Práctica de Programación Orientada a Objetos
Junio 2013

Jesús López García

jlopez2856@alumno.uned.es
jesuslopgar@gmail.com
Tel: 647994781

Centro asociado: Albacete

Índice.

Contenido

1. Introducción

La práctica del presente curso va a estar basada en el legendario arcade “R-Type”. Esto nos servirá para estudiar y practicar los mecanismos de la Programación Orientada a Objetos y hacer uso de sus características gráficas.

Historia del juego

R-Type es un videojuego de género matamarcianos creado por Irem en 1987 para las máquinas recreativas o Arcade.

En este juego el jugador controla a un caza espacial llamado R-9 que se caracteriza por un láser que se puede cargar para aumentar la fuerza de impacto contra los enemigos y a la vez, con una cápsula, llamada Force (fuerza) que se deja anclar a la nave y otorga un arma definida de acuerdo con el color o la letra que tiene el Power-Up que se recoge a los siguientes. El poder de "Fuerza" aumenta conforme se recogen más armas. También existen extras para aumentar la velocidad de despliegue de la nave, cápsulas fijas sobre el techo y debajo de la base de R-9 y misiles detectores (homing) como añadido eficiente al disparo que ya poseemos. Lo que más distinguió a R-Type de los juegos de la competencia era la novedad de poder utilizar la cápsula "Force" tanto para atacar como para defenderse de los disparos enemigos y que esta se podía lanzar y separar de la nave para utilizarla como un satélite, además de forma libre, lo que habilitaba al jugador de anclar "Fuerza" en la cola de la nave y así dispara hacia atrás, es decir hacia la izquierda de la pantalla. R-Type fue el único juego que disponía de estas características y de un apartado técnico impecable. Daba igual la cantidad de enemigos o su tamaño en pantalla, R-Type no se ralentizaba ni se dañaba la apariencia en otros objetos que simultáneamente se movían sobre la pantalla. Este era un error muy común en aquellos tiempos pero Irem demostró su maestría en crear un shooter horizontal sin estos errores comunes. Este último hecho solo se entiende para la versión original de R-Type en máquina recreativa. Las conversiones para consolas y ordenadores solían tener algunos fallos pero era debido a que el hardware no cumplía con los requisitos para poder programar a un R-Type vistoso y sin ralentizaciones.

2. Enunciado de la práctica

Implementación Obligatoria

El alumno deberá implementar un juego del estilo R-Type satisfaciendo los siguientes requisitos:

1- El juego comenzará con una pantalla de bienvenida a partir de la cual se podrá seleccionar el modo de juego (FACIL, NORMAL, COMPLICADO, IMPOSIBLE) y comenzar a jugar.

2- El juego constará de un único nivel donde el jugador deberá acabar con una horda de naves alienígenas. El número de alienígenas con los que acabar dependerá del modo de juego seleccionado. Fácil=10, Normal=15, Complicado=20, Imposible=30.

3- El jugador controlará la nave aliada y dispondrá de 1 sola vida.

4- Las naves alienígenas serán controladas por el ordenador.

5- Las naves alienígenas no disparan.

6- No hay que implementar relieve. Es decir, no hay que mostrar ningún tipo de suelo o techo como en el juego original.

7- La nave aliada podrá moverse arriba (Tecla Q), abajo (Tecla A), izquierda (Tecla O) y derecha (Tecla P). Así mismo podrá disparar su laser utilizando la tecla ESPACIO.

8- El área de movimiento permitido para la nave será toda la pantalla, aunque habrá que comprobar que la nave no salga de estos límites.

9- El disparo que realiza la nave aliada es continuo, es decir, no es necesario esperar a que el misil disparado abandone la pantalla para que la nave aliada pueda volver a disparar.

10- La nave aliada sólo puede realizar un tipo de disparo que se desplazará horizontalmente hacia la derecha de la pantalla, sin variar su trayectoria y a velocidad constante.

11- Las naves alienígenas se mueven a velocidad constante y podrán ser de dos tipos:

a. Nave Alienígena Tipo A. Aparecen por la parte derecha de la pantalla y se mueven horizontalmente hacia la izquierda a velocidad constante sin variar su trayectoria, es decir, su coordenada "y" no varía en todo el desplazamiento.

b. Nave alienígena Tipo B. Aparecen por la parte derecha de la pantalla y se mueven horizontalmente hacia la izquierda a velocidad constante. La principal diferencia con las Naves de Tipo A es que éstas pueden variar su trayectoria, es decir, en su desplazamiento horizontal pueden variar su coordenada "y" de manera aleatoria.

12- La velocidad a la que se mueven las naves alienígenas dependerá del modo de juego seleccionado. Todas las naves se mueven a la misma velocidad.

13- Cuando las naves alienígenas alcancen la parte izquierda de la pantalla volverán a aparecer por la parte derecha de ésta.

14- Se deberán de detectar dos tipos de colisiones.

a. Las colisiones entre la nave aliada y las naves alienígenas, lo que supondrá el final del juego.

b. Las colisiones entre los misiles disparados por la nave aliada y las naves alienígenas, lo que supondrá la destrucción de la nave alienígena contra la que ha chocado el misil.

15- Si el jugador finaliza el nivel del juego deberá aparecer un mensaje de felicitación y se volvería a mostrar el menú inicial.

3. Análisis de la aplicación realizada

La aplicación R-Type está formada por distintos tipos de objetos que interactúan entre ellos, Una nave aliada que será manejada por el usuario, naves alienígenas que aparecen con el objetivo de colisionar y destruir la nave aliada, la cual podrá realizar disparos para destruirlas, todo ello dentro de una pantalla donde deberán moverse todos estos objetos. Requiere de la habilidad del jugador para conseguir la victoria sorteando los obstáculos con que se enfrenta la nave aliada.

Entendiendo la aplicación desde un punto de vista geométrico, todos estos objetos tienen varias cualidades comunes, independientemente de su forma y objetivo en el juego. Todos tienen una imagen y una posición en todo momento, al igual que un comportamiento cuando colisionen entre ellos.

La labor del programador es, siguiendo el patrón de normas ofrecido en el enunciado de la practica y conociendo la historia del juego, hacer que estos objetos interactúen dentro de esta pantalla, en la cual algunos como las naves alienígenas serán manejados de forma automática por la máquina y otros como la nave aliada y los disparos requieren de la intervención del usuario, que utilizando el teclado, permitirá que la nave aliada se mueva y dispare dentro de nuestra pantalla, destruyendo o no las naves alienígenas.

Desde ese punto de vista geométrico se diseñarán e implementarán las clases necesarias para dar a los objetos funcionalidad de movimiento, atendiendo al teclado el objeto que corresponda, cuando el usuario pulse las teclas indicadas y la actuación de estos objetos se base en las normas proporcionadas en el enunciado.

Se le ha dado el titulo de R-Type StarWars Edition para darle un toque de originalidad al juego, utilizando las imágenes de los vehículos de la saga StarWars en nuestras naves. Para la nave Aliada se utiliza la imagen de la nave Jedi de la Alianza Rebelde que pilotaba Obi Wan, mientras que las naves alienígenas se representan con vehículos del Imperio.

