|  |  |
| --- | --- |
| TRabajo práctico – programación 1  El camino de Gondolf | Grupo 2 - INTEGRANTES  Mateo Balbona Arvigo - matu-2001@hotmail.com.ar  Matias Flores – floresmatias641@gmail.com  Franco Ariel Nuttini Rolon – franconuttini@hotmail.com.ar |

El objetivo del trabajo práctico es programar un videojuego en Java, donde un personaje debe derrotar enemigos mediante hechizos en un mapa con obstáculos. Para lograrlo se crearon distintas clases que crean distintos aspectos del juego, desde el personaje del mago, las rocas que sirven de obstáculos, los enemigos y los hechizos. A continuación, se detalla cada clase y el propósito que cumple.

**Clase Mago**

En esta clase se crea al personaje del mago, se determina su tamaño, movimiento, vida y energía para lanzar hechizos.

Para crear al mago se utiliza un rectángulo, y se utilizan las siguientes variables de instancia:

1. Alto
2. Ancho
3. X
4. Y
5. Margen
6. tamañoMago

Las variables alto, ancho, x e y se utilizan para determinar el tamaño que tendrá el rectángulo que representa al mago, y tamañoMago determina el tamaño que tendrán las imágenes que se superponen al rectángulo. La variable margen es un valor de referencia para determinar cuando el mago llega al margen de la pantalla y de esa manera no sobrepasarlo.

Sobre el rectángulo del mago se dibujan 4 imágenes distintas dependiendo de donde esté mirando, y se determina en las siguientes variables:

1. izq
2. der
3. arriba
4. abajo

La variable direccionHechizo determina el sentido en el que está mirando el mago (arriba, abajo, derecha, izquierda). El nombre quedó del principio del trabajo, cuando los hechizos estaban incluidos en la clase mago.

Las últimas dos variables con energiaMagica y vida, que determinan la cantidad de hechizos que puede lanzar el mago y el total de vida que tiene, respectivamente.

El constructor Mago inicializa las variables necesarias para poder dibujar al mago en la pantalla.

El método dibujar “imprime” en pantalla al mago, considerando primero la variable direccionHechizo, que determina a donde está mirando el mago. Dependiendo del valor de esa variable se dibuja al mago con la imagen correspondiente (arriba, abajo, derecha o izquierda).

El método quitarVida le resta vida al mago cuando colisiona con un murciélago.

Los métodos booleanos dentroLimiteIzquierdo, dentroLimiteDerecho, dentroLimiteSuperior y dentroLimiteInferior sirven para evitar que el mago se salga de la pantalla, comparando un punto del mago (determinado por los métodos limiteDerecho, limiteIzquierdo, limiteSuperior y limiteInferior) con los bordes de la pantalla y evitando que siga avanzando cuando se choquen.

Los métodos moverDerecha, moverIzquierda, moverArriba y moverAbajo son los que determinan cuando avanza el mago en la patalla, dependiendo de donde esté mirando el mago. El movimiento se da restando o sumando de a dos en los valores de X e Y.

**Clase Roca**

Esta clase genera las rocas que hacen de obstáculos en el mapa.

Sus variables de instancia son:

1. X
2. Y
3. Alto
4. Ancho
5. Imagen Piedra

El constructor Roca inicializa las variables de instancia, y el método Roca crea un arreglo de 5 lugares, donde se crean la instancia rocas. Se crean 5 rocas, una por cada lugar del arreglo, con los valores hardcodeados en cada instancia.

El método dibujar toma el arreglo de rocas y los dibuja en pantalla.

Las piedras se dibujan como rectángulos y para evitar que el mago colisione con ellas es necesario ubicar las cuatro puntas del rectángulo, lo que se logra con los métodos:

1. puntoDerechoSuperiorPiedra
2. puntoDerechoInferiorPiedra
3. puntoIzquierdoSuperiorPiedra
4. puntoIzquierdoInferiorPiedra

Teniendo ubicados los puntos de cada roca, los métodos:

1. limiteDerechoEnPiedra
2. limiteIzquierdoEnPiedra
3. limiteSuperiorEnPiedra
4. limiteInferiorEnPiedra

se comparan con un punto del mago y cuando colisionan el mago deja de avanzar. Por ejemplo, el punto izquierdo del mago colisiona con el limite derecho de las piedras, para evitar que el mago siga avanzando se debe determinar que la coordenada X del mago sea menor o igual al punto superior derecho, pero mayor o igual al punto superior izquierdo de la roca, a la vez la coordenada Y del mago debe ser mayor o igual al punto izquierdo superior y menor o igual al punto inferior izquierdo de la piedra.

**Clase Fondo**

La clase fondo establece la imagen de fondo en la pantalla del juego. Sus variables de instancia son X, Y y la imagen fondoPantalla.

El constructor fondo inicializa las variables de instancia.

El método Fondo establece los valores de X e Y.

El método dibujar imprime en pantalla el fondo.

## **Clase HechizoAgua**

En esta clase se define el comportamiento y los atributos del hechizo de tipo agua. El hechizo se puede lanzar hacia un punto y desaparece cuando sale de la pantalla.

### Variables de instancia principales:

1. **x, y**  
   Coordenadas del hechizo en pantalla.
2. **agua**  
   Imagen animada del proyectil de agua en movimiento.
3. **aguaExplosion**  
   Imagen animada de la explosión del hechizo.
4. **activo**  
   Booleano que indica si el hechizo está volando o no.
5. **estadoExplotar**  
   Booleano que indica si debe mostrarse la explosión.
6. **dx, dy**  
   Desplazamientos en x e y calculados según el objetivo (velocidad en cada eje).
7. **velocidad**  
   Cantidad de píxeles que el hechizo avanza por tick.
8. **costoAgua**  
   Energía mágica que cuesta lanzar este hechizo.
9. **diametroExplosion**  
   Tamaño del área de efecto del hechizo.

### Métodos principales:

* **lanzar(...)**  
  Inicializa la posición y la dirección del hechizo, calculando dx y dy para que se dirija hacia el punto objetivo con la velocidad correcta.
* **avanzar()**  
  Actualiza la posición del hechizo sumando dx y dy. Si se sale de los límites del área jugable, se desactiva.
* **dibujar(Entorno e)**  
  Dibuja la imagen del hechizo en movimiento si está activo.
* **dibujarExplosion(Entorno e, Point punto)**  
  Muestra la imagen animada de la explosión en un punto determinado si el hechizo explotó.
* **dibujarAreaExplosion(...)**  
  Dibuja un círculo representando el área de efecto del hechizo.
* **cambiarEstado() / cambiarEstadoExplotar()**  
  Permiten cambiar los estados activo y estadoExplotar según la situación del hechizo.

**Clase HechizoFuego**

Esta clase representa el hechizo de fuego en el juego. Permite lanzar un proyectil animado en dirección a un punto determinado, con una velocidad definida, y genera una explosión con área de efecto al llegar a su destino. A diferencia del hechizo de agua, este hechizo tiene un costo energético más bajo y un área de explosión más grande.

### Variables de instancia principales:

1. **x, y**  
   Posición del hechizo en pantalla.
2. **fuego**  
   Imagen del proyectil en movimiento.
3. **fuegoExplosion**  
   Imagen de la animación al explotar el hechizo.
4. **activo**  
   Determina si el hechizo está en vuelo.
5. **estadoExplotar**  
   Indica si debe mostrarse la explosión.
6. **dx, dy**  
   Dirección y desplazamiento del hechizo (calculado al lanzarse).
7. **velocidad**  
   Velocidad de avance por cada tick.
8. **costoFuego**  
   Costo en energía mágica para lanzar el hechizo (por defecto: 5).
9. **diametroExplosion**  
   Tamaño del área de explosión (por defecto: 70 píxeles de diámetro).

