Documento de Requerimiento del Juego
Tetris

Programación Orientada a Objetos

Docente

Julio Cortes Trujillo

Ingeniería Electrónica

Juan Pablo Casella Galindo

20222005110

005-9

- Historia del juego **Tetris**

La historia del juego Tetris es una de las más fascinantes y curiosas del mundo de los videojuegos. Tetris fue creado en 1984 por el matemático ruso Alekséi Pázhitnov, que trabajaba en el centro de computación de la Academia de Ciencias de Moscú, en plena época comunista. Pázhitnov se inspiró en un juego tradicional llamado Pentominó, que consistía en encajar piezas geométricas formadas por cinco cuadrados en una caja de madera. Sin embargo, Pázhitnov simplificó las piezas a figuras de cuatro cuadrados, llamadas Tetrominos, y las hizo caer en una pantalla vertical, donde el jugador debía rotarlas y desplazarlas para formar líneas horizontales sin huecos. El nombre del juego proviene de la combinación de "tetra", que significa cuatro en griego, y el deporte favorito de su creador, el tenis.

Tetris fue programado en un ordenador Elektronika 60, que no tenía capacidades gráficas, por lo que Pázhitnov tuvo que usar caracteres alfanuméricos para representar las piezas. El juego se hizo muy popular entre los empleados del centro de computación y pronto se extendió por otras instituciones soviéticas. En 1986, el programador Vadim Gerásimov adaptó el juego para PC y lo distribuyó a través de redes informáticas. Así fue como Tetris llegó a manos de Robert Stein, un empresario británico que se interesó por el juego y contactó con Pázhitnov para negociar los derechos de distribución fuera de la URSS.

Sin embargo, la situación legal del juego era muy compleja, ya que los derechos pertenecían al Estado soviético y no al autor. Stein tuvo que lidiar con varias entidades burocráticas para obtener una licencia, mientras que otras compañías occidentales también se disputaban los derechos de Tetris para diferentes plataformas. Entre ellas estaba Nintendo, que consiguió la licencia exclusiva para consolas y lanzó el juego para NES y Game Boy en 1989. Esta última versión fue la que catapultó el éxito mundial de Tetris, convirtiéndose en uno de los juegos más vendidos y populares de la historia.

Tetris ha sido adaptado a numerosas plataformas y sistemas operativos, con diferentes variantes y modos de juego. También ha sido objeto de estudios científicos sobre sus efectos cognitivos y matemáticos, así como de obras artísticas y culturales. Tetris es considerado un clásico indiscutible y un icono de los videojuegos, que ha trascendido las barreras políticas, geográficas y generacionales.

El juego Tetris es un ejemplo de un juego de lógica y habilidad que consiste en encajar piezas geométricas de diferentes formas y colores en una cuadrícula bidimensional. El objetivo es completar filas horizontales sin dejar huecos, lo que hace que las piezas desaparezcan y se libere espacio. El juego se vuelve más difícil a medida que aumenta la velocidad de caída de las piezas y se acumulan en la parte inferior de la pantalla. El juego termina cuando las piezas llegan al borde superior de la cuadrícula y no hay más espacio para colocarlas.

1. **Análisis**

1.1 **Análisis Conceptual**

¿Qué es un numero?

Un número en programación es un tipo de dato que representa una cantidad numérica. Los números se pueden clasificar en diferentes categorías según su naturaleza y el rango de valores que pueden almacenar. Algunas de las categorías más comunes son:

- Enteros: Son números que no tienen parte decimal y pueden ser positivos, negativos o cero. Por ejemplo, 5, -3, 0 son enteros.

- Reales: Son números que pueden tener parte decimal y pueden ser positivos, negativos o cero. Por ejemplo, 3.14, -0.5, 0.0 son reales.

- Complejos: Son números que tienen una parte real y una parte imaginaria, representada por la letra i. Por ejemplo, 2 + 3i, -1 - i, 0 + 0i son complejos.

- Racionales: Son números que se pueden expresar como el cociente de dos enteros. Por ejemplo, 1/2, -3/4, 0/1 son racionales.

- Irracionales: Son números que no se pueden expresar como el cociente de dos enteros. Por ejemplo, $\sqrt{2}$, π, e son irracionales.

Cada lenguaje de programación tiene sus propias reglas y formas de definir y manipular los números. Es importante conocer las características y limitaciones de cada tipo de número para evitar errores o pérdidas de precisión al realizar operaciones matemáticas.

¿Qué es una suma, resta división y multiplicación?

Una suma, resta, división y multiplicación en programación son operaciones aritméticas que se realizan con operadores específicos sobre valores numéricos. Estos operadores son los siguientes:

- Suma: (+) Este operador se utiliza para sumar dos valores. Por ejemplo: 5 + 3 = 8

- Resta: (–) Este operador se utiliza para restar dos valores. Por ejemplo: 10 - 7 = 3

- Multiplicación: (*) Este operador se utiliza para multiplicar dos valores. Por ejemplo: 4 * 6 = 24

- División: (/) Este operador se utiliza para dividir dos valores. Por ejemplo: 12 / 4 = 3

Estas operaciones se pueden realizar con diferentes tipos de datos numéricos, como enteros, decimales o fraccionarios. También se pueden combinar entre sí siguiendo las reglas de prioridad y asociatividad de

las operaciones matemáticas. En algunos casos, se pueden utilizar paréntesis para modificar el orden de las operaciones.

Estas operaciones son muy útiles en programación para realizar cálculos, asignar valores a variables o comparar expresiones. Dependiendo del lenguaje de programación que se utilice, la sintaxis y los resultados pueden variar ligeramente.

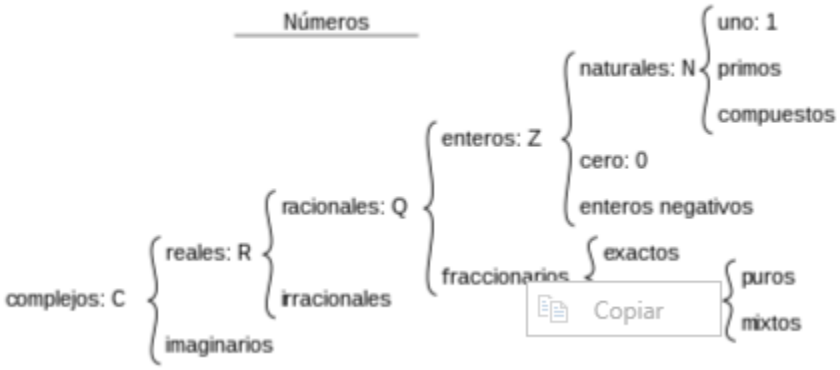


Imagen1. Tipos de números

¿Qué es un numero en programación?

