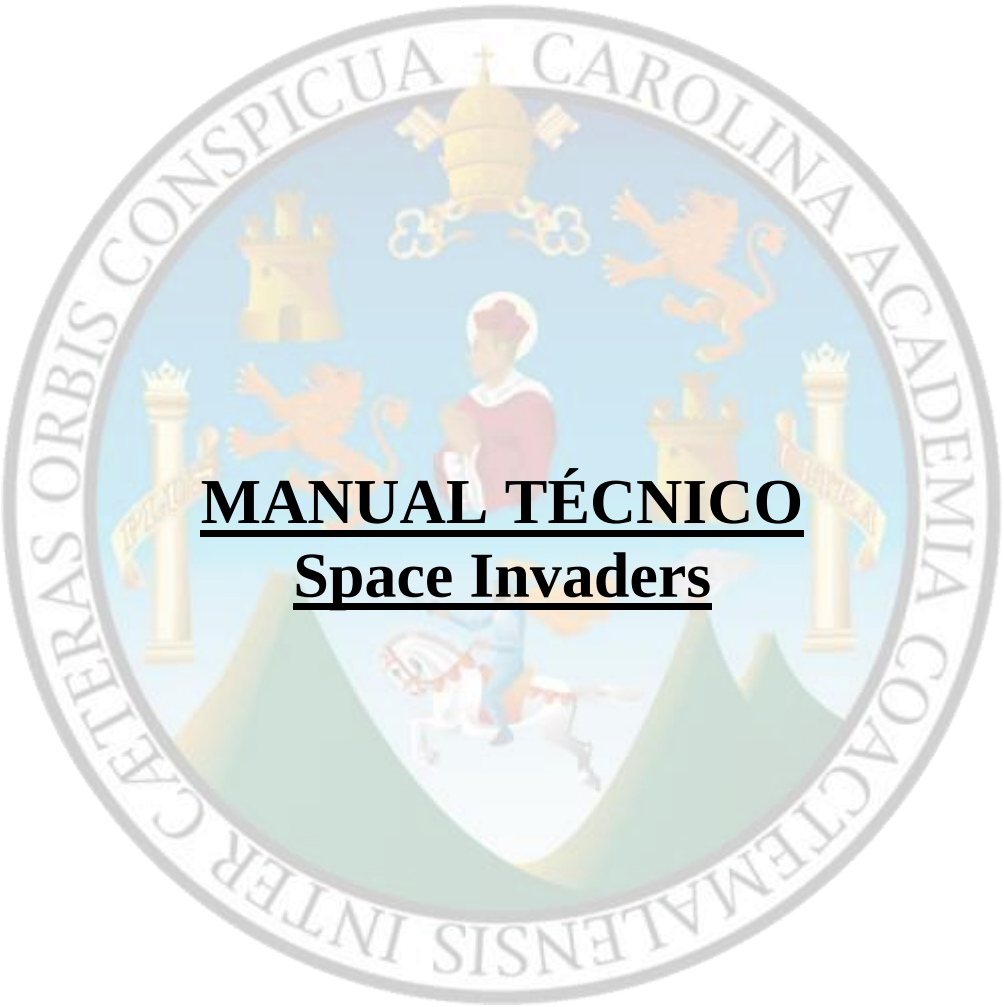
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1
CATEDRÁTICO: ING. WILLIAM ESTUARDO ESCOBAR ARGUETA
TUTOR ACADÉMICO: JOSUÉ RODOLFO MORALES CASTILLO



MANUAL TÉCNICO
Space Invaders

JOSÉ ALEXANDER LÓPEZ LÓPEZ
CARNÉ: 202100305
SECCIÓN: A

GUATEMALA, 20 DE JUNIO DE 2,024

ÍNDICE

INTRODUCCIÓN

El propósito de este manual técnico es proporcionar una guía exhaustiva y detallada sobre el desarrollo, la arquitectura y el mantenimiento del proyecto **Space Invaders**. Este documento está dirigido a desarrolladores y profesionales de TI que buscan comprender en profundidad los componentes internos del juego, sus dependencias de software, y el flujo de datos. Aquí se desglosan los detalles técnicos esenciales, incluyendo la configuración del entorno de desarrollo, la estructura del código, los patrones de diseño implementados y las mejores prácticas para la extensión y mejora del sistema.

El manual aborda cada aspecto del proyecto, comenzando con los requisitos del sistema necesarios para la correcta ejecución de la aplicación, pasando por una descripción detallada de los diferentes módulos y sus interacciones, hasta las instrucciones para la compilación, ejecución y depuración del juego. Además, se incluyen secciones dedicadas a la gestión de versiones, pruebas unitarias e integración continua, asegurando que el mantenimiento y la evolución del proyecto se realicen de manera eficiente y sin contratiempos.

El objetivo final de este manual es facilitar la comprensión y manipulación del código fuente, asegurando que los desarrolladores puedan integrarse rápidamente y trabajar de manera efectiva en el proyecto. Proporciona una base sólida para futuras actualizaciones y mejoras, permitiendo a los desarrolladores realizar modificaciones y añadir nuevas funcionalidades con confianza. A través de este documento, se busca mantener la coherencia y la calidad del código, así como garantizar que el videojuego Space Invaders siga siendo un proyecto robusto y sostenible a largo plazo.

OBJETIVOS

1. GENERAL

1.1. Proveer una guía técnica completa y detallada que permita a los desarrolladores y profesionales de TI comprender, mantener, y expandir el proyecto Space Invaders, asegurando la correcta configuración del entorno, el entendimiento profundo del código fuente y la implementación de mejores prácticas en el desarrollo de software.

2. ESPECÍFICOS

2.1. Proporcionar instrucciones detalladas para la instalación de todas las herramientas necesarias, como el sistema operativo, JDK, y el IDE, asegurando que los desarrolladores puedan configurar su entorno de desarrollo de manera eficiente y sin problemas.

2.2. Explicar la arquitectura del juego y la organización del código fuente, incluyendo los módulos principales, las clases, y sus interacciones.

2.3. Proveer directrices y mejores prácticas para el mantenimiento y la extensión del proyecto, incluyendo la implementación de nuevas funcionalidades, la realización de pruebas unitarias, y la gestión de versiones.

