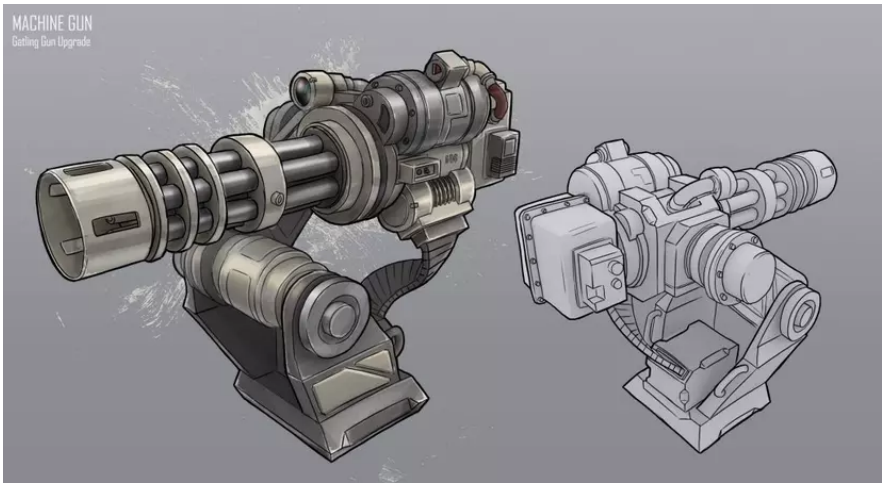
Tối ưu với Object Pooling Pattern trong Unity

Vì sao phải sử dụng Object Pooling?
Trong Unity, để sinh ra một phiên bản copy của một GameObject (thường là prefab), ta sẽ sử dụng hàm Instantiate(), nếu không sử dụng bản copy đó nữa chúng ta sẽ sử dụng hàm Destroy() hay DestroyImmediate().

Hàm Instantiate() sẽ phân bổ, cấp phát tài nguyên bộ nhớ cho copied-GameObject đó, hàm Destroy() xóa bỏ các tài nguyên đã cấp phát và nếu việc này xảy ra liên tục, Garbage Collector (GC) sẽ hoạt động liên tục làm phân mảnh bộ nhớ đồng thời trong frame đó phải chờ GC xử lý xong.

Memory fragmentation
Giả sử 10 khẩu súng có hàng trăm viên đạn như Gatling Gun bắn liên tục, các viên đạn sẽ được sinh ra và xóa đi trong memory liên tục như vậy sẽ ảnh hưởng rất lớn tới performance và đè nặng công việc cho Garbage Collector.



Gatling Gun
Object Pooling Pattern sinh ra nhằm giúp chúng ta tái sử dụng được các viên đạn trong ví dụ trên và giảm thiểu memory fragmentation do GC gây ra.

Object Pooling là gì?
Về ý tưởng, Object Pooling sẽ sinh ra các objects mà ta cần sử dụng trong game trước, sau đó deactive toàn bộ objects đó nếu chưa sử dụng, điều này có thể làm chậm quá trình khởi động của game.

Như vậy, thay vì Instantiate() các objects (như bullets) vào run-time, chúng ta chỉ việc lôi chúng từ Object Pool ra rồi bật active cho object đó, tương tự, thay vì Destroy(), chúng ta sẽ deactive object đó và gửi về lại Pool.

CPU xử lý active và deactive các objects tốt hơn rất nhiều so với việc sinh ra và xóa bỏ chúng trong bộ nhớ run-time, đồng thời chúng ta cũng tối ưu được cho số lượng công việc cần xử lý cho Garbage Collector (GC).

Implement Object Pooling Pattern
Để implement pattern này có rất nhiều source ở ngoài kia, ở đây mình chỉ trình bày ý tưởng của pattern.

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[System.Serializable]
public class Preallocation {
    public GameObject gameObject;
    public int count;
    public bool expandable;
}

public class ObjectPool : MonoSingleton<ObjectPool>
{
    public List<Preallocation> preAllocations;
    [SerializeField] List<GameObject> pooledGobjects;

    protected override void Awake()
    {
        base.Awake();

        pooledGobjects = new List<GameObject>();

        foreach (Preallocation item in preAllocations) {
            for (int i = 0; i < item.count; ++i)
                pooledGobjects.Add(CreateGobject(item.gameObject));
        }
    }

    public GameObject Spawn(string tag) {
        for (int i = 0; i < pooledGobjects.Count; ++i) {
            if (!pooledGobjects[i].activeSelf && pooledGobjects[i].tag == tag)
            {
                pooledGobjects[i].SetActive(true);
                return pooledGobjects[i];
```

```
            }
        }

        for (int i = 0; i < preAllocations.Count; ++i) {
            if (preAllocations[i].gameObject.tag == tag)
                if (preAllocations[i].expandable) {
                    GameObject obj = CreateGobject(preAllocations[i].gameObject);
                    pooledGobjects.Add(obj);
                    obj.SetActive(true);
                    return obj;
                }
        }

        return null;
    }

    GameObject CreateGobject(GameObject item) {
        GameObject gobject = Instantiate(item, transform);
        gobject.SetActive(false);
        return gobject;
    }
}
```

Awake(): sinh ra tất cả các objects cần có cho game
Spawn(): lấy các objects bằng tag của chúng
.expandable: phát sinh thêm object nếu ban đầu cấp phát cho Pool không đủ dùng.