La practica, consiste en desarrollar en java una versión del clásico juego R-Type. El programa debe cumplir con los siguientes requisitos, enumerados en el enunciado de la practica:

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 láser 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 varia 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.

FUNCIONAMIENTO DEL JUEGO.

El jugador controla una nave aliada, que aparece a la izquierda de la ventana de juego. Deberá eliminar todas las naves enemigas que aparecerán por la derecha de la pantalla, para ello, dispondrá de una sola vida.
La nave aliada se controla con las siguientes teclas: Q - arriba, A - Abajo, O - izquierda, P - derecha, SPACE disparo.
Las naves enemigas son de dos tipos, las de tipo A, son las primeras en aparecer y su movimiento es siempre en línea recta y a velocidad constante.
Las naves enemigas de tipo B, aparecerán justo cuando hayan terminado de salir las de tipo A y se moverán con desplazamientos en zig zag con distancias aleatorias hacia arriba y abajo y a velocidad constante.
Las naves enemigas que consigan alcanzar el final de la pantalla sin ser eliminadas, volverán a aparecer por la derecha de la pantalla de juego.
La única forma de eliminar la nave aliada es que se produzca una colisión con una nave enemiga.
Si la nave aliada es eliminada o todas las naves enemigas son eliminadas, finaliza el juego.
El jugador deberá seleccionar, en la barra de menú de la pantalla principal del juego, el nivel de dificultad y en función de la dificultad seleccionada, aumentara la velocidad del juego y el numero de naves enemigas.
Una vez seleccionada la dificultad de la partida, se pulsara JUGAR para comenzar la batalla. Si no se selecciona ninguna dificultad, por defecto, el juego iniciara en modo FACIL.
Una vez terminada la partida, se mostrara en pantalla un mensaje de felicitación, en caso de haber ganado la batalla, o un mensaje de consolación en caso de haber perdido, se pulsara el menú INICIO, para

volver a la pantalla principal del juego y poder comenzar una nueva partida.

ANALISIS Y DISEÑO DEL PROGRAMA.

El juego consta principalmente de cuatro elementos: nave aliada, nave enemiga alien A, nave enemiga alien B y misiles.
Las imágenes de las naves aliada, alienA, alienB, misiles, pantalla principal, pantalla de partida ganada, pantalla de partida perdida y fondo de la pantalla de juego, se implementan mediante archivos jpg y png que se almacenan dentro del directorio principal del proyecto en la carpeta images.
Se han implementado seis clases en total, una clase para cada uno de los elementos del juego (ALIADO, ALIENA, ALIENB, MISIL), una clase denominada BATALLA que implementa el control principal del juego y una clase inicial llamada RType que implementa la pantalla de inicio, el menú de opciones y lanza el juego.
La clases que implementan los elementos del juego, (ALIADO, ALIENA, ALIENB, MISIL), se han empaquetado en un package llamado naves, de esta forma, se facilita la reutilización de código, permitiendo utilizar la definición de las clases ALIADO, ALIENA, ALIENB y MISIL para la creación de otro programa, o poder sustituirlas en este.
El programa implementa un JFrame de tamaño fijo 800x600 y sobre este un JPanel. En este JPanel, se muestra tanto la zona de juego como la barra de menú y las pantallas de inicio y fin de juego.
El control sobre el estado del juego, se realiza mediante tres "banderas" booleanas:

- jugando: con valor true indica que estamos jugando una partida y con valor false, no se esta jugando.
  Por defecto se inicia con valor false.
- nueva_partida: con valor true indica que el programa esta listo para iniciar una nueva partida y con valor false que no lo esta.
  Por defecto se inicia con valor true.
- partida_ganada: con valor true indica que la partida se ha ganado y con valor false, que se ha perdido.
  Por defecto se inicia con valor true.

Por otro lado, se definen otras dos variable que también influyen sobre el control del programa:

- nivel: indica el nivel de dificultad del juego, el numero de naves enemigas que aparecerán.
- velocidad: indica la velocidad a la que se desplazaran las naves, tanto enemigas como la nave aliada.

Para seleccionar opciones en el juego, se ha implementado un JMenuBar con las siguientes opciones:

FACIL: selecciona el modo fácil del juego. Es el modo por defecto en caso de no seleccionar ningún nivel de dificultad.
Al seleccionar esta opción del menú, la variable nivel se actualiza a 10, mostrando un total de 10 naves alienA y 10 naves alienB, además el valor de la variable velocidad pasara a 1, la mínima velocidad.

