MANUAL TÉCNICO – JUEGO TETRIS EN CONSOLA (JAVA)

Facultad de Ingeniería – Escuela de Ciencias y Sistemas
Universidad de San Carlos de Guatemala
Introducción a la Programación y Computación 1 – Sección A

Proyecto: Implementación del Juego Tetris en Consola
Lenguaje: Java
Autor: *[Ramiro Castellanos-202302574]*
Fecha: Diciembre 2025

1. DESCRIPCIÓN GENERAL DEL SISTEMA

Este proyecto implementa el **juego Tetris completamente en consola**, con piezas rotables, puntajes, niveles, estadísticas y registro de mejores jugadores.

El sistema incluye:

- Gestión del tablero (20x10)
- 7 piezas oficiales (Tetrominos)
- Rotación de piezas
- Caída controlada y rápida
- Eliminación de líneas
- Sistema de puntuación y niveles
- Estadísticas globales
- Persistencia de datos con archivos
- Interfaz de texto con colores ANSI

2. ARQUITECTURA DEL PROYECTO

El sistema sigue una arquitectura **modular**, donde cada componente cumple una responsabilidad específica:

Módulo	Función
`main`	Menú principal e interacción general
`juegoPrincipal`	Control del ciclo de juego
`gestorJuego`	Lógica del juego y reglas
`tablero`	Gestión del área de juego
`pieza` y `tetromino`	Representación y rotación de piezas
`puntuacion`	Sistema de puntuación
`gestorArchivos`	Persistencia de datos
`utilidadesConsola`	Herramientas visuales para consola

3. ESTRUCTURA DE PAQUETES Y CLASES

practica1/
│
├── main.java
├── juegoPrincipal.java
├── gestorJuego.java
├── tablero.java
├── pieza.java
├── tetromino.java
├── puntuacion.java
├── gestorArchivos.java
└── utilidadesConsola.java

4. DESCRIPCIÓN TÉCNICA DE CLASES

4.1 main.java

- Gestiona el **menú principal**.
- Llama a módulos: puntajes, estadísticas, sistema de puntuación.
- Ejecuta la partida con `juegoPrincipal`.

```java
package practical;

import java.util.Scanner;

public class main {
    private static final String CARNE = "202302574";

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            mostrarMenuPrincipal(scanner);
        }
    }

    private static void mostrarMenuPrincipal(Scanner scanner) {
        utilidadesConsola.clearScreen();
        System.out.println("========================================");
        System.out.println("     TETRIS - CARNÉ: " + CARNE);
        System.out.println("========================================\n");

        System.out.println("         MENÚ PRINCIPAL");
        System.out.println("        ----------------");
        System.out.println("        1. Jugar Partida Nueva");
        System.out.println("        2. Ver Mejores Puntajes");
        System.out.println("        3. Ver Estadísticas Generales");
        System.out.println("        4. Sistema de Puntuación");
        System.out.println("        5. Salir del Juego");
        System.out.print("\n       Seleccione una opción (1-5): ");

        String opcion = scanner.nextLine();

```

4.2 juegoPrincipal.java

- Controla el **ciclo de vida completo de una partida**.
- Maneja entrada del usuario.
- Llama a `gestorArchivos` para guardar puntajes y estadísticas.

```java
package practical;

import java.util.Scanner;

public class juegoPrincipal {
    private gestorJuego gameManager;
    private long startTime;
    private String carne;

    public juegoPrincipal(String carne) {
        this.carne = carne;
    }

    public void start() {
        gameManager = new gestorJuego(carne);
        startTime = System.currentTimeMillis();

        Scanner scanner = new Scanner(System.in);
        utilidadesConsola.hideCursor();

        while (!gameManager.isGameOver()) {
            gameManager.draw();

            System.out.flush();
            String input = scanner.nextLine().trim();

            if (input.equalsIgnoreCase("ESC")) {
                System.out.println("\n  Cancelando partida...");
                utilidadesConsola.showCursor();
                return;
            }
```

4.3 gestorJuego.java

Implementa TODA la lógica del Tetris:

- Movimiento horizontal
- Rotación
- Soft Drop y Hard Drop
- Colocación de piezas
- Eliminación de líneas
- Verificación de Game Over
- Sistema de pausa

```java
package practical;

public class gestorJuego {
    private tablero board;
    private pieza currentPiece;
    private pieza nextPiece;
    private puntuacion scoreManager;
    private boolean isPaused;
    private boolean isGameOver;
    private boolean terminadoPorUsuario;
    private String carne;

    // CONSTRUCTOR
    public gestorJuego(String carne) {
        this.carne = carne;
        this.board = new tablero();
        this.scoreManager = new puntuacion();
        this.nextPiece = new pieza(tetromino.getRandomPieceIndex());
        this.spawnNewPiece();
        this.isPaused = false;
        this.isGameOver = false;
        this.terminadoPorUsuario = false;
    }

    // MÉTODO: Generar nueva pieza
    private void spawnNewPiece() {
        currentPiece = nextPiece;
        nextPiece = new pieza(tetromino.getRandomPieceIndex());

        if (board.isGameOver(currentPiece)) {
            isGameOver = true;
        }
```

