Introduction to Unity: Roll-A-Ball Basic Interactivity



- 1) Starting with the Unity Launcher:
 - a. Creating a new project: New \rightarrow



- i. Project name:
 - <u>Note</u> Name it LastNameFirstName_RollABall
- ii. Location: click on the dots to navigate where you want your project to live
- iii. 3D vs. 2D
 - Pick 3D for now
- iv. Hit Create Project

Create a new project with the second seco	ith Unity 2019.3.0a6			-		×
Templates			Settings			
©		◎	Project Name * DunsireNikki_RollABall			
2D	3D	3D With Extras	Location * C:\Users\ndunsir5.AD\Desktop			
Preview 3	۞					
High-Definition RP	Lightweight RP					
			CANCEL	CF	REATE	

- 2) Once Unity is open:
 - b) File \rightarrow Save scene as \rightarrow
 - Double click on the folder called it 'Scenes' to open it up i)
 - ii) Type a name for your scene file: "01 RollABall"
 - iii) Hit Save
 - c) Delete the "SampleScene"
- 3) Create a primitive plane. We can do this in two ways:





OR

c) This should place a plane at the Origin for us. If it's not at the origin, reset its value in the Inspector



Rename the plane to 'Ground' d)

Scale the plane up to 2, 1, 2 e)

						-
Scale	Х	2	Υ	1	Ζ	2

Note – you can't scale up in Y b/c this object is single sided. However, if you gave it a negative number, it i) would flip the plane the other direction and you wouldn't see it unless you rotated your camera underneath it. Making the non-renderable side transparent is called Backface Culling.

- 4) Create a Sphere (in the same why you created the plane)
 - b) Name it 'Player'
 - c) Make sure it's at the Origin
 - d) Move it up in Y: 0.5

Position X 0 Y 0.5 Z 0

i) <u>Note</u> – all objects in Unity take have default size of 1, 1, 1 or 1,2,1. If the Sphere is sitting at the Origin like the plane is and it's sitting half way through it, moving it up by half will rest it perfectly on top.



- 5) Let's add a material to the Ground
 - b) In the Project Tab \rightarrow Rt mouse click over the Assets folder \rightarrow Create \rightarrow Folder



- c) Rename this folder 'Materials'
- d) Highlight the Materials folder \rightarrow Rt mouse click \rightarrow Create \rightarrow Material
 - i) Name the new Material: 'Ground_mat'
 - ii) Note we should now see a new Material node in the Materials folder

e) In the Inspector \rightarrow Change the Albedo color to Blue (or a color of your choice) by clicking on the swatch

Inspector	🔀 Navigation	Services	∂ •≡	Color	•
Backgrou Shader (und Standard			rightness	Hite
Rendering Mod	e Opaque		\$		Saturation
Main Maps				R G	54
⊙ Albedo				A Hex Color	255 r # 363A65FF
⊙ Metallic	0		0	Presets Click to a	-= add new preset

f) From the Materials folder, drag the Ground material to the plane in the scene.



Player Control

- 1) Let's now setup the player behavior:
 - a) Setting up Contact:
 - i) Select the Sphere
 - ii) In the Inspector ightarrow Add Component
 - \Rightarrow Physics \rightarrow Rigidbody
 - ⇒ <u>Note</u> this will allow the Sphere to behavior properly when it bumps into things...i.e. it won't move through other objects.

OR

iii) <u>Note</u> – Because we created the primitive sphere in Unity, notice that it has a Sphere Collider already applied to it. This matters when we're working with objects that will interact with other objects...which we'll discuss more later. For now, make a mental note of it.

b) Setting up Movement:

i) <u>Note</u> – to setup movement via our keyboard that drives the sphere we'll need to use a script.

ii) Create a new folder within Assets called 'Scripts'

 \Rightarrow To create a new script, we can do this in one of three ways:

ssels		ł	נטטו נ מט וטו	now
ile Edit Ass	ets GameObject Component Wind	ow	Help	
(*)	Create	•	Folder	1
Hierard	Show in Explorer		C# Script	
Create *	Open		Invarceint	-
🚭 RollA	Delete		Shadaa	
Main C			Snader	
Groun	Open Scene Additive		Testing	'
Player	Import New Asset		Scene	
	Import Package	•	Prefab	
	Export Package		Audia Mina	
	Find References In Scene		Audio Mixer	
	Select Dependencies		Material	
			Lens Flare	
	Refresh Ctrl+R		Render Texture	
	Reimport		Lightmap Parameters	
	Reimport All		Sprites	
	Run API Updater		Animator Controller	
	Open C# Project		Animation	