NORMAL: selecciona el modo normal del juego.
Al seleccionar esta opción del menú, la variable nivel se actualiza a 15, mostrando un total de 15 naves alienA y 15 naves alienB, además el valor de la variable velocidad pasara a 2.

COMPLICADO: selecciona el modo compilado del juego.
Al seleccionar esta opción del menú, la variable nivel se actualiza a 20, mostrando un total de 20 naves alienA y 20 naves alienB, además el valor de la variable velocidad pasara a 3.

IMPOSIBLE: selecciona el modo imposible del juego.
Al seleccionar esta opción del menú, la variable nivel se actualiza a 30, mostrando un total de 30 naves alienA y 30 naves alienB, además el valor de la variable velocidad pasara a 4, la máxima velocidad.

JUGAR: Con esta opción del menú, comienza la partida, para ello actualiza el valor de la variable jugando a true.

INICIO: Con esta opción del menú, volvemos a la pantalla principal del juego desde cualquier pantalla de final de partida.
Al seleccionar esta opción, se reestablecen los valores de nivel de dificultad y velocidad a los valores por defecto, los del nivel fácil, 10 enemigos y velocidad 1 y se actualiza el valor de nueva_partida a true indicando que el programa esta preparado para comenzar una nueva partida.

SALIR: Con esta opción, termina el programa.

La nave aliada, esta implementada mediante un objeto ALIADO. Su movimiento se controla mediante teclado, para ello, se hace uso de las clases KeyEvent y KeyListener, que capturan las pulsaciones de teclado. La nave aliada se moverá por la pantalla en función de la velocidad del juego, desplazándola 1 píxel en la dirección adecuada si la velocidad es 1 (nivel de juego fácil), 2 píxeles si la velocidad es 2 (nivel de juego normal)

y así sucesivamente hasta la máxima velocidad de 4 píxeles (nivel de juego imposible).
Las naves enemigas tipo A y tipo B, se implementan mediante 2 Arrays de objetos ALIENA y ALIENB dentro de la clase ALIADO. Para cada array, se crean y se introducen en el tantas naves enemigas como indique el nivel de juego. Para mover las naves enemigas, se implementa un Timer, ya que las naves enemigas no se mueven en función de pulsaciones de teclado. De manera que a cada golpe de timer, se actualiza la posición de la nave en pantalla, en función del tipo de nave, Los alienA solo se moverán de derecha a izquierda en línea recta y con velocidad constante, mientras que los alienB también podrán moverse de arriba a abajo. La implementación de la velocidad para las naves enemigas se realiza de forma análoga a la nave aliada, para velocidad 1 se desplazaran 1 píxel en pantalla, para velocidad 2, 2 píxeles etc...
Los misiles, se implementan mediante un array de objetos MISIL dentro de la clase ALIADO, solo que al principio el array esta vacío. Se irán añadiendo misiles al array según se vayan disparando (pulsando la tecla SPACE). Por tanto, la creación de misiles estará controlada por KeyEvent y KeyListener mientras que el movimiento de los misiles en pantalla se hará en función del Timer, al igual que las naves enemigas. Una vez que el misil alcanza el final de la pantalla sin haber colisionado con nada, el misil será destruido y sacado del array de misiles.
El timer, también se utiliza para repintar en pantalla todos los elementos, de manera que a cada golpe de timer, se repintarían todas las naves y misiles con sus posiciones actualizadas, para ello se utiliza el método paint.
En cuanto al control de colisiones, se realiza mediante la comprobación del solapamiento de los rectángulos en los que están circunscritas las imágenes de la nave aliada, alienA, alienB y misiles.
Si comprobamos que el rectángulo de la nave aliada se solapa con el rectángulo de alguna nave enemiga, se ha producido una colisión y el juego termina, ya que se habría destruido la nave aliada.
Por otro lado, si el rectángulo de una nave enemiga se solapa con el rectángulo de un misil, esa nave enemiga seria destruida y es eliminada del su array correspondiente, array alienA si es una nave tipo A o array alienB si es una nave tipo B, de igual forma, el misil que ha colisionado con la nave enemiga, también es eliminado del array de misiles.
La implementación del método para comprobar el solapamiento de los rectángulos de las naves (colisiones), se ha realizado de forma manual, obteniendo las coordenadas X Y de los dos objetos que se quiere comprobar su colisión, y comprobando todas las posibles situaciones en que dos rectángulos pueden estar solapados, en función de estas coordenadas X Y.

