Universidad Nacional General Sarmiento

Asignatura: Programación 1

Comisión: 3

Grupo: 08

Trabajo Práctico: “Cuidando a los monos capuchinos”

| **Nombre y Apellido** | **Legajo/DNI** | **Email** |
| --- | --- | --- |
| Matias Verón | 38666723/2019 | veronmatias94@gmail.com |
| Pablo Meza | 36645140 | pablo.cursada@gmail.com |
| Elva Orue | 94423596 | orueelva@gmail.com |

**Introducción**

El presente trabajo tiene como objetivo crear un juego/aplicación vinculado a la especie de monos capuchinos. Se cuenta con el siguiente conocimiento acerca de la especie mencionada:

*“El mono capuchino o mono caí (Sapajus cay) es una especie de las regiones cálidas de nuestro país y se encuentra cada vez más amenazada por la caza furtiva para el comercio ilegal. Normalmente los monos capuchinos viven en zonas selváticas alimentándose de hojas y frutas. Sus depredadores naturales incluyen serpientes, felinos como pumas y aves rapaces como las águilas. Los monos son capaces de correr por el suelo, saltar y trepar a los árboles. También son muy habilidosos y se defienden de sus depredadores lanzando piedras.“*

El juego consiste en un mono que aparece en la parte inferior izquierda de la pantalla, cuyas únicas acciones que puede realizar son saltar y lanzar piedras para eliminar a los depredadores que eventual y aleatoriamente irán apareciendo en pantalla. Estos últimos, entrarán en escena desde la derecha de la pantalla moviéndose ininterrumpidamente hacia la derecha en dirección al mono, y lo harán moviéndose en el piso (para el caso de los tigres) y sobre las ramas de los árboles (para el caso de las serpientes que estarán presentes ahí). Será eliminado cada depredador que sea impactado por una piedra que lance el mono, y en el caso de que este último salte esquivando el contacto con el depredador, este finalmente seguirá su curso hasta desaparecer luego de traspasar el límite de la izquierda de la pantalla. El contacto directo del mono con alguno de los depredadores significará un “Game Over” concluyendo el juego.

**Descripción de Clases**

**Clase**: Mono

Esta clase tiene como objetivo ocuparse de la aparición y acciones relativas al personaje jugable del mono.

**Variables de instancia**

int **x** → asigna una coordenada “x” para la posición donde aparece el mono.

int **y** → asigna una coordenada “y” para la posición donde aparece el mono.

double **angulo** → es una variable vinculada al ángulo de la imagen del mono y es invocada dentro del bloque de código del método “dibujarse” (ver a continuación).

Image **img1** → en esta variable se cargará la imagen del mono que luego se dibujará en el juego.

int **piso →** es una variable que ayudará con el control de los instantes en que el mono está en el piso o no lo está.

Point **punto →** esta variable colaborará con la precisión de las coordenadas donde se ubica el mono en un rectángulo a definir.

Rectangle **rectanguloMono →** esta variable colabora en poner límites a las coordenadas “x” e “y” donde está la imagen del mono y ofrece un cierto margen de espacios (ancho y alto) para poder elaborar las colisiones con los otros objetos.

int **sePresiono →** esta variable cumple la función de ayudar con el control de gravedad, salto del mono y la presión de la tecla arriba para desencadenar el salto.

int **volviendo** → esta variable ayuda con el control del retorno del mono hacia el piso una vez que realizó el salto completo.

**Métodos**

**constructor (int x, int y)**

Recibe las coordenadas “x” e “y” de aparición del mono, y luego carga la imagen del mono.

**dibujarse (Entorno entorno)**

Recibe un parámetro del tipo Entorno para invocar un método “dibujarImagen” que recibirá los parámetros necesarios para dibujar al mono en la esquina inferior izquierda de la pantalla.

**saltando (int saltY)**

Este método controla la trayectoria hacia arriba del mono cuando realiza el salto, recibiendo como único parámetro un número entero.

**vuelvePiso (int volviendo)**

Este método realiza la operación inversa al método anterior con el objetivo de devolver el mono al piso desde donde inició el salto. También recibe un número entero como parámetro.

**boolean colision (Rectangle rectanguloPlat)**

Este método realiza una serie de controles respecto al contacto del mono con la plataforma, para confirmar o negar la colisión de ambos objetos. Retornará un valor booleano dependiendo del resultado de las evaluaciones de las condiciones.

