Programación I - Primer Semestre 2023
Trabajo Práctico: Lost Galaxian

En un lugar muy alejado de la galaxia existen unas tropas de naves llamadas Destructores Estelares, que invaden planetas y destruyen todo a su paso. Para combatir a los Destructores Estelares, la Súper Patrulla Espacial creó una nave de combate llamada Astro-MegaShip y nos encomendó la creación de una aplicación para entrenar a nuevos pilotos de la nave.

El objetivo de este trabajo práctico es desarrollar un video juego en el cual la Astro-MegaShip elimine la mayor cantidad de Destructores Estelares, sin ser destruida en el intento.

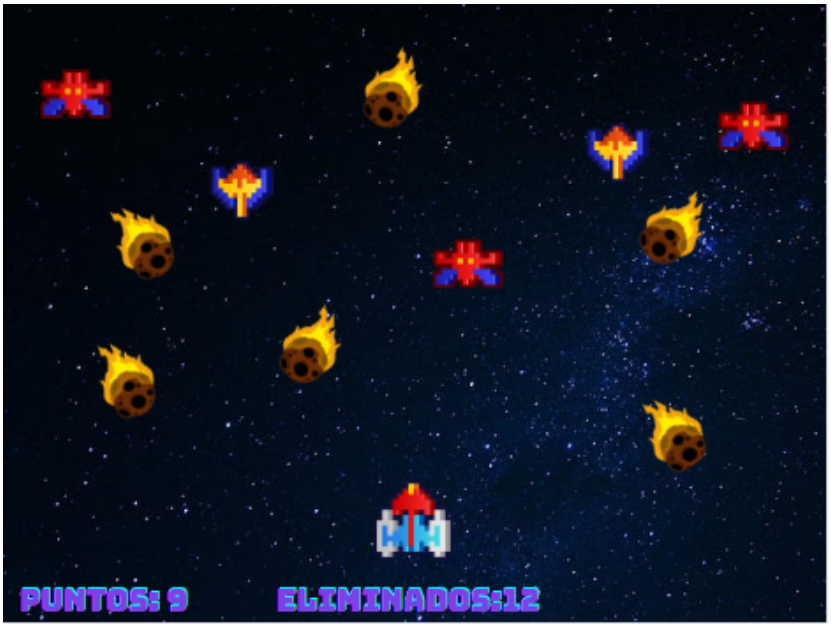

Figura 1: Ejemplo del juego.

El Juego: Lost Galaxian

Requerimientos obligatorios

1. Al comenzar el juego, en el centro inferior de la pantalla deberá verse Astro-MegaShip y algunos obstáculos representados por asteroides. La cantidad de obstáculos está a criterio de cada grupo, aunque deben ser no menos de 4 y no más de 6 en pantalla. (Ver Figura 1).

2. Los asteroides deben moverse en diagonal de arriba hacia abajo, algunos en diagonal hacia la derecha y otros en diagonal hacia la izquierda. Si la nave colisiona con alguno de estos obstáculos la misma es destruida.

3. La Astro-MegaShip sólo puede moverse de izquierda a derecha y de derecha a izquierda. Además, no puede salirse de los bordes de la pantalla. Para realizar, los movimientos de

la Astro-MegaShip se pueden usar las tecla de dirección (las flechas) o se pueden usar las teclas gamers, 'a', 'd'.

4. Los Destructores Estelares deben aparecer distribuidos aleatoriamente desde el sector superior de la pantalla, deben ir descendiendo y moviendose alternadamente hacia izquierda y derecha, dando la ilusión que la Astro-MegaShip avanza. Deben cuidar que los Destructores Estelares no se superpongan con los obstáculos ni entre ellos (un Destructor Estelar no puede volar sobre otro). Tampoco pueden salir por los límites izquierdo o derecho de la pantalla. La cantidad inicial de Destructores Estelares en pantalla deben ser entre 4 y 6.

5. Para defenderse de los Destructores Estelares, la Astro-MegaShip tiene proyectiles que se disparan al presionar la barra espaciadora. Sólo se puede disparar un proyectil a la vez. El proyectil debe desplazarse en línea recta hacia arriba desde la ubicación en la que estaba la Astro-MegaShip cuando se realizó el disparo. Cuando el proyectil hace contacto con un Destructor Estelar, lo destruye y no debe mostrarse más en pantalla. El proyectil que elimine a un Destructor Estelar, o colisiona con un asteroide, o llega a un extremo de la pantalla también debe desaparecer.

   Aclaramos que la destrucción o desaparición de un objeto de la pantalla, implica eliminar explícitamente el objeto, no vale invisibilizar el objeto o posicionarlo fuera de la pantalla.

