**Chapter 1:**

**Introduction to Immersive Technology**

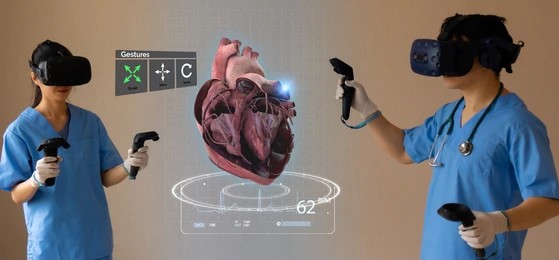
* 1. **What is Immersive Technology**

Immersive technology creates engaging digital experiences by making users feel present in a simulated or augmented environment. Unlike traditional screens, it allows interaction with virtual elements as if they were real.

The main types of immersive technologies include **Virtual Reality (VR)**, **Augmented Reality (AR)**, and **Mixed Reality (MR)**, which together are known as **Extended Reality (XR)**.

### 1.1.1. Virtual Reality (VR)

**Definition:**  
Virtual Reality is a fully immersive technology that places the user inside a completely digital environment, shutting out the real world.



**Image 1.1.1: Medical Training Using Virtual Reality: Collaborative 3D Heart Simulation**

**Example:**

* Playing VR games such as Beat Saber.
* Using flight simulators for pilot training.

**Devices Used:**

* VR Headsets (Oculus Quest, HTC Vive, PlayStation VR)
* Motion Controllers and Haptic Gloves

### 1.1.2. Augmented Reality (AR)

**Definition:**  
Augmented Reality overlays digital information, such as images, sounds, or 3D models, onto the real-world environment.



**Image 1.1.2: Augmented Reality in Retail: Interactive Virtual Shopping Experience**

**Example:**

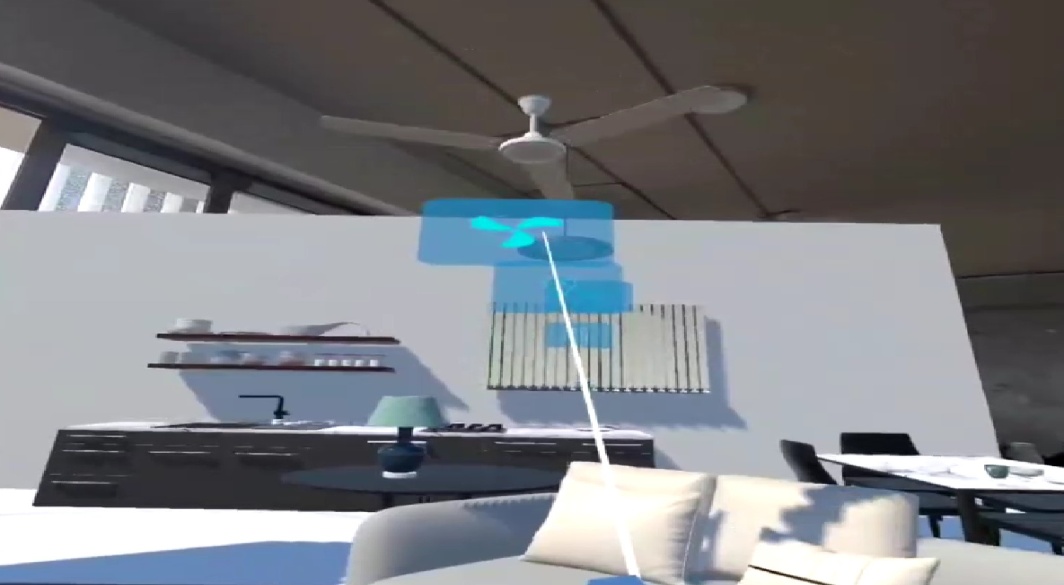
* Pokémon GO mobile game.
* IKEA AR app to place virtual furniture in real rooms.

**Devices Used:**

* Smartphones and Tablets with AR apps
* AR Glasses (Microsoft HoloLens, Google Glass, Magic Leap One)

### 1.1.3. Mixed Reality (MR)

**Definition:**  
Mixed Reality merges real and virtual environments, enabling real-time interaction between physical and digital objects.



