

ASTROMEGASHIP

GRUPO 4



1. e mayo de 2023

González Darío – Quintana Lucas - Lopez Ciro

# ASTROMEGASHIP

## Introducción:

El propósito de este informe es realizar un análisis del videojuego AstroMegaShip, creado por los miembros del Grupo N 4, López Ciro, Quintana Lucas, y González Darío. AstroMegaShip es un juego de acción espacial emocionante en el que los jugadores juegan como pilotos espaciales y deben enfrentar hordas de meteoritos y destructores estelares.

El juego esta desarrollado en java y es de dos dimensiones. Los jugadores tienen que controlar a la nave espacial MegaShip. El objetivo fundamental del juego es sobrevivir evitando todos los obstáculos y eliminar a las amenazas enemigas lo antes posible para evitar que sigan apareciendo otras.

## Descripción:

## Juego:

La clase Juego es la clase principal del juego AstroMegaShip.

### Variables de Instancia:

entorno: objeto de la clase Entorno que representa el entorno del juego.

img: objeto de la clase Image que almacena la imagen de fondo del juego.

imgPerdiste: objeto de la clase Image que almacena la imagen mostrada cuando se pierde el juego.

imgGanaste: objeto de la clase Image que almacena la imagen mostrada cuando se gana el juego.

megaShip: objeto de la clase MegaShip que representa la nave controlada por el jugador.

meteorito: arreglo de objetos de la clase Meteoritos que representa los meteoritos presentes en el juego.

disparo: arreglo de objetos de la clase Disparo que representa los disparos realizados por la nave.

destructorEstelar: arreglo de objetos de la clase DestructorEstelar que representa los destructores estelares en el juego.

disparoDestructor: arreglo de objetos de la clase DisparoDestructor que representa los disparos de los destructores estelares.

segundos: variable entera que lleva el conteo de los segundos transcurridos en el juego.

segundosDestructores: variable entera que lleva el conteo de los segundos para utilizarlos en la función de agregar nuevos destructores.

segundosMeteoritos: variable entera que lleva el conteo de los segundos para utilizarlos en la función de agregar nuevos meteoritos

perdiste: variable booleana que indica si el jugador ha perdido el juego.

puntos: variable entera que almacena la cantidad de puntos obtenidos por el jugador.

posicionDestruidoDestructor: Nos pasa cual es la posición del destructor destruido en el arreglo destructorEstelare.

posicionDestruidoMeteorito:Nos pasa cual es la posición del meteorito destruido en el arreglo de meteoritos

detectorDestruidosDestrucotres.cariable boolean que indica que se destruyó al menos un destructor.  
detectorDestruidosMeteorito:variable entera que cuenta cuantos meteoritos fueron destruidos para crear unos nuevos después de determinado tiempo.

### Métodos:

Juego(): constructor de la clase Juego que inicializa el entorno, instancia los objetos necesarios y asigna posiciones iniciales.

tick(): método que se ejecuta en cada iteración del juego. Se encarga de dibujar la pantalla, gestionar los movimientos de la nave, detectar colisiones, dibujar los elementos del juego, generar disparos de los destructores, verificar las colisiones con la nave y mostrar las pantallas de victoria o derrota.

detectarColisiones(): método que verifica si los disparos realizados por la nave colisionan con los meteoritos o destructores estelares y realiza las acciones correspondientes.

verificarColisionesMegaShip(): método que verifica si la nave colisiona con los meteoritos, destructores estelares o disparos de los destructores y determina si el jugador ha perdido el juego.

main(String[] args): método principal que crea una instancia de la clase Juego y comienza la ejecución del juego, además de llamar al método Sonido.

## Clase MetodosDelJuego:

La clase "MetodosDelJuego" proporciona algunos métodos utilizados en la clase juego para realizar diversas funciones.

### Métodos:

moverDisparos(Disparo[] disparo): Este método recorre el arreglo de disparos y los mueve hacia arriba. Si un disparo sale de la pantalla, se elimina estableciendo su valor en null.

detectarColisiones(Disparo[] disparo, Meteoritos[] meteorito, DestructorEstelar[] destructorEstelar, DisparoDestructor[] disparoDestructor, int puntos): Este método verifica las colisiones entre los disparos, los meteoritos y los destructores estelares. Si hay una colisión, se destruye el objeto correspondiente (meteorito o destructor estelar), se elimina el disparo y se incrementa la puntuación.

dibujarDisparos(Disparo[] disparo, Entorno entorno): Este método dibuja los disparos y los mueve hacia arriba en cada iteración.

dibujarMeteoritos(Meteoritos[] meteorito, Entorno entorno): Este método dibuja los meteoritos, aplica su movimiento y rotación en cada iteración.

dibujarDestructoresEstelares(DestructorEstelar[] destructorEstelar, Entorno entorno): Este método dibuja los destructores estelares, aplica su movimiento y muestra la cantidad restante de destructores estelares.

