From Eye to AI: Digital Phantoms for Medical Imaging

Lecture 4 – Coding in Unity3D

Joe Xing, Ph.D
CTO and Co-founder of C. Light Tech
Visiting professor at Tsinghua University

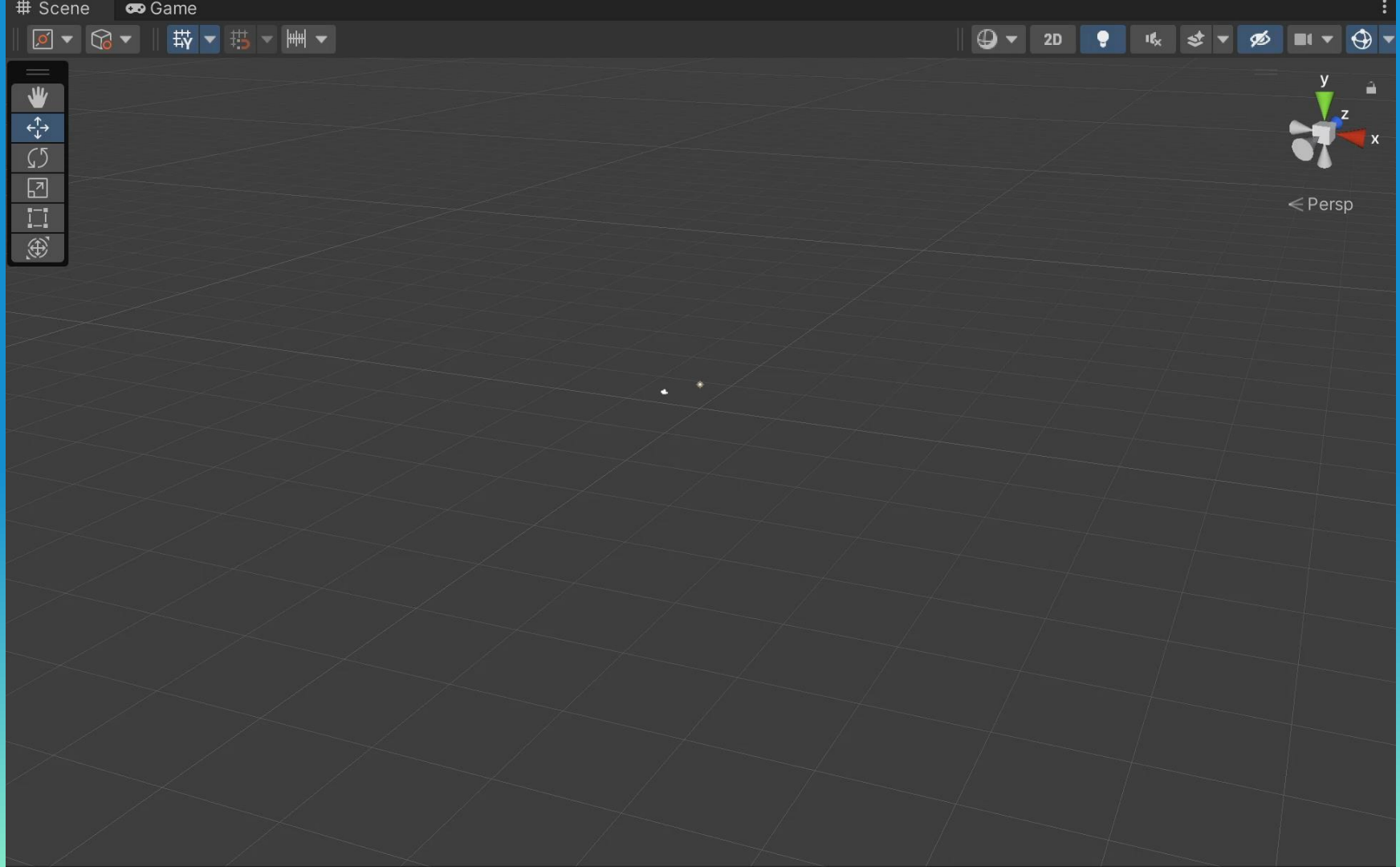
Fall Season 2025

C. LIGHT TECHNOLOGIES

TSINGHUA UNIVERSITY 1911

What is C#

- **C# (C-Sharp) is a modern, object-oriented programming language by Microsoft.**
- Designed for clarity, productivity, and integration with tools like Unity, .NET, and Visual Studio.
- Commonly used for game development, desktop apps, web APIs, and AR/VR.


Scene
Game
2D
y
z
x
Persp

Object Pooling

Spawn a game object versus Create a game object

Which one is more computationally heavy?

- Object Pooling improves performance by reusing inactive GameObjects instead of repeatedly creating and destroying them.
- Why using it?

- Reduces **Instantiate/Destroy** overhead.
- Minimizes **garbage collection** spikes.
- Ideal for bullets, enemies, particles, or effects.

