PROGRAMACIÓN I- PRIMER SEMESTRE 2021

Trabajo Practico: Sakura Ikebana Delivery

Comisión N° 3

Grupo 11

Alumnos:

Nombre	DNI	Legajo	Correo Electrónico
Lautaro Gamen	38827053	38827053/19	lautarog@outlook.com
Abel Fernando Acevedo	28881785	28881785/18	Marzoa3581@gmail.com
Federico Enrique Morales	41331140		fedemorales98@gmail.com

INTRODUCCIÓN

En el presente informe explicamos el desarrollo del juego “Sakura Ikebana Delivery”, el cual consiste en el control de un personaje principal llamado ***Sakura***, que debe entregar los pedidos (ramos de flores) en casas seleccionadas con una flecha, esquivando enemigos y matando los mismos con su arma principal llamada *Rasengan*. Para ello, primero detallaremos las diferentes clases implementadas, con sus respectivas variables y métodos. Luego profundizaremos en los inconvenientes surgidos durante el desarrollo del código y las respectivas soluciones implementadas. Finalmente, daremos una breve conclusión respecto a la resolución de este trabajo.

DESCRIPCIÓN

A continuación, se detalla en general cada clase implementada y sus correspondientes variables de instancia y métodos.

CLASE SAKURA

Modela a nuestro personaje principal, con todos sus atributos y límites con el entorno.

Tipo	Nombre	Descripción
Clase	Sakura	Creación del objeto Sakura.
Variables de instancia	x (int)	Instanciamos la posición x del rectangulo.
	y (int)	Instanciamos la posición y del rectangulo.
	ancho (double)	Instanciamos el ancho del rectangulo.
	alto (double)	Instanciamos la altura del rectangulo.
	Diámetro (double)	Instanciamos el ángulo del rectangulo
Métodos	Constructor	Sakura(x,y,ancho,alto,diametro).
	dibujar()	Dibuja rectángulos e imágenes.
	mover()	Mueve a sakura según las teclas apretadas.
	chocasteBordeSuperior() chocasteBordeInferior() chocasteBordeIzquierdo() chocasteBordeDerecho()	Define el límite de movimiento de sakura.
	noChocasteConVordeVerticalDeCuadra() noChocasteConBordeHorizontalDeCuadra()	Define los limites para no chocar Con las casas.
	disparar()	Crea el objeto Fireball en la clase Sakura.
	Setter and Getter	
	Y1(), y2(), x1(), x2()	posiciones del rectangulo para calcular colisiones.

CLASE CUADRA

Modela todas las cuadras del mapa

Tipo	Nombre	Descripción
Clase	Cuadra	Creacion del objeto cuadra.
Variables de instancia	x(double)	Instanciamos la posición x del rectangulo.
	y (double)	Instanciamos la posición y del rectangulo.
	ancho(double)	Instanciamos la posición ancho del rectangulo.
	alto(double)	Instanciamos la posición alto del rectangulo.
	Cuadra (Image) tierra	Instanciamos la imagen.
Métodos	Constructor Casa()	
	Dibujar()	Dibuja rectángulos e imágenes.
	matrizDeCuadras() Setter and Getter	Llama al constructor y crea los rectángulos que representan las cuadras del juego, a las que luego se va a cargar la imagen

CLASE CASA

Representa todas las casas del mapa

Tipo	Nombre	Descripción
Clase	Casa	Creacion del objeto casa.
Variables de instancia	x(double)	Instanciamos la posición x del rectangulo.
	y (double)	Instanciamos la posición y del rectangulo.
	ancho(double)	Instanciamos la posición y ancho del rectangulo.
	alto(double)	Instanciamos la posición alto del rectangulo.
	Casa (Image)	Instanciamos la imagen.
Métodos	Constructor Casa()	
	Dibujar()	Dibuja rectángulos e imágenes.
	matrizDeCasasVerticalDerecho() matrizDeCasasVerticalIzquierdo() matrizDeCasasHorizontalArriba() matrizDeCasasHorizontalAbajo()	Estos cuatro métodos dibujan las casas En diferentes posiciones.
	Setter and Getter	

CLASE NINJA

Modela los enemigos del juego. Su desplazamiento solo está definido de derecha a izquierda y de arriba hacia abajo

Tipo	Nombre	Descripción
Clase	Ninja	Creación del objeto Ninja.
Variables de instancia	x(double)	Instanciamos la posición x del rectángulo.
	y (double)	Instanciamos la posición y del rectángulo.
	ancho(double)	Instanciamos la posición y ancho del rectángulo.
	alto(double)	Instanciamos la posición alto del rectángulo.
	Cuadra (Image) ninja	Instanciamos la imagen.
Métodos	Constructor Ninja()	
	Dibujar()	Dibuja rectángulos e imágenes.
	mover() moverX()	Movimiento del ninja de derecha a izquierda y arriba hacia abajo.
	xIzquierdo() yTecho() ySuelo() xDerecho()	Posiciones del rectángulo para calcular colisiones.
	colisionNinjaSakura() colisionFireballNinja()	Comprueba si sakura choca Con los ninjas si el disparo, Choca con los ninjas.

CLASE FIREBALL

Modela los disparos de Sakura

Tipo	Nombre	Descripción
Clase	Fireball	Creación del objeto fireball.
	Image disparo	Instanciamos la imagen.
Variables de instancia	x (double)	Instanciamos la posición x del rectangulo.
	y (double)	Instanciamos la posición y del rectangulo.
	ancho (double)	Instanciamos el ancho del rectangulo.
	alto (double)	Instanciamos la altura del rectangulo.
	Diámetro (double)	Instanciamos el ángulo del rectangulo .
Métodos	Constructor	Firebal (x,y,ancho,alto,diametro).
	dibujar()	Dibuja rectángulos e imágenes.
	disparar() disparoDerecha()	Dirección en la que el disparo se mueve.
	Setter and Getter	
	y2(), x2(), y1()	Para calcular colisiones.
	disparar()	Crea el objeto Fireball en la clase Sakura.
	Setter and Getter	
	Y1(), y2(), x1(), x2()	posiciones del rectangulo para calcular colisiones.

CLASE FLECHAS

Modela las flechas que indicar y apuntan la casa en la que Sakura debe realizar la entrega.

