Universidad Nacional de General Sarmiento

Programación 1

Informe

TRABAJO PRÁCTICO

Juego Lost Galaxian

Apellido y nombre de los integrantes del grupo:

Berini Bruno Nicolás
DNI: 45867142
Email: berinibruno00@gmail.com

Guerra Josué
DNI: 41917055
Email del estudiante: josuehguerra@gmail.com

Docentes: Leonardo David Waingarten, Leonor Silvia Gutierrez

Comisión: 02

El proyecto Lost Galaxian

A continuación, documentaremos como fue todo el proceso de desarrollo del videojuego propuesto como trabajo practico en la materia programación 1. El mismo, cuenta con su respectiva clase principal (Main) donde se ejecuta el juego, y las demás clases necesarias para que se generen los objetos requeridos para su ejecución. El juego se ha desarrollado en el lenguaje Java. Se trata de un videojuego tipo Arcade clásico, donde el jugador controla una nave de combate (Astro-MegaShip) creada especialmente para combatir a los Destructores Estelares (Enemigos) con proyectiles.
El jugador puede ganar si se deben eliminan varios Destructores Estelares hasta llegar al jefe final, y derrotarlo. Y si la nave pierde todos sus puntos de vida, se destruye y el jugador pierde. Sin embargo, se cuentan con varios ítems para sobrevivir como vida extra. También se pueden sumar mas puntos con las estrellas.

Descripción de cada clase

Clase Juego (main):

Antes de comenzar con la clase, antes, desde la línea 3 hasta la 7 se importan las bibliotecas y los objetos necesarios para la correcta ejecución del juego, como la interface, el java.awt.color para la pantalla, etc.
Luego de importar lo necesario, nos introducimos en la clase, donde crearemos el entorno, los objetos, y las variables generales que utilizará la clase Juego.
Primeramente, creamos varios arreglos de Objetos tipo: Asteroide, Enemigo. De Proyectil Enemigo creamos dos arreglos porque uno lo vamos a usar para los proyectiles que disparan los Destructores Estelares comunes y otro es para los que dispara el Jefe Final.
Luego la pantalla se define en ancho = 800 por alto = 600.

Una vez dentro de la clase Juego, se crean variables de instancia tales como int puntaje o int velocidadEnemigos, así como también se crean objetos que luego usaremos a lo largo del desarrollo de esta clase, como el objeto fondo (que en realidad es un archivo .gif), gameover, y ganaste de la clase Image, o el objeto Nave de la clase Nave.
Respecto a esta última, la nave es una sola ya que es lo que controla el jugador, solo se usa una nave por partida. Además, cuenta con un proyectil que se recarga cuando sale de la pantalla del juego, para que el juego tenga mayor dificultad.
Respecto a otras variables, se crea un arreglo de objetos tipo Asteroide, así como un arreglo de objetos Enemigo, y otro arreglo más de objetos Proyectil. Cada uno de estos usa un ciclo for para crearse en el entorno. Este lo inicializamos luego de declarar lo anteriormente mencionado con entorno.iniciar().

Dentro del tick() se encuentran, de la línea 106 a la 120 los controles de la nave, derecha, izquierda y con la barra espaciadora disparar proyectilNave.

Luego se continua por los ítems, tales como extra vida (en el juego se representa con imagen del corazón) y el otro ítem extra puntaje (representado por una imagen de estrella)

Continuando en el desarrollo del tick() , siguen los proyectiles.
Primero, en el proyectilNave se define que si es distinto de nulo se genere uno, y si esta fuera del mapa (del entorno) se destruya u elimine el mismo.
Respecto a ionesEnemigos, se crean cuando se verifica la nulidad de los mismos. Luego se agregan condicionales, tales como, si colisiona a la nave (jugador), le quita vida haciéndole el daño definido al principio de la clase juego. Eso, lo realiza con el método Colisiona de la clase Detector. Para explicarlo brevemente, lo que hace es comparar la distancia de dos objetos y si es menor que cierto rango establecido, eso nos da True.

Mas abajo del código se define otro if que establece el ionesEnemigos[i] en null, en caso de que se salga del entorno y no haya eliminado antes debido a la colisión con la nave.
Respecto a los ionesJefeFinal se utilizan prácticamente los mismos condicionales, solo que este se activa cuando entramos en la dificultad/fase del Jefe Final.
En Asteroide también se utilizan similares condicionales, con la diferencia que en este se utiliza otro más extra que verifica si el asteroide es visible en el eje vertical de la pantalla. En caso que no lo sea, devuelve el asteroide a la posición Y = 0.

