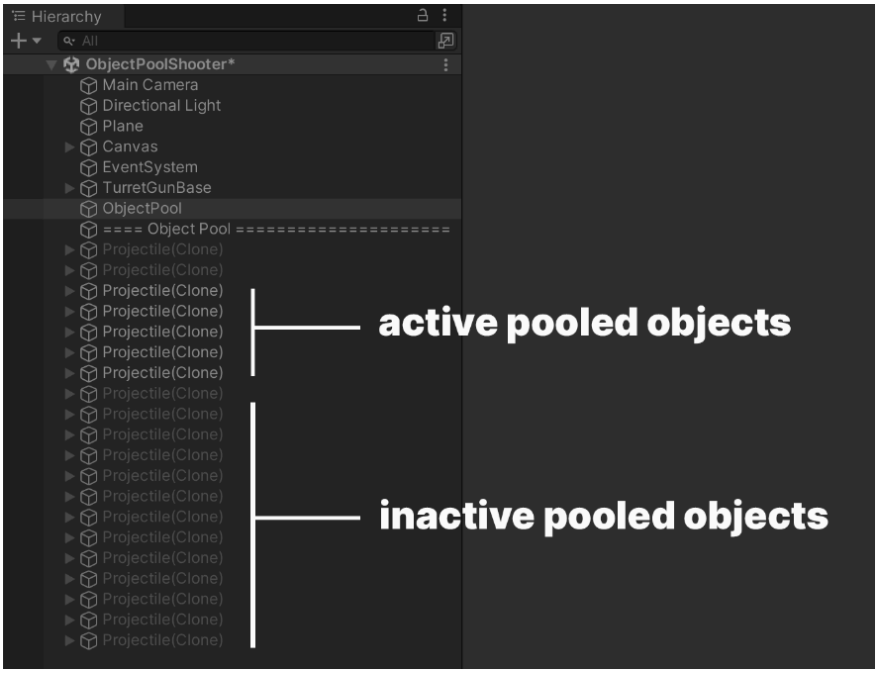
Atividade: Praticando Git e Resumo de Padrões de Projeto Mobile
Nome: Carlos Gabriel Reinheimer

Resumo do Padrão de Desenvolvimento: **Object Pooling**

Dentro do contexto de desenvolvimento de jogos, um quesito muito importante que devemos nos preocupar é com relação a performance. Um jogo que não "roda" bem tende a diminuir o engajamento de jogadores, além de deteriorar a experiência por si só. Com isso em mente, podemos aplicar um padrão de desenvolvimento que visa otimizar a performance reduzindo o processamento necessário pela CPU para executar métodos repetitivos de criação (Instantiate()) e destruição (Destroy()) de objetos, e isso é executado através do conceito de reutilização de objetos.



Com o padrão de *Object Pooling*, sempre que for necessário criar um objeto, em vez de instanciar um novo, o que causaria um uso maior de processamento da CPU além de alocação de memória, podemos apenas buscar um objeto disponível da *pool* e ativá-lo para uso. Assim que ele encerra seu ciclo, apenas desativamos ele novamente de forma que ele volte para a *pool* e possa ser recuperado novamente em outro momento. Isso é muito útil para jogos que necessitam utilizar muitos objetos similares frequentemente, como por exemplo uma torre disparando projéteis.

Além da redução do processamento da CPU, o padrão também auxilia na questão de alocação de memória. Cada vez que um objeto é criado ou destruído de forma primitiva, é necessário alocar e desalocar memória para o mesmo. Isso também auxilia na execução do *Garbage Collector (GB)* da engine - um sistema interno que ajuda a desalocar memória que não está sendo mais utilizada - visto que ele não ficará sobrecarregado, o que causa ainda mais problemas de performance.

Há muitas formas de implementar o sistema de Pooling, incluindo a API própria da Unity, porém abaixo eu apresento um sistema customizado:

```
public class GameObjectPooling : MonoBehaviour {

    public List<GameObject> pooling;
    public GameObject gameObjectPrefab;

    public GameObject GetGameObject(Transform spawnParent) {
                    foreach (var t in pooling.Where(t =>
!t.gameObject.activeInHierarchy)) {
            return t;
        }

        return CreateNewGameObject(spawnParent);
    }

          private GameObject CreateNewGameObject(Transform
spawnParent) {
            var newProjectile = Instantiate(gameObjectPrefab,
spawnParent);

        pooling.Add(newProjectile);
        return newProjectile;
    }
}
```

Como podemos ver, a classe possui uma lista, que possui todos os objetos da pool e um prefab. Há também dois métodos:

GetGameObject: faz uma busca na lista, buscando e retornando um objeto que esteja desativado na mesma.

CreateNewGameObject: caso todos os objetos da lista estejam em uso no momento, o sistema irá então criar de fato um objeto novo e irá adicioná-lo a lista da pool, de forma que seja necessário cada vez menos gastar processamento criando um objeto novo de fato. A lógica para desativar o objeto está presente em outros scripts.

Para utilizar essa *pool*, apenas buscamos uma referência dessa classe e chamamos o método **GetGameObject**, que irá retornar um objeto que podemos utilizar, e após desativarmos o mesmo, ele irá automaticamente ficar disponível na lista da *pool* para ser reutilizado novamente.

Referências:
https://learn.unity.com/tutorial/65df850fedbc2a082fb11029?uv=2022.3&projectId=65de084fedbc2a0699d68bfb