ENTREGA N°1
TRABAJO PRATICO N°2: "CUATRO EN LINEA V1.0"

Equipo:
Nientiendo

Integrantes:
- Franco Pomi - [106590]
- Ignacio Sotelo - [103923]

Manual de usuario:

- ### *Tablero:*

El tablero está conformado por filas, columnas, y distintos planos (Z) que representan la profundidad del tablero.

Ejemplo de tablero 6x6x3:

```
Z = 3
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Z = 2
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Z = 1
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```

- ***Configuración:***

Al iniciar el juego, el usuario debe indicar las dimensiones del tablero, la cantidad de fichas que deben coincidir para terminar el juego y luego que fichas utilizará:

```
Cuantas filas tendra el tablero?: 6
Cuantas Columnas tendra el tablero?: 6
Cuantos planos de profundidad tendra el tablero?: 3
Cuantos fichas tienen que coincidir para ganar?: 4
```

Luego se pedirá que cada jugador ingrese la ficha con la que quiere jugar (un carácter) y para terminar la carga, ingresas cero.

```
Ingresar fichas (caracter) para cada jugador
[0 para terminar]
Ingresar ficha para jugador 1: X
Ingresar ficha para jugador 2: O
Ingresar ficha para jugador 3: Y
Ingresar ficha para jugador 4: 0
```

- ***Cartas:***

Cada jugador posee 4 cartas, pero solo podrá utilizar 3 de ellas. Antes de cada turno, el jugador tiene la opción de utilizar alguna de sus cartas

```
Turno de jugador: 1, ficha X
Cartas restantes: 4
   1) [Carta Bloquear Turno]
   2) [Carta Juega Doble]
   3) [Carta Doble Ficha]
   4) [Carta Invertir Giro]
   0) [No usar carta]
  Ingresar carta       >
```

Cada carta tiene una función propia, siendo:

1- Bloquear Turno: el siguiente jugador pierde su turno
2- Juega Doble: el jugador actual ingresa dos fichas
3- Doble Ficha: el jugador deja caer dos fichas consecutivas una encima de la otra
4- Invertir Giro: invierte la orientación de los turnos

Una vez utilizada alguna carta, se mostrará al final de ella una leyenda indicando que tal carta ya no se encuentra disponible.

```
Turno de jugador: 1, ficha X
Cartas restantes: 3
   1) [Carta Bloquear Turno] ------> (NO DISPONIBLE)
   2) [Carta Juega Doble]
   3) [Carta Doble Ficha]
   4) [Carta Invertir Giro]
   0) [No usar carta]
  Ingresar carta       >
```

- ***Juego:***

  Luego de elegir una carta o no utilizar ninguna, se pide que se ingrese el número de la columna y la profundidad en la que se desea dejar caer la ficha:

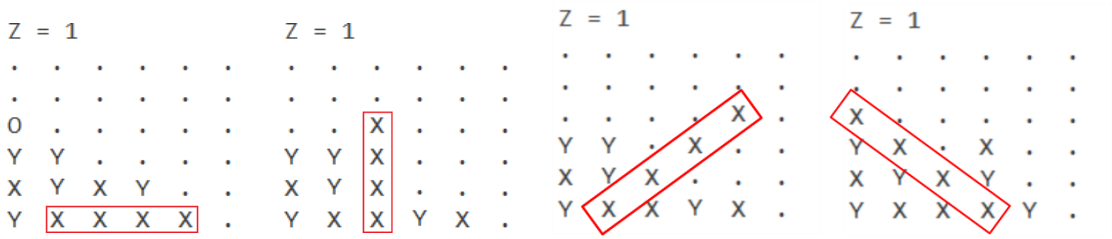
```
Ingresar columna     > 3
Ingresar profundidad > 1
```

  Y mostrando su ficha elegida al comienzo del juego en el tablero.

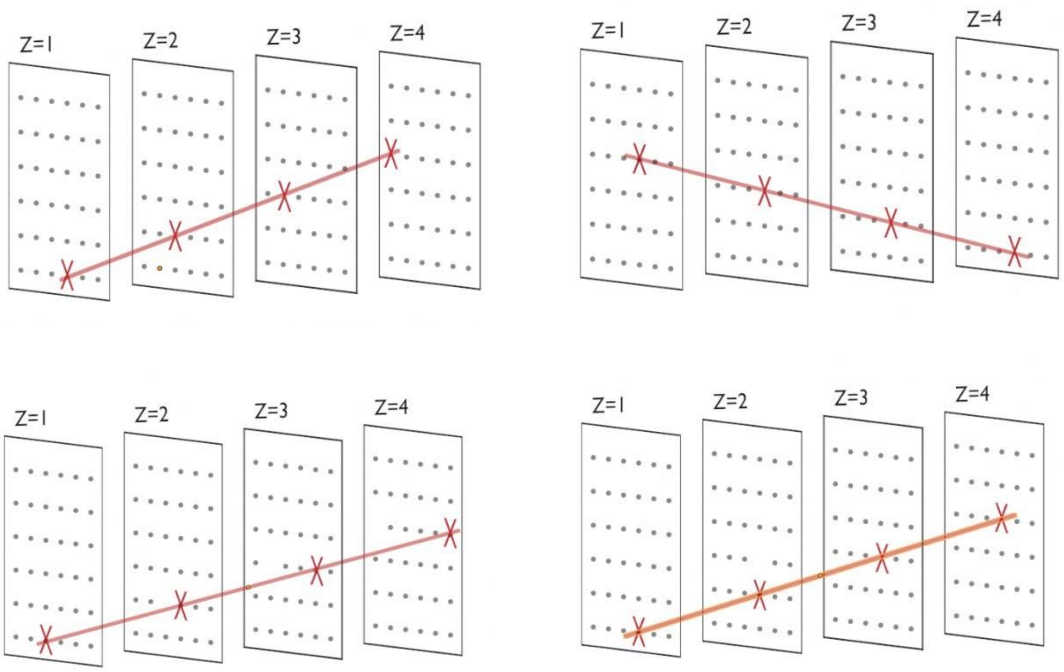
```
Z = 1
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . X . . .
```

- ***Ganador:***

  Gana el primer jugador que logre colocar N fichas en cualquier dirección, siendo N el valor ingresado al comienzo del juego.



Algunas posibles combinaciones posibles:



Manual de programador:

- ## *Columna.cpp:*

La clase Columna contiene un único atributo ‘celdas’ el cual es una lista de datos tipo ‘string’ en donde cada nodo contiene o un punto ‘.’ para representar una celda vacía, o una ficha.

Los métodos getter y setter, asignan y devuelven el dato que contiene el nodo.

