Zombie Shooter Project 1c

AINT155

Task 1. A script to move the player

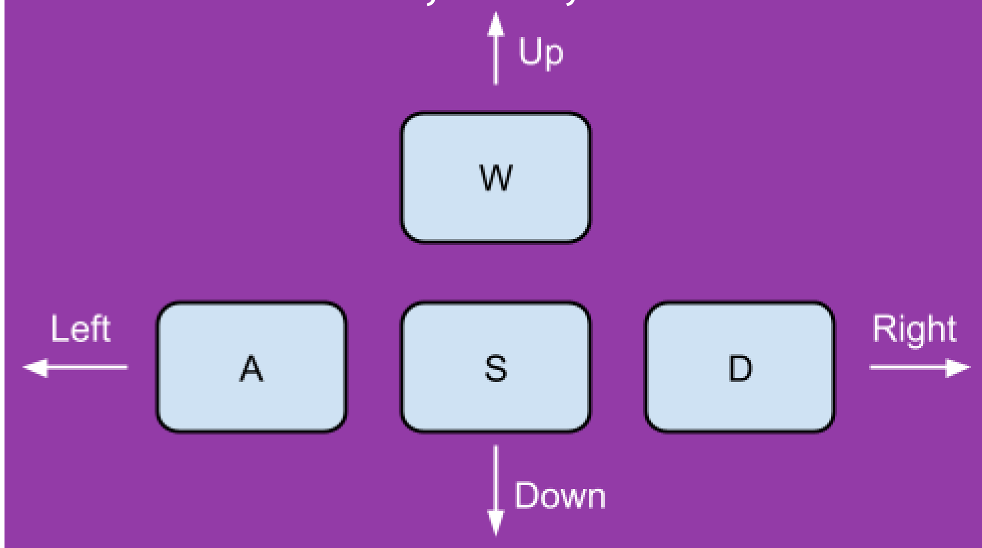
Explanation - What is a script?

- A text file with instructions that the Unity game engine reads
- The language used is C# (others can be used, but we will use C#)
- The scripts are often called **Classes**
- We use scripts to create **custom functionality** for our **GameObjects**
- We can attach scripts to **GameObjects**
- We can interact with any other components (**Rigidbody2D**, **Transform** etc) also on that GameObject

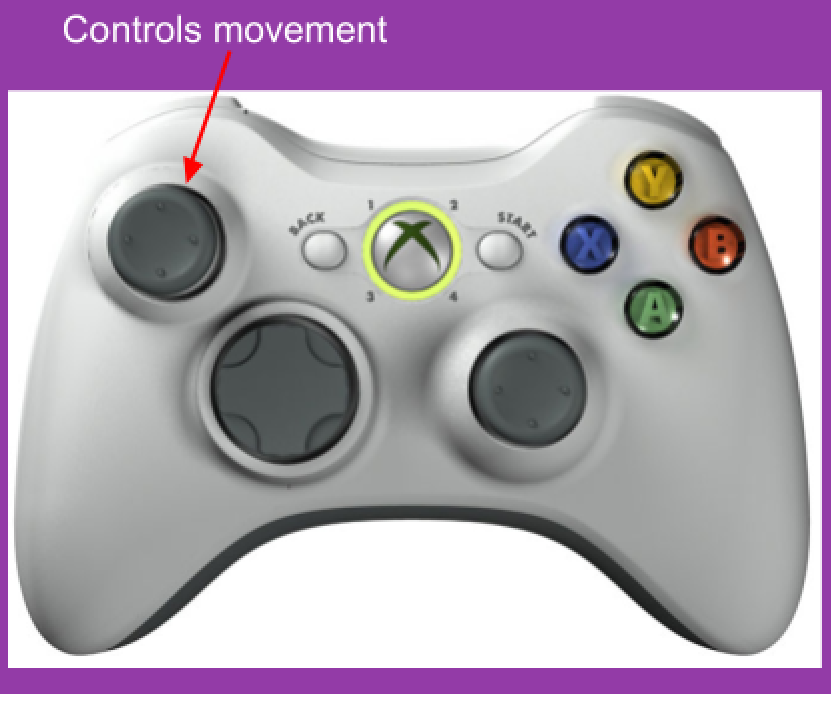
Explanation

- This script will **control** the **Player** character's **movements**
- The **movement** will be using **keyboard keys** or **joypad**

Keyboard keys



Joypad controls



Useful links

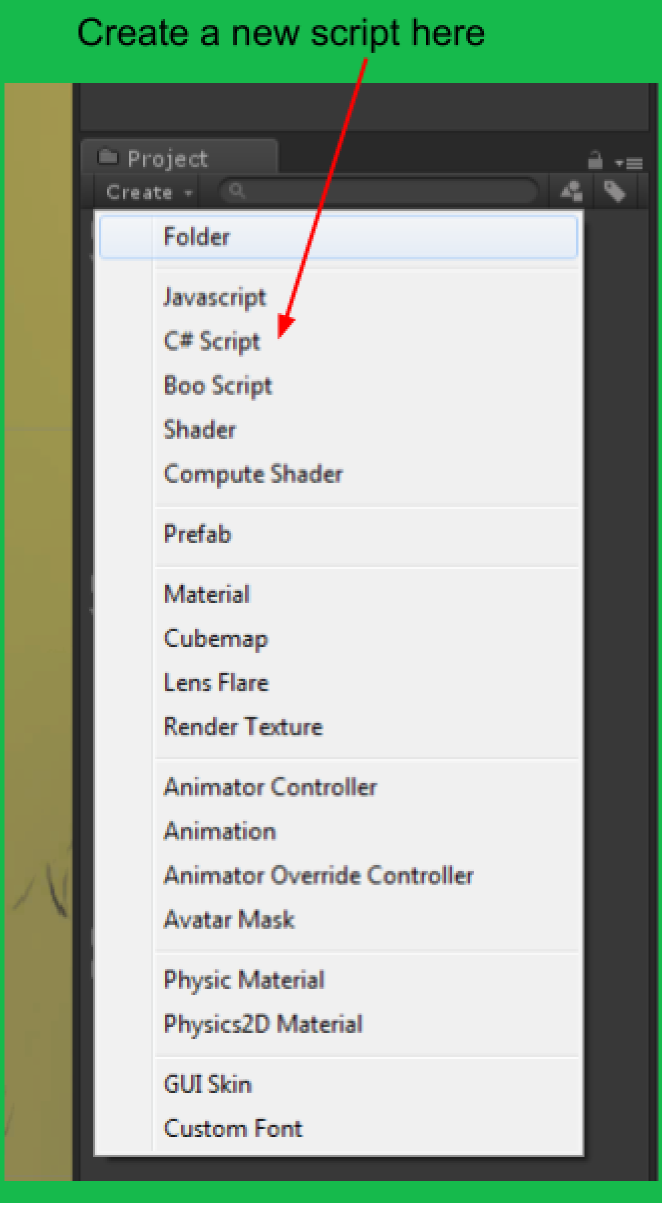
Getting input from an axis	GetAxis - Video
Update event functions	Update and FixedUpdate - Video
Attributes to show and hide properties	Attributes - Manual
A short explanation of O.O.P. Inheritance	Inheritance - Video
How to show decimal (float) values in C#	float (C# Reference) - MSDN Manual
How to get the type of an object in C#	typeof (C# Reference) - MSDN Manual

