Diseño y Programación Orientada a Objetos
Practica 2 Semestre

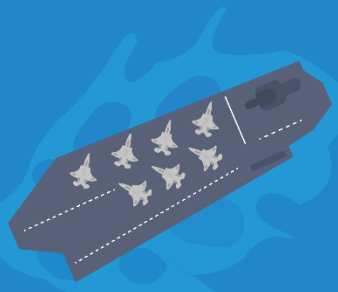
BATTLESHIP

Grupo C7

- **Esteban Ariel Gómez Agüero** (estebanariel.gomez@students.salle.url.edu)
- **David Deu Castells** (david.deu@students.salle.url.edu)
- **Alex Cano Gallego** (alex.cano@students.salle.url.edu)
- **Tomas Uzcudun** (tomas.uzcudun@students.salle.url.edu)
- **Kevin Eljarrat Ohayon** (kevin.eljarrat@students.salle.url.edu)

Project Leader:

Jordi Malé Carbonell (jordi.male@salle.url.edu)



Índice

Especificaciones del proyecto

Para este segundo semestre de Diseño y Programación Orientado a Objetos se nos ha pedido implementar un videojuego retro, utilizando todas las herramientas que hemos ido aprendiendo durante este curso. El videojuego que se nos ha pedido implementar no es otro que el *Hundir la flota*, dicho juego se juega entre dos jugadores donde ambos se atacan hasta acabar con los barcos del rival siguiendo un sistema de turnos.

El objetivo del proyecto era hacer una versión similar del videojuego, pero con un pequeño cambio, no se jugará contra otro jugador y el sistema de turnos será completamente sustituido. En nuestra versión, lucharemos contra la IA en lugar de otros jugadores, podremos enfrentarnos hasta contra 4 enemigos a la vez y, en lugar de esperar nuestro turno podremos atacar pasado cierto tiempo simulando la recarga de nuestros cañones.

El proyecto ha sido exclusivamente implementado con el lenguaje de programación Java debido a esto el apartado visual ha sido creado usando Swing, una librería gráfica que nos ha proporcionado las herramientas necesarias para mostrar y recoger información del usuario a través de diferentes pantallas.

El primer lugar, hicimos el registro y login de los usuarios en la plataforma, para ello hemos tenido que crear una base de datos para guardar la información de los usuarios. Esta base de datos fue creada con MySQL ya que es a lo que estábamos más acostumbrados todos los miembros del grupo.

En segundo lugar, para poder seguir con los diferentes aspectos del videojuego teníamos que implementar el menú principal. En esta ventana el usuario podría decidir entre diferentes opciones como: crear una nueva partida, cargar una partida guardada, mostrar las estadísticas de los jugadores y abrir el menú de ajustes.

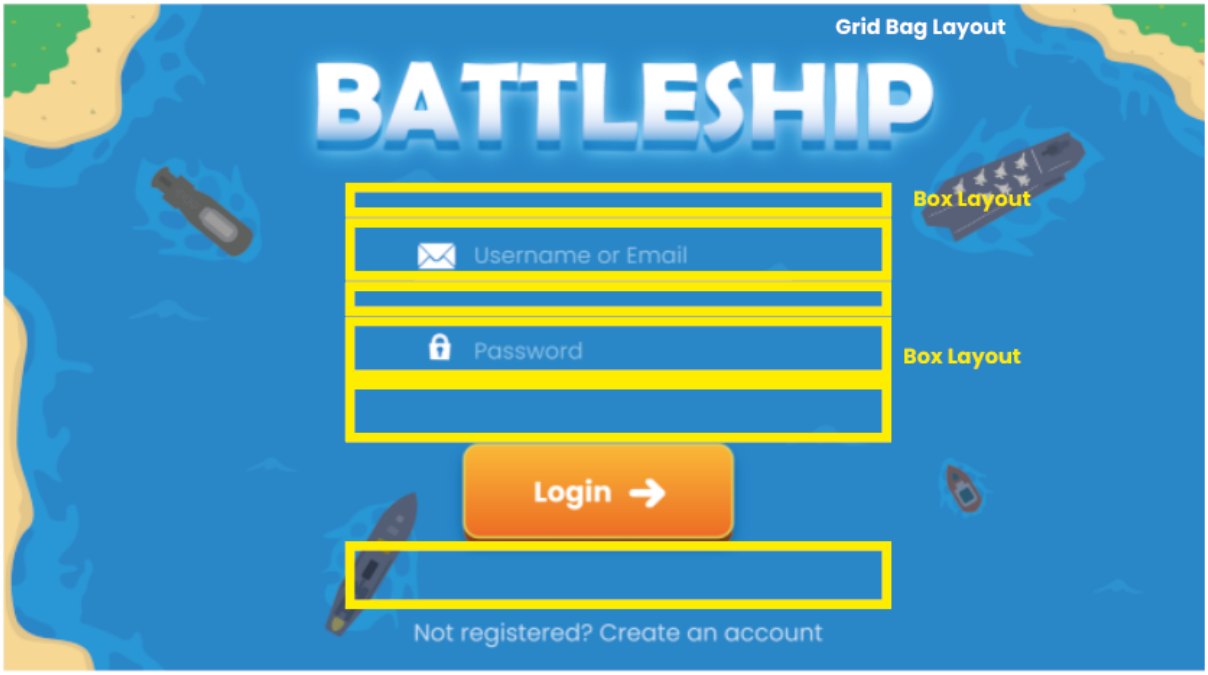
Cuando teníamos alguna de las ventanas creadas seguíamos implementado la lógica de estas de esta forma realizamos las siguientes implementaciones:

- Fase de preparación: el usuario seleccionará sus barcos y los podrá colocar en pantalla siguiendo los requerimientos marcados por el enunciado. Una vez el usuario acabe comprobaremos que todos los datos son correctos y, la IA se encargará de colocar sus barcos en sus respectivos tableros dando lugar a la partida.
- Fase de la partida: el usuario seleccionará que casillas atacar en su propio tablero mientras que la IA atacará cada vez que tenga disponible un ataque, ambos tendrán un tiempo de espera entre ataques. Cuando solo quede un jugador con vida acabará la partida. Por último, el usuario podrá parar la partida cuando quiera clicando un botón y guardando la partida asignándole un nombre.
- Ventana de ajustes: el usuario tendrá la opción de desconectarse o borrar su cuenta.
- Ventana de estadísticas: el usuario será capaz de ver sus propias estadísticas además de la de los demás usuarios que estén registrados en la aplicación.
- Cargar una partida: Cuando el usuario seleccione cargar partida aparecerá un menú desplegable donde se le mostrará todas las partidas guardadas, en caso de seleccionar una pasaremos directamente a la “fase de la partida” cargando toda la información que contenía el fichero.

