Trabajo práctico de Programación I

Universidad Nacional General Sarmiento
“Sakura Ikebana Delivery”

Integrantes: Matías Kalil Gómez, Matías Leandro Avila y Bertoni Jose Ignacio.
Legajos: 2018/41874133**,** 2020/39490535, 2019/38893843
Emails: matias_gomezu@hotmail.com ; leandroavmmo@gmail.com ; bertonigna@gmail.com
Profesoras: Verónica Moyano y Leonor Gutiérrez

Introducción

El trabajo práctico de Programación 1, del primer semestre del año 2021, es la realización de un juego llamado *"Sakura Ikebana Delivery"*. Este juego cuenta con un personaje principal llamado Sakura Ikebana. Nuestro personaje trabaja en una florería y está encargada de hacer las entregas de los ramos florales en las casas que estarán marcadas en el mapa. Se moverá entre las calles de este mapa para poder llegar a las manzanas en donde estarán las casas a las cuales deberá realizar la entrega de estos ramos.
En las calles del juego habrá ninjas que se moverán por ellas, sí sakura choca con alguno de estos ninjas perderá el juego. Nuestro protagonista contará con un poder especial que se activa al presionar la barra espaciadora: el Rasengan, el cual podrá lanzarse de a 1 sin limitaciones. Éste se lanzará en la dirección a la que sakura esté desplazándose, y si choca con un ninja lo elimina de la calle por un tiempo corto. El objetivo de este juego será hacer un puntaje igual a 20, donde cada entrega nos sumará 5 puntos en el contador de puntaje. Cuando esta condición esté cumplida, se ganará este juego.

Descripción

. Clase Sakura

Hacemos referencia al objeto "Sakura". Las variables de instancia: "x", "y", "alto","ancho", "direccion","angulo","sakura", "movimiento", "imagen4", "contNinjaElim";" ramo".

Siendo "x","y", "angulo" y "movimiento" de tipo double, "alto", "ancho", "direccion" y "contNinjaElim" son de tipo int, "sakura"y "ramo" es de tipo objeto Image, y "imagen4" que es de tipo String.

Los métodos aplicados en la clase Sakura son:

- **dibujar:** Este método de clase publico se encarga de dibujar el sakura
- **moverDerecha:** Este método de clase privado se encarga de mover el sakura a la derecha y limitar los movimientos del mismo según el entorno y el arreglo de manzanas.Además, genera una dirección con un valor entero predeterminado (1).
- **moverIzquierda:** Este método de clase privado se encarga de mover el sakura a la izquierda y limitar los movimientos del mismo según el entorno y el arreglo de manzanas.Además, genera una dirección con un valor entero predeterminado (2).
- **moverArriba:**Este método de clase privado se encarga de mover el sakura hacia arriba y limitar los movimientos del mismo según el entorno y el arreglo de manzanas.Además, genera una dirección con un valor entero predeterminado (3).
- **moverAbajo:**Este método de clase privado se encarga de mover el sakura hacia abajo y limitar los movimientos del mismo según el entorno y el arreglo de manzanas.Además, genera una dirección con un valor entero predeterminado (4).

- **movimientoRango**:Este método de clase publico se encarga de administrar todos los movimientos del personaje, y utilizarlos cuando esté presionado su debida tecla.
- **movimientoRangoManzanas:** Este método de clase privado se encarga de corroborar si el sakura está chocando o no con el la lista de objetos manzana.
- **colision:** Este método de clase booleano público se encarga de devolver si colisiona o no entre dos cuadrados.En caso afirmativo, true. Al contrario, false.
- **colisionCasa:** Este método de clase booleano se encarga de confirmar si, colisiona o no el sakura con el tipo de casa que corresponda.
- **sakuraDispara:** Este método público genera un nuevo rasengan con las direcciones del sakura en la posición actual.
- **habilidadEspecialRasengan:** Este método de clase publico se encarga de que cuando se aprieta una tecla especifica (espacio), se generar el rasengan. También se encarga de mover y limitar la trayectoria del rasengan por el mapa, ya sea cuando se pasa el entorno o del perímetro de las manzanas. Además, cuando se genera la colisión del rasengan con el ninja ambos se vuelven nulos.

