Unión Europea
Fondo Social Europeo
"El FSE invierte en tu futuro"

Proyecto de desarrollo de aplicaciones multiplataforma
Juego Bullet Hell Arcade

CICLO FORMATIVO DE GRADO SUPERIOR
Desarrollo de Aplicaciones Multiplataforma (IFCS02)

Curso 2022-23

Autor/a/es:
Mario González Resa

Tutor/a:
José Luis González Sánchez

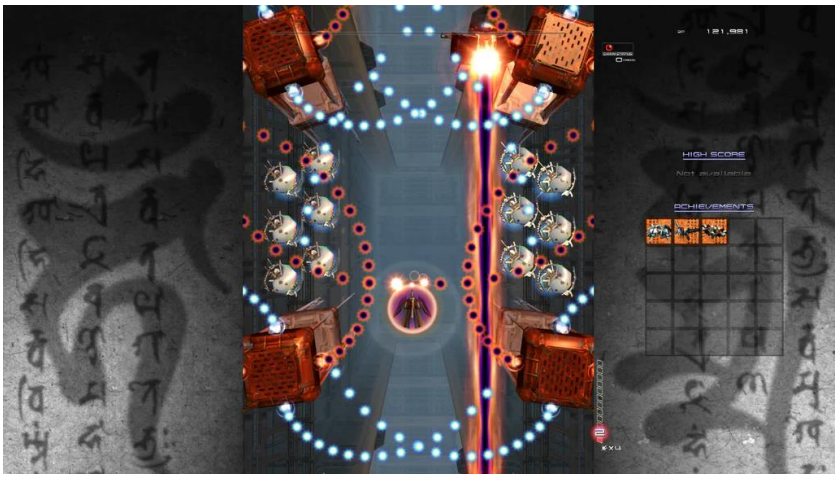
Departamento de Informática y Comunicaciones
I.E.S. Luis Vives

ÍNDICE

1 INTRODUCCION

1.1 DESCRIPCIÓN

Se trata de un juego del género **Bullet Hell**, es decir*, esquivar balas* y *sobrevivir* el *máximo tiempo* posible para así conseguir el *máximo de puntuación*; además de lograr generar una sensación de juego competitivo sin ser estrictamente un *multijugador*, usando una tabla de puntuaciones.



"Ikaruga" (2001) // Género: Bullet Hell

El usuario podrá jugar una partida de modo casual, aguantando el máximo posible para así obtener una puntuación más alta que la del resto de jugadores.

Se dispondrá de una base de datos donde se almacenarán tanto a los usuarios como sus puntuaciones más altas.

Igualmente, no todo el mundo posee conexión a internet las 24 horas del día, por lo que pienso que un modo **offline/sin iniciar sesión** es muy necesario, aunque se pierda la posibilidad de ver esa tabla de puntuaciones.

Con este proyecto, busco ampliar mis conocimientos sobre el mundo del desarrollo de software, en uno de los campos que más me llaman la atención junto al de las *Inteligencias Artificiales,* los **videojuegos**.

Personalmente también me lo planteo como un reto el hecho de lograr conectar tecnologías conocidas con desconocidas. También busco mejorar mis dotes de diseñador de interfaces, muy necesario.

2 REQUISITOS

2.1 REQUISITOS FUNCIONALES

El *usuario* podrá realizar las siguientes acciones:

- Jugar
- Iniciar Sesión

- Cerrar la Sesión
- Crear una Cuenta
- Ver una tabla de puntuaciones
- Ver su propio perfil
- Jugar de modo **offline** o **sin iniciar sesión**, en caso de no disponer de conexión a internet.
- Elegir entre tres dificultades.
- Cambiar la contraseña.
- Borrar la Cuenta

