Unity Components

GAME PROGRAMMING
C# PROGRAMMING AND UNITY

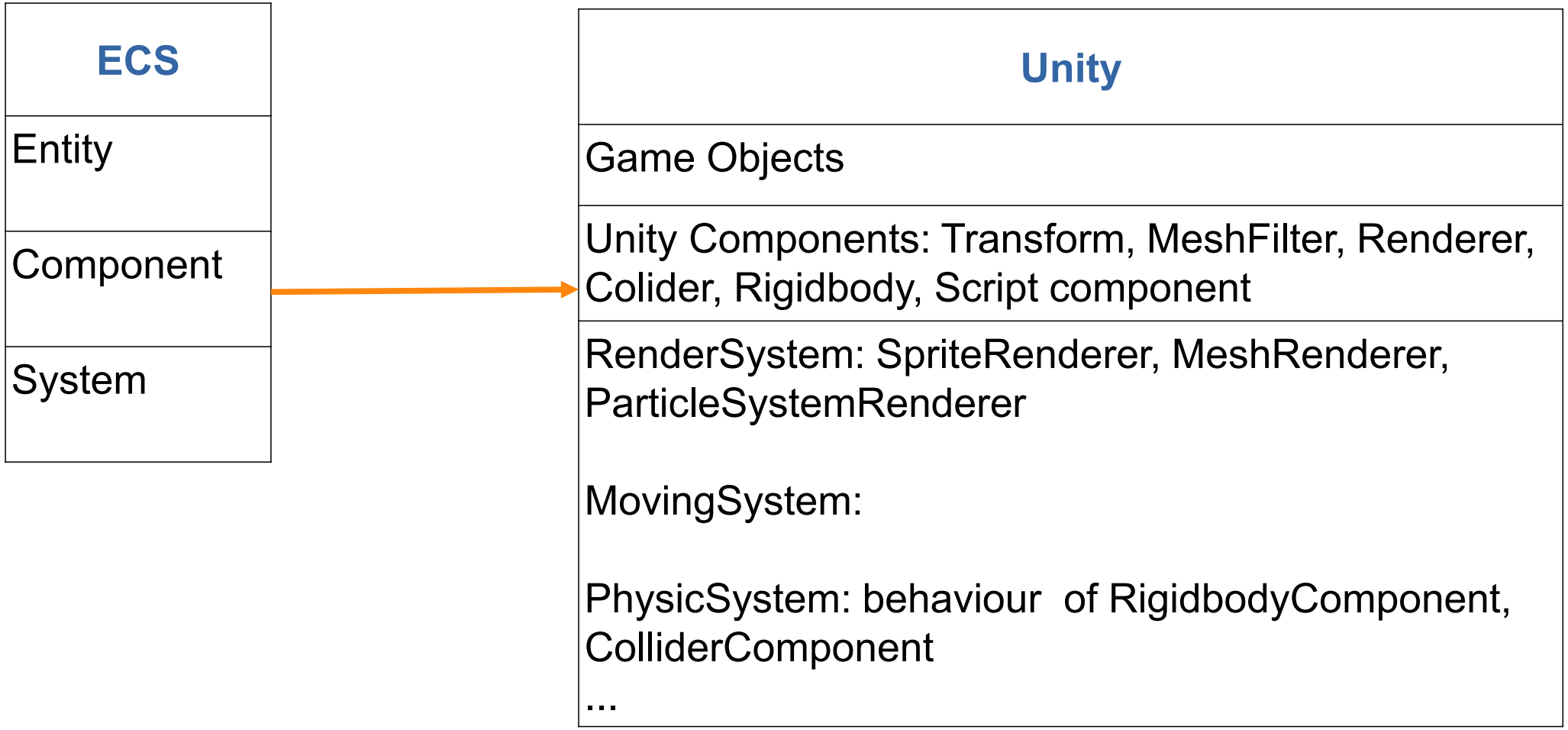
What is Unity's Component System

- The Entity Component System (ECS) is the core of the Unity Data-Oriented Tech Stack (DOTS)
- As the name indicates, ECS has three principal parts:
  - **Entities** — the entities, or things, that populate your game or program
  - **Components** — the data associated with your entities, but organized by the data itself rather than by entity. (This difference in organization is one of the key differences between an object-oriented and a data-oriented design)
  - **Systems** — the logic that transforms the component data from its current state to its next state - for example, a system might update the positions of all moving entities by their velocity times the time interval since the previous frame

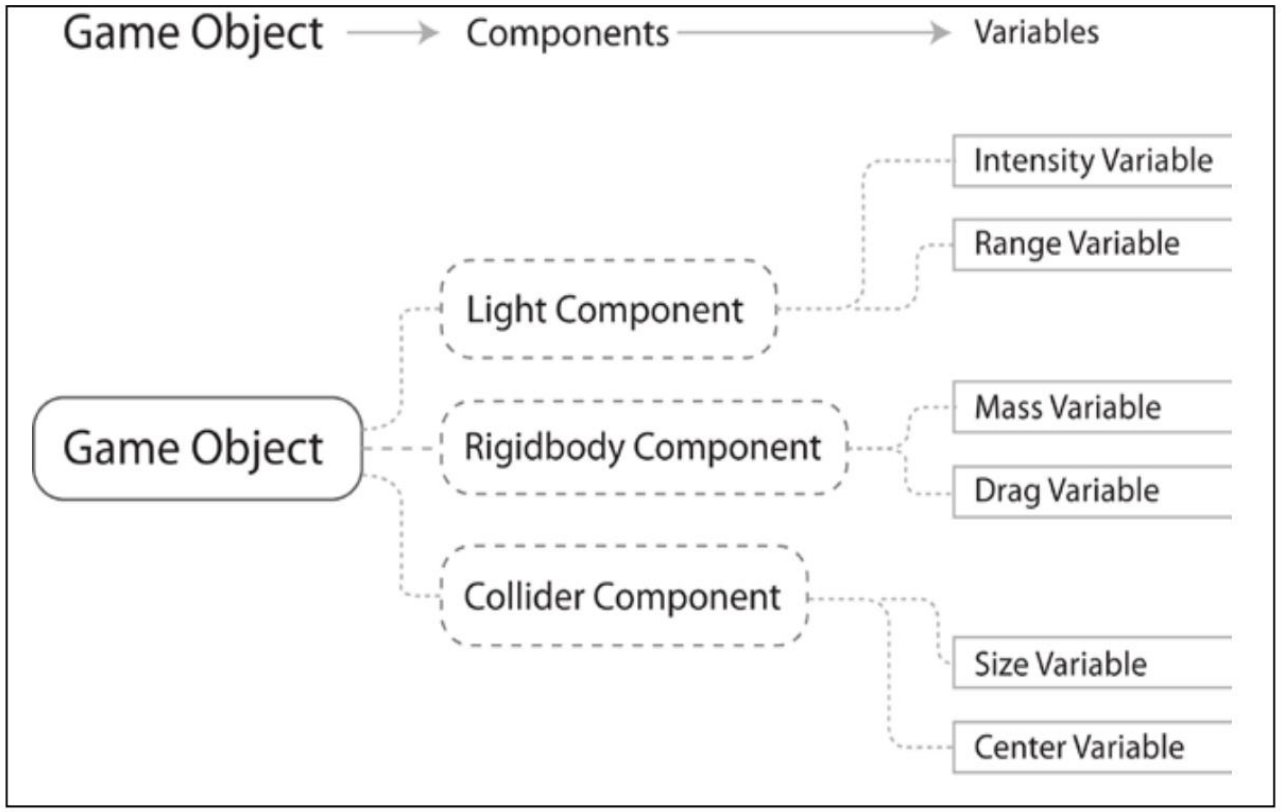
What is Unity's Component System

- In a component system, components are mix-and-match packets of functionality, and objects are built up as a collection of components, rather than as a strict hierarchy of classes.
- A component system is a different (and usually more flexible) approach to object-oriented programming (OOP) that constructs game objects through composition rather than inheritance
- In a component system, objects exist on a flat hierarchy, and different objects have different collections of components. An inheritance structure, in contrast, has different objects on completely different branches of the tree.
- The component arrangement facilitates rapid prototyping, because you can quickly mix and match components rather than having to refactor the inheritance chain when objects change

