**TEMASEK POLYTECHNIC**

**SCHOOL OF INFORMATICS & IT**

**DIPLOMA IN IMMERSIVE MEDIA & GAME DEVELOPMENT**

**AY2025/2026 APRIL SEMESTER (LEVEL 2)**

**GADV (CGE2C25)**

**Introduction to Unity**

**A picture containing clipart

Description automatically generated**

**GameObjects and Prefabs**

To see the additional comments and resources, make sure you select **All Markup** in the **Review/Tracking** pane

1. **Create a GameObject in the Unity Editor**

GameObjects are created using the Editor or by script. We’ll look at using the Editor here. Watch this brief Code Monkey [video](https://www.youtube.com/watch?v=uCA8Q3nwaz4) before starting the worksheet!

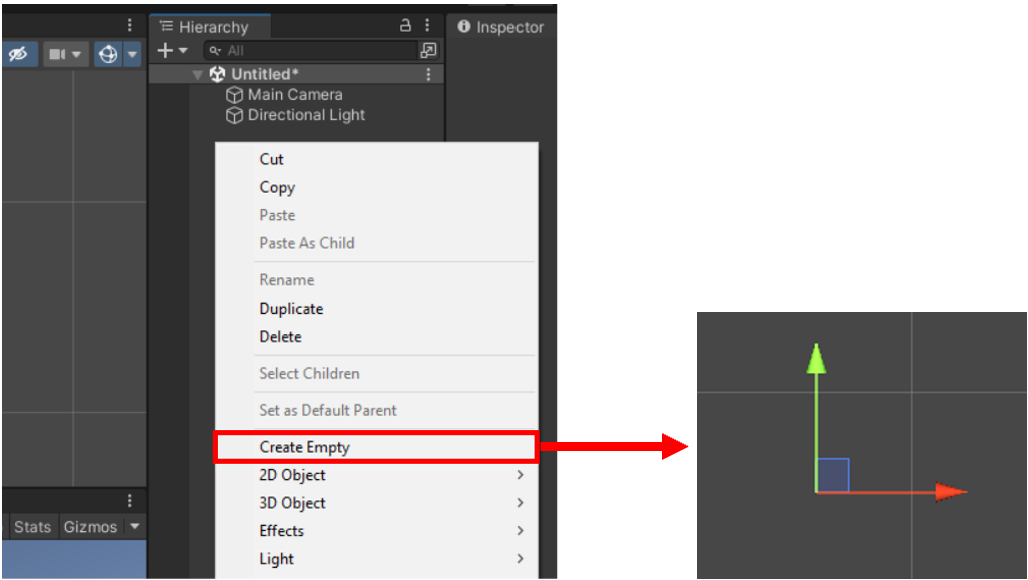
Okay, now follow these steps to create a GameObject.

1. In your **Worksheets** project, create a new folder: **Assets/Worksheets/Week\_01**.

Inside Week\_01, add these folders:

* **Scenes**
* **Scripts**

1. Select the menu item **File/New Scene** to create a new scene called **Week\_01**. Save this in your **Scenes** folder.
2. Right-click in the **Hierarchy** window and select **Create Empty**.



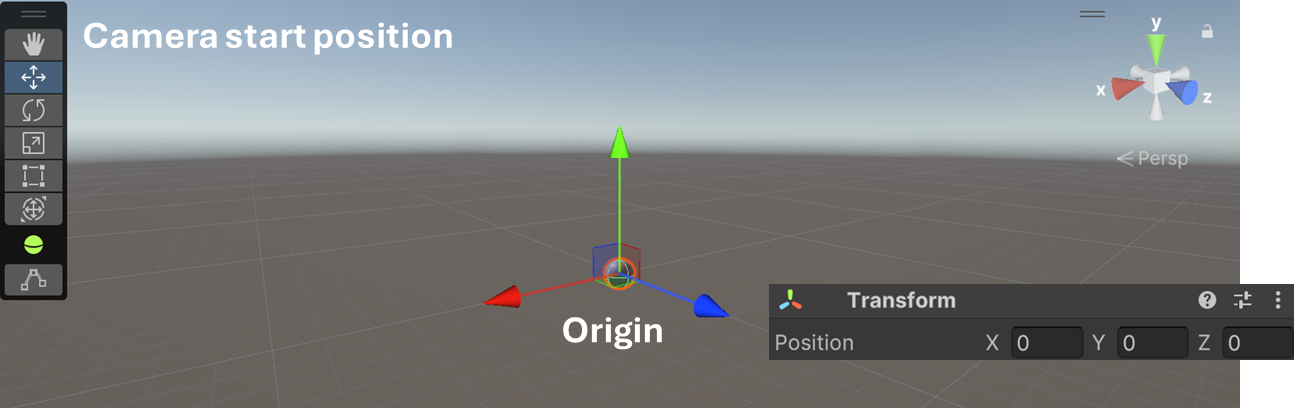
1. Click on the new GameObject.
2. Look in the Inspector. The GameObject is empty! This means that there is no component attached for creating or displaying a 2D image or 3D mesh, or any other kind of visual object.

In fact, the only component it has is a **Transform** component.

