Week 3: Foundations of Game Programming – Building a Solar System Simulation

Welcome to Week 3! This lecture introduces you to the core concepts of using a modern game engine, specifically Unity, by diving into our first major project: creating a realistic Solar System simulation.

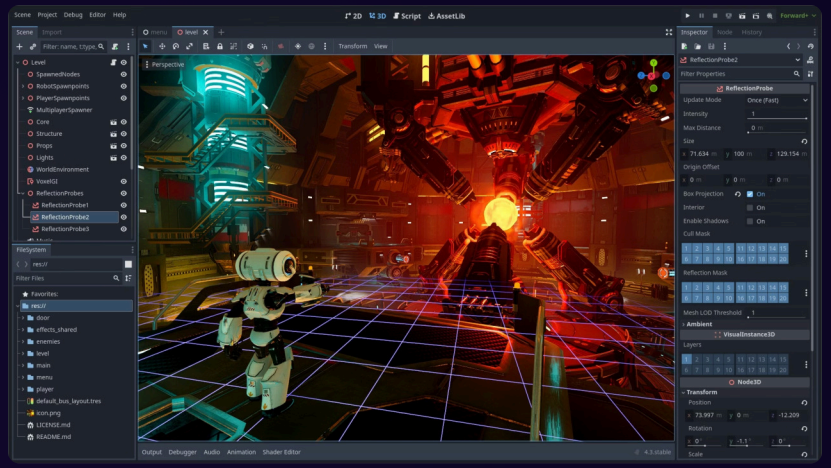
Over the next several cards, we will cover everything from setting up your development environment to building and testing the final simulation. We aim for a clear, instructional approach, ensuring all core concepts are defined and accompanied by relevant visuals.

1. Game Engine Overview: What is Unity?

Definition and Role

A game engine is a comprehensive software framework designed to facilitate the creation and development of video games. Think of it as a massive toolbox that provides all the necessary components for game creation—graphics rendering, physics simulation, sound, scripting, animation, AI, and more.

Unity is one of the most popular cross-platform game engines, used by millions of developers for everything from massive AAA titles to small indie games and educational simulations. Its primary advantage is its user-friendly interface and robust C# scripting environment, making it ideal for beginners.



Key Components

- The Editor: The visual interface where you arrange objects, design levels, and manage project assets.
- The Renderer: Handles drawing all the 3D and 2D graphics to the screen.
- The Physics Engine: Calculates interactions, collisions, and movement (crucial for our solar system's orbits).
- The Scripting API: Allows us to write code (C#) to define game logic and behavior.

2. Solar System Project Overview: Our First Simulation

Goal: Realistic Orbital Mechanics

The primary objective is to build a functional, visually appealing simulation of our solar system. We will focus on implementing basic gravitational and orbital physics to make the planets move realistically around the Sun.

Learning Focus: 3D Scene Setup

This project teaches fundamental skills like importing assets, setting up 3D space coordinates (transforms), managing materials, applying lighting, and integrating C# scripts to control movement and camera views.

Core Skill: Scripting Behaviors

We will write C# scripts to define the rotation of the planets and the logic for their elliptical orbits, ensuring the simulation runs smoothly and follows real-world principles.

This simulation provides a perfect, controlled environment to understand how objects interact in 3D space using a game engine's physics and scripting tools. It's a stepping stone to more complex game environments.

3. Unity Setup: Download and Installation

3.1. Unity Hub and Editor

Before starting, you need the Unity Hub—a central management tool for all your Unity projects and editor installations. It allows you to manage different versions of the Unity Editor required for various projects.

1. Go to the official Unity website and download the Unity Hub installer.
2. Install the Unity Hub, then open it.
3. In the Hub, navigate to the ‘Installs’ tab. Click ‘Install Editor’.
4. Choose a recommended, stable version (e.g., the latest Long-Term Support, or LTS, version). This is crucial for stability.
5. Ensure you select the ‘Windows Build Support’ or ‘Mac Build Support’ modules, along with the ‘Documentation’ for easy reference.

Unity setup requires a stable internet connection and significant disk space, as the editor package is large. Take your time to ensure all necessary components are included during the installation process.

4. Creating the Project and Downloading Assets

4.1. Creating the Unity Project

Every project begins with a template. For a 3D simulation like the solar system, we will use the core 3D template, which optimizes the environment for three-dimensional graphics.

