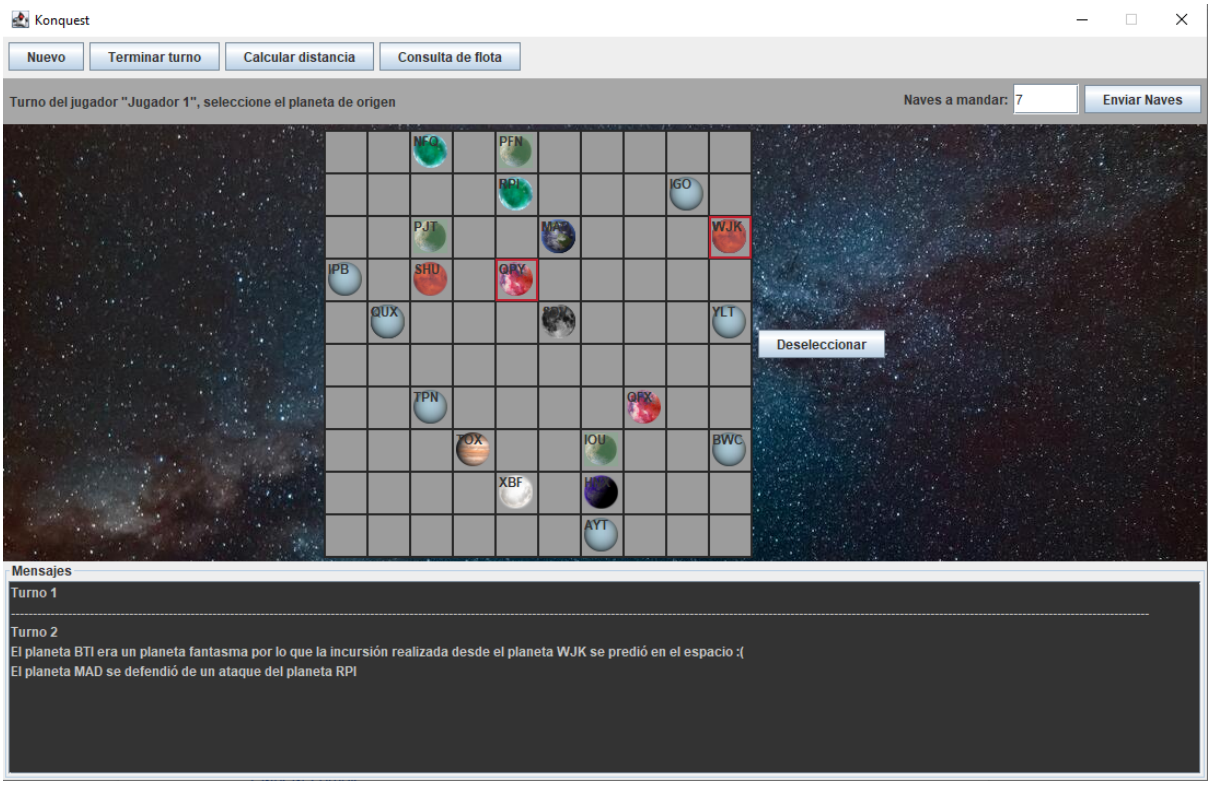


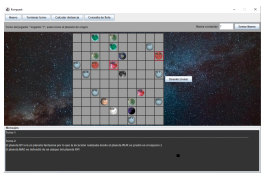
Konquest

Manual técnico

Versión: 0100

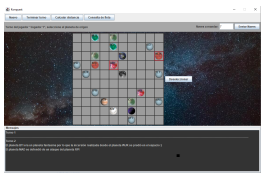
Fecha:17/05/2022

Desarrollador: Cristian Alejandro Vásquez Escobar, 202131936



Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

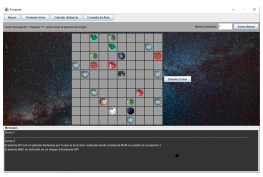


Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

Requerimientos para la ejecución del juego

- Una computadora
- Se necesita tener instalada la máquina virtual de java. Esto se puede hacer desde su página y para cualquier sistema operativo: Descargar Java
- Alguna terminal para poder interpretar comandos, si se encuentra en windows con el cmd o el windows powershell será suficiente.

