Proyecto
The Final Game

13 de Junio del 2018

Proyecto Final de CFGS Desarrollo de Aplicaciones Multiplataforma

Curso 2017 / 18

IES “Doctor Fleming” Oviedo

‘The Final Game’

Carlos Ramón Ferreras Menéndez
Pablo Díaz Rubio

Índice

The Final Classic

5 de Febrero del 2018

Visión general

Queremos hacer una aplicación multiplataforma, un juego que nos permita volver a experimentar la ambientación de los juegos de antaño en los cuales podías jugar durante cortos periodos sin tener que dedicarle demasiado tiempo a él mismo. Por el mismo motivo queremos que se representen los juegos más reconocidos de la época, entre los cuales podemos encontrarnos el clásico Wall Break que tantas horas de diversión nos ha generando en los antiguos móviles de pantalla de fósforo, pasando por el mítico Snake que nos proporcionó largas horas de concentración en busca de la manzana para aumentar nuestro tamaño y sin por supuesto olvidar al grande de los juegos 'retro' el inolvidable Comecocos en el cual huíamos sin descanso de los fantasmas por el laberinto en el que nos encontrábamos.

No obstante, aparte de lo anteriormente citado nos gustaría que no fuese algo de lo que ya podemos ver en la mayoría de las aplicaciones de la tienda de Aplicaciones, nos gustaría ofrecer al cliente una experiencia de juego fluida, intrigante y sobre todo sorprendente. Y para ello hacemos uso de eventos en la aplicación para así avanzar por las fases o niveles que cambiarán por completo las mecánicas de juego con una transición fluida, y muchas veces sorprendente para el jugador.

Público objetivo

Nuestra aplicación está dirigida un público amplio, dado que a él corresponde un extenso abanico generacional, y para poder complacer a este público de tan diversas edades la aplicación será sencilla para los más pequeños sin quitar la posibilidad de un gran reto para un público más adulto o experimentado. Además, estos últimos podrán revivir un recuerdo de los videojuegos de antaño.

Análisis situación

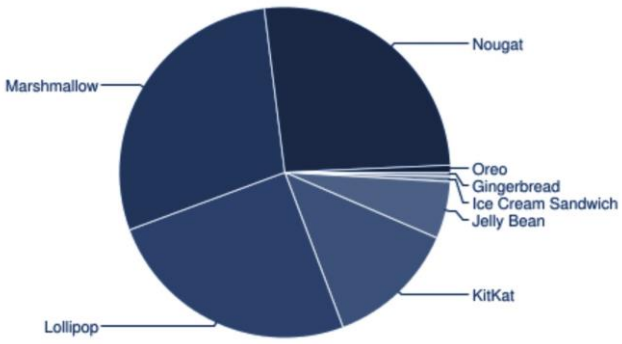
En el mercado actual de aplicaciones es bien cierto que en Android abundan las de entretenimiento y en concreto los videojuegos, no obstante, el público es muy amplio y en crecimiento. Además, muchas de las aplicaciones de este género tienen una cierta ausencia de calidad palpable, en nuestra mano estará intentar dar la mejor faceta al público de nuestra aplicación.

Situación actual

Herramientas utilizadas:

- ➔ Usaremos nuestro ordenador portátil personal para trabajar cuando estemos fuera del nuestro domicilio, y nuestro ordenador de sobremesa cuando estemos en el.
- ➔ Sistemas Operativos.
- ➔ A la hora de programar nuestro entorno de trabajo será Windows, en su última edición.
- ➔ Windows 10.
- ➔ Nuestra aplicación será sistemas Windows con una.
- ➔ Compatibilidad asegurada en sus versiones de Windows 7 y Windows 10.
- ➔ Y también para sistemas Android a partir de su API.
  - ◆ Esto la hará compatible con el 93,5% de dispositivos Android activos en el mundo.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.5%
4.1.x	Jelly Bean	16	1.9%
4.2.x		17	2.9%
4.3		18	0.8%
4.4	KitKat	19	12.8%
5.0	Lollipop	21	5.7%
5.1		22	19.4%
6.0	Marshmallow	23	28.6%
7.0	Nougat	24	21.1%
7.1		25	5.2%
8.0	Oreo	26	0.5%
8.1		27	0.2%



Datos oficiales de Google.

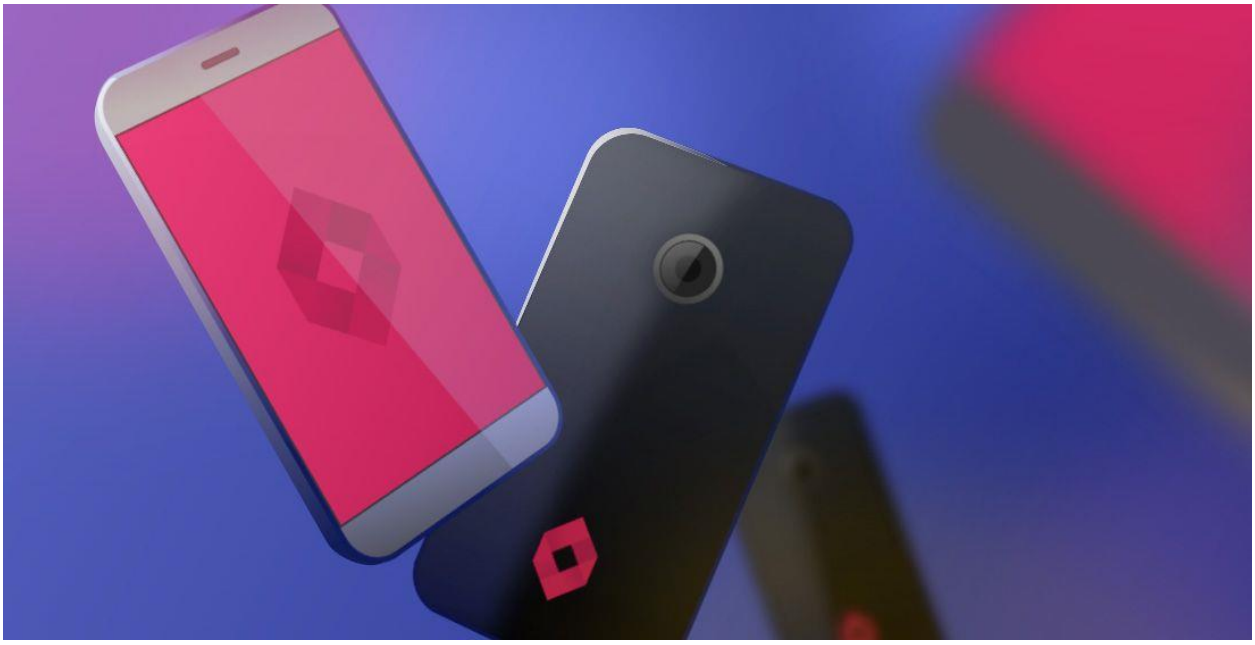
https://developer.android.com/about/dashboards

Lenguajes

- ➔ Java JDK 7+
- ➔ Android API.
- ➔ LibGDX o otra Librería en caso de verla más viable para nuestros objetivos.
- ➔ XML.

Software

- ➔ Android Studio IDE, con posibilidad de usar otro IDE en caso de ser requerido.
- ➔ Programas, ‘Open Source’ de diseño gráfico.



Desarrollo

Una vez empezamos con el desarrollo de nuestra aplicación decidimos utilizar como lenguaje de programación Java por su JVM y posible ampliación a otras plataformas, como IDE decidimos usar Eclipse y en cuanto programa de control de versiones acabamos usando Bitbucket, el cual nos permitía acceder de forma privada a el repositorio con nuestra aplicación.

-En esta parte del proyecto pasaremos explicando el funcionamiento de las clases de java y sus métodos más importantes.

División de las clases

- MainClass (main)
- Game
- KeyListenGame
- SaveDataManager
- SavePlayer
- SortedArrayList
- Player
- Ball
- AIPaddle
- Brick
- Token
- Point

MainClass

Importación de Librerías

```
package vFinal;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;
import javax.swing.SwingUtilities;
```

Clase

```
/**
 *
 * Clase que nos implementa el método main, extiende a JFrame para crear una
 * interfaz gráfica y implementa un ActionListener para la escucha de botones.
 * Declaramos una constante long como serialVersion, otra constante que nos
 * establezca el tamaño de ventana y otra para el alto. Como variables globales
 * de clase declaramos un Juego 'Game', los JPanel, JLabel, JButton, JTextArea
 * necesarios para la interfaz gráfica, un String que nos almacene el juego al
 * que estamos jugando, y un SaveDataManager para poder guardar la partida o la
 * puntuación.
 *
 * @author p.diaz
 *
 */
public class MainClass extends JFrame implements ActionListener {
```

Variables Globales a la Clase

```
	private static final long serialVersionUID = 2350704171326530882L;
	public static final int WIDTH = 800, HEIGHT = 600;
	private Game game;
	// Creamos panel con Layout centrada
	private JPanel menuPane, creditPane, howPane, statsPane;
	private String gameStage;
	private JButton startBtn, creditBtn, helpBtn, exitBtn, backBtn, backBtn2, backBtn3,
resumeBtn, statsBtn;
```

```
private JLabel titleLbl, namesLbl, howLbl, titleStatsLbl;
private JTextArea statsLbl;
private SaveDataManager sdc;
```

METODO MAIN

```
/**
 *
 * Método MAIN en el cual creamos el objeto de clase mediante un invokeLater de
 * SwingUtilities.
 *
 * @param args
 */
public static void main(String[] args) {
	SwingUtilities.invokeLater(new Runnable() {
		@Override
		public void run() {
			new MainClass();
		}
	});
}
```