Capturas de pantalla de la aplicación

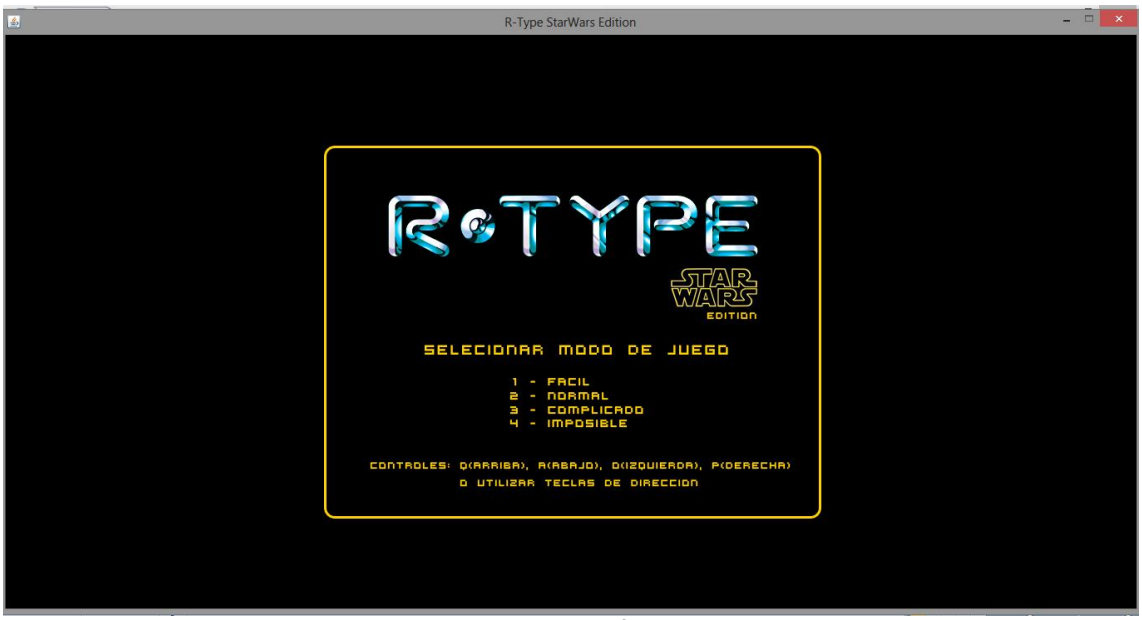


Imagen del menú inicial

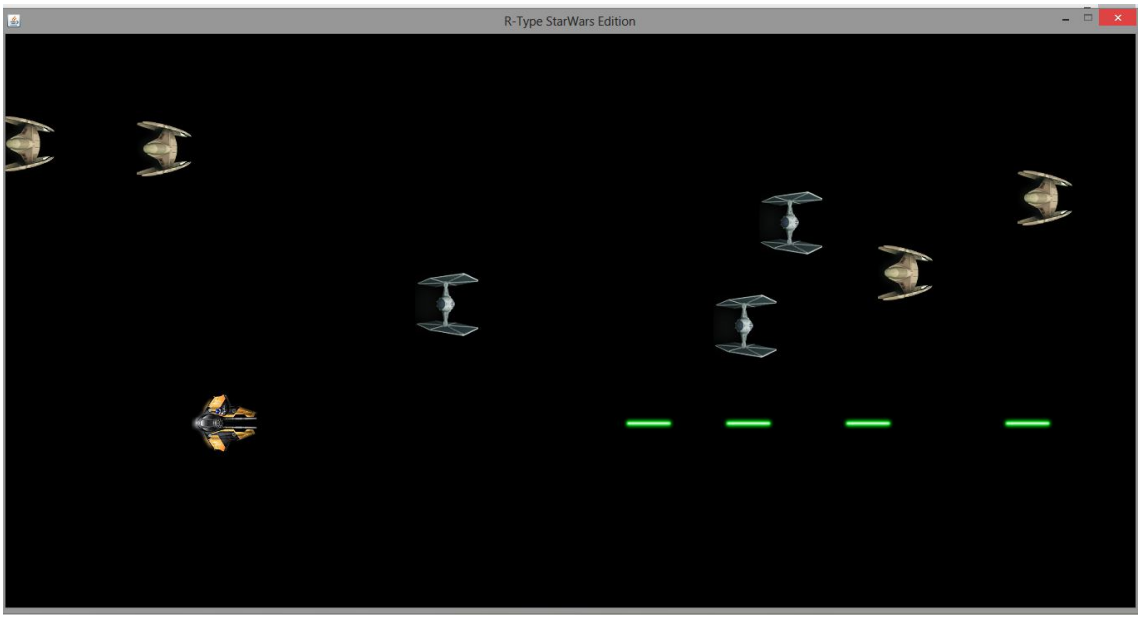
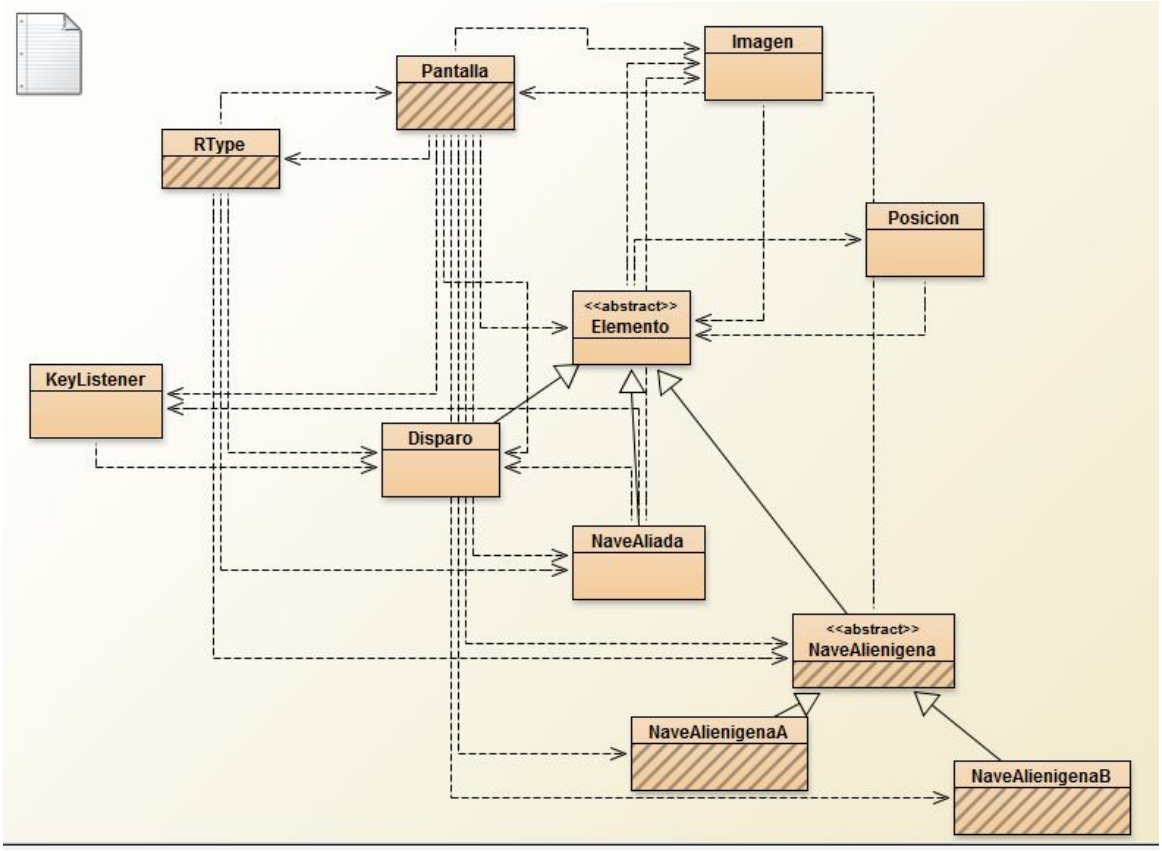


Imagen del juego en funcionamiento

Imagen del mensaje de victoria

Imagen del mensaje de derrota

4. Diagrama de clases



En el diagrama de clases se puede observar las relaciones entre las clases.
Las relaciones de herencia se muestran como flechas con punta hueca; las relaciones de tipo "usa" se muestran como flechas discontinuas con punta abierta.