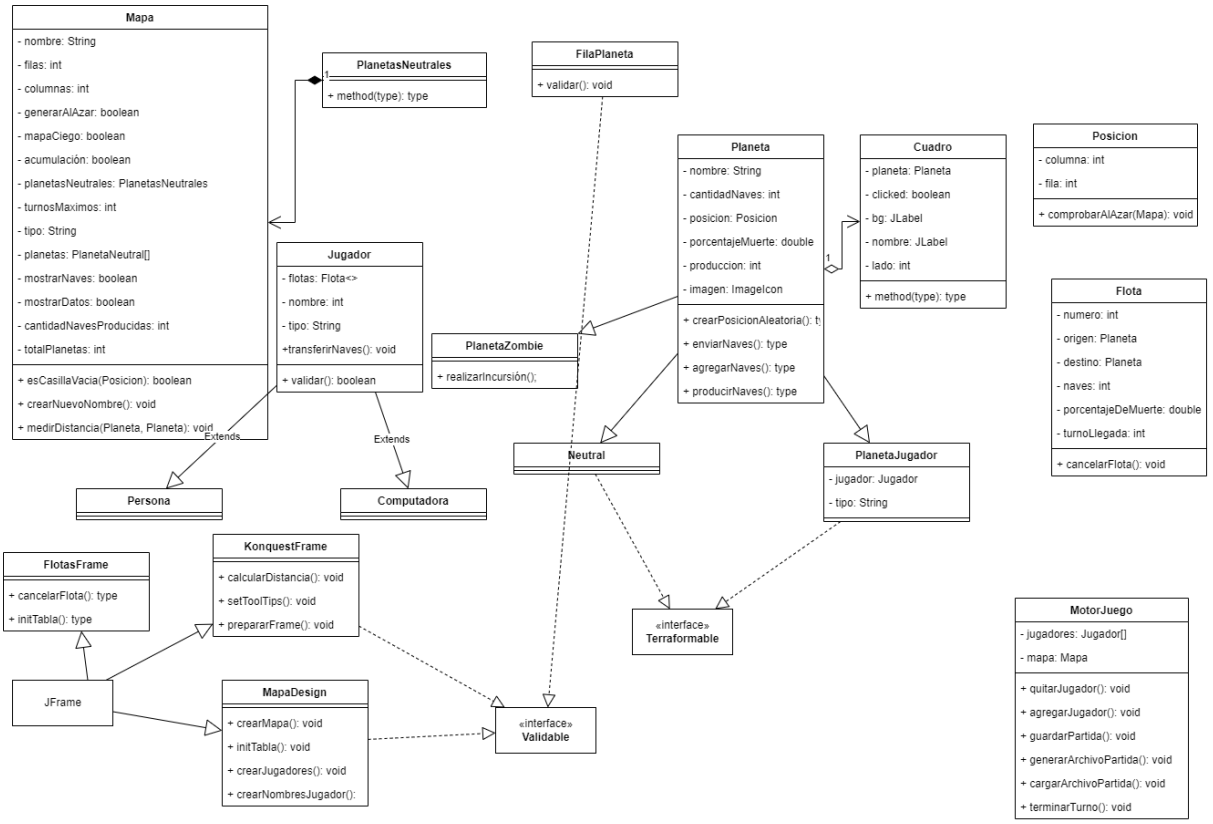


Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

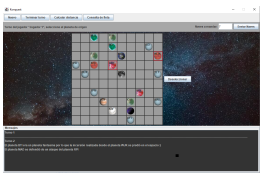
Elaboración del programa

Diagramas de clases UML



Herramientas

- Para la realización del programa “Mascota virtual” se utilizó el lenguaje de programación java en su versión 15
- El ambiente de desarrollo utilizado para el frontend es Apache Netbeans 12.6
- El ambiente de desarrollo utilizado para el backend es Intellij
- Fue realizado desde una laptop hp 15-db0 con windows 10.
- Para una mejor organización se utilizó el sistema de versiones git con github.
- Para el desarrollo se utilizó maven.
- Para la elaboración del archivo .jar se utilizó maven.