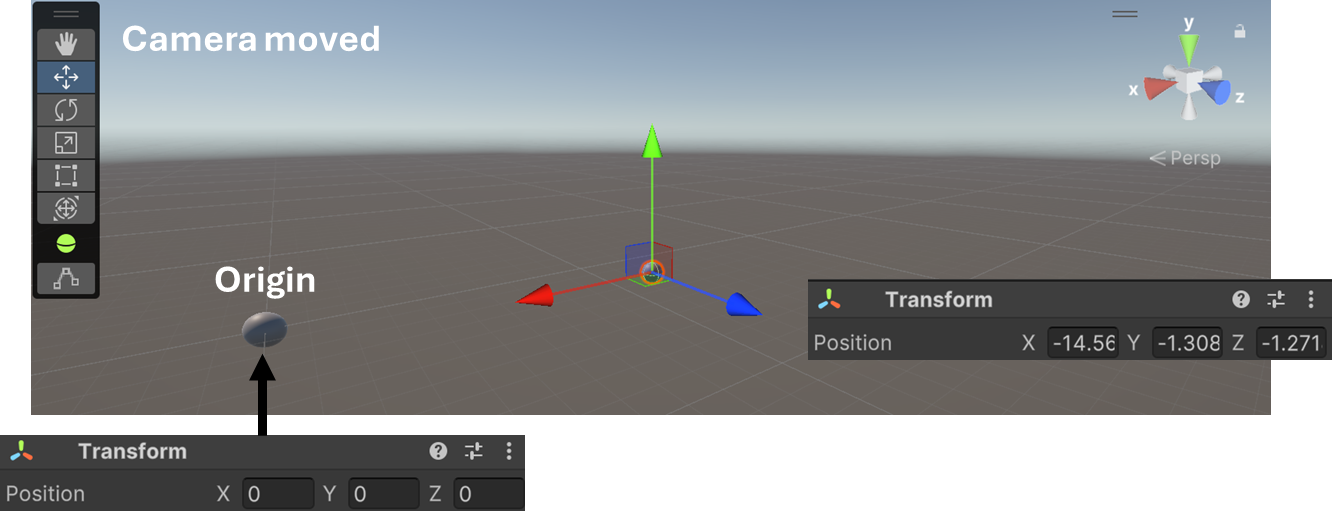
The Transform’s XYZ position values may not be set to 0, depending on where your camera is pointing. This is because the GameObject is created at the centre of the screen (i.e. wherever the camera is looking).

In the image below, the camera is intially at its default start position.

When you create a GameObject, its position is set along the camera’s view axis (the Z axis). The position displayed by the Transform component is (0,0,0).



If the camera is moved, a new GameObject will be positioned according to where the camera now points into the scene.



To position the GameObject at the origin, click the three dots as shown below, and select **Reset**.

Graphical user interface, application

Description automatically generated

The XYZ values will all be set to 0, which means the GameObject is at the origin.

Graphical user interface

Description automatically generated

As you move the camera around to build a scene, it’s very easy to create GameObjects in places you didn’t intend!

But you often need to ensure that objects are created at the origin. To do so, simply reset the transform. This is especially important when creating **prefabs** (covered later in GADV).

1. **Add a SpriteRenderer component to display an image**
2. We’ll add an image to the GameObject, to display a sprite. To do this, we need to add a component that can display an image.

A more technical word for display is ***render***. The Unity API has a component called **SpriteRenderer**.

Select the GameObject, click **Add Component**, search for **Sprite**, then select the **SpriteRenderer** component.

Graphical user interface

Description automatically generated

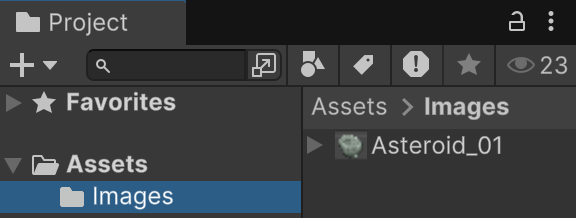
The SpriteRenderer is added as a component to the GameObject.



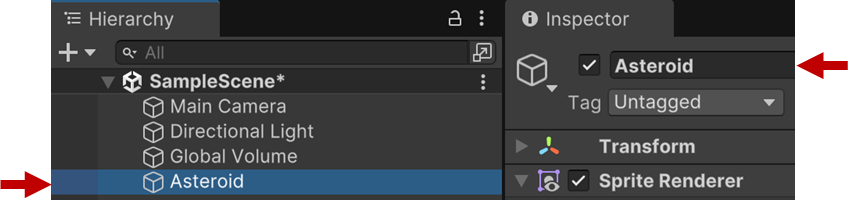
1. Now we have to select an image to show.

We’ll use **Asteroid\_01.png**.

