Zombie Shooter Project 1d

AINT152

Task 1. Make the player face the mouse

Explanation

- This script will rotate the **player** towards the **mouse pointer** along its **z-axis**

Do this

- In the **Project view**, create a new **C# Script** in the **Scripts Folder**
- Name the Script **MouseSmoothLook2D**

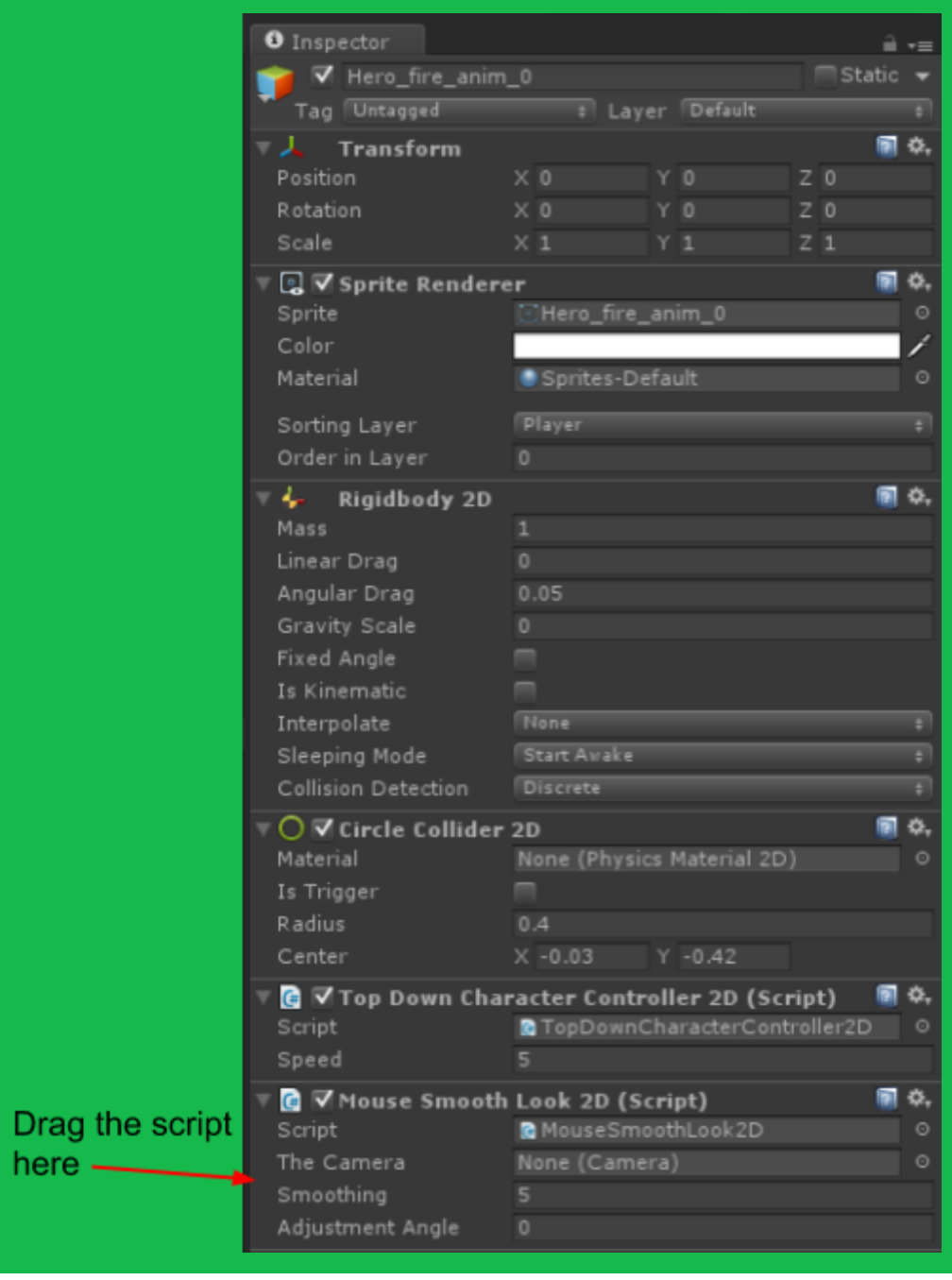
Do this

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

```
using UnityEngine;
using System.Collections;

public class MouseSmoothLook2D : MonoBehaviour {
    public Camera theCamera;
    public float smoothing = 5.0f;
    public float adjustmentAngle = 0.0f;

    void Update(){
        Vector3 target = theCamera.ScreenToWorldPoint( Input.mousePosition );
        Vector3 difference = target - transform.position;

        difference.Normalize();

        float rotZ = Mathf.Atan2( difference.y, difference.x ) * Mathf.Rad2Deg;
        Quaternion newRotation = Quaternion.Euler( new Vector3( 0.0f, 0.0f, rotZ + adjustmentAngle ));
        transform.rotation = Quaternion.Lerp(transform.rotation, newRotation, Time.deltaTime * smoothing);
    }
}
```

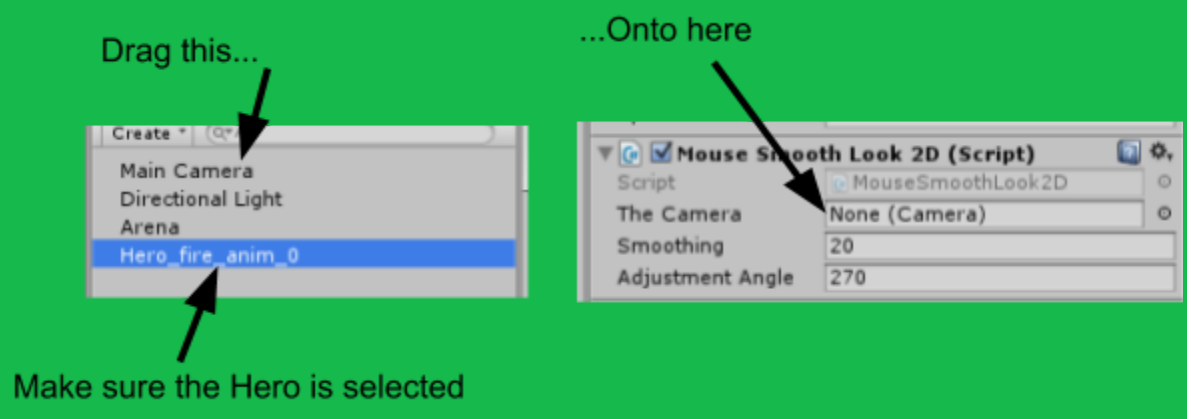
Do this

- In the **Unity Editor**, select the **Hero** GameObject in the **Hierarchy**