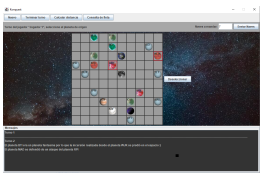
Clases

Se crearon diferentes tipos de clases y de paquetes

Flota

```java
public class Flota {

    private int numero;
    private int naves;
    private double porcentajeMuerte;
    private int turnoLlegada;
    private int turnoPartida;
    private Planeta origen;
    private Planeta destino;

    public Flota(int numero, int naves, double porcentajeMuerte, Planeta origen, Planeta destino, int turnoLlegada
        this.numero = numero;
        this.naves = naves;
        this.porcentajeMuerte = porcentajeMuerte;
        this.origen = origen;
        this.destino = destino;
        this.turnoLlegada = turnoLlegada;
        this.turnoPartida = turnoPartida;
    }

    public void aterrizar(Mapa mapa, KonquestFrame frame) throws ListaException {
        double porcentaje = Planeta.crearPorcentajeMuerteAleatorio();
        if (destino.isActivo()) {
            if (destino instanceof PlanetaFantasma) {
                ((PlanetaFantasma) destino).recibirIncursion( flota: this, mapa, frame);
            } else if (destino instanceof PlanetaJugador && origen instanceof PlanetaJugador && ((PlanetaJugador)
```

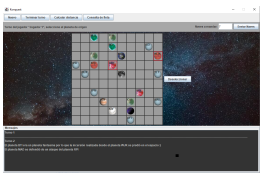
En esta clase se manejan las flotas, útiles para cuando ver cuándo llegan las inscursiones.



Mapa

```java
public class Mapa implements Serializable {

    private String nombre;
    private int filas;
    private int columnas;
    private boolean alAzar;
    private boolean mapaCiego;
    private boolean acumulativo;
    private int turnosMaximos;
    private String tipo;
    private int totalPlanetasNeutrales;
    private Lista<Planeta> planetas;
    private Lista<PlanetaNeutral> planetasNeutrales;
    private Lista<PlanetaJugador> planetasJugador;
    private boolean mostrarNavesNeutrales;
    private boolean mostrarEstadisticasPlanetasNeutrales;
    private int ProduccionPlanetasNeutrales;
    private Jugador[] jugadores;
    private int cantidadPlanetasFantasmas;
    private int cantidadPlanetasZombie;
    private int navesAtaqueZombie;
    private Lista<PlanetaFantasma> planetasFantasma;
    private Lista<PlanetaZombie> planetasZombie;
    private Cuadro[][] cuadros;

    public Mapa(String nombre, int filas, int columnas, boolean alAzar, boolean mapaCiego, boolean acumulativo, int
```

Aquí se manejaba cómo funcionaba el tablero del juego, contiene todas las opciones previamente preestablecidas y elementos como listas de planetas de cada clase.

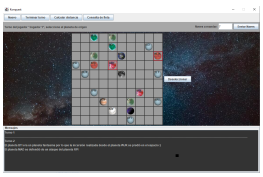


Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

```java
public boolean esPosicionOcupada(Posicion posicion) throws ListaException {
    for (int i = 0; i < planetas.obtenerLongitud(); i++) {
        Posicion posicionNodo = planetas.buscarIndice(i).getContenido().getPosicion();
        if (posicionNodo.esPosicionIgual(posicion)){
            return true;
        }
    }
    return false;
}

public void reiniciarListaPlanetas() {
    this.planetas = new Lista<Planeta>();
    this.planetasJugador = new Lista<PlanetaJugador>();
    this.planetasNeutrales = new Lista<PlanetaNeutral>();
    this.planetasZombie = new Lista<PlanetaZombie>();
    this.planetasFantasma = new Lista<PlanetaFantasma>();
}

public static int medirDistancia(Posicion pos1, Posicion pos2) {
    int columna1 = pos1.getColumna();
    int fila1 = pos1.getFila();
    int columna2 = pos2.getColumna();
    int fila2 = pos2.getFila();

    int distanciaX = columna1 - columna2;
    int distanciaY = fila1 - fila2;
```

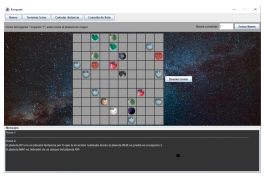
La medición de distancias se hizo en base al teorema de Pitágoras.