moverDisparosDestructor(DisparoDestructor[] disparoDestructor, Entorno entorno): Este método mueve los disparos de los destructores estelares hacia abajo en cada iteración y los dibuja en el entorno.

generarDisparosDestructor(DestructorEstelar[] destructorEstelar, DisparoDestructor[] disparoDestructor, int segundos): Este método genera disparos de los destructores estelares en momentos específicos del juego, determinados por el valor de la variable "segundos".

verificarColisionesMegaShip(MegaShip megaShip, Meteoritos[] meteorito, DestructorEstelar[] destructorEstelar, DisparoDestructor[] disparoDestructor, boolean perdiste): Este método verifica las colisiones entre la nave principal (MegaShip) y los meteoritos, destructores estelares y disparos de los destructores estelares. Si hay una colisión, se establece la variable "perdiste" en true y se elimina al MegaShip.

mostrarPantallaPerdiste(Entorno entorno, boolean perdiste, Image imgPerdiste): Este método muestra la pantalla de "perdiste" en el entorno si la variable "perdiste" es true.

mostrarPantallaGanaste(Entorno entorno, int puntos, Image imgGanaste): Este método muestra la pantalla de "ganaste" en el entorno si la puntuación alcanza 5 puntos.

## Clase DestructorEstelar:

### Atributos:

ancho: ancho del destructor estelar.

largo: largo del destructor estelar.

x: coordenada x del destructor estelar.

y: coordenada y del destructor estelar.

imaDestructor: imagen del destructor estelar.

alternador: variable para controlar el movimiento del destructor estelar.

### Métodos:

DestructorEstelar(double x, double y): constructor de la clase DestructorEstelar. Inicializa las coordenadas del destructor estelar y carga su imagen.

movDestructor(Entorno e): actualiza la posición del destructor estelar en función de su movimiento.

dibujarDestructor(Entorno e): dibuja el destructor estelar.

dibujarDestructorRestantes(Entorno e, int posicion): dibuja los destructores estelares restantes en el entorno, en la posición especificada.

disparar(): crea y devuelve un objeto de tipo DisparoDestructor, correspondiente al disparo del destructor estelar.

colisionaDisparo(Disparo disp): verifica si el destructor estelar colisiona con un disparo.

destruirDestructor(DestructorEstelar[] destructoresEstelares, int destructorEstelar): elimina un destructor estelar del arreglo estableciendo su valor en null.

## Clase Disparo:

### Atributos:

x: coordenada x del disparo.

y: coordenada y del disparo.

imagenBala: imagen del disparo.

### Métodos:

Disparo(double x, double y): constructor de la clase Disparo. Inicializa las coordenadas del disparo y carga su imagen.

moverArriba(): mueve el disparo hacia arriba.

dibujar(Entorno e): dibuja el disparo.

agregarDisparo(Disparo[] cargador, Disparo bala): agrega un disparo al arreglo de disparos, en la primera posición disponible.

## Clase DisparoDestructor:

### Atributos:

x: coordenada x del disparo destructor.

y: coordenada y del disparo destructor.

imgDispDestructor: imagen del disparo destructor.

### Métodos:

DisparoDestructor(double x, double y): constructor de la clase DisparoDestructor. Inicializa las coordenadas del disparo destructor y carga su imagen.

dibujarDisparo(Entorno e): dibuja el disparo destructor en el entorno especificado.

movDisparoDestructor(): actualiza la posición del disparo destructor en función de su movimiento.

agregarDisparo(DisparoDestructor[] arregloDisparosDestructor, DisparoDestructor p): agrega un disparo destructor al arreglo de disparos destructor, en la primera posición disponible.

## Clase MegaShip:

### Atributos:

x: coordenada x de la nave MegaShip.

y: coordenada y de la nave MegaShip.

imagen: imagen de la nave MegaShip.

### Métodos:

MegaShip(double x, double y): constructor de la clase MegaShip. Inicializa las coordenadas de la nave MegaShip y carga su imagen.

dibujar(Entorno e): dibuja la nave MegaShip en el entorno especificado.

moverDerecha(Entorno e): mueve la nave MegaShip hacia la derecha si no supera los límites del entorno.

moverIzquierda(Entorno e): mueve la nave MegaShip hacia la izquierda si no supera los límites del entorno.

colisionaMeteorito(Meteoritos meteo): verifica si la nave MegaShip colisiona con un meteorito dado.

colisionaDestructor(DestructorEstelar destruc): verifica si la nave MegaShip colisiona con un destructor estelar dado.

colisionaDispDestructor(DisparoDestructor dispDestr): verifica si la nave MegaShip colisiona con un disparo del destructor estelar dado.

disparar(): crea un nuevo objeto de la clase Disparo ubicado en la posición de la nave MegaShip.