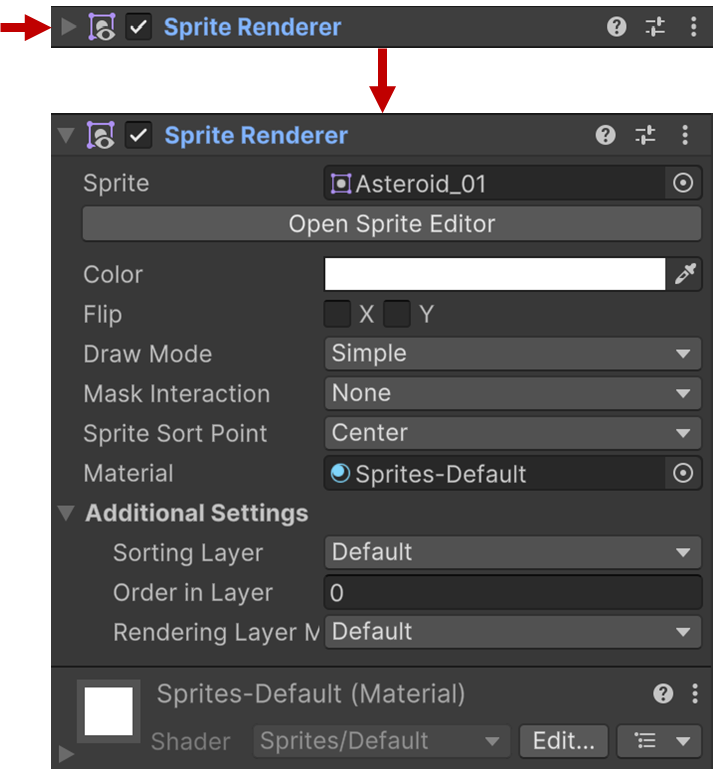
Create a new folder called Images in your project, then download the Asteroid\_01.png file from the Resources folder in LMS into Images.



Since we want the GameObject to represent an asteroid, rename it to **Asteroid**.



Select the Asteroid GameObject, then click the dropdown arrow next to its SpriteRenderer to expand the panel.



This shows all the available properties of the Sprite Renderer. We are able to change these properties. This is how Unity allows game designers who can’t code to make changes to the objects in a game.

IMPORTANT! If you see the button below in the Inspector, click it to install the 2D Sprite package. This is needed for working with sprites, and will be essential once you start working on your project.



Drag Asteroid\_01 from the Images folder into the Sprite field.

It doesn’t work!

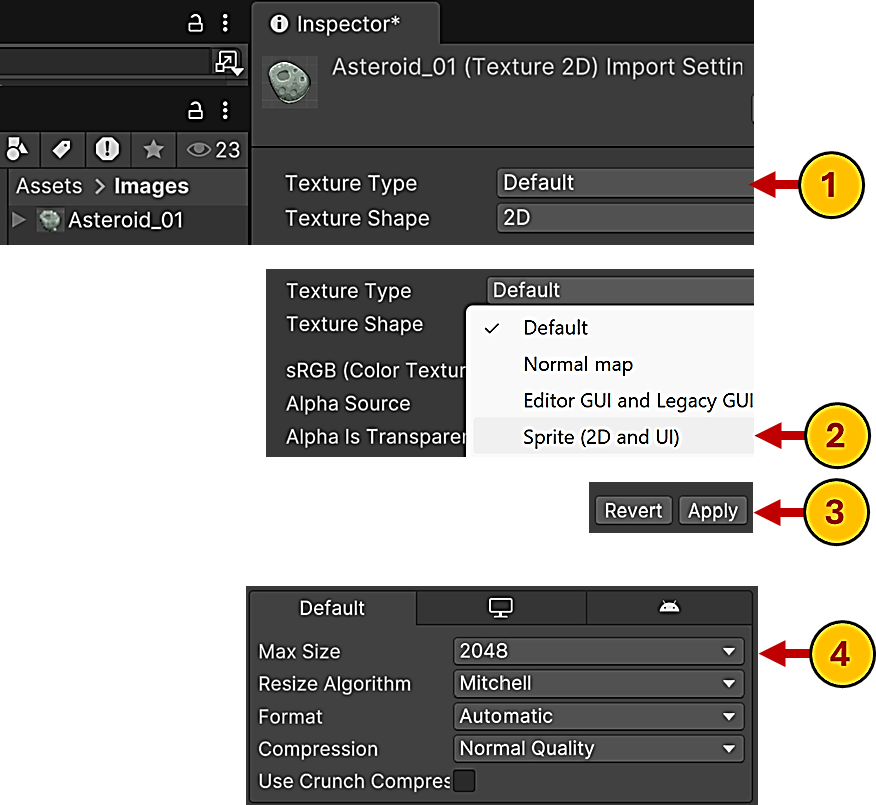


Unity automatically imported the asteroid image when we dragged it into the project (in the Images folder). But an image can be used for many different purposes, e.g. a texture, cursor icon, sprite, etc.

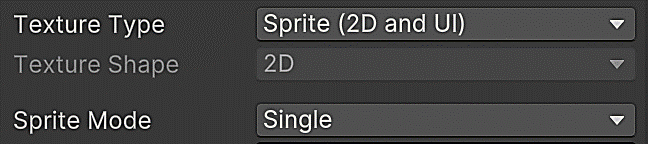
We need to set the texture type of the image to **Sprite**.

First, select the image in the Images folder, then follow these steps:

1. Select **Texture Type**
2. Set the type to **Sprite (2D and UI)**
3. Scroll to the bottom of the panel and click **Apply**.