MotorJuego

```java
public class MotorJuego {

    private final Lista<Jugador> jugadores;
    private final Mapa mapa;
    private KonquestFrame frame;
    private Jugador jugadorActivo;
    private int indiceJugadorActivo;
    private int turno;
    private int turnoZombie;
    private Lista<Flota> flotasZombies;

    public MotorJuego(Jugador[] jugadores, Mapa mapa) {
        this.jugadores = new Lista<Jugador>();
        for (int i = 0; i < jugadores.length; i++) {
            this.jugadores.agregar(jugadores[i]);
        }
        this.mapa = mapa;
    }

    public Lista<Jugador> getJugadores() { return jugadores; }

    public Mapa getMapa() { return mapa; }

    public void empezarPartida(KonquestFrame frame) throws ListaException {
        this.frame = frame;
        this.indiceJugadorActivo = 0;
```

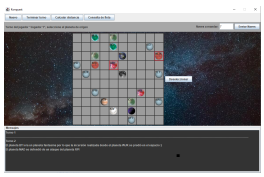
La clase motor juego se encarga de manejar el estado del juego, pues este contiene a los jugadores, al mapa, etc… cada cambio que pase en el juego pasa sobre esta clase.



Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

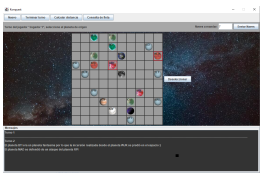
```java
public void cargarPlanetas(Lista planetas) throws ListaException {
    int longitud = planetas.obtenerLongitud();
    for (int i = 0; i < longitud; i++) {
        Planeta planeta = (Planeta) planetas.obtenerContenido(i);
        Posicion pos = planeta.getPosicion();
        int columna = pos.getColumna();
        int fila = pos.getFila();
        Cuadro planetaCuadro = frame.getCuadros()[columna][fila];
        planetaCuadro.setPlaneta(planeta);
        planetaCuadro.setIcon();
    }
}

public void nuevaFlota(Cuadro[] cuadrosClickeados, int naves) {
    Planeta origen = cuadrosClickeados[0].getPlaneta();
    Planeta destino = cuadrosClickeados[1].getPlaneta();
    jugadorActivo.agregarFlota(origen, destino, naves, turno);
}

public void aterrizarNaves() throws ListaException {
    for (int i = 0; i < jugadores.obtenerLongitud(); i++) {
        jugadores.obtenerContenido(i).aterrizarFlotas(turno, mapa, frame);
    }
    for (int i = 0; i < flotasZombies.obtenerLongitud(); i++) {
        Flota flota = flotasZombies.obtenerContenido(i);
        if (flota.getTurnoLlegada() == turno) {
```



Posición

```java
public class Posicion {

    private int columna;
    private int fila;

    public Posicion(int columna, int fila) {
        this.columna = columna;
        this.fila = fila;
    }

    public int getColumna() {
        return columna;
    }

    public int getFila() {
        return fila;
    }

    public boolean esPosicionFueraIndice(Mapa mapa) {
        return (mapa.getColumnas() < columna || mapa.getFilas() < fila || 0 > columna || 0 > fila);
    }
```

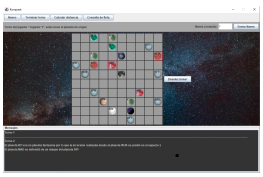
Clase hecha para simplificar la implementación de posiciones para ubicar ciertos aspectos en el tablero del juego.



Cuadro

```
private Planeta planeta;                                                    ⚠ 5
private boolean clicked;
private MotorJuego juego;
private JLabel bg;
private JLabel nombre;
private int ladoCuadro;

public final static Border LINEA_NEGRA = BorderFactory.createLineBorder(new Color( r: 40,  g: 40,  b: 40,
public final static Border LINEA_ROJA = BorderFactory.createLineBorder(new Color( r: 194,  g: 31,  b: 47,

public Cuadro(final MotorJuego juego) {
    this.juego = juego;
    clicked = false;
    setOpaque(false);
    setLayout(null);

    addMouseListener((MouseAdapter) mouseClicked(evt) → {
            KonquestFrame frame = juego.getFrame();
            if (planeta != null) {
                if (planeta instanceof PlanetaJugador && ((PlanetaJugador) planeta).getJugador().equals
                    if (frame.getIndiceCuadrosClickeados() == 0) {
                        seleccionar();
                        frame.getCuadrosClickeados()[0] = Cuadro.this;
                        frame.aumentarIndiceCuadrosClickeados();
                    } else if (frame.getIndiceCuadrosClickeados() == 1 && !frame.getCuadrosClickeados()
                        seleccionar();
```

Una subclase de JPanel, hecha para poder desplegar visualmente a cada planeta en el tablero, conteniendo también un label para poder visualizar el nombre del planeta. Además maneja mouselistener para el sistema de envío de flotas.

