Triggers

Objective

You will understand triggers and how to use them to make many different mechanics in games

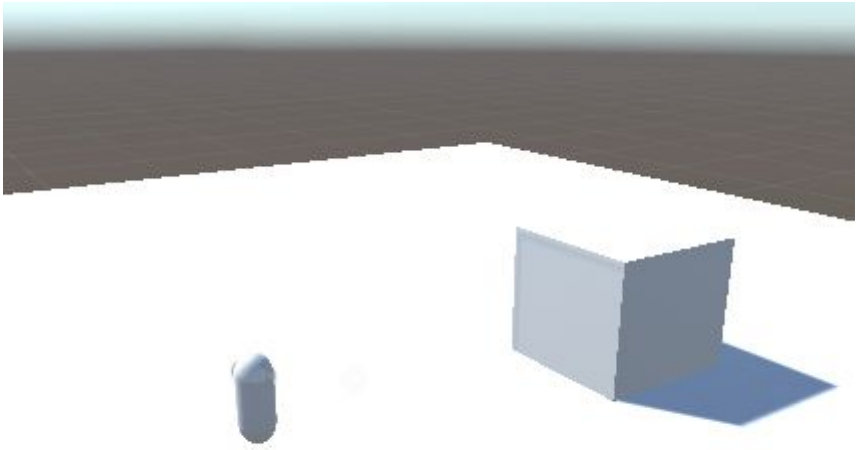
Basic Trigger

Making a trigger activate code when something moves into it and when something moves out of it.

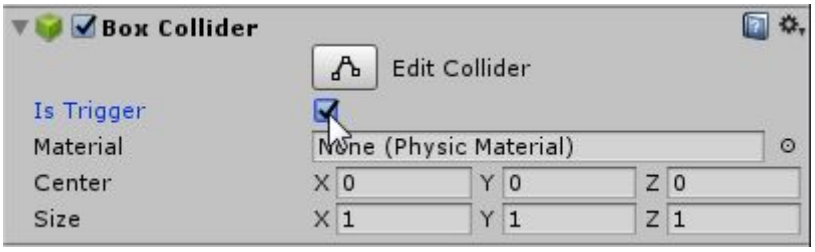
Setup

We have provided you with a RPG Package, Install this into your project.

- Locate the **AIE_Controller -> Prefabs -> FPS_Controller** and drag it into the scene
- Delete the **Main Camera** as the FPS_Controller has its own camera
- Make a large plane for the player to walk around
- Make a cube, scale it up and place it in the scene somewhere the player can move into it.



- On the Cube set the **Box Collider** to **Is Trigger**



- Rename the cube to **'MyTriggerObject'**
- Make a new script called **'TriggerCube'** and put it on the cube

You should now be able to run the game and walk right through the cube

Lesson

Update **TriggerCube** to the following:

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

public class TriggerCube : MonoBehaviour {

    //When something enters the Trigger
    void OnTriggerEnter()
    {
        Debug.Log("Happy to see you");
    }

    //When something exits the Trigger
    void OnTriggerExit()
    {
        Debug.Log("Sorry to see you go");
    }

    //When something exits the Trigger
    void OnTriggerStay()
    {
        Debug.Log("I wish you would leave");
    }