CLASES Y METODOS.

CLASE RType.

```
public class RType extends JFrame {
   JMenuBar barra_menu;
   JMenuItem elemento_facil;
   JMenuItem elemento_normal;
   JMenuItem elemento_complicado;
   JMenuItem elemento_imposible;
   JMenuItem elemento_jugar;
   JMenuItem elemento_inicio;
   JMenuItem elemento_salir;
   BATALLA batalla;
```

Es la clase principal. Se implementa con herencia a la clase JFrame de java. En esta clase se definen los elementos del menú de opciones del juego, así como un objeto de tipo BATALLA que implementara el desarrollo del juego.

METODOS DE LA CLASE RType.

public RType()

Es el constructor de la clase RType.
Crea la barra de menú con todas sus opciones y define para cada elemento de la barra de menú un ActionListener para poder capturar la pulsación del ratón sobre cada elemento.
Crea un objeto de la clase BATALLA, que será el encargado de controlar el desarrollo del juego.
Define el JFrame donde se va a mostrar el juego.

private class FacilActionListener implements ActionListener
public void actionPerformed(ActionEvent e)

Clase privada que implementa el ActionListener de la opción FACIL de la barra de menú.
Actualiza el nivel de la partida a 10 (numero de naves alienA y naves alienB) y la velocidad a 1.

private class NormalActionListener implements ActionListener {
public void actionPerformed(ActionEvent e) {

Clase privada que implementa el ActionListener de la opción NORMAL de la barra de menú.

Actualiza el nivel de la partida a 15 (numero de naves alienA y naves alienB) y la velocidad a 2.

private class ComplicadoActionListener implements ActionListener {
public void actionPerformed(ActionEvent e) {
Clase privada que implementa el ActionListener de la opción COMPLICADO de la barra de menú.
Actualiza el nivel de la partida a 20 (numero de naves alienA y naves alienB) y la velocidad a 3.
private class ImposibleActionListener implements ActionListener {
public void actionPerformed(ActionEvent e)
Clase privada que implementa el ActionListener de la opción IMPOSIBLE de la barra de menú.
Actualiza el nivel de la partida a 30 (numero de naves alienA y naves alienB) y la velocidad a 4.

private class JugarActionListener implements ActionListener
public void actionPerformed(ActionEvent e)
Clase privada que implementa el ActionListener de la opción JUGAR de la barra de menú.
Si no se esta jugando y se espera una nueva partida, pone los valores de las variables de control (jugando, nueva_partida, partida_ganada) a true e inicia los parámetros del juego.

private class InicioActionListener implements ActionListener
public void actionPerformed(ActionEvent e)
Clase privada que implementa el ActionListener de la opción INICIO de la barra de menú.
Si no se esta jugando, inicia el nivel del juego y la velocidad a los valores por defecto, nivel 10 y velocidad 1, y pone a verdadero la variable nueva_partida, indicando que el programa esta listo para iniciar una nueva partida.

private class SalirActionListener implements ActionListener
public void actionPerformed(ActionEvent e)
Clase privada que implementa el ActionListener de la opción SALIR de la barra de menú.
Termina la ejecución del programa.

CLASE BATALLA.

public class BATALLA extends JPanel implements ActionListener
private Timer timer;

```
    private ALIADO aliado;
    private int NIVEL = 10;
    private int VELOCIDAD = 1;
    private Image fondo;
    private boolean jugando = false;
    private boolean nueva_partida = true;
    private boolean partida_ganada = true;
```

En esta clase se implementa el desarrollo principal del juego. Se implementa con herencia a la clase JPanel de java e implementa ActionListener para hacer las capturas de las pulsaciones de teclado y un timer..
En esta clase se definen los elementos:
Objeto timer, para implementar el timer utilizado.
Objeto aliado del tipo ALIADO, implementa la nave aliada.
Variable NIVEL que almacena el nivel de juego, por defecto es 10, nivel fácil (indica el numero de naves alienA y alienB que aparecerán).
Variable VELOCIDAD, que almacena la velocidad del juego, velocidad del movimiento de las naves. Por defecto se inicia a 1, la velocidad en modo fácil.
Objeto fondo de tipo Image. Almacenara las imágenes del fondo de la pantalla de juego, la imagen de la pantalla de inicio y las imágenes de partida ganada y partida perdida.
Variable jugando de tipo boolean. Indica si se esta jugando una partida o no.
Variable nueva_partida. Indica si se esta esperando para comenzar una nueva partida.
Variable partida_ganada. Indica si la partida se ha ganado o no.