Tipo	Nombre	Descripción
Clase	Flecha	Creación del objeto Flecha.
	Image flecha	Instanciamos la imagen.
Variables de instancia	x (double)	Instanciamos la posición x del rectángulo.
	y (double)	Instanciamos la posición y del rectángulo.
	ancho (double)	Instanciamos el ancho del rectángulo.
	alto (double)	Instanciamos la altura del rectángulo.
Métodos	Constructor	Flecha (x, y).
	dibujar()	Dibuja rectángulos e imágenes.
	colisionFlechaSakura()	Comprueba colision sakura flecha.
	xDerecho() xIzquierdo() yTecho() ySuelo()	Para calcular colisiones.

PROBLEMAS ENCONTRADOS EN EL DESARROLLO DEL JUEGO Y SOLUCIONES IMPLEMENTADAS:

- El problema de las cuadras: En un principio los rectángulos que representarían las cuadras del mapa, no estaban pensados con la idea de que fueran objetos. Esto traía varios problemas al código. Por un lado, no se podía lograr ningún tipo de interacción con las cuadras. Por el otro, al no contar con propiedades, tampoco se le podían asignar imágenes. Para solucionarlo, implementamos la clase Cuadra.java, que modela todos los parámetros y limites necesarios para la construcción de cada cuadra. Sin embargo, producto de este inconveniente que no fue solucionado hasta avanzado momento en el proyecto, el tiempo de desarrollo del resto de código se vio afectado, ya que nos vimos obligados a repensar la lógica en la cual estábamos escribiendo el juego.

- Construcción de casas: a fin facilitar la comprensión del código y evitar el inconveniente surgido en el punto anterior, decidimos utilizar variantes de matrices de la clase Cuadras.java, para la construcción de las casas.

- Ancho, alto de Sakura y calles del mapa: Decidir el alto y ancho fue un verdadero problema, ya que, si las propiedades de Sakura coincidían demasiado con el ancho de la calle, provocaba una sensación incomoda en cuanto al movimiento del personaje. Por ello, se optó por reducir el tamaño de Sakura a fin de que el jugador pueda tener un mayor espacio de desplazamiento en las calles. Como contrapartida, esto nos sumó otro problema en cuanto a las flechas, detallado en el siguiente punto.

- Flechas o Lugar de entrega de flores: Recordemos que esta clase modela la flecha que apunta a la casa en la cual Sakura debe entregar las flores. Logramos resolver el problema de recolección y entrega de flores; reemplazando los métodos que calculaban la coincidencia de Sakura en los puntos X e Y con las flechas, por métodos que modelan la colisión de nuestro personaje con cada uno de nuestros objetivos.

CONCLUSION

Finalmente podemos concluir que la elaboración del juego “Sakura Ikebana Delivery” nos ayudo a poner en practica varios de los conocimientos adquiridos en clase, enfrentándonos a problemas reales en el desarrollo de código y trabajo en equipo.