Problemas Encontrados

El problema que encontramos con este objeto fue en la implementación de los límites de las manzanas. Al dibujar las manzanas y al hacer las colisiones del sakura con las manzanas no siempre funcionaban bien. El problema era que a veces chocaba antes o después, dependiendo de la dirección que este. Solucionamos el problema, notando que el radio del cuadrado de manzana no era el problema, sino que el problema era que había que modificar el x y el y del sakura o de la manzana.

. Clase Ninja

Hacemos referencia al objeto “Ninja” . Las variables de instancia: “x”,” y”,” alto” ,” ancho”,” velocidad”,” dirección”,” ninja”, “ángulo”.

Siendo “x”, “y”, “ángulo”, “velocidad” de tipo double, “alto”, “ancho”, “direccion” de tipo int, y “ninja” de tipo objeto Image.

Los métodos aplicados en la clase Ninja son:

- **generarNinjas:** Este método se encarga de dibujar los ninjas.
- **respawnNinjas:** Este método se encarga de recorrer el arreglo de ninjas, y verifica si alguno es nulo. En caso de que no haya nulos, se invoca al método **generarNinjas** y dibuja ninjas en esa posición; en caso de que se encuentren nulos se verifica la posición del arreglo y en base a esa posición se recrea el ninja.
- **mover:** Este método se encarga de mover el ninja en base a su dirección
- **moverDerecha:** Este método es para mover el ninja de izquierda hacia la derecha.
- **moverIzquierda:** Este método es para mover el ninja de derecha hacia la izquierda.

- **moverArriba:** Este método sirve para mover el ninja desde abajo hacia arriba.
- **moverAbajo:** Este método sirve para mover el ninja desde arriba hacia abajo.
- **colisionRasengan:** Este método nos devuelve un booleano indicando si se produce la colisión entre el rasengan y algún ninja, devolviendo “true” en caso de que se haya producido la colisión y “false” en caso de que no se produzca la colisión.
- **colisionSakura:** Este método nos devuelve un booleano indicando si se produce la colisión entre sakura y algún ninja, devolviendo “true” en caso de que se haya producido la colisión y “false” en caso de que no se produzca la colisión.

Problemas Encontrados

Uno de los principales problemas fue el respawneo de los ninjas. La solución la abordamos con la consulta a la profe. Determinamos que verificando si se encontraba nulo un ninja en el arreglo ninjas, nos de la posición en ese arreglo de ese ninja, y en base a esa posición se respawnear el ninja correspondiente.

. Clase Rasengan

Hacemos referencia al objeto “Rasengan”, que es lanzado por sakura, con las variables de instancia: “x”, “y”, “radio”, “velocidad”, “dirección”, “imagen4”.

Siendo “x”, “y”, “radio”, “velocidad” de tipo double, “direccion” de tipo int, y “iamgen4” de tipo objeto Image.

Los métodos aplicados en la clase Rasengan son:

- **mover:** Este método se encarga de mover el rasengan en base a su dirección.
- **moverDerecha:** Este método es para mover el rasengan de izquierda hacia la derecha.
- **moverIzquierda:** Este método es para mover el rasengan de derecha hacia la izquierda.
- **moverArriba:** Este método sirve para mover el rasengan desde abajo hacia arriba.
- **moverAbajo:** Este método sirve para mover el rasengan desde arriba hacia abajo.
- **movimientoRangoManzasRasengan:** Este método es utilizado para el movimiento del rasengan entre las manzanas, es decir evita que el rasengan traspase las manzanas.
- **colisionRasenganManzana:** Este método verifica si se produce una colisión entre el rasengan y la manzana, devolviendo un booleano “true” en caso de que se produzca esta colisión y “false” en caso de que no se produzca.
- **rangoRasengan:** Este método verifica que, si se produce que el rasengan se pasa del rango de las manzanas, es decir si colisiona con las manzanas inmediatamente el rasengan pasa a ser nulo. Esto se verifica mediante la invocación del método **movimientoRangoManzasRasengan y/o** si la posición en X o Y en rasengan es menor o igual a 0 o si es mayor igual al alto o ancho del entorno.

