Trabajo práctico 1: Tesoro Binario v1.0

Nombre: Daniel

Apellido: Mamani

Padrón: 109932

Mail: danymamani1002@gmail.com

Manual de usuario

Reglas del juego

- El objetivo es recuperar todos los tesoros del oponente.
- Cada jugador dispone de 4 tesoros para esconderlos en un tablero de 20 filas y 20 columnas (20x20).
- Solo se puede poner un tesoro por casillero.
- Gana el juego el jugador que haya recuperado primero todos los tesoros del oponente.

Desarrollo del juego

- Al iniciar el juego cada jugador deberá ingresar su nombre y las posiciones donde esconderá sus tesoros en el tablero.
- Mediante turnos, cada jugador pondrá un espía en algún casillero del tablero y luego podrá optar por mover de posición algún tesoro suyo.
- Los tesoros pueden moverse en diagonal, vertical y horizontal.
- En caso de que el espía encuentre un tesoro en esa posición, el casillero no estará disponible (no podrá ponerse ni espía ni tesoro en ese casillero) durante 5 turnos. El jugador contrincante pierde ese tesoro.
- Si un espía encuentra un espía enemigo ambos espías serán eliminados del tablero, dejando libre el casillero nuevamente.
- Si un tesoro se mueve a un casillero con un espía, el juego le avisará al oponente que su espía encontró un tesoro.
- Si un tesoro es movido a un casillero con un tesoro el juego le avisará al jugador para que pueda recuperarlo en su próximo turno.
- Los turnos se sucederán hasta que uno de los jugadores se quede sin tesoros disponibles.
- Gana el juego el primer jugador en recuperar todos los tesoros del oponente.

Estados de los casilleros del tablero

Durante todo el juego los casilleros del tablero mostrarán distintos símbolos representando el estado de los casilleros.

Los estados posibles que puede tomar un casillero son:

- **LIBRE (" - ") :** Este símbolo representa que el casillero está vacío y puede ponerse un tesoro o un espía en el mismo. Ambos jugadores pueden ver los casilleros vacíos.
- **TESORO (" T ")**: Este símbolo representa que el casillero contiene un tesoro. Cada jugador sólo puede ver sus propios tesoros.
- **ESPÍA (" E ")**: Este símbolo representa que el casillero contiene un espía. Cada jugador sólo puede ver sus propios espías.
- **OCUPADO (" O ")**: Este símbolo representa que el casillero está ocupado por un tesoro. Este estado lo verá un jugador que haya movido un tesoro donde hay un tesoro del oponente.
- **NO DISPONIBLE (" X ")**: Este símbolo representa que el casillero está inhabilitado, es decir, que no podrá utilizarse durante 5 turnos. Pasados los 5 turnos el casillero se mostrará como **LIBRE**. Ambos jugadores pueden ver los casilleros inhabilitados.

Manual del programador

Introducción

En esta sección se describirán que conforman el proyecto y su funcionamiento, con el fin de que el lector pueda modificar a su gusto algunos de los valores y parámetros de las funciones expuestas a su gusto.
El proyecto está conformado por seis archivos fuente:

- **main.cpp** - Archivo principal que contiene el llamado a las funciones principales para el funcionamiento del programa del juego Tesoro Binario.
- **Registro.cpp** - Archivo dónde se encuentran las funciones relacionadas con los casilleros del tablero. (Ej: inicializar el registro, destruirlo, etc).
- **tablero.cpp** - Archivo dónde se encuentran todas funciones relacionadas con el tablero del juego. (Ej: inicializar el tablero, indicar si el casillero está ocupado, etc).
- **Tesoro.cpp** - Archivo dónde están todas las funciones relacionadas con los tesoros de los jugadores. (Ej: inicializar tesoro).
- **Jugador.cpp** - Archivo dónde están las funciones relacionadas con el jugador. (Ej: inicializar el jugador y destruirlo).
- **TesoroBinario.cpp** - Archivo dónde están todas las funciones relacionadas con el desarrollo del juego Tesoro Binario. (Ej: inicializar Tesoro Binario, esconder los tesoros, poner espías, mostrar tablero, finalizar el juego, etc).

Aclaración : los archivos **Registro.cpp**, **tablero.cpp**, **Tesoro.cpp**, **Jugador.cpp** y **TesoroBinario.cpp** tienen sus archivos de cabecera (header files) correspondientes donde están las firmas de cada función.

Bloque **main.cpp**

Se importan todas las funciones principales del bloque **TesoroBinario.cpp** e inicia la función principal **main**.

```
#include "TesoroBinario.h"
int main(){
```

Se declaran dos variables de tipo string que almacenará el nombre de cada jugador.

una variable tesoroBinario de TesoroBinario y luego se la inicializa enviando los nombres de los jugadores por parámetro.