 Project Tab → Create (Over the Scripts Folder) → C# Script [Do this]

Project	Console		
Create *		(9	
🔻 😭 Favorites	Assets > Scripts		
All Materi All Model: All Prefab All Conflic All Conflic All Conflic Scenes Scripts	ols s sd tred Create	This folder	' is empty
	Show in Explorer	C# Script	
	Open	lavascrint	
	Delete	Shader	•
	Open Scene Additive	Testing	•

- Rename the Script to 'PlayerController'
- Drag the script onto the Sphere in the Scene
- Open up the Script. We can do this in a couple of different ways:
 - (i) Double click on it in the Scripts folder **OR**
 - (ii) Click on the Settings Gear of the script in the Inspector once it's added to the sphere

🔻 健 🗹 Player Controller (Script)	D 🖏	
Script 💽 PlayerController		Reset
Default-Material Shader Standard		Remove Component
Add Component]	Move Down
	,	Copy Component Paste Component As New
		Paste Component Values
		Edit Script

- (iii) <u>Note</u> this should open up Visual Studio which is where we're going to edit the script. If you're on a Mac, it might open MonoDevelop. Either is fine. Visual Studio won't compile on a Mac OS, as its PC based.
 - 1. For now, we are not going into the ins and outs of the code, we simply want to explore how this process works and how it's applied.
- (iv) In the Code Editor:
 - 1. Remove the text that's in the document and replace it with:

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
public class PlayerController : MonoBehaviour
{
   public float speed;
   private Rigidbody rb;
   void Start()
   {
        rb = GetComponent<Rigidbody>();
   }
   void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
        rb.AddForce(movement * speed);
   }
}
```



- 2. Save the code file
- Play test!
 - (i) In the Inspector \rightarrow change the Speed value to something other than 0

▼ 🕒 🔽	Player Controller (Script)	💽 🌣,
Scrip	PlayerController	0
Spee	d 3	
	Default-Material	🔯 🌣,
	Shader Standard	

(ii) Hit Play on the player controls



(iii) Use W,A,S,D to control the movement of the Sphere

Camera Setup

1) Position the Camera so that it's up above and angled down toward the player

🖊 Transform			💽 🕂 🐥
Position	X O	Y 10	Z -10
Rotation	X 45	Y 0	Z 0
Scale	X 1	Y 1	Z 1

<u>Note</u> – how we setup the camera in relationship to the player will matter. We want them pointing forward down the same axis. So, if Z is the forward axis of the camera, we want Z to be forward on the sphere as well. Otherwise, when we apply our script, our controls may end up backwards.

- 2) We want the camera to move relative to the Player. One thing we could do (but won't) is:
 - a) Hierarchy \rightarrow Drag the Main Camera object over the Player Object to Parent them.

	-
ⁱ ≔ Hierarchy	∂
Create * Q*All	
🔻 🚭 02_RollABall_PlayerSetupComplete*	*≡
Directional Light	
Ground	
▼ Player	
Main Camera	

- Note So right now the camera will follow the object as it moves. This is a typical 3rd person setup. The only problem with this however, is that as soon as the player starts rotating (which it's going to do on all three axis) the camera will do the same. So, we can't do it this way, we'll need to use a script.
 - \Rightarrow Drag the Main Camera back out of the Player in the Hierarchy to unparent them.
- ii) Select the camera
 - \Rightarrow Inspector \rightarrow
 - Add Component \rightarrow New Script \rightarrow

Add Compone	ent	Add Cor	mponent
۹.		Q	
Component	:	✓ New	Script
Navigation	+	Name	
Audio	>	NewBehaviourScri	pt
Video	►	Language	C Sharp
Rendering	► 🗋		
Layout			
Miscellaneous			
Analytics			
Scripts			
Event			
Network	•		
UI	•		
AR	- U		
New Script		Create	and Add

- Rename the script 'CameraController'
- Hit CreateAndAdd

 \Rightarrow Find the script in the Assets folder and move it into the Scripts folder

- \Rightarrow Open the Script Up in Visual Studio
 - Remove the text that's in:
 - Replace it with:

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
public class CameraController : MonoBehaviour
{
    public GameObject player;
    private Vector3 offset;
    void Start()
    {
        offset = transform.position - player.transform.position;
    }
    void LateUpdate()
    {
        transform.position = player.transform.position + offset;
    }
}
```

