Universidad de San Carlos de Guatemala.

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Introducción a la Programación y Computación 1 Sección C

Catedrático: Ing. Moisés Velásquez

Tutor académico: Carlos Campanero, William Corado

Practica no. 3

Manual Técnico

Nombre: Diego Enrique Arriaga Meléndez

Canet: 202003892

Fecha: 2/04/2021

El presente documento que va dirigido a un público con conocimientos técnicos sobre el área de programación. En este se encuentran los recursos utilizados para la realización del programa, entre estos recursos se encuentran el IDE (Integrated Development Environment) que se utilizó, el lenguaje en el que se programó y el equipo utilizado para desarrollar el programa.

Objetivos del Programa

GENERALES:

Otorgar al usuario un juego entretenido y fácil de entender para que cualquier persona pueda jugarlo.

ESPECIFICOS:

- Crear un juego al cual se pueda jugar en los momentos de aburrimiento.
- Diseñar un juego accesible para cualquier persona sin importar su edad.
- Diseñar una interfaz intuitiva, para que el usuario pueda moverse fácilmente en ella.

Índice

Especificaciones Técnicas

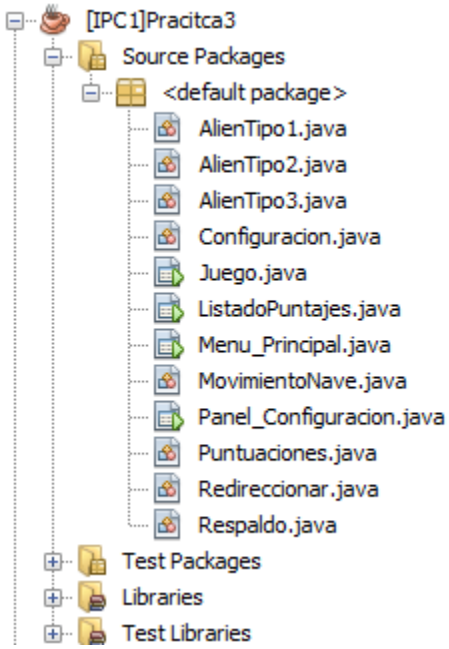
Hardware utilizado

- Maquina: Computadora Portátil HP 14-dq1005la.
- Procesador: Intel™ Core™ i7-1065G7 CPU @ 1.30GHz 1.50 GHz.
- RAM instalada: 8.00 GB (7.70 GB utilizable).
- Unidad de estado sólido de 256 GB Intel® + Memoria de 16 GB Intel® Optane™.
- Monitor LED de alta definición, diagonal 14,0 pulgadas.

Software utilizado

- Sistema perative: Windows 10 Home Single Language.
- Versión: 20H2.
- Lenguaje de Programación e IDE: Java.
- IDE: Neatbeans 8.2
- JDK: jdk1.8.0_111.

Listado de clases creadas.



Antes de explicar el código hay que aclarar que para diseñar los *JFrames* se utilizo la función *drag and drop* de *NetBeans*. Por lo que hay código que se genera automáticamente.

Objetos globales.

Existen dos clases tipo java que contienen objetos que son llamados por otras clases, estas clases son **Configuracion.java** y **Puntuaciones.java**.

Configuracion.java

En esta clase se encuentran diferentes variable las cuales funcionan como indicadores de las condiciones con las que se iniciaran un nuevo juego.

```java
public class Configuracion {

    static  int tiempo=90,movimiento=10,frecuencia=10000, puntos=0,derrotatipo1=0,
            derrotatipo2=0,derrotatipo3=0;
    //Para determinar si los objetos aparezcan o no
    static boolean mastiempo=false,ptextras=false,masvelocidad=false,menostiempo=false,
                   penalizacion=false,congelacion=false,derota=false;

    public Configuracion() {
    }

    //Setters
    public void setTiempo(int tiempo) {
        this.tiempo = tiempo;
    }

    public void setPuntos(int puntos) {
        this.puntos = puntos;
    }

    public static void setMovimiento(int movimiento) {
        Configuracion.movimiento = movimiento;
    }

    public void setFrecuencia(int frecuencia) {
        this.frecuencia = frecuencia;
    }

}
```

Puntuaciones.java

Esta clase posee un par de arreglos, uno que almacena las puntuaciones conseguidas en cada juego y el nombre del jugador que las consiguió. También posee un método que se encarga de almacenar una nueva puntuación al terminar cada partida.

```java
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;


public class Puntuaciones implements Serializable{
    //Se crean las variables que serviran para guardar los puntajes
    static int cont=0,puntuacion[]=new int[cont];
    static String apodo[]=new String[cont];
    //Constructor
    public Puntuaciones() {
    }
```

```java
//Metodo que se encarga de
public void nuevopuntaje(String nombre,int pt){
    if (cont==0) {
        cont++;
        puntuacion=new int[cont];
        apodo=new String[cont];
        puntuacion[cont-1]=pt;
        apodo[cont-1]=nombre;
    }else{
        //Se guardan los valores antiguos
        String ap[]=apodo;
        int puntu[]=puntuacion,con=cont;
        //Se actualizan los arreglos
        cont++;
        puntuacion=new int[cont];
        apodo=new String[cont];
        for (int i = 0; i < con; i++) {
            apodo[i]=ap[i];
            puntuacion[i]=puntu[i];
        }
        apodo[con]=nombre;
        puntuacion[con]=pt;
    }
}

//Getters y Setters

public static int getCont() {
    return cont;
}

public static int[] getPuntuacion() {
    return puntuacion;
}

public static String[] getApodo() {
    return apodo;
}

public static void setCont(int cont) {
    Puntuaciones.cont = cont;
}

public static void setPuntuacion(int[] puntuacion) {
    Puntuaciones.puntuacion = puntuacion;
}

public static void setAcronimo(String[] apodo) {
    Puntuaciones.apodo = apodo;
}
```

Clase Serializable

Una de las funciones principales del programa es guardar los puntajes conseguidos en cada partida y que estos se guarden incluso después de cerrar el juego. Por lo mismo hay una clase que se encarga de almacenar la información que posea la clase **Puntuacione.java** en variables privadas y no *static*, la clase encargada de esto es **Respaldo.java**.

```java
import java.io.Serializable;


public class Respaldo implements Serializable{
    //Se llama a la clase con las puntuaciones
    Puntuaciones puntuaciones=new Puntuaciones();
    //Se crean las variables que serviran para guardar los puntajes
    private int cont=0,puntuacion[]=new int[cont];
    private String apodo[]=new String[cont];

    //Constructor
    public Respaldo() {
        this.cont=puntuaciones.cont;
        this.puntuacion=puntuaciones.puntuacion;
        this.apodo=puntuaciones.apodo;
    }

    //Getters

    public int getCont() {
        return cont;
    }

    public int[] getPuntuacion() {
        return puntuacion;
    }

    public String[] getApodo() {
        return apodo;
    }

}
```

El constructor se encarga de que cada vez que se llame a la clase esta reciba de manera automática la información de la clase **Puntuaciones.java**.