## Clase Meteoritos:

### Atributos:

x: coordenada x del meteorito.

y: coordenada y del meteorito.

anguloRotacion: ángulo de rotación del meteorito.

angulo: ángulo de movimiento del meteorito.

imaMeteorito: imagen del meteorito.

### Métodos:

Meteoritos(double x, double y): constructor de la clase Meteoritos. Inicializa las coordenadas y ángulos del meteorito, y carga la imagen del meteorito.

dibujar(Entorno e): dibuja el meteorito.

movimientoMeteorito(Entorno e): actualiza la posición del meteorito en función de su ángulo de movimiento.

rotacionMeteorito(): actualiza el ángulo de rotación del meteorito.

colisionConMeteorito(Meteoritos[] meteorito): verifica las colisiones entre los meteoritos.

colisionConDestructor(Meteoritos[] meteorito, DestructorEstelar[] destructor): verifica las colisiones entre los meteoritos y los destructores estelares.

colisionaDisparo(Disparo disp): verifica si el meteorito colisiona con un disparo.

destruirMeteorito(Meteoritos[] meteoritos, int p): elimina un meteorito del arreglo estableciendo su valor en null.

## Clase Sonido:

### Atributos:

File ubicacion: Es un objeto de la clase File que representa la ubicación del archivo de sonido. Se crea a partir del directorio especificado como argumento del método.

AudioInputStream audioStream: Es un objeto de la clase AudioInputStream que se utiliza para obtener el archivo de sonido. Se obtiene llamamdo a getAudioInputStream() que le pasa la ubicación.

Clip clip: Es un objeto de la clase Clip que representa el archivo de sonido.

clip.open(audioStream): Llama al método open() para abrir el archivo de sonido. Se pasa como argumento el objeto audioStream que contiene el archivo de sonido.

clip.start(): Llama al método start() para reproducir el sonido desde el inicio.

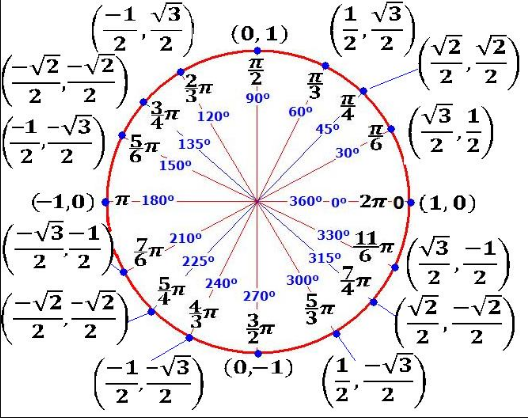
clip.loop(Clip.LOOP\_CONTINUOUSLY): Llama al método loop() para hacer que la música no pare de reproducirse.

### Métodos:

reproducirMusica(): este método es el encargado de reproducir la música de fondo del juego.

## Problemas:

### Pantalla:

Tuvimos un problema a la hora de poner las direcciones aleatorias de desplazamiento diagonal. Algunos meteoritos seguían derecho en vez de ir para la diagonal izquierda. Descubrimos luego de varios intentos de que estaba invertido los ángulos a lo que vemos nosotros normalmente, en vez de usar el 4/3 pi y 5/3 pi había que usar el pi/3 y ¾ pi

Nos costó entender que teníamos que ver el programa de una manera distinta a la que usamos normalmente en la vida.



### Eliminación de los Destructores Espaciales:

Pudimos crear un método para eliminar los meteoritos, no tuvimos ningún problema, pero a la hora para hacerlo en los destructores espaciales se nos complicó bastante.

Creíamos que sería fácil ya que pudimos hacerlo para los meteoritos entonces pensábamos que era más de lo mismo, al adaptar el método que usamos previamente notamos que al eliminar a un destructor espacial se eliminaban todos los que estaban a su derecha, algo que nos pareció sumamente raro y tuvimos que pensar demasiado para resolverlo.

Al final resulto ser que el problema no estaba en el método de eliminación del destructor espacial, sino en el momento de la creación, ya que no estábamos colocando que cuando uno de los destructores sea null no se dibuje, lo que hacía que se intente dibujar el null y al no poder se eliminaban a todos los destructores siguientes del array.

### Múltiples balas:

Cuando apretábamos espacio para disparar notamos que en vez de lanzar una bala se lanzaban 3 al mismo tiempo, tuvimos varias teorías como que si era la sensibilidad del teclado o que estaba programado para lanzar 3; pensamos en que la nave pueda lanzar una bala cada vez que la bala anterior se haya destruido, pero esta opción no era muy buena ya que tardaba mucho tiempo y hacia el juego imposible, al final optamos por consultarle al profesor cual podría ser el problema y nos brindó la información de que podríamos manipular los ticks a nuestro favor, gracias a esto pudimos controlar el tiempo de cada cuanto el AstroMegaShip puede disparar.