### Métodos principales:

* **lanzar(...)**  
  Calcula la dirección del hechizo desde su origen hasta el punto objetivo y lo activa.
* **avanzar()**  
  Mueve el hechizo sumando dx y dy a su posición. Si sale del área jugable, se desactiva.
* **dibujar(...)**  
  Muestra la imagen del hechizo en movimiento si está activo.
* **dibujarExplosion(...)**  
  Dibuja la explosión animada del hechizo en una coordenada dada si estadoExplotar es true.
* **dibujarAreaExplosion(...)**  
  Representa gráficamente el área de efecto del hechizo con un círculo.
* **cambiarEstado() / cambiarEstadoExplotar()**  
  Permiten desactivar el hechizo o cambiar su estado para activar la explosión.

## **Clase Menu**

La clase Menu es la encargada de representar gráficamente el panel lateral derecho del videojuego. Este menú contiene botones interactivos para que el jugador pueda seleccionar el tipo de hechizo que desea lanzar (fuego o agua), y visualmente destaca qué hechizo está activo mediante un borde de color.

### Variables de instancia principales:

1. **x, y**  
   Coordenadas centrales del menú en pantalla.
2. **menu**  
   Imagen de fondo del menú (archivo menu.png).
3. **ancho, alto**  
   Dimensiones de la imagen del menú.
4. **botonFuegoX, botonFuegoY, botonFuegoAncho, botonFuegoAlto**  
   Coordenadas y tamaño del botón para seleccionar el hechizo de fuego.
5. **botonAguaX, botonAguaY, botonAguaAncho, botonAguaAlto**  
   Coordenadas y tamaño del botón para seleccionar el hechizo de agua.

### Métodos principales:

* **dibujar(Entorno e)**  
  Dibuja el menú completo en pantalla, incluyendo los botones de fuego y agua, sus etiquetas y la imagen de fondo.
* **dibRecFuego(Entorno e)**  
  Dibuja un borde rojo alrededor del botón de fuego para indicar que ese hechizo está actualmente seleccionado.
* **dibRecAgua(Entorno e)**  
  Dibuja un borde azul alrededor del botón de agua para indicar que ese hechizo está seleccionado.
* **detectarClick(int mouseX, int mouseY)**  
  Detecta si el jugador hizo clic dentro del área del botón de fuego o agua:
  + Si el clic fue dentro del botón de fuego → retorna true
  + Si fue dentro del botón de agua → retorna false
  + En cualquier otro caso → también retorna false

El objetivo del trabajo práctico es programar un videojuego en Java, donde un personaje debe derrotar enemigos mediante hechizos en un mapa con obstáculos. Para lograrlo se crearon distintas clases que crean distintos aspectos del juego, desde el personaje del mago, las rocas que sirven de obstáculos, los enemigos y los hechizos. A continuación, se detalla cada clase y el propósito que cumple.

**Clase Murciélago**

Esta clase se encarga principalmente de dibujar al murciélago, hacer que siga al jugador y determinar si un murciélago fue eliminado o no.

Esta clase cuenta con varias variables de entorno que explicaremos:

* private int x, y : Indican en que posición en el eje se encuentra el murciélago.
* private double velocidad: La velocidad a la que el murciélago persigue al jugador.
* private double escala: Parametro que pasamos al dibujar la imagen para achicar o agrandar la imagen.
* private int vida: Vida con la que cuenta el murciélago (No nos dio tiempo a utilzarla)
* private int defensa: La defensa estaba pensada para que sea un porcentaje de daño que mitigue el murciélago, pero no fue implementada.
* private int daño: Daño que causa el murciélago al jugador cuando impacta con el
* private boolean estaVivo: Flag para verificar si sigue vivo o no el murciélago (No es utilizado en el proceso del juego)
* Image imgMurcielago: Cargamos la imagen .gif del murciélago

Metodos de Clase Murcielago

1. public void dibujar(Entorno e)

Metodo utilizado para dibujar en pantalla al Murcielago

1. public void moverHaciaJugador(int posJugadorX, int posJugadorY)

Método que calcula en base a una formula llamada “Formula Euclidiana” utilizada para medir distancias que consiste en calcular la diferencia en X e Y entre el murciélago y el jugador, luego calcula la distancia total entre ambos, calculando lo que llamamos un **vector de dirección** que indica la orientación o el sentido de un movimiento en un espacio (asegurándose que la distancia sea mayor que cero para que no estén en el mismo lugar), una vez que tenemos estos valores simplemente modificamos hacia donde debe moverse el murciélago tanto en X como en Y, y lo multiplicamos por la velocidad que tengamos seteada. Para que el murciélago no se pase y se quede dando vueltas alrededor del mago, lo que hicimos fue verificar si el murciélago estaba muy cerca (que la distancia que queda sea menor a la que va la velocidad) haciendo que el murciélago vaya directamente al jugador y no oscile.

1. Getters y setters

Métodos para obtener y modificar las propiedades de la clase sin romper el encapsulamiento.

**Clase Oleada**

Esta clase funciona fundamentalmente para gestionar todo lo relacionado con las oleadas de enemigos, es decir, que oleada es la ganadora, cuantos enemigos debe generar, el aumento de daño y cantidad de los enemigos que debe haber por oleada, etc.

Esta clase cuenta con varias variables de entorno que explicaremos:

* private int numOleadaActual: Indica el número de oleada actual
* private int murcielagosEnEstaOleada: Indica la cantidad de murciélagos que hay en esta oleada
* private int murcielagosDerrotados: Indica los murciélagos derrotados.
* private int murcielagosGenerados: Los mobs que se han generado
* private int estadoOleada: Flag para determinar el comportamiento de la oleada:
  + ///0 - En espera a siguiente oleada
  + ///1 - Mobs apareciendo y atacando
  + ///2 - Todos los mobs de la oleada generados, esperando a ser todos derrotados
  + ///3 - Oleada finalizada
* private int contadorDescanso: Contador para el tiempo de descanso entre oleadas Expresado en ticks (100 ticks = 1 segundo)
* private int tiempoDescansoEntreOleadas: Auxiliar para verificar que se alcanzó el tiempo de descanso
* private int cantMurcielagosIniciales: En la primera oleada apareceran N murcielagos
* private int incrementoMurcielagosPorOleada: En cada oleada se iran incrementando
* private static int oleadaGanadora = 2: Oleada que tiene que alcanzar el jugador para ganar
* private int dañoBaseMurcielago = 10: Daño inicial en la primera oledada
* private static int incrementoDañoPorOleada = 5: Cuanto aumenta el daño por oleada
* private int dañoActualMurcielago: Auxiliar para ir aumentando el daño por cada oleada

**Metodos de Clase Murcielago**

1. public void iniciarOleada()

Comienza la oleada, setea cuantos murciélagos hay en la oleada, resetea el contador de los enemigos generados y derrotados, inicializa el daño la oleada y lo va aumentando, cambia también el estado de la oleada.

1. public void actualizar(int mobsActivosEnPantalla)

Cambia el estado de la oleada según cuantos mobs se hayan generado, si ya se generaron todos, queda en espera a que todos sean derrotados, si todos fueron derrotados se pone en marcha el tiempo de espera entre oleadas y una vez pasado el tiempo inicia una nueva oleada.

1. public boolean necesitaGenerarEnemigo()

Verifica el estado de la oleada para que, según el estado y la cantidad de enemigos generados, quede en true si necesita generar y en false cuando ya no lo necesite.

1. public void mobGenerado() y public void mobDerrotado()

Utilizadas con un contador para verificar cuantos mobs fueron generados y cuantos mobs fueron derrotados, esto sirve para poder controlar el estado de la oleada.

1. Getters y Setters

Métodos para obtener y modificar las propiedades de la clase sin romper el encapsulamiento.

