Programación I

Trabajo Práctico: Sakura Ikebana Delivery

Profesores: Rodrigo Gastón González,

Nicolás Emanuel Cancela González.

Alumnos/Legajo: Ezequiel Nicolás Villafañe – 42457470/19

Nicolás Sassoni – 36905421/20

Comisión: COM-03

Año: 1er Semestre – 2021

Contenido

Introducción

El juego Sakura Ikebana Delivery se desenvuelve en una ciudad japonesa donde Sakura y su amiga de la infancia Ino son las dueñas de un negocio de arreglos florales típicos japoneses llamados ikebana.

El objetivo del mismo es entregar un ramo de flores Ikebana a distintas locaciones, evitando el ataque de ninjas y esquivando los obstáculos.

El presente trabajo consiste en desarrollar dicho juego utilizando lenguaje de programación JAVA. En las próximas líneas, se explicará en detalle el código implementado, las clases creadas, sus propios métodos y la forma en que fueron utilizadas.

Descripción

En este apartado nos enfocaremos en describir y explicar las variables de instancia y los métodos de cada clase implementada en el proyecto. También, desarrollaremos los problemas encontrados y las soluciones implementadas para cumplir con el objetivo de este proyecto.

Clase Sakura

Esta clase se encarga de la protagonista del juego, Sakura. Se establecen coordenadas x e y, dirección de movimiento, un ancho, alto y una imagen entre otras.

Variables de instancia

- `private double x:` Coordenada x del personaje.
- `private double y:` Coordenada y del personaje.
- `private double ancho:` Ancho del personaje.
- `private double alto:` Alto del personaje.
- `private int dirección:` Orientación del personaje 1: arriba 2: derecha 3: abajo 4: izquierda.
- `private Image sakuraImg:` Imagen del personaje.

Métodos

- `public Sakura(double x, double y, double ancho, double alto):` Inicializa las coordenadas x e y, ancho, alto, establece la dirección en 1 y carga la imagen del personaje
- `public void dibujar(Entorno entorno):` Dibuja la imagen del personaje en el entorno teniendo en cuenta las coordenadas x e y
- `public Rasengan disparar():` Retorna un objeto *Rasengan* teniendo en cuenta la dirección de Sakura.
- `public void moverse():` El personaje se mueve a una cierta dirección teniendo en cuenta el valor de la variable dirección. Por ej: Si la dirección es igual a 1 Sakura se mueve hacia arriba.
- `public int getDireccion():` Retorna la dirección en la que se mueve el personaje siendo 1 arriba, 2 derecha, 3 abajo y 4 izquierda

- `public void setDireccion(int direccion):` Se establece para que dirección se va a mover el personaje siendo 1 arriba, 2 derecha, 3 abajo y 4 izquierda.
- `public void moverArriba():` Mueve al personaje hacia arriba.
- `public void moverDerecha():` Mueve al personaje hacia la derecha.
- `public void moverAbajo():` Mueve al personaje hacia abajo.
- `public void moverIzquierda():` Mueve al personaje hacia la izquierda.
- `public double getX():` Retorna valor de la coordenada “x” del personaje.
- `public double getY():` Retorna el valor de la coordenada “y” del personaje.
- `public double getAncho():` Retorna el valor del ancho del personaje.
- `public double getAlto():` Retorna el valor del alto del personaje.
- `public Rectangulo getRect():` Retorna un objeto Rectángulo con las coordenadas x e y, el ancho y alto del personaje.

Problemas encontrados

Al momento de que Sakura realizaba un disparo el Rasengan (objeto que se crea en el método disparar()) no se comportaba de la manera esperada. Por lo tanto, se necesitaba pasar la información al Rasengan creado de la dirección en la cual Sakura se dirigía.

Soluciones implementadas

Se creó en primer lugar una variable llamada “dirección” el cual representa la actual dirección en la que Sakura se mueve y mediante el método “setDireccion(int dirección)” se establece la misma, luego se puede invocar al método moverse() para que Sakura logre recorrer dicha dirección. Cuando dispara, se crea el objeto Rasengan y el valor de la variable dirección se pasa por parámetro en el método constructor del mismo, de esta forma el Rasengan tendrá la información necesaria para moverse.

Clase Ninja

Esta clase se encarga de los personajes Ninjas. Se les establece una coordenada x e y, ancho, alto, velocidad y una imagen. Los ninjas pueden crear Púas e ir aumentando su velocidad de movimiento.

Variables de Instancia

- `private double x:` Coordenada del eje x.
- `private double y:` Coordenada del eje y.
- `private double ancho:` Ancho del personaje.
- `private double alto:` Alto del personaje.

- `private Image imgNinja:` Imagen del personaje.
- `private double velocidad:` Velocidad del personaje.

Métodos

- `public Ninja(double x, double y, double ancho, double alto, double velocidad):` Se inicializa las coordenadas, el ancho y alto y la velocidad inicial del ninja.
- `public void soltarPua():` Crea el objeto Púa teniendo en cuenta coordenadas "x" e "y "del Ninja.
- `public void dibujar(Entorno entorno):` Dibuja la imagen del personaje en el entorno teniendo en cuenta las coordenadas x e y del personaje.
- `public void moverDerecha():` Mueve el personaje hacia la derecha sobre el eje X teniendo en cuenta la velocidad.
- `public void moverIzquierda():` Mueve el personaje hacia la izquierda sobre el eje X teniendo en cuenta la velocidad.
- `public void moverArriba():` Mueve el personaje hacia arriba sobre el eje Y teniendo en cuenta la velocidad.
- `public void moverAbajo():` Mueve el personaje hacia abajo sobre el eje Y teniendo en cuenta la velocidad.
- `public double getX():` Devuelve el valor de la coordenada Y del personaje.
- `public double getY():` Devuelve el valor de la coordenada Y del personaje.
- `public void setX(double x):`Permite introducir otro valor a la coordenada X del personaje.
- `public void setY(double y):` Permite introducir otro valor a la coordenada Y del personaje.
- `public double getAncho():` Devuelve el valor del ancho del personaje.
- `public double getAlto():` Devuelve el valor del alto del personaje.
- `public void setVelocidad(double velocidad):` Permite modificar el valor de la velocidad.
- `public Rectangulo getRect():` Devuelve un objeto rectángulo teniendo en cuenta las coordenadas x e y, y el ancho y alto del personaje.

Clase Púas

Esta clase se encarga de controlar las propiedades y modelar el objeto Púas que es creado en la clase antes mencionada Ninja.