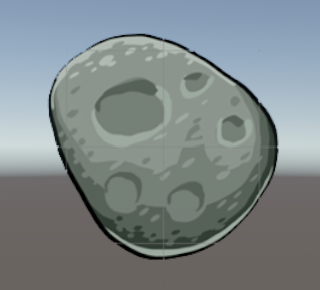


Also make sure that Sprite Mode is set to **Single**. A Sprite Mode of Multiple is used for animation using spritesheets.



Note the **Max Size** property (step 4 above). You can set this to various values. If the image is greater than Max Size, Unity will scale the resolution down. This is useful if you need to limit the amount of memory used by images.

Finally, the image appears!



1. **Prefabs**

**What Are Prefabs?**

A **Prefab** (short for *prefabricated object*) is a reusable GameObject template that can be created once and used multiple times in a scene.

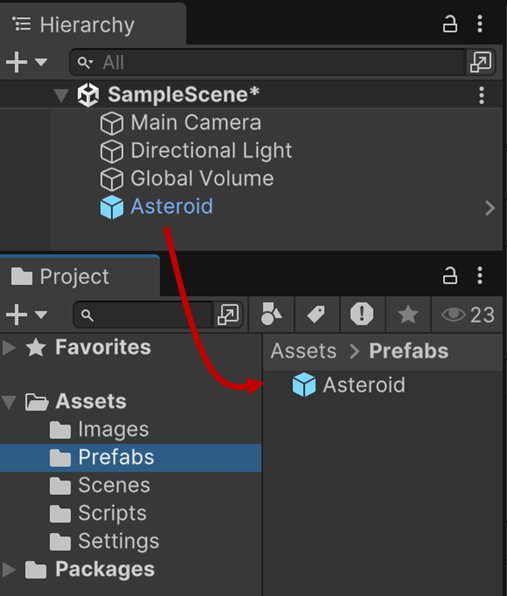
Prefabs are useful when you need to create many identical objects, such as enemies, power-ups, obstacles, or UI items such as buttons.

Instead of manually creating and setting up each GameObject, you can create a Prefab and instantiate (spawn) copies of it, making your workflow much more efficient.

1. **Creating a Prefab**

Let's turn the **Asteroid** GameObject from the previous section into a Prefab:

1. Open the **Project** window.
2. Create a new folder called **Prefabs** to keep things organized.
3. Drag the **Asteroid** GameObject from the **Hierarchy** window into the **Prefabs** folder in the **Project** window.
4. The GameObject turns blue in the **Hierarchy** panel, indicating it is now linked to a Prefab.



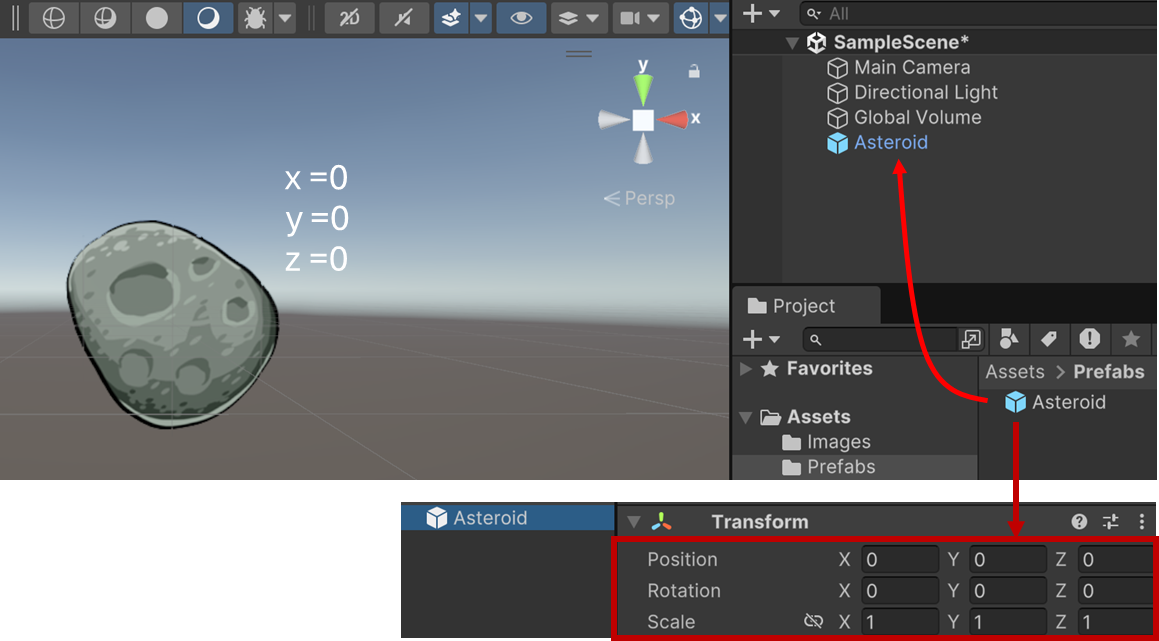
At this point, the **Asteroid** prefab is saved in the project.

You can delete the original GameObject from the **Hierarchy**—don’t worry, the Prefab still exists in the **Project** window!

IMPORTANT!

Check the GameObject’s transform properties before you create a prefab from it.