Un número en programación es un tipo de dato que representa una cantidad numérica. Los números se pueden clasificar en diferentes categorías según su naturaleza y el rango de valores que pueden almacenar. Algunas de las categorías más comunes son:

- Enteros: Son números que no tienen parte decimal y pueden ser positivos, negativos o cero. Por ejemplo, 5, -3, 0 son enteros. En Java, se pueden definir enteros usando la siguiente sintaxis:

java
int a = 5; // a es un entero positivo
int b = -3; // b es un entero negativo
int c = 0; // c es el entero cero

- Reales: Son números que pueden tener parte decimal y pueden ser positivos, negativos o cero. Por ejemplo, 3.14, -0.5, 0.0 son reales. En Java, se pueden definir reales usando la siguiente sintaxis:

java
double d = 3.14; // d es un real positivo
double e = -0.5; // e es un real negativo
double f = 0.0; // f es el real cero

- Complejos: Son números que tienen una parte real y una parte imaginaria, representada por la letra i. Por ejemplo, 2 + 3i, -1 - i, 0 + 0i son complejos. En Java, no hay un tipo de dato nativo para los complejos, pero se puede usar la clase Complex de la librería Apache Commons Math:

java
```
import org.apache.commons.math3.complex.Complex;

Complex g = new Complex(2,3); // g es un complejo con parte real 2 y parte imaginaria 3
Complex h = new Complex(-1,-1); // h es un complejo con parte real -1 y parte imaginaria -1
Complex i = Complex.ZERO; // i es el complejo cero
```

- Racionales: Son números que se pueden expresar como el cociente de dos enteros. Por ejemplo, 1/2, -3/4, 0/1 son racionales. En Java, no hay un tipo de dato nativo para los racionales, pero se puede usar la clase Fraction de la librería Apache Commons Math:

java
```
import org.apache.commons.math3.fraction.Fraction;

Fraction j = new Fraction(1,2); // j es un racional equivalente a 1/2
Fraction k = new Fraction(-3,4); // k es un racional equivalente a -3/4
Fraction l = Fraction.ZERO; // l es el racional cero
```

- Irracionales: Son números que no se pueden expresar como el cociente de dos enteros. Por ejemplo, $\sqrt{2}$, π, e son irracionales. En Java, se puede usar la clase Math para acceder a algunas constantes irracionales:

java
```
double m = Math.sqrt(2); // m es un irracional aproximado a la raíz cuadrada de 2
double n = Math.PI; // n es un irracional aproximado al número pi
double o = Math.E; // o es un irracional aproximado al número e
```

Cada lenguaje de programación tiene sus propias reglas y formas de definir y manipular los números. Es importante conocer las características y limitaciones de cada tipo de número para evitar errores o pérdidas de precisión al realizar operaciones matemáticas.

¿Qué es una interfaz gráfica en java?

Una interfaz gráfica en java es un conjunto de componentes visuales que permiten al usuario interactuar con una aplicación o un programa. Los componentes de una interfaz gráfica pueden ser botones, menús, etiquetas, campos de texto, imágenes, etc. Cada componente tiene una función específica y un aspecto determinado por el estilo de la interfaz.

Para crear una interfaz gráfica en java se puede utilizar la biblioteca Swing, que proporciona una serie de clases e interfaces para diseñar y manipular los componentes gráficos. La clase JFrame es la encargada de crear la ventana principal de la interfaz, donde se pueden añadir otros componentes como paneles, botones, etc. La clase JPanel es un contenedor que permite agrupar y organizar los componentes dentro de la ventana. La clase JButton es un componente que representa un botón que puede ejecutar una acción al ser pulsado. La clase JLabel es un componente que muestra un texto o una imagen. La clase JTextField es un componente que permite al usuario introducir texto. La clase ImageIcon es una clase que permite crear y mostrar imágenes.

Para añadir funcionalidad a los componentes gráficos se pueden utilizar los eventos y los oyentes. Un evento es una acción que ocurre en la interfaz gráfica, como por ejemplo el clic de un botón, el movimiento del ratón o la pulsación de una tecla. Un oyente es un objeto que se encarga de escuchar y responder a los eventos que le interesan. Para asociar un oyente a un componente se utiliza el método addXXXListener, donde XXX es el tipo de evento que se quiere escuchar. Por ejemplo, para asociar un oyente al clic de un botón se utiliza el método addActionListener. El oyente debe implementar la interfaz correspondiente al tipo de evento, como por ejemplo ActionListener para los eventos de acción. El oyente debe definir el método actionPerformed, que se ejecuta cuando ocurre el evento.

A continuación, se muestra un ejemplo de cómo crear una interfaz gráfica con Swing que contiene un botón y una etiqueta. Al pulsar el botón se cambia el texto de la etiqueta.

```
java
import javax.swing.*;
import java.awt.event.*;

public class EjemploInterfaz extends JFrame implements ActionListener {

// Creamos los componentes
private JButton boton;
private JLabel etiqueta;

// Constructor de la clase
public EjemploInterfaz() {
// Inicializamos los componentes
boton = new JButton("Pulsa aquí");
etiqueta = new JLabel("Hola");

// Añadimos el oyente al botón
boton.addActionListener(this);

// Creamos un panel para colocar los componentes
JPanel panel = new JPanel();
```

```
// Añadimos los componentes al panel
panel.add(boton);
panel.add(etiqueta);

// Añadimos el panel a la ventana
this.add(panel);

// Configuramos la ventana
this.setTitle("Ejemplo de interfaz");
this.setSize(300, 100);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setVisible(true);
}

// Método que se ejecuta cuando se pulsa el botón
public void actionPerformed(ActionEvent e) {
// Cambiamos el texto de la etiqueta
etiqueta.setText("Adiós");
}

// Método principal para ejecutar la aplicación
public static void main(String[] args) {
// Creamos una instancia de la clase
EjemploInterfaz ei = new EjemploInterfaz();
}
}
```

¿Qué es una clase estática?

Una clase estática en java es una clase que no tiene instancias y que solo puede acceder a los miembros estáticos de su propia clase o de otras clases. Una clase estática se declara con la palabra clave static dentro de otra clase, y se conoce como una clase anidada estática. Una clase estática puede tener métodos, campos, constructores y bloques de inicialización estáticos, pero no puede tener miembros no estáticos. Una clase estática puede acceder a los miembros estáticos de la clase que la contiene, pero no a los miembros no estáticos. Una clase estática puede ser usada para agrupar constantes o métodos que no dependen de una instancia de la clase que la contiene. Por ejemplo:

```
public class Calculadora {
  // Clase anidada estática
  public static class Operaciones {
    // Constante estática
    public static final double PI = 3.1416;
    // Método estático
    public static double suma(double a, double b) {
      return a + b;
    }
```

```
    // Constructor estático
    private Operaciones() {
      // No se puede instanciar
    }
  }
}

// Para usar la clase estática Operaciones
double resultado = Calculadora.Operaciones.suma(2, 3);
System.out.println(resultado); // Imprime 5.0
System.out.println(Calculadora.Operaciones.PI); // Imprime 3.1416
```

¿Qué es polimorfismo en Java?

El polimorfismo es una propiedad de la programación orientada a objetos que permite que un objeto pueda tener diferentes formas o comportamientos según el contexto. En Java, el polimorfismo se manifiesta de dos formas principales: la sobrecarga y la sobreescritura de métodos.