<u>METODOS DE LA CLASE BATALLA.</u>

```
    public BATALLA() {
```

Es el constructor de la clase BATALLA.
Crea el KeyListener para capturar las pulsaciones de teclado.
Inicia el JPanel.
Crea la nave aliada.
Crea el timer y lo inicia.

public void set_jugando(boolean j)
Actualiza el valor de la variable jugando a true o false.

public boolean get_jugando() {
Devuelve el valor de la variable jugando.

public void set_nueva_partida(boolean n)
Actualiza el valor de la variable nueva_partida a true o false.

public boolean get_nueva_partida()
Devuelve el valor de la variable nueva_partida.

public void set_partida_ganada(boolean pg)
Actualiza el valor de la variable partida_ganada a true o false.

public boolean get_partida_ganada()
Devuelve el valor de la variable partida_ganada.

public void set_nivel(int n)
Actualiza el valor de la variable nivel.

public void set_velocidad(int v)
Actualiza el valor de la variable velocidad.

public void paint(Graphics g)
Este método, se encarga de pintar / repintar los elementos del juego en función de la situación del juego.
Si se esta jugando dibuja el fondo de juego, nave aliada, misiles, alienA y alienB .
Si no se esta jugando y se espera una nueva partida, dibuja el menú principal.
Si no se esta jugando y no se espera una nueva partida y se ha ganado la partida dibuja el fondo de partida ganada.
Si no se esta jugando y no se espera una nueva partida y no se ha ganado la partida dibuja el fondo de partida perdida.

public void actionPerformed(ActionEvent e)
Este método, implementa el desarrollo principal del juego.
Se ejecuta cada vez que se produce un evento.
Si se esta jugando (variable jugando = true), movemos los misiles. Recorre todo el Array de misiles y actualiza sus posiciones en pantalla. Si el misil esta fuera de la pantalla de juego, lo pone como no visible.
Mueve la nave aliada
Mueve las naves alienA. Recorre todo el Array de naves alienA y actualiza sus posiciones en pantalla. Si la nave alienA no es visible, (ha sido destruida) la elimina del Array.

Mueve las naves alienB. Recorre todo el Array de naves alienB y actualiza sus posiciones en pantalla. Si la nave alienB no es visible, (ha sido destruida), la elimina del Array.
Comprueba si hay colisiones de misil con naves alienA. Para ello, por cada misil comprueba todas la naves del Array alienA en busca de colisión. Si hay colisión pone el indicador visible de la nave alienA y misil a falso indicando que no es visible.
Comprueba si hay colisiones de misil con naves alienB. Para ello, por cada misil comprueba todas la naves del Array alienB en busca de colisión. Si hay colisión, pone el indicador visible de la nave alienB y misil a falso indicando que no es visible.
Detectamos colisiones de nave aliada con naves alienA. Si detecta una colisión pone la nave alienA a no visible, el indicador de jugando a false indicando que se ha terminado la partida, el indicador de partida ganada a falso y nueva partida a falso.
Detectamos colisiones de nave aliada con naves alienB. Si detecta una colisión pone la nave alienB a no visible, el indicador de jugando a falso indicando que se ha terminado la partida, el indicador de partida ganada a falso y nueva partida a falso.
Limpiamos el array de misiles no validos, misiles que han salido de la pantalla o han hecho un impacto, para ello recorre el Array de misiles y elimina aquellos que no son visibles.
Limpiamos el array de alienA no validas, naves alienA que han sido destruidas para ello, recorre el Array de naves alienA y elimina aquellas que no son visibles.
Limpiamos el array de alienB no validas, naves alienB que han sido destruidas, para ello, recorre el Array de naves alienB y elimina aquellas que no son visibles.
Comprueba el numero de alienA y alienB en juego de manera que si es 0, termina la partida, ganando el juego. Actualiza el indicador jugando a falso indicando que se ha terminado el juego, el indicador partida ganada a true indicando que se ha terminado la partida y el indicador nueva partida a falso, indicando que no se espera una nueva partida.
Por ultimo, repinta todos los elementos de pantalla a sus nuevas posiciones.