El código sigue por la sección de Enemigos, donde se genera cada uno, con dibujo y movimiento (mover()).Dentro del ciclo que genera los objeto Enemigo es donde decidimos introducir uno de los requisitos opcionales, que es Destructores Estelares Kamikazes, en donde, si el enemigo se encuentra muy cerca de la nave, toma su posición para acercarse y colisionar con la misma. Esto lo logramos con el detector de colisiones y el método cambiarAngulo (que recibe el X y el Y que le pasemos como parámetro) de la clase Enemigo. Otro condicional es que el enemigo regresa hacia arriba si se está por ir fuera del entorno. Eso lo verificamos con el método estaEnEntorno de la clase Detector.
En la siguiente parte, generamos otro ciclo simplemente para comprobar si esta cerca de otro Enemigo y cambiamos la dirección con cambiarDireccion en caso de que eso sea cierto.
Respecto a si colisiona con un ProyectilNave, lo que devuelve es una suma de puntajes para el jugador y el Enemigo queda en null. Si colisiona con un asteroide, también usa

cambiarDireccion, y si el Enemigo colisiona con la nave, a la nave se le resta vida, pero el Enemigo queda en null directamente.
En cuanto a Destructores Estelares normales, el ultimo condicional que se encuentra regenera enemigos cada cierto tiempo establecido cuando el juego no ha sido ganado aún.

En la siguiente sección nos encontramos con los condicionales del Jefe Final, que se pueden resumir en los siguientes 3 puntos:
-El primer condicional detecta si hubo una coalision, y en caso de haberla, le resta vida al jefe y le suma puntos al jugador (nave).
-El segundo condicional verifica si la vida del Jefe es 0, y si lo es, lo pasa a nulo y el juego se da por terminado con una victoria.
-Por último, en el "else" es donde generamos la barra de vida del Jefe Final.

Luego, resumidamente, vienen las secciones de nave, donde se agrega o se quita vida según si hubo o no colision con algún Enemigo o proyectil enemigo, y respecto a los Niveles, van cambiando cuando se llega a cierto puntaje en el juego, siendo el ultimo nivel el más difícil ya que es donde se encuentra el Jefe Final.

Clase Nave

En esta clase se crea la Nave (Astroship), que es controlada por el jugador mediante los controles ya definidos en la clase Juego.

En la Clase primero definimos las siguientes variables de instancia:
double x,y;
double escala; (sirve para ajustar la escala de la imagen)
double angulo;
int vida;
Image img1;
Entorno entorno;
Luego en el constructor establecemos los siguientes argumentos de Nave(Entorno ent,double x, double y, double esc)

```
{
        this.vida=100;
        this.x = x;
        this.y = y;
        this.escala = esc;
        entorno = ent;
```

```
        img1 = Herramientas.cargarImagen("nave.png");
}
```

En el objeto img1 de tipo Imagen se carga la imagen de la nave con el método cargarImagen de la clase Herramientas que ya viene en el proyecto base.

```
public void dibujarse(){
        entorno.dibujarImagen(img1, this.x, this.y, this.angulo, escala);
    }
```

La función dibujarse retorna vacío ya que solo muestra la nave en pantalla con el método dibujarImagen de la clase entorno.

Por último, se mueve la nave con el método mover, izquierda con X en negativo y derecha con X en positivo. La nave solo se mueve sobre el eje horizontal X.

```
public void mover(double direccion) {
        if(!Detector.tocarBordeEjeX(this.x,entorno))
            this.x += direccion;
        else
            if(this.x > 0)
                this.x -= 10;
            else if(this.x < 5)
                this.x += 15;
```

Clase Enemigo

En la clase Enemigo se definen las siguientes variables de instancia:

```
double x,y;
    double angulo;
    double escala;
    double velocidad;
```

int vida; (esta variable solamente se usa para el jefe final, ya que los enemigos se eliminan directamente con un solo proyectil de la nave que colisione contra ellos)

```
  Image img;
  Entorno entorno;
```

Luego, similar a otras clases, cuenta con un método dibujar() que dibuja la imagen. Sin embargo, se agregan otra serie de métodos que resulta importante de destacar para entender lo que hacen.

En mover() se mueve el enemigo en el eje X e Y según una serie de condicionales if. Uno de ellos, es que cambia de angulo si toca el borde (medido en el eje horizontal). Para cambiar de angulo utilizamos el método void cambiarDireccion.

En moverSoloHorizontal() que se usa para el jefe final, el enemigo se mueve solamente sobre el eje X, ya que es enorme, y si va "bajando" en el eje Y hasta tocar la nave, sería injusto ya que siempre perderíamos. Usa el mismo condicional que mover()

Como se observa en el código del juego, hay dos métodos similares, cambiarDireccion y cambiarAngulo. La diferencia es que en cambiarAngulo se reciben dos parámetros, un X, Y para cambiar el ángulo del objeto en base a esas posiciones que pasemos.

Clase ProyectilEnemigo

Se utilizan casi las mismas variables de instancia que la clase Nave, ya que en vez de usarse un int vida se usa un double velocidad.

```
double x,y;
double angulo;
double escala;
double velocidad;
Image img;
Entorno entorno;
```

En el constructor lo que destaca es el ángulo, que se usa el valor de Pi ya que se mide en radianes.

Se utiliza un método Dibujar para poder dibujar la imagen de la variable de instancia.

Respecto al método mover() es distinto que otras clases, porque solamente se puede mover en el eje vertical, hacia abajo, donde aumenta el eje Y hacia el infinito positivo.