```
1:  #include "Columna.h"
2:
3:  Columna::Columna() {
4:      celdas = new Lista<std::string>;
5:  }
6:
7:  Columna::~Columna() {
8:      delete celdas;
9:   }
10:
11:  void Columna::agregarCelda(std::string dato) {
12:      celdas->agregar(dato);
13:   }
14:
15: void Columna::agregarCelda(std::string dato, int posicion) {
16:      celdas->agregar(dato, posicion);
17:   }
18:
19: void Columna::setCelda(std::string dato, int posicion) {
20:      celdas->asignar(dato, posicion);
21:   }
22:
23:  std::string Columna::getCelda(int posicion) {
24:      return celdas->obtener(posicion);
25:   }
26:
27:  int Columna::getTamanio() {
28:      return celdas->contarElementos();
29:   }
```

- *Plano.cpp:*

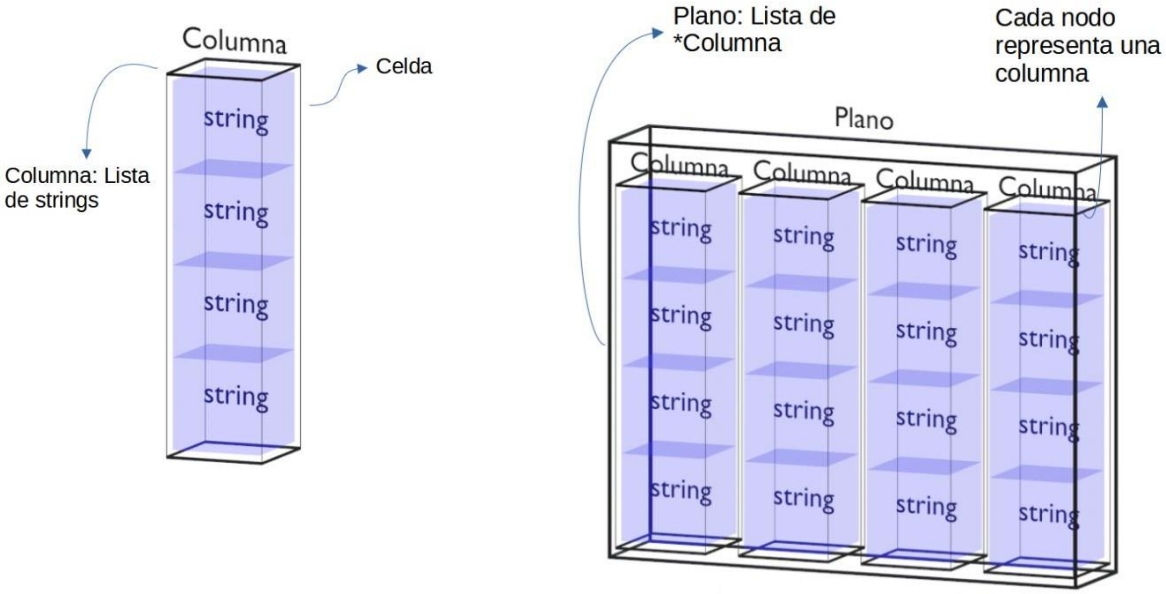
La clase Plano contiene un único atributo 'columnas' el cual es una lista de datos tipo punteros a 'Columna' en donde cada nodo contiene una columna, conformando así un tablero bidimensional.

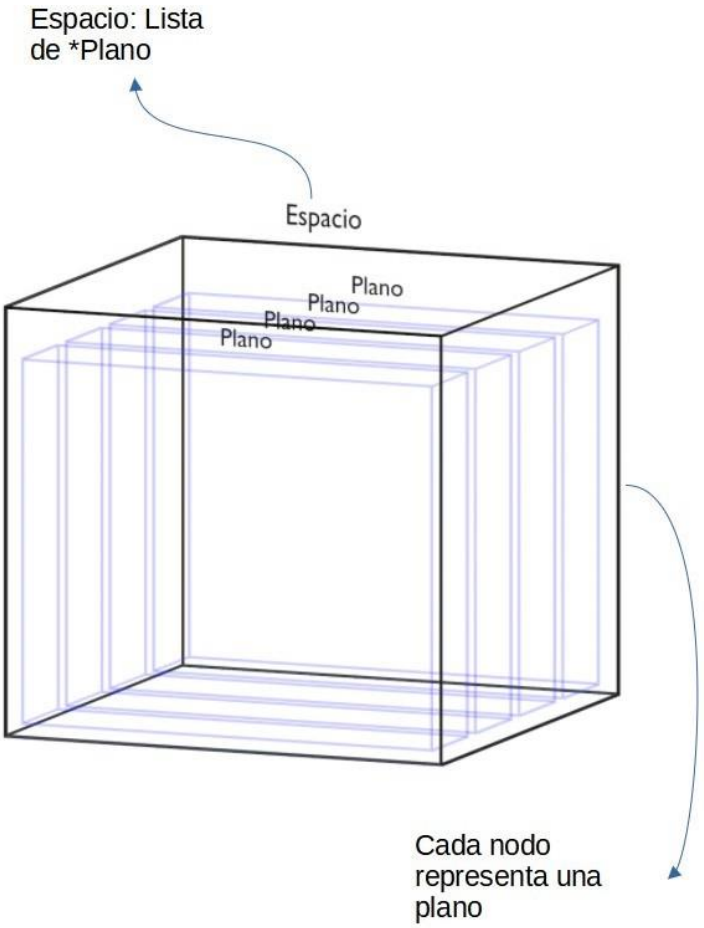
```
 1:  #include "Plano.h"
 2:
 3:  using namespace std;
 4:
 5:  Plano::Plano() {
 6:      columnas = new Lista<Columna*>;
 7:  }
 8:
 9:  Plano::~Plano() {
10:      delete columnas;
11:  }
12:
13:  void Plano::agregarColumna(Columna *columna) {
14:      columnas->agregar(columna);
15:  }
16:
17:  int Plano::getTamanio() {
18:      return columnas->contarElementos();
19:  }
20:
21:  int Plano::getDimensionColumna() {
22:      return columnas->obtener(1)->getTamanio();
23:  }
24:
25:  Columna* Plano::getColumna(int posicion) {
26:      return columnas->obtener(posicion);
27:  }
28:
29:  string Plano::getCelda( int x,  int y) {
30:      return columnas->obtener(x)->getCelda(y);
31:  }
32:
33:  void Plano::setCelda( string dato, int x,  int y) {
34:      columnas->obtener(x)->setCelda(dato, y);
35:  }
36:
37:  void Plano::imprimirPlano() {
38:      for(int i = 1; i <= columnas->obtener(1)->getTamanio(); i++) {
39:          cout << "           ";
40:          for( int j = 1; j <= columnas->contarElementos(); j++) {
41:              cout << " " << columnas->obtener(j)->getCelda(i) << " ";
42:          }
43:          cout << endl;
44:      }
45:  }
```

- ## *Espacio.cpp:*

La clase Espacio contiene un único atributo ‘planos’ el cual es una lista de datos tipo punteros a ‘Planos’ en donde cada nodo contiene un plano, conformando así un tablero tridimensional.

