🔯 🌣,
Positio	n :	х	-0.0353	Y	0.74871	Ζ	-0.4188
Rotati	on	х	0	Y	0	Z	0
Scale	3	х	1	Y	1	z	1
• • •	Test_sc	ri	pt (Scri	pt)	)	_	🔯 🌣,
Script			(e)te	st	script		0
	,	Ac	dd Comp	or	hent	_	

#### Assets

An asset is any resource that will be used as part of an object's component

- Scenes
- "Prefabs"
- Scripts
- Textures
- Animations
- Models
- Particles
- Sprites
- Etc.



#### **Shading and Materials**

- Unity provides several built-in shaders
  - Unity Standard shader
  - Can also write your own shader
    - Shaders are written in Cg/HLSL and wrapped in ShaderLab
- Manual Shader Documentation: <u>http://docs.unity3d.com/Manual/ShadersOverview.html</u>
- Standard Shader Documentation: http://docs.unity3d.com/Manual/shader-StandardShader.html
- Materials Documentation: <u>http://docs.unity3d.com/Manual/Materials.html</u>



O Inspector Services		
🙈 Stylize_Lava		🔯 🖬 -
Shader Standard (Sp	ecular setup)	
Rendering Mode	Opaque	
Main Maps		
O Albedo	1	
o Specular	<b>—</b> 1	
Smoothness		0.258
Source	Specular Alpha	
🔄 🛛 Normal Map		1
🗃 OHeight Map		0.08
© Occlusion		0 1
o Detail Mask		
Emission		
Tiling	X 1 Y	1
Offset	X 0 Y	0
Secondary Maps		
ODetail Albedo x2		
ONormal Map		1
Tiling	X I Y	1
Offset	X 0 Y	0
UV Set	UVO	
Forward Rendering Opt	tions	
Specular Highlights		
Reflections		
Advanced Options		
Enable GPU Instancing		
Double Sided Global Illum	ination	
O Inspector Services		<b>≙</b> •
Unlit Hologram		<b>a</b> 3
Shader Unlit/Tutoria	l/Hologram	
Albedo Texture		
hibedo rextare		
Tiling X 1	Y 1	
Offset X 0	Y 0	Sele
Tint Color		
Transparency		0.5
Cutout Threshold	O	0.22
Distance		0.88
Amplitude		1
Speed		1.11
Amount		0.2
Render Queue	Fi	om Shader 🕴 3000
Double Sided Global Illum	ination	

#### **Importing Textures**

• Process is the same as importing an external object. This time, instead of selecting an \*.FBX or \*.OBJ, select a \*.PNG, \*.JPG, etc. You can also place the images inside the Assets folder manually.



# **Using Textures**

- Click on the object you imported in the scene hierarchy and expand the shader properties in the Inspector.
- Click and drag the imported texture onto the square next to "Albedo" and your object should now have a texture on it.

Rendering Mode	Opaque		
Main Maps			
⊙ Albedo	1		
© Metallic	0		- 0
Smoothness		·	0.5
© Normal Map			
© Height Map			
© Occlusion			
© Emission	1		0
© Detail Mask			
Tiling	× 1	Y 1	
Offset	X 0	Y 0	
Secondary Maps			
O Detail Albedo x	2		
© Normal Map			1
Tiling	X 1	Y 1	
Offset	X 0	Y 0	
UV Set	UVO		

lambert1		🚺 🌣,
Shader Sta	andard	•
Rendering Mode Main Maps O Albedo	Opaque 🧷	*
No. 1		