##

Clase Juego

##

```
public class Juego extends InterfaceJuego
{

	private static final String[] UNUSED = null;

	// El objeto Entorno que controla el tiempo y otros
	private Entorno entorno;

	// Variables y métodos propios de cada grupo
	private Clip musiquita;
	//Imagenes
	private Image imagenFondo;
	private Image fueguillos;
	private Image gameOver;
	private Image Corazon;
	private Sakura sakura;
	//arreglos y matriz
	private Fireball[] fireballs;
	private Ninja [] ninjas ;
	private Ninja [] ninjas2 ;
	private Cuadra[][] cuadras;
	private Casa [][] casasVerticalDerecho;
	private Casa [][] casasVerticalIzquierdo;
	private Casa [][] casasHorizontalArriba;
	private Casa [][] casasHorizontalAbajo;
	private int direccion=1;
	private int unaBola;
	private int maxFire;
	private int vidasTotal;
	private int cantidadNinjas;
	Random rand = new Random();
	private boolean siChoco;
	private int puntaje;
	private int ninjasEliminados;
	private int time;
	private boolean esInmune;
	private double ticks;
	private double      velocidad;
	private boolean izquierda;
	private boolean derecha;
	private boolean arriba;
	private boolean abajo;
	private Flecha[] flecha;//AGREGADO::::
	private static Cuadra[] cuadras; //AGREGADO::::
	private Cuadra cuadraElegida; //AGREGADO::::
// Crea un array con posiciones nulas del disparo
	public Fireball[] fireListNull(int cantidad) {
```

```
            Fireball[] fuegos = new Fireball[cantidad];
            Arrays.fill(fuegos, null);
            return fuegos;
}
// Determina la altura del Piso por donde salen los ninjas del eje vertical
      double damePisoAlto() {
            double piso = (this.entorno.alto() - this.entorno.alto());
            return piso;
      }

      double damePisoAncho() {  // Ancho de la pantalla
            double piso = (this.entorno.ancho());
            return piso;
      }

// Creo ninjas en y
public Ninja dameNinjasEnY(double separadorX, double separadorY) {

      Ninja ladoVertical= new Ninja(148 + separadorX, damePisoAlto() +
      separadorY , 35 , 35 ,velocidad);

            return ladoVertical;
}

// Creo ninjas en x

public Ninja dameNinjasEnX(double separaX, double separaY) {

      Ninja ladoHorizontal= new Ninja(damePisoAncho() + separaX,separaY ,35
      ,35 ,velocidad);

            return ladoHorizontal;
}

// Arreglo de objeto ninja q salen del eje x 600
public Ninja [] dameNinjaXList(int num) {
      Ninja [] peloton= new Ninja[num];
      double separaX = 400;
      double separaY = 236;

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

            peloton[i] = dameNinjasEnX(separaX, separaY);

            separaX = separaX + (rand.nextInt(2) - 50);
            separaY = separaY + 256;
            }
      return peloton;
      }

// Arreglo de objeto ninja q salen del eje y 0
public Ninja [] dameNinjaList(int num) {
      Ninja [] peloton= new Ninja[num];
      double separadorX = 0;
      double separadorY = -90;

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

            peloton[i] = dameNinjasEnY(separadorX, separadorY);
            separadorX = separadorX + 503;
```

```
                separadorY = separadorY + (rand.nextInt(2) - 100);
        }
        return peloton;
}

// Comprueba si los ninjas murieron
        public boolean todosMuertos(Ninja[] lista) {
                for (Ninja soldado : lista) {
                        if(soldado !=null) {
                                return false;
                        }
                }
                return true;
        }

// Determina la altura del Corazon
double posicionCorazon() {
        double piso = (this.entorno.alto() - this.entorno.alto() / 6) - 30;
                return piso;
        }

//              SELECCIONA CASA PARA DEJAR EL RAMO
                public Cuadra elegirCuadra() {   //AGREGADO::::
                        Random cuadraElegida = new Random();
                        int aleatorio = cuadraElegida.nextInt(20);

                        System.out.println(aleatorio);
                    return cuadras[aleatorio];
                }



        public void crearCuadras() {     //AGREGADO::::

        //creo cuadrados "pasto" en cada posición del arreglo cuadra
                        Juego.cuadras[0] = new Cuadra(64, 44, 128, 88);
                        Juego.cuadras[1] = new Cuadra(232, 44, 128, 88);
                        Juego.cuadras[2] = new Cuadra(400, 44, 128, 88);
                        Juego.cuadras[3] = new Cuadra(568, 44, 128, 88);
                        Juego.cuadras[4] = new Cuadra(736, 44, 128, 88);
                        Juego.cuadras[5] = new Cuadra(64, 172, 128, 88);
                        Juego.cuadras[6] = new Cuadra(232, 172, 128, 88);
                        Juego.cuadras[7] = new Cuadra(400, 172, 128, 88);
                        Juego.cuadras[8] = new Cuadra(568, 172, 128, 88);
                        Juego.cuadras[9] = new Cuadra(736, 172, 128, 88);
                        Juego.cuadras[10] = new Cuadra(64, 300, 128, 88);
                        Juego.cuadras[11] = new Cuadra(232, 300, 128, 88);
                        Juego.cuadras[12] = new Cuadra(400, 300, 128, 88);
                        Juego.cuadras[13] = new Cuadra(568, 300, 128, 88);
                        Juego.cuadras[14] = new Cuadra(736, 300, 128, 88);
                        Juego.cuadras[15] = new Cuadra(64, 428, 128, 88);
                        Juego.cuadras[16] = new Cuadra(232, 428, 128, 88);
                        Juego.cuadras[17] = new Cuadra(400, 428, 128, 88);
                        Juego.cuadras[18] = new Cuadra(568, 428, 128, 88);
                        Juego.cuadras[19] = new Cuadra(736, 428, 128, 88);
                        Juego.cuadras[20] = new Cuadra(64, 556, 128, 88);
                        Juego.cuadras[21] = new Cuadra(232, 556, 128, 88);
                        Juego.cuadras[22] = new Cuadra(400, 556, 128, 88);
                        Juego.cuadras[23] = new Cuadra(568, 556, 128, 88);
                        Juego.cuadras[24] = new Cuadra(736, 556, 128, 88);
```

```
    }
    public void dibujarCuadras() {
        for (int i=0; i < cuadras.length; i++)
            Juego.cuadras[i].dibujar(entorno);
    }



Juego()
    {

// Inicializa el objeto entorno
this.entorno = new Entorno(this, "Sakura Ikebana Delivery - Grupo 11 - v1",
800, 600);

// Inicializar lo que haga falta para el juego

//musiquita = Herramientas.cargarSonido("prelude.wav");   //Musiquita 1
musiquita = Herramientas.cargarSonido("gravity-falls.wav");//Musiquita 2


musiquita.loop(Clip.LOOP_CONTINUOUSLY);


//IMAGENES
this.imagenFondo = Herramientas.cargarImagen("fondo-3.gif");
this.fueguillos = Herramientas.cargarImagen("fireball1.png");
this.Corazon = Herramientas.cargarImagen("corazon.png");
this.gameOver = Herramientas.cargarImagen("gameOver.gif");


// SE ASIGNA VALORES A LA VARIABLE  FIREBALL

maxFire = 1;
unaBola = 0;
fireballs = fireListNull(maxFire);

time = 3;// TIEMPO DE INMUNIDAD DESPUES DE SUFRIR DAÑO
cantidadNinjas = 2; // CANTIDAD DE NINJAS
this.ninjas = dameNinjaList(cantidadNinjas);//CREA UN ARREGLO DE NINJAS EN EL
EJE Y
this.ninjas2 = dameNinjaXList(cantidadNinjas);//CREA UN ARREGLO DE NINJAS EN
EL EJE X
esInmune = false;   // PARA CONTROLAR LA INMUNIDAD LUEGO DE UNA COLISION



puntaje = 0;              // PUNTAJES ACTUALES
ninjasEliminados = 0;
vidasTotal = 3;           // CANTIDAD DE VIDAS INICIALES

// Crea los objetos
this.sakura = new Sakura(400, 365, 25, 25, 0);

this.cuadras = new Cuadra[5][5];
this.cuadras = Cuadra.matrizDeCuadras();
//CREA OBJETOS CASAS
//VERTICAL DERECHO
this.casasVerticalDerecho = new Casa[5][4];
```

```
this.casasVerticalDerecho = Casa.matrizDeCasasVerticalDerecho();

//VERTICAL IZQUIERDO
this.casasVerticalIzquierdo = new Casa[5][4];