Constructores

```
/**
 * Constructor de la clase con el JFrame. Primero le establecemos unos valores
 * de titulo de frame, tamaño, operación ce cerrado por defecto, localización y
 * visibilidad.
 *
 * Inicializamos el gameStage a 'pong' ya que es el juego del inicio. Creamos el
 * juego insertándole por parámetros el ancho, alto, y gameStage que necesita y
 * finalmente le añadimos a esta clase un KeyListener para los eventos del
 * teclado.
 *
 *
 */
public MainClass() {
	// GENERAL
	setTitle("The Final Game");
	setPreferredSize(new Dimension(WIDTH, HEIGHT));
	setMaximumSize(new Dimension(WIDTH, HEIGHT));
	setMinimumSize(new Dimension(WIDTH, HEIGHT));
	setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	setResizable(false);
	setLocationRelativeTo(null);
	setVisible(true);

	// ACCIONES PREVIAS A LA CREACIÓN DEL FRAME.
	File f = new File(".//savedataTFG.txt");
	sdc = new SaveDataManager(f, game);
	sdc.iniSortedSaveManager();
	gameStage = "pong";
	game = new Game(WIDTH, HEIGHT, gameStage, this, sdc);
	game.setPaused(true);
```

```
this.addKeyListener(new KeyListenGame(game, this));

/**
 * Declaramos unos estilos para usar luego en nuestra clase.
 */
// - ESTILOS -
int btnWith = 270;
int btnHeight = 90;
Font btnFont = new Font("Arial", Font.BOLD, 40);
Font creditFont = new Font("Arial", Font.BOLD, 21);
Color btnColorPrimary = new Color(2, 167, 238);

/**
 * Creamos el menú de inicio con un GridLayout de 5 filas y 1 columna. Le
 * asignamos un ancho de toda la pantalla y lo añadimos a el frame.
 */
/**
 * / MENU INICIO
 */
menuPane = new JPanel();
menuPane.setLayout(new GridLayout(5, 0));
menuPane.setPreferredSize(new Dimension(WIDTH, HEIGHT));
menuPane.setBackground(new Color(250, 250, 250));
add(menuPane);
// Botón START
startBtn = new JButton("Nueva Partida");
startBtn.setActionCommand("startBtn");
startBtn.addActionListener(this);
startBtn.setFocusable(false);
startBtn.setFont(btnFont);
startBtn.setPreferredSize(new Dimension(btnWith, btnHeight));
startBtn.setForeground(Color.WHITE);
startBtn.setBackground(btnColorPrimary);
startBtn.setAlignmentX(CENTER_ALIGNMENT);
menuPane.add(startBtn);
// Botón CONTINUAR
resumeBtn = new JButton("Continuar");
resumeBtn.setActionCommand("resumeBtn");
resumeBtn.addActionListener(this);
resumeBtn.setFocusable(false);
resumeBtn.setFont(btnFont);
resumeBtn.setPreferredSize(new Dimension(btnWith, btnHeight));
resumeBtn.setForeground(Color.WHITE);
resumeBtn.setBackground(btnColorPrimary);
resumeBtn.setAlignmentX(CENTER_ALIGNMENT);
// Se añade reemplazando a comenzar
// Botón Créditos
creditBtn = new JButton("Créditos");
creditBtn.setActionCommand("creditBtn");
creditBtn.addActionListener(this);
creditBtn.setFocusable(false);
creditBtn.setFont(btnFont);
creditBtn.setPreferredSize(new Dimension(btnWith, btnHeight));
creditBtn.setForeground(Color.WHITE);
creditBtn.setBackground(btnColorPrimary);
```

```
creditBtn.setAlignmentX(CENTER_ALIGNMENT);
menuPane.add(creditBtn);
// Botón Instrucciones
helpBtn = new JButton("Instrucciones");
helpBtn.setActionCommand("helpBtn");
helpBtn.addActionListener(this);
helpBtn.setFocusable(false);
helpBtn.setFont(btnFont);
helpBtn.setPreferredSize(new Dimension(btnWith, btnHeight));
helpBtn.setForeground(Color.WHITE);
helpBtn.setBackground(btnColorPrimary);
helpBtn.setAlignmentX(CENTER_ALIGNMENT);
menuPane.add(helpBtn);
// Botón Stats
statsBtn = new JButton("Puntuaciones");
statsBtn.setActionCommand("statsBtn");
statsBtn.addActionListener(this);
statsBtn.setFocusable(false);
statsBtn.setFont(btnFont);
statsBtn.setPreferredSize(new Dimension(btnWith, btnHeight));
statsBtn.setForeground(Color.WHITE);
statsBtn.setBackground(btnColorPrimary);
statsBtn.setAlignmentX(CENTER_ALIGNMENT);
menuPane.add(statsBtn);
// Botón Salir
exitBtn = new JButton("Salir");
exitBtn.setActionCommand("exitBtn");
exitBtn.addActionListener(this);
exitBtn.setFocusable(false);
exitBtn.setFont(btnFont);
exitBtn.setPreferredSize(new Dimension(btnWith, btnHeight));
exitBtn.setForeground(Color.WHITE);
exitBtn.setBackground(btnColorPrimary);
exitBtn.setAlignmentX(CENTER_ALIGNMENT);
menuPane.add(exitBtn);

/**
 * / JPANEL CREDITOS
 */
creditPane = new JPanel();
creditPane.setLayout(new BorderLayout());
creditPane.setBackground(new Color(250, 250, 250));
// Botón Volver
backBtn = new JButton("Volver");
backBtn.setActionCommand("backBtn");
backBtn.addActionListener(this);
backBtn.setFocusable(false);
backBtn.setFont(creditFont);
backBtn.setForeground(Color.WHITE);
backBtn.setBackground(btnColorPrimary);
backBtn.setPreferredSize(new Dimension(btnWith, btnHeight));
backBtn.setAlignmentX(CENTER_ALIGNMENT);
creditPane.add(backBtn, BorderLayout.SOUTH);
// Label título créditos
titleLbl = new JLabel("<html><br>Créditos</html>");
```

```
titleLbl.setFont(creditFont);
titleLbl.setHorizontalAlignment(JLabel.CENTER);
creditPane.add(titleLbl, BorderLayout.NORTH);
// Label nombres y créditos
namesLbl = new JLabel (
"<html>The Final Game<br>Proyecto Final de GFGS Desarrollo de Aplicaciones
Multiplataforma<br>Curso 2017/18<br>IES Doctor Fleming, Oviedo<br>13 de Junio
del 2018<hr><br>Carlos Ramón Ferreras Menéndez<br>Pablo Díaz Rubio</html>");
namesLbl.setFont(creditFont);
namesLbl.setHorizontalAlignment(JLabel.CENTER);
creditPane.add (namesLbl, BorderLayout.CENTER);

/**
 * / PANEL INSTRUCCIONES
 */
howPane = new JPanel ();
howPane.setLayout(new BorderLayout ());
howPane.setBackground(new Color (64, 196, 255));
howPane.setBackground(new Color (250, 250, 250));
// Label instrucciones
howLbl = new JLabel (
"<html><center><b>PONG</b></center><br>Devuelve la pelota para vencer a la
Inteligencia Artificial, ¿Serás capaz de derrotar a lo
imposible?.<hr><br><center><b>WALL-BREAK</b></center><br>Si has llegado a este
punto sabrás que no puedes parar ahora. Destruye toda la pared de ladrillos
para subir de nivel.<hr><br><center><b>SNAKE</b></center><br>Aumenta tu
puntuación hasta lo inalcanzable comiendo manzanas para crecer como serpiente.
¿Podrás hacerlo?.<hr><br></html> ");
howLbl.setFont(creditFont);
howLbl.setHorizontalAlignment(howLbl.CENTER);
howPane.add(howLbl, BorderLayout.CENTER);
// Botón volver
backBtn2 = new JButton("Volver");
backBtn2.setActionCommand("backBtn2");
backBtn2.addActionListener(this);
backBtn2.setFocusable(false);
backBtn2.setFont(creditFont);
backBtn2.setForeground(Color.WHITE);
backBtn2.setBackground(btnColorPrimary);
backBtn2.setPreferredSize(new Dimension(btnWith, btnHeight));
howPane.add(backBtn2, BorderLayout.SOUTH);

/**
 * /PANEL PUNTUACIONES
 */
statsPane = new JPanel();
statsPane.setLayout(new BorderLayout());
statsPane.setBackground(new Color(250, 250, 250));
// Label titulo puntuaciones
titleStatsLbl = new JLabel("<html><center><b>PUNTUACIONES The Final Game
</b></center></html>");
titleStatsLbl.setFont(creditFont);
titleStatsLbl.setHorizontalAlignment(titleStatsLbl.CENTER);
statsPane.add(titleStatsLbl, BorderLayout.NORTH);
// JTextArea con scroll para puntuaciones.
```

```
        statsLbl = new JTextArea("PLAYERNAME / GAME REACHED / POINTS \n");
        statsLbl.setFont(creditFont);
        JScrollPane scrollPane = new JScrollPane(statsLbl);
        scrollPane.setVerticalScrollBarPolicy(
        ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        statsPane.add(scrollPane, BorderLayout.CENTER);
        // Botón Volver
        backBtn3 = new JButton("Volver");
        backBtn3.setActionCommand("backBtn3");
        backBtn3.addActionListener(this);
        backBtn3.setFocusable(false);
        backBtn3.setFont(creditFont);
        backBtn3.setForeground(Color.WHITE);
        backBtn3.setBackground(btnColorPrimary);
        backBtn3.setPreferredSize(new Dimension(btnWith, btnHeight));
        statsPane.add(backBtn3, BorderLayout.SOUTH);

    }
```

Métodos

```
    /**
     * Método usado al pulsar la tecla de 'ESC' el cual nos elimina todos los
     * paneles que podamos tener en el frame y nos añade el panel principal del
     * menú.
     */
    public void addMenu() {
        this.remove(howPane);
        repaint();
        revalidate();
        this.remove(creditPane);
        repaint();
        revalidate();
        this.remove(statsPane);
        repaint();
        revalidate();
        this.remove(game);
        // añadimos el menú inicio
        this.add(menuPane);
        repaint();
        revalidate();
    }
```

Implementaciones

```
    @Override
    public void actionPerformed(ActionEvent e) {

        // Acción de botón START
        if (e.getActionCommand().equals("startBtn")) {
            // quitamos el menú
            this.remove(menuPane);
```

```
            // revalidamos el juego de inicio
            gameStage = "pong";
            // añadimos el juego al frame
            add(game);
            game.repaint();
            game.revalidate();
            startBtn.setText("Continuar");

        }

        // Acción de botón CRÉDITOS
        else if (e.getActionCommand().equals("creditBtn")) {
            this.remove(menuPane);
            repaint();
            revalidate();
            this.add(creditPane);
            repaint();
            revalidate();

        }
        // Acción de botón INSTRUCCIONES
        else if (e.getActionCommand().equals("helpBtn")) {

            this.remove(menuPane);
            repaint();
            revalidate();
            this.add(howPane);
            repaint();
            revalidate();

        }

        // Acción de botón SALIR
        else if (e.getActionCommand().equals("exitBtn")) {

            System.exit(1);

        }

        // Acción de botón VOLVER desde créditos
        else if (e.getActionCommand().equals("backBtn")) {

            this.remove(creditPane);
            repaint();
            revalidate();

            this.add(menuPane);
            repaint();
            revalidate();

        }
        // Acción de botón VOLVER desde instrucciones
        else if (e.getActionCommand().equals("backBtn2")) {

            this.remove(howPane);
```

```
                repaint();
                revalidate();

                this.add(menuPane);
                repaint();
                revalidate();

            }
            // Acción de botón PUNTUACIONES
            else if (e.getActionCommand().equals("statsBtn")) {

                this.remove(menuPane);
                repaint();
                revalidate();

                statsLbl.setText(sdc.leerTxtStats());

                this.add(statsPane);
                repaint();
                revalidate();

            }

            // Acción de botón VOLVER desde puntuaciones
            else if (e.getActionCommand().equals("backBtn3")) {

                this.remove(statsPane);
                repaint();
                revalidate();

                this.add(menuPane);
                repaint();
                revalidate();

            }

        }

}
```

Game

Importación de Librerías

```
package v3;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
```

Clase

```
/**
 *
 * Clase Game 'Juego' que es un Panel que se puede implementar en cualquier JFrame
 * gracias a que extiende JPanel, esta clase es la que nos implementa el método
 * run() el cual nos va a servir para hacer el bucle del juego el cual va a
 * estar corriendo en un thread aparte para no bloquear la pantalla.
 *
 * En esta clase declaramos un serialVersionUID, un ancho de Panel y un alto. Un
 * Thread para el bucle del juego, varias variables boolean de control
 * inicializadas a false: running, paused, gameStarted, statsSave, godMode.
 * Declaramos otra variable boolean 'gameOver' que nos va a servir para detectar
 * cuando entramos en estado gameOver.
 *
 * Para poder acceder a las características de los objetos de las clases
 * respectivas declaramos: 'Ball','Player','AIPaddle','Token'.
 *
 * Declaramos una variable que almacene la etapa del juego en forma de String
 * (gameStage).
 *
 * Declaramos un int para almacenar la velocidad del juego.
 *
 * Declaramos un ArrayList<Brick>bricks para guardar los ladrillos que generemos
 * en la partida. Así mismo declaramos una variable para guardar la cantidad de
 * ladrillos, un margen y una cantidad de columnas de ladrillos(wallsQuantity);
 *
 * Declaramos una variable contador 'cont' para hacer un debug cada 100 vueltas
 * de bucle.
 *
 * Declaramos otra variable 'brickCount' para poder asignar un ID específico a
 * cada ladrillo del wall break.
 *
 * @author p.diaz
 *
 */
public class Game extends JPanel implements Runnable {
```

Variables Globales a la Clase

```
	private static final long serialVersionUID = -8033673042210859245L;
	private int width, height;
	private Thread thread;
	private boolean running = false;
	private boolean paused = false;
	private boolean gameStarted = false;
	private boolean gameOver;
	private boolean statsSaving = false;
	private Ball ball;
	private Player player;
	private String GameStage, playerName;
	private Token token;
	private AIPaddle AI;
	private boolean godMode = false;
	private int velocidadJuego;
	private MainClass main;
	private SaveDataManager sdc;
	// wall
	private ArrayList<Brick> bricks;
	private int bricksQuantity, margin, wallsQuantity;
	private int cont = 0;
	private int brickCount;
	// SaveGame
	private int letraABC;
	private int letraActual;
	private char[] letras = new char[26];
	private char[] nombre = new char[10];
	private String NombreJugador = "";
	private boolean nombreGuardado = false;
	private boolean guardarNombre = true;
	private String underscore = "_";
	private char barrabaja = 95;
	private boolean ckGOverSave = false;
	private int DebugCont = 0;
	private double coeficienteDeVelocidad = 1.5;
```

Constructores

```
	/**
	 *
	 * En este constructor pedimos por parámetros tanto el ancho como el alto de
	 * panel y el 'gameStage' en el que empiece el juego.
	 *
	 *
	 * @param width
	 * @param height
	 * @param game
	 */
	public Game(int width, int height, String game, MainClass main, SaveDataManager sdc)
{

		this.setPreferredSize(new Dimension(width, height));
		this.width = width - 18;
		this.height = height - 40;
		this.GameStage = game;
```

```
		this.main = main;
		this.sdc = sdc;
		crearElementosDelJuego();
		crearNombreJugadorDraw();
		start();
	}
```

Métodos

```
	/**
	 * Método DEBUG para revisar errores cada x vueltas de bucle.
	 */
	private void DEBUGsysos(int vueltasDeBucle) {
		if (DebugCont > vueltasDeBucle) {
			// System.out.println("running: " + running);
			// System.out.println("paused: " + paused);
			System.out.println("gameOver: " + gameOver);
			System.out.println("gameStarted: " + gameStarted);
			System.out.println("statsSaving: " + statsSaving);
			// System.out.println("godMode: " + godMode);
			System.out.println("nombreGuardado: " + nombreGuardado);
			System.out.println("guardarNombre: " + guardarNombre);
			System.out.println("ckGOverSave: " + ckGOverSave);
			System.out.println("Nombre Jugador:" + NombreJugador);
			System.out.println(" ------------------------------------ ");
			DebugCont = 0;
		}

		DebugCont++;

	}

	/**
	 * Método usado para resetear todos los juegos a su estado inicial en caso de
	 * que se vuelva a echar otra partida.
	 */
	public void newGame() {

		getPlayer().setScore(0);
		getPlayer().resetSnake();
		getBall().newBall();
		crearWall();
		for (Brick brick : bricks) {
			brick.setBrickAlive(true);
		}
		setGameStage("pong");
		setGameOver(false);
		setPaused(false);
		setStatsSaving(false);
		setGuardarNombre(false);

	}
```

```
	/**
	 * Método que nos obtiene el nombre que haya sido dibujado en
	 * dibujaNombre(); y nos lo guarda en el documento de texto de puntuaciones
	 * junto a su puntuación y juego al que se ha llegado.
	 */
	private void getNombreDibujado() {
		NombreJugador = creaNombre();
		if (NombreJugador.contains("_")) {
			NombreJugador = NombreJugador.substring(0,
			NombreJugador.indexOf(barrabaja));
		}

		System.out.println("HE GUARDADO UN NOMBRE = [ " + NombreJugador + " ].");
		sdc.appendTxtStats(NombreJugador, GameStage, getPlayer().getScore());

		setGuardarNombre(false);
		setNombreGuardado(true);
		setStatsSaving(false);

	}

	/**
	 * si el juego ha caído en game over, necesitamos que este método ponga a true
	 * la variable booleana StatsSaving ya que cada vez que se pierda es necesario
	 * introducir un nombre para saber de quién son las stats que se han creado.
	 */
	private void checkGameOverState() {
		setStatsSaving(true);
		setGuardarNombre(false);
		setNombreGuardado(false);
		ckGOverSave = false;
	}
```

