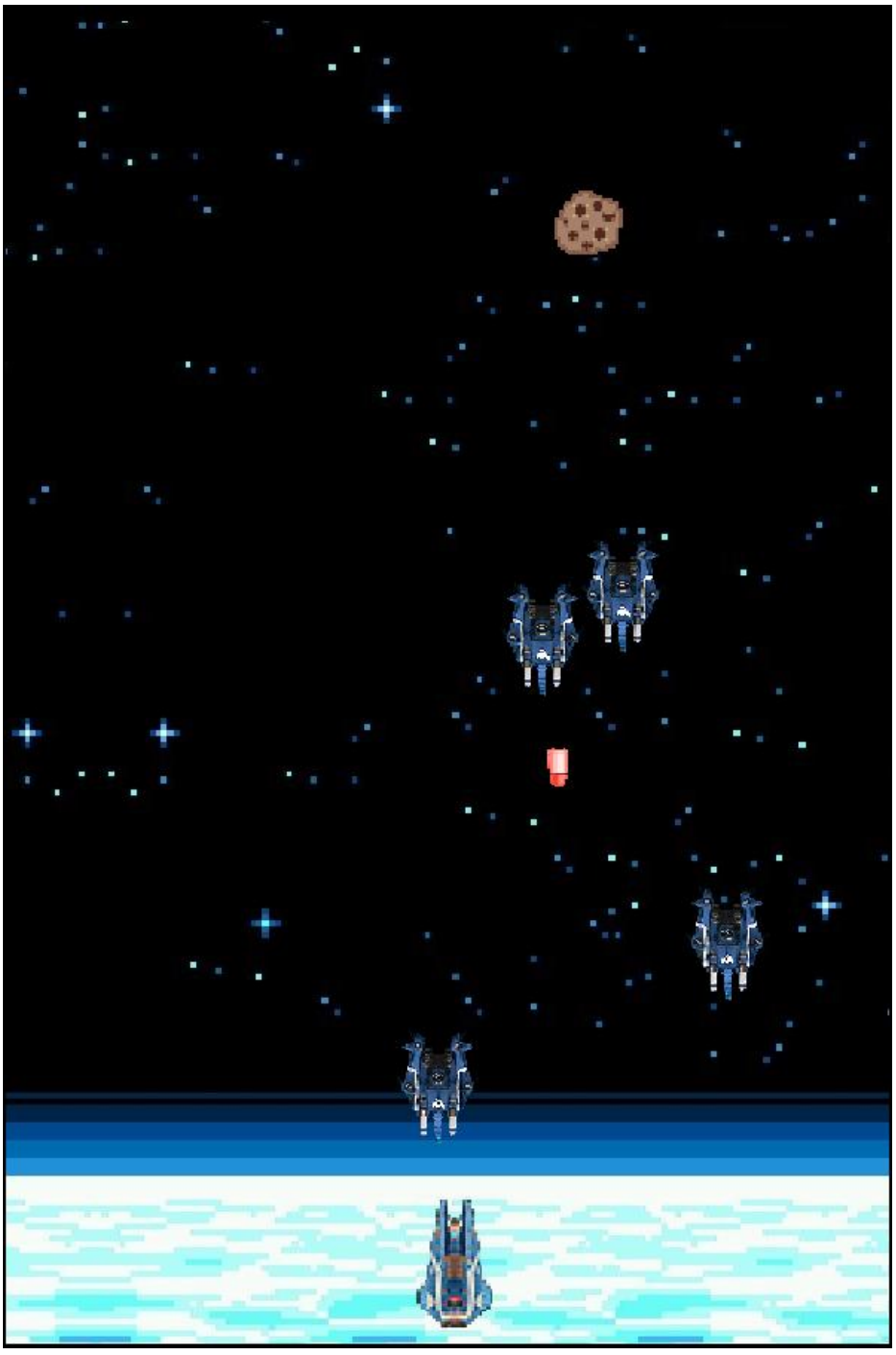
Programación I - Primer Semestre 2023
Trabajo Práctico: Lost Galaxian.