El método encargado de guardar la información en un archivo binario es el siguiente:

```java
//Guardar el puntaje
respaldo=new Respaldo();
try {
//Crear documento tipo binario
ObjectOutputStream oos;
oos=new ObjectOutputStream(new FileOutputStream("datos.bin"));
//Se elige la clase que se transformara en el documento binario
oos.writeObject(respaldo);
oos.close();
} catch (Exception e) {
    System.out.println(e);
}
```

Este método se encuentra únicamente en la clase **Juego.java**.

El método encargado de cargar el archivo cada vez que se abre el programa es:

```
Respaldo respaldo=new Respaldo();
//Se cargan las puntuaciones antiguas
ObjectInputStream ois;
try {
    //Tomar documento tipo binario
    ois=new ObjectInputStream(new FileInputStream("datos.bin"));
    //Se carga el valor del documento binario a la clase necesario
    Respaldo valores=(Respaldo)ois.readObject();

    puntuaciones.cont=valores.getCont();
    puntuaciones.apodo=valores.getApodo();
    puntuaciones.puntuacion=valores.getPuntuacion();
} catch (Exception e) {
}
```

Este método se encuentra únicamente en la clase **Menu_Principal.java**.

Clase **Menu_Principal.java**

Esta es la clase principal del programa, y genera el menú de inicio del juego

La primera parte del código que se creo en la clase es la encargada de cargar el archivo binario en el programa.

```
public Menu_Principal() {
    initComponents();
    //Se llama a la clase de puntuaciones
    Puntuaciones puntuaciones=new Puntuaciones();
    Respaldo respaldo=new Respaldo();
    //Se cargan las puntuaciones antiguas
    ObjectInputStream ois;
    try {
        //Tomar documento tipo binario
        ois=new ObjectInputStream(new FileInputStream("datos.bin"));
        //Se carga el valor del documento binario a la clase necesario
        Respaldo valores=(Respaldo)ois.readObject();

        puntuaciones.cont=valores.getCont();
        puntuaciones.apodo=valores.getApodo();
        puntuaciones.puntuacion=valores.getPuntuacion();
    } catch (Exception e) {
    }
}
```

El siguiente método es el que sirve al botón *Salir* para cerrar el juego en la ventana principal.

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    System.gc();
    System.exit(0);
}
```

Los siguientes métodos de la clase se encarga de mandar información a la clase **Redireccionar.java** para que esta se encargue de abrir la ventana respectiva al botón que presiono el botón y cerrar la ventana actual.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    //Se llama a la clase para redirigir el programa
    Redireccionar redi=new Redireccionar(1);
    dispose();
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    Redireccionar redi=new Redireccionar(2);
    dispose();
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    Redireccionar redi=new Redireccionar(3);
    dispose();
}
```

Clase **Redireccionar.java**

Esta es la clase que se encarga de abrir la ventana necesaria.

```
public class Redireccionar {

    public Redireccionar(int a) {
        //Se abre el frame dependiendo la opción que elija el usuario
        switch(a){
            case 1:
                //Se llaman a la clase necesaria
                Juego juego=new Juego();
                juego.setVisible(true);
                break;
            case 2:
                //Se llaman a la clase necesaria
                Panel_Configuracion configuracion=new Panel_Configuracion();
                configuracion.setVisible(true);
                break;
            case 3:
                //Se llaman a la clase necesaria
                ListadoPuntajes pt=new ListadoPuntajes();
                pt.setVisible(true);
                break;
            default:
                break;
        }
    }

}
```

Clase **Panel_Configuracion.java**

La siguiente clase sirve para que el jugador pueda modificar las características del juego.


```
public class Panel_Configuracion extends javax.swing.JFrame {
    //Se mandan a llamar a las clases necesarias
    Menu_Principal menu=new Menu_Principal();
    Configuracion confi=new Configuracion();
    public Panel_Configuracion() {
        initComponents();

         //Se establece el layout de los panels
         jPanel1.setLayout(null);
         jPanel2.setLayout(null);
         jPanel3.setLayout(null);
         panelItems.setLayout(null);
```

```
//Se establecen iconos para los labels
ImageIcon img=new ImageIcon("corazon.png");
Icon icono1=new  ImageIcon(img.getImage().getScaledInstance(labelCorazon.getWidth(), labelCorazon.getHeight(), Image.SCALE_DEFAULT));
labelCorazon.setIcon(icono1);

ImageIcon img2=new ImageIcon("mando.png");
Icon icono2=new  ImageIcon(img2.getImage().getScaledInstance(labelmando.getWidth(), labelmando.getHeight(), Image.SCALE_DEFAULT));
labelmando.setIcon(icono2);
```

```
        //Determinar si algun objeto ya esta seleccionado
        if (confi.mastiempo==true) {
            jRadioButton1.setSelected(true);
        }
        if (confi.ptextras==true) {
            jRadioButton2.setSelected(true);
        }
        if (confi.masvelocidad==true) {
            jRadioButton3.setSelected(true);
        }
        if (confi.menostiempo==true) {
            jRadioButton4.setSelected(true);
        }
        if (confi.penalizacion==true) {
            jRadioButton5.setSelected(true);
        }
        if (confi.congelacion==true) {
            jRadioButton6.setSelected(true);
        }
    }
```