La sobrecarga de métodos ocurre cuando se definen dos o más métodos con el mismo nombre, pero con diferentes parámetros en una clase. Por ejemplo, se puede tener un método llamado sumar que reciba dos números enteros y otro método llamado sumar que reciba dos números reales. El compilador determina cuál método invocar según el tipo de los argumentos que se pasen al llamar al método.

```
// Ejemplo de sobrecarga de métodos
public class Calculadora {

  // Método sumar que recibe dos enteros
  public int sumar(int a, int b) {
    return a + b;
  }

  // Método sumar que recibe dos reales
  public double sumar(double a, double b) {
    return a + b;
  }
}
```

La sobreescritura de métodos ocurre cuando se redefine un método heredado de una clase padre en una clase hija. Por ejemplo, se puede tener una clase Animal que tenga un método abstracto llamado hacerSonido y dos clases hijas llamadas Gato y Perro que implementen ese método de forma diferente. El polimorfismo permite que se pueda referenciar a un objeto de tipo Gato o Perro con una variable de tipo Animal y llamar al método hacerSonido, obteniendo el sonido correspondiente a cada subclase.

```
// Ejemplo de sobreescritura de métodos
public abstract class Animal {

  // Método abstracto hacerSonido
  public abstract void hacerSonido();
}

public class Gato extends Animal {

  // Método hacerSonido redefinido
  public void hacerSonido() {
    System.out.println("Miau");
  }
}

public class Perro extends Animal {

  // Método hacerSonido redefinido
  public void hacerSonido() {
    System.out.println("Guau");
  }
}
```

¿Qué es un archivo plano en java?

Un archivo plano en java es un tipo de archivo que contiene datos sin ningún tipo de estructura o formato. Los datos se almacenan como una secuencia de caracteres, sin separadores ni etiquetas que indiquen su significado. Un archivo plano puede ser leído o escrito por cualquier programa que pueda manejar archivos de texto, pero no ofrece ninguna ventaja en términos de seguridad, eficiencia o facilidad de uso. Un ejemplo de archivo plano en java es un archivo .txt que contiene el código fuente de una clase o un método.

1.2 **Análisis Programación**

Algunos de los posibles problemas de código al crear un juego como Tetris pueden incluir la comprensión de la lógica para las colisiones, rotaciones, eliminación de filas, movimiento de piezas y límites

Cliente	Docente
Usuario	Cualquier persona que desee jugar
Requerimiento Funcional	- Título, subtítulos e información - Que se pueda interactuar con el teclado y ratón - Que sea visible tenga una GUI (Interfaz gráfica) - Que los tetrominos tengan sus respectivas funciones - Que pueda registrar puntuación, tiempo

Mundo del Problema	- Para este problema en particular, se usar el sistema en base 10 u arábigo, y los números son enteros, pueden ser double, complejos etc. - Tiene una GUI (Interfaz gráfica) - Se está desarrollando a través de NetBeans
Requerimiento no funcional	- Se está desarrollando por NetBeans usando el GUI que tiene por defecto para hacer más fácil el desarrollo de la interfaz grafica - Tanto como los botones y demás tienen sus respectivas funciones y eventos predeterminados - Los colores y fondos de pantalla que se usan son predeterminados por el GUI de NeatBeans

Nombre	Cargar datos
Resumen	Dado iniciado el programa tiene que recibir las instrucciones
Entradas	
Flecha arriba, Flecha abajo, Flecha Izquierda, Flecha derecha, Tecla P	
Resultados	
Calcula los resultados para dar movimiento a los “Tetrominos” o dar pausa al codigo	

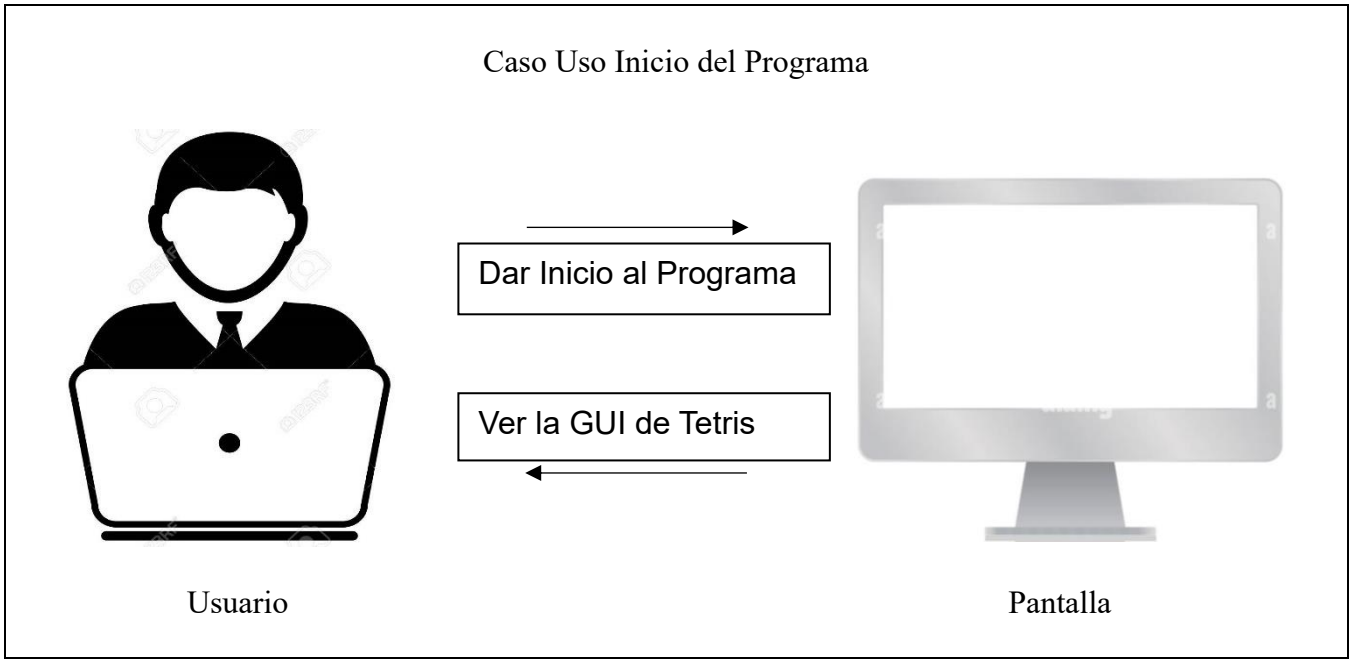
Nombre	Mostrar Resultados
Resumen	Dado iniciado el programa tiene que mostrar los resultados por el GUI
Entradas	
Flecha arriba, Flecha abajo, Flecha Izquierda, Flecha derecha, Tecla P	
Resultados	
Se mueve el “Tetromino” a través del GUI y la tecla P pausa el juego	

1.3 **Historia de Usuarios**

Historia de Usuario N.	1	**Titulo**	Cargar datos
Descripción	Como:	Usuario	
	Quiero:	Iniciar el programa	
	Para:	Probar el programa	
Criterios de Aceptación	• El programa tiene que iniciar • No deber de haber errores al iniciar el programa • El GUI tiene que mostrarse por pantalla • Las entradas tienen que registrarse por el programa		

Historia de Usuario N.	2	**Titulo**	Entregar los Datos
Descripción	Como:	Usuario	
	Quiero:	Iniciar el programa	
	Para:	Jugar	
Criterios de Aceptación	• Jugar Tetris sin ningún error en el programa		

2. **Diseño**



Entidad – Clase	Descripción
App o Tetris	Es la entidad o clase principal, lleva el orden como se utiliza los objetos de cada clase.
Tetris	Es la entidad más importante del mundo del problema, puesto que define su frontera.