public boolean impacto (int mpx1, int mpy1, int mpx2, int mpy2, int apx1, int apy1, int apx2, int apy2)

Comprueba si ha habido un impacto o no entre dos elementos. Para ello comprueba si los rectángulos de los dos elementos están solapados o no. Se pasan como parámetros los puntos superior izquierdo e inferior derecho de los 2 elementos que queremos comprobar.

Comprueba todas las situaciones en que dos rectángulos pueden estar solapados y devuelve true si los 2 rectángulos se solapan, hay impacto, y false en caso contrario.

public void dibujar_menu(Graphics g)
Muestra la pantalla principal del juego.

public void dibujar_fondo(Graphics g)
Muestra el fondo de la pantalla de juego.

public void dibujar_fondo_partida_ganada(Graphics g)
Muestra la imagen de partida ganada.

public void dibujar_fondo_partida_perdida(Graphics g){
Muestra la imagen de partida perdida.

public void dibujar_aliado(Graphics g)
Muestra en pantalla la imagen de la nave aliada, en las coordenadas (X,Y) en las que este situada la nave aliada.

public void dibujar_misiles(Graphics g)
Muestra en pantalla las imágenes de los misiles que hay activos en la pantalla, en las coordenadas (X,Y) en las que estén situados.

public void dibujar_aliena(Graphics g)
Muestra en pantalla las imágenes de las naves alienA que hay activas en la pantalla, en las coordenadas (X,Y) en las que estén situadas.

public void dibujar_alienb(Graphics g)
Muestra en pantalla las imágenes de las naves alienB que hay activas en la pantalla, en las coordenadas (X,Y) en las que estén situadas.

public void iniciar_parametros()
Inicia los parámetros del juego.
Borra los Arrays de misiles, naves alienA y naves alienB.
Inicia el nivel del juego al nivel de dificultad seleccionado y la velocidad del juego a la velocidad seleccionada, en función del nivel de dificultad.
Inicia la posición inicial (x,y) de la nave aliada.
Crea los Arrays alienA y alienB con el numero de naves especificadas por el nivel de dificultad seleccionado en el menú principal.

private class TAdapter extends KeyAdapter
public void keyReleased(KeyEvent e)
public void keyPressed(KeyEvent e)

Implementa una clase privada para capturar las pulsaciones de teclado y la liberación de la tecla pulsada.

CLASE ALIADO.

public class ALIADO

```
private int dx;
private int dy;
private int x;
private int y;
private int velocidad;
private int nivel;
private Image image_aliado;
private int alto;
private int ancho;
private ArrayList<MISIL> misiles;
private ArrayList<ALIENA> alienA;
private ArrayList<ALIENB> alienB;
```

Esta clase implementa la nave aliada.
La variable dx almacena la dirección en el eje X, en la que se moverá la nave aliada en función de la tecla pulsada.
La variable dy almacena la dirección en el eje Y, en la que se moverá la nave aliada, en función de la tecla pulsada.
Las variables (x,y), almacenan las coordenadas de la posición actual en la que se encuentra la nave aliada.
La variable velocidad, almacena la velocidad a la que se desplaza la nave aliada.
La variable nivel, almacena el nivel de dificultad del juego, el numero de naves alienA y alienB que aparecerán en pantalla.
La variable image_aliado, almacena la imagen de la nave aliada.
La variable alto, almacena el alto de la imagen de la nave aliada, que se utilizara para hacer las comprobaciones de las colisiones.
La variable ancho, almacena el ancho de la imagen de la nave aliada.
misiles, es un array de objetos MISIL, que almacena los misiles que han sido disparados y están activos.
alienA, es un array de objetos ALIENA, que almacena las naves enemigas de tipo A, que están en pantalla.
alienB, es un array de objetos ALIENB, que almacena las naves enemigas de tipo B, que están en pantalla.

METODOS DE LA CLASE ALIADO.

public ALIADO()
Constructor de la clase ALIADO.
carga la imagen de la nave aliada y obtiene su alto y ancho.