**Image 1.1.3: Mixed Reality Application: Smart Home Device Control Interface**

**Example:**

* Engineers using MR to interact with 3D digital prototypes.
* Medical students practicing surgery with holographic organs.

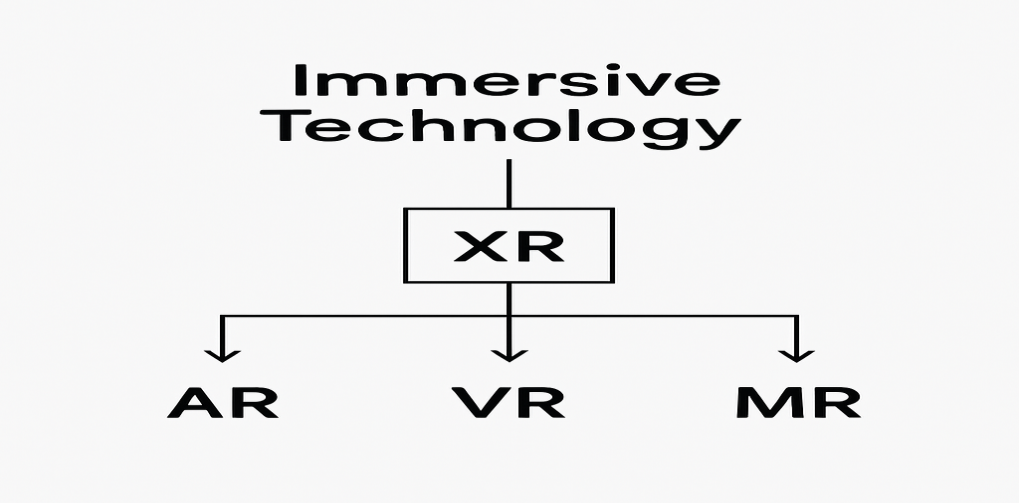
**Devices Used:**

* MR Headsets (Microsoft HoloLens 2, Magic Leap)
* High-performance MR-enabled PCs and sensors

### 1.1.4. Extended Reality (XR)

**Definition:**  
Extended Reality is the umbrella term that covers **VR, AR, and MR**.

**XR = VR + AR + MR**

****

**Applications of XR:**

* Education and Training
* Healthcare (surgical simulations, patient care)
* Manufacturing and Design
* Gaming and Entertainment
* Marketing and Remote Collaboration

**1.2 Why AR/VR skills are important**

AR/VR skills are enable immersive, interactive learning and training experiences that improve knowledge retention, practical skill application, and safety. These technologies facilitate hands-on practice in a risk-free environment, enhancing both soft and technical skills across industries like healthcare, retail, construction, and education. Additionally, AR/VR skills prepare individuals for future technology trends, offer a competitive advantage in the job market, and foster creativity and innovation in developing new applications and solutions.

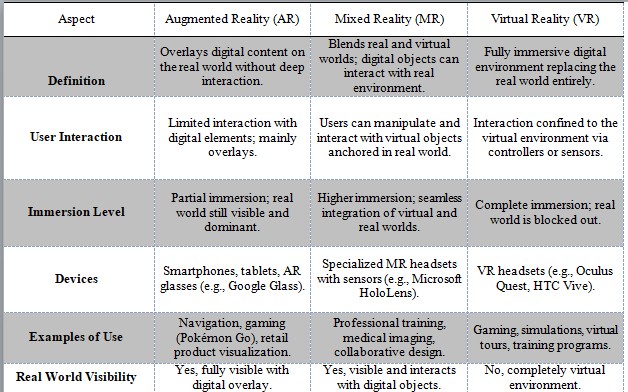
## 1.2.1 Key Reasons AR/VR Skills Matter

* **Immersive Learning & Training**: AR/VR creates environments where users can practice complex tasks, simulations, or scenarios that may be costly, dangerous, or hard to replicate in reality, leading to better skill development and retention.
* **Safe Practice Environment**: Mistakes made in VR are consequence-free, enabling users to build confidence and competence before applying skills in real-world situations.
* **Wide Industry Usage**: From healthcare surgeries to retail customer handling and engineering safety training, AR/VR skills enable professionals to adapt and excel in various fields.
* **Enhanced Engagement and Retention**: Learning with AR/VR is interactive and multisensory, which improves learner focus, motivation, and long-term retention of knowledge.
* **Future-Proofing Careers**: As AR/VR technologies evolve and integrate into more sectors, skills in these areas ensure individuals remain valuable and competitive in the rapidly changing tech job market.
* **Creativity and Innovation**: AR/VR training fosters out-of-the-box thinking, helping skills development in designing new tools, games, simulations, and marketing strategies.

