

Universidad ORT

Escuela de Ingeniería

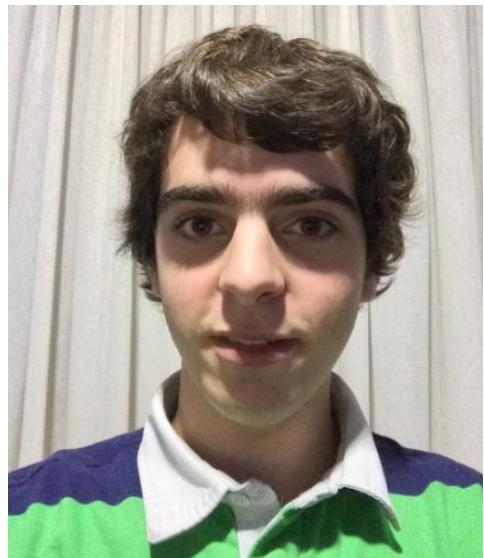
Obligatorio 2

Programación 2

Autores:

Fernando Agustín Hernández Gobertti (173631) – Ingeniería en Telecomunicaciones

Sebastián Effa Gallego (193248) – Ingeniería en Electrónica



Tutores:

Nicolás Fornaro (Teórico)

Matías Núñez (Práctico)

Fecha Límite de Entrega: 24

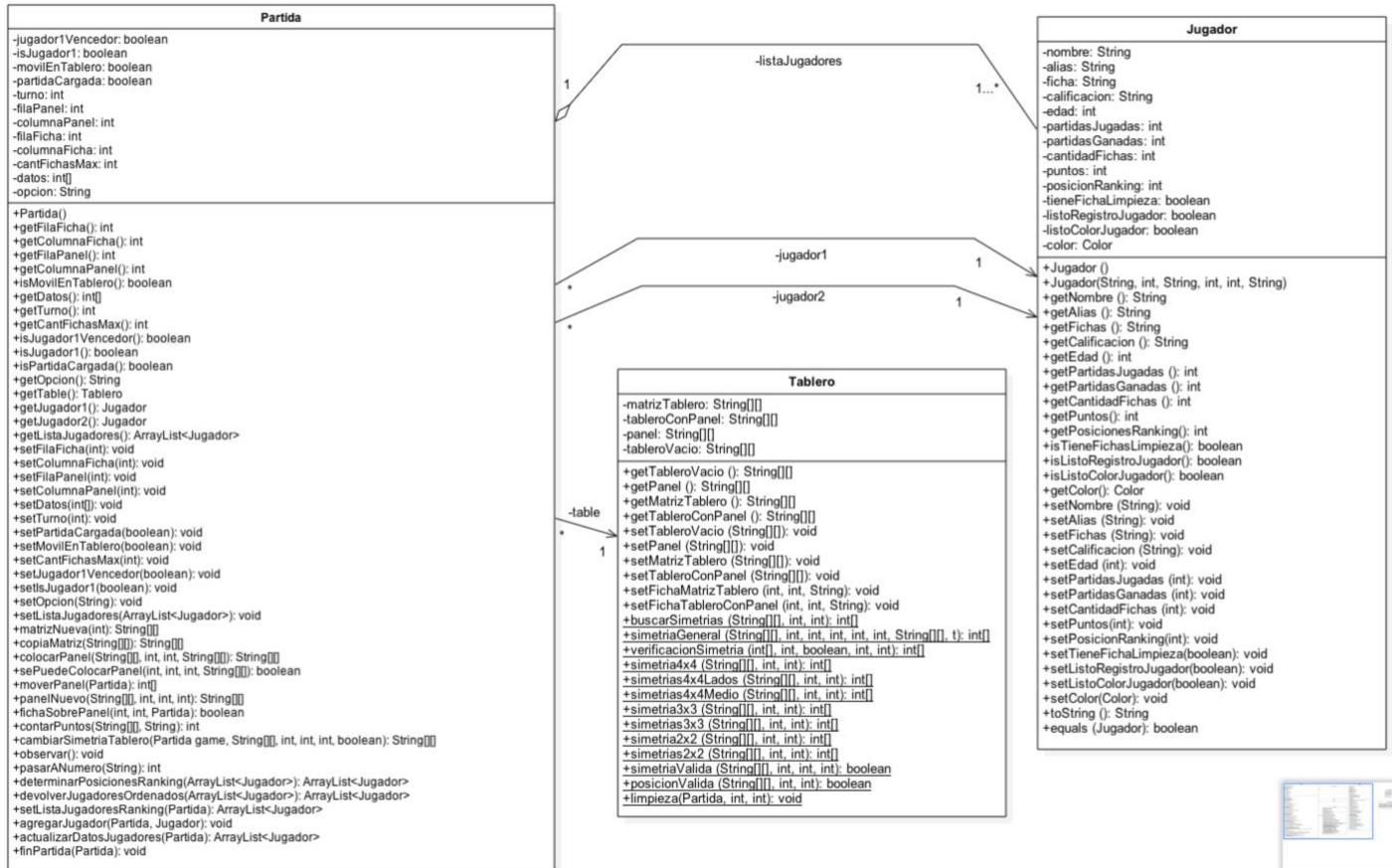
de Noviembre del 2014



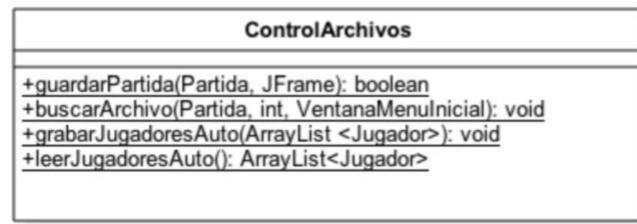
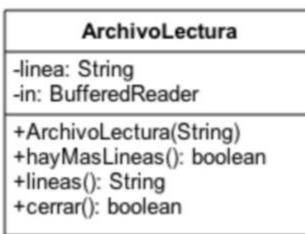
Índice del Trabajo Realizado:

Diagrama de Clases	3
Datos de Prueba	5
<i>Folleto del juego para promoción del producto:</i>	42, 43
<i>Página 1. PORTADA</i>	42
<i>Página 2. INTERIOR</i>	43
Emprendimiento de una Empresa de Videojuegos.....	44
Clases del Dominio (y su respectivo código):	45
Clase ArchivoLectura	45
Clase ControlArchivos.....	47
Clase Jugador	50
Clase Tablero.....	56
Clase Partida	73

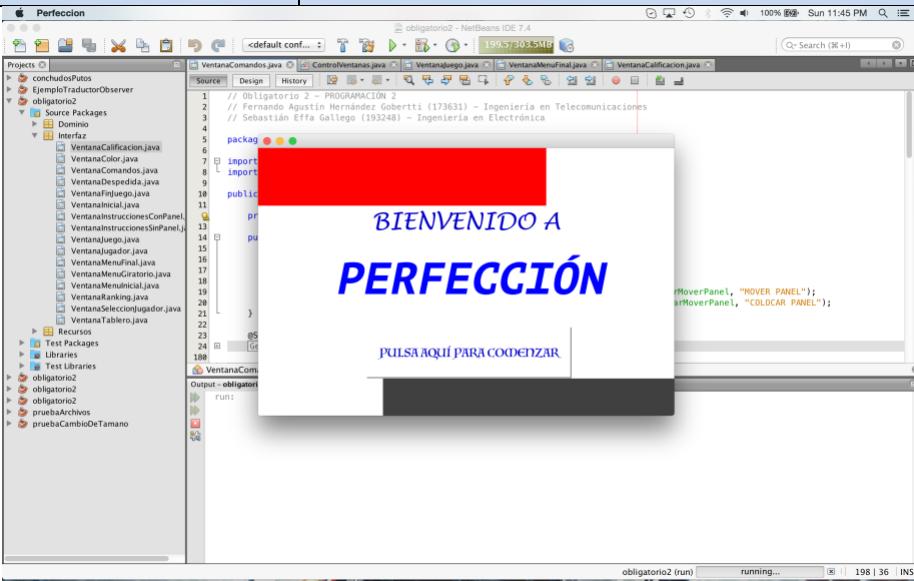
Diagrama de Clases (UML: *Unified Modelling Language*)



Nota: Creación en *StarUML* Versión 2.0.0-beta10 para MAC OS X



Datos de Prueba

ACCIÓN INGRESADA	ACCIÓN ESPERADA	RESULTADO
	 <p>The screenshot shows a Java application running in NetBeans. The window title is "BIENVENIDO A PERFECCIÓN". Inside the window, there is a message: "PULSA AQUÍ PARA CODENZAR". The application is developed in Java, with code visible in the source editor.</p>	OK

Las ventanas del trabajo se encuentran aproximadamente centradas en una pantalla de 15 pulgadas.

<p>Se precisa ayuda sobre lo que realizar en cada ventana para proseguir con el programa.</p>		<p>OK</p>
<p>Se desea salir del programa cliqueando en la cruz roja de la esquina superior izquierda del programa.</p>		<p>OK</p>

VentanaInicial

Se cliquea en el botón de "Comenzar" de la VentanaInicial.

Se cambia de ventana y muestra el Menu Inicial.

OK

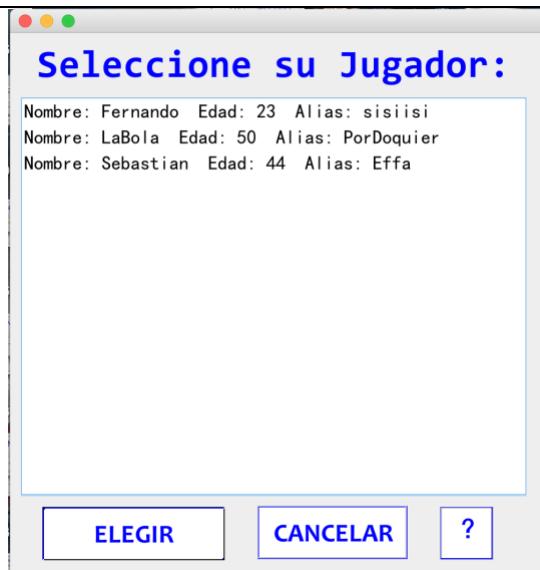


Decorado con funcionamiento apropiado y correcto brindado por el efecto de los colores.

VentanaMenúInicial

<p>Se cliquea el JButton de Ayuda (“?”).</p>	<p>Se despliega un mensaje de ayuda explicando lo que debe realizar en la ventana actual para continuar.</p> 	<p>OK</p>
<p>Se desea seleccionar mas de una vez un color al cambiar su decisión, o registrar jugadores varias veces diferentes.</p>	<p>Se procede a tocar las veces necesarias en las opciones correspondientes y los valores serán sobreescritos o agregados según el caso.</p>	<p>OK</p>
<p>Se desea seleccionar un jugador para jugar sin haber indicado de que jugador se trata al marcar los JRadioButton.</p>	 <p>Se avisa al usuario y se le solicita indique el jugador propicio.</p>	<p>OK</p>

Se selecciona el JButton de "Seleccionar Jugador".



OK

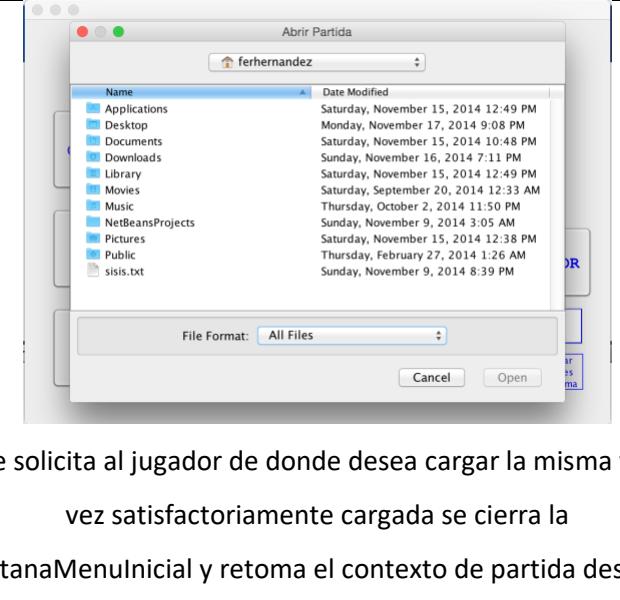
Se despliega una JList con los jugadores cargados previamente para que el jugador elija.

Se elije un jugador de la lista desplegada al querer seleccionar un jugador.

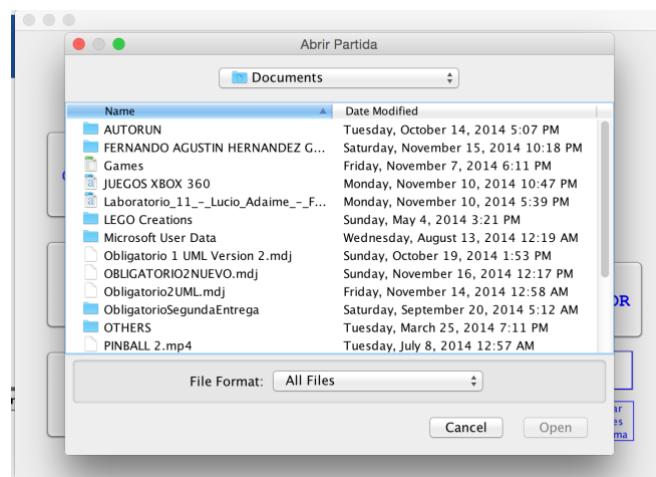


OK

Se le avisa al usuario y se setean los datos del jugador elegido con los datos del jugador indicado en los JRadioButton.

