Universidad ORT

Facultad de Ingeniería

Programación 2

Obligatorio 2 – Grupo M2A

Rodolfo Agustín Silva
(187061)

Andrés Lacaño
(158910)

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

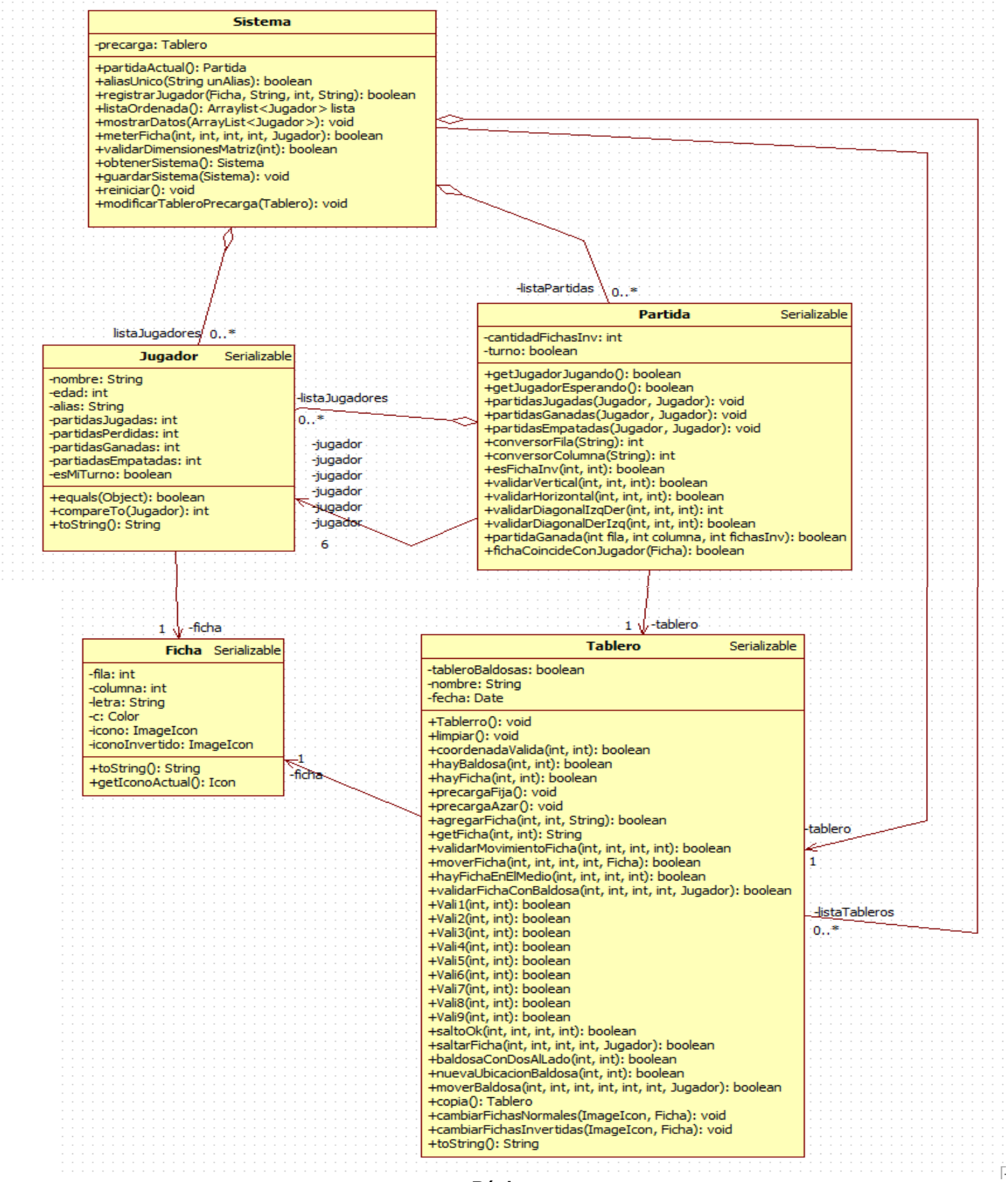
(158910) (187061)

Junio 2015

Índice

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

Diagrama de clases UML



Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

Datos de prueba

Descripción del dato a probar	Resultado esperado	Resultado obtenido
Dentro de la ventana Registrar Jugador		
" " (como nombre) " " (como edad) " " (como alias)	Mensaje pidiendo de ingresar los datos del jugador	ok
" " (como nombre) "20" (como edad) " " (como alias) (ver imagen 1)	Mensaje pidiendo de ingresar los datos del jugador	ok
"Jugador1" (como nombre) "20" (como edad) " " (como alias)	Mensaje pidiendo de ingresar los datos del jugador	ok
"Jugador1" (como nombre) "%" (como edad) "Jug1" (como alias) (ver imagen 2)	Mensaje diciendo que la edad no tiene formato valido	ok
"Jugador1" (como nombre) "20" (como edad) "Jug1" (como alias) (ver imagen 3)	Mensaje diciendo que el jugador se registro	ok
"Jugador2" (como nombre) "23" (como edad) "Jug1" (como alias) (ver imagen 4)	Mensaje diciendo que el alias ya esta en uso y se solo el alias para elegir otro	ok
Dentro de la ventana ranking		
Se intenta ingresar sin ninguna jugador en el sistema (ver imagen 5)	Mensaja diciendo que no se pueda ingresar porque no hay jugadores resgistrados	ok
Se ingresa a esta opcion con dos jugadores en el sistema, ambos con cero partidas	Muestra el ranking	ok

jugadas (ver imagen 6)		
Se ingresa a esta opcion con dos jugadores en el sistema, ambos con una partida jugada. El jugador1 con una partida ganada (ver imagen 7)	Muestra el ranking ordenado por partidas ganadas	ok
Se ingresa a esta opcion con dos jugadores en el sistema, ambos con tres partida jugada. El jugador1 con una partida ganada y el jugador2 con dos partidas ganadas (ver imagen 8)	Muestra el ranking ordenado por partidas ganadas	ok
Dentro de la ventana precarga		
Se selecciona precarga fija	El sistema acomoda las fichas de manera predeterminada en el tablero y se vuelve al menu principal	ok
Se selecciona precarga al azar	El sistema acomoda las fichas en el tablero aleatoriamente en el tablero y se vuelve al menu principal	ok
Se selecciona precarga por txt (ver imagen 9)	Se abre un fileChooser donde el usuario elijer un archivo .txt y se cargan las fichas en funcion del archivo seleccionado	ok
Se selecciona precarga manual	Se abre una nueva ventana donde se muestra el tablero solo con baldosas y donde se puede ingresar las posiciones de las fichas	ok
Dentro de la ventana Precarga Manual		

“E3” (posicion de ficha)	Mensaje diciendo que la posicion no es valida	ok
“F10” (posicion de ficha) (ver imagen 10)	Mensaje diciendo que la posicion no es valida	ok
“E&” (posicion de ficha)	Mensaje diciendo que la posicion no es valida	ok
“%6” (posicion de ficha) (ver imagen 11)	Mensaje diciendo que la posicion no es valida	ok
“e5” (posicion de ficha)	Mensaje diciendo que la posicion no es valida	ok
Se clickea en la cruz que cierra la ventana (ver imagen 12)	Mensaje que dice que las fichas se van a poner sobre el tablero al azar	ok
“E7” (posicion de ficha) (ver imagen 13)	Se actualiza el tablero mostrando la posicion de donde se puso la ficha	ok
Se clickea sobre el boton Salir sin haber terminado de poner las fichas en el tablero (ver imagen 14)	Mensaje que dice que las fichas no se terminaron de poner, por tanto se van a distribuir al azar	ok
Luego de haber ingresado las 6 fichas de ada jugador, se clickea sobre el boton Ingresar nuevamente	No realiza ninguna accion, pues se bloquea una ves se pusieron las fichas sobre el tablero	ok
Luego de haber ingresado las 6 fichas de ada jugador, se clickea sobre el boton Salir	Las fichas se distribuyen sobre el tablero de la forma que el jugador las ingreso	ok
Dentro de la ventana Jugar Baldosas		
Se selecciona jugar baldosas	Se abre otra ventana con las configuraciones de la partida	ok
Dentro de la ventana Configuracion de Partida		

Se selecciona jugar baldosas habiendo selecionada los mismo jugadores y “2” (cantidad fichas invertidas 1-6) y se preciona el boton jugar	Mensaje diciendo que se tiene que elegir un jugador diferente	ok
Se selecciona jugar baldosas habiendo selecionada dos jugadores distintos y “9” (cantidad fichas invertidas 1-6) y se preciona el boton jugar	Mensaje diciendo que la cantidad de fichas invertidas no es valida	ok
Se selecciona jugar baldosas habiendo selecionada dos jugadores distintos y “&” (cantidad fichas invertidas 1-6) y se preciona el boton jugar	Mensaje diciendo que la cantidad de fichas invertidas no es valida	ok
Se selecciona jugar baldosas habiendo selecionada dos jugadores distintos y “3” (cantidad fichas invertidas 1-6) y se preciona el boton jugar	Se abre una nueva ventana con el tablero de juego y las jugadas para realizar	ok
Dentro de la ventana TableroJ		
Recordamos que: Turno Jugador 1 con fichas blancas Turno Jugador 2 con fichas rojas		
Para los siguientes casos se elegio la precarga fija		
Selecciona mover fichas y clickea una ficha del	Mensaje diciendo que no se puede elegir una ficha del	ok

oponente (F5-F6)	oponente	
Selecciona saltar ficha y clickea una ficha del oponente (F7-F9)	Mensaje diciendo que no se puede elegir una ficha del oponente	ok
Selecciona mover baldosa y clickea una ficha del oponente (E9-F4-F5)	Mensaje diciendo que no se puede elegir una ficha del oponente	ok
Selecciona mover ficha y clickea una ficha suya (E7-E8)	Se realiza la jugada	ok
Selecciona mover ficha y clickea una ficha suya (E7-E9)	Mensaje diciendo que no se puede realizar ese movimiento	ok
Selecciona saltar ficha y clickea una ficha suya (E7-H7)	Mensaje diciendo que no se puede realizar ese movimiento	ok
Selecciona mover baldosa y clickea una ficha del oponente (E9-E4-E7)	Mensaje diciendo que no se puede realizar ese movimiento	ok
Selecciona cambiar icono de fichas normales (ver imagen 15)	Se abre un FilaChooser filtado por .jgp y se ponen la imagen sobre sus fichas no invertidas y no se cambia de turno	ok
Selecciona cambiar icono de fichas invertidas	Se abre un FilaChooser filtado por .jgp y se ponen la imagen sobre sus fichas invertidas y no se cambia de turno	ok
Selecciona reiniciar partida	Se vuelve a arrancar la partida, con la fichas en sus posiciones iniciales, los mismo jugadores y la misma cantidad de fichas invertidas	ok
Selecciona rendirse (ver imagen 16)	Mensaja diciendo que el jugador 2 es el ganador	ok

Clickea en la cruz para cerrar la ventana	Automaticamente pierde la partida y se muestra un mensaje que dice que el jugador 2 es el ganador	ok
Para los siguientes casos se se elige precarga manual y 1 ficha invertida para ganar Jugadores: Jugador1 Jugador2 Jugadas: S(Saltar Ficha), M(Mover Ficha), B(Mover Baldosa)		
S(H5-H7), M(H6-H5), S(G9-F9), M(F7-F6), M(F8-F7) (ver imagen 17)	Mensaje diciendo que el jugador1 es el ganador	ok
M(E7-E8), S(F7-H7), M(H5-G6), M(G8-H8), M(F8-G8), M(G5-H5) (ver imagen 18)	Mensaje diciendo que el jugador2 es el ganador	