Crea el Array de misiles, naves alienA y naves alienB.
inicia la posición inicial de la nave aliada y la velocidad y nivel de juego por defecto.

public void mover_aliado()
Actualiza las coordenadas x y de la nave aliada comprueba que no se salga de los limites de la pantalla.

public void set_velocidad(int v)
Actualiza la velocidad a la que se mueve la nave aliada.

public void set_nivel(int n)
Actualiza el nivel de dificultad.

public void set_posx(int px)
Actualiza la coordenada x de la posición de la nave aliada en pantalla.

public void set_posy(int py)
Actualiza la coordenada y de la posición de la nave aliada en pantalla.

public int get_posx_aliado()
Devuelve la coordenada x de la posición en la que se encuentra la nave aliada en pantalla.

public int get_posy_aliado()
Devuelve la coordenada y de la posición en la que se encuentra la nave aliada en pantalla.

public int get_alto_aliado()
Devuelve el alto de la imagen de la nave aliada.

public int get_ancho_aliado()
Devuelve el ancho de la imagen de la nave aliada.

public Image get_image_aliado()
Devuelve la imagen de la nave aliada.

public ArrayList<MISIL> get_misiles()
Devuelve el Array de misiles.

public ArrayList<ALIENA> get_aliena()
Devuelve el Array de naves alienA.

public ArrayList<ALIENB> get_alienb()
Devuelve el Array de naves alienB.

public void tecla_presionada(KeyEvent e)
Detecta la tecla pulsada y actualiza la dirección x o y correspondiente en la que se moverá la nave aliada o disparará un misil.

public void tecla_soltada(KeyEvent e)
Detecta cuando una tecla que ha sido pulsada, es liberada (soltada), y actualiza la dirección x o y en la que se moverá la nave aliada a 0, es decir, no se mueve si se suelta la tecla.

public void disparar()
Dispara un misil. Crea un nuevo misil, se le asigna su posición en pantalla en función de la posición de la nave aliada y se introduce en el Array de misiles.

public void crear_aliena(int num, int v)
Crea las naves alienA y la introduce en el Array de naves alienA.
La posición x de la nueva nave se pone al final de la pantalla, la posición y se calcula de forma aleatoria. Se pone un valor de retardo para la aparición de la nave en pantalla, de manera que hace que las naves aparezcan una detrás de otra en lugar de hacerlas todas a la vez.

public void crear_alienb(int num, int v)
Crea las naves alienB y las introduce en el Array de naves alienB.
La posición x de la nueva nave se pone al final de la pantalla, la posición y se calcula de forma aleatoria. Se pone un valor de retardo para la aparición de la nave en pantalla, de manera que hace que las naves aparezcan una detrás de otra en lugar de hacerlas todas a la vez.

public void borrar_arrays()
Borra los objetos de los Arrays de misiles, naves alienA y naves alienB.

CLASE MISIL.

public class MISIL
    private int x;
    private int y;
    private int velocidad;
    private int alto;
    private int ancho;
    private Image imagen_misil;
    boolean visible;

Implementa los misiles que dispara la nave aliada. Los misiles se desplazan de izquierda a derecha con velocidad constante y sin variar su trayectoria.
Las variables (X,Y) almacenan las coordenadas de la posición del misil en pantalla.
La variable velocidad, almacena la velocidad a la que se mueve el misil.
Las variables alto y ancho, almacenan el alto y el ancho de la imagen del misil.
La variable imagen_misil, almacena la imagen del misil.
La variable visible, indica si el misil es visible en pantalla, si no es visible es por que se ha salido de los limites de la pantalla.

METODOS DE LA CLASE MISIL.

public MISIL(int x, int y, int velocidad)

Constructor de la clase MISIL.
Se le pasan como parámetros la posición inicial del misil en pantalla (x,y) y la velocidad del misil.
Carga la imagen del misil, almacena su alto y ancho, lo hacemos visible.
Actualiza las coordenadas (x,y) y velocidad a la que se mueve.

public Image get_image_misil()

Devuelve la imagen del misil.

public int get_posx_misil()

Devuelve la coordenada x de la posición en pantalla del misil.

public int get_posy_misil()

Devuelve la coordenada y de la posición en pantalla del misil.

public boolean visible_misil()

Devuelve el valor visible del misil.

public int get_alto_misil()
Devuelve el alto de la imagen del misil.

public int get_ancho_misil()
Devuelve el ancho de la imagen del misil.

public void set_visible_misil(boolean v)
Actualiza el marcador visible del misil.

public void mover_misil()
Mueve el misil por la pantalla hacia la derecha. Comprueba si se ha salido de la pantalla, en cuyo caso actualiza la variable visible a falso para que el misil sea eliminado del array de misiles.