Problemas Encontrados

Con este objeto no tuvimos ningún problema.

. <u>Clase Manzana</u>

Hacemos referencia al objeto “Manzana”, que es lanzado por sakura, con las variables de instancia: “x”, “y”, “alto”, “ancho”, “angulo”, “color”, “casas[]”, “plantas”.

Siendo “x”, “angulo” e “y” de tipo double, “alto” y “ancho” de tipo int, “color” de tipo objeto Color y “plantas” de tipo objeto Image. Además de “casas[]” un arreglo de objetos Casa.

Los métodos aplicados en la clase Manzana son:

- **dibujar:** este método se encarga de dibujar las manzanas, que no sean la manzana en la posición 3, 7 o 11 del arreglo del objeto Manzana.
- **dibujarEsq:** este método se encarga de dibujar las manzanas en la posición 3, 7 y 11 del arreglo de objetos Manzana.

<u>Problemas Encontrados</u>

Uno de los problemas que tuve, que fue señalado en una consulta con la profesora asignada, era que estaba pasando las imágenes de las casas desde el constructor de Manzana que es donde contengo el arreglo de objetos Casa. Estaban asignadas a variables que eran de tipo string que contenía el nombre, y según la posición le pasaba esa variable. La solución fue asignar la imagen directamente en la clase Casa.
Otro problema, que fue al comienzo del desarrollo de la clase Manzana, fue el crear el arreglo de objetos Casa dentro del dibujador, en vez de dentro del constructor de Manzana, por lo cual cada vez que dibujaba un Objeto Manzana del arreglo manzanas dibujaba todo el contenido de ese arreglo de casas.

. <u>Clase Casa</u>

Hacemos referencia al objeto “Casa”, que se usa para dibujar casas dentro del objeto Manzana, con las variables de instancia: “x”, “y”, “alto”, “alto”, “ancho”, “tipo”, “angulo”, “casa”, “marca”, “casa1”, “casa2”, “casa3”, “casaObjetivo”.

Siendo “x”, “y”, “angulo” de tipo double, “alto”, “ancho”, “tipo” de tipo int, “casa” de tipo objeto Image, “marca” de tipo objeto Marca, “casa1”, “casa2”, “casa3” de tipo String, “casaObjetivo” de tipo booleano.

Los métodos aplicados en la clase Manzana son:

- **dibujar:** este método se encarga de dibujar las casas.

Problemas Encontrados

Uno de los problemas fue al cargar las imágenes de las casas. La solución fue colocar las imágenes bien dentro del proyecto, tenía las imágenes dentro del paquete juego, y no dentro de la carpeta src.
Otro problema fue al asignar la marca sobre la casaObjetivo, al tener diferente tipos de casas la marca se tendría que generar en distinta posición sobre la casa. La solución fue crear una variable tipo, que me dice cual casa es a la que le voy a poner la marca, y entonces con eso se determina la posición del objeto Marca, tres distintos posicionamientos uno para cada tipo de casa.

. Clase Marca

Hacemos referencia al objeto "Marca", que es la encargada de marcar en el mapa cuál es la casa objetivo a la que sakura debe entregar el ramo de flores, con las variables de instancia: "x", "y", "angulo", y "arrow".

Siendo "x", "y", "angulo" de tipo double, y "arrow" de tipo objeto Image.

Los métodos aplicados en la clase Manzana son:

- **dibujarMarca:** este método se encarga de dibujar la Marca.

Problemas Encontrados

Con este objeto no tuvimos ningún problema.

. Clase PantallaFinal

Hacemos referencia al objeto "PantallaFinal", que es una pantalla, que se activa cuando se gana el jugador suma 500 puntos, con las variables de instancia: "x", "y", "ancho", "alto", "colorPantalla".

Siendo "x", "y", de tipo double, "ancho" y "alto" de tipo int, y "colorPantalla" de tipo objeto Color.