## 1.2.2 Benefits for Organizations and Individuals

* Increased training effectiveness and reduced training costs.
* Improved product development, production, and remote collaboration.
* Ability to visualize and prototype designs virtually, identifying flaws early.
* Enhanced customer experiences through AR applications in retail and marketing.

**1.3 Difference between AR, VR, and MR**



**Table 1.2.1: Difference between AR, MR, VR**

**1.4 Summary**

This chapter introduces **Immersive Technologies** such as **Virtual Reality (VR), Augmented Reality (AR), Mixed Reality (MR), and Extended Reality (XR)**. It explains how these technologies create engaging digital experiences by blending or replacing the real world. Examples include VR gaming, AR mobile apps, and MR engineering prototypes. The chapter also highlights the **importance of AR/VR skills**, showing how they enable safe training, improve knowledge retention, and are widely applied in industries like **healthcare, education, retail, and manufacturing**. Finally, it compares AR, VR, and MR with clear definitions, use cases, and devices used.

**Chapter 2:**

**Introduction to Unity for AR/VR**

**2.1 What is Unity?**

Unity is a powerful and versatile cross-platform game engine used to create interactive 2D and 3D experiences such as games, simulations, and virtual/augmented reality applications. It offers a component-based architecture where game objects gain functionality through attachable components like visuals, physics, and scripts, primarily written in C#. The engine supports a wide range of platforms including mobile, desktop, consoles, VR, and AR devices.

****

**Image 2.1.1: Unity SDK logo.**

**2.2 Unity Editor Overview**

### 2.2.1 Scene View

### The Scene View is a visual workspace where developers construct and manipulate the game environment or virtual world. It displays the 3D or 2D representations of game objects, allowing users to move, rotate, and scale assets directly. This view provides essential spatial context for designing levels, setting up lighting and placing interactive elements.

### 2.2.2 Hierarchy Plane

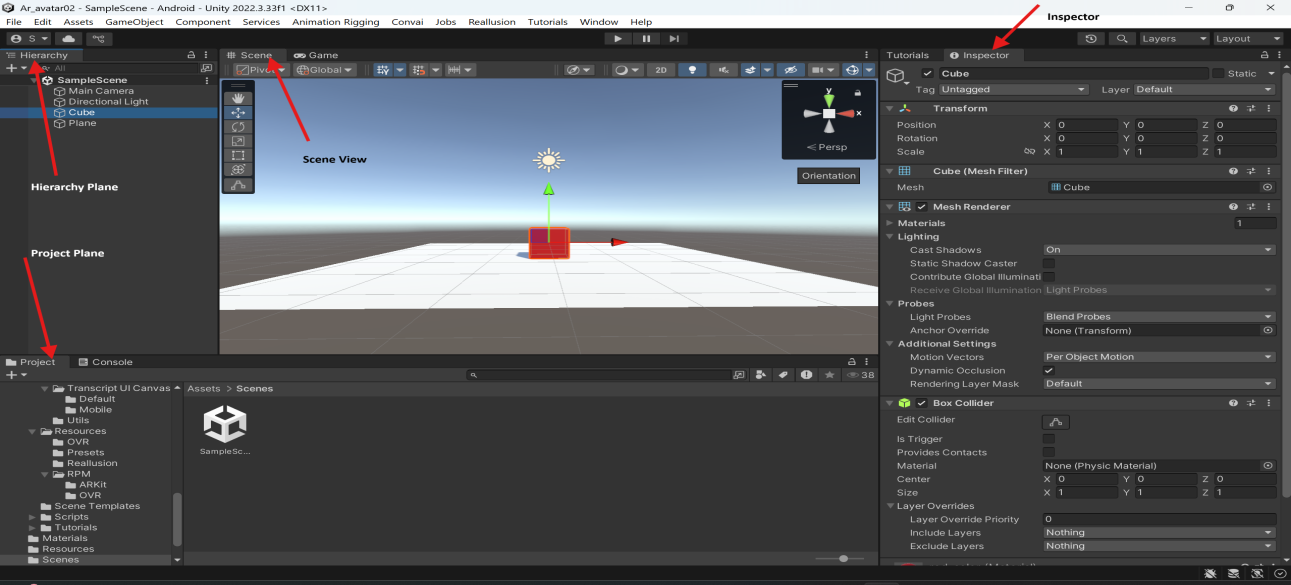
### The Hierarchy lists all the game objects present in the current scene in a tree structure. It organizes objects parented to one another, making it easier to select, group, and manage components of the scene. This panel reflects all assets visible and active in the Scene View.