Variables de Instancia:

- `private double x:` Coordenada del eje x.
- `private double y:` Coordenada del eje y.
- `private double ancho:` Ancho del obstáculo.
- `private double alto:` Alto del obstáculo
- `private Image puasImg:` Imagen del obstáculo

Métodos

- `public Puas(double x, double y, double ancho, double alto):` inicializa los valores de las coordenadas .
- `public void dibujar(Entorno entorno):` Dibuja la imagen del obstáculo en el entorno teniendo en cuenta las coordenadas x e y.
- `public Rectangulo getRect():` Devuelve un objeto rectángulo teniendo en cuenta las coordenadas x e y, y el ancho y alto del mismo.

Clase Ikebana

La clase del ikebana controla las características del objeto que representa al ramo de flores japonesa del juego.

Variables de Instancia:

- `private double x:` Coordenada del eje x.
- `private double y:` Coordenada del eje y.
- `private Image imgIkebana:` Imagen del Ikebana.

Métodos

- `public Ikebana(double x, double y):` Se inicializa las coordenadas del Ikebana.
- `public void dibujar(Entorno entorno):`Dibuja la imagen en el entorno teniendo en cuenta las coordenadas x e y.

- `public Rectangulo getRect():` Devuelve un objeto rectángulo teniendo en cuenta las coordenadas x e y, y el ancho y alto del mismo.

Clase Rasengan

Esta clase se encarga de modelar el hechizo de Sakura. Este objeto se puede mover en una única dirección una vez creado.

Variables de instancia

- `private double x:` Coordenada x del Rasengan.
- `private double y:` Coordenada y del Rasengan.
- `private double ancho:` Ancho del Rasengan.
- `private double alto:` Alto del Rasengan.
- `private int direccion:` Dirección en la que se mueve el Rasengan siendo 1 arriba, 2 derecha, 3 abajo y 4 izquierda.
- `private Image rasenganImg:` Imagen del rasengan.

Métodos

- `public Rasengan(double x, double y, double ancho, double alto, int direccion):` Se le establece las coordenadas x e y, ancho, alto y la dirección al rasengan.
- `public void dibujar(Entorno entorno):` Dibuja la imagen del Rasengan en el entorno teniendo en cuenta las coordenadas x e y
- `public void moverse():` El Rasengan se mueve a una cierta dirección teniendo en cuenta el valor de la variable dirección. Por ej: Si la dirección es igual a 1 el Rasengan se mueve hacia arriba.
- `private void moverDerecha():` Mueve al personaje hacia arriba.
- `private void moverDerecha():` Mueve al rasengan hacia la derecha.
- `private void moverAbajo():` Mueve al rasengan hacia abajo.
- `private void moverIzquierda():` Mueve al rasengan hacia la izquierda.
- `public double getX():` Devuelve el valor de la coordenada x.
- `public double getY():` Devuelve el valor de la coordenada y.
- `public Rectangulo getRect():` Devuelve un objeto Rectángulo con las coordenadas x e y, ancho y alto del Rasengan.

Problemas encontrados:

El Rasengan no recorría ninguna dirección, se mantiene inmóvil o se movía junto con Sakura.

Soluciones encontradas:

Se crearon cuatro métodos “private” de movimiento, estos son “moverArriba()”, “moverAbajo()”, “moverDerecha()”, “moverIzquierda()” y moverse(). Este último tiene como información el valor de la variable dirección. Por lo tanto, cuando el objeto Rasengan es creado sólo puede moverse en una única dirección.

Clase Rectangulo

Esta clase se encarga de representar un Rectángulo con coordenadas x e y, ancho y alto. Además, determina cuando dos rectángulos colisionan.

Variables de instancia.

- `private double x:` Coordenada x del rectángulo.
- `private double y:` Coordenada y del rectángulo.
- `private double ancho:` Ancho del rectángulo.
- `private double alto:` Alto del rectángulo.

Métodos

- `public Rectangulo(double x, double y, double ancho, double alto):` Se le asigna las coordenadas x e y, ancho y alto al rectángulo.
- `public static boolean colision(Rectangulo r1, Rectangulo r2):`

  Determina cuándo dos rectángulos colisionan. Es decir, cuando dos rectángulos se superponen en el eje de coordenadas.

Problemas encontrados:

No logramos encontrar una forma de realizar la colisión entre objetos.

Solución encontrada:

Debido a que todos los objetos pueden ser representados como rectángulos creamos esta clase que unifica todas las colisiones entre ellos. Todas las clases exceptuando Juego y Ciudad tienen un método getRect () que devuelve un objeto rectángulo con las dimensiones y las coordenadas. Utilizando el método static de Rectángulo se puede tratar las colisiones con dos objetos distintos si estos poseen el método getRect.

Clase Casa

Esta clase representa las casas dentro del juego, aquí se encuentra su modelización.

Variables de Instancia:

- `private double x:` Coordenada del eje x
- `private double y:` Coordenada del eje y
- `private double ancho:` Ancho de la casa
- `private double alto:` Alto de la casa
- `private Image imgCasa:` Imagen de la casa
- `private int direcCasa:` Valor que representa la orientación de la casa

Métodos

- `public Casa(double x, double y, double ancho, double alto, int direcCasa):` se inicializa las coordenadas de la casa, el ancho , el alto, y la orientacion de la casa.
- `private void elegirCasa():` Carga la imagen según la dirección de la casa.
- `public void dibujar(Entorno entorno):` Dibuja la imagen de la casa en el entorno teniendo en cuenta las coordenadas x e y.
- `public void setImgCasa(Image image):` Permite cambiar la imagen de las casas.
- `public double getX():` Devuelve el valor de la coordenada x
- `public double getY():` Devuelve el valor de la coordenada y
- `public Rectangulo getRect():` Devuelve un objeto Rectángulo con las coordenadas x e y, ancho y alto.

Problemas encontrados

Nos cruzamos con el problema de que cada casa tenga una imagen distinta. Y luego se necesitaba que en cualquier momento se pueda cambiar la imagen de la casa para implementar la Florería.

Solución implementada

Se creó un método elegirCasa() que establece la imagen de la casa teniendo en cuenta el valor de la variable direcCasa. Además se creo un metodo setImgCasa(Image image) para poder cambiar la imagen de la casa por otra de una florería.

Clase Manzana

En esta clase se encuentran las características de las Manzanas, coordenadas, alto, ancho. Las mismas van a contener las casas.

Variables de instancia

- private double x: Coordenada del eje X
- private double y: Coordenada del eje Y
- private double ancho: Ancho de la manzana
- private double alto: Alto de la manzana
- private Casa[] casas: Arreglo de casas
- private Image imgManzana: Dibuja la imagen de la manzana ingresando como parámetros la imagen, eje x e y.

Métodos

- public Manzana(double x, double y, double ancho, double alto): Inicializa las coordenadas de la manzana, el ancho, el alto y ubica cuatro casas en los bordes de la manzana.
- public void dibujar(Entorno entorno): Dibuja las manzanas e indica en qué posición dibujar las casas dentro de la misma.
- public double getAncho(): Devuelve el ancho de la manza.
- public double getAlto(): Devuelve el alto de la manzana.
- public Casa[] getCasas(): Devuelve el arreglo de casas que contiene la información de las casas que conforman la manzana.
- public Rectangulo getRect(): Devuelve el rectángulo conformado por la manzana.