**Clase**: Fondo

Esta clase tiene como objetivo crear las imágenes de fondo donde se desarrollarán los eventos del juego, simulando una animación de movimiento del escenario (además de los depredadores que avanzan).

**Variables de instancia**

Image **pastoFijo →** una variable del tipo Image donde se cargará una imagen que ilustra pasto.

Image **fondoAzul →** una variable del tipo Image donde se cargará una imagen de fondo azul.

Point **posicion →** es una variable para determinar la posición desde la cual se dibujará la imagen del fondo.

**Métodos**

**constructor (no recibe parámetros)**

Se cargarán las imágenes del fondo del juego y en la variable “posicion”, las coordenadas desde donde partirán los dibujos del fondo.

**dibujar (Entorno entorno, int rotacion)**

Recibe un parámetro del tipo Entorno para invocar un método “dibujarImagen” que recibirá los parámetros necesarios para dibujar el fondo de acuerdo a los límites que rigen las medidas de pantalla. Además, recibe un parámetro para la rotación de la imagen y que será invocado nuevamente en el método “dibujarImagen”.

**Clase**: Piedra

Esta clase tiene como objetivo ocuparse de la creación y eventos relacionados con la piedra que puede lanzar el mono para deshacerse de sus depredadores.

**Variables de instancia**

Entorno **entorno →** recibe una variable de tipo entorno para poder realizar operaciones utilizando las bibliotecas del package entorno.

Point **spawn →** este punto contendrá las coordenadas desde las cuales se creará la piedra (la misma posición donde esté el mono).

Rectangle **rectanguloPiedra** → esta variable colabora en poner límites a las coordenadas “x” e “y” donde está la imagen de la piedra y ofrece un cierto margen de espacios (ancho y alto) para poder elaborar las colisiones con los otros objetos.

Image **piedra →** una variable del tipo Image donde se cargará la imagen de la piedra.

**Métodos**

**constructor (int x, int y)**

Recibe como parámetros las coordenadas “x” e “y” de aparición de la piedra, y luego carga la imagen de la piedra.

**dibujarse (Entorno entorno)**

Recibe un parámetro del tipo Entorno para llamar un método “dibujarImagen” y un “dibujarRectangulo” que recibirán los parámetros necesarios (dentro de los cuales se hallan las variables de instancia) para dibujar a la piedra en la misma posición donde se halla el mono.

**moverDerecha (no recibe parámetros)**

Este método realizará el movimiento horizontal hacia la derecha de la piedra que sea creada y dibujada, lográndolo a través del incremento del valor del “x” de la variable del tipo Point “spawn” determinada como propiedad del objeto.

**boolean colision (Rectangle rectanguloDepred)**

Este método realiza una serie de controles respecto al contacto de la piedra con los depredadores, para confirmar o negar la colisión de ambos objetos. Retornará un valor booleano dependiendo del resultado de las evaluaciones de las condiciones.

**Clase**: Tigre

Esta clase tiene como objetivo ocuparse de la creación y eventos relacionados con los tigres que irán apareciendo como depredadores terrestres del mono, es decir, el valor de “y” de estos personajes será muy cercano o igual al valor de “y” donde se ubica el mono, ya que el movimiento de los tigres será horizontal hacia la izquierda en dirección al mono.

**Variables de instancia**

Point **punto →** este punto contendrá las coordenadas desde las cuales se creará el tigre, que será con un “x” que exceda el límite horizontal de la pantalla para luego avanzar hacia la izquierda en dirección al mono.

Image **tigre →** una variable del tipo Image donde se cargará la imagen del tigre.

Rectangle **rectanguloTigre** → esta variable colabora en poner límites a las coordenadas “x” e “y” donde está la imagen del tigre y ofrece un cierto margen de espacios (ancho y alto) para poder elaborar las colisiones con los otros objetos.

**Métodos**

**constructor (int x, int y)**

Recibe las coordenadas “x” e “y” de aparición del tigre, luego inicializa un rectángulo y carga la imagen del tigre.

**dibujarse (Entorno entorno)**

Recibe un parámetro del tipo Entorno para llamar un método “dibujarImagen” que recibirá los parámetros necesarios (dentro de los cuales se hallan las variables de instancia) para dibujar al tigre (que avanzará hacia la izquierda por medio de otro método. Ver a continuación).