## Implementación:

package juego;

//LIBRERIAS

import entorno.Entorno;

import java.awt.Image;

import entorno.InterfaceJuego;

import entorno.Herramientas;

public class Juego extends InterfaceJuego {

// DECLARACION DE VARIABLES

private Entorno entorno;

private Image img;

private Image gameOver = Herramientas.cargarImagen("gameover.gif");

private Image imgGanaste = Herramientas.cargarImagen("imagenganadora.gif");

private Image textGanaste = Herramientas.cargarImagen("ganasteTexto.gif");

private MegaShip megaShip;

private Meteoritos[] meteorito;

private Disparo[] disparo;

private DestructorEstelar[] destructorEstelar;

private DisparoDestructor[] disparoDestructor;

private int segundos;

private int segundosDestructores;

private int segundosMeteoritos;

private boolean perdiste = false;

private int puntos = 0;

private boolean detectorDestruidosDestructores = false;

private int detectorDestruidosMeteoritos = 0;

private int posicionDestruidoDestructor;

private int posicionDestruidoMeteorito;

public Juego() {

//Cargamos el entorno, Intanciamos objetos y les asignamos posiciones

this.entorno = new Entorno(this, "AstroMegaShip - Grupo N 4 - Lopez - Quintana - Gonzalez - V0.01", 800, 600);

this.img = Herramientas.cargarImagen("FondoEspacial.gif");

this.megaShip = new MegaShip(entorno.ancho() / 2, entorno.alto() - 50);

this.disparo = new Disparo[1];

this.meteorito = new Meteoritos[5];

for (int unMeteorito = 0; unMeteorito < 5; unMeteorito++) {

this.meteorito[unMeteorito] = new Meteoritos(30 + (entorno.ancho() / 5) \* unMeteorito,

-100 - (entorno.alto()) / 5);

}

this.destructorEstelar = new DestructorEstelar[5];

for (int unDestructor = 0; unDestructor < 5; unDestructor++) {

this.destructorEstelar[unDestructor] = new DestructorEstelar(60 + (entorno.ancho() / 5) \* unDestructor,

-100 - (entorno.alto() / 5) \* (Math.random() \* 4 + 1));

}

this.disparoDestructor = new DisparoDestructor[100];

//INICIAMOS EL ENTORNO

this.entorno.iniciar();

}

public void tick() {

if (megaShip != null) {

// DIBUJAR PANTALLA DEL JUEGO

entorno.dibujarImagen(img, entorno.ancho() / 2, entorno.alto() / 2, 2.2, 1.5);

// DETECCION DE TECLAS y MOVIMIENTOS DE LA NAVE

if (entorno.estaPresionada(entorno.TECLA\_DERECHA))

megaShip.moverDerecha(entorno);

if (entorno.estaPresionada(entorno.TECLA\_IZQUIERDA))

megaShip.moverIzquierda(entorno);

if (entorno.estaPresionada(entorno.TECLA\_ESPACIO)) {

Disparo.agregarDisparo(disparo, megaShip.disparar());

}

// FUNCIONES DE LOS DISPAROS

if (disparo != null) {

detectarColisiones();

}

MetodosDelJuego.moverDisparos(disparo);

// AGREGAMOS CONTADORES DEL TIEMPO

segundos++;

segundosDestructores++;

segundosMeteoritos++;

// DIBUJAMOS

if (megaShip != null) {

megaShip.dibujar(entorno);

}

MetodosDelJuego.dibujarDisparos(disparo, entorno);

MetodosDelJuego.dibujarMeteoritos(meteorito, entorno);

MetodosDelJuego.dibujarDestructoresEstelares(destructorEstelar, entorno);

MetodosDelJuego.moverDisparosDestructor(disparoDestructor, entorno);

// Colisiones Bala

detectarColisiones();

// AGREGAR DISPAROS DESTRUCTOR

agregarDisparosDestructor();

// AGREGAR NUEVOS METEORITOS Y DESTRUCTORES

if (detectorDestruidosMeteoritos > 0 && segundosMeteoritos >= 500) {

destructorEstelar[posicionDestruidoMeteorito] = new DestructorEstelar(

60 + (entorno.ancho() / 5) \* posicionDestruidoMeteorito,

-100 - (entorno.alto() / 5) \* (Math.random() \* 4 + 1));

detectorDestruidosMeteoritos--;

segundosMeteoritos = 0;

}

if (detectorDestruidosDestructores && segundosDestructores >= 1000) {

puntos--;

destructorEstelar[posicionDestruidoDestructor] = new DestructorEstelar(

60 + (entorno.ancho() / 5) \* posicionDestruidoDestructor,

-100 - (entorno.alto() / 5) \* (Math.random() \* 4 + 1));

detectorDestruidosDestructores = false;

segundosDestructores = 0;

