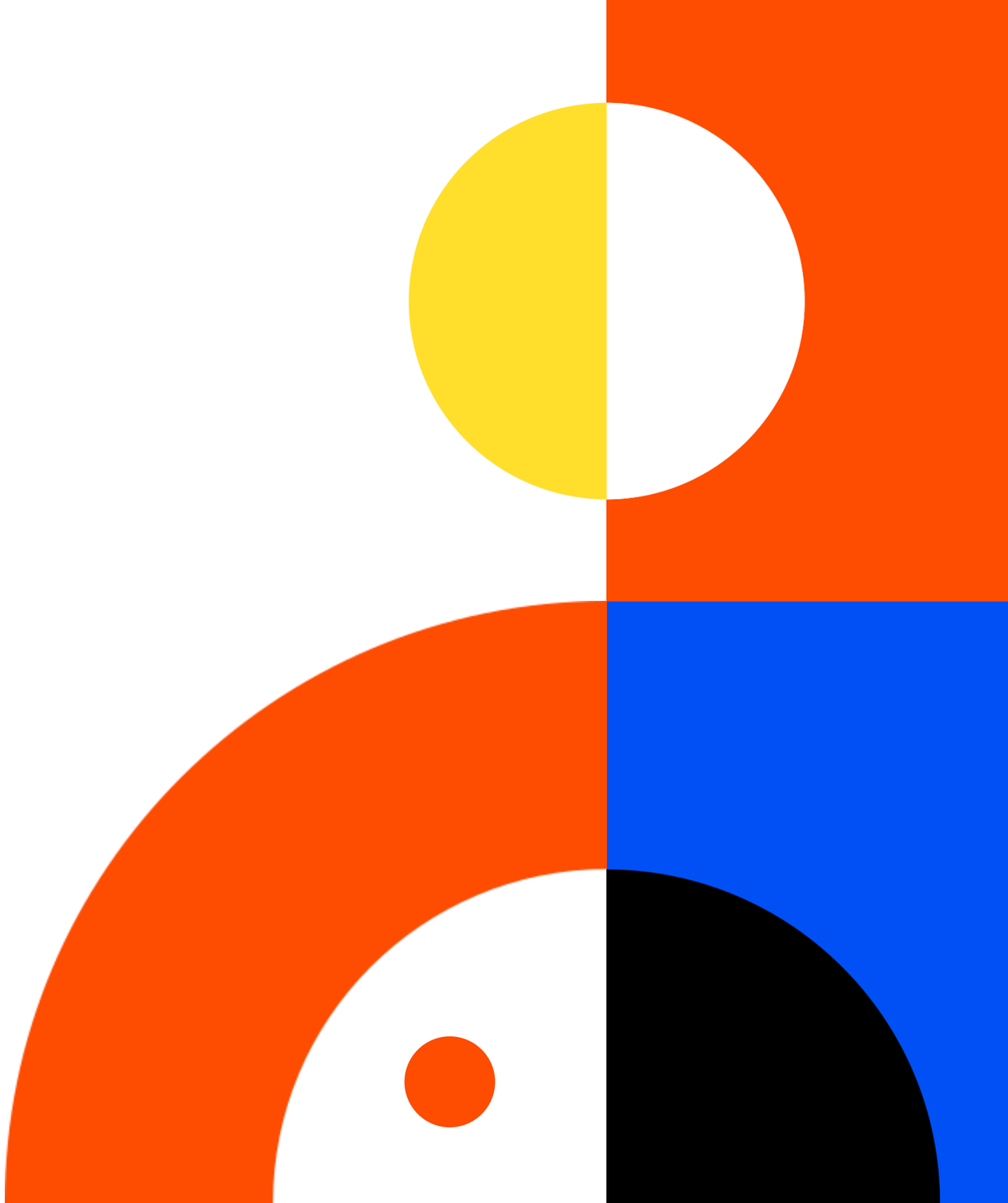
Лекція №6

Динамічне створення об’єктів

2025



План

1. Динамічне створення та видалення об'єктів
2. Префаби
3. Quaternion
4. Paralax-effect

Динамічне створення та видалення об'єктів

У Unity більшість об’єктів створюється під час розробки — ти просто перетягуєш їх у сцену.
Але часто треба створювати нові екземпляри **під час гри**, коли:
з’являється ворог;
гравець стріляє кулею;
потрібно згенерувати частинки, бонуси або блоки рівня.
Тоді використовується **динамічне створення** — тобто код створює новий об’єкт під час виконання.