Tetris
-Piezas1: GUI -Stats1: GUI -Tablero1: GUI
+Piezas() +Stats() +Tablero() +TetrisGUI()

Clase Piezas

Atributo	Valores Posibles	Comentarios
Pieza I	Cuatro valores: arriba, abajo, izquierda y derecha	La pieza I es fija
Pieza O	Cuatro valores: arriba, abajo, izquierda y derecha	La pieza O es fija
Pieza T	Cuatro valores: arriba, abajo, izquierda y derecha	La pieza T es fija
Pieza S	Cuatro valores: arriba, abajo, izquierda y derecha	La pieza S es fija
Pieza Z	Cuatro valores: arriba, abajo, izquierda y derecha	La pieza Z es fija
Pieza J	Cuatro valores: arriba, abajo, izquierda y derecha	La pieza J es fija
Pieza L	Cuatro valores: arriba, abajo, izquierda y derecha	La pieza L es fija

Clase Stats

Atributo	Valores Posibles	Comentarios
Tiempo	Desde 0 hasta 9:99	Iniciar en 0 y avanza hasta terminar el programa
Puntaje	Desde 0 hasta 9999	Iniciar en 0 y avanza hasta terminar el programa

Clase Tablero

Atributo	Valores Posibles	Comentarios
Panel	Es fijo	Se le muestra al usuario un GUI

Clase TetrisGUI

Atributo	Valores Posibles	Comentarios
Main	Maneja y da inicio al programa	Iniciar programa

3. Codificación o Construcción de la solución

TetrisGUI.java

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.BoxLayout;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
```

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JDialog;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import java.awt.Color;
import javax.swing.BorderFactory;
import java.awt.Dimension;

public class TetrisGUI extends JFrame{
	private JPanel panel;
	private Tablero zonaJuego;
	private Stats zonaStats;
	private JMenuBar menuMnb;
	private JMenu juegoMnu;
	private JMenuItem juegoNuevoItm;
	private JMenuItem salirItm;
	private JMenu configMnu;
	private JMenuItem controlesItm;

	public TetrisGUI(){
		this.setTitle("Super Amazing Sugoi Tetris");
		this.setSize(480,700);
		this.setLocationRelativeTo(null);
		//this.setResizable(false);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		menuMnb = new JMenuBar();
		juegoMnu = new JMenu("Juego");
		juegoNuevoItm = new JMenuItem("Juego nuevo");
		juegoNuevoItm.addActionListener(new ActionListener(){
					public void actionPerformed(ActionEvent e){
						zonaJuego.reset();
					}
				});
		salirItm = new JMenuItem("Salir");
		salirItm.addActionListener(new ActionListener(){
					public void actionPerformed(ActionEvent e){
						System.exit(0);
					}
				});
		juegoMnu.add(juegoNuevoItm);
		juegoMnu.add(salirItm);
		configMnu = new JMenu("");
		controlesItm = new JMenuItem("Controles");
		controlesItm.addActionListener(new ActionListener(){
					public void actionPerformed(ActionEvent e){
```

```
                                                dialogoControles(e);
                                        }
                                });
                configMnu.add(controlesItm);
                menuMnb.add(juegoMnu);
                menuMnb.add(configMnu);
                this.setJMenuBar(menuMnb);

                panel = new JPanel();
                panel.setLayout(new BoxLayout(panel,BoxLayout.X_AXIS));
                zonaJuego = new Tablero(this);
                zonaJuego.setMaximumSize(new Dimension(300,600));
                zonaJuego.setMinimumSize(new Dimension(300,600));
                zonaJuego.setBorder(BorderFactory.createEmptyBorder(10,10,10,0));
                panel.add(zonaJuego);
                zonaStats = new Stats();
                panel.add(zonaStats);
                this.setContentPane(panel);
        }

        public Tablero getTablero(){
                return zonaJuego;
        }

        public void ponerPiezaSiguiente(int[][] next, Color co){
                zonaStats.next(next, co);
        }

        public void ponerStats(int p, int l){
                zonaStats.setStats(p,l);
        }

        public void ponerNivel(int level){
                zonaStats.setNivel(level);
        }

        public void dialogoControles(ActionEvent e){
                zonaJuego.pausar();
                JDialog controles = new JDialog(this,"Controles",true);
                ImageIcon ic = new ImageIcon("Controles.jpg");
                JLabel etiqueta = new JLabel(ic);
                controles.add(etiqueta);
                controles.pack();
                controles.setLocationRelativeTo(this);
                controles.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
                controles.setVisible(true);
                zonaJuego.pausar();
```

```
	}

	public static void main(String[] args){
		long time = System.currentTimeMillis();
		long  controlNivel = System.currentTimeMillis();

		TetrisGUI f = new TetrisGUI();
		f.setVisible(true);
		while(true){			//Game loop.
			if(System.currentTimeMillis() - time > f.getTablero().getVelocidad()){
				f.getTablero().mover();
				time = System.currentTimeMillis();
			}
			if(System.currentTimeMillis() - controlNivel > 105000){
				f.getTablero().enviarNivel();
				controlNivel = System.currentTimeMillis();
			}

		}
	}
}
```