It will look like this:



• Save the script and go back to Unity

- iii) Now we need to connect the Player to the Camera so that our Script will work:
 - \Rightarrow Select the Main Camera
 - \Rightarrow Go to the Inspector
 - ⇒ In the Camera Controller (Script) drop down \rightarrow Drag the Player object from the Hierarchy to the Player slot of the Script to make the connection.



- \Rightarrow Play test!
- iv) <u>Note</u> for added functionality, try using the camera scripts that we used in our third person setup of the basic unity scene, instead of the one above!

Play Area Setup

- 1) Create an Empty Game Object
 - a) GameObject → CreateEmpty

C Unity 5.6.111 Personal (64bit) - 02_RollABall_PlayerSetupComplete.unity - Roll							
File Edit Assets	GameObject Component	Window Help					
🕅 🕂 🕄	Create Empty	Ctrl+Shift+N					
'≔ Hierarchy	Create Empty Child	Alt+Shift+N	_				
Create * Q*All	3D Object	+					

- b) <u>Note</u> we can use Empty Game Objects to organize and group things in the Hierarchy.
- c) Name this GameObject 'Walls'
- d) Verify that is sitting at the Origin. If it isn't, reset it so that it is.

e) Create a Cube

🚭 Unity 5.6.1f1 Perse	onal (64bit) - 02_RollABall_Pla	yerSetupComplete.unity	- RollABall - PC, Mac & Linu
File Edit Assets	GameObject Component	Window Help	
🖑 🕂 S	Create Empty	Ctrl+Shift+N	1
[™] Hierarchy	Create Empty Child	Alt+Shift+N	E Game
Create * Q*All	3D Object	•	Cube
▼ 🚭 02_RollABall	2D Object	•	Sphere
Directional Ligi Ground	Light	+	Capsule
Player	Audio		Cylinder

- f) Rename it 'WallWest'
- g) Verify that it is also at the Origin. If it isn't, reset it so that it is.

h) In the Hierarchy:

i) Drag WallWest into the Walls group to parent the cube to the empty game object we created.

'≡ Hierarchy	<u> </u>
Create * Q*All	
🔻 🚭 RollABall_Functionality*	+≡
🔘 Directional Light	
🔘 Ground	
🔘 Player	
问 Main Camera	
🔻 问 Walls	
🤍 WallWest	

ii) Adjust its Transforms:

🔻 🙏 🛛 Transform			🔯 🕸 🔅
Position	X -10	Y 1	Z 0
Rotation	X O	Y 0	Z 0
Scale	X 0.5	Y 2	Z 20.5

i) Duplicate the wall by hitting **Cntrl d**

- i) Rename it to WallEast
- ii) Adjust its Transforms:

🔻 🙏 🛛 Transform			💽 🕂 🐥
Position	X 10	Y 1	Z 0
Rotation	X O	Y O	Z 0
Scale	X 0.5	Y 2	Z 20.5

j) Duplicate the wall

- i) Rename it to WallNorth
- ii) Adjust its Transforms:

▼ 🙏 Transform			🚺 🐺 🐥
Position	X O	Y 1	Z 10
Rotation	X O	Y 90	Z 0
Scale	X 0.5	Y 2	Z 20.5

k) Duplicate the wall

- i) Rename it WallSouth
- ii) Adjust its Transforms:

🔻 🙏 🛛 Transform			🛐 🕂 🌣
Position	X O	Y 1	Z -10
Rotation	X O	Y 90	Z 0
Scale	X 0.5	Y 2	Z 20.5

I) Play Test!

