C# in Unity

Sebastian Grygorczuk - STEM Institute at CCNY

Today’s Agenda

We’re going to be going to be continuing with Chapter 8: Scripting 2

- Learn how to access components using your scripts
- Create movement
- Spawn enemies and obstacles from PreFabs
- Use Trigger and Timing Mechanics

Connecting to Components

Just like Ints, Floats, String and so on Classes can be called in as variables.

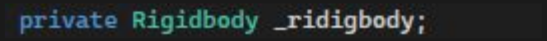
Each component is a class and we can declare them in the script.

The way we can connect them is by calling GetComponent<Class_Name>();

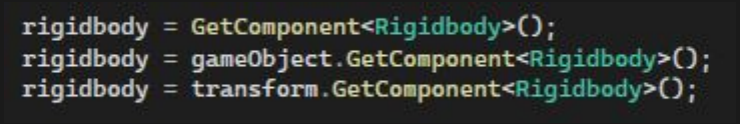
We can directly use GetComponent if the Script we're using and the Component we're trying to connect to are on the same Game Object.

C# is able to tell that you are trying to connect in the same Game Object so no necessary searching is required.

If you want to be more specific you can also use gameObject or transform which directly reference a Game Object the script is connected to.



```
private Rigidbody _ridigbody;
```



```
rigidbody = GetComponent<Rigidbody>();
rigidbody = gameObject.GetComponent<Rigidbody>();
rigidbody = transform.GetComponent<Rigidbody>();
```

Connecting to Children of Game Object

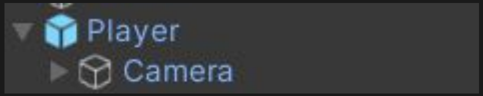
Sometime you will want to change or use an aspect of a child of the game object you are currently looking at.

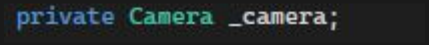
Similar to the case in the First Person Script we've use in the 3D terrain maps.

To do this we reference transform, which acts as a central line of the game object as it part of every game object.

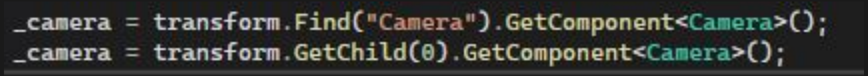
transform.Find("Child_Name") will be able to find the child and you can sue the same GetComponent Method to connect to it.

If you know the position in the Child List you could also call it by the index.





```
private Camera _camera;
```



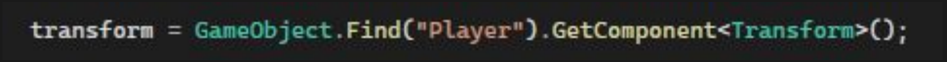
```
_camera = transform.Find("Camera").GetComponent<Camera>();
_camera = transform.GetChild(0).GetComponent<Camera>();
```

Connecting to a Object in the World

Sometime you will want to connect to a game object that’s not part of the current game object. Such as a scoreboard being updated, moving a camera to a new position or player health UI.

This is a very expensive function as it has to go through all of the game objects that are currently in the Hierarchy so you want to use it at the very start of the level once.

Using the capital GameObject refers to the class not the game object that the script is connected to, and using Find will scan through the whole Hierarchy ist to look for the Game Object with the given name.



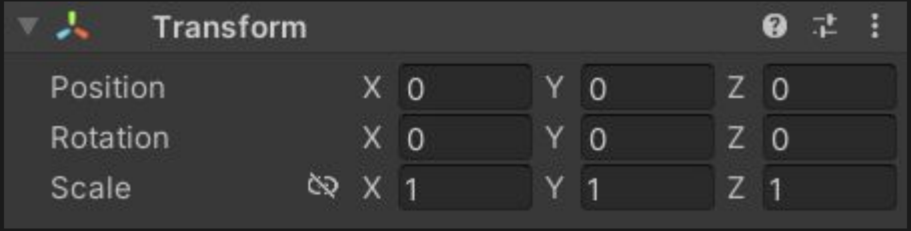
```
transform = GameObject.Find("Player").GetComponent<Transform>();
```

Transform and Vector 3

As we know every game object has a Transform component attached to it.

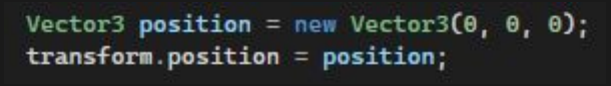
In that Transform component you have three Variables Position, Rotation and Scale. Each of those is using a Data type called Vector 3.

Vector 3 is a Array of 3 data types, in the case of Position, Rotation and Scale being float.



