# The Unity Journey: Building Your First 3D Solar System

Welcome to Week 3 & 4 of our Game Programming lecture! This presentation is your comprehensive guide to setting up the Unity game engine and starting your "Solar System Project." We will cover everything from core concepts to essential setup steps, ensuring you have a solid foundation for building interactive 3D worlds.

> This lecture is designed for beginner programmers and hobbyists. We focus on clarity, step-by-step instructions, and practical application. By the end, you will be familiar with the Unity ecosystem and ready to create your first celestial simulation.

# 1. Understanding the Game Engine: Your Creative Toolbox

A game engine is a software framework designed for the development of video games. Think of it as a massive toolbox that provides all the essential components you need to build a game, without having to code every single piece from scratch.

## Core Functions of a Game Engine

### Rendering

Draws 2D and 3D graphics to the screen (lights, shadows, textures, models).

### Physics

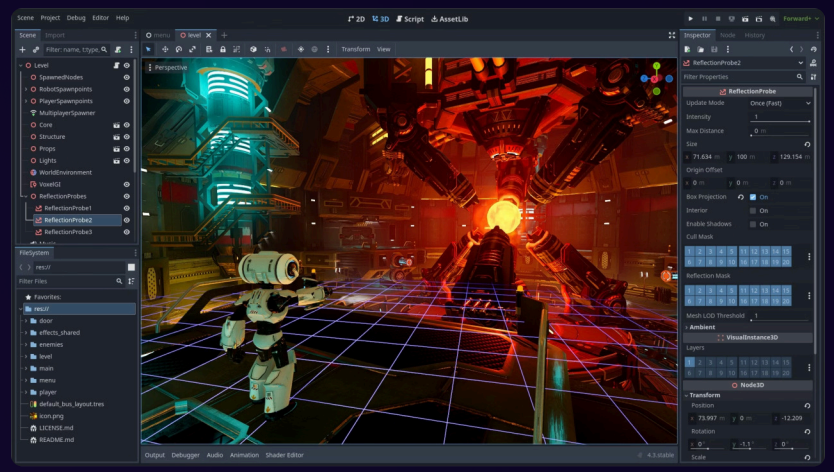
Handles realistic interaction, collision detection, and movement based on physical rules.

### Audio

Manages sound effects, music, and spatial audio (where sound comes from).

### Input

Processes player actions from controllers, keyboards, and mice.



Unity is one of the most popular game engines, offering a vast array of tools and supporting deployment to almost every platform imaginable, from PC to mobile and consoles. It uses C# as its primary scripting language.

# 2. Project Focus: The Solar System Simulation

Our first major project is creating a simplified 3D solar system. This project is chosen because it allows us to immediately apply fundamental game programming concepts like object manipulation, transformation, and rotation, while introducing the basic structure of a Unity scene.

The goal is to model the Sun and several planets, demonstrating orbital mechanics through simple C# scripts.

### Celestial Bodies

Creating 3D spheres to represent planets and the Sun.

### Orbital Movement

Using scripts to make planets rotate around the sun (the central point).

### Camera Control

Setting up a camera that can view the system effectively.

### Engine Familiarity

Gaining hands-on experience with the Unity Editor's transformation tools and component system.

This project serves as a clear, achievable goal for beginners, giving immediate visual feedback for the code you write. Success in this project means understanding the core concepts of **GameObject** and **Transform**.