Collectables Setup

- 1) Create a new cube
 - a) Reset it to the Origin if it isn't already there.
 - b) <u>Note</u> we can hide the Player for now if we want by clicking on checkbox next to the name of the object in the Inspector (this allows us to make an object active or not).



- c) Rename the Cube to 'PickUp'
- d) Adjust its Transforms:

· · · · · · · ·		
Inspector	🕻 Navigation	≧ *≡
PickUp		🗌 Static 🔻
Tag Untagged	Layer Defaul	t ‡
▼ 🙏 🛛 Transform		🛐 큐 🌣
Position	X 0 Y 0.5	Z 0
Rotation	X 45 Y 45	Z 45
Scale	X 0.5 Y 0.5	Z 0.5

- 2) Let's have the PickUp rotate as it sits there. To do this we'll need another script:
 - a) Select the PickUp
 - b) Inspector \rightarrow Add Component \rightarrow New Script
 - i) Name this Script 'Rotator'
 - ii) Click CreateAndAdd
 - c) Place the new script in the Scripts folder
 - d) Open the new Script in VS
 - i) Remove the default code as we did before
 - ii) Replace it with the following and then save the file:



- 4) Now we'd like to place duplicates of this cube around the play area as our pickup objects!
 - a. <u>Note</u> before we can do this however, we need to create what's called a **Prefrab**. A Prefab is like a template that contains a blueprint for a game object or game object family. In other words, it's kind of like creating an instanced object with set parameters. When we create a prefab, we can drag and drop the prefab into the scene. When we make changes to a prefab object, it will update all of the other duplicates.
 - a) Create a Prefab folder in our Project structure
 - b) Drag the PickUp object from our Hierarchy to the Prefabs folder



- c) <u>Note</u> anytime we drag something from the Hierarchy to the Project structure like this we create a new Prefab asset containing a blueprint of our game object.
- d) Let's organize our Hierarchy prior to creating our duplicates!
 - i) Create a new Empty Game Object and name it 'PickUps'
 - ii) Reset it to the Origin if it's not currently located there.
 - iii) In the Hierarchy \rightarrow Drag the first PickUp Object into the new PickUps group.

e) In the Top down view, move the first PickUp where you want it on the board and then duplicate it. Move the duplicates to your desired location around the board. You may need to adjust your pivot from Local to Global.

= Hierarchy	🔒 📲 🔤 🗰 Scene	S Game			*=
Create * Q*All	Shaded	* 2D 🔆 ≪	i) 🖬 🔹 Gizmos 🖜	Q*All	
V Content C	sComplete* -=				
Directional Light					<u></u>
					N N
Player				-	ж
Main Camera					
▶ Walls					
▼ PickUps					= ^T op
PickUp					
PickUp (1)					
PickUp (2)					
PickUp (3)					
PickUp (4)					
PickUp (5)					
PickUp (6)					
PickUp (7)					н
		-			c
		-			
		1		1	
			U		
			CS		

- f) Play Test!
- 5) Now let's change the color of the cubes (we will need a new material for this):
 - a) Select our current material in the Materials folder
 - b) Duplicate it (Cntrl d)
 - c) Rename it 'PickUp_Yellow'
 - d) In the Inspector \rightarrow Change the Albedo color to Yellow (or any other color that you want)
 - e) Let's apply this to our Prefab.
 - i) Drag the Material to one of the PickUp cubes
 - ii) Click on the cube
 - iii) In the Inspector \rightarrow
 - \Rightarrow Notice there's now a Prefab line item
 - \Rightarrow Click on the Overrides drop down
 - \Rightarrow Apply All

🟮 Inspector 🛛 🔀 Navig	ation	∂ *≡
PickUp (6)		🗌 Static 🔻
Tag Untagged	t Layer Def	ault +
Prefab Open	Select	Overrides 🔹
V L Transform	Overrides to 🥃 PickUp in 🚭 Scene	
Rotation	V 🜍 PickUp (6)	
Scale	Iransform Inderer	
Cube (Mesh Filter]	
🔻 🛃 🗹 Mesh Renderer	Click on individual item	ns to review, revert and apply.
Materials	Revert All	Apply All

- iv) Note this will apply the material to all of the cubes
- f) Let's make these a little more fun!
 - i) Click on the PickUp_Yellow Material
 - ii) In the Inspector \rightarrow Rendering Mode \rightarrow change to Transparent

