Laboratorio Nº1
"Batalla Naval"

Nombres: Renzo Dávila, Facundo Lella

Universidad Nacional de Cuyo

Resumen

El desarrollo y modelado del famoso juego Batalla Naval, comentando aquellos problemas aparecidos, soluciones encontradas, decisiones tomadas y la conclusión de este camino recorrido.

1. Introducción

Nuestro objetivo es crear un juego mesa utilizando las buenas prácticas de la programación orientada a objetos en el lenguaje Java 17. Para el desarrollo del juego utilizaremos las reglas provistas por el problema, que serán las siguientes:

1. Modelar cómo sería su aproximación para brindar una solución que permita hacer funcionar el sistema de batalla naval.

2. Simular una partida que se pueda jugar entre dos participantes.

3. Imprimir el estado actual del mapa de cada participante,en qué estado se encuentra, lista de barcos hundidos, lista de barcos a salvo todavía.

4. Imprimir un menú para que el usuario pueda ver la información del juego antes de poder jugar.

5. Implementar una interfaz en el diagrama de clases.

Como primer paso haremos una abstracción del problema el cual nos dará un comienzo. A través de esto podremos empezar a modelar nuestro diagrama UML, que tendrá clases, atributos, métodos y sus respectivas relaciones. Este paso es importante ya que nos define qué objetos y problemas deberemos solucionar. Mientras haya una mejor abstracción del problema, mejor será nuestro código e implementación.

2. Objetivos

. Correcto Modelado
. Buenas Prácticas de POO

. Cumplir las consignas propuestas

3. Proceso de experimentación

Para nuestra solución iniciamos leyendo las reglas acerca del juego “Batalla Naval”. El usuario es capaz de elegir el tamaño del tablero, cantidad de barcos, cantidad de disparos a realizar y cantidad de islas. A partir de esto realizamos simplificaciones que nos parecieron óptimas para su jugabilidad y nivel de código, limitando las opciones del usuario a 3:

Tamaño 10x10 - 5 barcos - 70 disparos - 4 Islas

Tamaño 7x7 - 4 barcos - 32 disparos - 3 Islas

Tamaño 5x5 - 2 barcos - 18 disparos - 2 Islas

Luego de haber aclarado las condiciones de juego comenzamos a dar los primeros pasos en la estructuración del diagrama de clases. En nuestro primer modelado identificamos las siguientes clases: Barco, Tablero, Jugador, HUB. Estas clases que fueron las primeras que dieron vida al proyecto, no resultaron tanto como esperábamos, por eso hablaremos un poco de qué tratan, de cómo fueron evolucionando y los problemas encontrados.

Barco: Este objeto es una clase abstracta que tiene 5 subclases que son los diferentes tipos de barco: Lancha, Crucero, Portaaviones, Buque, Submarino. Los atributos de Barco son tamaño e identificador (identificador: Letra que lo diferencia en el Tablero). Decidimos usar Barco como clase abstracta ya que no va a existir un objeto Barco, si no diferentes tipos y a futuro poder agregar más.

Tablero: Esta es una clase abstracta que tiene dos subclases TableroA (Atacante) y TableroD (Defensor). Sus atributos son una matriz (Board) y el tamaño, y métodos como mostrarTablero. Esto se hizo ya que había una necesidad de tener dos tableros distintos con distintos comportamientos y uno tendrá la posiciones de los barcos y el otro guardará los disparos hechos.

HUB: HUB es el “lobby” o “menú” del juego, te da opciones para iniciar el juego, leer reglas o salir. Algunos de sus métodos turnos, leerReglas e iniciarJuego

Jugador: Como atributos tiene un nombre, cantidad de barcos, lista de barcos restantes, lista de barcos perdidos, además de un TableroA y TableroD. Sus métodos eran mostrarBarcos, eliminarBarcos e inicializarBarcos.

Posición: con sus atributos fila y columna que nos permiten almacenar una posición en el tablero.

Luego de hacer este primer modelado e implementar código detectamos ciertos problemas que detallaremos a continuación:

- **Como saber a qué barco le estamos dando**. Al crear una matriz de caracteres colocamos identificadores de los barcos en esta. Por ejemplo un Crucero tiene identificador C entonces en la posición (4,3) hay una C, además, si teníamos otro objeto Crucero y lo ponemos en la posición (9,4) con el carácter C y el jugador contrario acierta en su ataque a esa posición, no sabremos cual de los dos barcos fue afectado.