Para acabar, creemos que es importante comentar que todo el proyecto ha sido implementado siguiendo una arquitectura por capas además de usar el patrón de diseño Modelo-Vista-Controlador. Con esto hemos conseguido unas vistas desacopladas del modelo de forma que hemos podido trabajar independientemente de la forma en la que hemos implementado las vistas, teniendo en cuenta que somos un grupo de 5 personas esto nos ha ayudado a poder organizarnos mejor.

Diseño de la interfaz gráfica

Vista del formulario para iniciar sesión



La ventana del login está formada principalmente por un **GridBagLayout**, el cual tendrá la imagen de fondo, y dentro el cual pondremos **6 Box Layout**.

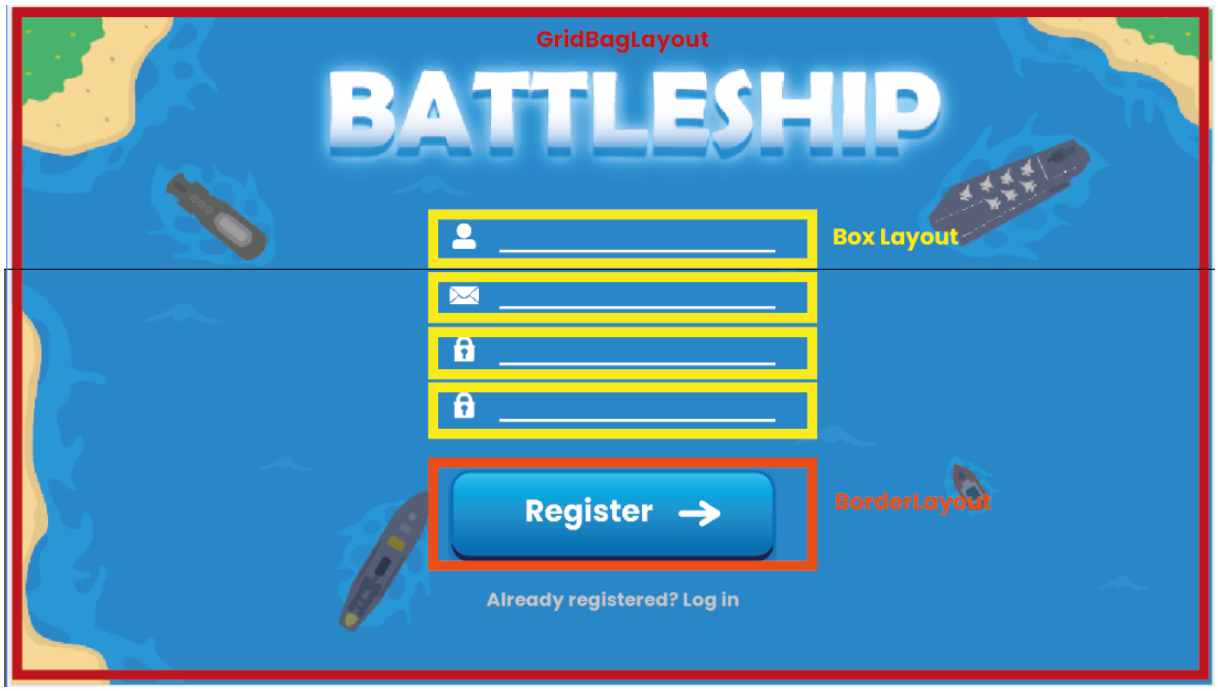
- Utilizamos un JPanel con un Box Layout para poner:
  - El icono del email como un JImagePanel.
  - El JTextField como input para insertar el email o el nombre del usuario.

- Seguidamente utilizamos otro JPanel con un Box Layout para poner:
  - El icono del candado como un JImagePanel.
  - El JPasswordField como input para insertar la contraseña del usuario.

- En tercer lugar, está el botón de login para poder iniciar sesión que será un JPanel con un BorderLayout con dentro:
  - Un JImagePanel con la imagen del botón.

- Y, por último, tendremos un JLabel que si es clicado te llevará a la ventana para poder registrar una cuenta nueva.

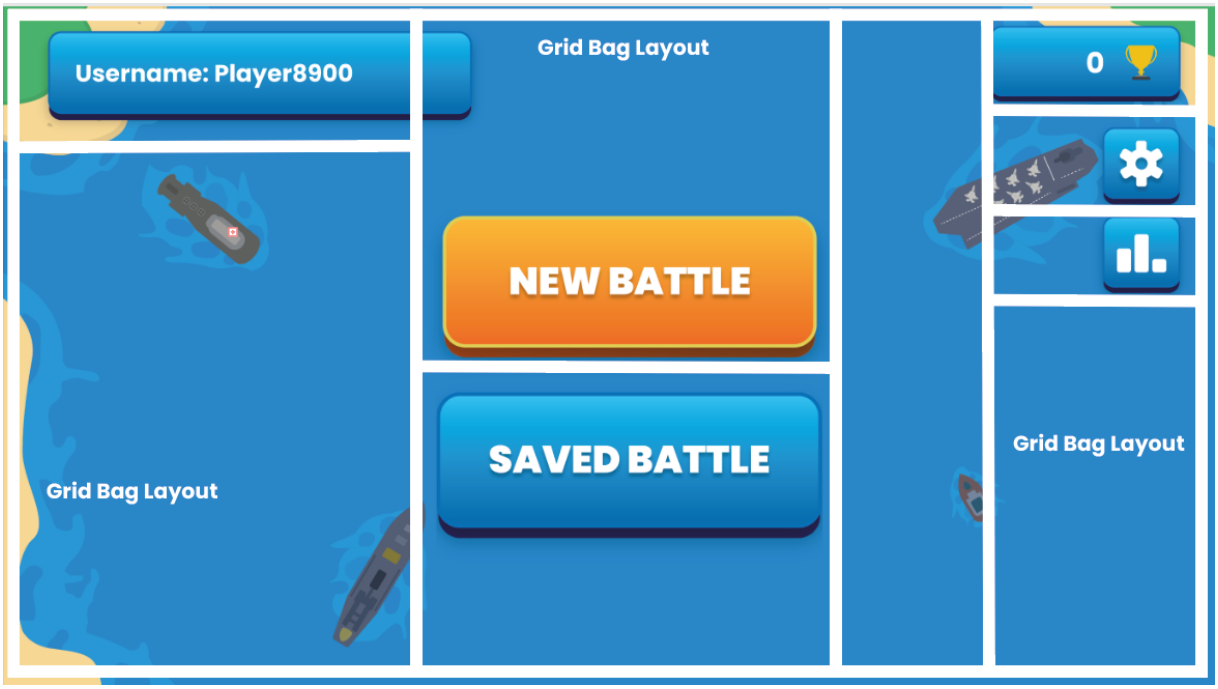
Vista del formulario para registrar un usuario



La ventana de registro está formada por un JImagePanel de la imagen de fondo con un GridBagLayout que contiene:

- Cuatro BoxLayouts con JTextFields y JPasswordFields para poder introducir el nombre del usuario, el correo, la contraseña y la confirmación de la contraseña.
- Un JPanel con un BorderLayout que tendrá un JImagePanel del botón para registrarse.
- Y por último un JLabel con el texto para poder pasar a la ventana de login en caso de que ya nos hayamos registrado.