Do this

- In the **Project view**, create a new Folder in the **Assets** folder
- Name the Folder **Scripts**

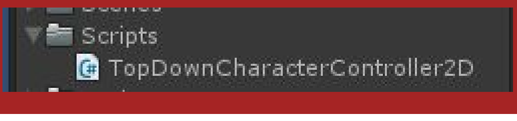
Do this

- In the **Scripts folder** in the **Project view**, create a new **C# Script**
- Name the Script **TopDownCharacterController2D**



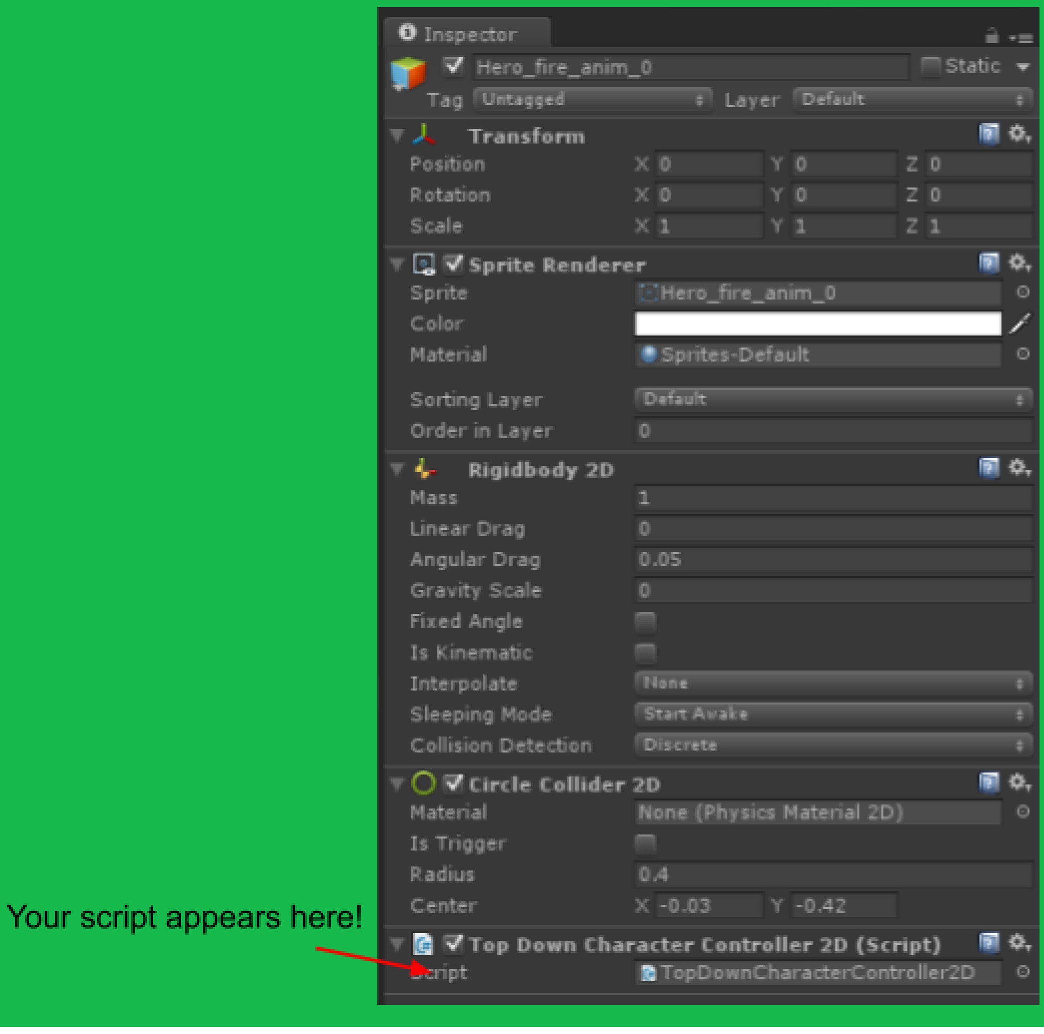
Check this

- Check the script is named **TopDownCharacterController2D**
- Check there are no spaces in the name!



Do this

- Drag the Script onto the **Hero** GameObject in the **Hierarchy**



Hero_fire_anim_0 Static
Tag Untagged Layer Default

Transform

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

Sprite Renderer

Sprite	Hero_fire_anim_0
Color	
Material	Sprites-Default
Sorting Layer	Default
Order in Layer	0

Rigidbody 2D

Mass	1
Linear Drag	0
Angular Drag	0.05
Gravity Scale	0
Fixed Angle	☐
Is Kinematic	☐
Interpolate	None
Sleeping Mode	Start Awake
Collision Detection	Discrete

