Universidad Nacional de General Sarmiento

Licenciatura en Sistemas
Tecnicatura Superior en Informática

Trabajo Práctico
Ta-Te-Toro

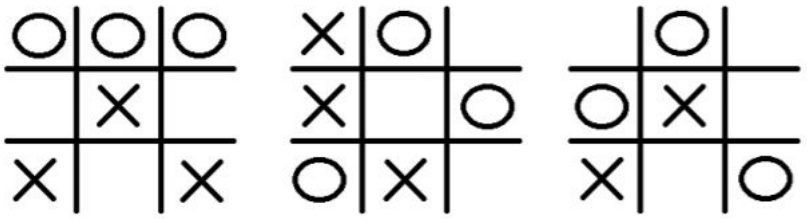
Programación 3
(2º semestre 2020)

Integrantes:
Sanchez Juan Ignacio - 39160465 - juani.scz95@gmail.com
Schmidt Maximiliano - 40664272 - maxischmidt2011@gmail.com
Sosa Martín Leonel - 36289472 - martinleonelarce@gmail.com

Consignas

El objetivo del trabajo práctico es implementar una aplicación para que dos usuarios jueguen al “ta-te-ti toroidal”. Este juego es similar al “ta-te-ti clásico” sobre un tablero de 3 × 3, pero se considera que las celdas están conectadas de **manera toroidal**. Es decir, un jugador gana si ubica tres piezas propias en tres celdas seguidas (en una misma fila, columna o diagonal), y se considera que una diagonal continua del otro lado del tablero.

Las siguientes posiciones son ganadoras para el jugador que juega con los círculos:



La aplicación debe proporcionar un mecanismo adecuado para que los jugadores especifiquen sus jugadas por turnos. Además, debe detectar cuando alguno de los jugadores ha ganado e informar en la interfaz.

Como objetivos opcionales no obligatorios, se pueden contemplar los siguientes elementos:

1. Contar cuantos turnos se han jugado y mostrar este dato en la interfaz.

2. Pedir los nombres (y las fotos) de los usuarios y mostrar sus nombres en pantalla y en los carteles que informan el resultado del juego.

3. Detectar que el juego ya terminará en un empate antes de que estén completos todos los casilleros, e informar a los usuarios.

Condiciones de entrega: El trabajo práctico se debe entregar por mail a los docentes de la materia. Además del código, se debe incluir un documento en el que se describa la implementación y se detallen las decisiones tomadas durante el desarrollo. El código responsable del funcionamiento de la aplicación debe estar claramente separado del código responsable de la interfaz. El trabajo práctico se puede hacer en grupos de hasta tres personas.

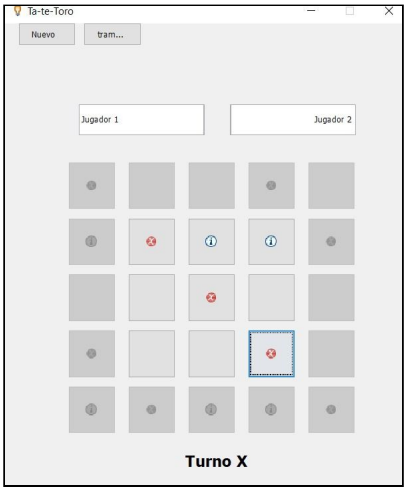
Resumen

En el presente trabajo se presenta el desarrollo de un juego en Java, utilizando una nueva tecnología llamada WindowsBuilder, previamente presentada por los docentes en clases.

El juego “Te-te-toro” consiste en un programa en donde dos jugadores mantienen una partida de “ta-te-ti ordinario” con reglas adicionales. Los jugadores deben seleccionar, utilizando el mouse, el casillero en donde quieran dejar marcada su jugada, posteriormente decidirá su jugada el jugador oponente hasta que haya un ganador. el juego cesará cuando uno de los dos jugadores gane.

En este apartado iremos elaborando los percances y aciertos a la hora de afrontar el trabajo práctico desde sus inicios.

Una puesta en marcha



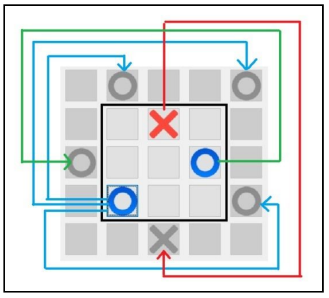
Al comenzar con el proyecto en un principio decidimos crear dos packages, uno “Interfaz” donde se hallarían las ventanas y sus anexos relacionados con la interfaz gráfica del juego, y otro “classes” donde se guardarían las clases utilizadas para representar los objetos complejos visibles en la interfaz gráfica, y clases adicionales (como enums).