6. Los Destructores Estelares realizan disparos de iones, con dirección recta hacia abajo, atravesando (sin destruir) los obstáculos a su paso. Si uno de estos disparos colisionan con la Astro-MegaShip, la destruye. Cuando un Destructor Estelar hace un disparo de iones este no se detiene aunque se elimine la nave que realizó el disparo.

7. Si un Destructor Estelar, un proyectil de iones o un asteroide alcanza la posición de la Astro-MegaShip y la toca, perdemos el juego.

8. Cuando la Astro-MegaShip haya eliminado algún Destructor Estelar, luego de una determinada cantidad de tiempo, a criterio de cada grupo, deben aparecer **nuevos** Destructores Estelares para tratar de que el juego no se quede sin enemigos.

9. Si en algún momento, el juego se queda sin enemigos en pantalla, el juego debe terminar y darse por ganado. Se deberá mostrar en pantalla el cartel correspondiente.

10. Durante todo el juego, deberá mostrarse en el inferior de la pantalla la cantidad de enemigos eliminados.

11. El código del proyecto **deberá tener un buen diseño** de modo que cada objeto tenga **bien delimitadas sus responsabilidades**.

0.1. Requerimientos opcionales

La implementación a entregar debe cumplir como mínimo con todos los requerimientos obligatorios planteados arriba, pero si el grupo lo desea, puede implementar nuevos elementos para enriquecer la aplicación. Sugerimos a continuación algunas ideas:

- Destructores Estelares especiales que sean kamikazes: cuando detectan la presencia de La Astro-MegaShip vuelan rápidamente hacia su ubicación para que ambas exploten.
- Varias vidas, o niveles de energía que vayan disminuyendo a medida que los Destructores Estelares hagan contacto con la Astro-MegaShip.
- Que aparezcan ítems en distintos lugares del juego que vayan cayendo que le den puntaje/vidas extra a la Astro-MegaShip.
- Niveles: La posibilidad de comenzar un nuevo nivel después de jugar determinada cantidad de tiempo o después de determinada cantidad de puntaje acumulado, e incrementar la dificultad y/o velocidad de cada nivel.
- Que aparezca un Destructor Estelar colosal como jefe final o de nivel.

Informe solicitado

Además del código, la entrega debe incluir un documento en el que se describa el trabajo realizado, que debe incluir, **obligatoriamente y como mínimo**, las siguientes secciones:

Carátula: Una carátula del documento con nombres, apellidos, legajos, y direcciones de email de cada integrante del equipo.

Introducción: Una breve introducción describiendo el trabajo práctico.

Descripción: Una explicación general de cada clase implementada, describiendo las variables de instancia y dando una breve descripción de cada método.

También deben incluirse los problemas encontrados, las decisiones que tomaron para poder resolverlos y las soluciones encontradas.

Implementación: Una sección de implementación donde se incluya el código fuente correctamente formateado y comentado, si corresponde.

Conclusiones: Algunas reflexiones acerca del desarrollo del trabajo realizado y de los resultados obtenidos. Pueden incluirse lecciones aprendidas durante el desarrollo del trabajo.

Condiciones de entrega

El trabajo práctico se debe hacer en grupos de **exactamente tres** personas.

El nombre del proyecto de Eclipse debe tener el siguiente formato: los apellidos en minúsculas de los tres integrantes, separados con guiones, seguidos de `-tp-p1`.
Por ejemplo, `amaya-benelli-castro-tp-p1`.

Cada grupo deberá hacer entrega tanto del informe como del código fuente del trabajo práctico. El informe, además, deberá ser entregado en versión impresa.

Consultar la modalidad de entrega de cada comisión.

- Si la modalidad de entrega es mediante email: especificar en el asunto del email, los apellidos en minúsculas de los tres integrantes. En el cuerpo del email colocar, nombres, apellidos, legajos, y direcciones de email de cada integrante del equipo. Adjuntar al email tanto el informe como el código fuente del trabajo práctico.
- Si la modalidad de entrega es mediante repositorio en Gitlab: asegurarse de que el repositorio sea privado y que el nombre del proyecto de Gitlab tenga los apellidos en minúsculas de los tres integrantes, separados con guiones, seguidos del string `-tp-p1`. Por ejemplo, `amaya-benelli-castro-tp-p1`. Consultar el username de cada docente y agregar a los docentes como `maintainer`'s. El informe del trabajo práctico puede incluirse directamente en el repositorio.

Fecha de entrega: Verificar en el moodle de la comisión correspondiente.

