Zombie Shooter Project 2a

Task 1. Create layers for damage and health

Explanation

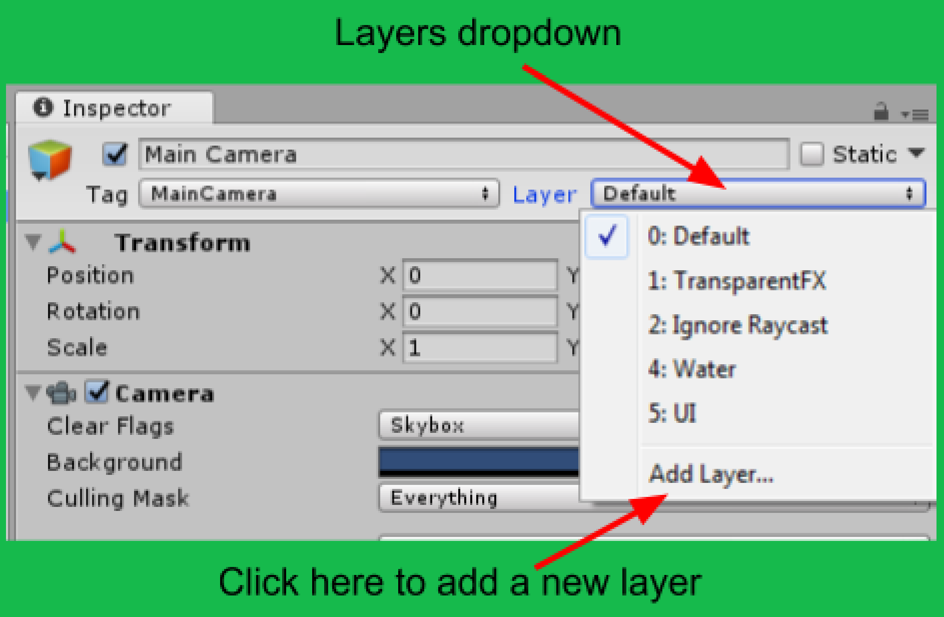
- Layers in Unity can be used to ignore or allow certain collisions and triggers.
- For example you may want a player bullet to damage enemies, but not the player
- Layers can be configured using the **Layer Collision Matrix** to either ignore or allow other layers to interact using the **OnTrigger** or **OnCollision** methods

Useful links

- Learn more about **Layers** — Layers - Manual
- Learn more about the **Layer Collision Matrix** — Layer Collision Matrix

Do this

- Select ANY GameObject in the **Hierarchy**
- Click the **Layers** dropdown at the top of the **Inspector**
- Click **Add Layer** to open the **Tags and Layers** manager



Explanation

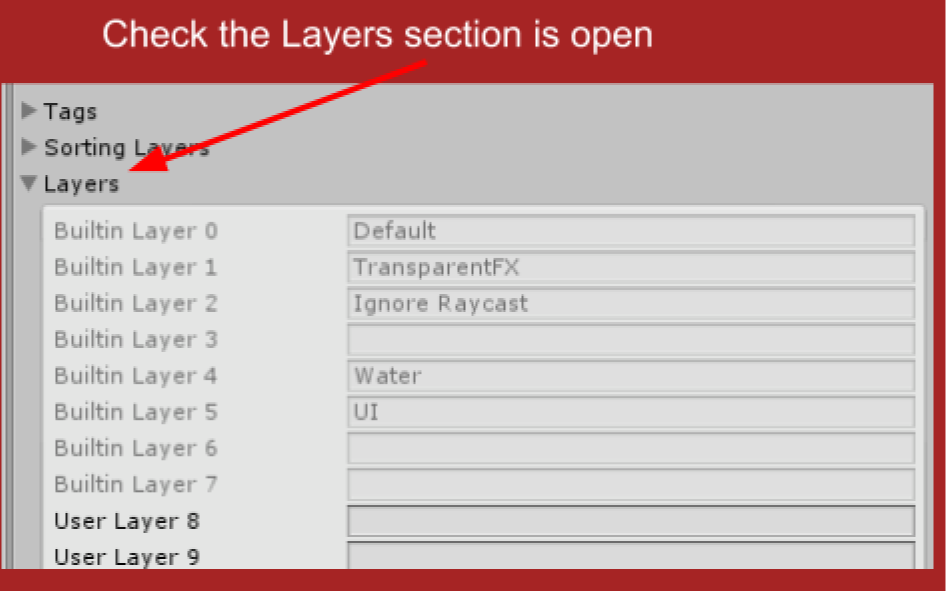
- The Tags and Layers Manager will allow you to add and remove **Tags**, **Layers** and **Sorting Layers**