# 3. Download and Install the Unity Platform

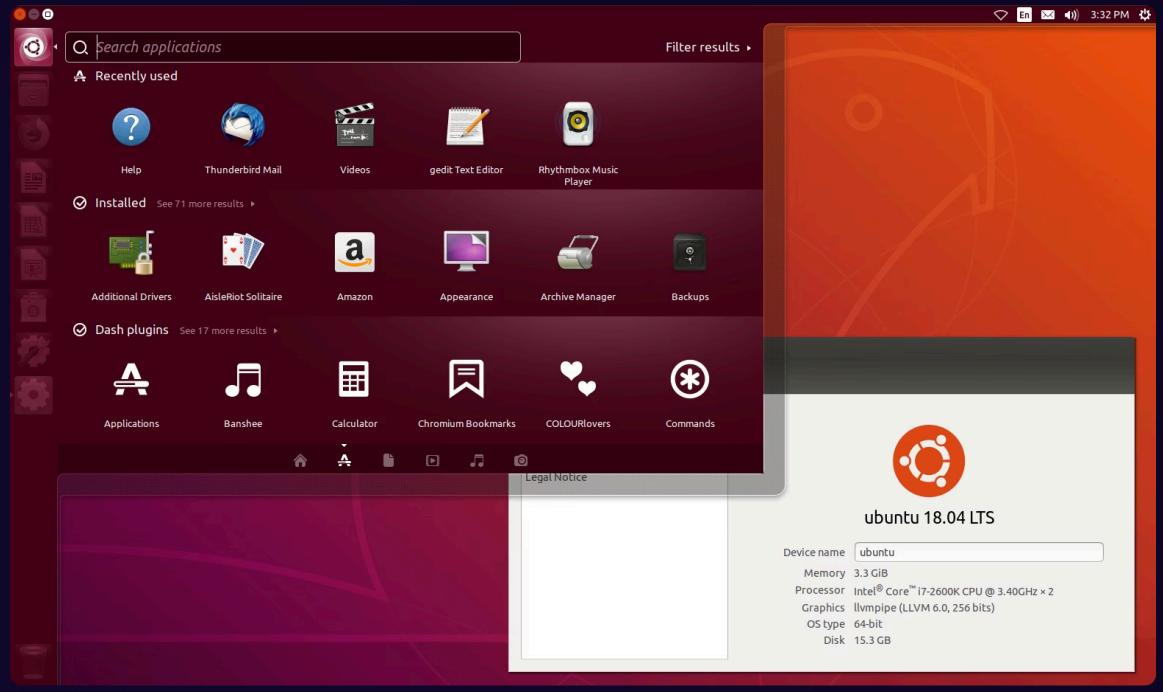
Before we can start building, we need the right tools. Unity is managed through the **Unity Hub**, which allows you to install multiple versions of the editor and manage all your projects easily.

## Step 1: Get the Unity Hub

- Go to the official Unity website and navigate to the Download page.
- Download and install the Unity Hub application.
- Sign in or create a free Unity ID. This is necessary for licensing and access to assets.

## Step 2: Install the Editor

- Inside the Hub, go to the **Installs** tab.
- Click **Install Editor**.
- We recommend installing the latest Long-Term Support (LTS) version, as it is the most stable and reliable for large projects.
- Crucially, ensure you select the **Visual Studio Community** module during installation, as this is the IDE (Integrated Development Environment) we will use for C# scripting.




Always use the LTS (Long-Term Support) version for stability in educational or commercial projects. It receives bug fixes and support for an extended period.

# 4. Unity Setup: Licensing and Modules

Proper setup is vital to avoid frustrating roadblocks later on. The Unity Hub manages your license, which is free for personal use, and ensures you have all the necessary components (modules) for your target platforms.

1

### Activate Personal License

After logging into Unity Hub, go to Preferences > License Management. Click **Activate New License** and choose the free **Personal** option. This grants you full access to the engine features without cost, provided your company revenue is below a certain threshold.

2

### Check Build Modules

If you forgot to install the correct modules (like Android support or WebGL support), you can add them later. In the Installs tab, click the gear icon next to your Unity version and select **Add Modules**. For our Solar System project, the default Windows/Mac/Linux build support is enough, but ensure the Documentation and Visual Studio modules are present.