Vista del menú principal



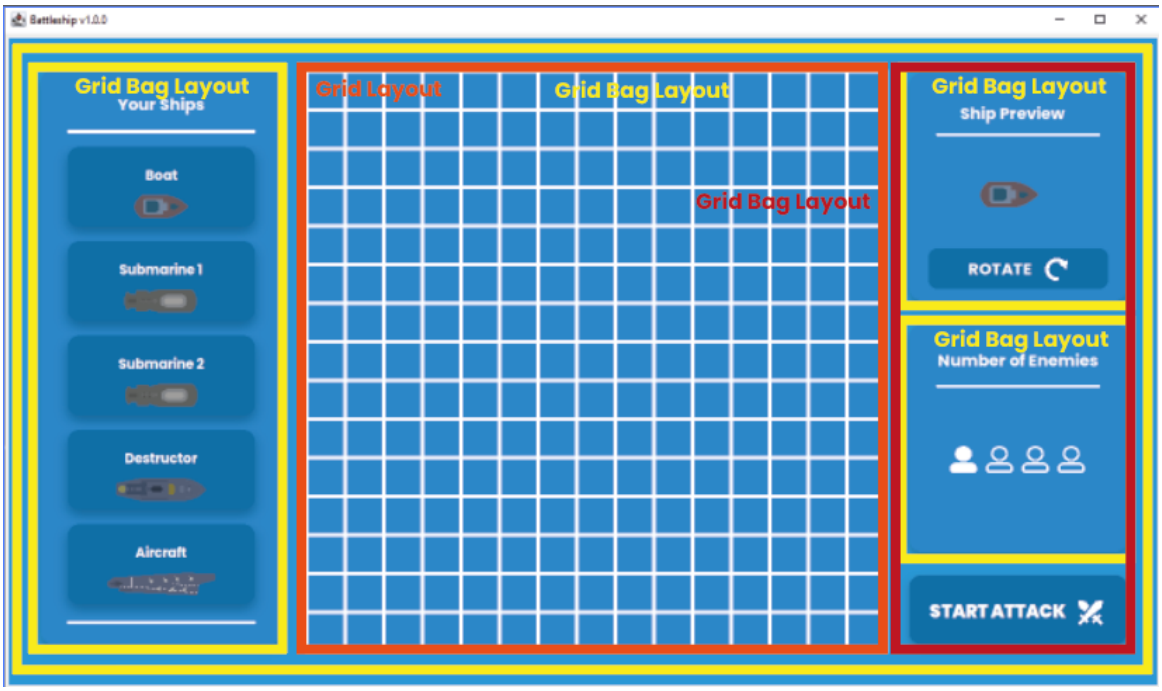
La ventana del menú inicial está formada por un GridBagLayout principal, el cual contendrá otros tres GridBagLayouts (Uno por cada columna):

- En la izquierda tenemos un JPanel GridBagLyout que contendrá separadores para que no esté al lado del borde del fondo, y el panel que contendrá el nombre del jugador.

- En la parte central tenemos otro JPanel GridBagLayout con dos botones que son JImagePanels:
  - Uno para comenzar una nueva partida.
  - Uno para cargar una partida guardada.

- En la columna de la derecha tendremos otro JPanel GridBagLayout donde utilizamos unos separadores para poder posicionar los elementos en la parte derecha de la vista.
  - Utilizamos un JPanel para hacer el panel de los trofeos.
  - Un JImagePanel para el icono de ajustes.
  - Un JImagePanel para el icono de estadísticas.

Vista de la preparación de la partida



La ventana del setup está formada también por un GridBagLayout que contendrá 3 partes:

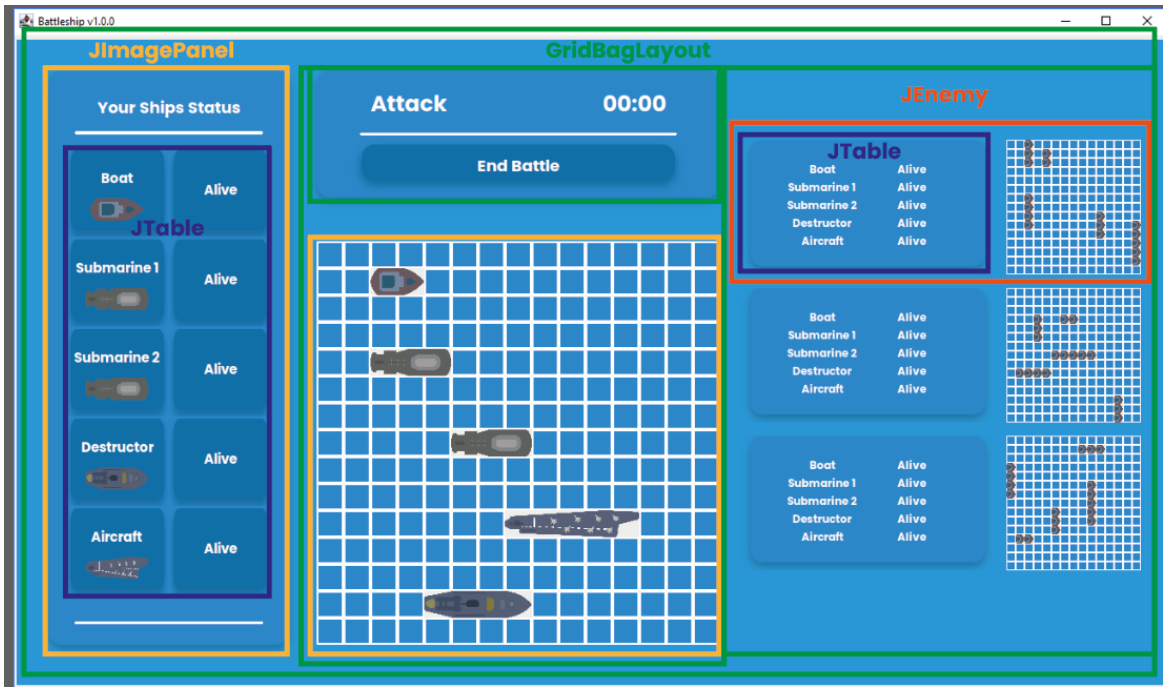
- En la parte de la izquierda tendremos un JImagePanel (para el fondo del panel) con un GridBagLayout que contendrá:
  - Un JLabel con el título del panel.
  - Cinco JPanels que serán los barcos que el usuario podrá posicionar.
- En la parte central de la vista tendremos el tablero donde posicionar los barcos. Es un JPanel con un GridPanel de 15x15 que contendrá en cada casilla:
  - Un JPanel con la coordenada de la casilla en el tablero y la imagen de este.