Transform

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

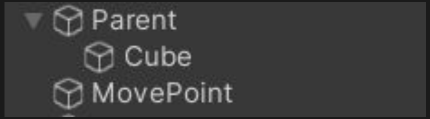


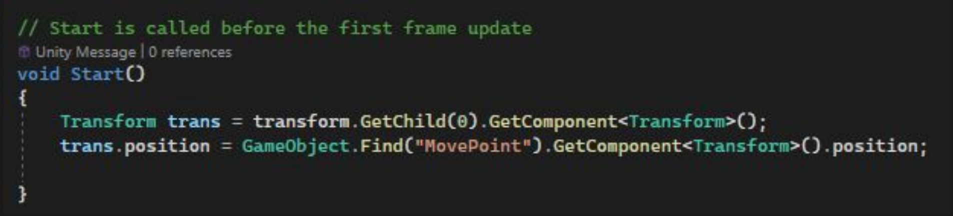
```
Vector3 position = new Vector3(0, 0, 0);
transform.position = position;
```

Challenge: Connection Data Transfer

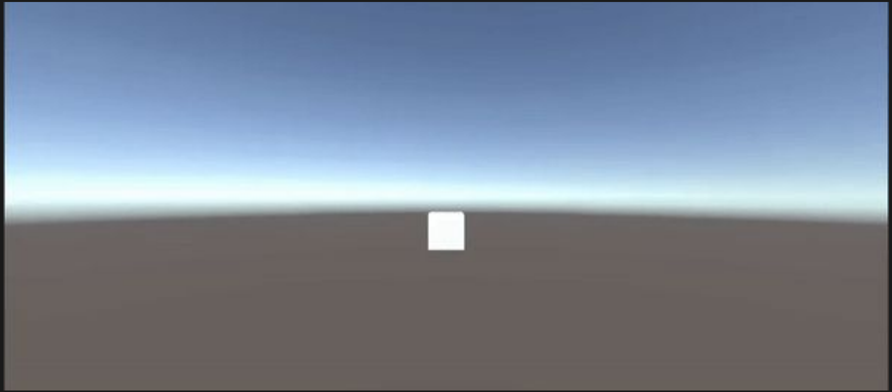
In Scene_1 you have two Game Objects, Parent which has a Cube nested in it and you have the MovePoint Object.

Using C# move the Cube and not the Parent to the MovePoint position.





```
// Start is called before the first frame update
Unity Message | 0 references
void Start()
{
    Transform trans = transform.GetChild(0).GetComponent<Transform>();
    trans.position = GameObject.Find("MovePoint").GetComponent<Transform>().position;

}
```

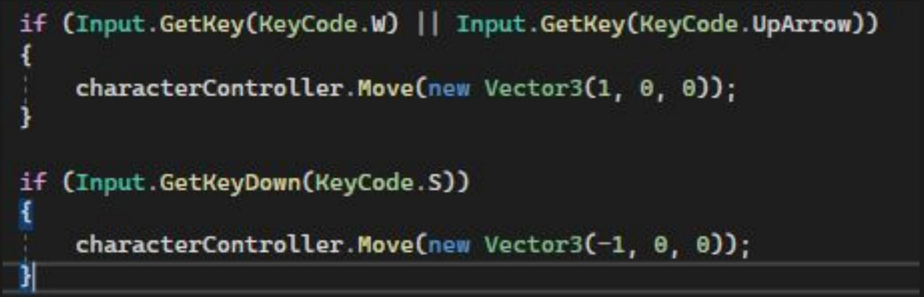


Inputs

Using the Input class you can call down different actions.

Using GetKey will make the action occur as long as the Key is down. Such cases could be like movement. Where you hold a key to move.

Using GetKeyDown would be true only once pressed, holding it would do nothing. Case for this could be casting a spell or performing an action.



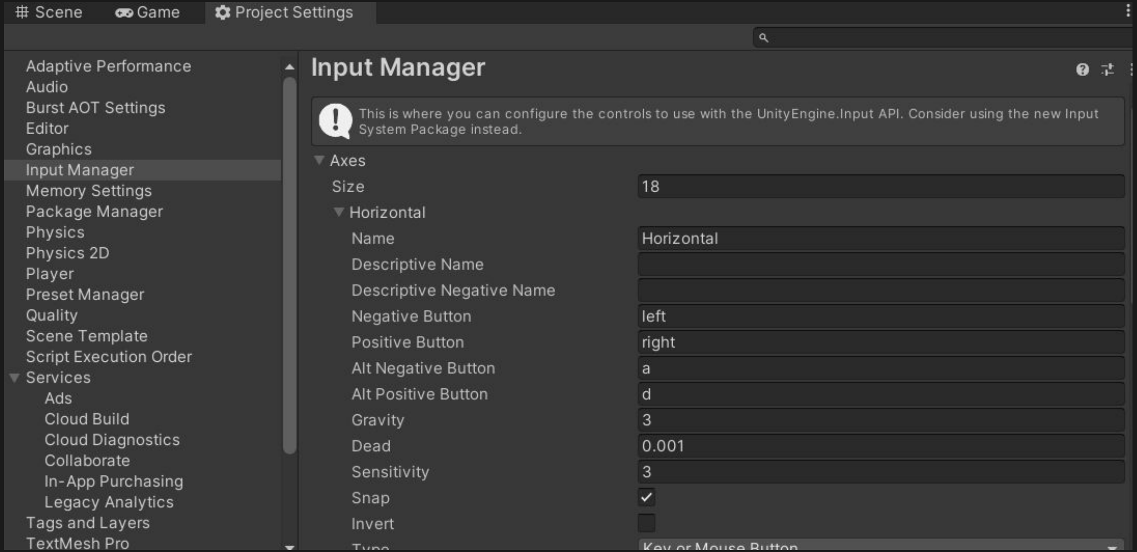
```
if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
{
    characterController.Move(new Vector3(1, 0, 0));
}

if (Input.GetKeyDown(KeyCode.S))
{
    characterController.Move(new Vector3(-1, 0, 0));
}
```