Después se llama a la función inicarJuego() que presentará el juego a los jugadores, pedirá sus nombres y las posiciones para esconder sus tesoros.

```
std::string jugador1;
std::string jugador2;
TesoroBinario *tesoroBinario = inicializarTesoroBinario(jugador1, jugador2);
iniciarJuego(tesoroBinario);
```

Se llama a la función jugar() que ejecuta el desarrollo del juego.

Luego se llama a la función finalizarJuego() que informa al ganador del juego una vez finalizado y cierra el juego.

Por último finaliza el la función main()

```
jugar(tesoroBinario);
        finalizarJuego(tesoroBinario);
  return 0;
}
```

Bloque **Registro.cpp**

Se incluye el archivo de cabecera donde están todas las firmas de las funciones implementadas en este bloque.

```
#include "Registro.h"
```

Se declaran dos constantes para representar el nombre de un jugador vacio y un id de un tesero nulo. Estas constantes se usan para inicializar los campos de registro.

```
static const std::string NOMBRE_VACIO = " ";
static const int ID_TESORO_NULO = 0;
```

Se implementa la función **inicializarRegistro** que recibe un puntero a una variable tipo Registro como parámetro para inicializar todos sus campos.

```
void inicializarRegistro(Registro *registro) {
```

```
        registro->ocupada = false;
        registro->estado = LIBRE;
        registro->jugador = NOMBRE_VACIO;
        registro->tesoroId = ID_TESORO_NULO;
}
```
Finaliza el bloque **Registro.cpp**

Bloque **tablero.cpp**

Se incluye el archivo de cabecera donde están todas las firmas de las funciones implementadas en el bloque.

```
#include "tablero.h"
```

Se implementa la función **incializarTablero** que crea un puntero de tipo tablero e inicializa todos sus campos para finalmente devolverlo.

```
Tablero *inicializarTablero() {
        Tablero *tablero = new Tablero();
        tablero->alto = ALTO;
        tablero->ancho = ANCHO;
        tablero->matriz = new Registro*[tablero->ancho];
        for(int fila = 0; fila < tablero->ancho; fila++) {
      tablero->matriz[fila] = new Registro[tablero->alto];
                for(int columna = 0; columna < tablero->alto; columna++) {
                        inicializarRegistro(&tablero->matriz[fila][columna]);
                }
        }
        return tablero;
}
```

Se implementa la función **esCasilleroOcupado** que recibe un putero de tipo tablero, un int fila y un int columna por parámetro. verifica si el casillero en la fila-columna recibidas está ocupado.

```
bool esCasilleroOcupado(Tablero *tablero, int fila, int columna){
   if(tablero->matriz[fila][columna].ocupada == true){
      return true;
   }

   return false;
}
```

Se implementa la función **destruirTablero** que libera la memoria reservada por el tablero.

```
void destruirTablero(Tablero *tablero){
   for(int fila = 0; fila < tablero->ancho; fila++) {
                        for(int columna = 0; columna < tablero->alto; columna++) {
                                delete &tablero->matriz[fila][columna];
                        }
                        delete tablero->matriz[fila];
           }

   delete tablero;
}
```

Bloque **Tesoro.cpp**

Se incluye el archivo de cabecera donde están todas las firmas de las funciones implementadas en el bloque.

```
#include "Tesoro.h"
```

Se implementa la función **inicializarTesoro** que recibe el id del tesoro por parámetro. Crea un puntero de tipo Tesoro, inicializa todos sus campos y luego lo devuelve.

```
Tesoro *inicializarTesoro(int id){
   Tesoro *tesoro = new Tesoro();
   tesoro->id = id;
   tesoro->columna = 0;
   tesoro->fila = 0;
   tesoro->turnosEnRecuperacion = 0;
   return tesoro;
}
```

Se implementa la función **destruirTesoro** que libera la memoria reservada por el tesoro recibido por parámetro.

```
void destruirTesoro(Tesoro *tesoro){
   delete tesoro;
}
```

Bloque **Jugador.cpp**

Se incluye el archivo de cabecera donde están todas las firmas de las funciones implementadas en el bloque.

Se declaran dos constantes, una que representa la cantidad de tesoros iniciales que tiene un jugador; y otra que representa el nombre del archivo que contendrá el estado del tablero del jugador.

```
#include "Jugador.h"

static const int CANTIDAD_TESOROS_INICIAL = 4;
static const std::string NOMBRE_ARCHIVO = "estadoTablero.txt";
```

Se implementa la función **incializarJugador** que recibe el nombre de un jugador por parámetro.
La función crea un puntero de tipo Jugador, inicializa todos sus campos y por último lo devuelve.

```
Jugador *inicializarJugador(std::string nombre){
   Jugador *jugador = new Jugador();
   jugador->nombre = nombre;
   jugador->cantidadTesoros = CANTIDAD_TESOROS_INICIAL;
   jugador->cantidadTesorosDescubiertos = 0;
   jugador->estadoTablero = NOMBRE_ARCHIVO;
   jugador->encontroTesoro = false;
   jugador->encontroEspia = false;
   jugador->gano = false;
   for(int i = 0; i < jugador->cantidadTesoros; i++){
      jugador->tesoros[i] = inicializarTesoro(i+1);
      jugador->tesorosDescubiertos[i] = inicializarTesoro(0);
   }
   return jugador;
}
```