Los métodos aplicados en la clase Manzana son:

- **dibujarPantallaFinal:** este método se encarga de dibujar la PantallaFinal.

Problemas Encontrados

Con este objeto no tuvimos ningún problema.

Implementación

Las relaciones entre los distintos objetos creados están establecidas en la clase Juego. Esta misma está codificada de la siguiente forma:

Dentro de public Juego():

- **En la línea 25:** Se instancia el objeto entorno.
- **De las líneas 29 a 31:** Se declaran los objetos: Pfinal, sakura, y rasengan.
- **En las líneas 36 y 37:** Se crean dos arreglos de objetos: Uno de objetos Manzana y otro de objetos Ninja.
- **De la 47 a la 58:** Se declaran las posiciones del arreglo de objetos Manzana.
- **De la 62 a la 68:** Se declaran las posiciones del arreglo de objetos Ninja.
- **En la línea 74:** Se llama al método de la clase Juego, llamado elegirCasaObjetivo() y se le asigna el valor retornado a una variable de esta clase. El método es el siguiente:

```
//Retorna la casaObj
//Ademas de cambiar el valor de la variable casaObjtivo en Casa a true.
public Casa elegirCasaObjetivo(Manzana manzanas[]) {
    //Eleccion de manzana random
    Random manzanaObjetivo = new Random();
    nm = manzanaObjetivo.nextInt(12);

    //Eleccion de casaObjetivo condicionando != 3 7 11
    if(nm != 3 && nm != 7 && nm != 11) {
        Random casaObjetivo = new Random();
        int co = casaObjetivo.nextInt(3);

        //Fijado de casaObtivo == true en la casa random elegida
        manzanas[nm].getCasas(co).setCasaObjetivo(true);

        //Retorno de casaObjetivo
        return casaObj = manzanas[nm].getCasas(co);

    } else {
        Random casaObjetivo = new Random();
        int co = 0;
        int ran = casaObjetivo.nextInt(2) + 1;
        if(ran == 1) {
            co = 3;
        } else {
            co = 4;
    }

    //Fijado de casaObjetivo == true en la casa random elegida
    manzanas[nm].getCasas(co).setCasaObjetivo(true);

    //Retorno de casaObjetivo
    return casaObj = manzanas[nm].getCasas(co);
     }
}
```

- **En la línea 79:** inicia el juego.

Dentro de public void tick():

- **De la linea 87 a la 98:** Se dibuja en el entorno del juego donde se ve la cantidad de puntos y los ninjas eliminados.

```
//Seteo del tamaÃ±o, color y tipo de Font.
entorno.cambiarFont("Arial", 20, Color.WHITE);   //Establece tipo de fuente , tamaño y color

//Dibujar en pantalla segÃºn puntaje.
//Aviso de cuando le falta 1 entrega para ganar!
if(puntaje >= 15) {    //condición que agrega una linea en el medio de la pantalla que notifica que falta un ramo para ganar
    entorno.escribirTexto("Puntaje: " + puntaje, 35, 25);
    entorno.escribirTexto("Ninjas eliminados: " + sakura.getcontNinjaElim(), 575, 25);
    entorno.escribirTexto("Â¡Una entrega mÃ¡s para ganar!", 210, 25);
} else {    //dibuja en pantalla cantidad de puntos obtenidos y cantidad de ninjas eliminados
    entorno.escribirTexto("Puntaje: " + puntaje, 35, 25);
    entorno.escribirTexto("Ninjas eliminados: " + sakura.getcontNinjaElim(), 575, 25);
}
```

- **De la línea 106 a 108:**
  En la línea 106 utiliza un método de la clase sakura que hace lo siguiente:

```
public void dibujar (Entorno e) {

    e.dibujarImagen(sakura, x, y, angulo); //dibuja la imagen del sakura en el entorno
    e.dibujarImagen(ramo, x-20, y+5, 6);  // dibuja la imagen del ramo en el entorno
}
```