In the image below, the GameObject’s transform position values are all set to 0, which means the object is at the origin.

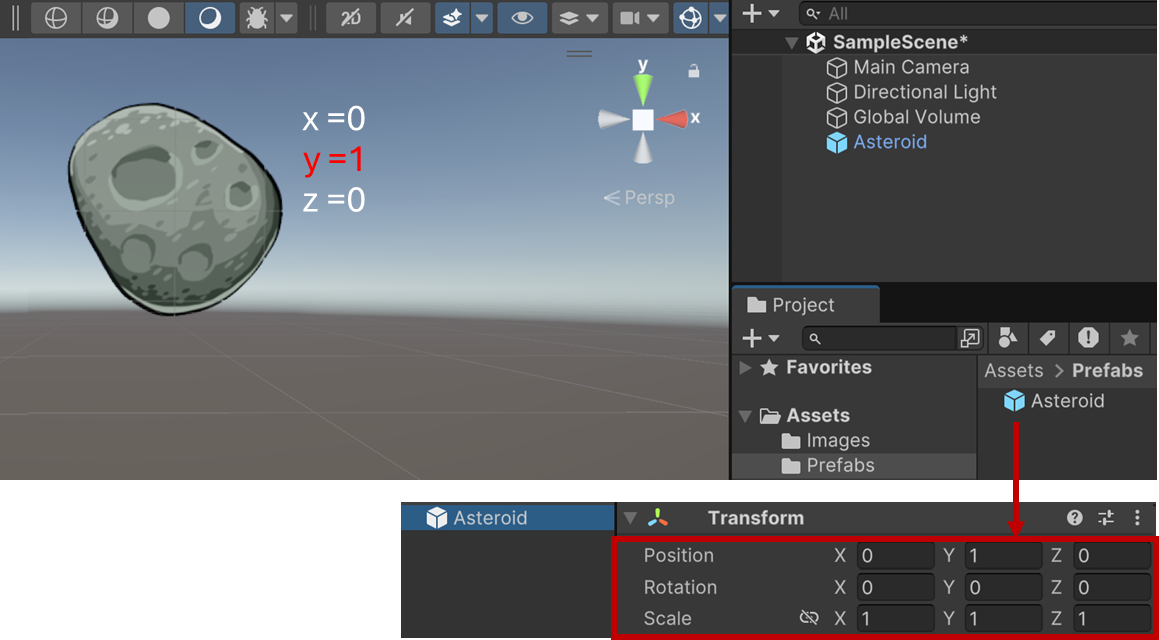


When you create a prefab from the GameObject, the prefab will have the same transform values.

So, if you drag the prefab into the Hierarchy, the instance created will have the same transform values, too.

In this case, the instance is positioned at the origin (0,0,0).

But if the GameObject you use to create the prefab is not at the origin, then when you drag the prefab into the Hierarchy, the instance will also not be at the origin.



This can be annoying.

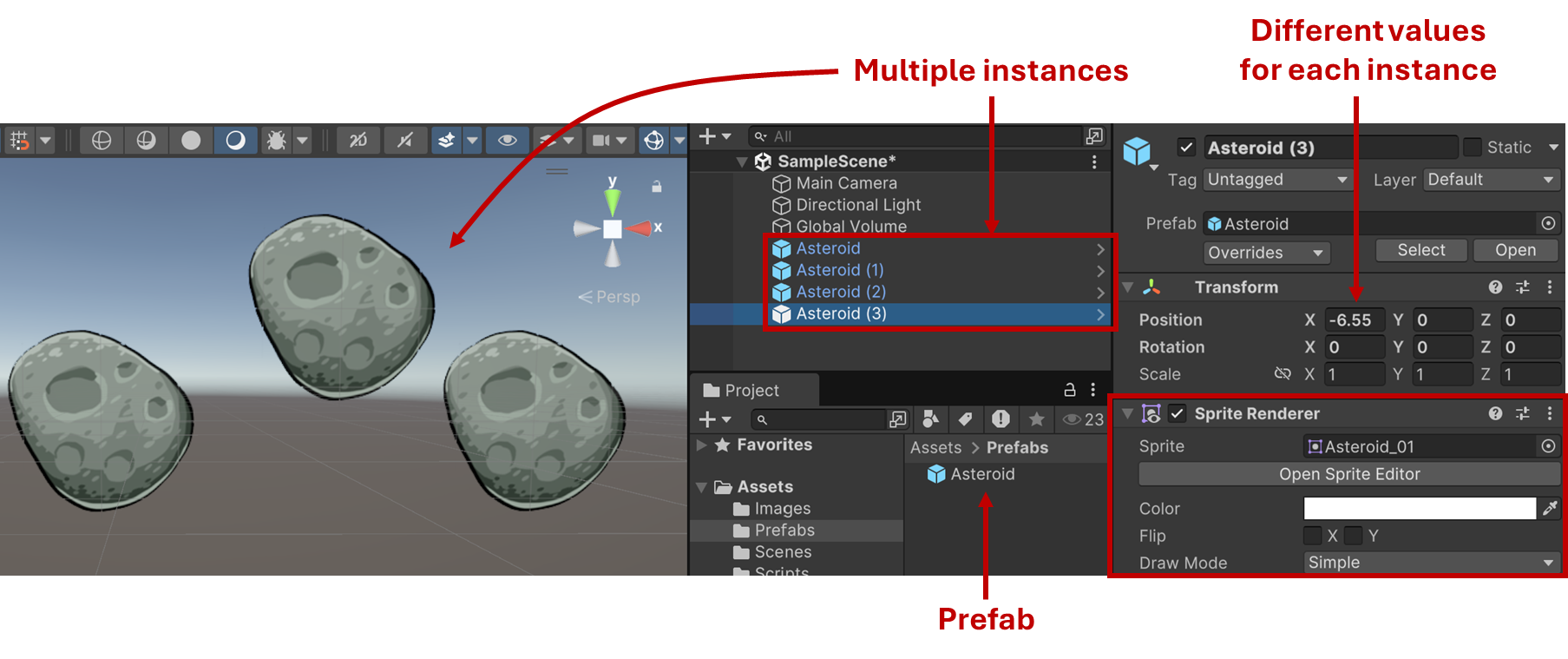
Or worse, you don’t realise and later must redo everything because the position or some other values are wrong.