0 Inspector 🔀 Navigation	Services	à •≡
Ground_Yellow		🔯 *.
Shader Standard		•
Rendering Mode	Transparent	+)
Main Maps		
⊙Albedo	len de la constante de la const	
© Metallic	0	0
0 11		0

- iii) Click on one of the PickUp cubes
- iv) In the Inspector \rightarrow Add Component \rightarrow Effects \rightarrow Halo
 - \Rightarrow Size: Adjust the size of the glow you want
 - \Rightarrow Color: Adjust the color of the glow

🔻 😳 🗹 Halo				\$,
Color				P
Size	0.59			
PickUp_Yellow		1	큐	\$,
▶ Shader Standard			_	•

\Rightarrow Hit Apply to update the Prefab





Collectables Picked Up

- 1) Now we need to tell the Player what to collide with and what happens when the collision happens.
- 2) Make the Player active again (if you hid it before)



3) Open up the PlayerController Script in VS

a) Add the following code right before the very last } and save:

```
void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Pick Up"))
    {
        other.gameObject.SetActive(false);
    }
}
```



b) In the Project Structure \rightarrow Prefab folder \rightarrow select the Prefab PickUp object



<u>Note</u> – in the code we're referencing a tag that we need to create. This tag will only be associated with our PickUps. This is how our Player will know to only pick up the PickUp objects and not things like the floor and walls. The naming convention that we used in the script needs to be consistent with what we create here. "Pick Up". It is space and case sensitive!!



- c) Click on the + button to create the new tag
- d) Name it "Pick Up"
- e) Hit Save
- f) Click back on the PickUp Prefab
- g) Inspector \rightarrow Tag \rightarrow Change to Pick Up



- 4) Play Test!
- 5) <u>Note</u> You probably noticed that our sphere is still bouncing off our cubes. This is because our cubes are still using colliders and we need them to be Triggers so that our Sphere can occupy the same space with each cube it runs into them and essentially absorb them (or make them inactive upon contact) as per our code. To do this:
 - a) Select the PickUp Prefab
 - b) In the Inspector \rightarrow Box Collider
 - i) Is Trigger: Check ON

🔻 🤪 🗹 Вох Collider		\$,
Is Trigger		
Material	None (Physic Material)	0
Center	X 0 Y 0 Z 0	
Size	X 1 Y 1 Z 1	

ii) Play Test!

- 6) Note right now everything appears to be working, but there is one issue and it has to do with performance and resource depletion based on how Unity optimizes its Physics. As a performance optimization, Unity calculates all the volumes of all the static colliders in the scene and holds this information in cache. This makes sense as static collider shouldn't move and this saves recalculating this information every frame. The problem with our scene is that our cubes are rotating. Anytime we move, rotate or scale a static collider, Unity will recalculate all the static colliders again and update the static collider cache. In doing this we're tapping our system's resources. As an alternative we can move, scale and rotate Dynamic colliders as often as we want and Unity won't re-cache any collider volumes. Unity expects us to move colliders, but we need to indicate which ones are dynamic. We do this by using the RigidBody component. Any game object with a collider and a Rigidbody is considered dynamic. Any game object with just a collider and NO Rigidbody is considered Static. Right now, our cubes don't have a Rigidbody so they're considered Static. So, Unity it is recalculating our Static collider cache every frame. So, we need to add a Rigidbody to the PickUp object Prefrab.
 - a) Select the Prefab
 - b) Inspector \rightarrow Add Component \rightarrow Physics \rightarrow Rigidbody

🔻 🙏 🛛 Rigidbody	🔟 🔅,
Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	
Is Kinematic	
Interpolate	None +
Collision Detection	Discrete +
▶ Constraints	
Ground_Yell	ow 💿 🖏
Shader Standard	
Ac	dd Component

- c) <u>Note</u> Now our Cubes are dynamic. However, if we hit play they'll just fall through the floor because 'Use Gravity' is checked on and b/c we changed the cubes to Triggers, they no longer collide with the floor. We could just uncheck 'Use Gravity', but although they would no longer respond to gravity, they would respond to physics forces. So rather than checking off Gravity:
 - i) Is Kinematic: Check ON
 - ⇒ Now our Cubes will not react to physics forces and they can be animated and moved via their transforms. This is great for things that have colliders like elevators and moving platforms as well as objects with Triggers like our cubes, that need to animate or move by their transforms.

