FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico N.º 2
BATALLA DIGITAL V2.0

[95.12] Algoritmos y Programación II
Curso Calvo
Primer cuatrimestre de 2023

Primera entrega: *16/06/23*

Apellido/s	Nombres	Padrón	Correo electrónico
Mokorel	Pablo	103029	pmokorel@fi.uba.ar
Neuman	Federico José	107049	fedtep@gmail.com
Zambrano Mercado	Amddy Erly	106540	azambranom@fi.uba.ar
Allelo	Agustin	108391	agustinallelo@gmail.com

Índice

1. Consigna

1.1. Objetivos

Generar una pieza de software que simule el funcionamiento del juego Batalla Digital en su versión Jugador contra Jugador.

1.2. Enunciado

Batalla Digital es un juego de mesa para N jugadores en el que se introducen M soldados/armamento por cada jugador en un tablero de X por Y por Z con el fin de que los soldados sobrevivan hasta el final, En cada turno el jugador saca una carta del mazo y ataca una posición del tablero con una mina, eliminando si hay un soldado o armamento en ella y dejando la casilla inactiva por tantos turnos dependiendo el poder de la mina, luego del disparo, el jugador puede optar por mover un soldado o armamento, ya sea horizontal, vertical o diagonal. Un soldado no se puede mover a una casilla inactiva y si se mueve a una casilla con un soldado contrario, se eliminan los 2 soldados. El juego termina cuando todos los jugadores menos uno se queda sin soldados, ganando el jugador con Soldados. Las cartas las pueden acumular o jugar. Hay 6 tipos de cartas, 3 establecidas por el enunciado y 3 por el grupo. La carta se juega al final de cada turno de cada jugador. Las cartas establecidas por el TP son: ataque químico, al atacar contamina 125 casilleros (5x5x5) por 10 turnos en el centro, 8 turnos el siguiente radio, y así. Un avión radar (si está en el aire puede detectar minas en su zona por cada turno) y un barco (si está en el agua puede disparar un misil una vez por cada turno, adicional a los disparos del turno). El terreno tiene 3 casilleros tipos de casilleros, uno es tierra, otro es agua y otro es aire. El nivel 1 a 5 del tablero es tierra o agua, y el resto de los niveles son aire

1.3. Interfaz de usuario

Toda la interfaz de usuario debe estar basada en texto, la entrada y salida y el estado del tablero tiene que mostrarse utilizando un Bitmap (ver librería) de una manera ideada por el grupo. No es necesario que se limpie la pantalla, simplemente escribir el estado del tablero luego de cada jugada.

1.4. Cuestionario

Responder el siguiente Cuestionario:

1. ¿Qué es un svn?
2. ¿Qué es git?
3. ¿Qué es Github?
4. ¿Qué es un valgrind?

1.5. Normas de entrega

Trabajo práctico grupal: 6 personas. Se deberá elegir un nombre de Grupo
Reglas generales: respetar el Apéndice A.
El tablero se debe implementar utilizando la clase Lista. Todos los objetos deben estar en memoria dinámica:
Se deberá subir un único archivo comprimido al campus, en un link que se habilitará para esta entrega. Este archivo deberá tener un nombre formado de la siguiente manera:
Nombre de Grupo-TP2.zip
Deberá contener los archivos fuentes (no los binarios), el informe del trabajo realizado, las respuestas al cuestionario, el manual del usuario y el manual del programador (Todo en el mismo

PDF).
La fecha de entrega vence el día lunes 16/06/23 a las 23.59hs.
Se evaluará: funcionalidad, eficiencia, algoritmos utilizados, buenas prácticas de programación, modularización, documentación, gestión de memoria y estructuras de datos.

1.6. Apéndice A

1. Usar las siguientes convenciones para nombrar identificadores.

   a) Clases y structs: Los nombres de clases y structs siempre deben comenzar con la primera letra en mayúscula en cada palabra, deben ser simples y descriptivos. Se concatenan todas las palabras. Ejemplo: Coche, Vehiculo, CentralTelefonica.

   b) Métodos y funciones: Deben comenzar con letra minúscula, y si está compuesta por 2 o más palabras, la primera letra de la segunda palabra debe comenzar con mayúscula. De preferencia que sean verbos. Ejemplo: arrancarCoche(), sumar().

   c) Variables y objetos: las variables siguen la misma convención que los métodos. Por Ejemplo: alumno, padronElectoral.

   d) Constantes: Las variables constantes o finales, las cuales no cambian su valor durante todo el programa se deben escribir en mayúsculas, concatenadas por "_". Ejemplo: ANCHO, VACIO, COLOR_BASE.

2. El lenguaje utilizado es C++, esto quiere decir que se debe utilizar siempre C++ y no C, por lo tanto una forma de darse cuenta de esto es no incluir nada que tenga .h, por ejemplo #include <iostream>.

3. No usar sentencias 'using namespace' en los .h, solo en los .cpp. Por ejemplo, para referenciar el tipo string en el .h se pone std::string.

4. No usar 'and' y 'or', utilizar los operadores '&&' y '||' respectivamente.

5. Compilar en forma ANSI. Debe estar desarrollado en linux con eclipse y g++. Utilizamos el estándar C++98.

6. Chequear memoria antes de entregar. No tener accesos fuera de rango ni memoria colgada.

7. Si el trabajo práctico requiere archivos para procesar, entregar los archivos de prueba en la entrega del TP. Utilizar siempre rutas relativas y no absolutas.

8. Entregar el informe explicando el TP realizado, manual de usuario y manual del programador.

9. Comentar el código. Todos los tipos, métodos y funciones deberían tener sus comentarios en el .h que los declara.

10. Modularizar el código. No entregar 1 o 2 archivos, separar cada clase o struct con sus funcionalidades en un .h y .cpp

11. No inicializar valores dentro del struct o .h.

12. Si cualquier estructura de control tiene 1 línea, utilizar siempre, por ejemplo: for(int i = 0; i <10; i++) { std::cout « i; }

2. Cuestionario

2.1. ¿Qué es un SVN?

SVN (Subversion) es un sistema de control de versiones de software utilizado para gestionar y rastrear los cambios realizados en archivos y directorios a lo largo del tiempo. Proporciona un historial completo de todas las modificaciones realizadas en un proyecto, lo que facilita la colaboración entre desarrolladores. SVN utiliza un modelo cliente-servidor, donde los desarrolladores pueden realizar cambios en sus copias locales y luego sincronizarlos con un repositorio centralizado.

2.2. ¿Qué es Git?

Git es otro sistema de control de versiones, pero a diferencia de SVN, es distribuido, lo que significa que no depende de un servidor central. Cada desarrollador tiene una copia completa del repositorio en su máquina local, lo que les permite trabajar de forma independiente y realizar cambios sin estar conectados a una red. Git es conocido por ser rápido, eficiente y robusto, y se utiliza ampliamente en proyectos de software de todos los tamaños.

2.3. ¿Qué es GitHub?

GitHub es una plataforma basada en web que utiliza el sistema de control de versiones Git. Es un servicio de alojamiento de repositorios que permite a los desarrolladores colaborar en proyectos y compartir su código. GitHub proporciona una interfaz fácil de usar para trabajar con Git y ofrece características adicionales, como seguimiento de problemas, administración de proyectos y colaboración en equipo. Muchos proyectos de código abierto y empresas utilizan GitHub para alojar y administrar sus repositorios de código fuente.

2.4. ¿Qué es Valgrind?

Valgrind es una herramienta de depuración y perfilado de código utilizada principalmente en entornos de desarrollo de software. Proporciona varias herramientas, siendo la más conocida Memcheck, que se utiliza para detectar errores de memoria, como fugas de memoria y acceso a memoria no válida. Valgrind también puede realizar análisis de rendimiento y detección de errores relacionados con hilos de ejecución. Es una herramienta poderosa para mejorar la calidad y el rendimiento del código, especialmente en proyectos escritos en C o C++.

3. Manual de usuario

3.1. Manual Batalla Digital

Batalla Digital es un juego de N jugadores en el cual a cada jugador se le asignan M soldados/armamentos en un tablero de X por Y por Z con el fin de que los soldados sobrevivan hasta el final.En cada turno el jugador saca una carta del mazo y ataca una posición del tablero con una mina, eliminando si hay un soldado o armamento en ella y dejando la casilla inactiva por tantos turnos dependiendo el poder de la mina, luego del disparo, el jugador puede optar por mover un soldado o armamento, ya sea horizontal, vertical o diagonal.
Normas:
A los jugadores se les asignan la cantidad de fichas que pidan por consola donde el 75 % de las fichas seran soldados y el 25 % armamentos.
Luego de la asigancion de fichas por jugadores, se le pedira a cada uno de ellos que las coloque en el tablero para luego comenzar a jugar.(los soldados y armemanentos solo puede ser colocados en tierra).
Al comenzar cada turno se apilara una carta del mazo del juego en la lista de cartas de cada jugador.
El primer jugador en hacer un movimiento es el jugador 1.
El jugador puede realizar tres acciones por turno, colocar una mina, mover un soldado o usar una carta.
Si el jugador logra matar a todos soldados del jugador contrario ya sea minando el casillero donde se encuentra cada uno , haciendo que pise una mina, o utlizando alguna carta , gana la partida.
Al finalizar el juego se le entregaran los archivos de imagen del mismo.

3.2. Compilación

Para compilar el programa solo bastara con ejecutar el comando make en la terminal de Linux dentro de la carpeta donde se encuentran los archivos de la aplicación.

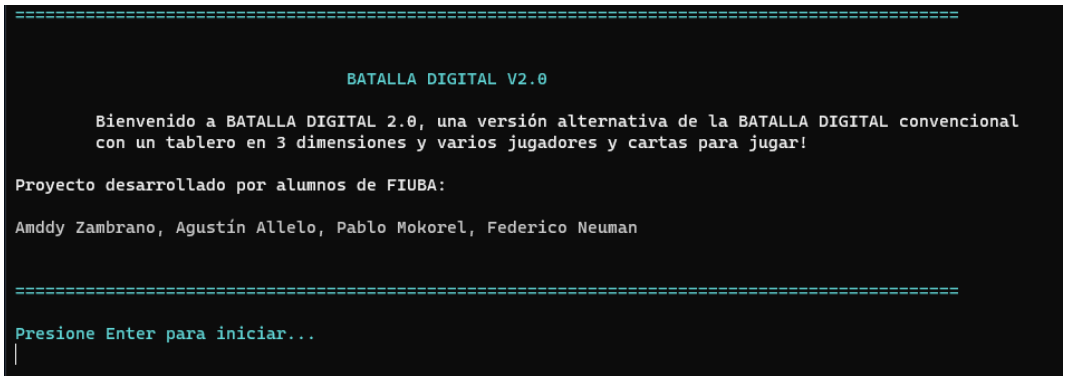
```
pablo@DESKTOP-VRAKFG1:/mnt/c/Users/Pablo/Desktop/TP2_ALGO_2-development/TP2$ make
g++ -g -pedantic -Wall -w -o carta.o  -c Carta.cpp
g++ -g -pedantic -Wall -w -o ficha.o  -c Ficha.cpp
g++ -g -pedantic -Wall -w -o casillero.o  -c Casillero.cpp
g++ -g -pedantic -Wall -w -o tablero.o  -c Tablero.cpp
g++ -g -pedantic -Wall -w -o jugador.o  -c Jugador.cpp
g++ -g -pedantic -Wall -w -o juego.o  -c Juego.cpp
g++ -g -pedantic -Wall -w -o interfaz.o  -c Interfaz.cpp
g++ -g -pedantic -Wall -w -o main.o  -c main.cpp
g++ -g -pedantic -Wall -w -o EasyBMP.o -c EasyBMP/EasyBMP.cpp
g++ -g -pedantic -Wall -w -o batalla_digitalv2 interfaz.o juego.o casillero.o tablero.o jugador.o carta.o ficha.o EasyBMP.o  main.o
rm *.o
```

3.3. Ejecución

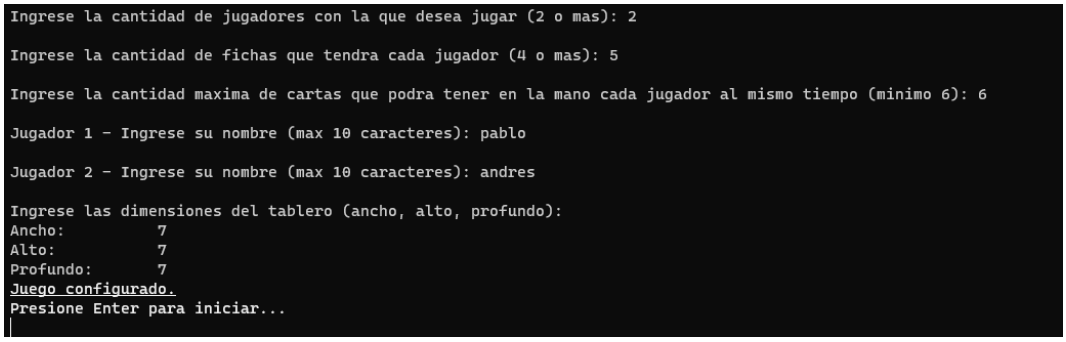
Luego de compilar el programa ya se puede proceder con la ejecución del mismo escribiendo por consola ./batalla_digitalv2.

3.4. Juego

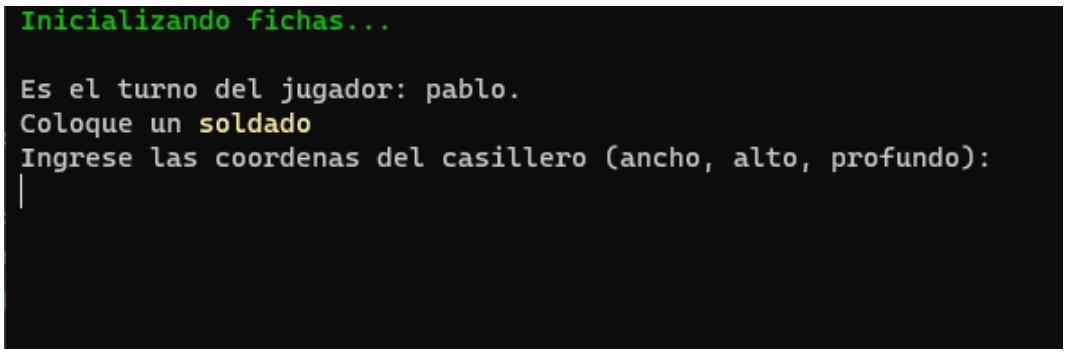
Una vez ejecutado el programa se mostrara una presentacion en pantalla.



Luego de la presentación, inicializa el juego pidiendo por pantalla la cantidad de jugadores, la cantidad de fichas por jugador, la cantidad máxima de cartas que puede tener cada jugador en la mano, los nombres de los jugadores y las dimensiones del tablero.



Después de inicializar los parámetros del juego, se procederá con la inicialización de las fichas. Esto consiste en que a cada jugador se le pedirá, hasta acabar sus fichas de soldado y armamento, que las coloque en el tablero.



Las fichas soldados/armamento solo pueden ser colocada en un tipo de casillero TIERRA, sino, no se permitira realizar el ingreso de las mismas.

```
Ingrese las coordenas del casillero (ancho, alto, profundo):
3
4
5

Ingrese las coordenas del casillero (ancho, alto, profundo):
2
5
4
0
Soldado colocado.
Coloque un soldado
```

Al terminar de ingresar las fichas soldado/armementos, el primer movimiento que se le pregunta al jugador es si quiere utilizar una carta de su lista de cartas.

```
Batalla iniciada, el juego ha comenzado!

Ha sacado una carta del mazo.

 Jugador: 'afsaf', tus cartas son:
        * 1 : Contamina 125 casilleros por 10 turnos en el centro, 8 turnos el siguiente radio y asi
Desea usar una carta? S/N
S

Cual carta quieres usar?:
```

En este ejemplo se ejecuto el ataque quimico.

```
Ha sacado una carta del mazo.

 Jugador: '1', tus cartas son:
        * 1 : Contamina 125 casilleros por 10 turnos en el centro, 8 turnos el siguiente radio y asi
Desea usar una carta? S/N

s

Cual carta quieres usar?: 1

Ingrese las coordenas del casillero (ancho, alto, profundo):
25
25
25
Ha realizado un ataque químico.
```

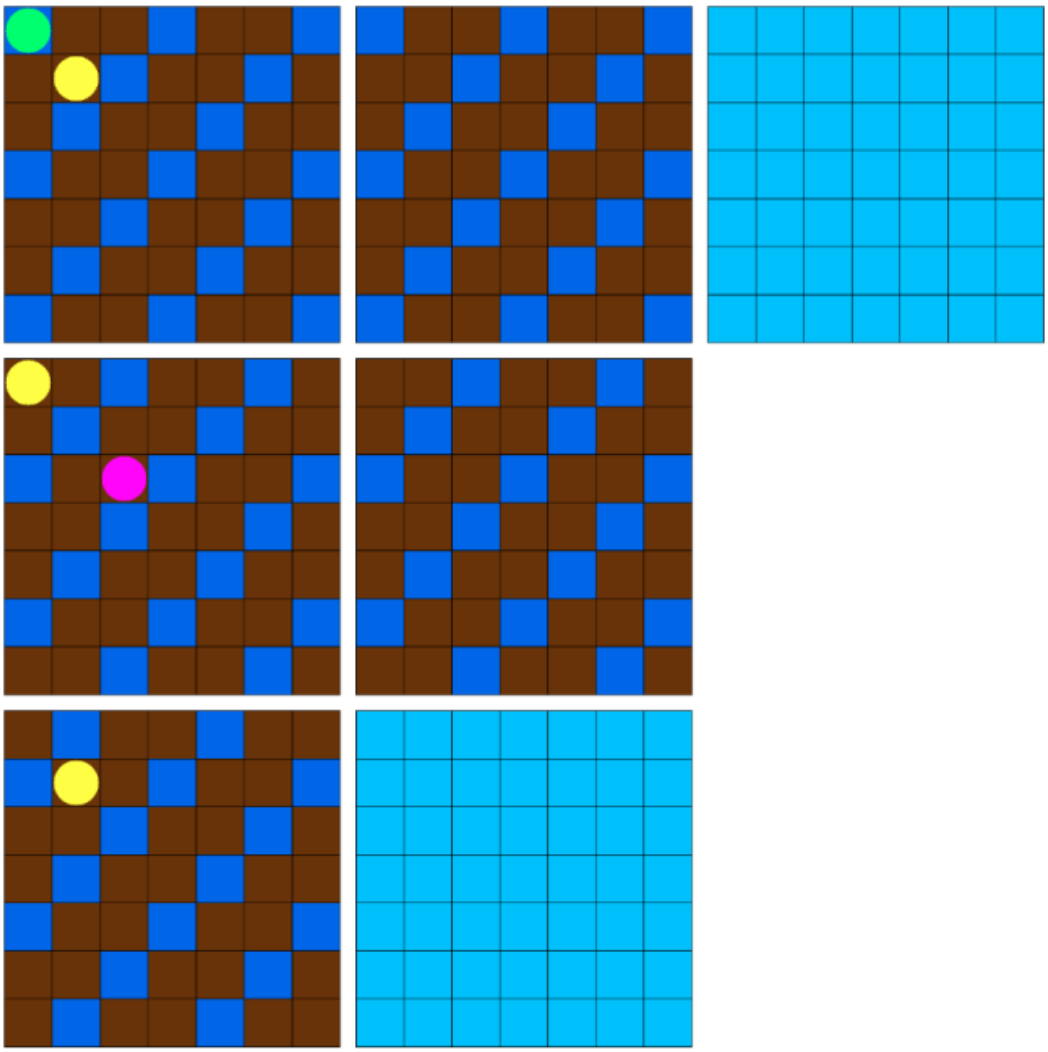
Luego de ejecutar un carta se pide al jugador que ingrese una mina en alguna posicion valida.

```
Coloque una mina
Ingrese las coordenas del casillero (ancho, alto, profundo):
1
1
1
```

Luego de colocar la mina, el jugador tiene que mover un soldado o armamento a una posicion valida (casillero adyacente).
Si en el casillero de destino hay un soldado enemigo se elimanaran ambos soldados y si hay una mina, esta explotara dejando inactivo al casillero por 5 turnos.

```
Desea mover un soldado? S/N
s
Ingrese las coordenadas donde se encuentre el soldado.
1
1
2
Ingrese las coordenadas de destino adyacente  del soldado.
1
1
3
Soldado movido.
Desea mover un armamento? S/N
s
Ingrese las coordenadas donde se encuentre el soldado.
2
2
3
Ingrese las coordenadas de destino adyacente  del soldado.
2
2
4
Soldado movido.
```

Al finalizar cada turno se imprimira un archivo .bmp con el tablero del juego.



