Informe del Bot

Segunda entrega

Ingeniería en informática

Programación 2

Grupo 3

Integrantes del grupo:

Benjamín Rielli

Federico Ferreira

Juan Cabrera

Nicolás Abreu

Docentes: María Parapar, Matías Olave

Fecha de informe 05/07/2022

Contenido

1. Decisiones Principales

UML

En cuanto al UML, y con el objetivo de facilitar la lectura de este, tomamos una serie de decisiones sobre la manera en la que representamos distintas clases. La clase “Coord” no está conectada con sus colaboradores en el diagrama ya que colabora con una gran cantidad de clases, las clases con las que Coord colabora son:

Barco, Tablero, ControladorJuego, CoordenadasNoAlineadas, LeerCoordenadas, Jugador, Robotina, Comando (que se encuentra dentro de Robotina) y Jugada.

La otra clase que no representamos en el diagrama dada su presencia en varias otras clases es Ident, clase simplemente utilizada para simular la función de Id presente en telegram, esta clase colabora con las siguientes clases:

ControladorJuego, JugadorIncorrecto, BarcoLargoIncorrecto, HistóricoEstadisticas, Jugada, Jugador, JuegoConsola, BatallaNaval, GestorBots, GestorPartidas, Usuario (que se encuentra dentro de la clase GestorPartidas) y Robotina.

Otra libertad que nos tomamos en el diagrama UML fue el cómo representamos los handlers, ya que estos usan distintas partes de la lógica de todo el programa decidimos mostrar cómo se conectan entre ellos, pero se puede inferir sus colaboraciones por sus nombres. Todos los handlers utilizan la clase Message, esto no es representado en el diagrama por simplicidad.

Tuvimos que generar dos archivos distintos para los UML ya que la pagina que utilizamos tenía un límite de carga que fue superado,

Código

En cuanto a código, preferimos justificar algunos patrones en el informe en vez de en el código por practicidad, este es el caso de los handlers, todos cumplen con SRP, tienen la única responsabilidad de responder a cierto mensaje, cumplen con LSP, ya que todos son intercambiables con todos sin representar ningún tipo de problema, también cumplen con Chain of Responsability y con Polimorfismo, responden a la misma llamada (Un mensaje de un Usuario) de manera distinta dependiendo de la naturaleza de dicha llamada.

Ident y Coord

Estas clases son utilizadas simplemente como tipos de datos, los utilizamos como si fueran, por ejemplo, strings y son recibidas por una cantidad de clases como parámetros, pero no tienen mas funcionalidad aparte de representar identificaciones y coordenadas.

Tests De Handlers

Implementamos los tests de los handlers simulando diferentes partidas utilizando la clase batalla naval, esta decisión fue tomada para ahorrar tiempo en los tests de los handlers, dado que hechos de otra manera serian muchísimo más tediosos.

Existe un test del handler de inicio que demuestra dos casos, el caso de que el mensaje entre al handler y el caso del que el mensaje no entre al handler, obviamos este test en todos los demás handlers dado que debido a herencia e interfaces todos funcionan exactamente igual.

2. Cambios

Incluimos la clase BatallaNaval, la cual une todas las piezas para soportar múltiples partidas de la batalla naval, esta clase se usa en el programa principal

mandándole los mensajes de los jugadores y luego la respuesta a cada mensaje se le envía tanto al remitente como al oponente de ser necesario.

A petición del docente pasamos la lógica que gestiona el agregado de los barcos a la clase ControladorJuego.

3. Nuevas Funcionalidades

Implementamos una serie de nuevas funcionalidades para facilitar el funcionamiento del Bot, para cumplir con algunos patrones y para implementar las nuevas funcionalidades prometidas. Estas nuevas funciones son: Persistencia, Handlers, Usuarios, Maquina de Estados, Temporizador de Turnos, Excepciones y Radar.