```
 1: #include "Espacio.h"
 3: using namespace std;
 4:
 5: Espacio::Espacio() {
 6:     planos = new Lista<Plano*>;
 7: }
 8:
 9: Espacio::~Espacio() {
10:     delete planos;
11: }
12:
13: void Espacio::agregarPlano(Plano* plano) {
14:     planos->agregar(plano);
15: }
16:
17: int Espacio::getTamanio() {
18:     return planos->contarElementos();
19: }
20:
21: int Espacio::getDimensionColumna() {
22:     return planos->obtener(1)->getDimensionColumna();
23: }
24:
25: Plano* Espacio::getPlano(int posicion) {
26:     return planos->obtener(posicion);
27: }
28:
29: string Espacio::getCelda( int x,  int y, int z) {
30:     return planos->obtener(z)->getColumna(x)->getCelda(y);
31: }
32:
33: void Espacio::setCelda(string dato, int x, int y, int z) {
34:     planos->obtener(z)->setCelda(dato, x, y);
35: }
36:
37: void Espacio::imprimirEspacio() {
38:
39:     for(int i = planos->contarElementos(); i > 0; i--) {
40:         cout << endl;
41:         cout << "            Z = " << i << endl;
42:         planos->obtener(i)->imprimirPlano();
43:         cout << endl << endl;
44:     }
45: }
```


Columna
Celda
Columna: Lista de strings
string
Plano: Lista de *Columna
Cada nodo representa una columna
Plano
Columna


Espacio: Lista de *Plano
Espacio
Plano
Cada nodo representa una plano

- *Tablero.cpp:*

La clase Tablero contiene un atributo de tipo '*Espacio ', en la cual se van a cargar las fichas y celdas funcionando como un tablero tridimensional.

También contiene como dato las dimensiones del tablero.

```
 1:  #include "Tablero.h"
 2:  #include "Constantes.h"
 3:   using namespace std;
 4:
 5: Tablero::Tablero(int numeroDeFilas, int numeroDeColumnas, int numeroDePro
fundidad) {
 6:
 7:     this->numeroDeFilas = numeroDeFilas;
 8:     this->numeroDeColumnas = numeroDeColumnas;
 9:     this->numeroDeProfundidad = numeroDeProfundidad;
10:
11:     espacio = new Espacio;
12: }
13:
14:  Tablero::~Tablero() {
15:     delete espacio;
16:  }
17:
18:  int Tablero::getNumeroDeFilas() {
19:     return numeroDeFilas;
20:  }
21:
22:  int Tablero::getNumeroDeColumnas() {
23:     return numeroDeColumnas;
24:  }
25:
26:  int Tablero::getNumeroDeProfundidad() {
27:     return numeroDeProfundidad;
28:  }
29:
30: void Tablero::setCelda(string dato, int x, int y, int z) {
31:     espacio->setCelda(dato, x, y, z);
32:  }
33:
34:  string Tablero::getCelda(int x, int y, int z) {
35:     return espacio->getCelda(x, y, z);
36:  }
37:
38:  int Tablero::getDimensionColumna() {
39:     return espacio->getDimensionColumna();
40:  }
41:
42:  void Tablero::iniciarTablero() {
43:     for(int i = 1; i <= numeroDeProfundidad; i++) {
44:         Plano* paux = new Plano;
45:         for(int j = 1; j <= numeroDeColumnas; j++) {
46:             Columna* caux = new Columna;
47:             for(int k = 1; k <= numeroDeFilas; k++) {
```