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

imagen 1

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

imagen 2

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

imagen 3


Registrar Jugador
Nombre
Jugador2
Edad
23
Alias
Jug1
Registrar
Registrar y
Mensaje
Error el alias ya esta en uso
Aceptar

imagen 4

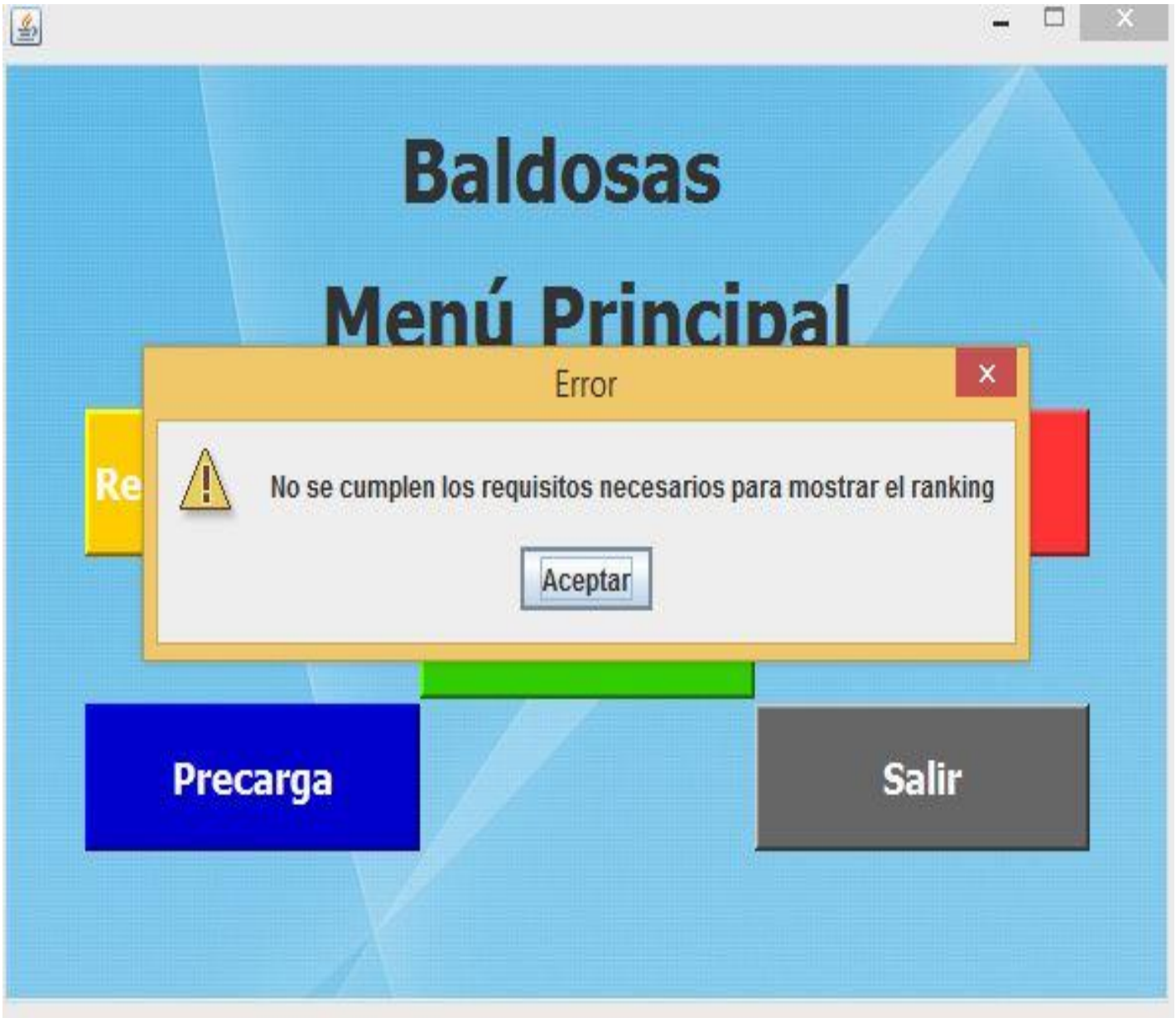

Baldosas
Menú Principal
Error
No se cumplen los requisitos necesarios para mostrar el ranking
Aceptar
Re
Precarga
Salir

imagen 5

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

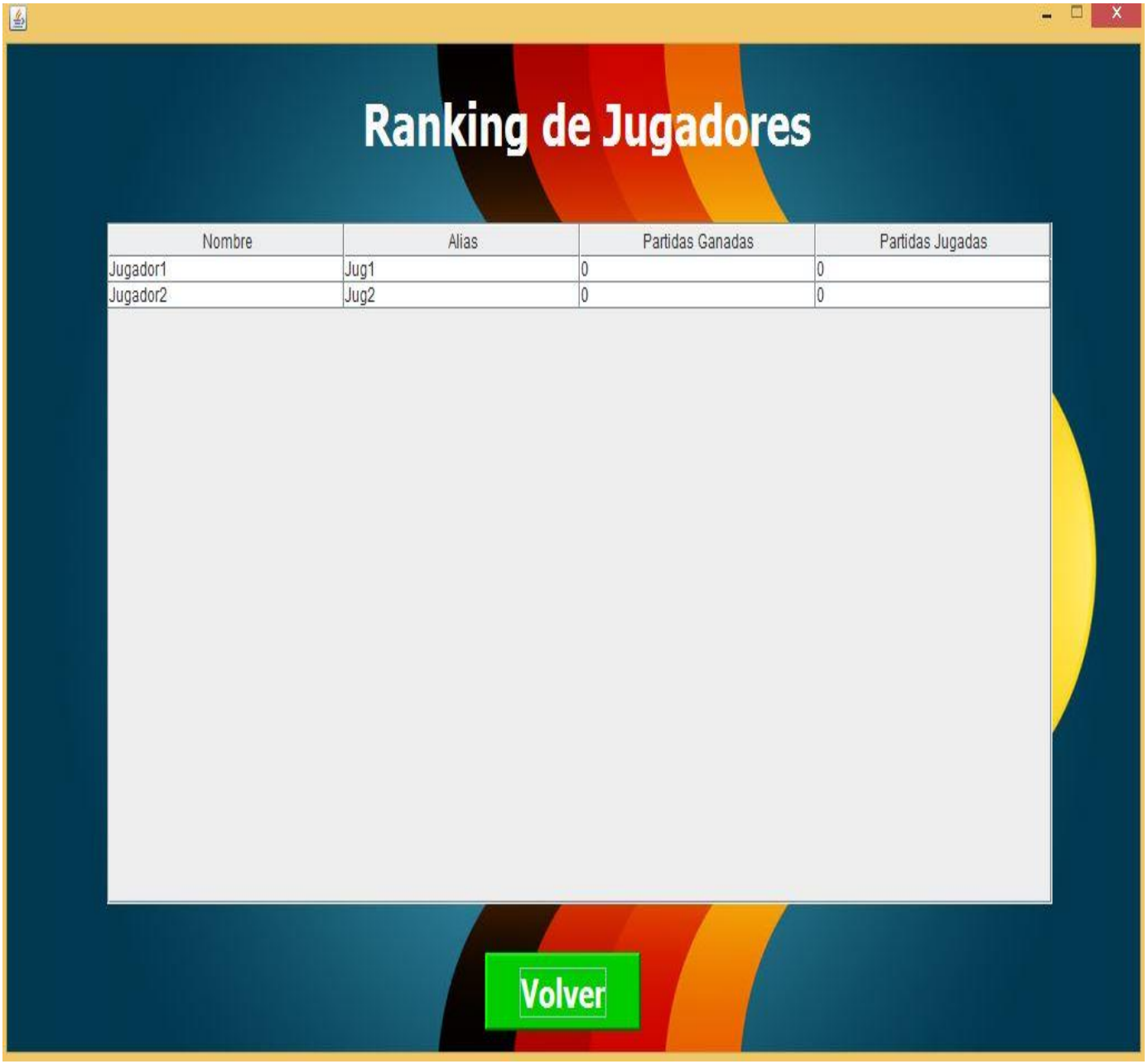


imagen 6


Ranking de Jugadores
Nombre
Alias
Partidas Ganadas
Partidas Jugadas
Jugador1
Jug1
1
1
Jugador 2
Jug2
0
1

imagen 7

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

imagen 8

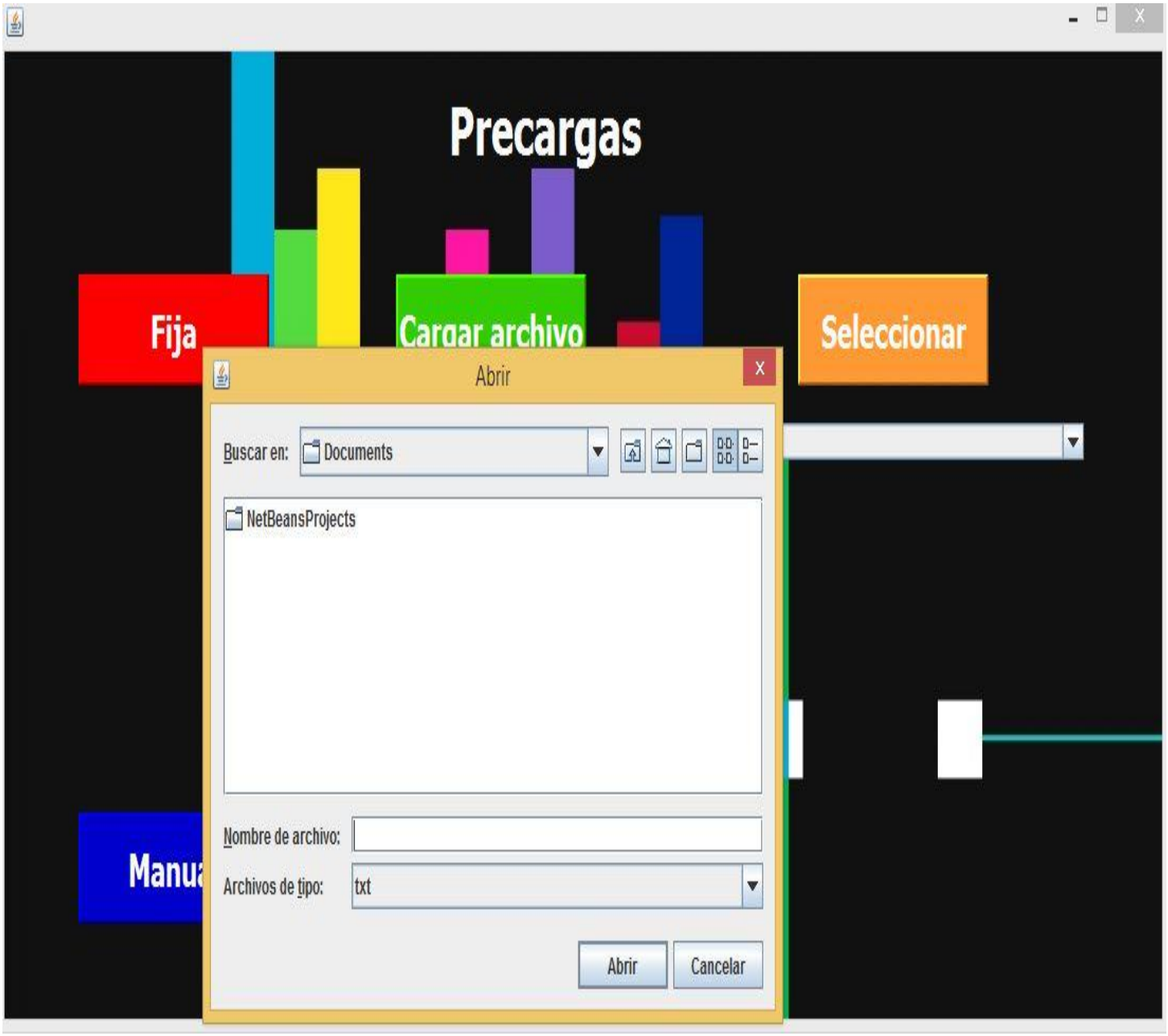

Precargas
Fija
Cargar archivo
Seleccionar
Abrir
Buscar en:
Documents
NetBeansProjects
Nombre de archivo:
Archivos de tipo:
txt
Abrir
Cancelar

imagen 9

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

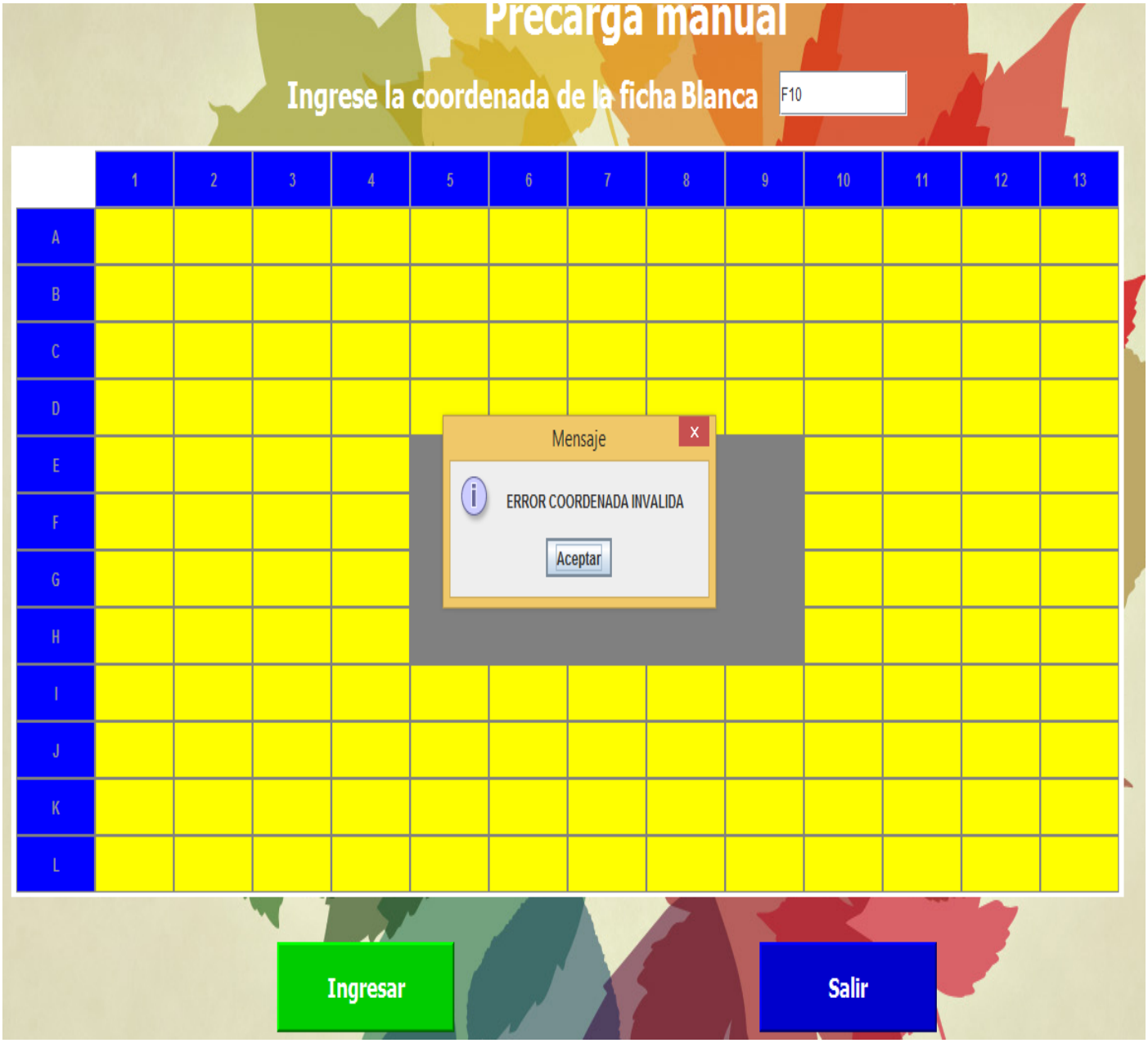


imagen 10


Precarga manual
Ingrese la coordenada de la ficha Blanca
%6
1 2 3 4 5 6 7 8 9 10 11 12 13
A B C D E F G H I J K L
Mensaje
ERROR COORDENADA INVALIDA
Aceptar