**Implementación**

A continuación, se incluye todo el código del juego separado por clases:

**CLASE JUEGO**

package juego;

import java.awt.Color;

import java.awt.Image;

import java.awt.Point;

import java.util.Random;

import javax.sound.sampled.Clip;

import entorno.Entorno;

import entorno.Herramientas;

import entorno.InterfaceJuego;

public class Juego extends InterfaceJuego

{

private Entorno entorno;

private boolean espacioPresionado = false;

private boolean enterPresionado = false;

private Point posMouse = new Point(0,0);

private boolean eligeHechizo = false;

private double contadorFuego = 0;

private double contadorAgua = 0;

private Point puntoColision = new Point(0,0);

//PERSONAJES - COSAS

private Mago gondolf;

private Murcielago[] murcielagos;

private Roca rocas;

private Fondo fondo;

private Menu menu;

private Image imagenMenu;

private Image imagenGameOver;

private Image imagenVictoria;

private Oleada gestionadorOleadas;

private HechizoFuego hechizoFuego;

private HechizoAgua hechizoAgua;

//IMAGENES - SONIDOS

private Clip game\_music;

private Clip sonidoVictoria;

private Clip sonidoGameOver;

private boolean sonidoVictoriaReproducido = false;

private boolean juegoGanado = false;

private boolean sonidoGameOverReproducido = false;

private boolean juegoTerminado = false;

private boolean enMenu = true;

///VARIABLES QUE CONTROLAN LA APARICION DE MOBS

private Random random;

private int cantMurcielagosMatados = 0;

private static int *cantMurcielagosTotales* = 50;

private static int *maxMurcielagosPantalla* = 10;

private int intervaloAparicion = 20;

private int contadorAparicion = 0;

///DIMENSIONES DE LA VENTANA DE JUEGO

private int anchoPantalla = 800;

private int altoPantalla = 600;

private int margenPantalla = 50;

private int margenAparicion = 50;

private int vidaMago;

private int energiaMago;

///CONSTRUCTOR

public Juego(){

///INICIAMOS OBJETOS ESENCIALES

this.entorno = new Entorno(this, "Proyecto para TP", 800, 600);

this.gondolf = new Mago (30,45,390,530);

this.rocas = new Roca();

this.fondo = new Fondo();

this.random = new Random();

this.menu = new Menu();

this.hechizoFuego = new HechizoFuego(gondolf.getX(), gondolf.getY(), posMouse);

this.hechizoAgua = new HechizoAgua(gondolf.getX(), gondolf.getY(), posMouse);

this.imagenMenu = Herramientas.*cargarImagen*("imagenes/juego-menu.png");

this.imagenVictoria = Herramientas.*cargarImagen*("imagenes/victoria.png");

this.sonidoVictoria = Herramientas.*cargarSonido*("sonido/sonido3.wav");

this.imagenGameOver = Herramientas.*cargarImagen*("imagenes/game-over.png");

this.sonidoGameOver = Herramientas.*cargarSonido*("sonido/sonido2.wav");

this.game\_music = Herramientas.*cargarSonido*("sonido/sonido1.wav");

this.game\_music.loop(Clip.***LOOP\_CONTINUOUSLY***);

vidaMago = gondolf.getVida();

energiaMago = gondolf.getEnergiaMagica();

///MOBS

this.murcielagos = new Murcielago[*cantMurcielagosTotales*];

////SETEAMOS ARRAY DE MURCIELAGOS EN NULL

for(int i = 0; i < *cantMurcielagosTotales*; i++) {

murcielagos[i] = null;

}

////INICIALIZAMOS GESTOR DE OLEADAS

int tiempoDescanso = 300;

int murcielagosBase = 10;

int incrementoMurcielagos = 2;

this.gestionadorOleadas = new Oleada(tiempoDescanso, murcielagosBase, incrementoMurcielagos);

///SE INICIA EL ENTORNO DEL JUEGO

this.entorno.iniciar();

}

//RELACION TIEMPO Y TICKS

//1 SEGUNDO = 100 TICKS

public void tick()

{

//DIBUJAR LA IMAGEN DE VICTORIA Y PAUSAR SONIDO DE INICIO CUANDO GANA

if (juegoGanado) {

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

if (!sonidoVictoriaReproducido) {

sonidoVictoria.start();

sonidoVictoriaReproducido = true;

}

return;

}

if (juegoTerminado) {

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

if (!sonidoGameOverReproducido) {

sonidoGameOver.start();

sonidoGameOverReproducido = true;

}

return; //COLOCAMOS ARRIBA DE **TODO** PARA QUE NO SE SIGAN DIBUJANDO MAS LOS PERSONAJES

}

///////////////////////////////////////////////////////////////////////////////////

/////////////////////////MENU DE JUEGO INICIO//////////////////////////////////////

if (enMenu) {

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

if (entorno.sePresiono(entorno.TECLA\_ENTER)) {

enMenu = false;

}

if (entorno.sePresiono(entorno.TECLA\_ESCAPE)) {

System.*exit*(0); //

}

return;

}

///////////////////////////////////////////////////////////////////////////////////

Point puntoLimiteDerechoMago = new Point (gondolf.limiteDerecho(), this.gondolf.getY()); //Delimita en un punto el límite derecho del mago

Point puntoLimiteIzquierdoMago = new Point (gondolf.limiteIzquierdo(), this.gondolf.getY()); //Delimita en un punto el límite izquierdo del mago

Point puntoLimiteSuperiorMago = new Point (this.gondolf.getX(), gondolf.limiteSuperior()); //Delimita en un punto el límite superior del mago

Point puntoLimiteInferiorMago = new Point (this.gondolf.getX(), gondolf.limiteInferior()); //Delimita en un punto el límite inferior del mago

/////////////////// DIBUJAR A GONDOLF, HECHIZOS, FONDO Y ROCAS DENTRO DEL MAPA/////////////////////

this.fondo.dibujar(entorno);

this.rocas.dibujar(entorno);

this.menu.dibujar(entorno);

this.gondolf.dibujar(entorno);

///////////////////ACA GONDOLF SE MUEVE,LANZA HECHIZOS/////////////////////

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

gondolf.getDireccion(false); // mirar a la derecha

if (gondolf.dentroLimiteDerecho() && rocas.limiteIzquierdoEnPiedra(puntoLimiteDerechoMago, rocas.rocas)==false) {

if (gondolf.limiteDerecho() + 2 < menu.getBordeIzquierdo()) {

gondolf.moverDerecha();

}

}

}

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

gondolf.getDireccion(true);

if (gondolf.dentroLimiteIzquierdo() && rocas.limiteDerechoEnPiedra(puntoLimiteIzquierdoMago, rocas.rocas)==false) {

this.gondolf.moverIzquirda();

}

}

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

if (gondolf.dentroLimiteSuperior() && rocas.limiteInferiorEnPiedra(puntoLimiteSuperiorMago, rocas.rocas)==false) {

this.gondolf.moverArriba();

}

}

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

if (gondolf.dentroLimiteInferior() && rocas.limiteSuperiorEnPiedra(puntoLimiteInferiorMago, rocas.rocas)==false) {

this.gondolf.moverAbajo();

}

}

///////////////////////ELEGIR QUE HECHIZO LANZAR CON COORDENADAS MOUSE//////////////////////

if (entorno.mousePresente() && entorno.sePresionoBoton(1) && entorno.mouseX() >= 600) {

eligeHechizo = menu.detectarClick(entorno.mouseX(), entorno.mouseY());

}

if (entorno.sePresionoBoton(entorno.BOTON\_IZQUIERDO)) {

posMouse.x = entorno.mouseX();

posMouse.y = entorno.mouseY();

if (posMouse.x >= 600) {

eligeHechizo = menu.detectarClick(posMouse.x, posMouse.y);

} else {

if (eligeHechizo) {

hechizoFuego.lanzar(gondolf.getX(), gondolf.getY(), posMouse.x, posMouse.y);

} else {

hechizoAgua.lanzar(gondolf.getX(), gondolf.getY(), posMouse.x, posMouse.y);