5. Descripción de las clases

Clase RType

La clase **Rtype** es la clase que contiene el método public static void main(String[] args). Será la clase encargada de lanzar el interface de usuario *IGU*. Para esta interface se han utilizado componentes de la librería *SWING* de Java, como son *JFrame* y *JPanel*, en los cuales se mostrarán los objetos gráficos que tengamos en la aplicación.

RType extiende *JFrame*, en ella se especifican las características que ha de tener el *JFrame* dentro del cual añadiremos un *JPanel* , el *JPanel* que será implementado en la clase **Pantalla** que se explicará más adelante.

En el constructor de **RType** lo que hacemos es básicamente inicializar algunas de las caracteristicas más importantes del *JFrame*, ponerle un titulo "R-Type StarWars Edition", unas dimensiones de ventana para las cuales se obtenemos las dimensiones de la pantalla del equipo en el que ejecutemos la aplicación por medio de los métodos:

```
private static final int
ANCHO_PANTALLA=java.awt.Toolkit.getDefaultToolkit().getScreenSize().width;
```

```
private static final int
ALTO_PANTALLA=java.awt.Toolkit.getDefaultToolkit().getScreenSize().height-80;
```

Le añadimos al *JFrame* un *JPanel* que crearemos pasándole las dimensiones de ventama que hemos obtenido add(new Pantalla(ANCHO_PANTALLA,ALTO_PANTALLA));

Y hacemos que se muestre la ventana y no pueda redimensionarse con setVisible(true); setResizable(false);

Finalmente en el método main tan solo creamos el *JFrame* con new RType();

Clase Pantalla

La clase **Pantalla** extiende *JPanel* e implementa el interface *Actionlistener* ya que en esta clase se escucharán los eventos del teclado para hacer que la Nave Aliada se mueva y dispare.

Pantalla es una de las clases más importantes de la aplicación ya que en ella se incluyen las siguientes funcionalidades:

-En su constructor se crea la Nave Aliada, el *ArrayList* que contendrá las *navesAlienigenas*, se añade el controlador de teclado de la *NaveAliada* para que el usuario pueda moverla y se establece el *timer* cada 10 ms. También se crean los mensajes de victoria, derrota y el menú inicial.

- Se sobrescribe el método paint de *JPanel* public void paint (Graphics grafico) agregando las sentencias que nos permitirán pintar los objetos del juego que existan en cada momento en sus posiciones y con sus imágenes

- *Logica del juego.* En esta clase se incorpora el método public void actionPerformed(ActionEvent e) que atendiendo a los tiempos que marque el *timer* declarado en el constructor, pintará los objetos en el *JPanel*, ya sean naves y disparos o mensajes de menú, victoria o derrota, dependiendo del momento en el que nos encontremos.

- Control de colisiones de objetos mediante el método private void checkCollisions(). En este método utilizamos el componente *Rectangle* para crear un rectángulo por cada objeto que hayamos creado de tipo NaveAliada, NaveAlienigena o Disparo y a estos rectángulos le damos las dimensiones de las imágenes que tengan los objetos. Una vez creados los rectángulos por cada objeto, utilizamos el método intersect() de *Rectangle* de la siguiente manera:

1. Si intersecta el rectángulo de la NaveAliada con el de alguna NaveAlienigena, significa que nos han derrotado y lanzamos el método derrota() .

2. Si intersecta una NaveAlienigena con un Disparo, ponemos invisible esa nave y ese disparo, para que puedan ser borrados de su colección en el siguiente ciclo de reloj.

- En el método private void jugar() que se llamará en cada tic de reloj, comprobaremos los elementos que estén visibles. Si están visibles los moveremos llamando a su método mover() y si no lo están los eliminaremos de sus respectivas colecciones. Cuando no quede ninguna NaveAlienigena en el *ArrayList* significara que todas esas naves han sido borradas por haber colisionado con un disparo y lanzaremos el método victoria().

Métodos de la clase:

public void paint (Graphics grafico)
Sobrescribe el método paint de *Jpanel*

public void actionPerformed(ActionEvent e)

private void crearMensajes()
Crea los objetos Imagen de los mensajes de victoria, derrota, menú inicial y sus posiciones.

private void victoria()
Lanza el mensaje de victoria y espera a que pulsemos la tecla C para reiniciar el juego

private void derrota()
Lanza el mensaje de derrota y espera a que pulsemos la tecla C para reiniciar el juego

private void reiniciar_juego()
Reinicia el juego, borrando las naves alienígenas y disparos que quedasen, reubica la nave aliada en su posición original y muestra de nuevo el menú

private void jugar()
Tras comprobar las colisiones, recorre las colecciones de disparos y naves alienígenas, moviéndolos si están visibles o borrándolos si no lo están.
Atendiendo al teclado, mueve la nave aliada o crea disparos y repinta el *JPanel* .

public ArrayList<NaveAlienigena> getNavesAlienigenas()

private void crearNavesAlienigenas()
Atendiendo a la dificultad que indique el usuario en el menú inicial añade las NavesAlienigenas correspondientes al *ArrayList navesAlienigenas*. La mitad de ellas serán de tipo A y la otra mitad de tipo B.

private void borrarNavesAlienigenas()
Recorre el *ArrayList navesAlienigenas* y borra todos sus elementos.

private void checkCollisions()
Comprueba las colisiones entre NaveAliada, NavesAlienigenas y Disparos poniendo visibles o invisibles los que hayan colisionado

Clase Imagen

La clase **Imagen** ha sido creada ya que todos los elementos del juego ya sean naves, disparos o mensajes, tienen una imagen asignada. Esta clase nos permite crear objetos de tipo imagen desde la ruta que le indiquemos. Esta clase podría haberse integrado dentro de la clase Elemento ya que no tiene demasiados atributos, pero se decidió separarla por ser mas versátil y permitir así el uso de elementos Imagen en otras clases que no sean Elementos, como en la clase Pantalla para mostrar el menú o los mensajes de victoria y derrota.

Métodos de la clase:

public int getAncho()
Devuelve el ancho de una imagen

public int getAlto()
Devuelve el alto de una imagen

public boolean getVisible()
Devuelve si una imagen es visible

public void setVisible(boolean visible)
Establece si una imagen es visible

public Image getImage()
Devuelve la imagen de un objeto Imagen

Clase Posicion

La clase **Posicion** ha sido creada debido a que todos los elementos del juego ya sean naves o disparos o mensajes tienen una posición asignada. Esta clase podría haberse integrado dentro de la clase Elemento ya que no tiene demasiados atributos, pero se decidió separarla por ser más versátil y permitir así crear posiciones para, por ejemplo, los mensajes de victoria o derrota y el menú inicial.