imagen 11


Precarga manual
Ingrese la coordenada de la ficha Blanca
1
2
3
4
5
6
7
8
9
10
11
12
13
A
B
C
D
E
F
G
H
I
J
K
L
Mensaje
Se pondran las fichas al azar
Aceptar
Ingresar
Salir

imagen 12

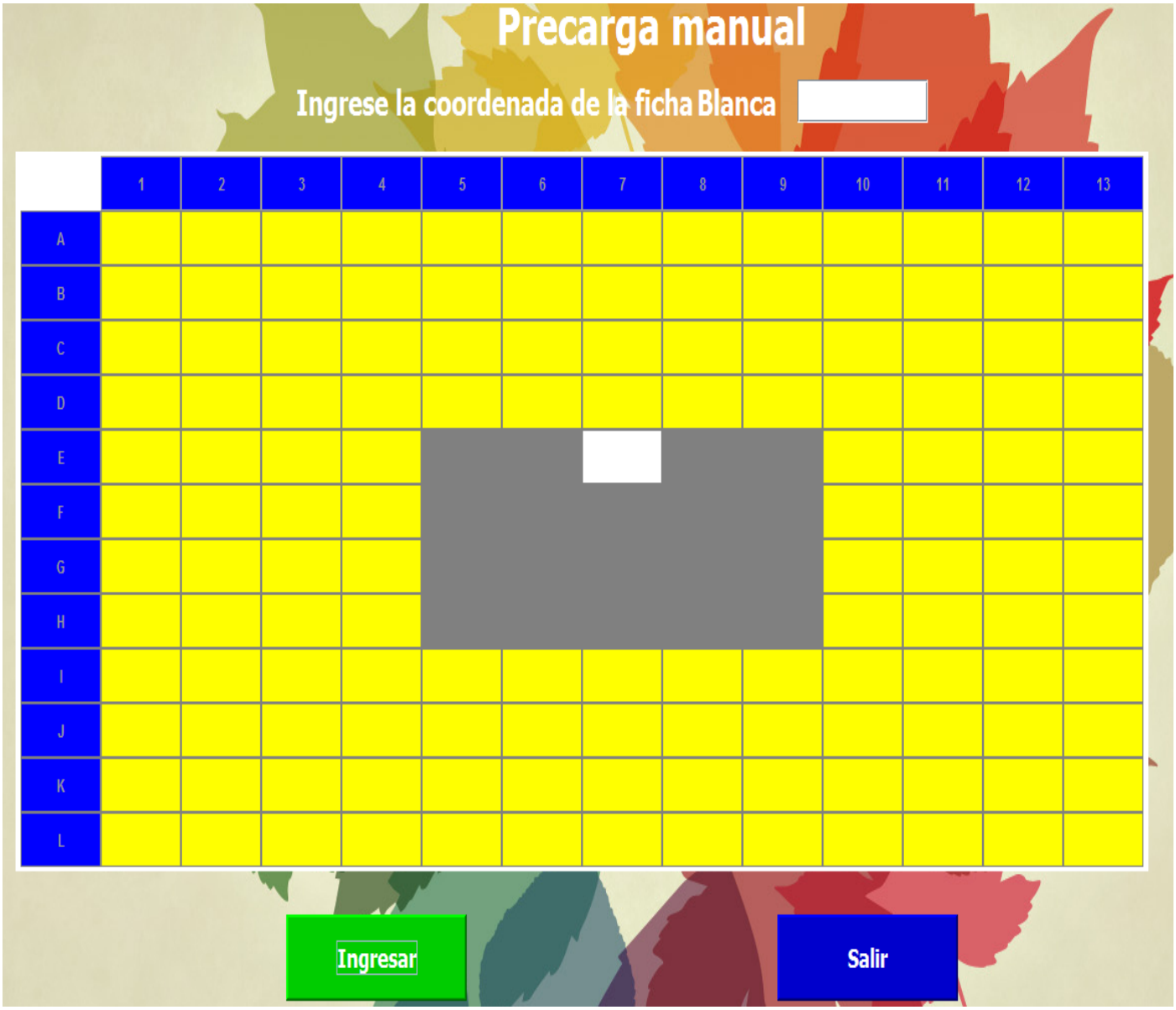

Precarga manual
Ingrese la coordenada de la ficha Blanca
1 2 3 4 5 6 7 8 9 10 11 12 13
A B C D E F G H I J K L
Ingresar
Salir

imagen 13

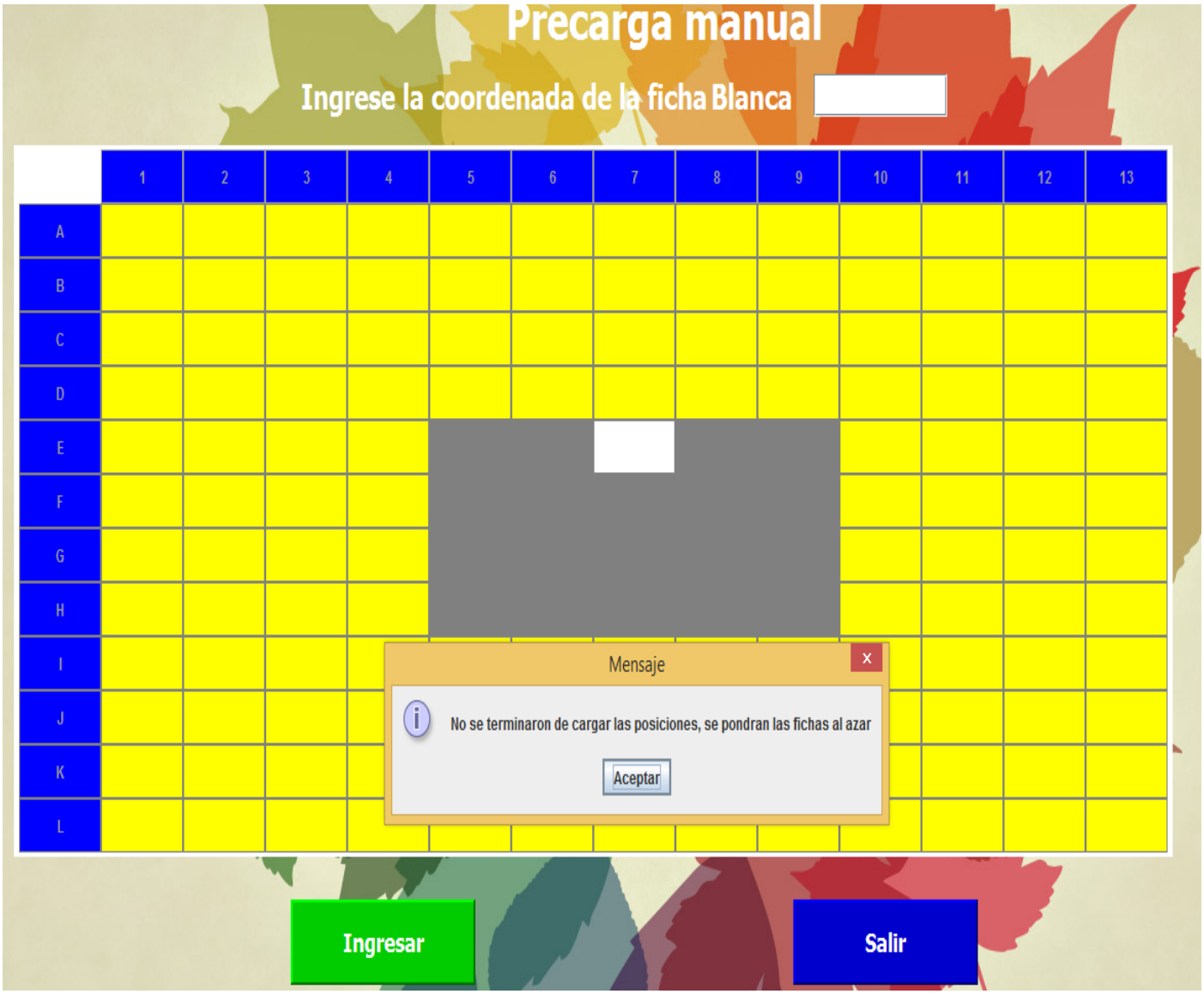

Precarga manual
Ingrese la coordenada de la ficha Blanca
1 2 3 4 5 6 7 8 9 10 11 12 13
A B C D E F G H I J K L
Mensaje
No se terminaron de cargar las posiciones, se pondran las fichas al azar
Aceptar
Ingresar
Salir

imagen 14

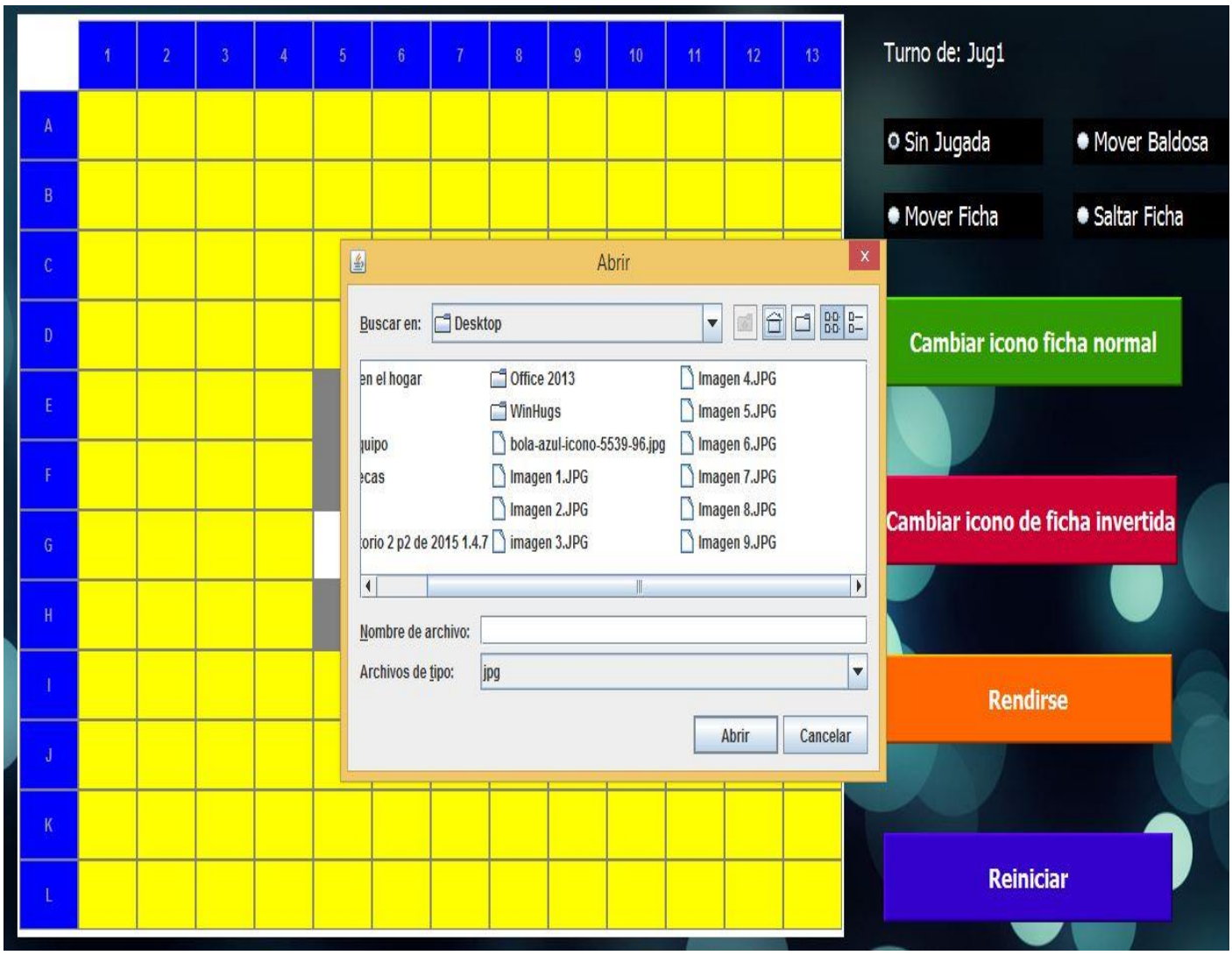

1 2 3 4 5 6 7 8 9 10 11 12 13
A B C D E F G H I J K L
Turno de: Jug1
Sin Jugada
Mover Baldosa
Mover Ficha
Saltar Ficha
Cambiar icono ficha normal
Cambiar icono de ficha invertida
Rendirse
Reiniciar
Abrir
Buscar en:
Desktop
en el hogar
Office 2013
WinHugs
bola-azul-icono-5539-96.jpg
Imagen 1.JPG
Imagen 2.JPG
imagen 3.JPG
Imagen 4.JPG
Imagen 5.JPG
Imagen 6.JPG
Imagen 7.JPG
Imagen 8.JPG
Imagen 9.JPG
Nombre de archivo:
Archivos de tipo:
jpg
Abrir
Cancelar

