Pathfinding setup

Task 1. Import the Pathfinding toolkit into your project

Do this

- Go to the DLE web page for AINT155
- In the Week 8 folder, download the .zip file called **Pathfinding toolkit**
- Unzip the files into your project

Task 2. Setup the Pathfinder prefab

Explanation

- We want to have pathfinding in our scene for our zombies to move around obstacles and chase the player
- We need the **Pathfinder** prefab in the **Pathfinding toolkit** folder
- It requires a little setup!

Do this

- Drag the **Pathfinder** prefab from the **Pathfinding toolkit** folder of the **Project view** into the scene

Do this

- Select the **Pathfinder** GameObject in the **Hierarchy**
- Set the Transform Position to X = 0, Y = 0 and Z = 0
- Set the Transform Rotation to X = 0, Y = 0 and Z = 0

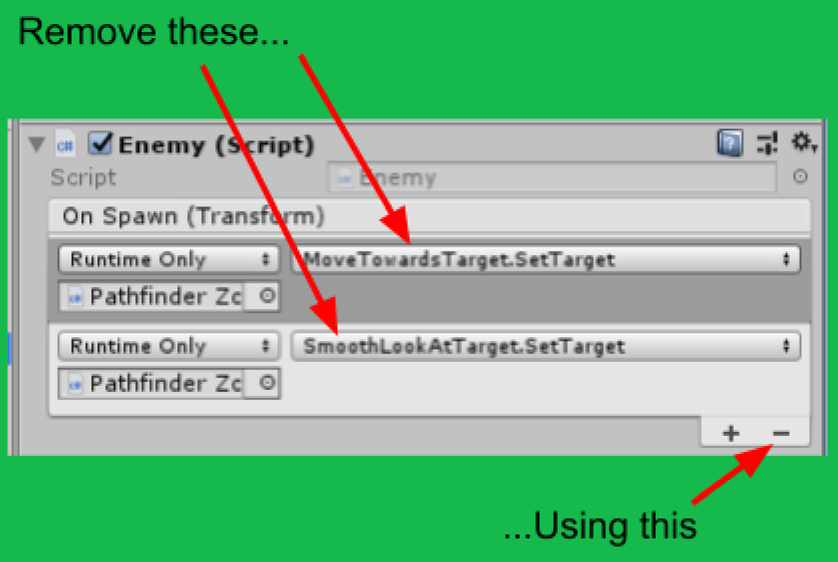
Task 3. Setup the zombie prefab for pathfinding

Explanation

- Now we setup our Zombie for pathfinding
- The Zombie already has some movement and looking components, we need to remove them and their connections to events
- We have 3 components to add and a custom script to make
- Then we wire the new event in and we are good to go!

Do this

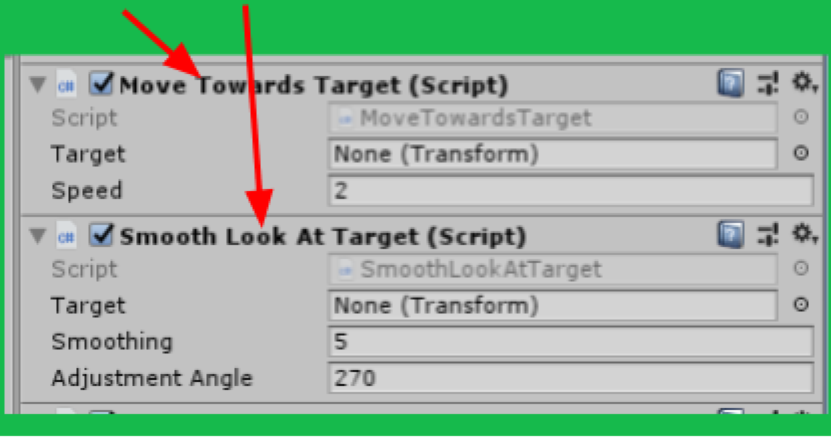
- Select your **Zombie** prefab in the **Project view**
- On the **Enemy** component in the Inspector, remove the following events using the "-" button
  - **MoveTowardsTarget.SetTarget**
  - **SmoothLookAtTarget.SetTarget**



Do this

- With your **Zombie** prefab still selected in the **Project view**
- Remove the Following components:
  - **MoveTowardsTarget**
  - **SmoothLookAtTarget**

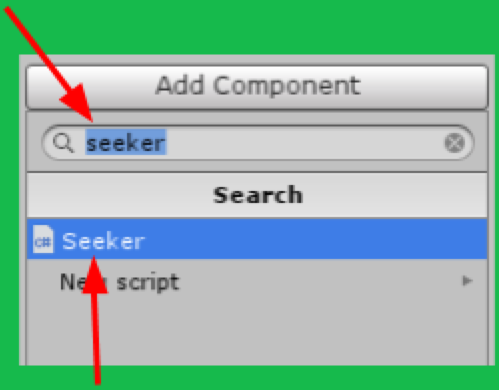
Right-click on name and select Remove Component



Do this

- With your **Zombie** prefab still selected in the **Project view**
- Click the **Add Component** button at the bottom of the **Inspector**
- Type **Seeker** into the search field
- Click the **Seeker** script when it appears

Type "seeker"

