

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



Josué David Figueroa Acosta

CARNÉ: 202307378

SECCIÓN: A

GUATEMALA, 14 DE FEBRERO DEL 2,024

ÍNDICE

ÍNDICE	1
INTRODUCCIÓN	1
OBJETIVOS	1
1. GENERAL	1
2. ESPECÍFICOS	1
ALCANCES DEL SISTEMA	1
ESPECIFICACIÓN TÉCNICA	1
• REQUISITOS DE HARDWARE	1
• REQUISITOS DE SOFTWARE	1
DESCRIPCIÓN DE LA SOLUCIÓN	2
LÓGICA DEL PROGRAMA	2
❖ NOMBRE DE LA CLASE	
Captura de las librerías usadas	2
➤ Librerías	2
➤ Variables Globales de la clase _(El nombre de su clase actual)	3
➤ Función Main	3
➤ Métodos y Funciones utilizadas	3

INTRODUCCIÓN

Este manual técnico está diseñado para proporcionar a los desarrolladores y técnicos toda la información necesaria sobre el juego, incluyendo su arquitectura, requisitos y detalles de implementación. Su propósito es asegurar que cualquier desarrollador pueda comprender, mantener y extender el código fuente del juego.

OBJETIVOS

1. GENERAL

- 1.1. El objetivo de este manual es servir como una guía completa y detallada sobre la estructura y funcionamiento del código fuente del juego, facilitando su mantenimiento y desarrollo futuro.

2. ESPECÍFICOS

- 2.1. Objetivo 1: Proveer una explicación detallada de la estructura del código, incluyendo la arquitectura del sistema, los módulos principales y su interacción.
- 2.2. Objetivo 2: Describir los requisitos técnicos necesarios, tanto de hardware como de software, para trabajar con el código y asegurar un desarrollo continuo y eficiente.

ALCANCES DEL SISTEMA

El objetivo de este manual es ofrecer una visión detallada y técnica del sistema, permitiendo a los desarrolladores entender cómo funciona el código internamente, cómo interactúan los distintos componentes y qué requisitos son necesarios para su correcto funcionamiento y desarrollo futuro.

ESPECIFICACIÓN TÉCNICA

● REQUISITOS DE HARDWARE

- Para trabajar con la aplicación y continuar con el desarrollo, el programador necesita los siguientes requisitos de hardware:

- Procesador: Intel Core i5 o equivalente.
- Memoria RAM: 8 GB o superior.
- Disco Duro: Al menos 500 MB de espacio libre.
- Tarjeta Gráfica: Compatible con OpenGL 3.0 o superior.
- Pantalla: Resolución mínima de 1280x720.

● REQUISITOS DE SOFTWARE

- Para trabajar con la aplicación y continuar con el desarrollo, el programador necesita los siguientes requisitos de software:

- Sistema Operativo: Windows 10, macOS 10.15+, o cualquier distribución de Linux moderna.
- Java Development Kit (JDK): Versión 8 o superior.
- IDE de Desarrollo: IntelliJ IDEA, Eclipse, NetBeans, o cualquier otro IDE compatible con Java.

DESCRIPCIÓN DE LA SOLUCIÓN

- Para diseñar y desarrollar el juego, primero analizamos los requerimientos del enunciado, los cuales establecían la necesidad de un juego interactivo donde un jugador controla un personaje que dispara a enemigos y recoge ítems. A partir de esto, se definieron los siguientes aspectos clave:
 - Arquitectura del Juego: Se decidió utilizar el lenguaje Java debido a su robustez y amplia adopción. La biblioteca Swing fue elegida para manejar la interfaz gráfica, proporcionando una estructura clara y manejable para el desarrollo.
- - Interacción y Controles: Se diseñaron controles intuitivos usando las teclas de dirección para el movimiento del personaje y la barra espaciadora para disparar. Esto garantiza una experiencia de usuario sencilla y accesible.
- - Mecánicas de Juego: Se implementaron mecánicas de juego que incluyen la generación aleatoria de enemigos y ítems, detección de colisiones y cálculo de puntajes. Estos elementos se aseguraron de cumplir con los objetivos del juego, manteniendo la jugabilidad y el desafío.
- - Flujo de Juego: El flujo del juego fue planeado para incluir inicio, pausa, reanudación y finalización, con estados claramente definidos y gestionados a través del código para asegurar una experiencia fluida.