Circle Collider 2D

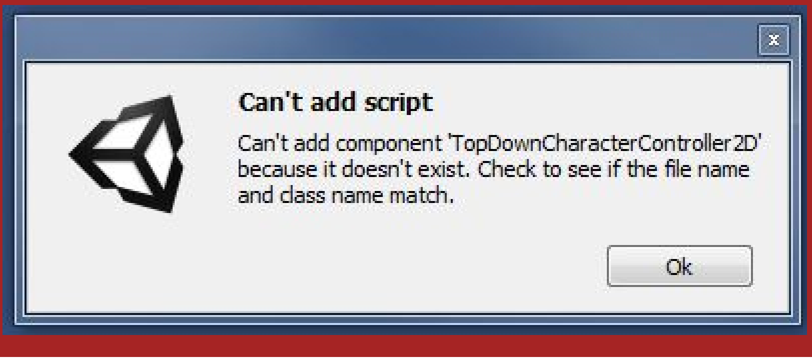
Material	None (Physics Material 2D)
Is Trigger	☐
Radius	0.4
Center	X -0.03 Y -0.42

Top Down Character Controller 2D (Script)

Script	TopDownCharacterController2D

Check this

- If you can't add the script to the Hero, you may get an error message like this
- Check below for an answer



Can't add script

Can't add component 'TopDownCharacterController2D' because it doesn't exist. Check to see if the file name and class name match.

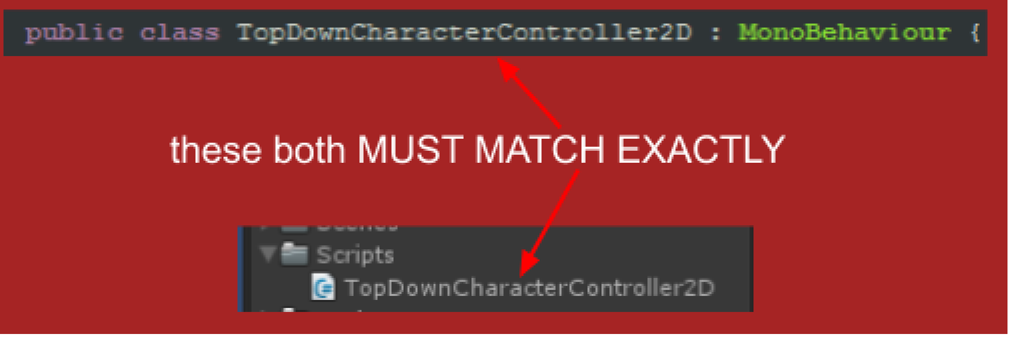
Do this

- In the **Project view** double click the **Script** to open it for editing in **Visual Studio**
- Look for this line in the code

```
public class TopDownCharacterController2D : MonoBehaviour {
```

Check this

- Check the name **TopDownCharacterController2D** matches **EXACTLY** the name of the script file



```
public class TopDownCharacterController2D : MonoBehaviour {
```

Explanation - Standard Libraries

- This script uses 2 **Standard libraries** that will be coded by default at the **top of the script file**
  - These libraries allow us to create code that interacts with the Unity game engine
- This code is added when you create a script by the **Unity Editor**

Check this

- Look at the top of your script file
- You will see code matching the code on the right
  - Note the semicolon at the end of each line!

```
using UnityEngine;
using System.Collections;
```

Explanation - What is UnityEngine?

- UnityEngine is the library that lets us interact with GameObjects and Components, along with many other parts of Unity
- We can create custom functionality using this library

Explanation - What is System.Collections?

- Allows us to use basic Arrays
- Arrays are lists of things, like GameObjects that we can manage
- For example, we will make a list of Zombies, Bullets and explosions later on

Useful links

- The using statement in C# allowing script to use libraries
- Collections, allowing script to use simple arrays

using statement - MSDN Manual
System.Collections - MSDN Manual

Note: UnityEngine is the library that allows us to interact with the Unity Game Engine

Explanation - Class declaration

- A class is a collection of C# code
- All the code in a class is contained within 2 curly braces

```
public class TopDownCharacterController2D : MonoBehaviour
{
  All code for this class is contained here!
}
```

Check this

- Your class declaration looks like this
  - Please see links above for more info

```
public class TopDownCharacterController2D : MonoBehaviour {
```

Explanation - MonoBehaviour

- This class **inherits** from **MonoBehaviour**
- This means our class (**TopDownCharacterController2D**) can leverage code from **MonoBehaviour**

```
public class TopDownCharacterController2D : MonoBehaviour {
```

- We can see the word **MonoBehaviour** after a colon (:), which is after our class name
- The colon (:) states our class inherits functionality from MonoBehaviour

Explanation - public

- The word **public** at the beginning of a class means we can use the script on our GameObjects
- The word **public** is an **accessor**, meaning "publicly available"
- Our **public** class is **publicly** available to use on **GameObjects**

Useful links

More about C# classes in Unity	Classes - Video
More about the MonoBehaviour library	MonoBehaviour - Manual
More about Accessors (public, private etc)	Access Modifiers - MSDN