Clase Ciudad

Esta clase se encarga de representar una ciudad formada por manzanas y calles. La misma está organizada en forma de grilla.

Variables de instancia

- private double ancho: Ancho de la ciudad
- private double alto: Alto de la ciudad
- private int callesVerticales: Cantidad de calles verticales de la ciudad
- private int callesHorizontales: Cantidad de calles horizontales de la ciudad
- private Manzana[][] manzanas: Manzanas de la ciudad
- private double anchoCalle: Ancho de las calles
- private Image imgCalles: Imagen de las calles de la ciudad

Métodos

- public Ciudad(double ancho, double alto,int callesVerticales, int callesHorizontales,double anchoCalle): Se le establece un ancho a la ciudad, un alto, cantidad de calles verticales, cantidad de calles horizontales y el ancho de las calles. Con esos valores se crean las manzanas
- public void dibujar(Entorno entorno): Se dibuja en el entorno la ciudad y las manzanas en su correspondiente x e y, ancho y alto.
- private void crearManzanas(): Inicializa las manzanas teniendo en cuenta el ancho de la ciudad, el alto, la cantidad de calles establecida, el ancho de las calles. Le establece una coordenada x e y a las manzanas tal que queden en forma de grilla.
- public Manzana[][] getManzanas(): Devuelve el arreglo bidimensional de las manzanas
- public double getAnchoCalle(): devuelve el ancho de las calles de la ciudad
- public int getCallesVerticales(): devuelve la cantidad de calles verticales
- public int getCallesHorizontales(): devuelve la cantidad de calles horizontales

Problemas encontrados

Método constructor con demasiado código lo que resultaba difícil de leer.

Solución encontrada

Se creó un método que se encarga de inicializar las posiciones del arreglo manzanas. Se calcula el tamaño de las manzanas y las coordenadas x e y que van a tener. Se obtuvo un código más limpio y fácil de leer.

Clase Juego

Aquí es donde se encuentran las variables de instancias, métodos que controlan el comportamiento del propio juego. En esta clase se muestra la interfaz y realiza la interacción con el jugador mostrando en pantalla de forma visible los objetos antes descritos.

Variables de instancia

- `private Sakura sakura:` personaje principal del juego
- `private Rasengan rasengan:` Hechizo que realiza sakura para defenderse
- `private Ciudad ciudad:` Ciudad por donde se mueven los personajes del juego
- `private Casa floreria:` Casa donde el personaje tiene que ir a buscar ramo de flores
- `private Ikebana ikebana:` Ramo de flores que sakura va a buscar a la floreria y luego entregar en una casa de la ciudad
- `private Casa casaEntrega:` La casa que sakura tiene que ir a entregar el ramo de flores
- `private Manzana[][] manzanas:` Arreglo bidimensional de las manzanas de la ciudad
- `private Ninja [] ninjas:` Ninjas enemigos que el personaje Sakura tiene que evitar.
- `private Ninja [] ninjasMuertos:` Ninjas enemigos que colisionaron con el Rasengan
- `private Puas [] puas:` Obstáculo que sueltan los ninjas para hacer que Sakura pierda el juego
- `private Random rand:` Generador de números aleatorios usado para elegir una casa para la entrega
- `private Image imgFlecha:` Imagen de una flecha para señalizar la florería y la casa para entregar
- `private Image fondoMenu:` Imagen para mostrar en el menú
- `private int anchoPantalla:` Valor del ancho de la pantalla
- `private int altoPantalla:` Valor del alto de la pantalla
- `private int puntaje:` Valor del puntaje obtenido en el juego, se incrementa cada vez que sakura entrega un pedido
- `private int muertes:` Cantidad de ninjas matados por sakura.
- `private int tickPuas:` variable que decrementará para determinar cuándo las púas desaparecen y aparecen en la pantalla.
- `private boolean juegoTerminado:` variable para determinar cuando el juego empieza o termina. Por ej. si sakura muere se considera como juego terminado.

Métodos

- `Juego():` Se inicializa las variables de instancia necesarias para el funcionamiento del juego. Se cargan las imágenes, se establece el valor inicial para los personajes del juego. Luego se inicia el entorno del juego.

Problemas encontrados
Código muy largo con la inicialización de los ninjas.
Solución encontrada:
Se creó un método aparte (iniciarNinjas()) para inicializar los ninjas y darle su ubicación inicial.

- `public void tick():` Es el punto de entrada del juego, aquí se decide si se ejecuta el método menu() o pantallaJuego() dependiendo del valor de juegoTerminado.
- `private void pantallaJuego():` Se encarga de llamar a los métodos, tales como, los movimientos de los personajes, colisiones entre las entidades, entre otros. A su vez, controla la estadía del objeto púas en pantalla y el contador llamado tickPuas.
- `private void menu():` Es el primer método que interactúa con el usuario o jugador, dibuja el fondo de pantalla del menú e indica los pasos a seguir para el juego. También es al lugar al que se regresa una vez finalizado el juego.
- `private void dibujarEntidades():` Se encarga de determinar cuándo dibujar todos los objetos del juego y luego dibujarlos en el entorno.
- `private void elegirCasa():` Este método se encarga de elegir la casa a la que se llevará el pedido. En primera instancia selecciona al azar la manzana y dentro de ella, de la misma forma, la casa, hay que hacer una salvedad y es que si la manzana es alguna de los extremos superiores o inferiores de la pantalla, el mismo método restringe las casas inaccesibles para el personaje Sakura como también evita que el pedido se entregue en la misma florería.

Problemas encontrados

Al utilizar la clase Random para generar los números aleatoriamente se los tenía que restringir ya que algunos correspondían a casas que era imposible acceder para Sakura. Este problema se solventó de forma provisoria con algunos if que evitaban la selección de estos. Pero el código era muy largo y difícil de leer.

Solución implementada

Se crearon arreglos con los números posibles de utilizar. Es así que con la Clase Random solo es necesario generar aleatoriamente un número correspondiente a alguna posición del arreglo. Con este cambio se logró reducir el código del método y se mejoró notablemente su comprensión.

- `private void soltarPuas():` Se determina cuando los ninjas sueltan las púas. Si el puntaje es mayor o igual a 20 puntos los ninjas que estén en pantalla soltaran las púas. Luego, cada un determinado tiempo los ninjas vuelven a soltar más púas en la posición que se encuentren en ese momento.
- `private void quitarPuas():` Quita las púas que hayan soltado los ninjas.
- `private void dibujarPuntaje():`Dibuja en pantalla los puntajes (En la esquina superior derecha) y las muertes obtenidas por el jugador(En la esquina superior izquierda).
- `private void señalizarFloreria():` Se dibuja una flecha arriba de la casa florería. Indica que el personaje Sakura tiene que ir a buscar un pedido allí.
- `private void iniciarNinjas():` `Se inicializa el arreglo de ninjas ubicándolos en en el centro` de las calles según los siguientes datos: la cantidad y ancho de calles, el ancho de las manzanas, el largo y el alto de la pantalla.