3

### Configure Visual Studio

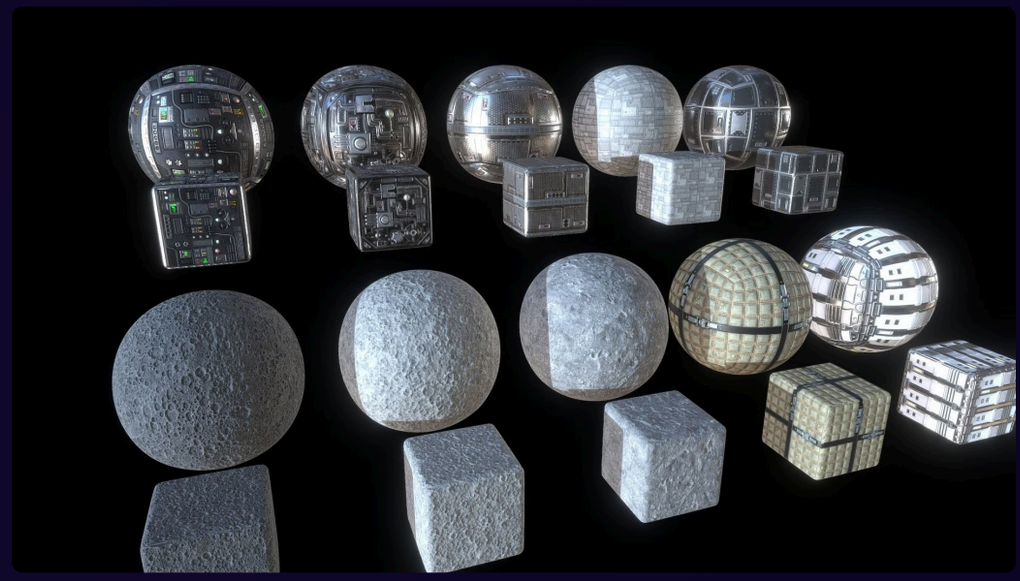
In the Hub's Preferences, ensure that **Visual Studio Community** is set as the default external script editor. This guarantees that when you double-click a C# file in Unity, it opens correctly in the IDE with all the required debugging tools.

# 5. Creating Your Unity Project

The creation process determines the starting environment of your project. For our 3D solar system, we must choose the correct template.

## Choosing the Right Template

- **Open Unity Hub** and navigate to the **Projects** tab.
- Click the blue **New Project** button.
- Select the **3D (Core)** template. This provides a clean, standard 3D environment optimized for performance and basic scene creation, which is perfect for our simulation.
- **Avoid** templates like "Universal Render Pipeline (URP)" or "High Definition Render Pipeline (HDRP)" for now, as they introduce extra complexity we don't need yet.



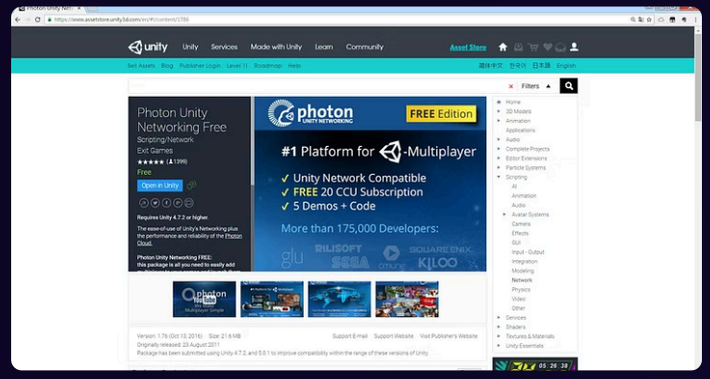
## Project Details

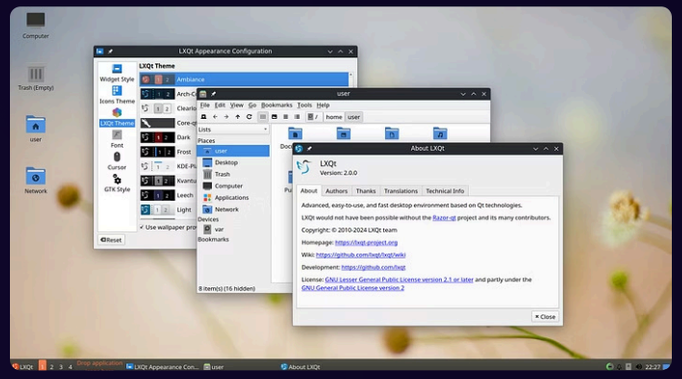
Give your project a clear, descriptive name (e.g., `SolarSystem_FirstProject`). Choose a suitable location on your hard drive where you can easily find it. Once named and located, click **Create Project**. Unity will take a few minutes to set up the initial files and load the editor.

> The project name is used for file organization and should not contain spaces or special characters if possible, although Unity can handle them.

# 6. Downloading Project Assets

While we could build all the planetary models and textures ourselves, utilizing pre-made assets is standard practice in game development. This allows us to focus on the programming mechanics rather than pure artistry.




### Access the Asset Store

You can find the Unity Asset Store either through a web browser or directly inside the Unity Editor (Window > Asset Store, though this redirects to the web in newer versions).