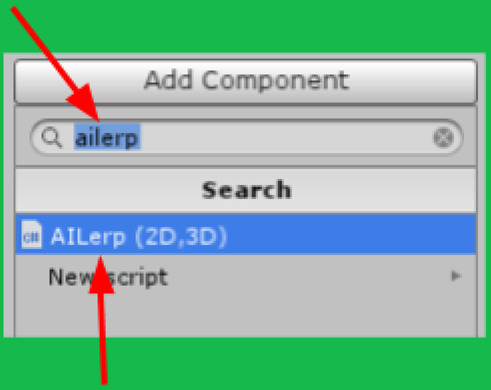


Click script to add component

Do this

- With your **Zombie** prefab still selected in the **Project view**
- Click the **Add Component** button at the bottom of the **Inspector**
- Type **ailerp** into the search field
- Click the **AILerp** script when it appears

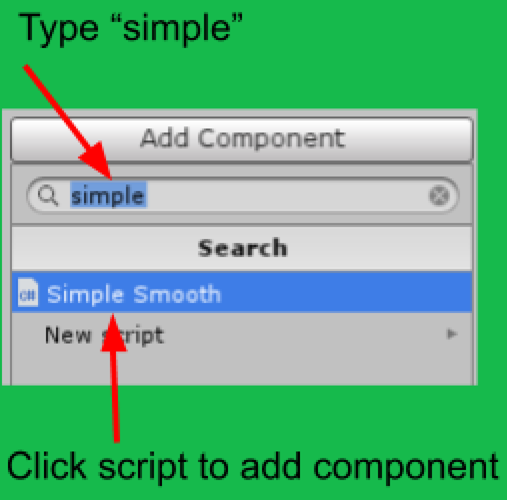
Type "ailerp"



Click script to add component

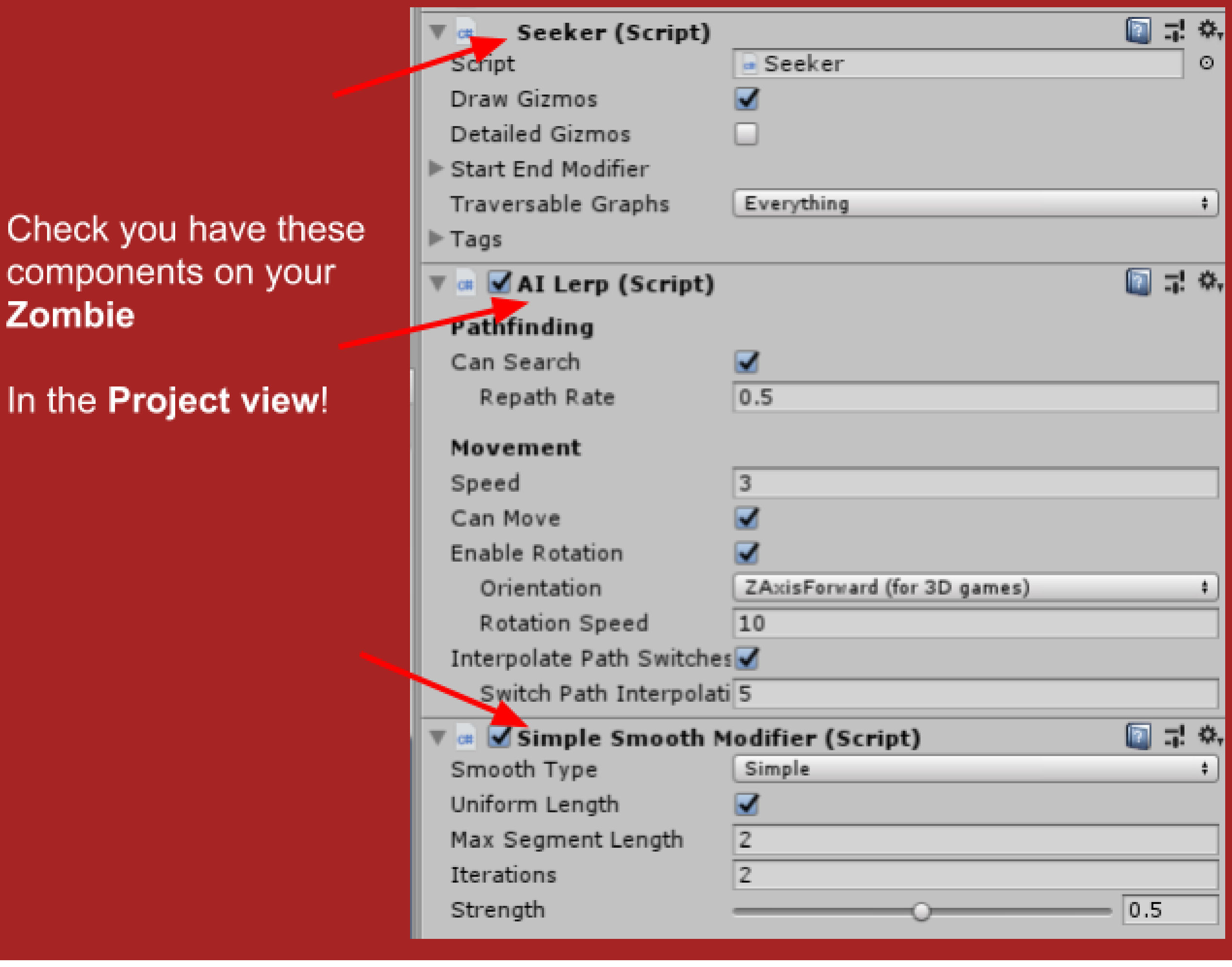
Do this

- With your **Zombie** prefab still selected in the **Project view**
- Click the **Add Component** button at the bottom of the **Inspector**
- Type **simple** into the search field
- Click the **Simple Smooth** script when it appears