4.4 tablero.java

- Matriz base 20x10.
- Maneja colisiones.
- Dibuja el tablero en consola.
- Implementa el algoritmo de limpieza de líneas.

```java
package practical;

public class tablero {
    // CONSTANTES
    private static final int ROWS = 20;
    private static final int COLS = 10;
    private static final char EMPTY_CELL = '·';
    private static final char FILLED_CELL = '■';

    private char[][] grid;

    // CONSTRUCTOR
    public tablero() {
        grid = new char[ROWS][COLS];
        clear();
    }

    // Limpiar el tablero
    public void clear() {
        for (int i = 0; i < ROWS; i++) {
            for (int j = 0; j < COLS; j++) {
                grid[i][j] = EMPTY_CELL;
            }
        }
    }

    // Verificar si una posición es válida para una pieza
    public boolean isValidPosition(pieza piece, int offsetX, int offsetY) {
        int[][] shape = piece.getShape();
        int pieceX = piece.getX() + offsetX;
        int pieceY = piece.getY() + offsetY;
```

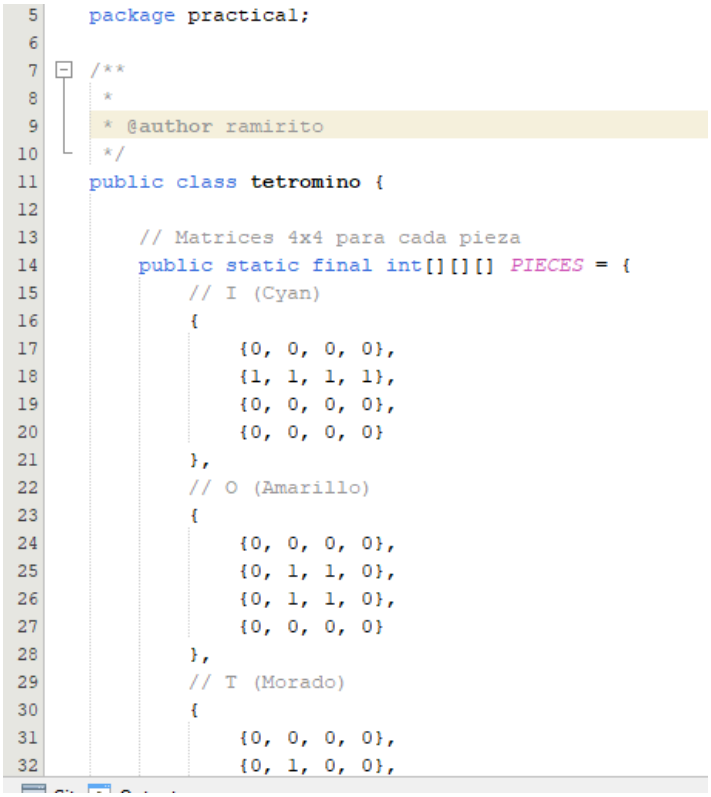
4.5 pieza.java

- Maneja posición (x,y), rotación y forma.
- Posee matrices 4x4 por diseño oficial.

```java
package practical;

/**
 *
 * @author ramirito
 */
public class pieza {

    private int[][] shape;
    private int x, y;
    private char type;

    public pieza(int pieceIndex) {
        this.shape = tetromino.getPiece(pieceIndex);
        this.type = tetromino.getPieceType(pieceIndex);
        this.x = 3; // Columna inicial (centro del tablero 10 columnas)
        this.y = 0; // Fila inicial
    }

    public pieza(int[][] shape, char type) {
        this.shape = shape;
        this.type = type;
        this.x = 3;
        this.y = 0;
    }

    public int[][] getShape() {
        return shape;
```

4.6 tetromino.java

- Define las **7 piezas clásicas** del Tetris.
- Genera piezas aleatorias.



```
5   package practical;
6
7   /**
8    *
9    * @author ramirito
10   */
11  public class tetromino {
12
13      // Matrices 4x4 para cada pieza
14      public static final int[][][] PIECES = {
15          // I (Cyan)
16          {
17              {0, 0, 0, 0},
18              {1, 1, 1, 1},
19              {0, 0, 0, 0},
20              {0, 0, 0, 0}
21          },
22          // O (Amarillo)
23          {
24              {0, 0, 0, 0},
25              {0, 1, 1, 0},
26              {0, 1, 1, 0},
27              {0, 0, 0, 0}
28          },
29          // T (Morado)
30          {
31              {0, 0, 0, 0},
32              {0, 1, 0, 0},
```