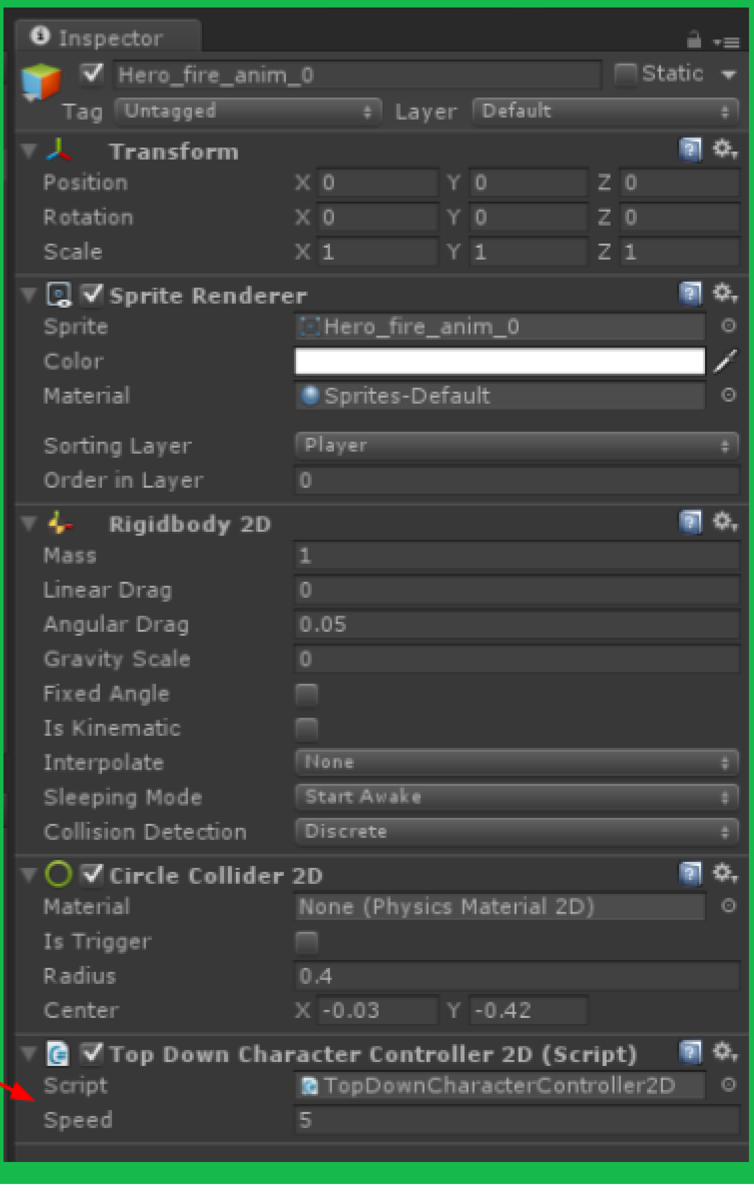
Do this

- Type out this code into your script file
- Make sure your code is ***EXACTLY*** the same!
  - Check **captial** and **lower case** lettering - **you have been warned!**
- **Replace** any other code there if it doesn't appear below

```
using UnityEngine;
using System.Collections;

public class TopDownCharacterController2D : MonoBehaviour {
    public float speed = 5.0f;
    Rigidbody2D rigidbody2D;

    void Start() {
        rigidbody2D = GetComponent<Rigidbody2D>();
    }

    void FixedUpdate(){
        float x = Input.GetAxis( "Horizontal" );
        float y = Input.GetAxis( "Vertical" );

        rigidbody2D.velocity = new Vector2( x, y ) * speed;
        rigidbody2D.angularVelocity = 0.0f;
    }
}
```

Do this

- In the **Unity Editor**, select the **Hero** GameObject in the **Hierarchy**
- From the **Project view** drag your **TopDownCharacterController2D** script to the **Hero** in the **Inspector**



Inspector

Hero_fire_anim_0 — Static

Tag: Untagged — Layer: Default

Transform

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

Sprite Renderer

Sprite	Hero_fire_anim_0
Color	
Material	Sprites-Default
Sorting Layer	Player
Order in Layer	0

Rigidbody 2D

Mass	1
Linear Drag	0
Angular Drag	0.05
Gravity Scale	0
Fixed Angle	
Is Kinematic	
Interpolate	None
Sleeping Mode	Start Awake
Collision Detection	Discrete

Circle Collider 2D

Material	None (Physics Material 2D)
Is Trigger	
Radius	0.4
Center	X -0.03 Y -0.42

Top Down Character Controller 2D (Script)

Script	TopDownCharacterController2D
Speed	5

Explanation - properties

- Look at the following line from our class:

```
public float speed = 5.0f;
```

- This is a **property**
- Its name is **speed**
- The **speed property** will **control** our **movement speed**
- Its **value** is 5.0
  - The "f" at the end of the value is a C# way of saying "this is a decimal number"
  - The value **5.0f** is known as the "default" value as it is assigned as soon as we created the property
- Its **type** is **float** (a decimal number)
- The **public** part allows us to edit the value:
  - From the Unity Editor as part of a Component
  - From other classes
- Note: the semicolon (;) at the end of the line!

Explanation - properties

- Look at the following line from our class:

```
Rigidbody2D rigidbody2D;
```

- Its name is **rigidbody2D**
- The **rigidbody2D property** will **move our Hero**
- Its **value** is set in the **Start** method
- Its **type** is **Rigidbody2D** (a Component)
- Note: the semicolon (;) at the end of the line!

Explanation - Start

- **Start** is an **Event Function** (or **method**)
- **Event Functions** are part of the **MonoBehaviour** code that we can use for our custom code
- **Start** runs **Once** when the GameObject is created

```
void Start(){

}
```

Explanation - FixedUpdate

- **FixedUpdate** is an **Event Function** (or **method**)
- **Event Functions** are part of the **MonoBehaviour** code that we can use for our custom code
- **FixedUpdate** runs **constantly** while the game is running, so any code inside the method will be running constantly!

```
void FixedUpdate(){

}
```

Useful links

- Please refer to the **Event Functions help sheet** on the **DLE website** for more information on **FixedUpdate**

Explanation - Our custom Start code

- Here is our custom code used inside the **Start** method
- This code gets the **Rigidbody2D** Component we placed on our **Hero GameObject** in the **Inspector**
- It then assigns the **Rigidbody2D** Component to our **rigidbody2D** property
- We can use it to move our character

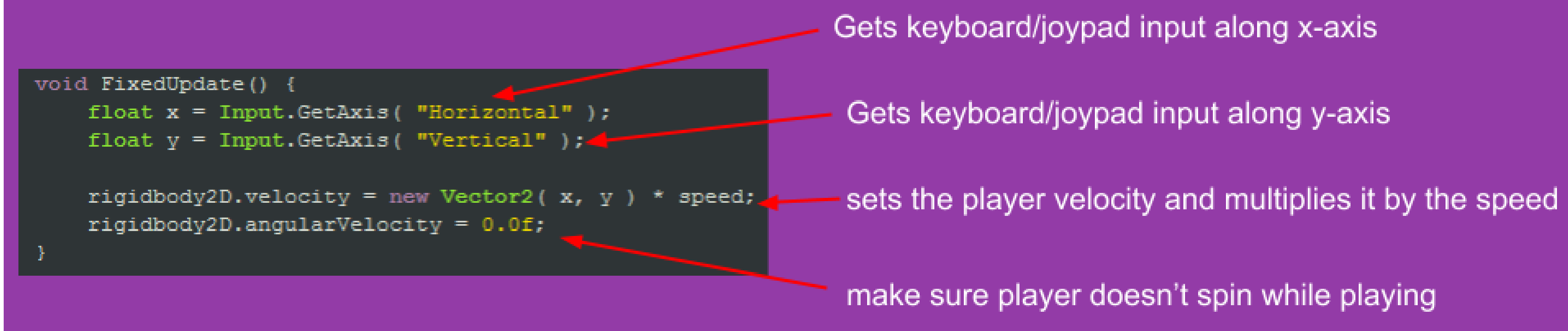
```
void Start(){
    rigidbody2D = GetComponent<Rigidbody2D>();
}
```