Dentro del package “interfaz” creamos una Windows Application llamada “Ventana” la cual en un principio iba a ser la única interfaz gráfica. Al poco tiempo nos dimos cuenta de que la clase comenzaba a alojar demasiado código, por lo que decidimos separar la zona del tablero en un JPanel independiente.

Más adelante decidiremos ir por un camino más modular y separar la interfaz gráfica en cuatro clases: un JFrame Ventana y tres JPanel Botonera, Jugadores y Tablero. Ventana ahora se encargará de contener y comunicar a los paneles entre sí, mientras que estos serán los que se encargarán de la interacción con el usuario.

Por otro lado, luego también crearemos un package independiente para contener los enums, y otro para los controladores (según consideramos necesario incluirlos) aprovechando de esta forma el patrón de arquitectura MVC

Consideraciones personales



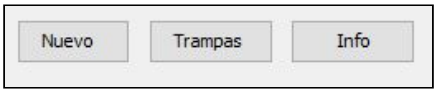
Desde un principio nos pareció buena idea implementar botones extra (desactivados) que representaran gráficamente las jugadas toroidales, para que sea más fácil observar el funcionamiento y comprender el sentido del juego que debíamos programar. A estos botones decidimos llamarlos “botones ghost” (btnGhost).

Implementamos el sistema de turnos y realizamos varias pruebas para ver el correcto funcionamiento del tablero, fue en ese momento que descubrimos que no era posible empatar y por consiguiente que el ítem 3 de los objetivos opcionales de la consigna (“Detectar que el juego ya terminará en un empate antes de que estén completos todos los casilleros, e informar a los usuarios.”) era imposible de realizar

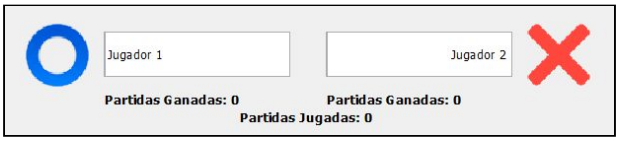
Avanzando con el juego

Una vez tuvimos todos los elementos capitales con los cuales el juego ya podía correr, comenzamos las primeras pruebas para encontrar errores importantes que hicieran fallar el programa, no cumplieran con las consignas dadas en el trabajo práctico o que atente con la separación del código con la interfaz. Como consecuencia de esto, implementamos varios cambios y desarticulamos Ventana en varias partes que después fueron exportadas a la ventana principal. Los elementos separados fueron dejados en el package Interfaz con los nombres de:

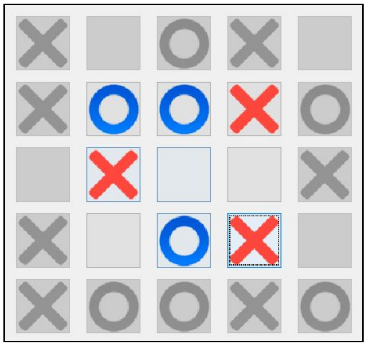
"Botonera" que es una barra horizontal de botones: "nuevo" con la opción de reiniciar la partida, "trampas" donde después pensábamos agregar diferentes mods o hacks para profundizar aún más el proyecto y el botón "info" donde depositamos toda la información competente acerca del proyecto, su versión y demás menesteres.



"Jugadores" donde se dejó dos JTextFields en donde ambos jugadores podrían escribir sus nombres y a sus lados correspondientes los iconos indicativos mostrando que tipo de marca tendrían al interactuar con el tablero.



"Tablero" que contiene los botones seleccionables, los botones "ghost" y también una JLabel que indica el turno de los jugadores.



Y como nexo de todas estas partes de la interfaz gráfica dejamos **"Ventana"**, aquí todas las partes funcionan sin estar ensimismadas en una sola Window Application, mejorando así la lectura, y generando un "encapsulamiento" de la parte gráfica.