Se implementa la función **destruirJugador** que recibe un puntero de tipo Jugador por parámetro. La función libera la memoria reservada por el puntero recibido.

```
void destruirJugador(Jugador *jugador){
   for(int i = 0; i < jugador->cantidadTesoros; i++){
      destruirTesoro(jugador->tesoros[i]);
      destruirTesoro(jugador->tesorosDescubiertos[i]);
   }
   delete jugador;
}
```

Bloque **TesoroBinario.cpp**

Se incluye el archivo de cabecera donde están todas las firmas de las funciones implementadas en el bloque.

```
#include "TesoroBinario.h"
```

Se declaran las constantes necesarias para que el código sea más claro y fácil de leer.

```
static const std::string CELDA_LIBRE = "-";
static const std::string CELDA_TESORO = "T";
static const std::string CELDA_ESPIA = "E";
static const std::string CELDA_NO_DISPONIBLE = "X";
static const std::string CELDA_OCUPADA = "O";
static const std::string SIN_JUGADOR = " ";
static const int ALTO_MAXIMO = ALTO;
static const int ANCHO_MAXIMO = ANCHO;
static const int ALTO_MINIMO = 0;
static const int ANCHO_MINIMO = 0;
static const int SIN_TESORO = 0;
static const int TESORO_RECUPERADO = 5;
static const char MOVER_TESORO = 'S';
```

Se implementa la función **inicializarTesoroBinario** que recibe el nombre de ambos jugadores por parámetro.
La función crea un puntero de tipo TesoroBinario, inicializa todos sus campos y después lo devuelve.

```
TesoroBinario *inicializarTesoroBinario(std::string nombreJugador1, std::string nombreJugador2){
   TesoroBinario *tesoroBinario = new TesoroBinario();
   tesoroBinario->tablero = inicializarTablero();
   tesoroBinario->jugador1 = inicializarJugador(nombreJugador1);
   tesoroBinario->jugador2 = inicializarJugador(nombreJugador2);
   tesoroBinario->finalizado = false;
   return tesoroBinario;
}
```

Se implementa la función **esPosicionValida.**

```
/**
* Devuelve true si fila y columna son validas, false sino.
**/
bool esPosicionValida(int fila, int columna){
   if(fila <= ANCHO_MINIMO || columna <= ALTO_MINIMO){
      std::cout << "Esa posicion no es valida, ingrese otra" << std::endl;
      return false;
   }
   else if(fila > ANCHO_MAXIMO || columna > ALTO_MAXIMO){
```

```
        std::cout << "Esa posicion no es valida, ingrese otra" << std::endl;
        return false;
    }
    else{
        return true;
    }
}
```

Se implementa la función **pedirPosicion**.

```
/**
* Pide una posición válida para un casillero.
**/
void pedirPosicion(int &fila, int &columna){
    std::cout << "Ingrese una posición" << std::endl;
    bool valida = false;
    while(valida == false){
        std::cin >> fila >> columna;
        valida = esPosicionValida(fila, columna);
    }
}
```

Se implementa la función **cargarCasillero.**

```
/**
* Le asigna el estado y el nombre del jugador recibidos por parámetro a la posición recibida.
**/
void cargarCasillero(Tablero *tablero, std::string jugador, int fila, int columna, EstadoDeRegistro
estado, int tesoroId){
    if(tablero->matriz[fila][columna].estado == TESORO){
        tablero->matriz[fila][columna].tesoroId = tesoroId;
    }
    tablero->matriz[fila][columna].estado = estado;
    if(estado != NO_DISPONIBLE){
        tablero->matriz[fila][columna].ocupada = true;
    }
    tablero->matriz[fila][columna].jugador = jugador;
}
```

Se implementa la función **definirCasillero.**

```
/**
* Define cómo se verá el estado recibido por parámetro en el tablero.
**/
std::string definirCasillero(EstadoDeRegistro estado){
    switch(estado){
        case LIBRE:
            return CELDA_LIBRE;
            break;
```

```
      case TESORO:
         return CELDA_TESORO;
         break;
      case ESPIA:
         return CELDA_ESPIA;
         break;
      case NO_DISPONIBLE:
         return CELDA_NO_DISPONIBLE;
         break;
      case OCUPADA:
         return CELDA_OCUPADA;
         break;
   }
}
```