Método PaintComponent

```
	/**
	 * Sobreescritura del método paintComponent que va llamando a cada método draw
	 * de cada clase necesaria en el momento indicado
	 *
	 * @param g
	 */
	@Override
	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		g.setColor(new Color(30, 30, 30));
		g.fillRect(0, 0, width, height);
		g.setFont(new Font("Arial", Font.PLAIN, 24));
		g.setColor(new Color(197, 0, 0));
		g.drawString("Score: " + player.getScore(), width - 160, 48);
		if (godMode) {
			g.setColor(new Color(197, 0, 0));
			g.drawString("GODMODE", width - 160, 24);
		}
		if (paused) {
			g.setFont(new Font("Arial", Font.PLAIN, 16));
```

```
                g.setColor(new Color(197, 0, 0));
                g.drawString("PAUSE (pulsa 'espacio' para continuar)", (width
                / 2) - 150, height / 2);
        } else {
                if (gameOver) {
                        g.setFont(new Font("Arial", Font.PLAIN, 20));
                        g.setColor(new Color(197, 0, 0));
                        g.drawString("GAME OVER", (width / 2) - 60, height / 2);
                        if (statsSaving) {
                                if (!nombreGuardado) {
                                        dibujaNombre(g);
                                        if (guardarNombre) {
                                                getNombreDibujado();
                                        }
                                }
                        } else {
                                g.drawString("(pulsa 'escape' para salir al menú)", (width
                                / 2) - 150, (height / 2) + 24);
                                g.drawString("(pulsa 'intro' para jugar de nuevo)", (width
                                / 2) - 150, (height / 2) + 48);
                        }
                } else {
                        if (getGameStage().equals("snake")) {
                                token.draw(g);
                        } else if (getGameStage().equals("pong")) {
                                AI.draw(g);
                                ball.draw(g);
                        } else if (getGameStage().equals("wall")) {
                                for (Brick brick : bricks) {
                                        if (brick.isBrickAlive()) {
                                                brick.draw(g);
                                        }
                                }
                                ball.draw(g);
                        }
                        player.draw(g, GameStage);
                }
        }

        // ----------------------------------------------
        g.dispose();
}

/**
 * método que controla el gameStage del juego haciendo que según suben los
 * puntos pase de un juego a otro.
 */
private void controlPartida() {
        if (player.getScore() >= 3 && ball.getX() <= (getWidth() / 2)) {

                setGameStage("wall");
        }
        if (player.getScore() >= (3 + brickCount)) {
                setGameStage("snake");
        }
```

```
}

/**
 * método ejecutado en el @Override del run
 */
private void runVersion3() {
	// capturo el tiempo en nanosegundos en este instante de ejecución
	long tiempoOrigenRepintado = System.currentTimeMillis();
	long tiempoOrigenMovimiento = System.currentTimeMillis();
	long tiempoOrigenFrameCount = System.currentTimeMillis();
	// establezco el tiempo de espera para que se ejecute el run de nuevo.
	long tiempoEsperaRepintado = 0;
	long tiempoEsperaMovimiento = 0;
	long segundo = 1000L;
	long frames = 60;
	int framesPerSecond = 0;

	tiempoEsperaRepintado = segundo / frames;
	tiempoEsperaMovimiento = (long) ((segundo / frames) * coeficienteDeVelocidad);

	// comienzo el bucle infinito del juego.
	while (running) {
		// capturo el tiempo nada más comenzar el bucle
		long tiempoActualRepintado = System.currentTimeMillis();
		long tiempoActualMovimiento = System.currentTimeMillis();
		long tiempoActualFrameCount = System.currentTimeMillis();
		controlPartida();
/**
 * si la diferencia entre el tiempo actual y el tiempo de origen es mayor o
 * igual a el tiempo de espera repintado. Igualamos el tiempo actual a el tiempo
 * de origen para que vuelva a entrar a la siguiente vuelta de bucle.
 *
 * imprimimos 60 frames por segundo.
 *
 */
		if ((tiempoActualRepintado - tiempoOrigenRepintado) >=
		tiempoEsperaRepintado) {
			tiempoOrigenRepintado = tiempoActualRepintado;
			// método que imprime el juego
			repaintPanel();
			// aumentamos un frame por cada vez que entra aquí.
			framesPerSecond++;
		}
		if ((tiempoActualMovimiento - tiempoOrigenMovimiento) >=
		tiempoEsperaMovimiento) {
			tiempoOrigenMovimiento = tiempoActualMovimiento;
			// método que mueve el juego
			parametrosMovimiento();
		}
		if ((tiempoActualFrameCount - tiempoOrigenFrameCount) >= segundo) {
			tiempoOrigenFrameCount = tiempoActualFrameCount;
			// imprimimos por pantalla los frames que se redibujan en un
			segundo y los
			// restablecemos a 0
			System.out.println("FPS= [" + framesPerSecond + " ]");
```

```
                        framesPerSecond = 0;
                }
        }
        stop();
}

/**
 * método que permite que el juego se mueva de una forma u otra según el juego
 * en el que se esté jugando.
 */
private void parametrosMovimiento() {
        if (!paused) {
                if (!gameOver) {
                        if (!godMode) {
                                checkGameOver();
                        }
                        move();
                        if (getGameStage().equals("snake")) {
                                token.snakeColision();
                        }
                        if (getGameStage().equals("wall")) {
                                ball.checkPaddleAndBrickCollision(player, bricks);
                        }
                        if (getGameStage().equals("pong")) {
                                ball.checkPaddleCollision(player, AI);
                        }
                } else {
                        if (ckGOverSave) {
                                checkGameOverState();
                        }
                }
        }
}
```

Método Run

```
/**
 * METODO RUN. Método necesario para la creación del bucle del juego, ya que
 * crea un nuevo thread y dentro de él se mete en un bucle infinito para que
 * podamos llamar al método dibujar y al método mover. Esto genera la ilusión de
 * un videojuego pudiendo establecer los frames per second a el valor preferido
 * y permitiendo que en cada uno de esos frames se muestre una imagen distinta a
 * la anterior dando así una ilusión de movimiento.
 */
@Override
public void run() {
        runVersion3();
}

/**
 * Variables necesarias que se han de cumplir para que Game Over = true;
 */
public void checkGameOver() {
        if (getGameStage().equals("snake")) {
```

```
		if (player.getXsnake() < 0 || player.getXsnake() > getWidth()) {
				gameOver = true;
				ckGOverSave = true;
			}
		if (player.getYsnake() < 0 || player.getYsnake() > getHeight()) {
				gameOver = true;
				ckGOverSave = true;
			}
		if (player.snakeCollision()) {
				gameOver = true;
				ckGOverSave = true;
			}
		}
		if (getGameStage().equals("pong")) {
			if (ball.getX() < -10 || ball.getX() > width + 10) {
				gameOver = true;
				ckGOverSave = true;
			}
		}
		if (getGameStage().equals("wall")) {
			if (ball.getX() < -10) {
				gameOver = true;
				ckGOverSave = true;
			}
		}

	}

	/**
	 * Método que crea los Objetos que van a ser usados en el juego.
	 */
	public void crearElementosDelJuego() {
		gameOver = false;
		gameStarted = true;
		ball = new Ball(this.width, this.height);
		player = new Player(this.width, this.height);
		token = new Token(this.width, this.height, getPlayer());
		AI = new AIPaddle(player, ball, this.width, this.height);
		// wall
		crearWall();
		// thread Start
		velocidadJuego = 30;

	}

	/**
	 *
	 * Este método sirve para crear una pared de ladrillos que cubra toda la
	 * pantalla independientemente de lo alta que sea.
	 *
	 */
	public void crearWall() {

		bricks = new ArrayList<Brick>();
		bricksQuantity = (int) ((this.height - 10) / player.getPlayerHeight());
```

```
                margin = (int) ((this.height - (player.getPlayerHeight() *
                bricksQuantity)) / 2);
                wallsQuantity = 3;
                brickCount = 0;
                for (int i = 1; i <= wallsQuantity; i++) {
                        for (int j = 0; j < bricksQuantity; j++) {
                                bricks.add(new Brick(
                                player,
                                (this.width - 20 - (player.getPlayerWidth() * i)),
                                margin + (j * player.getPlayerHeight()),
                                brickCount));
                                brickCount++;
                        }
                }

        }

        /**
         * Método que nos llama a el método repaint() dentro de un nuevo thread de
         * SwingUtilities.invokeLater
         */
        private void repaintPanel() {
                SwingUtilities.invokeLater(new Runnable() {
                        @Override
                        public void run() {
                                repaint();
                        }
                });
        }

        /**
         * Método sincronizado que comienza el bucle del juego.
         */
        public synchronized void start() {
                thread = new Thread(this);
                thread.start();
                running = true;
        }

        /**
         * Método sincronizado que permite parar el thread del bucle del juego.
         */
        public synchronized void stop() {
                try {
                        thread.join();
                        running = false;
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
```

Método Move

```
        /**
         * Método que permite que haya movimiento de los objetos del juego.
```

```
* métodos move específicos de cada juego y en todos método move de player.
*/
public void move() {
	// -------------------------------------------
	if (getGameStage().equals("pong")) {
		AI.move();
		ball.move(GameStage, godMode);
	} else if (getGameStage().equals("wall")) {
		ball.move(GameStage, godMode);
	} else if (getGameStage().equals("snake")) {
	// No hay un movimiento específico de snake ya que va implementado en player.
	}
	player.move(GameStage);

	// -------------------------------------------
}

/**
 * Método que nos devuelve un String con la posición del cursor en el momento de
 * la creación del nombre.
 *
 * @return
 */
public String getLetraCambiando() {
	return new String("letraActual: " + letraActual + ", letraABC: " + letraABC);
}

/**
 *
 * Método que inicializa las variables necesarias para poder guardar el nombre
 * del jugador.
 *
 */
private void crearNombreJugadorDraw() {
	for (int i = 0; i < letras.length; i++) {
		letras[i] = (char) (65 + i);
	}
	letraABC = 0;
	letraActual = 0;
	for (int i = 0; i < nombre.length; i++) {
		nombre[i] = 95;
	}
}

/**
 * Método que recorre el array de char para elegir la letra deseada
 *
 * @param posLetra
 * @return
 */
public char dameLetra(int posLetra) {
	for (int i = 0; i < letras.length; i++) {
		if (posLetra == i) {
			return letras[i];
		}
```

```
			}
			return barrabaja;
		}

		/**
		 * Método que nos rellena el array de char 'nombre' para completar el nombre
		 * deseado
		 *
		 * @return String del array de char 'nombre'
		 */
		public String creaNombre() {
			for (int i = 0; i < nombre.length; i++) {
				if (i == getLetraActual()) {
					nombre[i] = dameLetra(getLetraABC());
					return new String(nombre);
				}
			}
			return new String(nombre);
		}

		/**
		 * Metodo que nos dibuja en pantalla las instrucciones para crear un nombre para
		 * guardar la partida y nos lo va mostrando segun lo generamos.
		 *
		 * @param g
		 */
		public void dibujaNombre(Graphics g) {
			g.setFont(new Font("Arial", Font.PLAIN, 16));
			g.setColor(new Color(197, 0, 0));
			g.drawString("Introduce tu nombre: (flecha arriba, abajo, izquierda, y
derecha)", (width / 2) - 120,
					(height / 2) + 24);
			g.drawString("Pulsa 'intro' para guardar", (width / 2) - 120, (height / 2) +
48);
			g.drawString(creaNombre(), (width / 2) - 100, (height / 2) + 72);
		}

		// GETTERS Y SETTERS
		public int getWidth() {			return width;}
		public void setWidth(int width) {			this.width = width; }
		public int getHeight() {		return height;		}
		public void setHeight(int height) {			this.height = height;		}
		public Thread getThread() {			return thread;		}
		public void setThread(Thread thread) {			this.thread = thread;		}
		public boolean isGameStarted() {		return gameStarted; }
		public void setGameStarted(boolean gameStarted) {this.gameStarted=gameStarted; }
		public Ball getBall() {			return ball; }
		public void setBall(Ball ball) {		this.ball = ball;	}
		public Player getPlayer() {			return player;		}
		public void setPlayer(Player player) {		this.player = player;		}
		public int getCont() {			return cont; }
		public void setCont(int cont) {			this.cont = cont;	}
		public boolean isRunning() {			return running;		}
		public String getGameStage() {			return GameStage;	}
		public void setGameStage(String gameStage) {		GameStage = gameStage;		}
```

```
public boolean isGameOver() {          return gameOver;    }
public void setGameOver(boolean gameOver) {   this.gameOver = gameOver; }
public Token getToken() { return token;}
public void setToken(Token token) {    this.token = token; }
public AIPaddle getAI() { return AI;   }
public void setAI(AIPaddle aI) { AI = aI;     }
public boolean isGodMode() {     return godMode;     }
public void setGodMode(boolean godMode) {     this.godMode = godMode;}
public ArrayList<Brick> getBricks() {  return bricks;}
public void setBricks(ArrayList<Brick> bricks) {     this.bricks = bricks;     }
public int getBricksQuantity() {       return bricksQuantity;    }
public void setBricksQuantity(int bricksQuantity){
       this.bricksQuantity=bricksQuantity;}
public int getMargin() {         return margin;      }
public void setMargin(int margin) {          this.margin = margin;     }
public int getWallsQuantity() {        return wallsQuantity;     }
public void setWallsQuantity(int wallsQuantity) {this.wallsQuantity = wallsQuantity;}
public int getBrickCount() {           return brickCount;  }
public void setBrickCount(int brickCount) {         this.brickCount = brickCount;}
public int getVelocidadJuego() {       return velocidadJuego;    }
public void setVelocidadJuego(int velocidadJuego) {
       this.velocidadJuego = velocidadJuego;  }
public synchronized boolean getPaused() {           return paused;      }
public synchronized void setPaused(boolean paused) {       this.paused = paused;}
public synchronized boolean getRunning() {          return running;     }
public synchronized void setRunning(boolean running) {this.running = running;  }
public boolean isStatsSaving() {       return statsSaving; }
public void setStatsSaving(boolean statsSaving) {this.statsSaving = statsSaving;}
public String getPlayerName() {        return playerName;  }
public void setPlayerName(String playerName) {this.playerName = playerName;    }
public MainClass getMain() {           return main; }
public void setMain(MainClass main) {         this.main = main;   }
public SaveDataManager getSdc() {      return sdc;  }
public void setSdc(SaveDataManager sdc) {           this.sdc = sdc;     }
public char[] getLetras() {            return letras;      }
public void setLetras(char[] letras) {        this.letras = letras;     }
public char[] getNombre() {            return nombre;      }
public void setNombre(char[] nombre) {        this.nombre = nombre;     }
public String getUnderscore() {        return underscore;  }
public void setUnderscore(String underscore) {this.underscore = underscore;}
public static long getSerialversionuid() {return serialVersionUID;      }
public String getNombreJugador() {return NombreJugador;    }
public void setNombreJugador(String nombreJugador) {NombreJugador = nombreJugador;}
public boolean isNombreGuardado() {    return nombreGuardado;    }
public void setNombreGuardado(boolean nombreGuardado) {
       this.nombreGuardado=nombreGuardado;    }
public boolean isGuardarNombre() {     return guardarNombre;     }
public void setGuardarNombre(boolean guardarNombre) {
       this.guardarNombre = guardarNombre;    }
public char getBarrabaja() {return barrabaja; }
public void setBarrabaja(char barrabaja) {this.barrabaja = barrabaja;   }
public int getLetraABC() {return letraABC;}
public void setLetraABC(int letraStart) {this.letraABC = letraStart;    }
public int getLetraActual() {return letraActual;    }
```

```
	public void setLetraActual(int letraActual) {this.letraActual=letraActual;}
	public boolean isCkGOverSave() { return ckGOverSave; }
	public void setCkGOverSave(boolean ckGOverSave) {this.ckGOverSave = ckGOverSave;}
	public int getDebugCont() {return DebugCont;  }
	public void setDebugCont(int debugCont) {DebugCont = debugCont;   }
	public double getCoeficienteDeVelocidad() {return coeficienteDeVelocidad;}
	public void setCoeficienteDeVelocidad(double coeficienteDeVelocidad){
		this.coeficienteDeVelocidad = coeficienteDeVelocidad;	}

	/**
	 *
	 * Método antiguo de introducción de nombre. Obsoleto por motivos de estética.
	 *
	 * @deprecated
	 */
	public void statsSave() {
		playerName = JOptionPane.showInputDialog(main, "Introduce tu Nick:",
"Anonimo");
		if (playerName.equalsIgnoreCase("")) {
			playerName = "Anonimo";
		}
		playerName = playerName.replace("\\", "");
		getPlayer().setName(playerName);
		sdc.appendTxtStats(getPlayer().getName(), getGameStage(),
getPlayer().getScore());
		statsSaving = true;
	}

}
```