Input Manager

Unity has a way to hotkey several button to a nebulous action.

Some of those are Horizontal and Vertical which connects to A/D/Left Arrow Key/Right Arrow Key and W/S/Up Arrow Key/Down Arrow Key.

These action translate directly into float variables with A being -1, D being 1 and not holding either of the button's being 0.

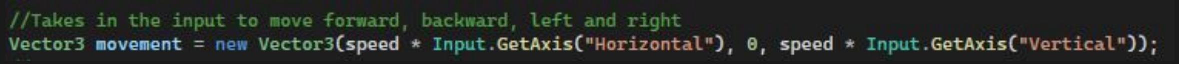


Input Manager

This is where you can configure the controls to use with the UnityEngine.Input API. Consider using the new Input System Package instead.

Axes

Size	18
Horizontal	
Name	Horizontal
Descriptive Name	
Descriptive Negative Name	
Negative Button	left
Positive Button	right
Alt Negative Button	a
Alt Positive Button	d
Gravity	3
Dead	0.001
Sensitivity	3
Snap	✓
Invert	



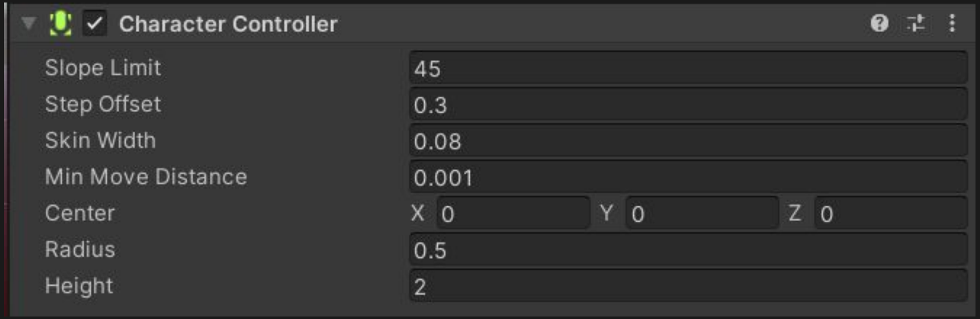
```
//Takes in the input to move forward, backward, left and right
Vector3 movement = new Vector3(speed * Input.GetAxis("Horizontal"), 0, speed * Input.GetAxis("Vertical"));
```

Character Controller

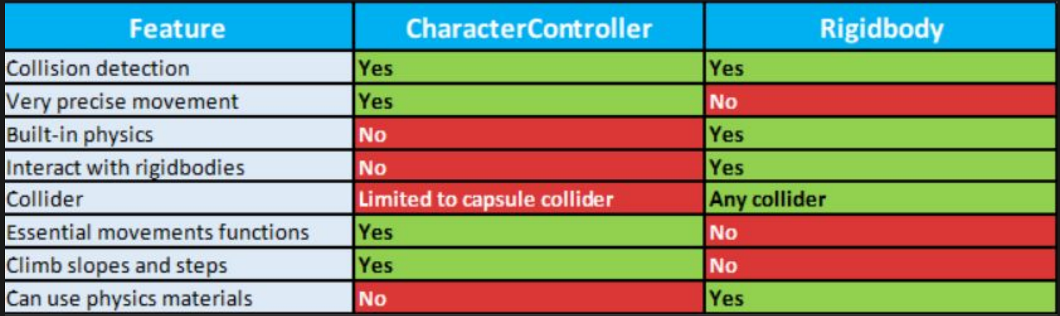
Character Controller is a component is similar to a Collider and Rigidbody combined into one with few benefit and downsided.

It's better for navigating levels but is limited to using the Capsule Collider and doesn't have Physics interactions.