Tablero.java

```
import javax.swing.BorderFactory;
import javax.swing.JPanel;
import java.awt.Color;
import javax.swing.JFrame;
import java.awt.GridLayout;
import java.util.Random;
import java.awt.event.KeyEvent;
import java.awt.event.KeyAdapter;
import javax.swing.JOptionPane;

public class Tablero extends JPanel{
	private JPanel[][] casillas;
	private int[][] mapa;				//matriz de enteros en donde se marca la posicion
de la figura cayendo
	private int ancho = 15;
	private int alto = 30;
	private Piezas[] p;
	private boolean llegarAbajo = true;
	private int[][] coordenadasPiezaCayendo;
	private int piezaActual;
	private int piezaSiguiente;
	private int[][] piezasPuestas;				//matriz de enteros en donde se marcan las
piezas ya colocadas
```

```
	private int velocidad = 800;
	private int velocidadActual = velocidad;
	private TetrisGUI owner;
	private boolean pausa = false;
	private Color fondoCasillas = Color.black;
	private final int puntosLlegarAbajo = 30;
	private final int puntosFilaCompleta = 100;
	private final int bonusFila = 15;
	private int puntaje = 0;
	private int lineas = 0;
	private int nivel = 1;

	public Tablero(TetrisGUI t){
		owner = t;
		casillas = new JPanel[alto][ancho];
		mapa = new int[alto][ancho];
		piezasPuestas = new int[alto][ancho];
		for(int i = 0; i < alto; i ++){
			for(int j = 0; j < ancho; j++){
				casillas[i][j] = new JPanel();
				casillas[i][j].setBorder(BorderFactory.createLineBorder(Color.black));
				casillas[i][j].setBackground(fondoCasillas);
				this.add(casillas[i][j]);
				mapa[i][j] = 0;
				piezasPuestas[i][j] = 0;

			}
		}
		coordenadasPiezaCayendo = new int[4][2];
		mapa = new int[alto][ancho];
		p = Piezas.values();
		Random rnd = new Random();
		piezaSiguiente = rnd.nextInt(7);                    //en este arreglo se guardan
los datos de las diferentes piezas
		this.setLayout(new GridLayout(alto,ancho));
		this.addKeyListener(new KeyAdapter() {
                                public void keyPressed(KeyEvent e){
                                        precionarTecla(e);
                                }
                                public void keyReleased(KeyEvent e){
                                        soltarTecla(e);
                                };
                        });
		this.setFocusable(true);
	}
```

```
	public void precionarTecla(KeyEvent e){ //Metodo encargado de la jugabilidad, moviendo la pieza de derecha a izquierda, o acelerando su caida
		if(e.getKeyCode() == KeyEvent.VK_LEFT){ //dependiendo del boton pulsado
			if(this.extremoIzq() > 0 && !colicionIzquierda()){
				Color aux = casillas[coordenadasPiezaCayendo[0][0]][coordenadasPiezaCayendo[0][1]].getBackground();
				for(int i = 0; i < 4; i ++){

	mapa[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]] = 0;

	casillas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]].setBackground(fondoCasillas);
					coordenadasPiezaCayendo[i][1]--;
				}
				for(int i = 0; i < 4; i ++){

	mapa[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]] = 1;

	casillas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]].setBackground(aux);
				}
			}
		}
		if(e.getKeyCode() == KeyEvent.VK_RIGHT){
			if(this.extremoDer() + 1 < ancho && !colicionDerecha()){
				Color aux = casillas[coordenadasPiezaCayendo[0][0]][coordenadasPiezaCayendo[0][1]].getBackground();
				for(int i = 0; i < 4; i ++){

	mapa[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]] = 0;

	casillas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]].setBackground(fondoCasillas);
					coordenadasPiezaCayendo[i][1]++;
				}
				for(int i = 0; i < 4; i ++){

	mapa[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]] = 1;

	casillas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]].setBackground(aux);
				}
			}
		}
		if(e.getKeyCode() == KeyEvent.VK_DOWN){
			velocidad = 15;
```

```
                }
        }

        public boolean colicionIzquierda(){                     //Metodo que comprueba si hay piezas a la
izquierda de nuestra pieza actual
                boolean hayColicion = false;
                for(int i = 0; i < 4; i ++){

        if(piezasPuestas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1] -1] == 1)
                                hayColicion = true;

                }
                return hayColicion;
        }

        public boolean colicionDerecha(){                       //Metodo que comprueba si hay piezas a la
derecha de nuestra pieza actual
                boolean hayColicion = false;
                for(int i = 0; i < 4; i ++){

        if(piezasPuestas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1] + 1] == 1)
                                hayColicion = true;

                }
                return hayColicion;
        }

        public boolean colicionVertical(){                      //Metodo que comprueba si hay piezas
abajo de nuestra pieza actual
                boolean hayColicion = false;
                for(int i = 0; i < 4; i ++){
                        if(piezasPuestas[coordenadasPiezaCayendo[i][0] +
1][coordenadasPiezaCayendo[i][1]] == 1)
                                hayColicion = true;

                }
                return hayColicion;
        }

        public void soltarTecla(KeyEvent e){                                    //Metodo encargado de
redibujar la pieza al rotarla
                if(e.getKeyCode() == KeyEvent.VK_UP){
                        Color aux =
casillas[coordenadasPiezaCayendo[0][0]][coordenadasPiezaCayendo[0][1]].getBackground();

                        for(int i = 0; i < 4; i ++){
```

```
                                mapa[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]]
= 0;

        casillas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]].setBackground(fon
doCasillas);
                        }

                        p[piezaActual].rotar(coordenadasPiezaCayendo,piezasPuestas);

                        for(int i = 0; i < 4; i ++){
                                mapa[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]]
= 1;

        casillas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]].setBackground(aux
);
                        }
                }
                if(e.getKeyCode() == KeyEvent.VK_DOWN){
                        velocidad = velocidadActual;
                }
                if(e.getKeyCode() == KeyEvent.VK_P){
                        pausa = !pausa;
                        if(pausa)
                                owner.setTitle(owner.getTitle() + " ****PAUSA****");
                        else
                                owner.setTitle(owner.getTitle().substring(0,26));
                }
        }

        public int extremoDer(){                        //Retorna la componente X mayor de las
coordenadas de la pieza actual
                int mayor = coordenadasPiezaCayendo[0][1];
                for(int i = 0; i < 4; i++){
                        if(coordenadasPiezaCayendo[i][1] > mayor)
                                mayor = coordenadasPiezaCayendo[i][1];

                }
                return mayor;
        }

        public int extremoIzq(){                        //Retorna la componente X menor de las
coordenadas de la pieza actual
                int menor = coordenadasPiezaCayendo[0][1];
                for(int i = 0; i < 4; i++){
                        if(coordenadasPiezaCayendo[i][1] < menor)
                                menor = coordenadasPiezaCayendo[i][1];
```

```
                }
                return menor;
        }

        public int extremoInf(){                        //Retorna la componente Y mayor de las
coordenadas de la pieza actual
                int mayor = coordenadasPiezaCayendo[0][0];
                for(int i = 0; i < 4; i++){
                        if(coordenadasPiezaCayendo[i][0] > mayor)
                                mayor = coordenadasPiezaCayendo[i][0];

                }
                return mayor;
        }

        public int extremoSup(){                        //Retorna la componente Y menor de las
coordenadas de la pieza actual
                int menor = coordenadasPiezaCayendo[0][0];
                for(int i = 0; i < 4; i++){
                        if(coordenadasPiezaCayendo[i][0] < menor)
                                menor = coordenadasPiezaCayendo[i][0];

                }
                return menor;
        }

        public int getVelocidad(){
                return velocidad;
        }

        public void gameOver(){
                Object[] opciones = {"Empezar un juego nuevo","Ragequit"};