### Search for Free Assets

Search for a package like "Solar System Kit" or "Space Textures." Always filter results for "Free Assets" to stay within budget. We need basic 3D sphere models and high-resolution planetary textures.

### Download and Add to My Assets

Once you find a suitable asset pack, click **Download** and then **Add to My Assets**. This links the package to your Unity account, making it available for import into any project.

Having pre-existing assets saves significant time and gives your simulation a polished, professional look right from the start.

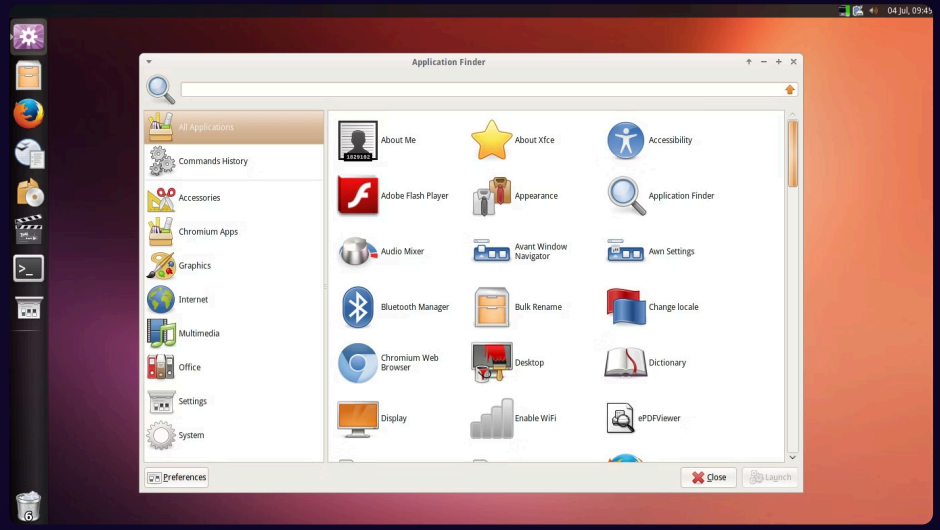
# 7. Importing Assets into the Project

After acquiring assets, they must be brought into your specific project folder. This process unpacks the files (models, textures, sounds, scripts) into the Unity file structure.

## The Import Process

1. In the Unity Editor, open the **Window** menu, then select **Package Manager**.
2. In the Package Manager window, change the drop-down menu from "Unity Registry" to **My Assets**. This displays all the assets linked to your account.
3. Locate the Solar System asset package you downloaded.
4. Click the **Download** button (if not already downloaded locally), and then click the **Import** button.
5. A dialog box will appear showing all the files in the package. Ensure all components (models, materials, scripts) are checked, and click **Import**.