Useful links

- Learn more about the **Tags and Layers Manager** — Tags and Layers - Manual

Check this

- The **Layers** section should be open on the **Tags and Layers Manager**
- Check you can see this:

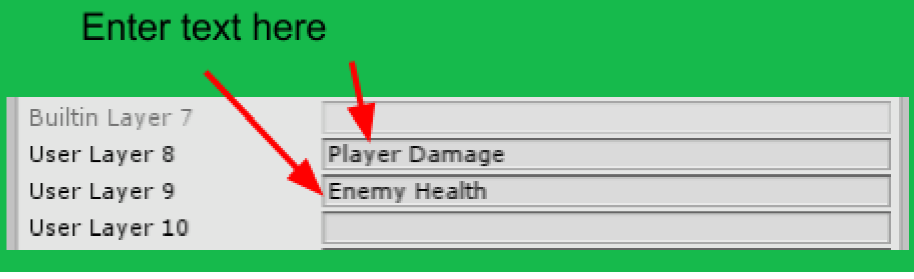


Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
Builtin Layer 3	
Builtin Layer 4	Water
Builtin Layer 5	UI
Builtin Layer 6	
Builtin Layer 7	
User Layer 8	
User Layer 9	

Do this

- Click in the text field of **User Layer 8**
- Type **Player Damage** in the text field

- Click in the text field of **User Layer 9**
- Type **Enemy Health** in the text field



Builtin Layer 7	
User Layer 8	Player Damage
User Layer 9	Enemy Health
User Layer 10	

Task 2. Setup the Player Damage Layer in the Collision Matrix

Explanation

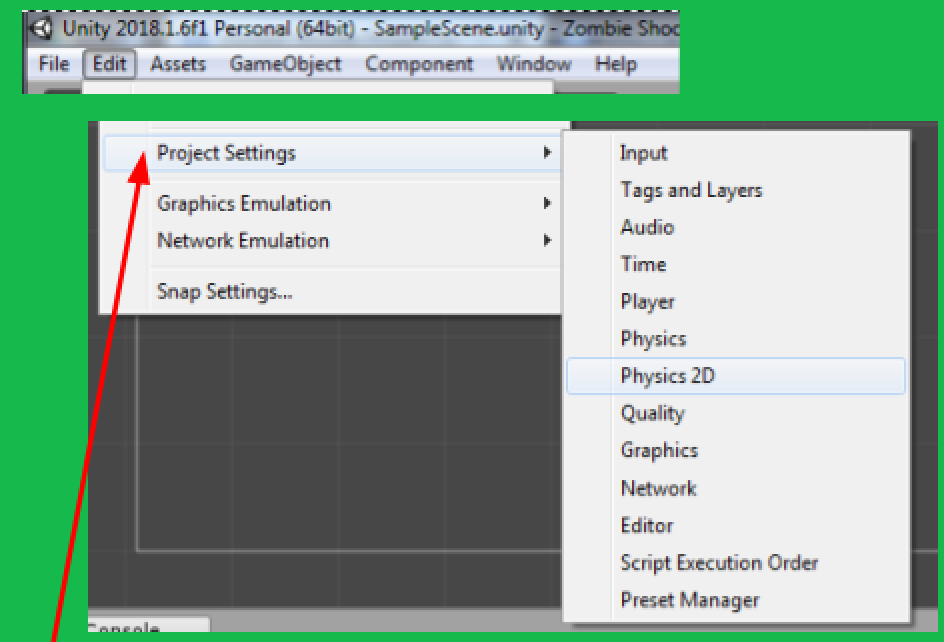
- We will set the **Player Damage** layer to ONLY interact with the **Default** and **Enemy Health** layers
- This will allow bullets using the **Player Damage** layer to only apply damage to enemies or destroy themselves if they hit a wall

Useful links

- Learn more about the **Layer Collision Matrix** [Layer Collision Matrix]