ALCANCES DEL SISTEMA

Configuración del Entorno de Desarrollo:

Instrucciones detalladas para la instalación y configuración de los requisitos del sistema, incluyendo el sistema operativo compatible, JDK, y el IDE recomendado.
Guía paso a paso para importar el proyecto en el IDE y asegurarse de que el entorno esté listo para el desarrollo y la ejecución del juego.

Descripción de la Arquitectura del Sistema:

Desglose completo de la arquitectura del juego, incluyendo diagramas y explicaciones de los componentes principales, como la nave, los enemigos, los ítems y las pantallas del juego.
Descripción de la estructura del código fuente, detallando las clases y métodos principales y cómo interactúan entre sí.
Flujo de Funcionalidades:

Explicación detallada del flujo de la aplicación desde la pantalla principal hasta el fin del juego, incluyendo el inicio de un nuevo juego, la carga de partidas guardadas, la visualización de puntuaciones máximas y la salida del juego.
Instrucciones sobre cómo utilizar cada funcionalidad del juego, con ejemplos y capturas de pantalla para ilustrar cada paso.

Mantenimiento y Extensión del Proyecto:

Directrices para el mantenimiento del código, incluyendo mejores prácticas para la escritura y organización del código, así como la implementación de pruebas unitarias. Procedimientos para la gestión de versiones y la integración continua, asegurando que las modificaciones se realicen de manera controlada y eficiente.
Estrategias para la extensión del juego, proporcionando pautas para añadir nuevas funcionalidades, mejorar las existentes y asegurar la compatibilidad y estabilidad del sistema.

Solución de Problemas y Depuración:

Consejos y técnicas para la depuración del código y la resolución de problemas comunes que puedan surgir durante el desarrollo y la ejecución del juego.
Lista de errores conocidos y sus soluciones, ayudando a los desarrolladores a identificar y corregir problemas de manera rápida y efectiva.

Documentación y Comentarios en el Código:

Importancia de mantener el código bien documentado y comentado para facilitar la comprensión y el mantenimiento del proyecto por parte de otros desarrolladores.
Ejemplos de buenas prácticas en la documentación y comentarios del código.
Al proporcionar una guía detallada y práctica sobre estos aspectos, el manual técnico de Space Invaders tiene como objetivo asegurar que cualquier desarrollador pueda integrarse rápidamente al proyecto, mantenerlo de manera eficiente y contribuir a su mejora y evolución continua.

ESPECIFICACIÓN TÉCNICA

Para asegurar un entorno de desarrollo adecuado y eficiente para el proyecto Space Invaders, es necesario que los programadores cuenten con ciertos requisitos de hardware y software. A continuación, se detallan los requisitos necesarios para trabajar con la aplicación y continuar con su desarrollo futuro:

- **REQUISITOS DE HARDWARE**
- Procesador:
- Procesador de 1 GHz o superior (se recomienda un procesador multinúcleo para un rendimiento óptimo).
- **Memoria RAM:**
- Al menos 4 GB de RAM (se recomiendan 8 GB o más para un desarrollo fluido y ejecución de múltiples herramientas de desarrollo simultáneamente).
- **Espacio en Disco:**

  Al menos 500 MB de espacio libre en disco para la instalación de JDK, IDE, y otros recursos necesarios.

  Espacio adicional para la creación y almacenamiento de archivos del proyecto, incluyendo archivos de juego guardados y binarios generados.
- **Tarjeta Gráfica:**

  No se requiere una tarjeta gráfica avanzada, pero se recomienda una tarjeta con soporte para gráficos 2D básicos.
- **Pantalla:**

  Resolución mínima de 1024x768 píxeles (se recomienda una resolución más alta para una mejor experiencia de desarrollo).

REQUISITOS DE SOFTWARE

  - Q Sistema Operativo:
  - Windows 7 o superior
  - macOS 10.12 (Sierra) o superior
  - Linux (cualquier distribución moderna)

- **Java Development Kit (JDK):**

  JDK 8 o superior. Es esencial tener el JDK instalado para compilar y ejecutar el proyecto. Se puede descargar desde el sitio oficial de Oracle o utilizar distribuciones de código abierto como OpenJDK.

- **Integrated Development Environment (IDE):**

  Un IDE es altamente recomendable para facilitar el desarrollo, la depuración y la gestión del código. Los IDEs recomendados son:

- IntelliJ IDEA
- Eclipse
- NetBeans

- **Control de Versiones:**

  Git debe estar instalado para la gestión de versiones del código fuente. Es útil para la colaboración y el seguimiento de cambios en el proyecto. Se puede descargar desde el sitio oficial de Git.

- **Dependencias de Java:**

  Asegurarse de tener instaladas todas las bibliotecas estándar de Java. No se requieren bibliotecas externas específicas para este proyecto, pero es crucial verificar que las bibliotecas estándar de Java (javax.swing, java.io, etc.) estén disponibles.

- **Herramientas de Compilación y Construcción:**

  Aunque el uso de herramientas como Maven o Gradle no es obligatorio para este proyecto, su utilización puede facilitar la gestión de dependencias y la automatización de tareas de construcción.

- **Herramientas de Pruebas Unitarias:**

  JUnit o TestNG para escribir y ejecutar pruebas unitarias, asegurando que el código nuevo y existente funcione correctamente.

  Herramientas de Integración Continua:

- Jenkins, Travis CI, o cualquier otra herramienta de integración continua para automatizar la construcción, prueba y despliegue del proyecto.

Siguiendo estos requisitos de hardware y software, los programadores estarán bien equipados para trabajar de manera efectiva con la aplicación Space Invaders, asegurando un desarrollo continuo y sostenible del proyecto.

DESCRIPCIÓN DE LA SOLUCIÓN

- Para desarrollar Space Invaders, primero se revisó minuciosamente el enunciado del proyecto para comprender todos los requisitos. Se identificaron las funcionalidades clave como el control de la nave, la aparición de enemigos, la interacción con ítems, y la gestión de diferentes pantallas del juego.
- Decidí modularizar el código, dividiendo el juego en componentes independientes como la nave, los enemigos, y la interfaz de usuario. Utilizando patrones de diseño orientados a objetos para garantizar un código mantenible y escalable.
- Se comenzó con prototipos iniciales para validar nuestras ideas y ajustamos el diseño según fuera necesario. Se implementó el control de la nave, asegurando que se moviera y disparara de manera fluida. Desarre tres tipos de enemigos con diferentes atributos y lógica de movimiento. Integré ítems potenciadores y debilitadores que afectan el temporizador y los puntos del jugador, añadiendo dinamismo al juego.
- Implementé las pantallas del juego, asegurando transiciones suaves y acciones intuitivas para el usuario. Realicé pruebas unitarias y funcionales para validar cada componente y el juego en su conjunto. Durante las pruebas, corregí errores y optimizé el rendimiento del juego.
- Este enfoque nos permitió cumplir con todos los requisitos del enunciado y desarrollar un Space Invaders funcional, robusto y mantenible.