```
48:                 caux->agregarCelda(ESPACIO_VACIO);
49:             }
50:             paux->agregarColumna(caux);
51:         }
52:         espacio->agregarPlano(paux);
53:     }
54: }
55:
56: void Tablero::mostrarTablero() {
57:     espacio->imprimirEspacio();
58: }
59:
```

- *Jugador.cpp:*

La clase Jugador contiene la información de un Jugador tal como las cartas disponibles, cartas utilizadas y fichas restantes.

Tiene una variable de tipo estática para contabilizar el número del jugador que al instanciar un nuevo Jugador se incrementa en uno su id.

```
1:  #include "Jugador.h"
2:  #include "Constantes.h"
3:
4:  using namespace std;
5:
6:  Jugador::Jugador(string ficha) {
7:      id = numero;
8:      numero++;
9:
10:     cartaBloquearTurno = false;
11:     cartaJuegaDoble = false;
12:     cartaDobleFicha = false;
13:     cartaInvertirGiro = false;
14:
15:    fichasRestantes  =  MAX_FICHAS;
16:    cartasRestantes  =  MAX_CARTAS;
17:
18:     this->ficha = ficha;
19: }
20:
21:  int Jugador::getNumero() {
22:     return id;
23:  }
24:
25:  string Jugador::getFicha() {
26:     return ficha;
27:  }
28:
29:  int Jugador::getCartasRestantes() {
30:     return cartasRestantes;
31:  }
32:
```

```
33:  int Jugador::getFichasRestantes() {
34:     return fichasRestantes;
35:  }
36:
37:  bool Jugador::getCartaBloquearTurno() {
38:     return cartaBloquearTurno;
39:  }
40:
41:  bool Jugador::getCartaJuegaDoble() {
42:     return cartaJuegaDoble;
43:  }
44:
45:  bool Jugador::getCartaDobleFicha() {
46:     return cartaDobleFicha;
47:  }
48:
49:  bool Jugador::getCartaInvertirGiro() {
50:     return cartaInvertirGiro;
51:  }
52:
53:  void Jugador::usarCartaBloquearTurno() {
54:     cartaBloquearTurno = true;
55:  }
56:
57:  void Jugador::usarCartaJuegaDoble() {
58:     cartaJuegaDoble = true;
59:  }
60:
61:  void Jugador::usarCartaDobleFicha() {
62:     cartaDobleFicha = true;
63:  }
64:
65:  void Jugador::usarCartaInvertirGiro() {
66:     cartaInvertirGiro = true;
67:  }
68:
69:  void Jugador::restarCarta() {
70:     cartasRestantes--;
71:  }
72:
```

- *Jugadores.cpp:*

La clase Jugadores contiene la lista de los jugadores que participan en el Juego, y el jugador actual.

Permite agregar un nuevo jugador, devolver el jugador actual, pasar al siguiente jugador y devolver la cantidad de jugadores.

```
 1:  #include "Jugadores.h"
 2:  #include "Constantes.h"
 3:   using namespace std;
 4:
 5:  Jugadores::Jugadores() {
 6:     jugadores = new Lista<Jugador*>;
 7:     idJugadorActual = 1;
 8:     cantidadJugadores = 0;
 9:  }
10:
11:  Jugadores::~Jugadores() {
12:     delete jugadores;
13:  }
14:
15:  void Jugadores::agregarJugador(string ficha) {
16:     jugadores->agregar(new Jugador(ficha));
17:     cantidadJugadores++;
18:  }
19:
20:  Jugador* Jugadores::getJugadorActual() {
21:     return  jugadores->obtener(idJugadorActual);
22:  }
23:
24:  void Jugadores::pasarJugador(int direccion) {
25:      if (direccion == ASCENDIENTE){
26:          idJugadorActual >= cantidadJugadores ? idJugadorActual = 1:idJug
adorActual ++;
27:      } else {
28:          idJugadorActual <= 1 ? idJugadorActual = cantidadJugadores : idJ
ugadorActual --;
29:      }
30:  }
31:
32:  int Jugadores::getCantidadJugadores() {
33:     return jugadores->contarElementos();
34:  }
35:
36:  int Jugadores::getIdJugadorSiguiente() {
37:     return idJugadorActual >= cantidadJugadores ? 2 : idJugadorActual;
38: }
39:
```

- *Juego.cpp:*

La clase Juego contiene la lista de Jugadores y el Tablero del juego. Esta clase contiene la parte lógica del juego, busca las intersecciones y manipula los datos a ingresar.

Es la clase que se debe instanciar a la hora de crear un nuevo juego.