- **Implementar una Interfaz con un propósito óptimo.** No conseguimos ver un comportamiento común entre algunas clases, todo lo posible también llevaba consigo atributos, los cuales no son compatibles con interfaces, por lo que no nos decantamos en donde podríamos crear una interfaz.

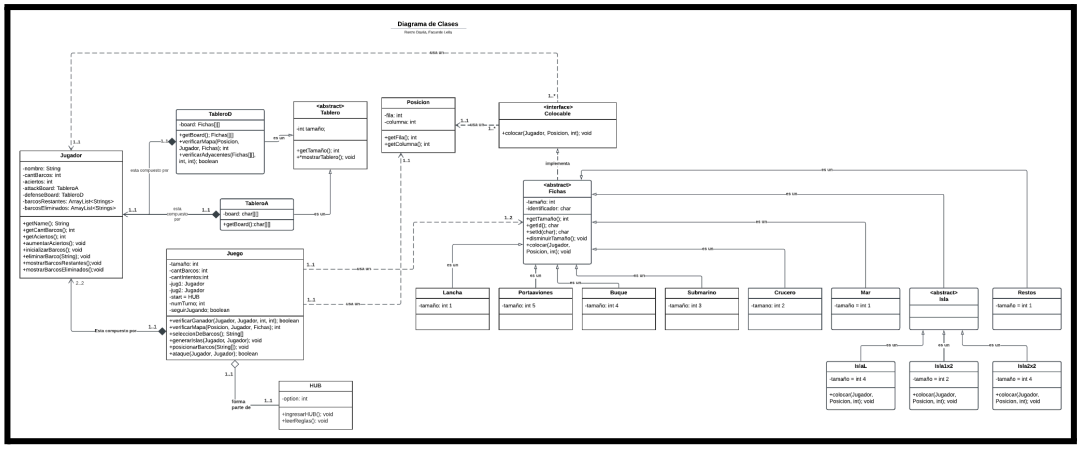
- **Implementar un método Polimórfico**

Luego de discutir y pensar hemos llegado a las siguientes soluciones lo cual nos lleva a modificaciones en nuestro modelado. Nuestra clase **Barco** pasará a ser la clase **Fichas** que tiene los mismos atributos que barco y sus subclases serán las mismas lancha, crucero, etc y se agregara la subclase Mar, Isla (con sus tipos de islas) y Restos (todas estas clases fueron creadas con el fin de poder colocarlas en el TableroD, el cual es de Fichas). Fichas sigue siendo una clase abstracta. Ahora el TableroD será una matriz de Fichas mientras que TableroA seguirá siendo de carácteres. Para resolver la implementación de la interfaz creamos Colocable que posee el método Colocar(), y Fichas lo va a implementar por lo cual todas sus subclases tendrán que implementarlo. Esto ,ya que, cada Ficha tiene su manera de ser colocada, un Submarino ocupa 3 y un Portaaviones 5, podríamos unificar y que sea un método de la clases Fichas pero luego deberíamos modificar ese método si quisiéramos incluir nuevas Fichas que se coloquen de una manera especial, por esa razón decidimos usar Colocable como interfaz.

Además decidimos que Hub ya no será el main, el nuevo main será Juego, ya que tiene más sentido que esta clase tenga la responsabilidad de llevar a cabo la partida. En resumen hay 5 grandes clases con sus respectivas subclases ya mencionadas: Fichas, Jugador, Juego , Tablero y Hub.