What is Unity's Component System



Components in Unity



Components are one of the three principle elements of an Entity Component System architecture. They represent the data of your game or program

- In Unity, components come in various forms. They can be for creating behavior, defining appearance, and influencing other aspects of an object's function in the game
- By attaching components to an object, you can immediately apply new parts of the game engine to your object

Components in Unity

- Common components of game production come built-in with Unity, such as the Rigidbody component mentioned earlier, down to simpler elements such as lights, cameras, particle emitters, and more
- To build further interactive elements of the game, you'll write scripts, which are also treated as components in Unity. Try to think of a script as something that extends or modifies the existing functionality available in Unity or creates behavior with the Unity scripting classes provided

Common Components in Unity

Components	Using / Datastore
Transform	Position, Rotation, scale
Renderer	Object appear on the screen
Collider	Define the shape of an object for the purposes of physical collisions
Rigidbody	Add motion to GameObject under the control of Unity's physics engine
Script	All code execution in Unity starts from code files linked to an object in the scene Scripts in Unity are more akin to individual OOP classes, and scripts attached to objects in the scene are object instances

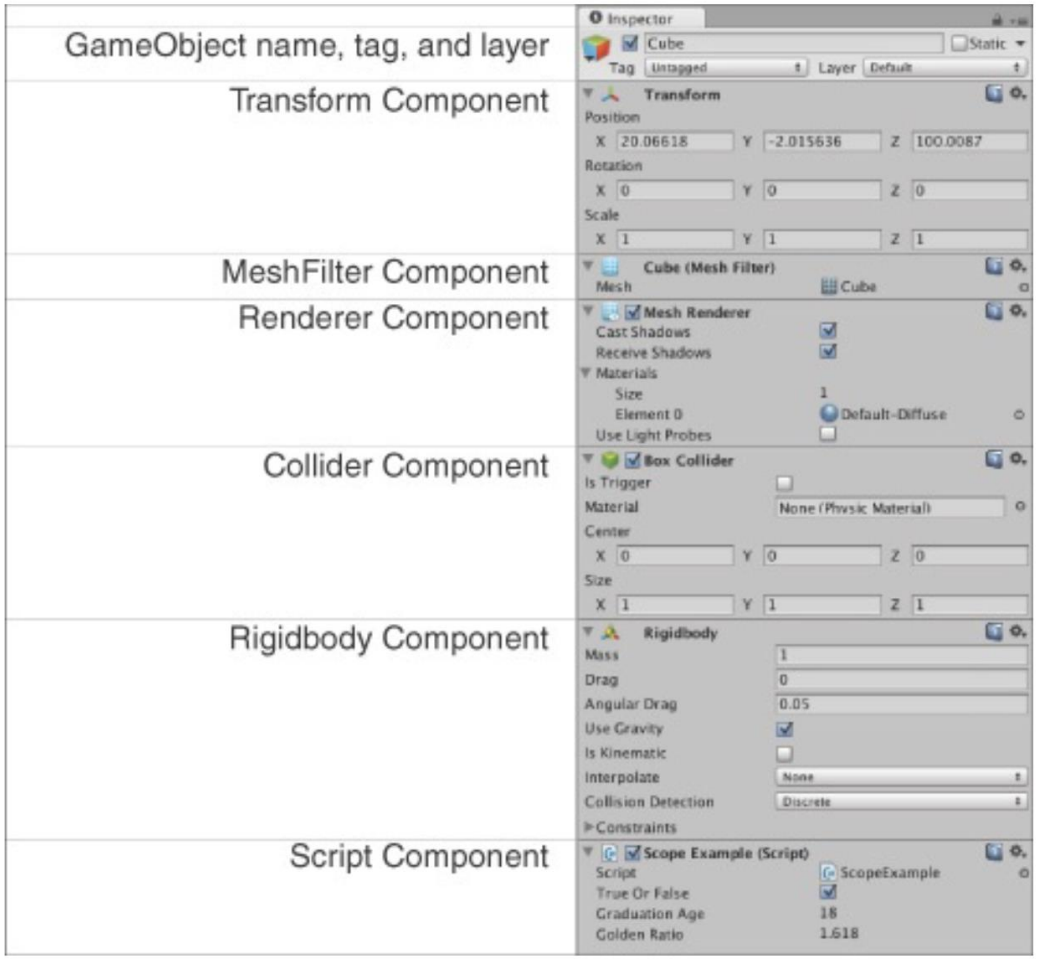


Figure 20.1 The Inspector pane showing various important components

The Transform Component

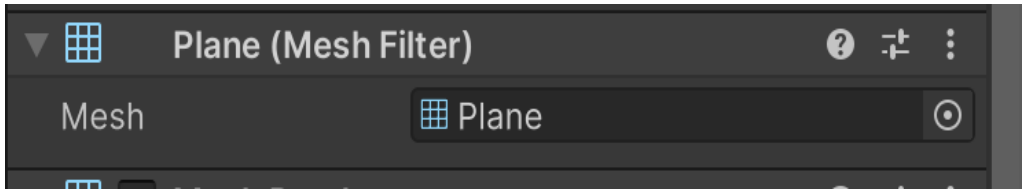
Transform: Position, Rotation, and Scale

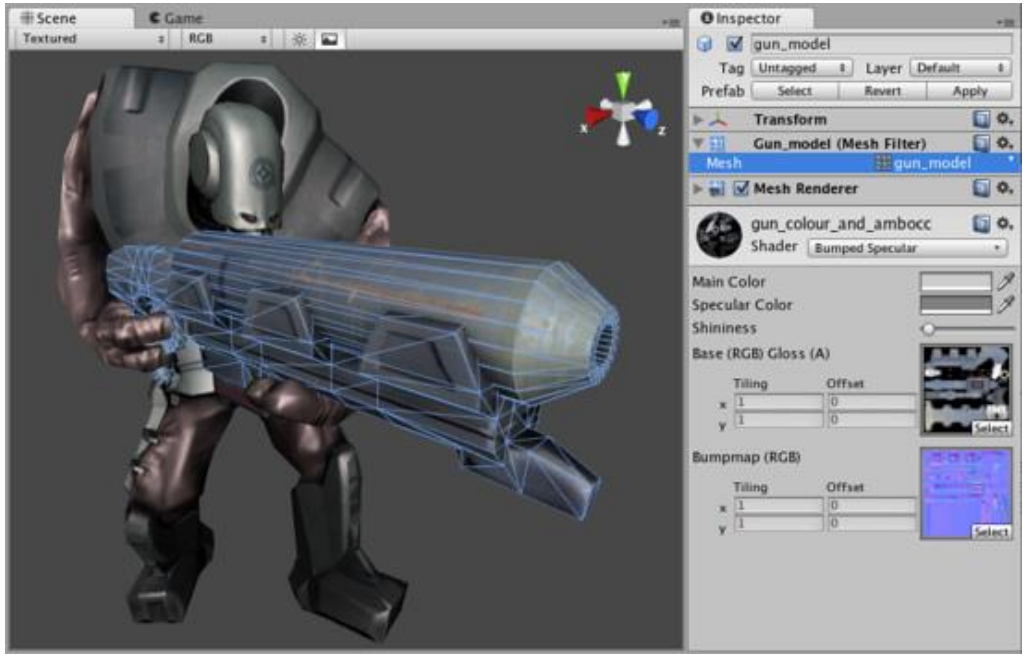
- Transform is a mandatory component that is present on all GameObjects
- Transform handles critical GameObject information like position (the location of the GameObject), rotation (the orientation of the GameObject), and scale (the size of the GameObject)
- Though the information is displayed in the Inspector pane, Transform is also responsible for the parent/child relationships in the Hierarchy pane. When one object is the child of another, it moves with that parent object as if attached to it