Las fichas se representaran de la siguiente manera:

- MINA: Verde brillante
- SOLDADO: Amarillo brillante
- ARMAMENTO: Magenta
- BARCO: Naranja
- AVIÓN: Gris
- BLOQUEADO: Rojo
- VENENO: Verde

4. Manual de programador

4.1. Mantenibilidad y extensibilidad

Con el fin de mantener y extender el juego, se opto por un desarrollo generico de las clases Jugador, Casillero e interfaz.

4.1.1. Clase Jugador

La clase Jugador representa a un jugador en el juego. Tiene los siguientes atributos y métodos:

```
class Jugador {
  friend class Ficha;
private:
  std::string nombreJugador;
  Pila<Ficha *> *fichas;
  int cantidadSoldados;
  Lista<Carta *> *cartas;
  int contadorTurnos; // Cuando suma un turno, resta uno al contador_bloqueado
                      // de cada ficha
public:
  /* ...
  Codeium: Refactor | Explain
  Jugador(std::string nombreJugador);
  /* ...
  Codeium: Refactor | Explain
  ~Jugador();
  /* ...
  std::string getNombre();
  /* ...
  Ficha *getFicha();
  /* ...
  void setFicha(TipoDeFicha tipoDeFicha);
  /* ...
  int getCantidadFichas();
  /* ...
  void setCantidadFichas(int cantidad);
  /* ...
  void setFichas(Pila<Ficha *> *nuevasFichas);
  /* ...
  Pila<Ficha *> *getPilaFichas();
  /* ...
  Lista<Carta *> *getCartas();
  /* ...
  Carta *getUltimaCarta();
  /* ...
  void setCartas(Lista<Carta *> *nuevasCartas);
  /* ...
  int getTurnos();
  /* ...
  void setTurnos(int nuevosTurnos);
};
#endif /* JUGADOR.H */
```

4.1.2. Clase Casillero

La clase Casillero representa una casilla en el tablero. Tiene los siguientes atributos y métodos:

```cpp
#ifndef CASILLERO_H_
#define CASILLERO_H_

#include "Constantes.h"
#include "Ficha.h"

class Casillero {
private:
  Ficha *ficha;
  Casillero ****casillerosAdyacentes;
  EstadoCasillero estado;
  TipoTerreno terreno;
  int contadorDeTurno;
public:
  /*...
  Codeium: Refactor | Explain
  Casillero();
  /*...
  Codeium: Refactor | Explain
  Casillero(Ficha *ficha);

  /*...
  Codeium: Refactor | Explain
  ~Casillero();
  /*...
  void asignarCasilleroAdyacente(int x, int y, int z,
                                 Casillero *casilleroAdyacente);
  /*...
  bool tieneAdyacente(unsigned int x, unsigned int y, unsigned int z);
  /*...
  bool esAdyacenteLineal(Casillero *);
  /* Pre : que exista la clase casillero previamente.···
  Ficha *getFicha();
  /*...
  void setFicha(Ficha *nuevaFicha);
  /*...
  void eliminarFicha();
  /*...
  Casillero *getAdyacente(unsigned int i, unsigned int j, unsigned int k);
  /*...
  EstadoCasillero getEstado();
  /*...
  unsigned int getLongitudFichasIguales(unsigned int i, unsigned int j,
```

```cpp
  /*...
  bool tienenMismaFicha(Casillero *casilleroAdyacente);
  /*...
  bool estaBloqueado();
  /*...
  bool estaEnvenenado();
  /*...
  void bloquear();
  /*...
  void desbloquear();
  /*...
  bool estaVacio();
  /*...
  void setContadorDeTurnos(int turnos);
  /*...
  int getContadorDeturnos();
  /*...
  void bajarTurno();
  /*...
  void envenenar();
  /*...
  TipoTerreno obtenerTerreno() const { return terreno; }
  /*...
  void asignarTerreno(TipoTerreno nuevoTerreno) { terreno = nuevoTerreno; }
};
#endif /* CASILLERO_H_ */
```

4.1.3. Clase Tablero

La clase Tablero representa una lista de casilleros que conforman el tablero. Tiene los siguientes atributos y métodos:

```cpp
#ifndef TABLERO_H_
#define TABLERO_H_


#include "Lista.h"
#include "Casillero.h"
#include "Ficha.h"

class Tablero{
private:
    Lista<Lista<Lista<Casillero *> *> *> * casilleros;
    int dimensiones[3];

public:
    /*...
    Tablero(unsigned int x, unsigned int y, unsigned int z);

    /*...
    ~Tablero();

    /*...
    int * getDimensiones();

    /*...
    bool existeCasillero(int x, int y, int z);

    /*...
    Casillero * getCasillero(unsigned int x, unsigned int y, unsigned int z);

};


#endif /* TABLERO_H_ */
```

4.1.4. Clase Ficha

La clase ficha represanta a todas las fichas del juego. Tiene los siguientes atributos y métodos:

```cpp
class Ficha {
private:

    EstadoFicha estado;
    TipoDeFicha tipoDeFicha;
    std::string identificadorDeJugador;
public:
    /*...
    Codeium: Refactor | Explain
    Ficha();
    /*...
    Codeium: Refactor | Explain
    Ficha(TipoDeFicha tipoDeFicha, std::string indetificadorDeJugador);
    /*...
    Codeium: Refactor | Explain
    Ficha(std::string identificadorDeJugador);
    /*...
    Ficha &operator=(const Ficha &otraFicha);
    /*...
    Codeium: Refactor | Explain
    Ficha(Ficha *fichaOriginal);
    /*...
    Codeium: Refactor | Explain
    ~Ficha();
    /*...
    TipoDeFicha getTipoDeFicha();
    /*...
    void setTipoDeFicha(TipoDeFicha tipoDeFicha);
    /*...
    bool esIgual(Ficha *ficha2);
    /*...
    void bloquear();
    /*...
    void desbloquear();
    /*...
    bool estaBloqueada();
    /*...
    std::string getIdentificadorDeJugador();
    /*...
    int getNumeroDeFicha();
};
```

4.1.5. Clase Carta

La clase ficha represanta a todas las cartas del juego. Tiene los siguientes atributos y métodos:

```cpp
#ifndef CARTA_H_
#define CARTA_H_
#include <string>

typedef enum {
    ATAQUE_QUIMICO,
    AVION_RADAR,
    BARCO_MISIL,
    REFUERZO,
    BOMBARDEO,
    ESPIONAJE
} funcion_t;

class Carta {
private:
    funcion_t funcion;

public:
    /*
     *pre: recibe un tipo de dato funcion_t.
     *post: crea una instancia de la clase carta, esta debe recibir un tipo funcion_t valido.
     */
    Codeium: Refactor | Explain
    Carta(funcion_t funcion);

    /*
     *pre: que exista previamente una instancia de la clase carta.
     *post: devuelve un tipo funcion_t que indica el tipo de carta que es.
     */
    funcion_t getFuncion();

    /*
     *pre: que exista previamente una instancia de la clase carta.
     *post: duelve una descripcion de la funcion de la carta.
     */
    std::string getDescripcion();
};

#endif /* CARTA_H_ */
```

4.1.6. Constantes

Aqui se encuentran todas las constantes del juego. Este archivo esta generado para jugar el juego planteado anteriormente.

```cpp
#ifndef CONSTANTES_H_
#define CONSTANTES_H_

#include <string>

#define VACIO '.'

enum TipoDeFicha { NO_DEFINIDA, MINA, SOLDADO, ARMAMENTO, BARCO, AVION };

enum EstadoFicha { FICHA_DESBLOQUEADA, FICHA_BLOQUEADA };

enum EstadoCasillero {
    CASILLERO_BLOQUEADO,
    CASILLERO_DESBLOQUEADO,
    CASILLERO_ENVENENADO
};

enum TipoTerreno { TIERRA, AGUA, AIRE };

#endif /* CONSTANTES_H_ */
```

4.1.7. Clases para manipular datos

Para la manipulación y estructura de datos se utilizaron listas, pilas y colas. Sus respectivos códigos con métodos y atributos se encuentran al final del informe, junto con todos los códigos del programa.

4.1.8. Clase Juego

Esta clase contiene la lógica del juego "Batalla Digital 2"(movimientos, validaciones y colocación de fichas). Esta clase no es reutilizable como las anteriores, ya que se creó únicamente

con el fin de jugar al juego planteado anteriormente.

4.1.9. Clase Interfaz

La clase Interfaz se utilizará con el fin de visualizar a los usuarios el juego y el ingreso de datos (dimensiones, colocación de fichas o movimientos). Esta clase es únicamente válida para el juego planteado anteriormente, por lo cual este código no es reutilizable como los anteriores.

De esta manera se permite al programador programar su propio juego y modificar o mantender el presentado, modificando la logica del mismo que se encuentra en Juego.cpp e Intefaz.cpp.

En las definiciones de las clases y archivos anteriormente presentadas se encuentran brevemente descriptos los métodos de cada una de ellas.

4.2. Reutilización

En tanto a la reutilización del código, se permitirán utilizar las clases Jugador, Casillero , Tablero, Ficha y Cartas con el fin de poder crear un nuevo juego con cualquier cantidad de juadores, teniendo la opción de crear tableros de nxm, jugadores con cualquier cantidad de fichas y cartas y movimientos de cualquier tipo.
Se recomienda modificar o crear un nuevo archivo juego.cpp e interfaz .cpp para crear un nuevo juego.

5. Consideraciones Generales

5.1. Estandarización

Lenguaje de Programación	C++ - G++ en Linux Ubuntu 18.04
Espaciado	4 espacios por nivel
Funciones	nombreFuncion()
Variables locales	variable
Clase	Clase
Métodos de clase	nombreMetodo()
Variables de clase	variable

5.2. Implementación

5.3. Makefile

Se creó un makefile para compilar y correr el programa principal y el testeo con Valgrind para verificar fugas de memoria o errores de ejecución graves.

Los mismos pueden utilizarse con:

```
$ make              // Compila el programa principal
$ make all          // Compila el programa principal$
$ make leak_test    // Ejecuta el programa para verificar fugas de memoria
```

6. Desarrollo

Para el desarrollo de la aplicación se optó por usar disntos TDAs.

Se comenzó desarrollando la clase Jugador, cuyos atributos contendrán las fichas, cartas y cantidades para su manipulación durante el juego.

Luego se procedió con el desarrollo de la clase Casillero y Tablero, importantes ya que dentro de ellos se desarrolla todo el juego.

Después se realizaron las clases Fichas y Cartas, importantes para que los jugadores puedan tener una noción del juego al que están jugando.

La clase Interfaz se programó luego de todas estas clases, pensando minuciosamente la mejor forma de representar los datos visualmente y que los jugadores tengan una buena experiencia al ejecutar el juego.

Luego de haber definido estas clases, se optó por realizar las funciones lógicas del juego (como verificaciones y movimientos), para luego poder realizar la lógica del mismo (BATALLA DIGITAL) en un archivo aparte.

A continuación se explicará brevemente para qué sirve cada clase de este juego.

6.1. Clase Jugador

Esta clase se creo para representar a un jugador en el juego. incluye a las variables para almacenar las fichas, cartas y turnos, junto con las funciones necesarias para poder acceder a sus atributos.

6.2. Clase Casillero

Esta clase representa un casillero del juego. posee las variables y atributos para almacenar una ficha, el estado del casillero y casilleros adyacentes. También tiene funciones para acceder y modificar estos valores, así como para realizar operaciones relacionadas con el juego, como verificar adyacencia, contar fichas iguales y realizar acciones en el casillero.

6.3. Clase Tablero

esta clase representa al tablero. Tiene variables y atributos para almacenar las dimensiones del tablero y una lista de casilleros. Posee diferentes tipos de terrenos según su posición. También establece las conexiones entre los casilleros adyacentes. Hay funciones para obtener las dimensiones, verificar la existencia de un casillero y obtener un casillero específico del tablero.

6.4. Clase Ficha

La clase representa una ficha, con variables para tipo, identificador de jugador y estado. Proporciona funciones para manipular y obtener información sobre la ficha, como establecer el tipo, verificar igualdad y bloqueo, y obtener el identificador del jugador

6.5. Clase Carta

La clase representa una carta de habilidad para utilizar en el juego. Tiene una función asociada que determina su efecto. Se puede obtener la descripción de la carta, que corresponde a una funcionalidad específica definida en un arreglo preestablecido.

6.6. Juego

Después de haber creado todas estas clases, se empezó a codificar el proceso del juego.

Como desarrollo base, se pensó únicamente en un juego en el cual, si se eliminaban los 4 soldados de un jugador (siendo minados), el contrario sería el ganador.

Al finalizar esta "primera parte"del juego, se implementó la segunda. En esta "segunda parte", se realizaron todas las validaciones de movimientos de soldados, verificando si en la casilla de destino se encontraba una mina o soldado enemigo.

Finalmente, se generaron las funciones de las cartas y la lógica de funcionamiento en cada turno de las mismas. Para esta parte, se pudieron utilizar funciones propias de eliminación, movimiento y manipulación de datos hechas anteriormente.

Luego de todas estas lógicas, se programó el desarrollo del juego y la manipulación de los jugadores dentro del mismo.

7. Código

En esta parte del informe se muestran los codigos fuentes del programa.