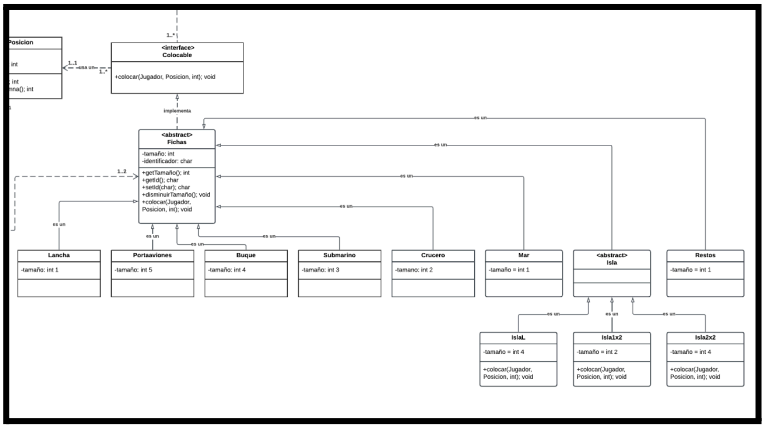
4. Diagrama final

Luego de todo el proceso del primer planteo y la aparición de problemas, se llegó a un resultado final, donde este fue el Diagrama UML resultante:

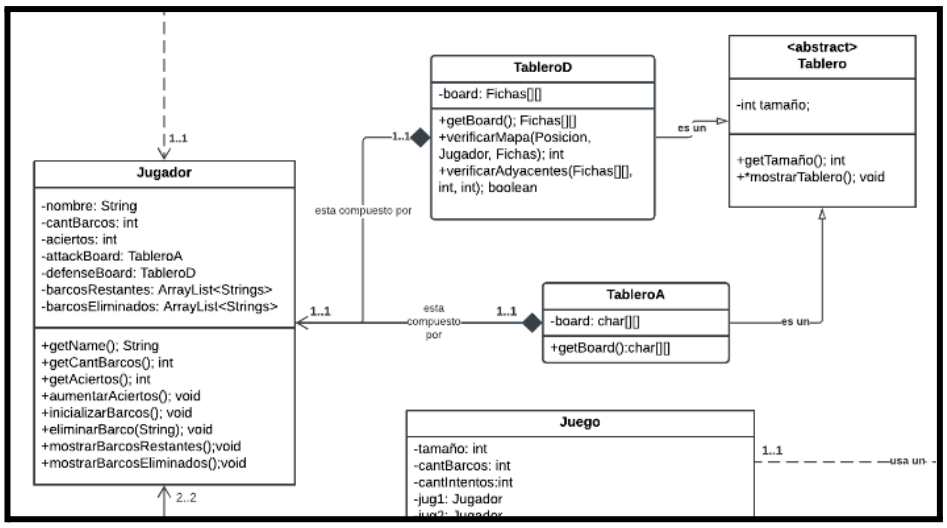


Ahora pasaremos a explicarlo por partes:

Empezaremos hablando sobre la parte de las Fichas, en donde tenemos todas aquellas posibilidades de objetos colocables en el Tablero:

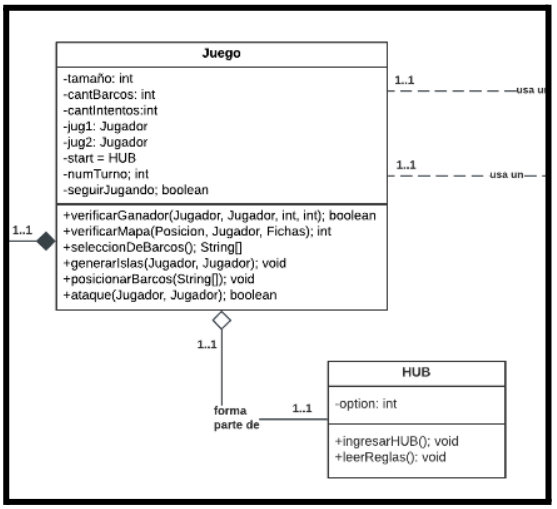


Podemos ver como la clase Fichas es la principal en esta sección, con sus atributos de tamaño e identificador , implementando a la interfaz Colocable con su método de colocar que se implementa en esta misma clase, con el fin de que cada una de sus subclases la sobreescriban o sobrecarguen y puedan colocarse de sus diferentes maneras. Y finalmente todas sus subclases: Lancha, Portaaviones, Buque, Submarino y Crucero son simplemente tipos de barcos que van a variar en tamaño. Las clases de Mar y Restos, serán para completar el tablero y el reemplazo de cuando se destruya una parte de un barco, respectivamente. La clase Isla servirá para un impedimento a la colocación de barcos, nos hubiera gustado añadirle una función en especial, como la implementación de tesoros, o el desbloquear toda la isla al dispararle, pero por un tema de tiempo y complejidad de código, decidimos dejarlas como una isla normal. Estas subclases de islas, sobrecargan el método de colocar de la superclase Fichas, ya que no es necesario saber la dirección hacia la que se colocará, únicamente selecciona una posición, y la coloca siempre de la misma manera.