LÓGICA DEL PROGRAMA

❖ **Clase: Main**:

```
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.Font;
6  import java.awt.GridBagConstraints;
7  import java.awt.GridBagLayout;
8  import java.awt.Insets;
9  import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11 import java.io.File;
12 import java.io.IOException;
13 import java.util.logging.Level;
14 import java.util.logging.Logger;
15 import javax.sound.sampled.AudioInputStream;
16 import javax.sound.sampled.AudioSystem;
17 import javax.sound.sampled.Clip;
18 import javax.swing.ImageIcon;
19 import javax.swing.JButton;
20 import javax.swing.JFrame;
21 import javax.swing.JLabel;
22 import javax.swing.JPanel;
23 import javax.swing.SwingUtilities;
24 import recursos.Constantes;
```

➢ **Librerías**

- java.awt: Utilizada para manejar componentes gráficos como JFrame, JPanel, JLabel, y JButton, permitiendo la creación de la interfaz de usuario.
- javax.sound.sampled: Usada para cargar y reproducir archivos de audio (AudioInputStream, Clip), necesario para la reproducción de música de fondo en el juego.

➢ **Variables Globales de la clase Main**

```
29     // VAriables
30     public static Escena scene;
31     public static boolean jeu = true;
32     public static Clip clip; // Variable para el clip de música
33
```

scene: Objeto de tipo Escena que representa el contenido principal del juego.

jeu: Variable booleana que controla el estado del juego.

clip: Variable estática de tipo Clip para la reproducción de música de fondo.

➢ **Función Main**

La función main se encarga de iniciar la aplicación creando y configurando la ventana principal (JFrame) del juego. Carga la música de fondo desde un archivo de audio, configura la imagen de fondo y los botones principales (Nuevo Juego, Cargar Juego, Puntuaciones, Salir) usando JButton y JPanel.

```
    // metodos
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            // Creación de la ventana de la aplicación
            JFrame fenetre = new JFrame("Space Invaders");
            fenetre.setSize(1000, 600);
            fenetre.setResizable(false);
            fenetre.setLocationRelativeTo(null);
            fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            try {...7 lines } catch (Exception ex) {...3 lines }
            // Crear una etiqueta para la imagen de fondo
            JLabel background = new JLabel();
            background.setLayout(new BorderLayout());

            // Cargar la imagen de fondo desde un archivo o URL
            ImageIcon backgroundImage = new ImageIcon("src/images/fondo_principal.jpg"); // Cambia la ruta por la de tu imagen

            background.setIcon(backgroundImage);

            // Panel para los botones con GridBagLayout para centrar los botones
            JPanel buttonPanel = new JPanel(new GridBagLayout());
            buttonPanel.setOpaque(false); // Hacer el panel transparente
            buttonPanel.setSize(100,100);
            // Estilo de fuente para los botones
            Font buttonFont = new Font("Futura", Font.BOLD, 22);
            // Cargar la imagen del JLabel
            ImageIcon labelImage = new ImageIcon("src/images/Icono.png"); // Cambia la ruta por la de tu imagen
            JLabel imageLabel = new JLabel(labelImage);
            JButton startButton = new JButton("Nuevo Juego");
```

➢ **Procedimientos, métodos y Funciones utilizadas**

A continuación se dará una explicación general de lo que hace cada función:

```java
public static void demarrerJeu() throws IOException {
    // ventana del juegos
    try {
            File audioFile = new File("src/sons/Intro.wav");
            AudioInputStream audioStream = AudioSystem.getAudioInputStream(audioFile);
            clip = AudioSystem.getClip();
            clip.open(audioStream);
            clip.loop(Clip.LOOP_CONTINUOUSLY);
        } catch (Exception ex) {
            System.out.println("Error al cargar el archivo de audio: " + ex.getMessage());
        }
    JFrame jeuFrame = new JFrame("Space Invaders");
    jeuFrame.setSize(Constantes.LARGO_VENTANA, Constantes.ALTURA_VENTANA);
    jeuFrame.setResizable(false);
    jeuFrame.setLocationRelativeTo(null);
    jeuFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jeuFrame.setAlwaysOnTop(true);
    scene = new Escena();
    Main.jeu=true;
    jeuFrame.setContentPane(scene);
    jeuFrame.setVisible(true);

}
```

demarrerJeu(): Inicia la ventana principal del juego (JFrame jeuFrame) donde se mostrará la escena del juego (Escena).

- **Clase: Escena:**

```java
import javax.swing.JPanel;

import Entidades.GrupoAliens;
import Entidades.Item1;
import Entidades.Item2;
import Entidades.Item3;
import Entidades.Item4;
import Entidades.ProyectilAlien;
import Entidades.ProyectilNave;
import Entidades.Nave;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;
import javax.swing.Timer;
import static juego.Main.clip;
import recursos.Tiempo;
import recursos.Teclas;
import recursos.Constantes;

```

➢ **Librerías**

- Java.awt: Proporciona las clases necesarias para manejar gráficos y eventos en la interfaz de usuario.
- javax.swing: Utilizada para componentes gráficos como JPanel, JFrame y JOptionPane, facilitando la creación de la interfaz de usuario.

➢ **Variables Globales de la clase Escena:**

fondo: Imagen de fondo del juego.
vaisseau: Objeto de tipo Nave que representa la nave del jugador.
groupeAliens: Objeto de tipo GrupoAliens que maneja el grupo de aliens en el juego.
tirVaisseau, tirAlien1, tirAlien2, tirAlien3: Objetos de proyectiles para la nave y aliens respectivamente.
item1, item2, item3, item4: Objetos de tipo Item que representan los items del juego.
scoreImage, timeImage: Imágenes utilizadas para representar el score y el tiempo en la interfaz.
afficheScore, afficheTexte, timerFont: Fuentes utilizadas para mostrar texto en la interfaz.
score: Puntaje actual del jugador.
timer: Objeto Timer utilizado para contar el tiempo restante en el juego.
tiempoRestante: Tiempo restante en segundos para la partida.
gameOverDisplayed: Variable booleana que controla la visualización del mensaje de Game Over.