### 2.2.3 Inspector Plane

### The Inspector displays detailed properties and components of the currently selected game object or asset. Developers use it to modify transform values (position, rotation, scale), add or remove components (scripts, colliders, renderers), and adjust properties such as material settings, behaviours and physics parameters.

### 2.2.4 Project Plane

### The Project Panel provides access to all assets, scripts, prefabs, textures, and resources included in the project. It acts as a file browser and manager, organizing files into folders for easy retrieval and drag-and-drop placement into the scene or hierarchy.

****

**Image 2.2.1: Unity** Editor Interface: Scene, Hierarchy, Project, and Inspector Panels

### 2.2.5 Game View

### The Game View shows the real-time output of the project, simulating what the player or user will see during gameplay or experience. It renders the scene as it will appear on the target device, enabling developers to test interactions, animations, and gameplay mechanics within the editor.

### 

**Image 2.2.2:** Game View Window in Unity Editor: Previewing Real-Time Scene Output

**2.3 Importing Assets in Unity Editor**

**Definition:**

Assets are the resources used to build a Unity project. These include **3D models, textures, audio, animations, prefabs, and scripts**. Importing assets means bringing external files or ready-made Unity content into your project so they can be used in scenes.

## 2.3.1 Importing 3D Models

* Unity supports common 3D model file formats like FBX, OBJ, and STL.
* To import, simply drag and drop the 3D model file into the **Project** panel or use **Assets > Import New Asset** from the menu.
* Upon import, Unity automatically processes the model and creates a usable asset.
* Models can then be dragged into the **Scene** or **Hierarchy** panel for placement.
* Unity provides options to configure the model's scale, mesh compression, and animation settings within the **Inspector**.

## 2.3.2 Importing Textures

* Textures such as PNG, JPEG, TIFF, or TGA files can be imported similarly by dragging them into the **Project** panel.
* Textures are used to add surface details to materials applied to 3D models or UI elements.
* Unity allows adjustments like wrap mode, filter mode, and compression quality in the **Inspector** after import.
* Multiple textures can be combined into materials inside Unity to define visual properties like colour, reflectivity, and transparency.

## 2.3.4 Importing Prefabs

* Prefabs are reusable game object templates that can include models, components, scripts, and more.
* Prefabs are created within Unity itself by dragging a configured game object from the **Hierarchy** into the **Project** panel.
* Prefabs can be imported if provided as Unity package files (.unity package), which are imported using **Assets > Import Package**.
* Using prefabs improves development efficiency by allowing duplication and easy updates to multiple instances across scenes.

## General Tips

* Always organize imported assets into descriptive folders within the **Project** panel for better management.
* Unity supports importing asset packages created in other engines or applications, expanding resources available.
* After import, assets can be optimized and customized through Unity’s powerful tools and scripting APIs.

## 2.4 Setting Up Build Settings in Unity (Android, iOS, XR)

Build settings in Unity configure the project for deployment on different platforms such as Android, iOS, and XR devices. Proper setup ensures that the application runs correctly and takes full advantage of platform-specific features.

## Accessing Build Settings

* Open Unity and go to **File > Build Settings**.
* This window shows a list of available platforms and settings for building the project.

