Risk: Proyecto Estructuras de Datos 2023-3

Diego Alejandro Albarracin Maldonado
Pontificia Universidad Javeriana
Bogotá, Colombia
di.albarracin@javeriana.edu.co

Carlos Antonio D'Silvestri Ruiz
Pontificia Universidad Javeriana
Bogotá, Colombia
carlosdesilvestrir@javeriana.edu.co

Giseth Valentina Villalobos Rocha
Pontificia Universidad Javeriana
Bogotá, Colombia
villalobos-g@javeriana.edu.co

***Abstract*—El siguiente documento corresponde al informe de la implementación de un sistema de apoyo para el juego de mesa *Risk*® como proyecto de la asignatura de *Estructuras de Datos*. El sistema se implementa como una aplicación que recibe comandos textuales, agrupados en componentes con funcionalidades específicas.**

Keywords: Template, stacks, queues, vector, linked list, set, iterator.

I. Introducción

Risk® es un juego de mesa basado en turnos, cuyo objetivo es la conquista de territorios a través de representaciones de ejércitos, eliminando jugadores oponentes en el proceso. Cada jugador inicia con una cierta cantidad de unidades de ejército ubicadas en algunos territorios del tablero. Para cumplir el objetivo, cada jugador debe atacar territorios vecinos, intentando ocuparlos en el ataque, lo cual se decide con el lanzamiento de dados. El juego termina cuando un jugador ha cumplido con su misión (que puede ser la ocupación de territorios específicos o incluso la ocupación de todos los territorios del tablero).

II. Componente 1: Diseño del Juego

En esta sección se explica a profundidad la lógica y el diseño sobre el cual se basa el código del juego.

A. Definición de TADs

Para la definición de los Tipos Abstractos de Datos se especifica el nombre de cada uno junto a los atributos (datos mínimos) que requiere y los métodos (operaciones) que puede llevar a cabo. Se omite en esta descripción todo tipo de constructor, o método *get* o *set*, sin embargo, se implementa en el código.

1) TAD Juego: Este TAD representa incluye los datos y operaciones involucradas en una partida del juego.

a) Datos mínimos:

- jugadores, vector tipo Jugador, representa el listado de jugadores por partida (mínimo 3, máximo 6)
- mundo, lista de lista (multi-lista) de cadena de caracteres, contiene los nombres de todos los países del juego agrupados por el nombre de los continentes.
- cartas, pila tipo Carta, representa la pila de cartas del juego barajadas
- intercambios, entero, almacena la cantidad de intercambios de cartas que se ha realizado en la partida.

b) Operaciones:

- llenarJugadores(): método que pregunta la cantidad de jugadores, el nombre de cada uno, el color con el que va a jugar y los añade en el vector de jugadores.
- llenarMundo(): método que lee, del archivo de texto, los países del tablero y los almacena en la multi-lista *mundo*.
- llenarBarajaCartas(): método que baraja las cartas con los ejércitos, comodín y misiones para almacenarlos en la pila de cartas.
- calcularInfanteria(): método que, de acuerdo al número de participantes, calcula la cantidad de ejércitos que tendrá cada jugador a su disposición inicialmente.
- validarEnCurso(): método que verifica el momento en que existe un ganador y el juego debe finalizar.
- establecerTurnos(): método que decide qué participante es el siguiente en tomar su turno y le pregunta qué acción quiere llevar a cabo.

c) Representación: A continuación, representación UML del TAD.

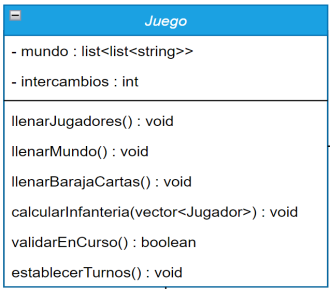

Fig. 1. TAD Juego con sus datos mínimos y operaciones.

2) TAD Jugador:

a) Datos mínimos:

- nombre, cadena de caracteres, nombre del jugador.
- color, cadena de caracteres, color escogido por el jugador para el desarrollo de la partida.
- ejercitos, entero, representa la cantidad de figuras de ejército con el que cuenta el jugador. Se asignan al principio de la partida mediante el cálculo de infantería y se va modificando cada vez que posiciona un ejército sobre un territorio o cuando adquiere nuevos ejércitos mediante el intercambio de cartas.