**moverIzquierda (no recibe parámetros)**

Este método realizará el movimiento horizontal hacia la izquierda del tigre que sea creado y dibujado previamente, lográndolo a través de la disminución del valor del “x” de la variable del tipo Point “punto” determinada como propiedad del objeto.

**Rectangle getEspTigre ()**

Este método colabora en obtener los puntos de posición del rectángulo que delimita al tigre en su imagen creada, para confirmar o negar en el main el momento de colisión con la piedra (en caso de darse).

Clase: **Pasto**

Esta clase tiene como objetivo crear la imágen de pasto donde se desarrollarán los eventos del juego, simulando una animación de movimiento del escenario (además de los depredadores que avanzan).

**Variables de instancia**

Point **punto →** este punto contendrá las coordenadas desde las cuales se creará el pasto.

Image **pasto →** una variable del tipo Image donde se cargará la imagen del pasto.

**Métodos**

**constructor (int x, int y)**

Recibe las coordenadas “x” e “y” de aparición del pasto y carga la imagen del mismo.

**dibujarse (Entorno entorno)**

Recibe un parámetro del tipo Entorno para llamar un método “dibujarImagen” que recibirá los parámetros necesarios (dentro de los cuales se hallan las variables de instancia) para dibujar el pasto (que avanzará hacia la izquierda por medio de otro método. Ver a continuación).

**moverIzquierda (no recibe parámetros)**

Este método realizará el movimiento horizontal hacia la izquierda del pasto que sea creado y dibujado previamente, lográndolo a través de la disminución del valor del “x” de la variable del tipo Point “punto” determinada como propiedad del objeto.

Clase: **Arbol**

Esta clase tiene como objetivo crear la imágen del árbol donde se ubicará su plataforma/rama, lugar donde el mono podrá situarse si efectúa el salto correcto.

**Variables de instancia**

Point **punto →** este punto contendrá las coordenadas desde las cuales se creará el árbol.

Image **arbol →** una variable del tipo Image donde se cargará la imagen del pasto.

**Métodos**

**constructor (int x, int y)**

Recibe las coordenadas “x” e “y” de aparición del árbol y carga la imagen del mismo.

**dibujarse (Entorno entorno)**

Recibe un parámetro del tipo Entorno para llamar un método “dibujarImagen” que recibirá los parámetros necesarios (dentro de los cuales se hallan las variables de instancia) para dibujar el árbol (que avanzará hacia la izquierda por medio de otro método. Ver a continuación).

**moverIzquierda (no recibe parámetros)**

Este método realizará el movimiento horizontal hacia la izquierda del árbol que sea creado y dibujado previamente, lográndolo a través de la disminución del valor del “x” de la variable del tipo Point “punto” determinada como propiedad del objeto. Se va moviendo a la par de su plataforma.

Clase: **Plataforma**

Esta clase creará la imagen de la plataforma en los árboles y colaborará con los controles que se hagan al respecto para observar la colisión entre el mono y la propia plataforma (cuando el mono llegue con un salto apropiado).

**Variables de instancia**

Point **punto →** este punto contendrá las coordenadas desde las cuales se creará el árbol.

int **x** → coordenada “x” de la plataforma.

int **y** → coordenada “y” de la plataforma.

int **alto** → alto de la plataforma.

int **ancho** → ancho de la plataforma.

Image **imagenPlataforma →** una variable del tipo Image donde se cargará la imagen del pasto.

Rectangle **rectanguloPlataforma** → esta variable colabora en poner límites a las coordenadas “x” e “y” donde está la imagen de la plataforma y ofrece un cierto margen de espacios (ancho y alto) para poder elaborar las colisiones con los otros objetos.

**Métodos**

**constructor (int x, int y)**

Recibe las coordenadas “x” e “y” de aparición del árbol y carga la imagen del mismo. También inicializa la variable de rectángulo con los valores de las otras variables inicializadas en este mismo método.

**dibujarPlataforma (Entorno entorno)**

Recibe un parámetro del tipo Entorno para llamar un método “dibujarImagen” y un “dibujarRectangulo” que recibirán los parámetros necesarios (dentro de los cuales se hallan las variables de instancia) para dibujar la plataforma y un rectángulo que ayudará con los límites del objeto plataforma (la imagen avanzará hacia la izquierda por medio de otro método. Ver a continuación).