To fix it, you’d have to edit the prefab to set the correct values (see **editing** below).

1. **Using a Prefab**

Now that you have a Prefab, you can create multiple instances of it.

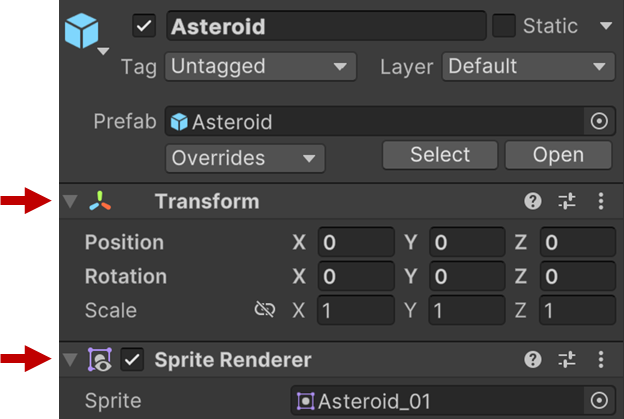
1. Drag the **Asteroid** Prefab from the **Project** window into the **Scene** or **Hierarchy**. Do this a few times.
2. You will see multiple **Asteroid** GameObjects, all linked to the original Prefab.



**Why Use Prefabs?**

* **Efficiency** – Instead of setting up objects manually, you can quickly spawn new instances.
* **Consistency** – All instances of a Prefab share the same components and properties.
* **Easy Updates** – If you change the Prefab (e.g., change its sprite, size, or collider), all instances update automatically.

Notice that each instance of the Asteroid prefab already has all the components and values it needs to display the asteroid image as a sprite.



You don’t need to do all the setting up for each sprite object again!

This is the main advantage of using prefabs. Your game objects can become very complex, so it’s useful to create a prefab for easy instantiation.

But what if you need to change a prefab after its been created?

1. **Editing a Prefab**

To modify a Prefab:

1. Double-click the Prefab in the **Project** window.
2. The **Prefab Editor** opens, allowing you to make changes.
3. Click **Back to Scene** when done.

All instances of the Prefab will reflect the changes!

**Exercise:**

Download Asteroid\_02.png.

A close-up of a asteroid

Description automatically generated

Make *all* the instances of the Asteroid prefab display Asteroid\_02.png.

1. **Overriding a Prefab**

If you modify a specific instance of a Prefab in the **Scene**, Unity marks it as having **overrides**. You can:

* **Apply** the changes to update the Prefab.
* **Revert** to restore the instance to match the Prefab.