Se implementa la función **cargarEstadoTablero.**

```
/**
* Carga el estado actual del tablero en un archivo y lo guarda en estadoTablero.
**/
void cargarEstadoTablero(Tablero *tablero, std::string estadoTablero, Jugador *jugador){
   std::ofstream estado;
   estado.open(estadoTablero.c_str());
   for(int fila = 0; fila < tablero->ancho; fila++) {
                        for(int columna = 0; columna < tablero->alto; columna++) {
         if(tablero->matriz[fila][columna].jugador == jugador->nombre ||
tablero->matriz[fila][columna].jugador == SIN_JUGADOR){
            estado << " " << definirCasillero(tablero->matriz[fila][columna].estado) << " ";
         }
         else if(tablero->matriz[fila][columna].ocupada == true){
            estado << " " << CELDA_LIBRE << " ";
         }
                        }
                        estado << std::endl;
          }
          estado << "Tesoros disponibles: " << jugador->cantidadTesoros << " Tesoros descubiertos: "
<< jugador->cantidadTesorosDescubiertos << std::endl;
   for(int i = 0; i < jugador->cantidadTesoros; i++){
      estado << "Tesoro " << jugador->tesoros[i]->id << " posición: " << jugador->tesoros[i]->fila << " "
<< jugador->tesoros[i]->columna << std::endl;
   }
          estado.close();
}
```

Se implementa la función **mostrarTablero**.

```
/**
* Muestra el estado actual del tablero del jugador de turno en pantalla.
**/
void mostrarTablero(std::string estadoTablero){
```

```
    std::ifstream lecturaEstado;
    lecturaEstado.open(estadoTablero.c_str());
    std::string linea;
    while(!lecturaEstado.eof()){
        std::getline(lecturaEstado, linea);
        std::cout << linea << std::endl;
    }
    lecturaEstado.close();
}
```

Se implementa la función **esconderTesoros.**

```
/**
* Esconde los tesoros del jugador en las posiciones que este indica, al iniciar el juego.
**/
void esconderTesoros(Jugador *jugador, Tablero *tablero){
    std::cout << "Ingrese nombre del jugador" << std::endl;
    std::cin >> jugador->nombre;
    std::cout << jugador->nombre << " ingresá las posiciones donde esconderás tus tesoros" <<
std::endl;
    for(int i = 0; i < jugador->cantidadTesoros; i++){
        pedirPosicion(jugador->tesoros[i]->fila, jugador->tesoros[i]->columna);
        cargarCasillero(tablero, jugador->nombre, jugador->tesoros[i]->fila-1,
jugador->tesoros[i]->columna-1, TESORO, jugador->tesoros[i]->id);
    }
    cargarEstadoTablero(tablero, jugador->estadoTablero, jugador);
    mostrarTablero(jugador->estadoTablero);
}
```

Se implementa la función **inciarJuego** que recibe un puntero de tipo TesoroBinario por parámetro, presenta el juego y lo prepara para que se pueda jugar.

```
void iniciarJuego(TesoroBinario *tesoroBinario){
    std::cout << "Bienvenidos al juego Tesoro Binario (V1.0)" << std::endl;
    esconderTesoros(tesoroBinario->jugador1, tesoroBinario->tablero);
    esconderTesoros(tesoroBinario->jugador2, tesoroBinario->tablero);
}
```

Se implementa la función **mostrarEstadoTableroGeneral** que recibe un puntero de tipo Tablero por parámetro. Muestra el tablero cargado con los estados de ambos jugadores a lo largo de la partida.

```
void mostrarEstadoTableroGeneral(Tablero *tablero){
    std::cout << "Estado general de la partida:" << std:endl;
    for(int fila = 0; fila < tablero->ancho; fila++) {
                        for(int columna = 0; columna < tablero->alto; columna++) {
            std::cout << definirCasillero(tablero->matriz[fila][columna].estado);
                        }
```

```
                    std::cout << "\n";
            }
}
```

Se implementa la función **recuperarTesoro.**

```
/**
* Marca el casillero con el tesoro recuperado como no disponible durante 5 turnos.
**/
void recuperarTesoro(Tablero *tablero, int fila, int columna, Jugador *jugador){
   jugador->tesorosDescubiertos[jugador->cantidadTesorosDescubiertos]->id =
tablero->matriz[fila][columna].tesoroId;
   jugador->tesorosDescubiertos[jugador->cantidadTesorosDescubiertos]->fila = fila;
   jugador->tesorosDescubiertos[jugador->cantidadTesorosDescubiertos]->columna = columna;
   jugador->tesorosDescubiertos[jugador->cantidadTesorosDescubiertos]->turnosEnRecuperacion++;
   jugador->cantidadTesorosDescubiertos++;
   jugador->encontroTesoro = true;
   cargarCasillero(tablero, SIN_JUGADOR, fila, columna, NO_DISPONIBLE, SIN_TESORO);
}
```