Luego de realizar todos estos cambios, también decidimos cambiar la implementación de nuestro package "Classes" para que estuviera en mayor concordancia con la idea antes presentada, creando un nuevo package "enums" con el fin de mejorar la comprension del codigo y ubicar mas eficientemente las clases del proyecto

Al notar que el proyecto se adecuaba sin problemas a los cambios y funcionaba sin mayores inconvenientes, decidimos que ya era momento de incluir nuestras propias imágenes y sprites con el fin de mejorar la parte estética del programa (hasta ese momento habíamos estado utilizando iconos por defecto de eclipse). A estas imágenes las depositamos en una folder denominada **"Recursos/Imagenes"** por fuera de las Interfaces ya creadas.

La logica detras del funcionamiento

A la hora de implementar la lógica, intentamos realizar una implementación de grafos, basado en el funcionamiento de las listas enlazadas, con algunas funcionalidades extra que nos serían útiles a la hora de recorrer el tablero en busca de ganadores.

El tablero está relacionado con una instancia de tablero virtual de tal forma que cada botón gráfico posee su representación lógica en un botón virtual (BtnVirtutal). Con esto logramos que, luego de construir el tablero virtual y enlazar los botones gráficos con los botones virtuales, cada botón conozca no solo sus vecinos tal y como funcionaria en caso de un grafo, sino que además sepa la orientación cardinal de estos. Permitiéndonos desplazarnos entre botones al llamar a los métodos de instancia de los botones virtuales.

Esto nos facilitó enormemente el armado de la función que busca si hay un ganador cada vez que hay un cambio de turno

Con el fin de mantener una numeración coherente y más fácil de comprender, decidimos crear los arrays con n+1 unidades, dejando el elemento (0) sin uso. Esto se dio por conveniencia, al resultarnos a todos los integrantes del grupo más cómodo comparar los botones del juego con el de un teclado numérico de un teléfono (T9). Consideramos el uso extra de recursos justificable al compararlo con las ventajas que obtendríamos a la hora de realizar el proyecto

De esta forma, un botón perteneciente a la interfaz gráfica está relacionado con el tablero virtual mediante un número asignado arbitrariamente por el programador (la numeración fue establecida buscando representar el estilo numérico de un teléfono T9 mencionado previamente)

A medida que fuimos cumpliendo de forma estricta las condiciones del trabajo práctico decidimos tomarnos ciertas libertades para mejorar y agregar elementos que potenciarán la jugabilidad y la interfaz gráfica a nuestro proyecto. Para este cometido decidimos separar aún más el código e implementar la clase **“TableroContoller”** quien ahora se encargaría de realizar toda la lógica referente al tablero, que hasta ese entonces se encontraba en el JPanel Tablero. De esta forma logramos encapsular el funcionamiento del tablero, dado que ahora los demás componentes que se comunican con el desconocen la existencia del controlador (hablan con tablero para solicitarle una función y tablero la deriva a su controlador). De esa forma queda separada la vista de la lógica, la clase se hace más pequeña y ordenada

Etapa Final: Ultimando Detalles

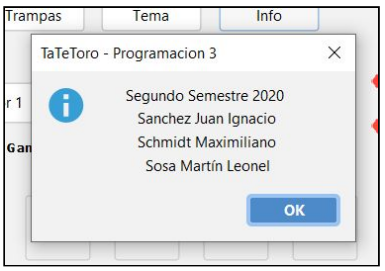
Luego de llegar a una estructura de proyecto que considerábamos adecuada nos centramos en privatizar métodos que no era necesario que permanezcan públicos, crear getters y setters donde fuera necesario y crear métodos adicionales descriptivos con el fin de mejorar la lectura del código y segmentar aún más la realización de procesos. Este cambio se ve principalmente en la clase TableroController, donde incluso hay funciones que solo se encargan de llamar a otro grupo de funciones, reduciendo de esta forma la complejidad de la función principal y abstrayendo la lectura de los procesos que son realizados a la de un título más descriptivo.