Do this

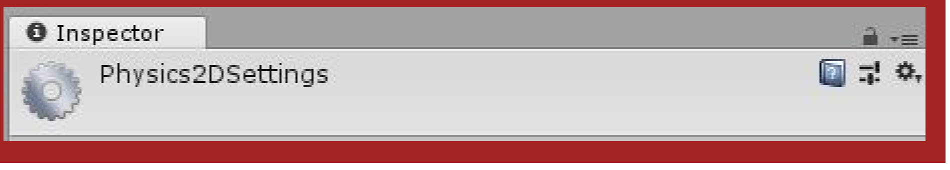
- On the top menu in the Unity Editor, click:
- **Edit** > **Project Settings** > **Physics 2D**



Project settings is near the bottom of the menu!

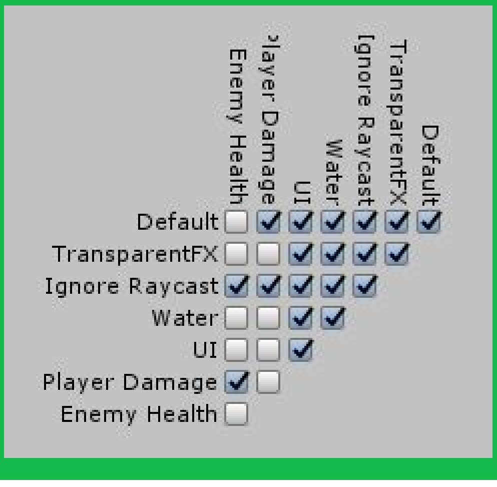
Check this

- Check the **Physics 2D Settings** is open in the Inspector



Do this

- In the **Layer Collision Matrix**, UNTICK the boxes to match the image to the right



	Enemy Health	Player Damage	UI	Water	Ignore Raycast	TransparentFX	Default
Default	☐	✓	✓	✓	✓	✓	✓
TransparentFX	☐	☐	✓	✓	✓	✓	
Ignore Raycast	✓	✓	✓	✓	✓		
Water	☐	☐	✓	✓			
UI	☐	☐	✓				
Player Damage	✓	☐					
Enemy Health	☐						

Task 3. Make a Bullet Prefab

Explanation

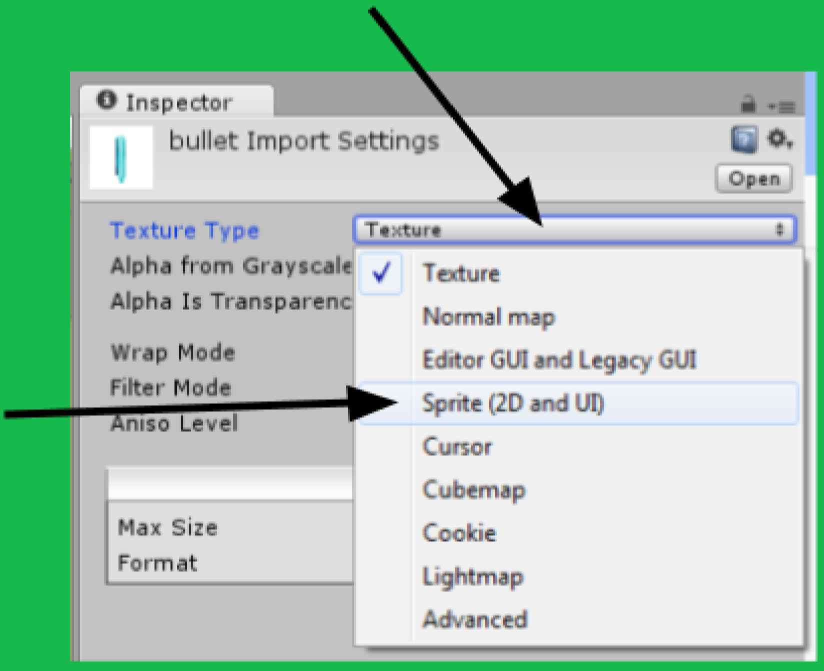
- We will create a **Prefab** (prefabricated GameObject) for our **Bullet**
- Using a **Prefab** means we don't need to keep our **Bullet** GameObject in the **Hierarchy**
- Using a **Prefab** also means we can use the **Bullet** in any Scene, while only having to make changes to one Bullet
- For an in-depth explanation, Please refer to the **Prefabs in Unity** document in the **Resources** section on the **DLE**