}

}

}

//encuadra el hechizo seleccionado

if (eligeHechizo==false) {

menu.dibRecAgua(entorno);

}else {

menu.dibRecFuego(entorno);

}

hechizoFuego.avanzar();

if (eligeHechizo == true && entorno.sePresionoBoton(entorno.BOTON\_IZQUIERDO) && entorno.mouseX()<600) {

energiaMago -= hechizoFuego.costoFuego();

}

hechizoFuego.dibujar(entorno);

hechizoAgua.avanzar();

if (eligeHechizo == false && entorno.sePresionoBoton(entorno.BOTON\_IZQUIERDO) && entorno.mouseX()<600) {

energiaMago -= hechizoAgua.costoAgua();

}

hechizoAgua.dibujar(entorno);

if(hechizoAgua.getX() == posMouse.x && hechizoAgua.getY() == posMouse.y) {

hechizoAgua.cambiarEstado();

}

if(hechizoFuego.estadoExplotar == true && contadorFuego <= 66.00) {

//hechizoFuego.dibujarAreaExplosion(entorno, puntoColision);

hechizoFuego.dibujarExplosion(entorno, puntoColision);

contadorFuego++;

}else{

hechizoFuego.setEstadoExplotar(false);

contadorFuego = 0;

}

if(hechizoAgua.estadoExplotar == true && contadorAgua <= 20.00) {

//hechizoAgua.dibujarAreaExplosion(entorno, puntoColision);

hechizoAgua.dibujarExplosion(entorno, puntoColision);

contadorAgua++;

}else{

hechizoAgua.setEstadoExplotar(false);

contadorAgua = 0;

}

//ENERGIA

entorno.cambiarFont(null, 30, Color.***RED***);

entorno.escribirTexto("Energía: " + energiaMago, 615, 585);

//VIDA DEL MAGO

entorno.cambiarFont(null, 30, Color.***GREEN***);

entorno.escribirTexto("Vida: " + vidaMago, 635, 550);

//////////////////////////////////////////////////////////////////////////////////////

////////////////////////////GESTION DE OLEADAS///////////////////////////////////////

int mobsActivos = cantMobsActivos();

this.gestionadorOleadas.actualizar(mobsActivos);

if(this.gestionadorOleadas.necesitaGenerarEnemigo()) {

this.contadorAparicion++;

if(this.contadorAparicion >= this.intervaloAparicion) {

if(cantMobsActivos() < *maxMurcielagosPantalla*) {

añadirMurcielagoEnPosicionAleatoria();

for(int i = 0; i < this.*cantMurcielagosTotales*; i++) {

if(this.murcielagos[i] != null && this.murcielagos[i].getDaño() == 10) {

this.murcielagos[i].setDaño(this.gestionadorOleadas.getDañoActualMurcielagos());

System.***out***.println("Daño:" + this.murcielagos[i].getDaño());

}

}

this.gestionadorOleadas.mobGenerado();

System.***out***.println("Mobs Activos:" + cantMobsActivos());

}else {

System.***out***.println("Maxima cantidad de mobs en pantalla alcanzado:" + *maxMurcielagosPantalla*);

}

this.contadorAparicion = 0; ///VUELVE EL CONTADOR A CERO PARA QUE SE CUMPLA EL INTERVALO DE APARICION

}

}

/////////////////////////////////////////////////////////////////////////////////////

chequearColisionesConHechizos(); //LLAMAMOS AL METODO PARA ELIMINAR MURCIELAGOS ANTES DE DIBUJAR

if (hechizoAgua.estadoExplotar == true || hechizoFuego.estadoExplotar == true) {

chequearColisionConExplosion();

}

//////////////////////CICLO PARA DIBUJAR LOS MURCIELAGOS///////////////////////

for (int i = 0; i < *cantMurcielagosTotales*; i++) {

Murcielago m = this.murcielagos[i];

if (m != null) {

m.moverHaciaJugador(this.gondolf.getX(), this.gondolf.getY());

//////////////////////LOGICA DAÑO AL MAGO CUANDO LOS ENEMIGOS CHOCAN CON EL///////////////////////

int margenDistancia = (int)m.getVelocidad();

double auxX = this.gondolf.getX() - m.getX();

double auxY = this.gondolf.getY() - m.getY();

double distanciaActual = Math.*sqrt*(auxX \* auxX + auxY \* auxY);

if (distanciaActual <= margenDistancia) {

this.murcielagos[i] = null;

vidaMago -= m.getDaño();

this.gestionadorOleadas.mobDerrotado();

} else {

m.dibujar(entorno);

}

////////////////////////////////////////////////////////////////////////////////

}

}

////////////////////////////////////////////////////////////////////////////////

//////////////////////TEXTOS RELACIONADOS CON MOBS///////////////////////

entorno.cambiarFont("consola", 20, Color.***WHITE***);

entorno.escribirTexto("Cant. Matados:" + this.cantMurcielagosMatados, 400, 20);

entorno.cambiarFont("consola", 20, Color.***WHITE***);

entorno.escribirTexto("Cant Mobs Oleada:" + this.gestionadorOleadas.getmurcielagosEnEstaOleada(), 400, 40);

entorno.cambiarFont("consola", 20, Color.***WHITE***);

entorno.escribirTexto("Oleada:" + this.gestionadorOleadas.getNumOleadaActual(), 10, 20);

//////////////////////FINALIZAR JUEGO Y MOSTRAR IMAGEN-SONIDO GAME-OVER)/////////////////////

if (this.gestionadorOleadas.getNumOleadaActual()-1 == this.gestionadorOleadas.getOleadaGanadora()) {

juegoGanado = true;

return;

}else if(vidaMago <= 0) {

juegoTerminado = true;

game\_music.stop();

return;

}

}

//////////////////////////////////////////////////////////////////////////////////////////////

private void añadirMurcielagoEnPosicionAleatoria() {

for(int i = 0; i < this.*cantMurcielagosTotales*;i++) {

if(this.murcielagos[i] == null) {

int nuevaX;

int nuevaY;

int anchoJugable = this.anchoPantalla - this.menu.getAncho();

int bordeRandom = this.random.nextInt(4); ///Se elige aleatoriamente un numero que representa los lados

switch(bordeRandom) {

case 0: ///Borde Superior

nuevaX = this.random.nextInt(anchoJugable);

nuevaY = -this.margenAparicion;

break;

case 1: ///Borde inferior

nuevaX = this.random.nextInt(anchoJugable);

nuevaY = this.altoPantalla + this.margenAparicion;

break;

case 2: ///Borde izquierdo

nuevaX = -this.margenAparicion;

nuevaY = this.random.nextInt(this.altoPantalla);

break;

case 3: ///Borde Derecho

nuevaX = anchoJugable;

nuevaY = this.random.nextInt(this.altoPantalla);

break;

default:

nuevaX = -this.margenAparicion;

nuevaY = -this.margenAparicion;

break;

}

this.murcielagos[i] = new Murcielago();

this.murcielagos[i].setX(nuevaX);

this.murcielagos[i].setY(nuevaY);

return;

}

}

}

public int cantMobsActivos() {

int aux = 0;

for(int i = 0; i < *cantMurcielagosTotales*; i++) {

if(this.murcielagos[i] != null) {

aux++;

}

}

return aux;

}

//////////////////////////METODOS MAGO ELIMINA MURCIELAGOS//////////////////////////////

private boolean magoFuegoColisionaCon(Murcielago m) {

return hechizoFuego.activo &&

distancia(m.getX(), m.getY(), hechizoFuego.getX(), hechizoFuego.getY()) < 30;

}

private boolean magoAguaColisionaCon(Murcielago m) {

return hechizoAgua.activo &&

distancia(m.getX(), m.getY(), hechizoAgua.getX(), hechizoAgua.getY()) < 30;

}

private double distancia(int x1, int y1, int x2, int y2) {

return Math.*sqrt*(Math.*pow*(x1 - x2, 2) + Math.*pow*(y1 - y2, 2));