Problema encontrado

Al no contar con calles determinadas, es decir, las calles estaban delimitadas por las manzanas no teníamos las coordenadas de las calles por ende no podíamos distribuir los ninjas por calle y que se situaran en el centro.

Solución implementada

Teniendo como datos las medidas del ancho de calles y largo o ancho de las manzanas, realizamos un cálculo porcentual que resulta en la ubicación de cada ninja en el centro de la calle por ejemplo: el primer ninja de las calles verticales se ubica en el ancho de la manzana más la mitad del ancho de la calle como su coordenada x y el segundo ninja en dos manzanas más el ancho de la primera calle más la mitad de la segunda calle. Generalizando esta serie se pueden agregar más o menos cantidad de ninjas.

- `private void restaurarNinjas():` Realiza un conteo de la cantidad de ninjas vivos y si este es menor o igual a 4, los reestablece en coordenadas predeterminadas.

Problema encontrado
Una vez eliminados los ninjas, se perdía la información de su ubicación, entonces iniciamos un nuevo objeto ninja con la información del eliminado, el resultado era que los nuevos ninjas no se movían.
Solución implementada
Creamos un arreglo para los ninjas eliminados en el cual, cuando un ninja era eliminado, esta referencia era pasada al arreglo de ninjasMuertos en la misma posición en la que se encontraba. De esta forma al momento de restaurarlos realizamos el proceso inverso, pasar esta referencia al arreglo de ninjas.

- `private void restaurarNinja():` Verifica si hay al menos un ninja vivo y al término de cierto tiempo lo restablece.
- `private void movimientoNinjas():` Indica el camino a recorrer por las calles verticales y horizontales, una vez alcanzan el borde de la pantalla los mismos reingresan por el lado opuesto de la calle, siempre en la misma dirección.
- `private void colisionNinjaRasengan():` Evalúa la colisión entre los Ninjas y el Rasengan. Al colisionar, llama al método sumarMuertes() y elimina al Rasengan y al ninja involucrados.
- `private void colisionRasengan():` Evalúa la colisión entre el Rasengan y las manzanas de la ciudad. Si colisiona, se elimina el rasengan.
- `private void movimientoSakura():` Evalúa cual tecla que se encuentra presionando el jugador y mueve a Sakura a la dirección de la tecla apretada. A su vez, controla que Sakura no atraviese los límites de la pantalla y no atraviese las manzanas de la ciudad, es decir, que únicamente pueda circular por las calles.

Problema encontrado
Lo solicitado en la consigna es que Sakura pueda moverse en 1 dirección a la vez, pero en principio utilizamos los condicionales “if” para describir esas direcciones, sin embargo, Sakura lograba moverse en diagonal.
Por otra parte, cuando Sakura colisionaba con las manzanas dejaba de moverse.
Solución implementada
Cambiamos el condicional “if” por un “else if” de esta forma Sakura solo puede ir en 1 dirección a la vez.
Con respecto a la colisión de Sakura con las manzanas, fue necesario implementar el método colosionSakuramanzanas(), el cual se encarga de retornar un True en caso de que Sakura colisione. Con esta información el método de movimientoSakura() a evalúa

si en la posición en la que se encuentra hay colisión, en caso de ser verdadero ejecuta el mismo movimiento en dirección opuesto, es decir se cancelan.

- `private boolean colisionSakuraManzanas():` Retorna True si Sakura colisiona con alguna de las manzanas y retorna False si sakura no colisiona con ninguna.
- `private void colisionSakuraNinjas():` Si sakura colisiona con algún ninja, se reinicia el juego. Esto significa que el Jugador ha perdido el juego.
- `private void colisionSakuraPuas():` Si sakura colisiona con alguna púa, se resetea el juego, es decir, que el Jugador ha perdido el juego.
- `private void sakuraEntrego():`Si Sakura lleva un Ikebana y colisiona con la casa seleccionada para entregar el pedido, el ikebana es eliminado. También, este método suma cinco puntos al Jugador.
- `private void sumarMuerte():` Suma 1 muerte al puntaje de Ninjas muertos.
- `private void sumarPuntos():` Suma 5 puntos al puntaje de entregas.
- `private void resetarJuego():`Restablece las variables a su valor inicial para poder volver a jugar.
- `private void colisionSakuraFloreria():` Si Sakura colisiona con la casa Floreria, se crea un ikebana con las coordenadas X e Y de Sakura.
- `private void movimientoIkebana():` Establece las coordenadas x e y del ikebana iguales a las de Sakura.
- `private void cambiarVelocidadNinjas():` Cuando el puntaje llega a 35 se cambia la velocidad de los ninjas.
- `private void movimientoRasengan():` comprueba que exista el objeto y controla el movimiento del Rasengan.

Clase Sakura

```
public class Sakura {
    private double x;
    private double y;
    private double ancho;
    private double alto;
    private int direccion;
    private Image sakuraImg;

    public Sakura(double x, double y, double ancho, double alto) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.direccion=1;
        this.sakuraImg=Herramientas.cargarImagen("imagenes/sakura.png");
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarImagen(sakuraImg, x, y, 0, 1);
    }

    public Rasengan disparar() {
        return new Rasengan(x,y,15,15,this.direccion);
      //se crea un disparo con la direccion de Sakura
    }

    public void moverse() {
        if(this.direccion == 1) {
            moverArriba();
        }else if(this.direccion == 2) {
            moverDerecha();
        }else if(this.direccion == 3) {
            moverAbajo();
        }else if(this.direccion == 4) {
            moverIzquierda();
        }
    }

    public int getDireccion() {
        return this.direccion;
    }
```

```
    public void setDireccion(int direccion) {
        this.direccion = direccion;
    }

    public void moverDerecha() {
        this.x+=2;
    }

    public void moverIzquierda() {
        this.x-=2;
    }

    public void moverArriba() {
        this.y-=2;
    }

    public void moverAbajo() {
        this.y+=2;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public double getAncho() {
        return ancho;
    }

    public double getAlto() {
        return alto;
    }

    public Rectangulo getRect() {
        return new Rectangulo(x, y, ancho, alto);
    }

}
```