➢ **Constructor:**

El constructor Escena() inicializa los componentes principales del juego, carga las imágenes de fondo y de los items, configura el temporizador y maneja eventos de teclado para interactuar con la interfaz.

```
70      public Escena() throws IOException {
71                  super();
72
73          try {...4 lines } catch (IOException e) {
            e.printStackTrace();
78      }
79              fondo = ImageIO.read(new File("src/images/fondo.jpg"));
80              // instancias de las teclas
81              this.setFocusable(true);
82              this.requestFocusInWindow();
83              this.addKeyListener(new Teclas());
84
85          // Iniciar el temporizador
            timer = new Timer(1000, new ActionListener() {...13 lines });
99      timer.start(); // Comenzar el temporizador
100
101              this.addKeyListener(new KeyAdapter() {...13 lines });
114             // instancias del tiempo
115             Thread chronoEcran = new Thread(new Tiempo());
                chronoEcran.start();
117     }
```

➢ **Métodos:**

- **paintComponent(Graphics g):** Método sobrecargado para dibujar todos los elementos visuales en la pantalla de juego. Dibuja el fondo, el score, el tiempo, la nave, los aliens, los proyectiles y los items. También gestiona la detección de colisiones y el estado de Game Over.

```
139
    public void paintComponent(Graphics g) {
141         super.paintComponent(g);
142         Graphics g2 = (Graphics2D) g;
143         // fondo de pantalla
144         g2.drawImage(fondo, 0, 0, Constantes.LARGO_VENTANA, Constantes.ALTURA_VENTANA, this);
145         // Establecer el color de fondo para el rectángulo
146         g.setColor(Color.BLUE); // O el color que prefieras
147         // Dibujar un único rectángulo grande detrás de ambos textos
148         g.fillRect(0, 0, 1000, 40); // Ajusta las coordenadas y el tamaño según sea necesario
149         g.drawImage(scoreImage, 20, 10, null); // Ajusta las coordenadas según sea necesario
150         // Dibujar el primer texto
151         g.setFont(afficheScore);
152         g.setColor(Color.WHITE); // O el color que prefieras para el texto
153         g.drawString("Score : " + score, 380, 25);
154         g.drawImage(timeImage, 350, 10, null); // Ajusta las coordenadas según sea necesario
155         // Dibujar el segundo texto
156         g.setFont(timerFont);
157         g.setColor(Color.WHITE); // O el color que prefieras para el texto
158         g.drawString("Tiempo: " + tiempoRestante + " s", 50, 25);
159         //diseño de nave
160         this.vaisseau.dessinVaisseau(g2);
161         // diseño aliens
162         this.groupeAliens.dessinAliens(g2);
163         // diseño proyectil nave
164         this.tirVaisseau.dessinTirVaisseau(g2);
165         this.groupeAliens.tirVaisseauToucheAlien(this.tirVaisseau);
166         // Inicio de juego
167         if(tiempoRestante ==90) {...5 lines }
```

- **guardarPuntaje(String nombre, int puntaje):** Método privado que guarda el puntaje del jugador en un archivo de texto. Utiliza BufferedWriter para escribir el puntaje junto con el nombre del jugador en el archivo.

```
    private void guardarPuntaje(String nombre, int puntaje) {
        File puntajeArchivo = new File("src/Puntajes/puntajes.txt");
        try {
            if (!puntajeArchivo.exists()) {
                puntajeArchivo.getParentFile().mkdirs(); // Crear directorio si no existe
                puntajeArchivo.createNewFile(); // Crear archivo si no existe
            }

            try (BufferedWriter writer = new BufferedWriter(new FileWriter(puntajeArchivo, true))) {
                writer.write(nombre + ": " + puntaje);
                writer.newLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error guardando el puntaje: " + e.getMessage(), "Error", JOptionPane.ERROR_ME
        }
    }
```

❖ **Clase: PuntuacionesFrame**:

```
import javax.swing.*;
import java.awt.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
```

➢ **Librerías**

- javax.swing: Utilizada para la interfaz gráfica de usuario, incluyendo componentes como JFrame, JList, DefaultListModel, JScrollPane y JOptionPane.
- java.awt: Proporciona clases para manejar gráficos y eventos en la interfaz de usuario.
- java.io: Utilizada para la lectura de archivos de texto con BufferedReader y FileReader.
- java.util: Utilizada para trabajar con colecciones como ArrayList y Collections, así como Comparator para ordenar las puntuaciones.

```
27
28          listModel = new DefaultListModel<>();
29          puntuacionesList = new JList<>(listModel);
30          puntuacionesList.setFont(new Font("Arial", Font.PLAIN, 16)); // Establecer una fuente y tamaño más grande
31
```

➢ **Variables Globales de la clase Escena:**

- **listModel:** DefaultListModel utilizado para almacenar y mostrar las puntuaciones en la interfaz.
- **puntuacionesList:** JList que muestra las puntuaciones almacenadas.

➢ **Constructor:**

```
22      public PuntuacionesFrame() {
23          super("Mejores Puntuaciones");
24          setSize(400, 300);
25          setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
26          setLocationRelativeTo(null);
27
28          listModel = new DefaultListModel<>();
29          puntuacionesList = new JList<>(listModel);
30          puntuacionesList.setFont(new Font("Arial", Font.PLAIN, 16)); // Establecer una fuente y tamaño más grande
31
32          cargarPuntuaciones();
33
34          getContentPane().add(new JScrollPane(puntuacionesList), BorderLayout.CENTER);
35      }
```

- El constructor PuntuacionesFrame() inicializa la ventana de la aplicación de mejores puntuaciones, estableciendo el título, tamaño, cierre de operación y ubicación. Carga las puntuaciones desde el archivo puntajes.txt utilizando el método cargarPuntuaciones(), y las muestra en una lista JList.