🔻 🙏 🛛 Rigidbody	🔯 🌣,		
Mass	1		
Drag	0		
Angular Drag	0.05		
Use Gravity			
Is Kinematic			
Interpolate	None +		
Collision Detection	Discrete +		
▶ Constraints			
Ground_Yell	ow 🛐 🖏		
Shader Standard			
Add Component			

- \Rightarrow Play Test!
 - Now things are working and performing the way they should

- d) To Recap:
 - i) Static Colliders shouldn't move (walls, floors, etc.)
 - ii) Dynamic Colliders can move and should have a RigidBody component along with their collider
 - iii) Standard RigidBodies are moved using Physics forces
 - iv) Kinematic RigidBodies are moved using their transforms

Keeping Score and UI

- 1) <u>Note</u> We need something to store the counted PickUps and then add to that stored number. We can do this by modifying the Player script. We also need something to display the count and let us know when we've 'won'.
- 2) Create a new UI text element (this will allow us to display text in our scene)
 - a) GameObject \rightarrow UI \rightarrow Text

Create Empty Ctrl+Shift+N Create Empty Child Alt+Shift+N 3D Object * 2D Object * Light * Audio * Video * UI * Particle System Image Camera Button Center On Children Button Make Parent Sider Clear Parent Sider Apply Changes To Prefab Break Prefab Instance Set as first sibling Ctrl+- Kove To View Ctrl+- Alt+F Aign With View Ctrl+- Alt+F Aign With View Ctrl+- Shift+F Aign Witew to Selected Toggle Active State	Gam	eObject Component Windo	ow Help		
Create Empty Child Alt+ Shift+ N 3D Object * 2D Object * 2D Object * Light * Audio * Video * UI * Particle System Image Camera Image Center On Children Button Make Parent Scrollbar Clear Parent Scrollbar Apply Changes To Prefab Break Prefab Instance Set as first sibling Ctrl+- Aving With View Ctrl+Alt+F Align With View Ctrl+Alt+F Align With View Ctrl+Shift+A		Create Empty	Ctrl+Shift+N	- 11	
3D Object haded 2D 2D Object haded 2D Light haded 2D Audio haded 2D Video haded Image Video Image Image Camera Button Toggle Center On Children Button Toggle Make Parent Sider Scrollbar Clear Parent Dropdown Input Field Set as first sibling Ctrl+= Scrollbar Set as first sibling Ctrl+= Canvas Set as first sibling Ctrl+= Canvas Aign With View Ctrl+shift+F Aign With View Scroll View Aign With View Ctrl+shift+F Event System Event System		Create Empty Child	Alt+Shift+N	# Sr	cene Game
2D Object • Light • Audio • Video • UI • Particle System Image Camera Raw Image Center On Children Button Make Parent Sider Clear Parent Sider Apply Changes To Prefab Break Prefab Instance Break Prefab Instance Dropdown Input Field Canvas Set as first sibling Ctrl+= Set as first sibling Ctrl+= Aign With View Ctrl+Alt+F Aign With View Ctrl+shift+F Aign With View Ctrl+Shift+A		3D Object		▶ Shac	led * 2D
Light > Audio > Video > UI > Particle System Image Camera Raw Image Center On Children Button Make Parent Stider Clear Parent Scrollbar Apply Changes To Prefab Dropdown Break Prefab Instance Dropdown Move To View Ctrl+= Atign With View Ctrl+shift+F Align View to Selected Event System		2D Object		•	
Audio Image Video Text UI Image Particle System Raw Image Camera Button Center On Children Button Make Parent Slider Clear Parent Sider Apply Changes To Prefab Dropdown Break Prefab Instance Dropdown Set as first sibling Ctrl+= Set as first sibling Ctrl+= Set as last sibling Ctrl+= Aign With View Ctrl+Alt+F Align With View Ctrl+Shift+F Align With View Ctrl+Shift+F Aign With View Ctrl+Shift+F Toggle Active State Alt+Shift+A		Light		•	