The MeshFilter Component

MeshFilter: The Model You See

- A MeshFilter component attaches a 3D mesh in your Project pane to a GameObject
- To see a model on screen, the GameObject must have both a MeshFilter that handles the actual 3D mesh information and a MeshRenderer that combines that mesh with a shader or material and displays the image on screen
- The MeshFilter creates a skin or surface for a GameObject, and the MeshRenderer determines the shape, color, and texture of that surface

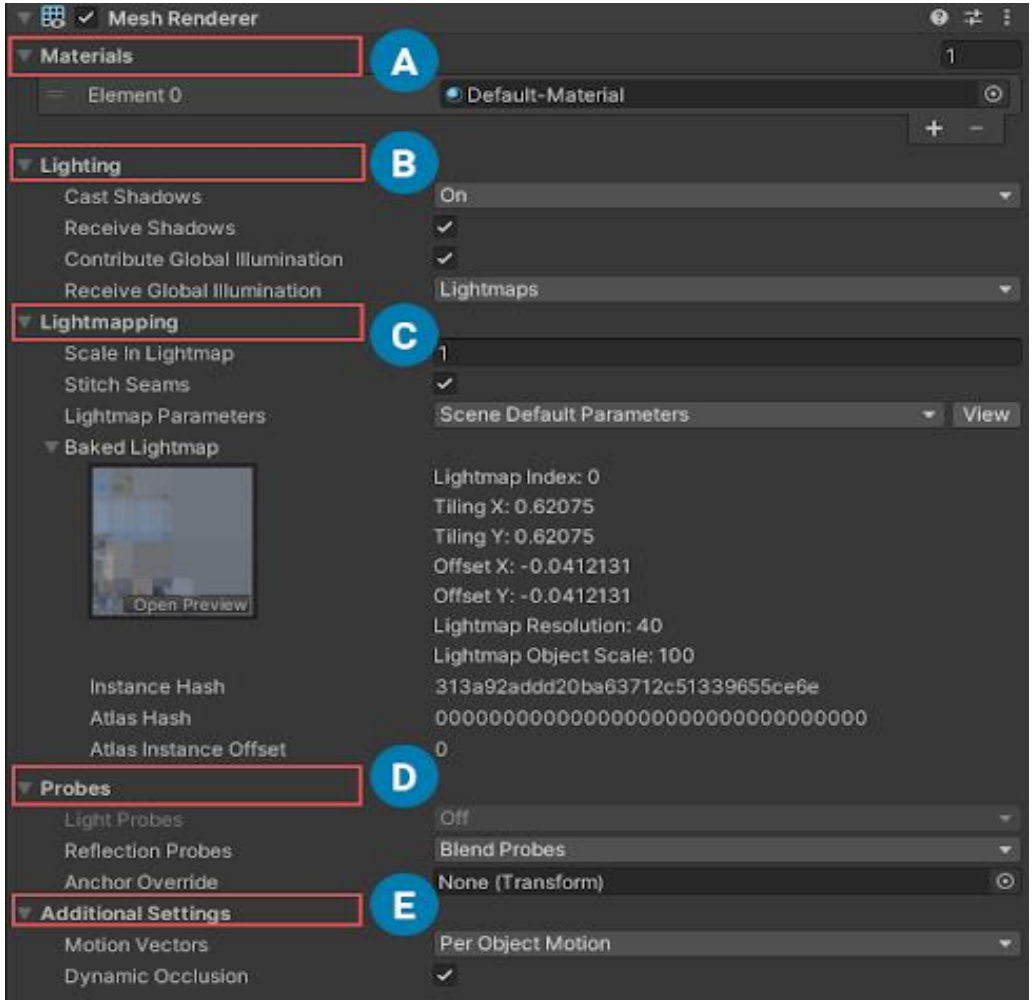




https://www.youtube.com/watch?v=m3pblzmvFpE

The Renderer Component

Renderer: Allows You to See the GameObject



- A Renderer component - in most cases, a MeshRenderer - allows you to see the GameObject in the Scene and Game panes
- The MeshRenderer requires a MeshFilter to provide 3D mesh data as well as at least one Material if you want it to look like anything other than an ugly magenta blob (Materials apply textures to objects, and when no Material is present, Unity defaults to solid magenta to alert you to the problem)

Renderers bring the MeshFilter, the Material(s), and lighting together to show the GameObject on screen

The Rigidbody Component

Rigidbody: The Physics Simulation

- The Rigidbody component controls the physics simulation of your GameObject
- The Rigidbody component simulates acceleration and velocity every FixedUpdate (generally every 50th of a second) to update the position and rotation of the Transform component over time. It also uses the Collider component to handle collisions with other GameObjects
- The Rigidbody component can also model things like gravity, drag, and various forces like wind and explosions
- Set isKinematic to true if you want to directly set the position of your GameObject without using the physics provided by Rigidbody

The Rigidbody Component

- Configure the Rigidbody
  - Mass: Sets the mass of the Rigidbody. Higher mass makes the object harder to move or affect by external forces
  - Linear Drag: Applies a damping force to slow down the Rigidbody's linear motion. Higher drag values result in slower movement
  - Angular Drag: Applies a damping force to slow down the Rigidbody's angular rotation. Higher angular drag values result in slower rotation
  - Gravity scale: Determines how much gravitational force affects the object
  - Constraints: You can limit the movement of the Rigidbody along certain axes by checking the corresponding checkboxes