# Lighting

- Lighting Documentation: <u>http://docs.unity3d.com/Manual/Lighting.html</u>
- Global Illumination Documentation:

http://docs.unity3d.com/Manual/GlobalIllumination.html

- Lighting is accomplished with the "Light" component.
  - Directional
  - Point
  - o Spot
  - Area (baked only)

ameObject Component Mobile Input Window Help			🔻 👩 🗹 Light	📵 🕸 🔅	
Create Empty	Ctrl+Shift+N		Туре	Directional	;
Create Empty Child	Alt+Shift+N	Scene C Game	Color		A
2D Object	Ś	10e0 20 % 40	Mode	Mixed	•
Light	>	Directional Light	Intensity	1	
Audio	>	Point Light	Indirect Multiplier	1	
UI	>	Spotlight	Shadow Type	Soft Shadows	:
Particle System		Area Light	Baked Shadow Angle	0	0
Camera		Reflection Probe	Realtime Shadows		
Center On Children		Light Probe Group	Strength		01
Make Parent			Resolution	Use Quality Settings	+
Clear Parent			Bias	-0	0.05
Apply Changes To Prefab			Normal Bias		0.4
Break Prefab Instance			Near Plane	0	0.2
Cat as first sibling	Chile -		Cookie	None (Texture)	0
Sec as hist sibling	Ctri+=	10-1-5	Cookie Size	10	
Set as last sibling	Cur+-	Dector	Draw Halo		
Move to View	Ctrl+Alt+F		Flare	None (Flare)	0
Align With View	Ctrl+Shift+F		Render Mode	Auto	•
Align View to Selected			Culling Mask	Everything	;
Toggle Active State	Alt+Shift+A				

# Scripting in Unity

Scripting in Unity is done in C#

Scripts are an example of a component that is associated with a game object

The skeletal structure of a typical script called *MyGameObject* is shown below:

```
using UnityEngine; // basic Unity-Engine objects
using System.Collections; // basic structures (ArrayList, HashTable,...)
public class MyGameObject : MonoBehaviour {
  void Start () {
    // ... initializations (like a constructor in Java)
  }
  void Update () {
    // ... insert code to be repeated every update cycle
  }
}
```

Script example

# Fundamental Classes: MonoBehavior

When you create a script in Unity, Unity creates a class that extends MonoBehaviour.

Contains functions and events that are available to standard scripts attached to Game Objects

- Awake, Start, Update, FixedUpdate
- OnCollisionEnter, OnCollisionStay, OnCollisionExit
- GetComponent, SendMessage, BroadcastMessage
- Destroy, Instantiate

For a full list of methods and documentation, see:

https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

# Fundamental Classes: GameObject

GameObject: A generic type from which all game objects are derived. This corresponds to anything that could be placed in your scene hierarchy.

GameObjects have an associated *name* and *tag*. You can find other gameObjects with *Find*, *FindWithTag*, *FindGameObjectsWithTag*, etc.

Here is an example of how to obtain the main camera reference by its name:

```
GameObject camera = GameObject.Find ( "Main Camera");
```

Suppose that we assign the tag "Player" with the player object and "Enemy" with the various enemy objects. We could access the object(s) through the tag using the following commands:

```
GameObject player = GameObject.FindWithTag( "Player");
GameObject [] enemies = GameObject.FindGameObjectsWithTag( "Enemy");
```

# Fundamental Classes: Transform

Transform: Every *game object* in Unity is associated with an object called its transform.

This object stores the position, rotation, and scale of the object. You can use the transform object to query the object's current position (*transform.position*) and rotation (*transform.eulerAngles*)

#### Vector3

Structure in Unity for representing 3D vectors and points.

This structure is used throughout Unity to pass 3D positions and directions around. It also contains functions for doing common vector operations.

Other classes can be used to manipulate vectors and points as well. For example the Quaternion and the Matrix4x4 classes are useful for rotating or transforming vectors and points.

Common methods: Cross, Dot, Normalize, Lerp, Reflect, Distance

For more information, see the documentation: <a href="https://docs.unity3d.com/ScriptReference/Vector3.html">https://docs.unity3d.com/ScriptReference/Vector3.html</a>

#### Quaternion

Quaternions are used internally by Unity to represent rotations.

There are some advantages to using quaternions over euler angles (gimbal lock, can be interpolated easily, etc)

Have x,y,z,w components and are non-commutative. Likely will never need to modify these components individually.