<p>Se es seleccionado el jugador o se le es asignando un color de la paleta de colores satisfactoriamente.</p>		<p>OK</p>
<p>Se presiona el JButton de “Registro Jugador”.</p>	<p>Se cierra la VentanaMenulinicial y se despliega la VentanaRegistroJugador.</p>	<p>OK</p>
<p>Se presiona el JButton de “Cargar Jugadores”.</p>	<p>Se le pregunta al usuario la dirección del archivo donde se los quiere cargar y se le avisa al usuario una vez cargados.</p>	<p>OK</p>
<p>El usuario seleccionado no es compatible con la forma requerida para leer los archivos con los datos de los Jugadores.</p>	<p>Se le indica al usuario del problema y se le solicita una nueva selección del archivo.</p>	<p>OK</p>
<p>Se desea cargar una partida previa, por lo que el usuario cliquea en el JButton “Cargar Partida”.</p>	 <p>Se le solicita al jugador de donde desea cargar la misma y una vez satisfactoriamente cargada se cierra la VentanaMenulinicial y retoma el contexto de partida desde el</p>	<p>OK</p>

que se había guardado la partida previamente.



Se selecciona un archivo con datos de Partida que no posee los datos correctos.

Se avisa al usuario del problema y se le solicita una nueva selección de archivo o un nuevo comienzo de partida.

OK

Se quiere continuar la Partida sin haber hecho todo lo solicitado en el Menu Inicial.

Se le avisa al usuario y se le es solicitado que continue con el registro y el cumplimiento de los requisitos previos al inicio del juego.

OK

Se desea continuar a la Partida habiendo registrado y cumplido los requerimientos del Menu Inicial, cliqueando en el JButton donde se lee "Continuar".

Desaparece la VentanaMenuInicial y aparece la VentanaTablero, previa al inicio del juego.

OK

Se presiona el JButton "Restaurar Jugadores En Sistema".



Se eliminan todos los jugadores ingresados hasta el momento.

OK

VentanaRegistroJugador

Se cliquea el JButton de Ayuda (“?”).



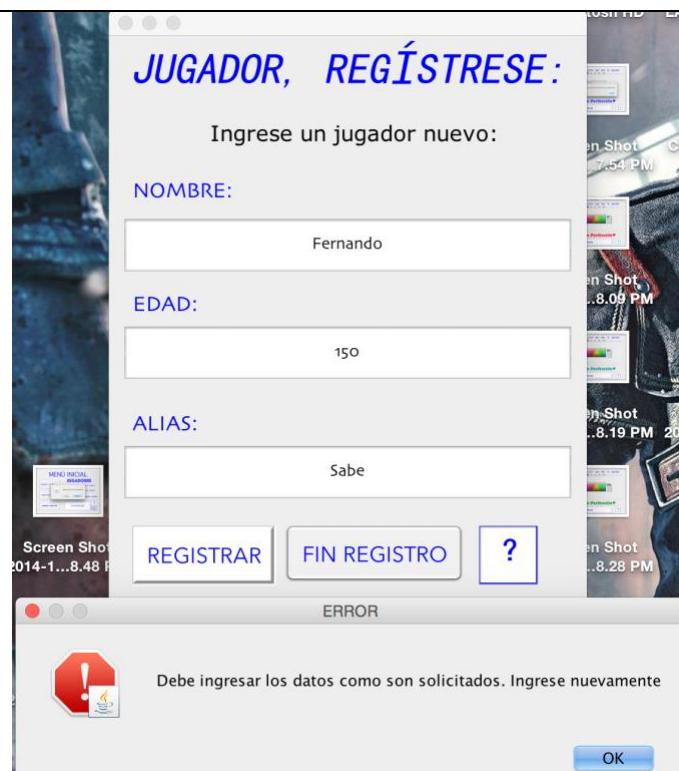
Se despliega un mensaje de ayuda explicando lo que debe realizar en la ventana actual para continuar.

OK

Se ingresan los datos y se selecciona el JButton “Registrar”.

Si los datos se encuentran correctos, se registra el jugador en el archivo de jugadores.

OK



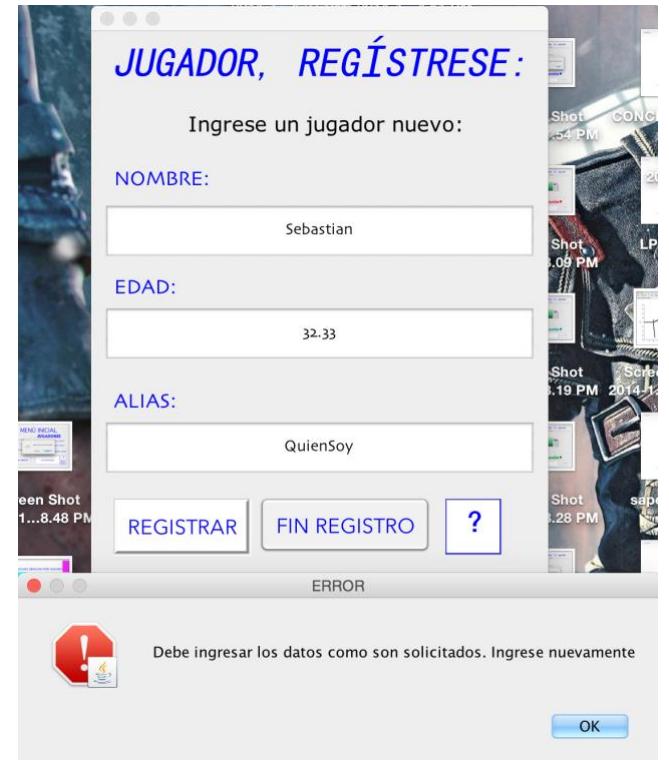
Se ingresa una edad irreal
(negativo o mayor a un máximo
estimado en 120 años).

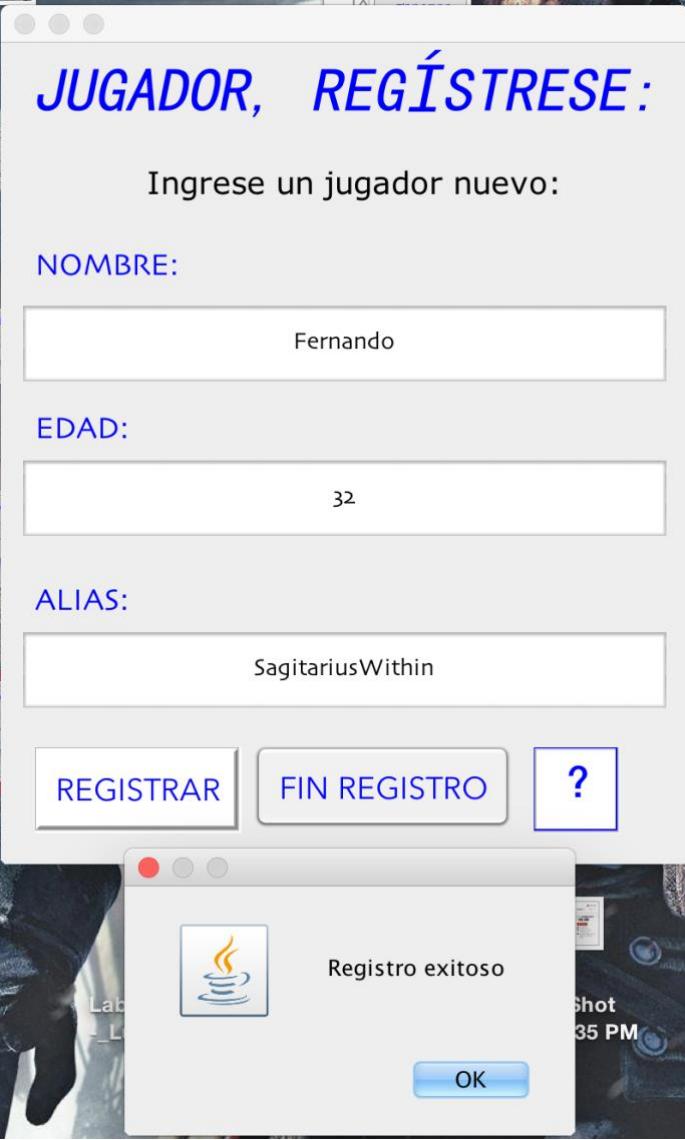
Ejemplo: -32, 193



OK

Se informa al usuario del error y se le solicita un nuevo ingreso

	de datos.	
<p>Se ingresa una edad con un formato diferente al esperado o necesitado.</p> <p>Ejemplo: "Veinticuatro" (String), 23.323 (float o double)</p>		OK
		<p>Se informa al usuario del error y se le solicita un nuevo ingreso</p>

	de datos.	
Se registra un jugador satisfactoriamente.	<p>JUGADOR, REGÍSTRESE:</p> <p>Ingrese un jugador nuevo:</p> <p>NOMBRE:</p> <p>Fernando</p> <p>EDAD:</p> <p>32</p> <p>ALIAS:</p> <p>SagittariusWithin</p> <p>REGISTRAR FIN REGISTRO ?</p>  <p>Se le permite al usuario ingresar tantos jugadores como quiera hasta que presione el JButton “Fin Registro”.</p>	OK
Se presiona el JButton “Fin Registro”.	<p>Se cierra la VentanaJugador y se abre la VentanaMenúInicial nuevamente, desde donde podrá en caso de así quererlo registrar más jugadores al seleccionar otra vez el JButton “Registro Jugador”.</p>	OK

VentanaSeleccionJugador

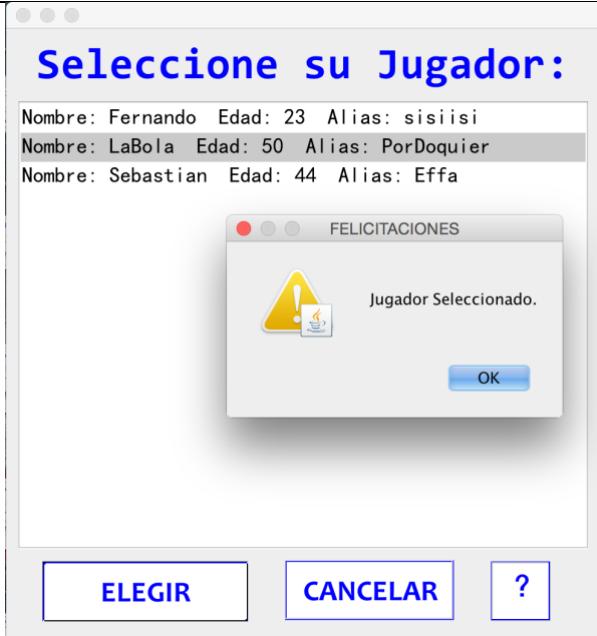
Se cliquea el JButton de Ayuda (“?”).



Se despliega un mensaje de ayuda explicando lo que debe realizar en la ventana actual para continuar.

OK

Se selecciona un jugador y se presiona el JButton “Elegir”.



Se registra el jugador respectivo, se cierra la VentanaSeleccionJugador y se abre nuevamente la VentanaMenuInicial.

OK

El jugador elegido es le mismo que el elegido por el otro jugador.	 <p>Se le avisa al jugador del problema y el programa vuelve a VentanaMenúInicial, cerrándose la VentanaSelecciónJugador.</p>	OK
Se cliquea el JButton de Ayuda ("?").	 <p>Se despliega un mensaje de ayuda explicando lo que debe realizar en la ventana actual para continuar.</p>	OK

Se elige un Color de la paleta de colores.



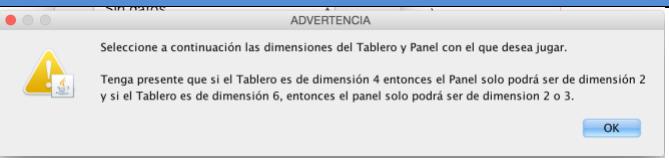
OK

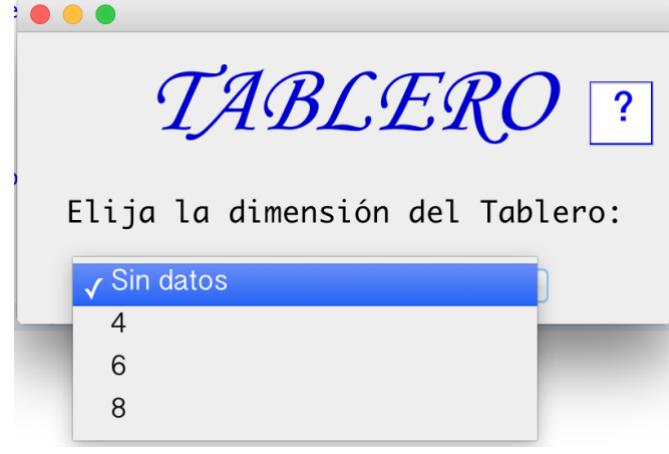


La Vista Previa (Preview) se actualiza cada vez que un nuevo color es seleccionado.