## 2.4.1 Setting Up for Android

* Select **Android** from the platform list and click **Switch Platform** to target Android devices.
* Configure settings under **Player Settings**:
  + Set the package name (e.g., com.companyname.productname).
  + Define the minimum and target Android API levels.
  + Configure graphics APIs like OpenGL ES3 or Vulkan.
  + Adjust resolution and presentation settings for device screens.
* If using AR Core or Unity XR plugins, ensure they are installed via **Package Manager** and configured.
* Connect an Android device or emulator to test builds via USB debugging.

## 2.4.2 Setting Up for iOS

* Select **iOS** and switch platform.
* In **Player Settings**:
  + Set the bundle identifier (e.g., com.companyname.productname).
  + Specify the minimum iOS version supported.
  + Configure orientation, resolution, and other device-specific settings.
* Install necessary XR or ARKit packages if developing AR applications.
* After building, Unity generates an Xcode project that must be compiled and deployed using Xcode.

## 2.4.3 Setting Up for XR (Extended Reality)

* XR projects require enabling XR plug-in management via **Edit > Project Settings > XR Plug-in Management**.
* Install and activate relevant XR platforms (e.g., Oculus, Windows Mixed Reality, ARKit, ARCore).
* Configure XR-specific player settings like stereo rendering mode, tracking, and input handling.
* Ensure the build target platform matches the intended XR device (Android for Quest, iOS for ARKit, etc.).
* Test XR builds on target hardware with devices connected or through simulation.

### 

**Image 2.4.1:** Unity Build Settings: Preparing Project for Android Deployment

### ****2.5 Common Components in Unity****

Components in Unity are **building blocks** that define the behaviour and properties of GameObjects. Every GameObject must have at least a **Transform component**, and additional components can be added to give it functionality.

## 2.5.1 Transform

The **Transform** component is fundamental to every GameObject in Unity. It defines the GameObject’s position, rotation, and scale in the scene, effectively determining where and how it appears in the virtual world. The Transform component also manages the parent-child hierarchy (parenting), which controls how objects move and rotate relative to each other.

## 2.5.2 Colliders

**Colliders** define the shape of an object for physical collision detection. They do not render visually but provide boundaries that detect interactions with other colliders. Unity supports various collider types, such as BoxCollider, SphereCollider, CapsuleCollider (3D), and BoxCollider2D (2D), among others. Colliders are essential for physics-based interactions like detecting hits, triggers, or obstacles.

## 2.5.3 Rigidbodies

The **Rigidbody** component enables physics simulation on GameObjects. It allows objects to respond to forces such as gravity, collisions, and applied impulses dynamically rather than being static. Adding a Rigidbody pairs with Colliders to create realistic behaviors including falling, bouncing, and sliding.