Botón para cerrar la ventana.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    menu.setVisible(true);
    System.gc();
    dispose();
}
```

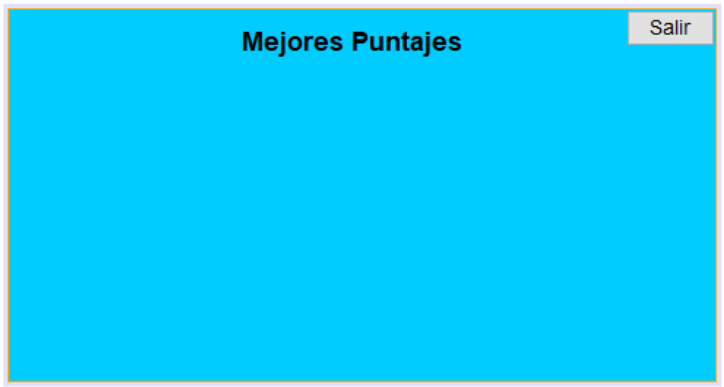
Botón para guardar los cambios.

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    //Se establece la frecuencia de salida de los objetos
    String frecuencia=boxFrecuencias.getSelectedItem().toString();
    switch(frecuencia){
        case "Poco frecuente":
            confi.setFrecuencia(15000);
            break;
        case "Normal":
            confi.setFrecuencia(10000);
            break;
        case "Muy frecuente":
            confi.setFrecuencia(5000);
            break;
    }

    //Se establece la velocidad de movimiento
    String velocidad=nivelVelocidad.getSelectedItem().toString();
    switch(velocidad){
        case "Normal":
            confi.setMovimiento(10);
            break;
        case "Rapido":
            confi.setMovimiento(20);
            break;
    }
```

```
    confi.setTiempo(tiempo);

    //Se establece si el objeto esta seleccionado o no
    if (jRadioButton1.isSelected()==true) {
        confi.mastiempo=true;
    }else if(jRadioButton1.isSelected()==false){confi.mastiempo=false;}

    if (jRadioButton2.isSelected()==true) {
        confi.ptextras=true;
    }else if(jRadioButton2.isSelected()==false){confi.ptextras=false;}

    if (jRadioButton3.isSelected()==true) {
        confi.masvelocidad=true;
    }else if(jRadioButton3.isSelected()==false){confi.masvelocidad=false;}

    if (jRadioButton4.isSelected()==true) {
        confi.menostiempo=true;
    }else if(jRadioButton4.isSelected()==false){confi.menostiempo=false;}

    if (jRadioButton5.isSelected()==true) {
        confi.penalizacion=true;
    }else if(jRadioButton5.isSelected()==false){confi.penalizacion=false;}

    if (jRadioButton6.isSelected()==true) {
        confi.congelacion=true;
    }else if(jRadioButton6.isSelected()==false){confi.congelacion=false;}

    JOptionPane.showMessageDialog(null,"Se guardo la configuracion del juego");
}
```

Clase **ListadoPuntajes.java**

La siguiente se encarga de acomodar los arreglos de puntajes y nombres en orden para poder mostrar los 5 mejores puntajes.



```java
public ListadoPuntajes() {
    initComponents();
    //Se le da layout al panel
    jPanell.setLayout(null);
    //Se crea un contador que depende de cuantos puntajes se posee
    int cont=0;
    if (puntuaciones.cont<5) {
        cont=puntuaciones.cont;
    }else{
        cont=5;
    }

    //Arreglos que almacenan los puntajes y los nombres
    String [] nombre=puntuaciones.apodo;
    int [] pt=puntuaciones.puntuacion;

     //Metodo burbuja
     int aux;
     String aux2;
     for (int i = 0; i < pt.length - 1; i++) {
         for (int j = 0; j < pt.length - i - 1; j++) {
             if (pt[j + 1] < pt[j]) {
                 //Ordenar nombres
                 aux2 = nombre[j + 1];
                 nombre[j + 1] = nombre[j];
                 nombre[j] = aux2;
                 //Ordenar puntos
                 aux = pt[j + 1];
                 pt[j + 1] = pt[j];
                 pt[j] = aux;
             }
         }
     }
     //Se crea variable para nuevo modelo de tabla
     DefaultTableModel modelo=new DefaultTableModel();

     //Se agregan columnas
     modelo.addColumn("Nombre");
     modelo.addColumn("Puntaje");

      //Se agregan filas
      int a=0;
      Object fila[]=new Object[2];
      for (int i = puntuaciones.cont-1; i >=0; i--) {
          if (a<cont) {
              fila[0]=nombre[i];
              fila[1]=pt[i];
              modelo.addRow(fila);
          }else{break;}
          a++;
      }
      //Se crea la tabla
      JTable listadopuntajes= new JTable(modelo);

      //Estilo de letra
      listadopuntajes.setFont(new Font("Arial", Font.PLAIN, 14));

      //Se crea el JScrollPane
      JScrollPane sp=new JScrollPane(listadopuntajes);
      sp.setBounds(100, 50, 300, 110);

      //Se agrega al panel
      jPanell.add(sp);
  }
```

Botón para cerrar la ventana y regresar a la ventana principal.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    menu.setVisible(true);
    dispose();
}
```

Clase **MovimientoNave.java**

La siguiente clase sirve como hilo que se encarga del movimiento de la nave en el juego.

```
public class MovimientoNave extends JFrame implements Runnable{
    //Se importa la clase con la configuración del juego
     Configuracion config=new Configuracion();
    //Label que guarda la imagen del juego
    JLabel lnave=new JLabel();
    //Variable de la posición en y de la nave
    static int y=280;
    double movimiento=config.movimiento;
    boolean a=true;
```

```
//Metodo para crear la nave en java
@Override
public void run() {
    while(a){
        lnave.setBounds(10,y, 45,45);
        ImageIcon img=new ImageIcon("nave.png");
        Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(lnave.getWidth(), lnave.getHeight(), Image.SCALE_DEFAULT));
        lnave.setIcon(iconol);
    }
}
```

Método que se llama cuando se pierde la partida porque se acaba el tiempo o porque los enemigos alcanzaron el extremo izquierdo de la pantalla.

```
//Metodo en caso que el jugador pierda
public void derrota(){
    lnave.setBounds(10,y, 45,45);
    ImageIcon img=new ImageIcon("explosion.png");
    Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(lnave.getWidth(), lnave.getHeight(), Image.SCALE_DEFAULT));
    lnave.setIcon(iconol);

}
```

```
//Metodo para darle una configuración de posición al label
public void setLnave(JLabel lnave) {
    this.lnave = lnave;
}

//Dar nuevo valor en y
public void sumaY(int y) {
    this.y +=y;
}
```

Movimiento de los Extraterrestres.

En el juego hay tres tipos de enemigos, el movimiento de cada uno se maneja en una clase distinta pero cada una es igual, la razón de esto es para poder diferenciarlo al momento de determinar una colisión entre el disparo de la nave y un enemigo, las clases que se encargan de los enemigos son **AlienTipo1.java**, **AlienTipo2.java** y **AlienTipo3.java**.