}
```

Results

When the player walks into the trigger the first message will play, and when the player walks out of the trigger the second message will play.
And when the player stays in the cube the third message will play;

Expanding what it can do.

Adding a few more things triggers can do

Setup

Continue from previous lesson.

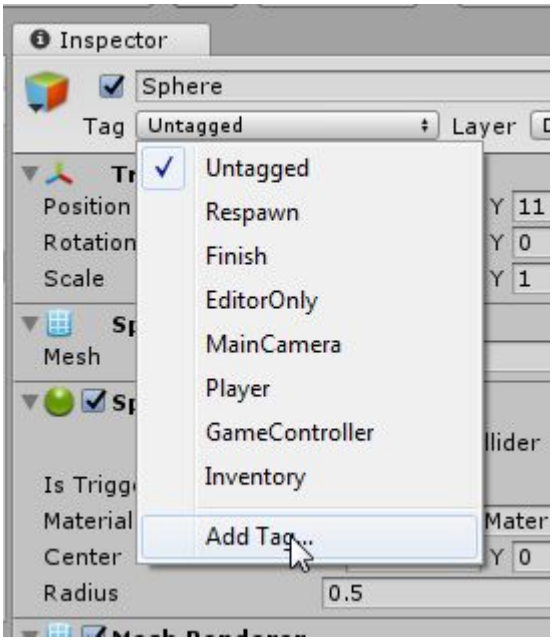
Lesson

Make a sphere above the trigger and add a rigidbody to it, press play and watch the sphere fall into the trigger and activate it.

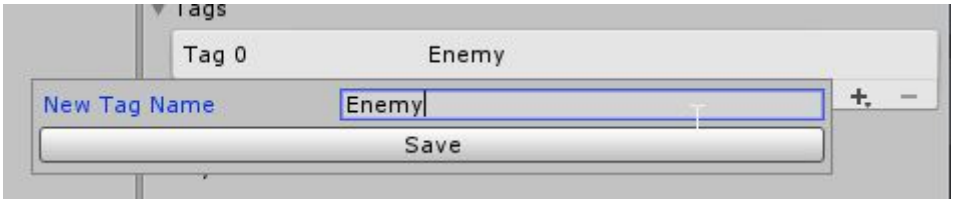
This may be what you want in you game but this might not be what you want.
Do you want an enemy triggering an event when they walk into the trigger by mistake?

Lets tag some objects so the trigger knows what is walking into it.

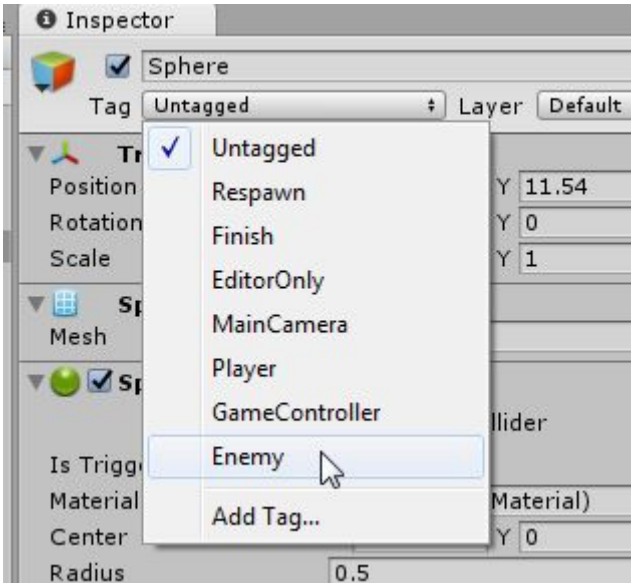
In the **Inspector** Click on **Tag -> Add Tag...**



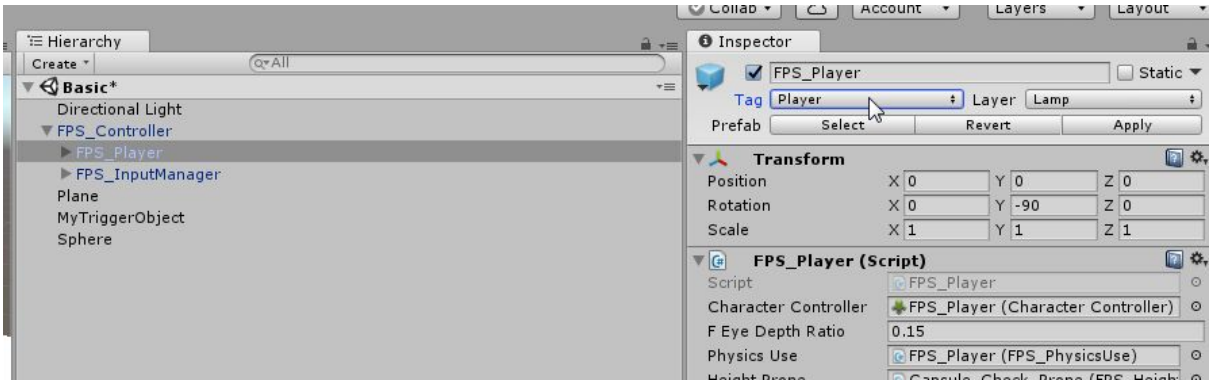
Add an **'Enemy'** tag and press **Save**



Select the Sphere and tag it as **Enemy**



If you look you will see that the player is already tagged as **Player**



Update the **TriggerCube** code to only allow objects with the **Player** tag to trigger it.