this.casasVerticalIzquierdo =  Casa.matrizDeCasasVerticalIzquierdo();

//HORIZONTAL ARRIBA
this.casasHorizontalArriba = new Casa[4][5];
this.casasHorizontalArriba = Casa.matrizDeCasasHorizontalArriba();

//HORIZONTAL ABAJO
this.casasHorizontalAbajo = new Casa[4][5];
this.casasHorizontalAbajo = Casa.matrizDeCasasHorizontalAbajo();


siChoco = false;    // Controla si chocan a sakura
ticks = 3;          // se usa para retardar algunos sucesos

izquierda = false;  // Booleanos para el movimiento del disparo
derecha = false;
arriba = false;
abajo = false;

this.objetivosRestantes = Flecha.objetivosRestantes();//llamada a numero
regresivo
//CREO OBJETOS FLECHA
this.flecha1 = new Flecha(145,44);            //pos(x,y) predefinida
this.flecha2 = new Flecha(650,300);           //pos(x,y) predefinida

this.flecha3 = new Flecha(483,556);           //pos(x,y) predefinida
this.flecha4 = new Flecha(315,172);           //pos(x,y) predefinida


// Inicia el juego!
this.entorno.iniciar();
}

public void tick()
        {
// Procesamiento de un instante de tiempo

//     BLOQUE PARA LAS IMAGENS DE FONDO

// CONDICIONAL QUE SE EJECUTA MIENTRAS LAS VIDAS SEAN MAYORES A CERO
if(vidasTotal > 0) {

      this.entorno.dibujarImagen(imagenFondo, 400, 300, 0, 1);
      this.sakura.dibujar(entorno);


if (this.objetivosRestantes > 0) {

//SI NO SE CAPTURO NINGUNA FLECHA (ya que sakura esta recorriendo el mapa)
if ( ( this.sakura.getX() != this.flecha1.getX()) && (this.sakura.getY()) !=
this.flecha1.getY())|| ((this.sakura.getX() != this.flecha2.getX()) &&
(this.sakura.getY()) !=  this.flecha2.getY())  || ( this.sakura.getX() !=
this.flecha3.getX()) && (this.sakura.getY()) !=  this.flecha3.getY())
|| ((this.sakura.getX() != this.flecha4.getX()) && (this.sakura.getY()) !=
this.flecha4.getY()) )
                        {
```

```
//SI LA PRIMERA FLECHA NO FUE CAPTURADA
      if (this.flecha1.getX() != 0 && this.flecha1.getY() != 0) {
      //DIBUJA
            this.flecha1.dibujar(entorno);
}
//SI LA PRIMERA FLECHA FUE CAPTURADA(=0),
//PERO NO SE CAPTURO LA SEGUNDA FLECHA(!=0)->no esta escondida
      if ((this.flecha1.getX() == 0 && this.flecha1.getY() == 0) &&

            this.flecha2.getX() != 0 && this.flecha2.getY() != 0) {
            //DIBUJA
            this.flecha2.dibujar(entorno);
      }                                    }

//SI LA PRIMERA Y SEGUNDA FLECHA FUERON CAPTURADAS(=0)->estan escondidas,
PERO NO SE CAPTURO LA TERCERA FLECHA(!=0)->no esta escondida
      if ((this.flecha1.getX() == 0 && this.flecha1.getY() == 0) &&
(this.flecha2.getX() == 0 && this.flecha2.getY() == 0)) &&
this.flecha3.getX() != 0 && this.flecha3.getY() != 0) {
            //DIBUJA
            this.flecha3.dibujar(entorno);
}


//SI LA PRIMERA, SEGUNDA Y TERCER FLECHA FUERON CAPTURADAS(=0)->estan
escondidas, PERO NO SE CAPTURO LA TERCERA CUARTA(!=0)->no esta escondida
if ((this.flecha1.getX() == 0 && this.flecha1.getY() == 0) &&

      (this.flecha2.getX() == 0 && this.flecha2.getY() == 0) &&
      (this.flecha3.getX() == 0 && this.flecha3.getY() == 0)) &&
      this.flecha4.getX() != 0 && this.flecha4.getY() != 0) {
            //DIBUJA
            this.flecha4.dibujar(entorno);
}

//SINO, NO TE VAS A DIBUJAR (Por el momento...)


      //SI SE CAPTURA LA PRIMERA FLECHA
if ((this.sakura.getX() == this.flecha1.getX()) && (this.sakura.getY() ==
this.flecha1.getY())) {


//ACTUALIZA PARAMETROS Y DIBUJA
this.objetivosRestantes -= 1;    //objetivosRestantes = 3

//ESCONDE FLECHA1
this.flecha1.setX(0);
this.flecha1.setY(0); }

//SI SE CAPTURA LA SEGUNDA FLECHA
if ((this.sakura.getX() == this.flecha2.getX()) && (this.sakura.getY() ==
this.flecha2.getY())) {


//ACTUALIZA PARAMETROS Y DIBUJA
this.objetivosRestantes -= 1;    //objetivosRestantes = 2

//ESCONDE FLECHA1
```

```
this.flecha2.setX(0);
this.flecha2.setY(0);

}


//SI SE CAPTURA LA TERCERA FLECHA
if ( (this.sakura.getX() == this.flecha3.getX()) && (this.sakura.getY() ==
this.flecha3.getY()) ) {


//ACTUALIZA PARAMETROS Y DIBUJA
this.objetivosRestantes -= 1;    //objetivosRestantes = 1

//ESCONDE FLECHA1
this.flecha3.setX(0);
this.flecha3.setY(0);

}

//SI SE CAPTURA LA CUARTA FLECHA
if ( (this.sakura.getX() == this.flecha4.getX()) && (this.sakura.getY() ==
this.flecha4.getY()) ) {


//ACTUALIZA PARAMETROS Y DIBUJA
this.objetivosRestantes -= 1;    //objetivosRestantes = 1

//ESCONDE FLECHA1
this.flecha4.setX(0);
this.flecha4.setY(0);

}


}


//*****************************************
//RECORRO LAS CUADRAS PARA PODER DIBUJARLAS
for (int i = 0; i < 5; i++) {
      for (int j = 0; j < 5; j++) {
            this.cuadras[i][j].dibujar(entorno);
      }
}


//*****************************************
//RECORRO CASAS VERTICALES DERECHO PARA PODER DIBUJARLAS
for (int i = 0; i < 5; i++) {
      for (int j = 0; j < 4; j++) {

            this.casasVerticalDerecho[i][j].dibujar(entorno);
      }
}

//RECORRO CASAS VERTICALES IZQUIERDO PARA PODER DIBUJARLAS
for (int i = 0; i < 5; i++) {
      for (int j = 0; j < 4; j++) {
            this.casasVerticalIzquierdo[i][j].dibujar(entorno);
      }
}
```

```
//RECORRO CASAS HORIZONTAL ARRIBA PARA PODER DIBUJARLAS
for (int i = 0; i < 4; i++) {
	for (int j = 0; j < 5; j++) {

	this.casasHorizontalArriba[i][j].dibujar(entorno);
	}
}

//RECORRO CASAS HORIZONTAL ABAJO PARA PODER DIBUJARLAS
	for (int i = 0; i < 4; i++) {
		for (int j = 0; j < 5; j++) {

		this.casasHorizontalAbajo[i][j].dibujar(entorno);
		}
	}



// MOVIMIENTO SAKURA PARA Q NO CHOQUE LAS CASAS //
//SI NO CHOCA CON LOS BORDES VERTICALES DE CUADRA, NI CON EL ENTORNO; PERMITE
MOV DE SAKURA
			if (this.sakura.noChocasteConBordeVerticalDeCuadra()) {

				if
((this.entorno.estaPresionada(this.entorno.TECLA_ARRIBA) &&
!this.sakura.chocasteBordeSuperior())) {
					this.sakura.mover("arriba");
				}
			}

//SI NO CHOCA CON LOS BORDES VERTICALES DE CUADRA, NI CON EL ENTORNO; PERMITE
MOV DE SAKURA
			if (this.sakura.noChocasteConBordeVerticalDeCuadra()) {

				if
((this.entorno.estaPresionada(this.entorno.TECLA_ABAJO) &&
!this.sakura.chocasteBordeInferior())) {
					this.sakura.mover("abajo");
				}


			}

//SI NO CHOCA CON LOS BORDES HORIZONTALES DE CUADRA, NI CON EL ENTORNO;
PERMITE MOV DE SAKURA
			if (this.sakura.noChocasteConBordeHorizontalDeCuadra()) {
				if
((this.entorno.estaPresionada(this.entorno.TECLA_IZQUIERDA)&&
!this.sakura.chocasteBordeIzquierdo())) {
					this.sakura.mover("izquierda");
				}
			}

//SI NO CHOCA CON LOS BORDES HORIZONTALES DE CUADRA, NI CON EL ENTORNO;
PERMITE MOV DE SAKURA
	if (this.sakura.noChocasteConBordeHorizontalDeCuadra()) {
		if ((this.entorno.estaPresionada(this.entorno.TECLA_DERECHA)&&
		!this.sakura.chocasteBordeDerecho())) {
					this.sakura.mover("derecha");
				}
```

```
                }


//#########################################################
//                  BLOQUE NINJAS                        //
//#########################################################



//     NINJA QUE APARECEN DE ARRIBA
for (int i = 0; i < ninjas.length; i++) {
      if (ninjas[i] != null) {
            this.ninjas[i].mover();
//Comprueba colision ninjas sakura
      if(this.ninjas[i].colisionNinjaSakura(this.sakura.x2(),

            this.sakura.y2(), this.sakura.y1())){
                  siChoco = true;


      }
      ninjas[i].dibujar(this.entorno);

}

if (unaBola > 0) {
for (int j = 0; j < fireballs.length; j++) {
      if(this.fireballs[j] !=null && this.ninjas[i] !=null) {
      //Comprueba colision ninjas en eje y con fireball
      if(this.ninjas[i].colisionFireballNinja(this.fireballs[j].x2(),

            this.fireballs[j].y2(), this.fireballs[j].y1())) {
                        this.fireballs[j] = null;

                        unaBola -=1;
                        this.ninjas[i] = null;