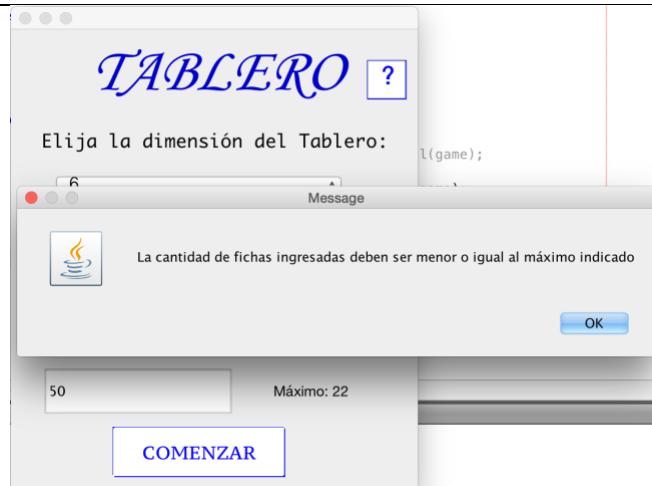
<p>Se presiona el JButton "Continuar" habiendo elegido un color oscuro.</p>	 <p>Se despliega un mensaje de confirmación al usuario, avisándole del problema circunstancial y preguntándole si desea seguir indistintamente o no.</p>	<p>OK</p>
---	--	------------------

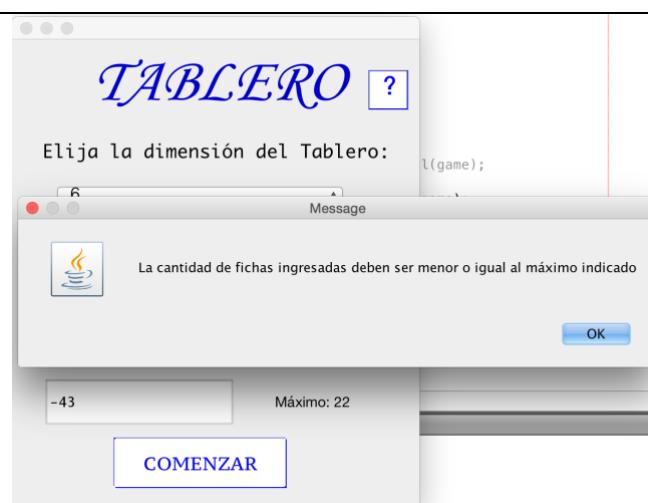
<p>Se presiona el JButton "Continuar" habiendo elegido un color relativamente claro (no dificultando la visibilidad de la ficha en el tablero).</p>	<p>La VentanaColor desaparece y se despliega nuevamente la VentanaMenuInicial con la opción de selección color del cual se acaba de seleccionar confirmada como ingresada.</p>	<p>OK</p>
---	--	------------------

<p>Se cliquea el JButton de Ayuda ("?").</p>	 <p>Se despliega un mensaje de ayuda explicando lo que debe realizar en la ventana actual para continuar.</p>	<p>OK</p>
--	---	------------------

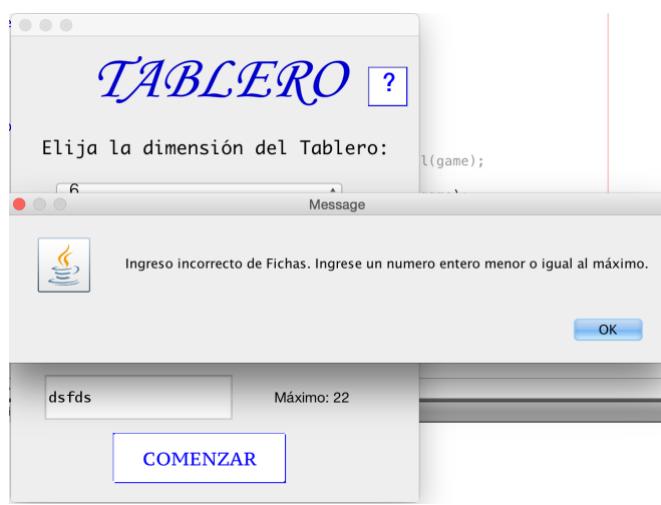
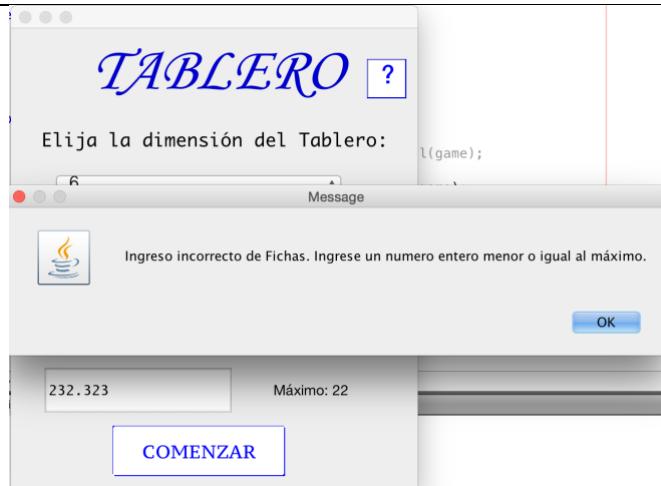
<p>Se quiere seleccionar la dimensión del Tablero y del Panel y se cliquea en el JComboBox respectivo.</p> <p>Tablero: 4, 6, 8</p> <p>Panel: 2, 3, 4</p>	 <p>Elija la dimensión del Tablero:</p> <ul style="list-style-type: none"> ✓ Sin datos 4 6 8 <p>Se despliega una lista con valores acordes.</p>	OK
<p>Se selecciona una dimensión del Tablero.</p> <p>Ejemplo: Se selecciona dimensión del Tablero 6, aparecen posibles dimensiones del Panel 2 y 3 únicamente.</p>	 <p>Elija la dimensión del Tablero:</p> <p>6</p> <p>Escoja la dimensión del Panel:</p> <ul style="list-style-type: none"> ✓ Sin datos 2 3 <p>Se despliegan únicamente las dimensiones posibles del Panel.</p>	OK
<p>Se ingresan datos aceptables sobre los mostrados.</p>	 <p>Elija la dimensión del Tablero:</p> <p>Sin datos</p>	OK

	 <p>Elija la dimensión del Tablero:</p> <p>6</p> <p>Escoja la dimensión del Panel:</p> <p>Sin datos</p> <p>Se modifica el tamaño de la VentanaTablero y deja visible las demás funcionalidades de la ventana.</p>	
<p>Se selecciona la opción predeterminada nuevamente del JComboBox referente a “Sin datos” (que no hace referencia a ninguna dimensión).</p>	 <p>Debe seleccionar algún dato deseado en ambas opciones</p> <p>OK</p> <p>Sin datos</p> <p>Se muestra un mensaje de repetir la operación otra vez.</p>	<p>OK</p>

Se seleccionar adecuadamente la dimensión del Tablero y del Panel.	 <p>Elija la dimensión del Tablero:</p> <p>6</p> <p>Escoja la dimensión del Panel:</p> <p>3</p> <p>Ingrese la cantidad de fichas:</p> <p>Máximo: 22</p> <p>COMENZAR</p>	OK
Se ingresa una cantidad de fichas de los jugadores negativa o mayor a la máxima mostrada.	 <p>La cantidad de fichas ingresadas deben ser menor o igual al máximo indicado</p> <p>OK</p>	OK



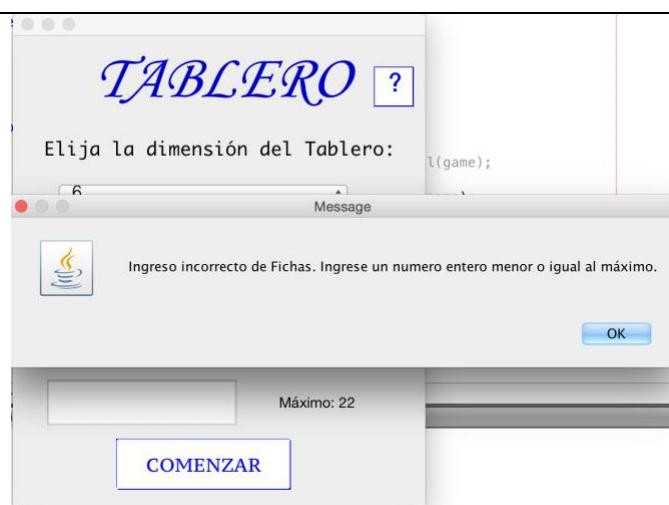
Se solicita un nuevo ingreso de datos y se le avisa al usuario de su error.



Se ingresa un valor no referente a un número entero.

Ejemplo: “Pepito” (String), 1.323
(float o double)

OK



Se solicita un nuevo ingreso de datos y se le avisa al usuario de su error.



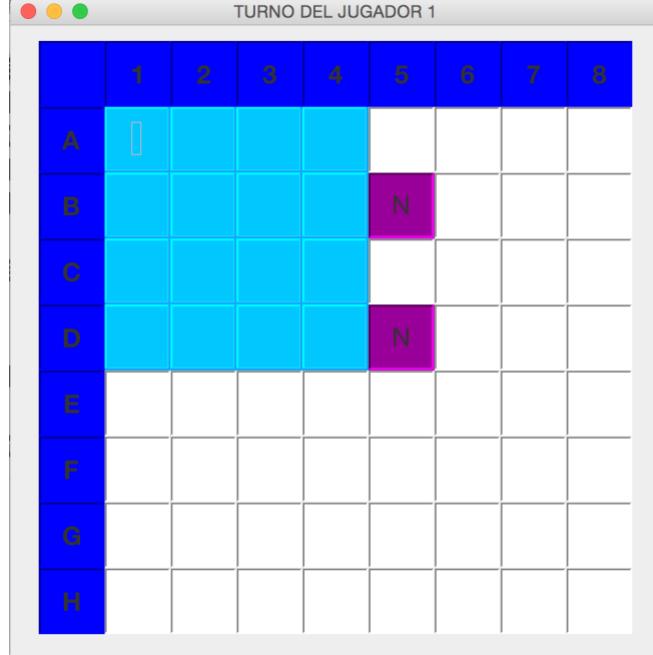
Se presiona el JButton "Comenzar"
una vez terminado todo lo
requerido en la VentanaTablero.

OK

Desaparece la VentanaTablero anterior y aparece la
VentanaJuego, la VentanaComandos y la
VentanalInstrucciones. Comienza el juego.

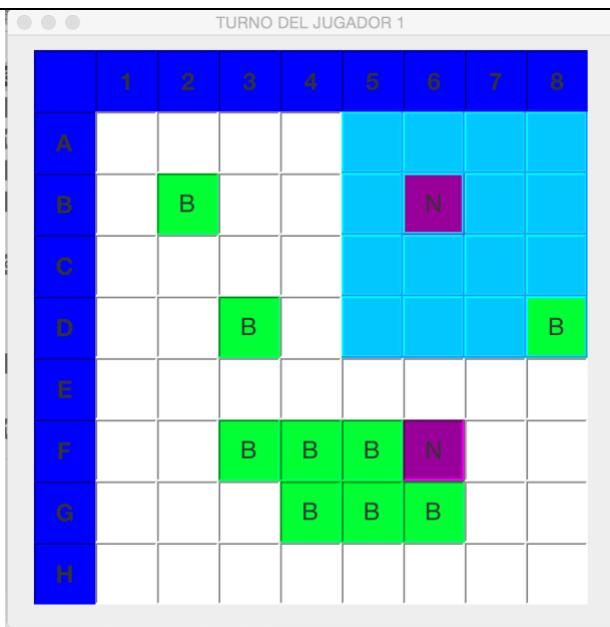
VentanaJuego

Se seleccionó una dimensión del Tablero y del Panel correcta. Ejemplo: Dimensión Tablero: 6 Dimensión Panel: 3	Las dimensiones desplegadas del Tablero y del Panel son las indicadas por el jugador en VentanaTablero.	OK
Se ingresan jugadas desde la VentanaComandos.	Las jugadas ingresadas son relacionadas con la VentanaJuego.	OK
Los turnos de los Jugadores cambian periódicamente.	Se modifica el título de la ventana acordemente.	OK
El jugador tiene fichas y selecciona una posición libre del tablero.	 <p>Se ingresa una ficha correspondiente del jugador, con la letra y el fondo correspondiente.</p>	OK
El jugador no tiene fichas disponibles y desea ingresar una ficha más al tablero.	Se le avisa al jugador de su error y se le pide realice otra jugada.	OK

<p>El jugador tiene fichas y selecciona una posición donde ya se encuentra una ficha.</p>	 <p>Se le avisa al usuario de su error y se le solicita realice otra jugada.</p>	OK
<p>Se ingresa el panel.</p>	 <p>Este aparece en la posición indicada con un efecto de relieve y tapa las fichas de abajo. El Panel se resalta en el tablero mediante un formato de relieve, bordes y color de fondo ("Background").</p>	OK

Se desea colocar fichas encima del panel.		OK
Se ingresa la ficha limpieza.		OK

Se mueve el panel.



OK

Todas las fichas de encima del panel se mueven con el mismo.

Se encuentra una simetría.