Nombres:

- DE LAURETIS, Nicolas
- Solis Hernan
- Sosa Lucas Ariel
- Stafforini Manuel

Mail:

- nicolasdelauretis122@gmail.com
- hernansolis94k@gmail.com
- gopfle@gmail.com
- manustafforini@gmail.com

Introducción:

La Astro-MegaShip en un intento de defender lo que queda de la raza humana después de un ataque interestelar a la tierra. Nuestra nave vaga por el espacio eliminando a los Destructores estelares, culpables del ataque a la tierra, mientras esquiva asteroides para poder recuperar la paz en nuestro universo.

Equipamos a la Astro-MegaShip con una coraza superresistente que aguanta más de un golpe de un Ion de los destructores, o colisiones con asteroides, pero no es irrompible. Podemos recuperar durabilidad con los Corazones de Destructores que flotan como basura espacial.

Descripción:

Asteroides:

La clase "Asteroides" se utiliza para representar y controlar el comportamiento de los asteroides en el juego. Proporciona métodos para mover, dibujar, cambiar la trayectoria, acelerar y detectar colisiones con una nave espacial y otros asteroides. El problema principal fue establecer hitbox congruente con la imagen del asteroide.

```
import java.awt.Image;
import
java.awt.Rectangle;
import
entorno.Entorno;
import
entorno.Herramientas;
```

Importa las clases necesarias para trabajar con imágenes, rectángulos y el entorno del juego.

```
public class Asteroides {
        private double x;
        private double y;
        private double velocidad;
        private double angulo;
        private int radio;

        Image imagenAst =
Herramientas.cargarImagen("Asteroid.gif");
```

La clase Asteroides tiene variables de instancia privadas “x”, “y”, “velocidad”, “ángulo” y “radio”. Representan la posición, la velocidad, el ángulo y el radio del asteroide. A su vez tiene un objeto Image, llamado “imagenAst” para guardar la imagen del asteroide.

```
public Asteroides (double x, double y, double velocidad,
                                double angulo, int radio)
 {
        this.x = x;
        this.y = y;
        this.velocidad = velocidad;
        this.angulo = angulo;
        this.setRadio(radio);
}
```

Este constructor inicializa las propiedades del asteroide “x”, “y”, “velocidad” , “ángulo” y “radio”.

```
public void mover(int mod) {
        y += velocidad * Math.sin(angulo);
        x += velocidad * Math.cos(angulo) * mod;
}
```

El método “mover” actualiza la posición del asteroide según su velocidad y angulo. El parámetro mod se utiliza para controlar la dirección del movimiento.

```
public void cambiarTrayectoria() {
        angulo += Math.PI / 2;
}
```

El método “cambiarTrayectoria” cambie el ángulo del asteroide sumándole π/2 (90 grados) a su ángulo actual.

```
public void acelerar() {
        velocidad += 0.05;
}
```

El método “acelerar” aumenta la velocidad del asteroide.

```
public void dibujarse(Entorno entorno) {
        entorno.dibujarImagen(imagenAst, this.x, this.y, -1.55, 0.05);
}
```

El método “dibujarse” se utiliza para dibujar el asteroide en el entorno del juego (Entorno). Utiliza el método dibujarImagen para mostrar la imagen del asteroide en la posición especificada (x, y).

```
public Rectangle asteroideHitbox() {
        return new Rectangle((int) this.x, (int) this.y, 54, 54);
}
```

El método “asteroideHitbox” devuelve un objeto Rectangle que representa el cuadro de colisión o límite del asteroide. Se utiliza para la chequear si hay colisiones.

```
public boolean colision(Spaceship navecita,
Asteroides[]asteroidesArr){
        for (int i = 0; i < asteroidesArr.length; i++) {
        if (asteroidesArr[i] != null) {
        if(navecita.navecitaHitbox().intersects(asteroidesArr[i].astero
ideHitbox()))
                asteroidesArr[i] = null;
```

El método “colision” comprueba si hay una colisión entre la nave espacial (navecita) y los asteroides en el arreglo asteroidesArr. Recorre todos los asteroides en el arreglo y, si un asteroide no es nulo, verifica si la hitbox de la nave espacial (navecitaHitbox()) se interseca con la hitbox del asteroide (asteroideHitbox()). Si hay una colisión, se establece el asteroide en null, se reduce en 1 la cantidad de vidas de la nave espacial y se retorna true. Si no hay colisión con ningún asteroide, el método retorna false.