Check this

- Check your **Zombie** Prefab in the **Project view** has the following components


Check you have these components on your **Zombie**
In the **Project view**!
Seeker (Script)
AI Lerp (Script)
Simple Smooth Modifier (Script)

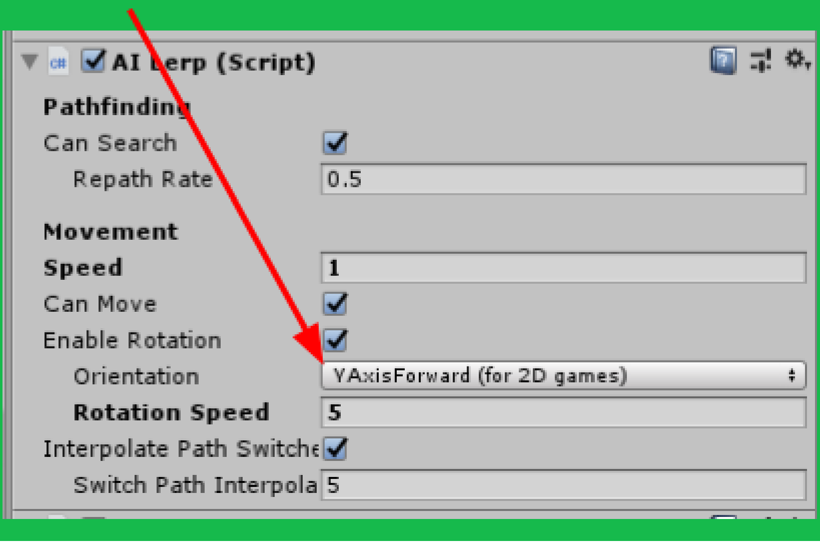
Seeker (Script)
Script: Seeker
Draw Gizmos: ✓
Detailed Gizmos: ☐
Start End Modifier
Traversable Graphs: Everything
Tags

AI Lerp (Script)
Pathfinding
Can Search: ✓
Repath Rate: 0.5
Movement
Speed: 3
Can Move: ✓
Enable Rotation: ✓
Orientation: ZAxisForward (for 3D games)
Rotation Speed: 10
Interpolate Path Switches: ✓
Switch Path Interpolati: 5

Simple Smooth Modifier (Script)
Smooth Type: Simple
Uniform Length: ✓
Max Segment Length: 2
Iterations: 2
Strength: 0.5

Do this

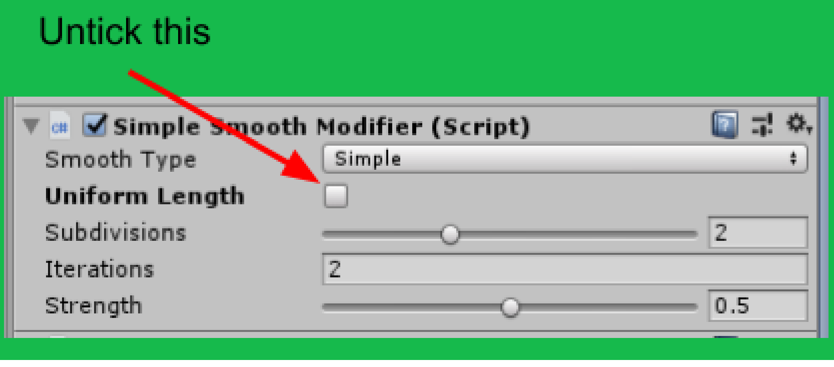
- With your **Zombie** prefab still selected in the **Project view**
- On the **AILerp** component, set the Orientation to **YAxisForward (for 2D games)**



AI Lerp (Script)
Pathfinding
Can Search: ✓
Repath Rate: 0.5
Movement
Speed: 1
Can Move: ✓
Enable Rotation: ✓
Orientation: YAxisForward (for 2D games)
Rotation Speed: 5
Interpolate Path Switche: ✓
Switch Path Interpola: 5

Do this

- With your **Zombie** prefab still selected in the **Project view**
- On the **Simple Smooth Modifier** component, **unitck** the **Uniform Length**



Smooth Type: Simple
Uniform Length: ☐
Subdivisions: 2
Iterations: 2
Strength: 0.5

Task 4. Custom zombie pathfinder script

Explanation

- Now our Zombie prefab has the components for pathfinding, we want to find the player when it spawns in the scene
- We already have this functionality in the Enemy component, we just need to wire it up to the new pathfinding components
- We will create a small script to deal with initialising the pathfinding and getting the player Transform

Do this

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

Do this

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

```
using UnityEngine;
using Pathfinding;

public class EnemyPathFinder : MonoBehaviour {

    public Transform target;
    private IAstarAI ai;

    private void Start() {
        ai = GetComponent<IAstarAI>();
    }

    private void Update () {
        if (target != null && ai != null) {
            ai.destination = target.position;
            ai.SearchPath();
        }
    }

    public void SetTarget(Transform newTarget) {
        target = newTarget;
    }
}
```