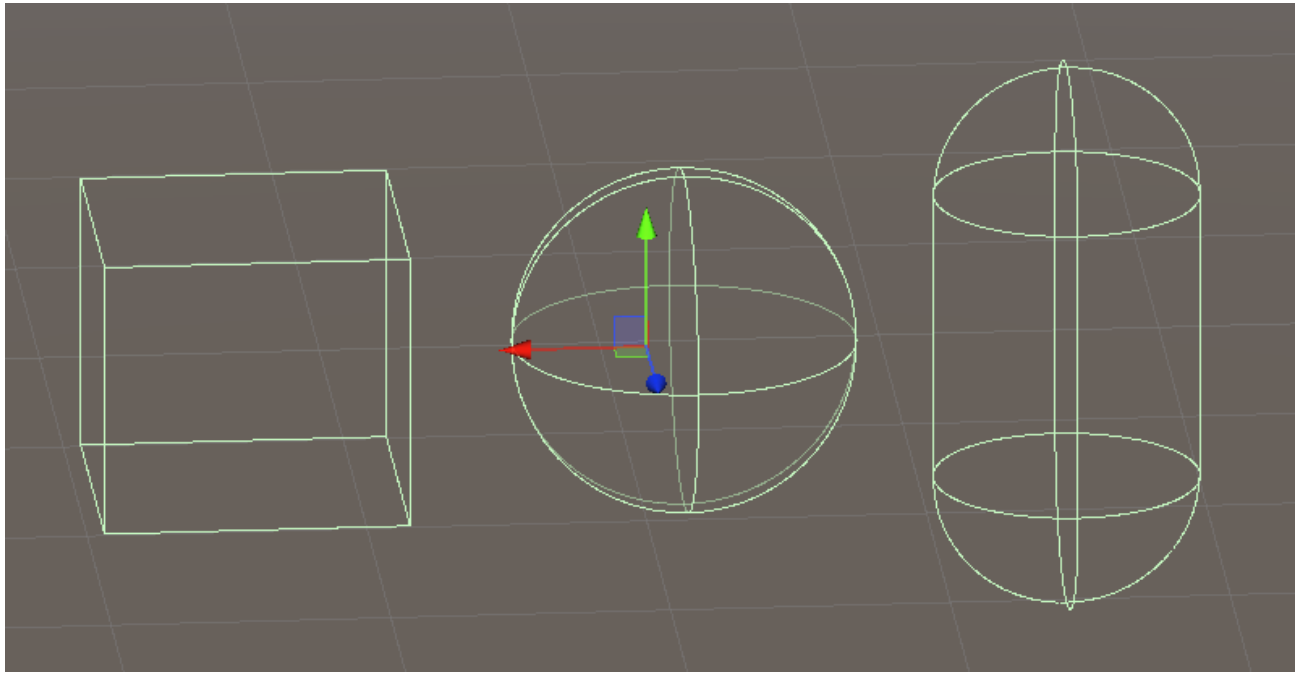
The Collider Component

Collider: The Physical Presence of the GameObject

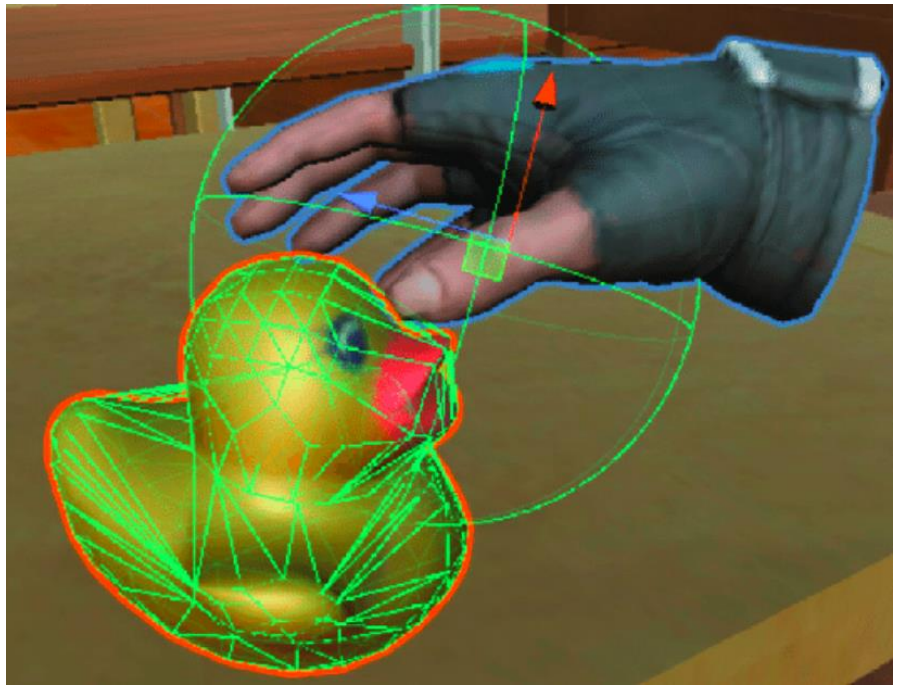
- A Collider component enables a GameObject to have a physical presence in the game world and collide with other objects
- Unity has four different kinds of Collider components, which ve arranged below in order of their speed. Calculating whether another object has collided with a Sphere Collider is extremely fast, but calculating whether an object has collided with a MeshCollider is much slower:
  - Sphere Collider: The fastest collision shape to calculate. A ball or sphere
  - Capsule Collider: A pipe with spheres at each end. The second fastest type
  - Box Collider: A rectangular solid. Useful for crates and other boxy things
  - Mesh Collider: A collider formed from a 3D mesh. Although useful andaccurate, mesh colliders are much, much slower than any of the other three.Also, only Mesh Colliders with Convex set to tr ue can collide with otherMesh Colliders

The Collider Component

- Colliders are components which provide collision detection through the various green lines (or boundaries)
- They define the shape of game objects for the purposes of physical collision