7.1. Makefile

Listing 1: Makefile

```
CFLAGS=-g -pedantic -Wall -w
CC=g++

all: batalla_digitalv2 clear

#Make Main

run: batalla_digitalv2
	./batalla_digitalv2

batalla_digitalv2: carta.o ficha.o casillero.o tablero.o jugador.o juego.o interfaz.o main.o easyBMP.o
	$(CC) $(CFLAGS) -o batalla_digitalv2 interfaz.o juego.o casillero.o tablero.o jugador.o carta.o ficha.o EasyBMP.o main.o

#Make Obj

main.o: main.cpp
	$(CC) $(CFLAGS) -o main.o  -c main.cpp

juego.o: Juego.cpp Juego.h
	$(CC) $(CFLAGS) -o juego.o  -c Juego.cpp

interfaz.o: Interfaz.cpp Interfaz.h
	$(CC) $(CFLAGS) -o interfaz.o  -c Interfaz.cpp

tablero.o: Tablero.cpp Tablero.h
	$(CC) $(CFLAGS) -o tablero.o  -c Tablero.cpp

casillero.o: Casillero.cpp Casillero.h
	$(CC) $(CFLAGS) -o casillero.o  -c Casillero.cpp

jugador.o: Jugador.cpp  Jugador.h
	$(CC) $(CFLAGS) -o jugador.o  -c Jugador.cpp

ficha.o: Ficha.cpp  Ficha.h
	$(CC) $(CFLAGS) -o ficha.o  -c Ficha.cpp

carta.o: Carta.cpp  Carta.h
	$(CC) $(CFLAGS) -o carta.o  -c Carta.cpp

easyBMP.o:
	$(CC) $(CFLAGS) -o EasyBMP.o -c EasyBMP/EasyBMP.cpp


#Make Clear

clear:
```

```
47          rm *.o
48
49  install:
50          cp batalla_digitalv2 ../batalla_digitalv2
51
52  leak_test:
53          valgrind --leak-check=full ./batalla_digitalv2
```

7.2. Main

Listing 2: main.cpp

```
1   #include "Carta.h"
2   #include "Cola.h"
3   #include "Constantes.h"
4   #include "EasyBMP/EasyBMP.h"
5   #include "Interfaz.h"
6   #include "Juego.h"
7   #include "Jugador.h"
8   #include "Tablero.h"
9   #include <iostream>
10  #include <string>
11
12  int main() {
13    try{
14      Juego *juego = new Juego();
15      juego->jugarBatallaDigital();
16      delete juego;
17    }
18    catch(...){
19      throw "Hay un error en el inicio del juego.";
20    }
21  }
```

7.3. Casillero

Listing 3: Casillero.cpp

```
1   #include "Casillero.h"
2   #include "Constantes.h"
3   #include <iostream>
4   Casillero::Casillero() {
5     this->ficha = NULL;
6     this->estado = CASILLERO_DESBLOQUEADO;
7     this->terreno = TIERRA;
8
9     this->casillerosAdyacentes = new Casillero ***[3];
10    for (int i = 0; i < 3; i++) {
11      this->casillerosAdyacentes[i] = new Casillero **[3];
12      for (int j = 0; j < 3; j++) {
13        this->casillerosAdyacentes[i][j] = new Casillero *[3];
14      }
15    }
16  }
```

```
17
18  Casillero::Casillero(Ficha *nuevaFicha) {
19
20    this->ficha = nuevaFicha;
21    this->estado = CASILLERO_DESBLOQUEADO;
22
23    this->casillerosAdyacentes = new Casillero ***[3];
24    for (int i = 0; i < 3; i++) {
25      this->casillerosAdyacentes[i] = new Casillero **[3];
26      for (int j = 0; j < 3; j++) {
27        this->casillerosAdyacentes[i][j] = new Casillero *[3];
28      }
29    }
30  }
31
32  Casillero::~Casillero() {
33
34    if (this->ficha) {
35      delete this->ficha;
36    }
37
38    for (int i = 0; i < 3; i++) {
39      for (int j = 0; j < 3; j++) {
40        delete[] this->casillerosAdyacentes[i][j];
41      }
42      delete[] this->casillerosAdyacentes[i];
43    }
44    delete[] this->casillerosAdyacentes;
45  }
46  EstadoCasillero Casillero::getEstado() { return estado; };
47
48  void Casillero::asignarCasilleroAdyacente(int x, int y, int z,
49                                            Casillero *
                                                casilleroAdyacente) {
50    if ((x < -1 || x > 1) || (y < -1 || y > 1) || (z < -1 || z > 1)) {
51      throw "Coordenadas invalidas. Se toma el mismo casillero como
            origen por "
52            "lo tanto"
53            "las coordenadas deben estan comprendidas entre -1 y 1";
54    }
55
56    this->casillerosAdyacentes[x + 1][y + 1][z + 1] = casilleroAdyacente
        ;
57  }
58
59  bool Casillero::tieneAdyacente(unsigned int x, unsigned int y,
      unsigned int z) {
60    // Puede que este mal
61    return (this->casillerosAdyacentes[x][y][z] != NULL);
62  }
63
64  bool Casillero::esAdyacenteLineal(Casillero *casillero) {
65
66    Casillero *ca;
67    for (int k = 0; k < 3; ++k) {
68      for (int j = 0; j < 3; ++j) {
69        for (int i = 0; i < 3; ++i) {
70          ca = this->getAdyacente(i, j, k);
```

```
71          if (ca == casillero) {
72            if ((k == 1 && j == 1 && i != 1) || (k == 1 && j != 1 && i
                  == 1) ||
73                (k != 1 && j == 1 && i == 1)) {
74              return true;
75            }
76          }
77        }
78      }
79    }
80    return false;
81  }
82
83  Casillero *Casillero::getAdyacente(unsigned int i, unsigned int j,
84                                     unsigned int k) {
85    Casillero *c = this->casillerosAdyacentes[i][j][k];
86    return c;
87  }
88
89  unsigned int Casillero::getLongitudFichasIguales(unsigned int i,
        unsigned int j,
90                                                   unsigned int k) {
91
92    if (!this->tieneAdyacente(i, j, k) || !this->ficha ||
93        (i == 1 && j == 1 && k == 1)) {
94      return 0;
95    }
96
97    Casillero *casilleroAdyacente = this->getAdyacente(i, j, k);
98
99    try {
100     if (this->tienenMismaFicha(casilleroAdyacente)) {
101       return (1 + casilleroAdyacente->getLongitudFichasIguales(i, j, k
              ));
102     }
103   } catch (...) {
104   }
105
106   return 0;
107 }
108
109 Ficha *Casillero::getFicha() {
110   return this->ficha;
111 }
112
113 void Casillero::setFicha(Ficha *nuevaFicha) {
114   if (this->ficha || this->estaBloqueado()) {
115     throw("No se puede poner una ficha en el casillero ocupado o
            bloqueado");
116   }
117
118   this->ficha = nuevaFicha;
119 }
120
121 void Casillero::eliminarFicha() {
122
123   if (!(this->ficha)) {
124     throw("No hay ficha para quitar");
```

```
125     }
126     delete this->ficha;
127     ficha = NULL;
128   }
129
130   bool Casillero::estaBloqueado() {
131     return (this->estado == CASILLERO_BLOQUEADO);
132   }
133   bool Casillero::estaEnvenenado() { return estado ==
        CASILLERO_ENVENENADO; }
134   void Casillero::bloquear() { this->estado = CASILLERO_BLOQUEADO; }
135
136   void Casillero::desbloquear() { this->estado = CASILLERO_DESBLOQUEADO;
        }
137
138   void Casillero::envenenar() { this->estado = CASILLERO_ENVENENADO; }
139
140   bool Casillero::tienenMismaFicha(Casillero *casilleroAdyacente) {
141
142     return (this->ficha->esIgual(casilleroAdyacente->getFicha()));
143   }
144
145   void Casillero::setContadorDeTurnos(int turnos) {
146     this->contadorDeTurno = turnos;
147   }
148
149   bool Casillero::estaVacio() {
150
151     if (this->ficha == NULL) {
152       return true;
153     }
154
155     return false;
156   }
157
158   void Casillero::bajarTurno() { this->contadorDeTurno--; }
159   int Casillero::getContadorDeturnos() { return this->contadorDeTurno; }
```

Listing 4: Casillero.h

```
1   #ifndef CASILLERO_H_
2   #define CASILLERO_H_
3
4   #include "Constantes.h"
5   #include "Ficha.h"
6
7   class Casillero {
8   private:
9     Ficha *ficha;
10    Casillero ****casillerosAdyacentes;
11    EstadoCasillero estado;
12    TipoTerreno terreno;
13    int contadorDeTurno;
14  public:
15    /*
16     * pre: ---
17     * Post: crea un Casillero con la Ficha vacia e inicializa
18     * sus 26 casillerosAdyacentes
```

```
19      */
20     Casillero();
21     /*
22      * pre: recibe un tipo de dato del tipo Ficha
23      * Post: crea un Casillero con la Ficha indicadas
24      */
25     Casillero(Ficha *ficha);
26
27     /*
28      * pre:---
29      * Post: libera toda la memoria solicitada para el Casillero
30      */
31     ~Casillero();
32     /*
33      * Pre: recibe las coordenadas respecto del mismo casillero tomando
            como
34      * referencia que el origen es la posicion del casillero.
35      * Post: asigna el casillero recibido como adyacente en la posicion
            recibida
36      * respecto del mismo casillero. Lanza una excepcion si las
            coordenadas son
37      * invÃąlidas
38      */
39     void asignarCasilleroAdyacente(int x, int y, int z,
40                                    Casillero *casilleroAdyacente);
41     /*
42      * Pre: que exista la clase casillero previamente, recibe las
            coordenadas
43      * respecto del mismo Casillero que se desea checkear.
44      * Post: devuelve true si el Casillero tiene un Casilllero adyacente
45      * existente ubicado en las coordenadas recibidas y false en caso de
            que no.
46      */
47     bool tieneAdyacente(unsigned int x, unsigned int y, unsigned int z);
48     /*
49      * pre : que exista la clase casillero previamente, recibe un tipo
            de dato
50      * Casillero. Post: devuelve verdadero si el casillero pasado por
            parametro es
51      * adyacente en linea con el casillero actual
52      */
53     bool esAdyacenteLineal(Casillero *);
54     /* Pre : que exista la clase casillero previamente.
55      * Post: devuelve la Ficha del Casillero
56      */
57     Ficha *getFicha();
58     /*
59      * pre: que exista la clase casillero previamente, Recibe un tipo de
            dato
60      * Ficha. Post: establece la Ficha indicada en el Casillero.
61      */
62     void setFicha(Ficha *nuevaFicha);
63     /*
64      * Pre: que exista la clase casillero previamente  y this->ficha !=
            NULL.
65      * Post: Eliminar ficha en memoria.
66      */
67     void eliminarFicha();
```

```
68      /*
69       * Pre: que exista la clase casillero previamente ,recibe
             coordenadas i,j,k.
70       * Post: devuelve el casillero adyacente que esta en la matriz de
             adyacentes.
71       */
72      Casillero *getAdyacente(unsigned int i, unsigned int j, unsigned int
             k);
73      /*
74       * pre: exista una instancia de casillero previamente.
75       * post: devuelve el estado del casillero.
76       */
77      EstadoCasillero getEstado();
78      /*
79       * Pre: que exista la clase casillero previamente, recibe
             coordenadas i,j,k.
80       * Post: devuelve un entero positivo que indica la cantidad de
             fichas iguales
81       * respecto de la posicion recibida
82       */
83      unsigned int getLongitudFichasIguales(unsigned int i, unsigned int j
            ,
84                                            unsigned int k);
85      /*
86       * Pre: Exista una instancia de la clase casillero,recibe una
             casillero
87       * valido. Post: devuelve verdadero si la ficha de ambos casilleros
             es igual.
88       */
89      bool tienenMismaFicha(Casillero *casilleroAdyacente);
90      /*
91       * Pre: Exista una instancia de la clase casillero.
92       * Post: devuelve True si estado = bloqueado.
93       */
94      bool estaBloqueado();
95      /*
96       * Pre: Exista una instancia de la clase casillero.
97       * Post: devuelve True si estado = envenenado.
98       */
99      bool estaEnvenenado();
100     /*
101      * Pre:  Exista una instancia de la clase casillero.
102      * Post: cambia estado a bloqueado
103      */
104     void bloquear();
105     /*
106      * Pre:  Exista una instancia de la clase casillero.
107      * Post: cambia estado a desbloqueado
108      */
109     void desbloquear();
110     /*
111      * pre:  Exista una instancia de la clase casillero.
112      * Post: devuelve verdadero si el casillero no posee una ficha
113      */
114     bool estaVacio();
115     /*
116      * pre:  Exista una instancia de la clase casillero, reciba una
             variable int.
```

```
117      * Post: cambia la cantidad de turnos en el casillero.
118      */
119     void setContadorDeTurnos(int turnos);
120     /*
121      * pre:  Exista una instancia de la clase casillero.
122      * Post: devuelve el contador de turnos del casillero.
123      */
124     int getContadorDeturnos();
125     /*
126      * pre:  Exista una instancia de la clase casillero.
127      * Post: baja en 1 el contador de turnos del casillero.
128      */
129     void bajarTurno();
130     /*
131      * pre:  Exista una instancia de la clase casillero.
132      * Post: cambia el estado del casillero a ENVENENADO.
133      */
134     void envenenar();
135     /*
136      * pre:  Exista una instancia de la clase casillero.
137      * Post: devuelve el tipo de terreno de el casillero.
138      */
139
140     TipoTerreno obtenerTerreno() const { return terreno; }
141     /*
142      * pre:  Exista una instancia de la clase casillero.
143      * Post: asigna el tipo de terreno al casillero.
144      */
145
146     void asignarTerreno(TipoTerreno nuevoTerreno) { terreno =
            nuevoTerreno; }
147 };
148 #endif /* CASILLERO_H_ */
```

7.4. Tablero

Listing 5: Tablero.cpp

```
1  #include "Tablero.h"
2
3  Tablero::Tablero(unsigned int x, unsigned int y, unsigned int z) {
4    this->dimensiones[0] = x;
5    this->dimensiones[1] = y;
6    this->dimensiones[2] = z;
7
8    this->casilleros = new Lista<Lista<Lista<Casillero *> *> *>();
9
10   for (unsigned int k = 0; k < z; k++) {
11     Lista<Lista<Casillero *> *> *fila = new Lista<Lista<Casillero *>
           *>();
12
13     for (unsigned int j = 0; j < y; j++) {
14       Lista<Casillero *> *columna = new Lista<Casillero *>();
15
16       for (unsigned int i = 0; i < x; i++) {
17         Casillero *casillero = new Casillero();
```

```
18
19          if (k < 5) {
20            // Los niveles del 1 al 5 tienen casilleros de tierra y agua
21            // intercalados
22            if ((i + j + k) % 3 != 0) {
23              casillero->asignarTerreno(TIERRA);
24            } else {
25              casillero->asignarTerreno(AGUA);
26            }
27          } else {
28            // Los niveles superiores representan el aire
29            casillero->asignarTerreno(AIRE);
30          }
31
32          columna->altaFinal(casillero);
33        }
34
35        fila->altaFinal(columna);
36      }
37
38      this->casilleros->altaFinal(fila);
39    }
40
41    for (unsigned int i = 0; i < x; i++) {
42      for (unsigned int j = 0; j < y; j++) {
43        for (unsigned int k = 0; k < z; k++) {
44          Casillero *casillero = this->getCasillero(i, j, k);
45
46          for (int l = -1; l < 2; l++) {
47            for (int m = -1; m < 2; m++) {
48              for (int n = -1; n < 2; n++) {
49                if (this->existeCasillero(i + l, j + m, k + n)) {
50                  Casillero *casilleroAdyacente =
51                      this->getCasillero(i + l, j + m, k + n);
52                  casillero->asignarCasilleroAdyacente(l, m, n,
53                                                       casilleroAdyacente
                                                           );
54                } else {
55                  casillero->asignarCasilleroAdyacente(l, m, n, NULL);
56                }
57              }
58            }
59          }
60        }
61      }
62    }
63  }
64
65  Tablero::~Tablero() {
66
67    for (int k = 0; k < this->dimensiones[2]; k++) {
68      for (int j = 0; j < this->dimensiones[1]; j++) {
69        for (int i = 0; i < this->dimensiones[0]; i++) {
70          delete this->getCasillero(i, j, k);
71        }
72        delete this->casilleros->obtener(k + 1)->obtener(j + 1);
73      }
74      delete this->casilleros->obtener(k + 1);
```

```
75    }
76    delete this->casilleros;
77  }
78
79  int *Tablero::getDimensiones() { return this->dimensiones; }
80
81  bool Tablero::existeCasillero(int x, int y, int z) {
82    return (x >= 0 && x < this->getDimensiones()[0] && y >= 0 &&
83            y < this->getDimensiones()[1] && z >= 0 &&
84            z < this->getDimensiones()[2]);
85  }
86
87  Casillero *Tablero::getCasillero(unsigned int x, unsigned int y,
88                                   unsigned int z) {
89
90    return this->casilleros->obtener(z + 1)->obtener(y + 1)->obtener(x +
          1);
91  }
```

Listing 6: Tablero.h

```
1   #ifndef TABLERO_H_
2   #define TABLERO_H_
3
4
5   #include "Lista.h"
6   #include "Casillero.h"
7   #include "Ficha.h"
8
9   class Tablero{
10  private:
11      Lista<Lista<Lista<Casillero *> *> *> * casilleros;
12      int dimensiones[3];
13
14  public:
15      /*
16       * Pre: recibe las longitudes de las tres dimensiones con las que
             se desea crear el tablero
17       * Post: inicializa un tablero de x ancho, y alto, z profundidad
             de casilleros vacios y
18       * asigna sus respectivos casilleros adyacentes
19       */
20      Tablero(unsigned int x, unsigned int y, unsigned int z);
21
22      /*
23       * pre: ---
24       * Post: libera la memoria solicitada para la creacion del tablero
25       */
26      ~Tablero();
27
28      /*
29       * pre: ---
30       * Post: devuelve un puntero a las dimensiones del Tablero
31       */
32      int * getDimensiones();
33
34      /*
35       * Pre: recibe las coordenadas x,y,z del Casillero que se desea
```

```
            verificar
36          * Post: devuelve true si existe un Casillero en el Tablero
                ubicado en las
37          * coordenadas recibidas, devuelve false si no existe y lanza una
                excepcion
38          * si las coordenadas son invÃąlidas
39          * quito el unsigned int porque chequea numeros negativos
40          */
41         bool existeCasillero(int x, int y, int z);
42
43         /*
44          * Pre: recibe las coordenadas x,y,z del casillero que se desea
                obtener
45          * Post: devuelve el casillero que se encuentra en la coordenada
                recibida
46          */
47         Casillero * getCasillero(unsigned int x, unsigned int y, unsigned
               int z);
48
49     };
50
51
52
53     #endif /* TABLERO_H_ */
```

7.5. Carta

Listing 7: Carta.cpp

```
1   #include "Carta.h"
2
3   std::string funcionalidades[6] = {
4       "Contamina 125 casilleros por 10 turnos en el centro, 8 turnos el
            siguiente radio y asi",
5       "Detecta minas en cada turno",
6       "Si esta en el agua, puede disparar un misil una vez por turno",
7       "Se incorpora un soldado",
8       "Puede tirar 3 minas por el turno en el que se usa",
9       "Permite seleccionar una ficha e identificar si es de un enemigo"
            };
10
11  Carta::Carta(funcion_t funcion) { this->funcion = funcion; }
12
13  funcion_t Carta::getFuncion() { return this->funcion; }
14
15  std::string Carta::getDescripcion() { return funcionalidades[this->
        funcion]; }
```

Listing 8: Carta.h

```
1   #ifndef CARTA_H_
2   #define CARTA_H_
3   #include <string>
4
5   typedef enum {
6       ATAQUE_QUIMICO,
```

```
7       AVION_RADAR,
8       BARCO_MISIL,
9       REFUERZO,
10      BOMBARDEO,
11      ESPIONAJE
12  } funcion_t;
13
14  class Carta {
15  private:
16      funcion_t funcion;
17
18  public:
19      /*
20       *pre: recibe un tipo de dato funcion_t.
21       *post: crea una instancia de la clase carta, esta debe recibir un
             tipo funcion_t valido.
22       */
23      Carta(funcion_t funcion);
24
25      /*
26       *pre: que exista previamente una instancia de la clase carta.
27       *post: devuelve un tipo funcion_t que indica el tipo de carta que
             es.
28       */
29      funcion_t getFuncion();
30
31      /*
32       *pre: que exista previamente una instancia de la clase carta.
33       *post: duelve una descripcion de la funcion de la carta.
34       */
35      std::string getDescripcion();
36  };
37
38  #endif /* CARTA_H_ */
```