Video Image UI Image Particle System Image Camera Raw Image Center On Children Button Make Parent Slider Clear Parent Slider Apply Changes To Prefab Dropdown Break Prefab Instance Dropdown Set as first sibling Ctrl+= Set as last sibling Ctrl+= Aign With View Ctrl+Alt+F Align With View Ctrl+Shift+F Aign With View to Selected Event System		Audio		•	
UI Text Particle System Image Camera Raw Image Center On Children Button Make Parent Slider Clear Parent Slider Apply Changes To Prefab Dropdown Break Prefab Instance Dropdown Set as first sibling Ctrl+= Set as first sibling Ctrl+= Aign With View Ctrl+Alt+F Align With View to Selected Event System Toggle Active State Alt+Shift+A		Video		١.	
Particle System Image Camera Raw Image Center On Children Button Make Parent Stider Clear Parent Stider Apply Changes To Prefab Dropdown Break Prefab Instance Input Field Set as first sibling Ctrl+= Set as last sibling Ctrl+- Move To View Ctrl+Alt+F Align With View to Selected Event System Toggle Active State Alt+Shift+A		UI		•	Text
Camera Raw Image Center On Children Button Make Parent Toggle Clear Parent Slider Apply Changes To Prefab Dropdown Break Prefab Instance Input Field Set as first sibling Ctrl+= Set as last sibling Ctrl+Alt+F Aign Vith View Ctrl+Alt+F Align View to Selected Event System		Particle System			Image
Center On Children Button Make Parent Toggle Clear Parent Slider Apply Changes To Prefab Dropdown Break Prefab Instance Input Field Set as first sibling Ctrl+= Set as last sibling Ctrl+. Move To View Ctrl+Alt+F Arign View to Selected Scroll View Toggle Active State Alt+Shift+A		Camera			Raw Image
Make Parent Toggle Clear Parent Slider Apply Changes To Prefab Scrollbar Break Prefab Instance Dropdown Set as first sibling Ctrl+= Set as last sibling Ctrl+- Move To View Ctrl+Alt+F Align With View to Selected Scroll View Toggle Active State Alt+Shift+A		Center On Children			Button
Clear Parent Slider Apply Changes To Prefab Scrollbar Break Prefab Instance Dropdown Set as first sibling Ctrl+= Set as last sibling Ctrl+- Move To View Ctrl+Alt+F Align With View Ctrl+Shift+F Align View to Selected Event System Toggle Active State Alt+Shift+A		Make Parent			Toggle
Apply Changes To Prefab Scrollbar Break Prefab Instance Dropdown Set as first sibling Ctrl+= Set as last sibling Ctrl+- Move To View Ctrl+Alt+F Align With View Ctrl+Shift+F Align View to Selected Event System Toggle Active State Alt+Shift+A		Clear Parent			Slider
Break Prefab Instance Dropdown Set as first sibling Ctrl+= Set as last sibling Ctrl+- Move To View Ctrl+Alt+F Align With View Ctrl+Shift+F Align View to Selected Event System Toggle Active State Alt+Shift+A		Apply Changes To Prefab			Scrollbar
Set as first sibling Ctrl+= Set as last sibling Ctrl+- Move To View Ctrl+Alt+F Align With View Ctrl+Shift+F Align View to Selected Event System Toggle Active State Alt+Shift+A		Break Prefab Instance			Dropdown Input Field
Set as last sibling Ctrl+- Canvas Move To View Ctrl+Alt+F Panel Align With View Ctrl+Shift+F Scroll View Align View to Selected Event System Toggle Active State Alt+Shift+A		Set as first sibling	Ctrl+=		
Move To View Ctrl+Alt+F Panel Align With View Ctrl+Shift+F Scroll View Align View to Selected Event System Toggle Active State Alt+Shift+A		Set as last sibling	Ctrl+-		Canvas
Align With View Ctrl+Shift+F Align View to Selected Event System Toggle Active State Alt+Shift+A		Move To View	Ctrl+Alt+F		Panel
Align View to Selected Event System Toggle Active State Alt+Shift+A		Align With View	Ctrl+Shift+F		Scroll View
Toggle Active State Alt+Shift+A		Align View to Selected			Event System
		Toggle Active State	Alt+Shift+A		F