Finalmente, para mejorar la parte estética del proyecto, decidimos cambiar las imágenes del círculo y la equis por otras que guardarán mayor correlación entre sí, manteniendo la misma estructura de color sólido (sin degrede), al mismo tiempo creamos dos imagenes más que se utilizarían para marcar la jugada ganadora

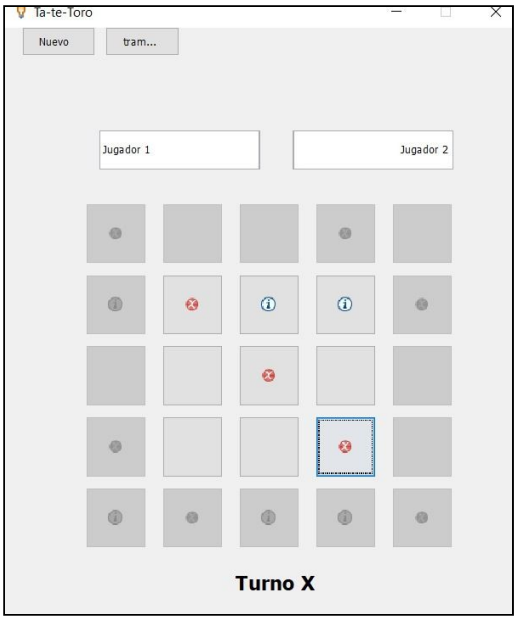
Una función que nos resultó interesante y útil fue la de mostrar antes de clickear un botón, como quedaría la jugada. Esto es sumamente útil cuando se visualizan los botones ghost dado que permiten al usuario comprender mejor el funcionamiento del tablero y planear estrategias con mayor facilidad al ver el impacto visual de su jugada antes de realizarla..

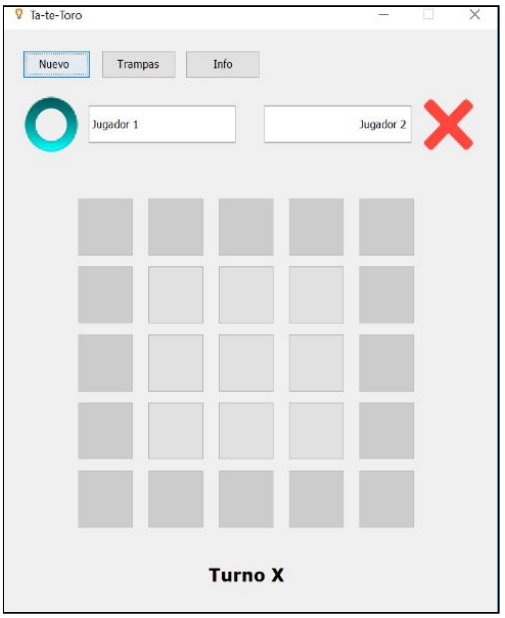
Un extra que decidimos incorporar fue que el botón **“Trampas”** de la Botonera, activará o cancelará la visualización de los botones especiales (btnGhost) toroidales con el fin de una mejor vista de las jugadas hechas por los jugadores.

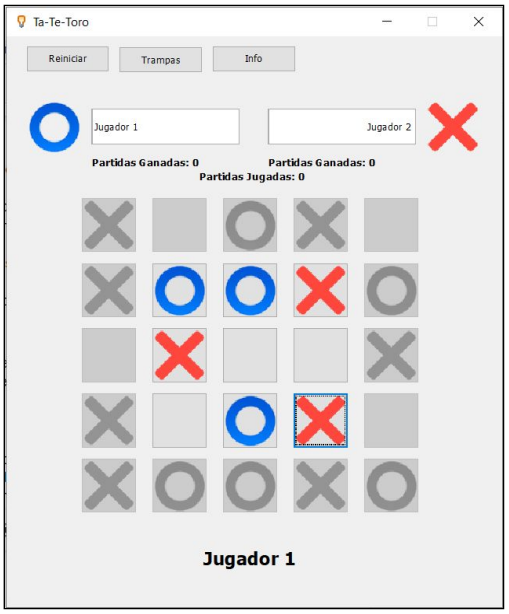
Otra extra fue el botón **“Info”** de Botonera, el cual al clickearse revela información básica de trabajo práctico, como nombre del proyecto, materia e integrantes del proyecto.