2.2 REQUISITOS NO FUNCIONALES

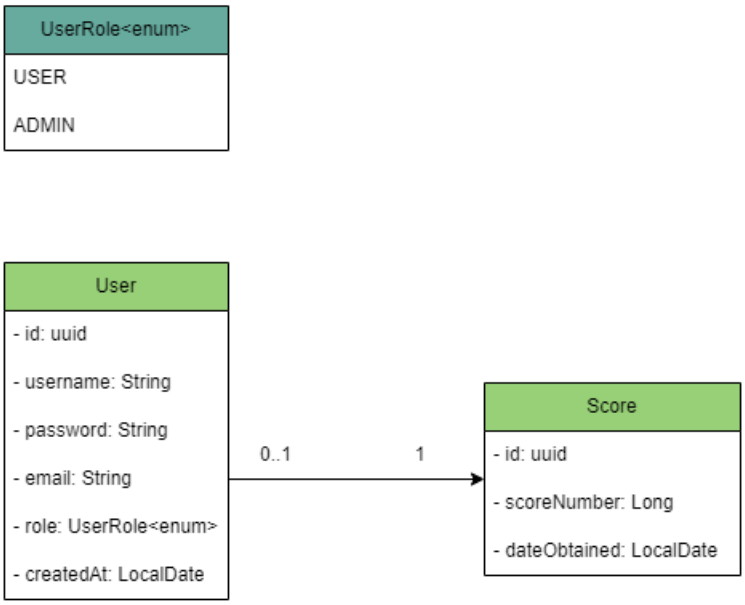
- La base de datos será SQL.
- Se dispondrá de un servicio API-REST que comunique la base de datos con la aplicación.
- El usuario iniciara sesión con su nombre de usuario y su contraseña, luego la aplicación almacenara el token correspondiente y este será utilizado, mientras no caduque, para realizar la comunicación con el servidor.
- La puntuación se subirá de manera automática si la puntuación es superior a la obtenida anteriormente.
- Para poder actualizar la contraseña, se deberá disponer de la contraseña actual, como medida de seguridad.

2.3 REQUISITOS DE INFORMACIÓN

Se dispondrán de **dos** entidades:

- Usuarios
- Puntuaciones

La relación entre ellas será simple, un usuario podrá tener desde 0..1 puntuación y la puntuación solo podrá pertenecer a un único usuario.



"Diagrama de clases"

"Diagrama entidad-relación"

3 ANÁLISIS DE MERCADO

La idea del proyecto surgió principalmente de **tres** videojuegos:

- ***Akane*** (2018)

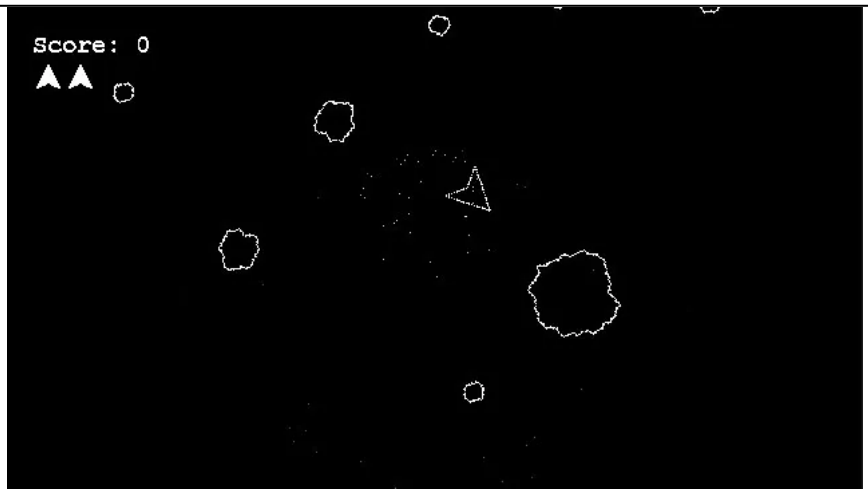
- ***Nier: Automata*** (2017)

En el **primer caso**, es un juego basado en sobrevivir el mayor tiempo posible sobre una **arena de combate,** sobre la cual no dejan de aparecer enemigos. Desgraciadamente este juego no cuenta con tabla de puntuaciones, por lo que su aspecto competitivo se reduce únicamente a hablar con amigos o subir la puntuación máxima conseguida a algún foro por internet. Igualmente me resulta interesante el reto que ofrece, porque siempre perderás con tan solo un toque de algún enemigo.

En el **segundo caso**, me inspire en los pequeños fragmentos que, a su vez, se inspiraban en el propio género **bullet hell.** El resto del juego no tiene nada que ver, pero las opciones jugables, y el permitir destruir esas propias balas de distintas formas me resulto interesante.

Finalmente, mi idea **final** de diseño gráfico y en parte jugable proviene de un videojuego bastante más antiguo:

- ***Asteroids*** (1979)



Para el enfoque arcade este último caso es perfecto, y cuenta con las mismas razones que por ejemplo *Nier*, el poder destruir elementos, y el reto ofrecido por *Akane*, traducido a los distintos niveles de dificultad que serán ofrecidos.

En la perspectiva más estricta, ninguno de esos tres juegos son un *bullet hell* en su totalidad, pero sí que encuentro elementos individuales que podrían hacer uno muy competitivo y divertido.

4 ANÁLISIS TECNOLÓGICO

4.1 BACKEND

4.1.1 Gestión de la información

La gestión de la información será gestionada usando una **base de datos relacional** o **SQL**, debido a que, para este problema en específico, pienso que es la mejor solución.