```
public class AlienTipo1 extends JFrame implements Runnable{
    //Se importa la clase con la configuración del juego
    Configuracion config=new Configuracion();
    AlienTipo2 tipo2=new AlienTipo2();
    AlienTipo3 tipo3=new AlienTipo3();

    JLabel label=new JLabel();
    int posicionx=1014,posiciony,movimiento=config.movimiento,
            desplazamientoy=21,vida=2,x=posicionx;

    @Override
    public void run() {
        x=posicionx;
        while (x>= 0) {
            //Se comprueba la vida del alien
            if (vida<=0) {
                config.puntos+=10;
                config.derrotatipo1++;
                break;
            }
```

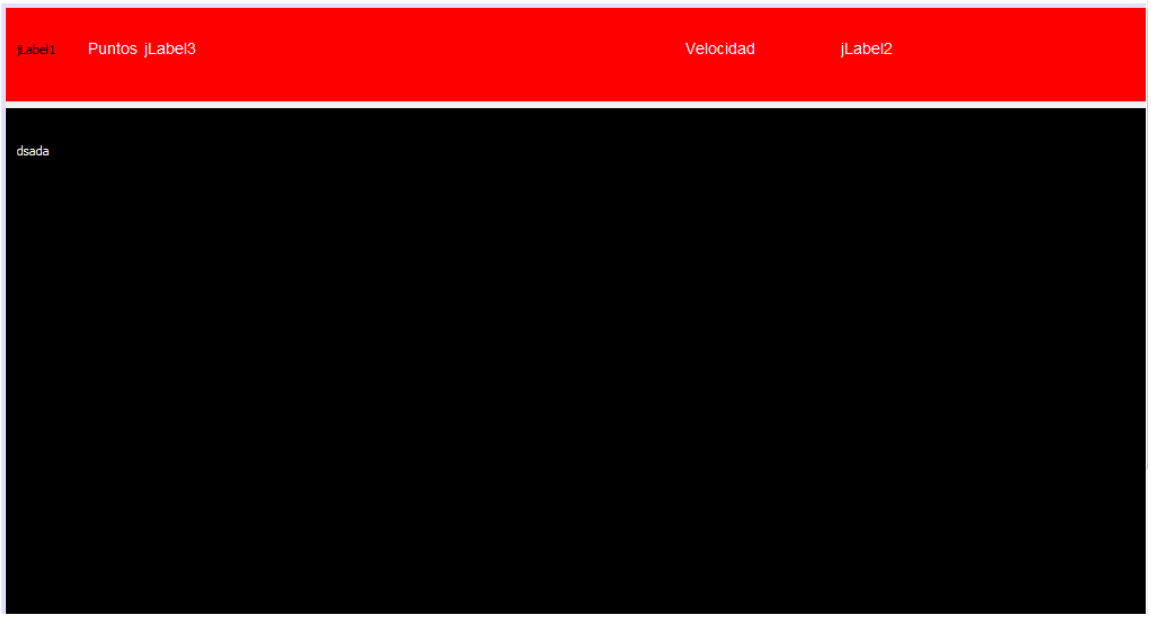
```
    //Mover el label en y
    posiciony+=desplazamientoy;
    label.setBounds(x,posiciony, 45,45);
    ImageIcon img=new ImageIcon("alien1.png");
    Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(label.getWidth(), label.getHeight(), Image.SCALE_DEFAULT));
    label.setIcon(iconol);
    //Se cambia la direccion que se movera en y
    desplazamientoy*=-1;
    //Tiempo de espara antes de volver a moverse
    try {
        Thread.sleep(500);
    } catch (InterruptedException ex) {
        Logger.getLogger(AlienTipo1.class.getName()).log(Level.SEVERE, null, ex);
    }
    x-=movimiento;
}
```

```
    //En caso de que derrotaran al alien
    if (vida<=0) {
        label.setBounds(x,posiciony, 45,45);
        ImageIcon img=new ImageIcon("explosion.png");
        Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(label.getWidth(), label.getHeight(), Image.SCALE_DEFAULT));
        label.setIcon(iconol);
        try {
            //Tiempo de espera
            Thread.sleep(500);
        } catch (InterruptedException ex) {
            Logger.getLogger(AlienTipo1.class.getName()).log(Level.SEVERE, null, ex);
        }
        label.setBounds(1200,1200, 45,45);
        label.setText("");
        label.setIcon(null);
    }
    //En caso de que el alien haya llegado al final
    if (x<=0) {
        config.derota=true;
    }
}
```

```
    //Setters
    public void setLabel(JLabel label) {
        this.label = label;
    }

    public void setPosicionx(int posicionx) {
        this.posicionx = posicionx;
    }

    public void setPosiciony(int posiciony) {
        this.posiciony = posiciony;
    }

    public void setMovimiento(int movimiento) {
        this.movimiento = movimiento;
    }

    //Getters
    public JLabel getLabel() {
        return label;
    }
}
```

Clase **Juego.java**.

Esta clase es la que se encarga de generar la partida.



Esta clase posee varias subclases que servirán como hilos de diferentes acciones dentro del juego.

```
public class Juego extends javax.swing.JFrame {
    //Se llama al menu de inicio
    Menu_Principal menu=new Menu_Principal();
    //Se manda a llamar la clase con la configuración del juego y listado de puntaje
    Configuracion config=new Configuracion();
    Puntuaciones puntuaciones=new Puntuaciones();
    Respaldo respaldo=new Respaldo();
    //Se manda a llamar las clases que contienen los hilos necesarios
    MovimientoNave mn=new MovimientoNave();
    AlienTipo1 atipo1[]=new AlienTipo1[8];
    AlienTipo2 atipo2[]=new AlienTipo2[16];
    AlienTipo3 atipo3[]=new AlienTipo3[16];
    //Arreglos de hilos para los tipos de alien
    Thread htipo1[]=new Thread[8], htipo2[]=new Thread[16], htipo3[]=new Thread[16];


    //Variables que reciben el valor de la configuracion
    int frecuencia=config.frecuencia,puntos=config.puntos,movimiento=config.movimiento,
        movimientonave=config.movimiento,tiempo_ex;
```

Subclase Aparcion_Objetos

Esta subclase se encarga de que en la pantalla aparezcan los objetos que puede tocar el jugador, y la acción que realizan cuando el jugador lo toca.