                        Herramientas.play("burning.wav");
                        //     puntaje = puntaje + 5;
                        ninjasEliminados ++;
                  }
            }
      }
}


if(this.ninjas[i] != null && this.ninjas[i].getY() > 900) {
      this.ninjas[i] = null;
}


      }


if(todosMuertos(ninjas)) {       //CONTROLA SI TODOS LOS SOLDADOS MURIERON DE
SER ASI CREA UNA NUEVA LISTA
      ninjas = dameNinjaList(cantidadNinjas);

}
```

```
// NINJAS QUE APARECEN DE DERECHA A IZQUIERDA
for (int i = 0; i < ninjas2.length; i++) {
	if (ninjas2[i] != null) {
		this.ninjas2[i].moverEjeX();
//Comprueba colision ninjas sakura

	if(this.ninjas2[i].colisionNinjaSakura(this.sakura.x2(),

		this.sakura.y2(), this.sakura.y1())){
			siChoco = true;


	}
	ninjas2[i].dibujar(this.entorno);

}



if(unaBola > 0) {
for (int j = 0; j < fireballs.length; j++) {
	if(this.fireballs[j] !=null && this.ninjas2[i] !=null) {
//Comprueba colision ninjas fireballs

	if(this.ninjas2[i].colisionFireballNinja(this.fireballs[j].x2(),

		this.fireballs[j].y2(), this.fireballs[j].y1())) {
			this.fireballs[j] = null; // Si hubo colision elimino
fireball

	unaBola -=1;
	this.ninjas2[i] = null;		// Si hubo colision elimino

	Herramientas.play("burning.wav");
	//	puntaje = puntaje + 5;
	ninjasEliminados ++;
		}
	}
}
}



	if(this.ninjas2[i] != null && this.ninjas2[i].getX() < -200) {
		this.ninjas2[i] = null;
	}
}


if(todosMuertos(ninjas2)) {	//CONTROLA SI TODOS LOS SOLDADOS MURIERON DE
SER ASI CREA UNA NUEVA LISTA
	ninjas2 = dameNinjaXList(cantidadNinjas);

}



//#################################################################
//			BLOQUE DE CONTROL DE FIREBALL
//#################################################################
```

```
if (this.entorno.sePresiono(entorno.TECLA_ESPACIO) && unaBola < maxFire) {
	Herramientas.play("fireball.wav");
	unaBola = unaBola + 1;
	int disponible = 0;
	if(disponible < 1) {
		for (int i = 0; i < fireballs.length; i++) {
			if(fireballs[i] == null && disponible == 0) {
				fireballs[i] = this.sakura.disparar(); //Llamo a
disparo desde el objeto sakura

				disponible +=1;
			}

		}
		disponible = 0;
}



//preguntamos la direccion en la que esta apuntando el personaje principal
if(entorno.estaPresionada(this.entorno.TECLA_ESPACIO) &&
entorno.estaPresionada(this.entorno.TECLA_ARRIBA))
	{
		arriba = true;
		abajo = false;
		derecha = false;
		izquierda = false;
		this.direccion = 1;
	}

else if(entorno.estaPresionada(this.entorno.TECLA_ESPACIO) &&
entorno.estaPresionada(this.entorno.TECLA_DERECHA))
	{
		arriba = false;
		abajo = false;
		derecha = true;
		izquierda = false;
		this.direccion = 2;
	}



else if(entorno.estaPresionada(this.entorno.TECLA_ESPACIO) &&
entorno.estaPresionada(this.entorno.TECLA_ABAJO))
				{
					arriba = false;
					abajo = true;
					derecha = false;
					izquierda = false;
					this.direccion = 3;
				}

else if(entorno.estaPresionada(this.entorno.TECLA_ESPACIO) &&
entorno.estaPresionada(this.entorno.TECLA_IZQUIERDA))
				{
					arriba = false;
					abajo = false;
					derecha = false;
					izquierda = true;
					this.direccion = 4;
```

```
                }

else if(entorno.estaPresionada(this.entorno.TECLA_ESPACIO))
                {
                        arriba = false;
                        abajo = false;
                        derecha = false;
                        izquierda = false;
                        this.direccion = 5;
                }



if(unaBola > 0) {
// Disparamos en la direccion que camina el personaje
      for (int i = 0; i < fireballs.length; i++) {
            if(fireballs[i] !=null) {
                  if(derecha == true && izquierda == false && arriba ==
            false && abajo == false) {
                        fireballs[i].disparar(this.direccion);
                        fireballs[i].dibujar(this.entorno);

                }
            if(izquierda == true && derecha == false && arriba == false &&
             abajo == false) {

                  fireballs[i].disparar(this.direccion);

                  fireballs[i].dibujar(this.entorno);

            }
if(arriba == true && derecha == false && izquierda == false && abajo ==
false) {

      fireballs[i].disparar(this.direccion);

      fireballs[i].dibujar(this.entorno);

}
if(abajo == true && derecha == false && izquierda == false && arriba ==
false) {

      fireballs[i].disparar(this.direccion);

      fireballs[i].dibujar(this.entorno);

}
if(abajo == false && derecha == false && izquierda == false && arriba ==
false) {

      fireballs[i].disparar(this.direccion);

      fireballs[i].dibujar(this.entorno);

}

// Si el fireball pasa los limetes maximos y minimos de la pantalla se
elimina
```

```
if(fireballs[i].getX() > 800 || fireballs[i].getY() < 0 ||
fireballs[i].getX() < 0 || fireballs[i].getY() > 600) {
            fireballs[i] = null;
      if(unaBola > 0) {
            unaBola -=1;
      }
}

}
}
}


//#####################################################################
//                  DIBUJA LAS BALAS Y LOS CORAZONES                 //
//#####################################################################



if(vidasTotal > 0) {
      int separacion = 0;
      for(int i = 0; i < vidasTotal ; i++) {
            this.entorno.dibujarImagen(Corazon, 450 - separacion, 20, 0,
0.1);
                  separacion = separacion + 60;
      }
}

if((maxFire - unaBola) > 0) {
      int separacion = 0;
      for(int i = 0; i < (maxFire - unaBola) ; i++) {
            this.entorno.dibujarImagen(fueguillos, 40 + separacion, 100, 0,
0.8);
                  separacion = separacion + 60;

      }

}







// SUMA DE VARIABLE TICKS USADA COMO CONTROL DE TIEMPO
ticks++;

//#########################################################################
//    CONTROL DE COLISION SAKURA, INMUNIDAD Y DISMINUCION DE VIDAS //
//#########################################################################

if(siChoco && !esInmune ) { // AL SUFRIR UN DANIO SE OTORGA 3 SEG DE INMUNIDAD
            Herramientas.play("ooh.wav");
            siChoco = false;
            esInmune = true;
            time = 0;
            vidasTotal --;
}
```

```
if(esInmune && time < 3 ) { //CONTROLA EL TIEMPO DE INMUNIDAD
             siChoco = false;
             if(ticks% 100 == 0) {
             time = time + 1;

             }
}else {
      esInmune = false;
}


//#####################################################################
//                  MUESTRA EL PUNTAJE ACTUAL                        //
//#####################################################################


this.entorno.cambiarFont(Font.SANS_SERIF, 30, Color.WHITE);
this.entorno.escribirTexto("Puntos  " + puntaje, 10, 50);
this.entorno.escribirTexto("Muertos  " + ninjasEliminados, 640, 50);

}

else{


//#####################################################################
//                         GAME OVER                                 //
//#####################################################################

      musiquita.stop();
      Herramientas.play("smb_gameover.wav");
      this.entorno.dibujarImagen(gameOver, 400, 300, 0);
      this.entorno.cambiarFont(Font.SANS_SERIF, 40, Color.WHITE);
      this.entorno.escribirTexto(" Presione Enter para salir " , 145, 545);

}







//#####################################################################
//           SI SE PRESIONA ENTER , CIERRA EL ENTORNO                //
//#####################################################################

if(this.entorno.sePresiono(this.entorno.TECLA_ENTER)) {

      this.entorno.dispose();
}


}

@SuppressWarnings("unused")
      public static void main(String[] args)
      {
```

```
		Juego juego = new Juego();
	}

}
```