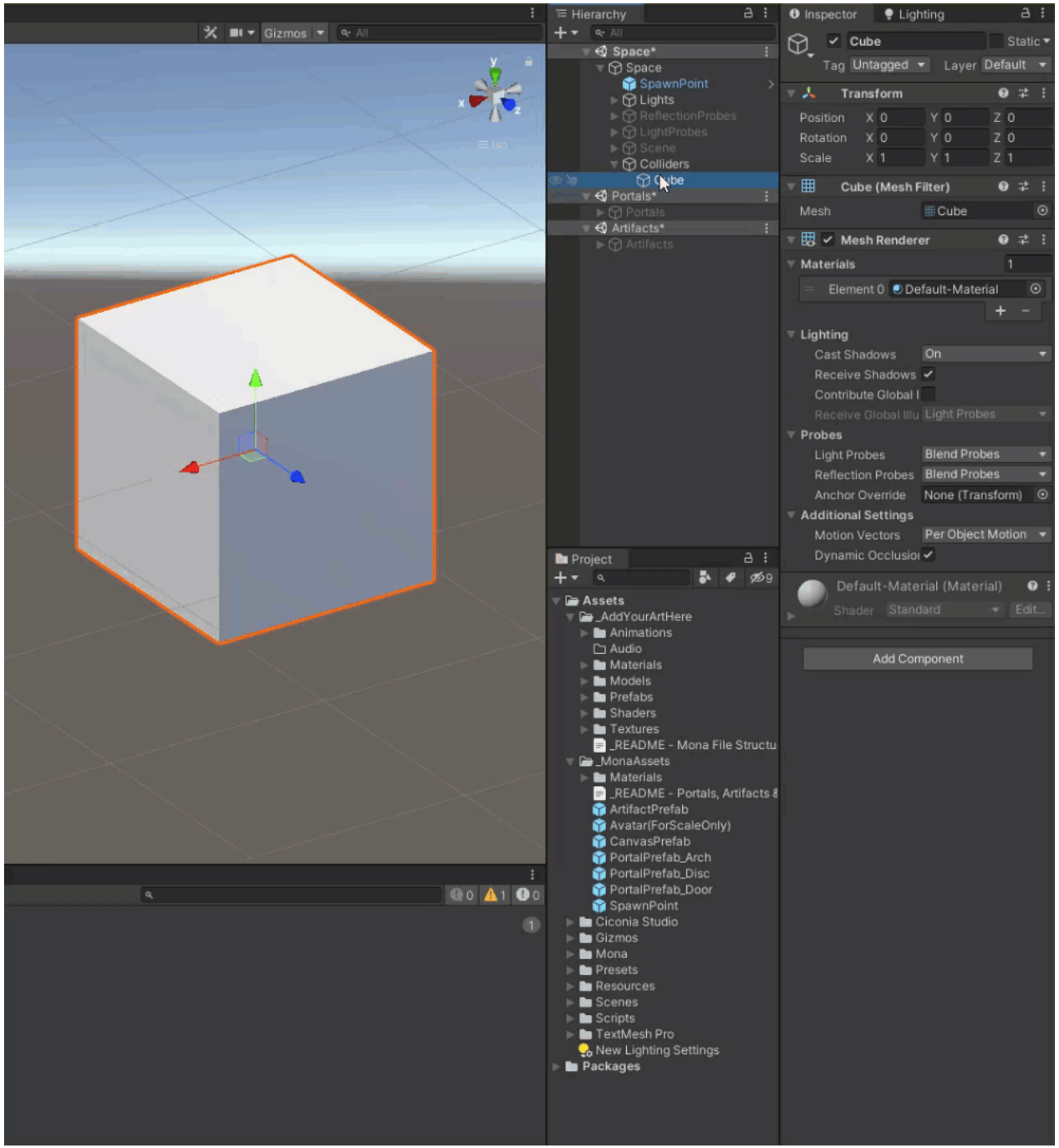


Box Collider Sphere Collider Capsule Collider

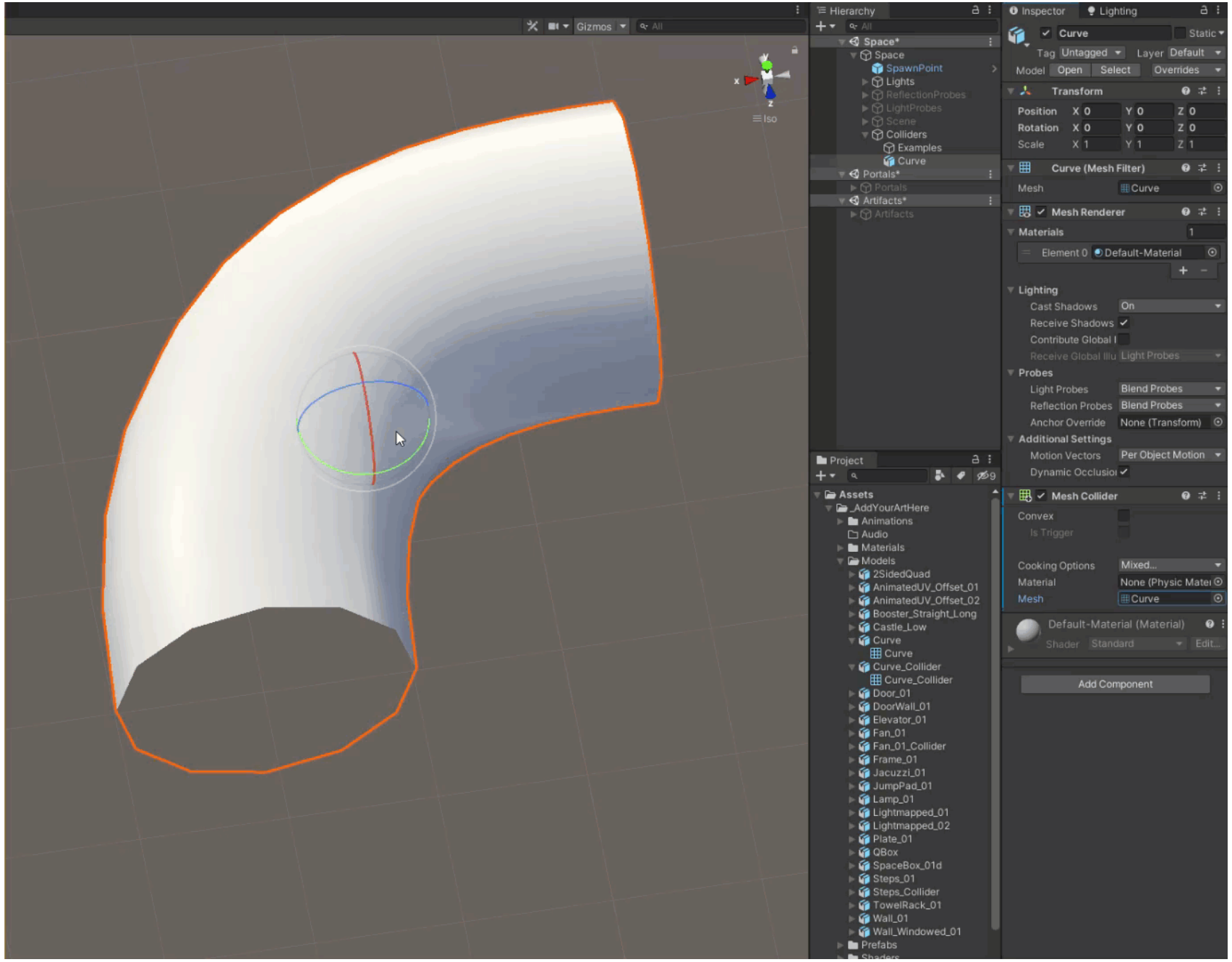


Mesh Collider

The Collider Component



The Collider Component



Script Components

Runmning code in Unity: Script components

- All code execution in Unity starts from code files linked to an object in the scene
- Ultimately, this code execution is all part of the component system described earlier; game objects are built up as a collection of components, and that collection can include scripts to execute

Listing 1.1 Code template for a basic script component

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HelloWorld : MonoBehaviour {
    void Start() {
        // do something once
    }