```
//When something enters the Trigger
void OnTriggerEnter(Collider other)
{
    if (other.tag == "Player")
    {
        Debug.Log("Happy to see you");
    }
}
```

```
//When something exits the Trigger
void OnTriggerExit(Collider other)
{
    if (other.tag == "Player")
    {
        Debug.Log("Sorry to see you go");
    }
}
```

```
//When something stays within the Trigger
void OnTriggerStay(Collider other)
{
    if (other.tag == "Player")
    {
        Debug.Log("I wish you would leave");
    }
}
```

Run the game and you should see that only the player activates the trigger

Results

Now only the player will activate a trigger.

Teleport

Now we can access the object entering the trigger, let's move it around

Setup

Create a new trigger the same way we did the previous lesson, but this time make a new script called **'TeleportScript'** and attach it to it.

Change values

Update **TeleportScript**

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

public class TeleportScript : MonoBehaviour {

    //When something enters the Trigger
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            other.transform.position = new Vector3(0, 0, 0);
        }
    }

}
```

Once this change has been made get the player to walk into the trigger, they should be moved to position 0,0,0.

Change the numbers

```
ion = new Vector3(0, 0, 0);
```

and have the player teleported to different positions.

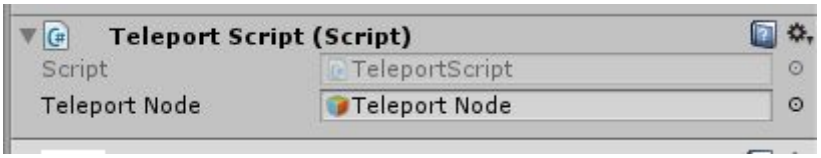
So instead of hard coding values, lets get a reference from another object.

Make a sphere, give it a colourful material and remove its collider.
Rename the sphere to 'Teleport Node'

Update the script to below, it will require them to drag the new Teleport Node into the public variable.

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

public class TeleportScript : MonoBehaviour {

    public GameObject teleportNode;

    //When something enters the Trigger
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            Vector3 nodeLocation = teleportNode.transform.position;
            other.transform.position = nodeLocation;
        }
    }

}
```

Update this



Once the node is dragged into the script students should be able to teleport to the sphere by walking into the trigger box.

A common request is to have the player teleport facing a certain direction.
For this we can not only copy the position of the teleport node but the rotation as well

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

public class TeleportScript : MonoBehaviour {

    public GameObject teleportNode;

    //When something enters the Trigger
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            Vector3 nodeLocation = teleportNode.transform.position;
            other.transform.position = nodeLocation;
            Quaternion nodeRotation = teleportNode.transform.rotation;
            other.transform.rotation = nodeRotation;

        }
    }

}
```

Now they should be able to rotate the node to have the player face in the right direction

If you add this to the script you can get a line in the editor to show the level designer where the teleport is going to.

```
    {
        if (other.tag == "Player")
        {
            Vector3 nodeLocation = teleportNode.transform.position;
            other.transform.position = nodeLocation;
            Quaternion nodeRotation = teleportNode.transform.rotation;
            other.transform.rotation = nodeRotation;

        }
    }

    void OnDrawGizmos()
    {
        if (teleportNode) {
            Gizmos.color = Color.blue;
            Gizmos.DrawLine(transform.position, teleportNode.transform.position);
        }
    }


}
```

Results

You should be able to teleport all over the map.

Make a RPG

Objective

Make a basic RPG using triggers to control the game.

Setup

- Load the scene in **TownKit -> Scenes -> Example-Scene-001**
- Make an empty game object called **'GameController'**
- Make script called **UIController**, attach it to the GameController
- Make an cube gameobject and place it in the scene somewhere as as trigger
- Rename the cube to **BasicTrigger**
- Set its collider to a **Trigger**
- Create a script called **TriggerDisplay** and place it on the **BasicTrigger**
- Remove the **Mesh Filter** and **Mesh Renderer**