```
  1:  #include "Juego.h"
  2:  #include "Constantes.h"
  3:
  4:  using namespace std;
  5:
  6:  int Jugador::numero = 1;
  7:
  8: Juego::Juego(int filas, int columnas, int profundidad, int fichasPorCoin
cidir) {
  9:    tablero = new Tablero(filas, columnas, profundidad);
 10:    tablero->iniciarTablero();
 11:
 12:    jugadores = new Jugadores;
 13:
 14:    direccion = ASCENDIENTE;
 15:
 16:    this->fichasPorCoincidir  =  fichasPorCoincidir;
 17:
 18:    finDeJuego = false;
 19: }
 20:
 21:  Juego::~Juego() {
 22:    delete  tablero;
 23:    delete jugadores;
 24:  }
 25:
 26: bool Juego::colocarFicha(string ficha, int columna, int profundidad) {
 27:    int finDeColumna = tablero->getDimensionColumna();
 28:    bool ganadorEncontrado = false, fichaColocada = true;
 29:
 30:    while((tablero-
>getCelda(columna, finDeColumna, profundidad) != ESPACIO_VACIO) && finDeColumn
a >= MIN_COLUMNA) {
 31:        finDeColumna--;
 32:    }
 33:
 34:    if(finDeColumna < MIN_COLUMNA) {
 35:        fichaColocada = false;
 36:    }else{
 37:
 38:          tablero->setCelda(ficha, columna, finDeColumna, profundidad);
 39:
 40:          ganadorEncontrado = buscarGanadorX(columna, finDeColumna, profu
ndidad, ficha) + buscarGanadorY(columna, finDeColumna, profundidad, ficha) +
41:                              buscarGanadorZ(columna, finDeColumna, profu
ndidad, ficha) + buscarGanadorDiagonalA(columna, finDeColumna, profundidad, fi
cha) +
 42:                              buscarGanadorDiagonalB(columna, finDeColumn
a, profundidad, ficha) + buscarGanadorDiagonalC(columna, finDeColumna, profund
idad, ficha) +
 43:                              buscarGanadorDiagonalD(columna, finDeColumn
a, profundidad, ficha) + buscarGanadorDiagonalE(columna, finDeColumna, profund
idad, ficha) +
 44:                              buscarGanadorDiagonalF(columna, finDeColumn
a, profundidad, ficha) + buscarGanadorDiagonalG(columna, finDeColumna, profund
```

```
idad, ficha) +
 45:                                 buscarGanadorDiagonalH(columna, finDeColumn
a, profundidad, ficha);
 46:
 47:          if(ganadorEncontrado) {
 48:              cout << endl <<"            **** Gano el Jugador " << jugad
ores->getJugadorActual()->getNumero() << " ****" << endl << endl;
 49:              finDeJuego = true;
 50:          }
 51:    }
 52:
 53:    return fichaColocada;
 54:  }
 55:
 56:  void Juego::mostrarTablero() {
 57:    tablero->mostrarTablero();
 58:  }
 59:
 60:  void Juego::agregarJugador(string ficha) {
 61:    jugadores->agregarJugador(ficha);
 62:  }
 63:
 64:  Jugador* Juego::getJugadorActual() {
 65:    return  jugadores->getJugadorActual();
 66:  }
 67:
 68:  void Juego::pasarJugador(int direccion) {
 69:    jugadores->pasarJugador(direccion);
 70:  }
 71:
 72:  Tablero* Juego::getTablero() {
 73:    return tablero;
 74:  }
 75:
 76: void Juego::seleccionarColumna(int columna, int profundidad, int cartaSe
leccionada) {
 77:    if(columna > VACIO && columna <= tablero->getDimensionColumna()) {
 78:
 79:        if(this->colocarFicha(jugadores->getJugadorActual()-
>getFicha(), columna, profundidad)) {
 80:
 81:            if(cartaSeleccionada == DOBLE_FICHA) {
 82:                this->colocarFicha(jugadores->getJugadorActual()-
>getFicha(), columna, profundidad);
 83:            }
 84:
 85:            if(cartaSeleccionada != JUGAR_NUEVAMENTE) {
 86:                jugadores->pasarJugador(direccion);
 87:            }
 88:
 89:        } else {
 90:            cout << "Columna completa, selecciona otra" << endl;
 91:        }
 92:    } else {
 93:        cout << "** La columna debe estar entre 1 y " << tablero-
>getDimensionColumna()  << " ** " << endl;
 94:    }
 95:  }
 96:
 97:  int Juego::pedirCarta() {
```

```
 98:     int cartaSeleccionada = NINGUNA;
 99:     bool usarCarta = false;
100:
101:     cout << "_____" << endl <<
 endl;
102:     cout << "Turno de jugador: " << jugadores->getJugadorActual()-
>getNumero() << ", ficha " << jugadores->getJugadorActual()-
>getFicha()<< endl;
103:     cout << "Cartas restantes: " << jugadores->getJugadorActual()-
>getCartasRestantes() << endl;
104:
105:     while(!usarCarta) {
106:         if(jugadores->getJugadorActual()->getCartasRestantes() > VACIO) {
107:
108:             cout << "   1) [Carta Bloquear Turno]";
109:
110:             if(jugadores->getJugadorActual()->getCartaBloquearTurno()) {
111:                 cout << " ------> (NO DISPONIBLE)";
112:             }
113:
114:             cout << endl << "   2) [Carta Juega Doble]";
115:
116:             if(jugadores->getJugadorActual()->getCartaJuegaDoble()) {
117:                 cout << " -------- > (NO DISPONIBLE)";
118:             }
119:
120:             cout << endl << "   3) [Carta Doble Ficha]";
121:
122:             if(jugadores->getJugadorActual()->getCartaDobleFicha()) {
123:                 cout << " ----------> (NO DISPONIBLE)";
124:             }
125:
126:             cout << endl << "   4) [Carta Invertir Giro]";
127:
128:             if(jugadores->getJugadorActual()->getCartaInvertirGiro()) {
129:                 cout << " ----------> (NO DISPONIBLE)";
130:             }
131:
132:             cout << endl << "   0) [No usar carta]";
133:
134:             cout << endl << "  Ingresar carta       > ";
135:
136:             cin >> cartaSeleccionada;
137:
138:
139:
140:               //PUEDE PONERSE UN SWITCH (ABAJO)
141:
142:             if(cartaSeleccionada == SALTEAR_TURNO) {
143:                 if(!jugadores->getJugadorActual()-
>getCartaBloquearTurno()) {
144:                     jugadores->getJugadorActual()-
>usarCartaBloquearTurno();
145:                     jugadores->getJugadorActual()->restarCarta();
146:                     usarCarta = true;
147:                 } else {
148:                     cartaSeleccionada = NINGUNA;
149:                 }
150:
```