- En la parte derecha de la vista tendremos un JPanel con GridBagLayout que contendrá:
  - Arriba del todo un JImagePanel con GridBagLayout que contiene:
    - Un JLabel con el título del panel.
    - Un JImagePanel del barco para la preview.
    - Un JPanel para el botón de rotación, formado por:
      - Un JImagePanel (para el fondo).
      - Un JLabel para el texto del botón.
      - Un JImagePanel para el icono de rotación.

  - Debajo tenemos otro JImagePanel con GridBagLayout que contiene:
    - Un JLabel con el título del panel.
    - Cuatro JImagePanels con las imágenes de usuarios para seleccionar el número de enemigos. Estos cuatro iconos estarán dentro un JPanel con GridBagLayout.

  - Por último, tendremos el botón para comenzar el ataque que es un JImagePanel también con un GridBagLayout que contendrá:
    - Un JLabel con el texto del botón.
    - Un JImagePanel con el icono del botón de ataque.

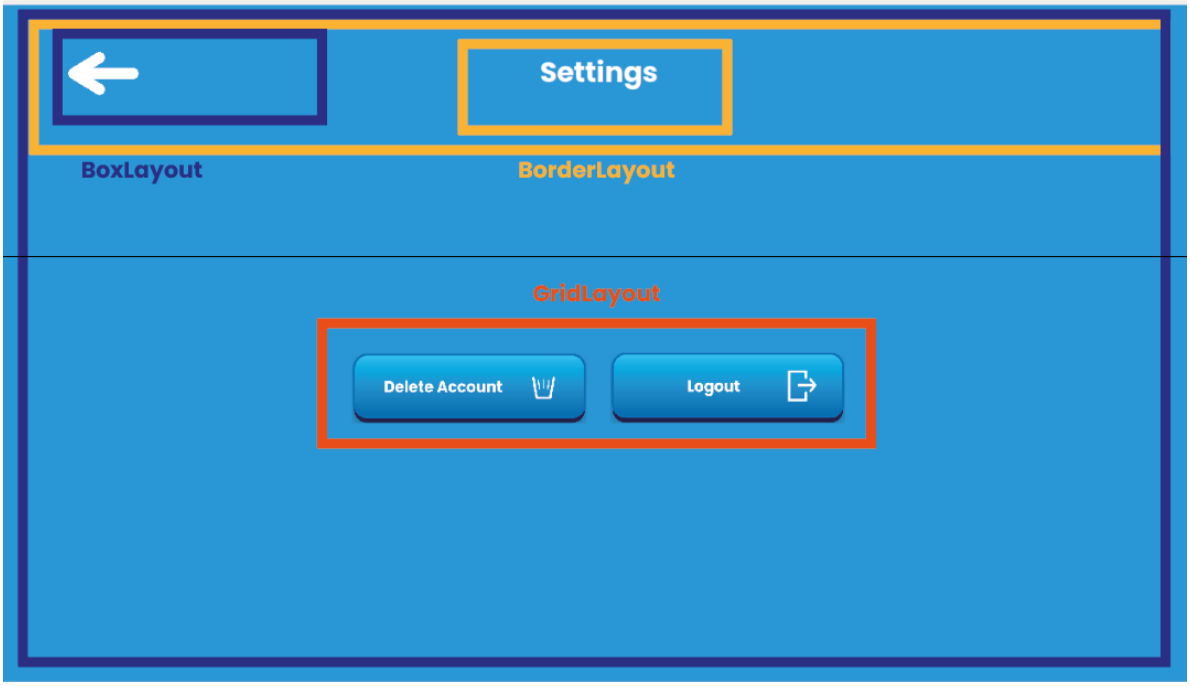
Vista de la fase de juego



En la ventana del juego, tendremos al igual que en las anteriores, un JPanel con un color de fondo con un GridBagLayout que contendrá:

- En la parte izquierda un JImagePanel con el fondo de imagen para el estado de los barcos del jugador, el cual contendrá lo siguiente:
    - Un JLabel en la parte superior del panel, que será el título de este.
    - Una JTable (2x5), la primera columna para mostrar el nombre y la imagen del barco y una segunda columna para mostrar el estado en el que se encuentra el barco (hundido o vivo).
- En la parte central tendremos otro JPanel con un GridBagLayout que contendrá:
    - Un JImagePanel con GridBagLayout para poder mostrar:
        - El estado de ataque (si el jugador puede atacar o debe esperar)
        - El tiempo de duración de la partida
        - Y por último el botón para poder terminar la partida.
    - Un JPanel con un GridLayout (15x15) donde cada celda tendrá un JPanel con las coordenadas de la celda y la imagen que se debe mostrar.
- Por último, en la parte derecha tendremos otro GridBagLayout que contendrá una clase personalizada (JEnemy) la cual está formada por:
    - Una JTable personalizada (2x5) con el nombre del barco y el estado de este.
    - Un GridPanel de 15x15 (tablero enemigo).

Vista del menú de ajustes



En la ventana de ajustes tendremos un JPanel como fondo principal de color azulito que será un BorderLayout. Dentro de este tendremos:

- En la parte superior del BorderLayout (North) tendremos otro BorderLayout que contendrá:
  - Un BoxLayout para el botón de “back” junto a un separador para que no esté al borde de la ventana.
  - Un JLabel BorderLayout con el título de la ventana.

- Mientras que en la parte central (CENTER), tendremos un GridLayout que contendrá:
  - Un JImagePanel para el botón de eliminar cuenta de usuario.
  - Un JImagePanel para volver a la pantalla de login.

Vista de las estadísticas.