    void Update() {
        // do something every frame
    }
}
```

Include namespaces for Unity and .NET/Mono classes.

The syntax for inheritance

Put code here that runs once.

Put code here that runs every frame.

Script Components

- As you've probably surmised from this description, in Unity, scripts are components - not all scripts, mind you, only scripts that inherit from MonoBehaviour, the base class for script components
- MonoBehaviour defines the invisible groundwork for attaching components to game objects, and inheriting from it provides a couple of automatically run methods that you can implement
- Those methods include Start(), called once when the object becomes active (which is generally as soon as the scene with that object has loaded), and Update(), which is called everyframe. Your code is run when you put it inside these predefined methods

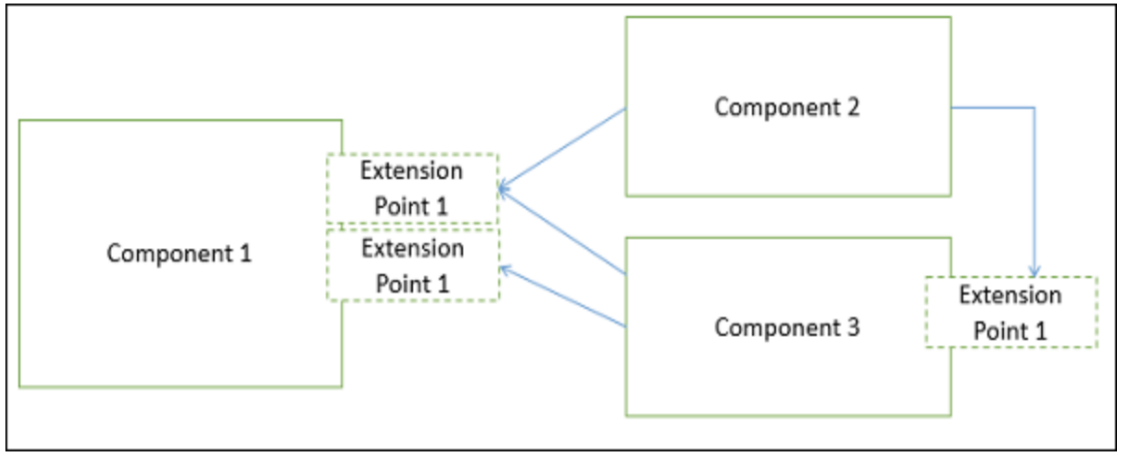
Advantages of Component-Based Design (CBD)

There are many advantages of developing the applications using the CBD:

1.Ease of deployment − As new compatible versions become available, it is easier to replace existing versions with no impact on the other components or the system as a whole

2.Reduced cost − The use of third-party components allows you to spread the cost of development and maintenance

3.Ease of development − Components implement well-known interfaces to provide defined functionality, allowing development without impacting other parts of the system



Advantages of Component-Based Design (CBD)

4. **Reusable** − The use of reusable components means that they can be used to spread the development and maintenance cost across several applications or systems