```
//Clase para la apararicion de objetos
class Aparicion_Objetos implements Runnable{
    //Posiciones X y Y de los 6 objetos
    int x1,y1;
    int tiempo=config.tiempo;
    @Override
    public void run() {
        while (tiempo>0) {
            //Variable para la aparición random de un objeto
            int c=(int)(Math.random()*(6-1+1)+1);
            //Condicional para que salga el que da mas tiempo
            if (config.mastiempo==true && c==1) {
                //Se dan un valor random a la posicion inicial del obeto
                x1=(int)(Math.random()*(900-300+1)+300);
                y1=(int)(Math.random()*(515-50+1)+50);
                //Movimiento del item
                while (x1>-65) {
                    int a=Math.abs((x1+45)-55),
                    b=Math.abs((y1+45)-(mn.y+45));
                    if ((0 <= a && a< 45) && (0 <= b && b< 45)) {
                        tiempo+=10;
                        Litem.setIcon(null);
                        break;
                    }
```

```
    //Se le da una posicion y una imagen al objeto
    Litem.setBounds(x1,y1, 45,45);
    ImageIcon img=new ImageIcon("mastiempo.png");
    Icon icono1=new  ImageIcon(img.getImage().getScaledInstance(Litem.getWidth(), Litem.getHeight(), Image.SCALE_DEFAULT));
    Litem.setIcon(icono1);
    //Se mueve al objeto
    x1-=movimiento;
    //Se espera para que se vuelva a mover el objeto
    try {
        Thread.sleep(300);
    } catch (InterruptedException ex) {
        Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
    }
}
Litem.setIcon(null);
```

```
        //Tiempo de espera para el siguiente item
        try {
            Thread.sleep(frecuencia);
        } catch (InterruptedException ex) {
            Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    //Condicional para que salga el que da mas puntos
    if (config.ptextras==true && c==2) {
        //Se dan un valor random a la posicion inicial del obeto
        x1=(int)(Math.random()*(900-300+1)+300);
        y1=(int)(Math.random()*(515-50+1)+50);
        //Movimiento del item
        while (x1>-65) {
            int a=Math.abs((x1+45)-55),
            b=Math.abs((y1+45)-(mn.y+45));
            if ((0 <= a && a< 45) && (0 <= b && b< 45)) {
                puntos+=10;
                Litem.setIcon(null);
                break;
            }
            //Se le da una posicion y una imagen al objeto
            Litem.setBounds(x1,y1, 45,45);
            ImageIcon img=new ImageIcon("maspuntos.png");
            Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(Litem.getWidth(), Litem.getHeight(), Image.SCALE_DEFAULT));
            Litem.setIcon(iconol);
            //Se mueve al objeto
            x1-=movimiento;
            //Se espera para que se vuelva a mover el objeto
            try {
                Thread.sleep(300);
            } catch (InterruptedException ex) {
                Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        Litem.setIcon(null);
        //Tiempo de espera para el siguiente item
        try {
            Thread.sleep(frecuencia);
        } catch (InterruptedException ex) {
            Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    //Condicional para que salga el que aumenta la velocidad
    if (config.masvelocidad==true && c==3) {
        //Se dan un valor random a la posicion inicial del obeto
        x1=(int)(Math.random()*(900-300+1)+300);
        y1=(int)(Math.random()*(515-50+1)+50);
        //Movimiento del item
        while (x1>-65) {
            int a=Math.abs((x1+45)-55),
            b=Math.abs((y1+45)-(mn.y+45));
            if ((0 <= a && a< 45) && (0 <= b && b< 45)) {
                movimientonave+=10;
                Litem.setIcon(null);
                break;
            }
            //Se le da una posicion y una imagen al objeto
            Litem.setBounds(x1,y1, 45,45);
            ImageIcon img=new ImageIcon("velocidad.png");
            Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(Litem.getWidth(), Litem.getHeight(), Image.SCALE_DEFAULT));
            Litem.setIcon(iconol);
            //Se mueve al objeto
            x1-=movimiento;
            //Se espera para que se vuelva a mover el objeto
            try {
                Thread.sleep(300);
            } catch (InterruptedException ex) {
                Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        Litem.setIcon(null);
        //Tiempo de espera para el siguiente item
        try {
            Thread.sleep(frecuencia);
        } catch (InterruptedException ex) {
            Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

```
//Condicional para que salga el que reduzca el tiempo
if (config.menostiempo==true && c==4) {
    //Se dan un valor random a la posicion inicial del obeto
    x1=(int)(Math.random()*(900-300+1)+300);
    y1=(int)(Math.random()*(515-50+1)+50);
    //Movimiento del item
    while (x1>-65) {
        int a=Math.abs((x1+45)-55),
        b=Math.abs((y1+45)-(mn.y+45));
        if ((0 <= a && a< 45) && (0 <= b && b< 45)) {
            tiempo-=10;
            Litem.setIcon(null);
            break;
        }
        //Se le da una posicion y una imagen al objeto
        Litem.setBounds(x1,y1, 45,45);
        ImageIcon img=new ImageIcon("menostiempo.png");
        Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(Litem.getWidth(), Litem.getHeight(), Image.SCALE_DEFAULT));
        Litem.setIcon(iconol);
        //Se mueve al objeto
        x1-=movimiento;
        //Se espera para que se vuelva a mover el objeto
        try {
            Thread.sleep(300);
        } catch (InterruptedException ex) {
            Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    Litem.setIcon(null);
    //Tiempo de espera para el siguiente item
    try {
        Thread.sleep(frecuencia);
    } catch (InterruptedException ex) {
        Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