- territorios, lista de tipo Territorio, representa el listado de territorios que posee un jugador.
- cartas, lista de tipo Carta, consiste en el conjunto de cartas que almacena un jugador y que, luego, puede intercambiar por ejércitos.

b) Operaciones:

- colocarEjercitos(): método que permite que el jugador sitúe sus ejércitos (uno por uno) en el territorio que escoja en cada turno y éste territorio se añada a su lista de territorios.
- ocuparTerritorio(): método que permite al jugador añadir el territorio conquistado a su listado de territorios.
- tomarCarta(): método que permite que el jugador obtenga una nueva carta en su lista de cartas y se elimine de la pila de cartas del juego.
- obtenerNuevasUnidades(): método que valida la forma en que el jugador desea obtener nuevas unidades y, con base a ello, calcula la cantidad de ejércitos nuevos que puede reclamar y lo reclama.
- atacarTerritorioVecino(): método que le pregunta al usuario el territorio que quiere atacar e invoca el lanzamiento de dados por parte del jugador atacante y el jugador defensor.
- lanzarDados(): contiene la lógica que permite realizar el lanzamiento de dados por parte del jugador.
- agregarEjercitos(): método que permite que el jugador desplace parte de sus ejércitos a uno de sus territorios vecinos.

c) Representación: A continuación, representación UML del TAD de Jugador.

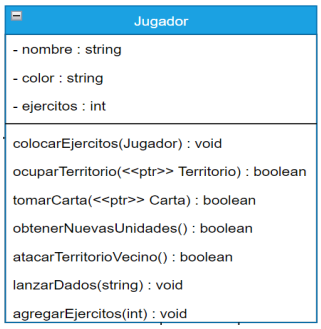

Fig. 2. TAD Jugador con sus datos mínimos y operaciones.

3) TAD Territorio:

a) Datos mínimos:

- nombre, cadena de caracteres, nombre del país/territorio
- ejercitos, entero, representa la cantidad de ejércitos que están posicionados en este territorio.
- vecinos, vector de tipo Territorio, representa el listado de países con los que está conectado un territorio

b) Operaciones:

- recopilarVecinos(): método que permite llenar el vector de vecinos a través de la lectura de un archivo de texto.
- mostrarVecinos(): método que le muestra al usuario los países con los que conecta el territorio seleccionado.

c) Representación: Representación UML del TAD de Territorio.

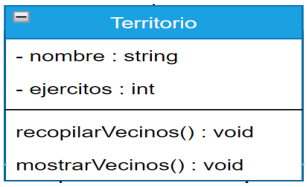

Fig. 3. TAD Territorio con sus datos mínimos y operaciones.

4) TAD Carta:

a) Datos mínimos:

- tipo, caracter, define el tipo de carta que representa, puede ser I - Infantería, C - Caballería, A - Artillería, J - Comodín/Joker, M - Misión.
- territorio, cadena de caracteres, representa el territorio que indica la carta. Si es carta de tipo Comodín o Misión, este campo es igual a "N/A".

b) Operaciones:

- mostrarContenido(): método que permite mostrar el tipo de carta en cuestión.

c) Representación: Representación UML del TAD Carta.

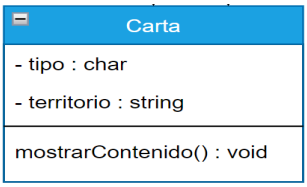

Fig. 4. TAD Carta con sus datos mínimos y operaciones.

B. Diagrama de clases

A continuación se muestra la representación de las clases (TADs) del sistema mediante un diagrama que las relaciona entre sí.

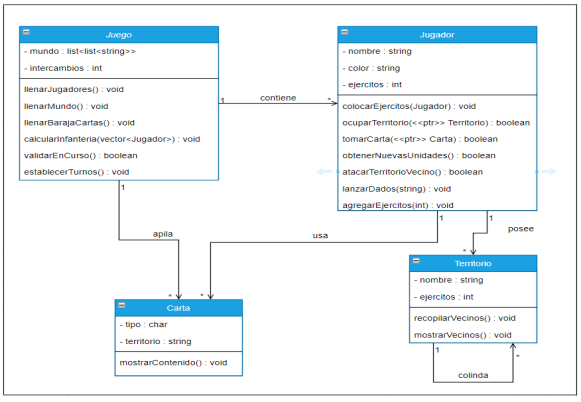

Fig. 5. Diagrama de clases.

III. Componente 2: Configuración del Juego

En esta sección se explica a profundidad el comportamiento de cada una de las funciones usadas para este componente.

A. Imprimir Logo

La función ***imprimirLogoInicio()***, como su nombre lo dice, imprime en consola un logo similar al del juego *Risk*®. Para lograrlo, se hizo uso de modificadores de impresión de texto con el fin de alinear los caracteres de la manera correcta para alcanzar el resultado esperado.

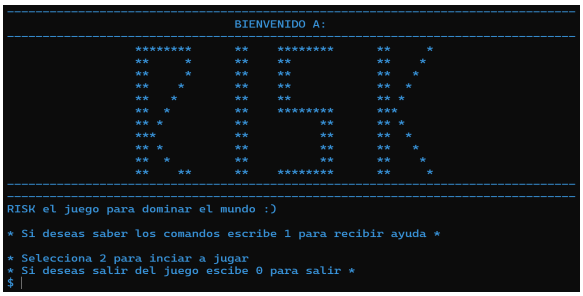

Fig. 6. Impresión del logo de *Risk*® en la consola.