En esta sección la clase principal es el Jugador, con sus atributos de Nombre, cant Barcos que es el número de barcos que le quedan en la partida, un número de aciertos con el fin de compararlos con el otro jugador, obviamente su tablero atacante y defensor (el primero donde recibe los disparos, y el segundo donde se encuentra la distribución de los diferentes objetos, y finalmente dos listas donde se guardaran los barcos restantes, y los barcos que haya perdido.

Además se muestra la clase Abstracta de Tablero, de la cual heredan TableroA y TableroD, donde se puede ver la diferencia en el tipo de tablero, y además en TableroD vemos dos metodos: verificarMapa y verificarAdyacente, estos métodos nos sirven para controlar si una ficha es aplicable en una posición del tablero.



Finalmente la clase principal de nuestro modelado, donde se lleva a cabo el correcto funcionamiento del juego, con sus atributos de tamaño, cantBarcos y cantIntentos, que son elegidos por el usuario, sus respectivos jugadores 1 y 2, el atributo start que nos permite ingresar al HUB al comienzo de la ejecución (en donde se puede dar una breve explicación de cómo funciona el juego), numTurno que nos permite conocer cuántos disparos se han realizado y finalmente seguirJugando que nos dice cuando se concretó la partida y es hora de decidir un ganador. Entre sus métodos tenemos verificarGanador, que justamente decide quién ganó la batalla. Además la selección y el posicionamiento de los barcos, que como su nombre indica, en la primera se le permite al usuario escoger sus barcos favoritos, y el segundo posicionar estos en el campo de batalla.Luego tenemos el método ataque, el cual lleva al jugador a realizar su disparo. Finalmente la generación de islas, la cual con la ayuda del módulo Random de Java, crea una cierta cantidad de islas aleatorias y las posiciona en el tablero.

5. Implementación

En esta sección vamos a entrar en profundidad en, nuestro código, de cómo decidimos implementar nuestro modelo en un lenguaje de programación orientado a objetos como lo es Java

Antes de empezar vamos a comentar un poco cómo nos organizamos entre nosotros a la hora de trabajar. Nunca hubo una asignación bruta de tareas, en donde cada uno trabajara completamente por separado, si no que era un permanente intercambio de ideas y posibles soluciones al trayecto de desarrollo que enfrentabamos.

Algo que se repite mucho en nuestra implementación, es pedirle opciones al usuario, por lo que la estructura “switch” nos acompañó en todo nuestro recorrido, esta fue la forma de implementar esta toma de decisiones normalmente:

```
do{
    try{
        option = scanner.nextInt();
    }catch(InputMismatchException e){
        scanner.next();
    }

    switch (option) {
        case 1:
        System.out.println("Mi tablero");
            jActual.deffenseBoard.mostrarTablero();
            System.out.println("Barcos restantes: ");
            jActual.mostrarBarcosRestantes();
            System.out.println("Barcos perdidos: ");
            jActual.mostrarBarcosEliminados();
            System.out.println("Atacaras en 5 segundos ");
            Thread.sleep(5000);
            break;

        case 2:
            break;
        default:
            System.out.println("Opcion invalida");
            break;
    }
} while (option > 2 || option < 1);
```

Simplemente, luego de mostrar las opciones por pantalla, realizamos una estructura do-while con el fin de repetir el proceso hasta que el usuario ingrese una opción válida. Dentro de esta estructura nos topamos con una estructura del tipo try-catch, para manejar las anomalías del ingreso erróneo de datos (en este caso pide un entero, si se ingresa un carácter o cadena, el código se rompería). Finalmente la estructura switch para decidir qué opción fue elegida, en este caso únicamente hay 2 opciones, y un default con el fin de arrojar un mensaje en caso que la opción ingresada sea errónea.