Динамічне створення

Unity дозволяє “клонувати” об’єкти або **префаби** через метод Instantiate()

```
public class Spawner : MonoBehaviour
{
    public GameObject enemyPrefab;

    void Start()
    {
        // Create one enemy at the origin
        Instantiate(enemyPrefab, Vector3.zero, Quaternion.identity);
    }

    void Update()
    {
        // Press Space to spawn a random enemy
        if (Input.GetKeyDown(KeyCode.Space))
        {
            Vector3 pos = new Vector3(Random.Range(-5f, 5f), 0, 0);
            Instantiate(enemyPrefab, pos, Quaternion.identity);
        }
    }
}
```

Навіщо це потрібно

- Створення **ворогів/снарядів/об’єктів UI** під час гри.
- Генерація **рівнів або середовищ** (як у Minecraft або Terraria).
- Динамічне завантаження контенту без попереднього розміщення на сцені.

Destroy()

Destroy() — це **метод Unity**, який **видаляє об’єкти або компоненти** під час виконання гри.
Він зупиняє оновлення (Update, FixedUpdate тощо) і фізично прибирає об’єкт із сцени.

- Destroy(Object obj);
- Destroy(Object obj, float t);

```
void Update()
{
// Delete this GameObject instantly
    if (Input.GetKeyDown(KeyCode.Space))
        Destroy(gameObject);
}
```

```
void Start()
{
// Destroy after 3 seconds
    Destroy(gameObject, 3f);
}
```

Подія OnDestroy

OnDestroy() — це **вбудований метод Unity**, який викликається **перед тим**, як об’єкт знищується.
Це свого роду “деструктор” у світі Unity.

```
public class Enemy : MonoBehaviour
{
    void OnDestroy()
    {
        // Called automatically before the GameObject
is destroyed
        Debug.Log("Enemy destroyed!");

        // Example: unsubscribe from event
        GameManager.OnEnemyDied -= HandleDeath;
    }
}
```

Навіщо це потрібно

- Щоб **відписуватись від подій**, якщо об’єкт більше не існує (щоб уникнути помилок типу *MissingReferenceException*).
- Щоб **звільнити ресурси** (таймери, завдання, ефекти).
- Щоб **створити ефект вибуху або частинок** після знищення.

Додавання і видалення компонентів

```
void Start()
{
 // Add component Rigidbody2D
    var rigitbody = gameObject.AddComponent<Rigidbody2D>();
 // Remove Rigidbody2D only
    Destroy(rigitbody);
}
```

Префаби

Prefab — це “шаблон” об’єкта.
У ньому зберігається вся структура: компоненти, налаштування, матеріали, дочірні об’єкти тощо.
Коли ти створюєш копії через Instantiate(prefab), Unity автоматично відтворює весь об’єкт з його властивостями.