##

Clase Casa

##

```
public class Casa {

	private Image casa;
	private double x;
	private double y;
	private double alto;
	private double ancho;

	public Casa double x, double y,double alto, double ancho) {
		this.x = x;
		this.y = y;
		this.alto = alto;
		this.ancho = ancho;
	}

public void dibujar(Entorno e) {
		e.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0,
Color.GREEN);
		//e.dibujarImagen(casa, x, y, alto);
	}


// Matrizes para la creacion de las casas del rectangulos
public static Casa[][] matrizDeCasasVerticalDerecho() {
	Casa[][] c = new Casa[5][4];

	double posInicialX = 110;
	double posInicialY = 44;  //pos en Y de la cuadra inicial

	for (int i = 0; i < 5; i++) {
		for (int j = 0; j < 4; j++) {
			c[i][j] = new Casa(posInicialX,posInicialY,20,20);
			posInicialX += 168;//DISTANCIA ENTRE X DE CADA CASA = 168
			}
			posInicialX = 110;  //se REINICIA para ubicar las casas de
abajo en el mismo X de los de arriba
			posInicialY += 128;//DISTANCIA ENTRE Y DE CADA CUADRA =
128
		}

		return c;
```

```
	}


public static Casa[][] matrizDeCasasVerticalIzquierdo() {
	Casa[][] c = new Casa[5][4];

	double posInicialX = 186;
	double posInicialY = 44;  //pos en Y de la cuadra inicial

	for (int i = 0; i < 5; i++) {
		for (int j = 0; j < 4; j++) {
			c[i][j] = new Casa(posInicialX,posInicialY,20,20);
			posInicialX += 168;//DISTANCIA ENTRE X DE CADA CASA = 168
		}
			posInicialX = 186;  //se REINICIA para ubicar las casas de
abajo en el mismo X de los de arriba
			posInicialY += 128;//DISTANCIA ENTRE Y DE CADA CUADRA =
128
		}

		return c;
	}


public static Casa[][] matrizDeCasasHorizontalArriba() {
	Casa[][] c = new Casa[4][5];

	double posInicialX = 64;  //pos en X de la cuadra inicial
	double posInicialY = 140;

	for (int i = 0; i < 4; i++) {
		for (int j = 0; j < 5; j++) {
			c[i][j] = new Casa(posInicialX,posInicialY,20,40);
			posInicialX += 168;//DISTANCIA ENTRE X DE CADA CASA = 168
			}
			posInicialX = 64;   //se REINICIA para ubicar las casas de
abajo en el mismo X de los de arriba
			posInicialY += 128;//DISTANCIA ENTRE Y DE CADA CUADRA =
128
		}

		return c;
	}

public static Casa[][] matrizDeCasasHorizontalAbajo() {
	Casa[][] c = new Casa[4][5];

	double posInicialX = 64;  //pos en X de la cuadra inicial
	double posInicialY = 76;

	for (int i = 0; i < 4; i++) {
		for (int j = 0; j < 5; j++) {
			c[i][j] = new Casa(posInicialX,posInicialY,20,40);
			posInicialX += 168;//DISTANCIA ENTRE X DE CADA CASA = 168
			}
			posInicialX = 64;   //se REINICIA para ubicar las casas de
abajo en el mismo X de los de arriba
			posInicialY += 128;//DISTANCIA ENTRE Y DE CADA CUADRA =
128
```

```
                }

                return c;
        }


        //GETTERS

        public double getX() {
                return x;
        }
        public double getY() {
                return y;
        }
        public double getAlto() {
                return alto;
        }
        public double getAncho() {
                return ancho;
        }
}
```