Clase Ninja

```
public class Ninja {

    private double x;
    private double y;
    private double ancho;
    private double alto;
    private Image imgNinja;
    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.imgNinja=Herramientas.cargarImagen("imagenes/imgninja.png");
    }

    public Puas soltarPua() {
        return new Puas(x, y, 10, 10);
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarImagen(imgNinja, x, y, 0,0.70);

    }
    public void moverDerecha() {
        this.x+= this.velocidad;
    }

    public void moverIzquierda() {
        this.x-=this.velocidad;
    }

    public void moverArriba() {
        this.y -= this.velocidad;
    }

    public void moverAbajo() {
        this.y +=this.velocidad;
    }



    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 double getAlto() {
        return alto;
    }

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

    public Rectangulo getRect() {
        return new Rectangulo(x, y, ancho, alto*2);
    }

}
```

Clase Puas

```
public class Puas {

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

    public Puas(double x, double y, double ancho, double alto) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.puasImg=Herramientas.cargarImagen("imagenes/puas.png");
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarImagen(puasImg, x, y, 0,1);
    }

    public Rectangulo getRect() {
        return new Rectangulo(x, y, ancho, alto);
    }

}
```

Clase Ikebana

```
public class Ikebana {

    private double x;
    private double y;
    private Image imgIkebana;

    public Ikebana(double x, double y) {
        this.x = x;
        this.y = y;
        this.imgIkebana =Herramientas.cargarImagen("imagenes/ikebana.png"
                                                   );
    }

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

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

    public void dibujar(Entorno entorno) {
        entorno.dibujarImagen(imgIkebana, x, y, 0, 0.5);
    }

}
```

Clase Rasengan

```
public class Rasengan {

    private double x;
    private double y;
    private double ancho;
    private double alto;
    private int direccion;
    private Image rasenganImg;

    public Rasengan(double x, double y, double ancho, double alto,
int direccion) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.direccion = direccion;
        this.rasenganImg= Herramientas.cargarImagen("imagenes/rasengan.pn
                                                    g");
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarImagen(rasenganImg, x, y, 0,0.05);
    }

    public void moverse() {
        if(this.direccion == 1) {
            moverArriba();
        }else if(this.direccion == 2) {
            moverDerecha();
        }else if(this.direccion == 3) {
            moverAbajo();
        }else if(this.direccion == 4) {
            moverIzquierda();
        }
    }

    private void moverDerecha() {
        this.x+=3;
    }

    private void moverIzquierda() {
        this.x-=3;
    }
```

```
    private void moverArriba() {
        this.y-=3;
    }

    private void moverAbajo() {
        this.y+=3;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public Rectangulo getRect() {
        return new Rectangulo(x, y, ancho, alto);
    }
}
```

Clase Rectangulo

```
public class Rectangulo {

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

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


    public static boolean colision(Rectangulo r1, Rectangulo r2) {

        // r1.bordeSuperior > r2.bordeInferior
        if((r1.y - (r1.alto/2)) > (r2.y + (r2.alto/2))) {
            return false;
        //r1.bordeInferior < r2.bordeSuperior
        }else if((r1.y + (r1.alto/2)) < (r2.y - (r2.alto/2))) {
```

```
            return false;
        //r1.bordeIzquiedo > r2.bordeDerecho
        }else if((r1.x - (r1.ancho/2)) > (r2.x + (r2.ancho/2))) {
            return false;
        // r1.bordeDerecho < r2.bordeIzquierdo
        }else if((r1.x+ (r1.ancho/2)) < (r2.x - (r2.ancho/2))) {
            return false;
        }else {
            return true;
        }
    }
}
```

Clase Casa

```
public class Casa {

    private double x;
    private double y;
    private double ancho;
    private double alto;
    private Image imgCasa;
    private int direcCasa;

    public Casa(double x, double y, double ancho, double alto,
                int direcCasa) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.direcCasa=direcCasa;
        elegirCasa();
    }




private void elegirCasa(){
        if (this.direcCasa==0)
            this.imgCasa=Herramientas.cargarImagen("imagenes/casaArriba.p
                                                   ng");
        else if (this.direcCasa==1)
            this.imgCasa=Herramientas.cargarImagen("imagenes/casaAbajo.pn
                                                    g");
```

```
        else if (this.direcCasa==2)
            this.imgCasa=Herramientas.cargarImagen("imagenes/casaIzquierd
                                                   a.png");
        else if (this.direcCasa==3)
            this.imgCasa=Herramientas.cargarImagen("imagenes/casaDerecha.
                                                    png");
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarImagen(imgCasa, x, y, 0,0.8);
    }

    public void setImgCasa(Image image) {
        this.imgCasa = image;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public Rectangulo getRect() {
       return new Rectangulo(x, y, ancho+5, alto+5);
      //hit box de entrega del ikebana o se entiende como área de    colision
    }

}
```

Clase Manzana

```
public class Manzana {

    private double x;
    private double y;
    private double ancho;
    private double alto;
    private Casa[] casas;
    private Image imgManzana;

    public Manzana(double x, double y, double ancho, double alto) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
```

```
        this.imgManzana = Herramientas.cargarImagen("imagenes/
pasto.png");
        this.casas = new Casa[4];
        //casa arriba
        casas[0] = new Casa(x,
                            (alto/2)+((alto/3)/2),
                            ancho/3, alto/3,
                           0);
        //casa abajo
        casas[1] = new Casa(x,
                           y+(alto/2)-((alto/3)/2),
                           ancho/3,
                           alto/3,1);
        //casa izquieda
        casas[2] = new Casa(x-(ancho/2)+((ancho/3)/2),
                           y,
                            ancho/3,
                            alto/3,
                           2);
        //casa derecha
        casas[3] = new Casa(x+(ancho/2)-((ancho/3)/2),
                            y,
                            ancho/3,
                            alto/3,
                           3);
    }




    public void dibujar(Entorno entorno) {
        entorno.dibujarImagen(imgManzana, x, y, 0, 1);
        casas[0].dibujar(entorno);
        casas[1].dibujar(entorno);
        casas[2].dibujar(entorno);
        casas[3].dibujar(entorno);
    }

    public double getAncho() {
        return ancho;
    }

    public double getAlto() {
        return alto;
    }

    public Casa[] getCasas() {
```

```
        return casas;
    }

    public Rectangulo getRect() {
        return new Rectangulo(x, y, ancho, alto);
    }
}
```