**moverAIzquierda (double velAvanceArbol)**

Este método realizará el movimiento horizontal hacia la izquierda de la plataforma que sea creada y dibujada previamente, lográndolo a través de la disminución del valor del “x” de la variable del tipo Point “punto” determinada como propiedad del objeto. Se va moviendo a la par de su árbol.

**Rectangle getEspPlataforma(no recibe parámetros)**

Este método colabora en obtener los puntos de posición del rectángulo que delimita a la plataforma en su imagen creada, para confirmar o negar en el main el momento de colisión con el mono (en caso de darse).

Clase: **Temporizador**

Esta clase despliega texto en la pantalla que muestra el tiempo de avance en segundos del juego iniciado.

**Variables de instancia**

int **minutos →** un valor entero para medir minutos.

int **segundos →** un valor entero para medir segundos.

int **miliSegundos →** un valor entero para medir milisegundos.

**Métodos**

**constructor (int seg)**

Recibe un parámetro entero e iguala su valor con la variable de instancia “segundos”.

**tiempo (no recibe parámetros)**

Realiza operaciones para medir el paso del tiempo en segundos durante la vigencia del juego.

**Implementación**

**Clase Juego**

package juego;

import java.awt.Color;

//import javax.sound.sampled.Clip;

//import entorno.Entorno;

//import entorno.Herramientas;

//import entorno.InterfaceJuego;

import entorno.\*;

public class Juego extends InterfaceJuego

{

// El objeto Entorno que controla el tiempo y otros

private Entorno entorno;

// otras variables del juego aqui

// Variables y métodos propios de cada grupo

// ...

//fondo

private Fondo fondo = new Fondo();

Pasto pasto []= new Pasto[1];

Mono mono;

private Piedra piedra;

private Tigre tigre;

int gravedadMono;

int piso = 520;

int pisoMono = 520;

Tigre tigres[] = new Tigre[3];

Arbol arbol[]= new Arbol[2];

Plataforma plataforma[]= new Plataforma[1];

int miliSegundos;

int minutos;

int segundos;

Temporizador tempo;

int suma ;

Juego()

{

// Inicializa el objeto entorno

this.entorno = new Entorno(this, "Selva Mono Capuchino - Grupo 08 Elva-Pablo-Mati - v1", 800, 600);

// Inicializa el objeto entorno, pero aun no lo inicia.

mono = new Mono(50, piso);

mono.dibujarse(entorno);

gravedadMono = 0;

miliSegundos = 0;

int minutos = 0;

int segundos = 0;

pasto[0] = null ;

arbol[0] = null ;

arbol[1] = null ;

plataforma[0]= null;

this.piedra = null;

this.tigre = null;

tigres[0] = null;

tigres[1] = null;

tempo = new Temporizador(segundos);

// Inicializar lo que haga falta para el juego

// ...

// El objeto Entorno que controla el tiempo y otros

// ….

Herramientas.loop("Sans.wav");

// Inicia el juego!

this.entorno.iniciar();

}

/\*\*

\* Durante el juego, el método tick() será ejecutado en cada instante y

\* por lo tanto es el método más importante de esta clase. Aquí se debe

\* actualizar el estado interno del juego para simular el paso del tiempo

\* (ver el enunciado del TP para mayor detalle).

\*/

public void tick()

{

// Procesamiento de un instante de tiempo

tempo.tiempo();

fondo.dibujar(entorno, 0);

// //pasto

// //pasto[0]

if (this.pasto[0] == null) {

//anda:

this.pasto[0] = new Pasto(450, piso);

}

// pasto[0]

if( (this.pasto[0] != null) && (pasto[0].punto.x < this.entorno.ancho()- this.entorno.ancho()+ 10.00) ) {

this.pasto[0]=null;

}

//pasto[0]

if(this.pasto[0] != null) {

//pasto

this.pasto[0].dibujarse(entorno);

this.pasto[0].moverIzquierda();

}

// limiteIz = this.entorno.ancho()- this.entorno.ancho()- 50.00);

entorno.cambiarFont("Arial", 26, Color.white);

entorno.escribirTexto("Tiempo: "+tempo.minutos + " : "+tempo.segundos ,500,200 );

if (entorno.sePresiono(entorno.TECLA\_ARRIBA) && mono.sePresiono < 1 ) {

mono.sePresiono+=1;

gravedadMono= 400;

Herramientas.play("Salto.wav");

}

if( piso+gravedadMono > piso && mono.y > 150 && pisoMono+gravedadMono > piso ) {

mono.saltando(10);

gravedadMono-=10;