5. **Modification of technical complexity** − A component modifies the complexity through the use of a component container and its services
6. **Reliability** − The overall system reliability increases since the reliability of each individual component enhances the reliability of the whole system via reuse
7. **System maintenance and evolution** − Easy to change and update the implementation without affecting the rest of the system
8. **Independent** − Independency and flexible connectivity of components. Independent development of components by different group in parallel. Productivity for the software development and future software development

Practice – Assets folder structure

Hint: In order to organize the Assets folder, create the following folders:

Example 1

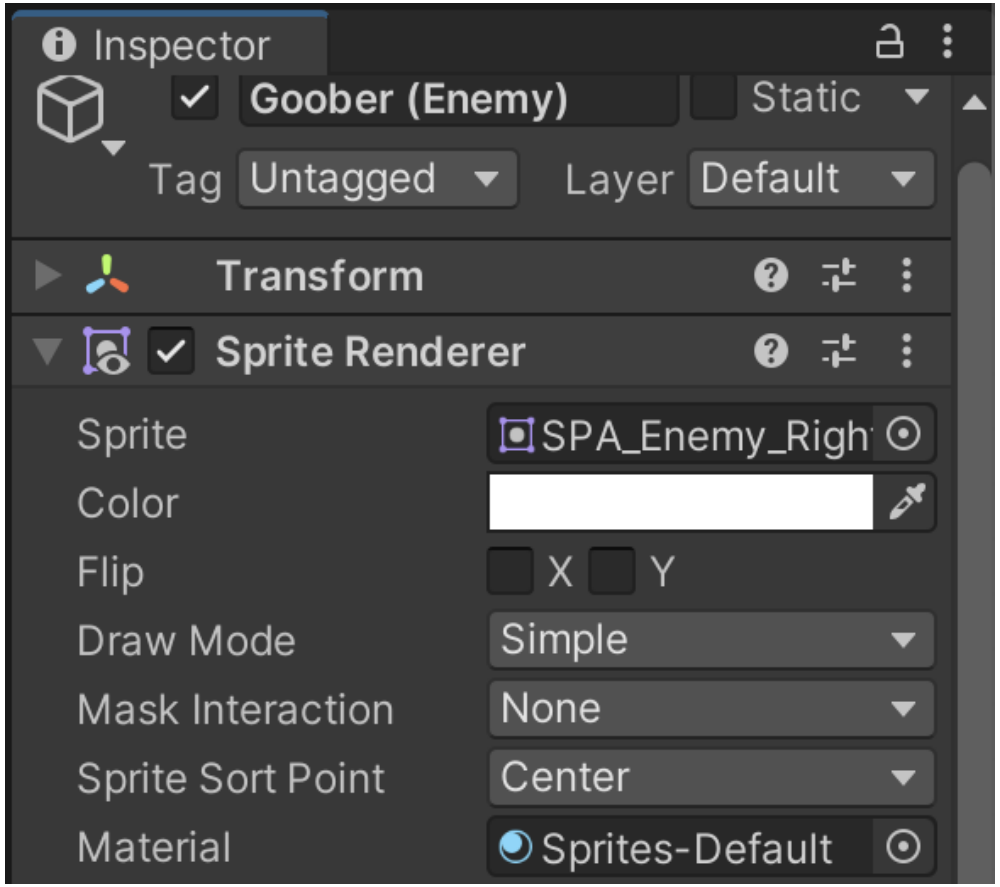
```
Assets
+---Art
| +---Materials
| +---Models
| +---Textures
+---Audio
| +---Music
| \---Sound
+---Code
| +---Scripts  # C# scripts
| \---Shaders  # Shader files and shader graphs
+---Docs   # Wiki, concept art, marketing material
+---Level   # Anything related to game design in Unity
| +---Prefabs
| +---Scenes
| \---UI
```

Practice 1 – Create an enemy

1.Create an empty game object, reset its transform, and rename it Goober (Enemy)

2.Add the Sprite Renderer component to Goober

- Drag and drop an image (SPA_Enemy_Right.png) into the Sprite field
- Set the Sorting Layer to ‘Enemies’



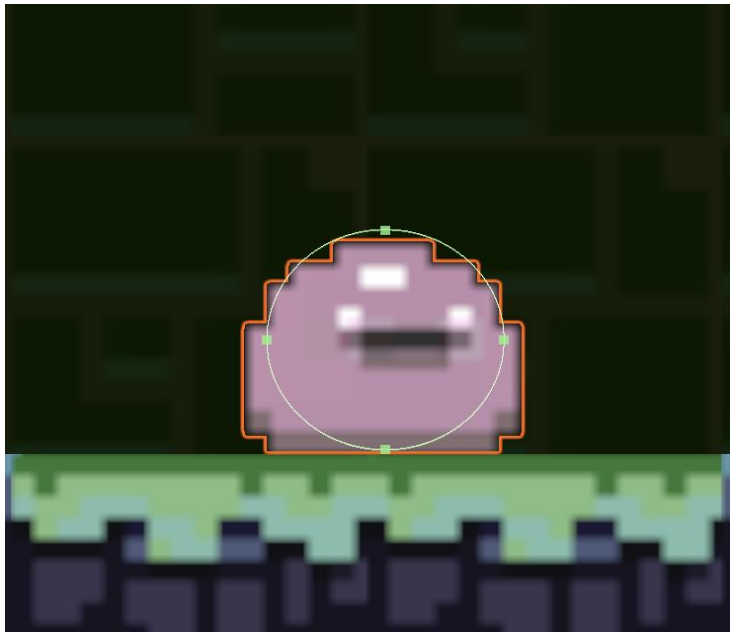
Practice 1 – Create an enemy

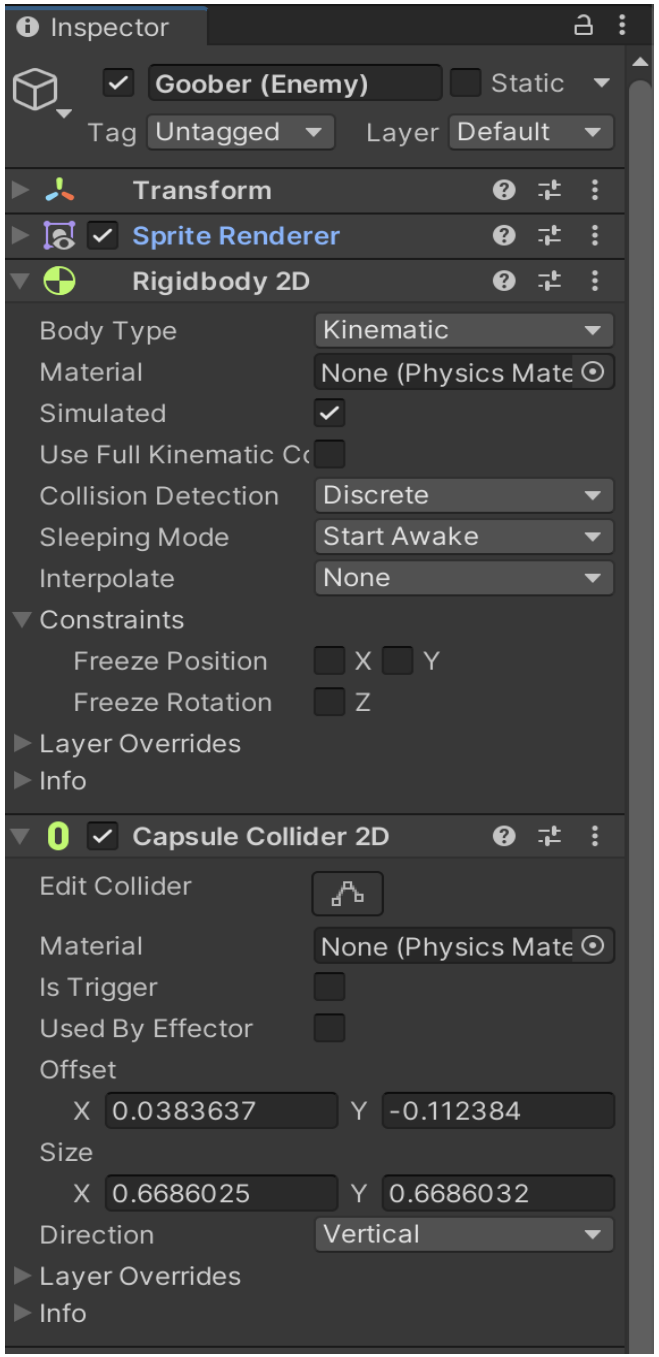
3.Add Rigidbody 2D component to Goober

- Body Type: Kinematic

4.Add Capsule Collider 2D component to Goober

- Click 'Edit Collider' and adjust the collider's scale

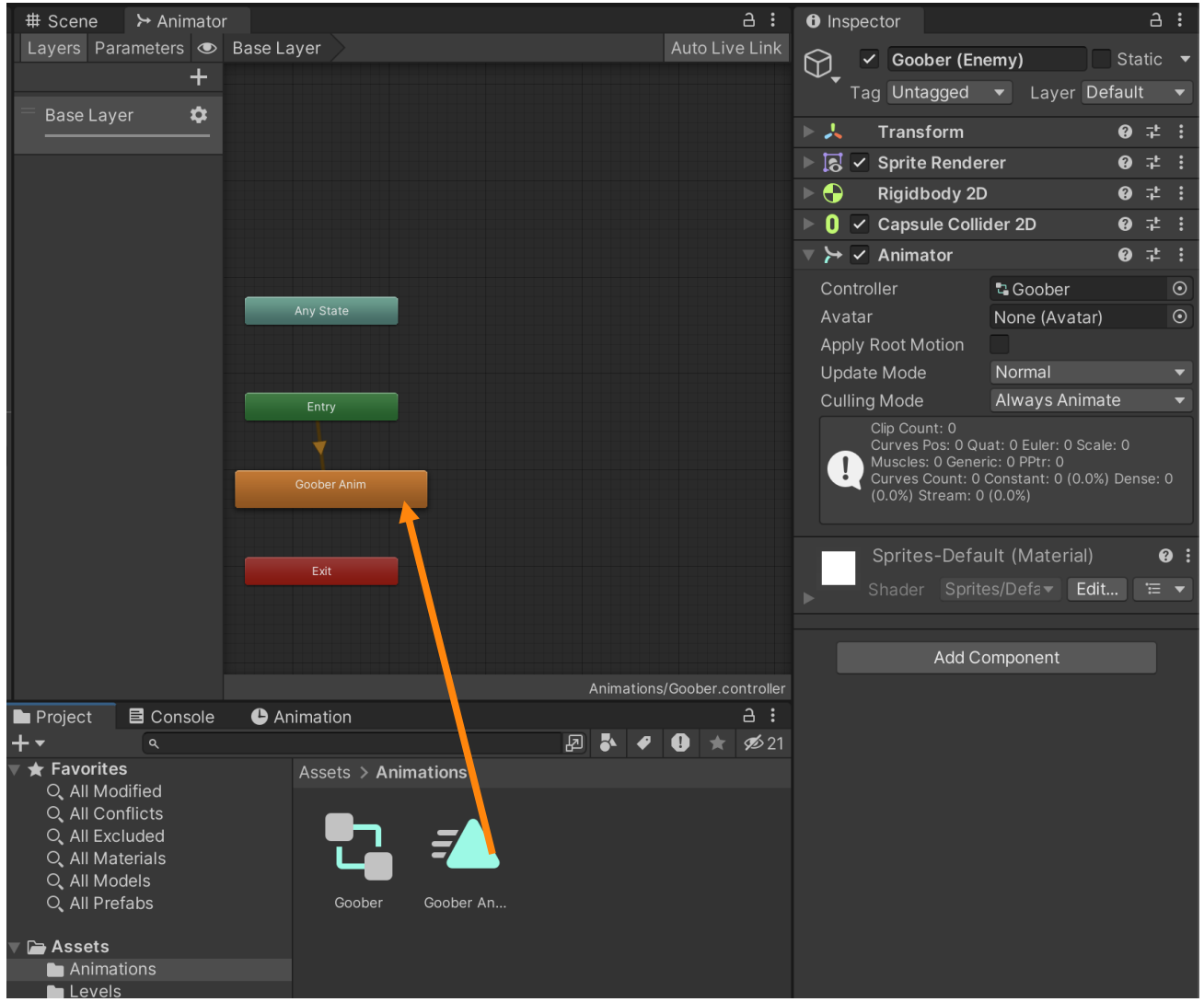




Practice 1 – Create an enemy