OK

Se le avisa al usuario y se la remarca cambiándole el color a la sección de la simetría hasta que termine el turno siguiente,

	cuento de fichas que se realizan en la VentanaComandos.	
El jugador se queda sin fichas para jugar.	Se le avisa al jugador del hecho y se le obliga a ingresar otra jugada. Si el panel ya se encuentra colocado y si ya colocó la ficha limpieza, deberá aguardar moviendo el panel hasta que el oponente termine sus jugadas.	OK
No hay más espacios libres.	Se le avisa al jugador del hecho y le obliga a colocar el panel o utilizar la ficha limpieza. En caso de no tener ningún jugador disponibles las jugadas anteriores, el juego terminará.	OK
Termina el juego.	Se cuentan las fichas del tablero y del panel para llegar a un ganador de la partida.	OK

VentanaComandos

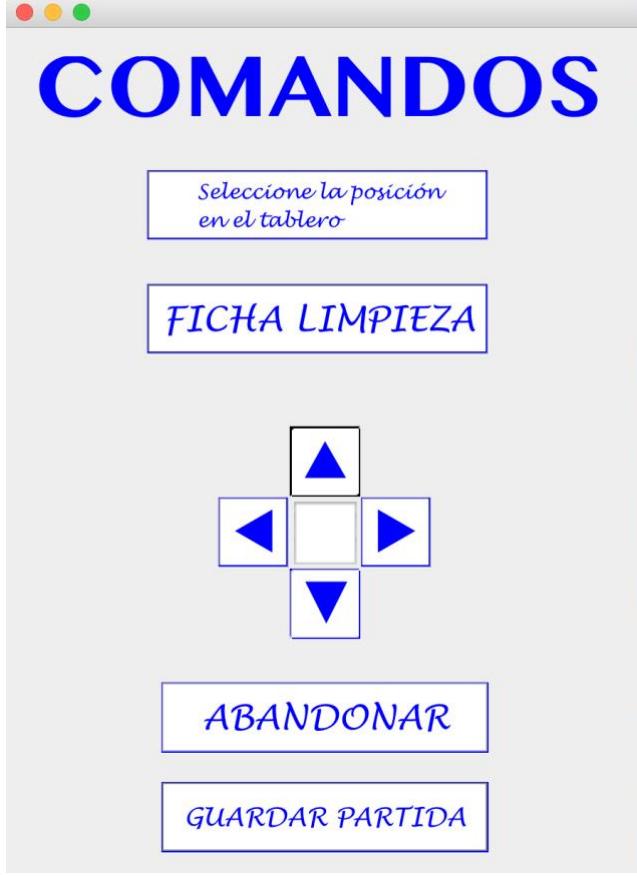
Se realizan acciones en VentanaComandos referentes a cambios a realizar en VentanaJuego.	Se realizan los cambios en VentanaJuego.	OK
--	--	----

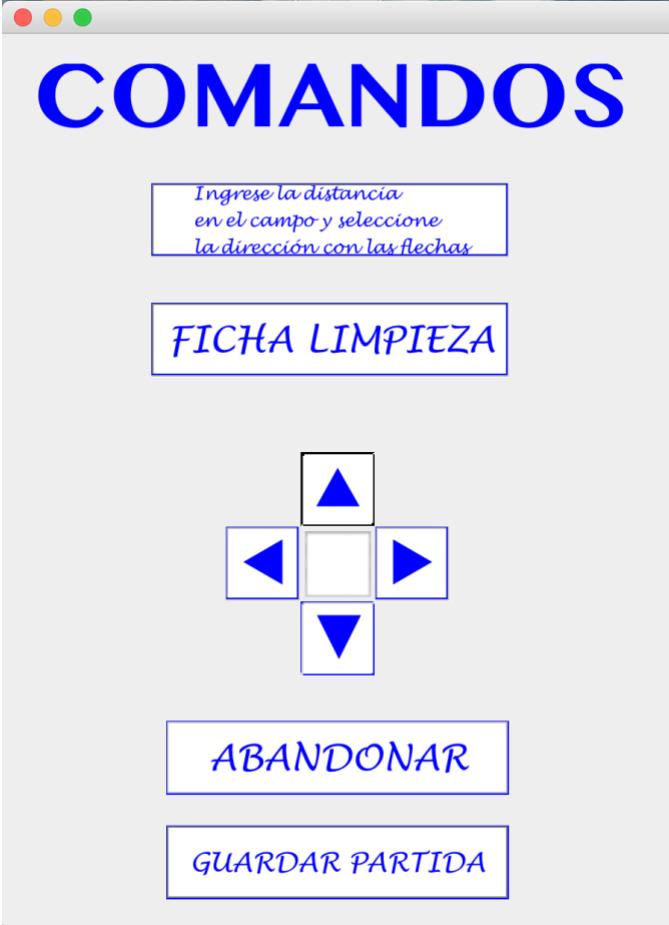


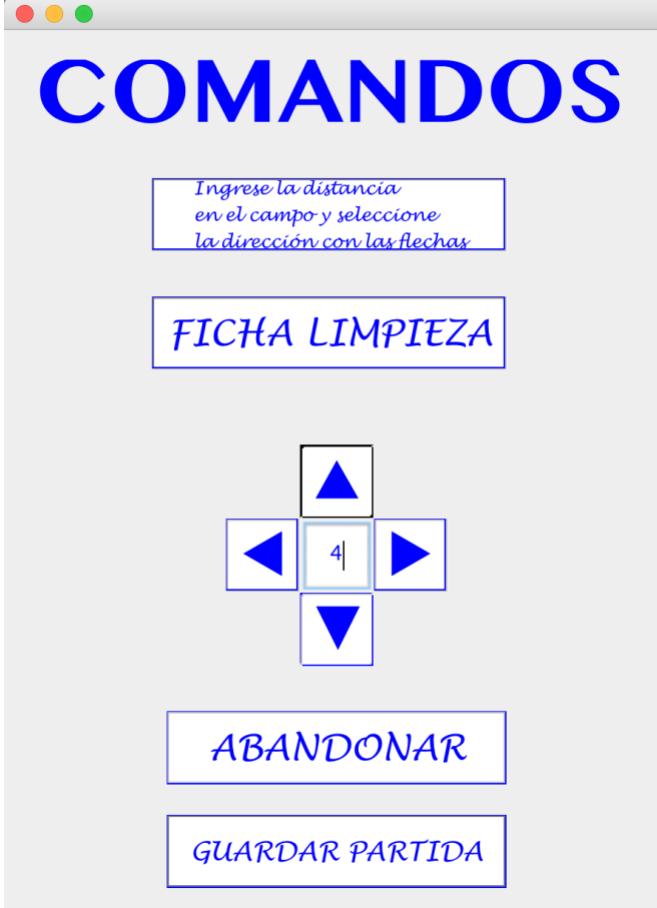
El jugador se queda sin fichas y quiere colocar otra ficha.

Se le avisa al jugador del problema y se le solicita una nueva jugada.

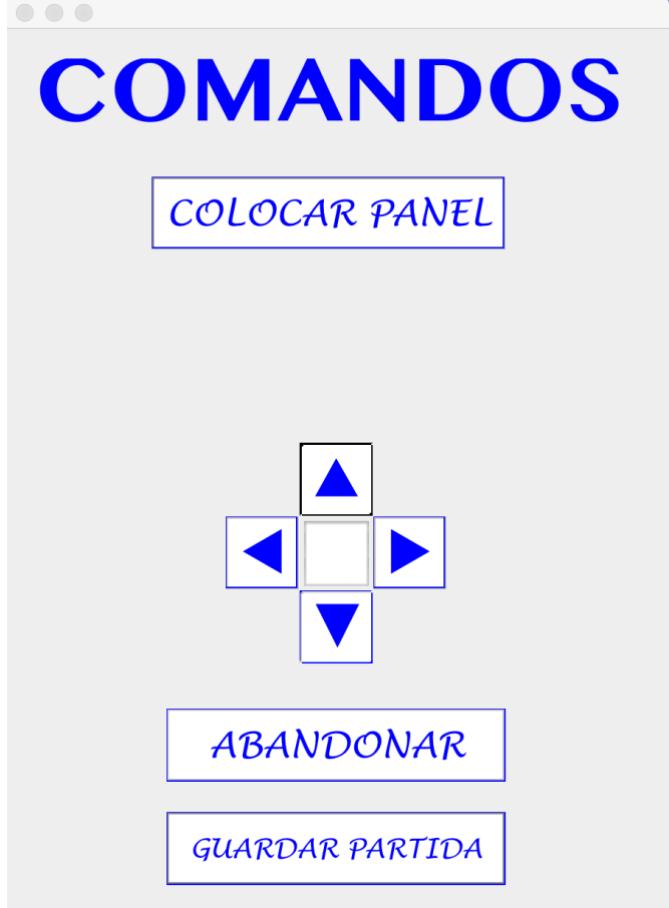
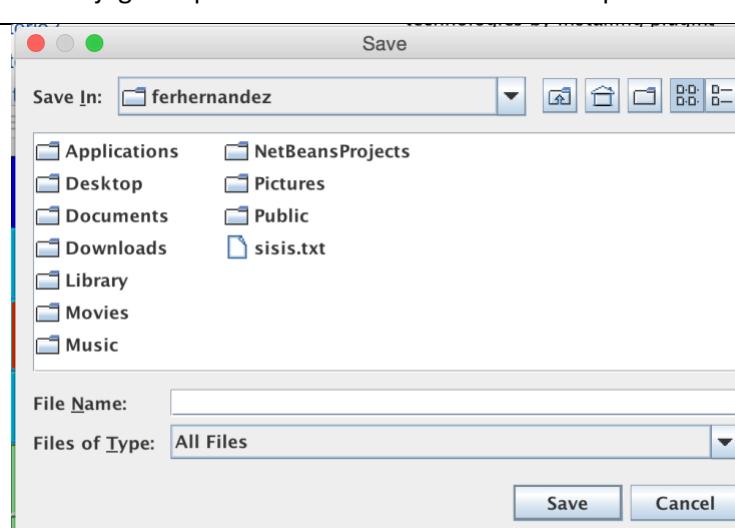
OK

<p>El jugador posee la posibilidad de ingresar el Panel al Tablero y desea hacerlo.</p>		OK
<p>El Panel ya se encuentra en el Tablero.</p>	<p>El JButton “Colocar Panel” se encuentra visible y con la mencionada inscripción. Al cliquearlo cambia su texto indicándole al Jugador lo que realizar.</p>	OK

	 <p>El jugador cliquea sobre el JButton "Mover Panel" ya que quiere mover el Panel.</p> <p>El JButton cambia de texto indicando como proceder (gracias a HTML).</p> <p>OK</p>	
--	--	--

<p>El jugador sigue las instrucciones y mediante el uso de los JButton como flechas y del JTextField indicar la dirección y cantidad de espacios en que lo desea mover.</p>		OK
<p>El jugador quiere mover el panel sin presionar el JButton "Mover Panel".</p>	<p>Se le avisa al usuario de su error y le solicita un nuevo ingreso de jugada.</p>	OK

		
<p>El jugador tiene habilitada la ficha limpieza y desea usarla. Para ello, cliquea sobre el JButton "Ficha Limpieza"</p> <p>Se le pide al usuario la posición en donde la desea colocar en la VentanaJuego.</p>		OK

<p>El jugador no tiene la ficha limpieza por un uso previo de la misma.</p>		<p>OK</p>
<p>El usuario desea guardar la partida para poder seguirla nuevamente en un futuro. En consecuencia, presiona el JButton "Guardar Partida".</p>	 <p>La partida en su totalidad es guardada como objeto en un archivo indicado por el usuario, de extensión automática</p>	<p>OK</p>

	".txt". A continuación, el juego termina.	
El juego termina al haberse agotado las jugadas o no haber espacios libres.	La VentanaComandos desaparece junto a la VentanaJuego y la VentanaInstrucciones.	OK

VentanaInstrucciones

Se selecciona el JButton "Aceptar" o la cruz roja de la esquina superior izquierda.	Se cierra la ventana, dejando para continuar la partida.	OK
Los elementos de la VentanaInstrucciones se modifican automáticamente de acuerdo al contexto de la Partida.		OK
El jugador del turno no posee fichas para ingresar.	Se elimina esa etiqueta de las instrucciones para ese jugador y se reordena la posición de las demás etiquetas acordemente.	OK
El jugador del turno ya no posee la ficha limpieza porque ya la uso.	Se elimina esa etiqueta de las instrucciones para ese jugador.	OK
El panel se encuentra colocado en el tablero.	Se modifica la etiqueta referente a colocar el panel y se muestra las indicaciones de los pasos a seguir para mover el	OK

panel.

VentanaFinJuego

Decorado con funcionamiento apropiado y correcto brindado por el efecto de los colores.



OK



GANADOR: SISIISI (JUGADOR 2)

CONTINUAR

CONTINUAR



GANADOR: SISIISI (JUGADOR 2)



Despliegue correcto del vencedor de la partida, mostrándose el Alias del ganador como marca distintiva.

OK

Se presiona el JButton “Aceptar” para continuar.

Se cierra la VentanaFinJuego y luego se abre la VentanaRanking.

OK

VentanaRanking

Despliegue automático de una lista ordenada decrecientemente por puntos de Jugadores.

OK

Incorporación de Persistencia.