7.6. Jugador

Listing 9: Jugador.cpp

```
1   #include "Jugador.h"
2   #include <iostream>
3
4   Jugador::Jugador(std::string nombreJugador) {
5     this->nombreJugador = nombreJugador;
6     this->fichas = new Pila<Ficha *>;
7     this->cartas = new Lista<Carta *>;
8     this->cantidadSoldados = 0;
9     this->contadorTurnos = 0;
10  }
11
12  Jugador::~Jugador() {
13    this->cartas->iniciarCursor();
14
15    while (this->cartas->avanzarCursor()) {
16      delete this->cartas->obtenerCursor();
17    }
```

```
18    while (!this->fichas->estaVacia()) {
19      delete this->fichas->desapilar();
20    }
21    delete this->cartas;
22    delete this->fichas;
23  }
24
25  std::string Jugador::getNombre() { return nombreJugador; }
26  void Jugador::setFichas(Pila<Ficha *> *nuevasFichas) {
27    // Limpia la pila de fichas existente
28    while (!fichas->estaVacia()) {
29      delete fichas->desapilar();
30    }
31    delete fichas;
32
33    // Asigna la nueva pila de fichas
34    fichas = nuevasFichas;
35  }
36  Ficha *Jugador::getFicha() {
37    if (fichas->estaVacia()) {
38      std::cout << "ESTA VACIA LA PILA" << std::endl;
39    }
40    return fichas->desapilar();
41  }
42  Lista<Carta *> *Jugador::getCartas() { return cartas; }
43
44  void Jugador::setCartas(Lista<Carta *> *nuevasCartas) {
45    this->cartas = nuevasCartas;
46  }
47  int Jugador::getTurnos() { return contadorTurnos; }
48
49  void Jugador::setTurnos(int nuevosTurnos) {
50    this->contadorTurnos = nuevosTurnos;
51  }
52
53  Pila<Ficha *> *Jugador::getPilaFichas() { return this->fichas; }
```

Listing 10: Jugador.h

```
1   #ifndef JUGADOR_H_
2   #define JUGADOR_H_
3   #include "Carta.h"
4   #include "Constantes.h"
5   #include "Ficha.h"
6   #include "Lista.h"
7   #include "Pila.h"
8   #include <string>
9
10  class Jugador {
11    friend class Ficha;
12  private:
13    std::string nombreJugador;
14    Pila<Ficha *> *fichas;
15    int cantidadSoldados;
16    Lista<Carta *> *cartas;
17    int contadorTurnos; // Cuando suma un turno, resta uno al
          contador_bloqueado
18                        // de cada ficha
```

```
19  public:
20    /*
21     *Pre: recibe un string, una ficha valida y un entero mayor a 0
22     * Post: inicializar atributos, crea maso de carta en memoria
23     */
24    Jugador(std::string nombreJugador);
25    /*
26     * Pre :---
27     * Post: libera toda la memoria de la instancia jugador.
28     */
29    ~Jugador();
30    /*
31     * Pre :exista una instancia de jugador previamente.
32     * Post: devuelve el nombre del jugador.
33     */
34    std::string getNombre();
35    /*
36     * Pre :exista una instancia de jugador previamente.
37     * Post: devuelve Ficha.
38     */
39    Ficha *getFicha();
40    /*
41     * Pre :exista una instancia de jugador previamente, recibe un
           TipoDeFicha.
42     * Post: cambia el estadoDeFicha de una Ficha.
43     */
44    void setFicha(TipoDeFicha tipoDeFicha);
45    /*
46     * Pre :exista una instancia de jugador previamente.
47     * Post: devuelve la cantidad de fichas.
48     */
49    int getCantidadFichas();
50    /*
51     * Pre :exista una instancia de jugador previamente, recibe una
           cantidad.
52     * Post: cambia la cantidad de fichas.
53     */
54    void setCantidadFichas(int cantidad);
55    /*
56     * Pre :exista una instancia de jugador previamente, recibe una pila
           de fichas.
57     * Post: cambia las fichas del la pila de fichas.
58     */
59    void setFichas(Pila<Ficha *> *nuevasFichas);
60    /*
61     * Pre :exista una instancia de jugador previamente.
62     * Post: devuelve la pila de fichas
63     */
64    Pila<Ficha *> *getPilaFichas();
65    /*
66     * Pre :exista una instancia de jugador previamente.
67     * Post: devuelve la lista de cartas
68     */
69    Lista<Carta *> *getCartas();
70    /*
71     * Pre :exista una instancia de jugador previamente.
72     * Post: devuelve la ultima carta de la pila de cartas.
73     */
```

```
74    Carta *getUltimaCarta();
75    /*
76     * Pre :exista una instancia de jugador previamente.
77     * Post: cambia las cartas de la lista de cartas.
78     */
79    void setCartas(Lista<Carta *> *nuevasCartas);
80    /*
81     * Pre :exista una instancia de jugador previamente.
82     * Post: devuelve la cantidad de turnos.
83     */
84    int getTurnos();
85    /*
86     * Pre :exista una instancia de jugador previamente.
87     * Post: cambia la cantidad de turnos.
88     */
89    void setTurnos(int nuevosTurnos);
90  };
91  #endif /* JUGADOR.H */
```