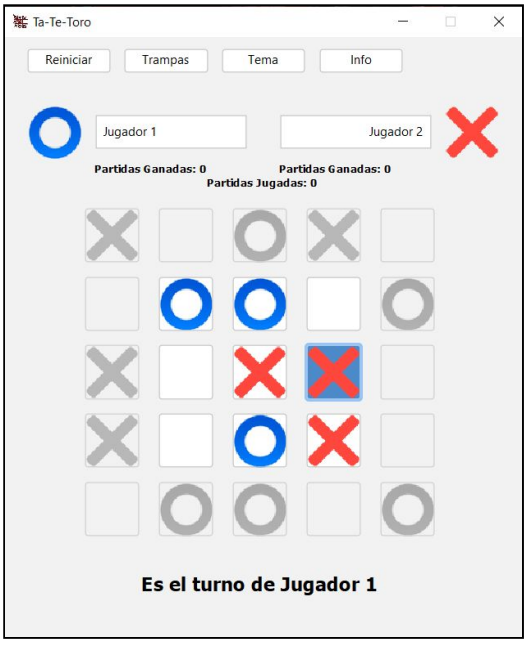


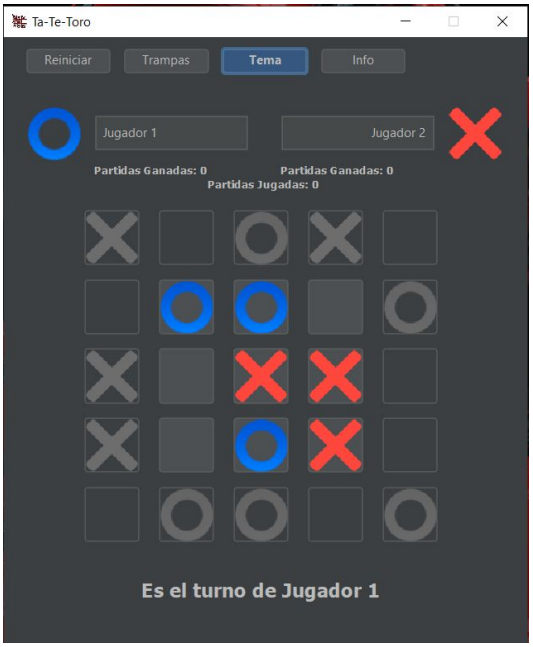
Por último nos pareció oportuno incluir la posibilidad al usuario de alternar la colorización de la interfaz gráfica según sus preferencias. Esto es posible realizarlo a traves del boton **“Tema”** de la botonera, que permite cambiar entre un tema claro y oscuro del Look And Feel FlatLaf, que es open-source (https://www.formdev.com/flatlaf/)


Ta-te-Toro
Nuevo
tram...
Jugador 1
Jugador 2
Turno X


Ta-te-Toro
Nuevo
Trampas
Info
Jugador 1
Jugador 2
Turno X


Ta-Te-Toro
Reiniciar
Trampas
Info
Jugador 1
Jugador 2
Partidas Ganadas: 0
Partidas Ganadas: 0
Partidas Jugadas: 0
Jugador 1

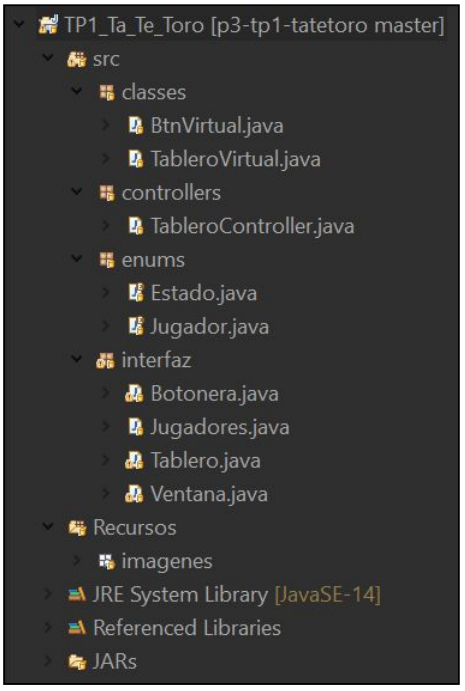

Ta-Te-Toro
Reiniciar
Trampas
Tema
Info
Jugador 1
Jugador 2
Partidas Ganadas: 0
Partidas Ganadas: 0
Partidas Jugadas: 0
Es el turno de Jugador 1


Ta-Te-Toro
Reiniciar
Trampas
Tema
Info
Jugador 1
Jugador 2
Partidas Ganadas: 0
Partidas Ganadas: 0
Partidas Jugadas: 0
Es el turno de Jugador 1

Interfaces y denominación de clases

Para la implementación de la aplicación de decidió separar en cuatro (4) interfaces:

1. classes
2. controllers (capa lógica)
3. enums
4. Interfaz (capa visual)



CLASSES

BtnVirtual
private Estado estado;
private int nBoton;
private BtnVirtual arriba;
private BtnVirtual abajo;
private BtnVirtual derecha;
private BtnVirtual izquierda;