                int n = JOptionPane.showOptionDialog(owner,"Se acabo el juego,
PERDEDOR!","Game Over",JOptionPane.YES_NO_OPTION,
JOptionPane.PLAIN_MESSAGE,null,opciones,opciones[1]);
                if(n == JOptionPane.YES_OPTION)
                        reset();
                if(n == JOptionPane.NO_OPTION || n == JOptionPane.CLOSED_OPTION)
                        System.exit(0);
        }

        public void reset(){
                for(int i = 0; i < alto; i ++){
                        for(int j = 0; j < ancho; j++){
                                casillas[i][j].setBackground(fondoCasillas);
                                mapa[i][j] = 0;
                                piezasPuestas[i][j] = 0;
```

```
                }
            }
            velocidad = 800;
            this.llegarAbajo = true;
            puntaje = 0;
            lineas = 0;
            nivel = 1;
            enviarStats();
            owner.ponerNivel(nivel);
            this.mover();
        }

        public void pausar(){
            pausa = !pausa;
        }

        public boolean enFila(int[] filas, int coo){                    //Metodo para hacer que una
coordenada repetida no sea contada como una fila completa
            for(int i = 0; i < 4; i++){
                if(filas[i] == coo){
                    return true;
                }
            }
            return false;
        }

        public void enviarPiezaSiguiente(){
            owner.ponerPiezaSiguiente(p[piezaSiguiente].getPieza(),p[piezaSiguiente].getColor());
        }

        public void enviarStats(){
            owner.ponerStats(puntaje,lineas);
        }

        public void enviarNivel(){
            if(velocidad >= 0){
                velocidad -=40;
                velocidadActual = velocidad;
                nivel++;
                owner.ponerNivel(nivel);
            }
        }

        public void comprobarFilas(){                                   //Metodo para comprobar si se
completo una fila
            int[] filas = new int[4];
```

```
                int contador = 0;
                int numFilas = 0;

                for(int i = 0; i < 4; i++){
                        for(int j = 0; j < ancho; j++){
                                if(piezasPuestas[coordenadasPiezaCayendo[i][0]][j] == 1){
                //Si una fila de piezasPuestas tiene solo unos, significa que esta completa
                                        contador++;
                                }
                        }
                        if(contador == ancho && !enFila(filas,coordenadasPiezaCayendo[i][0])){
                        //Se almacenan las coordenadas de las filas completas en el arreglo filas
                                filas[numFilas] = coordenadasPiezaCayendo[i][0];
                                System.out.println("Coordenada: " + coordenadasPiezaCayendo[i][0]);
                                System.out.println("filas[" + numFilas + "]: " + filas[numFilas]);

                                numFilas++;
                        }
                        contador = 0;
                }

                if(numFilas > 0){               //Si se completa una fila luego de poner una pieza...
                        for(int i=0;i <= numFilas-1;i++){
                                System.out.print("|" + filas[i] + "|");
                        }
                         for(int i = 0; i < numFilas - 1; i++){                                 //Algoritmo
Burbuja, para ordenar las coordenadas de las filas de mayor a menor
                                for(int j = 0; j < numFilas - i - 1; j++){
                                        if(filas[j + 1] < filas[j]){
                                                int aux = filas[j + 1];
                                                filas[j + 1] = filas[j];
                                                filas[j] = aux;
                                        }
                                }
                        }

                        System.out.println("numFilas: " + numFilas);
                        for(int i=0;i <= numFilas-1;i++){
                                System.out.print("|" + filas[i] + "|");
                        }

                        System.out.println("numFilas: " + numFilas);

                        puntaje += (puntosFilaCompleta*numFilas) + (bonusFila*(numFilas-1));
                        lineas += numFilas;
```

```
                        for(int i = filas[numFilas-1]; i > 0; i--){                    //Esta seccion
se encarga de borrar las filas completas, y de bajar todas las demas
                                for(int j = 0; j < ancho; j++){
                                        if(i - numFilas < 0){
                                                piezasPuestas[i][j] = 0;
                                                casillas[i][j].setBackground(fondoCasillas);
                                        }
                                        else{
                                                piezasPuestas[i][j] = piezasPuestas[i-numFilas][j];
                                                casillas[i][j].setBackground(casillas[i-
numFilas][j].getBackground());
                                        }
                                }
                        }



                }
        }

        public void mover(){
                if(!pausa){                             //Movimiento uniforme hacia abajo del juego
                        if(llegarAbajo){
                                this.piezaNueva();
                                llegarAbajo = false;
                                enviarPiezaSiguiente();
                        }
                        else{
                                if(extremoInf() + 1 < alto && !colicionVertical()){
                                        Color aux =
casillas[coordenadasPiezaCayendo[0][0]][coordenadasPiezaCayendo[0][1]].getBackground();
                                        for(int i = 0; i < 4; i ++){

        mapa[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]] = 0;

        casillas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]].setBackground(fon
doCasillas);
                                                coordenadasPiezaCayendo[i][0]++;
                                        }
                                        for(int i = 0; i < 4; i ++){

        mapa[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]] = 1;

        casillas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]].setBackground(aux
);
                                        }
                                }
                                else {
```

```
                                        llegarAbajo = true;
                                        for(int i = 0; i < 4; i ++){

        piezasPuestas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]] = 1;
                                        }
                                        puntaje += puntosLlegarAbajo;

                                        comprobarFilas();
                                        enviarStats();

                                        for(int i = 0; i < alto; i ++){
                                                for(int j = 0; j < ancho; j++){
                                                        System.out.print("|"+piezasPuestas[i][j]+"|");
                                                }
                                                System.out.println("");
                                        }
                                        this.mover();
                                }
                        }
                }
        }

        public void piezaNueva(){                               //Genera una nueva pieza aleatoria
                Random rnd = new Random();
                piezaActual = piezaSiguiente;
                piezaSiguiente = rnd.nextInt(7);
                //piezaActual = 0;

                coordenadasPiezaCayendo = p[piezaActual].getPieza();

                for(int i = 0; i < 4; i++){

        if(piezasPuestas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]] == 1){
                                gameOver();
                                break;
                        }
                }
                for(int i = 0; i < 4; i++){
                        mapa[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]] = 1;

        casillas[coordenadasPiezaCayendo[i][0]][coordenadasPiezaCayendo[i][1]].setBackground(p[p
iezaActual].getColor());
                }
        }

}
```

