Proyecto #2 – Batalla naval

Programación Orientada a Objetos

Ingeniería en Computación

Integrantes:

- Alexander Artavia Quesada

- Armando Castro RUIZ

- Joan Sánchez Chinchilla

Profesora: Samanta Ramijan

FECHA DE ENTREGA: martes 19/10/2019

**Tabla de contenidos**

1. Descripción del problema…………..……............................3

2. Diagrama de clases UML……………………………………..4

3. Diseño del programa…………………………………………..4

3.1 Algoritmos usados………………………………………5

3.2 Bibliotecas utilizadas……………………………………6

4. Análisis de resultados………………………………………….6

4.1 Objetivos alcanzados……………………………………6

4.2 Objetivos no alcanzados………………………………..7

5. Manual de Usuario………………………………………………7

6. Conclusiones personales……………………………………….8

7. Autoevaluaciones………………………………………………10

**1. Descripción del problema**

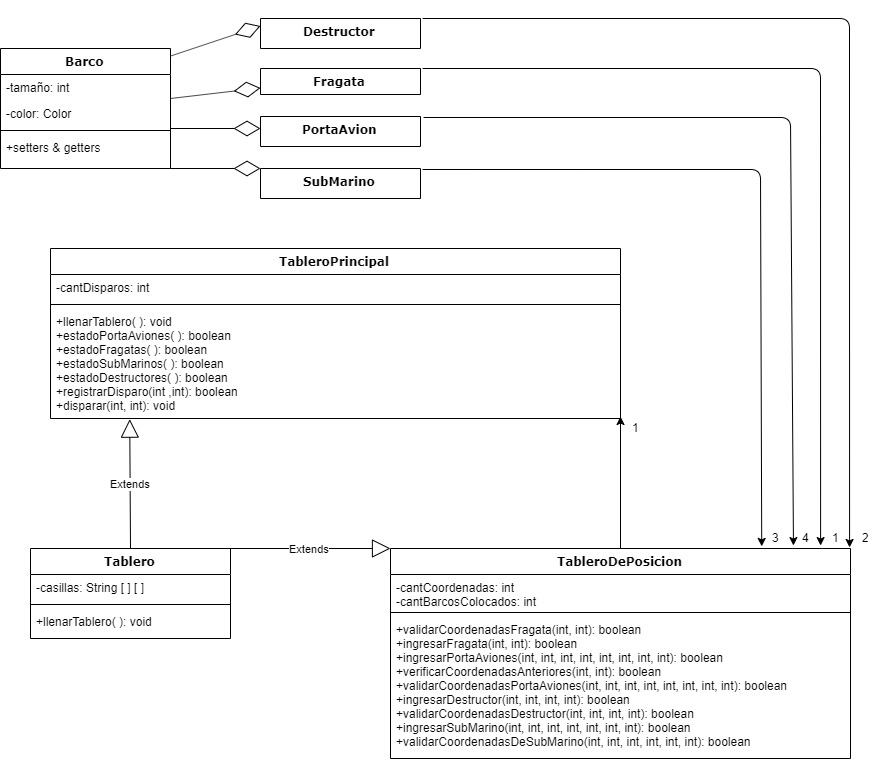
Batalla naval es, a nivel programático un reto relativamente sencillo, pero que apremia la atención al detalle.  
Puesto que se basa en matrices y objetos sueltos dentro de ellas para funcionar, el problema se podría plantear desde muchísimos puntos de vista.

Representar el tablero es la base del programa, siendo este el medio más tangible para que el usuario interactúa con el mismo, debería ser una representación lo suficientemente sencilla para no resultar intimidatoria y lo suficientemente clara para ser entendida a simple vista o bien con un poco de experimentación.

Otra parte esencial para este proyecto es la representación de los barcos, esto es más que nada un reto programático, puestos en la matriz pueden ser cualquier cosa a nivel visual, pero hay que evaluar con cuidado la información relevante para evitar la sobre complicación del problema y tener en cuenta que comparten características, aunque sean distinto tipos de naves.