}else{

if ( mono != null && plataforma[0] != null && mono.colision(plataforma[0].getEspPlataforma()) && mono.sePresiono <= 2 && mono.x <= (plataforma[0].x + (plataforma[0].rectanguloPlataforma.height/2)) ) {

pisoMono = plataforma[0].getEspPlataforma().y - (plataforma[0].rectanguloPlataforma.height) ;

mono.vuelvePiso(pisoMono);

}else {

gravedadMono = 0;

mono.vuelvePiso(piso);

}

}

if( mono.y == piso || mono.y == pisoMono ) {

mono.sePresiono=0;

}

System.out.println( "eje Y monoY: "+mono.y );

if (this.arbol[0] == null) {

this.plataforma[0] = new Plataforma(this.entorno.ancho()+50, 400 );

this.arbol[0] = new Arbol(this.entorno.ancho()+50, piso);

}

//arbol[0]

if(this.arbol[0] != null) {

//arbol

this.arbol[0].dibujarse(entorno);

this.arbol[0].moverIzquierda();

//plataforma

this.plataforma[0].dibujarPlataforma(entorno);

this.plataforma[0].moverAIzquierda(1.5);

}

// System.out.println( "eje x plataforma: "+plataforma[0].getEspPlataforma().x );

// arbol[0]

if( (this.arbol[0] != null) && (arbol[0].punto.x < this.entorno.ancho()- this.entorno.ancho()- 150.00) ) {

this.arbol[0]=null;

this.plataforma[0]=null;

}

//Mono

// System.out.println("gravedad mono: "+gravedadMono);

mono.dibujarse(entorno);

if (this.piedra == null && entorno.sePresiono(entorno.TECLA\_ESPACIO)) {

this.piedra = new Piedra(mono.x, mono.y);

}

if(this.piedra != null) {

this.piedra.dibujarse(entorno);

this.piedra.moverDerecha();

}

if( this.piedra != null && piedra.spawn.x > this.entorno.ancho()+50.00 )

this.piedra=null;

// Prueba end

// tigre

if (this.tigre == null) {

this.tigre = new Tigre(this.entorno.ancho()+50, piso);

}

// tigres[0] = null;

if (this.tigres[1] == null) {

this.tigres[1] = new Tigre(this.entorno.ancho()+50, piso);

}

// tigres[0]

if( (this.tigres[1] != null) && (tigre.punto.x <= 400 || this.tigres[1].punto.x <= 400) ){

this.tigres[1].dibujarse(entorno);

this.tigres[1].moverIzquierda();

}

//

if(this.tigre != null) {

this.tigre.dibujarse(entorno);

this.tigre.moverIzquierda();

}

// tigres[0]

if( (this.tigres[1] != null) && (tigres[1].punto.x < this.entorno.ancho()- this.entorno.ancho()- 50.00) )

this.tigres[1]=null;

if( (this.tigre != null) && (tigre.punto.x < this.entorno.ancho()- this.entorno.ancho()- 50.00) )

this.tigre=null;

// if (tigre != null && piedra != null && piedra.colision(tigre.getSpawn())) {

// tigre=null;

// }

if ( tigre != null && piedra != null && piedra.colision(tigre.getEspTigre()) ) {

tigre=null;

piedra = null;

Herramientas.play("Piedra.wav");

suma = piedra.spawn.x + piedra.rectanguloPiedra.width/2;

}

// System.out.println( "piedra rectang.x: "+ suma );

// piedra.spawn.x + piedra.rectanguloPiedra.width/2

// System.out.println( "eje x tigre: "+tigres[1].rectanguloTigre.x );

}

@SuppressWarnings("unused")

public static void main(String[] args)

{

Juego juego = new Juego();

}

}

**Clase Arbol**

package juego;

import java.awt.\*;

import entorno.\*;

public class Arbol {

Point punto;

private Image arbol;

public Arbol(int x, int y) {

this.punto = new Point(x,y);

this.arbol = Herramientas.cargarImagen("arbol.png");

}

public void dibujarse(Entorno entorno) {

entorno.dibujarImagen(arbol, this.punto.x, 380, 0, 0.35);

}

void moverIzquierda() {

this.punto.x -=1.5;