- **En la línea 107:** utiliza este método de clase sakura:

```
public void movimientoRango(Entorno entorno,Manzana[] manzanas) {

    if (entorno.estaPresionada(entorno.TECLA_DERECHA)) {
        moverDerecha(entorno, manzanas);    //si se preciona tecla derecha dentro en entorno y fuera de las manzanas
    }else if (entorno.estaPresionada(entorno.TECLA_IZQUIERDA))  {
        moverIzquierda(manzanas);   //si se preciona tecla izquierda mover dentro entorno y fuera de las manzanas
    }else if (entorno.estaPresionada(entorno.TECLA_ABAJO)){
        moverAbajo(entorno,manzanas);  //si se preciona tecla abajo mover dentro entorno y fuera de las manzanas
    }else if (entorno.estaPresionada(entorno.TECLA_ARRIBA)){
        moverArriba(manzanas);              //si se preciona tecla arriba mover dentro entorno y fuera de las manzanas
    }

}
```

Este método de clase publico se encarga de administrar todos los movimientos del personaje, y utilizarlos cuando esté presionado su debida tecla. Así mismo, este método utiliza los métodos privados de la clase sakura moverDerecha, moverIzquierda, moverAbajo, moverArriba:

```
private void moverDerecha(Entorno x,Manzana[] manzanas) {
    if (!(this.x+10+this.ancho/2>x.ancho())) {  // verifica que solo se pueda mover por el entorno
          this.x+=this.movimiento;
    } if (movimientoRangoManzanas(manzanas)) { // verifica si colisiona o no con las manzanas
        this.x-=this.movimiento;
    }
    direccion=1; // valor determinado para derecha=1
    }
```

Se encarga de mover el sakura a la derecha y limitar los movimientos del mismo dentro del entorno y sin colisionar dentro del arreglo de manzanas.

```
private void moverIzquierda(Manzana[] manzanas) {
    if (!(this.x+1-this.ancho<=0)) {  // verifica que solo se pueda mover por el entorno
        this.x-=this.movimiento;
    }if (movimientoRangoManzanas(manzanas)) {  // verifica si colisiona o no con las manzanas
        this.x+=this.movimiento;
    }
    direccion=2; // valor determinado para derecha=2
    }
```

Se encarga de mover el sakura a la derecha y limitar los movimientos del mismo dentro del entorno y sin colisionar dentro del arreglo de manzanas.

```
private void moverAbajo(Entorno x, Manzana[] manzanas) {
    if (!(this.y+12+this.alto>x.alto())) {   // verifica que solo se pueda mover por el entorno
          this.y+=this.movimiento;
    }if (movimientoRangoManzanas(manzanas)) {  // verifica si colisiona o no con las manzanas
        this.y-=this.movimiento;
    }
    direccion=4; // valor determinado para derecha=4
}
```

Se encarga de mover el sakura hacia abajo y limitar los movimientos del mismo dentro del entorno y sin colisionar dentro del arreglo de manzanas.

```
 private void moverArriba(Manzana[] manzanas) {
     if (!(this.y-50-this.alto<0) ) {   // verifica que solo se pueda mover por el entorno
         this.y-=this.movimiento;
     }if (movimientoRangoManzanas(manzanas)) {  // verifica si colisiona o no con las manzanas
         this.y+=this.movimiento;
     }
     direccion=3;   //valor determinado para derecha=3
 }
```

Se encarga de mover el sakura hacia arriba y limitar los movimientos del mismo dentro del entorno, sin colisionar dentro del arreglo de manzanas.

```
private boolean movimientoRangoManzanas(Manzana[] manzanas) {
    int cont=0;
    for (int i=0 ; i<manzanas.length;i++) {
        if (colision(this.x+90,this.y+70,this.ancho,this.alto,manzanas[i].getX(),manzanas[i].getY(),manzanas[i].getAlto()+28,manzanas[i].getAncho()+15)==true) { //verifica la colision
            cont+=1;   //los va contando
        }else {
            cont=cont+0;
        }
    }
    if (cont>0) {   //si la colision es mayor a 1 es verdadera
        return true;
    }else {
        return false; // si nunca colisiona con ninguna manzana es false
    }

  }
```

Para que en cualquiera de los cuatro movimientos de sakura, no se pudiera caminar sobre las manzanas, se implementó un método private de la clase sakura movimientoRangoManzanas() que da true si alguna de las casas colisiona con el sakura y false si no, para poder hacer la restricción del movimiento en tal caso.