Métodos de la clase:

public void setX(int x)
Establece la coordenada x de una posición

public void setY(int y)
Establece la coordenada y de una posición

int getX()
Devuelve la coordenada x de una posición

int getY()
Devuelve la coordenada y de una posición

Clase Elemento

La clase **Elemento** es la superclase abstracta de la que heredan todos los objetos que puedan dibujarse en nuestro *JPanel* a excepción del menú y los mensajes de victoria y derrota. Estos objetos son la NaveAliada, Disparo, NaveAlienigenaA y NaveAlienigenaB.

Ha sido diseñada así ya que todos esos objetos, aunque tienen particularidades también comparten importantes características como tener una Imagen, una Posicion en nuestro *JPanel* , unos límites de pantalla que hemos de tener en cuenta de forma distinta para cada elemento y un rectángulo asociado a cada imagen que nos ayudará a gestionar las colisiones.

No podremos instanciar objetos de esta clase ya que al ser muy genérica se ha declarado abstracta, los objetos que podrán instanciarse son los de sus subclases, donde se definirán sus particularidades.

Métodos de la clase:

Imagen getImagen()
Devuelve la Imagen asignada al Elemento.

Posicion getPosicion()
Devuelve la posición asignada al Elemento.

public void setImagen(String ruta)
Establece la imagen del Elemento

public void setVisible(boolean estado)
Establece la visibilidad del Elemento

public boolean getVisible()
Devuelve si el Elemento es visible

public int getLimiteDerecho()
Devuelve el limite derecho del *JPanel*, será tenido en cuenta por la nave aliada para no rebasarlo.

public int getLimiteInferior()
Devuelve el límite inferior del *JPanel*, será tenido en cuenta por la nave aliada y naves alienígenas para no rebasarlo.

public Rectangle getBounds()
Devuelve el rectángulo asociado al Elemento para gestionar las colisiones.

Clase NaveAliada

La clase **NaveAliada** extiende a Elemento. En ella se definirán las peculiaridades de esta nave como su imagen, la velocidad a la que ha de moverse, la posición en la que ha de aparecer, los limites que ha de respetar en su movimiento así como un controlador de teclado para que el usuario pueda moverla. Cuando se crea el KeyListener en el constructor se le pasa como parámetro la constante *VELOCIDAD_NAVEALIADA* que servirá para que este conozca cuantos pixeles ha de mover la nave en cada lectura de tecla de dirección.

Métodos de la clase:

public KeyListener getTeclado()
Devuelve el objeto controlador de teclado que se le ha asignado a NaveAliada

public ArrayList<Disparo> getDisparos()
Devuelve el *ArrayList disparosActivos*, con los disparos que se han creado hasta el momento al pulsar la tecla espacio.

public void disparar()
Añade un nuevo Disparo al *ArrayList disparosActivos* cada vez que se pulsa la tecla espacio, haciendo que la posición inicial de estos disparos parta del centro de la nave aliada.

public void borrarDisparosActivos()
Elimina los Disparos que haya en el *ArrayList disparosActivos*

public void mover(int dx, int dy)
Mueve la NaveAliada a la las coordenadas x e y que se le pasen por parámetro, siempre que estas se encuentren dentro de los limites de pantalla adecuados.

Clase KeyListener

La clase KeyListener extiende a KeyAdapter, crea un escuchador de eventos de teclado que controlará las teclas que han sido pulsadas y en base a ello modificara sus campos dx y dx que indican las coordenadas a las que ha de moverse la NaveAliada cada vez que se detecte una pulsación de teclas de dirección.

También guarda en sus variables espacio y tecla si se ha pulsado un espacio, que lo utilizaremos para crear disparos o si se ha pulsado alguna tecla necesaria como la 'c' cuando tras una victoria o derrota queremos continuar el juego.

Métodos de la clase:

public void keyPressed(KeyEvent e)
Detecta la pulsación de teclas de dirección 'O', 'P', 'Q', 'A' o las teclas que indican el nivel '1','2', '3', '4' o la tecla 'C' para continuar una partida.

public void keyReleased(KeyEvent e)
Detecta cuando dejamos de pulsar una tecla, haciendo que la nave deje de moverse en la dirección de la tecla que haya dejado de pulsarse. En este método detectamos si la tecla que se ha dejado de pulsar es el espacio, para generar un Disparo, ya que si esta funcionalidad la ejecutamos cuando se pulsa la tecla espacio en keyPressed en lugar de cuando se suelta, nos crearía Disparos demasiado rápido.

public int getDx()
Devuelve la coordenada x a la que debe moverse la NaveAliada.

public int getDy()
Devuelve la coordenada y a la que debe moverse la NaveAliada.

public int getEspacio()
Devuelve si hemos pulsado la tecla espacio

public int getTecla()
Devuelve que tecla se ha pulsado si ha sido '1','2','3','4' para seleccionar nivel o 'c' para continuar.

public void setEspacio(int espacio)
Establece a mano el valor booleano de la tecla espacio, útil para limpiar a mano el valor de esta variable.

Clase Disparo

La clase **Disparo** extiende a Elemento. En ella se definirán las peculiaridades del elemento Disparo como su Imagen o la velocidad a la que debe desplazarse por la pantalla por medio de la constante *VEL_DISPARO=8* que serán los pixeles que se moverá el disparo hacia la derecha en cada ciclo de reloj.

Métodos de la clase:

public void mover()
Sobrescribimos el método mover() de Elemento para hacer que se mueva de izquierda a derecha a la velocidad constante que se le establezca en *VEL_DISPARO*. Al salir por el límite derecho de la pantalla lo pondremos invisible para ser borrado en el siguiente ciclo de reloj.

Clase NaveAlienigena

La clase abstracta **NaveAlienigena** extiende a Elemento.

Tenemos naves alienígenas de dos tipos A y B, ambas tienen características similares aunque su forma de moverse varíe.

Esta clase será superclase de NaveAlienigenaA y NaveAlienigenaB, no podremos instanciar objetos de esta pero en ella inicializaremos algunos campos de la naves como su posición utilizando dos variables aleatorias para posicionar inicialmente las naves en distintas posiciones intentando que se ubiquen a lo alto de la pantalla y a lo ancho de 1500 pixeles a la derecha de la pantalla, para que luego vayan apareciendo por la derecha repartidas por la pantalla.

Se decidió este tipo de herencia para las naves alienígenas ya que nos permite crear una sola colección que contenga las naves alienígenas del tipo de la superclase NaveAlienigena aunque los objetos que contenga sean del tipo de las subclases NaveAlienigenaA y NaveAlienigenaB. Así cuando que haya que mover las naves recorremos una sola colección de objetos llamando a su método mover, que será sobrescrito en las subclases ya que cada tipo tiene distinta forma de moverse.

Clase NaveAlienigenaA

La clase **NaveAlienigenaA** extiende a la superclase NaveAlienigena, agregando las características que no estaban definidas en su superclase, como una imagen específica para el tipo A, o la velocidad a la que debe moverse, que se le pasará por parámetro en su constructor public NaveAlienigenaA(int dificultad) y dependerá de la dificultad que hayamos seleccionado en el menú inicial.

Métodos de la clase:

public void mover()
Sobrescribimos este método para hacer que el movimiento de la nave de tipo A sea de derecha a izquierda, a velocidad constante y controle que al salir por la parte izquierda de la pantalla vuelva a aparecer por la parte derecha.