Explanation - Pathfinding library

- To use the components in the pathfinding library, we need to import the library
- NOTE: this is not an official Unity supported library! You won't find documentation on the Unity website for it!
- **Links will be provided below to use the library - IF AVAILABLE**

```
using Pathfinding;
```

Useful links

- More information about **Pathfinding in 2D** Pathfinding in 2D

Explanation - target property

- The **Transform Component** of the **GameObject** we want to **pathfind towards**
- **target** is a **type** of **Transform**

```
public Transform target;
```

Useful links

- More information about **Transform** Transform - Scripting Reference

Explanation - ai property

- The **ai** property is a reference to the **AILerp** component
- NOTE: this is a reference to an **interface**, called **IAStartAI**

```
private IAstarAI ai;
```

Explanation - Start method

- **Start** is a method provided by Monobehaviour
- We want to get our **IAstarAI** component and store it in the **ai** property
- The **ai** property will be our way of telling the zombie to pathfind

```
private void Start() {
    ai = GetComponent<IAstarAI>();
}
```

Useful links

- More information about **Start** Start - Scripting Reference

Explanation - Line 1

- Store the IAstartAI component using GetComponent

```
private void Start() {
    ai = GetComponent<IAstarAI>();
}
```

Useful links

- More information about **GetComponent** GetComponent - Scripting Reference

Explanation - Update method

- **Update** is a method provided by Monobehaviour
- We will check we have a **target** and an **ai** component to do pathfinding with
- First we give the **ai** a **destination** (where the **target** is)
- Then we tell the **ai** to **search** for a **path** to the **destination**

```
private void Update () {
    if (target != null && ai != null) {
        ai.destination = target.position;
        ai.SearchPath();
    }
}
```

Useful links

- More information about **Update** Update - Scripting Reference

Explanation - Line 1

- We check if the **zombie** has a **target** and if it has an **ai** component (of type **IAstarAI**)

```
private void Update () {
    if (target != null && ai != null) {
        ai.destination = target.position;
        ai.SearchPath();
    }
}
```

Explanation - Line 2

- We set the ai destination property to the position of the target, using target.position

```
private void Update () {
    if (target != null && ai != null) {
        ai.destination = target.position;
        ai.SearchPath();
    }
}
```

Useful links

- More information about **Transform.position** [Transform.position - Scripting Reference]

Explanation - Line 3

- We tell the **ai** to perform a search for a path to the target using **SearchPath**
- **SearchPath** is a custom method of the **IAstarAI** interface

```
private void Update () {
    if (target != null && ai != null) {
        ai.destination = target.position;
        ai.SearchPath();
    }
}
```

Useful links

- More information about **SearchPath** [SearchPath]

Explanation - SetTarget method

- **SetTarget** will be called by our **Enemy** component, which will give us the player **Transform**
- When we get the **player** Transform, we can find a path to it!

```
public void SetTarget(Transform newTarget) {
    target = newTarget;
}
```

Explanation - Line 1

- Set the public property, **target** to the parameter, **newTarget**

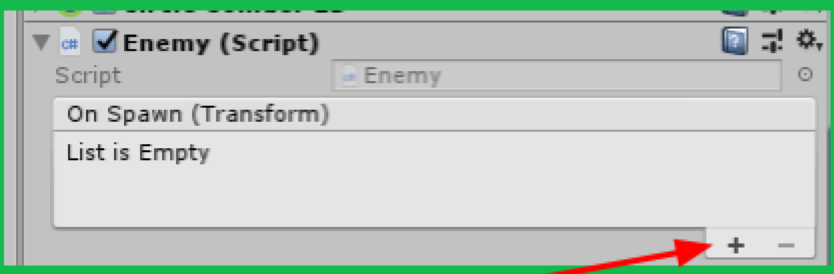
```
public void SetTarget(Transform newTarget) {
    target = newTarget;
}
```

Do this

- In the **Project view**, Add the **EnemyPathFinder** script to the **Zombie** prefab

Do this

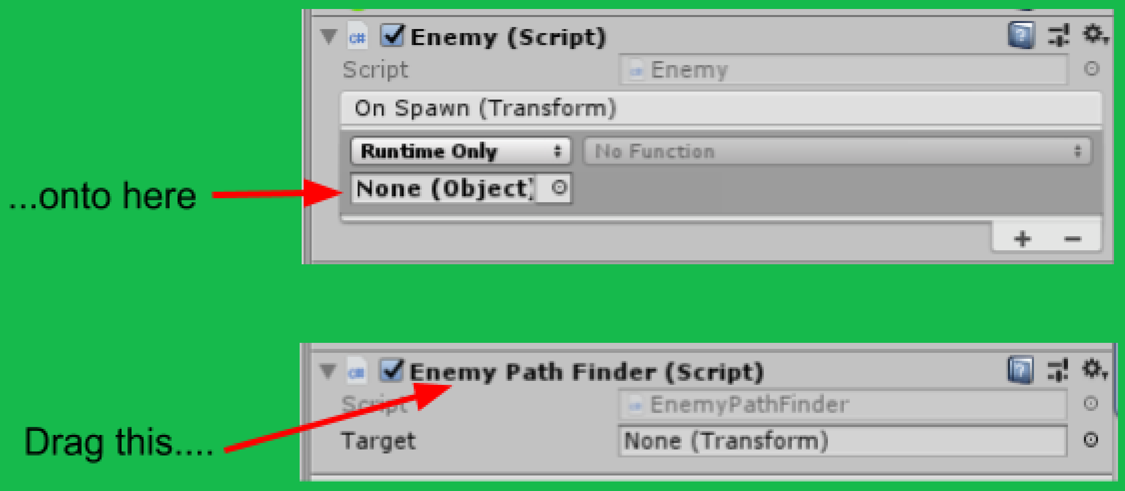
- In the **Project view**, with the **Zombie** prefab selected:
- Add a **listener** to the **On Spawn** event on the **Enemy** component, using the "+" button