7.7. Juego

Listing 11: Juego.cpp

```
1   #include "Juego.h"
2   #include "Casillero.h"
3   #include "Constantes.h"
4   #include "Ficha.h"
5   #include "Jugador.h"
6   #include "Pila.h"
7   #include "Tablero.h"
8
9   Juego::Juego() {
10    this->interfaz->mostrarPantallaInicial();
11    this->cantidadTurnosJuego = 0;
12    this->interfaz = new Interfaz();
13    this->hayGanador = NULL;
14    this->jugadorEnTurno = NULL;
15    this->turno = 0;
16    unsigned int cantidadJugadores = pedirCantidadJugadores();
17    unsigned int cantidadFichas = pedirCantidadFichas();
18    unsigned int cantidadCartas = pedirCantidadCartas();
19    this->cantidadDeFichas = cantidadFichas;
20
21    this->jugadores = new Lista<Jugador *>;
22    // mostrar Siguiente jugador
23    for (unsigned int i = 0; i < cantidadJugadores; i++) {
24      std::string nombre = pedirNombre(i + 1);
25
26      Pila<Ficha *> *fichas = new Pila<Ficha *>;
27      for (unsigned int j = 0; j < cantidadFichas; j++) {
28        Ficha *ficha = new Ficha(nombre);
29        fichas->apilar(ficha);
30      }
31
32      Jugador *nuevoJugador = new Jugador(nombre);
```

```
33        nuevoJugador->setFichas(fichas); // Actualiza la pila de fichas
              del jugador
34        this->jugadores->altaFinal(nuevoJugador);
35      }
36
37      // Setea jugador en turno.
38      this->jugadores->iniciarCursor();
39      this->jugadores->avanzarCursor();
40      this->jugadorEnTurno = this->jugadores->obtenerCursor();
41
42      unsigned int *dimensiones = new unsigned int[3];
43      pedirDimensionesJuego(cantidadJugadores, cantidadFichas, dimensiones
            );
44      this->tablero = new Tablero(dimensiones[0], dimensiones[1],
            dimensiones[2]);
45      delete[] dimensiones;
46
47      // Inicializa mazo de cartas (TDA Pila).
48      this->mazo = new Pila<Carta *>;
49      for (unsigned int i = 0; i < cantidadCartas; i++) {
50        funcion_t funcionalidad = getFuncionalidad(i % 6);
51
52        Carta *nuevaCarta = new Carta(funcionalidad);
53
54        this->mazo->apilar(nuevaCarta);
55      }
56      std::cout << "\033[1m\033[4mJuego configurado.\033[0m" << std::endl;
57    }
58
59    Juego::~Juego() {
60      delete this->interfaz;
61      delete this->hayGanador;
62
63      delete this->jugadorEnTurno;
64      delete this->jugadores;
65
66      delete this->tablero;
67
68      while (!this->mazo->estaVacia()) {
69        delete this->mazo->desapilar();
70      }
71
72      delete this->mazo;
73    }
74
75    bool Juego::determinarGanador() {
76      bool hayGanador;
77      if (this->hayGanador != NULL) {
78        hayGanador = true;
79        this->interfaz->mostrarGanador(this->hayGanador->getNombre());
80      } else {
81        hayGanador = false;
82      }
83      return hayGanador;
84    }
85
86    void Juego::cambiarDeJugadorActual() {
87      bool cursorAvanzado = this->jugadores->avanzarCursor();
```

```
88
89    if (cursorAvanzado) {
90      this->jugadorEnTurno = this->jugadores->obtenerCursor();
91    } else {
92      this->jugadores
93          ->iniciarCursor(); // Reiniciar el cursor al principio de la
                lista
94
95      if (this->jugadores->avanzarCursor()) {
96        this->jugadorEnTurno = this->jugadores->obtenerCursor();
97      } else {
98        // La lista estÃą vacÃŋa, no hay jugadores
99        this->jugadorEnTurno = NULL;
100     }
101   }
102   if (this->jugadorEnTurno != NULL) {
103     this->interfaz->mostrarJugadorEnTurno(this->jugadorEnTurno->
          getNombre());
104   } else {
105     std::cout << "No hay jugadores disponibles." << std::endl;
106   }
107 }
108
109 // FUNCIONES DE ENTRADA DE DATOS//
110 unsigned int Juego::pedirCantidadJugadores() {
111
112   int cantidadJugadores = 0;
113   bool cantidadValida = false;
114
115   while (!cantidadValida) {
116
117     this->interfaz->pedirCantidadJugadores();
118     try {
119       std::cin >> cantidadJugadores;
120       if (cantidadJugadores < 2) {
121         throw "Error Menos de 2 jugadores";
122       }
123       cantidadValida = true;
124     } catch (...) {
125       this->interfaz->ingresoInvalido();
126     }
127   }
128
129   return (unsigned int)cantidadJugadores;
130 }
131
132 unsigned int Juego::pedirCantidadFichas() {
133
134   int cantidadFichas = 0;
135   bool cantidadValida = false;
136
137   while (!cantidadValida) {
138     this->interfaz->pedirCantidadFichas();
139     try {
140       std::cin >> cantidadFichas;
141       if (cantidadFichas < 4) {
142         throw "Error menos de 4 fichas";
143       }
```

```
144         cantidadValida = true;
145       } catch (...) { // En caso de que ingrese un valor invÃąlido se le
              indica al
146                       // usuario que lo que ingreso es invÃąlido
147         this->interfaz->ingresoInvalido();
148       }
149     }
150
151     return (unsigned int)cantidadFichas;
152   }
153
154   std::string Juego::pedirNombre(int jugadorNumero) {
155     std::string nombre;
156     bool nombreValido = false;
157
158     while (!nombreValido) {
159       this->interfaz->pedirNombre(jugadorNumero);
160       try {
161         std::cin >> nombre;
162         if (nombre.length() > 10) {
163           throw("Nombre demasiado largo.");
164         }
165         nombreValido = true;
166       } catch (...) {
167         this->interfaz->ingresoInvalido();
168       }
169     }
170     return nombre;
171   }
172
173   void Juego::pedirDimensionesJuego(int cantidadJugadores, int
          cantidadFichas,
174                                     unsigned int *dimensiones) {
175     int ancho, alto, profundo;
176     bool diemensionesValidas = false;
177
178     while (!diemensionesValidas) {
179       this->interfaz->pedirDimensiones();
180       try {
181         std::cout << "Ancho:  \t";
182         std::cin >> ancho;
183
184         std::cout << "Alto:   \t";
185         std::cin >> alto;
186
187         std::cout << "Profundo:  \t";
188         std::cin >> profundo;
189
190         if (ancho < 1 || alto < 1 || profundo < 1 ||
191             ancho * alto * profundo <
192                 cantidadJugadores * cantidadFichas + cantidadJugadores)
                    {
193           throw "Dimesiones de juego invalidas o tablero demasiado chico
                  .";
194         }
195         diemensionesValidas = true;
196       } catch (...) {
197         this->interfaz->tableroChico();
```

```
198       }
199     }
200
201     dimensiones[0] = ancho;
202     dimensiones[1] = alto;
203     dimensiones[2] = profundo;
204   }
205
206   unsigned int Juego::pedirCantidadCartas() {
207
208     int cantidadCartas;
209     bool cantidad_valida = false;
210
211     while (!cantidad_valida) {
212       this->interfaz->pedirCantidadCartas();
213       try {
214         std::cin >> cantidadCartas;
215         if (cantidadCartas < 6) {
216           throw("La cantiadad mÃηnima de cartas es 6.");
217         }
218         cantidad_valida = true;
219       } catch (...) {
220         this->interfaz->ingresoInvalido();
221       }
222     }
223     return (unsigned int)cantidadCartas;
224   }
225
226   funcion_t Juego::getFuncionalidad(unsigned int indice) {
227
228     switch (indice) {
229     case ATAQUE_QUIMICO:
230       return ATAQUE_QUIMICO;
231     case AVION_RADAR:
232       return AVION_RADAR;
233     case BARCO_MISIL:
234       return BARCO_MISIL;
235     case REFUERZO:
236       return REFUERZO;
237     case BOMBARDEO:
238       return BOMBARDEO;
239     case ESPIONAJE:
240       return ESPIONAJE;
241     };
242
243     throw "Numero de carta no valido";
244   }
245
246   unsigned int Juego::soldadosDeJugadorEnTablero(Jugador *jugador) {
247
248     unsigned int contador = 0;
249     for (int k = 0; k < tablero->getDimensiones()[2]; k++) {
250       for (int j = 0; j < tablero->getDimensiones()[1]; j++) {
251         for (int i = 0; i < tablero->getDimensiones()[0]; i++) {
252           Casillero *casillero = tablero->getCasillero(i, j, k);
253           if (!casillero->estaVacio() &&
254               casillero->getFicha()->getIdentificadorDeJugador() ==
255                   jugador->getNombre()) {
```

```
256             contador++;
257           }
258         }
259       }
260     }
261     return contador;
262   }
263
264   /*
265    * Pre: el jugador elige una accion
266    * Post: ejecuta la accion y actualiza el tablero
267    */
268
269   /*
270    *Pre tiene que asignar un soldado
271    */
272   Jugador *Juego::validarSiHayGanador(Lista<Jugador *> *jugadores) {
273     if (jugadores == NULL) {
274       throw "La lista de jugadores esta vacia";
275     }
276     bool hayGanador = false;
277     int cantidadGanadores = 0;
278     Jugador *ganador = NULL;
279
280     jugadores->iniciarCursor();
281     while (jugadores->avanzarCursor()) {
282
283       if (this->soldadosDeJugadorEnTablero(jugadores->obtenerCursor())
              != 0) {
284         cantidadGanadores++;
285         ganador = jugadores->obtenerCursor();
286       }
287     }
288     if (hayGanador && cantidadGanadores == 1) {
289       return ganador;
290     }
291     return NULL;
292   }
293
294   /*
295    * Pre: Recibe una coordenada
296    * Post:
297    */
298   void Juego::matarFicha(Casillero *casillero) {
299
300     if (casillero == NULL) {
301       throw "El casillero estÃą vacÃo";
302     }
303     casillero->eliminarFicha();
304   }
305
306   void Juego::colocarArmamento(int x, int y, int z, Ficha *armamento) {
307     Jugador *jugador = this->jugadorEnTurno;
308     Casillero *casillero = this->tablero->getCasillero(x, y, z);
309     Ficha *fichaEnCasillero = casillero->getFicha();
310
311     try {
```

```
312       if (fichaEnCasillero != NULL && casillero->obtenerTerreno() ==
              TIERRA &&
313           (fichaEnCasillero->getIdentificadorDeJugador() !=
314                jugadorEnTurno->getNombre() ||
315            casillero->estaVacio()) &&
316           !casillero->estaBloqueado()) {
317
318         if (fichaEnCasillero->getTipoDeFicha() == SOLDADO) {
319           this->matarFicha(casillero);
320           delete armamento;
321
322         } else if (fichaEnCasillero->getTipoDeFicha() == MINA) {
323           casillero->bloquear();
324           this->matarFicha(casillero);
325
326         } else if (fichaEnCasillero->getTipoDeFicha() == ARMAMENTO) {
327           casillero->bloquear();
328           this->matarFicha(casillero);
329           delete armamento;
330         }
331       } else {
332         armamento->setTipoDeFicha(ARMAMENTO);
333         casillero->setFicha(armamento);
334       }
335       std::cout << "Armamento colocado." << std::endl;
336
337     } catch (...) {
338       this->interfaz->informarCasilleroNoDisponible();
339     }
340   }
341
342   void Juego::colocarMina(int x, int y, int z) {
343     Jugador *jugador = this->jugadorEnTurno;
344     Casillero *casillero = this->tablero->getCasillero(x, y, z);
345     Ficha *fichaEnCasillero = casillero->getFicha();
346     Ficha *mina = new Ficha(MINA, jugador->getNombre());
347     try {
348       if (fichaEnCasillero != NULL && casillero->obtenerTerreno() ==
              TIERRA &&
349           (fichaEnCasillero->getIdentificadorDeJugador() !=
350                jugadorEnTurno->getNombre() ||
351            casillero->estaVacio()) &&
352           !casillero->estaBloqueado()) {
353         if (fichaEnCasillero->getTipoDeFicha() == MINA) {
354           casillero->bloquear();
355           this->matarFicha(casillero);
356           casillero->setContadorDeTurnos(5);
357           delete mina;
358         } else {
359           // fichaEnCasillero es un soldado o armemento
360           casillero->bloquear();
361           this->matarFicha(casillero);
362           casillero->setContadorDeTurnos(5);
363           delete mina;
364         }
365
366       } else {
367         casillero->setFicha(mina);
```

```
368         std::cout << "Mina colocada." << std::endl;
369       }
370     } catch (...) {
371       this->interfaz->informarCasilleroNoDisponible();
372     }
373   }
374
375   void Juego::lanzarMisil(int x, int y, int z) {
376     Jugador *jugador = this->jugadorEnTurno;
377     Casillero *casillero = this->tablero->getCasillero(x, y, z);
378     Ficha *fichaEnCasillero = casillero->getFicha();
379
380     try {
381       if (fichaEnCasillero != NULL &&
382           (fichaEnCasillero->getTipoDeFicha() == SOLDADO ||
383            fichaEnCasillero->getTipoDeFicha() == ARMAMENTO)) {
384         this->matarFicha(casillero);
385         casillero->setContadorDeTurnos(3);
386         std::cout << "Se ha disparo un misil" << std::endl;
387       }
388     } catch (...) {
389       this->interfaz->informarCasilleroNoDisponible();
390     }
391   }
392
393   /*Pre: HAY QUE DESAPILAR LA PILA DE FICHAS Y CREAR EL SOLDADO ANTES DE
         LLAMAR
394    * A ESTA FUNCION Post:verifica y coloca el soldado.
395    */
396   void Juego::colocarSoldado(int x, int y, int z, Ficha *soldado) {
397     Jugador *jugador = this->jugadorEnTurno;
398     Casillero *casillero = this->tablero->getCasillero(x, y, z);
399     Ficha *fichaEnCasillero = casillero->getFicha();
400
401     try {
402       if (fichaEnCasillero != NULL &&
403           (fichaEnCasillero->getIdentificadorDeJugador() !=
404            jugadorEnTurno->getNombre()) &&
405           !casillero->estaBloqueado()) {
406
407         if (fichaEnCasillero->getTipoDeFicha() == SOLDADO) {
408           this->matarFicha(casillero);
409           delete soldado;
410         } else if (fichaEnCasillero->getTipoDeFicha() == MINA) {
411           casillero->bloquear();
412           this->matarFicha(casillero);
413         } else if (fichaEnCasillero->getTipoDeFicha() == ARMAMENTO) {
414           casillero->bloquear();
415           this->matarFicha(casillero);
416           delete soldado;
417         }
418       } else {
419         soldado->setTipoDeFicha(SOLDADO);
420         casillero->setFicha(soldado);
421         std::cout << "Soldado colocado." << std::endl;
422       }
423     }
424
```

```
425     catch (...) {
426       this->interfaz->informarCasilleroNoDisponible();
427     }
428   }
429
430   /* Pre: recibe una coordenada de origen y de destino.
431     Post: mueve el soldado a un casillero adyacente
432    */
433   void Juego::moverSoldado() {
434     int x1, y1, z1, x2, y2, z2;
435     bool movimientoValido = false;
436
437     while (!movimientoValido) {
438       std::cout << "Ingrese las coordenadas donde se \033[32mencuentre
              el soldado\033[0m." << std::endl;
439       this->interfaz->pedirCoordenadas(x1, y1, z1, this->tablero);
440       Jugador* jugador = this->jugadorEnTurno;
441       Casillero* casilleroOrigen = this->tablero->getCasillero(x1, y1,
              z1);
442       int *dim = this->tablero->getDimensiones();
443
444       if (casilleroOrigen->getFicha() != NULL && casilleroOrigen->
              getFicha()->getTipoDeFicha() == SOLDADO &&
445           casilleroOrigen->getFicha()->getIdentificadorDeJugador() ==
                  jugador->getNombre()) {
446
447         std::cout << "Ingrese las coordenadas de \033[32mdestino
                adyacente\033[0m del soldado." << std::endl;
448         this->interfaz->pedirCoordenadas(x2, y2, z2, this->tablero);
449
450         Casillero* casilleroDestino = this->tablero->getCasillero(x2, y2
                , z2);
451         if (casilleroDestino->esAdyacenteLineal(casilleroOrigen)) {
452           this->colocarSoldado(x2, y2, z2, casilleroOrigen->getFicha());
453           casilleroOrigen->setFicha(NULL);
454           std::cout << "\033[32mSoldado movido.\033[0m" << std::endl;
455           movimientoValido = true;
456         } else {
457           std::cout << "\033[31mLa coordenada de destino seleccionada no
                  es adyacente al soldado.\033[0m" << std::endl;
458
459           // Mostrar coordenadas adyacentes vÃąlidas
460           std::cout << "Coordenadas adyacentes vÃąlidas como ejemplo:";
461           if (x1 > 0) {
462             std::cout << " (" << x1 - 1 << ", " << y1 << ", " << z1 << "
                    )";
463           }
464           if (x1 < dim[0] - 1) {
465             std::cout << " (" << x1 + 1 << ", " << y1 << ", " << z1 << "
                    )";
466           }
467           if (y1 > 0) {
468             std::cout << " (" << x1 << ", " << y1 - 1 << ", " << z1 << "
                    )";
469           }
470           if (y1 < dim[1] - 1) {
471             std::cout << " (" << x1 << ", " << y1 + 1 << ", " << z1 << "
                    )";
```

```
472           }
473           if (z1 > 0) {
474             std::cout << " (" << x1 << ", " << y1 << ", " << z1 - 1 << "
                  )";
475           }
476           if (z1 < dim[2] - 1) {
477             std::cout << " (" << x1 << ", " << y1 << ", " << z1 + 1 << "
                  )";
478           }
479           std::cout << std::endl;
480         }
481
482       } else {
483         std::cout << "\033[31mNo hay un soldado del jugador actual en la
                casilla de origen seleccionada.\033[0m" << std::endl;
484       }
485
486     }
487   }
488
489
490
491   void Juego::moverArmamento() {
492     int x1, y1, z1, x2, y2, z2;
493     bool movimientoValido = false;
494
495     while (!movimientoValido) {
496       std::cout << "Ingrese las coordenadas donde se \033[32mencuentre
            el armamento\033[0m." << std::endl;
497       this->interfaz->pedirCoordenadas(x1, y1, z1, this->tablero);
498       Jugador* jugador = this->jugadorEnTurno;
499       Casillero* casilleroOrigen = this->tablero->getCasillero(x1, y1,
            z1);
500       int *dim = this->tablero->getDimensiones();
501
502       if (casilleroOrigen->getFicha() != NULL && casilleroOrigen->
            getFicha()->getTipoDeFicha() == ARMAMENTO &&
503           casilleroOrigen->getFicha()->getIdentificadorDeJugador() ==
                jugador->getNombre()) {
504
505         std::cout << "Ingrese las coordenadas de \033[32mdestino
              adyacente\033[0m del armamento." << std::endl;
506         this->interfaz->pedirCoordenadas(x2, y2, z2, this->tablero);
507
508         Casillero* casilleroDestino = this->tablero->getCasillero(x2, y2
              , z2);
509         if (casilleroDestino->esAdyacenteLineal(casilleroOrigen)) {
510           this->colocarArmamento(x2, y2, z2, casilleroOrigen->getFicha()
                );
511           casilleroOrigen->setFicha(NULL);
512           std::cout << "\033[32mArmamento movido.\033[0m" << std::endl;
513           movimientoValido = true;
514         } else {
515           std::cout << "\033[31mLa coordenada de destino seleccionada no
                es adyacente al armamento.\033[0m" << std::endl;
516
517           // Mostrar coordenadas adyacentes vÃąlidas
518           std::cout << "Coordenadas adyacentes vÃąlidas como ejemplo:";
```

```
519           if (x1 > 0) {
520             std::cout << " (" << x1 - 1 << ", " << y1 << ", " << z1 << "
                    )";
521           }
522           if (x1 < dim[0] - 1) {
523             std::cout << " (" << x1 + 1 << ", " << y1 << ", " << z1 << "
                    )";
524           }
525           if (y1 > 0) {
526             std::cout << " (" << x1 << ", " << y1 - 1 << ", " << z1 << "
                    )";
527           }
528           if (y1 < dim[1] - 1) {
529             std::cout << " (" << x1 << ", " << y1 + 1 << ", " << z1 << "
                    )";
530           }
531           if (z1 > 0) {
532             std::cout << " (" << x1 << ", " << y1 << ", " << z1 - 1 << "
                    )";
533           }
534           if (z1 < dim[2] - 1) {
535             std::cout << " (" << x1 << ", " << y1 << ", " << z1 + 1 << "
                    )";
536           }
537           std::cout << std::endl;
538         }
539
540       } else {
541         std::cout << "\033[31mNo hay un armamento del jugador actual en
                la casilla de origen seleccionada.\033[0m" << std::endl;
542       }
543
544     }
545   }
546
547
548
549
550
551   /* Pre:
552     Post: Agrega una carta al mazo del jugador (lista)
553    */
554
555   void Juego::sacarCartaDeMazo(Jugador *jugador) {
556
557     try {
558       if (!mazo->estaVacia()) {
559         Carta *carta = mazo->desapilar();
560         jugador->getCartas()->altaFinal(carta);
561       }
562     } catch (...) {
563       this->interfaz->mazoSinCartas();
564     }
565     std::cout << "Ha sacado una carta del mazo." << std::endl;
566   }
567   void Juego::ataqueQuimico() {
568     try {
569       int x = 0, y = 0, z = 0;
```

```
570       this->interfaz->pedirCoordenadas(x, y, z, this->tablero);
571       Casillero *casilleroAux = this->tablero->getCasillero(x, y, z);
572       // itera el tablero
573       for (int i = 0; i < 5; i++) {
574         for (int j = 0; j < 5; j++) {
575           for (int k = 0; k < 5; k++) {
576             // Envenana los adyacentes
577             Casillero *casilleroAux = this->tablero->getCasillero(x - i,
                      y - j, z - k);
578             casilleroAux->envenenar();
579             // El origen de coordenadas
580
581             if (i == 0 && j == 0 && k == 0) {
582               casilleroAux->setContadorDeTurnos(10);
583             } else if (i == 1 || j == 1 || k == 1) {
584               casilleroAux->setContadorDeTurnos(8);
585             } else if ((i == 2 || j == 2 || k == 2)) {
586               casilleroAux->setContadorDeTurnos(6);
587             } else if ((i == 3 || j == 3 || k == 3)) {
588               casilleroAux->setContadorDeTurnos(4);
589             } else if ((i == 4 || j == 4 || k == 4)) {
590               casilleroAux->setContadorDeTurnos(2);
591             }
592           }
593         }
594       }
595       std::cout << "Ha realizado un ataque quÃŋmico." << std::endl;
596
597     } catch (...) {
598       this->interfaz->ingresoInvalido();
599     }
600   }
601   void Juego::colocarAvion() {
602     try {
603       int x = 0, y = 0, z = 0;
604       std::cout << "Coloque un aviÃşn" << std::endl;
605       bool coordenadaValida = false;
606       while (!coordenadaValida) {
607         this->interfaz->pedirCoordenadas(x, y, z, this->tablero);
608         Casillero *casillero = this->tablero->getCasillero(x, y, z);
609         Ficha *fichaEnCasillero = casillero->getFicha();
610         if (fichaEnCasillero == NULL && casillero->obtenerTerreno() ==
                AIRE &&
611             !casillero->estaBloqueado()) {
612           Jugador *jugador = this->jugadorEnTurno;
613           Ficha *avion = new Ficha(AVION, jugador->getNombre());
614           casillero->setFicha(avion);
615           std::cout << "Avion colocado." << std::endl;
616           coordenadaValida = true;
617         } else {
618           std::cout << "Coordenada invÃąlida. El casillero no estÃą en
                  el aire o ya estÃą ocupado." << std::endl;
619         }
620       }
621     } catch (...) {
622       std::cout << "El aviÃşn no pudo ser colocado." << std::endl;
623     }
624
```

```
625 }
626
627 void Juego::colocarBarco() {
628   try {
629     int x = 0, y = 0, z = 0;
630     std::cout << "Coloque un barco" << std::endl;
631     bool coordenadaValida = false;
632     while (!coordenadaValida) {
633       this->interfaz->pedirCoordenadas(x, y, z, this->tablero);
634       Casillero *casillero = this->tablero->getCasillero(x, y, z);
635       Ficha *fichaEnCasillero = casillero->getFicha();
636       if (fichaEnCasillero == NULL && casillero->obtenerTerreno() ==
              AGUA &&
637           !casillero->estaBloqueado()) {
638         Jugador *jugador = this->jugadorEnTurno;
639         Ficha *barco = new Ficha(BARCO, jugador->getNombre());
640         casillero->setFicha(barco);
641         std::cout << "Barco colocado." << std::endl;
642         coordenadaValida = true;
643       } else {
644         std::cout << "Coordenada invÃąlida. El casillero no es de agua
                o ya estÃą ocupado." << std::endl;
645       }
646     }
647   } catch (...) {
648     std::cout << "El barco no pudo ser colocado." << std::endl;
649   }
650 }
651
652
653 void Juego::agregarTresMinas() {
654   std::cout << "Coloque un \033[1;32tres minas\033[0m" << std::endl;
655   for (int i = 0; i < 3; i++) {
656     try {
657       int x = 0, y = 0, z = 0;
658       this->interfaz->pedirCoordenadas(x, y, z, this->tablero);
659       this->colocarMina(x, y, z);
660     } catch (...) {
661       this->interfaz->ingresoInvalido();
662     }
663   }
664   std::cout << "Ha colocado las tres minas." << std::endl;
665 }
666
667 void Juego::identificarFichaEnCasillero() {
668   std::cout << "Identifique un \033[1;32casillero\033[0m" << std::endl
        ;
669   try {
670     int x = 0, y = 0, z = 0;
671
672     this->interfaz->pedirCoordenadas(x, y, z, this->tablero);
673     this->interfaz->mostrarFichaEnCasillero(tablero->getCasillero(x, y
          , z));
674
675   } catch (...) {
676     this->interfaz->ingresoInvalido();
677   }
678   std::cout << "El casillero se ha mostrado." << std::endl;
```

```
679 }
680
681 void Juego::agregarSoldado() {
682   std::cout << "Llame un \033[1;32refuerzo\033[0m. ";
683   std::cout << "Coloque  un \033[1;32soldado\033[0m" << std::endl;
684   try {
685     int x = 0, y = 0, z = 0;
686
687     this->interfaz->pedirCoordenadas(x, y, z, this->tablero);
688     this->colocarSoldado(x, y, z,
689                          new Ficha(SOLDADO, jugadorEnTurno->getNombre
                                 ()));
690
691   } catch (...) {
692     this->interfaz->ingresoInvalido();
693   }
694   std::cout << "Ha llamado un refuerzo." << std::endl;
695 }
696
697 void Juego::ejecutarCarta(unsigned int indice) {
698   switch (indice) {
699   case ATAQUE_QUIMICO:
700     this->ataqueQuimico();
701     break;
702   case AVION_RADAR:
703     this->colocarAvion();
704     break;
705   case BARCO_MISIL:
706     this->colocarBarco();
707     break;
708   case REFUERZO:
709     this->agregarSoldado();
710     break;
711   case BOMBARDEO:
712     this->agregarTresMinas();
713     break;
714   case ESPIONAJE:
715     this->identificarFichaEnCasillero();
716     break;
717   default:
718     throw "Numero de carta no valido";
719     break;
720   }
721 }
722
723 void Juego::usarCartaDeJugador(unsigned int posicionDeCarta) {
724   // try {
725   // Obtiene carta de jugador
726   Carta *carta = this->jugadorEnTurno->getCartas()->obtener(
          posicionDeCarta);
727   if (carta != NULL) {
728     this->ejecutarCarta(carta->getFuncion());
729     // Eliminar carta de jugador
730     this->jugadorEnTurno->getCartas()->remover(posicionDeCarta);
731   } else {
732     this->interfaz->jugadorSinCartas();
733   }
734   std::cout << "Carta tirada." << std::endl;
```

```
735 }
736
737 void Juego::inicializarFichas() {
738   // llevar a interfaz el mensaje
739   std::cout << "\033[1;32mInicializando fichas...\033[0m" << std::endl
          ;
740   unsigned int contador = 0;
741   // revisar logica por quÃ¨ jugador 2 de 2 no puede iniciarlizar
          fichas.
742   while (contador < this->jugadores->contarElementos()) {
743     Jugador *jugador = this->jugadorEnTurno;
744     this->interfaz->mostrarJugadorEnTurno(jugador->getNombre());
745     int n = this->cantidadDeFichas;
746     int cantidadSoldados = n * 0.75;   //
747     int cantidadArmamentos = n * 0.25; // 25% de las fichas
748     for (int i = 0; i < cantidadSoldados; i++) {
749       int x = 0, y = 0, z = 0;
750       Ficha *soldado = jugador->getFicha();
751       std::cout << "Coloque un \033[1;33msoldado\033[0m";
752       this->interfaz->pedirCoordenadas(x, y, z, this->tablero);
753       while (this->tablero->getCasillero(x, y, z)->obtenerTerreno() ==
              AGUA ||
754              this->tablero->getCasillero(x, y, z)->obtenerTerreno() ==
                  AIRE) {
755         this->interfaz->pedirCoordenadas(x, y, z, this->tablero);
756       }
757       this->colocarSoldado(x, y, z, soldado);
758     }
759     for (int in = 0; in < cantidadArmamentos; in++) {
760       int x2 = 0, y2 = 0, z2 = 0;
761       Ficha *armamento = jugador->getFicha();
762       std::cout << "Coloque un \033[1;32marmamento\033[0m";
763       this->interfaz->pedirCoordenadas(x2, y2, z2, this->tablero);
764       this->colocarArmamento(x2, y2, z2, armamento);
765     }
766     contador++;
767     this->cambiarDeJugadorActual();
768     this->interfaz->limpiarPantalla();
769   }
770   std::cout << "\033[1;31mFichas inicializadas.\033[0m\n" << std::endl
          ;
771   this->interfaz->limpiarPantalla();
772 }
773
774 void Juego::bajarContadorCasilleros() {
775   int *dim = this->tablero->getDimensiones();
776   for (int k = 0; k < dim[2]; k++) {
777     for (int j = 0; j < dim[1]; j++) {
778       for (int i = 0; i < dim[0]; i++) {
779         if (this->tablero->getCasillero(i, j, k)->getContadorDeturnos
                () > 0) {
780           this->tablero->getCasillero(i, j, k)->bajarTurno();
781         }
782       }
783     }
784   }
785 }
786
```

```
787 bool Juego::hayBarco(Jugador *jugador) {
788   int *dim = this->tablero->getDimensiones();
789   for (int k = 0; k < dim[2]; k++) {
790     for (int j = 0; j < dim[1]; j++) {
791       for (int i = 0; i < dim[0]; i++) {
792 
793         Ficha *fichaAux = this->tablero->getCasillero(i, j, k)->
              getFicha();
794         if ((fichaAux != NULL) && (fichaAux->getTipoDeFicha() == BARCO
              ) &&
795             (fichaAux->getIdentificadorDeJugador() == jugador->
                 getNombre())) {
796           return true;
797         }
798       }
799     }
800   }
801   return false;
802 }
803 
804 void Juego::ejecutarRadar(Jugador *jugador) {
805   int *dim = this->tablero->getDimensiones();
806   for (int k = 0; k < dim[2]; k++) {
807     for (int j = 0; j < dim[1]; j++) {
808       for (int i = 0; i < dim[0]; i++) {
809 
810         Ficha *fichaAux = this->tablero->getCasillero(i, j, k)->
              getFicha();
811         if ((fichaAux != NULL) && (fichaAux->getTipoDeFicha() == AVION
              ) &&
812             (fichaAux->getIdentificadorDeJugador() == jugador->
                 getNombre())) {
813           for (int z = k - 3; z < dim[2] - k; z++) {
814             for (int y = 0; y < 3; y++) {
815               for (int x = 0; x < 3; x++) {
816                 Casillero *casillero = this->tablero->getCasillero(i -
                      x,j- y,z);
817                 if (casillero->getFicha() != NULL &&
818                     casillero->getFicha()->getIdentificadorDeJugador()
                          ==
819                         jugador->getNombre()) {
820                   std::cout << "La mina estÃą en la posiciÃşn: X: " <<
                        i
821                             << ", Y: " << j << " , Z:" << k << std::
                                 endl;
822                 }
823               }
824             }
825           }
826         }
827       }
828     }
829   }
830 }
831 
832 void Juego::jugarBatallaDigital() {
833   this->interfaz->limpiarPantalla();
834   this->inicializarFichas();
```

```
835     while (this->estadoActual != FINALIZADO) {
836       this->interfaz->mostrarBatallaIniciada();
837       int x0 = 0, y0 = 0, z0 = 0, x1 = 0, y1 = 0, z1 = 0;
838       this->sacarCartaDeMazo(this->jugadorEnTurno);
839 
840       // Obtiene el nro de la carta y la ejecuta
841       this->interfaz->mostrarCartasJugador(this->jugadorEnTurno);
842       this->interfaz->pedirUsarCarta();
843       char opcionCarta;
844       std::cin >> opcionCarta;
845       if (opcionCarta == 'S' || opcionCarta == 's') {
846         unsigned int nroCarta =
847             this->interfaz->pedirNroCarta(this->jugadorEnTurno);
848         this->usarCartaDeJugador(nroCarta);
849       }
850 
851       this->interfaz->indicarColocarMina();
852       this->interfaz->pedirCoordenadas(x1, y1, z1, this->tablero);
853       this->colocarMina(x1, y1, z1);
854 
855       this->interfaz->indicarMoverSolado();
856       char opcionSoldado;
857       std::cin >> opcionSoldado;
858       if (opcionSoldado == 'S' || opcionSoldado == 's') {
859         this->moverSoldado();
860       }
861 
862       this->interfaz->indicarMoverArmamento();
863       char opcionArmamento;
864       std::cin >> opcionArmamento;
865       if (opcionArmamento == 'S' || opcionArmamento == 's') {
866         this->moverArmamento();
867       }
868       if (this->hayBarco(this->jugadorEnTurno)) {
869         std::cout << "Tiene un barco, puede lanzar un misil." << endl;
870         this->interfaz->pedirCoordenadas(x0, y0, z0, this->tablero);
871         this->lanzarMisil(x0, y0, z0);
872       }
873       this->ejecutarRadar(this->jugadorEnTurno);
874 
875       this->interfaz->mostrarTableroDeJugadorBitMap(this->tablero,
876                                                     this->
                                                         jugadorEnTurno)
                                                         ;
877 
878       this->interfaz->limpiarPantalla();
879 
880       this->cambiarDeJugadorActual();
881 
882       this->turno++;
883       this->hayGanador = this->validarSiHayGanador(this->jugadores);
884       this->estadoActual = this->determinarGanador() ? FINALIZADO :
             ENCURSO;
885     }
886     this->interfaz->mostrarGanador(this->hayGanador->getNombre());
887     this->interfaz->mostrarFinDelJuego();
888 
889   }
```