KeyListenGame

Importación de Librerías

```
package v3;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
```

Clase

```
/**
 * Clase que implementa el evento de escucha de teclado de nuestro juego.
 *
 * Declaramos un Juego para poder acceder a las variables de Jugador y
 * establecer el movimiento del mismo con respecto a la tecla presionada.
 *
 * Declaramos un objeto Main para poder volver al menú principal en caso de que se
 * pulse la tecla apropiada.
 *
 * @author p.diaz
 *
 */
public class KeyListenGame implements KeyListener {
```

Variables Globales a la Clase

```
	private Game game;
	private MainClass main;
```

Constructores

```
	/**
	 * constructor por defecto con los valores de los parámetros igualados a las
	 * variables globales de clase.
	 *
	 * @param game
	 * @param main
	 */
	public KeyListenGame(Game game, MainClass main) {
		this.game = game;
		this.main = main;
	}
```

Implementaciones

```
	@Override
	public void keyPressed(KeyEvent e) {

		// CONTROLES SAVEDATA
		if (game.isStatsSaving()) {
			if (e.getKeyCode() == KeyEvent.VK_UP) {
```

```
                        if (game.getLetraABC() < 25) {
                                game.setLetraABC(game.getLetraABC() + 1);
                        }
                }
                if (e.getKeyCode() == KeyEvent.VK_DOWN) {
                        if (game.getLetraABC() > 0) {
                                game.setLetraABC(game.getLetraABC() - 1);
                        }
                }
                if (e.getKeyCode() == KeyEvent.VK_LEFT) {
                        if (game.getLetraActual() > 0) {
                                game.setLetraActual(game.getLetraActual() - 1);
                        }
                }
                if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
                        if (game.getLetraActual() < 9) {
                                game.setLetraActual(game.getLetraActual() + 1);
                        }
                }
                if (e.getKeyCode() == KeyEvent.VK_ENTER) {
        // establecemos guardarNombre a true para que entre en el método guardarNombre;
                        game.setGuardarNombre(true);
                //      game.setStatsSaving(false);
                //      System.out.println(game.getNombreJugador());

                }

        } else {

        /**
         * Si el juego es el pong o el wallBreak establecemos que las teclas del teclado
         * de 'arriba' y 'w' nos suben el personaje y viceversa con 'abajo' y 's'.
         */
                if (game.getGameStage().equals("pong")) {
                        if (e.getKeyCode() == KeyEvent.VK_W) {
                                game.getPlayer().setUpAccel(true);
                        }
                        if (e.getKeyCode() == KeyEvent.VK_S) {
                                game.getPlayer().setDownAccel(true);
                        }
                } else if (game.getGameStage().equals("wall")) {
                        if (e.getKeyCode() == KeyEvent.VK_W) {
                                game.getPlayer().setUpAccel(true);
                        }
                        if (e.getKeyCode() == KeyEvent.VK_S) {
                                game.getPlayer().setDownAccel(true);
                        }

                /**
                 * Si el juego es el snake: y la serpiente esta quieta solo se podrá dejar de
                 * mover cuando se pulse la tecla 'd'/'derecha'. En caso de que la serpiente
                 * esté en movimiento se usará 'w'/'arriba' para desplazarse hacia arriba,
                 * 's'/'abajo' para desplazarse hacia abajo, 'a'/'izquierda' para desplazarse
                 * hacia la izquierda y 'd'/'derecha' para desplazarse hacia la derecha.
                 */
```

```
		} else if (game.getGameStage().equals("snake")) {
			if (!game.getPlayer().isMoving()) {
// e.getKeyCode() == KeyEvent.VK_W || e.getKeyCode() == KeyEvent.VK_S ||
				if (e.getKeyCode() == KeyEvent.VK_D) {
					game.getPlayer().setMoving(true);
				}
			}

			if (e.getKeyCode() == KeyEvent.VK_W) {
				if (game.getPlayer().getyDir() != 1) {
					game.getPlayer().setyDir(-1);
					game.getPlayer().setxDir(0);
				}
			}
			if (e.getKeyCode() == KeyEvent.VK_S) {
				if (game.getPlayer().getyDir() != -1) {
					game.getPlayer().setyDir(1);
					game.getPlayer().setxDir(0);
				}
			}
			if (e.getKeyCode() == KeyEvent.VK_A) {
				if (game.getPlayer().getxDir() != 1) {
					game.getPlayer().setxDir(-1);
					game.getPlayer().setyDir(0);
				}
			}
			if (e.getKeyCode() == KeyEvent.VK_D) {
				if (game.getPlayer().getxDir() != -1) {
					game.getPlayer().setxDir(1);
					game.getPlayer().setyDir(0);
				}
			}
		}

		// reiniciar juego
		if (e.getKeyCode() == KeyEvent.VK_ENTER) {
			game.newGame();
		}
	}

	// salir juego
	if (e.getKeyCode() == KeyEvent.VK_ESCAPE) {
		game.setPaused(true);
		main.addMenu();

	}
	// pausar juego
	if (e.getKeyCode() == KeyEvent.VK_SPACE) {
		if (game.getPaused()) {
			game.setPaused(false);
		} else {
			game.setPaused(true);
		}

	}
```

```
/**
 * CHEATS
 *
 * Esta tecla 'F1' sirve para poder el juego en GODMODE y así poder probar las
 * características del mismo en modo desarrollador. Estas teclas 'F2','F3','F4'
 * sirven para cambiar la velocidad del juego. 'ESC' sirve para volver al menu.
 * 'SPACE' sirve para poner el juego en pausa sin volver al menu. 'ENTER' sirve
 * para reiniciar el juego 'F5' nos cambia de juego a pruebas. 'F6' nos cambia
 * al pong. 'F7' nos cambia al wallBreak. 'F8' nos cambia al snake.
 */
	// velocidad lenta
	if (e.getKeyCode() == KeyEvent.VK_F4) {
		game.setCoeficienteDeVelocidad(4);
	}
	//velocidad normal
	if (e.getKeyCode() == KeyEvent.VK_F3) {
		game.setCoeficienteDeVelocidad(1.5);
	}
	// velocidad rapida
	if (e.getKeyCode() == KeyEvent.VK_F2) {
		game.setCoeficienteDeVelocidad(0.1);
	}
	// godMode
	if (e.getKeyCode() == KeyEvent.VK_F1) {
		if (game.isGodMode()) {
			game.setGodMode(false);
		} else {
			game.setGodMode(true);
		}
	}

	// cambiar a pruebas
	if (e.getKeyCode() == KeyEvent.VK_F5) {
		game.setGameStage("pruebas");
	}
	// cambiar a pong
	if (e.getKeyCode() == KeyEvent.VK_F6) {
		game.setGameStage("pong");
	}
	// cambiar a wall
	if (e.getKeyCode() == KeyEvent.VK_F7) {
		game.setGameStage("wall");
	}

	// cambiar a snake
	if (e.getKeyCode() == KeyEvent.VK_F8) {
		game.setGameStage("snake");

	}
	if (e.getKeyCode() == KeyEvent.VK_F9) {
		game.start();
	}
	if (e.getKeyCode() == KeyEvent.VK_F10) {
		game.stop();
	}
```

```
	}

	@Override
	public void keyReleased(KeyEvent e) {
		if (game.getGameStage().equals("pong")) {
			if (e.getKeyCode() == KeyEvent.VK_W) {
				game.getPlayer().setUpAccel(false);
			}
			if (e.getKeyCode() == KeyEvent.VK_S) {
				game.getPlayer().setDownAccel(false);
			}
		} else if (game.getGameStage().equals("wall")) {
			if (e.getKeyCode() == KeyEvent.VK_W) {
				game.getPlayer().setUpAccel(false);
			}
			if (e.getKeyCode() == KeyEvent.VK_S) {
				game.getPlayer().setDownAccel(false);
			}
		}
	}

	@Override
	public void keyTyped(KeyEvent arg0) {   }

}
```

SaveDataManager

Importación de Librerías

```
package v3;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.TreeSet;
import com.sun.tools.javac.util.List;
```

Clase

```
/**
 *
 * Clase SaveDataManager que nos sirve para poder guardar las puntuaciones de
 * los jugadores en un archivo .txt, ordenarlas de mayor puntuación a menor y
 * mostrarlas en el JPanel de Puntuaciones.
 *
 * @author p.diaz
 */
public class SaveDataManager {
```

Variables Globales a la Clase

```
	private  File file;
	private  Game game;
	private String Cabecera = "PLAYERNAME / GAME REACHED / POINTS ;\n";
	private  ArrayList<SavePlayer> jugadores = new ArrayList<SavePlayer>();
	private  SortedArrayList<SavePlayer> jugadoresOrdenados = new
SortedArrayList<SavePlayer>();
```

Constructores

```
	/**
	 * Constructor con un parámetro de tipo File para saber cuál es el archivo del
	 * savedata y otro parámetro de tipo Game que nos sirve para coger los datos del
	 * jugador de la partida.
	 *
	 * @param file
	 * @param game
	 */
	public SaveDataManager(File file, Game game) {
		this.file = file;
		this.game = game;
```

```
		if (!file.exists()) {
			crearTxtStats();
		}

	}
```