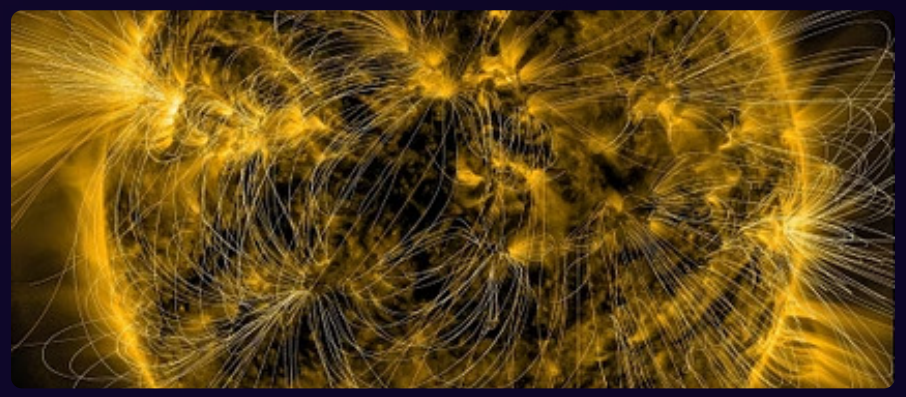
Open Unity Hub: Go to the 'Projects' tab and click 'New Project'.

Select Template: Choose the **3D (Core)** template. This provides a standard rendering pipeline suitable for our needs.

Name & Location: Give your project a clear name (e.g., "SolarSystemSim") and select a local directory to save it. Click 'Create Project'.

4.2. Downloading Project Assets

We won't create the planet textures from scratch. We will download pre-made assets—specifically, high-resolution textures for the Sun and planets—to ensure our simulation looks detailed and realistic.

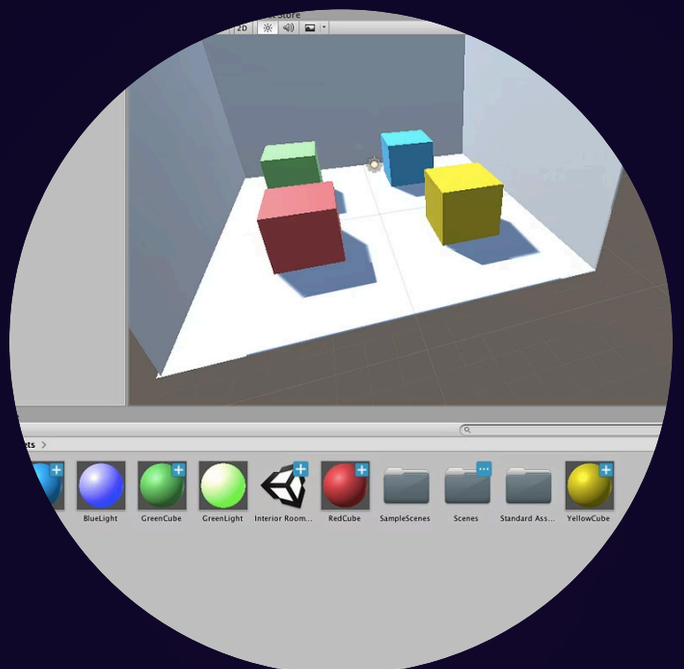


These assets, typically sourced from sites like NASA, will be imported directly into our project's 'Assets' folder to be applied as materials to our spheres.

5. Core Scene Components: Materials, Lights, and Cameras

Creating the Visual Foundation

Once the project is open, we define the visual look and feel of our space environment using materials, ensuring proper illumination with lighting, and setting up the viewer's perspective with the camera.



Materials and Textures

A **Material** defines how a surface looks. It includes the color, shininess, and, critically, the **Texture** (the image of the planet surface). We will create distinct materials for the Sun, Earth, Mars, etc., using the downloaded texture maps.

Lighting Setup

In a solar system, the Sun is the sole light source. We will use a powerful **Point Light** component centered on the Sun object to simulate its luminosity. We will also adjust the scene's ambient light to be pitch black to simulate the emptiness of space.

Camera Configuration

The **Camera** is what the user sees. For a solar system, we need a dynamic camera that can follow a specific planet or allow free movement. We will initially position the Main Camera to view the entire system and later attach a script to enable orbiting and zooming.

7. Setting Up the Project: Hierarchical Structure

Organizing the Scene for Clarity

A clean project hierarchy is essential for managing complexity. In our simulation, all celestial bodies must orbit the Sun. We achieve this by structuring the objects correctly in the Hierarchy panel.

1. The Sun (Root Object)

The Sun is the center of the system. It is placed at the global origin (0, 0, 0) and is the parent object for the entire system's logic.

2. Orbital Path Controllers

To simplify orbital physics, each planet is a child of an empty GameObject (or 'Orbit Controller'). These empty objects are also centered at (0, 0, 0) and the planet object is offset from its controller. Scripting the rotation of the controller makes the planet orbit the Sun.