}

// DESTRUCCION DEL MEGASHIP POR COLISION

verificarColisionesMegaShip();

// DESTRUCCION DE METEORITOS POR COLISIONES

Meteoritos.colisionConMeteorito(meteorito, detectorDestruidosMeteoritos);

Meteoritos.colisionConDestructor(meteorito, destructorEstelar, detectorDestruidosMeteoritos);

}

if (perdiste) {

megaShip = null;

MetodosDelJuego.mostrarPantallaPerdiste(entorno, perdiste, gameOver);

}

if (puntos == 5) {

megaShip = null;

MetodosDelJuego.mostrarPantallaGanaste(entorno, puntos, imgGanaste, textGanaste);

}

}

// METODOS

public void detectarColisiones() {

for (int bala = 0; bala < 1; bala++) {

if (disparo[bala] != null) {

for (int unElemento = 0; unElemento < 5; unElemento++) {

if (meteorito[unElemento] != null && meteorito[unElemento].colisionaDisparo(disparo[bala])) {

Meteoritos.destruirMeteorito(meteorito, unElemento);

disparo[bala] = null;

detectorDestruidosMeteoritos++;

posicionDestruidoMeteorito = unElemento;

break;

}

if (destructorEstelar[unElemento] != null

&& destructorEstelar[unElemento].colisionaDisparo(disparo[bala])) {

DestructorEstelar.destruirDestructor(destructorEstelar, unElemento);

disparo[bala] = null;

puntos++;

detectorDestruidosDestructores = true;

posicionDestruidoDestructor = unElemento;

break;

}

}

}

}

}

public void verificarColisionesMegaShip() {

if (megaShip != null) {

for (int i = 0; i < 5; i++) {

if (meteorito[i] != null && megaShip.colisionaMeteorito(meteorito[i])) {

perdiste = true;

megaShip = null;

break;

}

if (destructorEstelar[i] != null && megaShip.colisionaDestructor(destructorEstelar[i])) {

perdiste = true;

megaShip = null;

break;

}

if (disparoDestructor[i] != null && megaShip.colisionaDispDestructor(disparoDestructor[i])) {

perdiste = true;

megaShip = null;

break;

}

}

}

}

public void agregarDisparosDestructor() {

if (segundos == 100 || segundos == 150 || segundos == 200) {

int cantidadDisparos = 1;

if (segundos == 100 || segundos == 150) {

cantidadDisparos = 2;

}

MetodosDelJuego.generarDisparosAleatorios(destructorEstelar, disparoDestructor, cantidadDisparos);

segundos = 0;

}

}

@SuppressWarnings("unused")

public static void main(String[] args) {

Juego juego = new Juego();

// Agregamos Musica

Sonido sonido = new Sonido();

sonido.reproducirMusica("src/musica.wav");

}

}

package juego;

import entorno.\*;

import java.awt.Image;

public class MetodosDelJuego {

public static void moverDisparos(Disparo[] disparo) {

for (int bala = 0; bala < 1; bala++) {

if (disparo[bala] != null) {

if (disparo[bala].y < 0) {

disparo[bala] = null;

} else {

disparo[bala].moverArriba(); // Actualizar posición del disparo

}

}

}

}

public static void dibujarDisparos(Disparo[] disparo, Entorno entorno) {

for (int bala = 0; bala < disparo.length; bala++) {

if (disparo[bala] != null) {

disparo[bala].dibujar(entorno);

disparo[bala].moverArriba();

}

}

}

public static void dibujarMeteoritos(Meteoritos[] meteorito, Entorno entorno) {

for (int unMeteorito = 0; unMeteorito < meteorito.length; unMeteorito++) {

if (meteorito[unMeteorito] != null) {

meteorito[unMeteorito].dibujar(entorno);

meteorito[unMeteorito].movimientoMeteorito(entorno);

meteorito[unMeteorito].rotacionMeteorito();

}

}

}

public static void dibujarDestructoresEstelares(DestructorEstelar[] destructorEstelar, Entorno entorno) {

for (int unDestructor = 0; unDestructor < destructorEstelar.length; unDestructor++) {

if (destructorEstelar[unDestructor] != null) {

destructorEstelar[unDestructor].dibujarDestructor(entorno);

destructorEstelar[unDestructor].movDestructor(entorno);

int posicion = (unDestructor) \* 40;

destructorEstelar[unDestructor].dibujarDestructorRestantes(entorno, posicion);

}

}

}

public static void moverDisparosDestructor(DisparoDestructor[] disparoDestructor, Entorno entorno) {

for (int balaDestructor = 0; balaDestructor < disparoDestructor.length; balaDestructor++) {

if (disparoDestructor[balaDestructor] != null) {

disparoDestructor[balaDestructor].dibujarDisparo(entorno);

if (disparoDestructor[balaDestructor].y > entorno.alto()) {

disparoDestructor[balaDestructor] = null;