➢ **Métodos:**

- **cargarPuntuaciones():** Lee las puntuaciones desde el archivo puntajes.txt, separa el nombre y la puntuación de cada línea, crea objetos Puntuacion y los ordena de mayor a menor puntuación. Construye el modelo de lista listModel con las cinco mejores puntuaciones y lo muestra en puntuacionesList.

```
37      private void cargarPuntuaciones() {
38          List<Puntuacion> puntuaciones = new ArrayList<>();
39
40          try (BufferedReader br = new BufferedReader(new FileReader("src/Puntajes/puntajes.txt"))) {
41              String linea;
42              while ((linea = br.readLine()) != null) {
43                  // Separar nombre y puntuación
44                  String[] partes = linea.split("\\s+"); // Separar por espacios en blanco, ajustar según el delimitador
45                  if (partes.length >= 2) {
46                      String nombre = partes[0];
47                      int puntuacion = Integer.parseInt(partes[1]);
48                      puntuaciones.add(new Puntuacion(nombre, puntuacion));
49                  }
50              }
51
52              // Ordenar las puntuaciones de mayor a menor
53              Collections.sort(puntuaciones, Comparator.comparingInt(Puntuacion::getPuntuacion).reversed());
54              // Construir el modelo de lista con las puntuaciones
55              listModel.clear();
56              for (int i = 0; i < Math.min(puntuaciones.size(), 5); i++) {
57                  Puntuacion p = puntuaciones.get(i);
58                  String item = (i + 1) + ". " + p.getNombre() + " - " + p.getPuntuacion();
```

- **Puntuacion (Clase interna):** Representa una puntuación con un nombre y un valor de puntuación. Proporciona métodos para obtener el nombre y la puntuación.

```
    private static class Puntuacion {
        private String nombre;
        private int puntuacion;

        public Puntuacion(String nombre, int puntuacion) {
            this.nombre = nombre;
            this.puntuacion = puntuacion;
        }

        public String getNombre() {
            return nombre;
        }

        public int getPuntuacion() {
            return puntuacion;
        }
    }
```

❖ **Clase: Alien:**

```
package Entidades;

import javax.swing.ImageIcon;

import recursos.Constantes;

public class Alien extends Entidad {
```

➢ **Constructor:**

- Constructor Alien(int xPos, int yPos, String strImg1, String strImg2): Este constructor inicializa las variables de instancia heredadas de la clase Entidad y algunas específicas para el alienígena.

```
    public Alien(int xPos, int yPos, String strImg1, String strImg2) {

        // Inicialización de variables de la super classe
        super.xPos = xPos;
        super.yPos = yPos;
        super.largeur = Constantes.LARGO_ALIEN;
        super.altura = Constantes.ALTURA_ALIEN;
        super.dx = 0;
        super.dy = 0;
        super.vivo = true;
        super.strImg1 = strImg1;
        super.strImg2 = strImg2;
        super.strImg3 = "/images/alienMeurt.png";
        super.ico = new ImageIcon(getClass().getResource(super.strImg1));
        super.img = this.ico.getImage();
    }
```

➢ **Métodos:**

```
33      public void EligeImagen(boolean pos1) {
34
35          if(this.vivo == true) {
36              if(pos1 == true) {super.ico = new ImageIcon(getClass().getResource(strImg1));}
37               else {super.ico = new ImageIcon(getClass().getResource(strImg2));}
38              }
39              else {super.ico = new ImageIcon(getClass().getResource(strImg3));}
40              super.img = this.ico.getImage();
41      }
42
```

- EligeImagen(boolean pos1): Este método selecciona la imagen a mostrar dependiendo del estado del alienígena (vivo o muerto).

❖ **Clase: Entidades***:*

```
1   package Entidades;
2
3   import java.awt.Image;
4
5   import javax.swing.ImageIcon;
6
8   public abstract class Entidad {
```

➢ **Librerías**

- java.awt.Image: Se utiliza para representar imágenes en el juego.
- javax.swing.ImageIcon: Permite cargar imágenes desde el sistema de archivos o desde recursos empaquetados en el archivo JAR.

➢ **Variables globales de la clase Entidad**

```
protected int largeur, altura, xPos, yPos, dx, dy;
protected boolean vivo;
protected String strImg1, strImg2, strImg3;
protected ImageIcon ico;
protected Image img;
```

- largeur, altura: Representan las dimensiones (ancho y alto) de la entidad.
- xPos, yPos: Coordenadas x e y de la posición de la entidad en la pantalla.
- dx, dy: Velocidades de desplazamiento en las direcciones x e y.
- vivo: Indica si la entidad está viva o no.
- strImg1, strImg2, strImg3: Nombres de los archivos de imagen asociados a la entidad en diferentes estados.

- ico: Objeto ImageIcon que carga y gestiona las imágenes asociadas a la entidad.
- img: Objeto Image que representa la imagen cargada desde el ImageIcon.

➢ **Métodos:**

- **getLargeur():** Devuelve el ancho (largeur) de la entidad.
- **setLargeur(int largeur):** Establece el ancho (largeur) de la entidad.
- **getAltura():** Devuelve la altura (altura) de la entidad.
- **setAltura(int altura):** Establece la altura (altura) de la entidad.
- **getxPos():** Devuelve la posición x (xPos) de la entidad.
- **setxPos(int xPos):** Establece la posición x (xPos) de la entidad.
- **getyPos():** Devuelve la posición y (yPos) de la entidad.
- **setyPos(int yPos):** Establece la posición y (yPos) de la entidad.
- **getDx():** Devuelve el desplazamiento en x (dx) de la entidad.
- **setDx(int dx):** Establece el desplazamiento en x (dx) de la entidad.
- **getDy():** Devuelve el desplazamiento en y (dy) de la entidad.
- **setDy(int dy):** Establece el desplazamiento en y (dy) de la entidad.
- **isVivo():** Devuelve el estado de vida (vivo) de la entidad.
- **setVivo(boolean vivo):** Establece el estado de vida (vivo) de la entidad.
- **getStrImg1():** Devuelve el nombre del archivo de imagen 1 (strImg1) asociado a la entidad.
- **setStrImg1(String strImg1):** Establece el nombre del archivo de imagen 1 (strImg1) asociado a la entidad.
- **getStrImg2():** Devuelve el nombre del archivo de imagen 2 (strImg2) asociado a la entidad.
- **setStrImg2(String strImg2):** Establece el nombre del archivo de imagen 2 (strImg2) asociado a la entidad.
- **getStrImg3():** Devuelve el nombre del archivo de imagen 3 (strImg3) asociado a la entidad.
- **setStrImg3(String strImg3):** Establece el nombre del archivo de imagen 3 (strImg3) asociado a la entidad.
- **getIco():** Devuelve el ImageIcon (ico) asociado a la entidad.
- **setIco(ImageIcon ico): Establece** el ImageIcon (ico) asociado a la entidad.
- **getImg():** Devuelve el objeto Image (img) asociado a la entidad.