- **En la línea 108:** sakura utiliza el método habilidadEspecialRasengan() :

```
public void habilidadEspecialRasengan(Entorno entorno,Manzana[] manzanas, Rasengan[] rasengan,Ninja[] ninjas) {
    if (entorno.sePresiono(entorno.TECLA_ESPACIO)) {  //si se presiona espacio
        boolean dispararRasengan = false;
        for (int i = 0; i < rasengan.length && !dispararRasengan; i++) {
            if (rasengan[i] == null)
            {
                rasengan[i] = sakuraDisparar(); //genera un nuevo rasengan con posicion del sakura
                dispararRasengan = true;
            }
        }
    }

    //dibuja el rasengan y desaparece segun los limites en X e Y
    for (int i = 0; i < rasengan.length; i++) {
        if (rasengan[i] != null) {
            rasengan[i].mover();        //mueve el rasengan
            rasengan[i].Dibujar(entorno);  // dibuja el rasengan en el entorno
            rasengan[i].rangoRasengan(manzanas, rasengan, i, entorno);   //limita el rango de trayectoria del rasengan
        }
    }

    for (int i = 0; i < ninjas.length; i++) {
        if(ninjas[i]!=null && rasengan[0]!=null) {
                if(ninjas[i].colisionRasengan(rasengan[0],ninjas[i])==true) {  // si colisiona el rasengan con algun ninja
                    rasengan[0]=null;
                    ninjas[i]=null;
                    contNinjaElim = contNinjaElim  + 1;
                }
            }
        }

 }
```

En el cual, cuando se aprieta una tecla especifica (espacio), y se genera el rasengan por medio del método de la clase sakura sakuraDisparar. También se encarga de mover y limitar la trayectoria del rasengan por el mapa, ya sea cuando se pasa el entorno o del perímetro de las manzanas. Además, cuando se genera la colisión del rasengan con el ninja, ambos se vuelven nulos, contando en este además la cantidad de ninjas totales eliminados. Este método utiliza métodos de la clase Rasengan; tales como mover() (mueve el rasengan dentro del entorno en diferentes direcciones) , dibujar() (los dibuja en entorno), rangoRasengan() (pone nulo el rasengan si y sólo si colisionan con una manzana o con los bordes del entorno),También tiene un método de la clase Ninja

colisionRasengan (su función es verificar por medio de true o false si colisionan el rasengan con el ninja):

Dentro de la clase sakura

```
public Rasengan sakuraDisparar() {
    return new Rasengan(this.x,this.y,direccion,imagen4); // retorna un nuevo rasengan con x e y del sakura
}
```

Dentro de la clase Rasengan

```
public void rangoRasengan(Manzana[] manzanas, Rasengan[] rasengan, int i, Entorno entorno ) {
    if(rasengan[i].getX() <= 0 || rasengan[i].movimientoRangoManzasRasengan(manzanas, rasengan[i]) ) //
    {
        rasengan[i] = null;  // si se pasa del margen derecho del entorno o choca con una manzana se vuelve nulo
    }
    else if(rasengan[i].getX()  >= entorno.ancho()|| rasengan[i].movimientoRangoManzasRasengan(manzanas, rasengan[i]))
    {
        rasengan[i] = null; // si se pasa del margen izquierdo del entorno o choca con una manzana se vuelve nulo
    }
    else if(rasengan[i].getY() <= 0 || rasengan[i].movimientoRangoManzasRasengan(manzanas, rasengan[i]))
    {
        rasengan[i] = null; // si se pasa del margen inferior del entorno o choca con una manzana se vuelve nulo
    }
    else if(rasengan[i].getY()  >= entorno.alto()|| rasengan[i].movimientoRangoManzasRasengan(manzanas, rasengan[i]))
    {
        rasengan[i] = null;  // si se pasa del margen superior del entorno o choca con una manzana se vuelve nulo
    }
}
```