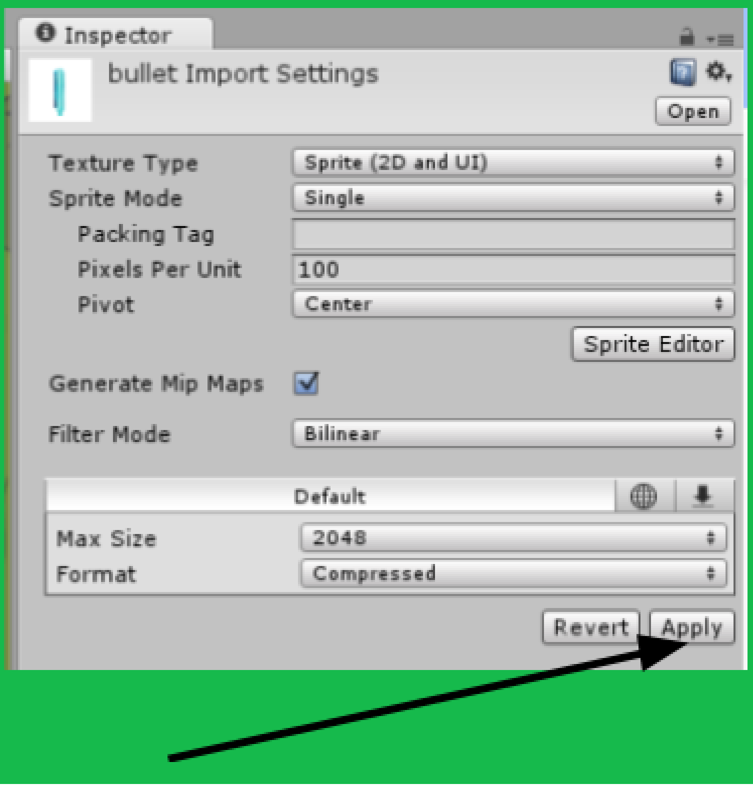
Useful links

- Learn more about Prefabs — Prefabs - Manual

Do this

- In the **Sprites folder** of the **Project view**, select the **Bullet** artwork
- In the **Inspector**, Set the **Texture Type** to **Sprite**
- Click **Apply**

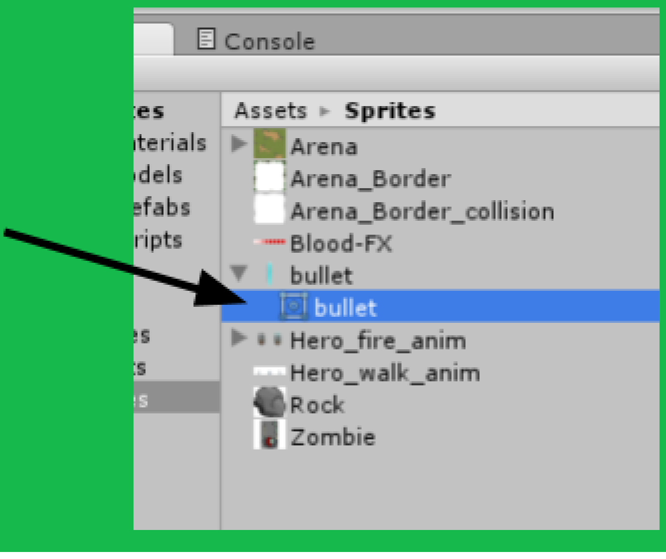




Texture Type: Sprite (2D and UI)
Sprite Mode: Single
Packing Tag:
Pixels Per Unit: 100
Pivot: Center
Generate Mip Maps: ✓
Filter Mode: Bilinear
Max Size: 2048
Format: Compressed