Factory Pattern – Spawning GameObjects or Agents

- **The Factory Pattern centralizes object creation, letting you spawn GameObjects dynamically without hardcoding their types or prefabs**

```
using UnityEngine;


public enum AgentType { Enemy, Ally }


[System.Serializable]
public struct AgentPrefab
{
public AgentType type;
public GameObject prefab;
}


public class AgentFactory : MonoBehaviour
{
[SerializeField] private AgentPrefab[] prefabs;


public GameObject Spawn(AgentType type, Vector3 position, Quaternion rotation)
{
var prefab = System.Array.Find(prefabs, p => p.type == type).prefab;
return Instantiate(prefab, position, rotation);
}
}
```

```
public class Spawner : MonoBehaviour
{
[SerializeField] private AgentFactory factory;


void Start()
{
// Spawn an enemy at a specific position
factory.Spawn(AgentType.Enemy, new Vector3(0, 0, 0), Quaternion.identity);
}
}
```

Centralizes and simplifies object creation.
Easy to add new types without changing existing code.
Works seamlessly with object pooling or dependency injection.

Poolable Object

An **interface** in C# is like a **contract** that defines a set of methods or properties that a class **must implement**, without defining how they work.

```
using UnityEngine;

namespace DigitalPhantom.ObjectPooling
{
    /// <summary>
    /// Base class for objects that can be pooled
    /// </summary>
    public abstract class PoolableObject : MonoBehaviour, IPoolableObject
    {
        [Header("Pool Settings")]
        [SerializeField] protected bool isActive = false;

        public bool IsActive
        {
            get => isActive;
            set => isActive = value;
        }

        public GameObject GameObject => gameObject;
```

In Unity, you can **inherit from `MonoBehaviour`** to use Unity's engine features, and **implement multiple interfaces** to modularize your behavior

```
using UnityEngine;

namespace DigitalPhantom.ObjectPooling
{
    /// <summary>
    /// Interface for objects that can be pooled
    /// </summary>
    public interface IPoolableObject
    {
        /// <summary>
        /// Called when the object is taken from the pool
        /// </summary>
        void OnPoolGet();

        /// <summary>
        /// Called when the object is returned to the pool
        /// </summary>
        void OnPoolReturn();

        /// <summary>
        /// Called when the object is created for the first time
        /// </summary>
        void OnPoolCreate();

        /// <summary>
        /// Whether this object is currently active in the scene
        /// </summary>
        bool IsActive { get; set; }
```

Pool can spawn any game objects

```
using UnityEngine;

namespace DigitalPhantom.ObjectPooling
{
    /// <summary>
    /// Ball object that can be pooled
    /// </summary>
    public class Ball : PoolableObject
    {
        [Header("Ball Settings")]
        [SerializeField] private float bounceForce = 5f;
        [SerializeField] private Color ballColor = Color.red;
        [SerializeField] private float size = 1f;

        private Rigidbody rb;
        private Renderer ballRenderer;
        private Vector3 originalScale;
```

Object Pool

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

namespace DigitalPhantom.ObjectPooling
{
    /// <summary>
    /// Generic object pool for managing reusable objects
    /// </summary>
    /// <typeparam name="T">Type of object to pool (must implement IPoolableObject)</typeparam>
    public class ObjectPool<T> where T : class, IPoolableObject
    {
        private readonly Queue<T> pool = new Queue<T>();
        private readonly List<T> activeObjects = new List<T>();
        private readonly System.Func<T> createFunction;
        private readonly int maxPoolSize;
        private readonly int initialPoolSize;
        private int totalCreated = 0;

        public int PoolSize => pool.Count;
        public int ActiveCount => activeObjects.Count;
        public int TotalCount => PoolSize + ActiveCount;
        public int TotalCreated => totalCreated;

        /// <summary>
```

Generic Design Patterns

Flexibility – The pool works for any object that implements a required interface.
Type Safety – Compile-time enforcement of required methods/properties.
Reusability – One generic pool can handle bullets, enemies, projectiles, etc.

Waterfall Manager

```
private void InitializePools()
{
    // Create a parent object for pooled objects
    poolParent = new GameObject("Pooled Objects").transform;
    poolParent.SetParent(transform);

    if (useBalls)
    {
        // Create ball pool
        ballPool = new ObjectPool<Ball>(
            () => ObjectFactory.CreateBall(poolParent, homeLocation),
            initialPoolSize,
            maxPoolSize
        );
    }
    else
    {
        // Create cube pool
        cubePool = new ObjectPool<PoolableCube>(
            () => ObjectFactory.CreateCube(poolParent, homeLocation),
            initialPoolSize,
            maxPoolSize
        );
    }
}
```

The pool doesn't care *how* the object is created — only that it *can be* created.

The **pool** handles *lifecycle* (Get, Return, Resize). The **factory** handles *creation details* (prefab, parent, transform, initialization).

THANKS

DO YOU HAVE ANY QUESTIONS?

contact@joexing.me