Clase NaveAlienigenaB

La clase **NaveAlienigenaB** al igual que NaveAlienigenaA extiende a la superclase NaveAlienigena, agregando las características que no estaban definidas en su superclase, como una imagen específica para el tipo B, o la velocidad a la que debe moverse, que se le pasará por parámetro en su constructor public NaveAlienigenaB(int dificultad) y dependerá de la dificultad que hayamos seleccionado en el menú inicial.

Métodos de la clase:

public void mover()

Sobrescribimos este método para hacer que el movimiento de la nave de tipo B sea de derecha a izquierda, a velocidad constante y controle que al salir por la parte izquierda de la pantalla vuelva a aparecer por la parte derecha.

Hasta aquí igual que el tipo A pero el tipo B puede además cambiar de dirección aleatoriamente, por lo que para este tipo se añade la funcionalidad de que cambie de dirección. Para que la nave no tiemble y el cambio de dirección sea suave, tenemos la constante *FREC_CAMBIO=50* donde indicaremos cada cuantos pixeles que avance la nave hacia la izquierda debemos hacer un cambio de dirección aleatorio. Esto quiere decir que la nave podría cada 50 pixeles y sin parar de avanzar, cambiar su dirección hacia arriba, hacia abajo o continuar de frente.

Nota sobre las velocidades en NaveAlienigenaA y NaveAlienigenaB :

Aunque el enunciado de la practica indica que la velocidad de las naves es la misma, se han declarado variables de velocidad distintas para cada tipo con el pensamiento de que se puedan poner distintas velocidades a cada tipo de nave en base a la dificultad seleccionada y así hacer más atractivo el juego.

6. Código fuente

Clase RType

```
import javax.swing.JFrame;

/**
 * La clase  Rtype es la clase que contiene el método public static void
main(String[] args).
 * Será la clase encargada de lanzar el interface de usuario IGU.
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */

public class RType extends JFrame
{

    private static final int
ANCHO_PANTALLA=java.awt.Toolkit.getDefaultToolkit().getScreenSize().width;
    private static final int
ALTO_PANTALLA=java.awt.Toolkit.getDefaultToolkit().getScreenSize().height-50;

    public RType()
    {
        add(new Pantalla(ANCHO_PANTALLA,ALTO_PANTALLA));

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(ANCHO_PANTALLA,ALTO_PANTALLA);
        setLocationRelativeTo(null);
        setTitle("R-Type StarWars Edition");
        setVisible(true);
        setResizable(false);

    }

    public static void main(String[] args)
    {
        new RType();
    }


}
```