## 2.5.4 Canvas

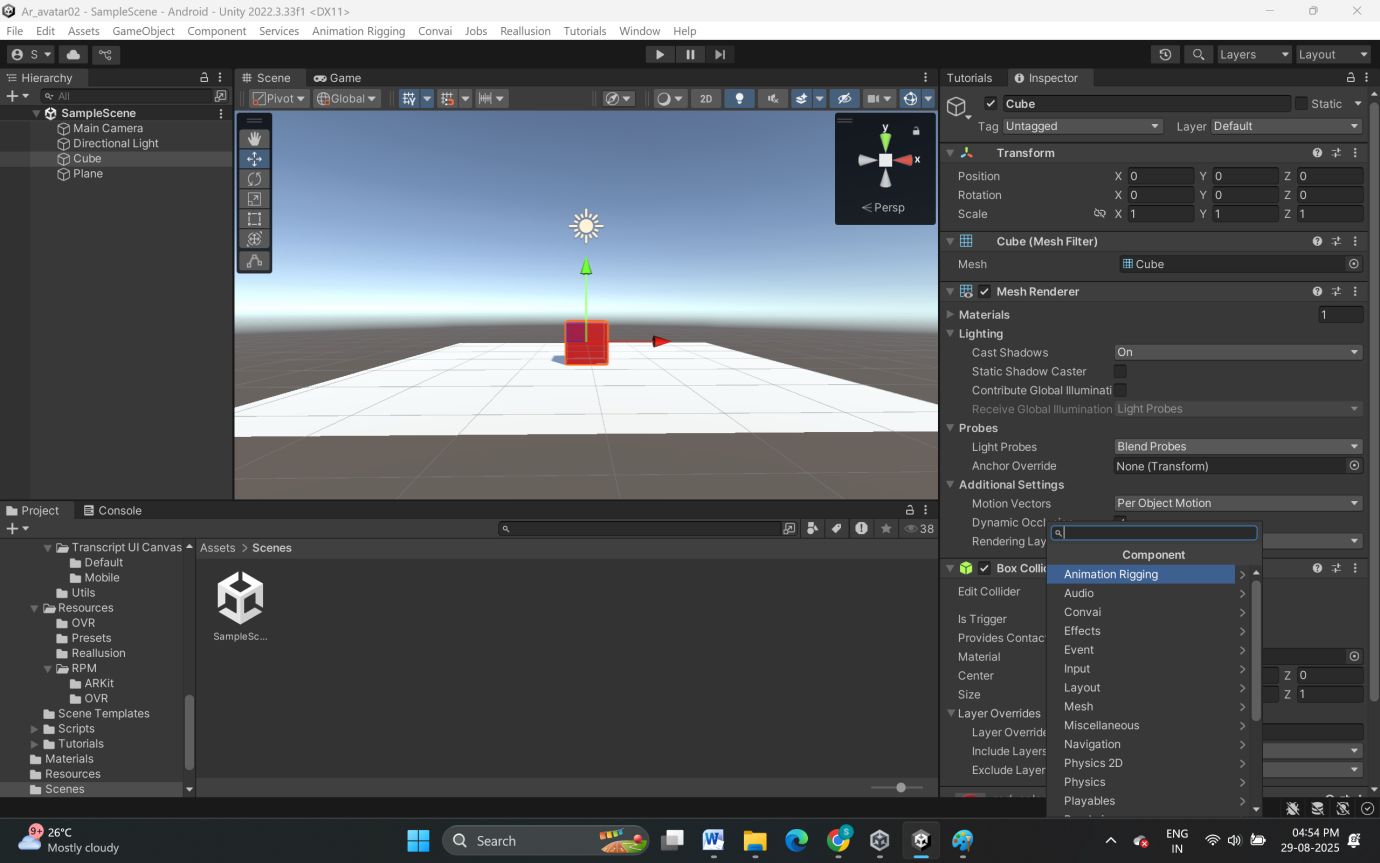
The **Canvas** component is the foundation for building user interfaces (UI) in Unity. It acts as a container for UI elements like buttons, text, and images, controlling how these elements are rendered and interact with the screen or world space. Canvas can be set to render in screen space (overlay or camera) or world space, enabling flexible UI designs.

## 2.5.5 Lighting

In Unity, lighting components control how light affects the scene and objects. Common lighting components include:

* **Directional Light:** Simulates sunlight with parallel light rays.
* **Point Light:** Emits light in all directions from a single point.
* **Spotlight:** Emits a cone-shaped light useful for focused illumination.

Lighting affects shadows, reflections, and overall ambiance, contributing significantly to the visual realism and mood of a scene.