Persistencia

La persistencia fue implementada en colaboración de Juan y Nicolas, esta consiste en una clase encargada de guardar todas las instancias existentes en un archivo Json, cuando el programa se inicia estos archivos Json pueden ser levantados por el código para restaurar la información de las instancias guardadas en los mismos.

Handlers

Los handlers fueron implementados por Federico y ligeramente modificados por Juan, estos implementan el patrón Chain of Responsability y utilizan código de la lógica para actuar acorde a los mensajes del usuario recibidos, devolviendo respuestas para indicarle al usuario como proceder.

Como indicamos en la sección de cambios, decidimos separar al Usuario del Jugador por motivos de practicidad, esta clase fue implementada por Nicolas y ligeramente modificada por Federico, en esta clase mantenemos los datos permanentes del Usuario como su nombre y estadísticas.

Reloj

El temporizador de turnos, o reloj, fue una de las funciones extra prometidas por nuestro equipo, fue implementada por Juan y es utilizada para controlar cuanto tiempo demora un jugador en efectuar su turno, dándole un tiempo límite y otorgándole la victoria al oponente en caso de agotarse.

Otra adición fue la clase Ident, esta clase fue implementada por Juan y es una adición temporal para simular las Ids de los usuarios de telegram mientras la conexión no está hecha.

Una serie de excepciones fueron añadidas por Benjamín para contemplar los distintos tipos de errores que un usuario podría cometer y cómo manejarlos, las excepciones añadidas son; BarcoYaExiste, JugadorIncorrecto, RadaresAgotados, BarcosSuperpuestos, CoordenadaFormatoIncorrecto, CoordenadasNoAlineadas, EstadoPartidaIncorrecto.

BarcoYaExiste

Esta excepción detecta cuando el jugador está tratando de colocar un barco que ya colocó previamente y le indica que ya realizó esa acción.

JugadorIncorrecto

Esta excepción detecta cuando el jugador está tratando de realizar una acción cuando no es su turno y le indica que es el turno del oponente.

RadaresAgotados

Esta excepción detecta cuando el jugador está tratando de usar un radar cuando ya no tiene más de este recurso y le avisa que sus radares han sido agotados.

BarcosSuperpuestos

Esta excepción detecta cuando el jugador está tratando de colocar un barco superpuesto a otro y le indica que esta acción no es posible.

CoordenadaFormatoIncorrecto

Esta excepción detecta cuando el jugador manda un set de coordenadas en un formato incorrecto, revisa cuál de las coordenadas en el set es incorrecta y devuelve al usuario la coordenada que escribió mal.

El return del atributo “Razón” es un enum que está incluido en la misma clase.

CoordenadasNoAlineadas

Esta excepción detecta cuando el jugador manda un set de coordenadas para colocar un barco y estas no están alineadas, le dice al usuario que no es posible colocar un barco con dicha configuración.

EstadoDePartidaIncorrecto

Esta excepción detecta cuando el jugador intenta realizar una acción en un momento no adecuado y le indica que el estado de la partida no es el correcto para esa acción.

Radar

El radar es un tipo de “ataque” que el jugador puede decidir usar, su función de la de revelar lo que hay en un área de 3x3 centrada en un punto seleccionado por el jugador, a pesar de no hacer daño este ataque puede dar la ventaja a su usuario revelando la ubicación de los barcos de manera más eficiente, inclusive si ningún barco es encontrado en su uso, se pueden descartar un total de 9 espacios en un solo turno. Dada esta gran utilidad el radar puede ser utilizado solo una vez por partida.

Robotina

Robotina es el nombre del oponente automático implementado en nuestro bot, se puede acceder a batallas contra ella utilizando el comando “Jugar contra bot”, esta función fue mayormente implementada por Juan, actúa como un usuario mandando mensajes a telegram como cualquier otra persona. Por simplicidad coloca todos sus barcos verticalmente, su manera de atacar implica simplemente tirar coordenadas random hasta que encuentra un barco, luego de eso se pone a probar atacando en los cardinales de ese punto.