} else {

disparoDestructor[balaDestructor].movDisparoDestructor();

}

}

}

}

public static void generarDisparosAleatorios(DestructorEstelar[] destructorEstelar,

DisparoDestructor[] disparoDestructor, int cantidad) {

for (int i = 0; i < cantidad; i++) {

int random = (int) (Math.random() \* 5);

if (destructorEstelar[random] != null) {

DisparoDestructor.agregarDisparo(disparoDestructor, destructorEstelar[random].disparar());

}

}

}

public static void mostrarPantallaPerdiste(Entorno entorno, boolean perdiste, Image gameOver) {

if (perdiste) {

entorno.dibujarImagen(gameOver, entorno.ancho() / 2, entorno.alto() / 2.5, 0);

}

}

public static void mostrarPantallaGanaste(Entorno entorno, int puntos, Image imgGanaste, Image textoGanaste) {

entorno.dibujarImagen(imgGanaste, entorno.ancho() / 2, entorno.alto() / 2.5, 0);

entorno.dibujarImagen(textoGanaste, entorno.ancho() / 2, entorno.alto() / 1.5, 0);

}

}

package juego;

import entorno.Entorno;

import java.awt.Image;

import entorno.Herramientas;

public class MegaShip {

private double x;

private double y;

private Image imagen;

public MegaShip(double x, double y) {

this.x = x;

this.y = y;

this.imagen = Herramientas.cargarImagen("MegaShip.gif");

}

public void dibujar(Entorno e) {

e.dibujarImagen(imagen, x, y, 0, 0.2);

}

public void moverDerecha(Entorno e) {

if (x + 20 < e.ancho()) { // LA NAVE MIDE 40

x += 2.5;

}

}

public void moverIzquierda(Entorno e) {

if (x - 20 > 0) {

x -= 2.5;

}

}

public boolean colisionaMeteorito(Meteoritos meteo) {

int anchoMegaShip = 20;

int largoMegaship = 40;

boolean a = y + largoMegaship >= -largoMegaship + meteo.y;

boolean b = y - largoMegaship <= largoMegaship + meteo.y;

boolean c = x + anchoMegaShip >= meteo.x - anchoMegaShip;

boolean d = x - anchoMegaShip <= meteo.x + anchoMegaShip;

return a && b && c && d;

}

public boolean colisionaDestructor(DestructorEstelar destruc) {

int anchoMegaShip = 20;

int largoMegaship = 40;

boolean a = y + largoMegaship >= -largoMegaship + destruc.y;

boolean b = y - largoMegaship <= largoMegaship + destruc.y;

boolean c = x + anchoMegaShip >= destruc.x - anchoMegaShip;

boolean d = x - anchoMegaShip <= destruc.x + anchoMegaShip;

return a && b && c && d;

}

public boolean colisionaDispDestructor(DisparoDestructor dispDestr) {

int anchoMegaShip = 13;

int largoMegaship = 38;

int anchoYlargoDisparo = 5;

boolean a = y + largoMegaship >= -anchoYlargoDisparo + dispDestr.y;

boolean b = y - largoMegaship <= anchoYlargoDisparo + dispDestr.y;

boolean c = x + anchoMegaShip >= dispDestr.x - anchoYlargoDisparo;

boolean d = x - anchoMegaShip <= dispDestr.x + anchoYlargoDisparo;

return a && b && c && d;

}

public Disparo disparar() {

return new Disparo(x, y - 35);

}

}

package juego;

import java.awt.Image;

import entorno.Entorno;

import entorno.Herramientas;

public class Disparo {

public double x;

public double y;

private Image imagenBala;

public Disparo(double x, double y) {

this.x = x;

this.y = y;

this.imagenBala = Herramientas.cargarImagen("disparoMegaShip.png");

}

public void moverArriba() {

this.y -= 4;

}

public void dibujar(Entorno e) {

e.dibujarImagen(imagenBala, x, y, 0, 0.02);

}

public static void agregarDisparo(Disparo[] cargador, Disparo bala) {

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

if (cargador[i] == null) {

cargador[i] = bala;

}

}

}

}

package juego;

import java.awt.Image;

import entorno.Entorno;

import entorno.Herramientas;

public class Meteoritos {

public double x;

public double y;

private double anguloRotacion;

private double angulo;

private Image imaMeteorito;

public Meteoritos(double x, double y) {

double aleatorio = (int) (Math.random() \* 10 + 1);

double modAngulo;

if (aleatorio <= 5) {

modAngulo = Math.PI / 3;

} else {

modAngulo = (Math.PI \* 3) / 4;

}

this.x = x;

this.y = y \* Math.random();

this.angulo = modAngulo;

this.anguloRotacion = 0;

this.imaMeteorito = Herramientas.cargarImagen("Meteorito.png");