- **setImg(Image img):** Establece el objeto Image (img) asociado a la entidad.

```
19      public int getLargeur() {...}
20
21      public void setLargeur(int largeur) {...3 lines }
24      public int getAltura() {...3 lines }
27      public void setAltura(int altura) {...3 lines }
        public int getxPos() {...3 lines }
        public void setxPos(int xPos) {...3 lines }
        public int getyPos() {...3 lines }
        public void setyPos(int yPos) {...3 lines }
42      public int getDx() {...3 lines }
45      public void setDx(int dx) {...3 lines }
48      public int getDy() {...3 lines }
51      public void setDy(int dy) {...3 lines }
54      public boolean isVivo() {...3 lines }
57      public void setVivo(boolean vivo) {...3 lines }
60      public String getStrImg1() {...3 lines }
63      public void setStrImg1(String strImg1) {...3 lines }
66      public String getStrImg2() {...3 lines }
69      public void setStrImg2(String strImg2) {...3 lines }
72      public String getStrImg3() {...3 lines }
75      public void setStrImg3(String strImg3) {...3 lines }
78      public ImageIcon getIco() {...3 lines }
81      public void setIco(ImageIcon ico) {...3 lines }
84      public Image getImg() {...3 lines }
87      public void setImg(Image img) {...3 lines }
90  }
91
```

❖ **Clase: GrupoAliens***:*

```
1   package Entidades;
2   import java.awt.Graphics;
3   import java.util.Random;
4
5   import juego.Main;
6   import recursos.Audio;
7   import recursos.Tiempo;
8   import recursos.Constantes;
```

➢ **Librerías:**

- **java.awt.Graphics:** Se utiliza para realizar operaciones de dibujo en el contexto gráfico, como dibujar imágenes.
- **java.util.Random**: Permite generar números aleatorios, utilizado para seleccionar un alien aleatorio que realice un disparo.

- **juego.Main:** No proporciona detalles explícitos sobre esta clase, pero se utiliza para acceder a la variable score de Main.scene.
- **recursos.Audio:** Proporciona métodos para reproducir sonidos en el juego.
- **recursos.Tiempo:** No se muestra explícitamente en el código proporcionado, pero parece relacionarse con el manejo del tiempo en el juego.
- **recursos.Constantes:** Contiene valores constantes utilizados en diferentes partes del juego, como posiciones iniciales, espaciado, velocidades, etc.

➢ **Variables Globales:**

```
12      // Variables
13      // tabla que contiene a los aliens
        private Alien tabAlien[][] = new Alien[8][5];
15      private boolean vaAbajo, pos1; // numeros
16      private int velocidad;
        private int[] tabAlienMuerto = {-1,-1}; // alien muerto
18      Random azar = new Random();
19      private int nombreAliens = Constantes.NUMERO_ALIENS;
20      private int contadorSonAlien = 0;
```

- **tabAlien[][]:** Matriz que contiene a los aliens organizados en filas y columnas.
- **vaAbajo:** Indica la dirección actual de movimiento de los aliens (hacia abajo o hacia arriba).
- **pos1:** Variable utilizada para determinar qué imagen de cada alien se debe mostrar.
- **velocidad:** Velocidad de movimiento de los aliens, definida en Constantes.VELOCIDAD_ALIEN.
- **tabAlienMuerto[]:** Almacena la posición del alien que ha sido eliminado.
- **nombreAliens:** Número total de aliens en la matriz.
- **contadorSonAlien:** Contador utilizado para reproducir sonidos de alienígenas en un ciclo determinado.

➢ **Métodos:**

- **iniciaTablaAliens():** Inicializa la matriz de aliens con posiciones y tipos de imágenes según la columna.

```java
private void iniciaTablaAliens() {
    // desplega la tabla de aliens
    for(int ligne = 0; ligne < 8; ligne++) {
        for(int colonne = 0; colonne < 5; colonne++) {
            int xPos = Constantes.X_POS_INI_ALIEN + (Constantes.LARGO_ALIEN + Constantes.ESPACIO_COLUMNAS_ALIEN) * colonne;
            int yPos = Constantes.Y_POS_INI_ALIEN + Constantes.ESPACIO_FILAS_ALIEN * ligne;
            String image1, image2;

            switch (colonne) {
                case 0:
                    image1 = "/images/alienBas1.png";
                    image2 = "/images/alienBas2.png";
                    break;
                case 1:
                case 2:
                    image1 = "/images/alienMilieu1.png";
                    image2 = "/images/alienMilieu2.png";
                    break;
                default:
                    image1 = "/images/alienHaut1.png";
                    image2 = "/images/alienHaut2.png";
                    break;
            }

            this.tabAlien[ligne][colonne] = new Alien(xPos, yPos, image1, image2);
        }
    }
}
```

- **dessinAliens(Graphics g)**: Dibuja los aliens en la pantalla, seleccionando la imagen adecuada según el estado y posición.

```java
public void dessinAliens(Graphics g) {
    if(Tiempo.compteTours % (100 - 10 * this.velocidad) == 0) {
        this.deplacementAliens();
    }
    // Diseño de aliens contenidos en la matriz
    for(int ligne = 0; ligne < 8; ligne++) {
        for(int colonne = 0; colonne < 5; colonne++) {
            if(this.tabAlien[ligne][colonne] != null) {
                this.tabAlien[ligne][colonne].EligeImagen(pos1);
                g.drawImage(this.tabAlien[ligne][colonne].getImg(), this.tabAlien[ligne][colonne].getxPos(), this.tabAlien[ligne][colonne].getyP
            }
        }
    }
}
```