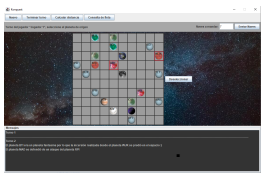


FilaPlaneta

```
public class FilaPlaneta implements Validable {

    protected int numeroFila;
    protected String nombre;
    protected int cantidadNaves;
    protected int producción;
    protected double porcentajeMuertes;
    protected Posicion posicion;
    protected JFrame parent;
    protected Mapa mapa;

    public FilaPlaneta(int numeroFila, String nombre, int cantidadNaves, int producción, double porcentajeMuert
        this.numeroFila = numeroFila;
        this.nombre = nombre;
        this.cantidadNaves = cantidadNaves;
        this.producción = producción;
        this.porcentajeMuertes = porcentajeMuertes;
        this.posicion = new Posicion(columnaPosicion, filaPosicion);
        this.parent = parent;
        this.mapa = mapa;
    }


    public String getNombre() { return nombre; }
```

Una clase creada para poder validar los datos de un planeta metidos dentro de una tabla para así después poder crear al planeta

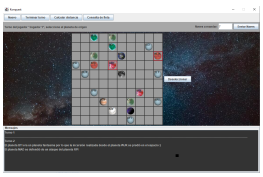
```
public class FilaPlanetaJugador extends FilaPlaneta {

    private String conquistador;
    private String tipo;

    public FilaPlanetaJugador(int numeroFila, String nombre, int cantidadNaves, int producción, double porcent
        super(numeroFila, nombre, cantidadNaves, producción, porcentajeMuertes, columnaPosicion, filaPosicion,
        this.conquistador = conquistador;
        this.tipo = tipo;
    }

    @Override
    public void validar() throws ValidacionesException, ListaException {
        //entero entre 0 y 0.999999
        if (porcentajeMuertes < 0 || porcentajeMuertes > 0.999999) {
            throw new ValidacionesException("Error de la tabla de jugadores, en la fila"+ numeroFila + "El por
```



Planeta

```
public abstract class Planeta {

    protected String nombre;
    protected int cantidadNaves;
    protected Posicion posicion;
    protected double porcentajeMuerte;
    protected int produccion;
    protected ImageIcon imagen;
    protected Cuadro cuadro;
    protected boolean activo;

    public Planeta(String nombre, int cantidadNaves, Posicion posicion, double porcentajeMuerte, int produccion)
        this.nombre = nombre;
        this.cantidadNaves = cantidadNaves;
        this.posicion = posicion;
        this.porcentajeMuerte = porcentajeMuerte;
        this.produccion = produccion;
        activo = true;
    }

    public void setCuadro(Cuadro cuadro) { this.cuadro = cuadro; }
```

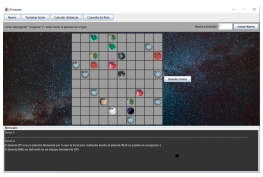
Clase abstracta que maneja a cada planeta, con atributos que le sirven al programa para la interacción y el cambio del estado.



Planeta Jugador

Subclase de planeta con la cualidad que esta tiene como miembro a un jugador y la implementación del recibo de incursiones.

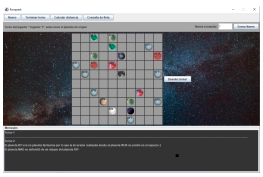
```
public class PlanetaJugador extends Planeta implements Terraformable, RecibirIncursion {

    private Jugador jugador;
    private String tipo;

    public PlanetaJugador(String nombre, int cantidadNaves, Posicion posicion, double porcentajeMuerte, int prod
        super(nombre, cantidadNaves, posicion, porcentajeMuerte, produccion);
        this.jugador = jugador;
        this.tipo = tipo;
        this.imagen = new ImageIcon(getClass().getResource( name: "/imagenes/planetas/" + tipo.toUpperCase() + ".
    }

    @Override
    public void recibirIncursion(Flota flota, Mapa mapa, KonquestFrame frame) throws ListaException {
        PlanetaJugador origen = (PlanetaJugador) flota.getOrigen();
        PlanetaJugador planeta = terraformar(flota);
        Cuadro cuadro = getCuadro();
        jugador.getPlanetas().eliminarContenido(this); //eliminarle el planeta al jugador destino
        origen.getJugador().getPlanetas().agregar(planeta);// agregarle el planeta al jugador origen
        activo = false;

        mapa.getPlanetasJugador().cambiarContenido( contenidoARemplazar: this, planeta);
        cuadro.setPlaneta(planeta);
        cuadro.actualizar();
```