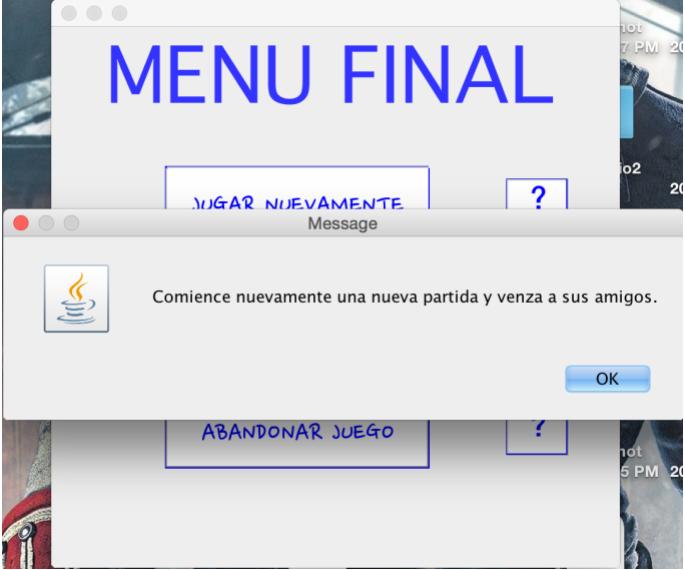
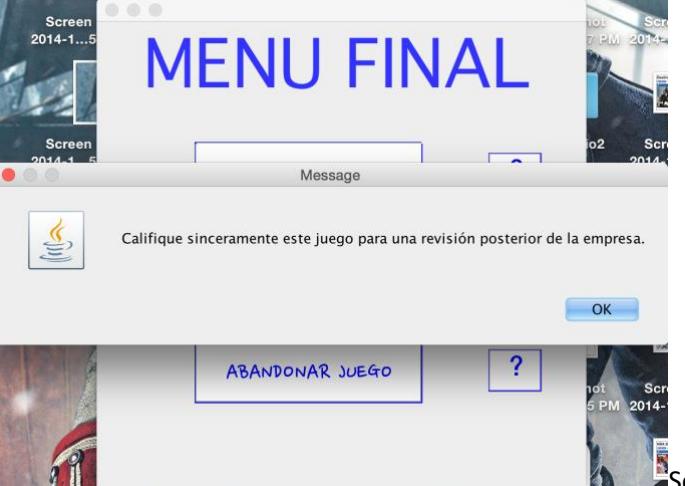
OK

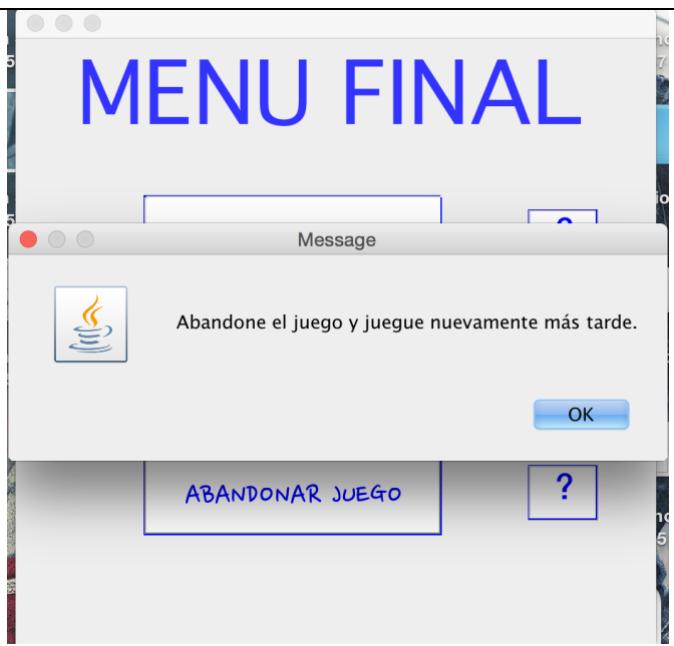
Se cliquea en el JButton “Continuar”.

Se cierra la VentanaRanking y se abre la VentanaMenuFinal.

OK

VentanaMenuFinal

<p>Se cliquea el JButton de Ayuda (“?”) que se encuentra al lado del JButton “Jugar Nuevamente”.</p>		<p>OK</p>
<p>Se cliquea el JButton de Ayuda (“?”) que se encuentra al lado del JButton “Calificar Juego”.</p>		<p>OK</p>

<p>Se cliquea el JButton de Ayuda (“?”) que se encuentra al lado del JButton “Abandonar Juego”.</p>	 <p>Se despliega un mensaje de ayuda explicando lo que sucedería si seleccionase “Abandonar Juego”.</p>	OK
<p>Se cliquea en el JButton “Jugar Nuevamente”.</p>	<p>Se cierra la VentanaMenuFinal y se abre la VentanaMenulinicial para brindar la oportunidad al usuario de un nuevo ingreso de jugadores.</p>	OK
<p>Se presiona el JButton “Calificar Juego”.</p>	<p>Se cierra la VentanaMenuFinal y se abre la VentanaCalificacion para asi los jugadores poder ingresar sus calificaciones.</p>	OK
<p>Se desea ingresar una calificación ya habiendo ingresado otra previamente por el mismo jugador.</p>	<p>Se le permite al usuario cambiar su comentario debido a un posible y real cambio de opinión por parte del mismo. El comentario del Jugador será sobreescrito con cada otro comentario ingresado para asi evitar un uso excesivo innecesario de datos en memoria.</p>	OK
<p>Se cliquea en el JButton “Abandonar Juego”.</p>	<p>Se cierra la VentanaMenuFinal y se abre la VentanaDespedida, última antes del fin del juego y del consiguiente cierre del programa.</p>	OK

VentanaCalificación

Se cliquea el JButton de Ayuda (“?”).



Se despliega un mensaje de ayuda explicando lo que debe realizar en la ventana actual para continuar.

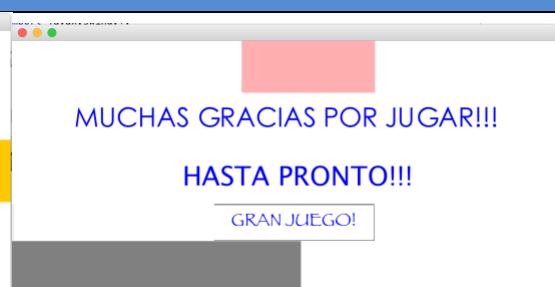
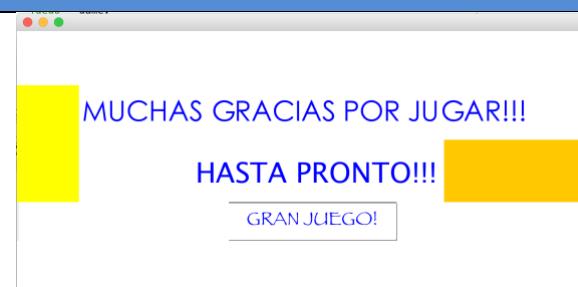
OK

Se presiona el JButton “Ingresar Calificaciones”.

Se registran los comentarios de ambos jugadores y se adjunta a su información personal. Se cierra la VentanaCalificación y se abre nuevamente la VentanaMenuFinal.

OK

VentanaDespedida



Decorado con funcionamiento apropiado y correcto brindado por el efecto de los colores.

OK

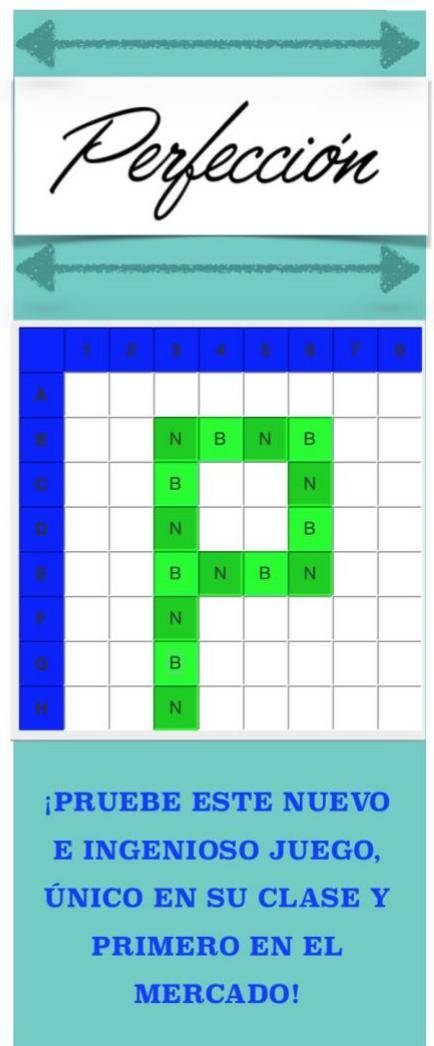
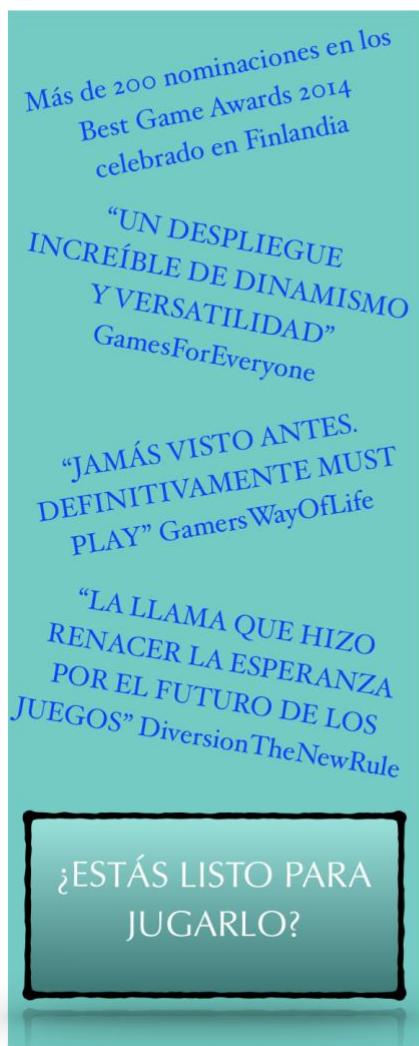
Se presiona el JButton “Gran Juego!”

Se cierra la VentanaDespedida y se termina el programa.

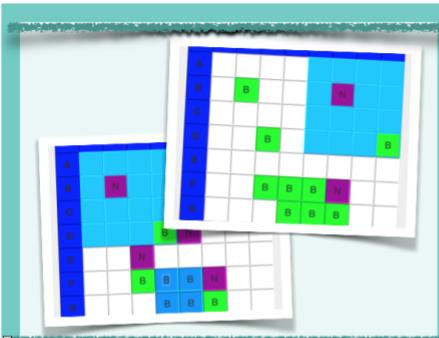
OK

Folleto del juego para promoción del producto:

Página 1. PORTADA



Nota: Creado en Pages versión 5.5 (2109) para MAC OS X.

Página 2. INTERIOR

El juego trata de generar simetrías mediante el ingreso de fichas del Jugador 1 y Jugador 2 para así abarcar la mayor cantidad de espacio del tablero con fichas propias.

Los jugadores poseerán un panel móvil de uso opcional que tapará las fichas de debajo de donde éste se coloque. Se podrá a su vez colocarle fichas encima que se moverán junto con el panel.

La partida termina cuando ambos jugadores colocaron todas sus fichas. El ganador resultará quien posee más cantidad de fichas visibles en el tablero.



Al comenzar la partida, se le solicitará su registro y la elección del tamaño del tablero y panel. Existen varios tamaños disponibles a su agrado.

¡¡¡Una vez listo, LA DIVERSIÓN COMENZARÁ!!!



¿PODRÁ CONVERTIRSE EN NUESTRO MEJOR JUGADOR?

Al finalizar la partida, poseerá varias opciones a su gusto para proseguir, viendo su posición en el ranking mundial de nuestro juego al igual que tener la posibilidad de calificar nuestro juego y que su comentario y puntaje lo sepa todo el mundo.

Nota: Creado en Pages versión 5.5 (2109) para MAC OS X.

Emprendimiento de una Empresa de Videojuegos

Concepto Asociado: Inversor Ángel (*Angel Investor*)

Entidad que brinda sus fondos y conocimientos (a diferencia de las entidades de capital de riesgo) fundamentales a un emprendimiento para comenzar con su negocio o servicio (conocido como capital semilla) a cambio usualmente de una ventaja financiera posterior. De esta manera, eligen particularmente los lugares donde invertir mediante una valoración de los planes de negocio brindadas. Por ello, se encuentran bajo un gran riesgo, por lo que necesitan lo que económicamente se conoce como ROI (Retorno sobre la Inversión) y que se refiere a la posibilidad de recuperar el dinero invertido. Para tener como positivo el aporte de una empresa todo inversor ángel espera un estimado mínimo de 10 veces en 5 años del apartado inicialmente en promedio.

El término proviene de la década del 70, utilizado en estudios de distintas Universidades de Estados Unidos (en especial, la de New Hampshire) sobre negocios pioneros en dicho país. Los propósitos de los agentes inversores NO tienen porque ser únicamente monetarios, ya que también sirve para actualizarse y expandir sus conocimientos a diferentes áreas de los negocios al igual que para enseñar a las nuevas generaciones y formar un buen legado.