Métodos

```
	/**
	 * Método que se lanza solo en caso de que no exista un savedata para crearlo e
	 * insertar la cabecera.
	 */
	public void crearTxtStats() {

		try {
			BufferedWriter bw = new BufferedWriter(new FileWriter(file));
			bw.write(Cabecera);
			bw.newLine();
			bw.flush();
			bw.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}


	/**
	 * Método que nos devuelve un String con los datos del .txt savedata.
	 *
	 * @return
	 */
	public  String leerTxtStats() {
		String txt = "";
		try {
			BufferedReader br = new BufferedReader(new FileReader(file));
			String linea = "";
			while ((linea = br.readLine()) != null) {
				if (!linea.equals("null")) {
					txt = txt + linea;
				}
			}
			txt = txt.replaceAll(";", ";\n");
			br.close();
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return txt;
	}


	/**
	 * Método que nos añade una nueva línea a nuestro savedata cogiendo el jugador,
	 * juego y puntos respectivos que se hayan conseguido antes de un gameover.
```

```
	 *
	 */
	public void appendTxtStats(String nombre, String juego, int puntos) {
		SavePlayer p = new SavePlayer(nombre,juego,puntos);
		jugadoresOrdenados.insertSorted(p);
		BufferedWriter bw;
		try {
			bw = new BufferedWriter(new FileWriter(file));
			bw.write(Cabecera);
			for (SavePlayer sp : jugadoresOrdenados) {
				bw.write(sp.toString());
			}
			bw.flush();
			bw.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	/**
	 * Método estático que nos sirve para ordenar los jugadores por puntuación y
	 * ordenarlos de mayor a menor.
	 */
	public void iniSortedSaveManager() {
		try {
			BufferedReader br = new BufferedReader(new FileReader(file));
			String linea = "";
			int cont = 0;
			while ((linea = br.readLine()) != null) {
				if (cont != 0) {
					SavePlayer sp = new SavePlayer(linea.split("/"));
					jugadores.add(sp);
				}
				cont++;
			}
			br.close();
			for (int i = 0; i < jugadores.size(); i++) {
				jugadoresOrdenados.insertSorted(jugadores.get(i));
			}
			BufferedWriter bw = new BufferedWriter(new FileWriter(file));
			bw.write(Cabecera);
			for (SavePlayer sp : jugadoresOrdenados) {
				bw.write(sp.toString());
			}
			bw.flush();
			bw.close();
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}

	}
}
```

SavePlayer

Importación de Librerías

```
package v3;
```

Clase

```
/**
 * Clase para ordenar los jugadores en relación a su puntuación. Nos almacena el
 * nombre, el juego y los puntos.
 *
 *
 * @author p.diaz
 *
 */
public class SavePlayer implements Comparable<SavePlayer> {
```

Variables Globales a la Clase

```
	private String nombre;
	private String game;
	private int puntos;
```

Constructores

```
	/**
	 * Constructor para meter automáticamente los valores mediante un split();
	 *
	 * @param campos
	 */
	public SavePlayer(String[] campos) {
		try {
			nombre = campos[0].replaceAll(" ", "");
			game = campos[1].replaceAll(" ", "");
			try {
				puntos = Integer.valueOf(campos[2].replaceAll(" ",
				"").replaceAll(";", ""));

			} catch (NumberFormatException e) {
				System.out.println(campos[2].replaceAll(" ", "").replaceAll(";",
				"") + " no se pudo Parsear");
				e.printStackTrace();
			}
		} catch (ArrayIndexOutOfBoundsException e) {
			e.printStackTrace();
		}
	}

	/**
	 *
	 * Constructor para meter a mano los valores.
	 *
```

```
	 * @param nombre
	 * @param game
	 * @param puntos
	 */
	public SavePlayer(String nombre, String game, int puntos) {
		super();
		this.nombre = nombre;
		this.game = game;
		this.puntos = puntos;
	}
```

Métodos

```
	@Override
	public String toString() {
		return nombre + " / " + game + " / " + puntos + " ;\n";
	}
	public String getNombre() {			return nombre;		}
	public void setNombre(String nombre) {		this.nombre = nombre;		}
	public String getGame() {		return game; }
	public void setGame(String game) {			this.game = game;	}
	public int getPuntos() {		return puntos;		}
	public void setPuntos(int puntos) {			this.puntos = puntos;		}

}
```

Implementaciones

```
	/**
	 * Método CompareTo necesario al implementar el Comparable <SavePlayer> en el
	 * cual comparamos por cantidad de puntos.
	 *
	 */
	@Override
	public int compareTo(SavePlayer o) {
		return o.getPuntos() - this.puntos;
	}
```

SortedArrayList

Importación de Librerías

```
package v3;

import java.util.ArrayList;
import java.util.Collections;
```

Clase

```
/**
 *
 * Clase Genérica que sobrescribe el ArrayList para que permita inserciones ordenadas.
 *
 * @author pablo
 *
 * @param <T>
 */
public class SortedArrayList<T> extends ArrayList<T> {
```

Métodos

```
 @SuppressWarnings("unchecked")
    public void insertSorted(T value) {
        add(value);
        Comparable<T> cmp = (Comparable<T>) value;
        for (int i = size()-1; i > 0 && cmp.compareTo(get(i-1)) < 0; i--)
            Collections.swap(this, i, i-1);
    }
}
```

Player

Importación de Librerías

```
package v3;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.util.ArrayList;
import java.util.List;
```

Clase

```
/**
 * Variables comunes a todos los juegos: ancho y alto de JPanel, Puntuacion,
 * Nombre del jugador.
 *
 * posición 'y' como doble ya que va a estar en movimiento y probablemente se
 * quede en una posición que no sea un numero entero.
 *
 * yVel: Numero aplicado para sumar o restar a la posición de 'y' y así darle
 * movimiento
 *
 * Booleanos upAccel y downAccel: sirven para determinar hacia qué lado se esta
 * moviendo nuestro jugador, ya sea hacia arriba o hacia abajo.
 *
 * Constante GRAVITY para que nuestra raqueta no se pare de golpe cuando dejamos
 * de apretar hacia arriba o hacia abajo. Multiplicando un numero por otro que
 * es menor que 1 conseguimos que nuestro valor que determina el movimiento se
 * vaya reduciendo poco a poco hasta llegar a cero.
 *
 * List<Point> snakePoints que va a guardar todos los puntos que conforman
 * nuestra serpiente y va a ser aumentado por cada vez que la serpiente
 * colisione con un token.
 *
 * xDir y yDir son las variables que determinan la dirección que tomara la
 * serpiente.
 *
 * Boolean isMoving determina si nuestra serpiente está en movimiento o esta
 * quieta.
 *
 * Boolean elongate nos sirve para aumentar el tamaño de la serpiente.
 *
 * Variable startSize es el tamaño de nuestra serpiente en un comienzo del
 * juego.
 *
 * Variables playerWidth y playerHeight nos dan el tamaño de nuestro jugador
 * "raqueta/paddle".
 *
 * @author pablo
 */
public class Player {
```

Variables Globales a la Clase

```
private int width, height;
private int score;
private String name;
private double y, yVel;
private boolean upAccel, downAccel;
private int x, xVel;
private final double GRAVITY = 0.94;
private List<Point> snakePoints;
private int xDir, yDir, countVel;
private boolean isMoving, elongate;
private final int startSize = 8, playerWidth = 20, playerHeight = 80;
```

Constructores

```
/**
 * Ya que tanto para el pong como para el wallBreak necesitamos una raqueta
 * usamos la misma.
 *
 * Inicializamos upAccel y downAccel a falso para que al principio nuestra
 * raqueta este quieta.
 *
 * establecemos la posición del jugador y la velocidad del mismo a 0 ya que en
 * un principio esta quieto.
 *
 * Inicializamos el List<> de nuestra serpiente, declaramos que la dirección
 * tanto de x como y es 0 para que continúe estática.
 * Inicializamos a 0 la variable de control de velocidad del juego.
 * Establecemos a false isMoving y elongate para que nuestra serpiente este quieta y
 * no crezca.
 * Añadimos los puntos iniciales al array de puntos de nuestra serpiente.
 *
 * @param width
 * @param height
 */
public Player(int width, int height) {
	// all
	this.width = width;
	this.height = height;

	// pong y wall
	upAccel = false;
	downAccel = false;
	y = (height / 2) - playerHeight;
	yVel = 0;
	x = 20;
	xVel = 0;

	// snake
	snakePoints = new ArrayList<Point>();
	xDir = 0;
	yDir = 0;
	countVel = 0;
	isMoving = false;
```

```
		elongate = false;

		snakePoints.add(new Point(x, (int) y));
		for (int i = 0; i < startSize; i++) {
			snakePoints.add(new Point(x, (int) (y - i * getPlayerWidth())));
		}

	}
```

Métodos

```
	/**
	 * Método que nos restablece nuestra serpiente a la posición inicial, establece
	 * que no se está moviendo y que no está creciendo. Además, nos restablece la
	 * cantidad de puntos que contiene a el valor inicial.
	 */
	public void resetSnake() {
		snakePoints = new ArrayList<Point>();
		xDir = 0;
		yDir = 0;

		isMoving = false;
		elongate = false;

		// le decimos donde esta colocada la cabeza.
		snakePoints.add(new Point(x, (int) y));
		// rellenamos el tamaño inicial de la serpiente
		for (int i = 0; i < startSize; i++) {
			snakePoints.add(new Point(x, (int) (y - i * getPlayerWidth())));
		}
	}

	/**
	 *
	 *
	 * Método que nos dibuja a nuestro jugador de una forma u otra con respecto a el
	 * juego en el que estemos jugando. Este método solo tiene valor gráfico en
	 * nuestro juego.
	 *
	 * @param g
	 * @param game
	 */
	public void draw(Graphics g, String game) {
		if (game.equals("pong")) {
			g.setColor(new Color(250, 250, 250));
			g.fillRect(x, (int) y, playerWidth, playerHeight);
		} else if (game.equals("wall")) {
			g.setColor(new Color(198, 150, 10));
			g.fillRect(x, (int) y, playerWidth, playerHeight);

		} else if (game.equals("snake")) {
			g.setColor(new Color(110, 150, 10));
			for (Point p : snakePoints) {
				g.fillRect(p.getX(), p.getY(), playerWidth, playerWidth);
```

```
                }
        }
}

/**
* Método que mueve nuestro jugador de forma distinta con respecto el juego que
* le pasemos por parámetro.
*
*
* dentro del juego pong solo necesitamos que nuestra raqueta se mueva de arriba
* a abajo asique solo hacemos que la y tenga cambios de valor.
*
* dentro del juego wallBreak solo necesitamos que nuestra raqueta se mueva de
* arriba a abajo asique solo hacemos que la y tenga cambios de valor.
*
* teniendo en cuenta que estos tres juegos son en realidad un único juego, no
* podemos dejar la velocidad de la serpiente con tan poco tiempo de reacción,
* así que, lo primero que hacemos es dividir la cantidad de veces que entra en
* el método move() para que así vaya a una velocidad asequible. Lo segundo que
* hacemos es detectar que nuestra serpiente se está moviendo o no para que
* cuando así sea le metamos las variables de los controles necesarios. Primero
* almacenamos en un Punto temporal los atributos de la cabeza de la
* serpiente(x,y,...). Lo segundo almacenamos en otro punto temporal los datos
* de la cola. Finalmente almacenamos en otro punto temporal un nuevo punto que
* es el que se va a desplazar la cabeza de forma que así parezca que hay
* movimiento. Ahora recorremos desde la cola hasta la cabeza todos los puntos
* de nuestra serpiente asignándole a cada uno de esos puntos el punto mas
* cercano a ellos por parte de la cabeza. Todo lo anterior solo nos movería la
* serpiente. En caso de hacerla crecer necesitamos que la variable boolean
* elongate sea true y así añadir al finar de nuestro List<Point>snakepoints la
* variable temporal que ya habíamos almacenado en un punto de nuestra cola de
* la serpiente. de esta forma cuando nuestra serpiente colisione con un token
* se habrá movido hacia la misma posición que estaba este token, pero como
* ahora nuestra lista de snakePoints es más larga, parecerá que la cola sigue
* en el mismo punto ya que ahí es donde se colocara nuestro último punto de la
* serpiente.
*
* @param game
*
*/

public void move(String game) {
        if (game.equals("pong")) {
                if (upAccel) {
                        yVel -= 2;
                } else if (downAccel) {
                        yVel += 2;
                } else if (!upAccel && !downAccel) {
                        yVel *= GRAVITY;
                }

                if (yVel >= 5) {
                        yVel = 5;
                } else if (yVel <= -5) {
                        yVel = -5;
```

```
                }
                y += yVel;
                if (y < 0)
                        y = 0;
                if (y > (height - playerHeight))
                        y = (height - playerHeight);
        } else if (game.equals("wall")) {
                if (upAccel) {
                        yVel -= 2;
                } else if (downAccel) {
                        yVel += 2;
                } else if (!upAccel && !downAccel) {
                        yVel *= GRAVITY;
                }
                if (yVel >= 5) {
                        yVel = 5;
                } else if (yVel <= -5) {
                        yVel = -5;
                }
                y += yVel;
                if (y < 0)
                        y = 0;
                if (y > (height - playerHeight))
                        y = (height - playerHeight);
        } else if (game.equals("snake")) {
                countVel++;
                // dividimos la velocidad a la mitad
                if (countVel > 2) {
                        if (isMoving) {
                                Point temp = snakePoints.get(0);
                                Point last = snakePoints.get(snakePoints.size() - 1);
                        // consigue el nuevo punto en el que se va a desplazar la cabeza
                                Point newStart = new Point(temp.getX() + xDir *
                                playerWidth, temp.getY() + yDir * playerWidth);
                                for (int i = snakePoints.size() - 1; i >= 1; i--) {
        // mueve desde la cola cada cuadrado parte de la serpiente a la posición donde
                        // se encontraba el cubo mas cercano suyo en dirección a la cabeza
                                        snakePoints.set(i, snakePoints.get(i - 1));
                                }
        // mueve la cabeza de la serpiente a donde se esté desplazando el usuario
                                snakePoints.set(0, newStart);
                                if (elongate) {
                                        snakePoints.add(last);
                                        elongate = false;
                                }
                        }
                        countVel = 0;
                }
        }
}

/**
 * Método que detecta cuando un punto de nuestra serpiente colisiona con otro
 * punto de sí misma.
 *
```

```
 * Para detectar esto simplemente comparamos la posición de la cabeza de nuestra
 * serpiente con la posición de todos los puntos de la misma. En caso de que
 * sean igual significa que la serpiente se ha chocado y el método devolverá
 * true.
 *
 * @return
 */
	public boolean snakeCollision() {
		int xHead = this.getXsnake();
		int yHead = this.getYsnake();
		for (int i = startSize; i < snakePoints.size(); i++) {
		  if (snakePoints.get(i).getX()==xHead && snakePoints.get(i).getY()==yHead) {
				return true;
			}
		}
		return false;
	}

	// PONG
	/**
	 * Creamos un rectángulo con la librería rectangle para que tengamos una raqueta
	 * más precisa y luego poder usar intersects();
	 *
	 * @return
	 */
	public Rectangle getRectanglePaddle() {
		return new Rectangle(getX(), (int) getY(), getPlayerWidth(),
		getPlayerHeight());
	}

	// --- GETTER Y SETTERS ---
	public double getY() {			return y;	}
	public void setY(double y) {			this.y = y;	}
	public double getyVel() {		return yVel; }
	public void setyVel(double yVel) {			this.yVel = yVel;	}
	public boolean isUpAccel() {			return upAccel;		}
	public void setUpAccel(boolean upAccel) {			this.upAccel = upAccel;	}
	public boolean isDownAccel() {			return downAccel;	}
	public void setDownAccel(boolean downAccel) {		this.downAccel = downAccel;}
	public int getX() {		return x;	}
	public void setX(int x) {		this.x = x;	}
	public int getxVel() {			return xVel; }
	public void setxVel(int xVel) {		this.xVel = xVel;	}
	public double getGRAVITY() {			return GRAVITY;		}
	// SNAKE
	public int getxDir() {			return xDir; }
	public void setxDir(int xDir) {		this.xDir = xDir;	}
	public int getyDir() {			return yDir; }
	public void setyDir(int yDir) {		this.yDir = yDir;	}
	public int getXsnake() {		return snakePoints.get(0).getX();	}
	public int getYsnake() {		return snakePoints.get(0).getY();	}
	public boolean isElongate() {			return elongate;	}
	public void setElongate(boolean elongate) {			this.elongate = elongate; }
	public boolean isMoving() {			return isMoving;	}
	public void setMoving(boolean isMoving) {			this.isMoving = isMoving; }
```

```
        public int getPlayerWidth() {           return playerWidth; }
        public int getPlayerHeight() {          return playerHeight;       }
        public int getScore() {          return score;}
        public void setScore(int score) {              this.score = score; }
        public int getWidth() {          return width;}
        public void setWidth(int width) {              this.width = width; }
        public int getHeight() {         return height;      }
        public void setHeight(int height) {            this.height = height;      }
        public String getName() {        return name; }
        public void setName(String name) {             this.name = name;   }
        public List<Point> getSnakePoints() {          return snakePoints; }
        public void setSnakePoints(List<Point> snakePoints) {this.snakePoints = snakePoints;}
        public int getCountVel() {       return countVel;    }
        public void setCountVel(int countVel) {        this.countVel = countVel; }
        public int getStartSize() {             return startSize;   }
}
```