Stats.java

```
import javax.swing.JPanel;
import javax.swing.BoxLayout;
import javax.swing.JLabel;
import java.awt.Color;
import javax.swing.BorderFactory;
import java.awt.GridLayout;
import java.awt.Dimension;
import javax.swing.Box;
import java.awt.Font;
import java.awt.Component;

//Clase que se encarga del puntaje, lineas, nivel, y mostrar la siguiente pieza.

public class Stats extends JPanel{
	private JLabel nextLbl;
	private JLabel nivelLbl;
	private JLabel lineasLbl;
	private JLabel linesLbl;
	private JLabel puntajeLbl;
	private JLabel scoreLbl;
	private JLabel levelLbl;
	private JPanel nextExternoPnl;
	private JPanel nextInternoPnl;
	private JPanel siguientePnl;
	private JPanel puntajePnl;
	private JPanel scorePnl;
	private JPanel lineasPnl;
	private JPanel linesPnl;
	private JPanel nivelPnl;
	private JPanel levelPnl;
	private JPanel[][] matrizPnl;

	public Stats(){
		this.setLayout(new BoxLayout(this,BoxLayout.Y_AXIS));

		nextLbl = new JLabel("Siguiente: ");
		nextLbl.setFont(new Font("System",Font.PLAIN,20));
		nextLbl.setForeground(Color.yellow);
		nextInternoPnl = new JPanel(new GridLayout(4,3));
		siguientePnl = new JPanel();
		nextExternoPnl = new JPanel();
		nextExternoPnl.setLayout(new BoxLayout(nextExternoPnl,BoxLayout.Y_AXIS));
		matrizPnl = new JPanel[4][4];
		for(int i = 0; i < 4; i++){
			for(int j = 0; j < 3; j++){
```

```
                matrizPnl[i][j] = new JPanel();
                matrizPnl[i][j].setBorder(BorderFactory.createLineBorder(Color.black));
                matrizPnl[i][j].setBackground(Color.black);
                nextInternoPnl.add(matrizPnl[i][j]);
            }
        }

        nextInternoPnl.setBorder(BorderFactory.createMatteBorder(0,30,0,30,Color.black));
        nextInternoPnl.setBackground(Color.black);
        nextInternoPnl.setMaximumSize(new Dimension(140,100));
        nextInternoPnl.setMinimumSize(new Dimension(140,100));
        siguientePnl.setBorder(BorderFactory.createMatteBorder(5,5,5,5,Color.gray));
        siguientePnl.setBackground(Color.blue);
        siguientePnl.add(nextLbl);
        siguientePnl.setMaximumSize(new Dimension(150,40));
        nextExternoPnl.add(siguientePnl);
        nextExternoPnl.add(nextInternoPnl);
        this.add(nextExternoPnl);

        this.add(Box.createRigidArea(new Dimension(0,30)));

        nivelLbl = new JLabel("Nivel");
        nivelLbl.setFont(new Font("System",Font.PLAIN,20));
        nivelLbl.setForeground(Color.yellow);
        nivelPnl = new JPanel();
        nivelPnl.setBorder(BorderFactory.createMatteBorder(5,5,0,5,Color.red));
        nivelPnl.setBackground(Color.blue);
        nivelPnl.add(nivelLbl);
        nivelPnl.setMaximumSize(new Dimension(150,40));
        this.add(nivelPnl);

        levelLbl = new JLabel("1");
        levelLbl.setFont(new Font("System",Font.PLAIN,20));
        levelLbl.setForeground(Color.yellow);
        levelPnl = new JPanel();
        levelPnl.setBorder(BorderFactory.createMatteBorder(0,5,5,5,Color.red));
        levelPnl.setBackground(Color.blue);
        levelPnl.add(levelLbl);
        levelPnl.setMaximumSize(new Dimension(150,40));
        this.add(levelPnl);

        this.add(Box.createRigidArea(new Dimension(0,30)));

        puntajeLbl = new JLabel("Puntaje: ");
        puntajeLbl.setFont(new Font("System",Font.PLAIN,20));
        puntajeLbl.setForeground(Color.yellow);
        puntajePnl = new JPanel();
```

```
        puntajePnl.setBorder(BorderFactory.createMatteBorder(5,5,0,5,Color.magenta));
        puntajePnl.setBackground(Color.blue);
        puntajePnl.add(puntajeLbl);
        puntajePnl.setMaximumSize(new Dimension(150,40));
        this.add(puntajePnl);

        scoreLbl = new JLabel("0");
        scoreLbl.setFont(new Font("System",Font.PLAIN,20));
        scoreLbl.setForeground(Color.yellow);
        scorePnl = new JPanel();
        scorePnl.setBorder(BorderFactory.createMatteBorder(0,5,5,5,Color.magenta));
        scorePnl.setBackground(Color.blue);
        scorePnl.add(scoreLbl);
        scorePnl.setMaximumSize(new Dimension(150,40));
        this.add(scorePnl);

        this.add(Box.createRigidArea(new Dimension(0,30)));

        lineasLbl = new JLabel("Lineas");
        lineasLbl.setFont(new Font("System",Font.PLAIN,20));
        lineasLbl.setForeground(Color.yellow);
        lineasPnl = new JPanel();
        lineasPnl.setBorder(BorderFactory.createMatteBorder(5,5,0,5,Color.orange));
        lineasPnl.setBackground(Color.blue);
        lineasPnl.add(lineasLbl);
        lineasPnl.setMaximumSize(new Dimension(150,40));
        this.add(lineasPnl);

        linesLbl = new JLabel("0");
        linesLbl.setFont(new Font("System",Font.PLAIN,20));
        linesLbl.setForeground(Color.yellow);
        linesPnl = new JPanel();
        linesPnl.setBorder(BorderFactory.createMatteBorder(0,5,5,5,Color.orange));
        linesPnl.setBackground(Color.blue);
        linesPnl.add(linesLbl);
        linesPnl.setMaximumSize(new Dimension(150,40));
        this.add(linesPnl);

    }

    public void next(int[][] n, Color c){
        for(int i = 0; i < 4; i++){
            for(int j = 0; j < 3; j++){
                matrizPnl[i][j].setBackground(Color.black);
            }
        }
        for(int i = 0; i < 4; i++){
```

```
                        matrizPnl[n[i][0]][n[i][1]-6].setBackground(c);
                }
        }

        public void setStats(int s, int l){
                scoreLbl.setText("" + s);
                linesLbl.setText("" + l);
        }

        public void setNivel(int l){
                levelLbl.setText("" + l);
        }


}
```