FUENTE basada para redactar la definición:

- ✓ Entrepreneur: <http://www.entrepreneur.com/article/52742>
- ✓ Investopedia: <http://www.investopedia.com/terms/a/angelinvestor.asp>
- ✓ Angel Capital Association:
<http://www.angelcapitalassociation.org/entrepreneurs/faqs/>

Clases del Dominio (y su respectivo código):

Package Dominio

Clase ArchivoLectura

```
// Obligatorio 2 - PROGRAMACIÓN 2
// Fernando Agustín Hernández Gobertti (173631) - Ingeniería en
Telecomunicaciones
// Sebastián Effa Gallego (193248) - Ingeniería en Electrónica

package Dominio;

import java.io.*;

public class ArchivoLectura {

    String linea = "";
    BufferedReader in;

    public ArchivoLectura (String unNombre){
        try{
            in = new BufferedReader (new FileReader (unNombre));
        }
        catch(FileNotFoundException e){}
    }

    public boolean hayMasLineas(){
```

```
try{
    linea = in.readLine();
}
catch(IOException e){
    linea = null;
}
return (linea!=null);
}

public String linea(){
    return linea;
}

public boolean cerrar(){
    boolean ok = true;
    try{
        in.close();
    }
    catch(Exception e){
        ok = false;
    }
    return ok;
}

}
```

Fin de Clase ArchivoLectura

Clase ControlArchivos

```
// Obligatorio 2 - PROGRAMACIÓN 2
// Fernando Agustín Hernández Gobertti (173631) - Ingeniería en
Telecomunicaciones
// Sebastián Effa Gallego (193248) - Ingeniería en Electrónica

package Dominio;

import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.filechooser.*;

public class ControlArchivos {

    //Método para Guardar una Partida al desearlo un usuario
    public static boolean guardarPartida(Partida game, JFrame
ventana) {
        boolean ok = false;
        try {
            JFileChooser guardar = new JFileChooser();
            int returnVal = guardar.showSaveDialog(ventana);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
                File direccion = guardar.getSelectedFile();
                ObjectOutputStream salida = new
ObjectOutputStream(new FileOutputStream(direccion));
                salida.writeObject(game);
                salida.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
        ok = true;
    }
} catch (Exception e) {}
return ok;
}

//Método para Buscar y Abrir un Archivo seleccionado por el
usuario, usado para leer jugadores o abrir una partida
public static String buscarArchivo(String titulo,
FileNameExtensionFilter filtro) {
    String direccion = null;
    JFileChooser buscador = new JFileChooser();
    buscador.setFileFilter(filtro);
    buscador.setDialogTitle(titulo);
    int opcion = buscador.showOpenDialog(buscador);
    buscador.setVisible(true);
    if (opcion == JFileChooser.APPROVE_OPTION) {
        direccion =
buscador.getSelectedFile().getAbsolutePath();
    }
    return direccion;
}

//Método que graba jugadores automaticamente al salir del
programa desde cualquier ventana
public static void grabarJugadoresAuto(ArrayList <Jugador>
jugadores){
    File archivo = new File("datosJugadores");
    if(!jugadores.isEmpty()){
        for(int i=0;i<jugadores.size();i++){

```

```
        jugadores.get(i).setTieneFichaLimpieza(true);  
    }  
    try{  
        ObjectOutputStream salida = new  
ObjectOutputStream(new FileOutputStream(archivo.getAbsolutePath()));  
        salida.writeObject(jugadores);  
        salida.close();  
    }catch(Exception e){}  
}  
  
}  
  
//Método que lee jugadores automáticamente al iniciar el  
programa en caso que existan  
public static ArrayList<Jugador> leerJugadoresAuto(){  
    ArrayList<Jugador> jugadores = new ArrayList<>();  
    File archivo = new File("datosJugadores");  
    if(archivo.exists()){  
        try{  
            ObjectInputStream entrada = new ObjectInputStream(new  
FileInputStream(archivo.getAbsolutePath()));  
            jugadores = (ArrayList <Jugador>)entrada.readObject();  
            entrada.close();  
        }  
        catch(Exception e){}  
    }  
    return jugadores;  
}  
}
```

Fin de Clase ControlArchivos

Clase Jugador

```
// Obligatorio 2 - PROGRAMACIÓN 2
// Fernando Agustín Hernández Gobertti (173631) - Ingeniería en
Telecomunicaciones
// Sebastián Effa Gallego (193248) - Ingeniería en Electrónica

package Dominio;

import java.awt.*;
import java.io.*;

public class Jugador implements Serializable{

    //Atributos de la clase Jugador
    private String nombre, alias, ficha, calificacion;
    private int edad, partidasGanadas, partidasJugadas,
cantidadFichas, puntos, posicionRanking;
    private boolean tieneFichaLimpieza, listoRegistroJugador,
listoColorJugador;
    private Color color;

    //Constructor por Defecto (sin Parámetros)
    public Jugador(){
        this.setNombre("Sin nombre.");
        this.setEdad(1);
        this.setAlias("Sin alias");
        this.setFicha("Sin ficha");
        this.setPartidasGanadas(0);
        this.setPartidasJugadas(0);
    }
}
```

```
        this.setCantidadFichas(0);
        this.setPuntos(0);
        this.setTieneFichaLimpieza(true);
        this.setListaRegistroJugador(false);
        this.setListaColorJugador(false);
    }

    //Constructor con parámetros
    public Jugador(String nombre, int edad, String alias, int
partidasJugadas, int partidasGanadas, String calificacion){
        this.setNombre(nombre);
        this.setEdad(edad);
        this.setAlias(alias);
        this.setPartidasJugadas(partidasJugadas);
        this.setPartidasGanadas(partidasGanadas);
        this.setCalificacion(calificacion);
    }

    //Métodos de la clase Jugador
    public String getNombre(){
        return nombre;
    }

    public int getEdad(){
        return edad;
    }

    public String getAlias(){
        return alias;
    }
```

```
public String getFicha(){
    return ficha;
}

public int getPartidasGanadas(){
    return partidasGanadas;
}

public int getPartidasJugadas(){
    return partidasJugadas;
}

public int getCantidadFichas(){
    return cantidadFichas;
}

public String getCalificacion() {
    return calificacion;
}

public Color getColor() {
    return color;
}

public int getPuntos() {
    return puntos;
}

public boolean isTieneFichaLimpieza() {
    return tieneFichaLimpieza;
```

```
}
```

```
public boolean isListoColorJugador() {
```

```
    return listoColorJugador;
```

```
}
```

```
public boolean isListoRegistroJugador() {
```

```
    return listoRegistroJugador;
```

```
}
```

```
public int getPosicionRanking() {
```

```
    return posicionRanking;
```

```
}
```

```
public void setNombre(String unNombre){
```

```
    this.nombre=unNombre;
```

```
}
```

```
public void setEdad(int unaEdad){
```

```
    this.edad=unaEdad;
```

```
}
```

```
public void setAlias(String unAlias){
```

```
    this.alias=unAlias;
```

```
}
```

```
public void setFicha(String unaFicha){
```

```
    this.ficha=unaFicha;
```

```
}
```

```
public void setPartidasGanadas(int unaPartidaGanada){  
    this.partidasGanadas=unaPartidaGanada;  
}  
  
public void setPartidasJugadas(int unaPartidaJugada){  
    this.partidasJugadas=unaPartidaJugada;  
}  
  
public void setCantidadFichas (int unaCantidadFichas){  
    this.cantidadFichas=unaCantidadFichas;  
}  
  
public void setCalificacion(String calificacion) {  
    this.calificacion = calificacion;  
}  
  
public void setTieneFichaLimpieza(boolean tieneFichaLimpieza) {  
    this.tieneFichaLimpieza = tieneFichaLimpieza;  
}  
  
public void setPuntos(int puntos) {  
    this.puntos = puntos;  
}  
  
public void setColor(Color color) {  
    this.color = color;  
}  
  
public void setListaRegistroJugador(boolean  
listaRegistroJugador) {
```

```
        this.listoRegistroJugador = listoRegistroJugador;
    }

    public void setListaColorJugador(boolean listoColorJugador) {
        this.listaColorJugador = listoColorJugador;
    }

    public void setPosicionRanking(int posicionRanking) {
        this.posicionRanking = posicionRanking;
    }

    @Override
    public String toString (){
        return "POSICIÓN: "+this.getPosicionRanking() + ".\n NOMBRE:  
" + this.getNombre() + ", \tEDAD: " + this.getEdad() + ", \tALIAS:  
" + this.getAlias() + "\nPartidas  
Jugadas:" + this.getPartidasJugadas() + ", \tPartidas Ganadas:  
" + this.getPartidasGanadas() + ", \tCalificación:  
" + this.getCalificacion() + "\n";
    }

    //Redefinición del Método equals
    public boolean equals(Jugador o){
        boolean ok = this.getAlias().equals(o.getAlias());
        return ok;
    }
}
```

Fin de Clase Jugador

Clase Tablero

```
// Obligatorio 2 - PROGRAMACIÓN 2
// Fernando Agustín Hernández Gobertti (173631) - Ingeniería en
Telecomunicaciones
// Sebastián Effa Gallego (193248) - Ingeniería en Electrónica

package Dominio;

import java.io.*;

public class Tablero implements Serializable{

    //Atributos de la clase Tablero
    private String[][] matrizTablero, tableroConPanel, panel,
tableroVacio;

    //Métodos de la clase Tablero
    public String[][] getTableroVacio() {
        return tableroVacio;
    }

    public void setTableroVacio(String[][] tableroVacio) {
        this.tableroVacio = tableroVacio;
    }

    public String[][] getPanel() {
        return panel;
    }
}
```

```
public void setPanel(String[][] panel) {  
    this.panel = panel;  
}  
  
public String[][] getMatrizTablero() {  
    return matrizTablero;  
}  
  
public void setMatrizTablero(String[][] matrizTablero) {  
    this.matrizTablero = matrizTablero;  
}  
  
public void setFichaMatrizTablero(int fila, int columna, String  
letra) {  
    this.matrizTablero[fila][columna] = letra;  
}  
  
public String[][] getTableroConPanel() {  
    return tableroConPanel;  
}  
  
public void setTableroConPanel(String[][] tableroConPanel) {  
    this.tableroConPanel = tableroConPanel;  
}  
  
public void setFichaTableroConPanel(int fila, int columna,  
String letra) {  
    this.tableroConPanel[fila][columna] = letra;  
}
```

```
//Método para buscar simetrías (utiliza métodos simetrías debajo)

public static int [] buscarSimetrias(String [][] tablero, int filaFicha, int columnaFicha){

    int [] datos = {0,0,0}; //Array cuyo primer valor refiere a la dimensión de la simetría (0 es que no hay), segundo y tercer valor son la posición

    datos      =      simetrias4x4Lados(tablero,      filaFicha,
columnaFicha);

    if(datos[0] == 0){

        datos      =      simetrias4x4Lados(tablero,      (filaFicha-1),
columnaFicha);

    }

    if(datos[0] == 0){

        datos      =      simetrias4x4Lados(tablero,      (filaFicha-2),
columnaFicha);

    }

    if(datos[0] == 0){

        datos      =      simetrias4x4Lados(tablero,      filaFicha,
(columnaFicha-1));

    }

    if(datos[0] == 0){

        datos      =      simetrias4x4Lados(tablero,      (filaFicha-2),
(columnaFicha-2));

    }

    if(datos[0] == 0){

        datos      =      simetrias4x4Medio(tablero,      filaFicha,
columnaFicha);

    }

}
```

```

        if(datos[0] == 0){
            datos = simetrias3x3(tablero, filaFicha, columnaFicha);
        }
        if(datos[0] == 0){
            datos = simetrias2x2(tablero, filaFicha, columnaFicha);
        }
        return datos;
    }

    //Método general de detección de simetrías utilizado en los
    métodos posteriores

    public static int [] simetriaGeneral (String [][][]
    matrizAuxiliar, int simetria, int filaFicha, int columnaFicha,
            int pasoFila, int pasoColumna, String [][] tablero, int
    [] datos){
        if((datos[0]==0)                                     &&
(simetriaValida(tablero,simetria,filaFicha,columnaFicha))){
            for(int fila = filaFicha; fila < pasoFila; fila++){
                for(int columna = columnaFicha; columna <
pasoColumna; columna++){
                    matrizAuxiliar[fila - filaFicha][columna -
columnaFicha] = tablero[fila][columna];
                }
            }
            if(simetria == 2){
                datos = simetria2x2 (matrizAuxiliar, filaFicha,
columnaFicha);
            }
            if(simetria == 3){
                datos = simetria3x3 (matrizAuxiliar, filaFicha,