En la ventana de ajustes tendremos un JPanel como fondo principal de color azul que será un BorderLayout. Dentro de este tendremos:

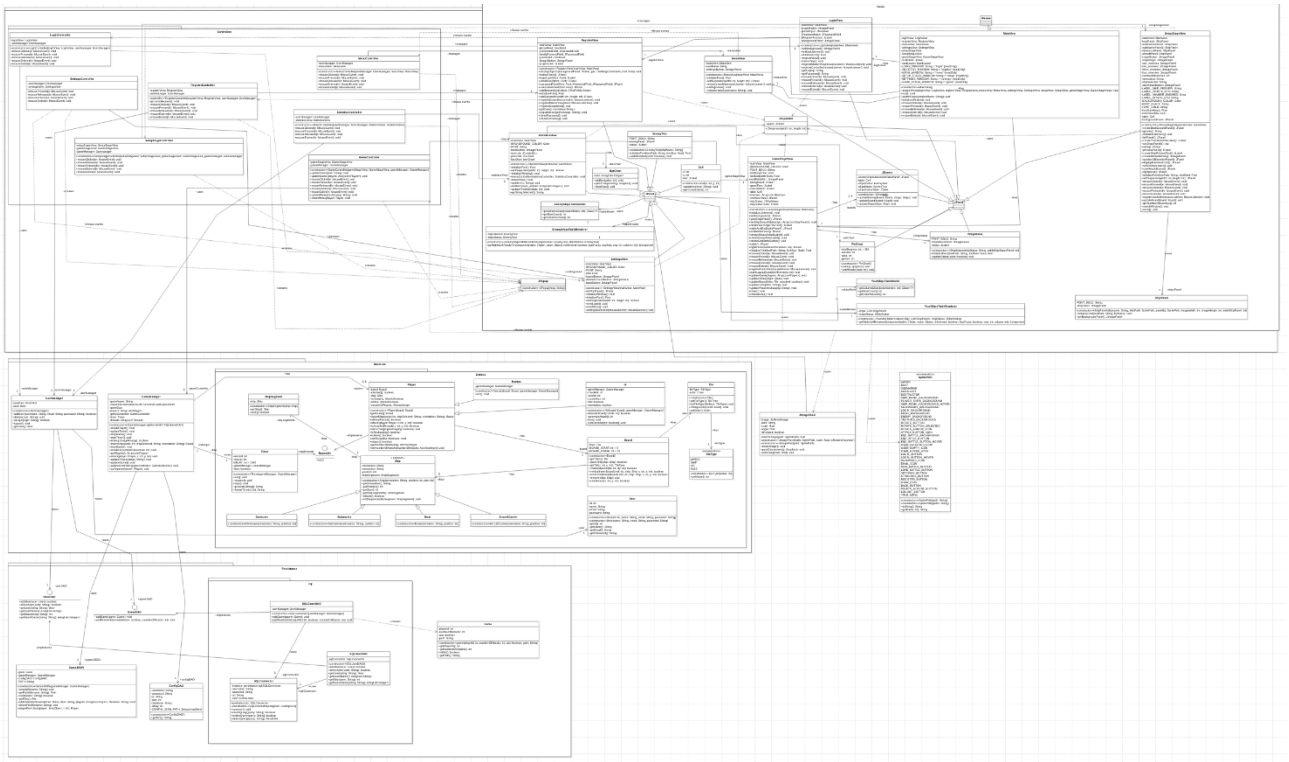
- En la parte superior del BorderLayout (North) tendremos otro BorderLayout que contendrá:
  - Un BoxLayout para el botón de “back” junto a un separador para que no esté al borde de la ventana.
  - Un JLabel BorderLayout con el título de la ventana.

- Mientras que en la parte central (CENTER), tendremos un GridLayout que contendrá:
  - Un JPanel con el texto “Username:”.
  - Un JComboBox para poder seleccionar al usuario del que queramos ver sus estadisticas.
  - Un PieChart que mostrara el porcentaje de victorias.
  - Un BarChart que mostrara los ataques que ha realizado el usuario en los diferentes games.

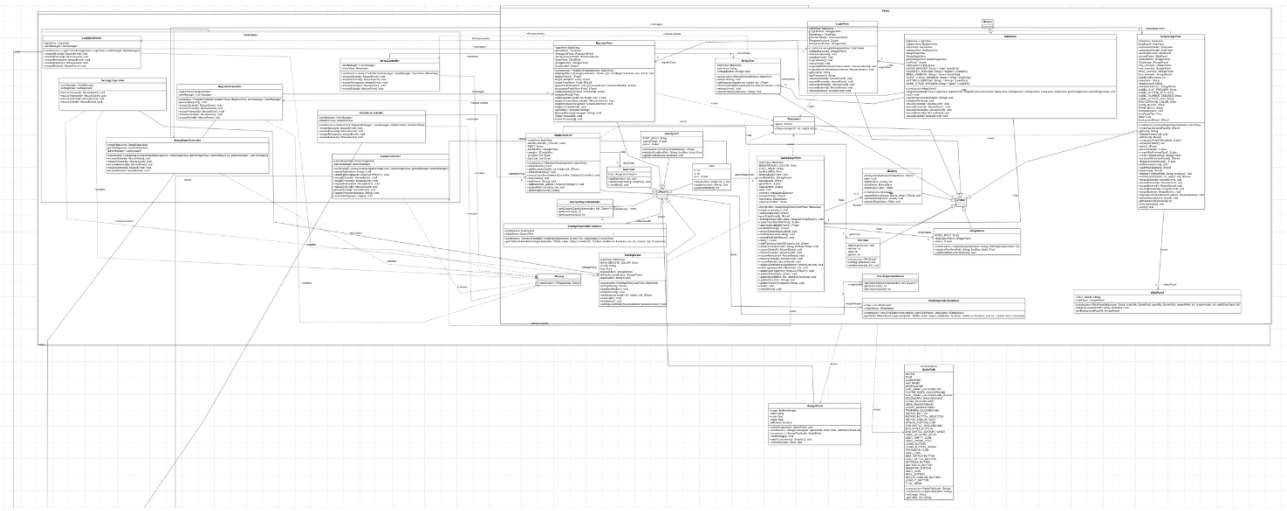
Diagrama de clases

Principalmente nuestro diseño se basa en separar el programa siguiendo una *layered architecture* además de utilizar el patrón de diseño Modelo-Vista-Controlador.

Esto es debido a queremos conseguir un sistema que sigua estos principios: fácil de diseñar y entender, facilidad para mantener el código, facilidad para la detección de errores y facilidad para la reutilización de código. Siguiendo esta arquitectura intentamos aumentar al máximo la abstracción y modularidad del sistema, separando las responsabilidades de cada clase enfocando cada una a un problema en concreto.