4.7 puntuacion.java

- Aplica reglas:
    - 1 línea = 100×nivel
    - 2 líneas = 300×nivel
    - 3 líneas = 500×nivel
    - 4 líneas = 800×nivel
- Calcula nivel dinámicamente.
- Lleva registro de Tetris realizados.

```java
package practical;

public class puntuacion {
    private int score;
    private int level;
    private int totalLines;
    private int tetrisCount;

    public puntuacion() {
        score = 0;
        level = 1;
        totalLines = 0;
        tetrisCount = 0;
    }

    // Método para sumar puntos por líneas eliminadas
    public void addLinesCleared(int lines) {
        totalLines += lines;

        // Calcular puntos según reglas originales
        int linePoints = 0;
        switch (lines) {
            case 1:
                linePoints = 100 * level;
                break;
            case 2:
                linePoints = 300 * level;
                break;
            case 3:
                linePoints = 500 * level;
                break;
            case 4:
```

4.8 gestorArchivos.java

- Guarda puntajes en **mejores_puntajes.txt**
- Guarda estadísticas en **estadisticas.txt**
- Ordena automáticamente el Top 10.

```java
package practical;

import java.io.*;
import java.text.SimpleDateFormat;
import java.util.Date;

public class gestorArchivos {
    private static final String HIGH_SCORES_FILE = "mejores_puntajes.txt";
    private static final String STATS_FILE = "estadisticas.txt";

    public static void saveScore(String playerName, int score, int level, int lines, long gameTime) {
        try {
            // Leer estadísticas actuales primero
            int[] stats = readStatsForUpdate();

            // Actualizar estadísticas
            stats[0]++; // PARTIDAS_JUGADAS
            stats[1] += lines; // TOTAL_LINEAS
            stats[2] += score; // TOTAL_PUNTOS
            stats[3] = Math.max(stats[3], score); // MEJOR_PUNTAJE

            // Calcular promedios
            double avgScore = (double) stats[2] / stats[0];
            double avgLevel = stats[0] > 0 ? (double) (stats[1] / 10 + 1) : 0; // Nivel promedio estimado

            // Guardar estadísticas actualizadas
            saveUpdatedStats(stats, avgScore, avgLevel);

            // Guardar en mejores puntajes
            updateHighScores(playerName, score, level, lines, gameTime);

        } catch (IOException e) {
```

4.9 utilidadesConsola.java

- Limpia pantalla.
- Oculta cursor.
- Muestra colores ANSI.

```java
package practical;

public class utilidadesConsola {
    // Códigos ANSI para colores
    public static final String RESET = "\033[0m";
    public static final String GREEN = "\033[32m";
    public static final String YELLOW = "\033[33m";
    public static final String RED = "\033[31m";
    public static final String CYAN = "\033[36m";

    public static void clearScreen() {
        System.out.print("\033[H\033[2J");
        System.out.flush();
    }

    public static void hideCursor() {
        System.out.print("\033[?25l");
    }

    public static void showCursor() {
        System.out.print("\033[?25h");
    }

    public static void printColor(String text, String color) {
        System.out.print(color + text + RESET);
    }
}
```

5. ALGORITMOS IMPORTANTES

5.1. Rotación de la pieza

1. Transponer matriz

2. Invertir columnas
   → Rotación horaria correcta para Tetris

5.2. Eliminación de líneas

- Recorrer filas de abajo hacia arriba.
- Identificar filas llenas.
- Reemplazar línea eliminada desplazando filas superiores.

5.3. Hard Drop

- Simular caída sin modificar pieza real.
- Determinar distancia máxima.
- Mover pieza real esa cantidad.
- Sumar puntos por celda ×2.

5.4. Ordenamiento de puntajes

- Leer archivos
- Insertar nuevo puntaje según su valor
- Limitar a Top 10

6. PERSISTENCIA DE DATOS (ARCHIVOS)

El sistema genera dos archivos:

mejores_puntajes.txt

Formato:

1. Juan,2500,3,30,2025-12-08 10:30,02:15

Campos:

1. Nombre
2. Puntaje
3. Nivel
4. Líneas
5. Fecha
6. Tiempo de partida

estadisticas.txt

Formato:

PARTIDAS_JUGADAS: 10
TOTAL_LINEAS: 150
TOTAL_PUNTOS: 8500
PROMEDIO_PUNTOS: 850.00
PROMEDIO_NIVEL: 1.80
MEJOR_PUNTAJE: 2500
TETRIS_REALIZADOS: 5
FECHA_ULTIMA_ACTUALIZACION: 2025-12-08 11:00:00

7. REQUERIMIENTOS TÉCNICOS

Software

- JDK 8+
- Terminal con soporte ANSI
- Editor o IDE (NetBeans recomendado)

Hardware

- 512MB RAM mínimo
- CPU 1.0 GHz

8. MANTENIMIENTO Y EXPANSIÓN

El sistema puede ampliarse fácilmente agregando:

- Caída automática por tiempo
- Sistema de sonido
- Tablero con colores per-pieza
- Guardado de partidas
- Implementación gráfica con JavaFX o Swing