Listing 12: Juego.h

```
#ifndef JUEGO_H_
#define JUEGO_H_

#include "Carta.h"
#include "Cola.h"
#include "Constantes.h"
#include "Ficha.h"
#include "Interfaz.h"
#include "Jugador.h"
#include "Tablero.h"
#include <iostream>
#include <string>

typedef enum { ENCURSO, FINALIZADO } Estado;
class Juego {

private:
  Lista<Jugador *> *jugadores;
  Pila<Carta *> *mazo;
  Jugador *jugadorEnTurno;
  Jugador *hayGanador;
  Tablero *tablero;
  Interfaz *interfaz;
  unsigned int cantidadMaximaCartas;
  unsigned int cantidadTurnosJuego;
  Estado estadoActual;
  unsigned int cantidadDeFichas;
  int turno;

public:
  /*
   * pre: ---
   * post: crea una instancia de juego.
   */
  Juego();

  /*
   * pre: exista una instancia de juego
   * post: destruyo la instancia de juego liberando memoria.
   */
  ~Juego();

  /*
   * pre: exista una instancia de juego y reciba el numero de jugador
   * en el juego.
   * post: devuelve el nombre ddel jugador.
   */
  std::string pedirNombre(int jugadorNumero);

  /*
   * pre: exista una instancia de juego.
   * post: cambia el jugadorEnTurno y mueve el cursor de la lista de
       jugadores.
   */
  void cambiarDeJugadorActual();

  /*
```

```
57      * pre: exista una instancia de juego.
58      * post: aumenta el contador de turno en 1.
59      */
60     void cambiarDeTurno();
61
62     /*
63      * pre: exista una instancia de juego.
64      * post: determina si hay un ganador, viendo si el hayGanador es
            distinto de
65      * NULL.
66      */
67     bool determinarGanador();
68
69     /*
70      * Pre:Exista una clase juego previamente. recibe la lista de
            jugadores.
71      * Post: comprueba si solo una de los jugadores tiene soldado, y de
            ser
72      * verdadero devuelve al jugador ganador.
73      */
74     Jugador *validarSiHayGanador(Lista<Jugador *> *jugadores);
75
76     /*
77      * pre: exista una instancia de juego y reciba un casillero.
78      * post: si el casillero no esta vacio elimina a la ficha en ese
            casillero.
79      */
80     void matarFicha(Casillero *casillero);
81
82     /*
83      * pre: exista una instancia de juego, recibe las coordenadas x,y,z
            y una
84      * Ficha. post: coloca una ficha del tipo armamento en las
            coordenadas x,y,z.
85      */
86     void colocarArmamento(int x, int y, int z, Ficha *soldado);
87
88     /*
89      * pre: exista una instancia de juego, recibe las coordenadas x,y,z
            y una
90      * Ficha.
91      * post: coloca una ficha del tipo mina en las coordenadas x,y,z.
92      */
93     void colocarMina(int x, int y, int z);
94
95     /*
96      * pre: exista una instancia de juego, recibe las coordenadas x,y,z.
97      * post: lanza un misil en la posicion x,y,z eliminando si hay
            soldados o
98      * armamento.
99      */
100    void lanzarMisil(int x, int y, int z);
101
102    /*
103     * pre: exista una instancia de juego, recibe las coordenadas x,y,z
            y una
104     * Ficha. post: coloca una ficha del tipo soldado en las coordenadas
105     * x,y,z.
```

```
106      */
107     void colocarSoldado(int x, int y, int z, Ficha *soldado);
108
109     /*
110      * pre: exista una instancia de juego, recibe las coordenadas
111      * x1,y1,z1,x2,y2,z2. post: si es una ficha del tipo soldado lo
           mueve desde la
112      * posicion incial x1,y1,z1 a la posicion final x2,y2,z2.
113      */
114     void moverSoldado();
115
116     /*
117      * pre: exista una instancia de juego, recibe las coordenadas
118      * x1,y1,z1,x2,y2,z2. post: si es una ficha del tipo armamento lo
           mueve desde
119      * la posicion incial x1,y1,z1 a la posicion final x2,y2,z2.
120      */
121     void moverArmamento();
122
123     /*
124      * pre: exista una instancia de juego, recibe un jugador.
125      * post: saca una carta del mazo y la agrega a la lista de cartas
           del jugador.
126      */
127     void sacarCartaDeMazo(Jugador *jugador);
128
129     /*
130      * pre: exista una instancia de juego, recibe un numero de carta.
131      * post: dependiendo del numero de carta indicada, busca en la lista
            de cartas
132      * del jugador y aplica la funcion de esa carta.
133      */
134     void usarCartaDeJugador(unsigned int posicionDeCarta);
135
136     /*
137      * pre: exista una instancia de juego, recibe un jugador.
138      * post: devuelve un el numero de carta a usar por el jugador.
139      */
140     unsigned int pedirNroCarta(Jugador *jugador);
141
142     /*
143      * pre: exista una instancia de juego
144      * post: aplica la funcion de la carta ataque quimico.
145      */
146     void ataqueQuimico();
147
148     /*
149      * pre: exista una instancia de juego
150      * post: aplica la funcion de la carta avion radar.
151      */
152     void colocarAvion();
153
154     /*
155      * pre: exista una instancia de juego
156      * post: aplica la funcion de la carta barco misil.
157      */
158     void colocarBarco();
159
```

```
160     /*
161      * pre: exista una instancia de juego
162      * post: aplica la funcion la carta bormbardeo.
163      */
164     void agregarTresMinas();
165 
166     /*
167      * pre: exista una instancia de juego
168      * post: aplica la funcion de la carta Refuerzos
169      */
170     void agregarSoldado();
171 
172     /*
173      * pre: exista una instancia de juego
174      * post: aplica la funcion de la carta espionaje.
175      */
176     void identificarFichaEnCasillero();
177 
178     /*
179      * pre: exista una instancia de juego
180      * post: baja el contador de turnos en el casillero.
181      */
182     void bajarContadorCasilleros();
183 
184     /*
185      * pre: exista una instancia de juego
186      * post: Inicializa el tablero colocando las fichas en los
            casilleros
187      * correspondientes.
188      */
189     void inicializarFichas();
190 
191     /*
192      * pre: exista una instancia de juego, recibe un jugador.
193      * post: duelve la cantidad de soldados que hay en el tablero de ese
            jugador.
194      */
195     unsigned int soldadosDeJugadorEnTablero(Jugador *jugador);
196 
197     /*
198      * pre: exista una instancia de juego
199      * post: devuelve la cantidad de jugadores que van a haber en el
            juego.
200      */
201     unsigned int pedirCantidadJugadores();
202 
203     /*
204      * pre: exista una instancia de juego
205      * post: devuelve la cantidad de fichas a usar en el juego.
206      */
207     unsigned int pedirCantidadFichas();
208 
209     /*
210      * pre: exista una instancia de juego, recibe la cantidad de
            jugadores, fichas
211      * y las dimensiones del tablero
212      * post: pide por pantalla la cantidad de jugadores,fichas y la
            dimension del
```

```
213      * tablero.
214      */
215     void pedirDimensionesJuego(int cantidadJugadores, int cantidadFichas
            ,
216                                unsigned int *dimensiones);
217
218     /*
219      * pre: exista una instancia de juego
220      * post: devuelve la cantidad de cartas a usar en el juego.
221      */
222     unsigned int pedirCantidadCartas();
223
224     /*
225      * pre: exista una instancia de juego, pide un indice.
226      * post: devuelve el tipo de funcion de la carta dependiendo del
            indice.
227      */
228     funcion_t getFuncionalidad(unsigned int indice);
229
230     /*
231      * pre: exista una instancia de juego, recibe un jugador y un tipo
            de ficha.
232      * post: devuelve una Ficha con indentidad de ese jugador y el
            tipoDeFicha
233      * seleccionado.
234      */
235     Ficha *ponerFicha(Jugador *jugador, TipoDeFicha tipo);
236
237     /*
238      * pre: exista una instancia de juego, recibe un indice.
239      * post: dependiendo del indice seleccionado hace uno de la
            funcionalidad
240      * de esa carta.
241      */
242     void ejecutarCarta(unsigned int indice);
243
244     /*
245      * pre: exista una instancia de juego, recibe un jugador.
246      * post: devuelve true o false si encuentra un barco del jugador.
247      */
248     bool hayBarco(Jugador *jugador);
249
250     /*
251      * pre: exista una instancia de juego, recibe un jugador.
252      * post: devuelve true o false si encuentra un avion del jugador.
253      */
254     bool hayAvion(Jugador *jugador);
255
256     /*
257      * pre: exista una instancia de juego.
258      * post: ejecuta el radar del avion.
259      */
260     void ejecutarRadar(Jugador *jugador);
261
262     /*
263      * pre: exista una instancia de juego.
264      * post: desarrolla toda la logica del juego batalla digital.
265      */
```

```
266     void jugarBatallaDigital();
267   };
268
269
270
271   #endif /* JUEGO_H_ */
```