```
151:             } else if(cartaSeleccionada == JUGAR_NUEVAMENTE) {
152:
153:                 if(!jugadores->getJugadorActual()-
>getCartaJuegaDoble()) {
154:                     jugadores->getJugadorActual()->usarCartaJuegaDoble();
155:                     jugadores->getJugadorActual()->restarCarta();
156:                     usarCarta = true;
157:                 } else {
158:                     cartaSeleccionada = NINGUNA;
159:                 }
160:             } else if(cartaSeleccionada == DOBLE_FICHA) {
161:
162:                 if(!jugadores->getJugadorActual()->getCartaDobleFicha())  {
163:                     jugadores->getJugadorActual()->usarCartaDobleFicha();
164:                     jugadores->getJugadorActual()->restarCarta();
165:                     usarCarta = true;
166:                 } else {
167:                     cartaSeleccionada = NINGUNA;
168:                 }
169:             } else if(cartaSeleccionada == INVERTIR_GIRO) {
170:
171:                 if(!jugadores->getJugadorActual()-
>getCartaInvertirGiro()) {
172:                     jugadores->getJugadorActual()-
>usarCartaInvertirGiro();
173:                     jugadores->getJugadorActual()->restarCarta();
174:                     usarCarta = true;
175:                 } else {
176:                     cartaSeleccionada = NINGUNA;
177:                 }
178:             } else if(cartaSeleccionada == SALIR) {
179:                 usarCarta = true;
180:             }
181:
182:         } else {
183:             usarCarta = true;
184:             cartaSeleccionada = NINGUNA;
185:         }
186:     }
187:
188:     return cartaSeleccionada;
189:  }
190:
191: void Juego::pedirPosicionesFicha(int cartaSeleccionada, int* input){
192:
193:       cout << "  Ingresar columna     > ";
194:       cin >> *input;
195:       int x = *input;
196:
197:       cout << "  Ingresar profundidad > ";
198:       cin >> *input;
199:       int z = *input;
200:
201:       seleccionarColumna(x, z, cartaSeleccionada);
202: }
203:
204: void Juego::interaccionesCartas(int cartaSeleccionada, int* input){
205:
206:       if(cartaSeleccionada == SALTEAR_TURNO) {
```

```
207:           cout << "El jugador " << getIdJugadorSiguiente() + 1 << " pierde
 un turno" << endl;
208:           pasarJugador(direccion);
209:       }
210:
211:       if(cartaSeleccionada == JUGAR_NUEVAMENTE) {
212:           mostrarTablero();
213:
214:           cout << "Jugador " << getJugadorActual()-
>getNumero() << " juega nuevamente: " << endl;
215:
216:           pedirPosicionesFicha(cartaSeleccionada,  &*input);
217:
218:           pasarJugador(direccion);
219:       }
220:
221:       if (cartaSeleccionada == INVERTIR_GIRO){
222:           direccion = (direccion == ASCENDIENTE ? DESCENDIENTE : ASCENDIE
NTE);
223:           /*
224:            * Como se llama 'pedirPosicionesFicha' antes que 'interacciones
Cartas', se pasa un jugador, antes desde pedirPosicionesFicha
225:            * llamo dos veces, 1ero vuelve al jugador actual y despues retr
ocede
226:            */
227:           pasarJugador(direccion);
228:           pasarJugador(direccion);
229:       }
230:   }
231:
232:   int Juego::getCantidadJugadores() {
233:     return jugadores->getCantidadJugadores();
234:   }
235:
236:   int Juego::getIdJugadorSiguiente() {
237:     return jugadores->getIdJugadorSiguiente();
238:   }
239:
240:   bool Juego::buscarGanadorX(int x, int y, int z, string ficha) {
241:     int dimension = tablero->getNumeroDeColumnas();
242:     bool conexion = false;
243:     int contador = 0;
244:
245:     for(int i = 1; i <= dimension; i++) {
246:         if(conexion) {
247:             if(tablero->getCelda(i, y, z) == ficha) {
248:                 contador++;
249:             } else {
250:                 conexion = false;
251:                 contador = 0;
252:             }
253:         } else {
254:             if(tablero->getCelda(i, y, z) == ficha) {
255:                 conexion = true;
256:                 contador++;
257:             }
258:         }
259:
260:         if(contador >= fichasPorCoincidir) {
```