Clase Pantalla

```
import java.util.ArrayList;

import javax.swing.JPanel;
import javax.swing.Timer;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Toolkit;
import java.awt.Rectangle;

/**
 * La clase Pantalla extiende JPanel e implementa el interface Actionlistener
 * ya que en esta clase se escucharán los eventos del teclado para hacer que
 * la Nave Aliada se mueva y dispare.
 * En esta clase se incluye:
 * -Pintado de los objetos existentes sobrescribiendo paint() de JPanel
 * -Logica del juego, establecemos el timer que marcará los tiempos
 * -Control de colisiones entre objetos
 *
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */


public class Pantalla extends JPanel implements ActionListener
{
    //Imagenes de los mensajes y sus posiciones
    private Imagen imagenMenu;
    private Imagen imagenVictoria;
    private Imagen imagenGameOver;

    private Posicion posicionMenu;
    private Posicion posicionVictoria;
    private Posicion posicionGameOver;

    private NaveAliada aliada;

    private int ancho; //Dimensiones de la pantalla de juego
    private int alto; //Dimensiones de la pantalla de juego

    private int dificultad; //Dificultad del juego

    //Coleccion que contendra las naves alienigenas A y B
    private ArrayList<NaveAlienigena> navesAlienigenas;

    Timer timer;

    public Pantalla(int ancho,int alto)
    {

        this.alto = alto;
        this.ancho = ancho;

        crearMensajes();

        aliada = new NaveAliada(alto/2);
        navesAlienigenas=new ArrayList<NaveAlienigena>();

        //Agrego el escuchador de teclado que creamos en NaveAliada, se crea
alli porque necesitamos pasarle cuantos pixeles ha de avanzar la nave
```

```
        //Aunque sea de la nave, lo utilizamos tambien para las opciones del
menu
        addKeyListener(aliada.getTeclado());

        setFocusable(true);
        setBackground(Color.BLACK);
        setDoubleBuffered(true);

        timer = new Timer (10, this);
        timer.start();

    }

    //Sobreescribe el método paint de Jpanel
    public void paint (Graphics grafico)
    {
        super.paint(grafico);
        Graphics2D grafico2 = (Graphics2D) grafico;

        RenderingHints rh =  new
RenderingHints(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);


rh.put(RenderingHints.KEY_RENDERING,RenderingHints.VALUE_RENDER_QUALITY);

        grafico2.setRenderingHints(rh);

        //Voy dibujando los elementos que tengan a true el atributo visible
        if (imagenMenu.getVisible())

grafico2.drawImage(imagenMenu.getImage(),posicionMenu.getX(),posicionMenu.getY
(),this);

        if (imagenVictoria.getVisible())

grafico2.drawImage(imagenVictoria.getImage(),posicionVictoria.getX(),posicionV
ictoria.getY(),this);

        if (imagenGameOver.getVisible())

grafico2.drawImage(imagenGameOver.getImage(),posicionGameOver.getX(),posicionG
ameOver.getY(),this);

        if (aliada.getVisible())

grafico2.drawImage(aliada.getImagen().getImage(),aliada.getPosicion().getX(),a
liada.getPosicion().getY(),this);

        if
(imagenVictoria.getVisible()==false&&imagenGameOver.getVisible()==false&&image
nMenu.getVisible()==false){
            ArrayList<Disparo> dis = aliada.getDisparos();

            //Dibujo los disparos que tenga en el arraylist
            for (int i=0;i<dis.size();i++)
            {
                Disparo d=(Disparo) dis.get(i);
                //if (d.getVisible())

grafico2.drawImage(d.getImagen().getImage(),d.getPosicion().getX(),d.getPosici
on().getY(),this);

            }

            //Dibujo las naves alienigenas que tenga en el arraylist
            for (int i=0;i<navesAlienigenas.size();i++)
            {
```

```
                NaveAlienigena na=navesAlienigenas.get(i);
                //if (na.getVisible())

grafico2.drawImage(na.getImagen().getImage(),na.getPosicion().getX(),na.getPos
icion().getY(),this);
            }
        }

        Toolkit.getDefaultToolkit().sync();
        grafico2.dispose();

    }


    public void actionPerformed(ActionEvent e)
    {
        if (imagenMenu.getVisible())
        {
            //leemos el teclado en el menu para establecer la dificultad del
juego
            if
(aliada.getTeclado().getTecla()>=1&&aliada.getTeclado().getTecla()<=4)
            {
                if (aliada.getTeclado().getTecla()==1)
                    dificultad=10;
                if (aliada.getTeclado().getTecla()==2)
                    dificultad=15;
                if (aliada.getTeclado().getTecla()==3)
                    dificultad=20;
                if (aliada.getTeclado().getTecla()==4)
                    dificultad=30;

                crearNavesAlienigenas();
                imagenMenu.setVisible(false);
                aliada.setVisible(true);
                addKeyListener(aliada.getTeclado());
            }
        }
        else if (imagenVictoria.getVisible())
        {
            victoria();
        }
        else if (imagenGameOver.getVisible())
        {
            derrota();
        }
        else
            jugar();

        repaint();

    }

    //Crea los objetos Imagen de los mensajes de victoria y derrota
    private void crearMensajes()
    {
        imagenMenu=new Imagen("images/menu.png");
        posicionMenu=new Posicion();
        posicionMenu.setX((ancho/2)-(imagenMenu.getAncho()/2));
        posicionMenu.setY((alto/2)-(imagenMenu.getAlto()/2));

        imagenVictoria=new Imagen("images/msg_victoria.png");
        posicionVictoria=new Posicion();
        posicionVictoria.setX((ancho/2)-(imagenVictoria.getAncho()/2));
        posicionVictoria.setY((alto/2)-(imagenVictoria.getAlto()/2));
        imagenVictoria.setVisible(false);

        imagenGameOver=new Imagen("images/msg_gameover.png");
```

```
        posicionGameOver=new Posicion();
        posicionGameOver.setX((ancho/2)-(imagenGameOver.getAncho()/2));
        posicionGameOver.setY((alto/2)-(imagenGameOver.getAlto()/2));
        imagenGameOver.setVisible(false);

    }

    //Lanza el mensaje de victoria y espera a que pulsemos la tecla C para
reiniciar el juego
    private void victoria()
    {
        aliada.setVisible(false);
        imagenVictoria.setVisible(true);
        if (aliada.getTeclado().getTecla()==10){
                reiniciar_juego();
        }
    }

    //Lanza el mensaje de derrota y espera a que pulsemos la tecla C para
reiniciar el juego
    private void derrota()
    {
        imagenGameOver.setVisible(true);
        if (aliada.getTeclado().getTecla()==10)
        {
                reiniciar_juego();
        }
    }

    //Reinicia el juego, borrando las naves alienígenas y disparos que
quedasen,
    //reubica la nave aliada en su posición original y muestra de nuevo el
menú
    private void reiniciar_juego()
    {
        borrarNavesAlienigenas();

        aliada.setVisible(false);
        aliada.borrarDisparosActivos();
        aliada.getPosicion().setX(0);
        aliada.getPosicion().setY(alto/2);

        imagenVictoria.setVisible(false);
        imagenGameOver.setVisible(false);

        imagenMenu.setVisible(true);
    }

    //Tras comprobrobar las colisiones, recorre las colecciones de disparos y
naves alienigenas,
    //moviendolos si estan visibles o borrandolos si no lo están.
    private void jugar()
    {

        aliada.setVisible(true);

        //Comprobamos las colisiones para poner invisibles los objetos que
hayan colisionado
        checkCollisions();

        ArrayList<Disparo> dis = aliada.getDisparos();

            //Recorremos la coleccionde disparos para si estan visibles
moverlos
            //y si no eliminarlos
            for (int i=0;i<dis.size();i++)
            {
                 Disparo d=(Disparo) dis.get(i);
```

```
                if(d.getVisible())
                    d.mover();
                else
                    dis.remove(i);
             }

            //Recorremos la coleccionde naves alienigenas para si estan
visibles moverlas
            //y si no eliminarlas
            if (navesAlienigenas.size()>0)
            {
                for (int i=0;i<navesAlienigenas.size();i++)
                {
                    NaveAlienigena na=navesAlienigenas.get(i);
                   if(na.getVisible())
                       na.mover();
                   else
                       navesAlienigenas.remove(i);
                }
            }
            else
                victoria();


aliada.mover(aliada.getTeclado().getDx(),aliada.getTeclado().getDy());
            aliada.disparar();

            repaint();

    }

    public ArrayList<NaveAlienigena> getNavesAlienigenas()
    {
        return navesAlienigenas;
    }

    //Atendiendo a la dificultad que indique el usuario en el menú inicial
añade las
    //NavesAlienigenas correspondientes al ArrayList navesAlienigenas.
    //La mitad de ellas serán de tipo A y la otra mitad de tipo B.
    private void crearNavesAlienigenas()
    {
        for (int i=0;i<this.dificultad;i++)
        {
            if (i%2==0)
                navesAlienigenas.add(new NaveAlienigenaA(dificultad));
            else
                navesAlienigenas.add(new NaveAlienigenaB(dificultad));

        }

    }

    //Recorre el ArrayList navesAlienigenas y borra todos sus elementos.
    private void borrarNavesAlienigenas()
    {

        for (int i=0;i<navesAlienigenas.size();i++)
        {
                navesAlienigenas.remove(i);
        }
        navesAlienigenas=new ArrayList<NaveAlienigena>();

    }

    //Comprueba las colisiones entre NaveAliada, NavesAlienigenas y Disparos
    //poniendo visibles o invisibles los que hayan colisionado
    private void checkCollisions()
```

```
    {

        //Ponemos invisibles los objetos que intersecten para ser borrados
posteriormente en el método jugar()

        Rectangle rAliada;
        Rectangle rDisparo;
        Rectangle rAlien;

        rAliada=aliada.getBounds();

        //Colision de alien y Aliada
        for (int i=0;i<navesAlienigenas.size();i++)
        {
            NaveAlienigena na=navesAlienigenas.get(i);
            rAlien=na.getBounds();

            if (rAliada.intersects(rAlien))
            {
                aliada.setVisible(false);
                derrota();

            }

        }

       //Colision de alien y disparos
       ArrayList<Disparo> dis = aliada.getDisparos();

       for (int i=0;i<dis.size();i++)
       {
                Disparo d = (Disparo) dis.get(i);

                rDisparo = d.getBounds();

                for (int j=0;j<navesAlienigenas.size();j++)
                {
                    NaveAlienigena na=navesAlienigenas.get(j);
                    rAlien=na.getBounds();

                    if (rDisparo.intersects(rAlien))
                    {
                        d.setVisible(false);
                        na.setVisible(false);
                        //dis.remove(i);
                        //navesAlienigenas.remove(j);
                    }

                }
       }
    }

}
```

Clase Imagen

```
import java.awt.Graphics2D;
import java.awt.Graphics;
import java.awt.Image;

import javax.swing.ImageIcon;

/**
 * La clase Imagen ha sido creada ya que todos los elementos del juego
 * ya sean naves, disparos o mensajes, tienen una imagen asignada.
 * Esta clase nos permite crear objetos de tipo imagen desde la ruta que le
indiquemos.
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */


public class Imagen
{
    private int alto;
    private int ancho;
    private int x;
    private int y;

    private Image png;//icono
    private boolean visible;


    public Imagen(String ruta)
    {

        ImageIcon ii = new ImageIcon(this.getClass().getResource(ruta));
        png = ii.getImage();

        visible=true;

    }


    public int getAncho()
    {
        return png.getWidth(null);
    }

    public int getAlto()
    {
        return png.getHeight(null);
    }

    public boolean getVisible()
    {
        return visible;
    }

    public void setVisible(boolean visible)
    {
        this.visible=visible;
    }

    public Image getImage()
    {
        return png;
    }


}
```

Clase Posicion

```
/**
 * La clase Posicion ha sido creada debido a que todos los elementos del juego
 * ya sean naves o disparos o mensajes tienen una posición asignada
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */

public class Posicion
{

    private int x;
    private int y;


    public Posicion()
    {
        x = 0;
        y = 0;
    }


    public void setX(int x)
    {
        this.x=x;
    }

    public void setY(int y)
    {
        this.y=y;
    }

    int getX()
    {
        return x;
    }

    int getY()
    {
        return y;
    }


}
```