CLASE ALIENA.

public class ALIENA
```
private int x;
private int y;
private int velocidad;
private int retardo;
private int alto;
private int ancho;
private Image imagen_aliena;
boolean visible;
```
Clase que implementa las naves enemigas tipo A.
Las variables (x,y) almacenan las coordenadas de la posición de la nave alienA en pantalla.
La variable velocidad, almacena la velocidad a la que se moverá la nave en pantalla.
La variable retardo, se utiliza para realizar un retardo en la aparición de la nave en pantalla, de esta manera, el conjunto de naves alienA, aparecerán en pantalla una detrás de otra en lugar de hacerlo todas a la vez.
Las variables alto y ancho, almacenan el alto y el ancho de la imagen de la nave alienA.
La variable imagen_aliena almacena la imagen de la nave.
La variable visible, indica si la nave alienA es visible o no.

METODOS DE LA CLASE ALIENA.

public ALIENA(int x, int y, int velocidad, int retardo)
Constructor de la nave alienA. Se le pasan como parámetros la posición inicial en pantalla (x,y), la velocidad a la que se moverá en pantalla y el retardo que tiene la nave para aparecer en pantalla.
Almacena la imagen de la nave alienA y obtiene su alto y ancho.
Hace la nave visible.
Actualiza las coordenadas (x,y) de la posición de la nave en pantalla.
Actualiza la velocidad de la nave y el retardo.

public Image get_image_aliena()
Devuelve la imagen de la nave alienA.

public int get_posx_aliena()
Devuelve la coordenada x de la posición de la nave en pantalla.

public int get_posy_aliena()
Devuelve la coordenada y de la posición de la nave en pantalla.

public boolean visible_aliena()
Devuelve el valor visible de la nave alienA.

public int get_alto_aliena()
Devuelve el alto de la imagen de la nave alienA.

public int get_ancho_aliena()
Devuelve el ancho de la imagen de la nave alienA.

public void set_visible_aliena(boolean v)
Actualiza el indicador de nave visible.

public void mover_aliena()
Mueve la nave alienA por la pantalla, solo por el eje x de derecha a izquierda. Si la nave tiene retardo, no la muestra en pantalla y reduce el retardo, esta acción evita que al salir las naves en pantalla por primera vez se solapen y salgan todas a la vez.

CLASE ALIENB.

```
public class ALIENB {
   private int x;
   private int y;
   private int velocidad;
   private int retardo;
   private int direccion_ejey;
   private int movimiento_ejey;
   private int alto;
   private int ancho;
   private Image imagen_alienb;
   boolean visible;
```

Clase que implementa las naves enemigas tipo B.
Las variables (x,y) almacenan las coordenadas de la posición de la nave en pantalla.
La variable velocidad, almacena la velocidad a la que se moverá la nave en pantalla,
La variable retardo, almacena el retardo que tendrá la nave para aparecer en pantalla.
La variable direccion_ejey, indica la dirección en la que se moverá la nave en el eje y. +1 se moverá hacia abajo, -1 se moverá hacia arriba.
La variable movimiento_ejey, indica la distancia, aleatoria, que se moverá la nave en el eje y, antes de cambiar de dirección.
Las variables alto y ancho, almacenan el alto y el ancho de la imagen de la nave alienB.
La variable imagen_alienb, almacena la imagen de la nave alienB.
La variable visible, indica si la nave es visible en pantalla o no.