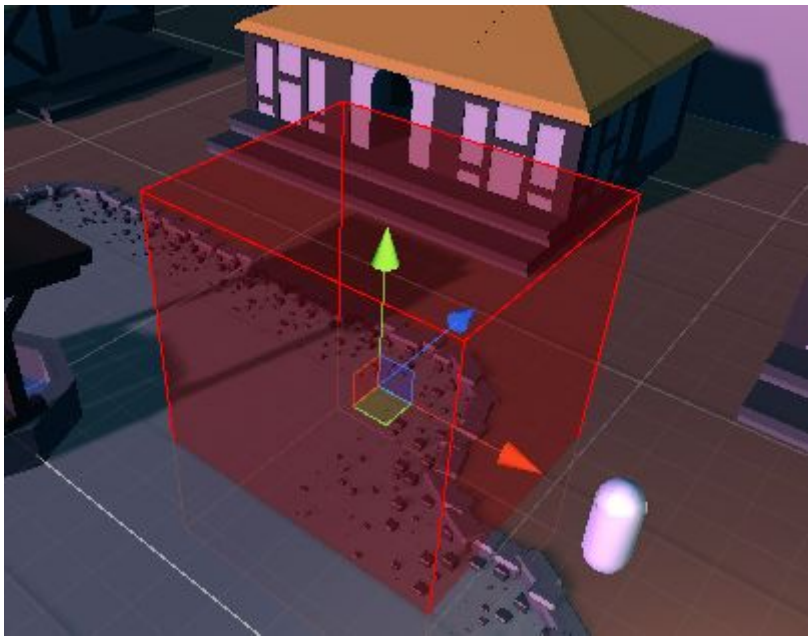
Lesson

Open the **TriggerDisplay** script up

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

[RequireComponent(typeof(BoxCollider))]
public class TriggerDisplay : MonoBehaviour {

    //This does some magic and draws a box that is only visible in the editor view. Great for debugging
    void OnDrawGizmos()
    {
        GetComponent<BoxCollider>().isTrigger = true;
        Vector3 drawBoxVector = new Vector3(
            this.transform.lossyScale.x * this.GetComponent<BoxCollider>().size.x,
            this.transform.lossyScale.y * this.GetComponent<BoxCollider>().size.y,
            this.transform.lossyScale.z * this.GetComponent<BoxCollider>().size.z
            );

        Vector3 drawBoxposition = this.transform.position + this.GetComponent<BoxCollider>().center;

        Gizmos.matrix = Matrix4x4.TRS(drawBoxposition, this.transform.rotation, drawBoxVector);
        Gizmos.color = new Color(1, 0, 0, 0.4f);
        Gizmos.DrawCube(Vector3.zero, Vector3.one);
        Gizmos.color = Color.red;
        Gizmos.DrawWireCube(Vector3.zero, Vector3.one);
    }
}
```

This is quite a complex script, it is what is called an **Editor script**.
It runs not only when the game runs, but all the time.

If you do this correctly the trigger cube should now look like this



So we now have a cube that we can only see in the scene view, not in the game.

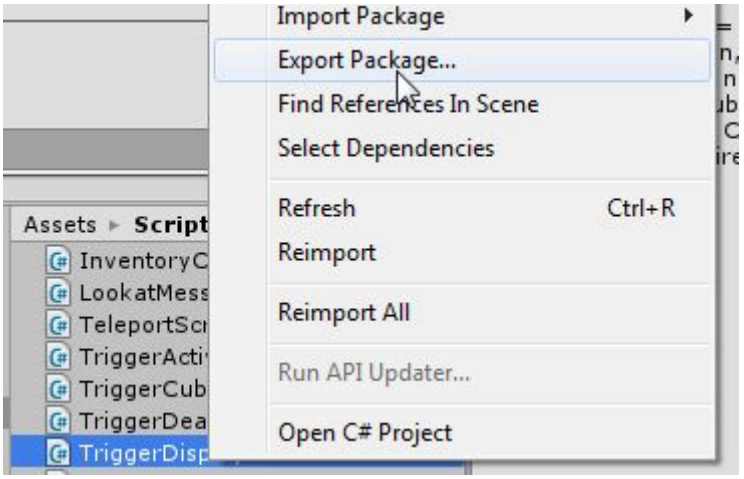
This will be the basis of all the following trigger scripts.

Now we are going to make a series of scripts to do a range of functions.
These used together are a set of powerful tools that can do a huge range of things.

We are going to be using this script over the next few weeks for various games. Because of this we don't want to have to write this over and over again.

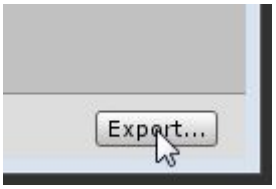
We are going to make a Unity package so we can import it everytime we want to use it.
This is a process that you should do for any handy scripts that you write.

Right click on the **TriggerDisplay** script and select **Export Package...**



With the window that pops up, ensure you untick **include dependencies**

Then click **Export…**



Save the file somewhere that you can get to easily later on. Note that the default location to save it is inside the Unity project, **DON'T SAVE IT HERE.**

Now everytime you want to use this script you can just double click on this Unity package and it will load into your project.

You can store multiple scripts and other assets inside a Unity package. They are very useful.

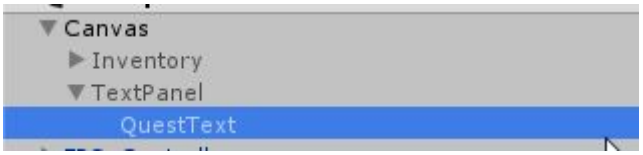
Show a Message

Open up the **UIController** script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class UIController : MonoBehaviour {

    public Text messageText;
    public GameObject messagePanel;
    private float displayTimer;
    private float displayLength;
    private bool isShowingMessage = false;

    //Function that gets called from other scipts, it gets passed a
    //string (What to say) and a float (how long to say it)
    public void ShowMessage(string message, float duration = 3)
    {
        //Set the messagePanel active
        messagePanel.SetActive(true);
        //Change the text to the message that was passed to this function
        messageText.text = message;
        //Set a bool saying that a message is currently being showen
        isShowingMessage = true;
        //Set a timer to remove the message after a set amount of time
        displayLength = duration;
        displayTimer = Time.time;
    }

    // Update is called once per frame
    void Update () {
        //If the script is currently showing a message
        if (isShowingMessage)
        {
            //Check if an amount of time has passed
            if(Time.time - displayTimer > displayLength)
            {
                //Deactivate the panel
                messagePanel.SetActive(false);
                isShowingMessage = false;
            }
        }
    }
}
```