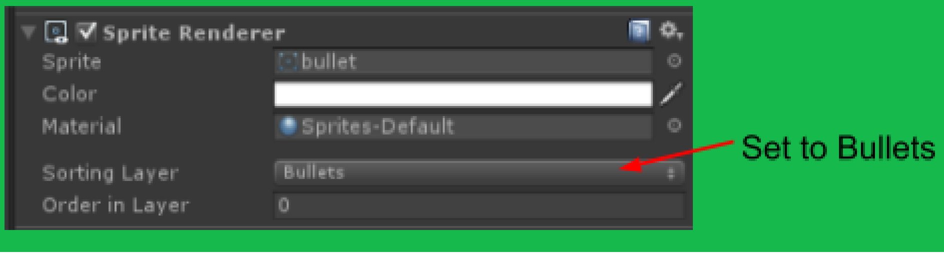
Do this

- In the **Sprites folder** of the **Project view**, select the **Bullet Sprite** we just created
- **Drag** the artwork into the **Hierarchy**



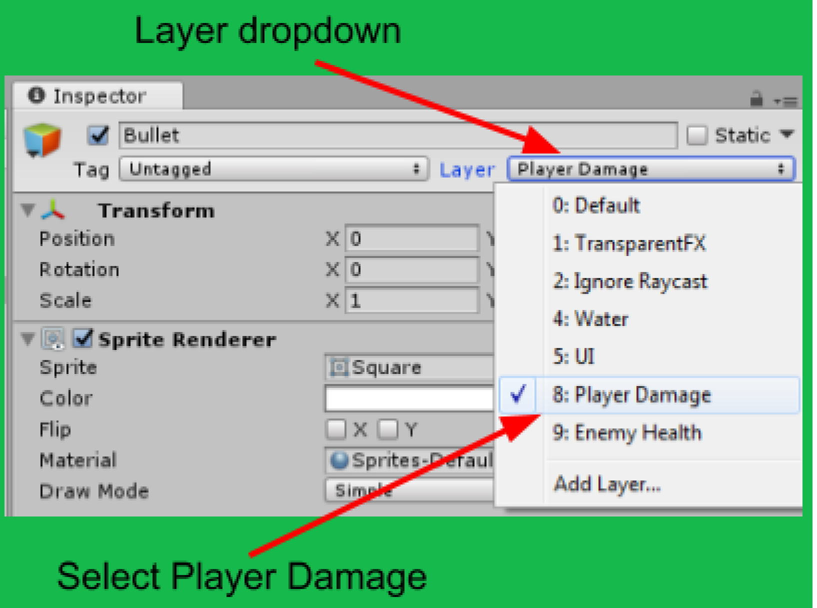
Do this

- On the **Sprite Renderer**, set the **Sorting Layer** to **Bullets**



Do this

- On the **GameObject**, at the top of the inspector, set the **Layer** to **Player Damage**



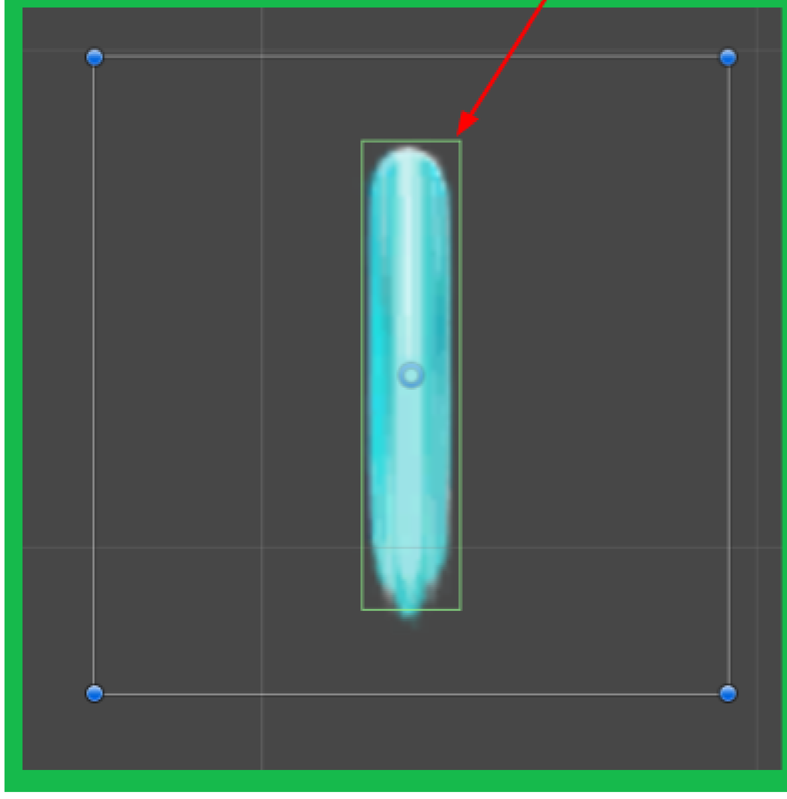
Do this

- Using the **Add Component** button, add a **Box Collider 2D** to the **Bullet**