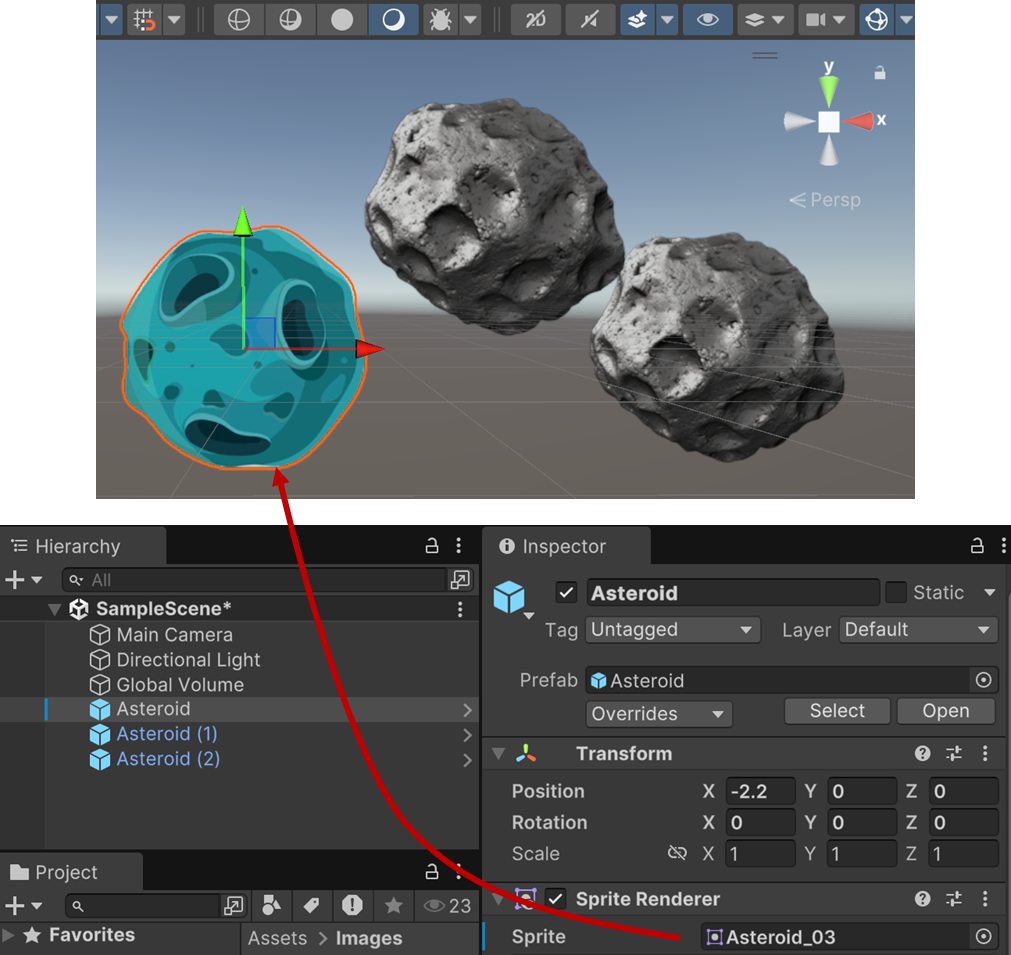
**Exercise:**

Download Asteroid\_03.png.

A blue planet with holes

Description automatically generated

Set only *one* instance of the prefabe in the scene to display Asteroid\_03.png.

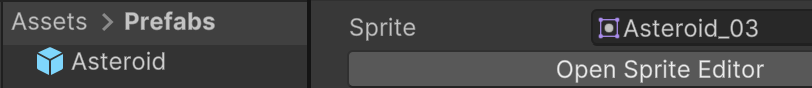


Note the **Overrides** dropdown.



Click the **Apply All** button. What happens?

Now look at the Asteroid prefab.



What is different?

Undo these changes, set the instance image to Asteroid\_03.png, then click the **Revert All** button. What happens now?

1. **Prefab Variants**

**What is a Prefab Variant?**

A **Prefab Variant** is a special type of prefab that is based on an existing prefab but allows for **custom modifications** while still inheriting properties from the original. This means that you can make changes to the variant without affecting the original prefab, and updates to the base prefab will still apply to the variant (unless overridden).

Prefab Variants are useful when you need different versions of the same object, such as:

* Different enemy types that share common properties but have unique appearances.
* Power-ups with the same behavior but different effects.
* UI elements that maintain a base layout but have different styles.

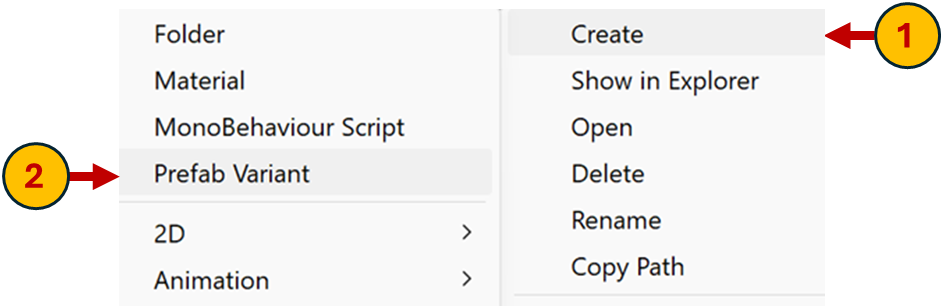
For example, if you have 5 different enemy types, you might think you have to create 5 separate prefabs for each type of enemy. But the enemies probably have a lot of functionality in common, e.g. rendering a 2D sprite or 3D model, finding a path to the player, moving towards the player, and so on.

Rather than create a whole new prefab for each enemy type, you can just create a base enemy type with all the common features, and then create a variant for each different type of enemy.

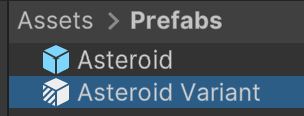
1. **Creating a Prefab Variant**

Let’s create a Prefab Variant based on the **Asteroid** prefab.

1. **Right-click** the existing **Asteroid** prefab in the **Project** window.
2. Select **Create Prefab Variant**.



1. Name the new variant **Asteroid\_Variant** (if not done automatically by Unity).



1. **Double-click** the variant to open it in the Prefab Editor.
2. **Using Prefab Variants**

Now that we have a Prefab Variant, we can modify it while still keeping a connection to the original prefab.

**Practical Examples**

**Example 1: Changing the Sprite**

1. Select **Asteroid\_Variant** in the Prefab Editor.
2. Change its **Sprite Renderer** to use **Asteroid\_02.png** instead of the original.
3. Save and return to the scene.
4. Drag both the **Asteroid** and **Asteroid\_Variant** into the scene. Notice that they look different, but they are still linked to the same base prefab.