- Una de las mayores ventajas ofrecidas por una base de datos **NoSQL** es el poder congelar la información, algo muy útil para mantener un historial de pedidos. Pero en este caso, al usuario solo le interesa mantener una única puntuación, la más alta obtenida, y el cambio de contraseña no afecta en nada a ese almacenamiento, por lo tanto, para poder mantener esa relación de forma adecuada, la mejor solución para este problema es aplicar una base de datos **SQL.**
- Debido a eso, quedo descartado el uso de **Firebase**.

- Para poder acceder a los datos de forma correcta, he decidido usar un servicio API-REST, ya que me permite acceder a los datos de la propia base de datos, y aplicar una serie de capas de seguridad para así mantener integra la información; desde aquí podría gestionar las distintas operaciones, como el registro de nuevos usuarios, los inicios de sesión más la generación de tokens…

- Para lanzar el servidor de la base de datos, se lanzará en un **contenedor** o **Docker,** con su respectivo **Docker-Compose**.

4.1.2 Servicio API-REST

- La **API-Rest** se realizaría con **Java** o **Kotlin**; ambas ofrecen un apoyo parecido, si bien *Kotlin* facilita la implementación reactiva gracias a las corrutinas y flujos, frente a *Java*, y el uso de flux y mono.

    - Estos lenguajes han sido seleccionados por un motivo de diseño del proyecto que se mencionó en la introducción, mi idea es juntar tecnología conocida con desconocida; de ese modo, el *backend* será realizado con tecnología que he usado a lo largo del curso.

    - Este servicio será reactivo, principalmente por la gestión de puntuaciones, que al ser accedida un X número de veces de forma continua, y esta a su vez necesitar información del usuario asociado, mejorará la velocidad del servicio.

    - Este servicio, además de contar con un lenguaje de los ya mencionados, se aplicaría un **framework,** y la decisión sería: **Ktor** o **Spring Boot.**

        - **Ktor:**

- **Puntos positivos:** Es ligero, y ofrece mayor control sobre sus distintas configuraciones.

- **Puntos negativos:** Es menos maduro que la contraparte, ofrece menos integraciones, por lo que se gana en control se pierde en eficacia y optimización de código. Solo es posible con Kotlin.

▪ **<u>Spring Boot</u>:**

- **Puntos positivos:** Tanto Java como Kotlin son soportados, gracias a su historial ofrece una seguridad en la implementación difícil de alcanzar.

- **Puntos negativos:** Si bien su implementación es relativamente más rápida, la configuración es más estricta que en Ktor, aunque subsanado con la cantidad ingente de información que hay por Internet, es un factor importante a tener en cuenta. Es bastante más pesado, y tarda más en ejecutarse por primera vez.

- Personalmente, prefiero implantar ***Spring Boot***, debido a la robustez que ofrece gracias a su madurez e historial.

- En la elección del lenguaje, usare **Kotlin**, debido a la facilidad de implementar la **reactividad** que busco.

- Además, para probar los diferentes *end points,* se hará uso de una plataforma diseñada para API llamada **Postman**.

4.2 FRONTEND

4.2.1 Motor de desarrollo

- **Unreal Engine** queda **descartado** desde un comienzo por su alta dificultad de aprendizaje tanto del programa como del lenguaje usado principalmente, **C++,** si bien ofrece una serie de ventajas que los demás motores no son capaces de cubrir, como su gran capacidad para gráficos 3D o entornos virtuales; son **ventajas** que para este proyecto **no** son **necesarias**. Además, es conocido por no manejar de forma correcta entornos 2D.
- La decisión se decide principalmente entre dos motores *alternativos*: **Unity** o **Godot.**

    - **Unity:**
        - **Puntos positivos:** Cuenta con una comunidad muy activa, el aprendizaje es de dificultad media y cuenta con una gran tienda de *assets* **oficial**. Gran número de plataformas a las que exportar el resultado final.

        - **Puntos negativos:** Solo permite un lenguaje de desarrollo, **C#,** es **software privativo**. Es un motor **pesado** para ejecutar en un ordenador **modesto**.

    - **Godot:**
        - **Puntos positivos:** En cuanto a curva de aprendizaje, es el más sencillo para comenzar, gracias al diseño por nodos. Al ser un **motor** de **código libre**, entre otras ventajas, es sencillo encontrar gran parte de ayuda por Internet. Permite el uso tanto de **C#,** de **C++** y finalmente, de su propio lenguaje, **GDScript**, para desarrollar el juego. Es el motor perfecto para el desarrollo en 2D. Es un motor **ligero** de ejecutar.