Click this to add a new listener

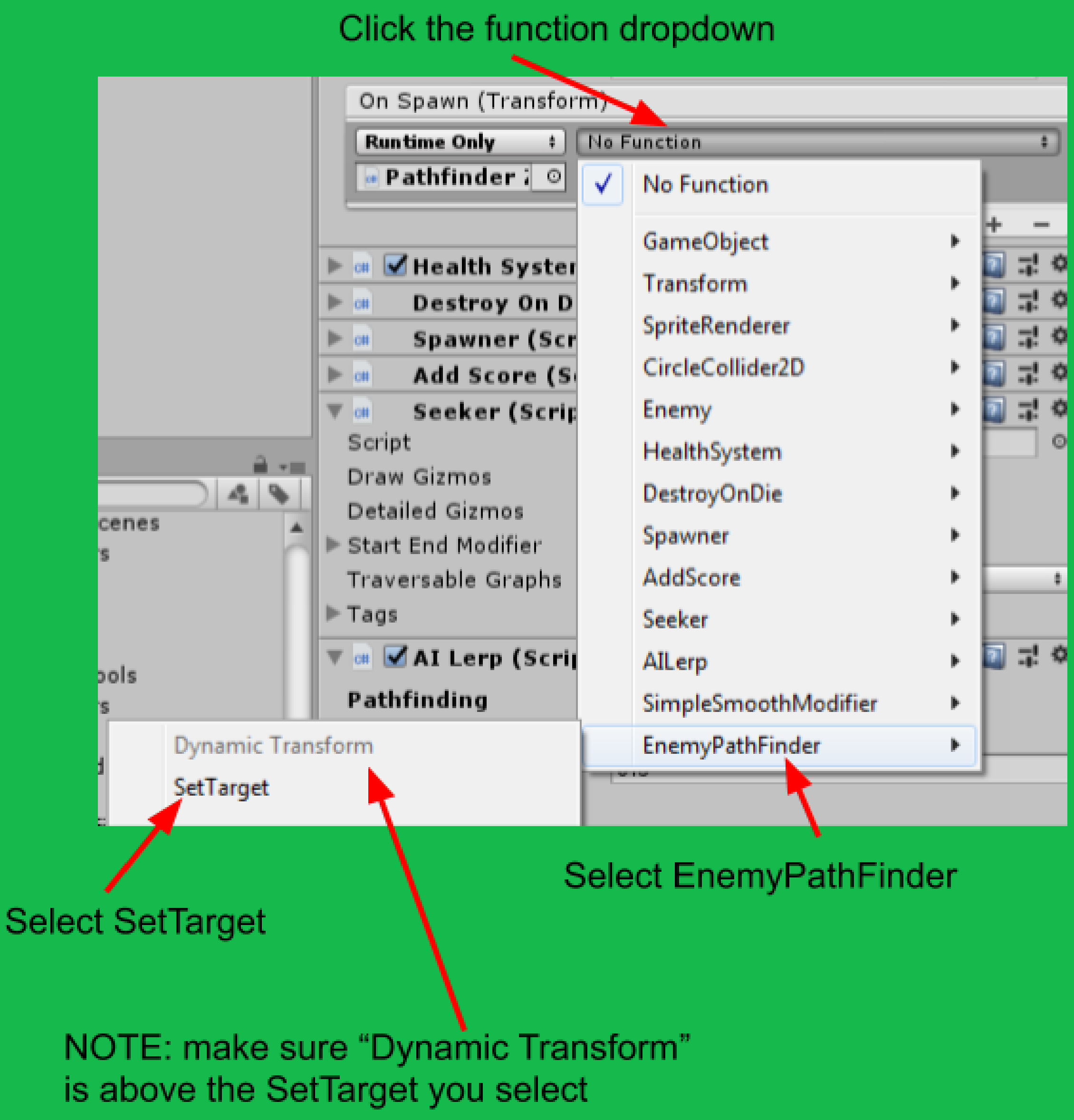
Do this

- In the **Project view**, with the **Zombie** prefab selected:
- Drag the **EnemyPathFinder** **component** onto the **listener** inlet
- **NOTE**: drag the **component** that is already on the **Zombie** prefab not the script!