```
//Condicional para que el que reduce los puntos
if (config.penalizacion==true && c==5) {
    //Se dan un valor random a la posicion inicial del obeto
    x1=(int)(Math.random()*(900-300+1)+300);
    y1=(int)(Math.random()*(515-50+1)+50);
    //Movimiento del item
    while (x1>-65) {
        int a=Math.abs((x1+45)-55),
        b=Math.abs((y1+45)-(mn.y+45));
        //Condicion para ver si los labels se esta tocando
        if ((0 <= a && a< 45) && (0 <= b && b< 45)) {
            //Condicion para comprobar que los puntos no queden negativos
            if (puntos>=10) {
                puntos-=10;
            }else{puntos=0;}
            Litem.setIcon(null);
            break;
        }
        //Se le da una posicion y una imagen al objeto
        Litem.setBounds(x1,y1, 45,45);
        ImageIcon img=new ImageIcon("menospuntos.png");
        Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(Litem.getWidth(), Litem.getHeight(), Image.SCALE_DEFAULT));
        Litem.setIcon(iconol);
        //Se mueve al objeto
        x1-=movimiento;
        //Se espera para que se vuelva a mover el objeto
        try {
            Thread.sleep(300);
        } catch (InterruptedException ex) {
            Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    Litem.setIcon(null);
    //Tiempo de espera para el siguiente item
    try {
        Thread.sleep(frecuencia);
    } catch (InterruptedException ex) {
        Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

```
//Condicional para que salga el que congela la nave
if (config.congelacion==true && c==6) {
    //Se dan un valor random a la posicion inicial del obeto
    xl=(int)(Math.random()*(900-300+1)+300);
    yl=(int)(Math.random()*(515-50+1)+50);
    //Movimiento del item
    while (xl>-65) {
        int a=Math.abs((xl+45)-55),
        b=Math.abs((yl+45)-(mn.y+45));
        if ((0 <= a && a< 45) && (0 <= b && b< 45)) {
            ImageIcon img=new ImageIcon("navecongelada.png");
            Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(labelNave.getWidth(), labelNave.getHeight(), Image.SCALE_DEFAULT));
            labelNave.setIcon(iconol);
            Litem.setIcon(null);
            hilol.stop();
            break;
        }
        //Se le da una posicion y una imagen al objeto
        Litem.setBounds(xl,yl, 45,45);
        ImageIcon img=new ImageIcon("congelacion.png");
        Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(Litem.getWidth(), Litem.getHeight(), Image.SCALE_DEFAULT));
        Litem.setIcon(iconol);
        //Se mueve al objeto
        xl-=movimiento;
        //Se espera para que se vuelva a mover el objeto
        try {
            Thread.sleep(300);
        } catch (InterruptedException ex) {
            Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    Litem.setIcon(null);
                //Tiempo de espera para el siguiente item
                try {
                    Thread.sleep(5000);
                    mn.setLnave(labelNave);
                    hilol=new Thread(mn);
                    hilol.start();
                    Thread.sleep(frecuencia-5000);
                } catch (InterruptedException ex) {
                    Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
    }
}
Aparicion_Objetos items=new Aparicion_Objetos();
```

Subclase Reloj

Esta subclase sirve como el hilo que se encarga de llevar control del tiempo de juego y actualizar el marcador en el encabezado.

```
//Clase para la función de reloj
class Reloj implements Runnable{
    //Guardar la configuración del tiempo
    int tiempo;
    @Override
    public void run() {
        boolean a=true;
        while (a) {
            tiempo=items.tiempo;
            labelreloj.setText(tiempo+" s");
            tiempo_ex=this.tiempo;
            try {
                Thread.sleep(1000);
                items.tiempo+=-1;
            }catch (InterruptedException ex) {
                Logger.getLogger(Reloj.class.getName()).log(Level.SEVERE, null, ex);
            }
            //En caso de que se acabe el tiempo
            if (tiempo<=0) {
                a=false;
            }
            labelreloj.setText(tiempo+" s");
        }
        //Se detiene el hilo
        mn.a=false;
```

```
//Se cambia el icono de la nave
ImageIcon img=new ImageIcon("explosion.png");
Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(labelNave.getWidth(), labelNave.getHeight(), Image.SCALE_DEFAULT));
labelNave.setIcon(iconol);

hilo1.stop();
hilo3.stop();
hilo4.stop();
hilo5.stop();
for (int j = 0; j < 8; j++) {
            htipo1[j].stop();
        }
        for (int j = 0; j < 16; j++) {
            htipo2[j].stop();
        }
        for (int j = 0; j < 16; j++) {
            htipo3[j].stop();
        }
//Se regresa al menú principal
JOptionPane.showMessageDialog(null,"Se termino el tiempo");
JOptionPane.showMessageDialog(null, "Su punttuacion es de "+puntos);
String acronimo=JOptionPane.showInputDialog(null,"Escriba su nombre");
puntuaciones.nuevopuntaje(acronimo, puntos);
        //Guardar el puntaje
        respaldo=new Respaldo();
        try {
        //Crear documento tipo binario
        ObjectOutputStream oos;
        oos=new ObjectOutputStream(new FileOutputStream("datos.bin"));
        //Se elige la clase que se transformara en el documento binario
        oos.writeObject(respaldo);
        oos.close();
        } catch (Exception e) {
            System.out.println(e);
        }
        //Regresar al menu principal
        menu.setVisible(true);
        System.gc();
        dispose();
    }
}
Reloj reloj=new Reloj();
```

Subclase DisparoN

Esta subclase sirve como hilo que se encarga del movimiento del disparo de la nave y la colision de este con algún enemigo, solo en caso que el hilo del mismo este corriendo.

```
//Disparo de la nave
//Posiciones que saldra el disparo
int posicionX,posicionY;
class DisparoN implements Runnable{
    @Override
    public void run() {
        posicionX=55;posicionY=mn.y+18;
        boolean colision=false;
        //Características del label
        disparo.setText("");
        disparo.setBackground(Color.white);
        disparo.setOpaque(true);
```

```
//Animación del disparo
while(posicionX<=2000){
    //Comprobar si el disparo coincide con alguno de los aliens tipo 1
    for (int i = 0; i < 8; i++) {
        //Se comprueba si el hilo del alien aun esta activo
        if (htipo1[i].isAlive()==true) {
            int a=Math.abs((atipo1[i].x+45)-(posicionX+15)),
                b=Math.abs((atipo1[i].posiciony+45)-(posicionY+15));
            if ((0<a && a<=44) && (0<b && b<44)) {
                atipo1[i].vida--;
                colision=true;
                disparo.setOpaque(false);
                break;
            }
        }

    }
//Comprobar si el disparo coincide con alguno de los aliens tipo 2
for (int i = 0; i < 16; i++) {
    //Se comprueba si el hilo del alien aun esta activo
    if (htipo2[i].isAlive()==true) {
        int a=Math.abs((atipo2[i].x+45)-(posicionX+15)),
        b=Math.abs((atipo2[i].posiciony+45)-(posicionY+15));
        if ((0<a && a<44) && (0<b && b<44)) {
            atipo2[i].vida--;
            colision=true;
            disparo.setOpaque(false);
            break;
        }
    }

}
//Comprobar si el disparo coincide con alguno de los aliens tipo 3
for (int i = 0; i < 16; i++) {
    //Se comprueba si el hilo del alien aun esta activo
    if (htipo3[i].isAlive()==true) {
        int a=Math.abs((atipo3[i].x+45)-(posicionX+15)),
        b=Math.abs((atipo3[i].posiciony+45)-(posicionY+15));
        if ((0<a && a<44) && (0<b && b<44)) {
            atipo3[i].vida--;
            colision=true;
            disparo.setOpaque(false);
            break;
        }
    }

}
            //Desplazamiento del disparo
            disparo.setBounds(posicionX, posicionY, 15, 15);
            try {
                Thread.sleep(50);
            } catch (InterruptedException ex) {
                Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
            }
            posicionX+=40;

            //En caso de que haya colisionado
            if (colision) {
                break;
            }
        }
    }

}
//Se inicializa la clase DisparoN
DisparoN disparoN=new DisparoN();
```