##

Clase Cuadra

##

```
public class Cuadra {
        private Image pasto;
        private double x;
        private double y;
        private double alto;
        private double ancho;


        public Cuadra int x, int y, double alto, double ancho) {
                this.x = x;
```

```
            this.y = y;
            this.alto = alto;
            this.ancho = ancho;
            this.pasto = Herramientas.cargarImagen("pasto.jpg");
      }

public void dibujar(Entorno e) {
      e.dibujarImagen(pasto, x, y,0, 0.26);
}


//crea muchos objeto Cuadra para que despues se le puedan poner imagenes
public static Cuadra[][] matrizDeCuadras() {//crea ubicaciones cuadras o
pastos
      Cuadra[][] cuadras = new Cuadra[5][5];


      int posInicialX = 64;     //es la mitad del ancho de la cuadra
      int posInicialY = 44;     //es la mitad del alto de la cuadra

      for (int i = 0; i < 5; i++) {
            for(int j = 0; j < 5; j++) {
                  cuadras[i][j] = new Cuadra(posInicialX,posInicialY,88,128);
                        posInicialX += 168;//en cada it de j, se construyan
las cuadras horizontales DISTANCIA ENTRE X DE CADA CUADRA = 200
                  }

                  posInicialY += 128;//en cada iteracion de i, salte a las
cuadras de abajo DISTANCIA ENTRE Y DE CADA CUADRA = 150
                  posInicialX = 64;  //se REINICIA para ubicar las cuadras
de abajo en el mismo x de los de arriba

                  /*
                   * 168 y 128 se sacaron con un calculo
                   */
            }

            return cuadras;
      }

public double getX() {
            return x;
      }

      public double getY() {
            return y;
      }

      public double getAlto() {
            return alto;
      }

      public double getAncho() {
            return ancho;
      }
}
```

##

Clase Flecha

##

```
public class Flecha {

	private double x;
	private double y;
	private double alto;
	private double ancho;



public Flecha(double x, double y) {
		this.x = x;
		this.y = y;
		this.alto = 10;
		this.ancho = 20;
	}

public void dibujar(Entorno e) {
	e.dibujarRectangulo(x, y, ancho, alto, 0, Color.MAGENTA);
	}

public static int objetivosRestantes() {
		int cuentaRegresiva = 3;
		cuentaRegresiva -= 1;
		return cuentaRegresiva;
	}

public boolean colisionFlechaSakura(double posSakuraX2 ,double posSakuraY2 ,
double posSakuraY1) {

		if( posSakuraX2 >= this.xIzquierdo() && posSakuraX2 <=
this.xDerecho() && posSakuraY2 >= yTecho() && posSakuraY1 <= ySuelo()) {
			return true;
		}
		return false;
	}
// posiciones del rectangulo para calcular colisiones
public double xDerecho() {
		double xd = x + ancho /2;
		return xd;
		}
public double xIzquierdo() {
		double xIz = x - ancho /2;
		return xIz;
	}
public double yTecho() {
	double yTc = y - alto /2;
	return yTc;
}
public double ySuelo() {
```

```
        double ySu = y + alto /2;
        return ySu;
}

//GETTERS
        public double getX() {
                return this.x;
        }
        public void setX(double x) {
                this.x = x;
        }

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

        public double getY() {
                return this.y;
        }
        public double getAlto() {
                return this.alto;
        }
        public double getAncho() {
                return this.ancho;
        }

}
```