You need to attach the Text Panel and the Quest text items to this script once written

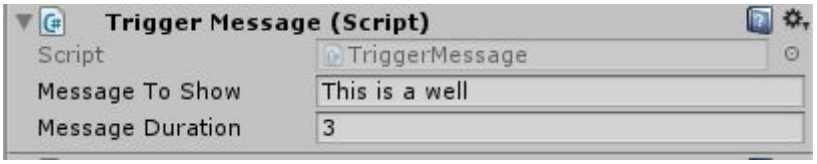
These can be found here



Now we have this set up we can call it from other scripts.
Let create a new script called **TriggerMessage** and attach it to a trigger box

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

[RequireComponent(typeof(TriggerDisplay))]
public class TriggerMessage : MonoBehaviour {

    public string messageToShow = "Default Message";
    public float messageDuration = 3;

    //Runs when the player moves into this trigger
    void OnTriggerEnter(Collider other)
    {
        //Checks if the tag of the object that enters is "Player"
        if (other.tag == "Player")
        {
            GameObject.FindObjectOfType<UIController>().
                ShowMessage(messageToShow, messageDuration);
        }
    }

}
```

You can now update the message displayed



And if you walk through the trigger it will display a message

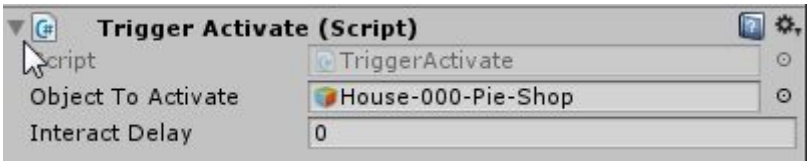
Activate an object

This script will allow us to place an object in the scene, then deactivate it. When the player walks through this trigger then it will appear.

Create a new script called **TriggerActivate** and place it on a trigger cube.

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

[RequireComponent(typeof(TriggerDisplay))]
public class TriggerActivate : MonoBehaviour
{

    public GameObject objectToActivate;
    public float interactDelay = 0;

    //Runs when the player moves into this trigger
    void OnTriggerEnter(Collider other)
    {

        if (other.tag == "Player")
        {
            //Run the function "DeactivateObjet" after [interactDelay] seconds
            Invoke("ActivateObject", interactDelay);
        }
    }

    //Deactivates an object
    private void ActivateObject()
    {
        objectToActivate.SetActive(true);
    }
}
```

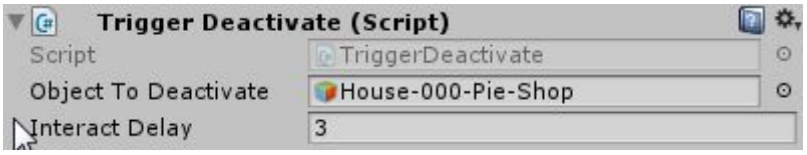
And from the front end we can assign the object to activate and put a delay on it (if any)



While we are at it, let's make a script called **TriggerDeactivate** to do the opposite

