Programación I - Segundo Semestre 2023
Trabajo Práctico: Plantas Invasoras

La ciudad de Felicidonia está siendo invadida por plantas mutantes que destruyen todo a su paso lanzando bolas de fuego. Para combatir esta plaga se contrató a uno de los integrantes de la Patrulla de Rescate Canina Layka, la cachorra rescatista.

El objetivo de este trabajo práctico es desarrollar un video juego en el cual Layka pueda hacerle frente a esta invasión de plantas mutantes sin perder la vida en el intento.

El Juego: Plantas invasoras

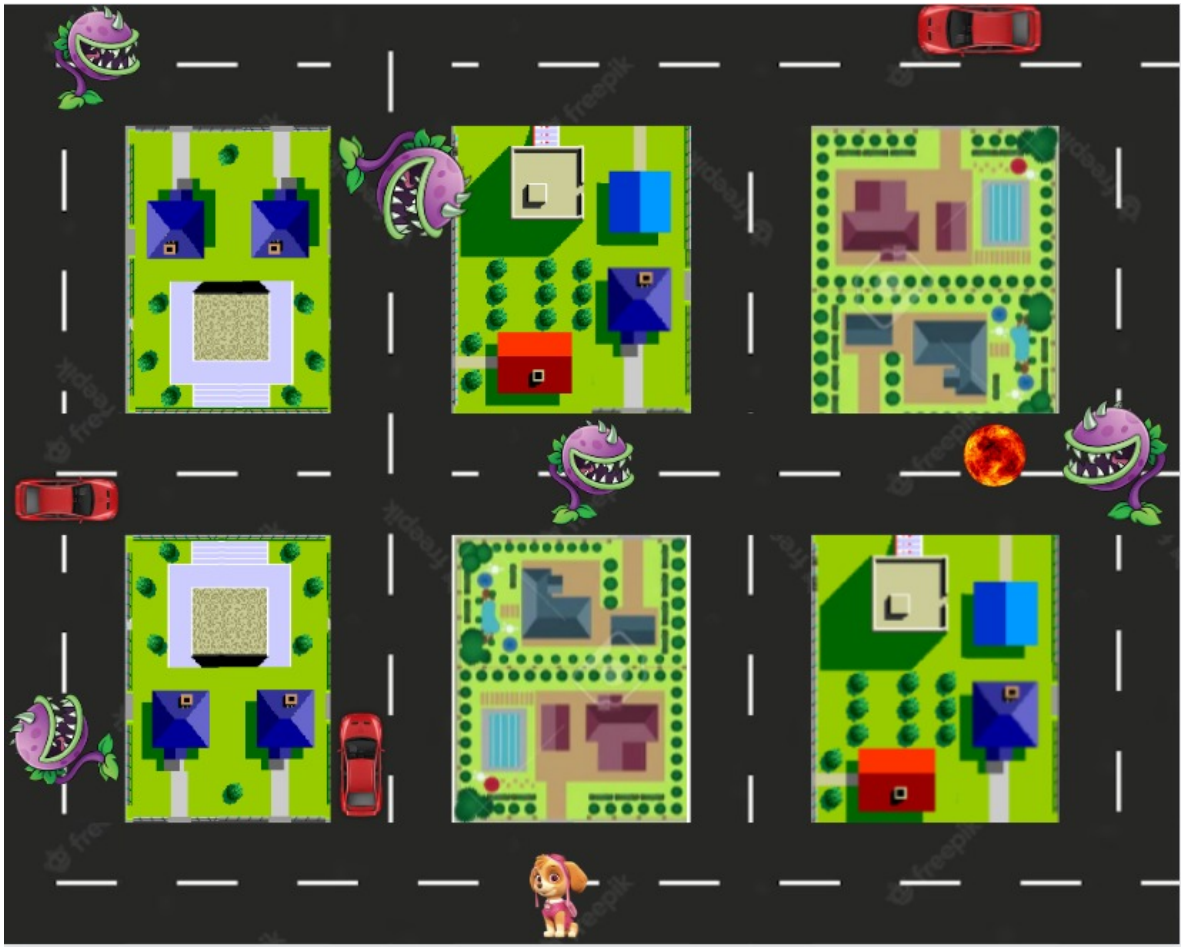

Figura 1: Ejemplo del juego.

El Juego: Plantas Invasoras

Requerimientos obligatorios

1. Al iniciar el juego, Layka debe aparecer aproximadamente en la parte inferior de la pantalla (ver Figura 1).

2. Layka sólo puede moverse por las calles de la ciudad, es decir, no puede atravesar por encima de las casas, sólo puede moverse por sobre las calles. Tampoco puede salirse de los límites de la pantalla.