        - **Puntos negativos: No** dispone de una tienda oficial de *assets*, por lo que el resultado final se debe de hacer a mano o como buenamente se pueda. El número de plataformas de exportación es más limitado. Los ambientes 3D no son su especialidad. Su motor de físicas es algo inferior al ofrecido en Unity. Es el menos conocido laboralmente.

- **Personalmente,** me encanta la programación funcional de **Python,** y lo más parecido a eso en este mundillo es *GDScritp***;** el que sea software de código libre me llama la atención. Finalmente, pienso que al ser una experiencia nueva y ofrecer la mejor curva de aprendizaje, me decantaría por **Godot**.

    - La **elección del lenguaje** en ese sentido es clara, usaría **GDScript,** ya que es un lenguaje desarrollado única y exclusivamente para este motor, y es el lenguaje que cuenta con la mejor implementación. Además, al ser tan parecido a *Python,* lo veo una buena forma de cimentar conocimientos parecidos para el mismo.

4.3 HERRAMIENTAS DE DISEÑO

- **Draw.io:** Editor de diagramas.

- **Paint:** Editor de imágenes.

4.4 HERRAMIENTAS ADICIONALES

- **Itch.io:** Plataforma de distribución digital. Haciendo uso *específico* de la tienda de **assets.**

5 BACKEND-EXPLICACIÓN

5.1 MODELOS

Siguiendo el diagrama de clases, contamos con dos elementos:

- **Usuario / User**
  - **ID:** Id que identifica al elemento en la base de datos. (***Primary Key***) (***UUID***)
  - **Username:** Nombre de usuario (***String***)
  - **Password:** Contraseña del usuario, almacenada *cifrada* con **Bcrypt** por seguridad. (***String***)
  - **Email:** Correo electrónico del usuario. (***String***)
  - **Role:** Rol del usuario; existen dos tipos:
    - **USER**
    - **ADMIN**
  - **CreatedAt:** Fecha donde se creó al usuario. (***LocalDate***)

- **Puntuación / Score**
  - **ID:** Id que identifica al elemento en la base de datos. (***Primary Key***) (***UUID***)
  - **UserId:** Id que relaciona al usuario con la puntuación. (***Foreing Key***) (***UUID***)
  - **ScoreNumber:** El número obtenido de la puntuación conseguida. (***Long***)
  - **DateObtained:** Fecha donde se obtuvo la puntuación. (**LocalDate**)

5.2 DTO

Según las distintas necesidades del programa se han desarrollado distintos DTO para poder facilitar el trabajo con los datos.

5.2.1 Usuarios

- **UserDTOLogin:** Usado para iniciar sesión.
  - **Username** (String)
  - **Password** (String)

- **UserDTORegister:** Usado para el registro de nuevos usuarios
  - **Username** (String)
  - **Password** (String)
  - **RepeatPassword** (String)
  - **Email** (String)

- **UserDTOCreate:** Usado para la creación de un nuevo usuario por parte de un administrador.
  - **Username** (String)
  - **Password** (String)
  - **Email** (String)
  - **Role** (UserRole)

- **UserDTOResponse:** Usado para dar una respuesta genérica.
    - **Username** (String)

- **UserDTOProfile:** Usado para mandar la información que un usuario puede ver sobre él mismo.
    - **Username** (String)
    - **Email** (String)
    - **CreatedAt** (String)
    - **Score** (ScoreDTOResponse?)
        - **Si el usuario es nuevo y no ha subido ninguna puntuación es posible que no tenga ninguna.**

- **UserDTOLeaderBoard:** Usado para facilitar el acceso según la puntuación obtenida y su posición.
    - **Position** (String)
    - **Username** (String)
    - **Score** (ScoreDTOResponse)

- **UserDTOPasswordUpdate:** Usado para la actualización de la contraseña de un usuario.
    - **ActualPassword** (String)
    - **NewPassword** (String)
    - **RepeatNewPassword** (String)

5.2.2 Puntuaciones

- **ScoreDTOCreate:** Usado para la creación de una nueva puntuación
    - **UserId** (String)
    - **ScoreNumber** (String)

- **ScoreDTOResponse:** Usado cuando se solicite una puntuación.
    - **ScoreNumber** (String)
    - **DateObtained** (String)

5.3 MAPPERS

Hacemos uso de distintos *mappers* como funciones de *extensión* para el paso de Modelo a DTO.

En estas funciones es donde, generalmente, indicamos la fecha de creación tanto de usuarios como de las puntuaciones. También, y para facilitar el envió y recibo de información, es donde se transformarán ciertos datos de un dato complejo a String (es decir, para facilitar el uso de **JSON** como formato para el intercambio de datos).

Además, aunque esta comprobación se realice en otra ubicación más idónea, se comprueba en el paso de **UserDTORegister** a Modelo el que las contraseñas pedidas coincidan.