Clase ProyectilNave

En proyectil nave, las variables de instancia son similares a las de proyectilEnemigo

```
double x,y;
double angulo;
double escala;
double velocidad;
Image img;
Entorno entorno;
```

En el método dibujar se genera la imagen img.

Y en el método mover, el Y es negativo ya que ya que el proyectil sube hacia Y infinito negativo.

Clase Asteroide

La clase Asteroide usa las siguientes variables de instancia

```
    double x,y;
    double angulo;
    double escala;
    double velocidad;
    Image img;
    Entorno entorno;
```

A continuación, se presenta el constructor de Asteroide, que usa Entorno, escala y velocidad. Respecto al ángulo, como se comento en otra clase, se usa el Math.PI ya que se mide en radianes.

```
  public Asteroide(Entorno ent, double esc, double vel)
    {
        this.entorno=ent;
        this.escala=esc;
        this.velocidad=vel;
        this.x=Math.random()*entorno.ancho();
        this.y=-Math.random()*entorno.alto()*0.2;

        img = Herramientas.cargarImagen("asteroide.png");
        angulo=Utiles.generarRandomDouble(0,Math.PI/2);
        velocidad=Utiles.generarRandomDouble(1,5);

  }
```

Al igual que en otros métodos, el método dibujar dibuja la imagen

```
    public void dibujar()
    {
        entorno.dibujarImagen(img, this.x, this.y, this.angulo,
this.escala);

    }


    public void mover() {
        this.x += Math.cos(this.angulo)*velocidad;
        this.y += Math.sin(this.angulo)*velocidad;
```

```
        if(this.x > entorno.ancho()*1.05) {
            this.x=-entorno.ancho()*0.05;
        }
        if(this.x < -entorno.ancho()*0.05) {
            this.x=entorno.ancho()*1.05;
        }
        if(this.y > entorno.alto()*1.05) {
            this.y=-entorno.alto()*0.05;
        }
        if(this.y < -entorno.alto()*0.05) {
            this.y=entorno.alto()*1.05;
        }

    }
```

Método cambiarAngulo

```
    public void cambiarAngulo() {
        if(this.angulo > Math.PI/2) {
            this.angulo=Math.PI/4;
        }
        else
        {
            this.angulo=3 * Math.PI/4;
        }
    }
```

Clase Detector

En la clase Detector no lleva variables de instancia. Y los métodos que están definidos sirven para saber si un objeto está tocando algún borde del eje x horizontal de la pantalla (método tocarBordeEjeX), otro método sirve para comparar dos objetos y verificar si colisionan (método Colisiona) . Por último, hay un método que detecta si un objeto está dentro de la pantalla, es decir, dentro del entorno (estarEnEntorno). Al ser métodos de verificación, todos devuelven un booleano.

Algo que tuvimos mucho en cuenta a la hora de corregir finalmente esta sección del código fue en el método Colisiona, que estábamos tratando de implementar un sistema quizás demasiado complejo y la solución fue optar por lo sencillo y eficaz de medir la distancia de los centros.

Clase Utiles

Se definen los métodos generarRandom, que funciona pasándole como parámetros un valor mínimo y un máximo enteros, y generarRandomDouble,, que tiene los mismos parámetros, pero en formato double.
Luego con otro método se obtienen los segundos actuales con el método segundosActuales. Antes de utilizarlo se importa java.util.Calendar. SegundosActuales lo utilizamos en la clase Juego para que revise cada 30 segundos a los enemigos si no se ganó el juego aun.

```
            if(enemigos[i] == null && jefeFinal == null &&
!faseJefeFinalGanado &&(
            Utiles.segundosActuales() == 00 ||
            Utiles.segundosActuales() == 30 )) {
                enemigos[i] = new Enemigo(entorno, 0.2 ,3, 10);
            }
```

Por último, utilizamos un método booleano hayEnemigos que verifica si quedo algún enemigo “vivo”.

Clase ItemPuntaje y Clase ItemVida

En este caso, al ser las dos muy similares, se pueden agrupar. Las variables de instancia que usa cada una son las mismas.

```
double x,y;
double angulo;
double escala;
double velocidad;
Image img;
Entorno entorno
```

Respecto al angulo y velocidad, se usa el método generarRandomDouble de la clase Utiles para que siempre sea aleatorio el ángulo y velocidad con el que se dibujen y se muevan por la pantalla los dos ítems.
Cuentan con método dibujar y mover también.

Implementacion

El código fuente con los respectivos comentarios se puede ver desde este link a la carpeta de Drive compartida:
https://drive.google.com/drive/folders/1bUncsqPUmmxRGNeXt-_D_irI24jAPBtE?usp=sharing

Conclusiones

Un reto a lo largo del desarrollo definitivamente fue hacer que funcione todo en conjunto. Esto explica porque en muchas condicionales preguntamos si los objetos que utilizan son distintos de nulo, incluso cuando ya se hizo al principio. Se debe principalmente a que a medida que el juego avanza puede que este haya sido eliminado o afecte a todo el entorno, por ejemplo, de pasar a combatir con los Destructores Estelares a pasar a combatir con el jefe Final.