Corazones:

La clase Corazones se utiliza para representar objetos de corazones en el juego. Los métodos permiten obtener y establecer las coordenadas del corazón, actualizar su posición, dibujarlo en el entorno de juego y obtener el área de colisión del corazón.

```
public class Asteroides {
double x;

double y;

double velocidad;

double angulo;

int radio;

}
```

La clase corazones tiene variables de instancia “x” representa la coordenada x del corazón, “y” la coordenada y del corazón, “velocidad” el movimiento, “img” se usa para cargar la imagen del corazón.

```
public Corazones(int startX, int startY) {
	this.x = startX;
	this.y = startY;
}
```

Este constructor recibe las coordenadas iniciales “startX”, “startY”. Y las asigna a las variables de instancia “x” e “y”.

```
public double getX() {
	return x;
}
```

El método “getX”, devuelve la coordenada actual del corazón en “x”.

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

El método “setX”, establece la coordenada “x” del corazón.

```
public double getY() {
	return y;
	}
```

El método “getY”, devuelve la coordenada actual del corazón en “y”.

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

El método “setY”, establece la coordenada “y” del corazón.

```
public void mover() {
	y += velocidad;
	}
```

Actualiza la posición del corazón moviéndolo hacia abajo según la velocidad establecida.

```
public void dibujarCorazones(Entorno e) {
	img = Herramientas.cargarImagen("corazon.png");
		e.dibujarImagen(img, x, y, 0, 1);


	}
```

El método “dibujarCorazones”, toma como parámetro el objeto “Entorno” y dibuja la imagen del corazón en la posición actual “x”, “y”.

```
public Rectangle corazonHitbox() {
	return new Rectangle(x, y, 13, 26);
}
```

El método “corazonHitbox”, retorna un objeto Rectangle que representa el área de colisión del corazón. El rectángulo se crea en la posición actual “x”, “y”. Tiene un ancho de 13 y una altura de 26.

Destructores:

La clase “Destructores” se utiliza para representar y controlar el comportamiento de los destructores en el juego. Tiene los métodos para mover, dibujar, verificar colisiones con una nave espacial y otros destructores, ajustar posiciones en caso de superposición y actualizar la posición de los proyectiles disparados por los destructores.

```
public class Destructores {
	private double x;
	private double y;
	private double velocidad;
	private int movimiento = 0;
	Image imagenDest = Herramientas.cargarImagen("destructor.png");
}
```

Las variables de instancia de la clase “Destructores” son las necesarias para describir el movimiento de los mismo, utilizando “x” para las coordenadas en x, “y” para las coordenadas en y, “velocidad” representa la velocidad del movimiento del destructor, “movimiento” alude al estado del movimiento e “imgenDest” guarda la imagen del destructor.

```
public Destructores(double x, double y, double velocidad) {
	this.x = x;
	this.y = y;
	this.velocidad = velocidad;
}
```

En este constructor se asignan los valores a las variables, recibe las coordenadas iniciales “x”, “y” y la velocidad.

```
public double getX() {
	return x;
}
```

Devuelve la coordenada “x” actual del destructor.

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

Establece la coordenada “x” del destructor.

```
public double getY() {
	return y;
}
```

Devuelve la coordenada “y” actual del destructor.

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

Establece la coordenada “y” del destructor.

```
public double getVelocidad() {
	return velocidad;
	}
```

Devuelve la velocidad actual del destructor.

```
public void setVelocidad(double velocidad) {
	this.velocidad = velocidad;
}
```

Establece la velocidad del destructor.

```
public int getMovimiento() {
	return movimiento;
	}
```

Devuelve el estado de movimiento actual del destructor.