- From the **Project view** drag your **MouseSmoothLook2D** script to the **Hero** in the **Inspector**



Do this

- In the **Unity Editor**, select the **Hero** GameObject in the **Hierarchy**
- From the **Hierarchy**, drag the **Main Camera** GameObject onto the **The Camera** property of the **MouseSmoothLook2D** script in the **Inspector**



Explanation - theCamera property

- We need the **camera position** in the scene so we can offset the **mouse position** from it while the **game** is **running**
- **Camera** is a custom class for the **Camera Component**
- **theCamera** is **Editable** in the **Unity Editor**, because it is a **public property**

```
public Camera theCamera;
```

Useful links

- Documentation on **Camera** — Camera

Explanation - smoothing property

- This **property** will control the **smoothness** of **rotation** the **player** uses when **rotating** towards the **mouse pointer**
- smoothing is **Editable** in the **Unity Editor**, because it is a **public property**
- smoothing is a **float**, a **decimal number**

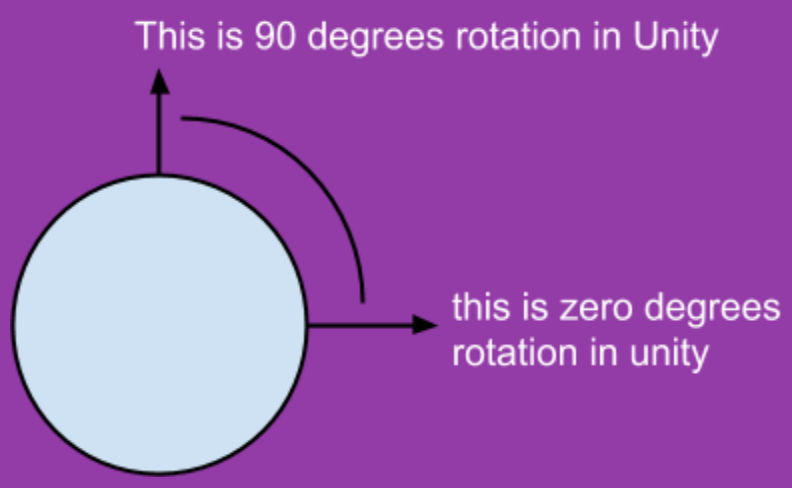
```
public float smoothing = 5.0f;
```

- 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 - adjustmentAngle property