Piezas.java

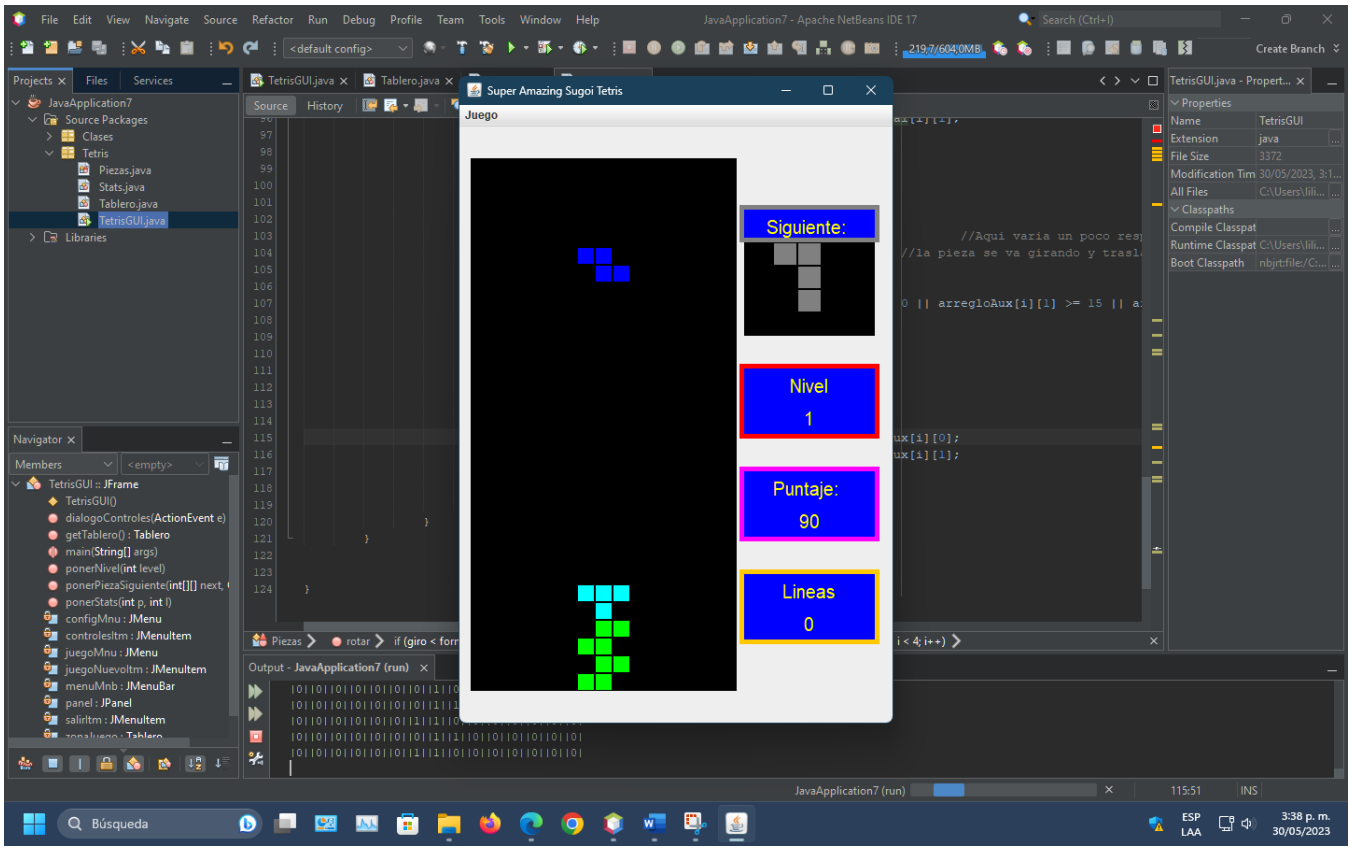
```
import java.awt.Color;

public enum Piezas{BARRA(Color.red), ZETA(Color.green), ESE(Color.blue), ELE(Color.yellow),
ELEINV(Color.gray), TE(Color.cyan), CUADRADO(Color.pink);
        Color color;
        int[] forma;
        int giro = 0;
        int[][] coordenadas = new int[][] {    //Todas las coordenadas que pueden usar las figuras
                                        { 0, 0},    //0
                                        { 0, 1},    //1
                                        {-1, 0},          //2
                                        { 1, 0},          //3
                                        {-1,-1},          //4
                                        { 0,-1},          //5
                                        { 1,-1},          //6
                                        { 0,-2}};         //7

        Piezas(Color c){
                this.color = c;
                switch(this.ordinal()){                   //Estos arreglos representan distintas
caracteristicas de las figuras. Los 4 primeros elementos son sus coordenadas para dibujarlos en un
plano
                                                //El 5to elemento es el numero de giros que puede dar.
Los 6to y 7mo elementos son deltas X e Y para poder dibujarlos correctamentes en el Tablero
                        case 0 :  forma = new int[]     {0,1,5,7,1,2,7};
                                break;
                        case 1 :  forma = new int[]    {0,2,5,6,1,1,7};
                                break;
                        case 2:  forma = new int[]      {0,3,5,4,1,1,7};
```

```
                        break;
                case 3 :  forma = new int[]     {0,1,5,6,3,1,7};
                        break;
                case 4 :  forma = new int[]     {0,1,5,4,3,1,7};
                        break;
                case 5 :  forma = new int[]     {0,1,2,3,3,0,7};
                        break;
                case 6 :  forma = new int[]     {0,5,6,3,0,1,6};
                        break;
            }
        }

        public Color getColor(){
            return color;
        }

        public int[][] getPieza(){                    //Genera las coordenadas de las figuras. Las
genera en base a las dimensiones del Tablero
            int[][] matrix = new int[4][2];
            for(int i = 0; i < 4; i++){
                matrix[i][1] = coordenadas[forma[i]][0] + forma[6];
                matrix[i][0] = coordenadas[forma[i]][1] + forma[5];
            }
            return matrix;
        }

        public void rotar(int[][] coordenadasPiezaActual, int[][] piezasPuestas){

            if(giro < forma[4] && forma[4] > 0){            //Este if es para gestionar la
rotacion de las piezas, mas especificamente a las L, L invertida y la T, que pueden girar sin problemas
                                                    //ademas de excluir al cuadrado por razones obvias
                int[][] arregloAux = new int[4][2];
                int[] eje = new int[]
{coordenadasPiezaActual[0][0],coordenadasPiezaActual[0][1]};        //Establece el eje de rotacion
                boolean rotacionPermitida = true;

                for(int i = 0; i < 4; i++){
                        arregloAux[i][0] = coordenadasPiezaActual[i][0];
                        arregloAux[i][1] = coordenadasPiezaActual[i][1];
                    }

                for(int i = 0; i < 4; i++){                         //Toda la rotacion se hace en
un arreglo auxiliar, para despues verificar si la pieza rotada no sale de los margenes
                                                                    //o choca con otras piezas
                    arregloAux[i][0] -= eje[0];
                    arregloAux[i][1] -= eje[1];
                    int aux = arregloAux[i][0];
```

```
                        arregloAux[i][0] = arregloAux[i][1];
                        arregloAux[i][1] = aux * -1;
                        arregloAux[i][0] += eje[0];
                        arregloAux[i][1] += eje[1];
                        if(arregloAux[i][0] >= 30 || arregloAux[i][0] < 0 || arregloAux[i][1] >=
15 || arregloAux[i][1] < 0 || piezasPuestas[arregloAux[i][0]][arregloAux[i][1]] == 1){
                                rotacionPermitida = false;
                                break;
                        }
                }

                if(rotacionPermitida){                  //Si se puede rotar sin problemas, modifica
el arreglo enviado por referencia
                        for(int i = 0; i < 4; i++){
                                coordenadasPiezaActual[i][0] = arregloAux[i][0];
                                coordenadasPiezaActual[i][1] = arregloAux[i][1];
                        }
                }
                giro++;
                if(giro >= 3)
                        giro = 0;
        }
        else if(forma[4] != 0){                         //Aqui se trata a la I, S , Z
                int[][] arregloAux = new int[4][2];
                int[] eje = new int[]
{coordenadasPiezaActual[0][0],coordenadasPiezaActual[0][1]};
                boolean rotacionPermitida = true;

                for(int i = 0; i < 4; i++){
                        arregloAux[i][0] = coordenadasPiezaActual[i][0];
                        arregloAux[i][1] = coordenadasPiezaActual[i][1];
                }

                for(int i = 0; i < 4; i++){
                        arregloAux[i][0] -= eje[0];
                        arregloAux[i][1] -= eje[1];
                        int aux = arregloAux[i][0];
                        arregloAux[i][0] = arregloAux[i][1]*-1;                 //Aqui varia
un poco respecto a lo de arriba, ya que en estas piezas no se puede estabecer un unico eje de rotacion
                        arregloAux[i][1] = aux;                                 //la pieza se va
girando y trasladando, en vez de solo girar en un eje
                        arregloAux[i][0] += eje[0];
                        arregloAux[i][1] += eje[1];
                        if(arregloAux[i][0] >= 30 || arregloAux[i][0] < 0 || arregloAux[i][1] >=
15 || arregloAux[i][1] < 0 || piezasPuestas[arregloAux[i][0]][arregloAux[i][1]] == 1){
                                rotacionPermitida = false;
                                break;
```

```
                        }
                    }

                    if(rotacionPermitida){
                        for(int i = 0; i < 4; i++){
                            coordenadasPiezaActual[i][0] = arregloAux[i][0];
                            coordenadasPiezaActual[i][1] = arregloAux[i][1];
                        }
                    }
                    giro = 0;
                }
            }


}
```

4. **Salida**



5. Conclusiones

No sé cómo llegue aquí…