}

public void dibujar(Entorno e) {

double escala = 0.5;

e.dibujarImagen(imaMeteorito, x, y, anguloRotacion, escala);

}

public void movimientoMeteorito(Entorno e) {

if (y < e.alto() + 100) {

this.x = x + Math.cos(angulo);

this.y = y + Math.sin(angulo);

} else {

this.y = -35;

this.x = 30 + (e.ancho() / 5) \* (1 + Math.random() \* 4);

this.y = y + Math.sin(angulo);

}

}

public void rotacionMeteorito() {

anguloRotacion += 0.015;

}

public static void colisionConMeteorito(Meteoritos[] meteorito, int detectorDestruidosMeteoritos) {

double ancho = 55;

double largo = 55;

for (int meteorito1 = 0; meteorito1 < meteorito.length; meteorito1++) {

if (meteorito != null) {

for (int meteorito2 = 0; meteorito2 < meteorito.length; meteorito2++) {

if (meteorito1 != meteorito2 && // Evitar comparar un objeto consigo mismo

meteorito[meteorito1] != null && meteorito[meteorito2] != null

&& meteorito[meteorito1].x + ancho >= meteorito[meteorito2].x

&& meteorito[meteorito1].x <= meteorito[meteorito2].x + ancho

&& meteorito[meteorito1].y + largo >= meteorito[meteorito2].y

&& meteorito[meteorito1].y <= meteorito[meteorito2].y + largo) {

Meteoritos.destruirMeteorito(meteorito, meteorito1);

Meteoritos.destruirMeteorito(meteorito, meteorito2);

detectorDestruidosMeteoritos++;

}

}

}

}

}

public static void colisionConDestructor(Meteoritos[] meteorito, DestructorEstelar[] destructor,

int detectorDestruidosMeteoritos) {

double anchoMeteorito = 55;

double largoMeteorito = 55;

double anchoDestructor = 35;

double largoDestructor = 35;

for (int meteorito1 = 0; meteorito1 < meteorito.length; meteorito1++) {

if (meteorito[meteorito1] != null) {

for (int destructorEstelar = 0; destructorEstelar < destructor.length; destructorEstelar++) {

if (destructor[destructorEstelar] != null

&& meteorito[meteorito1].x + anchoMeteorito >= destructor[destructorEstelar].x

&& meteorito[meteorito1].x <= destructor[destructorEstelar].x + anchoDestructor

&& meteorito[meteorito1].y + largoMeteorito >= destructor[destructorEstelar].y

&& meteorito[meteorito1].y <= destructor[destructorEstelar].y + largoDestructor) {

Meteoritos.destruirMeteorito(meteorito, meteorito1);

detectorDestruidosMeteoritos++;

break;

}

}

}

}

}

public boolean colisionaDisparo(Disparo disp) {

int anchoLargoMeteorito = 35;

int anchoLargoBala = 5;

boolean a = y + anchoLargoMeteorito >= -anchoLargoBala + disp.y;

boolean b = y - anchoLargoMeteorito <= anchoLargoBala + disp.y;

boolean c = x + anchoLargoMeteorito >= disp.x - anchoLargoBala;

boolean d = x - anchoLargoMeteorito <= disp.x + anchoLargoBala;

return a && b && c && d;

}

public static void destruirMeteorito(Meteoritos[] meteoritos, int p) {

meteoritos[p] = null;

}

}

package juego;

import entorno.Entorno;

import java.awt.Image;

import entorno.Herramientas;

public class DestructorEstelar {

public double ancho = 35;

public double largo = 35;

public double x;

public double y;

private Image imaDestructor;

private int alternador;

public DestructorEstelar(double x, double y) {

this.x = x;

this.y = y;

this.imaDestructor = Herramientas.cargarImagen("Destructor.png");

}

public void movDestructor(Entorno e) {

if (y < e.alto() + 100) {

if (x - ancho > 0 && x + ancho < e.ancho()) {

if (alternador < 100) {

this.x++;

alternador++;

}

if (alternador >= 99 && alternador < 200) {

this.x--;

alternador++;

}

if (alternador == 200) {

alternador = 0;

}

}

if (x - ancho == 0) {

this.x++;

}

if (x + ancho == e.ancho()) {

this.x--;

}

this.y++;

} else {

this.y = -largo;

}

}

public void dibujarDestructor(Entorno e) {

double angulo = 0;

double escala = .5;

e.dibujarImagen(imaDestructor, x, y, angulo, escala);

}

public void dibujarDestructorRestantes(Entorno e, int posicion) {

double angulo = 0;

double escala = .25;

double xAncho = e.ancho() - 20 - posicion;

double yAlto = e.alto() - 20;

e.dibujarImagen(imaDestructor, xAncho, yAlto, angulo, escala);