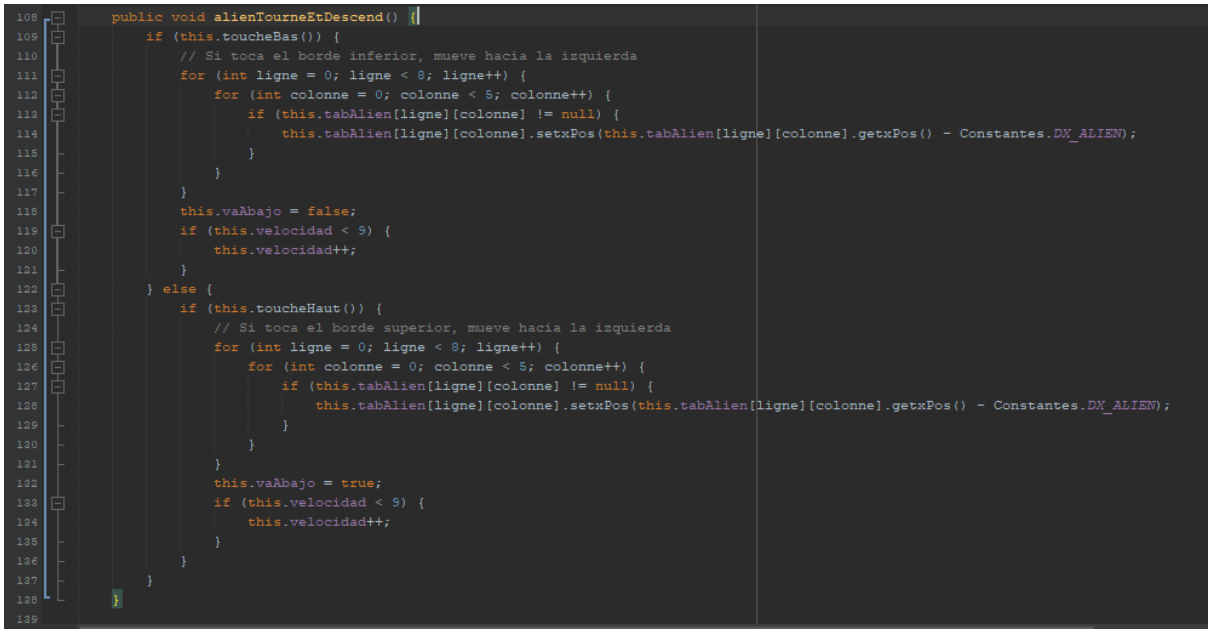
- **toucheHaut():** Verifica si algún alien toca el borde superior de la pantalla.

```java
private boolean toucheHaut() { // Cambiada la lógica para el movimiento horizontal
    boolean reponse = false;
    for(int ligne = 0; ligne < 8; ligne++) {
        for(int colonne = 0; colonne < 5; colonne++) {
            if(this.tabAlien[ligne][colonne] != null) {
                if(this.tabAlien[ligne][colonne].getyPos() < Constantes.MARGEN) {
                    reponse = true;
                    break;
                }
            }
        }
    }
    return reponse;
}
```

- **toucheBas():** Verifica si algún alien toca el borde inferior de la pantalla.

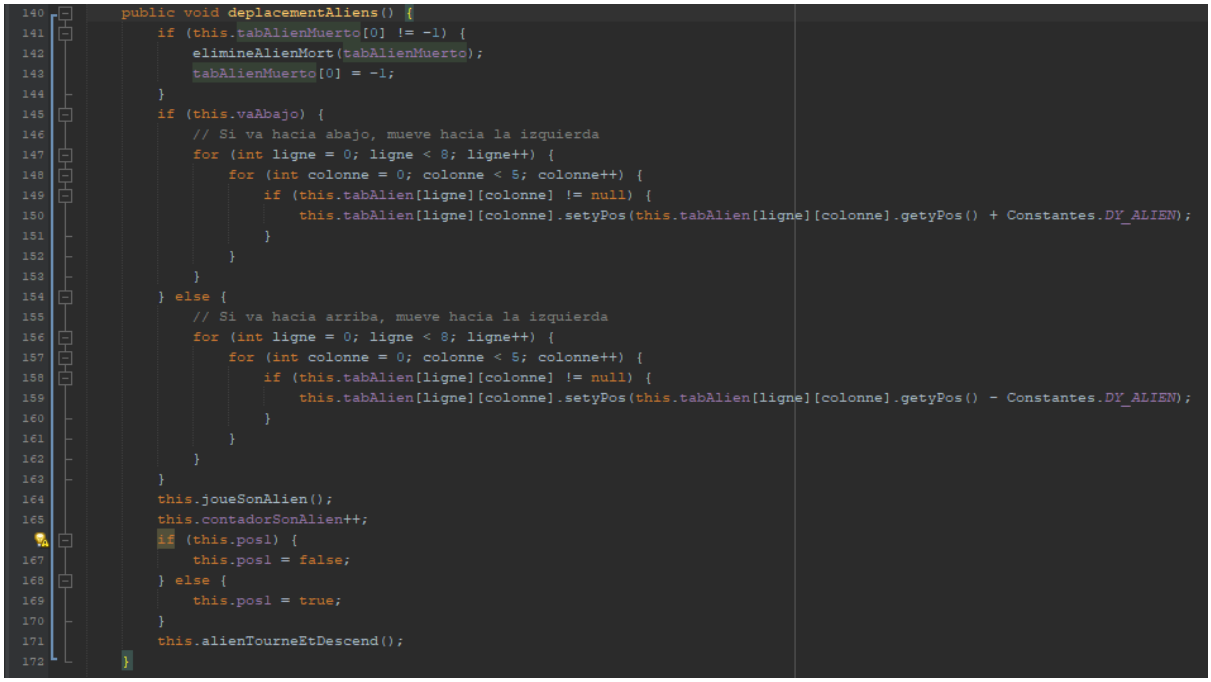
```java
private boolean toucheBas() { // Cambiada la lógica para el movimiento horizontal
    boolean reponse = false;
    for(int ligne = 0; ligne < 8; ligne++) {
        for(int colonne = 0; colonne < 5; colonne++) {
            if(this.tabAlien[ligne][colonne] != null) {
                if(this.tabAlien[ligne][colonne].getyPos() >
                Constantes.ALTURA_VENTANA - Constantes.ALTURA_ALIEN - Constantes.DY_ALIEN - Constantes.MARGEN) {
                    reponse = true;
                    break;
                }
            }
        }
    }
    return reponse;
}
```