Ball

Importación de Librerías

```
package v3;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.util.ArrayList;
```

Clase

```
/**
 * Clase Ball 'Pelota' que sirve como pelota en los juegos pong y wallBreak;
 *
 * Declaramos una posición (x,y) y una velocidad de x e y.
 *
 * Declaramos un ancho y alto de JPanel para saber por donde se puede mover la
 * pelota.
 *
 * @author p.diaz
 *
 */
public class Ball {
```

Variables Globales a la Clase

```
	private double xVel, yVel, x, y;
	private int width, height;
	private int diametro;
```

Constructores

```
	/**
	 * Constructor con parámetros ancho y alto para heredar sus propiedades y luego
	 * usarlas como delimitadores. En un principio la pelota empieza en el centro de
	 * nuestro JPanel y adopta una velocidad y dirección random para su movimiento.
	 *
	 * @param width
	 * @param height
	 */
	public Ball(int width, int height) {
		this.width = width;
		this.height = height;
		x = (width / 2) - 10;
		y = (height / 2) - 10;
		xVel = getRandomSpeed() * getRandomDirection();
		yVel = getRandomSpeed() * getRandomDirection();
		diametro = 20;

	}
```

```
/**
 * Metodo que sirve para dibujar la pelota.
 *
 * @param g
 */
public void draw(Graphics g) {
	g.setColor(new Color(255, 141, 0));
	g.fillOval((int) x, (int) y, 20, 20);
}

/**
 * Metodo que sirve para mover la pelota en el JPanel aplicandole el juego en el
 * que estemos para que adopte unas propiedades o otras. En caso del pong la
 * pared derecha o la del enemigo esta abierta en caso de que la IA no consiga
 * devolver la pelota(imposible) y en el wallbreak si choca en esta pared se
 * invierte la velocidad de su coordenada para que haya un rebote.
 *
 * En caso de que el parametro godmode sea true no hay posibilidad de que la
 * pelota se salga por ningun lado para imposibilitar así que el juego entre en
 * gameOver.
 *
 * @param game
 * @param godmode
 */
public void move(String game, boolean godmode) {
	if (game.equals("pong")) {
		x += xVel;
		y += yVel;

		if (y < 10) {
			yVel = -yVel;
		}
		if (y > height - 10) {
			yVel = -yVel;
		}
		if (godmode) {
			if (x < 10) {
				xVel = -xVel;
			}
		}

	} else if (game.equals("wall")) {
		x += xVel;
		y += yVel;

		if (y < 10) {
			yVel = -yVel;
		}
		if (y > height - 10) {
			yVel = -yVel;
		}
		if (x > width - 10) {
```

```
                        xVel = -xVel;
                }

                if (godmode) {
                        if (x < 10) {
                                xVel = -xVel;
                        }
                }

        } else if (game.equals("snake")) {
                x += xVel;
                y += yVel;

                yVel = 0;
                xVel = 0;
        } else if (game.equals("pruebas")) {
                x += xVel;
                y += yVel;

                if (y < 10) {
                        yVel = -yVel;
                }
                if (y > height - 10) {
                        yVel = -yVel;
                }
                if (x > width - 10) {
                        xVel = -xVel;
                }
                if (x < 10) {
                        xVel = -xVel;
                }
        }

}

/**
 * Método que devuelve true por cada vez que la pelota cruza el centro de
 * nuestra pantalla.
 *
 * @return
 */
public boolean crossCenter() {
        if (x >= (width / 2) - 10 && x <= (width / 2) + 10) {
                return true;
        }
        return false;
}

/**
 * Método que comprueba cuando la pelota colisiona con una raqueta para así
 * invertir su velocidad y se muestre como un rebote. En este método se pasan
 * por parámetro tanto el jugador como la raqueta de la AI para así poder
 * detectar ambos.
 *
 * Creamos un rectángulo que almacene el rectángulo de la pelota, otro que
```

```
	 * almacene el de la raqueta del jugador, este lo dividimos en dos partes,
	 * superior e inferior. Cada una de estas partes devuelve la dirección de x
	 * invertida, pero la dirección de y dependerá de que rectángulo colisione:
en
	 * el caso de que sea el superior y la velocidad de desplazamiento de la y sea
	 * positiva nos la invertirá si no no. En el caso de que la colisión sea con el
	 * rectángulo inferior y la velocidad de desplazamiento de la variable y de la
	 * pelota sea negativa, invertimos nuevamente su valor. esto nos da mas
	 * oportunidades a la hora de jugar y más flexibilidad en cuanto a la hora de
	 * devolver la pelota con la raqueta.
	 *
	 * En el caso de la raqueta de la AI, simplemente, invertimos la dirección de la
	 * coordenada x de la pelota en el momento de que esta colisione con el frontal
	 * de la raqueta. En ese mismo instante añadimos un punto a el jugador para que
	 * el juego pueda avanzar.
	 *
	 * @param player
	 * @param AI
	 */
	public void checkPaddleCollision(Player player, AIPaddle AI) {

		// RECTANGLE
		Rectangle rBall = getRectangleBall();
		Rectangle rPaddle = player.getRectanglePaddle();
		Rectangle rPaddleSup = new Rectangle((int) rPaddle.getX(),
		 (int) rPaddle.getY(), (int) rPaddle.getWidth(),
		 (int) rPaddle.getHeight() / 2);
		Rectangle rPaddleInf = new Rectangle((int) rPaddle.getX(),
		 (int) rPaddle.getCenterY(), (int) rPaddle.getWidth(),
		 (int) rPaddle.getHeight() / 2);
		if (rBall.intersects(rPaddle)) {
			if (rBall.intersects(rPaddleSup)) {
				if (yVel > 0) {
					yVel = -yVel;
				}
			}
			if (rBall.intersects(rPaddleInf)) {
				if (yVel < 0) {
					yVel = -yVel;
				}
			}
			xVel = -xVel;
		}

		// FRONTAL AI
		if (x >= width - (player.getPlayerWidth() + 20 + 10)) {
			if (y >= AI.getY() && y <= AI.getY() + player.getPlayerHeight()) {
				xVel = -xVel;
				player.setScore(player.getScore() + 1);
			}
		}
	}

	/**
	 * Método que detecta cuando la pelota choca con un ladrillo del Array de
```

```
	 * ladrillos o con el jugador.
	 *
	 * @param player
	 * @param bricks
	 */
	public void checkPaddleAndBrickCollision(Player player, ArrayList<Brick> bricks) {
		// Antes reconocíamos simplemente la colisión sin tener en cuenta de donde
		viene la pelota y donde golpea de nuestro paddle
//		if (x <= player.getPlayerWidth() + 20 + 10) {
//			if (y >= player.getY() && y <= player.getY() + player.getPlayerHeight())
{
//				xVel = -xVel;
//			}
//		}

		// RECTANGLE
		Rectangle rBall = getRectangleBall();
		Rectangle rPaddle = player.getRectanglePaddle();
		Rectangle rPaddleSup = new Rectangle((int) rPaddle.getX(), (int)
rPaddle.getY(), (int) rPaddle.getWidth(),
				(int) rPaddle.getHeight() / 2);
		Rectangle rPaddleInf = new Rectangle((int) rPaddle.getX(), (int)
rPaddle.getCenterY(), (int) rPaddle.getWidth(),
				(int) rPaddle.getHeight() / 2);
		if (rBall.intersects(rPaddle)) {
			if (rBall.intersects(rPaddleSup)) {
				if (yVel > 0) {
					yVel = -yVel;
				}
			}
			if (rBall.intersects(rPaddleInf)) {
				if (yVel < 0) {
					yVel = -yVel;
				}
			}
			xVel = -xVel;
		}



		for (int j = 0; j < bricks.size(); j++) {
			bricks.get(j).brickColision(this);
			if (!bricks.get(j).isBrickAlive()) {
				bricks.remove(j);
				break;
			}
		}
	}

	/**
	 * Método que genera una nueva pelota con una nueva dirección y velocidad.
	 */
	public void newBall() {

		x = (width / 2) - 10;
```

```
		y = (height / 2) - 10;
		xVel = getRandomSpeed() * getRandomDirection();
		yVel = getRandomSpeed() * getRandomDirection();
	}

	/**
	 * Método para generar una velocidad randomizada entre unos valores seguros.
	 *
	 * @return
	 */
	public double getRandomSpeed() {
		return (Math.random() * 5 + 2);
	}

	/**
	 * Método que nos genera una dirección aleatoria, ya sea 1 o -1 para poder
	 * multiplicarla por la velocidad y así nos de una velocidad con una dirección
	 * aleatoria. Si esto no se hiciese la velocidad siempre sería positiva, por lo
	 * cual nunca tendríamos una pelota que sale hacia la izquierda.
	 *
	 * @return
	 */
	public int getRandomDirection() {
		int rand = (int) (Math.random() * 2);
		if (rand == 1) {
			return 1;
		} else
			return -1;
	}


	//GETTERS Y SETTERS
	public double getxVel() { return xVel; }
	public int getWidth() {   return width;}
	public void setWidth(int width) {      this.width = width; }
	public int getHeight() {  return height;}
	public void setHeight(int height) {    this.height = height;     }
	public int getDiametro() {       return diametro;    }
	public void setDiametro(int diametro) {       this.diametro = diametro; }
	public void setxVel(double xVel) {            this.xVel = xVel;   }
	public double getyVel() {        return yVel; }
	public void setyVel(double yVel) {            this.yVel = yVel;   }
	public double getX() {           return x;    }
	public void setX(double x) {           this.x = x;  }
	public double getY() {           return y;    }
	public void setY(double y) {           this.y = y;  }
	public Rectangle getRectangleBall() {
		return new Rectangle((int) getX(), (int) getY(), diametro, diametro);
	}

}
```

AIPaddle

Importación de Librerías

```
package v3;

import java.awt.Color;
import java.awt.Graphics;
```

Clase

```
/**
 * Clase AIPaddle 'Raqueta Inteligencia Artificial' que nos crea un enemigo
 * invencible en el juego pong.
 *
 * Declaramos un ancho, alto de JPanel, una posición x,y y tan solo una
 * velocidad de y ya que solamente se va a mover por el eje y. En verdad no
 * haría falta declarar la mayoría de estas variables, pero están pensadas para
 * futuras implementaciones del juego en las cuales nuestra Inteligencia
 * artificial mejore y deje de ser invencible. Declaramos una constante Gravedad
 * por el mismo motivo.
 *
 * Declaramos una pelota para poder acceder a sus propiedades y un jugador por
 * el mismo motivo.
 *
 * @author p.diaz
 *
 */
public class AIPaddle {
```

Variables Globales a la Clase

```
	private int width, height;
	private double y, yVel;
	private boolean upAccel, downAccel;
	private int x;
	private final double GRAVITY = 0.94;
	private Ball ball;
	private Player player;
```