Instead use these to create/manipulate Quaternions: Quaternion.LookRotation, Quaternion.Angle, Quaternion.Euler, Quaternion.Slerp, Quaternion.FromToRotation, Quaternion.identity

For more information, see: <u>https://docs.unity3d.com/ScriptReference/Quaternion.html</u>

#### Matrix4x4

Structure for a 4x4 transformation matrix

Can perform translation, rotation, scale, shear, and perspective transformations using homogeneous transformations.

Column major: for the expression *mat[a, b]*, *a* refers to the row index, while *b* refers to the column index

In Unity, Matrix4x4 is used by several Transform, Camera, Material and GL functions.

Common methods/properties: determinant, inverse, transpose, LookAt, Ortho, Perspective, Rotate, Scale, Translate, TRS

For more information, see: <u>https://docs.unity3d.com/ScriptReference/Matrix4x4.html</u>

# Accessing Components:

It is often desirable to modify the values of components at run time.

Unity defines class types for each of the possible components, and you can access and modify this information from within a script.

To access public variables/methods from a component, use *GetComponent*.

Example:

```
// Get rigidbody component of this game object
Rigidbody rb = GetComponent <Rigidbody>();
```

```
// change this body's mass
rb.mass = 10f;
```

#### Accessing Members of Other Scripts

Often, game objects need to access members variables in other game objects.

Can use *GetComponent* to access public variables/methods in other scripts.

```
public class PlayerController : MonoBehaviour {
    public void DecreaseHealth () { ... } // decrease player 's health
}
public class EnemyController : MonoBehaviour {
    public GameObject player; // the player object
    void Start () {
        GameObject player = GameObject.Find( "Player");
    }
    void Attack () { // inflict health loss on player
        player.GetComponent<PlayerController>().DecreaseHealth();
}
```

# Colliders and Triggers:

Some events are generated by the user (e.g., input), some occur at regular time intervals (e.g.,Update()), and finally others are generated within the game itself.

Typically, colliders are physical objects that should not overlap, whereas triggers are invisible barriers that send a signal when crossed.

There are various event functions for detecting when an object enters, stays within, or exits, collider/trigger region. These include, for example:

- For colliders: void OnCollisionEnter(),void OnCollisionStay(),void OnCollisionExit()
- For triggers: void OnTriggerEnter(),void OnTriggerStay(),void OnTriggerExit()

#### Example: Rotate script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class RotateYFinal : MonoBehaviour
      // Degrees to rotate around Y-Axis per second
      public float rotationRate = 5.0f;
      // Update is called once per frame
      void Update()
            // Define the axis of rotation
            Vector3 axis = new Vector3(0, 1, 0);
            // Equivalently you could use Vector3.up
            // Calculate the amount to rotate the object this frame
            float amountToRotate = rotationRate * Time.deltaTime;
            // Access the transform component of this object and rotate
            this.transform.Rotate(axis, amountToRotate);
```

# Raycasting:

```
public static bool Raycast(
     Vector3 origin,
    Vector3 direction,
     out RaycastHit hitInfo,
     float maxDistance,
     int layerMask,
     QueryTriggerInteraction queryTriggerInteraction
```



);

Casts a ray, from point *origin*, in direction *direction*, of length *maxDistance*, against all colliders in the Scene.

You may optionally provide a LayerMask, to filter out any Colliders you aren't interested in generating collisions with.

Documentation: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html

# **Oculus Utilities for Unity**

OVRCameraRig is a Component that controls stereo rendering and head tracking. It maintains three child "anchor" Transforms at the poses of the left and right eyes, as well as a virtual "center" eye that is halfway between them.

This Component is the main interface between Unity and the cameras. It is attached to a prefab that makes it easy to add comfortable VR support to a scene.

Public Members:

1. Updated Anchors - Allows clients to filter the poses set by tracking. Used to modify or ignore positional tracking.

Game Object Structure:

1. TrackingSpace - A Game Object that defines the reference frame used by tracking. You can move this relative to the OVRCameraRig for use cases in which the rig needs to respond to tracker input. For example, OVRPlayerController changes the position and rotation of TrackingSpace to make the character controller follow the yaw of the current head pose.

# **OVRInput**