Apéndice: Implementación base

Junto con este enunciado, se les entrega el paquete `entorno.jar` que contiene la clase `Entorno`. Esta clase permite crear un objeto capaz de encargarse de la interfaz gráfica y de la interacción con el usuario. Así, el grupo sólo tendrá que encargarse de la implementación de la inteligencia del juego. Para ello, se deberá crear una clase llamada `Juego` que respete la siguiente signatura[1]:

```
public class Juego extends InterfaceJuego {
    private Entorno entorno;

    // Variables y metodos propios de cada grupo
    // ...

    public Juego() {
        // Inicializa el objeto entorno
        entorno = new Entorno(this, "Lost Galaxian - Grupo 3 - v1", 800, 600);

        // Inicializar lo que haga falta para el juego
        // ...

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

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

    public static void main(String[] args) {
       Juego juego = new Juego();
    }

}
```

El objeto `entorno` creado en el constructor del `Juego` recibe el juego en cuestión y mediante el método entorno.iniciar() se inicia el simulador. A partir de ahí, en cada instante de tiempo que pasa, el entorno ejecuta el método `tick()` del juego. éste es el método más importante de la clase `Juego` y aquí el juego debe actualizar su estado interno para simular el paso del tiempo. Como mínimo se deben realizar las siguientes tareas:

- Actualizar el estado interno de todos los objetos involucrados en la simulación.
- Dibujar los mismos en la pantalla (ver más abajo cómo hacer esto).
- Verificar si algún objeto aparece o desaparece del juego.

[1] **Importante:** las palabras clave "`extends InterfaceJuego`" en la definición de la clase son fundamentales para el buen funcionamiento del juego.

- Verificar si hay objetos interactuando entre sí (colisiones por ejemplo).
- Verificar si los usuarios están presionando alguna tecla y actuar en consecuencia (ver más abajo cómo hacer esto).

Para dibujar en pantalla y capturar las teclas presionadas, el objeto `entorno` dispone de los siguientes métodos, entre otros:

`void dibujarRectangulo(double x, double y, double ancho, double alto, double angulo, Color color)`
↪ Dibuja un rectángulo *centrado* en el punto (`x`,`y`) de la pantalla, rotado en el ángulo dado.

`void dibujarTriangulo(double x, double y, int altura, int base, double angulo, Color color)`
↪ Dibuja un triángulo *centrado* en el punto (`x`,`y`) de la pantalla, rotado en el ángulo dado.

`void dibujarCirculo(double x, double y, double diametro, Color color)`
↪ Dibuja un círculo *centrado* en el punto (`x`,`y`) de la pantalla, del tamaño dado.

`void dibujarImagen(Image imagen, double x, double y, double ang)`
↪ Dibuja la `imagen` *centrada* en el punto (`x`,`y`) de la pantalla rotada en el ángulo dado.

`boolean estaPresionada(char t)`
↪ Indica si la tecla `t` está presionada por el usuario en ese momento.

`boolean sePresiono(char t)`
↪ Indica si la tecla `t` fue presionada en este instante de tiempo (es decir, no estaba presionada en la última llamada a `tick()`, pero sí en ésta). Este método puede ser útil para identificar eventos particulares en un único momento, omitiendo `tick()` futuros en los cuales el usuario mantenga presionada la tecla en cuestión.

`void escribirTexto(String texto, double x, double y)`
↪ Escribe el `texto` en las coordenadas `x` e `y` de la pantalla.

`void cambiarFont(String font, int tamano, Color color)`
↪ Cambia la fuente para las próximas escrituras de texto según los parámetros recibidos.

Notar que los métodos `estaPresionada(char t)` y `sePresiono(char t)` reciben como parámetro un `char` que representa "la tecla" por la cual se quiere consultar, e.g., `sePresiono('A')` o `estaPresionada('+')`. Algunas teclas no pueden escribirse directamente como un `char` como por ejemplo las flechas de dirección del teclado. Para ellas, dentro de la clase entorno se encuentran las siguientes definiciones:

Valor	Tecla Representada
`TECLA_ARRIBA`	Flecha hacia arriba
`TECLA_ABAJO`	Flecha hacia abajo
`TECLA_DERECHA`	Flecha hacia la derecha
`TECLA_IZQUIERDA`	Flecha hacia la izquierda
`TECLA_ENTER`	Tecla “enter”
`TECLA_ESPACIO`	Barra espaciadora
`TECLA_CTRL`	Tecla “control”
`TECLA_ALT`	Tecla “alt”
`TECLA_SHIFT`	Tecla “shift”
`TECLA_INSERT`	Tecla “ins”
`TECLA_DELETE`	Tecla “del” (o “supr”)
`TECLA_INICIO`	Tecla “start” (o “inicio”)
`TECLA_FIN`	Tecla “end” (o “fin”)

De esta manera, para ver, por ejemplo, si el usuario está presionando la flecha hacia arriba se puede consultar, por ejemplo, si `estaPresionada(entorno.TECLA_ARRIBA)`.