<u>Note</u> – in order for text to behave the way it's supposed to, it needs to be a child of a Canvas.



b) Rename the text object 'CountText'

c) Inspector \rightarrow

i) In the text box type: Count

▼ <mark>Т</mark> ∎ Text (Script) Text	[] \$,
Count	
Character	

ii) Color: change to white

Best Fit		
Color		III III
Material	None (Material)	0

iii) We want the text to appear in the upper left corner of our screen in play mode:

\Rightarrow Open Anchor presets



 $\Rightarrow~$ Hold down Shift Alt and click on the upper left corner option



- \Rightarrow To give it a little more room:
 - Pos X: 10
 - Pox Y: -10

- 3) Now let's create the Win text
 - a) GameObject \rightarrow UI \rightarrow Text
 - b) Note this will place the new text in our current canvas
 - c) Rename it 'WinText'
 - d) Inspector \rightarrow
 - i) Color: white
 - ii) Font size: 24
 - iii) Text: You Win!!
 - iv) Open Anchor Presets



- \Rightarrow Hold down Shift and Alt
- \Rightarrow Select the center option
- \Rightarrow Change the Pos Y: 75

custom		Pos X	Pos	Y	Pos Z
5		0	0		0
		Width	Hei	ght	
		160	30		
🕨 Anchor P	resets				
Shift: Also	set pivo	t Alt:	Also set	position	
	left	center	right	stretch	
					7 1
-		<u> </u>			¥
					<u>i</u> i i i i i i i i i i i i i i i i i i
- ā 🗆					
a					
PP				• • • •	
Ξ Ξ					
8					0
		Щ			+
5					
tret	1			•	
in	-		L		1

- 4) Now let's adjust our PlayerController script to keep count of our objects and display them via our text:
 - a) Open up the PlayerController script in VS
 - b) Add the flowing lines of code (highlighted in yellow):

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
public class PlayerController : MonoBehaviour
{
    public float speed;
    public Text countText;
    public Text winText;
    private Rigidbody rb;
    private int count;
    void Start()
    {
        rb = GetComponent<Rigidbody>();
        count = 0;
        SetCountText();
        winText.text = "";
    }
    void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
        rb.AddForce(movement * speed);
    }
    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Pick Up"))
        {
            other.gameObject.SetActive(false);
            count = count + 1;
            SetCountText();
        }
    }
    void SetCountText()
    {
        countText.text = "Count: " + count.ToString();
        if (count >= 12)
        {
           winText.text = "You Win!";
        }
    }
}
```

- c) Save the script.
- d) Note if (count >= 8): you will need to change this value depending on how many pickup objects you have in your scene in order for the win screen to pop up!

- 5) Associate the text to Player (sphere)
 - a) Click on Player in Hierarchy
 - b) Inspector \rightarrow
 - i) Drag the CountText object to the CountText variable of the Player
 - ii) Drag the WinText object to the WinText variable to the Player



iii) Play test!

Building The Game

1) File \rightarrow Build Settings

Build Settings					
Scenes In Build					
✓ Scenes/01_RollABall			0		
			Add Open Scenes		
Platform					
PC, Mac & Linux Standalone	PC, Mac & Li	nux Standalone			
Universal Windows Platform	Target Platform	Window	5 💌		
tuos tuos	Architecture	x86	•		
1005 1005	Server Build				
iOS ios	Copy PDB files Create Visual Studio	Solution			
	Development Build				
	Autoconnect Profiler				
Android	Script Debugging				
WebGL	Scripts Only Build				
	Compression Metho	d Default	•		
Facebook		2 C. dure			
		Learn abo	ut Unity Cloud Build		
Player Settings		Build	Build And Run		

- a) Hit: Add Open Scene
- b) PC, Mac & Linux Standalone
- c) Target Platform: Windows OR Mac OS X
- d) Architecture: x86_64
- e) Click Build
 - i) Create a new folder within your project structure called 'Builds'
 - ii) Open this folder
 - iii) Name your .exe: "RollABall"
 - iv) Hit Save
 - v) Note To play your game open the .exe
 - vi) <u>Note</u> Do not delete the folders that gets created along with the .exe. The .exe needs that info to run your game.