Ahora veremos la estructura de dos de nuestras fichas:

```
public class Submarino extends Fichas{

    public Submarino(){
        super(3, 'S');
    }

    public void colocar(Jugador j, Posicion position, int option){

        int fila = position.getFila();
        int columna = position.getColumna();

        switch (option){
            case 1:
                for (int i = fila; i >= fila-2;i--){
                    j.deffenseBoard.getBoard()[i][columna] = this;
                }
                break;
            case 2:
                for (int i = fila; i <= fila+2;i++){
                    j.deffenseBoard.getBoard()[i][columna] = this;
                }
                break;
            case 3:
                for (int i = columna; i <= columna+2;i++){
                    j.deffenseBoard.getBoard()[fila][i] = this;
                }
                break;
            case 4:
                for (int i = columna; i >= columna-2;i--){
                    j.deffenseBoard.getBoard()[fila][i] = this;
                }
                break;
        }
```

```
public class Isla2x2 extends Isla{

    public Isla2x2(){
        super(4);
    }

    public void colocar(Jugador j, Posicion position){

        int fila = position.getFila(), columna = position.getColumna();

        for (int i=0;i<=1;i++){
            for (int n=0;n<=1;n++){
                j.deffenseBoard.getBoard()[fila+i][columna+n] = this;
            }

        }

    }
}
```

En estos fragmentos de código, podemos ver la implementación de la clase Submarino y de la Isla 2x2, sus clases “hermanas” por decirlo de alguna manera, tienen la misma forma de diseño, por lo que con analizar estas dos, bastará. Como se puede ver, Submarino es subclase de Fichas, y Isla 2x2 lo es de Islas que a su vez también es subclase de Fichas, poseen un constructor con parámetros ya establecidos, además del método colocar, que como mencionamos anteriormente varía entre nuestras fichas. Además se puede apreciar que nuestros barcos poseen un parámetro más que las islas, ¿esto a que se debe?, bueno, esto es así ya que a la hora de colocar nuestro barco, nosotros damos las opciones de colocarlo hacia arriba, abajo, izquierda o derecha, por lo que debemos ingresar como parámetro la opción dada por el usuario para posicionar el barco correctamente, mientras que en la Isla, nosotros definimos su posición, por lo que no necesitamos de información brindada por el usuario para su colocación.

Ahora veremos como es el funcionamiento del ataque entre jugadores:

```
Fichas fichaAlcanzada = jEnemigo.deffenseBoard.getBoard()[fila][columna];


if (fichaAlcanzada.getId() == 'M'){
    jEnemigo.attackBoard.getBoard()[fila][columna] = 'O';
    fichaAlcanzada.setId('O');
    System.out.println("¡Agua!, has fallado :(");
}else if(fichaAlcanzada.getId() == 'I'){
    jEnemigo.attackBoard.getBoard()[fila][columna] = 'I';
    fichaAlcanzada.setId('O');
    System.out.println("¡Fallaste!, diste en una isla");
    //Posible implementacion de alguna mecanica con las islas
}else{

    if (fichaAlcanzada.getTamanio() > 1){
        fichaAlcanzada.disminuirTamanio();
        jEnemigo.deffenseBoard.getBoard()[fila][columna] = new Restos();
        jEnemigo.attackBoard.getBoard()[fila][columna] = 'X';
        System.out.println("¡Diste en un objetivo!");
    }else{
```

```
}else{
    jEnemigo.attackBoard.getBoard()[fila][columna] = 'X';
    switch (fichaAlcanzada.getId()) {
        case 'L':
            jEnemigo.eliminarBarco("Lancha");
            break;
        case 'P':
            jEnemigo.eliminarBarco("Portaaviones");
            break;
        case 'B':
            jEnemigo.eliminarBarco("Buque");
            break;
        case 'S':
            jEnemigo.eliminarBarco("Submarino");
            break;
        case 'C':
            jEnemigo.eliminarBarco("Crucero");
            break;
    }

    fichaAlcanzada.setId('X');
    jEnemigo.disminuirCantBarcos();
    System.out.println("¡Hundiste una embarcacion!");
```