Se implementa la función **verEstadoCasillero**.

```
/**
* Le indica al jugador el estado del casillero en la posición recibida.
**/
void verEstadoCasillero(Tablero *tablero, int fila, int columna, Jugador *jugador, EstadoDeRegistro
nuevoEstado){
   switch(tablero->matriz[fila][columna].estado){
         case TESORO:
            if(nuevoEstado == ESPIA){
               std::cout << "Encontro un tesoro, recuperando..." << std::endl;
               std::cout << "El casillero no estara no disponible durante 5 turnos" << std::endl;
               recuperarTesoro(tablero, fila, columna, jugador);
            }else if(nuevoEstado == TESORO){
               std::cout << "Encontró un tesoro en la posición " << fila+1 << " " << columna+1 <<
std::endl;
               cargarCasillero(tablero, jugador->nombre, fila, columna, OCUPADA, SIN_TESORO);
            }
            break;
         case ESPIA:
            if(nuevoEstado == TESORO){
               std::cout << "Encontró un espia enemigo en la posición " << fila+1 << " " << columna+1 <<
std::endl;
               std::cout << "El casillero no estara no disponible durante 5 turnos" << std::endl;
               jugador->encontroEspia = true;
            }
            else{
            std::cout << "Encontro un espia enemigo, ambos fueron borrados del tablero" << std::endl;
            cargarCasillero(tablero, SIN_JUGADOR, fila, columna, LIBRE, SIN_TESORO);
            }
```

```
            break;
        case NO_DISPONIBLE:
            std::cout << "El casillero no está disponible" << std::endl;
            pedirPosicion(fila, columna);
            cargarCasillero(tablero, jugador->nombre, fila, columna, nuevoEstado, SIN_TESORO);
            break;
    }
}
```

Se implementa la función **ponerEspia**.

```
/**
* Coloca un espia en la posicón que indica el jugador.
**/
void ponerEspia(Tablero *tablero, Jugador *jugador){
   int fila, columna;
   std::cout << "Ingrese la pocion donde pondrá al espia" << std::endl;
   pedirPosicion(fila, columna);
   if(esCasilleroOcupado(tablero, fila-1, columna-1)){
      verEstadoCasillero(tablero, fila-1, columna-1, jugador, ESPIA);
   }
   else{
      cargarCasillero(tablero, jugador->nombre, fila-1, columna-1, ESPIA, SIN_TESORO);
      std::cout << "El espia fue colocado" << std::endl;
   }
}
```

Se implementa la función **seRecuperoTesoro**.

```
/**
* Devuelve true si el tesoro fue recuperado; false sino.
**/
bool seRecuperoTesoro(Tesoro &tesoro){
   if(tesoro.turnosEnRecuperacion == TESORO_RECUPERADO){
      std::cout << "El casillero está disponible de nuevo" << std::endl;
      return true;
   }
   if(tesoro.turnosEnRecuperacion > 0){
      tesoro.turnosEnRecuperacion++;
   }
   return false;
}
```

Se implementa la función **verificarTesorosRecuperados**.

```
/**
* Verifica si el tesoro fue recuperado para dejar el casillero libre.
**/
void verificarTesorosRecuperados(Tablero *tablero, Tesoro *tesoros, int
cantidadTesorosDescubiertos){
```

```
    for(int i = 0; i < cantidadTesorosDescubiertos; i++){
        if(seRecuperoTesoro(tesoros[i])){
            cargarCasillero(tablero, SIN_JUGADOR, tesoros[i].fila, tesoros[i].columna, LIBRE, SIN_TESORO);
        }
    }
}
```

Se implementa la función **esIdTesoroValido**.

```
/**
* Devuelve true si tesoroId es valida; false sino.
**/
bool esIdTesoroValido(int tesoroId, int cantidadTesoros){
    std::cin >> tesoroId;
    if(tesoroId >= 1 && tesoroId <= cantidadTesoros){
        return true;
    }

    std::cout << "Ese id no es valido, ingrese otro" << std::endl;
    return esIdTesoroValido(tesoroId, cantidadTesoros);
}
```

Se implementa la función **moverTesoro.**

```
/**
* Mueve un tesoro a la posición recibida por parametro.
**/
void moverTesoro(Tablero *tablero, Jugador *jugador, int nuevaFila, int nuevaColumna, Jugador
*oponente){
    int tesoroId = 0;
    std::cout << "Ingrese el id del tesoro que desea mover (Debe ser entre 1 y " <<
jugador->cantidadTesoros << ")" << std::endl;
    esIdTesoroValido(tesoroId, jugador->cantidadTesoros);
    pedirPosicion(nuevaFila, nuevaColumna);
    if(esCasilleroOcupado(tablero, nuevaFila-1, nuevaColumna-1)){
        verEstadoCasillero(tablero, nuevaFila-1, nuevaColumna-1, jugador, TESORO);
        if(jugador->encontroEspia){
            recuperarTesoro(tablero, nuevaFila-1, nuevaColumna-1, oponente);
        }
    }
    else{
        cargarCasillero(tablero, jugador->nombre, nuevaFila-1, nuevaColumna-1, TESORO, tesoroId);
        std::cout << "El tesoro con el id" << tesoroId << " fue movido de posición" << std::endl;
    }
    jugador->tesoros[tesoroId-1]->fila = nuevaFila;
    jugador->tesoros[tesoroId-1]->columna = nuevaColumna;
}
```