Explanation - Our custom FixedUpdate code

- Here is our custom code used inside the **FixedUpdate** method
- This code gets input from the keyboard or joypad and adds movement to the rigidbody

```
void FixedUpdate(){
    float x = Input.GetAxis( "Horizontal" );
    float y = Input.GetAxis( "Vertical" );

    rigidbody2D.velocity = new Vector2( x, y ) * speed;
    rigidbody2D.angularVelocity = 0.0f;
}
```

Explanation - code breakdown



```
void FixedUpdate() {
    float x = Input.GetAxis( "Horizontal" );
    float y = Input.GetAxis( "Vertical" );

    rigidbody2D.velocity = new Vector2( x, y ) * speed;
    rigidbody2D.angularVelocity = 0.0f;
}
```

Gets keyboard/joypad input along x-axis
Gets keyboard/joypad input along y-axis
sets the player velocity and multiplies it by the speed
make sure player doesn't spin while playing

Explanation - Line 1

- The first line (**highlighted in black**) creates a float variable named **x**
- **x** is assigned the value from **Input.GetAxis()**

```
void FixedUpdate(){
    float x = Input.GetAxis( "Horizontal" );
    float y = Input.GetAxis( "Vertical" );

    rigidbody2D.velocity = new Vector2( x, y ) * speed;
    rigidbody2D.angularVelocity = 0.0f;
}
```

- Note: the **axis** we are **getting input** from is **Horizontal** (left to right), or **along** the **X-axis**
- The axis **Horizontal** is already setup in the **Input Manager** for us

Explanation - Line 2

- The next line creates a float variable named **y**
- **y** is assigned the value from **Input.GetAxis()**

```
void FixedUpdate(){
    float x = Input.GetAxis( "Horizontal" );
    float y = Input.GetAxis( "Vertical" );

    rigidbody2D.velocity = new Vector2( x, y ) * speed;
    rigidbody2D.angularVelocity = 0.0f;
}
```

- Note: the **axis** we are **getting input** from is **Vertical** (up to down), or **along** the **Y-axis**
- The axis **Vertical** is already setup in the **Input Manager** for us

Useful links

Documentation on **Input.GetAxis()**	Input.GetAxis()
Documentation on the **Input Manager**	Input Manager

Explanation - Line 3

- Next we take our **values** for **x** and **y** and add them to our **Rigidbody2D velocity** property
- We first access the **Rigidbody2D Component**, using the **lower case rigidbody2D**
- We then access its velocity property using a full stop (.) like so: **rigidbody2D.velocity**
- We then assign the **velocity value** by creating a **new Vector2()** variable
- The **Vector2** variable **contains** 2 values (called x and y) we can put our **x** and **y** values into
- Finally, we **multiply** our new **velocity** by our **speed property**

```
void FixedUpdate(){
    float x = Input.GetAxis( "Horizontal" );
    float y = Input.GetAxis( "Vertical" );

    rigidbody2D.velocity = new Vector2( x, y ) * speed;
    rigidbody2D.angularVelocity = 0.0f;
}
```

Useful links

Documentation on **Rigidbody2D velocity**	Rigidbody2D.velocity
Documentation on **Vector2**	Vector2
Documentation on **Vector2 operator ***	Vector2.operator *

Explanation - Line 4

- The last line simply **stops** the **Player** from **spinning** if it gets hit by something.
- **Rigidbodies** react to in-game physics, so sometimes we need to disable parts so they work the way we want.
- We don't want the player spinning randomly while playing the game!

```
void FixedUpdate(){
    float x = Input.GetAxis( "Horizontal" );
    float y = Input.GetAxis( "Vertical" );

    rigidbody2D.velocity = new Vector2( x, y ) * speed;
    rigidbody2D.angularVelocity = 0.0f;
}
```

Useful links

Documentation on **Rigidbody2D angular velocity**	Rigidbody2D.angularVelocity

Task 2. Make the Camera follow the player

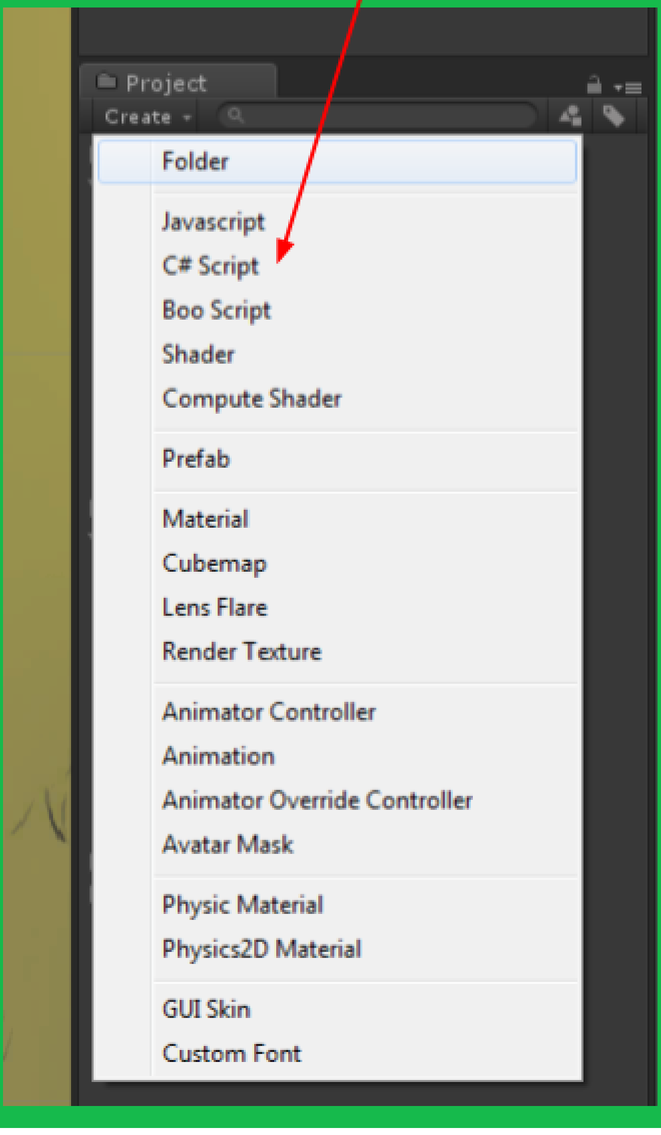
Explanation

- This script tells the **camera** to follow a target and smoothly **animate** its **movement**
- You can apply this script to lots of other things to move smoothly, not just the camera!