El primer paso es seleccionar la casilla donde se realiza el ataque, para luego verificar que tipo de Ficha se encuentra en esa posición. Si esta ficha es ‘Mar’ simplemente se marca con una ‘O’ y se notifica que se ha fallado el disparo, y continua con el siguiente turno. Si fuera una ficha del tipo ‘Isla’ sería algo parecido, pero se marcaría una ‘I’, además se notificó con un comentario que se podría hacer una mejora en las mecánicas de las islas, pero ya notificamos el porqué de su cancelación. Luego, si la ficha alcanzada no se trata de Mar ni de Isla, esto quiere decir que se le dio a un Barco, para esto hay dos opciones, que simplemente se le haya dado a un trozo del barco, o que se haya hundido el barco, para diferenciar esto analizaremos el tamaño de la ficha alcanzada, si este es mayor a 1, a el barco aún le quedan trozos por disparar, y lo único que debemos hacer es restarle uno al tamaño y posicionar una Ficha ‘Restos’ en este espacio, pero si su tamaño es 1, quiere decir que se destruyó el barco, en este último caso analizaremos su id con el fin de eliminarlo de los barcos del Jugador afectado.

Finalmente, mostraremos la generación de islas:

```
for (int j = 0;j<=1;j++){

    for (int i = 1;i<=cantIslas;i++){

        int option = rnd.nextInt(3);

        switch (option) {
            case 0:
                Isla1x2 isla1 = new Isla1x2();
                fila = rnd.nextInt(tamanio-1);
                columna = rnd.nextInt(tamanio-2);
                position = new Posicion(fila, columna);
                isla1.colocar(jActual, position);
                break;
            case 1:
                Isla2x2 isla2 = new Isla2x2();
                fila = rnd.nextInt(tamanio-2);
                columna = rnd.nextInt(tamanio-2);
                position = new Posicion(fila, columna);
                isla2.colocar(jActual, position);
                break;
            case 2:
                IslaL isla3 = new IslaL();
                fila = rnd.nextInt(tamanio-3);
                columna = rnd.nextInt(tamanio-2);
                position = new Posicion(fila, columna);
                isla3.colocar(jActual, position);
                break;
        }
    }

    jActual = j2;
```

a

Este fragmento inicia con dos estructuras del tipo for anidadas, la primera con el fin de dar dos vueltas, una por jugador, y la segunda para generar la cantidad de islas designada por los jugadores. Luego se puede ver que se genera un número aleatorio entre 0 y 2, para luego según lo que toco, se crea un tipo de isla, para después generarse otro par de números aleatorios en el rango de los tableros, con el fin de colocarlas en un lugar aleatorio del mapa. En este diseño se aprecia una aleatoriedad en ambos jugadores, es decir, no van tener las mismas islas, ni las mismas posiciones de estas, esto puede perjudicar a algún jugador, pero sentimos que es algo que hace al juego más emocionante.

Esta es la implementación que vamos a mostrar en este informe, pero igualmente, si se quiere entrar en mayor detalle, puede visitar el repositorio de GitHub para analizar con mayor profundidad el código: **Repositorio**

6. Paso a paso de la ejecución

En esta sección mostraremos cómo funciona la ejecución de nuestro código por consola, como primer paso será descargar el archivo BatallaNaval.jar. Luego ingresamos a la consola de nuestro dispositivo y entramos a la dirección donde se encuentre el archivo descargado, posiblemente sea en “Descargas” o “Downloads”.

Cuando nos encontremos en la carpeta donde se encuentra el archivo, ejecutaremos la siguiente línea de comando:

java -jar BatallaNaval.jar

Nos debería aparecer este mensaje:

```
PS C:\Users\renzo\Downloads> java -jar BatallaNaval.jar
Â¡Bienvenidos a Batalla Naval!
Seleccione una opcion:
1. Empezar a jugar
2. Leer reglas
3. Salir del juego
```

En este instante habría salido perfecto y ya empezamos a jugar, como se puede ver, nos muestra un menú, en donde nosotros deberemos ingresar la opción que deseemos, si no

tienen conocimiento sobre el juego Batalla Naval recomendamos que antes de jugar lea las reglas, para conseguir un buen desarrollo del juego.