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

Configura el movimiento, actualizando el estado de movimiento del destructor.

```
public void mover(double mod) {
	if (movimiento == 50) {
		movimiento = -50;
}
if (movimiento < 50 && movimiento >= 0) {
	x = x + mod;
	movimiento += 1;
}

if (movimiento < 0 && movimiento >= -50) {
	x = x - mod;
	movimiento += 1;
}

	y += velocidad;
}
```

Condición de movimiento en la coordenada "x" en rango -50 a 50 para generar movimiento de zigzag.

Condición de movimiento que indica al destructor hacia donde moverse si esta dentro del rango.

Condición de movimiento que indica al destructor hacia donde moverse si esta dentro del rango.

El destructor siempre se desplaza a una determinada velocidad en el eje "y".

```
public void acelerar() {
	velocidad += 0.05;
}
```

El método “acelerar” aumenta la velocidad.

```
public void dibujarse(Entorno entorno) {
	entorno.dibujarImagen(imagenDest, this.x, this.y, 0, 0.4);
}
```

Dibuja la imagen del destructor en la posición actual (x, y) utilizando el objeto Entorno proporcionado.

```
public Rectangle destructoresHitbox() {
	return new Rectangle((int) this.x, (int) this.y, 40, 60);
}
```

El método “destructoresHitbox” devuelve un Rectangle que representa el área de colision del destructor, un ancho “x” igual a 40 y un largo “y” igual a 60.

```
public boolean colision(Spaceship navecita, Destructores[]
destructoresArr) {

      for (int i = 0; i < destructoresArr.length; i++) {

            if (destructoresArr[i] != null) {

if(navecita.navecitaHitbox().intersects(destructoresArr[i].destru
ctoresHitbox()){

      int navecitaVidas = navecita.getVidas();

      destructoresArr[i] = null;

      navecitaVidas -= 1;

      navecita.setVidas(navecitaVidas);

      return true;

            }

      }

}

      return false;
```

Comprueba si hay colisión entre el destructor y una nave espacial (navecita). Recibe un arreglo de destructores (destructoresArr) para verificar la colisión con otros destructores. Si hay una colisión, resta una vida a la nave espacial, elimina el destructor del array y devuelve true. Si no hay colisión, devuelve false.

```
public void superponenDest(Destructores[] destructoresArr) {

	for (int i = 0; i < destructoresArr.length; i++) {

		for (int j = 1; j < destructoresArr.length; j++) {

				if (destructoresArr[i] != null &&
				destructoresArr[j] != null &&
				destructoresArr[i] != destructoresArr[j]) {

					if (j >= destructoresArr.length) {

						j = 0;

					}

	if
(destructoresArr[i].destructoresHitbox().intersects(destructoresArr
[j].destructoresHitbox())) {
	destructoresArr[j].setX(destructoresArr[i].getX() - 70);



				}

		}

	}
```

Comprueba si hay superposición entre los destructores. Recibe un arreglo de destructores como parámetro y ajusta la posición de los destructores si se superponen entre sí.

```
public void superponenAst(Destructores[] destructoresArr,
Asteroides[]asteroidesArr) {

      for (int i = 0; i < destructoresArr.length; i++) {

            for (int j = 0; j < asteroidesArr.length; j++) {

                  if (destructoresArr[i] == null || asteroidesArr[j]
== null) {

                              continue;

      } else {if
(destructoresArr[i].destructoresHitbox().intersects(asteroidesArr[j].
asteroideHitbox())) {


      destructoresArr[i].setY(destructoresArr[i].getY() - 30);

                                    }

                              }

                        }

                  }
```