}

private void chequearColisionesConHechizos() {

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

Murcielago murcielago = murcielagos[i];

if (murcielago != null && murcielago.getEstaVivo()) {

if (magoFuegoColisionaCon(murcielago)) {

puntoColision.x = murcielagos[i].getX();

puntoColision.y = murcielagos[i].getY();

murcielagos[i] = null; // lo eliminás del arreglo

cantMurcielagosMatados++;

hechizoFuego.cambiarEstadoExplotar();

hechizoFuego.cambiarEstado();

} else if (magoAguaColisionaCon(murcielago)) {

puntoColision.x = murcielagos[i].getX();

puntoColision.y = murcielagos[i].getY();

murcielagos[i] = null;

hechizoAgua.cambiarEstadoExplotar();

hechizoAgua.cambiarEstado();

cantMurcielagosMatados++;

}

}

}

}

//CUANDO HACE CONTACTO CON LOS VAMPIROS HACE UNA EXPLOSION QUE PERMITE QUE ELIMINE A LOS QUE ESTEN

//A SU ALREDEDOR

private boolean colisionConExplosionFuego(Murcielago m) {

int dx = Math.*abs*(m.getX()-puntoColision.x);

int dy = Math.*abs*(m.getY()-puntoColision.y);

double radio = hechizoFuego.getDiametroExplosion()/2;

if(dx \* dx + dy \* dy < radio \* radio) {

return true;

}

return false;

}

//COLISION AGUA CON VAMPIROS

private boolean colisionConExplosionAgua(Murcielago m) {

int dx = Math.*abs*(m.getX()-puntoColision.x);

int dy = Math.*abs*(m.getY()-puntoColision.y);

double radio = hechizoAgua.getDiametroExplosion()/2;

if(dx \* dx + dy \* dy < radio \* radio) {

return true;

}

return false;

}

private void chequearColisionConExplosion() {

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

Murcielago murcielago = murcielagos[i];

if (murcielago != null && murcielago.getEstaVivo()) {

if (colisionConExplosionFuego(murcielago) == true) {

murcielagos[i] = null; // lo eliminás del arreglo

cantMurcielagosMatados++;

}else if (colisionConExplosionAgua(murcielago)) {

murcielagos[i] = null; // lo eliminás del arreglo

cantMurcielagosMatados++;

}

}

}

}

*@SuppressWarnings*("unused")

public static void main(String[] args)

{

Juego juego = new Juego();

}

}

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**CLASE MAGO**

package juego;

import java.awt.Image;

import entorno.Entorno;

import entorno.Herramientas;

public class Mago {

private int alto;

private int ancho;

private int x;

private int y;

private int margen; // evita que el mago se salga de la pantalla

private double tamañoMago;

private boolean direccion; //IZQUIERDA O DERECHA

//imagenes

private Image izq;

private Image der;

private Image arriba;

private Image abajo;

private String direccionHechizo = "abajo"; // puede ser: "arriba", "abajo", "izquierda", "derecha"

private int energiaMagica = 100; // Energía inicial

private int vida = 100; // Vida inicial del mago

public Mago(int alto, int ancho, int x, int y){

this.x = x;

this.y = y;

this.margen = 20;

this.alto = alto;

this.ancho = ancho;

this.tamañoMago = 0.3; //tamaño de la imagen

this.arriba = Herramientas.cargarImagen("imagenes/mago-arriba.png");

this.abajo = Herramientas.cargarImagen("imagenes/mago-abajo.png");

this.izq = Herramientas.cargarImagen("imagenes/mago-der.png");

this.der = Herramientas.cargarImagen("imagenes/mago-izq.png");

}

//DIBUJAR AL MAGO SEGUN LA POSICION QUE MIRE

public void dibujar(Entorno e) {

if (direccionHechizo.equals("izquierda")) {

e.dibujarImagen(this.der, this.x, this.y, 0, this.tamañoMago);

} else if (direccionHechizo.equals("derecha")) {

e.dibujarImagen(this.izq, this.x, this.y, 0, this.tamañoMago);

} else if (direccionHechizo.equals("arriba")) {

e.dibujarImagen(this.arriba, this.x, this.y, 0, this.tamañoMago);

} else {

e.dibujarImagen(this.abajo, this.x, this.y, 0, this.tamañoMago);

}

}

///////////////////////////////////////////////////////////////////////////

public int getVida() {

return this.vida;

}

public void setVida(int vida) {

this.vida = vida;

}

public void quitarVida(int dañoMurcielago) {

this.setVida(this.getVida() - dañoMurcielago);

}

public boolean dentroLimiteIzquierdo() {

return this.x - this.ancho/2 - 2 > 0;

}

public boolean dentroLimiteDerecho(){

return this.x + this.ancho/2 + 2 < 800;

}

public int limiteDerecho() {

return this.x + this.ancho / 2;

}

public int limiteIzquierdo() {

return this.x - this.ancho / 2;

}

public int limiteSuperior() {

return this.y - this.alto / 2 + 1;

}

public int limiteInferior() {

return this.y + this.alto / 2 ;

}

//////PARA QUE NO SALGA DE LA PANTALLA Y NO PASE POR ABAJO DE MENU////////////

public boolean dentroLimiteSuperior() {

return this.limiteSuperior() > 0 + this.margen;

}

public boolean dentroLimiteInferior() {

return this.limiteInferior() < 600 - this.margen;

}

//////////////////////////////////////////////////////////////////////////////

public void moverDerecha() {

this.x += 2;

this.direccionHechizo = "derecha";

}

public void moverIzquirda() {

this.x -= 2;

this.direccionHechizo = "izquierda";

}

public void moverArriba() {

this.y -= 2;

this.direccionHechizo = "arriba";

}

public void moverAbajo() {

this.y += 2;

this.direccionHechizo = "abajo";

}

public int getAlto() {

return alto;

}

public int getAncho() {

return ancho;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public void getDireccion(boolean direccion) {

this.direccion = direccion;

}

public int getEnergiaMagica() {

return this.energiaMagica;

}

}

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**CLASE ROCA**

package juego;

import java.awt.Color;

import entorno.Entorno;

import entorno.Herramientas;

import java.awt.Image;

import java.awt.Point;

public class Roca {

private int x;

private int y;

private int alto;

private int ancho;

private Image piedra;

Roca[] rocas;

public Roca (Image piedra, int x, int y, int alto, int ancho) {

this.x = x;

this.y = y;

this.alto = alto;

this.ancho = ancho;

this.piedra = Herramientas.cargarImagen("imagenes/piedras.png");

}

Roca(){

this.rocas = new Roca[5];

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

if(i==0) {

rocas[i] = new Roca(piedra, 150,125,90,40);

}

if(i==1) {

rocas[i] = new Roca(piedra, 450,125,90,40);

}

if(i==2) {

rocas[i] = new Roca(piedra, 150,425,90,40);

}

if(i==3) {

rocas[i] = new Roca(piedra, 450,425,90,40);

}

if(i==4) {

rocas[i] = new Roca(piedra, 300,275,90,40);

}

}

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

public int getAlto() {

return alto;

}

public void setAlto(int alto) {

this.alto = alto;

}

public int getAncho() {

return ancho;

}

public void setAncho(int ancho) {

this.ancho = ancho;

}

public Image getPiedra() {

return piedra;

}

public void setPiedra(Image piedra) {

this.piedra = piedra;

}

public Roca[] getRocas() {

return rocas;

}

public void setRocas(Roca[] rocas) {

this.rocas = rocas;

}

public void dibujar(Entorno e) {

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

e.dibujarImagen(rocas[i].piedra, rocas[i].x, rocas[i].y, 0, 0.15);

}

}

//////PUNTOS DIAGONELES DE LAS PIEDRAS//////

public Point puntoDerechoSuperiorPiedra(Roca roca) {

Point punto = new Point(roca.getX() + roca.getAncho()/2, roca.getY() - roca.getAlto()/2);

return punto;

