Zombie Shooter Project 2e

Task 1. Make the Zombie move towards the player

Do this

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

Do this

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

```
using UnityEngine;

public class MoveTowardsObject : MonoBehaviour {

    public Transform target;
    public float speed = 5.0f;

    private void Update() {
        if( target != null ) {
            transform.position = Vector3.MoveTowards( transform.position, target.position, speed * 0.01f );
        }
    }
}
```

Explanation - target property

- The **Transform Component** of the **GameObject** we want to **move towards**
- **target** is a public property so it is **Editable** in the **Unity Editor**
- **target** is a **type** of **Transform**

```
public Transform target;
```

Useful links

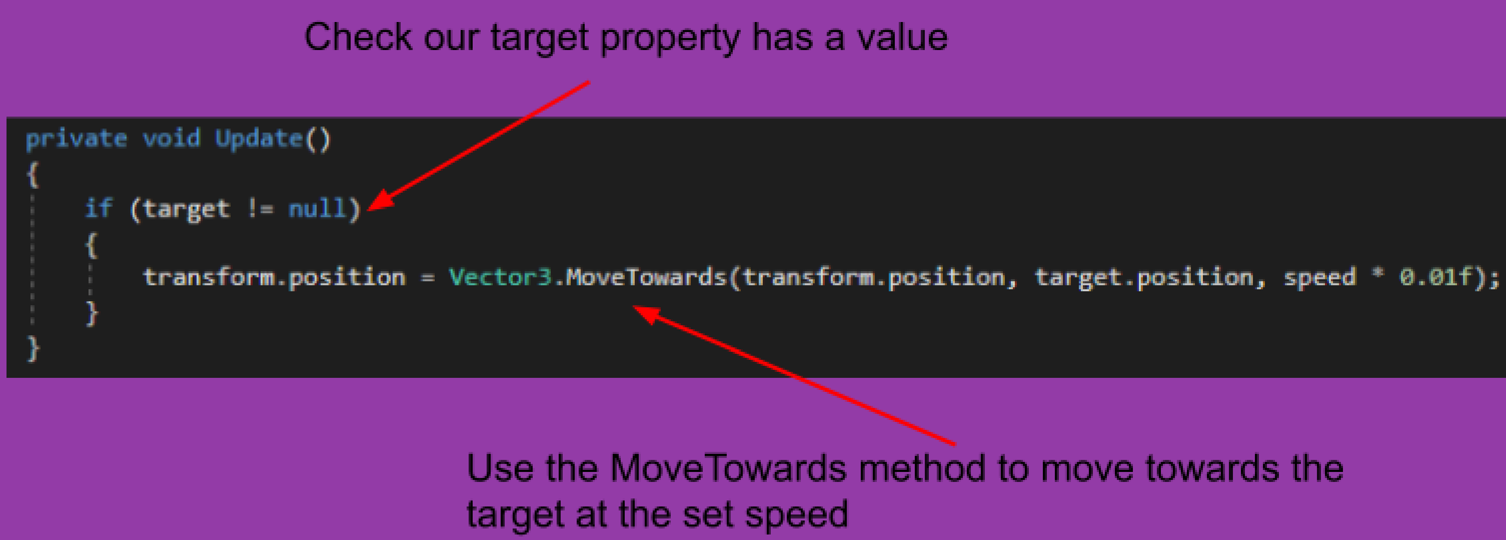
- More information about **Transform** [Transform]

Explanation - speed property

- The **speed** we will **move towards** our **target**
- **speed** is a **public property** so it is **Editable** in the **Unity Editor**
- **speed** is a **type** of **float**

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

Explanation - code breakdown



Explanation - Line 1

- We check the **target** property has been set
- We use the **!= operator** to check the **target** is **not null** (it has been set)