7.8. Interfaz

Listing 13: Interfaz.cpp

```
1   #include "Interfaz.h"
2   #include "Casillero.h"
3   #include "Constantes.h"
4   #include "EasyBMP/EasyBMP.h"
5   #include "Jugador.h"
6   #include "Nodo.h"
7   #include "Tablero.h"
8   #include <iostream>
9
10  void Interfaz::mostrarPantallaInicial() {
11    std::cout << "\n\n\033[1;36m
        ================================================="
12                "==============="
13                "==============================\n\n";
14    std::cout << "\033[1;36m\n\t\t\t\t BATALLA DIGITAL V2.0      \033[0m
        "
15              << std::endl;
16    std::cout
17        << "\n\t\033[1mBienvenido a BATALLA DIGITAL 2.0, una versiÃ§n
            alternativa"
18           " de la BATALLA DIGITAL convencional \n\tcon un tablero en 3
               "
19           "dimensiones"
20           " y varios jugadores y cartas para jugar!\033[0m"
21        << std::endl;
22    std::cout << "\n\033[1mProyecto desarrollado por alumnos de FIUBA
        :\033[0m"
23              << std::endl;
24    std::cout
25        << "\nAmddy Zambrano, AgustÃŋn Allelo, Pablo Mokorel, Federico
            Neuman"
26        << std::endl;
27    std::cout << "\n\n\033[1;36m
        ================================================="
28                "==============="
29                "==============================\n\n";
30
31    this->limpiarPantalla();
32  }
33
34  void Interfaz::mostrarControles(Tablero *t) {
35    int dim[3];
36    dim[0] = t->getDimensiones()[0];
37    dim[1] = t->getDimensiones()[1];
38    dim[2] = t->getDimensiones()[2];
```

```
    cout << "\n\
        nÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂ
        "
            "Âů";
    cout << "
        ÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂů\n\n
        ";

    std::cout << "#DISPOSICION DEL TABLERO:\n\n";
    for (int k = 1; k < dim[2] + 1; ++k) {
      for (int f = 0; f < k - 1; ++f) {
        std::cout << "\t";
      }
      std::cout << "Profundo:" << k << "\n";
      for (int f = 0; f < k; ++f) {
        std::cout << "\t";
      }
      std::cout << "     ancho:";
      for (int f = 1; f < dim[0] + 1; ++f) {
        std::cout << " " << f << "   ";
      }
      std::cout << "\n";
      for (int j = 1; j < dim[1] + 1; ++j) {
        for (int f = 0; f < k; ++f) {
          std::cout << "\t";
        }
        std::cout << "alto:" << j << "    ";

        for (int i = 1; i < dim[0] + 1; ++i) {
          try {
            char simbolo = t->getCasillero(i - 1, j - 1, k - 1)
                               ->getFicha()
                               ->getTipoDeFicha();
            std::cout << "  " << simbolo << "  ";
          } catch (...) {
            std::cout << "  "
                      << "-"
                      << "  ";
          }
        }
        std::cout << "\n";
      }
      std::cout << "\n\n";
    }
    std::cout << "\n#COORDENADAS\n";
    std::cout
        << "\nLas coordenadas que ingrese deben estar entre los rangos:
             \n";
    std::cout << "\n\t Ancho:\t\t1 - " << dim[0];
    std::cout << "\n\t Alto:\t\t1 - " << dim[1];
    std::cout << "\n\t Profundo:\t1 - " << dim[2];

    cout << "\n\
        nÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂ
        "
            "Âů";
```

```
  cout << "ÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂůÂů\n\n";
}
void Interfaz::mostrarBatallaIniciada() {
  std::cout
      << "\033[1;36m\033[4mBatalla iniciada, el juego ha comenzado!\033[0m\n"
      << std::endl;
}

void Interfaz::mostrarFicha(Ficha *ficha) {
  if (ficha == NULL) {
    cout << '-';
  }
  if (ficha->getTipoDeFicha() == MINA) {
    cout << 'M';
  }
  if (ficha->getTipoDeFicha() == AVION) {
    cout << '>';
  }
  if (ficha->getTipoDeFicha() == BARCO) {
    cout << 'B';
  }
  if (ficha->getTipoDeFicha() == SOLDADO) {
    cout << 'S';
  }
  if (ficha->getTipoDeFicha() == ARMAMENTO) {
    cout << 'A';
  }
}
void Interfaz::mostrarFinDelJuego() {
  std::cout << "\033[1;31m\033[4mFin del juego, gracias por jugar!\033[0m\n"
            << std::endl;
}

void Interfaz::mostrarGanador(std::string nombreGanador) {
  cout << "\n\n================================================================="
          "=";
  cout << "==============================\n\n";
  std::cout << "\n\tEl ganador del juego es: \n\n\t\t" << nombreGanador
            << std::endl;
  cout << "\n\n================================================================="
          "=";
  cout << "==============================\n\n";
}

void Interfaz::pedirNombre(int jugadorNumero) {
  std::cout << "\nJugador " << jugadorNumero
            << " - Ingrese su nombre (max 10 caracteres): ";
}
```

```
138 void Interfaz::pedirCantidadJugadores() {
139   std::cout << "\nIngrese la cantidad de jugadores con la que desea "
140                "jugar (2 o "
141                "mas): ";
142 }
143
144 void Interfaz::pedirCantidadFichas() {
145   std::cout << "\nIngrese la cantidad de fichas que tendra cada "
146                "jugador (4 o mas): ";
147 }
148
149 void Interfaz::pedirDimensiones() {
150   std::cout << "\nIngrese las dimensiones del tablero (ancho, alto, "
151                "profundo): "
152             << std::endl;
153 }
154
155 void Interfaz::tableroChico() {
156   std::cout << "\nEl tablero debe ser mas grande." << std::endl;
157 }
158
159 void Interfaz::pedirCantidadCartas() {
160   std::cout << "\nIngrese la cantidad maxima de cartas que podra tener
        en la "
161                "mano cada jugador al mismo tiempo (minimo 6): ";
162 }
163 void Interfaz::pedirUsarCarta() {
164   std::cout << "Desea usar una carta? S/N" << std::endl;
165 }
166
167 void Interfaz::pedirPosicionFichaABloquear() {
168   std::cout << "\nIngrese la posicion en el tablero de la ficha a
       bloquear: "
169             << std::endl;
170 }
171
172 void Interfaz::informarNoHayFicha() {
173   std::cout << "\nNo hay una ficha en la posicion del tablero
       ingresada."
174             << std::endl;
175 }
176
177 void Interfaz::pedirPosicionCasilleroABloquear() {
178   std::cout << "\nIngrese la posicion en el tablero del casillero a "
179                "bloquear: "
180             << std::endl;
181 }
182 void Interfaz::indicarColocarMina() {
183   std::cout << "Coloque una \033[1;31mmina\033[0m";
184 }
185
186 void Interfaz::indicarMoverSolado() {
187   std::cout << "Desea mover un soldado? S/N" << std::endl;
188 }
189
190 void Interfaz::indicarMoverArmamento() {
191   std::cout << "Desea mover un armamento? S/N" << std::endl;
192 }
```

```
193
194 void Interfaz::pedirCoordPonerFicha() {
195   std::cout << "\nIngrese la posicion en el tablero donde desea "
196                "colocar su ficha: "
197             << std::endl;
198 }
199
200 void Interfaz::informarCasilleroNoDisponible() {
201   std::cout << "\nEl casillero elegido no esta disponible" << std::
        endl;
202 }
203
204 void Interfaz::informarMovimientoDeFicha() {
205   std::cout << "\n La ficha se ha movido con ÃĽxito." << std::endl;
206 }
207
208 void Interfaz::pedirCoordOrigenMoverFicha() {
209   std::cout << "\nIngrese la posicion en el tablero de la ficha que "
210                "desea mover: "
211             << std::endl;
212 }
213
214 void Interfaz::pedirCoordDestinoMoverFicha() {
215   std::cout << "\nIngrese la posicion en el tablero a donde desea
        mover la "
216                "ficha elegida: "
217             << std::endl;
218 }
219
220 void Interfaz::preguntarUsarCarta() {
221   std::cout << "\nQuieres utilizar una carta? (S/N): ";
222 }
223
224 void Interfaz::preguntarNroCarta() {
225   std::cout << "\nCual carta quieres usar?: ";
226 }
227
228 void Interfaz::tocaPonerFicha(std::string nombreJugador, char simbolo,
229                               int cantidadFichas, std::string color) {
230   std::cout << "\nJugador: " << nombreJugador
231             << " te toca poner una ficha, tenes " << cantidadFichas
232             << " (usas la " << simbolo << "  -  Color: " << color << "
                  )."
233             << std::endl;
234 }
235
236 void Interfaz::tocaMoverFicha(std::string nombreJugador, char simbolo,
237                               std::string color) {
238   std::cout << "\nJugador: " << nombreJugador
239             << " te toca mover una ficha (usas la " << simbolo
240             << "  -  Color: " << color << ")." << std::endl;
241 }
242
243 void Interfaz::mostrarCartasJugador(Jugador *jugador) {
244   unsigned int contador = 0;
245   std::cout << "\n Jugador: '" << jugador->getNombre() << "', tus
        cartas son: ";
246   Lista<Carta *> *cartas = jugador->getCartas();
```

```
247     cartas->iniciarCursor();
248     while (cartas->avanzarCursor()) {
249       contador++;
250       Carta *carta = cartas->obtenerCursor();
251       std::cout << "\n\t * " << contador << " : " << carta->
              getDescripcion()
252                 << "\n";
253     }
254   }
255
256   void Interfaz::jugadorNoTieneCartaElegida() {
257     std::cout << "\nNo tienes la carta que elegiste usar." << std::endl;
258   }
259
260   void Interfaz::jugadorSinCartas() {
261     std::cout << "\nNo tienes cartas para usar." << std::endl;
262   }
263
264   void Interfaz::mazoSinCartas() {
265     std::cout << "\nEl mazo no tiene mas cartas." << std::endl;
266   }
267
268   void Interfaz::ingresoInvalido() {
269     std::cout << "\nEl valor ingresado es invalido, vuelva a intentar"
270               << std::endl;
271   }
272
273   void Interfaz::mostrarJugadorEnTurno(std::string nombreDeJugador) {
274     std::cout << "\nEs el turno del jugador: " << nombreDeJugador << "."
275               << std::endl;
276   }
277
278   void Interfaz::limpiarPantalla() {
279     std::cout << "\033[1mPresione Enter para iniciar...\033[0m" << std::
          endl;
280     std::cin.ignore(); // Limpiar el bÃžfer de entrada
281     std::cin.get();    // Esperar a que se ingrese una tecla
282
283     std::cout << "\033[H\033[2J\033[3J"; // CARACTER PARA LIMPIAR LA
          PANTALLA
284   }
285
286   void Interfaz::mostrarFichaEnCasillero(Casillero *casillero) {
287
288     std::cout << " ";
289     this->mostrarFicha(casillero->getFicha());
290     std::cout << "";
291   }
292   void Interfaz::pedirCoordenadas(int &x, int &y, int &z, Tablero *
        tablero) {
293
294     bool coordenadasValidas = false;
295
296     std::cout << "\nIngrese las coordenas del casillero (ancho, alto, "
297                  "profundo): "
298               << std::endl;
299
300     while (!coordenadasValidas) {
```

```
301       int *dimesiones = tablero->getDimensiones();
302       std::cin >> x;
303       std::cin >> y;
304       std::cin >> z;
305
306       if (x < 1 || y < 1 || z < 1 || x > dimesiones[0] || y > dimesiones
              [1] ||
307           z > dimesiones[2]) {
308         std::cout << "Coordenadas invÃąlidas. Intente nuevamente." <<
                std::endl;
309         coordenadasValidas = false;
310       } else {
311         coordenadasValidas = true;
312       }
313     }
314     x--;
315     y--;
316     z--;
317   }
318
319   unsigned int Interfaz::pedirNroCarta(Jugador *jugador) {
320     unsigned int posicionDeCarta;
321     bool cartaValida = false;
322
323     while (!cartaValida) {
324       this->preguntarNroCarta();
325       std::cin >> posicionDeCarta;
326
327       if (posicionDeCarta < 1 ||
328           posicionDeCarta > jugador->getCartas()->contarElementos()) {
329         this->ingresoInvalido();
330       } else {
331         cartaValida = true;
332       }
333     }
334
335     return posicionDeCarta;
336   }
337
338   void pintarCasillero(BMP &image, int xStart, int yStart, int xEnd, int
          yEnd,
339                        int red, int green, int blue) {
340     for (int x = xStart; x <= xEnd; ++x) {
341       for (int y = yStart; y <= yEnd; ++y) {
342         if (x == xStart || x == xEnd || y == yStart || y == yEnd) {
343           // Dibujar contorno negro
344           image(x, y)->Red = 0;
345           image(x, y)->Green = 0;
346           image(x, y)->Blue = 0;
347         } else {
348           // Pintar el interior del casillero
349           image(x, y)->Red = red;
350           image(x, y)->Green = green;
351           image(x, y)->Blue = blue;
352         }
353       }
354     }
355   }
```

```
356
357 void pintarFicha(BMP &image, int xStart, int yStart, int xEnd, int
    yEnd,
358                   int fichaRed, int fichaGreen, int fichaBlue) {
359   int centerX = (xStart + xEnd) / 2;
360   int centerY = (yStart + yEnd) / 2;
361   int radius = (xEnd - xStart) / 2 - 2;
362
363   for (int x = xStart; x <= xEnd; ++x) {
364     for (int y = yStart; y <= yEnd; ++y) {
365       int dx = x - centerX;
366       int dy = y - centerY;
367       if (dx * dx + dy * dy <= radius * radius) {
368         // Pintar dentro del cÃrculo
369         image(x, y)->Red = fichaRed;
370         image(x, y)->Green = fichaGreen;
371         image(x, y)->Blue = fichaBlue;
372       } else if (dx * dx + dy * dy >= (radius - 2) * (radius - 2) &&
373                  dx * dx + dy * dy <= radius * radius) {
374         // Pintar contorno del cÃrculo en color blanco
375         image(x, y)->Red = 255;
376         image(x, y)->Green = 255;
377         image(x, y)->Blue = 255;
378       }
379     }
380   }
381 }
382
383 void asignarColor(TipoTerreno terreno, EstadoCasillero estado, int &
    red,
384                   int &green, int &blue) {
385   switch (terreno) {
386   case AGUA:
387     red = (estado == CASILLERO_ENVENENADO) ? 85 : 0;
388     green = (estado == CASILLERO_BLOQUEADO) ? 85 : 100;
389     blue = (estado == CASILLERO_BLOQUEADO) ? 85 : 235;
390     break;
391   case TIERRA:
392     red = (estado == CASILLERO_BLOQUEADO) ? 128 : 102;
393     green = (estado == CASILLERO_ENVENENADO) ? 255 : 51;
394     blue = (estado == CASILLERO_ENVENENADO) ? 0 : 0;
395     break;
396   case AIRE:
397     red = (estado == CASILLERO_BLOQUEADO) ? 209 : 0;
398     green = (estado == CASILLERO_ENVENENADO) ? 235 : 191;
399     blue = (estado == CASILLERO_ENVENENADO) ? 247 : 255;
400     break;
401   }
402 }
403
404 // FunciÃ§n para asignar el color de la ficha
405 void asignarColorFicha(Ficha *ficha, int &red, int &green, int &blue)
    {
406   if (ficha == NULL) {
407     red = 255;
408     green = 255;
409     blue = 255; // Gris (por defecto)
410   } else {
```

```
411       switch (ficha->getTipoDeFicha()) {
412       case NO_DEFINIDA:
413         red = 255;
414         green = 255;
415         blue = 255;
416         break;
417       case MINA:
418         red = 0;
419         green = 255;
420         blue = 95; // Verde brillante
421         break;
422       case SOLDADO:
423         red = 255;
424         green = 255;
425         blue = 0; // Amarillo brillante
426         break;
427       case ARMAMENTO:
428         red = 255;
429         green = 0;
430         blue = 255; // Magenta
431         break;
432       case BARCO:
433         red = 255;
434         green = 165;
435         blue = 0; // Naranja
436         break;
437       case AVION:
438         red = 128;
439         green = 128;
440         blue = 128; // Gris
441         break;
442       }
443     }
444   }
445
446   void pintarFondoNegro(BMP &image) {
447     // Pintar fondo negro
448     for (int x = 0; x < image.TellWidth(); ++x) {
449       for (int y = 0; y < image.TellHeight(); ++y) {
450         image(x, y)->Red = 255;
451         image(x, y)->Green = 255;
452         image(x, y)->Blue = 255;
453       }
454     }
455   }
456   void Interfaz::mostrarTableroDeJugadorBitMap(Tablero* tablero, Jugador
        * jugador) {
457     int dim[3];
458     dim[0] = tablero->getDimensiones()[0];
459     dim[1] = tablero->getDimensiones()[1];
460     dim[2] = tablero->getDimensiones()[2];
461
462     int nivelesPorFila = 3;  // Cantidad de capas por cada fila de
          niveles
463     int pixelSize = 60;      // TamaÃšo de pÃŋxel para cada casillero (
          aumentado)
464     int layerGap = 20;       // Espacio entre capas (aumentado)
465     BMP image;
```

```
466     image.SetSize(dim[0] * pixelSize * nivelesPorFila + (nivelesPorFila
            - 1) * layerGap + 50,
467                   dim[1] * pixelSize * ((dim[2] + nivelesPorFila - 1) /
                          nivelesPorFila) +
468                       ((dim[2] + nivelesPorFila - 1) / nivelesPorFila -
                              1) * layerGap + 50);
469     image.SetBitDepth(24);
470     pintarFondoNegro(image);
471
472     for (int k = 0; k < dim[2]; ++k) {
473       int nivelActual = k / nivelesPorFila;     // Nivel actual en la
              fila de niveles
474       int nivelOffset = k % nivelesPorFila;     // Desplazamiento en la
              fila de niveles
475
476       for (int j = 0; j < dim[1]; ++j) {
477         for (int i = 0; i < dim[0]; ++i) {
478           Casillero* casillero = tablero->getCasillero(i, j, k);
479           Ficha* ficha = casillero->getFicha();
480
481           int xStart = (nivelActual * dim[0] + i) * pixelSize +
                  nivelActual * layerGap + 25;
482           int yStart = (j + nivelOffset * dim[1]) * pixelSize +
                  nivelOffset * layerGap + 25;
483           int xEnd = (nivelActual * dim[0] + i + 1) * pixelSize +
                  nivelActual * layerGap + 25;
484           int yEnd = (j + 1 + nivelOffset * dim[1]) * pixelSize +
                  nivelOffset * layerGap + 25;
485
486           EstadoCasillero estadoCasillero = casillero->getEstado();
487           TipoTerreno terreno = casillero->obtenerTerreno();
488           int red, green, blue;
489           asignarColor(terreno, estadoCasillero, red, green, blue);
490           pintarCasillero(image, xStart, yStart, xEnd, yEnd, red, green,
                   blue);
491           if (ficha != NULL && ficha->getIdentificadorDeJugador() ==
                  jugador->getNombre()) {
492             int fichaRed, fichaGreen, fichaBlue;
493             asignarColorFicha(ficha, fichaRed, fichaGreen, fichaBlue);
494             pintarFicha(image, xStart, yStart, xEnd, yEnd, fichaRed,
                    fichaGreen, fichaBlue);
495           }
496         }
497       }
498     }
499
500     std::cout << "Guardando tablero." << std::endl;
501     image.WriteToFile("tablero.bmp");
502     std::cout << "Tablero impreso." << std::endl;
503   }
```