```
261:             return true;
262:         }
263:     }
264:
265:     return false;
266:   }
267:
268:   bool Juego::buscarGanadorY(int x, int y, int z, string ficha) {
269:     int dimension = tablero->getNumeroDeFilas();
270:     bool conexion = false;
271:     int contador = 0;
272:
273:     for(int i = 1; i <= dimension; i++) {
274:         if(conexion) {
275:             if(tablero->getCelda(x, i, z) == ficha) {
276:                 contador++;
277:             } else {
278:                 conexion = false;
279:                 contador = 0;
280:             }
281:         } else {
282:             if(tablero->getCelda(x, i, z) == ficha) {
283:                 conexion = true;
284:                 contador++;
285:             }
286:         }
287:
288:         if(contador >= fichasPorCoincidir) {
289:             return true;
290:         }
291:     }
292:
293:     return false;
294:   }
295:
296:   bool Juego::buscarGanadorZ(int x, int y, int z, string ficha) {
297:     int dimension = tablero->getNumeroDeProfundidad();
298:     bool conexion = false;
299:     int contador = 0;
300:
301:     for(int i = 1; i <= dimension; i++) {
302:         if(conexion) {
303:             if(tablero->getCelda(x, y, i) == ficha) {
304:                 contador++;
305:             } else {
306:                 conexion = false;
307:                 contador = 0;
308:             }
309:         } else {
310:             if(tablero->getCelda(x, y, i) == ficha) {
311:                 conexion = true;
312:                 contador++;
313:             }
314:         }
315:
316:         if(contador >= fichasPorCoincidir) {
317:             return true;
318:         }
319:     }
```

```
320:
321:    return false;
322:  }
323:
324: bool Juego::buscarGanadorDiagonalA(int x, int y, int z, string ficha) {
325:    bool conexion = false;
326:    int contador = 0;
327:
328:  int  xA,  yA,  zA;
329:  int  xB,  yB,  zB;
330:
331:    xA = x;
332:    yA = y;
333:    zA = z;
334:
335:    while(xA > 1 && yA < tablero->getNumeroDeFilas() && zA > 1) {
336:        xA--;
337:        yA++;
338:        zA--;
339:    }
340:
341:    xB = xA;
342:    yB = yA;
343:    zB = zA;
344:
345:    while(xB <= tablero-
>getNumeroDeColumnas() && yB >= 1 && zB <= tablero-
>getNumeroDeProfundidad()) {
346:
347:        if(conexion) {
348:            if(tablero->getCelda(xB, yB, zB) == ficha) {
349:                contador++;
350:            } else {
351:                conexion = false;
352:                contador = 0;
353:            }
354:        } else {
355:            if(tablero->getCelda(xB, yB, zB) == ficha) {
356:                conexion = true;
357:                contador++;
358:            }
359:        }
360:
361:        if(contador >= fichasPorCoincidir) {
362:            return true;
363:        }
364:
365:        xB++;
366:        yB--;
367:        zB++;
368:    }
369:
370:    return false;
371:  }
372:
373: bool Juego::buscarGanadorDiagonalB(int x, int y, int z, string ficha) {
374:    bool conexion = false;
375:    int contador = 0;
376:
```

```
377:  int  xA,  yA,  zA;
378:  int  xB,  yB,  zB;
379:
380:     xA = x;
381:     yA = y;
382:     zA = z;
383:
384:     while(xA < tablero->getNumeroDeColumnas() && yA < tablero-
>getNumeroDeFilas() && zA < tablero->getNumeroDeProfundidad()) {
385:         xA++;
386:         yA++;
387:         zA++;
388:     }
389:
390:     xB = xA;
391:     yB = yA;
392:     zB = zA;
393:
394:     while(xB >= 1 && yB >= 1 && zB >= 1) {
395:
396:         if(conexion) {
397:             if(tablero->getCelda(xB, yB, zB) == ficha) {
398:                 contador++;
399:             } else {
400:                 conexion = false;
401:                 contador = 0;
402:             }
403:         } else {
404:             if(tablero->getCelda(xB, yB, zB) == ficha) {
405:                 conexion = true;
406:                 contador++;
407:             }
408:         }
409:
410:         if(contador >= fichasPorCoincidir) {
411:             return true;
412:         }
413:
414:         xB--;
415:         yB--;
416:         zB--;
417:     }
418:
419:     return false;
420:  }
421:
422: bool Juego::buscarGanadorDiagonalC(int x, int y, int z, string ficha) {
423:     bool conexion = false;
424:     int contador = 0;
425:
426:  int  xA,  yA,  zA;
427:  int  xB,  yB,  zB;
428:
429:     xA = x;
430:     yA = y;
431:     zA = z;
432:
433:     while(xA < tablero->getNumeroDeColumnas() && yA > 1 && zA > 1) {
434:         xA++;
```

```
435:         yA--;
436:         zA--;
437:     }
438:
439:     xB = xA;
440:     yB = yA;
441:     zB = zA;
442:
443:     while(xB >= 1 && yB <= tablero->getNumeroDeFilas() && zB <= tablero-
>getNumeroDeProfundidad()) {
444:
445:         if(conexion) {
446:             if(tablero->getCelda(xB, yB, zB) == ficha) {
447:                 contador++;
448:             } else {
449:                 conexion = false;
450:                 contador = 0;
451:             }
452:         } else {
453:             if(tablero->getCelda(xB, yB, zB) == ficha) {
454:                 conexion = true;
455:                 contador++;
456:             }
457:         }
458:
459:         if(contador >= fichasPorCoincidir) {
460:             return true;
461:         }
462:
463:         xB--;
464:         yB++;
465:         zB++;
466:     }
467:
468:     return false;
469:  }
470:
471: bool Juego::buscarGanadorDiagonalD(int x, int y, int z, string ficha) {
472:     bool conexion = false;
473:     int contador = 0;
474:
475:  int  xA,  yA,  zA;
476:  int  xB,  yB,  zB;
477:
478:     xA = x;
479:     yA = y;
480:     zA = z;
481:
482:     while(xA > 1 && yA > 1 && zA < tablero->getNumeroDeProfundidad()) {
483:         xA--;
484:         yA--;
485:         zA++;
486:     }
487:
488:     xB = xA;
489:     yB = yA;
490:     zB = zA;
491:
```