imagen 15

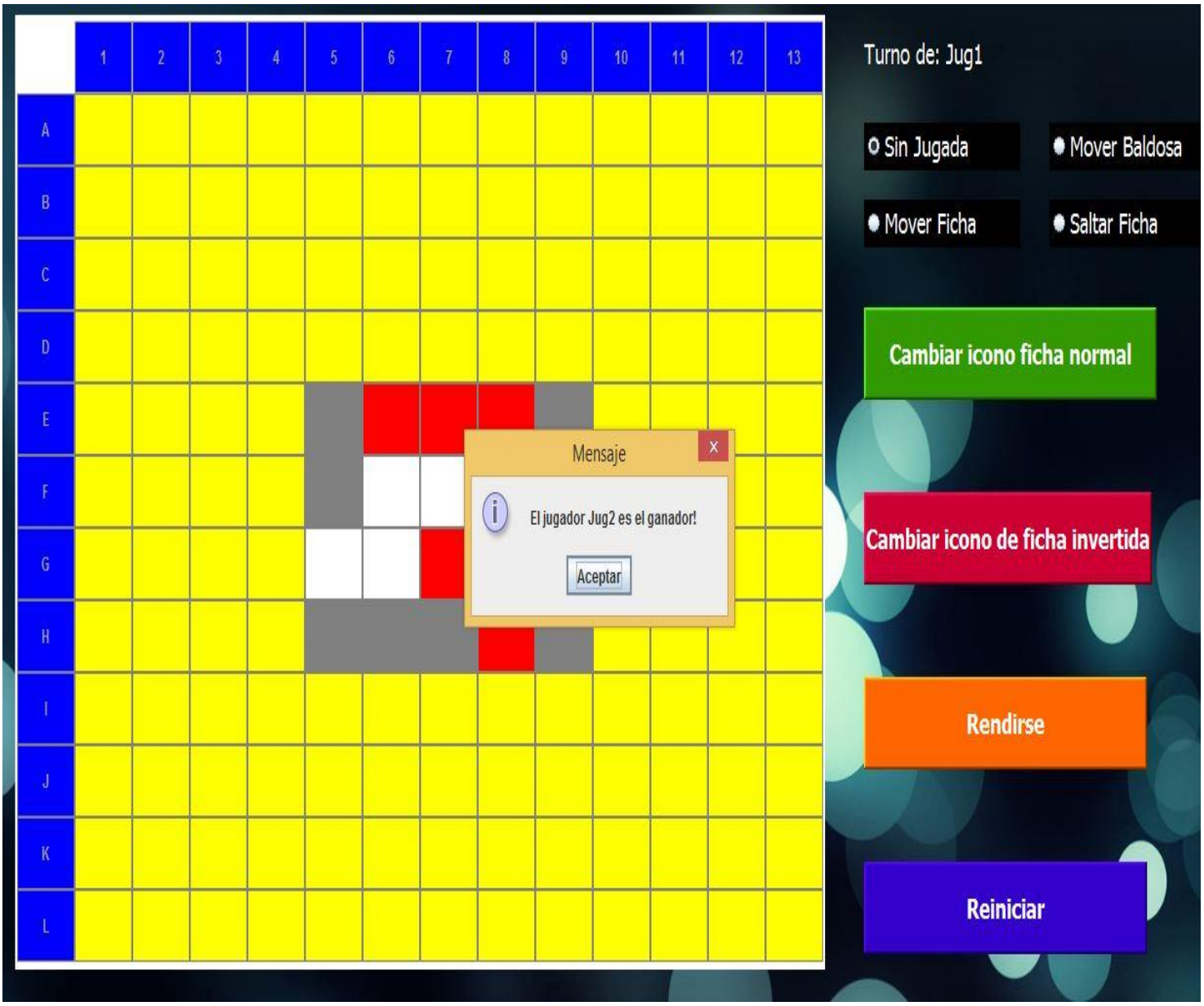

1 2 3 4 5 6 7 8 9 10 11 12 13
A B C D E F G H I J K L
Turno de: Jug1
Sin Jugada
Mover Baldosa
Mover Ficha
Saltar Ficha
Cambiar icono ficha normal
Cambiar icono de ficha invertida
Rendirse
Reiniciar
Mensaje
El jugador Jug2 es el ganador!
Aceptar

imagen 16

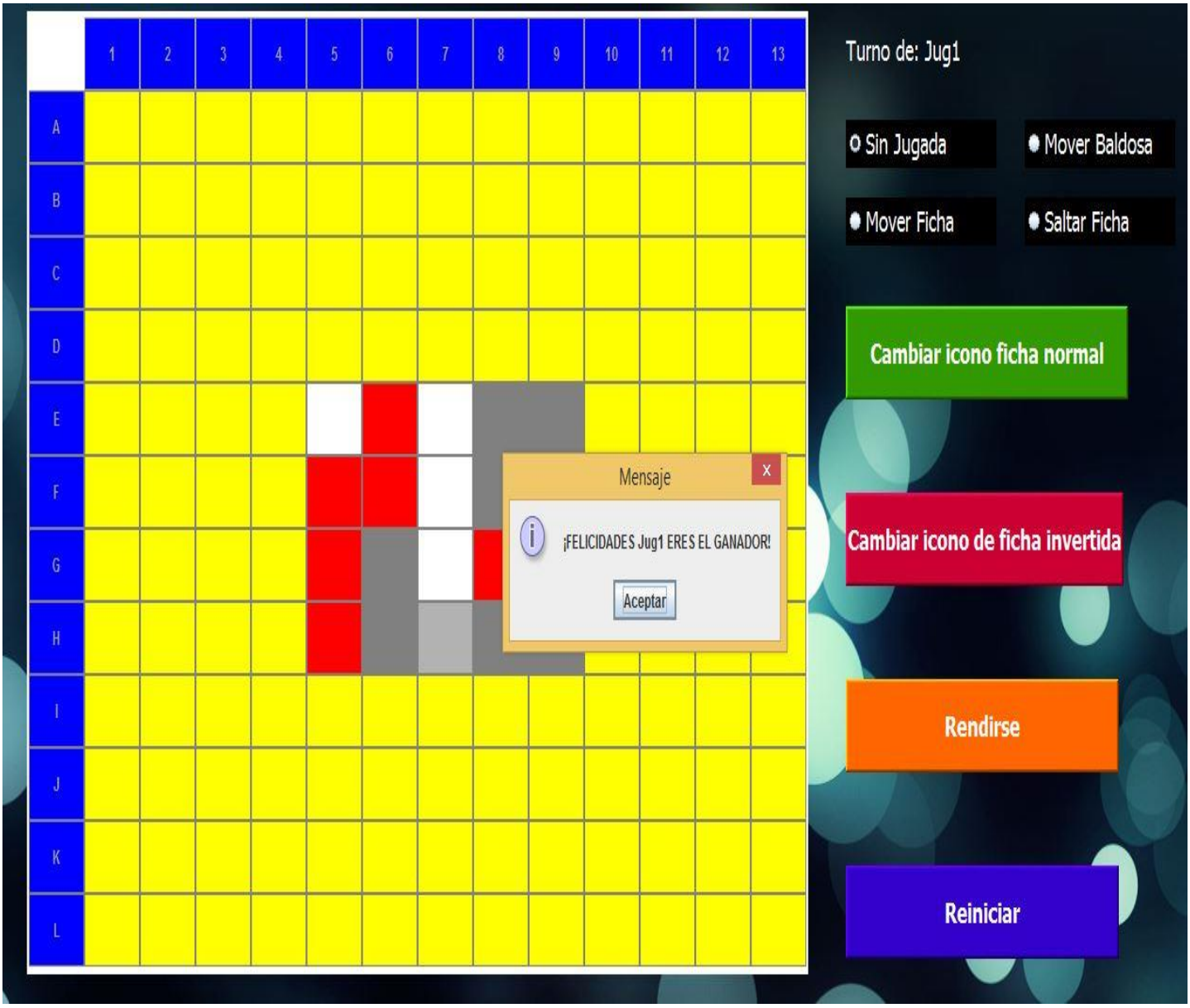

1 2 3 4 5 6 7 8 9 10 11 12 13
A B C D E F G H I J K L
Turno de: Jug1
Sin Jugada
Mover Baldosa
Mover Ficha
Saltar Ficha
Cambiar icono ficha normal
Cambiar icono de ficha invertida
Rendirse
Reiniciar
Mensaje
¡FELICIDADES Jug1 ERES EL GANADOR!
Aceptar

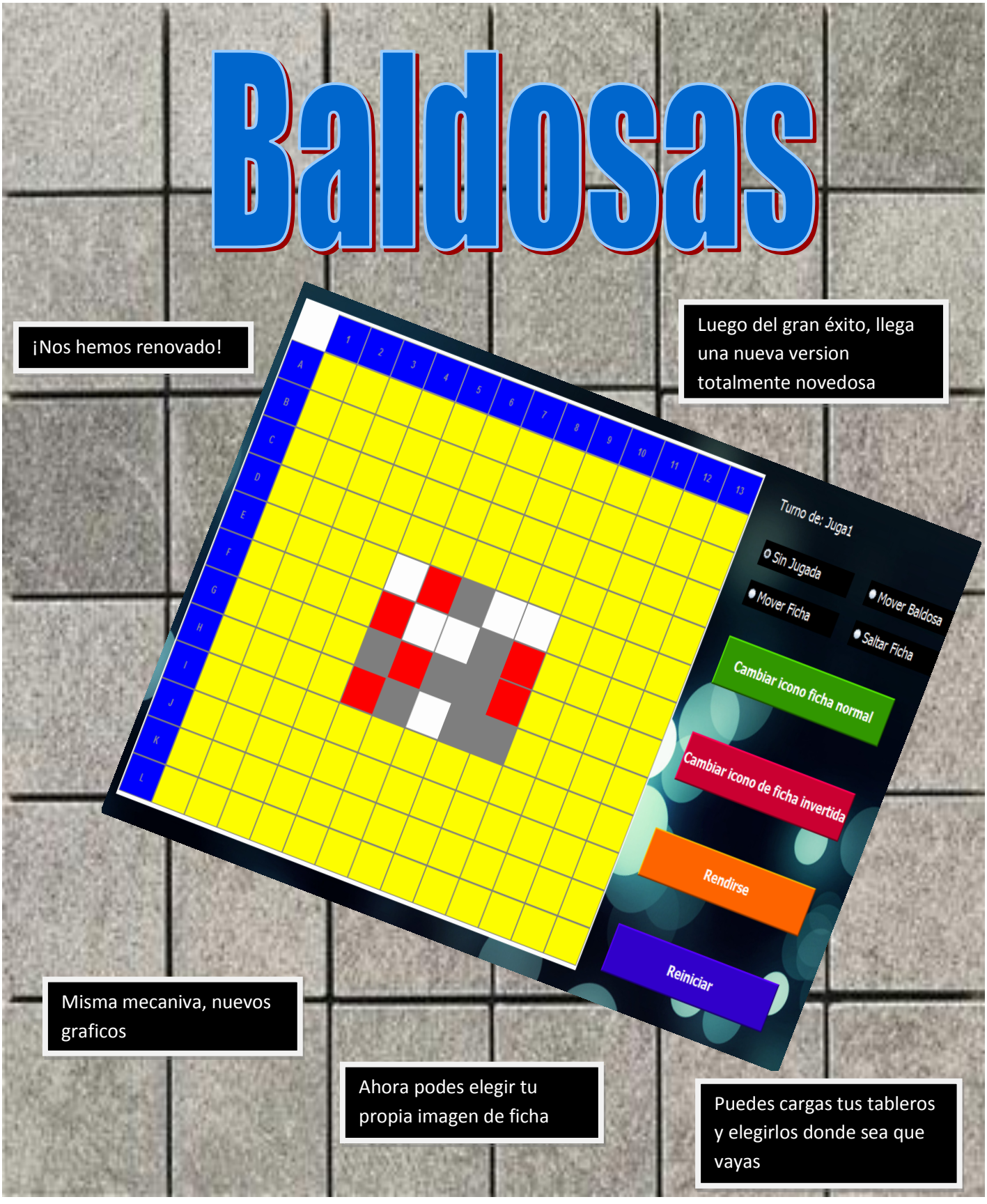
imagen 17

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

imagen 18

Folleto promocional de Baldosas

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)



Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

Definición de emprendedurismo

El emprendedurismo es el proceso en el cual una persona lleva una idea propia a convertirse en un proyecto concreto generando innovación y empleo. Esa persona es el emprendedor.