Listing 14: Interfaz.h

```
1   #ifndef INTERFAZ_H_
2   #define INTERFAZ_H_
3
4   #include "Carta.h"
5   #include "Jugador.h"
```

```
6  #include "Lista.h"
7  #include "Tablero.h"
8  #include "iostream"
9
10 using namespace std;
11
12 class Interfaz {
13 public:
14   /*
15    * Pre: ---
16    * Post: muestra por pantalla el menu principal del juego mostrando
          el titulo
17    * del mismo y dando bienvenida al usuario
18    */
19   void mostrarPantallaInicial();
20
21   /*
22    * Pre: ---
23    * Post: Muestra por pantalla que el juego ha iniciado.
24    */
25   void mostrarBatallaIniciada();
26
27   /*
28    * Pre:----
29    * Post: muestra en pantalla el simbolo de la ficha recibida sin dar
30    * salto de lÃηnea al final
31    */
32   void mostrarFicha(Ficha *);
33
34   /*
35    * Pre: recibe un string con el nombre del Jugador que ha ganado el
          juego
36    * Post: muestra un mensaje por pantalla indicando el nombre del
          ganador
37    */
38   void mostrarGanador(std::string);
39
40   /*
41    * Pre: Recibe un tablero valido
42    * Post: muestra por pantalla el la disposicion del tablero
          especificando
43    *        el ancho alto y profundo para orientar al ususario
44    */
45   void mostrarControles(Tablero *);
46
47   /*
48    * Pre: recibe el estado actual del tablero
49    * Post: muestra por pantalla las Fichas de los casilleros del
          tablero
50    * recibido imprimiendo tantos planos como profundidad tenga el
          tablero
51    * recibido. Por ejemplo si el tablero tiene una profundidad de 2
          niveles
52    * (z=2) se mostrara por pantalla 2 planos xy pertenecientes a cada
          nivel de
53    * profundidad
54    */
55   void mostrarTablero(Tablero *);
```

```
56
57      /*
58       * Pre: Recibe un tablero y un jugador
59       * Post: Muestra por pantalla el tablero del jugador.
60       */
61      void mostrarTableroDeJugador(Tablero *, Jugador *);
62      /*
63       * Pre: recibe el nÃžmero de jugador al que corresponde el nombre
64       * Post: pide por pantalla el nombre al jugador indicando su nÃžmero
65       * ejemplo: Jugador 1: TomÃąs, Jugador 2: Miguel, ett
66       */
67      void pedirNombre(int);
68
69      /*
70       * Pre: ---
71       * Post: pide por pantalla la cantidad de Jugadores que tendra el
             Juego
72       */
73      void pedirCantidadJugadores();
74
75      /*
76       * Pre: ---
77       * Post: pide por pantalla la cantidad de Fichas que tendra cada
             Jugador
78       */
79      void pedirCantidadFichas();
80
81      /*
82       * Pre: ---
83       * Post: pide por pantalla cada una de las dimensiones del tablero
84       */
85      void pedirDimensiones();
86
87      /*
88       * Pre: ---
89       * Post: imprime un mensaje de error indicando al usuario que el
             tablero
90       * debe ser mas grande
91       */
92      void tableroChico();
93
94      /*
95       * Pre: ---
96       * Post: pide por pantalla cual sera la cantidad maxima de cartas
             que tendra
97       * cada jugador
98       */
99      void pedirCantidadCartas();
100
101     /*
102      * Pre: ---
103      * Post: imprime un mensaje para ver si usa o no carta en el turno.
104      */
105     void pedirUsarCarta();
106
107     /*
108      * Pre: ---
109      * Post: imprime un mensaje para indicar que debe colocar una mina.
```

```
110      */
111     void indicarColocarMina();
112 
113     /*
114      * Pre: ---
115      * Post: imprime un mensaje para indicar que debe mover un soldado.
116      */
117     void indicarMoverSolado();
118 
119     /*
120      * Pre: ---
121      * Post: imprime un mensaje para indicar que debe mover un armamento
             .
122      */
123     void indicarMoverArmamento();
124 
125     /*
126      * Pre: ---
127      * Post: imprimer un mensaje que indica en fin del juego.
128      */
129     void mostrarFinDelJuego();
130 
131     /*
132      * Pre: Recibe un tablero y un jugador
133      * Post: imprime el tablero mediante bitmap, monstrando solo las
             fichas del
134      * jugador ingresado.
135      */
136     void mostrarTableroDeJugadorBitMap(Tablero *tablero, Jugador *
            jugador);
137 
138     /*
139      * Pre: ---
140      * Post: pide por pantalla la posicion de la ficha a bloquear
141      */
142     void pedirPosicionFichaABloquear();
143 
144     /*
145      * Pre: ---
146      * Post: imprime un mensaje de error indicando al usuario que la
             posicion
147      * ingresada es invalida
148      */
149     void informarNoHayFicha();
150 
151     /*
152      * Pre: ---
153      * Post: pide por pantalla la posicion del casillero a bloquear
154      */
155     void pedirPosicionCasilleroABloquear();
156 
157     /*
158      * Pre: ---
159      * Post: indica por pantalla que se ingreso un valor invalido
160      */
161     void ingresoInvalido();
162 
163     /*
```

```
164      * Pre: ---
165      * Post: muestra un mensaje en pantalla pidiendo al jugador que
             introduzca
166      * las 3 coordenadas del tablero donde desea ubicar su ficha
167      */
168     void pedirCoordPonerFicha();
169
170     /*
171      * Pre: ---
172      *Post: imprime un mensaje de error indicando al usuario que el
             casillero
173      *pedido no esta disponible
174      */
175     void informarCasilleroNoDisponible();
176
177     /*
178      * Pre: ---
179      * Post: pide por pantalla la posicion de la ficha que se desea
             mover
180      */
181     void pedirCoordOrigenMoverFicha();
182
183     /*
184      * Pre: ---
185      * Post: pide por pantalla la posicion de destino de la ficha que se
              desea
186      * mover
187      */
188     void pedirCoordDestinoMoverFicha();
189
190     /*
191      * Pre: ---
192      * Post: imprime un mensaje preguntado al usuario si quiere utilizar
              alguna
193      * de sus cartas
194      */
195     void preguntarUsarCarta();
196
197     /*
198      * Pre: ---
199      * Post: imprime un mensaje preguntado al usuario que numero de
             carta quier
200      * utilizar
201      */
202     void preguntarNroCarta();
203
204     /*
205      * Pre: recibe un string nombreJugador valido y un char simbolo
             valido
206      * Post: indica al jugador "nombreJugador" que debe poner una ficha
             cuyo
207      * simbolo en pantalla es "simbolo"
208      */
209     void tocaPonerFicha(std::string, char, int, std::string);
210
211     /*
212      * Pre: recibe un string nombreJugador valido y un char simbolo
             valido
```

```
213      * Post: indica al jugador "nombreJugador" que debe mober una ficha
             cuyo
214      * simbolo en pantalla es "simbolo"
215      */
216     void tocaMoverFicha(std::string, char, std::string);
217
218     /*
219      * Pre: Recibe una lista de cartas valida y un string nombreJugador
             valido
220      * Post: muestra por pantalla las "cartas" disponibles del jugador
221      * "nombreJugador"
222      */
223     /*
224     void mostrarCartasJugador(std::string, Lista<Carta*> *);
225   */
226
227     /*
228      * Pre: ---
229      * Post: imprime un mensaje de error indicando al usuario que no
             posee la
230      * carta elegida
231      */
232     void jugadorNoTieneCartaElegida();
233
234     /*
235      * Pre: ---
236      * Post: imprime por pantalla un mensaje indicando al usuario que el
              mismo
237      * no posee cartas en su mano
238      */
239     void jugadorSinCartas();
240
241     /*
242      * Pre: ---
243      * Post: Limpia la interfaz de juego
244      */
245     void limpiarPantalla();
246
247     /*
248      * Pre: ---
249      * Post: imprime por pantalla el nombre del jugador
250      */
251     void mostrarJugadorEnTurno(std::string nombreJugador);
252
253     /*
254      * Pre: ---
255      * Post: imprime un mensaje confirmando el movimiento realizado
256      */
257     void informarMovimientoDeFicha();
258
259     /*
260      * Pre: Recibe un jugador.
261      * Post: Imprime un mensaje mostrando las cartas del jugador.
262      */
263     void mostrarCartasJugador(Jugador *jugador);
264
265     /*
266      * Pre: ---
```

```
267      * Post: imprime un mensaje de que el jugador no tiene cartas.
268      */
269     void mazoSinCartas();
270 
271     /*
272      * Pre: ---
273      * Post: imprime mostrando la ficha en el casillero.
274      */
275     void mostrarFichaEnCasillero(Casillero *casillero);
276 
277     /*
278      * Pre: recibe las coordenadas x,y,z.
279      * Post: imprime un mensaje pidiendo una posicion y las almacena en
             x,y,z.
280      */
281     void pedirCoordenadas(int &x, int &y, int &z, Tablero *tablero);
282 
283     /*
284      * Pre: Recibe un jugador.
285      * Post:imprime un mensaje pidiendo la carta a usar del jugador y
286      * devuelve el numero de la carta a usar.
287      */
288     unsigned int pedirNroCarta(Jugador *jugador);
289 };
290 
291 #endif /* INTERFAZ_H_ */
```

7.9. Constantes

Listing 15: Constantes.h

```
1  #ifndef CONSTANTES_H_
2  #define CONSTANTES_H_
3  
4  #include <string>
5  
6  #define VACIO '.'
7  
8  enum TipoDeFicha { NO_DEFINIDA, MINA, SOLDADO, ARMAMENTO, BARCO, AVION
       };
9  
10 enum EstadoFicha { FICHA_DESBLOQUEADA, FICHA_BLOQUEADA };
11 
12 enum EstadoCasillero {
13   CASILLERO_BLOQUEADO,
14   CASILLERO_DESBLOQUEADO,
15   CASILLERO_ENVENENADO
16 };
17 
18 enum TipoTerreno { TIERRA, AGUA, AIRE };
19 
20 #endif /* CONSTANTES_H_ */
```

7.10. Lista

Listing 16: Lista.h

```
#ifndef LISTA_H_
#define LISTA_H_

#include "Nodo.h"

template <class T> class Lista {
private:
  Nodo<T> *primero;
  unsigned int tamanio;
  Nodo<T> *cursor;

public:
  /*
   * Post: constructor de lista vacia
   */
  Lista();
  /*
   * Post: copia la lista recibida
   */
  Lista(Lista<T> &otraLista);
  /*
   * Post: libera los recursos asociados a la lista
   */
  ~Lista();
  /*
   * Post: agrega elemento al final de Lista (posiciÃ§n
      contarElementos() + 1)
   */
  void altaFinal(T elemento);
  /*
   * Pre: posiciÃ§n pertenece al intervalo [1, contarElementos() + 1]
   * Post: agrega el elemento en la posiciÃ§n indicada
   */
  void altaPosicion(T elemento, unsigned int posicion);
  /*
   * Post: agrega todos los elementos de otraLista a partir de la
      posiciÃ§n
   * contarElementos() + 1
   */
  void agregar(Lista<T> &otraLista);
  /*
   * Pre: posiciÃ§n pertenece al intervalo [1, contarElementos()]
   * Post: devuelve el elemento en la posiciÃ§n indicada
   */
  T obtener(unsigned int posicion);
  /*
   * Pre: posiciÃ§n pertenece al intervalo: [1, contarElementos()]
   * Post: cambia elemento de posiciÃ§n indicada por elemento pasado
   */
  void asignar(T elemento, unsigned int posicion);
  /*
   * Pre: posiciÃ§n pertenece al intervalo: [1, contarElementos()]
   * Post: remueve de la Lista el elemento en la posiciÃ§n indicada
   */
  void remover(unsigned int posicion);
  /*
```

```
55      * Post: deja cursor de Lista preparado para hacer nuevo recorrido
            colocandolo
56      * en NULL
57      */
58     void iniciarCursor();
59     /*
60      * Pre: se ha iniciado un recorrido (invocando el mÃ©todo
            iniciarCursor())
61      * y desde entonces no se han agregado o removido elementos de la
            Lista.
62      *
63      * Post: mueve cursor, lo posiciona en siguiente elemento del
            recorrido
64      * El valor de retorno indica si el cursor quedÃ³ posicionado sobre
            un elemento
65      * o no (en caso de que la Lista estÃ© vacÃa o no existan mÃ¡s
            elementos por
66      * recorrer.)
67      */
68     bool avanzarCursor();
69     /*
70      * Pre: el cursor estÃ¡ posicionado sobre un elemento de la Lista
71      * (fue invocado el mÃ©todo avanzarCursor() y devolviÃ³ true)
72      *
73      * Post: devuelve el elemento en la posiciÃ³n del cursor.
74      */
75     T obtenerCursor();
76     /*
77      * Post: devuelve en booleano si la lista esta vacia o no
78      */
79     bool estaVacia();
80
81     /*
82      * Post: devuelve un entero con la cantidad de elementos en la lista
83      */
84     unsigned int contarElementos();
85
86     /*
87      * Post: Devuelve el ultimo elemento de la lista si esta no es nula
88      */
89
90     T bajaAlFinal();
91
92   private:
93     /*
94      * Pre: posicion pertenece al intervalo [1, contarElementos()]
95      * Post: devuelve el nodo en la posicion indicada
96      */
97     Nodo<T> *obtenerNodo(unsigned int posicion);
98   };
99
100  template <class T> Lista<T>::Lista() {
101    this->primero = NULL;
102    this->tamanio = 0;
103    this->cursor = NULL;
104  }
105
106  template <class T> Lista<T>::Lista(Lista<T> &otraLista) {
```

```
107     this->primero = NULL;
108     this->tamanio = 0;
109     this->cursor = NULL;
110 
111     this->agregar(otraLista); // copia los elementos de otraLista
112   }
113 
114   template <class T> Lista<T>::~Lista() {
115 
116     // mientras exista el primer nodo, se reasigna el puntero primero y
          se elimina
117     // el que era antes primer nodo
118     while (this->primero != NULL) {
119       Nodo<T> *aBorrar = this->primero;
120       this->primero = this->primero->getSiguiente();
121       delete aBorrar;
122     }
123   }
124 
125   template <class T> bool Lista<T>::estaVacia() { return (this->tamanio
       == 0); }
126 
127   template <class T> unsigned int Lista<T>::contarElementos() {
128     return this->tamanio;
129   }
130 
131   template <class T> void Lista<T>::altaFinal(T elemento) {
132     this->altaPosicion(elemento,
133                        this->tamanio + 1); // tamanio + 1 = nueva ultima
                            posicion
134   }
135 
136   template <class T>
137   void Lista<T>::altaPosicion(T elemento, unsigned int posicion) {
138 
139     if ((posicion > 0) &&
140         (posicion <= this->tamanio + 1)) { // valida la posicion de
              insercion
141       Nodo<T> *nuevo = new Nodo<T>(elemento);
142 
143       /* si la posicion donde se desea insertar es la primera
144        * se desplaza el primer nodo a la segunda posicion y
145        * se inserta el nuevo en la primera
146        */
147       if (posicion == 1) {
148         nuevo->setSiguiente(this->primero);
149         this->primero = nuevo;
150 
151       }
152 
153       /* si no, se apunta con anterior al nodo de la posiciÃşn anterior,
154        * el siguiente del nodo nuevo serÃą el siguiente del anterior,
155        * y el siguiente del anterior serÃą el nuevo
156        */
157       else {
158         Nodo<T> *anterior = this->obtenerNodo(posicion - 1);
159         nuevo->setSiguiente(anterior->getSiguiente());
160         anterior->setSiguiente(nuevo);
```

```
161       }
162
163       this->tamanio++;
164       this->iniciarCursor();
165     }
166   }
167
168   template <class T> void Lista<T>::agregar(Lista<T> &otraLista) {
169     otraLista.iniciarCursor(); // se inicializa el cursor de otraLista
          en NULL
170     while (
171         otraLista
172             .avanzarCursor()) { // mientras exista un nodo apuntado por
                el cursor
173       this->altaFinal(
174           otraLista
175               .obtenerCursor()); // se agrega al final el dato del nodo
                  apuntado
176     }
177   }
178
179   template <class T> T Lista<T>::obtener(unsigned int posicion) {
180     T elemento;
181     if ((posicion > 0) &&
182         (posicion <= this->tamanio)) { // si la posicion recibida es
              valida
183       elemento = this->obtenerNodo(posicion)
184                       ->getDato(); // obtenerNodo(posicion) retorna un
                          puntero al
185                                    // nodo del cual obtenemos el dato
186     } else {
187       throw "Indice fuera de rango";
188     }
189     return elemento;
190   }
191
192   template <class T> void Lista<T>::asignar(T elemento, unsigned int
        posicion) {
193     if ((posicion > 0) && (posicion <= this->tamanio)) {
194       this->obtenerNodo(posicion)->setDato(elemento);
195     }
196   }
197
198   template <class T> void Lista<T>::remover(unsigned int posicion) {
199
200     if ((posicion > 0) && (posicion <= this->tamanio)) {
201       Nodo<T> *removido;
202
203       if (posicion == 1) {
204         removido = this->primero;
205         this->primero = removido->getSiguiente();
206
207       } else {
208         Nodo<T> *anterior = this->obtenerNodo(posicion - 1);
209         removido = anterior->getSiguiente();
210         anterior->setSiguiente(removido->getSiguiente());
211       }
212
```

```
213       delete removido;
214       this->tamanio--;
215       this->iniciarCursor();
216     }
217   }
218
219   template <class T> void Lista<T>::iniciarCursor() { this->cursor =
        NULL; }
220
221   template <class T> bool Lista<T>::avanzarCursor() {
222
223     if (this->cursor == NULL) {
224       this->cursor = this->primero;
225
226     } else {
227       this->cursor = this->cursor->getSiguiente();
228     }
229
230     return (this->cursor != NULL);
231   }
232
233   template <class T> T Lista<T>::obtenerCursor() {
234     if (this->cursor == NULL) {
235       throw("");
236     }
237     T elemento = this->cursor->getDato();
238
239     return elemento;
240   }
241
242   template <class T> Nodo<T> *Lista<T>::obtenerNodo(unsigned int
        posicion) {
243
244     Nodo<T> *actual = this->primero;
245
246     // se avanza hasta localizar el dato de la posiciÃ§n pasada por
          argumento y se
247     // retorna
248     for (unsigned int i = 1; i < posicion; i++) {
249       actual = actual->getSiguiente();
250     }
251
252     return actual;
253   }
254
255   template <class T> T Lista<T>::bajaAlFinal() {
256     if (this->estaVacia()) {
257       throw "Lista sin elementos";
258     }
259
260     T elemento = this->obtener(this->tamanio);
261     this->remover(this->tamanio);
262
263     return elemento;
264   }
265
266   #endif /* LISTA_H_ */
```

7.11. Cola

Listing 17: Cola.h

```
#ifndef COLA_H_
#define COLA_H_


#include "Nodo.h"

template<class T>
class Cola {

private:
    Nodo <T> * inicio;
    Nodo <T> * final;
    unsigned int tamanio;

public:

    /*
     * Pre: -
     * Post: inicializa atributos en NULL y cero
     */
    Cola();


    /*
     * Pre: -
     * Post: hace delete de cada nodo de la cola
     */
    ~Cola();


    /*
     * Pre: Recibe un elemento de tipo T
     * Post: crea Nodo en el heap y agrega al final (cambia inicio,
         final y tamanio)
     */
    void push( T );


    /*
     * Pre: La cola no esta vacia
     * Post: saca el primer elemento de la cola y lo devuelve
     *          (lanza error si la cola esta vacia)
     */
    T pop();


    /*
     * Pre: -
     * Post: devuelve True si tamanio=0
     */
    bool estaVacia();

};

```

```
54
55  template<class T>
56  Cola<T>::Cola() {
57      this->inicio = NULL;
58      this->final = NULL;
59      this->tamanio = 0;
60  }
61
62
63  template<class T>
64  Cola<T>::~Cola() {
65
66      while ( !(this->estaVacia()) ){
67          this->pop();
68      }
69  }
70
71
72  template<class T>
73  void Cola<T>::push(T dato) {
74
75      Nodo<T> * nuevoNodo = new Nodo<T>(dato);
76
77      if ( !this->inicio ) {
78          this->inicio = nuevoNodo;
79          this->final = this->inicio;
80      }
81      else {
82          this->final->setSiguiente(nuevoNodo);
83          this->final = nuevoNodo;
84      }
85
86      ++(this->tamanio);
87  }
88
89
90  template<class T>
91  T Cola<T>::pop() {
92
93      if ( !(this->inicio) ) {
94          throw ("Cola vacia. No se puede obtener elemento");
95      }
96
97      T datoaBorrar = this->inicio->getDato();
98      Nodo<T> * nodoaBorrar = this->inicio;
99      this->inicio = this->inicio->getSiguiente();
100     delete nodoaBorrar;
101     --(this->tamanio);
102
103     return  datoaBorrar;
104 }
105
106
107 template<class T>
108 bool Cola<T>::estaVacia() {
109
110     return ( this->tamanio == 0 );
111 }
```

```
112
113
114
115 #endif /* COLA_H_ */
```