Dentro de la clase Rasengan

```
public void mover() {
     if(this.direccion == 0 ) {  //caso que la direccion del sakura es 0 mover derecha
        moverDerecha();
    } else if (this.direccion == 1) {//caso que la direccion del sakura es 1 mover derecha
        moverDerecha();
    } else if (this.direccion ==2) {//caso que la direccion del sakura es 2 mover izquierda
        moverIzquierda();
    }else if (this.direccion == 3) {//caso que la direccion del sakura es 3 mover abajo
        moverAbajo();
    } else if (this.direccion == 4) {//caso que la direccion del sakura es 4 mover arriba
        moverArriba();
    }
}

private void moverDerecha() {
    if(this.x >= 800) {
        this.x = 0;
    }
    this.x+=this.velocidad; //aumenta la x
}

private void moverIzquierda() {
    if(this.x <= 0) {
        this.x = 800;
    }
    this.x-=this.velocidad; //disminuye la x
}

private void moverArriba() {
    if(this.y >= 800) {
        this.y = 0;
    }
    this.y+=this.velocidad; //crece la y
}

private void moverAbajo() {
    if(this.y <= 0) {
        this.y = 800;
    }
    this.y-=this.velocidad; //decrece la y
}
```

Dentro de la clase Rasengan

```
public void Dibujar(Entorno entorno) {
    entorno.dibujarImagen(imagen4, x, y, radio); // dibuja el rasengan con la imagen
    }
```

- **De la línea 111 a 115:** Se recorre el arreglo de objetos de Manzana, y se dibuja dependiendo qué posición del arreglo de objetos Manzana se va a dibujar. Según esta condición establecida, se usa uno de los dos métodos para dibujar, de la clase Manzana.

```
//Dibujar manzanas en entorno segun si es 3, 7 o 11 || si es 3, 7 o 11 le dibuja 2 casas
for(int i=0;i<manzanas.length;i++) {
    if(i != 3 && i != 7 && i != 11) {
    manzanas[i].dibujar(entorno);
    } else {
        manzanas[i].dibujarEsq(entorno);
    }
}
```

Donde dibujarEsq() es el siguiente método:

```
public void dibujarEsq(Entorno e) {

    //Dibujar mazana
    e.dibujarRectangulo(x, y, ancho, alto, angulo, color);

    //Dibujar plantas
    e.dibujarImagen(plantas, x+20, y-35, 0);
    e.dibujarImagen(plantas, x-10, y, 0);
    e.dibujarImagen(plantas, x+40, y, 0);
    e.dibujarImagen(plantas, x+20, y+35, 0);

    // Dibujar casas en la manzana
    for(int i = 0;i<casas.length;i++) {
        if(i == 3 || i == 4) {
           casas[i].dibujar(e);
        }
    }
}
```

y dibujar() es el siguiente método:

```
    //metodo que dibuja las manzanas con las casas
    public void dibujar(Entorno e) {

        //Dibujar manzana
        e.dibujarRectangulo(x, y, ancho, alto, angulo, color);

        //Dibujar plantas
        e.dibujarImagen(plantas, x-42, y-35, 0);
        e.dibujarImagen(plantas, x-42, y-15, 0);
        e.dibujarImagen(plantas, x-25, y-15, 0);

        // Dibujar casas en la manzana
        for(int i = 0;i<casas.length;i++) {
            if(i != 3 && i != 4) {
               casas[i].dibujar(e);
            }
        }
    }
```