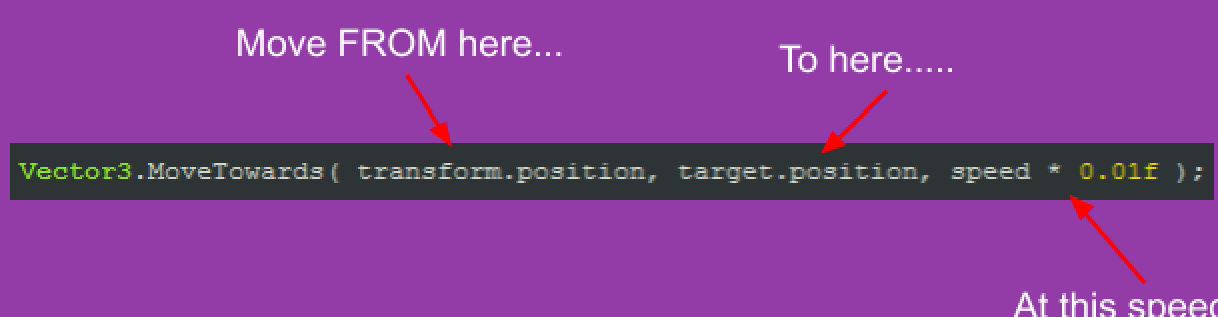
```
private void Update() {
    if( target != null ) {
        transform.position = Vector3.MoveTowards( transform.position, target.position, speed * 0.01f );
    }
}
```

Useful links

- More information about **!= operator** [!= operator]

Explanation - Line 2

- If the **target** is **set**, we can **move towards** the **target**!
- first, we get the **position** of our **GameObject**, using **transform.position**
- then, we use the **Vector3.MoveTowards** method to to our moving
- The **Vector3.MoveTowards** method takes **3 parameters**:



- Note: The **3rd parameter** is the **speed**, but it has been **reduced** by **multiplying** by a **small number**, this is a **small fix** for **easy use** in the **Unity Editor**
- This is because we want to use **larger numbers** for **speed** (1, 5, 10 etc) otherwise we would be using **tiny numbers** (0.0001f etc)

```
speed * 0.01f;
```

```
private void Update() {
    if( target != null ) {
        transform.position = Vector3.MoveTowards( transform.position, target.position, speed * 0.01f );
    }
}
```

Useful links

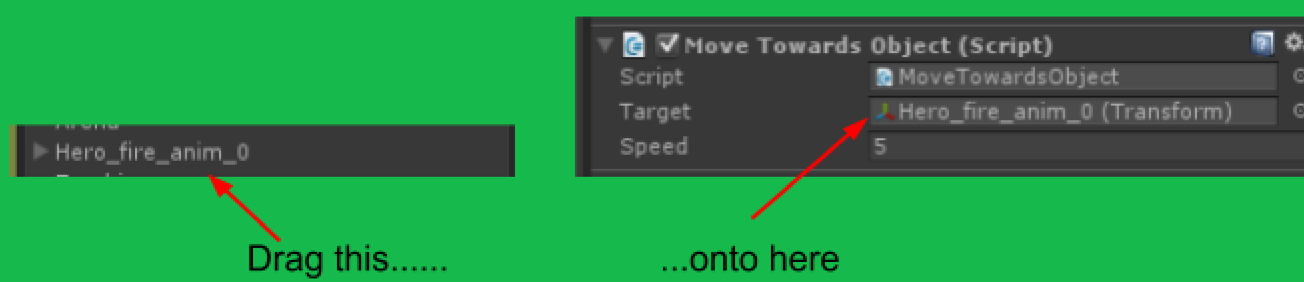
- More information about **Transform.position** [Transform.position]
- More information about **Vector3.MoveTowards** [Vector3.MoveTowards]

Do this

- In the **Unity Editor**, select the **MoveTowardsObject** script in the **Project view**
- **Drag** the **MoveTowardsObject** script onto the **Zombie** GameObject in the **Hierarchy**