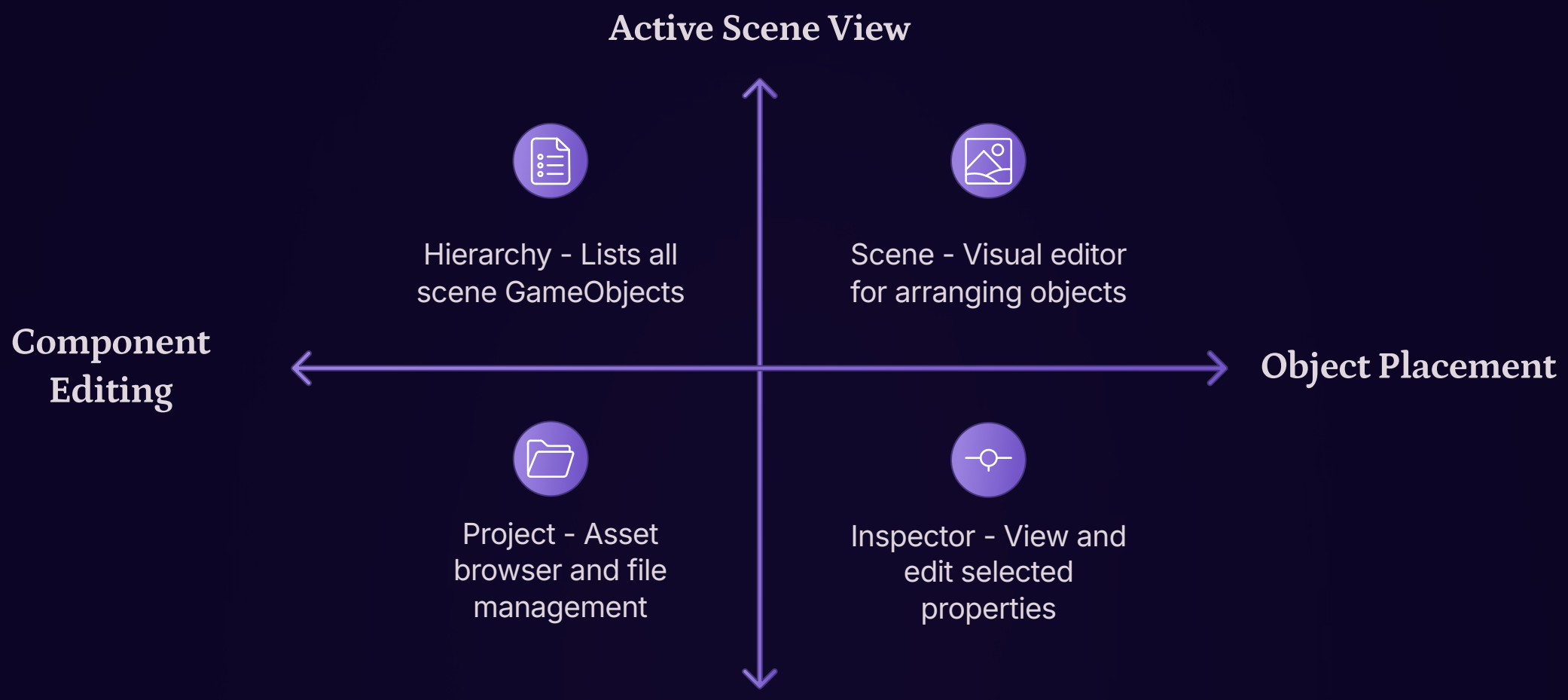
Once imported, the new folders and files will appear in the **Project Window** of the Unity Editor. It's good practice to keep the resulting folders organized.




> The Project Window in Unity is your file explorer. All models, textures, and scripts must reside here before they can be used in your game scenes.

# 8. Introduction to the Unity Editor

The Unity Editor is where the magic happens. It's a complex application, but we'll focus on the four key windows you'll use constantly. Mastery of these windows is crucial for efficient game development.




### Hierarchy Window

Lists every GameObject currently in the open scene. This is where you organize objects and see their parent-child relationships (e.g., Moon parented to Earth).

### Scene Window

Your 3D workspace. This is where you visually build, arrange, and navigate your game world (the Solar System). You manipulate objects using the Move, Rotate, and Scale tools.

### Project Window

Shows all the asset files (scripts, textures, models) in your entire project, similar to your computer's file explorer. **Do not confuse this with the Hierarchy!**

### Inspector Window

Displays the components and properties of the currently selected GameObject. This is where you modify transformation values, adjust material colors, or assign scripts.

# Next Steps: Writing Your First C# Script for Rotation

Now that the editor is set up and assets are imported, we are ready to bring our static solar system to life using code. Game programming is the art of giving objects instructions.

## Example: The Rotation Script

We will create a C# script called `Rotator` and attach it to our planets. This simple code makes the object spin around its own center, simulating a planet's rotation.

```
using UnityEngine;

public class Rotator : MonoBehaviour
{
    // Public variable for easy adjustment in the Inspector
    public float rotationSpeed = 50f;

    // Update is called once per frame
    void Update()
    {
        // Rotate the object around its Y axis
        transform.Rotate(Vector3.up * rotationSpeed * Time.deltaTime);
    }
}
```

In Unity, all movement and orientation is controlled by the **Transform component**, which every GameObject automatically possesses.

The key line is `transform.Rotate()`, which manipulates the object's **Transform component**. `Vector3.up` means we are rotating around the Y-axis. `Time.deltaTime` is essential for frame-rate independence, ensuring the rotation speed is the same regardless of how fast the player's computer is.

This is the foundation of all movement in Unity. In the next session, we will refine this script to handle orbital movement, requiring the planets to rotate around a central point (the Sun) instead of just their own axis.