Capa de Presentación



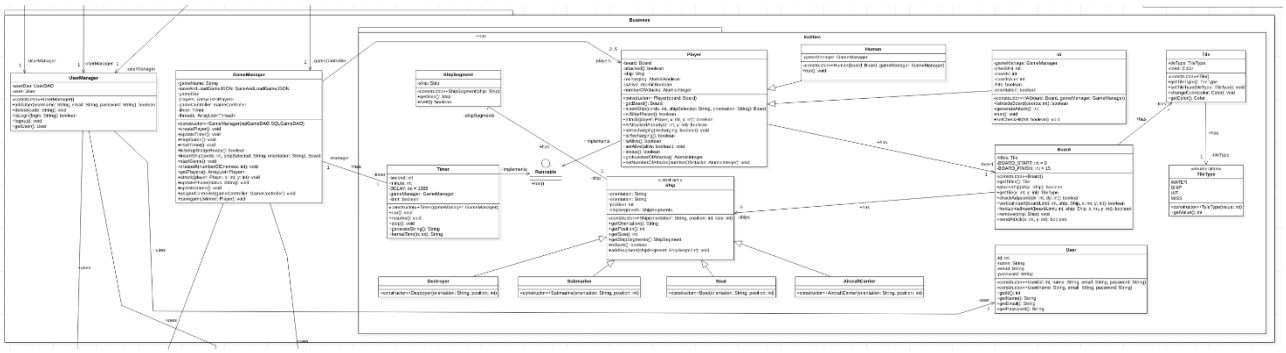
Hemos dividido está capa de dos paquetes grandes bien diferenciados los controlades y las vistas. En primer lugar, comentaremos como hemos enfocados las vistas y seguidamente hablaremos sobre los controladores.

Cuando nos reunimos hablamos sobre como enfocar las vistas, decidimos hacer un sistema de “cartas” donde todas las ventanas se cargaban al iniciar la aplicación y, nosotros nos encargaríamos de llenarlas de contenido según el usuario vaya navegando entre ella. Decidimos crear una clase que se encargaría de almacenar todas las vistas y rotarlas según las interacciones que realizará el usuario con la vista dicha clase es la “MainView”. Además de las vistas también creamos muchas clases auxiliares que nos ayudaban a reutilizar código y/o mejorar la legibilidad de este, todo esto nos ayudaba conseguir un “high cohesion” evitando dar muchas funcionalidades a una sola vista.

Cada vista esta conectada a un controlador de forma que seguimos el patrón de diseño Modelo-Vista-Controlador, cuando el usuario introduce información la vista la envía hacia los controladores donde estos decidirán si ir hacia el modelo para actualizar información y/o actualizarán la vista directamente desde el controlador, por ejemplo, cambiar una ventana después de pulsar un botón, para este tipo de acciones no necesariamente necesitamos ir hacia el modelo.

Por último, hablaremos sobre los controladores, estos son los encargados de comunicar la capa de presentación con la business. Su tarea principal será recoger la información introducida por el usuario en las vistas, comprobar si son correctas y enviarlas hacia los managers, esta era la única funcionalidad de los managers consiguiendo así un “high cohesion” evitando que realizarán muchas acciones diferentes.

Capa de Business



Para mantener el “low coupling” de la aplicación hemos creado dos mánager que se encargarán de realizar todas las acciones que comunicarán la capa de persistencia con la capa de presentación y los modelos. Los dos managers se encargarán de gestionar toda la lógica referente a los usuarios y las partidas, cada uno separado en un mánager respectivamente.

La clase “UserManager” se ocupará de realizar todas las operaciones lógicas relacionadas con el usuario, ya sea comprobar que el usuario exista, haciendo una consulta a la base de datos pasando por la capa de persistencia, como comprobar si los datos introducidos en el formulario de inicio de sesión son correctos. A su vez, se ayudará de la clase User para realizar diferentes acciones, siendo User la responsable de almacenar y gestionar la información del usuario respetando así el “information expert” que nos pide GRASP.

La clase “GameManager” se ocupará de realizar todas las operaciones lógicas relacionadas con la partida, será el encargado de crear la estructura base de la partida y hará uso de los modelos que ha su vez serán los encargados de realizar las operaciones que les involucren directamente a ellos, por ejemplo, cuando un usuario quiera insertar un barco en su tablero será la clase Jugador la que se encargará de comprobar que el barco pueda ser insertado. Al haber implementado este sistema de delegación seguimos respetando el “information expert” que nos pide GRASP, ya que si nos fijamos cada clase será la encargada de manipular y controlar su información.

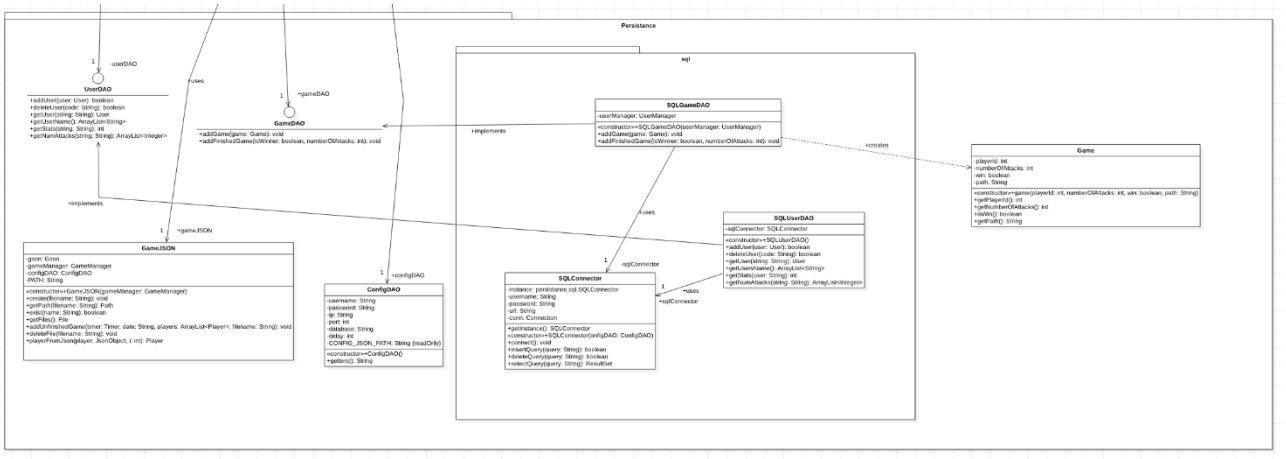
Por otro lado, hablaremos un poco de como esta pensada la lógica del juego, en nuestro caso tenemos 4 jugadores que se almacenarán en el GameManager, cada uno de estos tendrá un tablero donde están colocados sus barcos. Estos barcos también los tiene guardados el jugador en su propia clase, esto es así porque creemos que era más sencillo si nos guardábamos su referencia para comprobar que barcos seguían vivos y cuales estaban hundidos. Cabe destacar que cuando colocamos un barco en el tablero este se divide en varios objetos de tipo “ShipFragment” que se guardan dentro de las casillas del tablero con el fin de facilitarnos destruir los barcos modificando simplemente el tablero. Una vez teníamos todo esto simplemente dejábamos que los jugadores atacaran y íbamos controlado los barcos que les quedaban a los demás jugadores después de cada ataque cuando solo quedará un jugador con vida la partida se acaba.