Do this

- In the **Project view**, create a new **C# Script** in the **Scripts Folder**
- Name the Script **SmoothFollow2D**

Create a new script here



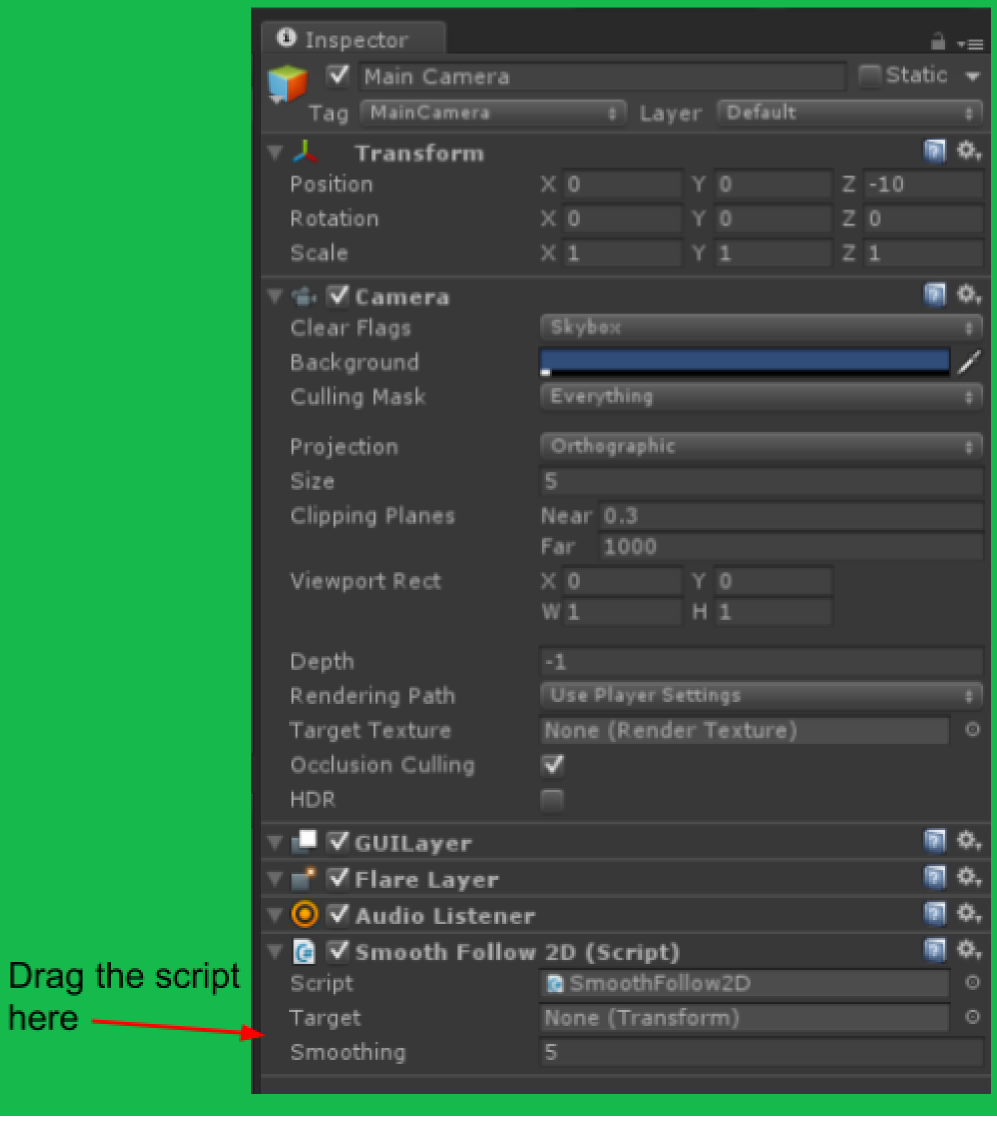
Do this

- Type out this code into your script file
- Make sure your code is ***EXACTLY*** the same!

```
using UnityEngine;
using System.Collections;

public class SmoothFollow2D : MonoBehaviour {
    public Transform target;
    public float smoothing = 5.0f;

    void FixedUpdate(){
        Vector3 newPos = new Vector3( target.position.x, target.position.y, transform.position.z );
        transform.position = Vector3.Lerp( transform.position, newPos, ( smoothing * 0.001f ));
    }
}
```

Do this

- In the **Unity Editor**, select the **Main Camera** GameObject in the **Hierarchy**
- From the **Project view** drag your **SmoothFollow2D** script to the **Main Camera** in the **Inspector**