Para mostrar el funcionamiento directamente pasaremos a jugar.

```
Inicia el juego
Ingrese el nombre del jugador 1:
Renzo
Ingrese el nombre del jugador 2:
Facu
Hola jugador Renzo y Facu
Opciones de tablero:
1. 12x12 - 7 barcos - 5 isla - 100 intentos
2. 10x10 - 5 barcos - 4 islas - 70 intentos
3. 7x7 - 3 barcos - 3 islas - 32 intentos
```

Al iniciar el juego nos pedirá nuestros nombres con el fin de tener una sesión más personalizada, y además la configuración de juego que deseemos jugar, nosotros creamos 3 configuraciones, pero se pueden crear más tranquilamente. En nuestro caso seleccionaremos la opción 3.

```
----------
Â¡Momento de ingresar los barcos a jugar!
-----
Ingrese el barco numero 1 que se va a jugar
1.Lancha (Tamanio 1)
2.Crucero (Tamanio 2)
3.Submarino (Tamanio 3)
4.Buque (Tamanio 4)
5.Portaaviones (Tamanio 5)
4
Ingrese el barco numero 2 que se va a jugar
1.Lancha (Tamanio 1)
2.Crucero (Tamanio 2)
3.Submarino (Tamanio 3)
4.Buque (Tamanio 4)
5.Portaaviones (Tamanio 5)
```

Luego de la configuración, saltará directamente a la selección de barcos, aquí ingresamos la opción que corresponda al barco decidido por los jugadores, como barco 1, nosotros seleccionamos un buque. En la configuración elegida, tenemos 3 barcos por lo que elegiremos un buque, un crucero y un portaaviones.

```
Turno de Renzo
------------------
   0  1  2  3  4  5  6
0 [ M  M  M  I  I  M  M ]
1 [ M  M  I  I  M  M  M ]
2 [ M  M  I  I  M  M  M ]
3 [ M  M  M  M  M  M  M ]
4 [ M  M  I  I  M  M  M ]
5 [ M  M  I  I  M  M  M ]
6 [ M  M  M  M  M  M  M ]
-----
Ingrese la fila donde desea colocar: Buque
3
-----
Ingrese la columna donde desea colocar: Buque
1
Opciones para colacar su barco:
1. Hacia arriba
2. Hacia abajo
3. Hacia la derecha
4. Hacia la izquierda
3
-----
El barco se coloco correctamente
-----
```

```
------------------
   0  1  2  3  4  5  6
0 [ M  M  M  I  I  M  M ]
1 [ M  M  I  I  M  M  M ]
2 [ M  M  I  I  M  M  M ]
3 [ M  B  B  B  B  M  M ]
4 [ M  M  I  I  M  M  M ]
5 [ M  M  I  I  M  M  M ]
6 [ M  M  M  M  M  M  M ]
-----
```

Aquí se puede ver el menú de colocación de barcos, cada jugador irá posicionando el barco uno por uno en el tablero, en este caso coloque el buque en la posición (3,1), siendo 3 la fila, y 1 la columna. Además lo coloque con dirección hacia la derecha, por lo que el buque quedó colocado desde (3,1) hasta (3,4) inclusive como se puede ver en la foto de la derecha. Este proceso se repetirá con todos los barcos seleccionados. ¡Recuerda que los barcos no pueden ponerse pegados unos a los otros por pedido del ejercicio!.

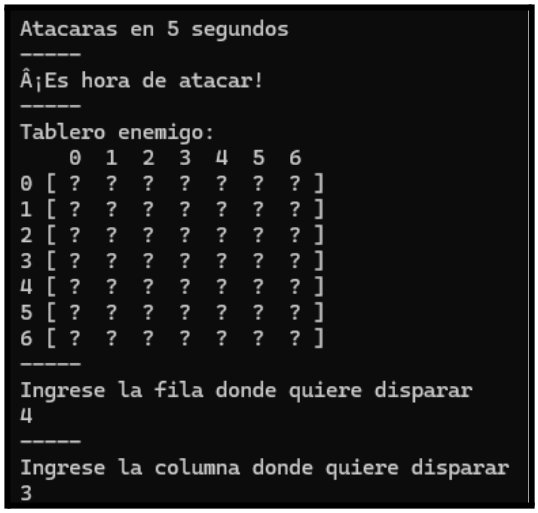
```
Tablero resultante:
   0  1  2  3  4  5  6
0 [ M  C  M  I  I  M  M ]
1 [ M  C  I  I  M  M  M ]
2 [ M  M  I  I  M  M  P ]
3 [ M  B  B  B  B  M  P ]
4 [ M  M  I  I  M  M  P ]
5 [ M  M  I  I  M  M  P ]
6 [ M  M  M  M  M  M  P ]
Presione Enter para colocar barcos del jugador 2
```