Do this

- In the **Project view**, with the **Zombie** prefab selected:
- On the **Enemy** component, click the function dropdown on the **OnSpawn** event
- Select **EnemyPathFinder** > **SetTarget**



Task 4. Setup walls and obstacles

Explanation

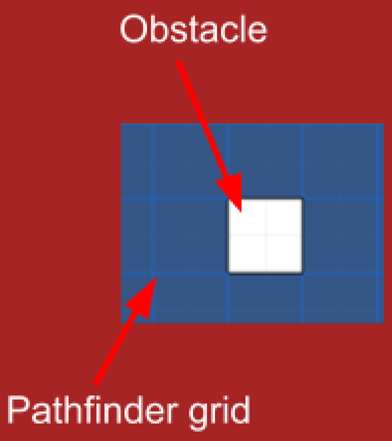
- We want walls and obstacles for the Zombie to chase us around
- If you already have these in your scene, make sure they follow these guidelines

Do this

- Create a new **Empty GameObject** using the **Create** button on the **Hierarchy**
- Add a **Box Collider 2D** to it
- Set the **Layer** to **Default**

Check this

- Check your new obstacle is on the **Pathfinder** grid



Explanation

- This pathfinding system uses a grid
- The unity **Tilemap** system also uses a grid
- You can apply a **Tilemap collider** to a **tilemap** and the pathfinding will conform to your level layout
- If you match up the grid size for your **Tilemap**, you can very easily build a level with a **Tilemap** and pathfinding built in!
- **NOTE**: the **Rapid Prototype 2D engine** builds levels using this technique!

Useful links

- More information about **Tilemaps** [Tilemaps - Manual]

Task 5. Setup player for wall collision

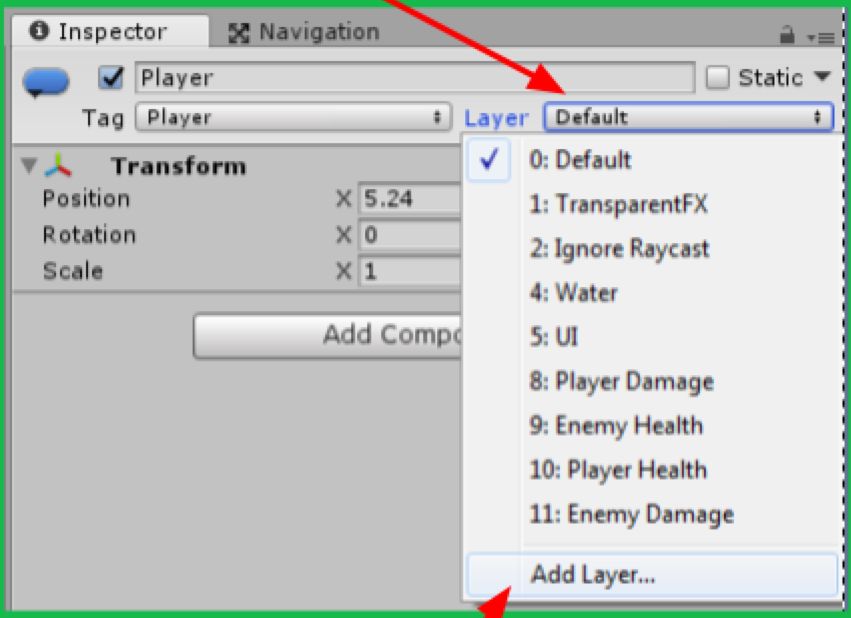
Explanation

- The player currently will walk through walls!
- We need to add a child GameObject with a Collider 2D and set the Layer to fix this

Do this

- Select the **Player** GameObject in the **Hierarchy**
- Click the **Layer** dropdown in the **Inspector**
- Select **Add Layer**

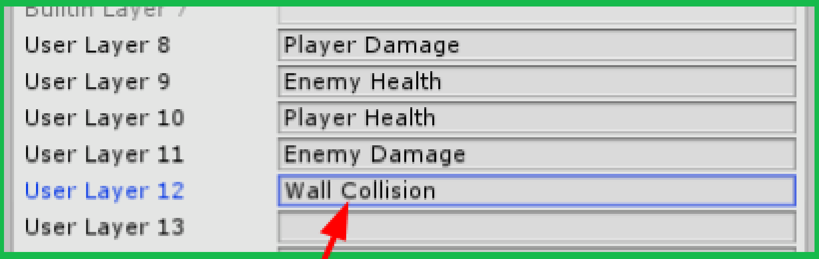
Layer dropdown



Select Add Layer

Do this

- In the **Tags & Layers** panel, add a new **Layer** by typing in **Wall Collision** into an empty layer