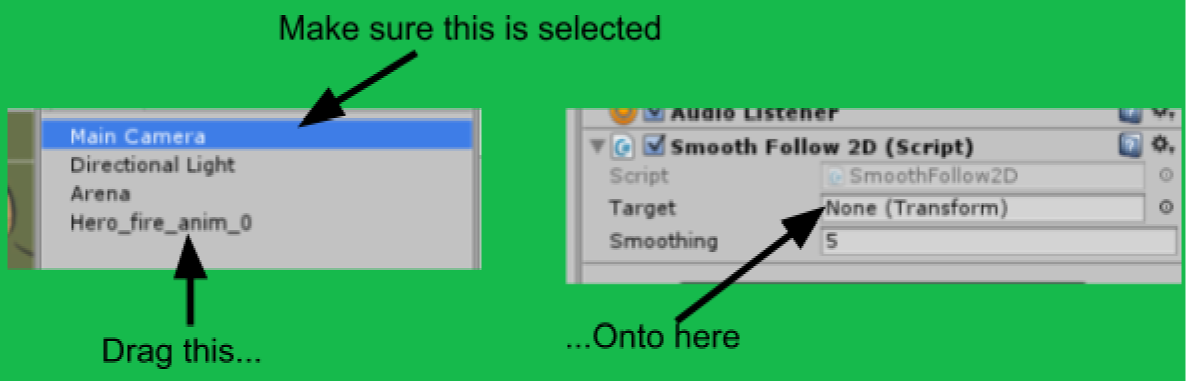
```
Main Camera                         Static
Tag  MainCamera       Layer  Default
Transform
Position          X 0    Y 0    Z -10
Rotation          X 0    Y 0    Z 0
Scale             X 1    Y 1    Z 1
Camera
Clear Flags       Skybox
Background
Culling Mask      Everything
Projection        Orthographic
Size              5
Clipping Planes   Near 0.3
                  Far  1000
Viewport Rect     X 0    Y 0
                  W 1    H 1
Depth             -1
Rendering Path    Use Player Settings
Target Texture    None (Render Texture)
Occlusion Culling ✓
HDR
GUILayer
Flare Layer
Audio Listener
Smooth Follow 2D (Script)
Script            SmoothFollow2D
Target            None (Transform)
Smoothing         5
```

Check this

- Your Camera Component has a property for 2D viewing
- It is called **Projection**
- Check the Projection is set to **Orthographic**
  - Orthographic is for 2D games
  - Perspective is for 3D games
- The Camera has a SmoothFollow2D script added to it

Do this

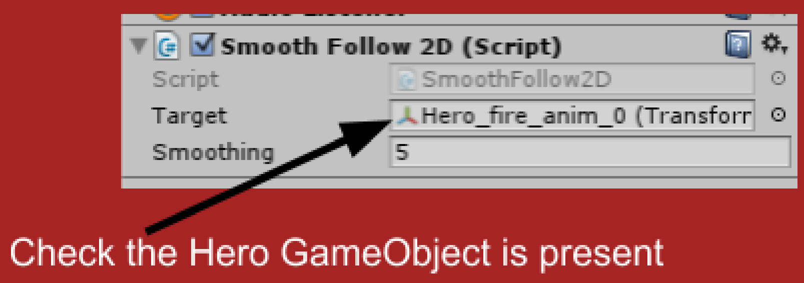
- In the **Unity Editor**, select the **Main Camera** GameObject in the **Hierarchy**
- Drag the **Hero** GameObject from the **Hierarchy** onto the **Target** property of the **SmoothFollow2D** Component



```
Main Camera
Directional Light
Arena
Hero_fire_anim_0

Smooth Follow 2D (Script)
Script      SmoothFollow2D
Target      None (Transform)
Smoothing   5
```

Check this

- The **SmoothFollow2D** script in the **Inspector** has the **Hero** GameObject in its **Target** property



```
Smooth Follow 2D (Script)
Script      SmoothFollow2D
Target      Hero_fire_anim_0 (Transforr
Smoothing   5
```

Useful links

- Documentation on **Camera** [Camera]

Explanation - target property

- The **camera** needs a **target** to **follow**, this **property** will hold a **reference** to it

```
public Transform target;
```

- target is **Editable** in the **Unity Editor**, because it is a **public property**
- target is a **Transform**, and as we know, every **GameObject** has a **Transform Component**, controlling its **position**, **rotation** and **scale**
- We will be using the **position property** of the **target Transform Component** later in this script

Useful links

- Documentation on **Transform** Transform

Explanation - smoothing property

- This **property** will control the **smoothness** of **movement** the **camera** uses when **animating** towards the **target**

```
public float smoothing = 5.0f;
```

- smoothing is **Editable** in the **Unity Editor**, because it is a **public property**
- smoothing is a **float**, a **decimal number**
- Note: we have **assigned a value** to **smoothing** when we **declared** it.
- This will be its **default value**, which can be **overridden** in the **Unity Editor**

Explanation - Our custom FixedUpdate code

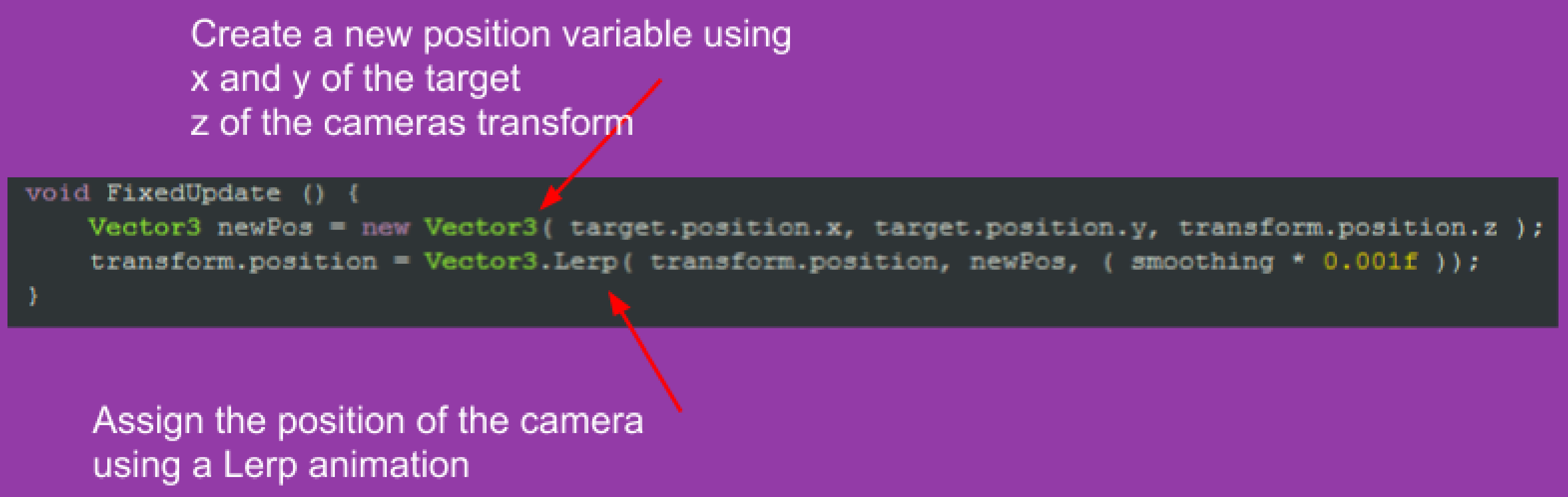
- Here is our custom code used inside the FixedUpdate method

```
void FixedUpdate(){
    Vector3 newPos = new Vector3( target.position.x, target.position.y, transform.position.z );
    transform.position = Vector3.Lerp( transform.position, newPos, ( smoothing * 0.001f ));
}
```