- We need to be able to **compensate** for the **artwork not facing the right way.**
- Unity sees angles as starting from the right:

```
public float adjustmentAngle = 0.0f;
```



- If our artwork were facing up (or 90 degrees according to Unity) we need a way of offsetting this so our artwork faces the right way
- Artwork facing up would need to be turned 3/4 of the way round until it faces to the right

Explanation - code breakdown Part 1

gets the screen position of the mouse pointer
and converts it to a point in the game world

Gets the difference between the player position
and the mouse point in the game world

```
void Update () {
    Vector3 target = theCamera.ScreenToWorldPoint( Input.mousePosition );

    Vector3 difference = target - transform.position;

    difference.Normalize();

    float rotZ = Mathf.Atan2( difference.y, difference.x ) * Mathf.Rad2Deg;

    Quaternion newRotation = Quaternion.Euler( new Vector3( 0.0f, 0.0f, rotZ + adjustmentAngle ));

    transform.rotation = Quaternion.Lerp( transform.rotation, newRotation, Time.deltaTime * smoothing );
}
```

Explanation - code breakdown Part 2

calculates the angle of the difference between the mouse pointer and the player in the game world

```
void Update () {
    Vector3 target = theCamera.ScreenToWorldPoint( Input.mousePosition );

    Vector3 difference = target - transform.position;

    difference.Normalize();

    float rotZ = Mathf.Atan2( difference.y, difference.x ) * Mathf.Rad2Deg;

    Quaternion newRotation = Quaternion.Euler( new Vector3( 0.0f, 0.0f, rotZ + adjustmentAngle ));

    transform.rotation = Quaternion.Lerp( transform.rotation, newRotation, Time.deltaTime * smoothing );
}
```

Creates a new rotation that includes our new calculated angle and the adjustment angle

Sets our new rotation by animating to it with a Lerp (Linear Interpolation)

Explanation - Line 1

- We **create** a new **Vector3** variable
- We use the **Camera's ScreenToWorldPoint()** method to **convert** our **mouse position** (**Input.mousePosition**) into a **Vector3** point in our **game world**

```
void Update(){
    Vector3 target = theCamera.ScreenToWorldPoint( Input.mousePosition );
    Vector3 difference = target - transform.position;

    difference.Normalize();

    float rotZ = Mathf.Atan2( difference.y, difference.x ) * Mathf.Rad2Deg;
    Quaternion newRotation = Quaternion.Euler( new Vector3( 0.0f, 0.0f, rotZ + adjustmentAngle ));
    transform.rotation = Quaternion.Lerp(transform.rotation, newRotation, Time.deltaTime * smoothing);
}
```

Useful links

- Documentation on **Camera.ScreenToWorldPoint** — Camera.ScreenToWorldPoint
- Documentation on **Input.mousePosition** — Input.mousePosition

Explanation - Line 2

- We create a new Vector3 variable
- We get the **difference** between the **player position** and the **target variable** we just created and store that

```
void Update(){
    Vector3 target = theCamera.ScreenToWorldPoint( Input.mousePosition );
    Vector3 difference = target - transform.position;

    difference.Normalize();

    float rotZ = Mathf.Atan2( difference.y, difference.x ) * Mathf.Rad2Deg;
    Quaternion newRotation = Quaternion.Euler( new Vector3( 0.0f, 0.0f, rotZ + adjustmentAngle ));
    transform.rotation = Quaternion.Lerp(transform.rotation, newRotation, Time.deltaTime * smoothing);
}
```

Useful links

- Documentation on **Vector3 operator -** — Vector3.operator -

Explanation - Line 3

- We then normalize the difference variable
- This will keep the **direction** the difference variable is pointing in, but reduce down the size of the values
- We **don't** need to **assign** this **value**, as it changes it internally