Verifica si hay superposición entre los destructores y los asteroides. Recibe arreglos de destructores y asteroides como parámetros y ajusta la posición de los destructores si se superponen con asteroides

```
public void destruccion(Destructores[] destructoresArr, Proyectil[]
proyectilesArr, Spaceship navecita) {

      for (int i = 0; i < destructoresArr.length; i++) {

            for (int j = 0; j < proyectilesArr.length; j++) {

                  if (destructoresArr[i] == null ||
proyectilesArr[j] == null) {

                              continue;

            } else {if
(destructoresArr[i].destructoresHitbox().intersects(proyectilesArr[j
].proyectilHitbox())) {

                  navecita.setPuedeDisparar(true);

                  navecita.setPuntaje((int) navecita.getPuntaje() +
1);

                  destructoresArr[i] = null;

                  proyectilesArr[0] = null;

                  // DECLARO E INICIALIZO SONIDO DE DESTRUCCION

File file5 = new File("src/explocion.wav");

AudioInputStream audioStream

=AudioSystem.getAudioInputStream(file5);

Clip clipExplocion = AudioSystem.getClip();

clipExplocion.open(audioStream5);

clipExplocion.start();

                  }

            }

      }
```

Comprueba si hay colisión entre los destructores y los proyectiles disparados por la nave. Recibe arreglos de destructores y otro de proyectiles, a su vez recibe un objeto de la clase Spaceship. Si hay colisión, aumenta el puntaje de la nave espacial, elimina el destructor y el proyectil, y habilita el disparo de la nave espacial.

```
public void disparo(Destructores destructor, Iones ion) {
	if (destructor != null && ion != null) {
		ion.setX((int) destructor.getX());
		ion.setY((int) destructor.getY());
	}
}
```

Recibe como parámetro dos objetos “destructor” de la clase Destructores e “ion” de la clase Iones. El método verifica estos sean distintos de null. Y luego asigna las coordenadas “x” del ion en las del destructor, realiza el mismo procedimiento para las coordenadas “y”.

```
public void fueraDePantalla(Destructores[] destructoresArr, Entorno entorno) {
	for (Destructores d : destructoresArr) {
		if (d != null) {
			if (d.getY() > entorno.alto()) {
				d.setY(10);

			}
		}
	}
}
```

Verifica si los destructores están fuera de la pantalla en la coordenada y, y si es así, los reposiciona en la parte superior de la pantalla.

Fondo:

Esta clase se encargar de cargar el fondo del juego en la posición necesaria, el fondo consiste en imágenes cargadas.

```
public class fondo {
	Entorno entorno;
	Juego juego;
	Image fondo, fondo2, suelo;
}
```

Las variables de instancia de esta clase son un objeto “entorno” de la clase Entorno, un objeto “juego” de la clase Juego y un objeto “fondo” de la clase Image la que almacena tres imágenes.

```
public fondo(Entorno e, Juego juego) {
	dibujar(e);
}
```

Este constructor inicializa los atributos de los objetos que son dados como parámetro.

```
public void dibujar(Entorno e) {
	fondo = Herramientas.cargarImagen("fondo.jpg");
	e.dibujarImagen(fondo, 0, 500, 0, 1);
	}
}
```

El método “dibujar” recibe como parámetro un objeto de la clase entorno para dibujar el fondo, carga una imagen de fondo desde un archivo utilizando la clase “Herramientas”. Luego se utiliza el método “dibujarImagen” de la clase Entorno que toma como parámetro la posición (0,500), el angulo de rotación (0) y la escala (1).

Iones:

Esta clase representa los iones del juego. Tiene métodos para mover el ion, dibujarlo en el entorno gráfico, verificar colisiones con un rectángulo de hitbox y comprobar si está fuera de la pantalla. Surgieron problemas a la hora de crear este objeto debido a que quisimos utilizar la misma clase para Iones, pero al destruirse el Destructor este borraba las coordenadas “x”, “y” y no generaba coordenadas para Ion, por lo tanto, hubo que generar una clase para Proyectil e Iones.

```
public class Iones {
  private int x;
  private int y;
  private double velocidad = 5;
  Image img;
}
```

Las variables de instancia “x”, “y” representan la posición en dichas coordenadas respectivamente. “velocidad” se encarga de establecer la velocidad, “img” guarda la imagen del ion.

```
public Iones(int startX, int startY) {
    this.x = startX;
    this.y = startY;
}
```