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

[RequireComponent(typeof(TriggerDisplay))]
public class TriggerDeactivate : MonoBehaviour {

    public GameObject objectToDeactivate;
    public float interactDelay = 0;

    //Runs when the player moves into this trigger
    void OnTriggerEnter(Collider other)
    {

        if (other.tag == "Player")
        {
                //Run the function "DeactivateObjet" after [interactDelay] seconds
                Invoke("DeactivateObjet", interactDelay);
        }
    }

    //Deactivates an object
    private void DeactivateObjet()
    {
        objectToDeactivate.SetActive(false);
    }
}
```

And it works exactly the same from the inspector



Teleport Self

This is very similar to what we have done before.
Same as before, create a script called **TriggerTeleportSelf** and place it on a triggerbox

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

public class TriggerTeleportSelf : MonoBehaviour {

    public Transform transformToTeleportObjectTo;

    //Runs when the player moves into this trigger
    void OnTriggerEnter(Collider other)
    {

        if (other.tag == "Player")
        {
            if (!transformToTeleportObjectTo)
            {
                Debug.LogWarning("You have not assigned a reference for this teleport script");
            }
            //Run the function "DeactivateObjet" after [interactDelay] seconds
            other.transform.position = transformToTeleportObjectTo.position;
            other.transform.rotation = transformToTeleportObjectTo.rotation;
        }
    }

    //Draws a line from the start to the end point when this trigger is selected
    void OnDrawGizmosSelected()
    {
        if (transformToTeleportObjectTo)
        {
            Debug.DrawLine(transform.position, transformToTeleportObjectTo.transform.position);
        }
    }
}
```

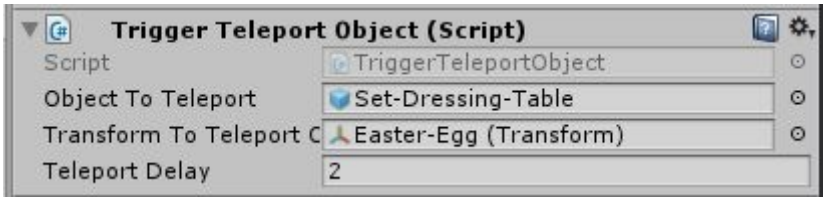
Remember to give it an object to teleport to

Teleport Object

Exactly the same as the previous script but you can move an object from one location to another

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

[RequireComponent(typeof(TriggerDisplay))]
public class TriggerTeleportObject : MonoBehaviour {

    public GameObject objectToTeleport;
    public Transform transformToTeleportObjectTo;
    public float teleportDelay = 0;

    //Runs when the player moves into this trigger
    void OnTriggerEnter(Collider other)
    {

        if (other.tag == "Player")
        {
            //Run the function "DeactivateObjet" after [interactDelay] seconds
            Invoke("TeleportObject", teleportDelay);
        }
    }

    //Moves an object from one point to another
    private void TeleportObject()
    {
        if (!transformToTeleportObjectTo || !objectToTeleport)
        {
            Debug.LogWarning("You have not assigned a reference for this teleport script");
        }

        objectToTeleport.transform.position = transformToTeleportObjectTo.position;
        objectToTeleport.transform.rotation = transformToTeleportObjectTo.rotation;
    }

    //Draws a line from the start to the end point when this trigger is selected
    void OnDrawGizmosSelected()
    {
        if (transformToTeleportObjectTo && objectToTeleport) {
            Debug.DrawLine(transformToTeleportObjectTo.position,objectToTeleport.transform.position);
        }
    }
}
```

Make sure you update the inspector



Play a sound

You know the drill, Script called **TriggerPlaySound** on a trigger cube

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

public class TriggerPlaySound : MonoBehaviour {

    public AudioClip clip;
    public float interactDelay = 0;

    //Runs when the player moves into this trigger
    void OnTriggerEnter(Collider other)
    {

        if (other.tag == "Player")
        {
            //Run the function "DeactivateObjet" after [interactDelay] seconds
            Invoke("PlaySound", interactDelay);
        }
    }

    private void PlaySound()
    {
        AudioSource.PlayClipAtPoint(clip, transform.position);
    }
}
```