Se implementa la función **finalizarTurno**.

```
/**
* Informa al jugador opnente si jugador principal encontró un tesoro y verifica si alguno de los dos ganó el juego.
**/
void finalizarTurno(Tablero *tablero, Jugador *principal, Jugador *oponente){
   if(principal->encontroTesoro == true){
      oponente->cantidadTesoros--;
      std::cout << oponente->nombre << ", " << principal->nombre << " encontró un tesoro tuyo" << std::endl;
   }
   principal->encontroTesoro = false;
   mostrarEstadoTableroGeneral(tablero);
}
```

Se implementa la función **definirGanador**.

```
/**
* Define si uno de los jugadores ganó el juego.
**/
void definirGanador(Jugador *jugador1, Jugador *jugador2){
   if(jugador1->cantidadTesoros == SIN_TESORO){
      jugador2->gano = true;
   }else if(jugador2->cantidadTesoros == SIN_TESORO){
      jugador1->gano = true;
   }
}
```

Se implementa la función **seFinalizoTesoroBinario**.

```
/**
* Devuelve true si uno de los jugadores ganó el juego; false sino.
**/
bool seFinalizoTesoroBinario(bool ganoJugador1, bool ganoJugador2){
   if(ganoJugador1 || ganoJugador2){
      return true;
   }
   return false;
}
```

Se implementa la función **jugarTurno**.

```
/**
* Ejecuta las acciones posibles que puede hacer un jugador durante su turno.
**/
void jugarTurno(Tablero *tablero, Jugador *principal, Jugador *oponente){
   definirGanador(principal, oponente);
   if(seFinalizoTesoroBinario(principal->gano, oponente->gano)){
      return;
   }
```

```
    verificarTesorosRecuperados(tablero, *principal->tesorosDescubiertos,
principal->cantidadTesorosDescubiertos);
    char opcion;
    int fila, columna;
    std::cout << principal->nombre << " ingresá la posición para poner un espia" << std::endl;
    ponerEspia(tablero, principal);
    std::cout << principal->nombre << " querés mover un tesoro? (Pulse S para si, N para no)" <<
std::endl;
    std::cin >> opcion;
    if(opcion == MOVER_TESORO){
        moverTesoro(tablero, principal, fila, columna, oponente);
    }
    cargarEstadoTablero(tablero, principal->estadoTablero, principal);
    mostrarTablero(principal->estadoTablero);
    finalizarTurno(tablero, principal, oponente);
}
```

Se implementa la función **jugar** que recibe un puntero tipo TesoroBinario como parámetro. La función ejecuta un ciclo en el que los jugadores jugarán por turnos mientras el juego no haya finalizado.

```
void jugar(TesoroBinario *tesoroBinario){
    while(tesoroBinario->finalizado == false){
        jugarTurno(tesoroBinario->tablero, tesoroBinario->jugador1, tesoroBinario->jugador2);
        jugarTurno(tesoroBinario->tablero, tesoroBinario->jugador2, tesoroBinario->jugador1);
        tesoroBinario->finalizado = seFinalizoTesoroBinario(tesoroBinario->jugador1->gano,
tesoroBinario->jugador2->gano);
    }
}
```

Se implementa la función **finalizarJuego** que recibe un puntero tipo TesoroBinario por parámetro.
La función felicita al ganador y luego libera toda la memoria reservada por el puntero tesoroBinario.

```
void finalizarJuego(TesoroBinario *tesoroBinario){
    if(tesoroBinario->jugador1->gano){
        std::cout << "!Felicidades " << tesoroBinario->jugador1->nombre << " ganaste¡" << std::endl;
    }
    else if(tesoroBinario->jugador2->gano){
        std::cout << "!Felicidades " << tesoroBinario->jugador2->nombre << " ganaste¡" << std::endl;
    }
    destruirJugador(tesoroBinario->jugador1);
    destruirJugador(tesoroBinario->jugador2);
    destruirTablero(tesoroBinario->tablero);
    delete tesoroBinario;
    std::cout << "El juego tesoro binario ha finalizado, gracias por jugar" << std::endl;
```

```
}
```

Bloque **Registro.h**

Se inicia el bloque incluyendo el componente iostream de la biblioteca estándar de C++.

```
#ifndef REGISTRO_H_
#define REGISTRO_H_

#include <iostream>
```

Se definen dos estructuras, una tipo EstadoRegistro que contiene todos los estados posibles que puede tomar un casillero en el tablero; y una tipo Registro que contiene las características necesarias para usarlo en el juego.

```
enum EstadoDeRegistro{
        OCUPADA,
        TESORO,
        ESPIA,
        NO_DISPONIBLE,
        LIBRE
};
typedef struct {
        bool ocupada;
        EstadoDeRegistro estado;
        std::string jugador;
        int tesoroId;
} Registro;
```