Constructores

```
	/**
	 * Constructor que nos iguala a las variables globales de clase los parámetros
	 * de Jugador, Pelota, Ancho y Alto.
	 *
	 * Establecemos la aceleración vertical a falso(upAccel, downAccel).
	 *
	 * Inicializamos la raqueta AI en su valor y a el centro de la pantalla y en su
	 * posición x a la derecha de la pantalla menos el margen de seguridad y el
	 * tamaño de ancho de la raqueta.
	 *
	 * @param player
```

```
     * @param ball
     * @param width
     * @param height
     */
    public AIPaddle(Player player, Ball ball, int width, int height) {
           this.player = player;
           this.width = width;
           this.height = height;
           this.ball = ball;
           upAccel = false;
           downAccel = false;
           y = (int) this.height / 2;
           yVel = 0;
           x = width - player.getPlayerWidth() - 20;
    }
```

Métodos

```
    /**
     * Método de dibujado de la raqueta AI.
     *
     * @param g
     */
    public void draw(Graphics g) {
           g.setColor(new Color(194, 32, 19));
           g.fillRect(x, (int) y, player.getPlayerWidth(), player.getPlayerHeight());
    }

    /**
     * Método de movimiento de la raqueta AI. Solamente iguala el centro de la
     * raqueta a el valor y de la pelota, por lo tanto siempre va a devolver la
     * pelota
     */
    public void move() {

           y = ball.getY() - 40;

           if (y < 0)
                  y = 0;
           if (y > height - player.getPlayerHeight())
                  y = height - player.getPlayerHeight();
    }


    //GETTERS Y SETTERS
    public int getY() {        return (int) y;     }
    public int getWidth() {          return width;}
    public void setWidth(int width) {       this.width = width; }
    public int getHeight() {  return height;      }
    public void setHeight(int height) {     this.height = height;      }
    public double getyVel() {        return yVel; }
    public void setyVel(double yVel) {      this.yVel = yVel;   }
    public boolean isUpAccel() {     return upAccel;     }
    public void setUpAccel(boolean upAccel) {     this.upAccel = upAccel;    }
    public boolean isDownAccel() {          return downAccel;   }
```

```
    public void setDownAccel(boolean downAccel) { this.downAccel = downAccel;}
    public int getX() {       return x;     }
    public void setX(int x) {        this.x = x;  }
    public Ball getBall() {          return ball; }
    public void setBall(Ball ball) {       this.ball = ball;   }
    public Player getPlayer() {            return player;      }
    public void setPlayer(Player player) {       this.player = player;      }
    public double getGRAVITY() {           return GRAVITY;     }
    public void setY(double y) {           this.y = y;  }

}
```

Brick

Importación de Librerías

```
package v3;

import java.awt.Color;
import java.awt.Graphics;
import java.util.Random;
```

Clase

```
/**
 * Clase Brick del juego wallBreak. Esta clase nos permite crear ladrillos con
 * diferentes características en nuestro juego.
 *
 * Declaramos un ancho, alto de JPanel, además de una posición x, y en la cual va
 * a estar colocado nuestro ladrillo. Finalmente le damos un ID para poder
 * identificar a el ladrillo.
 *
 * Declaramos un Jugador para poder sacar características del mismo, un Random
 * para nuestro segundo constructor el cual nos da un color aleatorio para hacer
 * un degradado con los ladrillos. Además, en un futuro puede que se implemente
 * colocación aleatoria de ladrillos en la pantalla.
 *
 * Declaramos un boolean brickAlive para establecer un estado de ladrillo (vivo o
 * muerto). Declaramos variables para hacer el degradado de los
 * ladrillos(sumaColor).
 *
 * @author p.diaz
 *
 */
public class Brick {
```

Variables Globales a la Clase

```
	private int width, height, x, y, ID;
	private Player player;
	private Random r;
	private boolean brickAlive;
	private int sumaColor, sumaColor2;
```

Constructores

```
	/**
	 * Constructor sobrecargado en desuso para futuras implementaciones del juego.
	 *
	 * @param player
	 * @param width
	 * @param height
	 * @param x
	 * @param y
	 */
```

```
public Brick(Player player, int width, int height, int x, int y) {
	this.width = width;
	this.height = height;
	this.x = x;
	this.y = y;
	this.player = player;
	// this.ball = ball;
	r = new Random();
	brickAlive = true;
}

/**
 * Constructor de ladrillos que le pasamos por parámetro la posición en la que
 * va a estar colocado nuestro ladrillo, un jugador para obtener sus
 * características y un ID para identificar nuestro ladrillo.
 *
 * @param player
 * @param x
 * @param y
 * @param ID
 */
public Brick(Player player, int x, int y, int ID) {
	this.x = x;
	this.y = y;
	this.player = player;
	// this.ball = ball;
	this.ID = ID;
	r = new Random();
	sumaColor = r.nextInt(50);
	sumaColor2 = r.nextInt(50) + 50;
	brickAlive = true;
}
```

Métodos

```
/**
 * Método de dibujado de cada ladrillo en relación a su ID.
 *
 * @param g
 */
public void draw(Graphics g) {
	// g.setColor(new
	// Color(r.nextInt(150)+100,r.nextInt(150)+100,r.nextInt(150)+100));
	g.setColor(new Color(50, 50, 50));
	g.fillRect(x, (int) y, player.getPlayerWidth(), player.getPlayerHeight());
	if (ID < 6) {
	  g.setColor(new Color(150 + sumaColor2, 70 + sumaColor2, 50 + sumaColor2));
	} else if (ID < 12) {
		g.setColor(new Color(150 + sumaColor, 70 + sumaColor, 50 + sumaColor));
	} else if (ID < 18) {
		g.setColor(new Color(150, 70, 50));
	}
		g.fillRect(x + 1, ((int) y) + 1, player.getPlayerWidth() - 2,
		player.getPlayerHeight() - 2);
```

```
	}

	/**
	 * Metodo que le pasamos por parametro una pelota para determinar si la posición
	 * de esta colisiona con la posición de alguno de nuestros ladrillos.
	 *
	 * en caso de que así sea establecemos el ladrillo como no vivo, la pelota rebota, y
la puntuacion del jugador aumenta.
	 *
	 * ademas devolvemos true en caso de que exista colisión.
	 *
	 * @param ball
	 * @return
	 */
	public boolean brickColision(Ball ball) {

if (ball.getX() + 10 >= this.x && ball.getX() + 10 <=(this.x +  player.getPlayerWidth())) {
	if (ball.getY() >= this.y && ball.getY() <=    (this.y + player.getPlayerHeight())) {
				setBrickAlive(false);
				ball.setxVel(ball.getxVel() * -1);
				player.setScore(player.getScore() + 1);
				return true;
			}
		}
		setBrickAlive(true);
		return false;
	}

	// GETTER Y SETTERS
	public boolean isBrickAlive() {		return brickAlive;  }
	public void setBrickAlive(boolean brickAlive) {this.brickAlive = brickAlive;   }
	public int getX() {		return x;    }
	public void setX(int x) {		this.x = x;  }
	public int getY() {		return y;    }
	public void setY(int y) {		this.y = y;  }
	@Override
	public String toString() {
		return "Point [x=" + x + ", y=" + y + "]";
	}
	public void imprimir() {
		System.out.println("Point [x=" + x + ", y=" + y + "]");
	}
	public int getWidth() {		return width;}
	public void setWidth(int width) {		this.width = width; }
	public int getHeight() {		return height;	}
	public void setHeight(int height) {		this.height = height;	}
}
```

Token

Importación de Librerías

```
package v3;

import java.awt.Color;
import java.awt.Graphics;
```

Clase

```
/**
 * Clase "Manzana" para el juego Snake.
 * @author pablo
 */
public class Token {
```

Variables Globales a la Clase

```
	/**
	 * Declaramos variables de posición x,y para colocar el token Declaramos un
	 * jugador que es el que interactúa con los tokens pasando por encima de ellos
	 * Declaramos un ancho y un alto heredado de la ventana del JPanel Declaramos un
	 * tamaño de token
	 */
	private int x, y;
	private Player snake;
	private int width, height;
	private int sizeToken;
```

Constructores

```
	/**
	 * Creamos un constructor parametrizado por un ancho, alto y jugador para
	 * heredar sus propiedades y usarlas en relación a nuestro token
	 *
	 * Inicializamos x,y a 20 y llamamos al método changePosition()
	 *
	 * Inicializamos el tamaño de nuestro token a un 10% mas grande que el ancho de
	 * nuestro jugador.
	 *
	 * @param width
	 * @param height
	 * @param s
	 */
	public Token(int width, int height, Player s) {
		this.width = width;
		this.height = height;
		x = 20;
		y = 20;
		snake = s;
		changePosition();
	sizeToken=(int)(snake.getPlayerWidth()+(snake.getPlayerWidth()*0.1));

	}
```

Métodos

```
	/**
	 * Método que asigna un valor aleatorio tanto a x como a y delimitado por el
	 * ancho y el alto de nuestro JPanel, además aumenta el valor de la misma
	 * variable hasta que esta sea divisible por el ancho del jugador dando un resto de
0.
	 */
	public void changePosition() {
		x = (int) (Math.random() * (width - snake.getPlayerWidth()));
		while (x % snake.getPlayerWidth() != 0)
			x++;
		y = (int) (Math.random() * (height - snake.getPlayerWidth()));
		while (y % snake.getPlayerWidth() != 0)
			y++;
	}

	/**
	 * Método de dibujado de nuestro token.
	 * @param g
	 */
	public void draw(Graphics g) {
		g.setColor(new Color(159, 20, 20));
		g.fillRect(x, y, sizeToken, sizeToken);
	}

	/**
	 * Método que devuelve true cuando nuestro jugador colisiona con el token
	 * @return
	 */
	public boolean snakeColision() {
		int snakeX = snake.getXsnake() + 2;
		int snakeY = snake.getYsnake() + 2;


		if (snakeX >= x - 1 && snakeX <= (x + sizeToken + 1))
			if (snakeY >= y - 1 && snakeY <= (y + sizeToken + 1)) {
				changePosition();
				snake.setScore(snake.getScore() + 1);
				snake.setElongate(true);
				return true;
			}
		return false;

	}

}
```

Point

Importación de Librerías

```
package v3;
```

Clase

```
/**
 * Clase Point 'Punto' que sirve para darle valores a un punto de la serpiente.
 *
 * Declaramos una posición x,y ya que es lo único que va a ser necesitado por cada punto.
 *
 * @author p.diaz
 *
 */
public class Point {
```

Variables Globales a la Clase

```
	private int x, y;
```

Constructores

```
	/**
	 * Constructor que nos inicializa la posición del punto a 0.
	 */
	public Point() {
		x = 0;
		y = 0;
	}

	/**
	 * Constructor que nos permite inicializar los puntos x,y en el valor deseado.
	 * @param x
	 * @param y
	 */
	public Point(int x, int y) {
		this.x = x;
		this.y = y;
	}
```

Métodos

```
	public int getX() {		return x;	}
	public void setX(int x) {		this.x = x;	}
	public int getY() {		return y;	}
	public void setY(int y) {		this.y = y;	}
	@Override
	public String toString() {return "Point [x=" + x + ", y=" + y + "]";	}
	public void imprimir() {	System.out.println("Point [x=" + x + ", y=" + y + "]");}
}
```

Funcionamiento

Instalación

Al ser un .jar ejecutable para poder jugar habría que descargarse java de la web y una vez hecho esto nuestro juego ya estaría listo para ser jugado.

Para descargar java solo hay que ir a https://java.com/es/download/.

Instrucciones

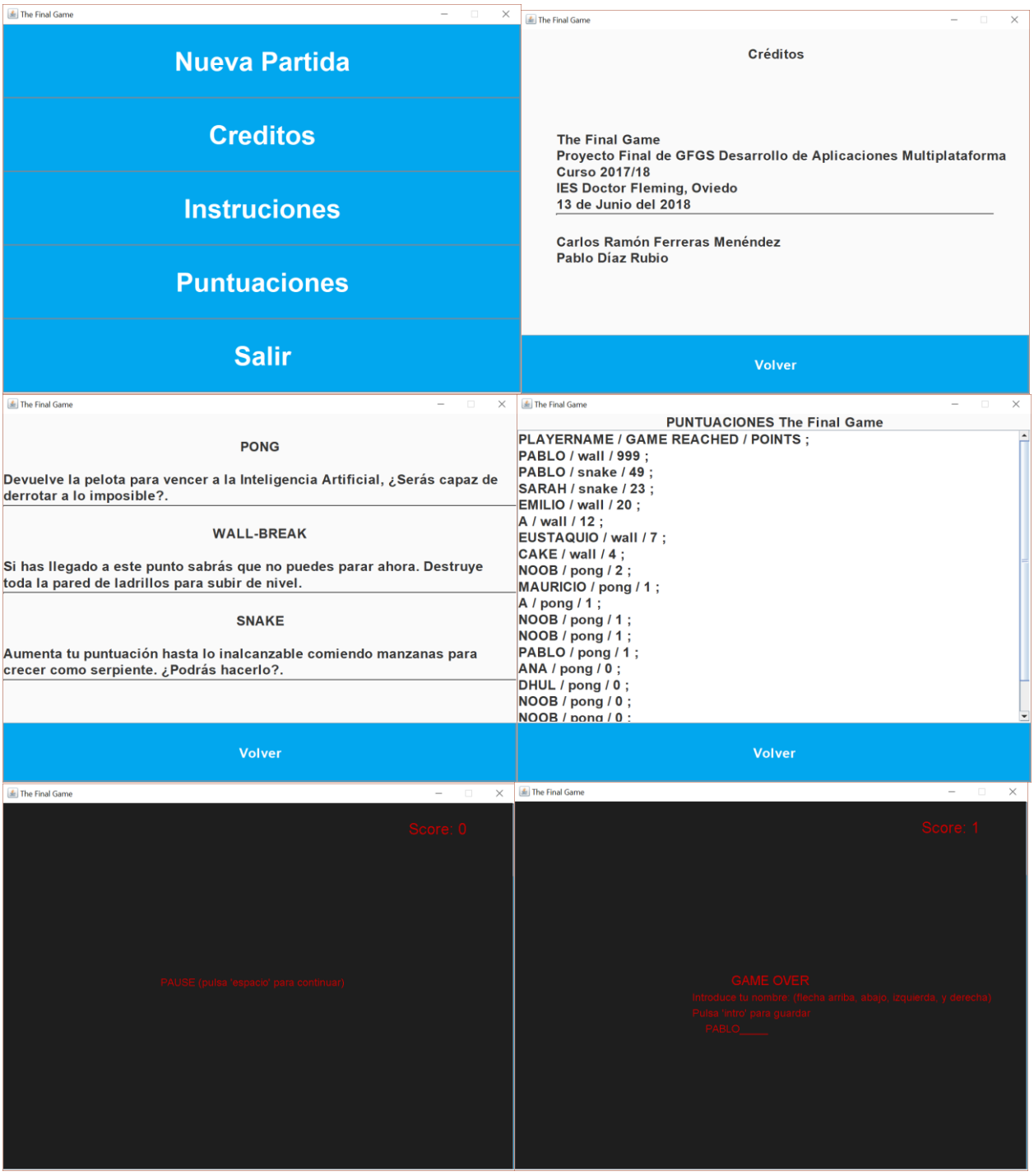
Si estas en el ‘pong’ o el ‘wallBreak’ para moverte solo tienes que usar las teclas ‘W’(arriba) y ‘S’(abajo). En el caso del ‘snake’ para moverte tienes que utilizar también la ‘A’(izquierda) y la ‘D’(derecha).
Para pausar el juego hay que pulsar la tecla ‘ESPACIO’, y para jugar un nuevo juego hay que pulsar ‘INTRO’.
Si lo que estás buscando es el GODMODE solo tienes que pulsar ‘F1’.