Unity Character Controller vs Rigidbody



Slope Limit	45
Step Offset	0.3
Skin Width	0.08
Min Move Distance	0.001
Center	X 0 Y 0 Z 0
Radius	0.5
Height	2



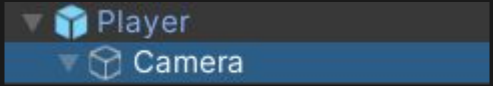
Feature	CharacterController	Rigidbody
Collision detection	Yes	Yes
Very precise movement	Yes	No
Built-in physics	No	Yes
Interact with rigidbodies	No	Yes
Collider	Limited to capsule collider	Any collider
Essential movements functions	Yes	No
Climb slopes and steps	Yes	No
Can use physics materials	No	Yes

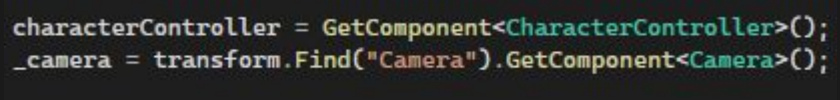
Making a Player Controller

To create a First Person controller we create an empty object named Player and attach the Camera as a child.

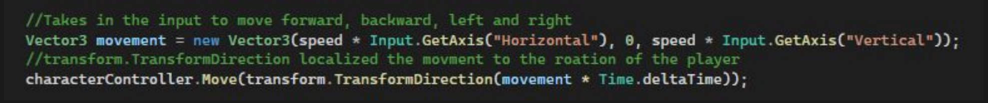
We will connect to the Character Controller to move the character and we will connect to the camera to edit it's rotation.

Using the Hotkeyed Horizontal And Vertical Axis Input we create a movement variable that will be multiple by Time.deltaTime and feed into the characterController using the Move function.

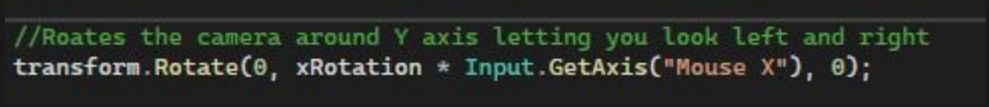




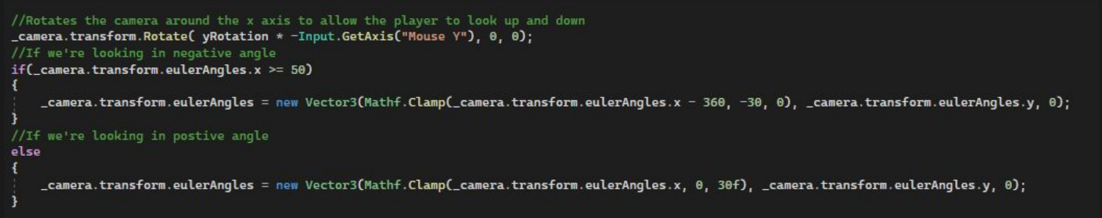
```
characterController = GetComponent<CharacterController>();
_camera = transform.Find("Camera").GetComponent<Camera>();
```



```
//Takes in the input to move forward, backward, left and right
Vector3 movement = new Vector3(speed * Input.GetAxis("Horizontal"), 0, speed * Input.GetAxis("Vertical"));
//transform.TransformDirection localized the movment to the roation of the player
characterController.Move(transform.TransformDirection(movement * Time.deltaTime));
```



```
//Roates the camera around Y axis letting you look left and right
transform.Rotate(0, xRotation * Input.GetAxis("Mouse X"), 0);
```



```
//Rotates the camera around the x axis to allow the player to look up and down
_camera.transform.Rotate( yRotation * -Input.GetAxis("Mouse Y"), 0, 0);
//If we're looking in negative angle
if(_camera.transform.eulerAngles.x >= 50)
{
    _camera.transform.eulerAngles = new Vector3(Mathf.Clamp(_camera.transform.eulerAngles.x - 360, -30, 0), _camera.transform.eulerAngles.y, 0);
}
//If we're looking in postive angle
else
{
    _camera.transform.eulerAngles = new Vector3(Mathf.Clamp(_camera.transform.eulerAngles.x, 0, 30f), _camera.transform.eulerAngles.y, 0);
}
```

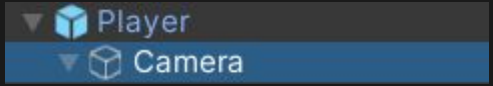
Making a Player Controller

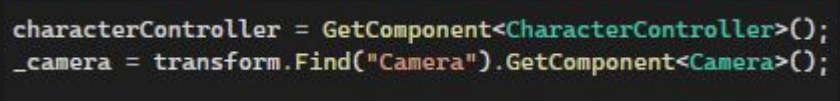
Next we deal with rotation, turning left or right is pretty simple. Just using the mouse X input to rotate around the Y axis.