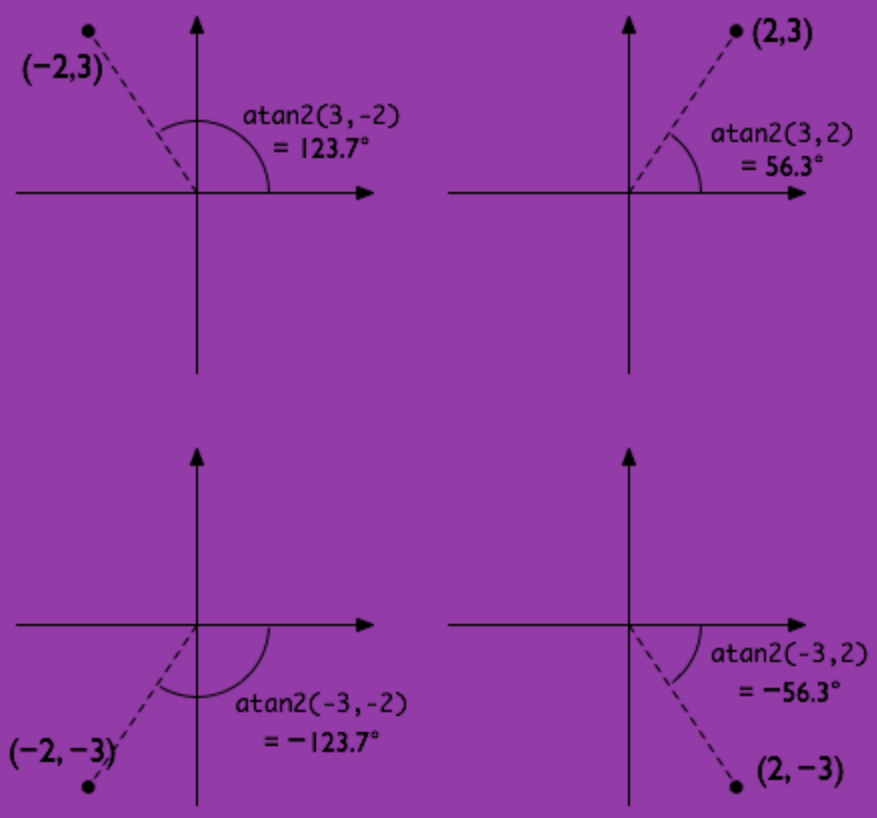
```
void Update(){
    Vector3 target = theCamera.ScreenToWorldPoint( Input.mousePosition );
    Vector3 difference = target - transform.position;

    difference.Normalize();

    float rotZ = Mathf.Atan2( difference.y, difference.x ) * Mathf.Rad2Deg;
    Quaternion newRotation = Quaternion.Euler( new Vector3( 0.0f, 0.0f, rotZ + adjustmentAngle ));
    transform.rotation = Quaternion.Lerp(transform.rotation, newRotation, Time.deltaTime * smoothing);
}
```