```

```
columnaFicha);
    }
    if(simetria == 4){
        datos = simetria4x4 (matrizAuxiliar, filaFicha,
columnaFicha);
    }
}
return datos;
}

//Método general de verificación de simetría utilizado en los
métodos posteriores

public static int [] verificacionSimetria(int [] datos, int
dimension, boolean haySimetria, int filaFicha, int columnaFicha){
    if(haySimetria){
        if(dimension == 2){
            datos[0] = 2;
            datos[1] = filaFicha;
            datos[2] = columnaFicha;
        }
        if(dimension == 3){
            datos[0] = 3;
            datos[1] = filaFicha;
            datos[2] = columnaFicha;
        }
        if(dimension == 4){
            datos[0] = 4;
            datos[1] = filaFicha;
            datos[2] = columnaFicha;
        }
    }
}
```

```
        }

        return datos;
    }

    //Método para detectar las simetrías de 4x4
    public static int [] simetria4x4 (String [][] aux4x4, int
filaFicha, int columnaFicha){

        int [] datos = {0,0,0}; //Array cuyo primer valor refiere a
        la dimensión de la simetría (0 es que no hay), segundo y tercer valor son
        la posición

        try{

            datos = verificacionSimetria(datos, 4, aux4x4
[0][0].equals(aux4x4 [0][3]) && aux4x4 [0][1].equals(aux4x4 [0][2]) &&
                aux4x4 [1][0].equals(aux4x4 [1][3]) && aux4x4
[1][1].equals(aux4x4 [1][2]) &&
                aux4x4 [2][0].equals(aux4x4 [2][3]) && aux4x4
[2][1].equals(aux4x4 [2][2]) &&
                aux4x4 [3][0].equals(aux4x4 [3][3]) && aux4x4
[3][1].equals(aux4x4 [3][2]), filaFicha, columnaFicha);

            datos = verificacionSimetria(datos, 4, datos[0] ==
0 && aux4x4 [0][0].equals(aux4x4 [3][0]) && aux4x4 [1][0].equals(aux4x4
[2][0]) &&
                aux4x4 [0][1].equals(aux4x4 [3][1]) && aux4x4
[1][1].equals(aux4x4 [2][1]) &&
                aux4x4 [0][2].equals(aux4x4 [3][2]) && aux4x4
[1][2].equals(aux4x4 [2][2]) &&
                aux4x4 [0][3].equals(aux4x4 [3][3]) && aux4x4
[1][3].equals(aux4x4 [2][3]), filaFicha, columnaFicha);

        }
        catch(Exception e){

    }
```

```

        datos [0] = 0;
    }
    return datos;
}

//Método para verificar las simetrías de 4x4 donde la pieza
colocada se encuentre en sus extremos

public static int [] simetrias4x4Lados(String [][] tablero, int
filaFicha, int columnaFicha){

    int [] datos = {0,0,0}; //el primer valor refiere a la
dimension de la simetria (0 es que no hay), segundo y tercer valor son la
posición

    String [][] aux4x4 = new String [4][4];
    if(simetriaValida(tablero,4,filaFicha,columnaFicha)){
        for (int fila = filaFicha; fila < (filaFicha + 4);
fila++) {
            for (int columna = columnaFicha; columna <
(columnaFicha + 4); columna++) {
                aux4x4[fila - filaFicha][columna -
columnaFicha] = tablero[fila][columna];
            }
        }
        datos = simetria4x4 (aux4x4, filaFicha, columnaFicha);
    }

    //se buscan simetrías en distintas posiciones
    datos = simetriaGeneral(aux4x4, 4, filaFicha-3,
columnaFicha, filaFicha+1, columnaFicha+4, tablero, datos);
    datos = simetriaGeneral(aux4x4, 4, filaFicha, columnaFicha-
3, filaFicha+4, columnaFicha+1, tablero, datos);
    datos = simetriaGeneral(aux4x4, 4, filaFicha-3,

```

```

columnaFicha-3, filaFicha+1, columnaFicha+1, tablero, datos);

try{
    for (int fila = 0; fila < 3; fila++) {
        for (int columna = 0; columna < 3; columna++) {
            if(aux4x4[fila][columna].equals(" ")){
                datos [0] = 0;
            }
        }
    }
    catch (Exception e){
        datos [0] = 0;
    }
    return datos;
}

//Método para verificar el resto de las simetrías de 4x4 que no
se verificaron en el método simetrias4x4Lados(String[][][], int, int)
public static int [] simetrias4x4Medio(String [][] tablero, int
filaFicha, int columnaFicha){

    int [] datos = {0,0,0}; //Array cuyo primer valor refiere a
la dimensión de la simetría (0 es que no hay), segundo y tercer valor son
la posición

    String [][] aux4x4 = new String [4][4];
    if(simetriaValida(tablero,4,(filaFicha-1),(columnaFicha-
1))){
        for (int fila = (filaFicha-1); fila < (filaFicha + 3);
fila++) {
            for (int columna = (columnaFicha-1); columna <
(columnaFicha + 3); columna++) {

```

```

aux4x4[fila] - (filaFicha-1)][columna] -
(columnaFicha-1)] = tablero[fila][columna];
}
}

datos = simetria4x4 (aux4x4, (filaFicha-1),
(columnaFicha-1));
}

//se buscan simetrías en distintas posiciones
datos = simetriaGeneral(aux4x4, 4, filaFicha-2,
columnaFicha-1, filaFicha+2, columnaFicha+3, tablero, datos);
datos = simetriaGeneral(aux4x4, 4, filaFicha-2,
columnaFicha-2, filaFicha+2, columnaFicha+2, tablero, datos);
datos = simetriaGeneral(aux4x4, 4, filaFicha-1,
columnaFicha-2, filaFicha+3, columnaFicha+2, tablero, datos);

try{
    for (int fila = 0; fila < 4; fila++) {
        for (int columna = 0; columna < 4; columna++) {
            if(aux4x4[fila][columna].equals(" ")){
                datos [0] = 0;
            }
        }
    }
    catch (Exception e){
        datos [0] = 0;
    }
    return datos;
}

//Método para detectar las simetrías de 3x3

```

```

public static int [] simetria3x3 (String [][] aux3x3, int
filaFicha, int columnaFicha){

    int [] datos = {0,0,0}; //Array cuyo primer valor refiere a
    la dimension de la simetria (0 es que no hay), segundo y tercer valor son
    la posición

    try{
        datos      =     verificacionSimetria      (datos,      3,
aux3x3[0][0].equals(aux3x3[0][2]) && aux3x3[1][0].equals(aux3x3[1][2]) &&
                           aux3x3[2][0].equals(aux3x3[2][2]),     filaFicha,
columnaFicha);
        datos      =     verificacionSimetria      (datos,      3,
aux3x3[0][0].equals(aux3x3[2][0]) && aux3x3[0][1].equals(aux3x3[2][1]) &&
                           aux3x3[0][2].equals(aux3x3[2][2]),     filaFicha,
columnaFicha);
    }
    catch(NullPointerException e){
        datos [0] = 0;
    }
    return datos;
}

//Método para verificar las simetrías de 3x3 respecto a la
última pieza colocada

public static int [] simetrias3x3(String [][] tablero, int
filaFicha, int columnaFicha){

    int [] datos = {0,0,0}; //Array cuyo primer valor refiere a
    la dimension de la simetria (0 es que no hay), segundo y tercer valor son
    la posición

    String [][] aux3x3 = new String [3][3];
    if(simetriaValida(tablero,3,filaFicha,columnaFicha)){

```

```
        for (int fila = filaFicha; fila < (filaFicha + 3);
fila++) {
            for (int columna = columnaFicha; columna <
(columnaFicha + 3); columna++) {
                aux3x3[fila - filaFicha][columna -
columnaFicha] = tablero[fila][columna];
            }
        }
        datos = simetria3x3 (aux3x3, filaFicha, columnaFicha);
    }

    //se buscan simetrías en distintas posiciones
    datos = simetriaGeneral(aux3x3, 3, filaFicha-2,
columnaFicha, filaFicha+1, columnaFicha+3, tablero, datos);
    datos = simetriaGeneral(aux3x3, 3, filaFicha-2,
columnaFicha-2, filaFicha+1, columnaFicha+1, tablero, datos);
    datos = simetriaGeneral(aux3x3, 3, filaFicha, columnaFicha-
2, filaFicha+3, columnaFicha+1, tablero, datos);
    datos = simetriaGeneral(aux3x3, 3, filaFicha-1,
columnaFicha, filaFicha+2, columnaFicha+3, tablero, datos);
    datos = simetriaGeneral(aux3x3, 3, filaFicha-1,
columnaFicha-2, filaFicha+2, columnaFicha+1, tablero, datos);
    datos = simetriaGeneral(aux3x3, 3, filaFicha-2,
columnaFicha-1, filaFicha+1, columnaFicha+2, tablero, datos);
    datos = simetriaGeneral(aux3x3, 3, filaFicha, columnaFicha-
1, filaFicha+3, columnaFicha+2, tablero, datos);
    datos = simetriaGeneral(aux3x3, 3, filaFicha-1,
columnaFicha-1, filaFicha+2, columnaFicha+2, tablero, datos);
    try{
        for (int fila = 0; fila < 3; fila++) {
            for (int columna = 0; columna < 3; columna++) {
```

```
if(aux3x3[fila][columna].equals(" ")){
    datos [0] = 0;
}
}

}

}

catch (Exception e){
    datos [0] = 0;
}

return datos;
}

//Método para detectar las simetrías de 2x2
public static int [] simetria2x2(String [][] aux2x2, int
filaFicha, int columnaFicha){

    int [] datos = {0,0,0}; //Array cuyo primer valor refiere a
    la dimensión de la simetría (0 es que no hay), segundo y tercer valor son
    la posición

    try{
        datos     =     verificacionSimetria(datos,     2,     aux2x2
[0][0].equals(aux2x2 [0][1])  &&  aux2x2 [1][0].equals(aux2x2 [1][1]),
filaFicha, columnaFicha);
    }
    catch(Exception e){
        datos [0] = 0;
    }
    try{
        datos     =     verificacionSimetria(datos,     2,     aux2x2
[0][0].equals(aux2x2 [1][0])  &&  aux2x2 [0][1].equals(aux2x2 [1][1]),
filaFicha, columnaFicha);
    }
}
```

```
        }

        catch(Exception e){
            datos [0] = 0;
        }

        return datos;
    }

    //Método para verificar las simetrías de 2x2 respecto a la
última pieza colocada

    public static int [] simetrias2x2(String [][] tablero, int
filaFicha, int columnaFicha){

        int [] datos = {0,0,0}; //Array cuyo primer valor refiere a
la dimensión de la simetría (0 es que no hay), segundo y tercer valor son
la posición

        String [][] aux2x2 = new String [2][2];
        if(simetriaValida(tablero,2,filaFicha,columnaFicha)){
            for (int fila = filaFicha; fila < (filaFicha + 2);
fila++) {
                for (int columna = columnaFicha; columna <
(columnaFicha + 2); columna++) {
                    aux2x2[fila - filaFicha][columna -
columnaFicha] = tablero[fila][columna];
                }
            }
            datos = simetria2x2 (aux2x2, filaFicha, columnaFicha);
        }

        //se buscan simetrías en distintas posiciones
        datos = simetriaGeneral(aux2x2, 2, filaFicha-1,
columnaFicha, filaFicha+1, columnaFicha+2, tablero, datos);
        datos = simetriaGeneral(aux2x2, 2, filaFicha-1,
```

```
columnaFicha-1, filaFicha+1, columnaFicha+1, tablero, datos);
    datos = simetriaGeneral(aux2x2, 2, filaFicha, columnaFicha-
1, filaFicha+2, columnaFicha+1, tablero, datos);
try{
    for (int fila = 0; fila < 2; fila++) {
        for (int columna = 0; columna < 2; columna++) {
            if(aux2x2[fila][columna].equals(" ")){
                datos [0] = 0;
            }
        }
    }
    catch (Exception e){
        datos [0] = 0;
    }
    return datos;
}

//Método para verificar si una simetría es válida
public static boolean simetriaValida (String[][] matriz, int
dimensionSimetria, int fila, int columna){
    boolean validez = false;
    boolean hayBlancas = false;
    boolean hayNegras = false;
    if((fila + (dimensionSimetria - 1)) < matriz.length &&
(columna + (dimensionSimetria - 1)) < matriz[0].length){
        validez=true;
    }
    if(fila>=0 && columna>=0 && validez){
        for (int f = fila; f < (fila + dimensionSimetria); f++)
```

```

{
    for (int c = columna; c < (columna +
dimensionSimetria); c++) {
        if(matriz[f][c].equals(" ")){
            validez = false;
        }
        if(matriz[f][c].equals("B")){
            hayBlancas = true;
        }
        if(matriz[f][c].equals("N")){
            hayNegras = true;
        }
    }
    return validez && hayBlancas && hayNegras;
}

//Método para verificar si se puede colocar una ficha en la
posición indicada
public boolean posicionValida (String[][] matriz, int fila, int
columna){
    boolean validez = false;
    try{
        if(matriz[fila][columna].equals(" ") &&
fila<matriz.length && columna<matriz[0].length)
            validez=true;
    }
    catch(Exception e){}
    return validez;
}

```

```
}
```

```
//Método para realizar la jugada limpieza (eliminar las fichas  
de una fila y columna dadas)  
public static void limpieza(Partida game, int fila, int  
columna){  
    String[][] copiaTablero =  
    game.copiaMatriz(game.getTable().getMatrizTablero());  
    if(game.isMovilEnTablero()) {  
        copiaTablero =  
        game.copiaMatriz(game.getTable().getTableroConPanel());  
        for(int i=0;i<game.getTable().getMatrizTablero().length;i++){  
            for(int j=0;j<game.getTable().getMatrizTablero()[0].length;j++){  
                if(i==fila-1){  
                    copiaTablero[i][j] = " ";  
                }  
                if(j==columna-1){  
                    copiaTablero[i][j] = " ";  
                }  
            }  
        }  
        game.getTable().setTableroConPanel(copiaTablero);  
    }  
    if(!game.isMovilEnTablero()){  
        for(int i=0;i<game.getTable().getMatrizTablero().length;i++){  
            for(int j=0;j<game.getTable().getMatrizTablero()[0].length;j++){  
                if(i==fila-1){  
                    copiaTablero[i][j] = " ";  
                }  
                if(j==columna-1){  
                    copiaTablero[i][j] = " ";  
                }  
            }  
        }  
        game.getTable().setTableroConPanel(copiaTablero);  
    }  
}
```

```
if(i==fila-1){  
    copiaTablero[i][j]=" ";  
}  
if(j==columna-1){  
    copiaTablero[i][j]=" ";  
}  
}  
}  
game.getTable().setMatrizTablero(copiaTablero);  
}  
}  
}
```

[Fin de Clase Tablero](#)