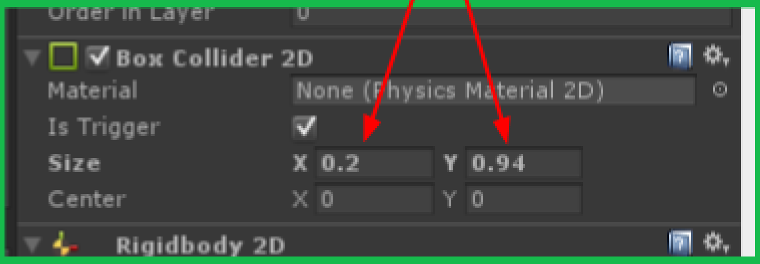
Do this

- Adjust the size of the **Box Collider 2D** to fit around the **Bullet artwork**
- Use the **X** and **Y** values to resize the green collider box

Collider is the green box



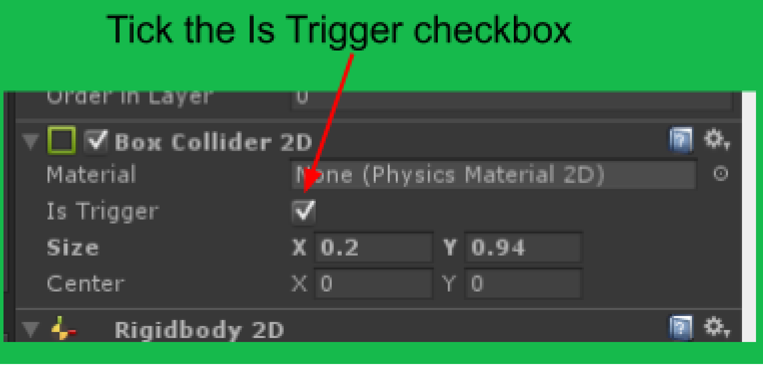
Resize the collider using the X and Y values to fit the bullet artwork



Order in Layer 0
Box Collider 2D
Material None (Physics Material 2D)
Is Trigger ✓
Size X 0.2 Y 0.94
Center X 0 Y 0
Rigidbody 2D

Do this

- Tick the **Is Trigger** checkbox on the Box Collider 2D



Box Collider 2D
Material None (Physics Material 2D)
Is Trigger ✓
Size X 0.2 Y 0.94
Center X 0 Y 0
Rigidbody 2D

Explanation - Triggers on Collider Components

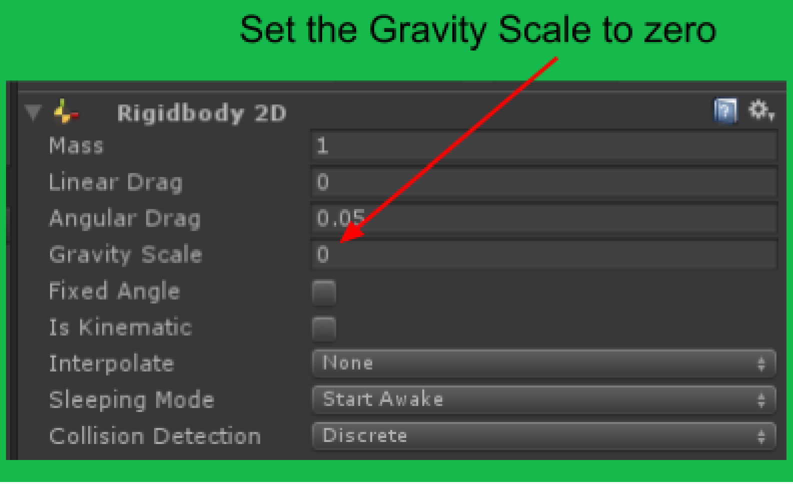
- All Colliders have an **Is Trigger** Checkbox
- When **Is Trigger** is **not ticked** the GameObject will react to colliding with other GameObjects (if it **also** has a Rigidbody attached!)
  - React like bouncing all over the place and other cool physics simulation stuff
- When **Is Trigger** is **ticked**, the GameObject (like our Bullet) will **pass through** other GameObjects
  - Why would we want to do this?
- The GameObject will still send out a **signal** when it **passes through** something!
- We can pick up on this **signal** using a method called **OnTriggerEnter**
  - Our Bullet script (we will create shortly) will use this to damage Zombies and destroy itself