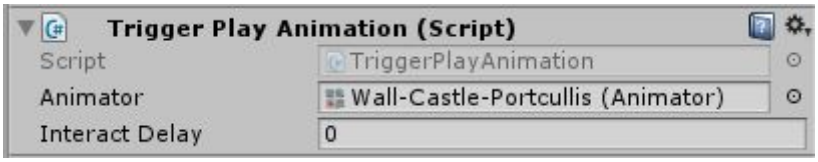
You will need to provide an audio clip, you can find some in the **Townkit -> Sounds**

Play an animation

Script called **TriggerPlayAnimation** on a triggerbox

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

public class TriggerPlayAnimation : MonoBehaviour {

    public Animator animator;
    public float interactDelay = 0;

    //Runs when the player moves into this trigger
    void OnTriggerEnter(Collider other)
    {

        if (other.tag == "Player")
        {
            //Run the function "DeactivateObjet" after [interactDelay] seconds
            Invoke("PlayAnimation", interactDelay);
        }
    }

    private void PlayAnimation()
    {
        animator.SetTrigger("play");
    }
}
```

To make this work you need an object with an animator that has a trigger Parameter called **play.**



We have provided you with one on the **Portcullis**.
Please ask your teacher if you need more information about this.

Advanced Trigger Use

Everything past here is getting a little more complex. You don't need to know any of this but it will help your games out a lot if you at least attempt them and put them in your code library.

Pick up an Item

Lets allow the player to pick up things

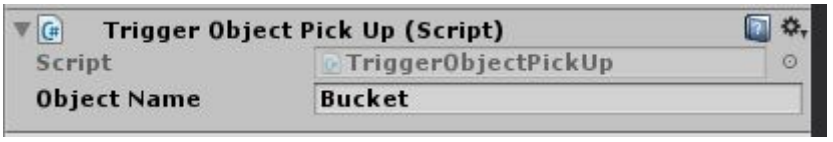
Create a new script called **TriggerObjectPickup** and place it on a object in the scene (perhaps the bucket near the well).

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

[RequireComponent(typeof(TriggerDisplay))]
public class TriggerObjectPickUp : MonoBehaviour {

    public string objectName = "This thing";

    //Runs when the player moves into this trigger
    void OnTriggerEnter(Collider other)
    {
        //Checks if the tag of the object that enters is "Player"
        if (other.tag == "Player")
        {
            GameObject.FindObjectOfType<UIController>().
                ShowMessage("You have picked up a " + objectName, 2);
            gameObject.SetActive(false);
        }
    }

}
```

> *Note: You might notice some magic here. Since we are saying **RequireComponent** in this script it automatically adds the **TriggerDisplay** script. And if you look at the TriggerDisplay script it has a **RequireComponent** for a box collider, and that script also automatically set the box collider to a trigger.*

This is very similar to the previous script, but it makes the object it's attached to disappear

Change the object name to something appropriate



Also you might want to enlarge the box collider to make it east to walk into.

Now if you walk into the bucket you can pick it up.. Kinda.

But now that you have picked it up there's not much you can do with it. Let’s make an inventory for all objects we have picked up.

Create a new script called **InventoryController** and attach it to the **GameController**

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class InventoryController : MonoBehaviour {

    public List<string> inventoryList = new List<string>();

    public GameObject inventoryPanel;
    public Text inventoryText;

    //Add an item to your inventory
    public void AddItem(string itemName)
    {
        inventoryList.Add(itemName);
        UpdateInventory();
    }

    //Check if an item is in your inventory
    public bool CheckItem(string itemName)
    {
        if(itemName == "")
        {
            return true;
        }
        return inventoryList.Contains(itemName);
    }

    //Checks to see if the player has pressed I to open the inventory
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.I))
        {
            UpdateInventory();
            inventoryPanel.SetActive(!inventoryPanel.activeSelf);

        }
    }

    //Loop through all the items in your inventory and write them to screen
    private void UpdateInventory()
    {
        string inventoryString = "";

        foreach (string item in inventoryList)
        {
            inventoryString += item + "\n";
        }

        inventoryText.text = inventoryString;
    }
}
```

Make sure you update the Inspector

You can find the objects here

And update the **TriggerObjectPickup** script

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

[RequireComponent(typeof(TriggerDisplay))]
public class TriggerObjectPickUp : MonoBehaviour {

    public string objectName = "This thing";

    //Runs when the player moves into this trigger
    void OnTriggerEnter(Collider other)
    {
        //Checks if the tag of the object that enters is "Player"
        if (other.tag == "Player")
        {
            GameObject.FindObjectOfType<UIController>().ShowMessage("You have picked up a " + objectName, 2);
            GameObject.FindObjectOfType<InventoryController>().AddItem(objectName);
            gameObject.SetActive(false);
        }
    }

}
```