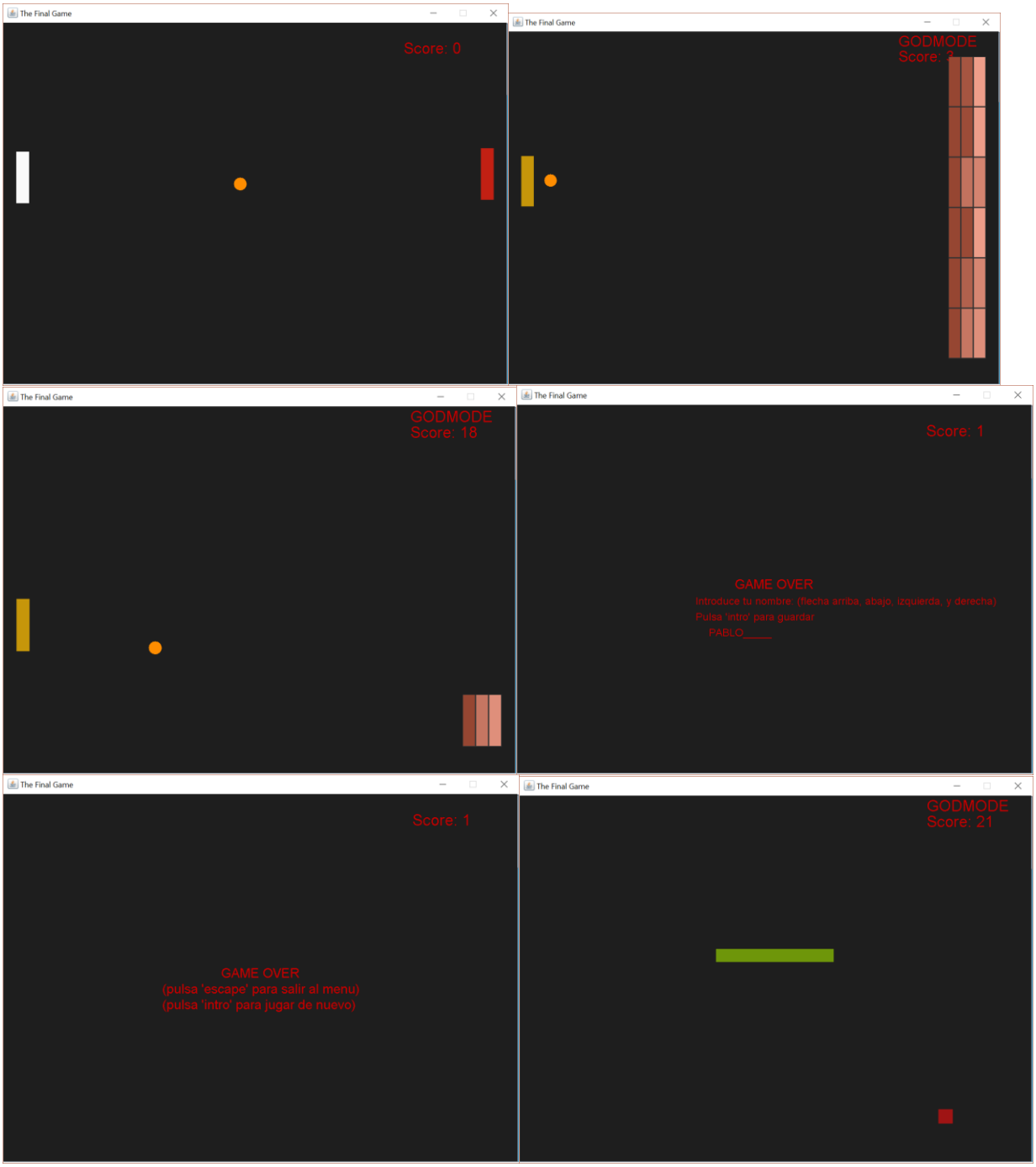
Cuando estés en la pantalla de GAMEOVER y quieras guardar tu nombre has de utilizar las flechas de dirección del teclado.

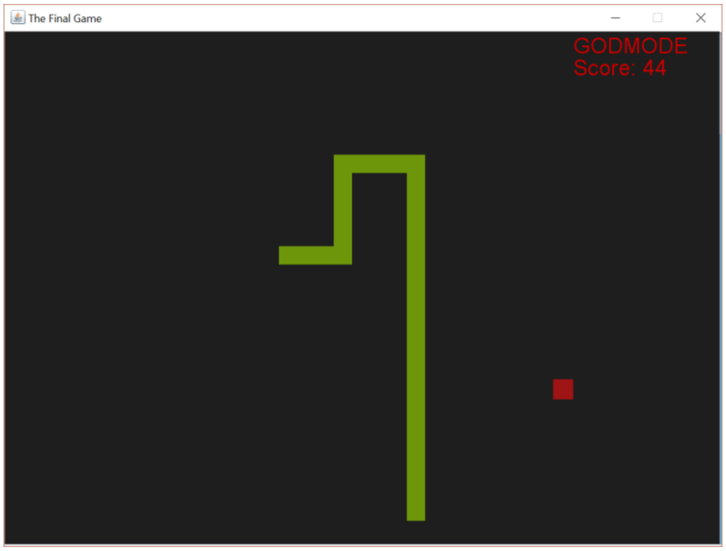
Resultados

Capturas de pantalla

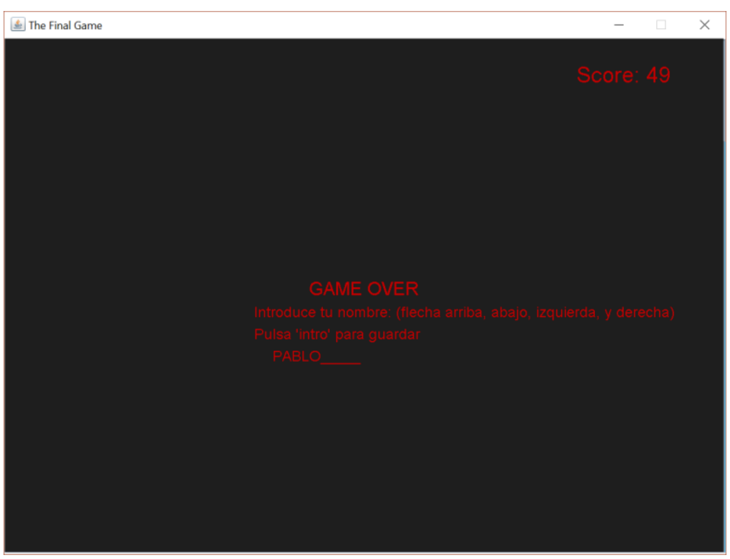
En estas capturas se puede ver el paso por todo el juego y sus diferentes secciones y estados del juego.

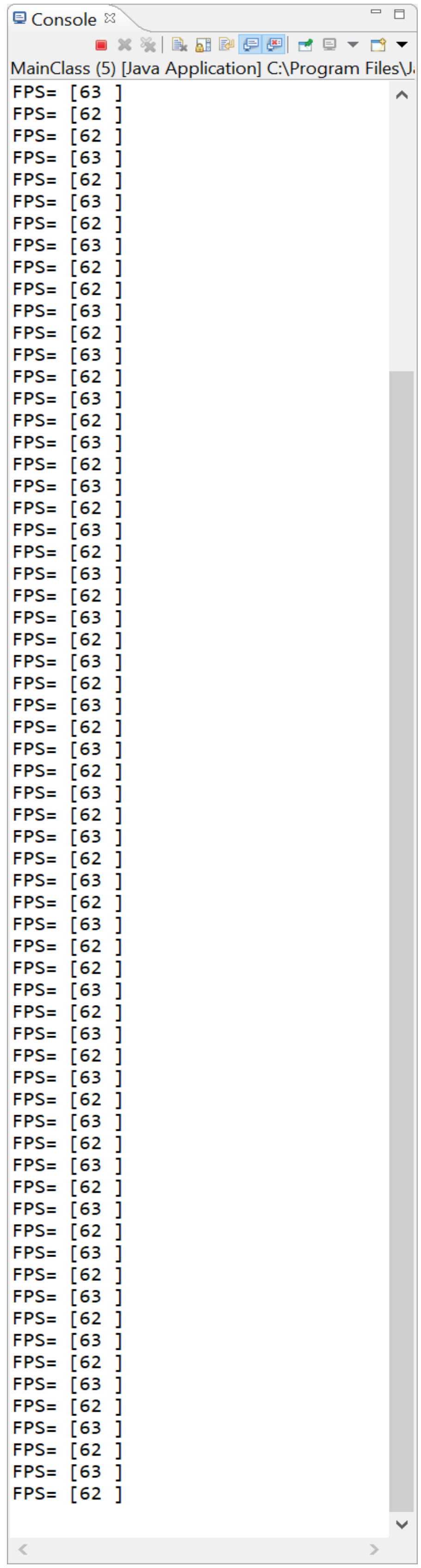



The Final Game
Score: 0
The Final Game
GODMODE
Score: 3
The Final Game
GODMODE
Score: 18
The Final Game
Score: 1
GAME OVER
Introduce tu nombre: (flecha arriba, abajo, izquierda, y derecha)
Pulsa 'intro' para guardar
PABLO____
The Final Game
Score: 1
GAME OVER
(pulsa 'escape' para salir al menu)
(pulsa 'intro' para jugar de nuevo)
The Final Game
GODMODE
Score: 21


The Final Game
GODMODE
Score: 44


The Final Game
Score: 48


The Final Game
Score: 49
GAME OVER
Introduce tu nombre: (flecha arriba, abajo, izquierda, y derecha)
Pulsa 'intro' para guardar
PABLO_____


Console
MainClass (5) [Java Application] C:\Program Files\J
FPS= [63]
FPS= [62]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]
FPS= [63]
FPS= [62]

Conclusión

Resumen General

Finalmente cabe destacar que es un proyecto con unas posibilidades de ampliación muy viables, y también se puede exportar a otras plataformas gracias a la compilación de la JVM (máquina virtual de Java). Aunque el código Java no se ejecuta de forma nativa, la JVM es plenamente capaz de proporcionar servicios relacionados con el sistema operativo, como el disco I/O y el acceso a la red, si los privilegios adecuados se conceden. La JVM permite a los usuarios decidir el nivel de protección adecuado, según una ACL. Por ejemplo, el acceso a disco y de red está habilitado normalmente para aplicaciones de escritorio. En la actualidad, los programas Java pueden ejecutarse en Microsoft Windows, Mac OS X, Linux y sistemas operativos Solaris. Para aplicaciones móviles, los plugins de los navegadores se utilizan en dispositivos basados en Windows y Mac, Android tiene soporte incorporado para Java.

Según lo expuesto anteriormente, y gracias a la flexibilidad aportada por la máquina virtual de Java, este proyecto ha sido realizado en su integridad en el IDE de Eclipse, abriendo así, un amplio abanico de opciones a la hora de exportar nuestro proyecto a las diferentes plataformas que hay en la actualidad, ya que con muy poco trabajo podemos abarcar usuarios de todos estos tipos de dispositivos.

Propuesta de Ampliación

En un futuro es posible que se implemente otro de los juegos favoritos de la época, el Comecocos, esto se decidirá en relación a un nuevo análisis del mercado y de los clientes que reciba el juego en su primera versión.

Otra de las propuestas de ampliación es insertar sonido en nuestro juego, que iría implementado tanto como música de fondo como en las colisiones de los objetos como efectos FX.

Una última propuesta de ampliación, aunque ya sería un poco más ambiciosa, sería el ‘hosteo’ de la aplicación de forma online con un servidor el cual nos guardaría los datos de las partidas de todos los jugadores a medida que vayan jugando y luego esos datos nos permitirían hacer un nuevo panel en el juego con las estadísticas online a tiempo real de todos los jugadores de nuestra aplicación.