Una vez tratados los puntos anteriores, lo siguiente es cómo operar la información que se encuentra en la matriz, con los objetos dentro de ellas y la manera en la que interactúan entre ellos para producir un juego funcional, si bien esta parte no es extremadamente difícil, definitivamente requerirá de mucha atención al detalle y solides en el trabajo invertido en ella.

**2. Diagrama de clases UML**



**3. Diseño del programa**

Inicialmente el programa estuvo pensado en un diseño simple en base a la consola. Donde el usuario pudiera interactuar usando el teclado para las diferentes acciones, como para ingresar las coordenadas al colocar un barco en posición o al realizar un disparo. Sin embargo, se planteó el realizar una implementación gráfica, lo cual daría una imagen más elegante al proyecto.   
El programa esta formado por 9 clases y 4 ‘jframes’ donde se trabaja la parte gráfica. Gracias a la variedad de clases el programa es más ordenado, dentro de ellas está la clase main, dos clases abstractas y clases heredadas.

**3.1 Algoritmos usados**

Para solventar el problema de tener varios barcos y tableros, se utilizaron dos clases abstractas: Barco y Tablero. La clase barco contiene dos atributos que son aplicables para todos los barcos, tamaño y color. Gracias a estos atributos se puede saber cuántas casillas ocupa cada uno en el tablero (tamaño) y de qué color se podía representar gráficamente.

La clase tablero cuenta con dos arreglos y una matriz de tamaño 10x10, los cuales se heredan.  
La matriz siempre se va a llenar completamente con espacios vacíos, ya que dentro del constructor colocamos un método que se encarga de ello usando un while. Dichos espacios en blanco representan el “agua” dentro del juego.  
Desde la subclase TableroDePosicion se agregan los distintos barcos dentro de la matriz heredada y las coordenadas que se usaron para ello se almacenan en CoordenadasI y coordenadasJ. Para lograr esto se implementó un método para cada barco específico, todos hacen el mismo trabajo, lo que los hace diferente es la cantidad de entradas que posee cada uno. Cada uno de ellos verifica si las coordenadas ingresadas son correctas, si son las primeras en ingresar o no y se lleva un conteo de la cantidad que son ingresadas correctamente. Gracias a ese conteo se puede recorrer los arreglos fácilmente a la hora de comparar las coordenadas con otras, este trabajo lo realiza un método llamado verificar coordenadas anteriores. Al ser todo verficado correctamente se retorna el valor booleano ‘true’, de lo contrario, ‘false’. Finalmente se agrega la letra inicial de cada barco en las casillas correspondientes.

La subclase TableroPrincipal cuenta con dos algoritmos principales, dentro de los cuales se puede registrar un disparo, es decir, tomar las coordenadas de un disparo y almacenarlas, antes de verificarlas claro. Para la verificación se utiliza el mismo algoritmo del TableroDePosicion, un ciclo recorre los arreglos coordenadasI y coordenadasJ, solo que, en este caso, en lugar de usar una cantidad de coordenadas como contador, se emplea la cantidad de disparos realizados.  
Terminado el proceso de verificación se dispara, para ello simplemente se accede al TableroDePosicion y se consulta que hay en la matriz en las coordenadas del disparo. Si hay un espacio vacío se sabe que dio en el agua, de haber una “P” el disparo acertó en el PortaAviones, con una “F” sabemos que acertó en una Fragata y así con cada inicial del barco. Por último, se utiliza un método por cada barco para saber su estado. Gracias al atributo tamaño se ideó un algoritmo que restara uno a su tamaño cada vez que un barco recibía un disparo, de esta manera al llegar a cero, es porque fue destruido.

**4.2 Bibliotecas utilizadas**

Se hizo uso de las siguientes bibliotecas:

- java.awt.Color: Con ella se pueden importar diferentes colores, los cuales se usan como atributo en los barcos y en la implementación gráfica.   
- java.util.ArrayList: Con ella se puede implementar el uso de arreglos.

- javax.swing.JOptionPane: Con ella se pueden mostrar ventanas con mensajes informativos o errores dentro del apartado gráfico.