Para los barcos decidimos utilizar herencia para diferenciar los diferentes barcos, aunque esto no sería necesario si, por ejemplo, hubiéramos añadido un atributo dentro de la clase “Ship” que fuera tipo. Pero creemos que al utilizar herencia mejorábamos la comprensión sobre como funcionaba el sistema.

Para los jugadores también usamos la herencia además de tener una clase “Player” abstracta donde definimos funciones que deberán ser implementadas tanto el “Human” como en “IA”, por ejemplo, la función para atacar que será común para ambas clases. Además, todos los jugadores implementan la clase Runnable que nos permitirá crear threads para gestionar los ataques y los tiempos de recarga de los jugadores.

Por último, comentaré como hemos gestionado el sistema de threads para los jugadores, en el caso del usuario utilizamos una variable para saber si esta recargando, es decir, si ha disparado hace poco y todavía no ha pasado el tiempo de “recarga”. Nuestro thread estará constantemente comprobando si ha pasado el tiempo definido y, cuando sea así cambiará el valor de nuestra variable indicando de que ya ha recargado, avisará al manager que tiene que actualizar la vista para informar al usuario. Por otro lado, tenemos la IA, en el caso de esta hemos simplificado el sistema, no utilizan la variable de recarga, sino que entre ataques detenemos el thread de forma que esperará el tiempo marcado entre ataques. Cada vez que realizan un ataque avisan al manager que tiene que actualizar la vista mostrando al usuario la casilla atacada.

Capa de Persistance



En esta capa encontramos todas las clases relacionadas con la persistencia del sistema, es decir, nos permitirán leer, guardar, crear y modificar información.

Lo más destacable de la clase deberían ser las interfaces donde están definidas todas las funciones que deberían implementar las clases que implementan dicha interfaz. En nuestro caso contamos con un tipo de persistencia en la base de datos para guardar la información de las partidas y los usuarios registrados. Además, también contamos con un fichero JSON donde guardaremos las partidas de forma local de los jugadores. Para cada una de estas operaciones tenemos una interfaz por si en algún momento futuro queremos cambiar la base de datos o el formato del fichero dicho cambio sea sencillo de realizar simplemente añadiendo nuevas clases que implementen estas interfaces.

Para que una clase no se encargue de toda la persistencia del sistema hemos creado diferentes clases. Para los ficheros JSON tenemos la clase “GameJSON” que se encargará de guardar las partidas en un fichero JSON además de leer dichos ficheros cuando sea necesario para cargar las partidas. Por otro lado, para la persistencia en la base de datos tenemos dentro de un paquete llamado SQL, las clases SQLGameDAO y SQLUserDAO donde cada uno como su nombre indica se encargan de gestionar la persistencia de la partida y los usuarios respectivamente.

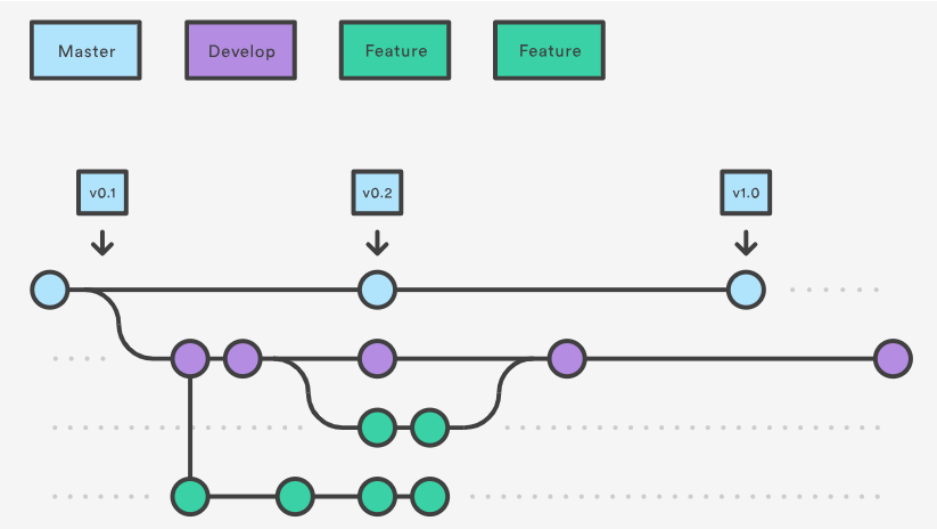
Por último, tenemos la clase ConfigDAO donde tendremos los datos para conectarnos a la base de datos además de configuraciones del propio juego, como el tiempo de recargara entre ataques. Este fichero se leerá una vez iniciada la aplicación y dará la información al SQLConnector que se encargará de iniciar la conexión con la base de datos además pasará la configuración de la partida al GameManager.

Metodologías de desarrollo

Para el desarrollo del proyecto nos hemos apoyado en dos herramientas que ha sida la pieza central para mantener un trabajo ordenado y estructurado y, que nos han ayudado organizarnos mejor como equipo.

Una de estas herramientas es Git un control de versiones que nos ayudará ir trabajado de forma simultanea en diferentes ramas organizándonos mejor como equipo, ya que si estuviéramos trabajado todos sobre un mismo código haría que nos confundiéramos y dificultaría en gran medida el desarrollo del proyecto. Con esta herramienta nos distribuíamos en diferentes grupos para ir implementando todas las funcionalidades del código.

La forma en la que nos distribuíamos era creando un sistema de ramas donde creábamos una rama por cada funcionalidad que teníamos que implementar, una vez acabada dicha funcionalidad esta se incluía en su rama de origen donde se iba manteniendo el código actualizado a la última versión, un ejemplo de como funciona este sistema sería la siguiente imagen:



En la imagen anterior se puede ver claramente el esquema seguido, teníamos una rama de desarrollo que era la raíz de todas las ramas auxiliares que íbamos creando donde añadíamos una nueva funcionalidad. Cuando dicha funcionalidad estaba acabada introducíamos los cambios de esta a la rama de desarrollo. Este proceso se ha seguido hasta acabar el juego, cuando llego este momento introducimos la rama de desarrollo a master que sería una versión estable de nuestro proyecto.