Moving up and down is similarly simple by changing the rotation of the camera around the X axis.

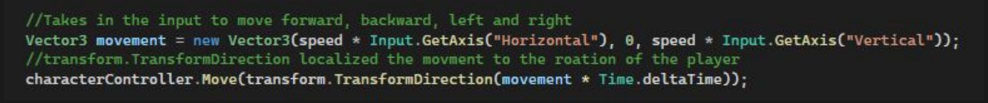
However you will be met with a camera that can make 360 turns around the body of the player which we don't want.

To fix that we use an If statement that checks if the direction that's currently looked in is negative or positive and then we clamp the input between two rangers. 0 to 30 and -30 to 0.

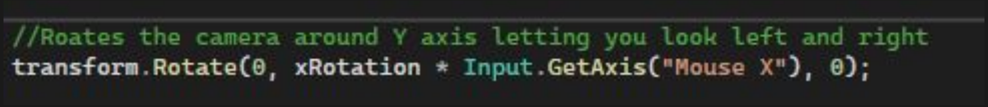




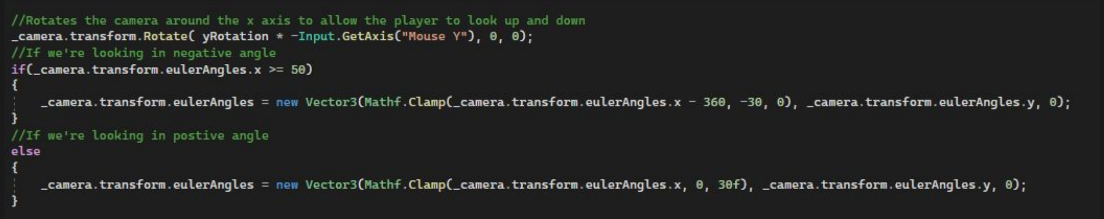
```
characterController = GetComponent<CharacterController>();
_camera = transform.Find("Camera").GetComponent<Camera>();
```



```
//Takes in the input to move forward, backward, left and right
Vector3 movement = new Vector3(speed * Input.GetAxis("Horizontal"), 0, speed * Input.GetAxis("Vertical"));
//transform.TransformDirection localized the movment to the roation of the player
characterController.Move(transform.TransformDirection(movement * Time.deltaTime));
```



```
//Roates the camera around Y axis letting you look left and right
transform.Rotate(0, xRotation * Input.GetAxis("Mouse X"), 0);
```



```
//Rotates the camera around the x axis to allow the player to look up and down
_camera.transform.Rotate( yRotation * -Input.GetAxis("Mouse Y"), 0, 0);
//If we're looking in negative angle
if(_camera.transform.eulerAngles.x >= 50)
{
    _camera.transform.eulerAngles = new Vector3(Mathf.Clamp(_camera.transform.eulerAngles.x - 360, -30, 0), _camera.transform.eulerAngles.y, 0);
}
//If we're looking in postive angle
else
{
    _camera.transform.eulerAngles = new Vector3(Mathf.Clamp(_camera.transform.eulerAngles.x, 0, 30f), _camera.transform.eulerAngles.y, 0);
}
```

Spawning Objects

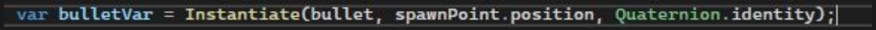
To spawn object you first have to have a reference to what you are spawning. Thus in your script you will have a public GameObject bullet variable that you'll be able to drag a prefab into from the Project View.

You will also need a location of where you want to spawn this object that you drag from the hierarchy view of the player.

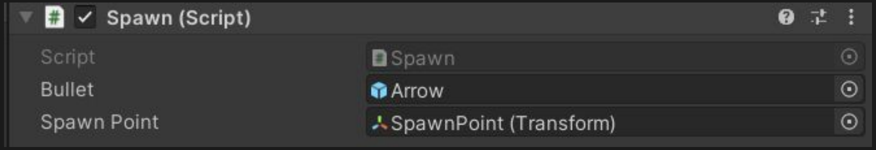
To create the object using script you will use Instate(gameObject, place, rotation) just leave the last one as is.

As you notice you will have Game Object with (Clone) next to them appear in the Hierarchy that means they are installed.

If you want to confined your object to another you can use the transform.Parent() and the specify the transform you want it to be attached to.