}

public Point puntoIzquierdoSuperiorPiedra(Roca roca) {

Point punto = new Point(roca.getX() - roca.getAncho()/2, roca.getY() - roca.getAlto()/2);

return punto;

}

public Point puntoDerechoInferiorPiedra(Roca roca) {

Point punto = new Point(roca.getX() + roca.getAncho()/2, roca.getY() + roca.getAlto()/2);

return punto;

}

public Point puntoIzquierdoInferiorPiedra(Roca roca) {

Point punto = new Point(roca.getX() - roca.getAncho()/2, roca.getY() + roca.getAlto()/2);

return punto;

}

//////LIMITE DE LAS PIEDRAS PARA QUE EL MAGO NO LAS ATRAVIESE//////

public boolean limiteDerechoEnPiedra(Point punto, Roca [] rocas) {

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

if (punto.x <= puntoDerechoSuperiorPiedra(rocas[i]).x && punto.x >= puntoIzquierdoSuperiorPiedra(rocas[i]).x &&

punto.y >= puntoDerechoSuperiorPiedra(rocas[i]).y && punto.y <= puntoDerechoInferiorPiedra(rocas[i]).y) {

return true;

}

}

return false;

}

public boolean limiteIzquierdoEnPiedra(Point punto, Roca [] rocas) {

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

if (punto.x >= puntoIzquierdoSuperiorPiedra(rocas[i]).x && punto.x <= puntoDerechoSuperiorPiedra(rocas[i]).x &&

punto.y >= puntoIzquierdoSuperiorPiedra(rocas[i]).y && punto.y <= puntoIzquierdoInferiorPiedra(rocas[i]).y) {

return true;

}

}

return false;

}

public boolean limiteSuperiorEnPiedra(Point punto, Roca [] rocas) {

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

if (punto.x <= puntoDerechoSuperiorPiedra(rocas[i]).x && punto.x >= puntoIzquierdoSuperiorPiedra(rocas[i]).x &&

punto.y >= puntoIzquierdoSuperiorPiedra(rocas[i]).y && punto.y <= puntoIzquierdoInferiorPiedra(rocas[i]).y) {

return true;

}

}

return false;

}

public boolean limiteInferiorEnPiedra(Point punto, Roca [] rocas) {

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

if (punto.x <= puntoDerechoInferiorPiedra(rocas[i]).x && punto.x >= puntoIzquierdoInferiorPiedra(rocas[i]).x &&

punto.y <= puntoIzquierdoInferiorPiedra(rocas[i]).y && punto.y >= puntoIzquierdoSuperiorPiedra(rocas[i]).y ) {

return true;

}

}

return false;

}

}

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**CLASE MURCIELAGO**

package juego;

import java.awt.Image;

import entorno.Entorno;

import entorno.Herramientas;

public class Murcielago {

private int x, y;

private double velocidad;

private double escala;

private int vida;

private int defensa;

private int daño;

private boolean estaVivo;

Image imgMurcielago;

public Murcielago() {

this.x = -100;

this.y = -100;

this.velocidad = 2.0;

this.escala = 0.1;

this.vida = 100;

this.defensa = 100;

this.daño = 10;

this.setEstaVivo(true);

imgMurcielago = Herramientas.cargarImagen("imagenes/bat.gif");

}

////////GETTERS Y SETTERS////////////

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

public double getVelocidad() {

return velocidad;

}

public void setVelocidad(double velocidad) {

this.velocidad = velocidad;

}

public double getEscala() {

return escala;

}

public void setEscala(double escala) {

this.escala = escala;

}

public int getVida() {

return vida;

}

public void setVida(int vida) {

this.vida = vida;

}

public int getDefensa() {

return defensa;

}

public void setDefensa(int defensa) {

this.defensa = defensa;

}

public boolean getEstaVivo() {

return estaVivo;

}

public void setEstaVivo(boolean estaVivo) {

this.estaVivo = estaVivo;

}

public int getDaño() {

return daño;

}

public void setDaño(int d) {

this.daño = d;

}

//////////////////////// DIBUJAR MURCIELAGO ///////////////////////

public void dibujar(Entorno e) {

e.dibujarImagen(this.imgMurcielago, this.x, this.y, 0, this.escala);

}

//////////////////////// MOVIMIENTO HACIA EL JUGADOR///////////////////////

public void moverHaciaJugador(int posJugadorX, int posJugadorY) {

/// Calcula la diferencia en X e Y entre el murciélago y el jugador

double x = posJugadorX - this.x;

double y = posJugadorY - this.y;

/// Calcula la distancia total entre el murciélago y el jugador

///OBTENEMOS LA DISTANCIA MAS CORTA EN LINEA RECTA

///ENTRE DOS PUNTOS EN UN MAPA (EN ESTE CASO LA PANTALLA) PERMITIENDONOS SABER CUANTOS "PASOS" DEBE DAR

///EL MURCIELAGO PARA LLEGAR AL JUGADOR COMBINANDO MOVIMIENTOS VERTICALES Y HORIZONTALES.

double distancia = Math.sqrt(x \* x + y \* y);

/// Evita dividir por cero si el murciélago ya está en la posición del jugador

if (distancia > 0) {

/// Crea un vector de direccion y calcula los componentes del vector

/// (un vector de longitud 1 que apunta hacia el jugador)

/// Tiene direccion 1 para que la distancia siempre sea la misma para que apunte perfectamente al jugador

/// y no "cuanta" distancia hay, asi podemos controlarlo con la variable "Velocidad"

double direccionX = x / distancia;

double direccionY = y / distancia;

/// Actualiza la posición del murciélago

/// Multiplica el vector de dirección y lo normaliza con la velocidad seteada

///

this.x += direccionX \* this.velocidad;

this.y += direccionY \* this.velocidad;

///evita que el murciélago "oscile" alrededor del jugador.

///calcula el valor absoluto entre el X e Y del murcielago y el X e Y del jugador

///y compara la distancia que le falta para llegar al jugador y si es verdadero,

///la posicion del murcielga queda igual a la posicion del jugador para que no se sobrepase

///

///Supongamos que la posicion del murcielago es (98,100) y la del jugador es (100,100) y tenemos una velocidad de 5

///la proxima vez que se mueva el murcielago quedaria en (103,100) lo cual se pasaria la posicion X, esto evita que eso pase

if (Math.abs(this.x - posJugadorX) < this.velocidad && Math.abs(this.y - posJugadorY) < this.velocidad) {

this.x = posJugadorX;

this.y = posJugadorY;

}

}

}

//////////////////////// DESACTIVAR MURCIELAGO///////////////////////

public void desactivar() {

this.estaVivo = false;

}

}

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**CLASE OLEADA**

package juego;

public class Oleada {

private int numOleadaActual;

private int murcielagosEnEstaOleada;

private int murcielagosDerrotados;

private int murcielagosGenerados;

private int estadoOleada; ///0 - En espera a siguiente oleada

///1 - Mobs apareciendo y atacando

///2 - Todos los mobs de la oleada generados, esperando a ser todos derrotados

///3 - Oleada finalizada

private int contadorDescanso; ///Expresado en ticks (100 ticks = 1 segundo)

private int tiempoDescansoEntreOleadas;

private int cantMurcielagosIniciales; ///En la primera oleada apareceran N murcielagos

private int incrementoMurcielagosPorOleada; ///y cada oleada se iran incrementando

private static int *oleadaGanadora* = 2; ///Oleada que tiene que alcanzar el jugador para ganar

private int dañoBaseMurcielago = 10; ///Daño inicial en la primera oledada

private static int *incrementoDañoPorOleada* = 5; ///Cuanto aumenta el daño por oleada

private int dañoActualMurcielago;

public Oleada(int tiempoDescansoEntreOleadas, int cantMurcielagosIniciales, int incrementoMurcielagos) {

this.tiempoDescansoEntreOleadas = tiempoDescansoEntreOleadas;

this.cantMurcielagosIniciales = cantMurcielagosIniciales;

this.incrementoMurcielagosPorOleada = incrementoMurcielagos;

this.numOleadaActual = 0;

this.estadoOleada = 0;

this.contadorDescanso = tiempoDescansoEntreOleadas;

this.dañoActualMurcielago = dañoBaseMurcielago;

