TECNICAS AVANZADAS DE PROGRAMACIÓN

POOLING

Álvaro San Juan Cervera
alvaro.san@live.u-tad.com

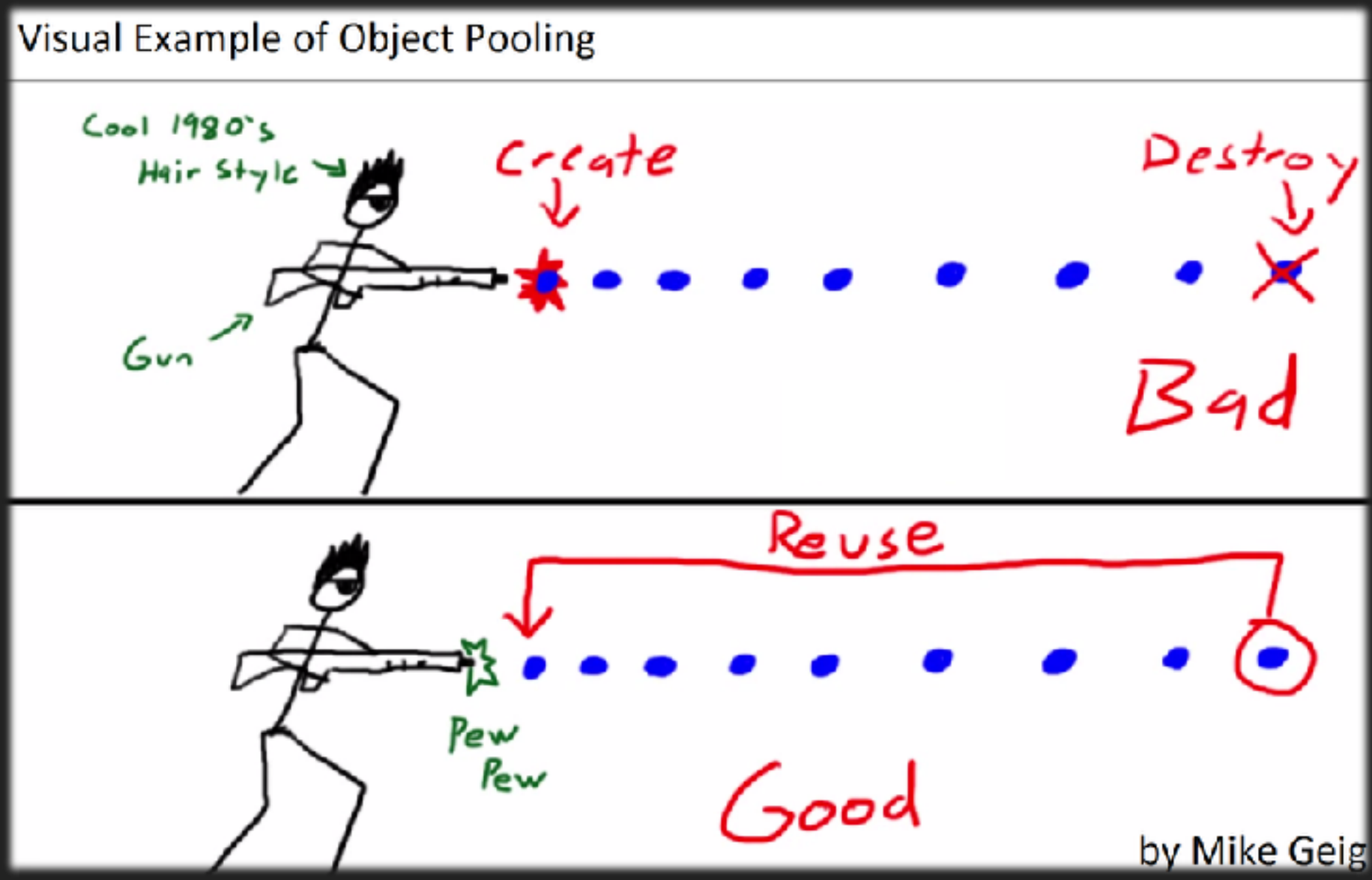
¿QUÉ QUEREMOS CORREGIR?