LÓGICA DEL PROGRAMA

❖ NOMBRE DE LA CLASE

```
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.text.Position;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
```

➤ Librerías

- javax.swing.*: Proporciona clases para crear interfaces gráficas de usuario (GUI) con componentes como botones, cuadros de texto, etc.
 - javax.swing.filechooser.FileNameExtensionFilter: Filtra archivos en un cuadro de diálogo de selección de archivos por su extensión.
- javax.swing.text.Position: Representa una posición dentro de un documento de texto.
- java.awt.*: Contiene clases para componentes gráficos de bajo nivel, manejo de eventos, y dibujo.
- java.awt.event.*: Proporciona clases e interfaces para el manejo de eventos de AWT (Abstract Window Toolkit).
- java.io.File: Representa un archivo o directorio en el sistema de archivos.
- java.io.FileInputStream: Permite leer bytes de un archivo.

- java.io.FileOutputStream: Permite escribir bytes en un archivo.
- java.io.IOException: Señala que se ha producido una excepción de E/S (entrada/salida).
- java.io.ObjectInputStream: Deserializa objetos previamente serializados desde un flujo de entrada.
- java.io.ObjectOutputStream: Serializa objetos y los escribe en un flujo de salida.
- java.util.ArrayList: Implementa una lista dinámica basada en un array que puede cambiar de tamaño.
- java.util.List: Interfaz que define una estructura de datos de lista que permite duplicados y acceso secuencial.
- java.util.Random: Genera números aleatorios.

➤ **Variables Globales de la clase _ (El nombre de su clase actual)**

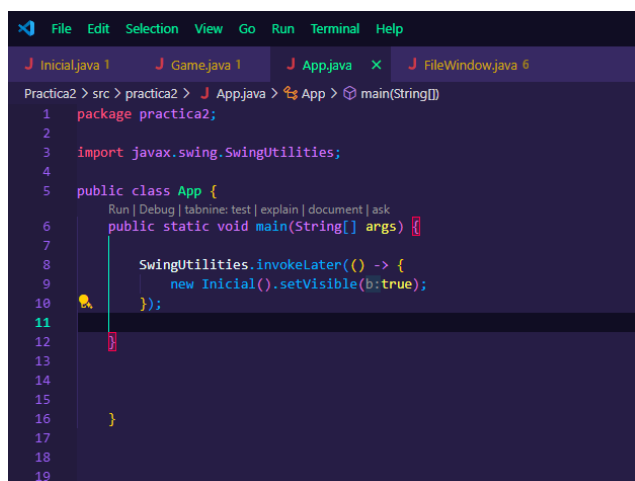
```
private Player player;
private List<Bullet> bullets;
private List<Items> items;
private JLayeredPane panel;
private Enemies enemigosThread;
private ImageIcon backgroundImage;
private MediaPlayer2 mediaPlayer2;
private Timer itemTimer;
private int timeRemaining22;
private JLabel timeLabel;
private JLabel pointsLabel;
private long lastShootTime = 0;
private final long shootCooldown = 500;
private JFrame mainFrame;
private MediaPlayer mediaPlayer;
private volatile boolean timerunning = true;
```

- Player player: Representa al jugador en el juego.
- List<Bullet> bullets: Almacena los proyectiles disparados por el jugador.
- List<Items> items: Almacena los ítems generados en el juego.

- JLayeredPane panel: Panel principal donde se dibujan los elementos del juego.
- Enemies enemigosThread: Hilo que controla el comportamiento de los enemigos.
- ImageIcon backgroundImage: Imagen de fondo del juego.
- MediaPlayer2 musicPlayer2: Reproduce la música y efectos de sonido del juego.
- int timeRemaining22: Tiempo restante del juego en segundos.
- JLabel timeLabel: Etiqueta que muestra el tiempo restante.
- JLabel pointsLabel: Etiqueta que muestra los puntos del jugador.
- long lastShootTime: Marca de tiempo del último disparo.
- final long shootCooldown: Intervalo mínimo entre disparos consecutivos.
- JFrame mainFrame: Marco principal de la aplicación.
- MediaPlayer musicPlayer: Reproduce la música de fondo del juego.
- volatile boolean timerunning: Indica si el temporizador del juego está en funcionamiento.