Do this

- Select the **Zombie** GameObject in the **Hierarchy**
- Drag the **Hero** GameObject onto the **target** inlet of the **MoveTowardsObject** Component



Task 2. Make the Zombie face the player at all times

Do this

- In the **Project view**, create a new **C# Script** in the **Scripts Folder**
- Name the Script **SmoothLookAtTarget2D**

Do this

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

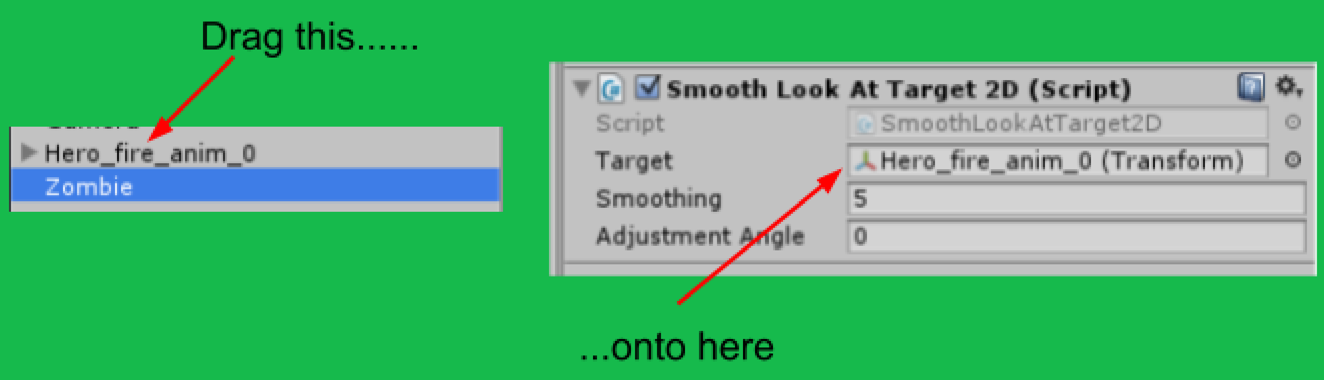
```
using UnityEngine;

public class SmoothLookAtTarget2D : MonoBehaviour {

    public Transform target;
    public float smoothing = 5.0f;
    public float adjustmentAngle = 0.0f;

    private void Update() {
        if( target != null ) {

            Vector3 difference = target.position - transform.position;

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

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

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

Do this

- In the **Unity Editor**, select the **SmoothLookAtTarget2D** script in the **Project view**
- **Drag** the **SmoothLookAtTarget2D** script onto the **Zombie** GameObject in the **Hierarchy**

Do this

- Select the **Zombie** GameObject in the **Hierarchy**
- Drag the **Hero** GameObject onto the **target** inlet of the **SmoothLookAtTarget2D** Component



Explanation - target property

- The **Transform Component** of the **GameObject** we want to **rotate towards**
- **target** is a **public property** so it is **Editable** in the **Unity Editor**
- **target** is a **type** of **Transform**