- java.util.Scanner: Con ella se puede interactuar con el usuario recibiendo datos de él. Sin embargo, no se usa en el proyecto. Se implementa para pruebas y en un inicio por si no se realizaba el apartado gráfico.

**5. Análisis de resultados**

Los resultados finales de este proyecto son muy prometedores. Logramos afrontar los problemas de la mejor manera y supimos enfocarnos en los principales problemas desde un inicio.

**5.1 Objetivos alcanzados**

-Se logró completar al 100% el proyecto. Todo funciona de la manera esperada y con buen rendimiento. Se puede jugar una partida de inicio a fin.  
-El despliegue de información y los controles del juego son sencillos pero elegantes.

-Le dimos buen uso a la herencia para desarrollar de la mejor manera posible los múltiples objetos de barco y tablero.

-Se logró implementar un apartado gráfico.

-El entendimiento sobre los objetos en java y como se relacionan mejoró gracias a este proyecto

-El instanciar objetos para darles uso es muy notable dentro del programa, así como el uso de los modificadores de visibilidad.

**5.2 Objetivos no alcanzados**

-No se trabajó el proyecto utilizando git desde el inicio. Simplemente se agregó el proyecto el día de entrega. Esto debido a nuestro sistema de trabajo y que se nos olvidó.

**6. Manual de usuario**

Para comenzar a jugar primero debe contar con la aplicación NeatBeans, es una plataforma de programación, dentro de la cual deberá agregar las clases que conforman el juego.   
Una vez teniendo esto se procede a presionar la tecla F6 o bien, buscar el icono de ‘Ejecutar proyecto’ color verde en la parte superior de Neatbeans.   
Segundos después de desplegará una ventana grafica con 3 botones: Comenzar partida, ¿Cómo jugar? y Salir.  
Al presionar el botón de ¿Cómo jugar?, se mostrará una nueva ventana con información sobre el juego, las reglas, controles y todo lo necesario para desarrollar una partida.   
Al oprimir Salir, el programa simplemente se cerrará y, por último, el botón e Comenzar partida. En el momento que se presiona éste, se mostrará en pantalla los tableros de juego para el jugador 1. Donde, primeramente, se deben colocar los barcos en el tablero de posición. Para ello se debe escoger cual barco se desea colocar y seleccionar las filas y columnas respectivas para cada barco en donde se quieren colocar sus casillas. Como nota importante, las filas en los tableros están representadas con letras de la A a la J y las columnas son números del 0 al 9.  
Luego de que el jugador 1 coloque los barcos continua el jugador con el mismo procedimiento, cuando este termina, aparece nuevamente la pantalla del jugador 1 donde podrá empezar a disparar.  
Para disparar solo debe presionar el botón dentro del Tablero principal donde desea hacerlo y listo.  
Se mostrara información en pantalla sobre la cantidad de disparos, a que barco se impactó y la cantidad de barcos restantes.  
El primero en hundir los barcos del otro ganará la partida, lo cual se notificará por medio de un mensaje al respectivo ganador.

**7. Conclusiones personales**

**Alexander Artavia Quesada**

**Conclusión:** Utilizar una clase abstracta para los barcos es una buena aplicación de este tipo de clase, además de que se utiliza herencia, debido a que los cuatro barcos disponibles para jugar, tienen en común su vida o tamaño, además de su color, que funciona como una representación gráfica del barco en el tablero y ayuda a diferenciar barcos, aparte de su tamaño, el color es más fácil para un usuario.

También utilizar una clase abstracta para un tablero, ayuda para que se deriven tableros para cada jugador y también para el tablero de posición, al que se le agregan coordenadas y cantidad de barcos, además de un tablero principal el cual será de utilidad para contar y realizar los disparos al tablero enemigo. Al usar swing para la parte gráfica, se guardan las imágenes de fondo en un paquete para ser utilizados y ordenados de una buena manera. Swing permite de una manera sencilla ingresar botones, etiquetas y demás. Una parte del código de la parte grafica se crea de manera automática cuando se coloca un botón, por ejemplo, lo inicializa y marca donde colocar código para que realice una o más acciones, por lo que esto es muy útil y ahorra espacio. Por lo que, identificar esas dos clases (barco y tablero) fue muy útil para luego crear las otras clases y el uso de swing ayuda y simplifica bastante la realización de una parte grafica del juego.

