Programación I - Primer Semestre 2022
Trabajo Práctico: Attack on Titan, Final Season

En la Isla Paradis existen unos malvados gigantes de forma humanoide, llamados kyojines, que invaden las ciudades y aplastan todo a su paso. Para combatir a los kyojines, las Fuerzas Armadas de la humanidad, crearon La Legión de Reconocimiento, encargada de exterminarlos.

Mikasa Ackerman, nuestra heroína y la soldada más letal de La Legión de Reconocimiento, tiene la tarea de exterminar a todos los kyojines que invadieron la ciudad.

El objetivo de este trabajo práctico es desarrollar un video juego en el cual Mikasa elimine a la mayor cantidad posible de kyojines, sin perder la vida en el intento.

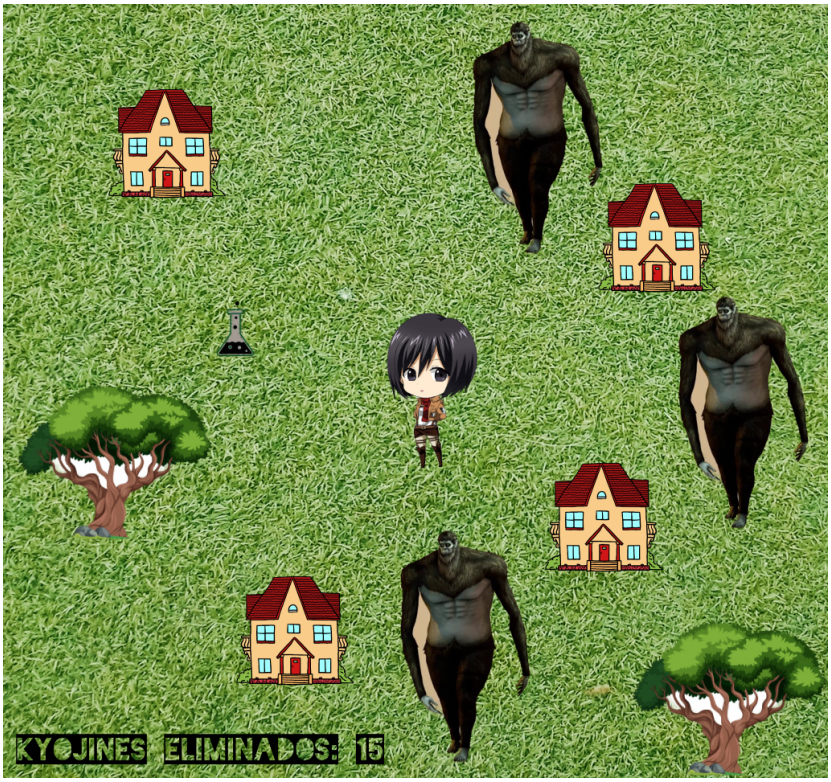

Figura 1: Ejemplo del juego.

El Juego: Attack on Titan, Final Season

Requerimientos obligatorios

1. Al comenzar el juego, en pantalla deberá verse una ciudad con algunos obstáculos. La cantidad de obstáculos es a criterio de cada grupo, aunque deben ser no menos de 4 obstáculos, y no más de 6 obstáculos. Los distintos personajes del juego no pueden

atravesar los obstáculos. Los obstáculos pueden ser cualquier elemento que no se mueva, como por ejemplo, edificios, árboles, ó algún otro elemento habitual de una ciudad.

En el centro de la ciudad, debe ubicarse Mikasa (ver Figura 1).

2. Mikasa puede moverse libremente por toda la ciudad, siempre que no haya un obstáculo delante. Durante todo momento, Mikasa tiene una orientación determinada. Si se presionan las flechas izquierda y derecha, la orientación de Mikasa debe rotar en el sentido correspondiente. Si se presiona la flecha arriba, Mikasa debe avanzar en la orientación establecida previamente. Cuando hay un obstáculo en su camino, Mikasa no puede avanzar, debe rodear el obstáculo. Además, Mikasa no puede salirse de los bordes de la pantalla. (En lugar de las flechas, se pueden usar las teclas gamers, 'w', 'a', 's', 'd').

3. Los kyojines deben aparecer distribuidos aleatoriamente en la pantalla, cuidando de que no se superpongan con los obstáculos. La cantidad inicial de kyojines en pantalla deben ser entre 4 y 6. Además, los kyojines no se pueden superponer entre ellos (no puede ir caminando un kyojin encima de otro).

   Los kyojines no pueden salirse de los límites de la pantalla.

4. Los kyojines pueden caminar libremente por toda la ciudad, siempre que no haya un obstáculo delante, en ese caso, deben rodear al obstáculo.

   Una característica de los kyojines es que pueden detectar la presencia humana y perseguirlos. Por este motivo, los kyojines deben caminar hacia la posición de Mikasa.

5. Si un kyojin alcanza la posición de Mikasa y la toca, perdemos el juego.

6. En el país Marley se desarrolló el suero *kyojin no kessei*, capaz de transformar temporalmente a una persona en un kyojin. Durante el juego, deben aparecer dosis del suero a intervalos aleatorios y en posiciones aleatorias. Cuando Mikasa camina sobre el suero, lo toma y se transforma temporalmente en una kyojina, lo que le brinda poderes. Si un kyojin toca a Mikasa, cuando se encuentra en estado de kyojina, el kyojin muere y Mikasa vuelve a su estado normal.

7. Cuando se presione la barra espaciadora, Mikasa debe lanzar un proyectil anti-kyojin. El proyectil lanzado debe desplazarse en la misma orientación en la que estaba Mikasa.

   Cuando el proyectil hace contacto con un kyojin, lo mata y el kyojin no debe mostrarse más en pantalla. El proyectil que elimine a un kyojin también debe desaparecer de la pantalla. Si el proyectil llega a un extremo de la pantalla, también debe desaparecer. **Aclaramos que tanto el kyojin como el proyectil deben ser eliminados, no vale únicamente "ocultar" las imágenes del juego.**

8. Cuando Mikasa haya eliminado algún kyojin, luego de una determinada cantidad de tiempo, a criterio de cada grupo, deben aparecer nuevos kyojines 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, y debe mostrarse un 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**.

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:

- Varias vidas, ó niveles de energía que vayan disminuyendo a medida que los kyojines toquen a Mikasa.
- Que aparezcan ítems en distintos lugares del juego que le den puntaje/vidas extra a Mikasa.
- Niveles: La posibilidad de comenzar un nuevo nivel después de jugar determinada cantidad de tiempo ó después de determinada cantidad de puntaje acumulado, e incrementar la dificultad y/ó velocidad de cada nivel.
- Que aparezca un kyojin colosal jefe de nivel.

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:

Encabezado Un encabezado ó 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 entregar tanto el informe **en pdf** como el código fuente del trabajo práctico **exportando el proyecto completo como archive file**. El código fuente debe incluir todos los archivos necesarios para que el proyecto se pueda ejecutar en cualquier equipo con Eclipse.

Fecha de entrega: Hasta el martes 24 de mayo de 2022, 18:00 hs.

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 métodos propios de cada grupo
    // ...

    public Juego() {
        // Inicializa el objeto entorno
        entorno = new Entorno(this, "Attack on Titan - 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)`.