- **alienTourneEtDescend():** Maneja el movimiento y rotación de los aliens cuando alcanzan los bordes superior e inferior.



```
public void alienTourneEtDescend() {
    if (this.toucheBas()) {
        // Si toca el borde inferior, mueve hacia la izquierda
        for (int ligne = 0; ligne < 8; ligne++) {
            for (int colonne = 0; colonne < 5; colonne++) {
                if (this.tabAlien[ligne][colonne] != null) {
                    this.tabAlien[ligne][colonne].setxPos(this.tabAlien[ligne][colonne].getxPos() - Constantes.DX_ALIEN);
                }
            }
        }
        this.vaAbajo = false;
        if (this.velocidad < 9) {
            this.velocidad++;
        }
    } else {
        if (this.toucheHaut()) {
            // Si toca el borde superior, mueve hacia la izquierda
            for (int ligne = 0; ligne < 8; ligne++) {
                for (int colonne = 0; colonne < 5; colonne++) {
                    if (this.tabAlien[ligne][colonne] != null) {
                        this.tabAlien[ligne][colonne].setxPos(this.tabAlien[ligne][colonne].getxPos() - Constantes.DX_ALIEN);
                    }
                }
            }
            this.vaAbajo = true;
            if (this.velocidad < 9) {
                this.velocidad++;
            }
        }
    }
}
```

- **deplacementAliens():** Controla el desplazamiento horizontal y vertical de los aliens, gestionando también la velocidad y la reproducción de sonidos.



```
public void deplacementAliens() {
    if (this.tabAlienMuerto[0] != -1) {
        elimineAlienMort(tabAlienMuerto);
        tabAlienMuerto[0] = -1;
    }
    if (this.vaAbajo) {
        // Si va hacia abajo, mueve hacia la izquierda
        for (int ligne = 0; ligne < 8; ligne++) {
            for (int colonne = 0; colonne < 5; colonne++) {
                if (this.tabAlien[ligne][colonne] != null) {
                    this.tabAlien[ligne][colonne].setyPos(this.tabAlien[ligne][colonne].getyPos() + Constantes.DY_ALIEN);
                }
            }
        }
    } else {
        // Si va hacia arriba, mueve hacia la izquierda
        for (int ligne = 0; ligne < 8; ligne++) {
            for (int colonne = 0; colonne < 5; colonne++) {
                if (this.tabAlien[ligne][colonne] != null) {
                    this.tabAlien[ligne][colonne].setyPos(this.tabAlien[ligne][colonne].getyPos() - Constantes.DY_ALIEN);
                }
            }
        }
    }
    this.joueSonAlien();
    this.contadorSonAlien++;
    if (this.posl) {
        this.posl = false;
    } else {
        this.posl = true;
    }
    this.alienTourneEtDescend();
}
```

- **tirVaisseauToucheAlien(ProyectilNave tirVaisseau):** Detecta si un proyectil disparado por la nave impacta y elimina a un alien, actualizando el puntaje del juego.

```
174     public void tirVaisseauToucheAlien(ProyectilNave tirVaisseau) {
175     for(int ligne = 0; ligne < 8; ligne++) {
176         for(int colonne = 0; colonne < 5; colonne++) {
177             if(this.tabAlien[ligne][colonne] != null) {
178                 if(tirVaisseau.tueAlien(this.tabAlien[ligne][colonne]) == true) {
179                     this.tabAlien[ligne][colonne].vivo = false; // On tue l'alien
180                     tirVaisseau.yPos = -1; // On tue le tir
181                     this.tabAlienMuerto[0] = ligne;
182                     this.tabAlienMuerto[1] = colonne;
183 
184                     // Determinar el valor según la columna
185                     if(colonne == 0 ) {
186                         Main.scene.score = Main.scene.score + Constantes.VALOR_ALIEN_TIPO3;
187                     } else if(colonne == 1 || colonne == 2) {
188                         Main.scene.score = Main.scene.score + Constantes.VALOR_ALIEN_TIPO2;
189                     } else {
190                         Main.scene.score = Main.scene.score + Constantes.VALOR_ALIEN_TIPO1;
191                     }
192                     break;
193                 }
194             }
195         }
196     }
197 }
```

- **elimineAlienMort(int[] tabAlienMort):** Elimina un alien de la matriz cuando ha sido derrotado.

```
199     private void elimineAlienMort(int[] tabAlienMort) {
200         this.tabAlien[tabAlienMort[0]][tabAlienMort[1]] = null;
201         this.nombreAliens--;
202     }
```

- **choixAlienQuiTire():** Selecciona aleatoriamente un alien vivo que disparará.

```
204     public int[] choixAlienQuiTire() {
205         int positionAlien[] = {-1,-1};
206         if(this.nombreAliens != 0) {
207             do {
208                 int ligne = azar.nextInt(7); // Tiro al azar de los aliens
209                 for(int colonne = 4; colonne >= 0; colonne--) {
210                     if(tabAlien[ligne][colonne] != null) {
211                         positionAlien[0] = this.tabAlien[ligne][colonne].getxPos();
212                         positionAlien[1] = this.tabAlien[ligne][colonne].getyPos();
213                         break;
214                     }
215                 }
216             } while(positionAlien[0] == -1);
217         }
218         return positionAlien;
219     }
```

- **joueSonAlien():** Reproduce sonidos de alienígenas en ciclos determinados.

```java
private void joueSonAlien() {
    int compteur = this.contadorSonAlien % 4;
    if(compteur == 0) {
        Audio.playSound("/sons/sonAlien1.wav");
    } else if(compteur == 1) {
        Audio.playSound("/sons/sonAlien2.wav");
    } else if(compteur == 2) {
        Audio.playSound("/sons/sonAlien3.wav");
    } else {
        Audio.playSound("/sons/sonAlien4.wav");
    }
}
```

- **getNombreAliens():** Obtiene el número actual de aliens vivos en la matriz.

```java
public int getNombreAliens() {
    return nombreAliens;
}
```

- **positionAlienLePlusGauche():** Calcula la posición más alta en la pantalla ocupada por los aliens en la columna más a la izquierda.

```java
public int positionAlienLePlusGauche() {
    int posHaut = Constantes.ALTURA_VENTANA; // Inicializar posición más arriba

    // Iterar sobre los aliens para encontrar la posición más a la izquierda verticalmente
    for (int colonne = 0; colonne < 5; colonne++) {
        for (int ligne = 0; ligne < 8; ligne++) {
            if (this.tabAlien[ligne][colonne] != null) {
                int yPos = this.tabAlien[ligne][colonne].getyPos();

                // Actualizar posición más arriba
                if (yPos < posHaut) {
                    posHaut = yPos;
                }
            }
        }
    }

    return posHaut;
}
}
```