B. Ayuda

La función ***ayuda()*** contiene la información del funcionamiento de todos los comandos, lo anterior con el objetivo de dar una guía al usuario sobre el funcionamiento del sistema.

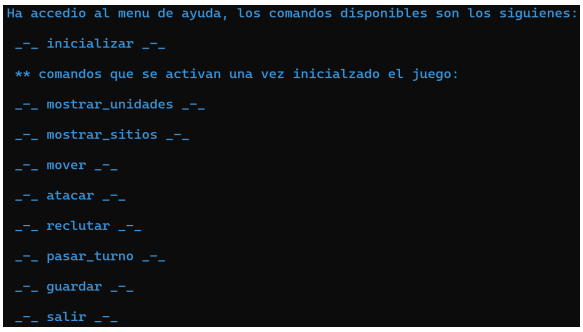

Fig. 7. Salida en pantalla al invocar la función ***ayuda()***.

Cuando es invocada, el usuario puede seleccionar el comando sobre el cual quiere obtener más información y el programa inmediatamente imprimirá en consola la información relacionada con ese comando.

C. Imprimir Dados

La función ***int imprimirDados(int caso)*** se encarga de obtener como parámetro un número entero en un rango de 1 a 6 (número de casos de resultado al lanzar un dado) para luego enviar ese número a una estructura de control *switch*, la cual se encargará de redireccionar el programa, según corresponda, al caso en donde se ejecuta la impresión de un dado con la cara del caso. Posteriormente, se retorna un número entero que contiene el resultado del lanzamiento y el cual será usado

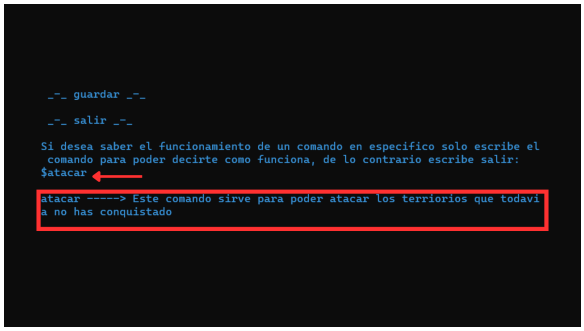

Fig. 8. Salida en pantalla al seleccionar el comando *atacar*.

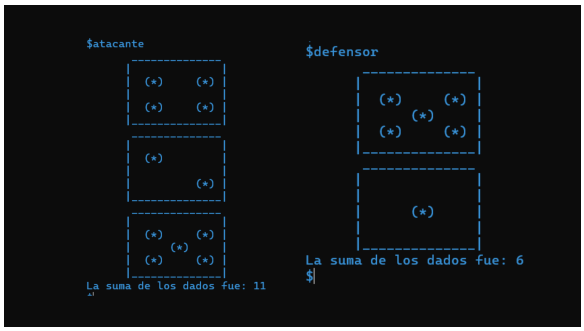

Fig. 9. Salida en pantalla de la función *imprimirDados()*.

para determinar las condiciones de victoria. La función ***int imprimirDados(int caso)*** únicamente es invocada dentro de la función *lanzarDados()*.

D. Condiciones de Victoria

La función ***void condicionesVictoria(vector<int> resultsDadoAtacante, int resultDadoDefensor)*** recibe como parámetros el vector *resultsDadoAtacante* y la variable *resultDadoDefensor*. Dentro de la función, primero se imprime el contenido dentro del vector, luego se le solicita al usuario que ingrese el valor de los 2 dados que quiere conservar para compararlos con el resultado del defensor. A modo de prevención, se realiza una validación para que el usuario no ingrese valores que no existen dentro del vector y lo mantendrá en un bucle hasta que no ingrese valores correctos.

Por último, se valida la suma de los 2 dados del *atacante* y la suma de los dados del *defensor* y se realizan las siguientes comparaciones para determinar el ganador según las reglas del juego:

- ***Atacante > Defensor*:** Si esta condición se cumple, el ganador es el *atacante* y el *defensor* pierde una unidad de ejército en el territorio en el cual está siendo atacado.
- ***Atacante > Defensor*:** Si esta condición se cumple, el *defensor* es el ganador y el *atacante* pierde una unidad de ejército en el territorio desde donde está atacando.
- ***Atacante == Defensor*:** Si esta condición se cumple significa que hay un empate. En este caso, el *defensor* gana y el *atacante* pierde una unidad de ejército desde el territorio desde el cual está atacando.