El Centro de Innovación y Emprendimientos (CIE) de la Universidad ORT es un sitio que promueve y desarrolla la generación de nuevos emprendedores. Establece un vínculo entre emprendedores, inversores y la sociedad.

El CIE estimula a los nuevos emprendedores a generar soluciones a las necesidades del mercado y aporta soluciones a la sociedad. Busca potenciar la creatividad que cada uno tiene a través de talleres, conferencias y actividades de vinculación. Genera redes de contacto con el sector productivo transformando la idea del emprendedor en un negocio viable económicamente y de interés en el mercado.

Los jóvenes emprendedores ven potenciada su creatividad mediante la metodología de trabajo del CIE. Inicialmente presentan su proyecto y trabajan bajo su tutela. De ser aceptados por el Comité de Selección continúan trabajando bajo el apoyo del CIE y presentan su idea ante inversores públicos y/o privados para la financiación del mismo.

De esta forma el CIE promueve a los jóvenes a que emprendan y les proporciona las herramientas para poder hacerlo.

Bibliografía:

http://www.mific.gob.ni/Portals/0/Portal%20Empresarial/u15.%20emprendedurismo.pdf
http://cie.ort.edu.uy/37/1/que_es_el_cie.html
http://fi.ort.edu.uy/9072/5/cie:_un_lugar_donde_las_ideas_valen.html
http://fa.ort.edu.uy/7811/8/cie.html

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

Listado impreso de clases

Clase Sistema:

```
package dominio;


import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.Collections;

import javax.swing.ImageIcon;


public class Sistema implements Serializable {


    private ArrayList<Jugador> listaJugadores;

    private ArrayList<Partida> listaPartidas;

    private ArrayList<Tablero> listaTableros;

    private Tablero precargar;
```

```
public Sistema() {
    listaJugadores = new ArrayList<Jugador>();
    listaPartidas = new ArrayList<Partida>();
    listaTableros = new ArrayList<Tablero>();
    precargar = new Tablero();
    precargar.precargaAzar();
}

public ArrayList<Jugador> getListaJugadores() {
    return listaJugadores;
}

public void agregarJugador(Jugador unJugador) {
    listaJugadores.add(unJugador);
}

public ArrayList<Partida> getListaPartidas() {
    return listaPartidas;
}

public void agregarPartida(Partida unaPartida) {
    unaPartida.setTablero(precargar.copia());
    listaPartidas.add(unaPartida);
}
```

```
    public ArrayList<Tablero> getListaTableros() {
        return listaTableros;
    }

    public void agregarTablero(Tablero unTablero) {
        listaTableros.add(unTablero);
    }

    public Tablero getPrecarga() {
        return precargar;
    }

    public Partida partidaActual() {
        return listaPartidas.get(listaPartidas.size() - 1);
    }

    //registro un jugador
    public boolean registrarJugador(String unNombre, int unaEdad, String
unAlias) throws Exception {
        boolean ok = false;
        Jugador unJugador = new Jugador(unNombre, unaEdad, unAlias, 0, 0, 0, 0,
false);
        if (!aliasUnico(unAlias)) {
            this.listaJugadores.add(unJugador);
            ok = true;
        }
```

```
        return ok;
    }

    //verifico que el alias sea unico
    public boolean aliasUnico(String unAlias) {
        boolean ok = false;
        for (int i = 0; i < this.listaJugadores.size(); i++) {
            if (this.listaJugadores.get(i).getAlias().equals(unAlias)) {
                ok = true;
            }
        }
        return ok;
    }

    //creo una lista auxiliar y meto todos los jugadores y los ordeno por
nombre
    public ArrayList<Jugador> listaOrdenada() {
        ArrayList<Jugador> lista = new ArrayList<Jugador>();
        for (int i = 0; i < this.getListaJugadores().size(); i++) {
            Jugador unJ = this.getListaJugadores().get(i);
            lista.add(unJ);
        }
        Collections.sort(lista);
        return lista;
    }
```

```
    //paso la lista ordenada por parametros y muestro los datos de los
jugadores en pantalla
    public void mostrarDatos(ArrayList<Jugador> lista) {
        for (int i = 0; i < lista.size(); i++) {
            Jugador unJugador = lista.get(i);
            System.out.println("El jugador " + unJugador.getAlias()
                    + " jugo un total de " + unJugador.getPartidasJugadas() + "
veces"
                    + " y gano un total de " + unJugador.getPartidasGanadas() +
" partidas");
        }
    }

    public void precargarFija() {
        precargar.precargaFija();
    }

    public void precargaAzar() {
        precargar.precargaAzar();
    }

    public boolean agregarFichaPrecargaMano(String fila, String columna, String
color) {
        return precargar.agregarFicha(Partida.conversorFila(fila),
Partida.conversorColumna(columna), color);
    }
```

```
    public boolean moverFicha(int filaOrigen, int columnaOrigen, int
filaDestino, int columnaDestino) throws Exception {

        return partidaActual().moverFicha(filaOrigen, columnaOrigen,
filaDestino, columnaDestino);

    }


    public boolean meterFicha(int numeroF, int numeroC, int numeroFila, int
numeroColumna, Jugador jugadorJugando) {

        boolean ok = false;

        if (partidaActual().getTablero().coordenadaValida(numeroFila,
numeroColumna)) {

            if (partidaActual().getTablero().hayBaldosa(numeroFila,
numeroColumna)) {

                if (partidaActual().getTablero().hayFicha(numeroF, numeroC)) {

                    if (!partidaActual().getTablero().hayFicha(numeroFila,
numeroColumna)) {

                        if
(jugadorJugando.getFicha().getLetra().equals(partidaActual().getTablero().getTa
bleroFichas()[numeroF][numeroC].getLetra().toUpperCase())) {

                            ok = true;

                        }

                    }

                }

            }

        }

        return ok;

    }
```

```
    public boolean hayFichaEnElMedio(int numeroFilaOrigen, int
numeroColumnaOrigen, int numeroFilaDestino, int numeroColumnaDestino) {

        return partidaActual().getTablero().hayFichaEnElMedio(numeroFilaOrigen,
numeroColumnaOrigen, numeroFilaDestino, numeroColumnaDestino);

    }


    public boolean saltoOk(int numeroFilaOrigen, int numeroColumnaOrigen, int
numeroFilaDestino, int numeroColumnaDestino) {

        return partidaActual().getTablero().saltoOk(numeroFilaOrigen,
numeroColumnaOrigen, numeroFilaDestino, numeroColumnaDestino);

    }


    public boolean saltaFicha(int numeroFilaOrigen, int numeroColumnaOrigen,
int numeroFilaDestino, int numeroColumnaDestino) throws Exception {

        return partidaActual().saltarFicha(numeroFilaOrigen,
numeroColumnaOrigen, numeroFilaDestino, numeroColumnaDestino);

    }


    public boolean validarFichaConBaldosa(int filaD, int columnaD, int filaFO,
int columnaFO, Jugador jugadorJugando) {

        return partidaActual().getTablero().validarFichaConBaldosa(filaD,
columnaD, filaFO, columnaFO, jugadorJugando);

    }


    public boolean baldosaConDosAlLado(int numeroFilaOrigen, int
numeroColumnaOrigen) {

        return
partidaActual().getTablero().baldosaConDosAlLado(numeroFilaOrigen,
numeroColumnaOrigen);
```

```
    }


    public boolean nuevaUbicacionBaldosa(int numeroFilaDestino, int
numeroColumnaDestino) {

        return
partidaActual().getTablero().nuevaUbicacionBaldosa(numeroFilaDestino,
numeroColumnaDestino);

    }


    public boolean moverBaldosa(int numeroFilaOrigen, int numeroColumnaOrigen,
int numeroFilaDestino, int numeroColumnaDestino, int numeroFilaOFicha, int
numeroColumnaOFicha, Jugador jugadorJugando) throws Exception {

        return partidaActual().moverBaldosa(numeroFilaOrigen,
numeroColumnaOrigen, numeroFilaDestino, numeroColumnaDestino, numeroFilaOFicha,
numeroColumnaOFicha, jugadorJugando);

    }


    public boolean validarDimensionesMatriz(int dimension) {

        boolean esValida;

        switch (dimension) {

            case 1:

                esValida = true;

                break;

            case 2:

                esValida = true;

                break;

            case 3:

                esValida = true;
```

```
                break;
            default:
                esValida = false;
        }
        return esValida;
    }

    public boolean fichaCoincideConJugador(Ficha unaFicha) {
        return partidaActual().fichaCoincideConJugador(unaFicha);
    }

    public Sistema obtenerSistema() {
        Sistema sis;
        try {
            File aqui = new File("").getAbsoluteFile();
            String direccionEntera = aqui.getPath() + "\\obligatorio2";
            BufferedInputStream inBuffer = new BufferedInputStream(new
FileInputStream(direccionEntera));
            ObjectInputStream stream = new ObjectInputStream(inBuffer);
            sis = (Sistema) stream.readObject();
            stream.close();
        } catch (IOException | ClassNotFoundException err) {
            sis = new Sistema();
        }
        return sis;
    }
```

```
    public void guardarSistema(Sistema sis) throws FileNotFoundException,
IOException {
        try {
            File aqui = new File("").getAbsoluteFile();
            String direccionEntera = aqui.getPath() + "\\obligatorio2";
            BufferedOutputStream outBuffer = new BufferedOutputStream(new
FileOutputStream(direccionEntera));
            ObjectOutputStream stream = new ObjectOutputStream(outBuffer);
            stream.writeObject(sis);
            stream.flush();
            stream.close();
        } catch (IOException err) {
            err.printStackTrace();
        }
    }

    public void reiniciar() {
        partidaActual().setTurno(true);
        partidaActual().setTablero(precargar.copia());
    }

    public void cambiarIconoNormal(ImageIcon icono) {
        partidaActual().cambiarIconoNormal(icono);
    }
```

```
    public void cambiarIconoInvertido(ImageIcon icono) {
        partidaActual().cambiarIconoInvertido(icono);
    }

    public void modificarTableroPrecarga(Tablero tablero) {
        precargar = tablero;
    }
}
```