TableroVirtual
BtnVirtual[] btnArray;
int[] botonesConsecutivos;

CONTROLLERS

TableroController
boolean hayGanador;
TableroVirtual tableroVirtual;
Jugador turno;
JButton[] botones;
JButton[] botonesEspeciales;
Jugadores jugadores;

ENUMS

Estado {LIBRE,JUGADOR1,JUGADOR2}

Jugador {JUGADOR1,JUGADOR2}

INTERFAZ (capa visual)

Botonera
Jugadores
Tablero
Ventana

Apéndice: Métodos de ejemplo

TableroController

```
public void sejuega(int nBtn, JButton Jbtn, JButton JbtnGhost, JButton JbtnGhost_2, JButton
JbtnGhost_diag) {
		if (!botonEstaOcupado(nBtn) && !hayGanador()) {
			if (esTurnoJugador1()) {
				setIcons(Jbtn, JbtnGhost, JbtnGhost_2, JbtnGhost_diag,
"/imagenes/circulo.png");
				tableroVirtual.setEstadoBtn(nBtn, Estado.JUGADOR1);
				actualizarCartelTurno(jugadores.getJugador2name());
				setTurno(Jugador.JUGADOR2);

			}
			else {
				setIcons(Jbtn, JbtnGhost, JbtnGhost_2, JbtnGhost_diag,
"/imagenes/equis.png");
				tableroVirtual.setEstadoBtn(nBtn, Estado.JUGADOR2);
				actualizarCartelTurno(jugadores.getJugador1name());
				setTurno(Jugador.JUGADOR1);
			}
			evaluarSiHayGanador(nBtn);
		}
	}

private void evaluarSiHayGanador(int nBtn) {
	if (tableroVirtual.hay3Consecutivos(nBtn)) {
		hayGanador(true);
		marcarLineaGanadora(tableroVirtual.getBotonesConsecutivos(),nBtn);
		mostrarAlertaGanador(nBtn);
		mostrarConsultaContinuarJuego();
	}
}


private void setIcons(JButton Jbtn, JButton JbtnGhost, JButton JbtnGhost_2, JButton JbtnGhost_diag,
String directorio) {
	Jbtn.setIcon(new ImageIcon(Tablero.class.getResource(directorio)));
	if (JbtnGhost != null)
		JbtnGhost.setIcon(new ImageIcon(Tablero.class.getResource(directorio)));
	if (JbtnGhost_2 != null)
		JbtnGhost_2.setIcon(new ImageIcon(Tablero.class.getResource(directorio)));
	if (JbtnGhost_diag != null)
		JbtnGhost_diag.setIcon(new ImageIcon(Tablero.class.getResource(directorio)));
}
```

Tablero

```
public void reiniciar() {
        tableroController.limpiarTablero();
        tableroController.reiniciarJuego();
}
```

Ejemplo de funcionamiento de un botón

```
JButton btn1 = new JButton("");
btn1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                tableroController.sejuega(1, btn1, btn1ghost, btn1ghost_2, btn1diag);
                //tableroController.actualizarCartelTurno(lbCartelInfo);
        }
});
btn1.setBounds(82, 77, 57, 57);
add(btn1);
botones[1] = btn1;
```

TableroVirtual

```
public boolean hay3Consecutivos(int nBtn) { // uso operador del tipo short-circuiting para no evaluar las condiciones restantes en cuanto una resulta verdadera
        BtnVirtual btn = btnArray[nBtn];
        return compararBotones(btn, btn.derecha(), btn.derecha().derecha()) ||
                compararBotones(btn, btn.abajo(), btn.abajo().abajo()) ||
                compararBotones(btn, btn.abajoDerecha(), btn.abajoDerecha().abajoDerecha()) ||
                compararBotones(btn, btn.abajoIzquierda(), btn.abajoIzquierda().abajoIzquierda());
}

private boolean compararBotones(BtnVirtual btn1, BtnVirtual btn2, BtnVirtual btn3) {
        if(btn1.estado() == btn2.estado() && btn1.estado() == btn3.estado()) { //la igualdad es transitiva
                botonesConsecutivos[0] = btn1.nBoton();
                botonesConsecutivos[1] = btn2.nBoton();
                botonesConsecutivos[2] = btn3.nBoton();
                return true;
        }
        return false;
}
```