Se declaran todas las funciones utilizadas en el bloque **Registro.cpp**.

```
/**
* Inicializa todos los campos del puntero registro recibido por parámetro.
**/
void inicializarRegistro(Registro *registro);
#endif //REGISTRO_H_
```

Bloque **tablero.h**

Se inicia el bloque incluyendo el archivo "*Registro.h*" para poder acceder a todas sus funciones y biblotecas importadas.

```
#ifndef TABLERO_H_
#define TABLERO_H_
#include "Registro.h"
```

Se definen dos constantes que representan el ancho y alto que tendrá el tablero.
Se define una estructura de tipo Tablero que contiene las dimensiones del tablero y una matriz dinámica de Registros que lo representa.

```
typedef struct{
   Registro **matriz;
   int ancho;
   int alto;
}Tablero;
```

Se declaran todas las funciones utilizadas en el bloque **tablero.cpp.**

```
/**
* Reserva la memoria necesaria para crear un tablero y luego inicializa sus casilleros.
*/
Tablero *inicializarTablero();
/**
* Devuelve true si el casillero en la posición fila-columna está ocupado; false sino.
**/
bool esCasilleroOcupado(Tablero *tablero, int fila, int columna);
/**
* Libera la memoria reservada para el tablero.
*/
void destruirTablero(Tablero *tablero);
#endif // TABLERO_H_
```

Bloque **Tesoro.h**

Se inicia el bloque definiendo una estructura de tipo Tesoro que contiene las características necesarias para su uso en el juego.

```
typedef struct{
   int id;
   int fila;
   int columna;
   int turnosEnRecuperacion;
}Tesoro;
```

Se declaran todas las funciones utilizadas en el bloque **Tesoro.cpp.**

```
/**
* Reserva la memoria necesaria para un tesoro e inicializa todos sus campos en 0,
* excepto el id que lo recibe por parámetro.
**/
Tesoro *inicializarTesoro(int id);
```

```
/**
* Libera la memoria reservada para tesoro.
**/
void destruirTesoro(Tesoro *tesoro);
#endif // _TESORO_H_
```

Bloque **Jugador.h**

Se inicia el bloque incluyendo el archivo "*Tesoro.h"* para poder acceder a todas sus funciones y bibliotecas importadas.

Se incluye el componente iostream de la biblioteca estándar de C++.

```
#ifndef _JUGADOR_H_
#define _JUGADOR_H_
#include "Tesoro.h"
#include <iostream>
```

Se define una estructura de tipo Jugador que contiene todas las características de un jugador para el juego.

```
typedef struct{
   std::string nombre;
   int cantidadTesoros;
   int cantidadTesorosDescubiertos;
   Tesoro *tesorosDescubiertos[4];
   Tesoro *tesoros[4];
   std::string estadoTablero;
   bool encontroTesoro;
   bool encontroEspia;
   bool gano;
}Jugador;
```

Se declaran todas las funciones utilizadas en el bloque **Jugador.cpp.**

```
/**
* Reserva la memoria necesaria para crear un jugador e inicializa todos sus campos.
**/
Jugador *inicializarJugador(std::string nombre);
/**
* Libera la memoria reservada para el jugador.
**/
void destruirJugador(Jugador *jugador);
#endif // _JUGADOR_H_
```

Bloque **TesoroBinario.h**

Se inicia el bloque incluyendo los archivos “*tablero.h” y “Jugador.h*” para acceder a todas sus funciones y sus bibliotecas importadas.

```
#ifndef TESOROBINARIO_H_
#define TESOROBINARIO_H_
#include "tablero.h"
#include "Jugador.h"
```

Se define una estructura de tipo TesoroBinario que contiene el tablero, los jugadores y un booleano que representa si el juego se finalizó.

```
typedef struct {
        Tablero *tablero;
        bool finalizado;
        Jugador *jugador1;
        Jugador *jugador2;
} TesoroBinario;
```

Se declaran las firmas de las funciones principales utilizadas en el bloque **TesoroBinario.cpp**.

```
/**
* Reserva la memoria necesaria para una estructura TesoroBinario e incializa todos sus campos.
**/
TesoroBinario *inicializarTesoroBinario(std::string nombreJugador1, std::string nombreJugador2);
/**
* Prepara el tablero con los tesoros de los jugadores escondidos es las posiciones que ellos indicaron.
**/
void iniciarJuego(TesoroBinario *tesoroBinario);
/**
* Muestra por pantalla el estado general del tablero después del turno de ambos jugadores.
**/
void mostrarEstadoTableroGeneral(Tablero *tablero);
/**
* Ejecuta el juego para cada jugador mediante turnos.
**/
void jugar(TesoroBinario *tesoroBinario);
/**
* Libera toda la memoria reservada por TesoroBinario.
**/
void finalizarJuego(TesoroBinario *tesoroBinario);
#endif // TESOROBINARIO_H_
```

Informe del T.P

Introducción

El fin del siguiente documento es explicar las ideas, suposiciones y estrategias que se siguieron para poder resolver el enunciado del TP.