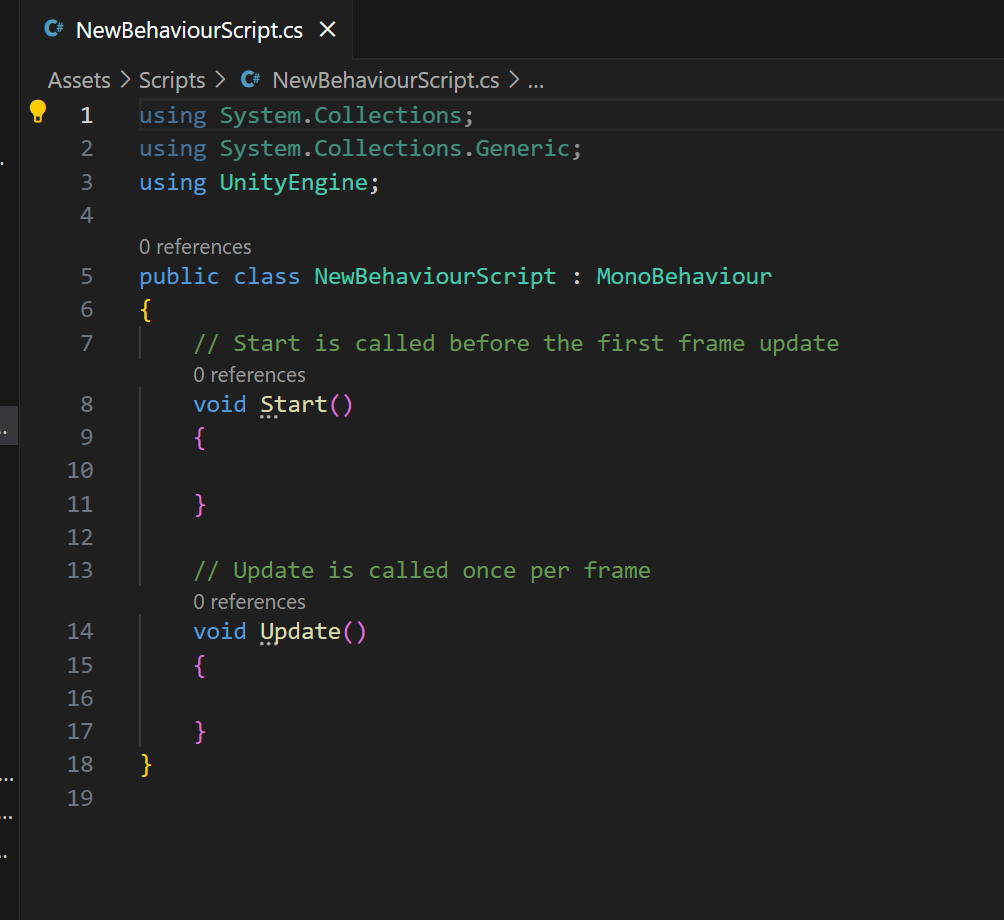
**Image 2.5.1:** Adding and Configuring Components in Unity’s Inspector Panel

**2.6 C# in Unity**

## 2.6.1 Definition and Role

In Unity, C# scripts are attached to GameObjects to create interactivity—moving objects, responding to input, detecting collisions, and managing game state. C# is chosen for its modern, object-oriented structure, wide .NET support, and ease of use for beginners and professionals.