**Joan Sánchez Chinchilla**

**Conclusión:** Fue muy útil emplear herencia para este proyecto. Resolvió varios problemas a la vez y ayudó a reciclar código.   
Este proyecto fue una buena experiencia tanto en el ámbito personal como el educativo. Logré emplear los conocimientos que obtuve en los últimos meses de clases y además tuve la oportunidad de mezclar esos conocimientos con un apartado visual que resultó mejor de lo que esperaba, además de pasarlo muy bien. El uso de una matriz y arreglos nos dio la solución a este proyecto, considero que dándoles un buen uso se pueden llegar a implementar muchísimas cosas chivas y resolver muchos problemas o bien almacenar datos de manera efectiva.

**Armando Castro Ruiz**

**Conclusión:** El polimorfismo y herencia demostraron ser herramientas verdaderamente útiles, sobre todo a la hora de generar clases muy parecidas entre sí que necesitan alguna distinción, además a veces es más eficiente utilizar ciertas implementaciones de manera más "abstracta" evitando la literalidad en la que uno podría caer a la hora de pensar en cómo programar un objeto.

**8. Autoevaluaciones**

**Nota:** Las evaluaciones se califican con el rango de 1 a 4. Donde 4 es lo mejor y 1 lo peor.

**Alexander Artavia Quesada**

Autoevaluación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterio | 1 | 2 | 3 | 4 |
| Respeto a los demás |  |  |  | x |
| Tolerancia a las diferencias |  |  |  | x |
| Contribución |  |  |  | x |
| Integración al grupo |  |  |  | x |

Coevaluación de Joan Sanchez

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterio | 1 | 2 | 3 | 4 |
| Respeto a los demás |  |  |  | x |
| Tolerancia a las diferencias |  |  |  | x |
| Contribución |  |  |  | x |
| Integración al grupo |  |  |  | x |

Coevaluación de Armando Castro

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterio | 1 | 2 | 3 | 4 |
| Respeto a los demás |  |  |  | x |
| Tolerancia a las diferencias |  |  |  | x |
| Contribución |  |  |  | x |
| Integración al grupo |  |  |  | x |

**Joan Sanchez Chinchilla**

Autoevaluación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterio | 1 | 2 | 3 | 4 |
| Respeto a los demás |  |  |  | x |
| Tolerancia a las diferencias |  |  |  | x |
| Contribución |  |  |  | x |
| Integración al grupo |  |  |  | x |

Coevaluación de Alexander Artavia

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterio | 1 | 2 | 3 | 4 |
| Respeto a los demás |  |  |  | x |
| Tolerancia a las diferencias |  |  |  | x |
| Contribución |  |  |  | x |
| Integración al grupo |  |  |  | x |

Coevaluación de Armando Castro

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterio | 1 | 2 | 3 | 4 |
| Respeto a los demás |  |  |  | x |
| Tolerancia a las diferencias |  |  |  | x |
| Contribución |  |  |  | x |
| Integración al grupo |  |  |  | x |

**Armando Castro Ruiz**

Autoevaluación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterio | 1 | 2 | 3 | 4 |
| Respeto a los demás |  |  |  | x |
| Tolerancia a las diferencias |  |  |  | x |
| Contribución |  |  |  | x |
| Integración al grupo |  |  |  | x |

Coevaluación de Alexander Artavia

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterio | 1 | 2 | 3 | 4 |
| Respeto a los demás |  |  |  | x |
| Tolerancia a las diferencias |  |  |  | x |
| Contribución |  |  |  | x |
| Integración al grupo |  |  |  | x |

Coevaluación de Joan Sanchez

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterio | 1 | 2 | 3 | 4 |
| Respeto a los demás |  |  |  | x |
| Tolerancia a las diferencias |  |  |  | x |
| Contribución |  |  |  | x |
| Integración al grupo |  |  |  | x |