Clase Tablero:

```
package dominio;


import java.io.Serializable;

import java.util.Date;

import java.util.Random;

import javax.swing.ImageIcon;


public class Tablero implements Serializable {


    private Ficha[][] tableroFichas;

    private boolean[][] tableroBaldosas;

    private String nombre;

    private Date fecha;


    public Tablero(Ficha[][] tabF, boolean[][] tabB) {

        this.setTableroFichas(tabF);

        this.setTableroBaldosas(tabB);

    }


    public Ficha[][] getTableroFichas() {

        return tableroFichas;

    }


    public void setTableroFichas(Ficha[][] tableroFichas) {
```

```
        this.tableroFichas = tableroFichas;
    }

    public boolean[][] getTableroBaldosas() {
        return tableroBaldosas;
    }

    public void setTableroBaldosas(boolean[][] tableroBaldosas) {
        this.tableroBaldosas = tableroBaldosas;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public Date getFecha() {
        return fecha;
    }

    public void setFecha(Date fecha) {
        this.fecha = fecha;
```

```
    }

    public Tablero() {
        limpiar();
    }

    public void limpiar() {
        Ficha[][] tFichas = new Ficha[13][14];
        this.setTableroFichas(tFichas);
        boolean[][] tBaldosas = new boolean[13][14];
        this.setTableroBaldosas(tBaldosas);
        for (int i = 5; i < 9; i++) {
            for (int j = 5; j < 10; j++) {
                this.getTableroBaldosas()[i][j] = (true);
            }
        }
    }

    public boolean coordenadaValida(int filaDestino, int columnaDestino) {
        boolean ok = false;
        if ((filaDestino >= 0 && filaDestino <= 12) && (columnaDestino >= 0 &&
columnaDestino <= 13)) {
            ok = true;
        }
        return ok;
    }
```

```
public boolean hayBaldosa(int numeroFila, int numeroColumna) {
    return this.getTableroBaldosas()[numeroFila][numeroColumna];
}

public boolean hayFicha(int numeroFila, int numeroColumna) {
    return (this.getTableroFichas()[numeroFila][numeroColumna] != null);
}

public String getFicha(int f, int c) {
    return tableroFichas[f][c].getLetra();
}

public void precargaFija() {
    limpiar();
    this.getTableroFichas()[5][5] = new Ficha(5, 5, "B");
    this.getTableroFichas()[5][7] = new Ficha(5, 7, "B");
    this.getTableroFichas()[6][8] = new Ficha(6, 8, "B");
    this.getTableroFichas()[7][7] = new Ficha(7, 7, "B");
    this.getTableroFichas()[7][9] = new Ficha(7, 9, "B");
    this.getTableroFichas()[8][5] = new Ficha(8, 5, "B");
    this.getTableroFichas()[5][6] = new Ficha(5, 6, "R");
    this.getTableroFichas()[6][5] = new Ficha(6, 5, "R");
    this.getTableroFichas()[6][7] = new Ficha(6, 7, "R");
    this.getTableroFichas()[7][5] = new Ficha(7, 5, "R");
```

```
        this.getTableroFichas()[7][8] = new Ficha(7, 8, "R");
        this.getTableroFichas()[8][6] = new Ficha(8, 6, "R");
    }

    public void precargaAzar() {
        int i = 1;
        Random rnd = new Random();
        limpiar();
        while (i < 7) {
            int fila = (int) (rnd.nextDouble() * 4 + 5);
            int columna = (int) (rnd.nextDouble() * 5 + 5);
            if (tableroFichas[fila][columna] == null) {
                tableroFichas[fila][columna] = new Ficha(fila, columna, "B");
                i++;
            }
        }
        while (i < 13) {
            int fila = (int) (rnd.nextDouble() * 4 + 5);
            int columna = (int) (rnd.nextDouble() * 5 + 5);
            if (tableroFichas[fila][columna] == null) {
                tableroFichas[fila][columna] = new Ficha(fila, columna, "R");
                i++;
            }
        }
    }
```

```
    public boolean agregarFicha(int fila, int columna, String color) {
        boolean ok = false;
        if (this.coordenadaValida(fila, columna)) {
            if (hayBaldosa(fila, columna) && !hayFicha(fila, columna)) {
                this.getTableroFichas()[fila][columna] = new Ficha(fila,
columna, color);
                ok = true;
            }
        }
        return ok;
    }

    public boolean validarMovimientoFicha(int filaOrigen, int columnaOrigen,
int filaDestino, int columnaDestino) {
        boolean ok = false;
        if ((Math.abs(filaOrigen - filaDestino) == 1 && Math.abs(columnaOrigen
- columnaDestino) == 1) || (Math.abs(filaOrigen - filaDestino) == 0 &&
Math.abs(columnaOrigen - columnaDestino) == 1) || (Math.abs(filaOrigen -
filaDestino) == 1 && Math.abs(columnaOrigen - columnaDestino) == 0)) {
            if (coordenadaValida(filaDestino, columnaDestino)) {
                ok = true;
            }
        }
        return ok;
    }
```

```
    public void moverFicha(int filaOrigen, int columnaOrigen, int filaDestino,
int columnaDestino, Ficha fichaAux) throws Exception {

        if (hayBaldosa(filaDestino, columnaDestino)) {

            if (validarMovimientoFicha(filaOrigen, columnaOrigen, filaDestino,
columnaDestino)) {

                if (!hayFicha(filaDestino, columnaDestino)) {

                    getTableroFichas()[filaDestino][columnaDestino] = fichaAux;

                    getTableroFichas()[filaOrigen][columnaOrigen] = null;

                } else {

                    throw new NullPointerException("Ya hay una ficha en esa
posicion");

                }

            } else {

                throw new NullPointerException("Imposible realizar ese
movimiento con la ficha");

            }

        } else {

            throw new NullPointerException("No hay baldosa en la posicion de
destino");

        }

    }


    public boolean hayFichaEnElMedio(int numeroFilaOrigen, int
numeroColumnaOrigen, int numeroFilaDestino, int numeroColumnaDestino) {

        int fila;

        int columna;

        if (numeroFilaOrigen == numeroFilaDestino) {
```

```
            fila = numeroFilaOrigen;
        } else {
            fila = Math.max(numeroFilaOrigen, numeroFilaDestino) - 1;
        }
        if (numeroColumnaOrigen == numeroColumnaDestino) {
            columna = numeroColumnaOrigen;
        } else {
            columna = Math.max(numeroColumnaOrigen, numeroColumnaDestino) - 1;
        }
        return hayFicha(fila, columna);
    }


    public boolean validarFichaConBaldosa(int filaD, int columnaD, int filaFO,
int columnaFO, Jugador jugadorJugando) {
        boolean ok = false;
        if (coordenadaValida(filaFO, columnaFO)) {
            Ficha fichaAux = getTableroFichas()[filaFO][columnaFO];
            if (fichaAux != null &&
jugadorJugando.getFicha().getLetra().toUpperCase().equals(fichaAux.getLetra().t
oUpperCase())) {
                if (saltoOk(filaFO, columnaFO, filaD, columnaD) ||
validarMovimientoFicha(filaFO, columnaFO, filaD, columnaD)) {
                    ok = true;
                }
            }
        }
        return ok;
```

```
    }


    public int Vali1(int numeroFilaOrigen, int numeroColumnaOrigen) {

        int cont = 0;

        if (getTableroBaldosas()[numeroFilaOrigen - 1][numeroColumnaOrigen] ==
true) {

            cont++;

        }

        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen - 1] ==
true) {

            cont++;

        }

        if (getTableroBaldosas()[numeroFilaOrigen + 1][numeroColumnaOrigen] ==
true) {

            cont++;

        }

        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen + 1] ==
true) {

            cont++;

        }

        return cont;

    }


    public int Vali2(int numeroFilaOrigen, int numeroColumnaOrigen) {

        int cont = 0;

        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen - 1] ==
true) {
```

```
            cont++;

        }

        if (getTableroBaldosas()[numeroFilaOrigen + 1][numeroColumnaOrigen] ==
true) {

            cont++;

        }

        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen + 1] ==
true) {

            cont++;

        }

        return cont;

    }


    public int Vali3(int numeroFilaOrigen, int numeroColumnaOrigen) {

        int cont = 0;

        if (getTableroBaldosas()[numeroFilaOrigen + 1][numeroColumnaOrigen] ==
true) {

            cont++;

        }

        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen + 1] ==
true) {

            cont++;

        }

        return cont;

    }


    public int Vali4(int numeroFilaOrigen, int numeroColumnaOrigen) {
```

```
        int cont = 0;
        if (getTableroBaldosas()[numeroFilaOrigen - 1][numeroColumnaOrigen] ==
true) {
            cont++;
        }
        if (getTableroBaldosas()[numeroFilaOrigen + 1][numeroColumnaOrigen] ==
true) {
            cont++;
        }
        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen + 1] ==
true) {
            cont++;
        }
        return cont;
    }

    public int Vali5(int numeroFilaOrigen, int numeroColumnaOrigen) {
        int cont = 0;
        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen + 1] ==
true) {
            cont++;
        }
        if (getTableroBaldosas()[numeroFilaOrigen - 1][numeroColumnaOrigen] ==
true) {
            cont++;
        }
```

```
        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen - 1] ==
true) {
            cont++;
        }
        return cont;
    }

    public int Vali6(int numeroFilaOrigen, int numeroColumnaOrigen) {
        int cont = 0;
        if (getTableroBaldosas()[numeroFilaOrigen - 1][numeroColumnaOrigen] ==
true) {
            cont++;
        }
        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen - 1] ==
true) {
            cont++;
        }
        if (getTableroBaldosas()[numeroFilaOrigen + 1][numeroColumnaOrigen] ==
true) {
            cont++;
        }
        return cont;
    }

    public int Vali7(int numeroFilaOrigen, int numeroColumnaOrigen) {
        int cont = 0;
```

```
        if (getTableroBaldosas()[numeroFilaOrigen - 1][numeroColumnaOrigen] ==
true) {
            cont++;
        }
        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen - 1] ==
true) {
            cont++;
        }
        return cont;
    }

    public int Vali8(int numeroFilaOrigen, int numeroColumnaOrigen) {
        int cont = 0;
        if (getTableroBaldosas()[numeroFilaOrigen - 1][numeroColumnaOrigen] ==
true) {
            cont++;
        }
        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen + 1] ==
true) {
            cont++;
        }
        return cont;
    }

    public int Vali9(int numeroFilaOrigen, int numeroColumnaOrigen) {
        int cont = 0;
```

```
        if (getTableroBaldosas()[numeroFilaOrigen + 1][numeroColumnaOrigen] ==
true) {

            cont++;

        }

        if (getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen - 1] ==
true) {

            cont++;

        }

        return cont;

    }


    public boolean saltoOk(int numeroFilaOrigen, int numeroColumnaOrigen, int
numeroFilaDestino, int numeroColumnaDestino) {

        boolean ok = false;

        if ((Math.abs(numeroFilaOrigen - numeroFilaDestino) == 2 &&
Math.abs(numeroColumnaOrigen - numeroColumnaDestino) == 2) ||
(Math.abs(numeroFilaOrigen - numeroFilaDestino) == 0 &&
Math.abs(numeroColumnaOrigen - numeroColumnaDestino) == 2) ||
(Math.abs(numeroFilaOrigen - numeroFilaDestino) == 2 &&
Math.abs(numeroColumnaOrigen - numeroColumnaDestino) == 0)) {

            if ((coordenadaValida(numeroFilaDestino, numeroColumnaDestino)) &&
(hayFichaEnElMedio(numeroFilaOrigen, numeroColumnaOrigen, numeroFilaDestino,
numeroColumnaDestino))) {

                ok = true;

            }

        }

        return ok;

    }
```

```
    public boolean saltarFicha(int numeroFilaOrigen, int numeroColumnaOrigen,
int numeroFilaDestino, int numeroColumnaDestino, Jugador jugadorJugando) throws
Exception {

        boolean ok = false;

        if (coordenadaValida(numeroFilaDestino, numeroColumnaDestino)) {

            if (hayBaldosa(numeroFilaDestino, numeroColumnaDestino)) {

                if (!hayFicha(numeroFilaDestino, numeroColumnaDestino)) {

                    Ficha fichaAux =
getTableroFichas()[numeroFilaOrigen][numeroColumnaOrigen];

                    if
(jugadorJugando.getFicha().getLetra().toUpperCase().equals(fichaAux.getLetra().
toUpperCase())) {

                        if (saltoOk(numeroFilaOrigen, numeroColumnaOrigen,
numeroFilaDestino, numeroColumnaDestino)) {

                            if (fichaAux.getLetra().equals("R") ||
fichaAux.getLetra().equals("B")) {


fichaAux.setLetra(fichaAux.getLetra().toLowerCase());

                            } else {


fichaAux.setLetra(fichaAux.getLetra().toUpperCase());

                            }


getTableroFichas()[numeroFilaDestino][numeroColumnaDestino] = fichaAux;


getTableroFichas()[numeroFilaOrigen][numeroColumnaOrigen] = null;

                            ok = true;

                        }

                    } else {
```

```
                        throw new NullPointerException("Imposible seleccionar
la ficha del rival");

                    }

                } else {

                    throw new ArrayIndexOutOfBoundsException("Ya hay ficha en
la posicion de destino");

                }

            } else {

                throw new NullPointerException("No hay baldosa en la posicion
de destino");

            }

        } else {

            throw new NullPointerException("No hay baldosa en la posicion de
partida");

        }

        return ok;

    }


    public boolean baldosaConDosAlLado(int numeroFilaOrigen, int
numeroColumnaOrigen) {

        boolean ok = false;

        int cont = 0;

        if (numeroFilaOrigen != 0 && numeroColumnaOrigen != 0 &&
numeroFilaOrigen != 12 && numeroColumnaOrigen != 13) {

            cont = Vali1(numeroFilaOrigen, numeroColumnaOrigen);

        } else {

            if (numeroFilaOrigen == 0 && numeroColumnaOrigen != 0 &&
numeroColumnaOrigen != 13) {
```

```
                cont = Vali2(numeroFilaOrigen, numeroColumnaOrigen);
            } else {
                if (numeroFilaOrigen == 0 && numeroColumnaOrigen == 0) {
                    cont = Vali3(numeroFilaOrigen, numeroColumnaOrigen);
                } else {
                    if (numeroFilaOrigen != 0 && numeroFilaOrigen != 12 &&
numeroColumnaOrigen == 0) {
                        cont = Vali4(numeroFilaOrigen, numeroColumnaOrigen);
                    } else {
                        if (numeroFilaOrigen == 12 && numeroColumnaOrigen != 0
&& numeroColumnaOrigen != 13) {
                            cont = Vali5(numeroFilaOrigen,
numeroColumnaOrigen);
                        } else {
                            if (numeroFilaOrigen != 12 && numeroFilaOrigen != 0
&& numeroColumnaOrigen == 13) {
                                cont = Vali6(numeroFilaOrigen,
numeroColumnaOrigen);
                            } else {
                                if (numeroFilaOrigen == 12 &&
numeroColumnaOrigen == 13) {
                                    cont = Vali7(numeroFilaOrigen,
numeroColumnaOrigen);
                                } else {
                                    if (numeroFilaOrigen == 12 &&
numeroColumnaOrigen == 0) {
                                        cont = Vali8(numeroFilaOrigen,
numeroColumnaOrigen);
                                    } else {
```

```
                                        if (numeroFilaOrigen == 0 &&
numeroColumnaOrigen == 13) {

                                            cont = Vali9(numeroFilaOrigen,
numeroColumnaOrigen);

                                        }

                                    }

                                }

                            }

                        }

                    }

                }

            }

        }

        if (cont <= 2) {

            ok = true;

        }

        return ok;

    }


    public boolean nuevaUbicacionBaldosa(int numeroFilaDestino, int
numeroColumnaDestino) {

        boolean ok = false;

        int cont = 0;

        if (coordenadaValida(numeroFilaDestino, numeroColumnaDestino)) {

            if (numeroFilaDestino != 0 && numeroColumnaDestino != 0 &&
numeroFilaDestino != 12 && numeroColumnaDestino != 13) {

                cont = Vali1(numeroFilaDestino, numeroColumnaDestino);
```

```
            } else {
                if (numeroFilaDestino == 0 && numeroColumnaDestino != 0 &&
numeroColumnaDestino != 13) {
                    cont = Vali2(numeroFilaDestino, numeroColumnaDestino);
                } else {
                    if (numeroFilaDestino == 0 && numeroColumnaDestino == 0) {
                        cont = Vali3(numeroFilaDestino, numeroColumnaDestino);
                    } else {
                        if (numeroFilaDestino != 0 && numeroFilaDestino != 12
&& numeroColumnaDestino == 0) {
                            cont = Vali4(numeroFilaDestino,
numeroColumnaDestino);
                        } else {
                            if (numeroFilaDestino == 12 && numeroColumnaDestino
!= 0 && numeroColumnaDestino != 13) {
                                cont = Vali5(numeroFilaDestino,
numeroColumnaDestino);
                            } else {
                                if (numeroFilaDestino != 12 &&
numeroFilaDestino != 0 && numeroColumnaDestino == 13) {
                                    cont = Vali6(numeroFilaDestino,
numeroColumnaDestino);
                                } else {
                                    if (numeroFilaDestino == 12 &&
numeroColumnaDestino == 13) {
                                        cont = Vali7(numeroFilaDestino,
numeroColumnaDestino);
                                    } else {
```

```
                                        if (numeroFilaDestino == 12 &&
numeroColumnaDestino == 0) {
                                            cont = Vali8(numeroFilaDestino,
numeroColumnaDestino);
                                        } else {
                                            if (numeroFilaDestino == 0 &&
numeroColumnaDestino == 13) {
                                                cont = Vali9(numeroFilaDestino,
numeroColumnaDestino);
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
            if (cont >= 1) {
                ok = true;
            }
            return ok;
        }
```

```
    public boolean moverBaldosa(int numeroFilaOrigen, int numeroColumnaOrigen,
int numeroFilaDestino, int numeroColumnaDestino, int numeroFilaOFicha, int
numeroColumnaOFicha, Jugador jugadorJugando) throws Exception {

        boolean ok = false;

        if (!hayFicha(numeroFilaOrigen, numeroColumnaOrigen)) {

            if (getTableroBaldosas()[numeroFilaDestino][numeroColumnaDestino]
== false) {

                if (baldosaConDosAlLado(numeroFilaOrigen, numeroColumnaOrigen))
{

                    if (nuevaUbicacionBaldosa(numeroFilaDestino,
numeroColumnaDestino)) {

                        if (validarFichaConBaldosa(numeroFilaDestino,
numeroColumnaDestino, numeroFilaOFicha, numeroColumnaOFicha, jugadorJugando)) {


getTableroBaldosas()[numeroFilaDestino][numeroColumnaDestino] = true;


getTableroBaldosas()[numeroFilaOrigen][numeroColumnaOrigen] = false;

                            if (!hayFicha(numeroFilaDestino,
numeroColumnaDestino)) {

                                if (coordenadaValida(numeroFilaDestino,
numeroColumnaDestino)) {

                                    if (saltarFicha(numeroFilaOFicha,
numeroColumnaOFicha, numeroFilaDestino, numeroColumnaDestino, jugadorJugando))
{

                                        ok = true;

                                    } else {

                                        Ficha fichaAux =
getTableroFichas()[numeroFilaOFicha][numeroColumnaOFicha];

                                        moverFicha(numeroFilaOFicha,
numeroColumnaOFicha, numeroFilaDestino, numeroColumnaDestino, fichaAux);
```