5.Add an Animator component to 'Goober' (optional)

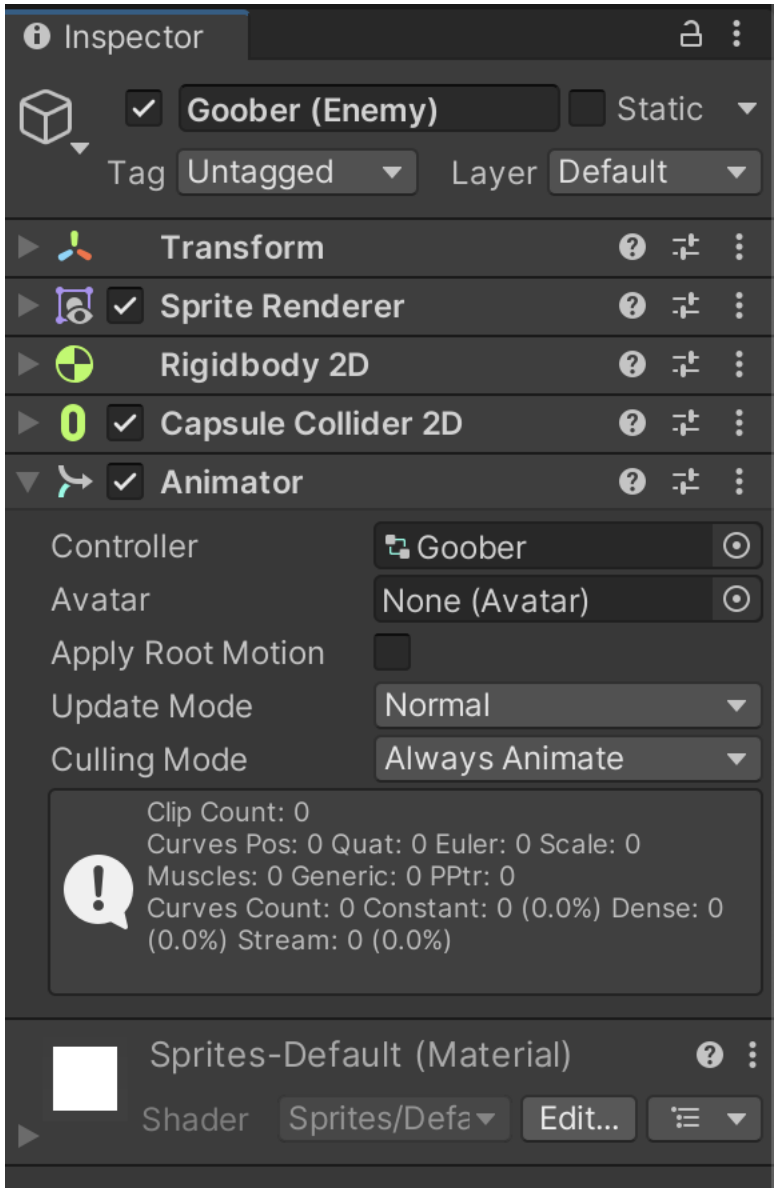
- In the **Assets/Animations** folder, right-click and select **Create > Animator Controller**
- Open the Animator window (navigate to **Window > Animation > Animator**). Rename the controller to **Goober Controller** (**Goober.controller**, also referred to as **Goober Animator Controller**)
- In the Animator window, drag and drop the downloaded **Goober Animation** from the **Assets > Animations** folder onto the Animator panel



Practice 1 – Create an enemy

6.Add an Animator component to 'Goober' (optional)

- Select the **Goober** GameObject and add the Animator Component
- Next, drag and drop the **Goober Animator Controller** into the **Controller** field of the Animator Component

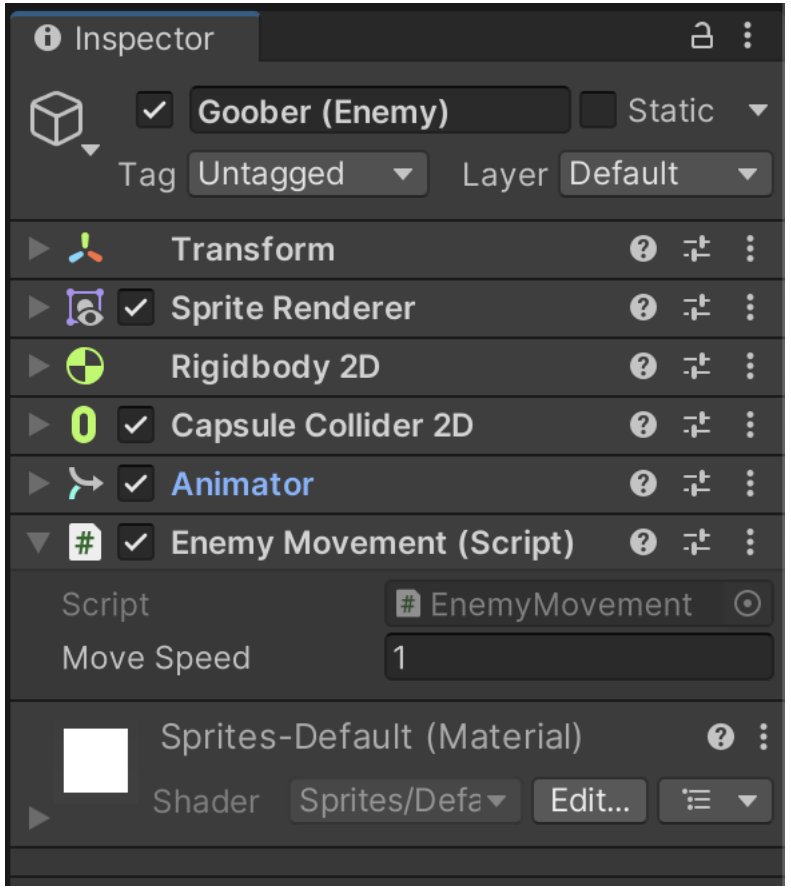


Practice 1 – Create an enemy

7.Add an Script component to 'Goober'

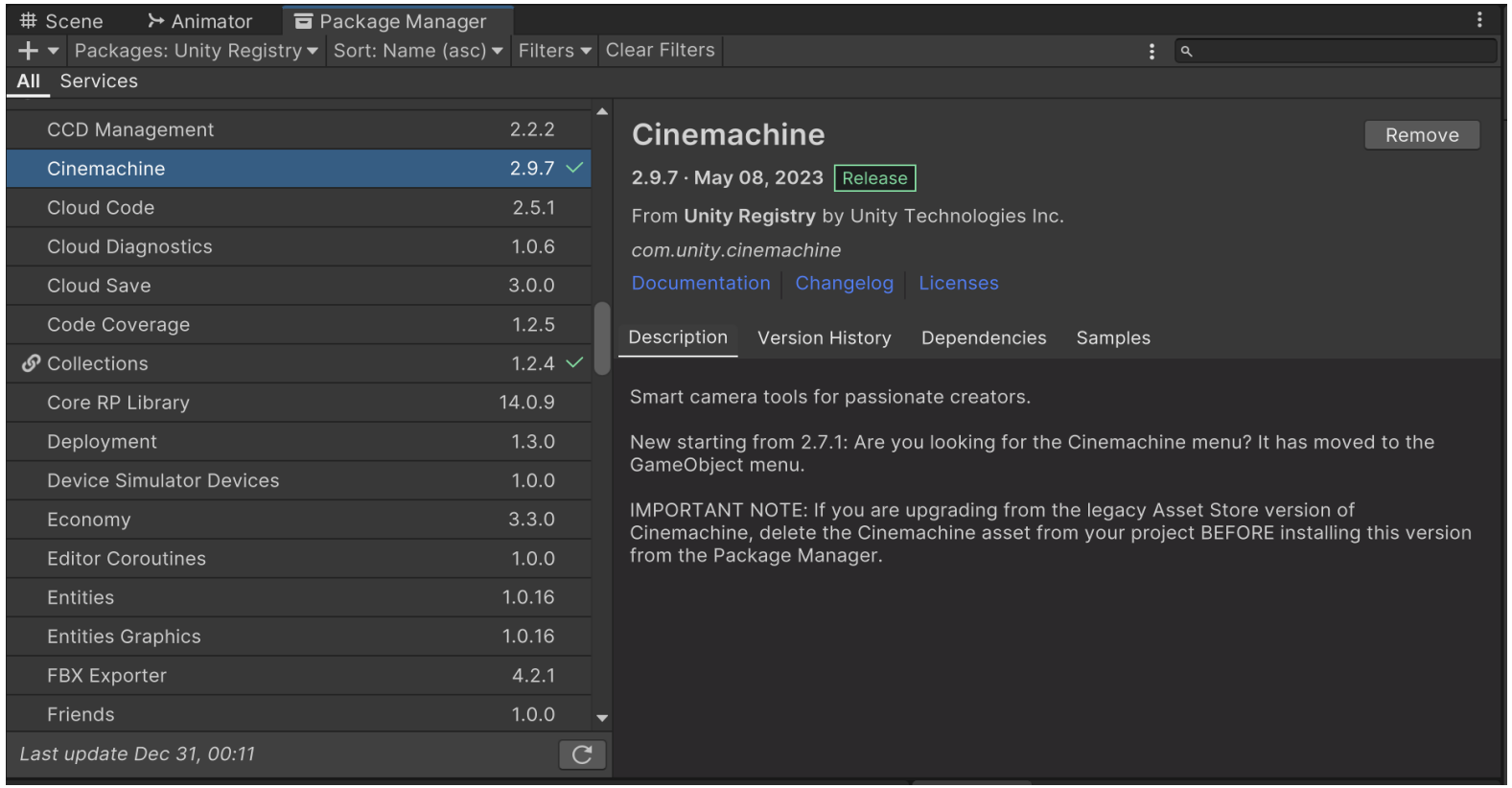
- Select **the Goober** GameObject, drag and drop the **EnemyMovement.cs** script into the Inspector window

8.Press Play and test the movement of Goober



Practice 2 – Add Cinemachine Follow Camera

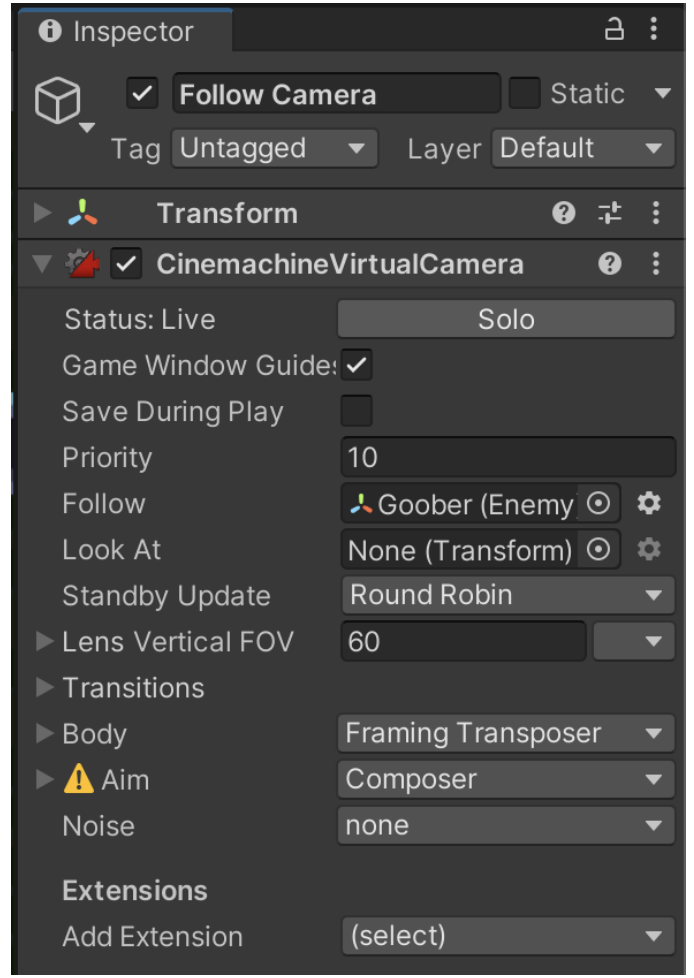
1. **Import Cinemachine:** Start by ensuring you've installed the Cinemachine package in your Unity project. You can do this using the Package Manager (**Window > Package Manager > Cinemachine**)



Practice 2 – Add Cinemachine Follow Camera

2.Create a Virtual Camera: Right-click in the Hierarchy window, select **Cinemachine > Virtual Camera**. This creates a new Cinemachine virtual camera

- Body: Framing Transposer
- Follow: Goober (For testing. In real game, Virtual Camera should follow the player)



3.Press Play and test the virtual camera