Clase Ciudad

```
public class Ciudad {

    private double ancho;
    private double alto;
    private int callesVerticales;
    private int callesHorizontales;
    private Manzana[][] manzanas;
    private double anchoCalle;
    private Image imgCalles;

    public Ciudad(double ancho, double alto,int callesVerticales,
                        int callesHorizontales,double anchoCalle) {

        this.anchoCalle = anchoCalle;
        this.ancho = ancho;
        this.alto = alto;
        this.callesHorizontales = callesHorizontales;
        this.callesVerticales = callesVerticales;
        this.manzanas = new Manzana[callesHorizontales+1][callesVerticales+1];

        this.imgCalles=Herramientas.cargarImagen("imagenes/calles.png");
        crearManzanas();
}

    public void dibujar(Entorno entorno) {
        entorno.dibujarImagen(imgCalles, ancho/2, alto/2, 0,1);
        for (int i = 0; i < manzanas.length; i++) {
              //se dibujan las manzanas
            for (int j = 0; j < manzanas[i].length; j++) {
                manzanas[i][j].dibujar(entorno);
            }
        }
    }

    private void crearManzanas() {
        double anchoManzana = (this.ancho - (this.anchoCalle*callesVerticales))/
                                (callesVerticale s+1);  // calcula el ancho de las manzanas
```

```
        double altoManzana =(this.alto - (this.anchoCalle*callesHorizontales))/
                            (callesHorizontales+1); // calcula el alto de las manzanas
        double x = anchoManzana/2; // pos x inicial de la primera manzana
        double y = altoManzana/2; // pos y inicial de la primera manzana

        for (int i = 0; i < manzanas.length; i++) {
            x = anchoManzana/2;
            for (int j = 0; j < manzanas[i].length; j++) {
                manzanas[i][j] = new Manzana(x,y,anchoManzana,altoManzana);
                x += anchoManzana + anchoCalle;
            }
            y += altoManzana + anchoCalle;
        }
    }

    public Manzana[][] getManzanas() {
        return manzanas;
    }
    public double getAnchoCalle() {
        return anchoCalle;
    }

    public int getCallesVerticales() {
        return callesVerticales;
    }
    public int getCallesHorizontales() {
        return callesHorizontales;
    }
}
```