}

}

**Clase Fondo**

package juego;

import java.awt.\*;

import entorno.\*;

public class Fondo {

private Image pastoFijo;

private Image fondoAzul;

private Point posicion;

public Fondo() {

this.pastoFijo = Herramientas.cargarImagen("pastoFijo-1.png");

fondoAzul = Herramientas.cargarImagen("fondo-celeste.png");

this.posicion = new Point (0,400);

}

public void dibujar(Entorno entorno, int rotacion) {

entorno.dibujarImagen(fondoAzul, 250, 270, rotacion, 2.1);

entorno.dibujarImagen(pastoFijo, 350, 625, rotacion, 0.5);

}

}

**Clase Mono**

package juego;

import java.awt.Color;

import java.awt.Image;

import java.awt.Point;

import java.awt.Rectangle;

import entorno.Entorno;

import entorno.Herramientas;

public class Mono

{

// Variables de instancia

int x;

int y;

double angulo;

Image img1;

int piso = 520;

Point punto;

Rectangle rectanguloMono;

int sePresiono = 0;

int volviendo;

public Mono(int x, int y)

{

this.x = x;

this.y = y;

this.punto = new Point(x,y);

this.rectanguloMono= new Rectangle(this.punto.x, this.punto.y, 40, 60);

img1 = Herramientas.cargarImagen("mono.png");

}

public void dibujarse(Entorno entorno)

{

//entorno.dibujarRectangulo(this.x , this.y , rectanguloMono.width, rectanguloMono.height, 0, Color.black );

entorno.dibujarImagen(img1, this.x, this.y, this.angulo, 0.12);

}

public void saltando(int saltY) {

this.y-=saltY;

}

public void vuelvePiso(int volviendo) {

if( this.y < volviendo ) {

this.y+=10;

}

}

//colision referencia

public boolean colision( Rectangle rectanguloPlat ) {

//if ( (this.spawn.x + (rectanguloPiedra.width/2 ) >= rectanguloDepred.x + ( rectanguloDepred.width/2 ) ) && ( this.spawn.y+ (rectanguloPiedra.height/2) >= rectanguloDepred.y + ( rectanguloDepred.width/2 ) ) ) {

int espacioMonoX = this.x + (rectanguloMono.width/2 );

int espacioPlatX = rectanguloPlat.x - ( rectanguloPlat.width/2 );

int espacioMonoY = this.y + (rectanguloMono.height/2);

int espacioPlatY = rectanguloPlat.y - ( rectanguloPlat.width );

if ( (espacioMonoX > espacioPlatX ) && ( espacioMonoY > espacioPlatY ) ) {

// if ( (rectanguloMono.wih >= espacioPlatX ) && ( espacioMonoY >= espacioPlatY ) ) {

return true;

}

return false;

}

}

**Clase Pasto**

package juego;

import java.awt.\*;

import entorno.\*;

public class Pasto {

Point punto;

private Image pasto;

public Pasto(int x, int y) {

this.punto = new Point(x,y);

this.pasto = Herramientas.cargarImagen("pastoFijo-2.png");

}

public void dibujarse(Entorno entorno) {

entorno.dibujarImagen(pasto, punto.x, 620, 0, 1.5);

}

void moverIzquierda() {

this.punto.x -=1.5;

}

}

**Clase Piedra**

package juego;

import java.awt.\*;

import entorno.\*;

public class Piedra {

private Entorno entorno;

Point spawn;

Rectangle rectanguloPiedra;

private Image piedra;

public Piedra(int x, int y) {

this.spawn = new Point(x,y);

this.rectanguloPiedra= new Rectangle(this.spawn.x, this.spawn.y, 20, 20);

this.piedra = Herramientas.cargarImagen("piedra.png");

}

public void dibujarse(Entorno entorno) {

// entorno.dibujarCirculo(spawn.x , spawn.y , 60.0, Color.white );

entorno.dibujarRectangulo(spawn.x, spawn.y, rectanguloPiedra.width, rectanguloPiedra.height, 0, null);

entorno.dibujarImagen(piedra, this.spawn.x, this.spawn.y, 0, 0.056);

}

void moverDerecha() {

this.spawn.x +=8;