METODOS DE LA CLASE ALIENB.

```
   public ALIENB(int x, int y, int velocidad, int retardo)
```

Constructor de la clase ALIENB.
Se le pasa como parámetros la posición de la nave en pantalla (x,y)
la velocidad de la nave y el retardo, el tiempo que tarda en aparecer por primera vez en pantalla.
Almacena la imagen de la nave alienB y obtiene su alto y ancho.
Hace la nave visible.
Actualiza las coordenadas (x,y) de la posición de la nave en pantalla.
Actualiza la velocidad y el retardo de la nave.
Inicia la variable direccion_ejey a 1, indicando que inicialmente la nave se moverá hacia abajo en el eje y.
Inicia la variable movimiento_ejey a 0.

public Image get_image_alienb()
Devuelve la imagen de la nave alienB.

public int get_posx_alienb()
Devuelve la coordenada x de la posición de la nave en pantalla.

public int get_posy_alienb()
Devuelve la coordenada y de la posición de la nave en pantalla.

public boolean visible_alienb()
Devuelve el valor visible de la nave alienB.

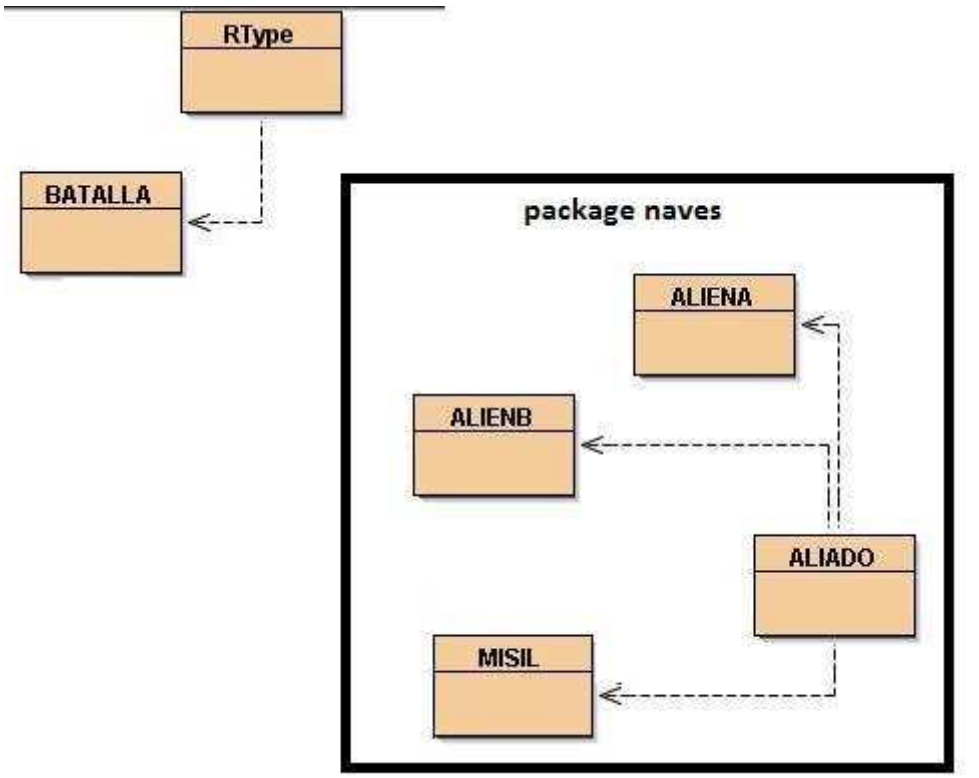
public int get_alto_alienb()
Devuelve el alto de la imagen de la nave alienB.

public int get_ancho_alienb()
Devuelve el ancho de la imagen de la nave alienB.

public void set_visible_alienb(boolean v)
Actualiza el indicador de nave visible.

public void mover_alienb()
Mueve la nave por la pantalla. Si la nave tiene retardo, espera y reduce el retardo. El movimiento sobre el eje y lo realiza en zig zag, pero con distancias de desplazamiento aleatorias, de esta forma, la nave se desplaza hacia arriba una distancia aleatoria, después hacia abajo, también una distancia aleatoria. Si la nave llega al limite inferior o superior de la pantalla y no ha completado la distancia, cambia de dirección.
Si la nave llega al final de la pantalla (izquierda), aparece de nuevo por la derecha.

DIAGRAMA DE CLASES.



Por un lado tenemos la clase principal RType y la clase BATALLA.
Por otro lado, esta el package naves, que esta compuesto por las clases ALIADO, MISIL, ALIENA, ALIENB.
En la figura anterior, se muestra la relación entre las distintas clases.
Al crear el package naves, se facilita la reutilización de código, ya que este paquete se podría utilizar para crear otro juego distinto.