En la línea 119: un ninja del arreglo ninjas que es para poder aplicar el método de la clase juego, metodo respawnNinjas:

```
public void respawnNinjas(Ninja[] ninjas, Entorno entorno) {
    for (int i = 0; i < ninjas.length; i++) {
        if (ninjas[i] !=null) {
            ninjas[i].generarNinjas(ninjas, entorno, i); // dibuja y mueve los ninjas en entorno
        }
        else{
              switch(i) {      //busca el caso y lo regenera otra vez al ninja en una posicion mas atras segun
                               //el caso.
              case 0: ninjas[i] = new Ninja(-500,20*29,2, 1);  //caso ninja[0]
                      break;
              case 1: ninjas[i] = new Ninja(1500, 20*10,2,2);  //caso ninja[1]
                      break;
              case 2: ninjas[i] = new Ninja(-500, 20*20,2, 1); //caso ninja[2]
                      break;
              case 3: ninjas[i] = new Ninja(40*10,-500,2,3);   //caso ninja[3]
                      break;
              case 4: ninjas[i] = new Ninja(18*10,1000,2,4);   //caso ninja[4]
                      break;
              case 5: ninjas[i] = new Ninja(62*10,1000,2,4);   //caso ninja[5]
                      break;
              }
        }
     }
}
```

En este método además se implementa el método generarNinjas de la clase ninjas que dibuja y mueve todos los ninjas por las calles en direcciones diferentes en cada calle. Regenerandose/reapariciendo cuando toque el extremo inverso de la salida inicial.

```
    private void generarNinjas(Ninja[] ninjas, Entorno entorno,int i) {
            ninjas[i].Dibujarse(entorno);    // dibuja los ninjas en entorno
            ninjas[i].mover();  //mueve los ninjas
    }
```

- **De la 128 a 133:** Un condicional que evalúa si sakura está colisionando con la casa donde debe entregar el ramo de flores. El condicional es el siguiente:

```
// Condicinal para colision de sakura con casaObj
//Asignacion de puntos por llegada a casaObj y Reseteo de casaObj
if(sakura.colisionCasa(casaObj)) {
    casaObj.setCasaObjetivo(false); // pone variable casa objetivo en falso de la casaObj anterior
    casaObj = elegirCasaObjetivo(manzanas);  // eligue una nueva casaObj
    casaObj.setCasaObjetivo(true);   //pone variable casa objetivo en true de la nueva casaObj
    puntaje = puntaje + 5;  //suma 5 al puntaje total
}
```

Donde también se usa el método elegirCasaObjetivo() de la clase Juego, antes mencionado

- .**De 135 a 140:**
  Este bucle recorre la cadena de ninjas y si colisiona algún ninja con sakura se cierra el juego, el método colisionSakura es de la clase Ninja y devuelve true o false dependiendo si colisionó o no.

```
for (int i = 0; i < ninjas.length; i++) {
    if(ninjas[i]!=null && sakura!=null)  { //si el no es nulo y sakura no nulo
        if(ninjas[i].colisionSakura(sakura,ninjas[i])==true) { //verifica si sakura colisiona con algun ninja
            entorno.dispose();    //cierra automaticamente el entorno
        }
     }
}
```

- **De la 147 a la 150:** Un condicional que verifica si el puntaje es mayor o igual que 20, si esto pasa, dibuja la pantalla final del juego.

```
if(puntaje >= 20) {
    Pfinal.dibujarPantallaFinal(entorno); //dibuja el objeto pantalla final
    entorno.cambiarFont("Arial", 40, Color.BLACK);   //settea tipo de fuente, tamaño y color de la misma
    entorno.escribirTexto("Â¡Ganaste el juego!", 220, 300);  // escribre sobre la pantalla un texto
}
```

Conclusión

En conclusión todos los integrantes afirman haber ganado un mejor manejo de objetos. Aprender de manera correcta la separación de los métodos específicos de cada objeto, sumado a desarrollar una mejor organización a la hora de la resolución de los muchos errores obtenidos mediante el desarrollo, lo cual nos deja a los integrantes de este grupo una experiencia positiva. Los errores del trabajo en el grupo, primeramente se trataban de resolver en solitario y si no se podía, luego se abordaban grupalmente. La distribución del trabajo de desarrollo, traducido al trabajo en conjunto para poder cumplir las fechas estipuladas, desembocó en una retroalimentación entre integrantes lo que generaba mejoras en nuestro código. Cerrando esto, el trabajo práctico deja una sensación positiva en cada integrante del grupo.