Useful links

- Please refer to the **Event Functions help sheet** on the **DLE website** for more information on **FixedUpdate**

Explanation - code breakdown


Create a new position variable using
x and y of the target
z of the cameras transform
Assign the position of the camera
using a Lerp animation

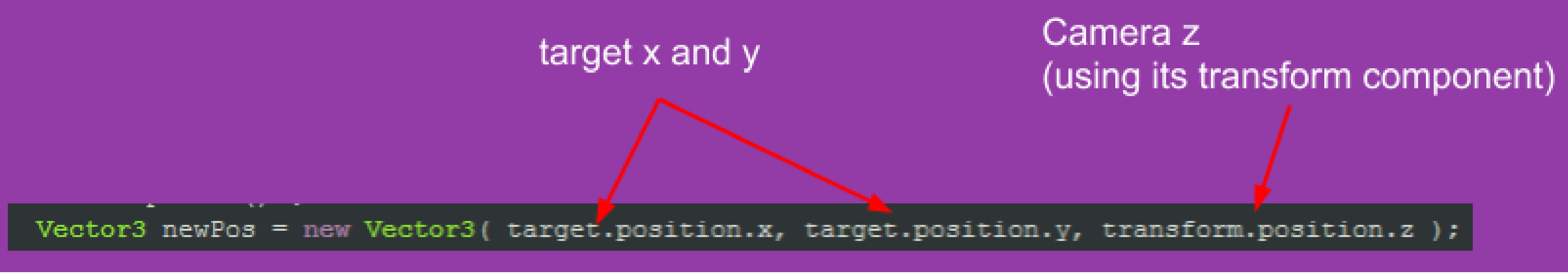
```
void FixedUpdate () {
    Vector3 newPos = new Vector3( target.position.x, target.position.y, transform.position.z );
    transform.position = Vector3.Lerp( transform.position, newPos, ( smoothing * 0.001f ));
}
```

Explanation - Line 1

- We create a new **Vector3** variable called **newPos** (short for new position)
- We assign the **newPos** variable's **x** and **y** values to the **target's x and y position:**
- x = target.position.x
- y = target.position.y
- We then set the **newPos z value** to its **own z value**, so it **doesn't change**
- Otherwise the **camera** will be **zooming** in and out of the screen!

```
void FixedUpdate(){
    Vector3 newPos = new Vector3( target.position.x, target.position.y, transform.position.z );
    transform.position = Vector3.Lerp( transform.position, newPos, ( smoothing * 0.001f ));
}
```

- Remember:



```
Vector3 newPos = new Vector3( target.position.x, target.position.y, transform.position.z );
```

Useful links

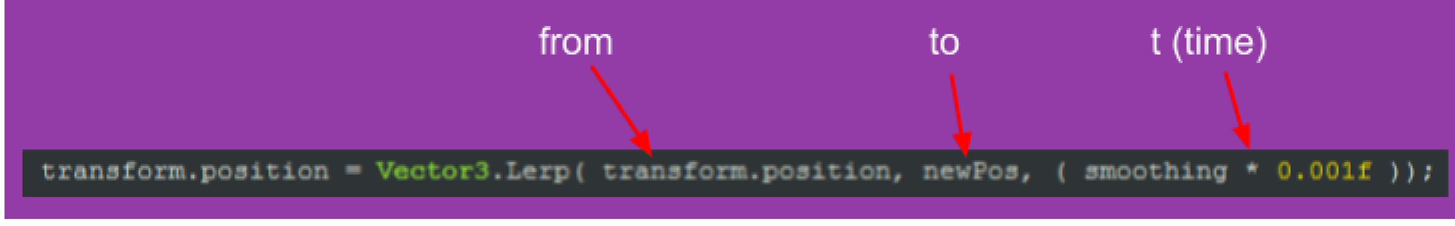
- Documentation on **Vector3** — Vector3

Explanation - Line 2

- We assign our **transform's position** here, performing the **smooth animation** to smoothly animate following the Player
- This **animation** is achieved using a method on the **Vector3** class called **Lerp** (or **Linear Interpolation**)
- **Vector3.Lerp** takes 3 parameters:
- **from** - where it starts animating from (requires a Vector3)
- **to** - where it animates to (requires a Vector3)
- **t** - time, how long it takes to animate (requires a float)

```
void FixedUpdate(){
    Vector3 newPos = new Vector3( target.position.x, target.position.y, transform.position.z );
    transform.position = Vector3.Lerp( transform.position, newPos, ( smoothing * 0.001f ));
}
```

- Note for the **from** parameter we use the **camera's own position, transform.position**
- For the **to** parameter we use the **newPos** variable we created on the previous line
- For the **t** parameter we apply the smoothing
- This will smooth more quickly or slowly depending on the size of the number we give
- The **smoothing** value is **reduced** here by **multiplying** it by a **very small number**
- This is so we don't have to type **tiny decimal values** in the **Unity Editor**, just **normal ones** like 5 or 20



```
transform.position = Vector3.Lerp( transform.position, newPos, ( smoothing * 0.001f ));
```

Useful links

- Documentation on **Vector3.Lerp** — Vector3.Lerp

More useful links

Documentation on **classes**	Classes - Video
The using directive for libraries	using Directive (C# Reference) - MSDN Manual
Collections, allowing script to use simple arrays	C# System.Collections Directive (C# Reference) MSDN Manual
Introduction to scripting in Unity	Unity scripting - Manual
About GameObjects and Components	Controlling GameObjects using Components - Manual
Manual entry about Event Functions	Event Functions - Manual
Order of execution of Event Functions	Execution order of event functions - Manual
About 2D gameplay in Unity	Gameplay in 2D - Manual

INTERACTIVE SYSTEMS STUDIO

CGD