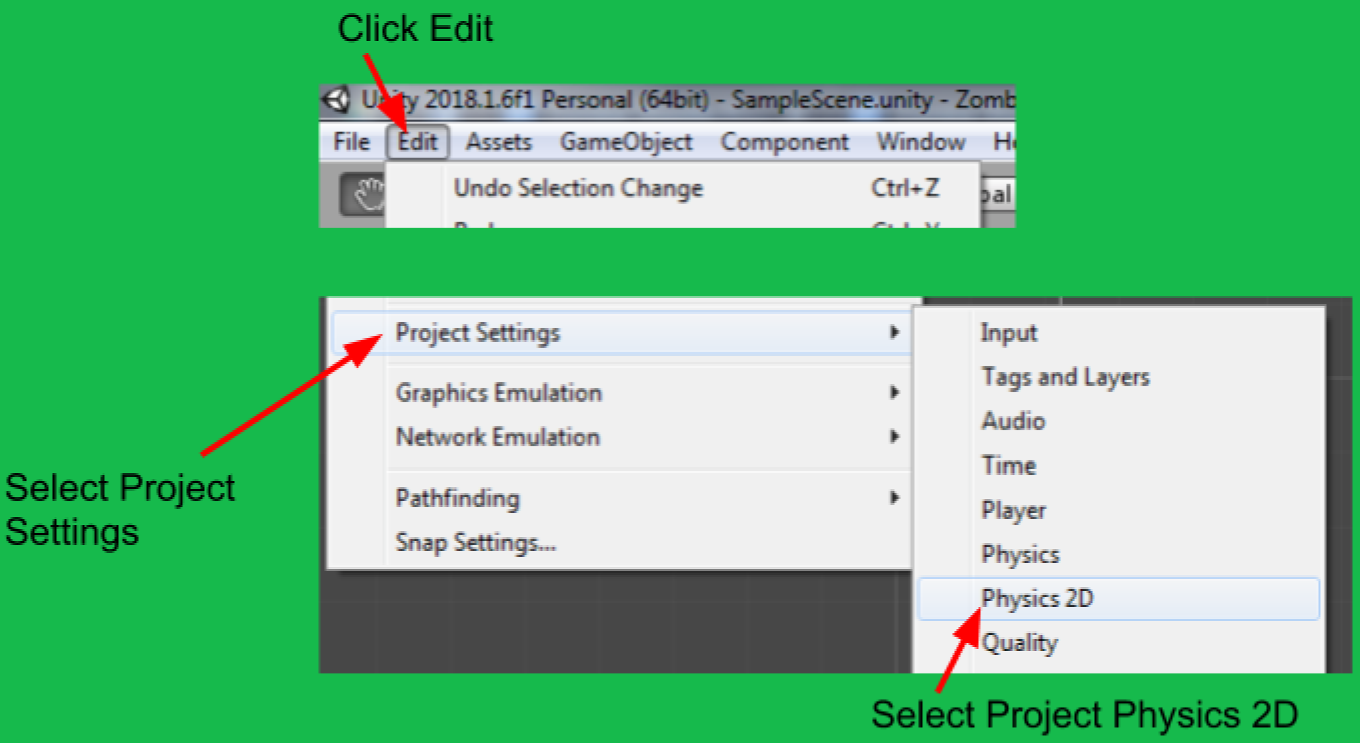


Builtin Layer 7	
User Layer 8	Player Damage
User Layer 9	Enemy Health
User Layer 10	Player Health
User Layer 11	Enemy Damage
User Layer 12	Wall Collision
User Layer 13	

Type Wall Collision into an empty layer

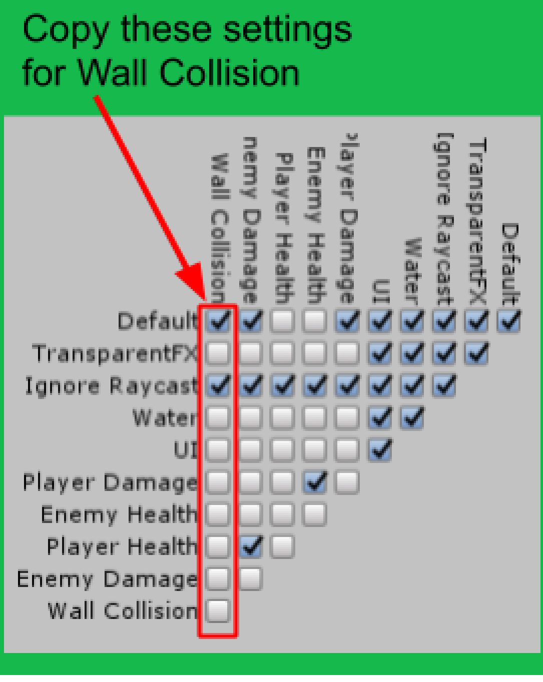
Do this

- Open the **Physics 2D Settings** panel:
- Go to the **top menu** in the **Unity Editor**
- Select **Edit** > **Project Settings** > **Physics 2D**



Do this

- Copy the settings shown in the image below for **Wall Collision**



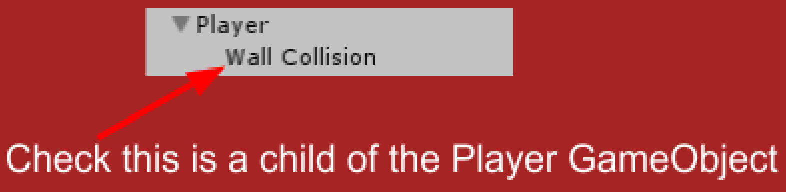
	Wall Collision	nemy Damage	Player Health	Enemy Health	›layer Damage	UI	Water	Ignore Raycast	TransparentFX	Default
Default	✓	✓	☐	☐	✓	✓	✓	✓	✓	✓
TransparentFX	☐	☐	☐	☐	☐	✓	✓	✓	✓	
Ignore Raycast	✓	✓	✓	✓	✓	✓	✓	✓		
Water	☐	☐	☐	☐	☐	✓	✓			
UI	☐	☐	☐	☐	☐	✓				
Player Damage	☐	☐	☐	✓	☐					
Enemy Health	☐	☐	☐	☐						
Player Health	☐	✓	☐							
Enemy Damage	☐	☐								
Wall Collision	☐									