```
public Transform target;
```

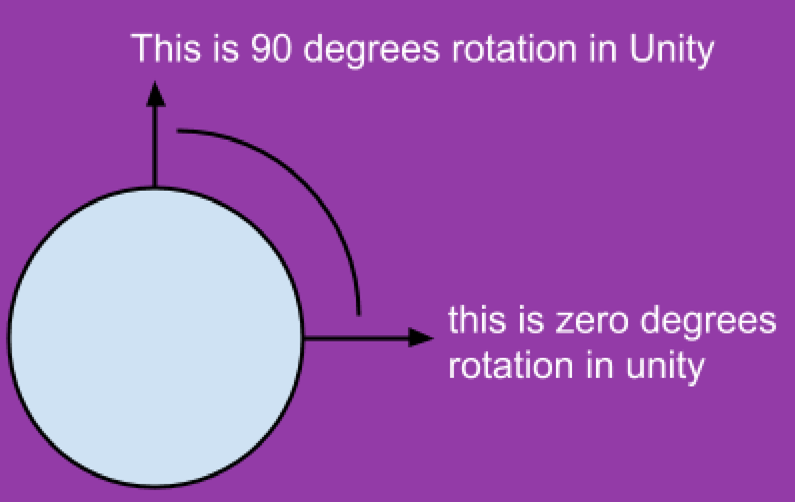
Explanation - smoothing property

- This **property** will control the **smoothness** of **rotation** the used when **rotating** towards the **target**
- **smoothing** is a **public property** so it is **Editable** in the **Unity Editor**
- **smoothing** is a float, a decimal number

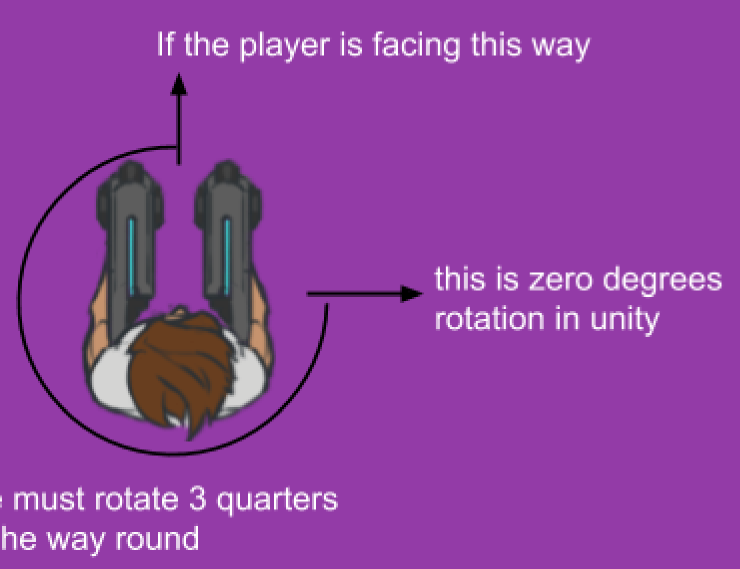
```
public float smoothing = 5.0f;
```

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:



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




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

Explanation - Our custom Update code

- Our custom Update method code looks like this

```
private void Update() {
    if( target != null ) {

        Vector3 difference = target.position - transform.position;

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

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

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

Explanation - Our custom Update code

Check our target has been set

get the difference between our position and the target position

```
private void Update()
{
    if (target != null)
    {
        Vector3 difference = target.position - transform.position;

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

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

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

get our z-axis rotation using Atan2()

animate to the new rotation using a Lerp

Create a new rotation adding our new z-axis and the adjustmentAngle

Explanation - Line 1

- We check the **target** property has been set
- We use the **!= operator** to check the **target** is **not null** (it has been set)

```
private void Update() {
    if( target != null ) {

        Vector3 difference = target.position - transform.position;

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

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

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

Useful links

- More information about **!= operator** [!= operator]

Explanation - Line 2

- We create a new **Vector3** variable
- We get the **difference** between the **target position** and the **transform position** (our current position!)