}

public boolean colision( Rectangle rectanguloDepred ) {

//if ( (this.spawn.x + (rectanguloPiedra.width/2 ) >= rectanguloDepred.x + ( rectanguloDepred.width/2 ) ) && ( this.spawn.y+ (rectanguloPiedra.height/2) >= rectanguloDepred.y + ( rectanguloDepred.width/2 ) ) ) {

int espacioPiedraX = this.spawn.x + (rectanguloPiedra.width/2 );

int espacioDepredX = rectanguloDepred.x - ( rectanguloDepred.width/2 );

int espacioPiedraY = this.spawn.y+ (rectanguloPiedra.height/2);

int espacioDepredY = rectanguloDepred.y - ( rectanguloDepred.width/2 );

if ( (espacioPiedraX >= espacioDepredX ) && ( espacioPiedraY >= espacioDepredY ) ) {

return true;

}

return false;

}

}

**Clase Plataforma**

package juego;

import java.awt.Color;

import java.awt.Image;

import java.awt.Point;

import java.awt.Rectangle;

import entorno.Entorno;

import entorno.Herramientas;

public class Plataforma {

// variables de Instancia

Point punto;

int x;

int y;

int alto;

int ancho;

private Image imagenPlataforma;

Rectangle rectanguloPlataforma;

// contructor

public Plataforma(int x, int y) {

this.x = x;

this.y = y;

this.punto = new Point(x,y);

this.ancho = 220;

this.alto = 25;

this.rectanguloPlataforma = new Rectangle(punto.x, punto.y, this.ancho, this.alto);

this.imagenPlataforma = Herramientas.cargarImagen("plataforma.png");

}

public void dibujarPlataforma(Entorno entorno) {

entorno.dibujarRectangulo(x, y, rectanguloPlataforma.width, rectanguloPlataforma.height, 0, Color.black );

entorno.dibujarImagen(imagenPlataforma, this.x, 400, 0.0, 0.4);

}

void moverAIzquierda( double velAvanceArbol ) {

this.x -=velAvanceArbol;

punto.x-=velAvanceArbol;

}

public Rectangle getEspPlataforma() {

rectanguloPlataforma.x = punto.x;

rectanguloPlataforma.y = punto.y;

return this.rectanguloPlataforma;

}

}

**Clase Temporizador**

package juego;

public class Temporizador {

int minutos;

int segundos;

int miliSegundos;

public Temporizador( int seg ){

this.segundos= seg;

}

public void tiempo() {

miliSegundos+=1550;

segundos= miliSegundos/100000;

if( segundos > 59 ) {

minutos+=1;

this.miliSegundos=0;

this.segundos = 0;

if(minutos > 59) {

minutos=0;

}

}

}

}

**Clase Tigre**

package juego;

import java.awt.\*;

import entorno.\*;

public class Tigre {

Point punto;

private Image tigre;

Rectangle rectanguloTigre;

public Tigre(int x, int y) {

this.punto = new Point(x,y);

this.rectanguloTigre = new Rectangle(punto.x, punto.y, 140, 90);

this.tigre = Herramientas.cargarImagen("tigre.png");

}

public void dibujarse(Entorno entorno) {

entorno.dibujarImagen(tigre, this.punto.x, this.punto.y, 0, 0.3);

// entorno.dibujarRectangulo(punto.x, punto.y, rectanguloTigre.width, rectanguloTigre.height, 0, Color.black);

}

void moverIzquierda() {

this.punto.x -=5;

}

public Rectangle getEspTigre() {

rectanguloTigre.x = punto.x;

rectanguloTigre.y = punto.y;

return this.rectanguloTigre;

}

}

**Conclusiones**

El presente proyecto presentó algunas dificultades para su comienzo y en su desarrollo, siendo algunas vinculadas a debates de ideas o metodologías a implementar en el juego final. Otros inconvenientes como se han presentado a lo largo de este informe, han estado vinculados a las formas en que se debía lograr el funcionamiento de algunos bloques de código y/o comportamientos y delimitaciones de los objetos, siendo necesario en algunos momentos escribir y borrar algunas líneas de código para mejorar lo máximo posible la versión final de este proyecto.

Algunas lecciones aprendidas se han suscitado (por ejemplo) en la experiencia en el momento de hacer aparecer la piedra y los enemigos que se mueven por la pantalla, su momento de aparición, sus cambios de valores (sea null o cualquier otro valor necesario), pensar los métodos de cada clase, etc.

Otros momentos de aprendizaje también se han desprendido de la puesta en marcha de la creación de las colisiones entre los distintos objetos, donde también se tuvo que repensar/corregir en los casos que luego se ha obtenido un mejoramiento en el código con respecto a su predecesor.