Arquitectura de Software:

El juego es un arcade en el que el objetivo es obtener la mayor puntuación antes de morir. El jugador controla una nave espacial y debe disparar a asteroides mientras los esquiva para evitar colisiones y mantenerse con vida. El juego es sencillo y fácil de entender, brindando una experiencia de juego emocionante.

Controles:

W: Moverse hacia adelante
A: Rotar a la izquierda
S: Moverse hacia atrás
D: Rotar a la derecha
Espacio: Disparar
Alt + F4: Salir del Juego

Agile:

El enfoque ágil se utiliza en el desarrollo del juego para garantizar una entrega continua y adaptativa. Se emplean metodologías ágiles como Scrum o Kanban para gestionar el desarrollo, las tareas y las iteraciones. El equipo trabaja en colaboración, fomentando la comunicación constante y la retroalimentación para garantizar un proceso de desarrollo ágil y eficiente.

Patrones de diseño utilizados en el proyecto:

Patrón Singleton:
Clase: GameManager
Descripción: La clase GameManager implementa el patrón Singleton al tener una única instancia de sí misma. Esto garantiza que solo haya una instancia activa del GameManager en todo el juego, lo que permite un acceso global y centralizado a su funcionalidad, como el seguimiento de la puntuación, el control de las balas y la gestión del estado del juego.

Patrón Observer:
Clase: GameManager
Descripción: La clase GameManager utiliza el patrón Observer para notificar a otros objetos sobre cambios en el contador de balas. Define el evento OnBulletCountChanged y el delegado BulletCountChanged, y luego notifica a los suscriptores cuando el contador de balas cambia mediante el método UpdateBulletCount(). Esto permite a otros componentes o sistemas del juego reaccionar dinámicamente a los cambios en el contador de balas y realizar acciones correspondientes.

Patrón Object Pool:
Clase: EnemySpawner
Descripción: La clase EnemySpawner utiliza el patrón Object Pool para administrar un grupo de enemigos disponibles y enemigos activos. Al inicio, se llena un "pool" con enemigos inactivos, y cuando se necesita un enemigo, se obtiene uno del pool en lugar de crear uno

nuevo. Esto evita los costos de rendimiento asociados con la creación y destrucción frecuente de objetos enemigos, mejorando así la eficiencia del juego.

Patrón Command:
Clase: ShipController
Descripción: Se utiliza el patrón Command para encapsular y ejecutar los comandos de movimiento de la nave. La clase ShipController implementa una interfaz ICommand que define el método Execute(float input). Esto permite crear diferentes comandos de movimiento y asignarlos dinámicamente al controlador de la nave. Los comandos encapsulan la lógica de movimiento y se pueden ejecutar y deshacer según sea necesario, proporcionando flexibilidad y modularidad en el manejo de las acciones de movimiento de la nave.

Estos patrones de diseño contribuyen a la organización, modularidad y reutilización del código en el desarrollo del juego, lo que resulta en un código más limpio, mantenible y escalable.

Bases de Datos:

Sql vs Sqlite:
SQL y SQLite son dos conceptos relacionados pero diferentes en el ámbito de las bases de datos.

SQL es un lenguaje de programación utilizado para administrar y manipular bases de datos relacionales. Es un estándar ampliamente aceptado que permite crear, consultar, modificar y eliminar datos en una base de datos. SQL se utiliza con diferentes sistemas de gestión de bases de datos (DBMS) como MySQL, PostgreSQL, Oracle, SQL Server, entre otros.

Por otro lado, SQLite es un motor de base de datos relacional ligero y autónomo que implementa SQL. A diferencia de otros DBMS, SQLite no se ejecuta como un proceso independiente en un servidor, sino que se incorpora directamente en la aplicación. Esto significa que la base de datos SQLite se almacena como un archivo local en el sistema de archivos del dispositivo y no requiere una conexión de red para acceder a ella.

Algunas diferencias clave entre SQL y SQLite son:

Escalabilidad: SQL se utiliza en sistemas de gestión de bases de datos más grandes y escalables que admiten múltiples usuarios y grandes volúmenes de datos. SQLite, por otro lado, está diseñado para aplicaciones de menor escala y es adecuado para aplicaciones de escritorio, móviles o web más pequeñas.

Implementación: SQL es un lenguaje estándar que se utiliza con diferentes DBMS, mientras que SQLite es un motor de base de datos específico con su propia implementación.

Persistencia: Las bases de datos SQL generalmente se mantienen en un servidor y conservan su estado incluso después de que se apague el servidor. SQLite, al ser un archivo local, es persistente en el sentido de que los datos se conservan en el archivo, pero si el archivo se borra o se pierde, los datos se perderán.

Configuración y administración: Los sistemas de bases de datos SQL requieren una configuración y administración más complejas, como instalación, configuración del servidor, asignación de recursos, seguridad, copias de seguridad, etc. SQLite, al ser una biblioteca incorporada en la aplicación, no requiere una configuración ni administración especial.

Porque he elegido sqlite:

Portabilidad: SQLite es un motor de base de datos incorporado y liviano que no requiere una instalación separada o configuración del servidor. Esto significa que puedes incluir la biblioteca SQLite directamente en tu proyecto de Unity, lo que facilita la distribución y la portabilidad del juego. No es necesario configurar una conexión de red o depender de un servidor externo para acceder a la base de datos.

Rendimiento: SQLite es conocido por su rendimiento rápido y eficiente, especialmente en aplicaciones de pequeña a mediana escala. Debido a que SQLite almacena la base de datos en un archivo local en el sistema de archivos, las operaciones de lectura y escritura son rápidas y no requieren una conexión de red.

Tamaño reducido: SQLite tiene una huella de memoria y almacenamiento pequeña en comparación con los sistemas de gestión de bases de datos más grandes.