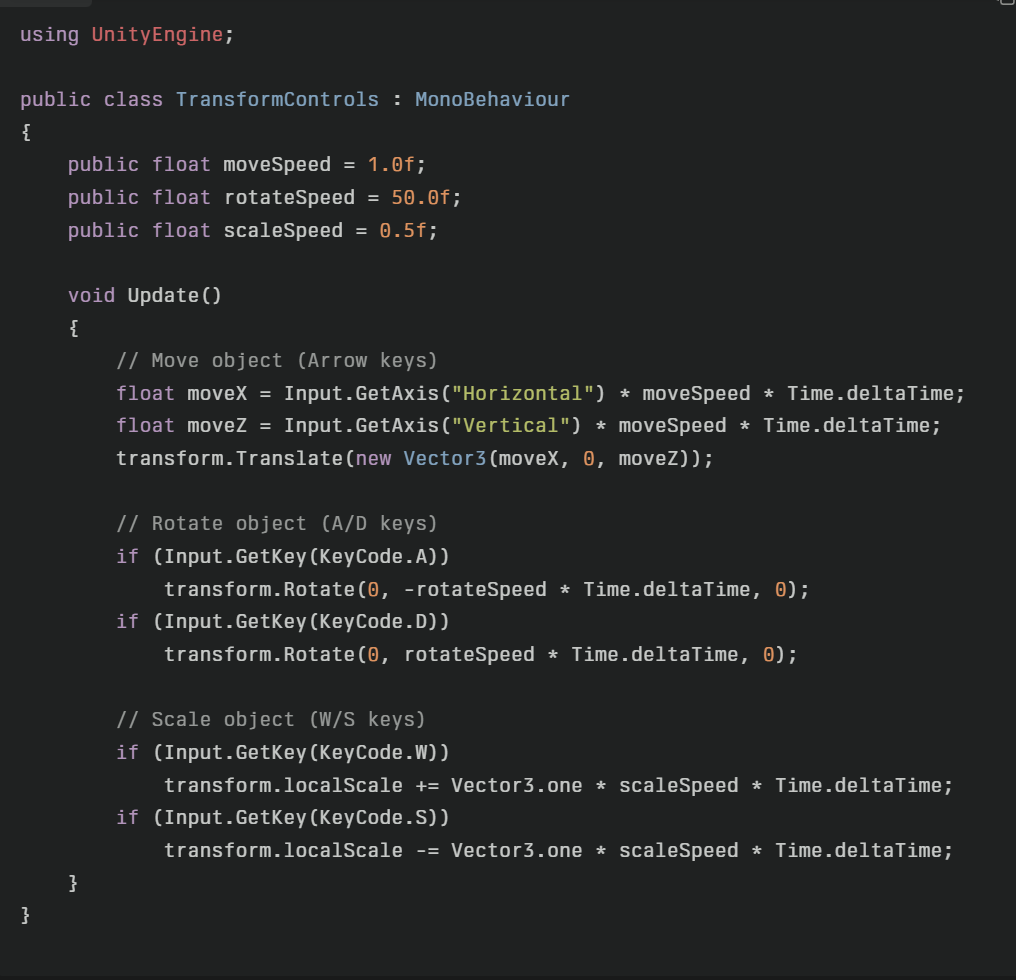
## 2.6.2 Default syntax of C# in Unity



**Image 2.6.1:** Syntax of C# script

* using System.Collections;
  + Imports the System.Collections namespace, enabling the use of collections such as lists and arrays in the script.
* using System.Collections.Generic;
  + Imports the System.Collections.Generic namespace to allow use of generic collections like List<T>, Dictionary<K,V>, etc.
* using UnityEngine;
  + Imports Unity’s core library, which contains basic game development classes such as MonoBehaviour, GameObject, and others.
* public class NewBehaviourScript : MonoBehaviour
  + Declares a new public class named NewBehaviourScript that inherits from MonoBehaviour. By inheriting MonoBehaviour, the class gains access to Unity’s event functions (e.g., Start, Update).
* { ... }
  + Encloses the body of the class where all fields and methods are defined.
* void Start()
  + Declares the Start method. This is a Unity event function called once before the first frame update (when the script instance is enabled or the scene starts).
* void Update()
  + Declares the Update method. This is a Unity event function called every frame while the script is active. It’s commonly used for checking input, updating movement, or running frame-by-frame logic.

## 2.6.3 c# script to Control a Transformation of GameObject



**Image 2.6.2:** Transformation Control C# script for GameObject

**2.7 Summary**

This chapter focuses on Unity as a powerful, cross-platform game engine widely used to develop AR and VR experiences. It describes the Unity Editor interface, including the Scene View for building environments, Hierarchy for organizing game objects, Inspector for modifying properties, Project panel for managing assets, and Game View for live previews. The chapter explains importing assets such as 3D models, textures, and prefabs to build content. It then details configuring build settings in Unity for different platforms including Android, iOS, and XR devices, emphasizing how to tailor projects for deployment. Lastly, the chapter covers common Unity components—Transform, Colliders, Rigidbodies, Canvas, and Lighting—explaining their roles in positioning, physics interaction, UI design, and scene illumination.

**Chapter 3:**

**Augmented Reality (AR)**

**3.1 What is AR?**

Augmented Reality (AR) is a technology that overlays real-time digital information, such as 3D graphics, images, or sounds, onto the user's view of the real world through devices like smartphones, tablets, or AR glasses. This blending of virtual content with the physical environment enhances the user's perception and interaction with their surroundings, creating an immersive and interactive experience.



**Image 3.1.1:** Example for AR 3D Model in real world

**3.2 Types of AR**

**3.2.1 Marker-Based AR**

Uses physical markers like QR codes or images that an AR device recognizes to trigger the display of virtual content. Example: IKEA’s app that lets users visualize furniture by scanning product catalog images.



**Image 3.2.1:** Example for Marker-Based AR