Laboratorio N°1 Paradigmas de la Programación: Batalla Naval

Farrés Martín, Olivares Agustín, Ruiz Joaquin
Lic. en Ciencias de la Computación - FING - UNCuyo 21 de septiembre de 2023

Abstract

En el presente trabajo se abordará el proyecto propuesto en el ejercicio integrador del Laboratorio N°1 de la materia Paradigmas de la Programación, este proyecto se basa en representar el juego de mesa “Batalla Naval” en el lenguaje de programación Java para aprender y fijar conceptos del Paradigma Orientado a Objetos.

1. Introducción

A lo largo de este informe se explicará el proceso de desarrollo del juego “Batalla Naval” trasladado a Java haciendo uso de sus reglas originales con un agregado de habilidades a los barcos. Para su implementación se hará uso de los conceptos del Paradigma orientado a objetos, acompañado con buenas prácticas y especificando las distintas clases a través de un UML que permita ver el nivel de abstracción que se utilizó. Los problemas principales a resolver son los 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.
6. Debe existir al menos algún método polimórfico.

Se adaptará el juego para cumplir tanto con los objetivos planteados como también respetar las reglas de juego, por lo tanto habrá que abstraer el juego en objetos. Se representarán dichos objetos mediante clases en java.

2. Metodología

Planificación

Para resolver el problema planteado, se realizó una breve investigación sobre el juego “Batalla Naval” para obtener información sobre las reglas y el formato del juego. De dicha información se determinó que cada jugador va a poseer 2 tableros de juego de 10*10 casilleros donde, en uno se posicionan los barcos y en el otro se lleva un registro de los disparos realizados al otro jugador, la cantidad total de barcos es de 10, teniendo 5 barcos para cada jugador. La máxima cantidad de turnos es de 100 (cantidad de casilleros), es decir, cada jugador puede disparar como máximo 100 veces.

Figura 1: Ejemplo tablero estándar de batalla naval

Para el agregado de las distintas habilidades de los barcos se realizó una lluvia de ideas para poder definir cuales se agregarían y el cómo serán implementadas de acuerdo al nivel de ventaja que éstas otorguen al jugador que las utilice. Para la creación de islas en el mapa se realizó un algoritmo que cree estructuras de isla de forma aleatoria y limitada para no obstaculizar la partida.

Para representar el tablero, se decidió implementar una matriz de 10x10, donde cada espacio está ocupado por un objeto de la clase “Cell” que contendrá el estado de esa posición, tanto si en este lugar se encuentra un bote, está vacío o si recibió un disparo, se optó por esta implementación por la facilidad que otorga para efectuar los distintos controles y actualizaciones

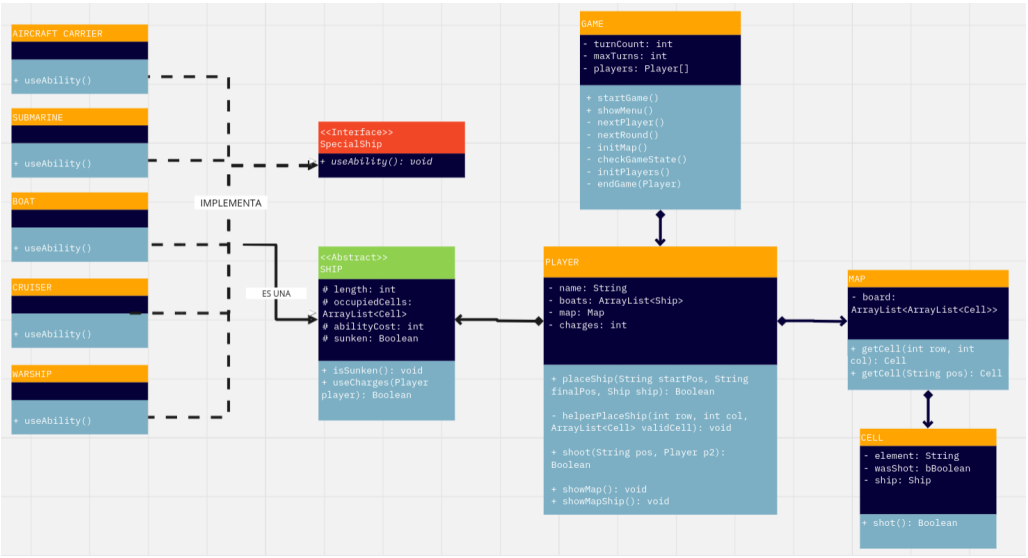
Diseño

Para representar el juego de batalla naval, lo dividimos en 5 clases principales: Game, Player, Map, Cell y Ship.

- Player:
  - Atributos:
    - name: String, el nombre del jugador
    - boats: ArrayList<Ship>, una lista que contiene sus barcos
    - map: Map, una matriz que tendrá información de sus barcos y la información que haya recolectado del enemigo a través de los disparos (si es agua, disparo o hundido)
    - charges: int, un contador de “cargas”, estas cargas son consumibles y permiten al jugador utilizar habilidades de barcos
  - Métodos:

- Game:
    - Atributos:
        - maxTurns: int, por defecto el máximo de turnos será 100, pero el jugador puede optar por jugar menos turnos
        - players: ArrayList<Player>, una lista de jugadores
    - Métodos:
        - método

- Clase:
    - Atributos:
        - atributo:
    - Métodos:
        - método



Agregar: en un principio se había optado agregar un método llamado “nextPlayer()” para iterar jugadores en la clase Game, pero luego de la clase sobre colecciones decidimos que era mejor tener los jugadores en un arraylist y recorrer los jugadores por medio de iterator, sin necesidad de crear el método mencionado. Al igual que el atributo “turnCount”, en vez de guardar en el objeto de la clase Game un contador, decidimos encerrar la lógica del juego dentro de un bucle for que iterara hasta la cantidad deseada de turnos (máximo 100).

Implementación

En un principio se optó por agregar un método llamado “nextPlayer()” para iterar jugadores en la clase Game, pero luego de haber estudiado las colecciones de objetos decidimos que era mejor tener los jugadores en un arraylist y recorrer los jugadores por medio de iterator, sin necesidad de crear el método mencionado. Al igual que el atributo “turnCount”, en vez de guardar en el objeto de la clase Game un contador, decidimos encerrar la lógica del juego dentro de un bucle for que iterara hasta la cantidad deseada de turnos (máximo 100).

- Semana 1: Acercamiento inicial de diagrama UML y desarrollo de clase Ship, Game, Player y Map.
- Semana 2: Ship pasa a ser una superclase, se desarrollan 6 subclases: Boat y SpecialShip, que heredan de Ship, y las clases AircraftCarrier, Cruiser, Submarine y Warship, que son subclases de SpecialShip.
- Semana 3: Desarrollo de clase MapElement e implementación de JavaDoc.