- Pensemos en una escena típica de un videojuego:
  - Tenemos un personaje con un fusil que hace un gran número de disparos por segundo
  - Además aparecen un gran número de enemigos en pantalla que también realizan disparos
  - Además tenemos explosiones constantes, casquillos de bala cayendo…
  - Toda esta acción hace que en el momento más crítico, perdamos un frame y nuestro personaje muera

¿QUÉ SON LOS OBJECT POOLS?

- Instanciar y destruir objetos puede ser una tarea costosa si lo hacemos frecuentemente
- En Unity el problema se agrava por el recolector de basura
- Instanciar y destruir objetos con demasiada frecuencia puede producir picos en los tiempos de frame (que tienen que mantenerse por debajo de 16,67ms para mantener 60fps estables)
- Si instanciamos los objetos antes de que los necesitemos (por ejemplo, en la pantalla de carga del nivel), podremos reutilizarlos más adelante para evitar instanciarlos durante la partida
- Además, si en vez de destruirlos los devolvemos al pool, ahorraremos destrucciones de objetos (y en caso de Unity, llamadas al recolector de basura)

VISUALMENTE



¿CUÁNDO UTILIZARLO?

- El caso típico es cuando tenemos un objeto que vamos a instancias múltiples veces y posee un ciclo de vida corto (por ejemplo, un proyectil)

¿QUÉ NECESITA?

- Tiene tres componentes básicos:
  - El **cliente**, que solicita los objetos
  - El **mánager** que gestiona los objetos
  - Los **objetos** del pool
- La forma básica de funcionamiento es:
  - El **mánager instancia** en el pool todos los objetos que estime que va a necesitar antes de comenzar la partida
  - En tiempo de partida, el **cliente solicita al mánager** objetos cuando los necesite
  - Cuando los objetos dejen de ser necesarios, se **devuelven los objetos** al pool

ASPECTOS A TENER EN CUENTA

- El mánager tiene que controlar qué objetos están disponibles en caso de que un cliente los solicite
- ¿Cómo estimamos la cantidad de objetos que vamos a meter en el pool?
- ¿Qué hacemos si un cliente nos solicita un objeto que no tenemos disponible?
- Seguramente, tengamos que devolver manualmente los objetos a su estado inicial

EJEMPLO DE IMPLEMENTACIÓN

- Vamos a definir una clase PoolManager que será nuestro mánager:

```
public class PoolManager : Singleton<PoolManager> {
        private Dictionary<string, List<GameObject>> pool;
        private Transform poolParent;

    }
```

- Nuestro pool irá implementado con un **diccionario <String, List <GameObject>>**
- Además, todas las instancias serán hijas de un **poolParent**, para que la escena quede más limpia

EJEMPLO DE IMPLEMENTACIÓN

- Para hacer la carga inicial de un objeto al pool:

```
public void Load(GameObject prefab, int quantity = 1) {
    var goName = prefab.name;
    if (!pool.ContainsKey(goName)) {
        pool[goName] = new List<GameObject>();
    }

    for (int i = 0; i < quantity; i++) {
        var go = Instantiate(prefab);
        go.name = goName;
        go.transform.SetParent(poolParent, false);
        go.SetActive(false);
        pool[go.name].Add(go);
    }
}
```

EJEMPLO DE IMPLEMENTACIÓN

- Para sacar un objeto del pool:

```
public GameObject Spawn(GameObject prefab) {
    if (!pool.ContainsKey(prefab.name)||pool[prefab.name].Count == 0) {
        Load(prefab, 1);
    }

    var l = pool[prefab.name];
    var go = l[0];
    l.RemoveAt(0);
    go.SetActive(true);
    go.transform.SetParent(null, false);
    return go;
}
```

EJEMPLO DE IMPLEMENTACIÓN

- Para devolver un objeto al pool:

```
public void Despawn(GameObject go) {
    if (!pool.ContainsKey(go.name)) {
        pool[go.name] = new List<GameObject>();
    }
    go.SetActive(false);
    go.transform.SetParent(poolParent, false);
    pool[go.name].Add(go);
}
```

EJERCICIO

- Vamos a implementar un spawner de cubos que lance un cubo con velocidad, color y dirección aleatorios cada poco tiempo (uno cada frame)
- Además, periódicamente destruiremos los cubos que lleven existiendo más de 5 segundos
- Vamos a ver si ganamos algo de rendimiento por utilizar pooling
- Ahora vamos a generar más tipos de objetos, además meteremos operaciones costosas en el Awake