3. Individual Planets

The individual planet models are children of their respective Orbit Controllers. This ensures that when the controller rotates, the planet follows the orbit, and the planet's own rotation script handles its spin.

4. Moons and Satellites

Moons (like Earth's Moon) become children of their respective planet objects. This establishes a secondary orbital system where the Moon orbits the Earth, and the Earth (with the Moon) orbits the Sun.

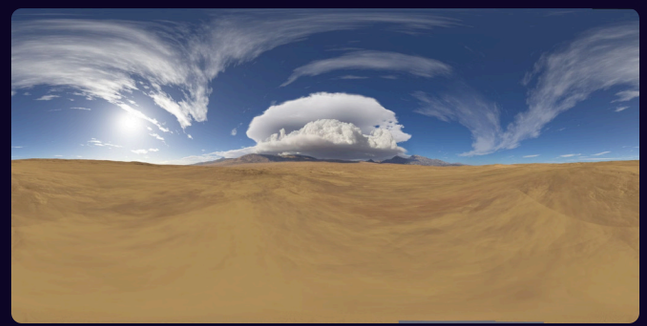
8. Level Design and Final Polish: Audio and Testing

From Functional Simulation to Immersive Experience

Level Design in a simulation involves structuring the environment. Here, it means precisely scaling the planets and setting their initial distances, although we must often compress the astronomical scale significantly to fit within the scene view.

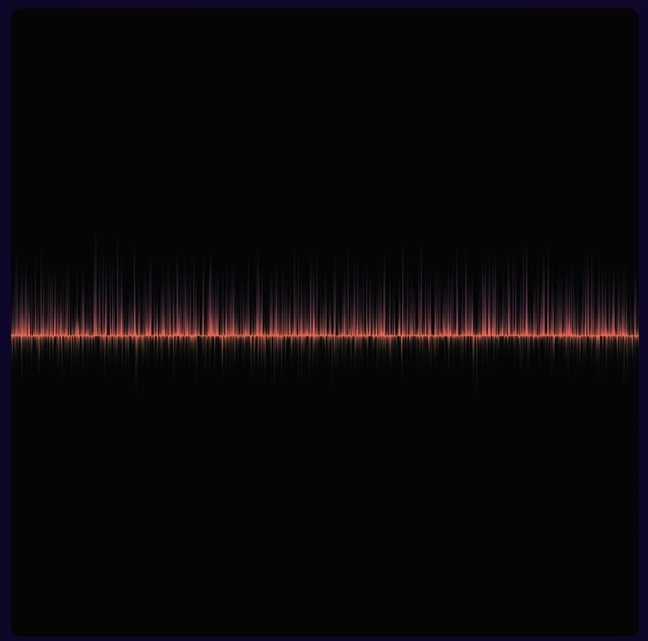
Aesthetics and Scale

We adjust the size ratios of the planets (e.g., Jupiter is much larger than Earth) and their distance ratios to visually convey scale. We also apply a 'Space Skybox'—a 360-degree cubemap texture of the distant stars—to complete the cosmic environment.



Audio Implementation

Audio is crucial for immersion. We add an **Audio Listener** to the camera and an **Audio Source** component to the scene, playing a continuous loop of ambient space music or quiet, deep background sound effects. This enhances the user experience.



Build and Test

Once the simulation runs correctly in the Unity Editor, we need to create a standalone application (**Build**). In the 'Build Settings,' we select the target platform (PC/Mac) and click 'Build.' We then test the executable file to ensure the performance, physics, and controls work outside the editor.

9. Finishing Up: Consistent Learning and Next Steps

Reflecting on Game System Instructions

This project successfully incorporated all the necessary elements of a foundational game programming lecture, focusing on clarity, integrity, and hands-on application.

Proper Definitions Included

Each topic (Game Engine, Material, Prefab) was clearly defined, ensuring a strong conceptual groundwork for beginner students.

Detailed Learning Material

We provided extensive text and detailed steps covering the setup and implementation of core features, making the content self-contained and useful for non-presenting study.

Relevant and Real Imagery

The presentation was rich with specific, real-world images related to Unity's interface, 3D assets, and development processes to enhance engagement.

Code Examples Provided

Simple C# code snippets were included to illustrate how the Rotation and Orbit behaviors are implemented, demonstrating the connection between scripting and visual output.

Consistency and Integrity Maintained

The entire presentation maintained a clear, instructional flow, building the Solar System simulation step-by-step with consistent terminology and a focus on essential, non-vague information.