```
492:    while(xB <= tablero->getNumeroDeColumnas() && yB <= tablero-
>getNumeroDeFilas() && zB >= 1) {
493:
494:        if(conexion) {
495:            if(tablero->getCelda(xB, yB, zB) == ficha) {
496:                contador++;
497:            } else {
498:                conexion = false;
499:                contador = 0;
500:            }
501:        } else {
502:            if(tablero->getCelda(xB, yB, zB) == ficha) {
503:                conexion = true;
504:                contador++;
505:            }
506:        }
507:
508:        if(contador >= fichasPorCoincidir) {
509:            return true;
510:        }
511:
512:        xB++;
513:        yB++;
514:        zB--;
515:    }
516:
517:    return false;
518:  }
519:
520: bool Juego::buscarGanadorDiagonalE(int x, int y, int z, string ficha) {
521:    bool conexion = false;
522:    int contador = 0;
523:
524:    int yA, zA;
525:    int yB, zB;
526:
527:    yA = y;
528:    zA = z;
529:
530:    while(yA > 1 && zA < tablero->getNumeroDeProfundidad()) {
531:        yA--;
532:        zA++;
533:    }
534:
535:    yB = yA;
536:    zB = zA;
537:
538:    while(yB <= tablero->getNumeroDeFilas() && zB >= 1) {
539:
540:        if(conexion) {
541:            if(tablero->getCelda(x, yB, zB) == ficha) {
542:                contador++;
543:            } else {
544:                conexion = false;
545:                contador = 0;
546:            }
547:        } else {
548:            if(tablero->getCelda(x, yB, zB) == ficha) {
549:                conexion = true;
```

```
550:                 contador++;
551:             }
552:         }
553:
554:         if(contador >= fichasPorCoincidir) {
555:             return true;
556:         }
557:
558:         yB++;
559:         zB--;
560:     }
561:
562:     return false;
563:  }
564:
565: bool Juego::buscarGanadorDiagonalF(int x, int y, int z, string ficha) {
566:     bool conexion = false;
567:     int contador = 0;
568:
569:     int yA, zA;
570:     int yB, zB;
571:
572:     yA = y;
573:     zA = z;
574:
575:     while(yA < tablero->getNumeroDeFilas() && zA < tablero-
>getNumeroDeProfundidad()) {
576:         yA++;
577:         zA++;
578:     }
579:
580:     yB = yA;
581:     zB = zA;
582:
583:     while(yB >= 1 && zB >= 1) {
584:
585:         if(conexion) {
586:             if(tablero->getCelda(x, yB, zB) == ficha) {
587:                 contador++;
588:             } else {
589:                 conexion = false;
590:                 contador = 0;
591:             }
592:         } else {
593:             if(tablero->getCelda(x, yB, zB) == ficha) {
594:                 conexion = true;
595:                 contador++;
596:             }
597:         }
598:
599:         if(contador >= fichasPorCoincidir) {
600:             return true;
601:         }
602:
603:         yB--;
604:         zB--;
605:     }
606:
607:     return false;
```

```
608:  }
609:
610: bool Juego::buscarGanadorDiagonalG(int x, int y, int z, string ficha) {
611:    bool conexion = false;
612:    int contador = 0;
613:
614:    int xA, yA;
615:    int xB, yB;
616:
617:    xA = x;
618:    yA = y;
619:
620:    while(xA < tablero->getNumeroDeColumnas() && yA > 1) {
621:        xA++;
622:        yA--;
623:    }
624:
625:    xB = xA;
626:    yB = yA;
627:
628:    while(xB >= 1 && yB <= tablero->getNumeroDeFilas()) {
629:
630:        if(conexion) {
631:            if(tablero->getCelda(xB, yB, z) == ficha) {
632:                contador++;
633:            } else {
634:                conexion = false;
635:                contador = 0;
636:            }
637:        } else {
638:            if(tablero->getCelda(xB, yB, z) == ficha) {
639:                conexion = true;
640:                contador++;
641:            }
642:        }
643:
644:        if(contador >= fichasPorCoincidir) {
645:            return true;
646:        }
647:
648:        xB--;
649:        yB++;
650:    }
651:
652:    return false;
653:  }
654:
655: bool Juego::buscarGanadorDiagonalH(int x, int y, int z, string ficha) {
656:
657:    bool conexion = false;
658:    int contador = 0;
659:
660:    int xA, yA;
661:    int xB, yB;
662:
663:    xA = x;
664:    yA = y;
665:
```

```
666:    while(xA < tablero->getNumeroDeColumnas() && yA < tablero-
>getNumeroDeFilas()) {
667:        xA++;
668:        yA++;
669:    }
670:
671:    xB = xA;
672:    yB = yA;
673:
674:    while(xB >= 1 && yB >= 1) {
675:
676:        if(conexion) {
677:            if(tablero->getCelda(xB, yB, z) == ficha) {
678:                contador++;
679:            } else {
680:                conexion = false;
681:                contador = 0;
682:            }
683:        } else {
684:            if(tablero->getCelda(xB, yB, z) == ficha) {
685:                conexion = true;
686:                contador++;
687:            }
688:        }
689:
690:        if(contador >= fichasPorCoincidir) {
691:            return true;
692:        }
693:
694:        xB--;
695:        yB--;
696:    }
697:
698:    return false;
699:  }
700:
701:  bool Juego::juegoFinalizado() {
702:    return finDeJuego;
703:  }
704:
```