Subclase Actu

Esta subclase es un hilo que constantemente actualiza los datos que muestra.

```
//Metodo para actualizar los marcadores
class Actu implements Runnable{

    @Override
    public void run() {
        boolean a=true;
        while(a){
            labelVelocidad.setText(movimiento+" Km/s");
            puntos=config.puntos;
            labelpuntos.setText(puntos+"");
            //Comprobar si los aliens ya llegaron a la tierra
            if (config.derota==true) {
                //Se cierran los hilo
                for (int i = 0; i < 8; i++) {
                   htipo1[i].stop();
                }
                for (int i = 0; i < 16; i++) {
                   htipo2[i].stop();
                }
                for (int i = 0; i < 16; i++) {
                   htipo3[i].stop();
                }
                hilo1.stop();
                hilo2.stop();
                hilo3.stop();
                hilo4.stop();
   ImageIcon img=new ImageIcon("explosion.png");
   Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(labelNave.getWidth(), labelNave.getHeight(), Image.SCALE_DEFAULT));
   labelNave.setIcon(iconol);
   JOptionPane.showMessageDialog(null,"Los aliens llegaron a la tierra\n"
                       + "FIN de la partida");

   JOptionPane.showMessageDialog(null, "Su punttuacion es de "+puntos);
   String acronimo=JOptionPane.showInputDialog(null,"Escriba su nombre");
   puntuaciones.nuevopuntaje(acronimo, puntos);
   //Guardar el puntaje
   respaldo=new Respaldo();
   try {
       //Crear documento tipo binario
       ObjectOutputStream oos;
       oos=new ObjectOutputStream(new FileOutputStream("datos.bin"));
       //Se elige la clase que se transformara en el documento binario
       oos.writeObject(respaldo);
       oos.close();
   } catch (Exception e) {
       System.out.println(e);
   }
   //Regresar al menu principal
   a=false;
   menu.setVisible(true);
   dispose();
}
```

```
//Comprobar si ya se derrotaron a todos los enemigos tipo 1
if (config.derrotatipo1==8) {
    movimiento+=10;
    //Dar mayor movimiento a todos los aliens tipo 2 y 3
    for (int i = 0; i < 16; i++) {
        atipo2[i].movimiento+=10;
    }
    for (int i = 0; i < 16; i++) {
        atipo3[i].movimiento+=10;
    }
    config.derrotatipo1++;
}
//Comprobar si ya se derrotaron a todos los enemigos tipo 2
if (config.derrotatipo2==16) {
    movimiento+=15;
    //Dar mayor movimiento a todos los aliens tipo 3
    for (int i = 0; i < 16; i++) {
        atipo3[i].movimiento+=15;
    }
    config.derrotatipo2++;
}
if (config.derrotatipo3==16) {
    hilo1.stop();
    hilo2.stop();
    hilo3.stop();
    hilo4.stop();
    try {
        Thread.sleep(1500);
    } catch (InterruptedException ex) {
        Logger.getLogger(Juego.class.getName()).log(Level.SEVERE, null, ex);
    }
    Reloj reloj=new Reloj();
    puntos+=tiempo_ex;
    JOptionPane.showMessageDialog(null, "Gano la partida");
    JOptionPane.showMessageDialog(null, "Su punttuacion es de "+puntos);
    String acronimo=JOptionPane.showInputDialog(null,"Escriba su nombre");
    puntuaciones.nuevopuntaje(acronimo, puntos);
    //Guardar puntuacion
    respaldo=new Respaldo();
    try {
        //Crear documento tipo binario
        ObjectOutputStream oos;
        oos=new ObjectOutputStream(new FileOutputStream("datos.bin"));
        //Se elige la clase que se transformara en el documento binario
        oos.writeObject(respaldo);
        oos.close();
    } catch (Exception e) {
        System.out.println(e);
    }
    //Regresar al menu principal
    menu.setVisible(true);
    dispose();
    config.derrotatipo3++;
    a=false;
}

Actu actu=new Actu();

//Hilos que se manejan
Thread hilo1=new Thread(mn);
Thread hilo2=new Thread(reloj);
Thread hilo3=new Thread(disparoN);
Thread hilo4=new Thread(items);
Thread hilo5=new Thread(actu);
```

```
public Juego() {
    initComponents();
    panelJuego.setLayout(null);
    //Se reinician los contadores necesarios
    config.derota=false;
    config.puntos=0;
    config.derrotatipo1=0;
    config.derrotatipo2=0;
    config.derrotatipo3=0;
    //Arreglos que almacenen los labels que corresponden a los aliens, dependiendo al tipo que petenezca
    JLabel tipos1[]={tipo1_1,tipo1_2,tipo1_3,tipo1_4,tipo1_5,tipo1_6,tipo1_7,tipo1_8},
           tipos2[]={tipo2_1,tipo2_2,tipo2_3,tipo2_4,tipo2_5,tipo2_6,tipo2_7,tipo2_8,
                  tipo2_9,tipo2_10,tipo2_11,tipo2_12,tipo2_13,tipo2_14,tipo2_15,tipo2_16},
           tipos3[]={tipo3_1,tipo3_2,tipo3_3,tipo3_4,tipo3_5,tipo3_6,tipo3_7,tipo3_8,
                  tipo3_9,tipo3_10,tipo3_11,tipo3_12,tipo3_13,tipo3_14,tipo3_15,tipo3_16};

    //Se llenan los labels para los aliens
    //Tipo 1
       //Posicion y de los aliens tipo 1
       int y=21;
    for (int i = 0; i < 8; i++) {
       //Se instancia la clase
       atipo1[i]=new AlienTipo1();
       atipo1[i].setLabel(
              tipos1[i]);
       //Posicion y de inicio
       atipo1[i].setPosiciony(y);
       //Posicion que debe tener el siguiente alien
       y+=66;

    }
//Tipo 2
    //Posicion y de los aliens tipo 2
    y=21;
for (int i = 0; i < 8; i++) {
    //Se instancia la clase
    atipo2[i]=new AlienTipo2();
    atipo2[i].setLabel(
           tipos2[i]);
    //Posicion X y Y de inicio
    atipo2[i].setPosicionx(1071);
    atipo2[i].setPosiciony(y);
    //Posicion que debe tener el siguiente alien
    y+=66;
}
    y=21;
for (int i = 8; i < 16; i++) {
    //Se instancia la clase
    atipo2[i]=new AlienTipo2();
    atipo2[i].setLabel(
           tipos2[i]);
    //Posicion X y Y de inicio
    atipo2[i].setPosicionx(1128);
    atipo2[i].setPosiciony(y);
    //Posicion que debe tener el siguiente alien
    y+=66;
}
```

```
//Tipo 3
    //Posicion y de los aliens tipo 3
    y=21;
for (int i = 0; i < 8; i++) {
    //Se instancia la clase
    atipo3[i]=new AlienTipo3();
    atipo3[i].setLabel(
            tipos3[i]);
    //Posicion X y Y de inicio
    atipo3[i].setPosicionx(1185);
    atipo3[i].setPosiciony(y);
    //Posicion que debe tener el siguiente alien
    y+=66;
}
    y=21;
for (int i = 8; i < 16; i++) {
    //Se instancia la clase
    atipo3[i]=new AlienTipo3();
    atipo3[i].setLabel(
            tipos3[i]);
    //Posicion X y Y de inicio
    atipo3[i].setPosicionx(1242);
    atipo3[i].setPosiciony(y);
    //Posicion que debe tener el siguiente alien
    y+=66;
}
```

```
//Se inician los hilos de los aliens
//Tipo 1
for (int i = 0; i < 8; i++) {
    htipo1[i]=new Thread(atipo1[i]);
    //Se inicia el hilo
    htipo1[i].start();
}
//Tipo 2
for (int i = 0; i < 16; i++) {
    htipo2[i]=new Thread(atipo2[i]);
    //Se inicia el hilo
    htipo2[i].start();
}
//Tipo 3
for (int i = 0; i < 16; i++) {
    htipo3[i]=new Thread(atipo3[i]);
    //Se inicia el hilo
    htipo3[i].start();
}
```

```
   //Se crea el icono de los puntos
   ImageIcon img=new ImageIcon("coin.png");
   Icon iconol=new  ImageIcon(img.getImage().getScaledInstance(labelMoneda.getWidth(), labelMoneda.getHeight(), Image.SCALE_DEFAULT));
   labelMoneda.setIcon(iconol);
   //Se crea el icono del reloj
   img=new ImageIcon("reloj.png");
   iconol=new  ImageIcon(img.getImage().getScaledInstance(labelMoneda.getWidth(), labelMoneda.getHeight(), Image.SCALE_DEFAULT));
   labelReloj.setIcon(iconol);


  //Se dan las caracterizticas a los labels de los hilos externos
  mn.setLnave(labelNave);
  //Se inicia los hilos necesarios
  hilo1.start();
  hilo2.start();
  hilo4.start();
  hilo5.start();
}
```

