Reporte De Programación de Sistemas de Videojuegos de **Maximo Blasquez**

**Patrones Implementados:**

* PROTOTYPE: Patrón de diseño que te permite clonar objetos sin la necesidad de crear un objeto de 0. Es el único patrón implementado en el proyecto por las siguientes razones. Es un patrón que viene parcialmente implementando en Unity con los prefabs y principalmente de la clase INSTANTIATE. Tambien, la clase prototype es usada en el script enemyVFX para la instanciación de las partículas.

**Patrones no Implementados:**

* POOL: Patrón de diseño que nos permite hacer una piscina de objetos para almacenar y reutilizar los objetos agregados. Si esta no tiene, Los crea. La principal observación del proyecto es que los spawner lo único que hacía era generar enemigos sin fin y que lo primero que se tenía que agregar sería una pool.

Para eso, mire un tutoriales y páginas para entender como armar una y termine armando una pool de pools (que no use, solo la arme para el enemigo). Lo que hacía era poner un límite en la pool a usar y los generaba hasta cierta cantidad. Al llegar al límite dejaba de crear y toda instanciación que viniera después la reutilizaba permitiendo un mejor rendimiento a la hora de generar objetos. Hubo un problema y es que se generaban algunos enemigos y se iban para el costado (lo dice el warning del navMesh) pero lo arregle usando el warp de Unity (teletransporta un objeto a X posicion).

* FLYWEIGHT: Patrón de diseño que nos permite mantener más objetos desde la RAM sin la necesidad de instanciarlos desde un solo objeto todo el tiempo. En el proyecto la use para los sonidos y los VFX de los enemigos. Lo que había notado, era que por cada enemigo que spawneaba tenía una lista de sonidos y Partículas que se instanciaban de manera random en sus dichos script. Para optimizarlo, cree dos scriptable objects, uno de vfx y otro de sonidos respectivamente. Luego cambiara las variables/listas donde se guardaban los objetos y pondría los scriptable objects en su lugar. Luego los llamaría en sus respectivas funciones. Como extra, Se usaba el wrapper del RandomComponent en los script, yo lo que hice es moverlo a las listas de los dos scriptable objects para ser llamadas desde ahí.
* SERVICE LOCATOR: Service Locator es un patrón de diseño que nos permite llamar otras funciones a través (obviamente) de un service Locator. Creas un “Service” donde definimos los parámetros que queramos crear de nuestro objeto y lo llamamos a nuestro script donde usemos una función relacionada a eso. Yo por ejemplo la use para llamar los sonidos de explosión de los enemigos. Para que funcionen, Los sripts tipo service tienen que ser game objects.
* WRAPPER/DECORATOR: Aunque estaba implementado tuvimos que hacer uno de 0. Patron de diseño que nos permite agregar parámetros a un objeto sin modificar su comportamiento base (Por ejemplo, refactoring guru utiliza el ejemplo de cuando nos ponemos una campera, abrigo, ect). Yo hice mi propio healthComponent básico para agregárselos tanto a las estructuras como a los enemigos. Definimos al wrapper como variable arriba para invocarla y luego les pasamos y definimos sus parámetros a los objetos, que en este caso, le agregamos un sistema de vida. Mi única contra personal es que la vida se la defino en el awake al correr el proyecto.