Clase Elemento

```
import java.awt.Rectangle;

/**
 * La clase Elemento es la superclase abstracta de la que heredan todos los
objetos
 * que puedan dibujarse en nuestro JPanel a excepción del menú y los mensajes
 * de victoria y derrota. Estos objetos son la NaveAliada, Disparo,
NaveAlienigenaA y NaveAlienigenaB.
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */

public abstract class Elemento
{

    private Posicion posicionElemento;
    private Imagen imagenElemento;

    private boolean visible;

    //limite inferior que los Elementos no deben rebasar. Es lo alto de la
pantalla del equipo menos 80px para salvar la barra de tareas.
    private static final int
LIMITE_INFERIOR=java.awt.Toolkit.getDefaultToolkit().getScreenSize().height-
80;
    //limite derecho que los Elementos no deben rebasar. Este se utiliza
principalmente en la NaveAliada
    private static final int
LIMITE_DERECHO=java.awt.Toolkit.getDefaultToolkit().getScreenSize().width;


    public Elemento()
    {
        posicionElemento = new Posicion();
        visible=true;
    }

    Imagen getImagen()
    {
        return imagenElemento;
    }

    Posicion getPosicion()
    {
        return posicionElemento;
    }

    public  void setImagen(String ruta)
    {
        imagenElemento = new Imagen(ruta);
    }

    public void setVisible(boolean estado)
    {
        visible=estado;
    }

    public boolean getVisible()
    {
        return visible;
    }

    public int getLimiteDerecho()
    {
        return LIMITE_DERECHO;
```

```
    }

    public int getLimiteInferior()
    {
        return LIMITE_INFERIOR;
    }

    //Devuelve el rectangulo asociado al elemento para comprobar las
colisiones entre objetos
    public Rectangle getBounds()
    {
        int x=this.getPosicion().getX();
        int y=this.getPosicion().getY();
        int ancho=this.getImagen().getAncho();
        int alto=this.getImagen().getAncho();

        return new Rectangle(x,y,ancho,alto);

    }

    //El metodo mover se sobrescribe en las subclases
    public void mover()
    {

    }

}
```

Clase NaveAliada

```
import java.util.ArrayList;

/**
 * La clase NaveAliada extiende a Elemento. En ella se definirán las
peculiaridades de esta nave
 * como su imagen, la velocidad a la que ha de moverse, la posición en la que
ha de aparecer,
 * los limites que ha de respetar en su movimiento así como un controlador de
teclado para que
 * el usuario pueda moverla
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */

public class NaveAliada extends Elemento
{

    private static final int VELOCIDAD_NAVEALIADA=4;

    private KeyListener teclado;
    private ArrayList<Disparo> disparosActivos;

    public NaveAliada(int centro)
    {
       setImagen("images/aliada_obiwan_m.png");

       //Le añadimos a este objeto un controlador de teclado, pasando como
parametro la velocidad a la que queremos
       //que se mueva la NaveAliada
       teclado=new KeyListener(VELOCIDAD_NAVEALIADA);

       //creo el arraylist para almacenar los disparos
       disparosActivos=new ArrayList<Disparo>();

       //situo la nave en el centro del alto la pantalla
       this.getPosicion().setY(centro);
       setVisible(false);


    }

    public KeyListener getTeclado()
    {
        return teclado;
    }

    public ArrayList<Disparo> getDisparos()
    {
        return disparosActivos;
    }

    //cuando llamemos a disparar, agregamos elementos al arraylist de
disparos, situandolos en el centro de la nave
    public void disparar()
    {
        Disparo disparo;

        if (teclado.getEspacio()==1)
        {

            //Creamos y configuramos la posicion donde se crea el disparo
            disparo=new Disparo();
            //Coordenada X es donde acaba la NaveAliada
            disparo.getPosicion().setX(this.getPosicion().getX() +
this.getImagen().getAncho());
```

```
            //Coordenada Y es la mitad de la altura de la NaveAliada menos la
mitad de la altura del disparo, para que quede centrado
            disparo.getPosicion().setY(this.getPosicion().getY() +
(this.getImagen().getAlto()/2) - (disparo.getImagen().getAlto()/2));

            disparosActivos.add(disparo);

            teclado.setEspacio(0);
        }
    }


    public void borrarDisparosActivos()
    {
        //Elimino todos los elementos del arraylist de disparos
        for (int i=0;i<disparosActivos.size();i++)
        {
                disparosActivos.remove(i);
        }
        //Reinicializo la coleccion para asegurar que todos los elementos sean
borrados
        disparosActivos=new ArrayList<Disparo>();
    }

    public void mover(int dx, int dy)
    {

        if (this.getPosicion().getX()+dx<=(this.getLimiteDerecho()-
this.getImagen().getAncho())&&
            this.getPosicion().getX()+dx>=0)
        {
            this.getPosicion().setX(this.getPosicion().getX()+dx);
        }

        if (this.getPosicion().getY()+dy<=(this.getLimiteInferior()-
this.getImagen().getAlto())&&
            this.getPosicion().getY()+dy>=0)
        {
            this.getPosicion().setY(this.getPosicion().getY()+dy);
        }

    }

}
```

Clase KeyListener

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


/**
 * La clase KeyListener extiende a KeyAdapter, crea un escuchador de eventos
de teclado
 * que controlará las teclas que han sido pulsadas y en base a ello modificara
sus campos
 * dx y dx que indican las coordenadas a las que ha de moverse la NaveAliada
cada vez que
 * se detecte una pulsación de teclas de dirección.
 *
 * También guarda en sus variables espacio y tecla si se ha pulsado un
espacio,
 * que lo utilizaremos para crear disparos o si se ha pulsado alguna tecla
necesaria como la 'c'
 * cuando tras una victoria o derrota queremos continuar el juego.
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */


public class KeyListener extends KeyAdapter
{

    private int dx;
    private int dy;
    private int avance;
    private int espacio;
    private int tecla;

    //constructor para escuchador de teclado simple. Cuando no necesitamos
aplicar un avance, por ejemplo en menus
    public KeyListener()
    {
        dx=0;
        dy=0;
        espacio=0;
        tecla=0;
    }

    //constructor para escuchador de teclado que aplique el avance que le
pasemos por parametro. Para la nave aliada
    public KeyListener(int avance)
    {
        dx=0;
        dy=0;
        this.avance=avance;
        espacio=0;
        tecla=0;
    }

    public int getDx()
    {
        return dx;
    }

    public int getDy()
    {
        return dy;
    }

    public int getEspacio()
    {
```

```
        return espacio;
    }

    public int getTecla()
    {
        return tecla;
    }

    public void setEspacio(int espacio)
    {
        this.espacio=espacio;
    }



    public void keyPressed(KeyEvent e)
    {

        int key = e.getKeyCode();

        if (key==KeyEvent.VK_LEFT||key==KeyEvent.VK_O){
                dx =- avance;

        }

        if (key==KeyEvent.VK_RIGHT||key==KeyEvent.VK_P){
                dx = avance;