Clase Partida

```
// Obligatorio 2 - PROGRAMACIÓN 2
// Fernando Agustín Hernández Gobertti (173631) - Ingeniería en
Telecomunicaciones
// Sebastián Effa Gallego (193248) - Ingeniería en Electrónica

package Dominio;

import java.util.*;
import java.io.*;

public class Partida extends Observable implements Serializable{

    //Atributos de la clase Partida
    private boolean jugador1Vencedor, isJugador1, movilEnTablero,
partidaCargada;
    private int turno, filaPanel, columnaPanel, filaFicha,
columnaFicha, partida, cantFichasMax;
    private int[] datos = new int[3];
    private String opcion;
    private Tablero table = new Tablero();
    private Jugador jugador1, jugador2;
    private ArrayList <Jugador> listaJugadores;

    //Constructor Sin Parámetros de la Clase Partida
    public Partida() {
        turno = 1;
        filaPanel = 0;
```

```
        columnaPanel = 0;
        partida = 1;
        movilEnTablero = false;
        partidaCargada = false;
        opcion = " ";
        jugador1 = new Jugador();
        jugador2 = new Jugador();
    }

//Métodos de Acceso y Modificación de la clase Partida

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

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

public boolean isPartidaCargada() {
    return partidaCargada;
}

public void setPartidaCargada(boolean partidaCargada) {
    this.partidaCargada = partidaCargada;
}

public int getFilaFicha() {
    return filaFicha;
```

```
}
```

```
public int getColumnaFicha() {  
    return columnaFicha;  
}
```

```
public int getFilaPanel() {  
    return filaPanel;  
}
```

```
public int getColumnaPanel() {  
    return columnaPanel;  
}
```

```
public int getCantFichasMax() {  
    return cantFichasMax;  
}
```

```
public boolean isMovilEnTablero() {  
    return movilEnTablero;  
}
```

```
public String getOpcion() {  
    return opcion;  
}
```

```
public int[] getDatos() {  
    return datos;  
}
```

```
public Tablero getTable() {
    return table;
}

public int getTurno() {
    return turno;
}

public boolean isJugador1Vencedor() {
    return jugador1Vencedor;
}

public boolean isJugador1() {
    return isJugador1;
}

public Jugador getJugador1(){
    return jugador1;
}

public Jugador getJugador2(){
    return jugador2;
}

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

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

```
}
```

```
public void setFilaFicha(int filaFicha) {
```

```
    this.filaFicha = filaFicha;
```

```
}
```

```
public void setColumnaFicha(int columnaFicha) {
```

```
    this.columnaFicha = columnaFicha;
```

```
}
```

```
public void setFilaPanel(int filaPanel) {
```

```
    this.filaPanel = filaPanel;
```

```
}
```

```
public void setColumnaPanel(int columnapanel) {
```

```
    this.columnaPanel = columnapanel;
```

```
}
```

```
public void setMovilEnTablero(boolean movilEnTablero) {
```

```
    this.movilEnTablero = movilEnTablero;
```

```
}
```

```
public void setOpcion(String opcion) {
```

```
    this.opcion = opcion;
```

```
}
```

```
public void setDatos(int[] datos) {
```

```
    this.datos = datos;
```

```
}
```

```
public void setTurno(int turno) {  
    this.turno = turno;  
}  
  
public void setJugador1Vencedor(boolean isJugador1Vencedor) {  
    this.jugador1Vencedor = isJugador1Vencedor;  
}  
  
public void setIsJugador1(boolean isJugador1) {  
    this.isJugador1 = isJugador1;  
}  
  
public void setCantFichasMax(int cantFichasMax) {  
    this.cantFichasMax = cantFichasMax;  
}  
  
//Método para crear e inicializar una Matriz de Strings  
(apropiada para la partida)  
public static String[][] matrizNueva(int dimension) {  
    String[][] matriz = new String[dimension][dimension];  
    for (int fila = 0; fila < matriz.length; fila++) {  
        for (int columna = 0; columna < matriz[0].length; columna++) {  
            matriz[fila][columna] = " ";  
        }  
    }  
    return matriz;  
}  
  
//Método para copiar una matriz
```

```
public String[][] copiaMatriz(String[][] matriz) {  
    String[][] matrizCopia = new String[matriz.length][matriz[0].length];  
    for (int fila = 0; fila < matriz.length; fila++) {  
        for (int column = 0; column < matriz[0].length; column++) {  
            matrizCopia[fila][column] = matriz[fila][column];  
        }  
    }  
    return matrizCopia;  
}  
  
//Método para colocar el panel en el tablero  
public String[][] colocarPanel(String[][] panel, int fila, int column, String[][] tablero) {  
    String[][] copiaTablero = copiaMatriz(tablero);  
    if (((fila + panel.length) <= tablero.length) && ((column + panel[0].length) <= tablero[0].length)) {  
        for (int i = 0; i < panel.length; i++) {  
            for (int j = 0; j < panel[0].length; j++) {  
                copiaTablero[fila + i][column + j] = panel[i][j];  
            }  
        }  
    }  
    return copiaTablero;  
}  
  
//Método para comprobar si se puede colocar el panel en cierta  
posición
```

```
public boolean sePuedeColocarPanel(int dimension, int fila, int
columna, String[][] tablero) {
    return (((fila + dimension) <= tablero.length) && ((columna
+ dimension) <= tablero[0].length) && fila >= 0 && columna >= 0);
}

//Método que verifica y retorna la nueva posición del panel y
si éste se pudo mover
public int[] moverPanel(Partida game) {
    int filaNueva = 0;
    int columnaNueva = 0;
    String[][] tableroConPanel =
game.getTable().getTableroConPanel();
    int dimension = game.getTable().getPanel().length;
    int filaPanel = game.getFilaPanel();
    int columnaPanel = game.getColumnaPanel();
    String direccion = game.getOpcion().substring(1,2);
    int distancia =
Integer.parseInt(game.getOpcion().substring(2,3));
    int[] retorno = {0, 0, 0};
    if (direccion.equals("I")) {
        filaNueva = filaPanel;
        columnaNueva = (columnaPanel - distancia);
    }
    if (direccion.equals("D")) {
        filaNueva = filaPanel;
        columnaNueva = (columnaPanel + distancia);
    }
    if (direccion.equals("A")) {
        filaNueva = (filaPanel - distancia);
```

```

        columnaNueva = columnaPanel;
    }

    if (direccion.equals("B")) {
        filaNueva = (filaPanel + distancia);
        columnaNueva = columnaPanel;
    }

    if (sePuedeColocarPanel(dimension, filaNueva, columnaNueva,
    tableroConPanel)) {
        retorno[0] = 1;
        retorno[1] = filaNueva;
        retorno[2] = columnaNueva;
    }

    return retorno;
}

//Método para actualizar la matriz panel una vez se coloca o se
mueve sobre el tablero

public String[][] panelNuevo(String[][] tableroConPanel, int
filaPanel, int columnaPanel, int dimension) {
    String[][] panelNuevo = matrizNueva(dimension);
    for (int fila = filaPanel; fila < (filaPanel + dimension);
fila++) {
        for (int columna = columnaPanel; columna <
(columnaPanel + dimension); columna++) {
            panelNuevo[fila - filaPanel][columna -
columnaPanel] = tableroConPanel[fila][columna];
        }
    }
    return panelNuevo;
}

```

```
//Método para comprobar si una ficha se coloca sobre el panel  
(si no es así, la ficha también se coloca en el tablero sin panel)  
public boolean fichaSobrePanel(int filaFicha, int columnaFicha,  
Partida game) {  
    return (filaFicha >= game.getFilaPanel() && filaFicha <=  
game.getFilaPanel() + game.getTable().getPanel().length &&  
        columnaFicha >= game.getColumnaPanel() &&  
columnaFicha <= game.getColumnaPanel() +  
game.getTable().getPanel().length);  
}  
  
//Método para contar puntos por jugada  
public int contarPuntos(String[][] matriz, String criterio) {  
    int puntos = 0;  
    for (int fila = 0; fila < matriz.length; fila++) {  
        for (int columna = 0; columna < matriz[0].length;  
columna++) {  
            if (matriz[fila][columna].equals(criterio)) {  
                puntos++;  
            }  
        }  
    }  
    return puntos;  
}  
  
//Método para sustituir las fichas en una simetría  
public String[][] cambiarSimetriaTablero(Partida game,  
String[][] tablero, int fila, int columna, int dimensionSimetria, boolean  
tableroSinPanel) {
```

```

String[][] copiaTablero = copiaMatriz(tablero);
for (int i = fila; i < (fila + dimensionSimetria); i++) {
    for (int j = columna; j < (columna + dimensionSimetria); j++) {
        if (game.getTurno() % 2 != 0 && !tableroSinPanel){
            copiaTablero[i][j] = "B";
        }
        if (game.getTurno() % 2 == 0 && !tableroSinPanel){
            copiaTablero[i][j] = "N";
        }
        if(tableroSinPanel && !fichaSobrePanel(i, j, game)){
            if (game.getTurno() % 2 != 0){
                copiaTablero[i][j] = "B";
            }
            if (game.getTurno() % 2 == 0){
                copiaTablero[i][j] = "N";
            }
        }
    }
}
return copiaTablero;
}

```

//Método usado en el Observer, para actualizar el tablero desde otra ventana

```

public void observar(){
    setChanged();
    notifyObservers();
}

```

```
//Método para pasar un String a número entero
public int pasarANumero(String letra) {
    char l = letra.charAt(0);
    int num = l - 64;
    if (num < 0) {
        num = Integer.parseInt("") + 1;
    }
    return num;
}

//Método para enumerar las posiciones según los respectivos
puntajes de los Jugadores
//(luego de ordenar la lista por puntajes)
public ArrayList<Jugador> determinarPosicionesRanking(ArrayList<Jugador>
listaJugadores) {
    ArrayList<Jugador> jugadores = listaJugadores;
    for(int i=0;i<jugadores.size();i++){
        jugadores.get(i).setPosicionRanking(i+1);
    }
    return jugadores;
}

//Método que devuelve un ArrayList de Jugadores dado en forma
ordenada
public ArrayList<Jugador>
devolverJugadoresOrdenados(ArrayList<Jugador> listaJugadores) {
    Collections.sort(listaJugadores, new Comparator() {
        public int compare(Object param1, Object param2) {
            Jugador participante1 = (Jugador) param1;
```

```
Jugador participante2 = (Jugador) param2;
int aux = (participante2.getPartidasGanadas()) -
participante1.getPartidasGanadas();
if (aux == 0) {
    aux = (participante2.getPartidasJugadas()) -
participante1.getPartidasJugadas();
}
return aux;
});
return listaJugadores;
}

//Método que acomoda la lista de jugadores adecuadamente para
mostrar en el Ranking
public ArrayList setListaJugadoresRanking(Partida juego){
    ArrayList<Jugador> jugadores = new ArrayList<Jugador> ();
    jugadores = actualizarDatosJugadores(juego);
    jugadores = devolverJugadoresOrdenados(jugadores);
    jugadores = determinarPosicionesRanking(jugadores);
    return jugadores;
}

//Método para agregar un jugador a una lista haciendo
verificaciones correspondientes
public void agregarJugador(Partida game, Jugador jugador) {
    ArrayList<String> alias = new ArrayList<>();
    ArrayList<Jugador> listaJugadores =
game.getListaJugadores();
    if(!listaJugadores.isEmpty()){


```

```
for (int i = 0; i < listaJugadores.size(); i++) {
    alias.add(listaJugadores.get(i).getAlias());
}
int pos = alias.indexOf(jugador.getAlias());
if (pos == -1) {
    listaJugadores.add(jugador);
} else {
    listaJugadores.get(pos).setNombre(jugador.getNombre());
    listaJugadores.get(pos).setEdad(jugador.getEdad());
}
game.setListaJugadores(listaJugadores);
}

//Método para actualizar los datos de un jugador
//(el caso de que este no se encuentre en la lista no debería
verificarse pero se hace por precaución)
public ArrayList actualizarDatosJugadores(Partida juego) {
    ArrayList<String> alias = new ArrayList<>();
    ArrayList<Jugador> listaJugadores =
juego.getListajugadores();
    Iterator<Jugador> it = listaJugadores.iterator();
    while(it.hasNext()){
        Jugador player = it.next();
        alias.add(player.getAlias());
    }
    int pos = alias.indexOf(juego.getJugador1().getAlias());
    if (pos == -1) {
        listaJugadores.add(juego.getJugador1());
    } else {
```

```
        listaJugadores.set(pos, juego.getJugador1());
    }
    pos = alias.indexOf(juego.getJugador2().getAlias());
    if (pos == -1) {
        listaJugadores.add(juego.getJugador2());
    } else {
        listaJugadores.set(pos, juego.getJugador2());
    }
    return listaJugadores;
}

//Método para contar puntos y determinar el ganador al
finalizar una partida

public void finPartida(Partida game) {
    int             puntosJ1Tablero      =
game.contarPuntos(game.getTable().getMatrizTablero(), "B");
    int             puntosJ2Tablero      =
game.contarPuntos(game.getTable().getMatrizTablero(), "N");
    int             puntosJ1Panel       =
game.contarPuntos(game.getTable().getPanel(), "B");
    int             puntosJ2Panel       =
game.contarPuntos(game.getTable().getPanel(), "N");
    if (game.isMovilEnTablero()) {
        if (puntosJ1Tablero + puntosJ1Panel > puntosJ2Tablero +
puntosJ2Panel) {
            game.setJugador1Vencedor(true);
        } else {
            game.setJugador1Vencedor(false);
        }
    }
}
```

```
if (!game.isMovilEnTablero()) {  
    if (puntosJ1Tablero > puntosJ2Tablero) {  
        game.setJugador1Vencedor(true);  
    } else {  
        game.setJugador1Vencedor(false);  
    }  
}  
}  
}
```

[Fin de Clase Partida](#)