Clase Juego

```
public class Juego extends InterfaceJuego {
    // El objeto Entorno que controla el tiempo y otros
    private Entorno;

    // Variables y métodos propios de cada grupo
    // ...
    private Sakura sakura;
    private Rasengan rasengan;
    private Ciudad;
    private Casa floreria;
    private Ikebana;
    private Casa casaEntrega;
    private Manzana[][] manzanas;
    private Ninja [] ninjas;
```

```
private Ninja [] ninjasMuertos;
private Puas [] puas;
private Random rand;
private Image imgFlecha;
private Image fondoMenu;
private int anchoPantalla;
private int altoPantalla;
private int puntaje;
private int muertes;
private int tickPuas;
private boolean juegoTerminado;

Juego(){
    // Inicializa el objeto entorno
    this.anchoPantalla = 800;
    this.altoPantalla = 600;
    this.entorno = new Entorno(this, "Sakura Ikebana Delivery - Grupo 03 - v1",
                               anchoPantalla, altoPantalla);

    // Inicializar lo que haga falta para el juego
    // ...
    this.puntaje = 0;
    this.muertes = 0;
    this.juegoTerminado=true;



    this.ciudad = new Ciudad(anchoPantalla,altoPantalla,3,3,40);
    this.manzanas = ciudad.getManzanas();

    this.sakura = new Sakura(anchoPantalla/2, altoPantalla/2, ciudad.getAnchoCalle()/2,
                             this.ciudad.getAnchoCalle()/2);
    this.rasengan=null;
    this.ikebana = null;
    this.rand = new Random();
    this.casaEntrega = null;

    this.floreria = manzanas[0][manzanas.length-1].getCasas()[1];
    // la floreria siempre es la casa de abajo de la manzana superior derecha
    this.floreria.setImgCasa(Herramientas.cargarImagen("imagenes/floreria.png"));
    //se le cambia la imagen a la floreria
    this.imgFlecha = Herramientas.cargarImagen("imagenes/flecha.png");
    this.fondoMenu=Herramientas.cargarImagen("imagenes/fondopantalla.png");

    this.ninjas = new Ninja [ciudad.getCallesHorizontales()+ciudad.getCallesVerticales()];
    iniciarNinjas();
    this.ninjasMuertos =
            new Ninja [ciudad.getCallesHorizontales()+ciudad.getCallesVerticales()];
```

```
        this.puas = new Puas[ciudad.getCallesHorizontales()+ciudad.getCallesVerticales()];
        this.tickPuas = 700;

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

    public void tick(){
        if (juegoTerminado)
            menu();
        else
            pantallaJuego();
    }

    private void pantallaJuego(){
        dibujarEntidades();

        //contadores
        this.tickPuas--;

        elegirCasa();

        restaurarNinjas();
        restaurarNinja();


        //movimiento de entidades
        movimientoSakura();
        movimientoRasengan();
        movimientoNinjas();
        movimientoIkebana();

        //colisiones
        colisionSakuraNinjas();
        colisionRasengan();
        colisionNinjaRasengan();
        colisionSakuraPuas();
        colisionSakuraFloreria();

        sakuraEntrego();

//las puas aparecen a partir de un cierto puntaje a la vez que cambia la velocidad de mov ninja
        soltarPuas();
        if(this.tickPuas == 150) {
            quitarPuas();
        }
        cambiarVelocidadNinjas();
    }
```

```
private void dibujarEntidades() {
    ciudad.dibujar(entorno);
    dibujarPuntaje();

    señalizarFloreria();

    //dibuja la flecha arriba de la casa
    if(this.casaEntrega != null) {
        entorno.dibujarImagen(this.imgFlecha, this.casaEntrega.getX(),
                              this.casaEntrega.getY()-30, 0, 0.77);
    }

    sakura.dibujar(entorno);

    if(this.ikebana != null) {
        this.ikebana.dibujar(entorno);
    }

    if(this.rasengan != null){
        this.rasengan.dibujar(entorno);
    }




    for (int i = 0; i < puas.length; i++) {
        if(this.puas[i] != null) {
            this.puas[i].dibujar(entorno);
        }
    }

    for (int i = 0; i < ninjas.length; i++) {
        if(ninjas[i]!=null)
            ninjas[i].dibujar(entorno);
    }
}

private void menu(){
    entorno.dibujarImagen(fondoMenu, anchoPantalla/2, altoPantalla/2, 0,1);
    if(entorno.estaPresionada(entorno.TECLA_ENTER))
        this.juegoTerminado=false;
}

private void soltarPuas() {
    if(this.tickPuas <= 0 && this.puntaje>=20) {
        this.tickPuas = 700;
        for (int i = 0; i < ninjas.length; i++) {
```

```
            if(this.ninjas[i]!=null) {
                this.puas[i] = this.ninjas[i].soltarPua();
            }
        }
    }
}

private void quitarPuas() {
    for (int i = 0; i < puas.length; i++) {
        if(this.puas[i] != null) {
            this.puas[i] = null;
        }
    }
}

private void dibujarPuntaje() {
    entorno.dibujarRectangulo(120/2, 30/2, 120, 30, 0, Color.black);
    entorno.dibujarRectangulo(anchoPantalla-(120/2), 30/2, 120, 30, 0, Color.black);
    entorno.cambiarFont("Lucida Console", 20, Color.yellow);
    entorno.escribirTexto("Score :" + Integer.toString(puntaje), anchoPantalla-115, 20);
    entorno.escribirTexto("Kills :" + Integer.toString(muertes), 2, 20);
}

private void elegirCasa() {
    if(this.casaEntrega == null && this.ikebana != null) {
        int fila = rand.nextInt(manzanas.length);
        int columna = rand.nextInt(manzanas[0].length);
        Manzana manzanaElegida = manzanas[fila][columna];
       // se elige una manzana aleatoriamente

        /* casa 0 = arriba
         * casa 1 = abajo
         * casa 2 = izquierda
         * casa 3 = derecha
         */

        if(fila == 0) { // manzanas superiores
            if(columna == 0) { // manzana superior izquierda
                int [] numeros = {1,3}; // solo se elije la casa 1 abajo y casa 3 derecha
                int casa = rand.nextInt(2); // posicion 0 o 1
                this.casaEntrega = manzanaElegida.getCasas()[numeros[casa]];
            }else if(columna == manzanas[0].length-1) { // manzana superior derecha
                int [] numeros = {1};
                int casa = rand.nextInt(1);
                this.casaEntrega = manzanaElegida.getCasas()[numeros[casa]];
                    // solo elige la casa de la izquierda (no elige floreria)
            }else {
                int [] numeros = {1,2,3};
```

```
                int casa = rand.nextInt(3);
                this.casaEntrega = manzanaElegida.getCasas()[numeros[casa]];
                                                        // no elige las casas de arriba
            }

        }else if(fila == manzanas.length-1) { // manzanas inferiores
            if(columna == 0) {
                int [] numeros = {0,3};
                int casa = rand.nextInt(2);
                this.casaEntrega = manzanaElegida.getCasas()[numeros[casa]];
                                        // solo elige las casas de la derecha y arriba
            }else if(columna == manzanas[0].length-1) {
                int [] numeros = {0,2};
                int casa = rand.nextInt(2);
                this.casaEntrega = manzanaElegida.getCasas()[numeros[casa]];
                                       //solo elige las casas de la izquierda y arriba
            }else {
                int [] numeros = {0,2,3};
                int casa = rand.nextInt(3);
                this.casaEntrega = manzanaElegida.getCasas()[numeros[casa]];
                                                        //no elige las casas de abajo
            }
        }else { // si no elige la fila superior y la fila inferior



    if(columna == 0) {
                int [] numeros = {0,1,3};
                int casa = rand.nextInt(3);
                this.casaEntrega = manzanaElegida.getCasas()[numeros[casa]];
                                                    // no elige las casas de la izquierda
            }else if(columna == manzanas[0].length-1) {
                int [] numeros = {0,1,2};
                int casa = rand.nextInt(3);
                this.casaEntrega = manzanaElegida.getCasas()[numeros[casa]];
            }else {
                int [] numeros = {0,1,2,3};
                int casa = rand.nextInt(4);
                this.casaEntrega = manzanaElegida.getCasas()[numeros[casa]];
            }
        }
    }
}

private void señalizarFloreria() {
    if(this.ikebana == null) {
        entorno.dibujarImagen(imgFlecha, this.floreria.getX(),
                              this.floreria.getY()-40, 0, 0.77);
```

```
        }
    }

    private void iniciarNinjas(){
        int contCalles=1;
        int contManzanas=1;
        // crea los ninjas de las calles verticales
        for (int i = 0; i < (ninjas.length)/2; i++) {
            this.ninjas[i]=new Ninja((manzanas[0][0].getAncho()*(contManzanas))+
                                     (ciudad.getAnchoCalle()*contCalles/2), //eje x
                                     20,
                                     ciudad.getAnchoCalle()/2,
                                     ciudad.getAnchoCalle()/2,
                                     1);
            contCalles+=2;
            contManzanas++;
        }
        contCalles=1;
        contManzanas=1;
        // crea los ninjas de las calles horizontales
        for (int i = (ninjas.length)/2; i < ninjas.length; i++) {
            this.ninjas[i]=new Ninja(40,
                                     (manzanas[0][0].getAlto()*(contManzanas))+