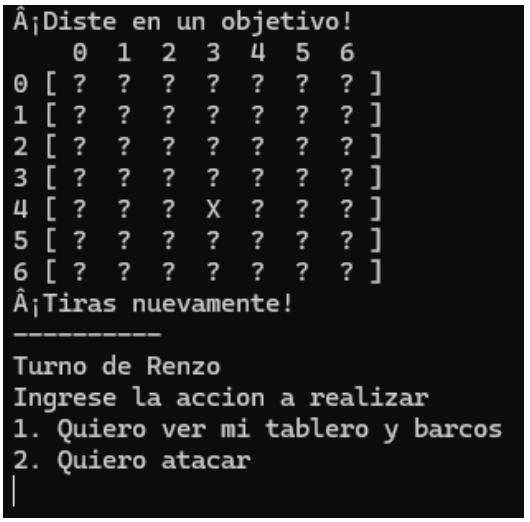
Luego de colocar todos los barcos se nos mostrará el tablero resultante, para seguir con el posicionamiento del jugador dos, presiona enter, trata de desplazarte hacia abajo en la terminal, con el fin de tapar tu tablero, para que a la hora de pasar el dispositivo al jugador dos, no pueda verlo.

```
Â¡Comienza el juego!
----------
Turno de Renzo
Ingrese la accion a realizar
1. Quiero ver mi tablero y barcos
2. Quiero atacar
```

Luego de posicionar los barcos con ambos jugadores, se mostrará el siguiente Menú, deberá ingresar la opción que quiera, al seleccionar la número 1, se mostrará su tablero y dos listas, con sus barcos restantes, y sus barcos eliminados, luego de esto en 5 segundos le tocará

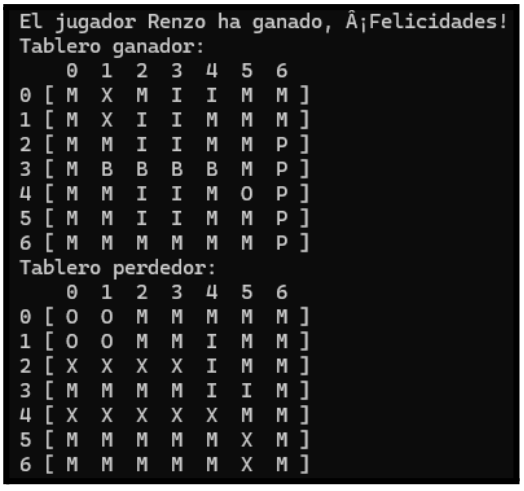
atacar.





En la foto de la izquierda se muestra el menú, de ataque, en donde se ve el tablero enemigo, ahora debemos seleccionar una posición, en mi caso seleccione la posición (4,3) y acerté el disparo, en el caso de acertar tendrás otro turno, si fallas pasara al otro jugador.

Esto se repetirá hasta que se acaben los disparos, en el caso de la configuración elegida 32 por jugador, 64 en total, o que se hundan todos los barcos de un jugador.



Finalmente, al realizar los disparos correspondientes, cuando se hayan hundido todos los barcos de un jugador, habrá un ganador. Para terminar se mostrarán ambos tableros.

7. Conclusiones

El desarrollo del juego **Batalla Naval** nos permitió enfrentar diversos desafíos técnicos y lógicos, desde la estructura y organización del tablero hasta la interacción en ataques, posicionamiento y selección de opciones entre jugadores. Todo esto a través de la implementación de clases que gestionan las fichas, barcos, y el tablero de juego, hemos logrado simular con una buena precisión a nuestro parecer la mecánica de la batalla naval.

En conclusión, el trabajo ha llegado a las expectativas de nuestros inicios cumpliendo con todas las normativas propuestas por el ejercicio y resolviendo con el uso de la POB.