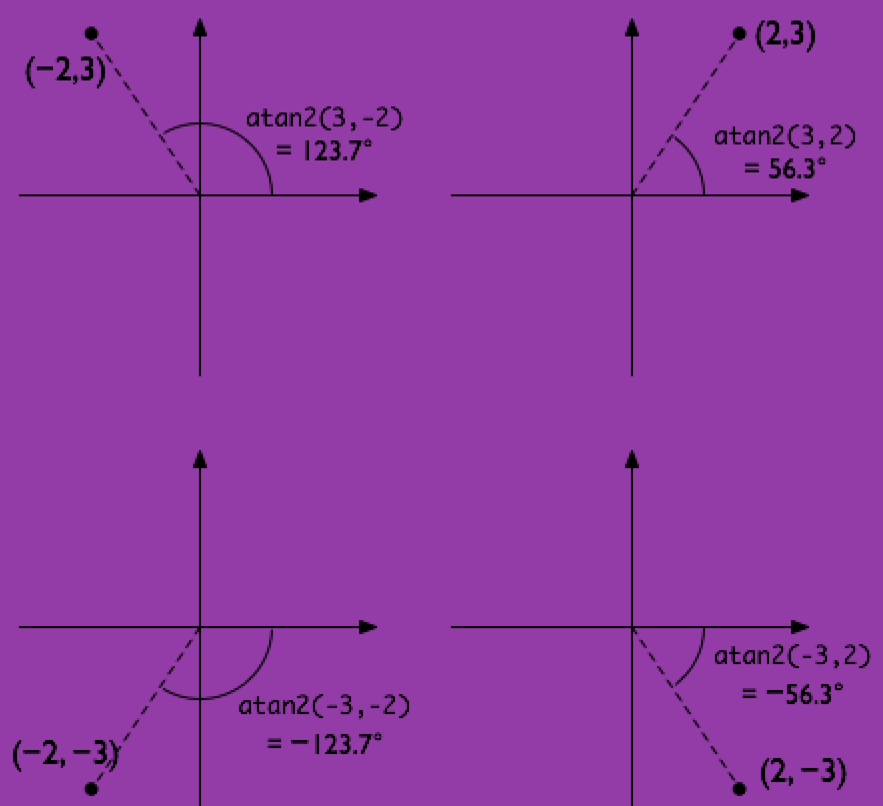
```
private void Update() {
    if( target != null ) {

        Vector3 difference = target.position - transform.position;

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

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

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

Useful links

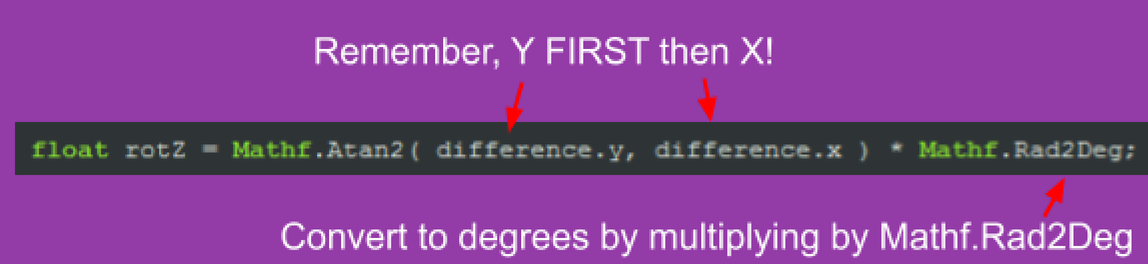
- More information about **Vector3.operator -** [Vector3.operator -]

Explanation - Line 3

- 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




```
private void Update() {
    if( target != null ) {

        Vector3 difference = target.position - transform.position;

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

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

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

Useful links

- More information about **Mathf.Atan2** [Mathf.Atan2]
- More information about **Mathf.Rad2Deg** [Mathf.Rad2Deg]

Explanation - Line 4

- 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 **target position**, with our **adjustment angle** in here

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

Note X and Y are set to zero

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

```
private void Update() {
    if( target != null ) {

        Vector3 difference = target.position - transform.position;

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

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

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

Useful links

- More information about **Quaternion.Euler** — [Quaternion.Euler]
- More information about **Vector3** — [Vector3]

Explanation - Line 5

- Here we set our **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**

Animate from here

Animate to here

Over this time

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

```
private void Update() {
    if( target != null ) {

        Vector3 difference = target.position - transform.position;

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

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

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

Useful links

- More information about **Transform.rotation** — [Transform.rotation]
- More information about **Quaternion.Lerp** — [Quaternion.Lerp]
- More information about **Time.deltaTime** — [Time.deltaTime]