                                     (ciudad.getAnchoCalle()*contCalles/2),
                                     ciudad.getAnchoCalle()/2,
                                     ciudad.getAnchoCalle()/2,
                                     1);
            contCalles+=2;
            contManzanas++;
        }
    }

    private void movimientoNinjas(){
        for (int i = 0; i < (ninjas.length)/2; i++) {
        //movimiento de ninjas calles verticales
            if(ninjas[i]!=null) {
                ninjas[i].moverAbajo();
                if (ninjas[i].getY()>=altoPantalla+ninjas[i].getAlto()) {
                //cuando llegan al final de la pantalla regresan al inicio
                    ninjas[i].setY(0);
                }
            }
        }
        for (int i = (ninjas.length)/2; i < ninjas.length; i++) {
        //movimiento de ninjas calles horizontales
            if(ninjas[i]!=null) {
                ninjas[i].moverDerecha();
                if (ninjas[i].getX()>=anchoPantalla+ninjas[i].getAncho()) {
```

```
                ninjas[i].setX(0);
            }
        }
    }
}
private void colisionNinjaRasengan() {
    for (int i = 0; i < ninjas.length; i++) {
        if(ninjas[i]!=null && this.rasengan != null) {
            if(Rectangulo.colision(this.ninjas[i].getRect(), this.rasengan.getRect())) {
                sumarMuerte();
                this.ninjasMuertos[i]=this.ninjas[i];
                //se agrega el ninja a el arreglo de ninjas muertos
                this.ninjas[i] = null;
                this.rasengan = null;
            }
        }
    }
}

private void restaurarNinjas() {
    int cantVivos=0;
    for (int i = 0; i < ninjas.length; i++) { // se cuenta la cantidad de ninjas vivos
        if(ninjas[i]!=null) {
            cantVivos++;
        }
    }
    if(cantVivos <=4) {
        for (int i = 0; i < ninjasMuertos.length/2; i++) {
    // se recorren las posiciones del arreglo de ninjas muertos calles verticales
            if(ninjasMuertos[i] != null) {
                this.ninjasMuertos[i].setY(-40);
                this.ninjas[i] = this.ninjasMuertos[i];
                this.ninjasMuertos[i]=null;
            }
        }
        for (int i = ninjasMuertos.length/2; i < ninjasMuertos.length; i++) {
            // se recorren las posiciones del arreglo de ninjas muertos calles horizontales
            if(ninjasMuertos[i] != null) {
                this.ninjasMuertos[i].setX(-40);
                this.ninjas[i] = this.ninjasMuertos[i];
                this.ninjasMuertos[i]=null;
            }
        }
    }
}

private void restaurarNinja() {
        for (int i = 0; i < ninjasMuertos.length/2; i++) {
```

```
            if(this.ninjasMuertos[i] != null) {
                this.ninjasMuertos[i].setY(-200); //para que tarden en aparecer
                this.ninjas[i] = this.ninjasMuertos[i];
                this.ninjasMuertos[i]=null;
                return;
            }
        }
        for (int i = ninjasMuertos.length/2; i < ninjasMuertos.length; i++) {
            if(ninjasMuertos[i] != null) {
                this.ninjasMuertos[i].setX(-200); //para que tarden en aparecer
                this.ninjas[i] = this.ninjasMuertos[i];
                this.ninjasMuertos[i]=null;
                return;
            }
        }
    }

    private void movimientoRasengan() {

        // si se presiona la tecla espacio y no existe un disparo, se crea un disparo
        if(this.entorno.sePresiono(entorno.TECLA_ESPACIO) && this.rasengan == null) {
            this.rasengan = sakura.disparar();
        }

        // se mueve
        if(this.rasengan != null) {
            this.rasengan.moverse();
        }
    }

    private void colisionRasengan() {
        if(this.rasengan != null){
            // colision con manzanas
            for (int i = 0; i < manzanas.length; i++) {
                for (int j = 0; j < manzanas[i].length; j++) {
                    if(Rectangulo.colision(this.rasengan.getRect(), manzanas[i][j].getRect())) {
                        this.rasengan = null;
                        return;
                    }
                }
            }
            //colision con limites
            if(this.rasengan.getX() <= 0 || this.rasengan.getX() >= anchoPantalla ||
            this.rasengan.getY() <= 0 || this.rasengan.getY() >= altoPantalla){
                this.rasengan = null;
                return;
            }
        }
```

```
}

private void movimientoSakura() {
    // Limites y movimientos arriba abajo izquierda y derecha
    if(this.entorno.estaPresionada(entorno.TECLA_DERECHA) &&
    sakura.getX()< anchoPantalla - (sakura.getAncho()/2)) {
        sakura.setDireccion(2);
        sakura.moverse();
        if(colisionSakuraManzanas()) {
            sakura.moverIzquierda();
        }
    }else if(this.entorno.estaPresionada(entorno.TECLA_IZQUIERDA) &&
    sakura.getX() > (sakura.getAncho()/2)) {

        sakura.setDireccion(4);
        sakura.moverse();
        if(colisionSakuraManzanas()) {
            sakura.moverDerecha();
        }
    }else if(this.entorno.estaPresionada(entorno.TECLA_ARRIBA) &&
    sakura.getY() > (sakura.getAlto()/2)) {
        sakura.setDireccion(1);
        sakura.moverse();
        if(colisionSakuraManzanas()) {
            sakura.moverAbajo();
        }
    }else if(this.entorno.estaPresionada(entorno.TECLA_ABAJO) &&
    sakura.getY() < altoPantalla -(sakura.getAlto()/2)) {
        sakura.setDireccion(3);
        sakura.moverse();
        if(colisionSakuraManzanas()) {
            sakura.moverArriba();
        }
    }
}

private boolean colisionSakuraManzanas() {
    for (int i = 0; i < manzanas.length; i++) {
        for (int j = 0; j < manzanas[i].length; j++) {
            if(Rectangulo.colision(sakura.getRect(), manzanas[i][j].getRect())) {
                return true;
            }
        }
    }

    return false;
}
```

```
private void colisionSakuraNinjas() {
    for (int i = 0; i < ninjas.length; i++) {
        if(ninjas[i] != null) {
            if(Rectangulo.colision(this.ninjas[i].getRect(), this.sakura.getRect())) {
                resetarJuego();
            }
        }
    }
}

private void colisionSakuraPuas() {
    for (int i = 0; i < puas.length; i++) {
        if(this.puas[i] != null) {
            if(Rectangulo.colision(this.puas[i].getRect(), this.sakura.getRect())) {
                resetarJuego();
            }
        }
    }
}

private void sakuraEntrego() {
    if(this.casaEntrega != null) {
        if(Rectangulo.colision(this.sakura.getRect(), this.casaEntrega.getRect()) &&
        this.ikebana != null) {
            this.ikebana = null;
            sumarPuntos();
            this.casaEntrega = null;
        }
    }
}

private void sumarMuerte() {
    this.muertes ++;
}

private void sumarPuntos() {
    this.puntaje += 5;
}

private void resetarJuego() {
    this.juegoTerminado = true;
    this.rasengan = null;
    this.casaEntrega = null;
    this.ikebana = null;
    quitarPuas();
    iniciarNinjas();
    this.sakura =
```

```
        new Sakura(anchoPantalla/2, altoPantalla/2, ciudad.getAnchoCalle()/2,
                  this.ciudad.getAnchoCalle()/2);

        this.puntaje = 0;
        this.muertes = 0;
    }

    private void colisionSakuraFloreria() {
        if(this.ikebana == null &&
        Rectangulo.colision(this.sakura.getRect(), this.floreria.getRect())) {
                // si no existe un ikebana y sakura colisiona con la floreria
            this.ikebana = new Ikebana(this.sakura.getX(), this.sakura.getY());
                // se crea un ikebana con las x e y de sakura
        }
    }

    private void movimientoIkebana(){
        if (this.ikebana!=null){
            this.ikebana.setX(this.sakura.getX());
            this.ikebana.setY(this.sakura.getY()+5);
        }
    }

    private void cambiarVelocidadNinjas(){
        if (this.puntaje==35) {
            for (int i = 0; i < ninjas.length; i++) {
                if (this.ninjas[i]!=null) {
                    this.ninjas[i].setVelocidad(2);
                }
            }
        }
    }

    @SuppressWarnings("unused")
    public static void main(String[] args)
    {
        Juego = new Juego();
    }
}
```

Conclusiones

El presente trabajo logró cumplir con las pautas principales como son: que el personaje Sakura inicie en el centro de la pantalla, pueda moverse por las calles e interactúe con los ninjas y pueda utilizar la técnica Rasengan, entre otras.

A su vez, implementamos satisfactoriamente funcionalidades extras como los obstáculos, la florería, una pequeña dificultad adicional, música de fondo, un menú que cuenta con un fondo de pantalla y donde se regresa al terminar la partida.

Por otra parte aprendimos que el mismo proyecto puede ser desarrollado de distintas formas y que trabajar en equipo enriquece la resolución de problemas. Gracias a que hay distintas perspectivas y formas de pensar un mismo código. Como resultado final, las implementaciones suelen ser más eficaces que las iniciales.

Finalmente gracias a este proyecto adquirimos nuevas prácticas como trabajar con repositorios, lo cual mejoró notablemente el trabajo en equipo y mejoramos nuestras habilidades interpersonales.