Do this

- Select the **Player** GameObject in the **Hierarchy**
- Click the **Create** button at the top of the **Hierarchy** panel
- Select **Create Empty Child**
- Rename the new GameObject to **Wall Collision**

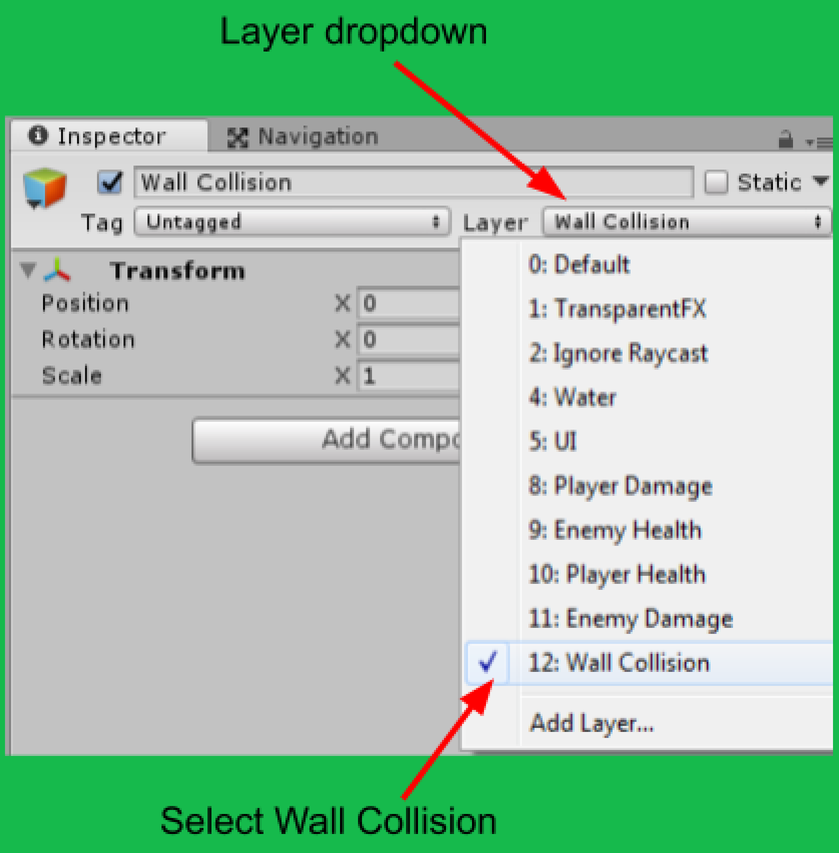
Check this

- Check your new **Wall Collision** GameObject is a child of the **Player** in the **Hierarchy**



Do this

- Select the **Wall Collision** GameObject we just created
- Set its **Layer** to **Wall Collision**



Wall Collision
Static
Tag Untagged
Layer Wall Collision

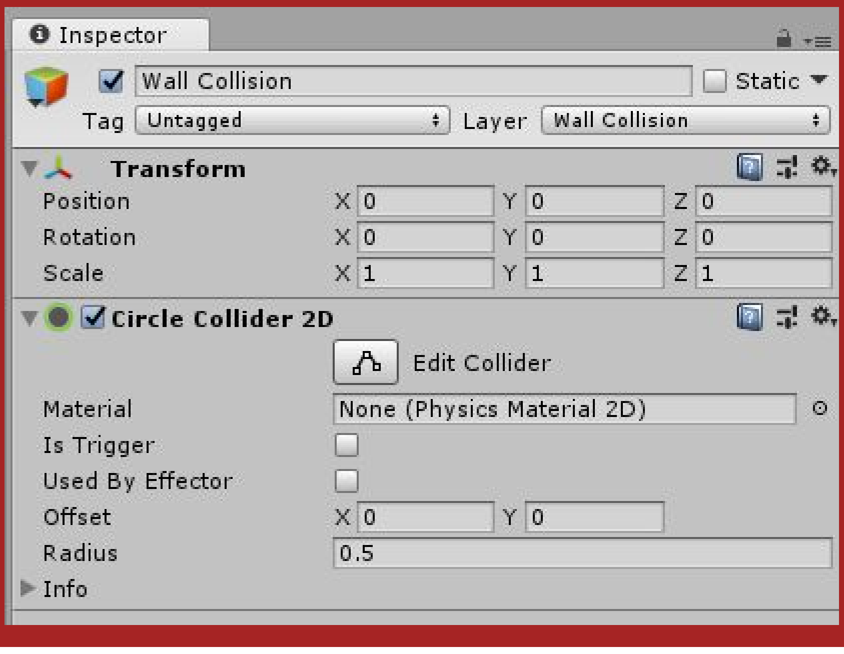
Transform
Position X 0
Rotation X 0
Scale X 1

Do this

- Select the **Wall Collision** GameObject we just created
- Click the Add Component button
- Select **Physics 2D** > **Circle Collider 2D**

Check this

- Check your **Wall Collision** GameObject is setup like the image below:



Wall Collision
Static
Tag Untagged
Layer Wall Collision

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

Circle Collider 2D
Material None (Physics Material 2D)
Is Trigger ☐
Used By Effector ☐
Offset X 0 Y 0
Radius 0.5