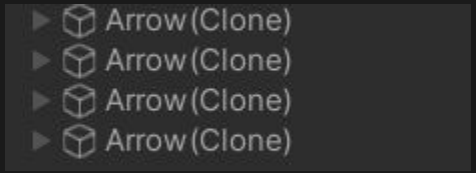


```
var bulletVar = Instantiate(bullet, spawnPoint.position, Quaternion.identity);
```

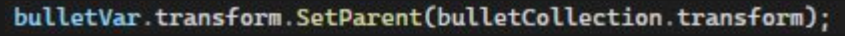


Spawn (Script)

Script	Spawn
Bullet	Arrow
Spawn Point	SpawnPoint (Transform)



Arrow(Clone)
Arrow(Clone)
Arrow(Clone)
Arrow(Clone)



```
bulletVar.transform.SetParent(bulletCollection.transform);
```

Sebastian Grygorczuk - STEM Institute at CCNY

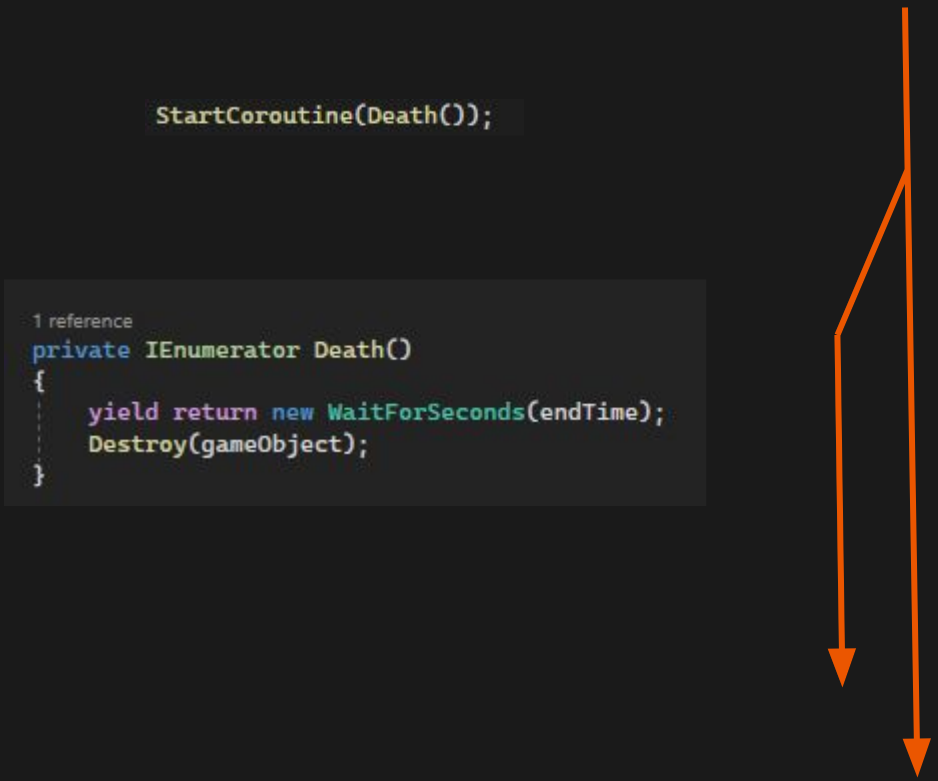
Coroutines & Destroying Objects

You game is all running in order, the code at the top of the method executes first and the code towards the bottom excused last.

Coroutines allow you to break out of this and create another path where two methods can run simoutnely.

You need to define the function as IEnumerator being the return type and use the yield keyword as the return. This tells you to wait before executing the code below.

You will also have to enclose the Method in StartCorutine() method.



```
StartCoroutine(Death());
```

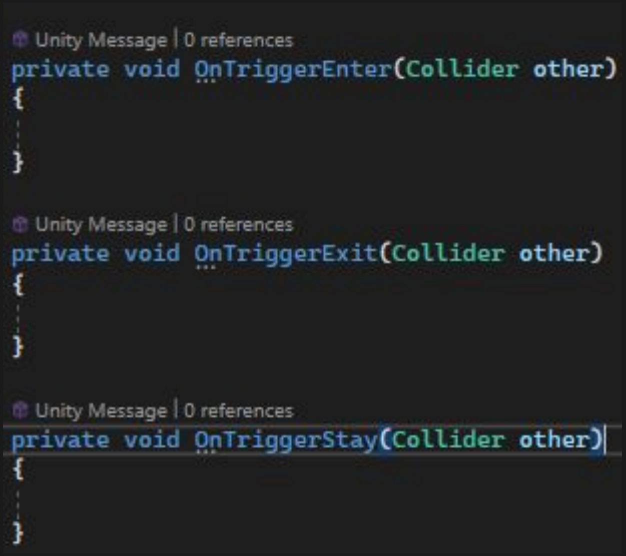
```
1 reference
private IEnumerator Death()
{
    yield return new WaitForSeconds(endTime);
    Destroy(gameObject);
}
```