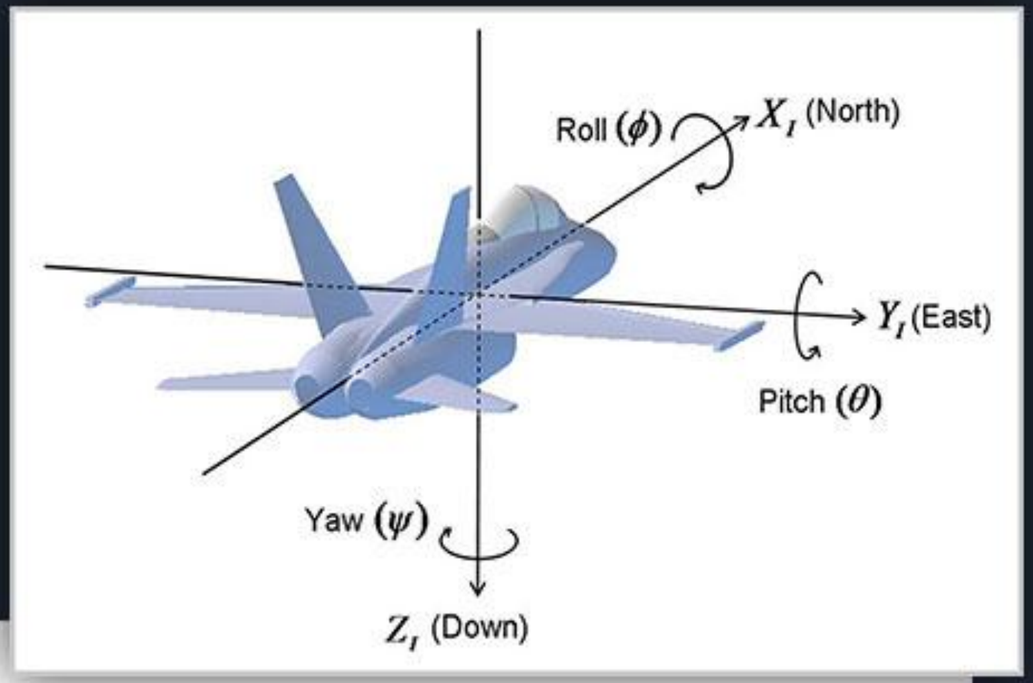
Навіщо це потрібно

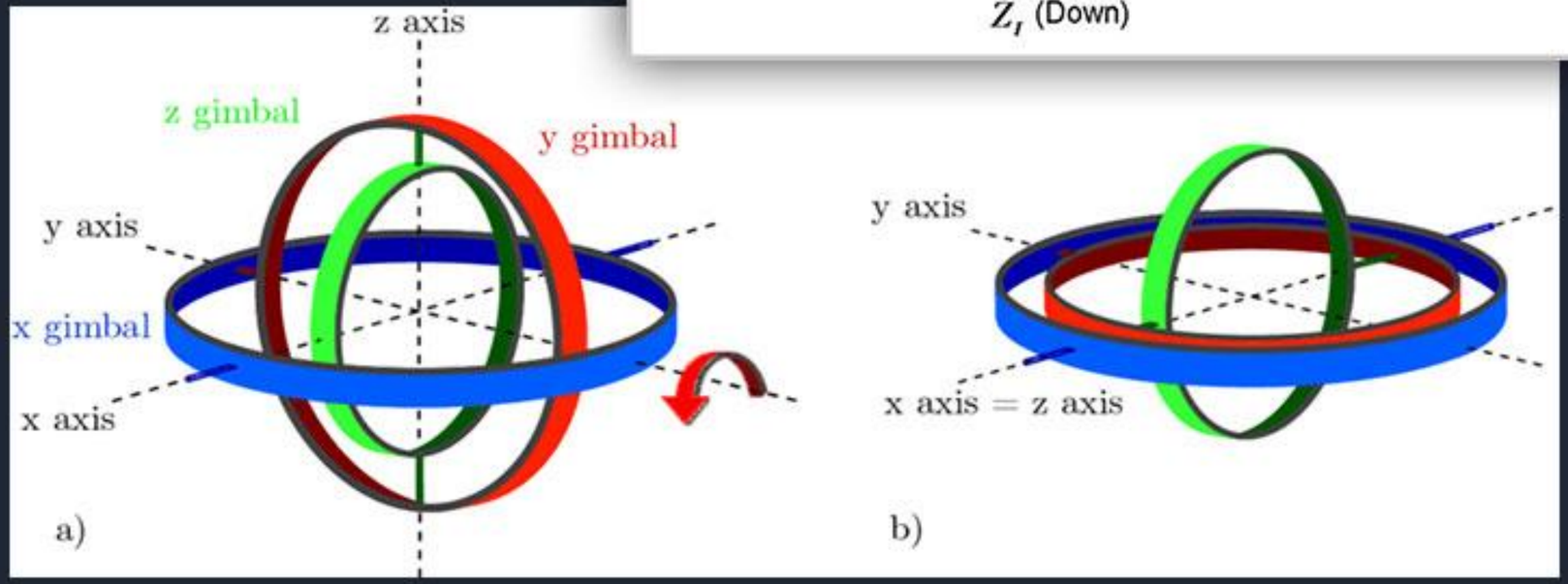
- Повторне використання об’єктів — замість копіювання вручну.
- Зручне оновлення: змінюєш оригінальний префаб — всі копії оновлюються.
- Масова генерація ідентичних елементів (NPC, UI-кнопок, блоків тощо).

Quaternion

Quaternion — це математичний тип, який представляє обертання в 3D-просторі.
Він потрібен, бо звичайні кути (Euler Angles) можуть викликати проблему “gimbal lock” — коли об’єкт не може повернутись у певний напрям.

What are Quaternions?





// Простий поворот на 90 градусів навколо осі Y

transform.rotation = Quaternion.Euler(0, 90, 0);

// Об’єкт дивиться на точку в просторі

transform.rotation = Quaternion.LookRotation(target.position - transform.position);

Parallax Effect (Ефект паралаксу)

Паралакс — це візуальний ефект, при якому об’єкти, що знаходяться далі, рухаються повільніше, ніж ближчі.
Людське око сприймає це як глибину, тому сцена виглядає “живою”.
У 2D іграх це часто використовується для фонів: небо, гори, дерева рухаються з різною швидкістю.

Навіщо це потрібно

- Створює **ілюзію тривимірного простору** в 2D;
- Робить сцену **динамічнішою і красивішою**;
- Використовується у платформерах, шутерах, меню або навіть у UI-анімаціях.

```
public class ParallaxEffect : MonoBehaviour
{
    public float parallaxFactor = 0.5f; // smaller = slower
    private Transform cam;
    private Vector3 lastCamPos;

    void Start()
    {
        cam = Camera.main.transform;
        lastCamPos = cam.position;
    }

    void LateUpdate()
    {
        Vector3 delta = cam.position - lastCamPos;
        transform.position += new Vector3(delta.x * parallaxFactor, delta.y * parallaxFactor, 0);
        lastCamPos = cam.position;
    }
}
```