El constructor asigna los valores iniciales para las coordenadas en “x”, “y”.

```
public double getX() {
    return x;
  }
```

Devuelve la posición horizontal “x” del ion.

```
public double setX() {
    this.x=x;
  }
```

Establece la posición horizontal “x” del ion según el valor proporcionado.

```
public double getY() {
    return y;
}
```

Devuelve la posición vertical “y” del ion.

```
public double setY() {
    this.y=y;
}
```

Establece la posición vertical “y” del ion según el valor proporcionado.

```
public void mover() {
      y += velocidad;
}
```

Actualiza la posición vertical “y” del ion al incrementarla según la velocidad de movimiento.

```
public void dibujarIones(Entorno e) {
      img = Herramientas.cargarImagen("ion.png");
      e.dibujarImagen(img, x, y, 0, 0.10);
  }
```

Dibuja la imagen del ion en el juego utilizando la clase Herramientas y el método dibujarImagen() del objeto pasado como parámetro que tiene atributos de la clase Entorno.

```
public Rectangle ionHitbox() {
    return new Rectangle(x, y, 13, 26);
}
```

Devuelve un objeto de tipo Rectangle que representa el área rectangular de colisión del ion. El rectángulo se crea con las coordenadas x e y del ion y un ancho de 13 y alto de 26.

```
  public Iones fueraDePantalla(Iones Iones) {
    if ((int) Iones.getY() >= 1600) {
      Iones = null;
    }
    return Iones;
  }
}
```

Comprueba si el ion está fuera de la pantalla según su posición vertical “y”. Si el ion ha alcanzado una posición mayor o igual a 1600, se establece como null. El método devuelve el ion.

Proyectil:

la clase Proyectil representa un proyectil en el juego. Tiene métodos para mover el proyectil, dibujarlo en el entorno gráfico y verificar colisiones con un rectángulo de hitbox.

```
public class Proyectil {
	private int x;
	private int y;
	private double velocidad = 5;
	Image img;
}
```

Las variables de la clase Proyectil representan “x” para la posición en la coordenada x, “y” para la coordenada y. “velocidad” establece la velocidad e “img” guarda la imagen.

```
public Proyectil(int startX, int startY) {
		this.x = startX;
		this.y = startY;
	}
```

El constructor asigna los valores iniciales del proyectil, inicializando las variables de instancia “x”, “y” en los parámetros dados al constructor.

```
public double getX() {
    return x;
  }
```

Devuelve la posición horizontal “x” del proyectil.

```
public double setX() {
    this.x=x;
  }
```

Establece la posición horizontal “x” del proyectil según el valor proporcionado.

```
public double getY() {
    return y;
}
```

Devuelve la posición vertical “y” del proyectil.

```
public double setY() {
    this.y=y;
}
```

Establece la posición vertical “y” del proyectil según el valor proporcionado.

```
public double getVelocidad() {
            return velocidad;
      }
```

Devuelve la velocidad de movimiento del proyectil.

```
public void setVelocidad(double velocidad) {
      this.velocidad = velocidad;
      }
```

Establece la velocidad de movimiento del proyectil.

```
public void mover() {
      y -= velocidad;
}
```

Actualiza la posición vertical “y” del proyectil al decrementarla según la velocidad de movimiento.

```
public void dibujarProyectil(Entorno e) {
            img = Herramientas.cargarImagen("proyectil.png");
            e.dibujarImagen(img, x, y, 0, 0.10);


      }
```

Dibuja la imagen del proyectil en el juego utilizando la clase Herramientas y el método dibujarImagen() del objeto pasado como parámetro que tiene atributos de la clase Entorno.

```
public Rectangle proyectilHitbox() {
      return new Rectangle(x, y, 10, 10);
      }
}
```

Devuelve un objeto de tipo Rectangle que representa el área rectangular de colisión del proyectil. El rectángulo se crea con las coordenadas “x” e “y” del proyectil y un ancho y alto de 10.