Now if you play the game you can pick up the Bucket and see it in your inventory (press I)

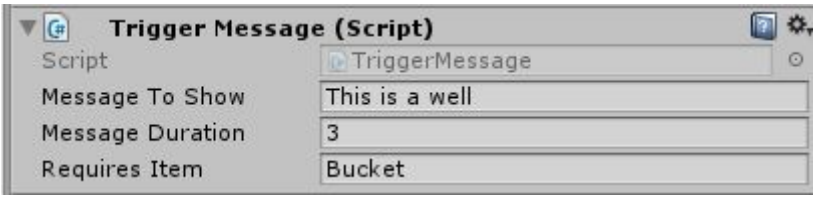
There is not much use if picking up an object if we can't use it.

Open the **TriggerMessage** Script and add a few more lines

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

[RequireComponent(typeof(TriggerDisplay))]
public class TriggerMessage : MonoBehaviour {

    public string messageToShow = "Default Message";
    public float messageDuration = 3;

    public string requiresItem = "";

    //Runs when the player moves into this trigger
    void OnTriggerEnter(Collider other)
    {
        //If this trigger requires an item, then check to see if the player has that item
        if (!GameObject.FindObjectOfType<InventoryController>().CheckItem(requiresItem))
        {
            //If they dont, Stop running this function.
            return;
        }


        //Checks if the tag of the object that enters is "Player"
        if (other.tag == "Player")
```

Now if we check in the inspector we see a new field. If it is empty the script will work as normal.

If it is not empty then it will only work if you have picked up an item with that name



Update previous scripts

Go through all the previous scripts and put the same code in to them to make the scripts work only with a required Item.

Look at Trigger

So far all of these triggers have been based on you walking through a trigger box.
We can also use other ways to trigger event.

This is a Look at triggger that will activate when you look at it.

Create a script called **LookatMessage** and place it on a trigger box

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

public class LookatMessage : MonoBehaviour {

    public float distanceToTrigger = 5;
    public float timeToTrigger = 1;
    [Range(0.01f, 0.2f)]
    public float accuracy = 0.05f;
    public bool activateOnce = true;
    private float timer;

    public string messageToDisplay = "You looked at me!";
    public float messageDuration = 2;

    public string requiresItem = "";

    // Update is called once per frame
    void Update () {

        if (!GameObject.FindObjectOfType<InventoryController>().CheckItem(requiresItem))
        {
            return;
        }

        float distanceToPlayer = Vector3.Distance(Camera.main.transform.position, transform.position);

        if (distanceToPlayer > distanceToTrigger)
        {
            timer = 0;
            return;
        }
```

Continued below

```
        //Get the vector from the players camera forward
        Vector3 playerForward = Camera.main.transform.forward;

        //Get the vector from the object to the player
        Vector3 objectToPlayer = Vector3.Normalize(transform.position - Camera.main.transform.position);

        //Draw debug rays of both vectors
        Debug.DrawRay(Camera.main.transform.position, playerForward * distanceToTrigger);
        Debug.DrawRay(Camera.main.transform.position, objectToPlayer * distanceToTrigger, Color.yellow);

        //If both vectors are alligned you will get a Dot Product of 1
        if (Vector3.Dot(playerForward, objectToPlayer) < 1 - accuracy)
        {
            timer = 0;
            return;
        }

        timer += Time.deltaTime;

        if (timer >= timeToTrigger)
        {
            //Trigger the event you want triggered.
            //Change this line to change the thing you want to trigger. Play sound, Pick Up etc...
            GameObject.FindObjectOfType<UIController>().
             ShowMessage(messageToDisplay, messageDuration);
            timer = 0;
            //Deactivate this script once run
            if (activateOnce)
            {

                this.enabled = false;
            }

        }

    }
```

This will trigger a message on the screen when you look at an item after a certain amount of time.

Distance To Trigger - How far away do you need to be to allow this trigger to work
Time To Trigger - How long you need to look at the object to allow this trigger to work
Accuracy - The accuracy of your direction of facing for this trigger to work

0 = looking EXACTLY at the object (almost imposible)

1 = looking somewhere between looking exactly at the object to looking at 90 degrees away from the object.

Active Once - Deactivate this script once triggered

All the other variables are taken from the **TriggerMessage** script.

Now if you have a look at the comments in the script you should be able to make a copy of this script do activate animations, sounds etc...