Do this

- Using the **Add Component** button, add a **Rigidbody 2D** to the **Bullet**

Do this

- Set the **Gravity Scale** to zero on the **Rigidbody 2D** Component



Rigidbody 2D
Mass 1
Linear Drag 0
Angular Drag 0.05
Gravity Scale 0
Fixed Angle
Is Kinematic
Interpolate None
Sleeping Mode Start Awake
Collision Detection Discrete

Explanation - Prefabs? what are they?

- A **Prefab** is a **GameObject** we can use in any **Scene**
- The reason being, a **Prefab** is actually a separate file!
- We can edit the **Prefab** once and use it anywhere
- **Prefabs** are often used for things we want to spawn and destroy a lot, like:
  - Bullets
  - Zombies
  - Explosions
- **Prefabs** can also be used on the player GameObject, so we can use it in multiple scenes

Do this

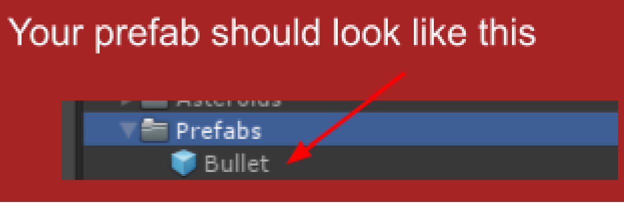
- In the **Assets folder** of the **Project view**, create a new **folder**
- Name it **Prefabs**

Do this

- Select the **Bullet** GameObject in the **Hierarchy**
- Drag the **Bullet** GameObject into the **Prefabs folder** you just created in the **Project view**
- **You have just created a Prefab!**

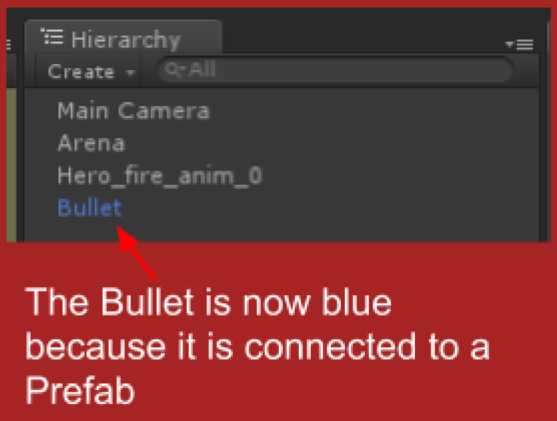
Check this

- Your new Prefab will be in the Prefabs folder in the Project view
- Note: **Prefabs** always have a **Blue Cube** icon in the **Project view**



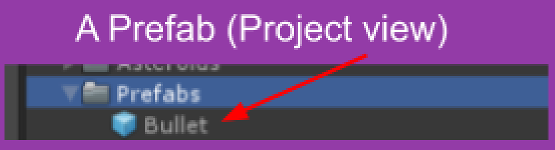
Check this

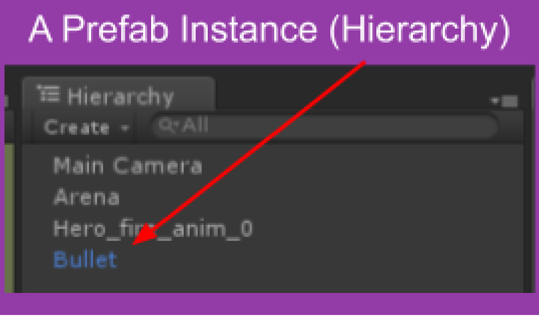
- The **Bullet** GameObject in the Hierarchy has changed colour to Blue