RayoVeloz:

La clase RayoVeloz representa un rayo en el juego. Tiene métodos para mover el rayo, dibujarlo en el entorno gráfico y verificar colisiones con un rectángulo de hitbox.

```
public class RayoVeloz {


  private int x;
  private int y;
  private double velocidad = 1;
  Image img;
}
```

Las varibales de instancia de la clase RayoVeloz, representan su posición en “x” e “y”, a través de coordenadas, “velocidad” indica la velocidad con la que se mueve, “img” almacena la imagen.

```
public RayoVeloz(int startX, int startY) {
    this.x = startX;
    this.y = startY;
  }
```

El constructor recibe las coordenadas iniciales startX y startY del rayo y las asigna a los atributos correspondientes “x” e “y”.

```
public double getX() {
    return x;
  }
```

Devuelve la posición horizontal “x” del rayo.

```
public double setX() {
    this.x=x;
  }
```

Establece la posición horizontal “x” del rayo según el valor proporcionado.

```
public double getY() {
    return y;
}
```

Devuelve la posición vertical “y” del rayo.

```
public double setY() {
    this.y=y;
}
```

Establece la posición vertical “y” del rayo según el valor proporcionado.

```
public void mover() {
      y += velocidad;
}
```

Actualiza la posición vertical “y” del ion al incrementarla según la velocidad.

```
  public void dibujarRayos(Entorno e) {
    img = Herramientas.cargarImagen("rayo.png");
    e.dibujarImagen(img, x, y, 0, 0.05);
  }
```

Dibuja la imagen del rayo utilizando la clase Herramientas y el método dibujarImagen() del objeto pasado como parámetro que tiene atributos de la clase Entorno.

```
public Rectangle rayoHitbox() {
    return new Rectangle(x, y, 13, 26);
  }
```

Devuelve un objeto de tipo Rectangle que representa el área rectangular de colisión del rayo. El rectángulo se crea con las coordenadas “x” e “y” del rayo y un ancho y alto de 13 y 26 respectivamente.

Spaceship:

La clase Spaceship se encarga de representar la nave espacial en el juego, controlar su movimiento, disparar proyectiles, manejar las vidas y puntuación, y verificar colisiones con otros objetos en el juego. En esta clase surgieron problemas con el hitbox ya que surgieron problemas para que la configuración utilizada sea fiel al objeto ya que requería manejar las escalas en pixeles por lo que se tuvo q hacer de forma manual.

```
public class Spaceship {
      private double x;
      private double y;
      private double velocidad;
      private int vidas;
      private boolean puedeDisparar = true;
      private int puntaje;
      Proyectil p;
      public Proyectil proyectiles[] = new Proyectil[1];
      Image img1;
}
```

Las variables de instancia de la clase Spaceship representan su movimiento en “x” e “y”, la “velocidad” de movimiento de la nave, la cantidad de “vidas”, el estado “puedeDisparar” da el atributo para que la misma dispare, “puntaje” almacena el puntaje, el objeto “p” de la clase Proyectil es un arreglo que define los proyectiles de la nave, “img1” almacena la imagen de la nave.

```
public Spaceship(double x, double y, int vidas, double velocidad) {
	this.x = x;
	this.y = y;
	this.vidas = vidas;
	this.velocidad = velocidad;
	img1 = Herramientas.cargarImagen("nave.png");
}
```

El constructor de esta clase recibe las coordenadas iniciales `x` e `y` de la nave, la cantidad de vidas “vidas” y la velocidad de movimiento “velocidad”, y las asigna a los atributos correspondientes.

```
public double setX() {
    this.x=x;
  }
```

Establece la posición horizontal “x” del rayo según el valor proporcionado.

```
public double setY() {
    this.y=y;
  }
```

Establece la posición vertical “y” del rayo según el valor proporcionado.

```
public void dibujarse(Entorno entorno) {
	entorno.dibujarImagen(img1, this.x, this.y, 0, 2);
}
```

Dibuja la imagen de la nave utilizando la clase Herramientas y el método dibujarImagen del objeto “entorno”.

```
public void aumentarVelocidad() {
	this.velocidad += 0.25;
}
```