}

////////GETTERS Y SETTERS////////////

public int getNumOleadaActual() {

return numOleadaActual;

}

public void setNumOleadaActual(int numOleadaActual) {

this.numOleadaActual = numOleadaActual;

}

public int getmurcielagosEnEstaOleada() {

return murcielagosEnEstaOleada;

}

public void setmurcielagosEnEstaOleada(int murcielagosXoleada) {

this.murcielagosEnEstaOleada = murcielagosXoleada;

}

public int getMurcielagosDerrotados() {

return murcielagosDerrotados;

}

public void setMurcielagosDerrotados(int murcielagosDerrotados) {

this.murcielagosDerrotados = murcielagosDerrotados;

}

public int getMurcielagosGenerados() {

return murcielagosGenerados;

}

public void setMurcielagosGenerados(int murcielagosGenerados) {

this.murcielagosGenerados = murcielagosGenerados;

}

public int getEstadoOleada() {

return estadoOleada;

}

public void setEstadoOleada(int estadoOleada) {

this.estadoOleada = estadoOleada;

}

public int getContadorDescanso() {

return contadorDescanso;

}

public void setContadorDescanso(int contadorDescanso) {

this.contadorDescanso = contadorDescanso;

}

public int getTiempoDescansoEntreOleadas() {

return tiempoDescansoEntreOleadas;

}

public void setTiempoDescansoEntreOleadas(int tiempoDescansoEntreOleadas) {

this.tiempoDescansoEntreOleadas = tiempoDescansoEntreOleadas;

}

public int getCantMurcielagosIniciales() {

return cantMurcielagosIniciales;

}

public void setCantMurcielagosIniciales(int cantMurcielagosIniciales) {

this.cantMurcielagosIniciales = cantMurcielagosIniciales;

}

public int getIncrementoMurcielagosPorOleada() {

return incrementoMurcielagosPorOleada;

}

public void setIncrementoMurcielagosPorOleada(int incrementoMurcielagosPorOleada) {

this.incrementoMurcielagosPorOleada = incrementoMurcielagosPorOleada;

}

public int getOleadaGanadora() {

return *oleadaGanadora*;

}

public int getDañoActualMurcielagos() {

return this.dañoActualMurcielago;

}

/////////////////////////////////////

////////////////////////////////////////

public void iniciarOleada() {

this.numOleadaActual++;

System.***out***.println("Inicia oleada:" + this.numOleadaActual);

this.murcielagosEnEstaOleada = this.cantMurcielagosIniciales + (this.numOleadaActual - 1) \* this.incrementoMurcielagosPorOleada;

this.murcielagosGenerados = 0;

this.murcielagosDerrotados = 0;

this.dañoActualMurcielago = this.dañoBaseMurcielago + (this.numOleadaActual - 1) \* this.incrementoMurcielagosPorOleada; ///Aumenta el daño por oleada

this.estadoOleada = 1; ///Generando Mobs

}

public void actualizar(int mobsActivosEnPantalla) {

switch(this.estadoOleada) {

case 0: ///Descanso entre oleadas

this.contadorDescanso--;

if(this.contadorDescanso <= 0) {

iniciarOleada();

}

break;

case 1: ///Genera Mobs hasta que la cantidad Maxima sea alcanzada

if(this.murcielagosGenerados >= this.murcielagosEnEstaOleada) {

this.estadoOleada = 2;

}

break;

case 2: ///Esperando a que los mobs sean derrotados

if(mobsActivosEnPantalla == 0) {

this.estadoOleada = 0;

this.contadorDescanso = this.tiempoDescansoEntreOleadas;

}

break;

case 3: ///La oleada termino

break;

}

}

public boolean necesitaGenerarEnemigo() {

return this.estadoOleada == 1 && this.murcielagosGenerados < this.murcielagosEnEstaOleada;

}

public void mobGenerado() {

this.murcielagosGenerados++;

}

public void mobDerrotado() {

this.murcielagosDerrotados++;

}

}

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**CLASE MENU**

package juego;

import java.awt.Color;

import java.awt.Image;

import entorno.Entorno;

import entorno.Herramientas;

public class Menu {

private int x;

private int y;

private Image menu;

private int ancho;

private int alto;

// Botones para hechizos

private int botonFuegoX, botonFuegoY, botonFuegoAncho, botonFuegoAlto;

private int botonAguaX, botonAguaY, botonAguaAncho, botonAguaAlto;

public Menu() {

this.menu = Herramientas.cargarImagen("imagenes/menu.png");

this.x = 700;

this.y = 300;

this.ancho = menu.getWidth(null);

this.alto = menu.getHeight(null);

// Botón de FUEGO

this.botonFuegoAncho = 150;

this.botonFuegoAlto = 150;

this.botonFuegoX = x - 70;

this.botonFuegoY = y + 70;

// Botón de AGUA

this.botonAguaAncho = 100;

this.botonAguaAlto = 100;

this.botonAguaX = x - 40;

this.botonAguaY = y -80;

}

public int getAncho() {

return ancho;

}

public int getX() {

return x;

}

public int getBordeIzquierdo() {

return x - ancho / 2;

}

public void dibujar(Entorno e) {

// Dibujar botón de fuego

e.dibujarRectangulo(botonFuegoX + botonFuegoAncho / 2, botonFuegoY + botonFuegoAlto

/ 2, botonFuegoAncho, botonFuegoAlto, 0,Color.red);

e.escribirTexto("Fuego", botonFuegoX + 10, botonFuegoY + 25);

// Dibujar botón de agua

e.dibujarRectangulo(botonAguaX + botonAguaAncho / 2, botonAguaY + botonAguaAlto

/ 2, botonAguaAncho, botonAguaAlto, 0, Color.BLUE);

e.escribirTexto("Agua", botonAguaX + 15, botonAguaY + 25);

e.dibujarImagen(this.menu, x, y, 0);

}

public void dibRecAgua(Entorno e) {

e.dibujarRectangulo(botonFuegoX+69, botonFuegoY-25, 165, 5, 0, Color.BLUE);

e.dibujarRectangulo(botonFuegoX+69, botonFuegoY-185, 165, 5, 0, Color.BLUE);

e.dibujarRectangulo(botonFuegoX+150, botonFuegoY-105, 5, 160, 0, Color.BLUE);

e.dibujarRectangulo(botonFuegoX-12, botonFuegoY-105, 5, 160, 0, Color.BLUE);

}

public void dibRecFuego(Entorno e) {

e.dibujarRectangulo(botonFuegoX+69, botonFuegoY-10, 165, 5, 0, Color.red);

e.dibujarRectangulo(botonFuegoX+69, botonFuegoY+140, 165, 5, 0, Color.red);

e.dibujarRectangulo(botonFuegoX+150, botonFuegoY+65, 5, 152, 0, Color.red);

e.dibujarRectangulo(botonFuegoX-12, botonFuegoY+65, 5, 152, 0, Color.red);

}

// Método para detectar si se hizo clic en un botón

public boolean detectarClick(int mouseX, int mouseY) {

if (mouseX >= botonFuegoX && mouseX <= botonFuegoX + botonFuegoAncho &&

mouseY >= botonFuegoY && mouseY <= botonFuegoY + botonFuegoAlto) {

return true;

}

if (mouseX >= botonAguaX && mouseX <= botonAguaX + botonAguaAncho &&

mouseY >= botonAguaY && mouseY <= botonAguaY + botonAguaAlto) {

return false;

}return false;