➤ Función Main

La función Main fue utilizada para poder ejecutar el inicio del programa.



```

File Edit Selection View Go Run Terminal Help
J Inicial.java 1 J Game.java 1 J App.java x J FileWindow.java 6
Practica2 > src > practica2 > J App.java > App > main(String[])
1 package practica2;
2
3 import javax.swing.SwingUtilities;
4
5 public class App {
6     public static void main(String[] args) {
7
8         SwingUtilities.invokeLater(() -> {
9             new Inicial().setVisible(b:true);
10        });
11
12
13
14
15
16    }
17
18
19

```


➤ Procedimientos, métodos y Funciones utilizadas

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

public Game(JFrame frame)

- Constructor de la clase **Game**. Inicializa los componentes del juego, configura el panel, el jugador, los enemigos y los temporizadores. También se encarga de iniciar la música de fondo y configurar los escuchadores de eventos.

```
40  public Game(JFrame frame) {
```

@Override public void paintComponent(Graphics g)

- Sobrescribe el método **paintComponent** para dibujar el fondo, el jugador y las balas en el panel del juego. Utiliza **Graphics2D** para realizar el dibujo.

```
167  public void paintComponent(Graphics g) {
```

@Override public void actionPerformed(ActionEvent e)

- Método que se ejecuta cada vez que el temporizador del juego se activa. Mueve al jugador, las balas y los ítems, verifica las colisiones y actualiza el panel de dibujo.

```
181  public void actionPerformed(ActionEvent e) {
```

private void checkCollisions()

- Verifica las colisiones entre las balas y los enemigos, los ítems y el jugador, y entre los enemigos y el jugador. Actualiza el estado del juego en consecuencia (remueve enemigos destruidos, actualiza puntos, etc.).

```
193  private void checkCollisions() {
```

private void generateRandomItems()

- Genera ítems aleatorios en posiciones aleatorias del panel. Los ítems pueden ser de diferentes tipos y se añaden a la lista de ítems y al panel.

```
268  private void generateRandomItems() {
```

private void addTime(int seconds)

- Añade o resta tiempo al temporizador del juego. Actualiza la etiqueta que muestra el tiempo restante.

```
283 private void addTime(int seconds) {
```

private void updatePointsLabel()

- Actualiza la etiqueta que muestra los puntos del jugador. Cambia el texto de la etiqueta según los puntos actuales del jugador.

```
289 private void updatePointsLabel() {
```

private void showExplosion(int x, int y)

- Muestra una animación de explosión en una posición específica del panel. La animación se elimina después de un breve período de tiempo.

```
294 private void showExplosion(int x, int y) {
```

private void closeAndOpenInitial()

- Cierra el juego actual y abre la ventana inicial del juego. Detiene todos los temporizadores e hilos, y limpia el panel del juego.

```
428 private void closeAndOpenInitial() {
```

private void showEndGameDialog(String message)

- Muestra un diálogo al final del juego con un mensaje personalizado (como "¡Game Over!" o "¡Has ganado!"). Pide al jugador su nombre para guardar su puntuación y reinicia el juego.

```
450 private void showEndGameDialog(String message) {
```

public void stoptime()

- Detiene el temporizador principal del juego estableciendo `timerunning` en `false`.

```
460 public void stoptime(){
```

public void saveGameData()

- Guarda el estado actual del juego (incluyendo la posición de los enemigos, los puntos del jugador y el tiempo restante) en un archivo

binario. Permite al jugador seleccionar el nombre del archivo y la ubicación de guardado mediante un **JFileChooser**.

```
464  public void saveGameData() {
```

private String getImagePathForColumn(int columnIndex)

- Devuelve la ruta de la imagen correspondiente a una columna específica de enemigos. Se utiliza al cargar datos del juego para determinar qué imagen usar para los enemigos.

```
505  private String getImagePathForColumn(int columnIndex) {
```

public void loadGameData()

- Carga el estado del juego desde un archivo binario. Restaura la posición de los enemigos, los puntos del jugador y el tiempo restante. Permite al jugador seleccionar el archivo de carga mediante un **JFileChooser**.

```
520  public void loadGameData() {
```

public void Startgame()

- Inicia el temporizador del juego, el temporizador de ítems y el hilo de los enemigos. Se utiliza para reanudar el juego desde un estado guardado.

```
593  public void Startgame(){
```