Colliders and Triggers

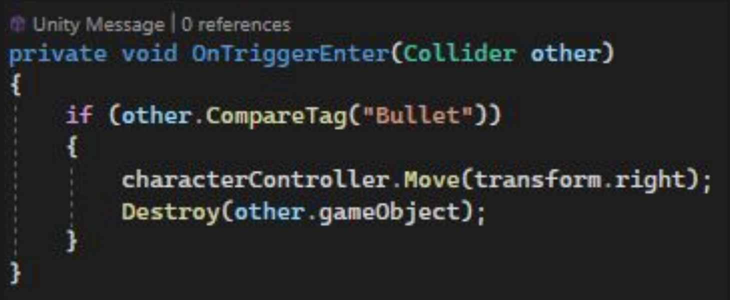
As we know collider have the Trigger toggle, once that's selected the collider turn into a non physical object that will allow you to perform action if something comes inside it.

In this case we only want to perform this action if a object with the Bullet Tag.

When that happens we push the object that got hit with the bullet and make sure that the bullet game object is destroyed.



```
Unity Message | 0 references
private void OnTriggerEnter(Collider other)
{

}

Unity Message | 0 references
private void OnTriggerExit(Collider other)
{

}

Unity Message | 0 references
private void OnTriggerStay(Collider other)
{

}
```

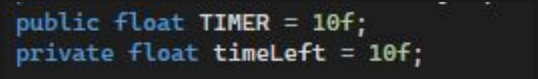


```
Unity Message | 0 references
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Bullet"))
    {
        characterController.Move(transform.right);
        Destroy(other.gameObject);
    }
}
```

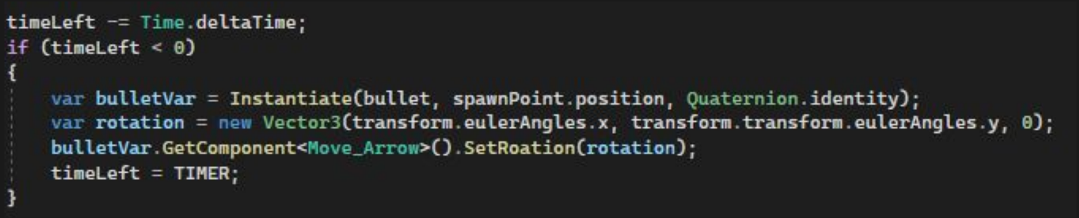
Timer

Timers are useful in many situation, in this case we will be creating one that will shoot bullets at the player.
You will need two variables the timeLeft which will be used to count down the time till an action should occur and TIMER which will reset the timeLeft once the action been performed.

Then in the update function we continuously subtract Time.deltaTime from the timeLeft and once it hit's below zero the action is triggered.



```
public float TIMER = 10f;
private float timeLeft = 10f;
```



```
timeLeft -= Time.deltaTime;
if (timeLeft < 0)
{
    var bulletVar = Instantiate(bullet, spawnPoint.position, Quaternion.identity);
    var rotation = new Vector3(transform.eulerAngles.x, transform.transform.eulerAngles.y, 0);
    bulletVar.GetComponent<Move_Arrow>().SetRoation(rotation);
    timeLeft = TIMER;
}
```

Sebastian Grygorczuk - STEM Institute at CCNY

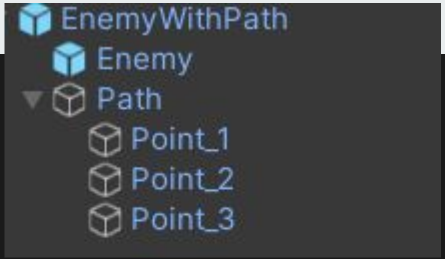
Enemy Following a Path

Creating an enemy will require you to create an array or list that will store the points that you want to move between. It will also require that the Enemy has a Parent where the enemy and the path exit as children.

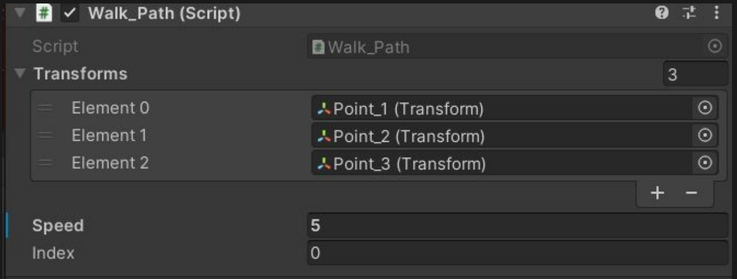
If you were to leave the path under the enemy they'd never reach it as it would move the same distance that the enemy has.

Now you will want to use Vector3.MoveTowards(currentPlace, Goal, step)

This will move the object towards the current goal in step.