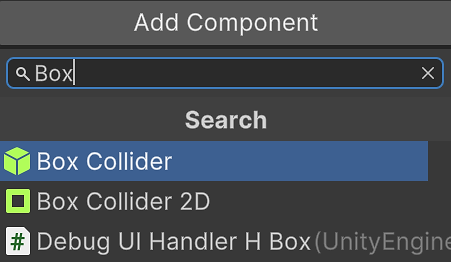
**Example 2: Adjusting Size and Color**

* 1. Select **Asteroid\_Variant**.
  2. Scale it up by setting **Transform → Scale** to (1.5, 1.5, 1).
  3. Change the **Sprite Renderer → Color** to a reddish tint.
  4. Save and return to the scene.

Now the variant renders a different sprite, but it still shares other properties with the original prefab.

**Example 3: Adding Extra Components**

1. Select **Asteroid\_Variant**.
2. Click **Add Component** and add a **Box Collider 2D**.



1. Save and return to the scene. This variant now has collision detection, while the original does not.

**Why Are Prefab Variants Important?**

* **Efficiency** – Easily create multiple versions of an object without duplicating work setting up a separate prefab for each version.
* **Inheritance** – Changes to the base prefab still apply to variants unless overridden, which means you can propagate (push) changes to the different versions just by changing the base prefab..
* **Flexibility** – Customize only the necessary properties while keeping the rest consistent, e.g. change the appearance of objects (such as different enemies) but keep the same base functionality (seeking the player, moving towards the player, etc.).

**Summary**



In this worksheet, you learned the fundamentals of **GameObjects** and **Prefabs** in Unity. You explored how to create and manage GameObjects, attach components to give them functionality, and use **Prefabs** to efficiently reuse objects in your scenes. You also discovered **Prefab Variants**, which allow you to create modified versions of a base prefab while still inheriting its properties.

Key takeaways:

* **GameObjects** are the building blocks of Unity scenes.
* **Components** define what a GameObject can do (e.g., rendering a sprite, detecting collisions, etc.).
* **Prefabs** allow you to create reusable objects for more efficient game development.
* **Prefab Variants** let you create different versions of an object while keeping a connection to the original.

Next, you’ll take your Unity knowledge further by learning how to add **script components** using **C#** to control GameObject behavior dynamically!

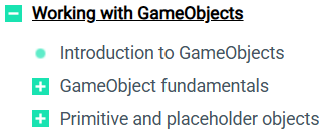
Eventually, working with Unity will become second nature to you, and you’ll look back at these worksheets as very simple and beginner-level!

Congratulations for getting so far!😊

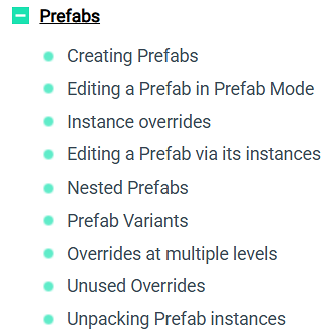
**Suggested Resources**

You should look for your own tutorials and resources, but here are a few good links for you to explore.

* [Unity Learn: Prefabs](https://learn.unity.com/tutorial/prefabs-e?uv=2019.4): Unity’s tutorial about using prefabs. It’s for an older version of Unity, but is still relevant.
* [Unity Manual: GameObjects](https://docs.unity3d.com/Manual/working-with-gameobjects.html):



* [Unity Manual: Prefabs](https://docs.unity3d.com/Manual/Prefabs.html):



* [Unity Official Tutorial – Prefabs](https://www.youtube.com/watch?v=H1OkG3a1w-o): A very short tutorial from Unity’s official YouTube channel.
* [Unity Learn: Nested Prefabs](https://learn.unity.com/tutorial/introduction-to-nested-prefabs-1?uv=2019.2): Nested prefabs aren’t covered in the worksheet, but it’s useful to know about this.
* [Ketra Games: A Beginner Level Introduction to Prefabs](https://www.youtube.com/watch?v=XYqxteBu3uI): A YouTube tutorial about using prefabs for a 2D shooter game.
* [Ketra Games: Easily Create Different Enemy Types in Unity Using Prefab Variants](https://www.youtube.com/watch?v=ivT8wpwhSzs): A YouTube tutorial about using prefab variants for a 2D shooter game, following on from the previous video.
* [Kodeco.com: GameObjects & Prefabs](https://www.kodeco.com/books/unity-apprentice/v1.0/chapters/3-gameobjects-prefabs): An excellent and respected tutorial site. This is for Unity 2020, but is still very much relevant.
* [What are prefabs in Unity?](https://www.youtube.com/watch?v=Tu7PMgR5DIk): A brief YouTube tutorial with examples.
* [Unity Unite Berlin 2018 - Technical Deep Dive into the New Prefab System](https://www.youtube.com/watch?v=J9WFcPbxOhQ): A long video presentation (59 minutes) by the Unity prefab system developers at Unity’s Unity conference. It’s from 2018, but is still relevant and very informative. These types of video are excellent, since they are professional developers talking to other professional developers. Some of the concepts might be a bit hard to follow at times, but it’s worth watching!