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

```
                                    ok = true;
                                }
                            } else {
                                throw new NullPointerException("Ya hay una
baldosa en esa posicion");
                            }
                        } else {
                            throw new NullPointerException("La posicion
elegida ya contiene ficha");
                        }
                    } else {
                        throw new NullPointerException("Imposible mover la
baldosa a esa ubicacion");
                    }
                } else {
                    throw new NullPointerException("Imposible seleccionar
esa baldosa");
                }
            } else {
                throw new NullPointerException("Imposible mover esa
baldosa");
            }
        } else {
            throw new NullPointerException("Ya hay una baldosa en esa
posicion");
        }
    } else {
```

```
            throw new NullPointerException("Imposible mover una baldosa con
ficha");
        }
        return ok;
    }

    public Tablero copia() {
        Tablero tableroCopiado = new Tablero();
        tableroCopiado.setNombre(this.nombre);
        tableroCopiado.setFecha(this.getFecha());
        for (int i = 5; i < 9; i++) {
            for (int j = 5; j < 10; j++) {
                if (this.hayFicha(i, j)) {
                    tableroCopiado.agregarFicha(i, j, this.getFicha(i,
j).charAt(0) + "");
                }
            }
        }
        return tableroCopiado;
    }

    public void cambiarFichasNormales(ImageIcon icono, Ficha ficha) {
        int ancho = getTableroFichas().length;
        int largo = getTableroFichas()[0].length;
        for (int i = 0; i < ancho; i++) {
            for (int j = 0; j < largo; j++) {
```

```
                if (hayFicha(i, j)) {
                    Ficha unaF = getTableroFichas()[i][j];
                    if
(unaF.getLetra().toUpperCase().equals(ficha.getLetra().toUpperCase())) {
                        unaF.setIcono(icono);
                    }
                }
            }
        }
    }

    public void cambiarFichasInvertidas(ImageIcon icono, Ficha ficha) {
        int ancho = getTableroFichas().length;
        int largo = getTableroFichas()[0].length;
        for (int i = 0; i < ancho; i++) {
            for (int j = 0; j < largo; j++) {
                if (hayFicha(i, j)) {
                    Ficha unaF = getTableroFichas()[i][j];
                    if
(unaF.getLetra().toLowerCase().equals(ficha.getLetra().toLowerCase())) {
                        unaF.setIconoInvertido(icono);
                    }
                }
            }
        }
    }
```

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

```
    @Override
    public String toString() {
        return "Nombre: " + nombre + ", Fecha: " + fecha;
    }
}
```