Useful links

- Documentation on **Vector3.Normalize** — Vector3.Normalize

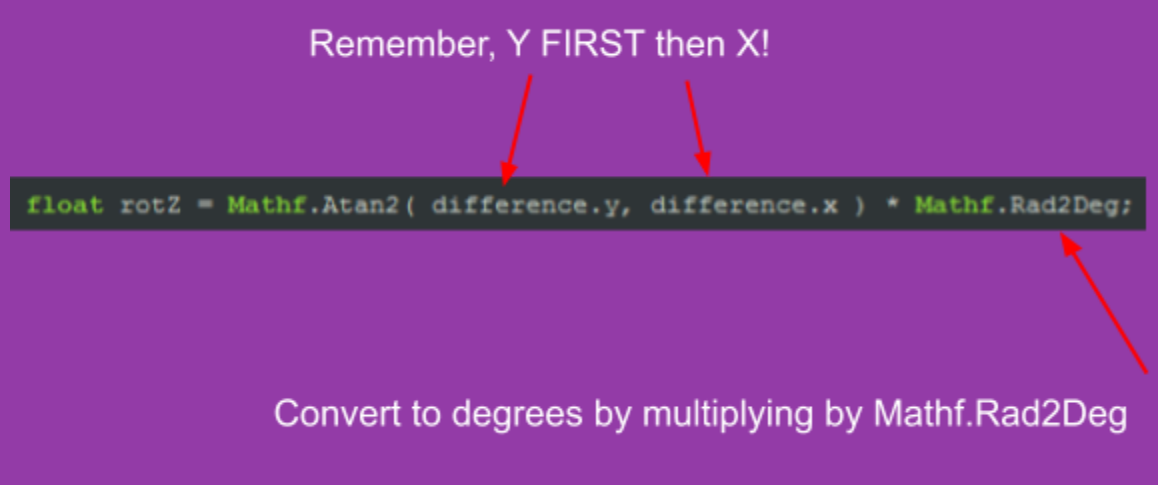
Explanation - Line 4

- We create a **float variable** to store our **z-axis**
- We use the **mathf.Atan2()** method to get the angle our difference variable is facing
- This works by assuming our starting x and y are both 0
- Then, by using **trigonometry** we can **input** the **x** and **y** of the **difference variable** to work out which way it's facing **compared** to the **starting point**



- The **Mathf.Atan2()** method will **return** an **angle** in **radians**, we will need to **convert** it to **degrees**
- We can do this using the **Mathf.Rad2Deg** property

```
void Update(){
    Vector3 target = theCamera.ScreenToWorldPoint( Input.mousePosition );
    Vector3 difference = target - transform.position;

    difference.Normalize();

    float rotZ = Mathf.Atan2( difference.y, difference.x ) * Mathf.Rad2Deg;
    Quaternion newRotation = Quaternion.Euler( new Vector3( 0.0f, 0.0f, rotZ + adjustmentAngle ));
    transform.rotation = Quaternion.Lerp(transform.rotation, newRotation, Time.deltaTime * smoothing);
}
```



Useful links

- **What are radians?** — Math is fun - radians
- Documentation on **Mathf.Atan2** — Mathf.Atan2
- Documentation on **Mathf.Rad2Deg** — Mathf.Rad2Deg

Explanation - Line 5

- We create a new **Quaternion** (deals with rotations)
- We use **Quaternion.Euler** to work with **degrees**, not **radians**
- We create a new **Vector3** INSIDE the **Quaternion.Euler** method as a **parameter**
- We can **assign** our new **z-axis** from the **mouse pointer**, with our **adjustment angle** in here

```
void Update(){
    Vector3 target = theCamera.ScreenToWorldPoint( Input.mousePosition );
    Vector3 difference = target - transform.position;

    difference.Normalize();

    float rotZ = Mathf.Atan2( difference.y, difference.x ) * Mathf.Rad2Deg;
    Quaternion newRotation = Quaternion.Euler( new Vector3( 0.0f, 0.0f, rotZ + adjustmentAngle ));
    transform.rotation = Quaternion.Lerp(transform.rotation, newRotation, Time.deltaTime * smoothing);
}
```

Note X and Y are set to zero

Z is set to our mouse pointer angle plus our adjustment angle

```
Quaternion newRotation = Quaternion.Euler( new Vector3( 0.0f, 0.0f, rotZ + adjustmentAngle ));
```

Useful links

- Documentation on **Quaternion** — Quaternion
- Documentation on **Quaternion.Euler** — Quaternion.Euler

Explanation - Line 6

- Here we set our **player rotation**!
- We use a **Lerp** to **smoothly animate** towards our desired **angle**
- We use **Time.DeltaTime** to calculate how fast to **animate** (this is common inside an **Update()** method)
- Note our **smoothing** property is used to **speed up** or **slow down** the **value** from **Time.deltaTime**

```
void Update(){
    Vector3 target = theCamera.ScreenToWorldPoint( Input.mousePosition );
    Vector3 difference = target - transform.position;

    difference.Normalize();

    float rotZ = Mathf.Atan2( difference.y, difference.x ) * Mathf.Rad2Deg;
    Quaternion newRotation = Quaternion.Euler( new Vector3( 0.0f, 0.0f, rotZ + adjustmentAngle ));
    transform.rotation = Quaternion.Lerp(transform.rotation, newRotation, Time.deltaTime * smoothing);
}
```

Animate from here

Animate to here

Over this time

```
transform.rotation = Quaternion.Lerp( transform.rotation, newRotation, Time.deltaTime * smoothing );
```

Useful links

- Documentation on **Transform.rotation** — Transform.rotation
- Documentation on **Quaternion.Lerp** — Quaternion.Lerp
- Documentation on **Time.deltaTime** — Time.deltaTime