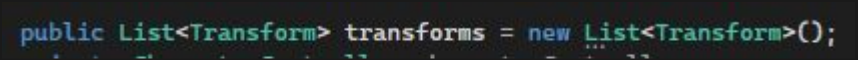


```
EnemyWithPath
  Enemy
  Path
    Point_1
    Point_2
    Point_3
```

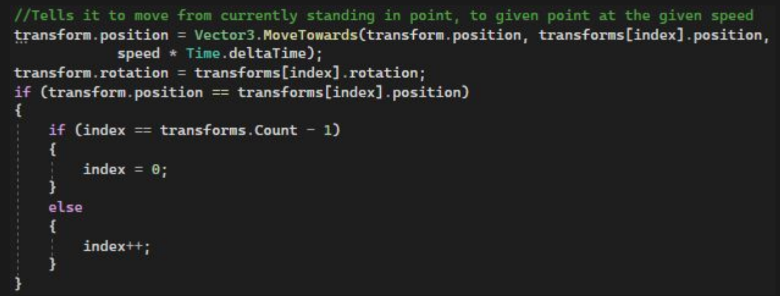


Walk_Path (Script)

Script	Walk_Path
Transforms	3
Element 0	Point_1 (Transform)
Element 1	Point_2 (Transform)
Element 2	Point_3 (Transform)
Speed	5
Index	0



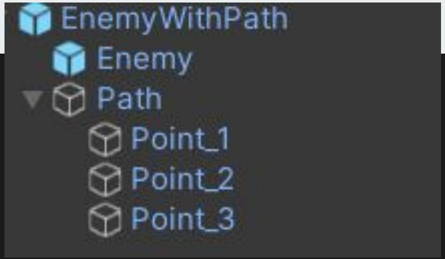
```
public List<Transform> transforms = new List<Transform>();
```



```
//Tells it to move from currently standing in point, to given point at the given speed
transform.position = Vector3.MoveTowards(transform.position, transforms[index].position,
            speed * Time.deltaTime);
transform.rotation = transforms[index].rotation;
if (transform.position == transforms[index].position)
{
    if (index == transforms.Count - 1)
    {
        index = 0;
    }
    else
    {
        index++;
    }
}
```

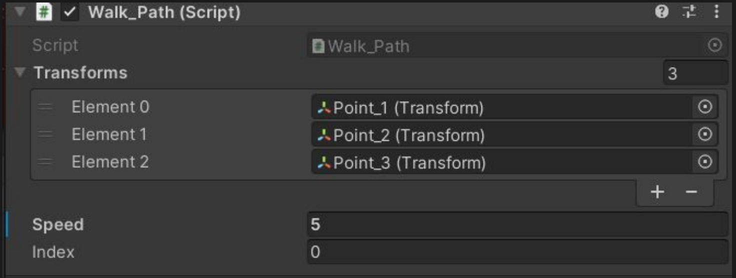
Sebastian Grygorczuk - STEM Institute at CCNY

Enemy Following a Path



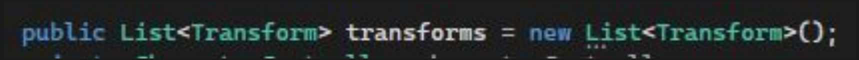
Once the enemy has reached a location you will want to give them the next location using the if statements.

If the enemy has reach the last location in the path you will want to reset them to 0th position.

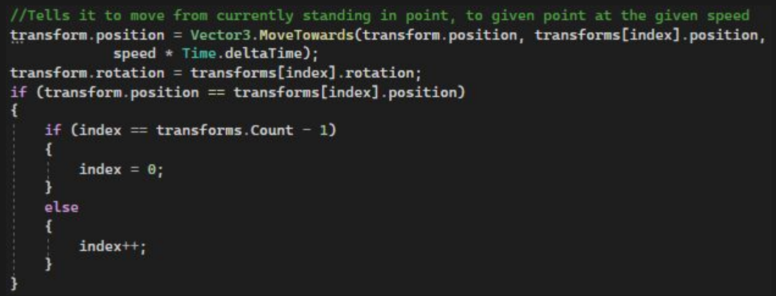


Walk_Path (Script)

Script	Walk_Path
Transforms	3
Element 0	Point_1 (Transform)
Element 1	Point_2 (Transform)
Element 2	Point_3 (Transform)
Speed	5
Index	0



```
public List<Transform> transforms = new List<Transform>();
```



```
//Tells it to move from currently standing in point, to given point at the given speed
transform.position = Vector3.MoveTowards(transform.position, transforms[index].position,
            speed * Time.deltaTime);
transform.rotation = transforms[index].rotation;
if (transform.position == transforms[index].position)
{
    if (index == transforms.Count - 1)
    {
        index = 0;
    }
    else
    {
        index++;
    }
}
```

Sebastian Grygorczuk - STEM Institute at CCNY