Incrementa la velocidad de movimiento de la nave en 0.25 unidades.

```
public int getPuntaje() {
	return puntaje;
}
```

Devuelve el puntaje actual de la nave.

```
public void setPuntaje(int puntaje) {
	this.puntaje = puntaje;
}
```

Establece el puntaje de la nave según el valor proporcionado.

```
public boolean puedeDisparar() {
	return puedeDisparar;
}
```

Devuelve un booleano que indica si la nave puede disparar en el momento actual.

```
public void setPuedeDisparar(boolean puedeDisparar) {
	this.puedeDisparar = puedeDisparar;
}
```

Establece el valor de la variable `puedeDisparar` según el valor booleano proporcionado

```
public void moverDerecha(Entorno entorno) {
	if (this.x < entorno.ancho() - 30) {
		this.x += this.velocidad * 2;
	}
}
```

Mueve la nave hacia la derecha. Verifica que la nave no salga del límite derecho del entorno.

```
public void moverIzquierda(Entorno entorno) {
	if (this.x > 35) {
		this.x -= this.velocidad * 2;
	}
}
```

Mueve la nave hacia la izquierda. Verifica que la nave no salga del límite izquierdo del entorno.

```
public int getVidas() {
	return vidas;
}
```

Devuelve la cantidad de vidas de la nave.

```
public void setVidas(int vidas) {
	this.vidas = vidas;
}
```

Establece la cantidad de vidas de la nave según el valor proporcionado.

```
public Rectangle navecitaHitbox() {
	return new Rectangle((int) this.x, (int) this.y, 32, 64);
	}
```

Devuelve un objeto de tipo Rectangle que representa el área rectangular de colisión de la nave. El rectángulo se crea con las coordenadas “x”, “y” de la nave y un ancho y alto de 32 y 64 respectivamente.

```
public void disparar() {
	if (puedeDisparar) {
		p = new Proyectil((int) x, (int) y);
		proyectiles[0] = p;
		// Se declara y usa sonido proyectil
			File file2 = new File("src/proyectil.wav");
			AudioInputStream audioStream2 =
			AudioSystem.getAudioInputStream(file2);
			Clip clipProyectil = AudioSystem.getClip();
			clipProyectil.open(audioStream2);
			clipProyectil.start();
	}
	puedeDisparar = false;
}
```

Crea un nuevo proyectil y lo agrega al arreglo de proyectiles de la nave. Además, desactiva la posibilidad de disparar nuevamente hasta que el proyectil salga de la pantalla. Se agrega sonido al proyectil utilizando la clase Clip.

```
public Proyectil fueraDePantalla(Proyectil proyectil) {
	if ((int) proyectil.getY() <= 0) {
		proyectil = null;
	}
	return proyectil;
}
```

Verifica si un proyectil está fuera de la pantalla. Si es así, lo marca como nulo, eliminándolo.

```
public boolean colisionConIon(Spaceship navecita, Iones[] ionesArr) {
	boolean bandera = false;
	for (int i = 0; i < ionesArr.length; i++) {
		if (ionesArr[i] != null)
			{if
(navecita.navecitaHitbox().intersects(ionesArr[i].ionHitbox())) {
				ionesArr[i] = null;
				bandera = true;
			}
		} else if (ionesArr[i] == null) {
			continue;
		}
	}
	return bandera;
}
```

Verifica si la nave colisiona con algún objeto de tipo Iones en el arreglo “ionesArr”. Si hay una colisión, elimina el objeto Iones del arreglo y devuelve true.

Conclusión:

Durante el desarrollo del trabajo practico, nos dimos cuenta lo complicado que es implementar correctamente todos los objetos y que congenien correctamente unos con otros.

Además, lo difícil que es hacer hitboxes que sean fieles a los objetos mostrados, es algo que desde un primer momento se muestra explícitamente.

A pesar de estas complicaciones, el desarrollo del juego en las ultimas 3 semanas aproximadamente, fue muy ameno porque supimos trabajar en equipo y congeniar en todo momento.