}

}

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**CLASE HECHIZOFUEGO**

package juego;

import java.awt.Color;

import java.awt.Image;

import java.awt.Point;

import entorno.Entorno;

import entorno.Herramientas;

public class HechizoFuego {

private int x;

private int y;

private Image fuego;

private Image fuegoExplosion;

boolean activo;

boolean estadoExplotar;

private double dx, dy;

private int velocidad = 5;

private int costoFuego;

private int diametroExplosion;

public HechizoFuego(int x, int y, Point objetivo) {

this.x = x;

this.y = y;

this.activo = false;

this.fuego = Herramientas.cargarImagen("imagenes/hechizo-fuego.png");

this.fuegoExplosion = Herramientas.cargarImagen("imagenes/fuego-explosion.gif");

this.estadoExplotar = false;

this.costoFuego = 5;

this.diametroExplosion = 70;

}

public void lanzar(int origenX, int origenY, int destinoX, int destinoY) {

this.x = origenX;

this.y = origenY;

double distancia = Math.sqrt(Math.pow(destinoX - origenX, 2) + Math.pow(destinoY - origenY, 2));

this.dx = velocidad \* (destinoX - origenX) / distancia;

this.dy = velocidad \* (destinoY - origenY) / distancia;

this.activo = true;

}

public void dibujar(Entorno e) {

if (activo) {

e.dibujarImagen(this.fuego, this.x, this.y, 0);

}

}

public void dibujarExplosion(Entorno e, Point punto) {

if (estadoExplotar == true) {

e.dibujarImagen(this.fuegoExplosion, punto.x, punto.y, 0, 1);

}

}

public boolean isEstadoExplotar() {

return estadoExplotar;

}

public void setEstadoExplotar(boolean estadoExplotar) {

this.estadoExplotar = estadoExplotar;

}

public void cambiarEstadoExplotar() {

estadoExplotar = true;

}

public void dibujarAreaExplosion(Entorno e, Point puntoExplosion) {

e.dibujarCirculo(puntoExplosion.x, puntoExplosion.y, diametroExplosion, null);

}

public int getVelocidad() {

return velocidad;

}

public void setVelocidad(int velocidad) {

this.velocidad = velocidad;

}

public int getCostoFuego() {

return costoFuego;

}

public void setCostoFuego(int costoFuego) {

this.costoFuego = costoFuego;

}

public int getDiametroExplosion() {

return diametroExplosion;

}

public void setDiametroExplosion(int diametroExplosion) {

this.diametroExplosion = diametroExplosion;

}

public void avanzar() {

if (!activo) return;

this.x += dx;

this.y += dy;

// Área jugable: ancho < 600, alto entre 0 y 600

if (x < 0 || x > 600 || y < 0 || y > 600) {

this.activo = false;

}

}

public void cambiarEstado() {

this.activo = false;

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

public Image getFuego() {

return fuego;

}

public Image getFuegoExplosion() {

return fuegoExplosion;

}

public void setFuegoExplosion(Image fuegoExplosion) {

this.fuegoExplosion = fuegoExplosion;

}

public void setFuego(Image fuego) {

this.fuego = fuego;

}

public int costoFuego() {

return costoFuego;

}

}

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**CLASE HECHIZOAGUA**

package juego;

import java.awt.Color;

import java.awt.Image;

import java.awt.Point;

import entorno.Entorno;

import entorno.Herramientas;

public class HechizoAgua {

private int x;

private int y;

private Image agua;

private Image aguaExplosion;

boolean activo;

boolean estadoExplotar;

private double dx, dy;

private int velocidad = 5;

private int costoAgua;

private int diametroExplosion;

public HechizoAgua(int x, int y, Point objetivo) {

this.x = x;

this.y = y;

this.activo = false;

this.agua = Herramientas.cargarImagen("imagenes/hechizo-agua.gif");

this.aguaExplosion = Herramientas.cargarImagen("imagenes/agua-explosion.gif");

this.costoAgua = 0;

this.estadoExplotar = false;

this.diametroExplosion = 30;

}

public void lanzar(int origenX, int origenY, int destinoX, int destinoY) {

this.x = origenX;

this.y = origenY;

double distancia = Math.sqrt(Math.pow(destinoX - origenX, 2) + Math.pow(destinoY - origenY, 2));

this.dx = velocidad \* (destinoX - origenX) / distancia;

this.dy = velocidad \* (destinoY - origenY) / distancia;

this.activo = true;

}

public void dibujar(Entorno e) {

if (activo) {

e.dibujarImagen(this.agua, this.x, this.y, 0,0.5);

}

}

public void dibujarExplosion(Entorno e, Point punto) {

if (estadoExplotar == true) {

e.dibujarImagen(this.aguaExplosion, punto.x, punto.y, 0, 0.5);

}

}

public void avanzar() {

if (!activo) return;

this.x += dx;

this.y += dy;

// Área jugable: ancho < 600, alto entre 0 y 600

if (x < 0 || x > 600 || y < 0 || y > 600) {

this.activo = false;

}

}

public int getCostoAgua() {

return costoAgua;

}

public void setCostoAgua(int costoAgua) {

this.costoAgua = costoAgua;

}

public int getDiametroExplosion() {

return diametroExplosion;

}

public void setDiametroExplosion(int diametroExplosion) {

this.diametroExplosion = diametroExplosion;

}

public void dibujarAreaExplosion(Entorno e, Point puntoExplosion) {

e.dibujarCirculo(puntoExplosion.x, puntoExplosion.y, diametroExplosion, null);

}

public boolean isEstadoExplotar() {

return estadoExplotar;

}

public void setEstadoExplotar(boolean estadoExplotar) {

this.estadoExplotar = estadoExplotar;

}

public void cambiarEstadoExplotar() {

estadoExplotar = true;

}

public void cambiarEstado() {

this.activo = false;

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

public Image getAgua() {

return agua;

}

public void setAgua(Image fuego) {

this.agua = agua;

}

public int costoAgua() {

return costoAgua;

}

}

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**CLASE FONDO**

package juego;

import java.awt.Image;

import entorno.Entorno;

import entorno.Herramientas;

public class Fondo {

private int x;

private int y;

private Image fondoPantalla;

public void fondo (Image fondoPantalla, int x, int y) {

this.x = x;

this.y = y;

this.fondoPantalla = Herramientas.cargarImagen("imagenes/fondo-pantalla.png");

}

// Fondo(){

// fondo(this.fondoPantalla, 400, 300);

// }

public void dibujar(Entorno e) {

e.dibujarImagen(this.fondoPantalla, this.x, this.y, 0, 1.8);

}

}

**Conclusiones**

Durante el desarrollo de este trabajo, unos de los desafíos mas grandes fue determinar los invariantes de representación, para que cada clase haga lo que le corresponda y nada más. Al ser un concepto relativamente nuevo para nosotros delimitar con precisión nos llevó a colaborar entre nosotros y discutir sobre que funciones debe realizar cada clase.

Otro desafío fue el sistema de oleadas. Diseñamos una clase Oleada dedicada que gestiona la progresión de la dificultad, controlando el número creciente de enemigos y el aumento de su daño con cada nueva fase, una de las tantas dificultades fue el aumento de daño por oleada, dado que buscabamos que se pudiera crear otro mob que funcionase la clase con ese Mob, por lo tanto decidimos no pasar ningún objeto Murciélago a la clase dificultando más la interacción con el Mob.

También un gran desafío fue la generación de murciélagos, logramos que los enemigos aparecieran de puntos aleatorios en cada uno de los cuatro extremos de la pantalla con la librería randomizadora y seccionando cada parte de los extremos de la pantalla con números del 0 al 4 identificándolos como un extremo por cada número, otro punto complicado fue el intervalo de tiempo en que cada murciélago tenía que aparecer con lo cual se calculó la cantidad de ticks que hacía por segundo el programa y en base a eso hacer contadores y que se reinicien cada que queríamos que se generen mobs.