Suposiciones

- Se supone que el usuario no cometerá errores al ingresar tipos de datos, es decir si se le pide un número ingresará un número y no una letra y otro signo.
- Se supone que el usuario puede cometer errores al ingresar un número en un rango, es decir si se le pide un número entre 1 y 4, el usuario ingresa 5 o 0 y se le pide que ingresé otro número.

Archivos del programa

El programa está compuesto por los siguientes archivos:

- **main.cpp -** Archivo principal que contiene el llamado a las funciones principales para el funcionamiento del programa del juego Tesoro Binario.
- **Registro.cpp** - Archivo dónde se encuentran las funciones relacionadas con los casilleros del tablero. (Ej: inicializar el registro, destruirlo, etc).
- **Registro.h** - Archivo de cabecera donde se encuentran las funciones relacionadas con los casilleros del tablero.
- **tablero.cpp -** Archivo dónde se encuentran todas funciones relacionadas con el tablero del juego. (Ej: inicializar el tablero, indicar si el casillero está ocupado, etc).
- **tablero.h** - Archivo de cabecera dónde se encuentran todas funciones relacionadas con el tablero del juego.
- **Tesoro.cpp** - Archivo dónde están todas las funciones relacionadas con los tesoros de los jugadores. (Ej: inicializar tesoro).

- **Tesoro.h** - Archivo de cabecera dónde están todas las funciones relacionadas con los tesoros de los jugadores.
- **Jugador.cpp** - Archivo dónde están las funciones relacionadas con el jugador. (Ej: inicializar el jugador y destruirlo).
- **Jugador.h** - Archivo de cabecera dónde están las funciones relacionadas con el jugador.
- **TesoroBinario.cpp** - Archivo dónde están todas las funciones relacionadas con el desarrollo del juego Tesoro Binario. (Ej: inicializar Tesoro Binario, esconder los tesoros, poner espías, mostrar tablero, finalizar el juego, etc).
- **TesoroBinario.h** - Archivo de cabecera dónde están todas las funciones relacionadas con el desarrollo del juego Tesoro Binario.

Desarrollo del juego

1. Todas las llamadas a las funciones principales para que el juego funcione se hacen desde el main. El juego está conformado por un ciclo while que se ejecutará hasta que uno de los jugadores haya ganado.
2. Primero se crea un estructura tipo tablero que mostrará a los jugadores el estado del juego a lo largo del programa. Está estructura se inicializará con " - " representando que todos los casilleros del tablero están vacíos.
3. Luego se le presenta el programa al usuario y se le solicita a cada jugador su nombre y las posiciones donde esconderá sus tesoros.
4. Después se guardará el estado del tablero en un archivo de texto correspondiente a cada jugador, y se mostrará por pantalla. Los casilleros que muestren " T " representan que hay un tesoro en ese casillero.
5. Al finalizar ambos turnos se mostrará el estado del tablero general con los tesoros de ambos jugadores.
6. Una vez que ambos jugadores escondieron sus tesoros inicia el juego, empezando a ejecutarse el ciclo while.
7. Durante cada turno al jugador se le pedirá que ingresé la posición donde quiere poner un espía y luego se le preguntará si quiere mover uno de sus tesoros.

8. Después de que el jugador haya ingresado tanto la posición donde pondrá un espía o moverá un tesoro, el programa llamará una función que le informará por pantalla la situación sucedida y ejecutará las acciones correspondientes a dicha situación.
9. Si alguno de los jugadores el juego, se muestra un mensaje por pantalla felicitando al ganador. Luego se libera toda la memoria reservada durante el juego, y finalmente se muestra por pantalla un mensaje que informa que el juego ha finalizado.

Cuestionario

1. ¿Qué es un Debug?

Un Debug o debugging es el nombre que se le da al proceso utilizado en programación para identificar y eliminar cualquier error en un programa para que esté funcione y pueda cumplir su propósito. Dichos errores, también llamados Bugs, pueden presentarse como errores de sintaxis o de lógica.

2. ¿Qué es un "Breakpoint"?

Un Breakpoint es un punto de interrupción en la ejecución del proceso o programa en ejecución. Permite especificar detenciones del programa, para ver valores de memoria, registros o variables durante la ejecución.

3. ¿Qué es “Step Into”, “Step Over” y “Step Out”?

- Step Into: ejecuta la siguiente línea de código. Si esa línea es una llamada a otra función, el programa entrará en esa función.
- Step Over: a diferencia de Step Into, en caso de que la siguiente línea sea una función, la ejecuta sin entrar en ella.
- Step Out: sale de la función actual.