3. Si se presionan las flechas direccionales (izquierda, abajo, arriba, derecha) Layka debe moverse en la dirección correspondiente. Sólo debe moverse en una dirección a la vez, y sólo mientras la tecla correspondiente a esa dirección esté presionada. Si no se presiona ninguna tecla direccional Layka debe quedarse quieta. (En lugar de las flechas, se pueden usar las teclas gamers, 'w', 'a', 's', 'd').

4. En pantalla deberá verse una ciudad con una grilla de calles, cada calle tiene una mano de ida y otra de vuelta. La cantidad de calles es a criterio de cada grupo, aunque como mínimo deben ser tres horizontales (se sugiere entre 3 y 5 horizontales) y mínimo tres calles verticales (se sugiere entre 3 y 5 verticales). Además, deben verse tres o 4 casas por manzana (ver Figura 1).

5. Por la ciudad transitan autos a los que Layka debe esquivar cambiando de carril o moviéndose hacia una calle donde no haya autos para no ser atropellada. Si Layka es atropellada perdemos el juego.

6. Cada planta mutante debe desplazarse únicamente por una misma calle y siempre en la misma dirección. Por ejemplo, si una planta dada se mueve de izquierda a derecha debe hacerlo siempre de izquierda a derecha, en cambio, si se mueve de arriba hacia abajo, debe hacerlo siempre de arriba hacia abajo. La cantidad de plantas mutantes queda a criterio de cada grupo, aunque como mínimo deben ser cuatro y se sugiere que sean entre cuatro y seis plantas presentes en todo momento en pantalla. Las plantas mutantes no se pueden superponer entre ellas (no pueden ir caminando una planta encima de otra), aunque sí pueden cruzarse por la misma calle. Cuando una planta mutante llega a un borde de la pantalla puede volver a aparecer en el borde contrario o rebotar. Si una planta mutante toca a Layka perdemos el juego.

7. Cuando se presione la barra espaciadora Layka puede lanzar un Rayo destructor. El Rayo destructor es un proyectil de energía que se desplaza en la misma dirección en la que Layka iba caminando. Cuando el Rayo hace contacto con una planta mutante, la mata y esta misma no debe mostrarse más en pantalla. El Rayo que destruye a una planta también debe desaparecer de la pantalla. **Aclaramos que tanto la planta mutante como el Rayo deben ser eliminados, no vale únicamente «ocultar» las imágenes del juego.**

8. El Rayo también debe ser eliminado cuando llegue al final de la pantalla o cuando haga contacto con un auto (sin destruir el auto).

9. Cada planta mutante puede lanzar bolas de fuego que arrasa con todo a su paso, destruyendo los vehículos a su paso. Laika sólo podrá esquivarlo moviéndose a otra calle o contraatacando con su rayo destructor, si estos proyectiles hacen contacto se genera una colisión y ambos desaparecen. Si una bola de fuego hace contacto con Layka perdemos el juego.

10. Cuando Layka haya eliminado alguna planta mutante, luego de determinada cantidad de tiempo, a criterio de cada grupo, deben aparecer nuevas plantas mutantes. Siempre hay que respetar la cantidad mínima de cuatro plantas en el juego.

11. Durante todo el juego deberán mostrarse los siguientes datos en pantalla: Deberá mostrarse en la esquina superior o inferior derecha de la pantalla el puntaje acumulando. Para calcular el puntaje, por cada planta eliminada se debe incrementar el puntaje en 5 puntos. Además, deberá mostrarse en la esquina superior izquierda de la pantalla, la cantidad de plantas eliminadas.

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

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:

- Tipos de enemigos: Que aparezcan nuevas plantas mutantes (más rápidas, más difíciles de eliminar, etc).
- Dos jugadores: Permitir el juego para dos jugadores de manera que compitan entre sí
- Niveles: La posibilidad de comenzar un nuevo nivel después de jugar determinada cantidad de tiempo o de puntaje acumulados, e incrementar la dificultad y/o velocidad de cada nivel.
- Ganar el juego: Al pasar cierto tiempo, conseguir una cierta cantidad de puntaje acumulado o llegar a una casa refugio, que el juego se dé por ganado.
- Que aparezca una Planta mutante colosal como jefe final o de nivel
- Vidas: poner vidas y cada vez que muera Layka se le reste una vida
- Item: que permitan a Layka tener más resistencia ante los choques de los autos

Informe solicitado

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

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

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 entregar tanto el informe como el código fuente del trabajo práctico.

Consultar la modalidad de entrega de cada comisión (email, google drive, git, etc).

- 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, "Plantas Invasoras - 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)`.