        }

        if (key==KeyEvent.VK_UP||key==KeyEvent.VK_Q){
                dy =- avance;
        }

        if (key==KeyEvent.VK_DOWN||key==KeyEvent.VK_A){
                dy = avance;
        }

        if (key==KeyEvent.VK_1){
                tecla = 1;
        }

        if (key==KeyEvent.VK_2){
                tecla = 2;
        }

        if (key==KeyEvent.VK_3){
                tecla = 3;
        }

        if (key==KeyEvent.VK_4){
                tecla = 4;
        }

        if (key==KeyEvent.VK_C){
                tecla = 10;
        }


        espacio=0;

    }

    public void keyReleased(KeyEvent e)
    {

        int key = e.getKeyCode();
```

```
        if (key==KeyEvent.VK_LEFT||key==KeyEvent.VK_O){
            dx = 0;
        }

        if (key==KeyEvent.VK_RIGHT||key==KeyEvent.VK_P){
            dx = 0;
        }

        if (key==KeyEvent.VK_UP||key==KeyEvent.VK_Q){
            dy = 0;
        }

        if (key==KeyEvent.VK_DOWN||key==KeyEvent.VK_A){
            dy = 0;
        }

        if (key==KeyEvent.VK_SPACE)
            espacio=1;

        tecla = 0;

    }

}
```

Clase Disparo

```
/**
 * La clase Disparo extiende a Elemento. En ella se definirán las
peculiaridades
 * del elemento Disparo como su Imagen o la velocidad a la que debe
desplazarse por la pantalla
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */

public class Disparo extends Elemento
{
    // instance variables - replace the example below with your own
    boolean visible;

    private static final int VEL_DISPARO=8;

    public Disparo()
    {
        this.setImagen("images/laser_m.png");
        visible=true;
    }

    public void mover()
    {
        //Movimiento horizontal
        this.getPosicion().setX(this.getPosicion().getX()+VEL_DISPARO);

        //Cuando el disparo sale de la pantalla desaparece, para ser borrado
posteriormente
        if(this.getPosicion().getX()>getLimiteDerecho())
            this.visible=false;
    }

}
```

Clase NaveAlienigena

```
import java.util.Random;

/**
 * Esta clase será superclase de NaveAlienigenaA y NaveAlienigenaB, no
podremos instanciar objetos
 * de esta pero en ella inicializaremos algunos campos de la naves como su
posición utilizando
 * dos variables aleatorias para posicionar inicialmente las naves en
distintas posiciones.
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */


public abstract class NaveAlienigena extends Elemento
{

    private boolean visible;
    Random Aleatorio;
    //constantes para espaciar las naves entre si
    //el margen en pixeles de alto para que las naves no aparezcan ni muy
arriba ni muy abajo
    //el espaciado en pixeles a lo ancho para que vayan saliendo poco a poco e
intentando que no se solapen
    private static final int MARGENES_ALTO=200;
    private static final int ESPACIADO_ANCHO=1500;

    public NaveAlienigena()
    {
        Aleatorio=new Random();

        //Creamos dos variables aleatorias para posicionar inicialmente las
naves en distintas posiciones
        //intentando que se ubiquen a lo alto de la pantalla y a lo ancho de
1500 pixeles a la derecha
        //de la pantalla, para que luego vayan apareciendo por la derecha
repartidas por la pantalla
        int indiceY=Aleatorio.nextInt(getLimiteInferior()-MARGENES_ALTO);
        int indiceX=Aleatorio.nextInt(ESPACIADO_ANCHO);

        this.getPosicion().setX(getLimiteDerecho()+indiceX);
        this.getPosicion().setY(indiceY);

        this.visible=true;
    }


    public void mover()
    {
        //Este metodo será sobreescrito en las subclases NaveAlienigenaA y
NaveAlienigenaB
        //ya que cada tipo tiene distinto movimiento.

    }


}
```

Clase NaveAlienigenaA

```
import java.util.Random;

/**
 * La clase NaveAlienigenaA extiende a la superclase NaveAlienigena, agregando las características
 * que no estaban definidas en su superclase, como una imagen específica para el tipo A,
 * o la velocidad a la que debe moverse, que se le pasará por parámetro en su constructor.
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */



public class NaveAlienigenaA extends NaveAlienigena
{

    private int velocidad;


    public NaveAlienigenaA(int dificultad)
    {
        if (dificultad==10)
            velocidad=2;

        if (dificultad==15)
            velocidad=4;

        if (dificultad==20)
            velocidad=5;

        if (dificultad==30)
            velocidad=7;

        setImagen("images/alien_A_m.png");

    }

    public void mover()
    {
        //Movimiento Horizontal
        getPosicion().setX(getPosicion().getX()-velocidad);

        //CONTROL DE MARGEN ANCHO
        if ((getPosicion().getX()+getImagen().getAncho())<0)
            getPosicion().setX(getLimiteDerecho());

    }

}
```

Clase NaveAlienigenaB

```
import java.util.Random;

/**
 * La clase NaveAlienigenaB  al igual que NaveAlienigenaA extiende a la
superclase NaveAlienigena,
 * agregando las características que no estaban definidas en su superclase,
como una imagen específica
 * para el tipo B, o la velocidad a la que debe moverse, que se le pasará por
parámetro en su constructor.
 *
 * @author Jesus Lopez Garcia
 * @version 1.1
 */


public class NaveAlienigenaB extends NaveAlienigena
{

    private int velocidad;
    private static final int FREC_CAMBIO=50; //Cada cuantos pixeles cambiara
la nave de direccion

    private boolean visible;
    Random Aleatorio;
    private int direccionV;//Direccion vertical. 0 sigue de frente, 1 sube, 2
baja.
    private int contador;


    public NaveAlienigenaB(int dificultad)
    {
        if (dificultad==10)
            velocidad=2;

        if (dificultad==15)
            velocidad=4;

        if (dificultad==20)
            velocidad=5;

        if (dificultad==30)
            velocidad=7;

        Aleatorio=new Random();

        setImagen("images/alien_B_m.png");
        direccionV=Aleatorio.nextInt(3);
        contador=FREC_CAMBIO;

    }


    public void mover()
    {

        //Movimiento horizontal
        //this.getPosicion().setX(this.getPosicion().getX()-this.velocidad);
        getPosicion().setX(getPosicion().getX()-velocidad);

        int direccion=Aleatorio.nextInt(100);

        //Nave B Puede moverse de arriba a abajo aleatoriamente. Controla el
movimiento vertical para que la nave no tiemble
        //la variable contador hace que solo cambie de direccion cada tantos
pixeles como se establezcan en frec_cambio
        if(contador==0)
```

```
        {
            direccionV=Aleatorio.nextInt(3);
            contador=FREC_CAMBIO;
        }

        if (direccionV==2)
            getPosicion().setY(getPosicion().getY()+velocidad);

        if (direccionV==1)
            getPosicion().setY(getPosicion().getY()-velocidad);


        //CONTROL DE MARGEN  ANCHO
        if ((getPosicion().getX()+getImagen().getAncho())<0)
            getPosicion().setX(getLimiteDerecho());

        //CONTROL DE MARGEN NAVE B - ALTO
        if (getPosicion().getY()<0)
        {
            getPosicion().setY(0);
            direccionV=Aleatorio.nextInt(3);
        }

        if ((getPosicion().getY()+getImagen().getAlto())>getLimiteInferior())
        {
            getPosicion().setY(getLimiteInferior()-getImagen().getAlto());
            direccionV=Aleatorio.nextInt(3);
        }

        contador--;

    }

}
```