##

Clase Sakura

##

```
public class Sakura {

// Variables de instancia

        private Image sakura;
        private int x;
        private int y;
        private double ancho;
        private double alto;
        private double diametro;

        // Constructor
public Sakura(int x, int y, double ancho, double alto, double diametro) {

                this.x = x;
                this.y = y;
                this.ancho = ancho;
                this.alto = alto;
```

```
            this.diametro = diametro;
            this.sakura = Herramientas.cargarImagen("fantasma.png");

}


public void dibujar(Entorno e) {
      //e.dibujarRectangulo(x, y, ancho, alto, 0, Color.BLUE);
      e.dibujarImagen(sakura, this.x, this.y, 0, 0.35);

}




//MUEVE A SAKURA SEGUN DIRECCIONES
            public void mover String palabra) {
                  if (palabra.equals("arriba")) {         //mover arriba
                        this.y = this.y - 3 ;
                  }
                  if (palabra.equals("abajo")) {          //mover abajo
                        this.y = this.y + 3;
                  }
                  if (palabra.equals("izquierda")) {      //mover izquierda
                        this.x = this.x - 3;
                  }
                  if (palabra.equals("derecha")) {        //mover derecha
                        this.x = this.x + 3;
                  }
            }


//DEFINE LIMITE DE MOV CON EL ENTORNO
public boolean chocasteBordeSuperior() {
      if (this.y - this.alto/2 < 0) {
            return true; //CHOQUE
      } else {
            return false;       //NO CHOQUE
            }
      }

//DEFINE LIMITE DE MOV CON EL ENTORNO
public boolean chocasteBordeInferior() {
      if (this.y + this.alto/2 > 600) {
            return true;
      } else {
            return false;
      }
}

            //DEFINE LIMITE DE MOV CON EL ENTORNO
            public boolean chocasteBordeIzquierdo() {
                  if (this.x - this.ancho/2 < 0) {
                        return true;
                  } else {
                        return false;
                  }
```

```
		}

		//DEFINE LIMITE DE MOV CON EL ENTORNO
		public boolean chocasteBordeDerecho() {
			if (this.x + this.ancho/2 > 800) {
				return true;
			} else {
				return false;
			}
		}



//DEFINE LIMITE DE MOV CON BORDE VERTICAL DE CUADRAS
public boolean noChocasteConBordeVerticalDeCuadra() {
	if (this.x > 138 && this.x < 154 || this.x > 307 && this.x < 323 ||
	this.x > 476 && this.x < 492 || this.x > 645 && this.x < 661) {
			return true;
	} else {
			return false;
	}
}



//DEFINE LIMITE DE MOV CON BORDE HORIZONTAL DE CUADRAS
public boolean noChocasteConBordeHorizontalDeCuadra() {
	if (this.y > 100 && this.y < 116 || this.y > 228 && this.y < 244 ||
	 this.y > 356 && this.y < 372 || this.y > 484 && this.y < 500 ) {
			return true;
	} else {
			return false;
	}
}



// Creo disparo


	public Fireball disparar() {

		return new Fireball(this.x, this.y - this.alto / 5, 10,10,0);
	}

	// Setter and Getter
	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 getAncho() {
		return ancho;
	}

	public void setAncho(double ancho) {
		this.ancho = ancho;
	}

	public double getAlto() {
		return alto;
	}

	public void setAlto(double alto) {
		this.alto = alto;
	}
	public double getDiametro() {
		return diametro;
	}


	// posiciones del rectangulo para calcular colisiones
	public double y1() {
		double y1 = y - (diametro /2);
		return y1;
	}

	public double y2() {
		double y2 = y + (diametro /2);
		return y2;
	}
	public double x2() {
		double x2 = x + (diametro/2) ;
		return x2;
	}
	public double x1() {
		double x1 = x - (diametro/2);
		return x1;
	}


}
```

##

Clase Ninja

##

```
public class Ninja {

	private Image ninja;
	private double x;
	private double y;
	private double ancho;
	private double alto;
	private double velocidad;

	public Ninja (double x,double y,double ancho,double alto,double
velocidad) {
		this.x = x;
		this.y = y;
		this.ancho = ancho;
		this.alto = alto;
		this.velocidad = velocidad;
		this.ninja = Herramientas.cargarImagen("ninja.gif");
	}

	public void dibujar(Entorno e) {

		//e.dibujarRectangulo(x, y, ancho, alto, 0, Color.CYAN);
		e.dibujarImagen(ninja, x, y, 0, 0.35);
	}

	// Movimiento del ninja
	public void mover() {
		this.y += (0.5);

	}

	public void moverEjeX() {
		this.x -= (0.5);
	}


	public double getX() {
		return x;
	}

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

	public double getY() {
		return y;
	}

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

	public double getAncho() {
		return ancho;
	}

	public void setAncho(double ancho) {
		this.ancho = ancho;
	}
```

```
	public double getAlto() {
		return alto;
	}

	public void setAlto(double alto) {
		this.alto = alto;
	}

	public double getVelocidad() {
		return velocidad;
	}

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



	// posiciones del rectangulo para calcular colisiones
	public double xDerecho() {
		double xd = x + ancho /2;
		return xd;
	}

	public double xIzquierdo() {
		double xIz = x - ancho /2;
		return xIz;
	}

	public double yTecho() {
		double yTc = y - alto /2;
		return yTc;
	}

	public double ySuelo() {
		double ySu = y + alto /2;
		return ySu;
	}



//#########################################################
//                  METODO DE CONTROL DE COLISION      //
//#########################################################


public boolean colisionNinjaSakura(double fireX2 ,double fireY2 , double
fireY1) {

	if( fireX2 >= this.xIzquierdo() && fireX2 <= this.xDerecho() && fireY2
	>= yTecho() && fireY1 <= ySuelo()) {
				return true;
			}
			return false;
		}
public boolean colisionFireballNinja(double fireX2 ,double fireY2 , double
fireY1 ) {
```

```
        if( fireX2 >= this.xIzquierdo() && fireX2 <= this.xDerecho() && fireY2
        >= yTecho() && fireY1 <= ySuelo()) {
                        return true;
        }
                return false;
}

}
```

##

Clase Fireball

##

```
public class Fireball{
        private Image disparo;
        private double x;
        private double y;
        private double alto;
        private double ancho;
        private double diametro;

        public Fireball(double x, double y, double alto, double ancho, double
diametro) {
                this.x = x;
                this.y = y;
                this.alto = alto;
                this.ancho = ancho;
                this.disparo = Herramientas.cargarImagen("disparo.png");
        }


        public void dibujar(Entorno e) {
        //      e.dibujarRectangulo(x, y, ancho, alto, 0, Color.YELLOW);
                e.dibujarImagen(disparo, this.x, this.y, 0);
        }


//Direccion del disparo se toma en sentido de las agujas del reloj (arriba =
1, derecha = 2,etc)
                public void disparar(int i)
                {
                        if(i==1)//arriba
                        {
                                this.y -= 4;
                        }
                        if(i==2)//derecha
                        {
                                this.x += 4;
                        }
                        if(i==3 )//abajo
```

```
                {
                        this.y += 4;
                }
                if (i==4) //izquierda
                {
                        this.x -= 4;
                }
                if (i==5) // Por defecto derecha
                {
                        this.x += 4;
                }
        }

public void disparoDerecha() {
        this.x += 4;
}


// Setter and Getter
public double getX() {
        return x;
}

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

public double getY() {
        return y;
}

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

public double getAlto() {
        return alto;
}

public void setAlto(double alto) {
        this.alto = alto;
}

public double getAncho() {
        return ancho;
}

public void setAncho(double ancho) {
        this.ancho = ancho;
}

public double getDiametro() {
        return diametro;
}


// Para calcular colisiones
public double y2() {
        double y2 = y + (diametro /2);
```

```
            return y2;
      }
      public double x2() {
            double x2 = x + (diametro/2) ;
            return x2;
      }
      public double y1() {
            double y1 = y - (diametro /2);
            return y1;
      }



}
```