Planeta Neutral

```java
public class PlanetaNeutral extends Planeta implements Terraformable, RecibirIncursion {

    public PlanetaNeutral(String nombre, int cantidadNaves, Posicion posicion, double porcentajeMuerte, int produc
        super(nombre, cantidadNaves, posicion, porcentajeMuerte, produccion);
        imagen = new ImageIcon(getClass().getResource( name: "/imagenes/planetas/NEUTRAL.png"));
    }


    public String toString(boolean[] valores) {
        /*mostrar naves = 0
        mostrar estadisticas = 1*/
        if (!valores[0] && valores[1])
            return "nombre: " + nombre + ", porcentaje de muerte: " + porcentajeMuerte + ", producción: " + produc
        if (valores[0] && valores[1])
            return "nombre: " + nombre + ", naves: " + cantidadNaves + ", porcentaje de muerte: " + porcentajeMuer
        if (valores[0] && !valores[1])
            return "nombre: " + nombre + ", naves: " + cantidadNaves;
        return "nombre: " + nombre;
    }


    @Override
    public void recibirIncursion(Flota flota, Mapa mapa, KonquestFrame frame) throws ListaException {
        PlanetaJugador origen = (PlanetaJugador) flota.getOrigen();
```

Muy parecido a su padre Planeta, con la diferencia de que implementa el recibirIncursiones a su manera.



Planeta Zombie

```java
public class PlanetaZombie extends Planeta {

    public PlanetaZombie(String nombre, int cantidadNaves, Mapa mapa) throws ListaException {
        super(nombre, cantidadNaves, crearPosiciónAleatoria(mapa), crearPorcentajeMuerteAleatorio(), crearProducci
        imagen = new ImageIcon(Objects.requireNonNull(getClass().getResource( name: "/imagenes/planetas/ZOMBIE.png"
        //TODO
    }

    @Override
    public void producirNaves(boolean esAcumulable) {

    }

    public void realizarIncursion(Mapa mapa, Lista<Flota> flotasZombie, int turno) throws ListaException {
        Cuadro[][] cuadros = mapa.getCuadros();

        int columna = (int) Math.floor(Math.random() * ((mapa.getColumnas() - 1) - 0 + 1) + 0);
        int fila = (int) Math.floor(Math.random() * ((mapa.getFilas() - 1) - 0 + 1) + 0);
        Posicion posicionAleatoria = new Posicion(columna, fila);
        while (!mapa.esPosicionOcupada(posicionAleatoria) || cuadros[columna][fila].getPlaneta() instanceof Planet
            columna = (int) Math.floor(Math.random() * ((mapa.getColumnas() - 1) - 0 + 1) + 0);
            fila = (int) Math.floor(Math.random() * ((mapa.getFilas() - 1) - 0 + 1) + 0);
            posicionAleatoria = new Posicion(columna, fila);
```

Subclase de Planeta, con la diferencia de que tiene un método en el que realiza sus incursiones a un planeta aleatorio.

Planeta fantasma

```java
public class PlanetaFantasma extends Planeta implements RecibirIncursion {

    public PlanetaFantasma(String nombre, Mapa mapa) throws ListaException {
        super(nombre , crearCantidadDeNavesAleatoria(), crearPosiciónAleatoria(mapa), crearPorcentajeMuert
        imagen = new ImageIcon(getClass().getResource( name: "/imagenes/planetas/NEUTRAL.png"));
    }

    public String toString(boolean[] valores){
        /*mostrar naves = 0
        mostrar estadisticas = 1*/
        if(!valores[0] && valores[1])
            return  "nombre: " + nombre + ", porcentaje de muerte: " + porcentajeMuerte + ", producción: "
        if(valores[0] && valores[1])
            return  "nombre: " + nombre + ", naves: " + cantidadNaves + ", porcentaje de muerte: " + porce
        if(valores[0] && !valores[1])
            return  "nombre: " + nombre + ", naves: " + cantidadNaves;
        return "nombre: " + nombre;
    }


    @Override
    public void recibirIncursion(Flota flota, Mapa mapa, KonquestFrame frame) throws ListaException {

        Planeta origen = flota.getOrigen();
```