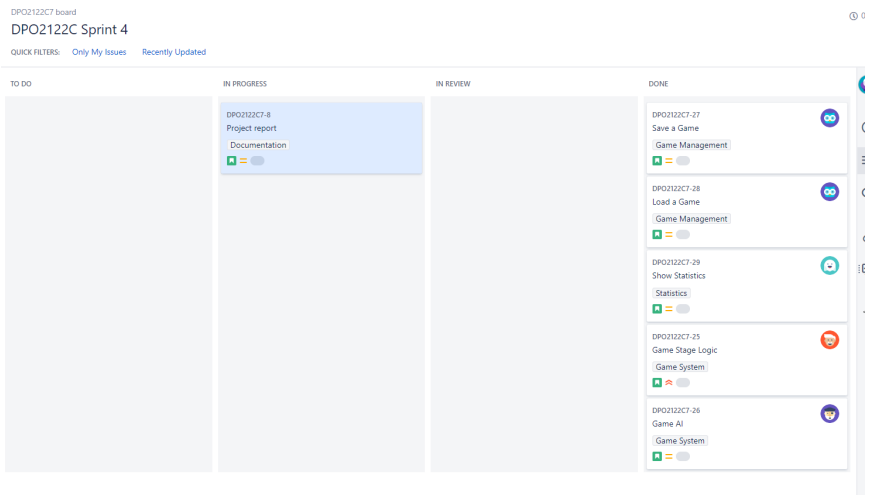
La otra herramienta que hemo usado ha sido Jira la cuál nos ha ayudado a organizarnos y saber en todo momento a que se estaban dedicando nuestros compañeros. Hemos seguido una metodología de desarrollo muy conocida llamada *Agile.* Más concretamente la metodología específica *Scrum.*

El uso de Jira en el proyecto era muy específico dividir el proyecto en 4 semanas, realizando esprints que tenían una duración de dos semanas, cada sprint tenía como objetivo acabar unas *Historias* previamente seleccionadas, donde cada una de estas historias se separaba en pequeñas funcionalidades que había que ir implementado por uno o varios miembros del grupo.

Gracias a Jira este proceso era mucho más fáciles ya que nos permitía crear las historias en “cartas” donde estaban especificados los requerimientos y objetivos de cada historia. Cada una de estas “cartas” podía ser asignada a un miembro del grupo en concreto así sabíamos si alguien se estaba encargado de ese apartado y nos olvidábamos de confundirnos e implementar dos personas la misma funcionalidad.

Por último, un apartado interesante del Jira es que nos permitirá poner las tareas seleccionadas para esa semana en diferentes categorías que son las siguientes:

- To do: La historia ha sido escogida para este sprint y todavía no se ha comenzado a implementar.
- In progress: La historia ha sido asignada a un miembro del grupo y este ha comenzado a implementarla.
- In Review: Cuando la historia ha sido acabada por el miembro del grupo pasa a este apartado donde permanecerá hasta que el resto de los compañeros revisen la funcionalidad implementada y den el visto bueno a dicha historia. En caso de no conseguir esto volverá al estado de In Progress donde tendrá que ser corregida por el miembro que la tuviera asignada.
- Done: Una vez las historias han sido revisadas y aprobadas pasan a este estado que indica que dicha historia esta acabada y funciona correctamente.



Estas han sido las herramientas que hemos utilizado y como las hemos implementado en nuestro desarrollo. Seleccionábamos al finalizar un sprint las historias que íbamos realizar para el siguiente y, si veíamos que algún compañero tenía dificultades mover una historia del estado de “In Progress” nos poníamos en contacto con el para tratar de ayudarlo y acabar el proyecto lo antes posible.

Dedicación

Respecto al tiempo dedicado en el proyecto, hemos procurado ir al día con los sprint de Jira, como bien nos ha guiado en las clases de mentoría nuestro Project Leader, Jordi Malé. A continuación, comentaremos el tiempo estimado para las cuatro etapas que conforman el proyecto, estas son: Comprensión de las especificaciones, análisis y diseño, codificación y documentación.

Una vez disponible el enunciado del proyecto, cada miembro del equipo tomo su tiempo para contemplar una primera hojeada y posteriormente poner sobre la mesa todas las dudas e inquietudes de forma conjunta. Cabe mencionar que, los apartados estaban muy bien detallados y por tanto no tuvimos que dedicar mucho tiempo a esta etapa.

Sin embargo, el análisis y diseño nos llevó mucho más tiempo, ya que tuvimos que pensar cómo estructurar el UML perfectamente debido a las dimensiones del proyecto. El hecho de dedicar un tiempo considerable al diagrama fue un punto a favor a la hora de codificar el código, ya que partíamos de tener una estructura predeterminada. En lo que respecta a la base de datos, es un aspecto a tener en cuenta, puesto que, hasta ahora, no habíamos trabajado con un programa que estuviera conectado a una base de datos.

Referente a la codificación del proyecto, nos ha resultado con gran diferencia la etapa más desafiante del proyecto. Por último, cabe mencionar la documentación la cual no nos ha supuesto mucha carga de trabajo, dado que simplemente constaba de explicar lo que habíamos desarrollado en el proyecto.

Conclusiones

En conclusión, este proyecto nos ha servido para reforzar aspectos teóricos de la asignatura, así como para mejorar las habilidades de trabajar en equipo.

Referente al contenido del proyecto, sin lugar a duda, ha sido un proyecto desafiante, para muchos de nosotros ha sido el proyecto de mayor envergadura con el que nos hemos enfrentado. Para el desarrollo de este, ha sido necesaria la constante búsqueda de información en internet, ya sea para resolver dudas o para buscar la solución al problema.

El proyecto principalmente ha tenido dos grandes bloques, estos son: La arquitectura por capas y la librería gráfica SWING. Entender bien cómo funciona el patrón MVC (Model View Controller) al principio nos resultó algo complicado, sobre todo el entender el propósito de su implementación para posteriormente desarrollarlo fue un proceso largo. No obstante, nos ha servido de gran ayuda de cara a tener una estructura coherente y saber cómo lidiar con los problemas en el apartado de codificación. Respecto a la parte gráfica del juego, ha sido la parte más dificultosa puesto que partíamos de prácticamente de ningún conocimiento a cerca de esta librería, por lo que tuvimos que investigar mucho e ir realizando pruebas sin temor a equivocarnos, sin duda, observar un error y poder corregirlo ha sido la mejor manera de aprender.

En cuanto al trabajo en equipo, ha sido crucial tener una comunicación fluida entre todos los miembros del equipo, aunque cada uno se encargará de una parte del proyecto, todos estábamos informados en qué punto se encontraba el compañero para cualquier cosa que necesitará. Este proyecto nos ha aportado grandes beneficios que nos encontraremos en el ámbito laboral, como, por ejemplo: Mantener reuniones semanales con un Project leader para hablar sobre el estado del proyecto, escuchar y debatir de manera respetuosa las ideas de los compañeros y sobre todo saber complementarnos los unos con los otros.

Bibliografía

Stackoverflow. (s. f.). Stack Overflow – Where Developers Learn, Share & Build Careers. https://stackoverflow.com

JavaTM Plataform, Standard Edition 18 API Specification. API Java. https://docs.oracle.com/en/java/javase/18/docs/api/index.html