Clase Partida:

```
package dominio;


import java.io.Serializable;

import java.util.ArrayList;

import javax.swing.ImageIcon;


public class Partida implements Serializable {


    private Tablero tablero;

    private Jugador jGanador;

    private Jugador jPerdedor;

    private ArrayList<Jugador> listaJugadores;

    private Jugador jugador1;

    private Jugador jugador2;

    private int cantidadFichasInv;

    private boolean turno;

    private Jugador jugadorJugando;

    private Jugador jugadorEsperando;


    public Partida(Jugador jugador1, Jugador jugador2, int cantidadFichasInv) {

        this.jugador1 = jugador1;

        this.jugador1.setFicha(new Ficha("B"));

        this.jugador2 = jugador2;

        this.jugador2.setFicha(new Ficha("R"));
```

```
        this.cantidadFichasInv = cantidadFichasInv;
        turno = true;
    }

    public Partida() {
        Jugador unJ = new Jugador();
        Tablero unT = new Tablero();
        this.setJugador1(jugador1);
        this.setJugador2(jugador2);
        this.setTablero(unT);
        this.setJGanador(unJ);
        this.setJPerdedor(unJ);
        this.setTurno(turno);
        this.listaJugadores = new ArrayList<>();
    }

    public Tablero getTablero() {
        return tablero;
    }

    public void setTablero(Tablero unTablero) {
        tablero = unTablero;
    }

    public Jugador getJGanador() {
```

```
    return jGanador;
}

public void setJGanador(Jugador unJGanador) {
    jGanador = unJGanador;
}

public Jugador getJPerdedor() {
    return jPerdedor;
}

public void setJPerdedor(Jugador unJPerdedor) {
    this.jPerdedor = unJPerdedor;
}

public ArrayList<Jugador> getListaJugadores() {
    return listaJugadores;
}

public void setListaJugadores(ArrayList<Jugador> listaJugadores) {
    this.listaJugadores = listaJugadores;
}

public Jugador getJugador1() {
    return jugador1;
```

```
}

public void setJugador1(Jugador unJugador1) {
    this.jugador1 = unJugador1;
}

public Jugador getJugador2() {
    return jugador2;
}

public void setJugador2(Jugador unJugador2) {
    this.jugador2 = unJugador2;
}

public Jugador jugadorEnJugando() {
    if (turno) {
        return jugador1;
    } else {
        return jugador2;
    }
}

public Jugador jugadorEnEspera() {
    if (turno) {
        return jugador1;
```

```
    } else {
        return jugador2;
    }
}

public int getCantFichasInv() {
    return cantidadFichasInv;
}

public void setCantFichasInv(int unaCantidadFichasInv) {
    this.cantidadFichasInv = unaCantidadFichasInv;
}

public boolean getTurno() {
    return turno;
}

public void setTurno(boolean unTurno) {
    this.turno = unTurno;
}

public Jugador getJugadorJugando() {
    return jugadorJugando;
}
```

```
public void setJugadorJugando(Jugador unJugJugando) {
    this.jugadorJugando = unJugJugando;
}

public Jugador getJugadorEsperando() {
    return jugadorEsperando;
}

public void setJugadorEsperando(Jugador unJugEsperando) {
    this.jugadorEsperando = unJugEsperando;
}

public void precargarFija() {
    tablero.precargaFija();
}

public void precargaAzar() {
    tablero.precargaAzar();
}

public void partidasJugadas(Jugador unJ1, Jugador unJ2) {
    unJ1.setPartidasJugadas(unJ1.getPartidasJugadas() + 1);
    unJ2.setPartidasJugadas(unJ2.getPartidasJugadas() + 1);
}
```

```
public void partidasGanadas(Jugador unJPerdedor, Jugador unJGanador) {
    this.setJPerdedor(unJPerdedor);
    this.setJGanador(unJGanador);
    unJPerdedor.setPartidasPerdidas(unJPerdedor.getPartidasPerdidas() + 1);
    unJGanador.setPartidasGanadas(unJGanador.getPartidasGanadas() + 1);
}

public void partidasEmpatadas(Jugador unJ1, Jugador unJ2) {
    unJ1.setPartidasEmpatadas(unJ1.getPartidasEmpatadas() + 1);
    unJ1.setPartidasEmpatadas(unJ2.getPartidasEmpatadas() + 1);
}

public static int conversorFila(String coordenadas) {
    int numeroFila = coordenadas.charAt(0) - 64;
    return numeroFila;
}

public static int conversorColumna(String coordenadas) {
    int numeroColumna;
    if (coordenadas.length() == 1) {
        numeroColumna = coordenadas.charAt(0) - 48;
    } else {
        try {
            numeroColumna = Integer.parseInt(coordenadas);
        } catch (NumberFormatException ex) {
```

```
                numeroColumna = -1;
            }
        }
        return numeroColumna;
    }

    private boolean esFichaInv(int fila, int columna) {
        return
(this.tablero.getTableroFichas()[fila][columna].getLetra().equals("r")
                ||
this.tablero.getTableroFichas()[fila][columna].getLetra().equals("b"));
    }

    private boolean validarVertical(int fila, int columna, int fichasInv, int
cantFichasInv) {
        int cantFichas = 1;
        int i = fila + 1;
        while ((i < 12) && (this.tablero.hayFicha(i, columna))
                &&
this.tablero.getTableroFichas()[i][columna].getLetra().toUpperCase().equals(thi
s.tablero.getTableroFichas()[fila][columna].getLetra().toUpperCase())) {
            if
(this.tablero.getTableroFichas()[i][columna].getLetra().equals("r")
                    ||
this.tablero.getTableroFichas()[i][columna].getLetra().equals("b")) {
                cantFichasInv++;
            }
            cantFichas++;
```

```
            i++;
        }
        i = fila - 1;
        while ((i >= 0) && (this.tablero.hayFicha(i, columna))
                &&
this.tablero.getTableroFichas()[i][columna].getLetra().toUpperCase().equals(thi
s.tablero.getTableroFichas()[fila][columna].getLetra().toUpperCase())) {


            if
(this.tablero.getTableroFichas()[i][columna].getLetra().equals("r")
                    ||
this.tablero.getTableroFichas()[i][columna].getLetra().equals("b")) {
                cantFichasInv++;
            }
            cantFichas++;
            i--;
        }
        return ((cantFichas >= 4) && (cantFichasInv >= fichasInv));
    }


    private boolean validarHorizontal(int fila, int columna, int fichasInv, int
cantFichasInv) {
        int cantFichas = 1;
        int j = columna + 1;
        while ((j < 13) && (this.tablero.hayFicha(fila, j))
                &&
this.tablero.getTableroFichas()[fila][j].getLetra().toUpperCase().equals(this.t
ablero.getTableroFichas()[fila][columna].getLetra().toUpperCase())) {
```

```
            if (this.tablero.getTableroFichas()[fila][j].getLetra().equals("r")
                    ||
this.tablero.getTableroFichas()[fila][j].getLetra().equals("b")) {
                cantFichasInv++;
            }
            cantFichas++;
            j++;
        }
        j = columna - 1;
        while ((j >= 0) && (this.tablero.hayFicha(fila, j))
                &&
this.tablero.getTableroFichas()[fila][j].getLetra().toUpperCase().equals(this.t
ablero.getTableroFichas()[fila][columna].getLetra().toUpperCase())) {

            if (this.tablero.getTableroFichas()[fila][j].getLetra().equals("r")
                    ||
this.tablero.getTableroFichas()[fila][j].getLetra().equals("b")) {
                cantFichasInv++;
            }
            cantFichas++;
            j--;
        }
        return (cantFichas >= 4) && (cantFichasInv >= fichasInv);
    }
```

```
    private boolean validarDiagonalIzqDer(int fila, int columna, int fichasInv,
int cantFichasInv) {

        int cantFichas = 1;

        int i = fila + 1;

        int j = columna + 1;

        while ((j < 13) && (i < 12) && (this.tablero.hayFicha(i, j))

                &&
this.tablero.getTableroFichas()[i][j].getLetra().toUpperCase().equals(this.tabl
ero.getTableroFichas()[fila][columna].getLetra().toUpperCase())) {

            cantFichas++;


            if (this.tablero.getTableroFichas()[i][j].getLetra().equals("r")

                    ||
this.tablero.getTableroFichas()[i][j].getLetra().equals("b")) {

                cantFichasInv++;

            }

            j++;

            i++;

        }

        j = columna - 1;

        i = fila - 1;

        while ((j >= 0) && (i >= 0) && (this.tablero.hayFicha(i, j))

                &&
this.tablero.getTableroFichas()[i][j].getLetra().toUpperCase().equals(this.tabl
ero.getTableroFichas()[fila][columna].getLetra().toUpperCase())) {

            cantFichas++;
```

```
            if (this.tablero.getTableroFichas()[i][j].getLetra().equals("r")
                    ||
this.tablero.getTableroFichas()[i][j].getLetra().equals("b")) {
                cantFichasInv++;
            }
            j--;
            i--;
        }
        return (cantFichas >= 4) && (cantFichasInv >= fichasInv);
    }

    private boolean validarDiagonalDerIzq(int fila, int columna, int fichasInv,
int cantFichasInv) {
        int cantFichas = 1;
        int i = fila - 1;
        int j = columna + 1;
        while ((j < 13) && (i >= 0) && (this.tablero.hayFicha(i, j))
                &&
this.tablero.getTableroFichas()[i][j].getLetra().toUpperCase().equals(this.tabl
ero.getTableroFichas()[fila][columna].getLetra().toUpperCase())) {
            cantFichas++;

            if (this.tablero.getTableroFichas()[i][j].getLetra().equals("r")
                    ||
this.tablero.getTableroFichas()[i][j].getLetra().equals("b")) {
                cantFichasInv++;
            }
```

```
            j++;
            i--;
        }
        j = columna - 1;
        i = fila + 1;
        while ((j >= 0) && (i < 12) && (this.tablero.hayFicha(i, j))
                &&
this.tablero.getTableroFichas()[i][j].getLetra().toUpperCase().equals(this.tabl
ero.getTableroFichas()[fila][columna].getLetra().toUpperCase())) {
            cantFichas++;

            if (this.tablero.getTableroFichas()[i][j].getLetra().equals("r")
                    ||
this.tablero.getTableroFichas()[i][j].getLetra().equals("b")) {
                cantFichasInv++;
            }
            j--;
            i++;
        }
        return (cantFichas >= 4) && (cantFichasInv >= fichasInv);
    }

    public boolean partidaGanada(int fila, int columna, int fichasInv) {
        int cantFichasInv = 0;
        if (esFichaInv(fila, columna)) {
            cantFichasInv = 1;
```

```
        }
        return (validarVertical(fila, columna, fichasInv, cantFichasInv))
                || (validarHorizontal(fila, columna, fichasInv, cantFichasInv))
                || (validarDiagonalDerIzq(fila, columna, fichasInv,
cantFichasInv))
                || (validarDiagonalIzqDer(fila, columna, fichasInv,
cantFichasInv));
    }

    public boolean fichaCoincideConJugador(Ficha unaFicha) {
        boolean ok = false;
        if (getTurno()) {
            if (unaFicha.getLetra().equals("B") ||
unaFicha.getLetra().equals("b")) {
                ok = true;
            }
        } else {

            if (unaFicha.getLetra().equals("R") ||
unaFicha.getLetra().equals("r")) {
                ok = true;
            }

        }
        return ok;
    }
```

```
    public boolean moverFicha(int filaOrigen, int columnaOrigen, int
filaDestino, int columnaDestino) throws Exception {

        boolean ok = false;

        Ficha fichaAux =
getTablero().getTableroFichas()[filaOrigen][columnaOrigen];

        if
(jugadorEnJugando().getFicha().getLetra().toUpperCase().equals(fichaAux.getLetr
a().toUpperCase())) {

            getTablero().moverFicha(filaOrigen, columnaOrigen, filaDestino,
columnaDestino, fichaAux);

            ok = true;

            setTurno(!getTurno());

        }

        return ok;

    }


    public boolean saltarFicha(int numeroFilaOrigen, int numeroColumnaOrigen,
int numeroFilaDestino, int numeroColumnaDestino) throws Exception {

        boolean ok = false;

        if (getTablero().saltarFicha(numeroFilaOrigen, numeroColumnaOrigen,
numeroFilaDestino, numeroColumnaDestino, jugadorEnJugando())) {

            setTurno(!getTurno());

            ok = true;

        } else {

            ok = false;

        }

        return ok;

    }
```

```
    public boolean moverBaldosa(int numeroFilaOrigen, int numeroColumnaOrigen,
int numeroFilaDestino, int numeroColumnaDestino, int numeroFilaOFicha, int
numeroColumnaOFicha, Jugador jugadorJugando) throws Exception {

        boolean ok = true;

        if (getTablero().moverBaldosa(numeroFilaOrigen, numeroColumnaOrigen,
numeroFilaDestino, numeroColumnaDestino, numeroFilaOFicha, numeroColumnaOFicha,
jugadorEnJugando())) {

            ok = true;

            setTurno(!getTurno());

        } else {

            throw new Exception("Imposible mover la badosa");

        }

        return ok;

    }


    public void cambiarIconoNormal(ImageIcon icono) {

        Ficha ficha = new Ficha("R");

        if (turno) {

            ficha = new Ficha("B");

        }

        tablero.cambiarFichasNormales(icono, ficha);

    }


    public void cambiarIconoInvertido(ImageIcon icono) {

        Ficha ficha = new Ficha("r");

        if (turno) {
```

```
            ficha = new Ficha("b");
        }
        tablero.cambiarFichasInvertidas(icono, ficha);
    }
}
```

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

Clase Jugador:

```
package dominio;


import java.io.Serializable;


public class Jugador implements Serializable, Comparable<Jugador> {


    private Ficha ficha;

    private String nombre;

    private int edad;

    private String alias;

    private int partidasJugadas;

    private int partidasPerdidas;

    private int partidasGanadas;

    private int partidasEmpatadas;

    private boolean esMiTurno;


    public Jugador() {

        this.setNombre("Sin nombre");

        this.setEdad(0);

        this.setAlias("Sin alias");

        this.setPartidasJugadas(0);

        this.setPartidasPerdidas(0);

        this.setPartidasGanadas(0);

        this.setPartidasEmpatadas(0);
```

```
        this.setEsMiTurno(true);
    }

    public Jugador(String unNombre, int unaEdad, String unAlias, int
unaPartidasJugadas, int unaPartidasPerdidas, int unaPartidasGanadas, int
unaPartidaEmpatada, boolean unTurno) {
        this.setNombre(unNombre);
        this.setEdad(unaEdad);
        this.setAlias(unAlias);
        this.setPartidasJugadas(unaPartidasJugadas);
        this.setPartidasPerdidas(unaPartidasPerdidas);
        this.setPartidasGanadas(unaPartidasGanadas);
        this.setPartidasEmpatadas(unaPartidaEmpatada);
        this.setPartidasEmpatadas(0);
        this.setEsMiTurno(unTurno);
    }

    public Ficha getFicha() {
        return ficha;
    }

    public void setFicha(Ficha unaFicha) {
        this.ficha = unaFicha;
    }

    public String getNombre() {
```

```
    return nombre;
}

public void setNombre(String unNombre) {
    this.nombre = unNombre;
}

public int getEdad() {
    return edad;
}

public void setEdad(int unaEdad) {
    this.edad = unaEdad;
}

public String getAlias() {
    return alias;
}

public void setAlias(String unAlias) {
    this.alias = unAlias;
}

public int getPartidasJugadas() {
    return partidasJugadas;
```

```
}

public void setPartidasJugadas(int unaPartidasJugadas) {
    this.partidasJugadas = unaPartidasJugadas;
}

public int getPartidasPerdidas() {
    return partidasPerdidas;
}

public void setPartidasPerdidas(int unaPartidasPerdidas) {
    this.partidasPerdidas = unaPartidasPerdidas;
}

public int getPartidasGanadas() {
    return partidasGanadas;
}

public void setPartidasGanadas(int unaPartidasGanadas) {
    this.partidasGanadas = unaPartidasGanadas;
}

public int getPartidasEmpatadas() {
    return partidasEmpatadas;
}
```

```
public void setPartidasEmpatadas(int unaPartidaEmpatada) {
    this.partidasEmpatadas = unaPartidaEmpatada;
}

public boolean getEsMiTurno() {
    return esMiTurno;
}

public void setEsMiTurno(boolean unTurno) {
    esMiTurno = unTurno;
}

@Override
public String toString() {
    return "\nNombre del jugador: " + this.getNombre()
            + ", \nEdad del jugador: " + this.getEdad()
            + ", \nAlias del jugador: " + this.getAlias();
}

@Override
public boolean equals(Object obj) {
    Jugador unJugador = ((Jugador) obj);
    return this.getAlias().equals(unJugador.getAlias());
}
```

```
    @Override
    public int compareTo(Jugador unJugador) {
        int retorno = unJugador.getPartidasGanadas() -
this.getPartidasGanadas();
        if (retorno == 0) {
            retorno = this.getAlias().compareTo(unJugador.getAlias());
        }
        return retorno;
    }
}
```

Andres Lacaño (158910)
Rodolfo Agustín Silva (187061)

Clase Ficha:

```
package dominio;


import java.awt.Color;

import java.io.Serializable;

import javax.swing.Icon;

import javax.swing.ImageIcon;


public class Ficha implements Serializable {


    private int fila;

    private int columna;

    private String letra;

    private Color c;

    private ImageIcon icono;

    private ImageIcon iconoInvertido;


    public Ficha() {

        this.setFila(0);

        this.setColumna(0);

        this.setLetra("");

    }


    public Ficha(String unaLetra) {

        this.setLetra(unaLetra);
```

```
}

public Ficha(int unaFila, int unaColumna, String unaLetra) {
    this.setFila(unaFila);
    this.setColumna(unaColumna);
    this.setLetra(unaLetra);
}

public ImageIcon getIconoInvertido() {
    return iconoInvertido;
}

public void setIconoInvertido(ImageIcon iconoInvertido) {
    this.iconoInvertido = iconoInvertido;
}

public ImageIcon getIcono() {
    return icono;
}

public void setIcono(ImageIcon icono) {
    this.icono = icono;
}

public Color getC() {
```

```
    return c;
}

public void setC(Color unC) {
    this.c = unC;
}

public int getFila() {
    return fila;
}

public int getColumna() {
    return columna;
}

public void setFila(int unaFila) {
    fila = unaFila;
}

public void setColumna(int unaColumna) {
    this.columna = unaColumna;
}

public String getLetra() {
    return letra;
```

```
}

public void setLetra(String unaLetra) {
    this.letra = unaLetra;
}

@Override
public String toString() {
    String r = "";
    if (letra.contains("B")) {
        r = "B";
    } else {
        if (letra.contains("R")) {
            r = "R";
        } else {
            if (letra.contains("b")) {
                r = "b";
            } else {
                if (letra.contains("r")) {
                    r = "r";
                }
            }
        }
    }
    return r;
```

```
    }


    public Icon getIconoActual() {
        Icon ico;
        if (letra.contains("B") || letra.contains("R")) {
            ico = icono;
        } else {
            ico = iconoInvertido;
        }
        return ico;
    }
}
```