}

public DisparoDestructor disparar() {

return new DisparoDestructor(x, y + 20);

}

public boolean colisionaDisparo(Disparo disp) {

boolean a = y + largo >= -5 + disp.y;

boolean b = y - largo <= 5 + disp.y;

boolean c = x + ancho >= disp.x - 5;

boolean d = x - ancho <= disp.x + 5;

return a && b && c && d;

}

public static void destruirDestructor(DestructorEstelar[] destructoresEstelares, int destructorEstelar) {

destructoresEstelares[destructorEstelar] = null;

}

}

package juego;

import java.awt.Image;

import entorno.Entorno;

import entorno.Herramientas;

public class DisparoDestructor {

public double x;

public double y;

private Image imgDispDestructor;

public DisparoDestructor(double x, double y) {

this.x = x;

this.y = y;

this.imgDispDestructor = Herramientas.cargarImagen("disparoDestructor.png");

}

public void dibujarDisparo(Entorno e) {

double angulo = 0;

double escala = 0.05;

e.dibujarImagen(imgDispDestructor, x, y, angulo, escala);

}

public void movDisparoDestructor() {

this.y += 3;

}

public static void agregarDisparo(DisparoDestructor[] arregloDisparosDestructor, DisparoDestructor p) {

for (int disparo = 0; disparo < arregloDisparosDestructor.length; disparo++) {

if (arregloDisparosDestructor[disparo] == null) {

arregloDisparosDestructor[disparo] = p;

break;

}

}

}

}

package juego;

import java.io.File;

import javax.sound.sampled.AudioInputStream;

import javax.sound.sampled.AudioSystem;

import javax.sound.sampled.Clip;

public class Sonido {

void reproducirMusica(String directorio) {

try {

File ubicacion = new File(directorio); // Objeto que contiene la ubicacion del archivo de sonido

if (ubicacion.exists()) // Comprobamos que el archivo exista

{

AudioInputStream audioStream = AudioSystem.getAudioInputStream(ubicacion); // Objeto que trae el archivo

// de sonido a eclipse

Clip clip = AudioSystem.getClip(); // Objeto que contiene el archivo de sonido

clip.open(audioStream); // Abre la musica

clip.start(); // Reproduce la musica

clip.loop(Clip.LOOP\_CONTINUOUSLY); // Hace que la musica vuevla a empezar

} else {

System.out.println("No se pudo encontrar el archivo");

}

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

## Conclusiones:

Hemos concluido el trabajo con éxito y pudimos concretar el proyecto, pero esto no fue una tarea para nada fácil, ya que se nos presentaron una cantidad enorme de complicaciones de las cuales pudimos salir y aprendimos varias lecciones que sin duda utilizaremos en los próximos trabajos en grupo.

Trabajar en grupo no es algo para nada sencillo, es muy distinto a realizar un programa solo por tu cuenta, gracias a este trabajo pudimos percatarnos de varios factores que hacen que trabajar en equipo, si no se tienen en claro estos tips, pueda ser un verdadero dolor de cabeza.

En primer lugar, hemos aprendido sobre la importancia de la planificación y el diseño antes de comenzar a programar. Definir claramente las clases, los atributos y los métodos necesarios permite tener una visión general del proyecto y avanzar de manera más eficiente. Este tip no lo hemos tenido presente lo que hizo que el desarrollo del programa fuera verdaderamente complejo ya que se improvisó sobre la marcha.

Hemos aprendido que la manera de escribir código de uno puede ser muy distinta a la del compañero, esto ha sido una dificultad ya que cuando realizábamos distintos métodos algunos lo hacían utilizando getters y setters y otros cargaban los datos directamente con los constructores o modificaban directamente la variable; algunos creaban dos clases, una para constructores y otra para los métodos y otros trabajaban todo en la misma clase. Estas diferencias hicieron que el código sea completamente distinto eh incompatible y se tuvo que optar por tomar una forma de programar y adaptarse a la misma.

La comunicación es muy importante, nos hemos encontrado con el problema de que dos integrantes estaban realizando los mismos métodos de manera separada y sin informar, lo que provocó que se hayan realizado los mismos métodos o clases enteras dos veces. Esto provocó que tengamos múltiples versiones del juego, algunas con arraylist, otras sin arraylist, otras con 7 clases, otras con 12.

Como grupo se tuvo mucha paciencia, empatía y se trabajó de una forma agradable y amistosa, los integrantes estaban abiertos a cambios de enfoque y a ajustarse a la opción más viable y conveniente. La buena actitud que provocó que tengamos ganas de realizar el proyecto e hizo que el trabajo en equipo sea muy ameno.

También todos los integrantes mostraron tener interés sobre el proyecto lo que hizo que se realizaran implementaciones por cuenta propia y avanzado más de lo esperado en poco tiempo.