Acciones de Teclado

Al frame se le agrego un evento que se activa siempre que el usuario toca una tecla en su teclado.

```
private void formKeyPressed(java.awt.event.KeyEvent evt) {
    //En caso de que toque la tecla de arriba
    if (KeyEvent.VK_UP==evt.getKeyCode()) {
        //Primero se comprueba si ya se alcanzo el limite
        if (mn.y>25) {
            //Comprobando si la nave NO se encuentra congelada
            if (hilo1.isAlive()==true) {
              mn.sumaY(-movimientonave);
            }
        }
    }
    //En caso de que toque la tecla de abajo
    if (KeyEvent.VK_DOWN==evt.getKeyCode()) {
        //Primero se comprueba si ya se alcanzo el limite
        if (mn.y<515) {
            //Comprobando si la nave NO se encuentra congelada
            if (hilo1.isAlive()==true) {
               mn.sumaY(movimientonave);
            }
        }
    }
    //Disparo
    if (KeyEvent.VK_SPACE==evt.getKeyCode()) {
        //Comprobando si la nave NO se encuentra congelada
            if (hilo1.isAlive()==true) {
                hilo3=new Thread(disparoN);
                hilo3.start();
            }
    }
```

```
//Menú de pausa
if (KeyEvent.VK_ESCAPE==evt.getKeyCode()) {

    //Pausar arreglos
       hilo1.suspend();
       hilo2.suspend();
       hilo3.suspend();
       hilo4.suspend();
       hilo5.suspend();
       for (int i = 0; i < 8; i++) {
           htipo1[i].suspend();
       }
       for (int i = 0; i < 16; i++) {
           htipo2[i].suspend();
       }
       for (int i = 0; i < 16; i++) {
           htipo3[i].suspend();
       }
    //Arreglo de opciones
    Object [] opciones={"Reanudar","Salir"};
    //Cuadro de dialogo
    int i=JOptionPane.showOptionDialog(this, "Pausa", "", JOptionPane.YES_NO_OPTION,
           JOptionPane.QUESTION_MESSAGE, null, opciones, opciones[0]);
```

```
//Opción de reanudar
if (i==0) {
    hilo1.resume();
    hilo2.resume();
    hilo3.resume();
    hilo4.resume();
    hilo5.resume();
    for (int j = 0; j < 8; j++) {
        htipo1[j].resume();
    }
    for (int j = 0; j < 16; j++) {
        htipo2[i].resume();
    }
    for (int j = 0; j < 16; j++) {
        htipo3[i].resume();
    }
}

//opcion de salir
if (i==1) {
    menu.setVisible(true);
    //Se paran los hilos
    hilo1.stop();
    hilo2.stop();
    hilo3.stop();
    hilo4.stop();
    hilo5.stop();
    for (int j = 0; j < 8; j++) {
        htipo1[j].stop();
    }
    for (int j = 0; j < 16; j++) {
        htipo2[j].stop();
    }
    for (int j = 0; j < 16; j++) {
        htipo3[j].stop();
    }
    System.gc();
    dispose();
}

        /*En caso que se salga del menu de pausa de cualquier otra manera que
            no sea tocando los botenes del mismo**/
        hilo1.resume();
        hilo2.resume();
        hilo3.resume();
        hilo4.resume();
        hilo5.resume();
        for (int j = 0; j < 8; j++) {
                htipo1[j].resume();
        }
        for (int j = 0; j < 16; j++) {
                htipo2[j].resume();
        }
        for (int j = 0; j < 16; j++) {
                htipo3[j].resume();
        }
    }
}
```

Diagrama de Flujo