Explanation - Prefabs and Prefab Instances

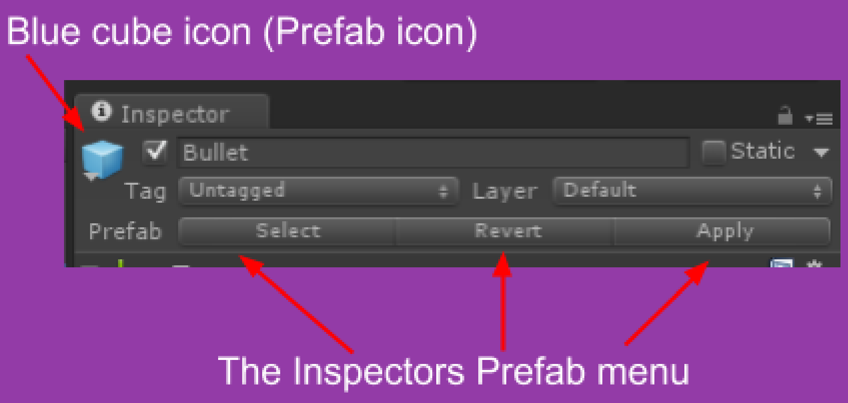
- A **Prefab**, as explained previously is a GameObject in the **Project view**
- A **Prefab Instance** is a GameObject in the **Hierarchy connected** to a Prefab





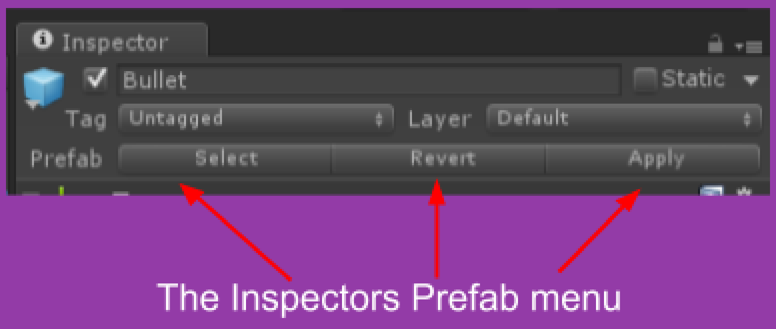
Explanation - Prefab Instances in the Inspector

- As explained, a Prefabbed GameObject in the Hierarchy is blue
- The **Inspector** also shows some differences!



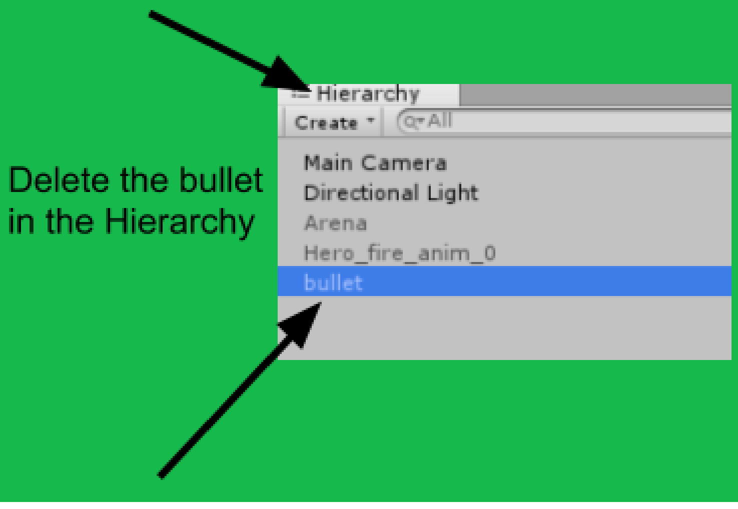
Explanation - Inspectors Prefab Menu

- When you make changes to a **Prefab Instance**, they may not appear on your **Prefab**
- The buttons **Select**, **Revert** and **Apply** allow you syncronise changes between your **Prefab Instance** and your **Prefab**
- **Select** - **Selects** the **Prefab** (in the **Project view**) connected to the **Prefab Instance**
- **Revert** - Any **changes** made to the **Prefab Instance** are replaced with the Prefab settings
- **Apply** - Apply **changes** made from the **Prefab Instance** to the **Prefab**



Do this

- Select the **Bullet** GameObject (**Prefab Instance**) in the **Hierarchy**
- **Delete** the **Bullet** GameObject in the **Hierarchy**
- You have a **Bullet Prefab** in your **Project view**, we can make changes to that


Delete the bullet in the Hierarchy
Hierarchy
Create
All
Main Camera
Directional Light
Arena
Hero_fire_anim_0
bullet