Programación I - Primer Semestre 2021
Trabajo Práctico: Sakura Ikebana Delivery

Sakura Haruno junto a su amiga de la infancia Ino Yamanaka, tienen un negocio de arreglos florales típicos japoneses *(ikebana)*. Ino se encarga de preparar los arreglos florales, y Sakura se encarga del delivery de los mismos.

La florería Yamanaka-Haruno está ubicada en una aldea ninja, que suele ser atacada por ninjas enemigos. Esto complica el trabajo de Sakura que debe entregar los pedidos evitando y combatiendo a los ninjas enemigos.

El objetivo de este trabajo práctico es desarrollar un video juego en el cual Sakura entregue los pedidos sin perder la vida a manos de los ninjas enemigos.

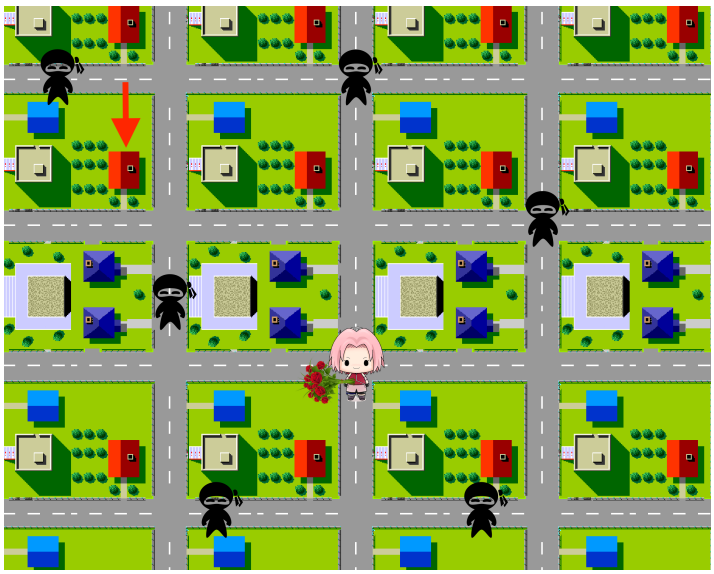

Figura 1: Ejemplo del juego.

El Juego: Sakura Ikebana Delivery

Requerimientos obligatorios

1. Al iniciar el juego, Sakura debe aparecer aproximadamente en el centro de la pantalla (ver Figura 1).

2. Sakura solo puede moverse por las calles de la ciudad, es decir, no puede cruzar por encima de las casas, solo 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) Sakura deberá moverse en la dirección correspondiente. Solo debe moverse en una dirección a la vez, y

solo mientras la tecla correspondiente a esa dirección esté presionada. Si no se presiona ninguna tecla direccional, Sakura deberá quedarse quieta. (En lugar de las flechas, se pueden usar las teclas gamers, ‘w‘, ‘a‘, ‘s‘, ‘d‘).

4. En pantalla deberá verse una ciudad/aldea, con una grilla de calles. 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 dos o tres casas por manzana (ver Figura 1).

5. Durante el juego, Sakura debe entregar un pedido (un ramo de flores o ikebana) en una casa determinada aleatoriamente. Una vez entregado, se genera el siguiente pedido nuevamente para otra casa determinada aleatoriamente. La casa donde Sakura debe entregar el pedido deberá estar marcada con una flecha.

   Cuando Sakura pasa caminando por enfrente de la casa marcada, se considera al pedido entregado y se cuenta como una entrega hecha. Al realizar la entrega la casa deberá dejar de estar marcada y deberá aparecer alteatoriamente otra casa marcada para realizar el siguiente pedido.

6. Cada ninja enemigo debe desplazarse únicamente por una única calle, y siempre en la misma dirección. Por ejemplo, si un ninja dado se mueve de izquierda a derecha, deberá hacerlo siempre de izquierda a derecha, en cambio, si se mueve de arriba hacia abajo, deberá hacerlo siempre de arriba hacia abajo.

   La cantidad de ninjas enemigos queda a criterio de cada grupo, aunque como mínimo deben ser cuatro y se sugiere que sean entre cuatro y seis ninjas presentes en todo momento en pantalla. Los ninjas no se pueden superponer entre ellos (no puede ir caminando un ninja encima de otro), aunque sí pueden cruzarse por la misma calle.

   Cuando un ninja enemigo llega a un borde de la pantalla debe reaparecer en el borde contrario.

7. Si un ninja toca a Sakura perdemos el juego.

8. El gran maestro Gama-Sennin le enseñó a Sakura su técnica secreta, el *Rasengan*. Cuando Sakura se ve acorralada por ninjas enemigos, puede usar el Rasengan para eliminar a los ninjas que se cruzan por su camino.

   Cuando se presione la barra espaciadora, Sakura debe lanzar un Rasengan. El Rasengan es un proyectil de energía que se desplaza en la misma dirección en la que Sakura iba caminando.

   Cuando el Rasengan hace contacto con un ninja, lo mata y el ninja no deberá mostrarse más en pantalla. El Rasengan que destruya a un ninja también deberá desaparecer de la pantalla. **Aclaramos que tanto el ninja como el Rasengan deben ser eliminados, no vale únicamente “ocultar” las imágenes del juego.**

9. Cuando Sakura haya eliminado algún ninja, luego de determinada cantidad de tiempo, a criterio de cada grupo, deberán aparecer nuevos ninjas. Siempre hay que respetar la cantidad mínima de cuatro ninjas en el juego.

10. Durante todo el juego deberán mostrarse los siguientes datos en pantalla:

- En la esquina superior derecha de la pantalla, el puntaje acumulado. Para calcular el puntaje, por cada pedido entregado se debe incrementar el mismo en 5 puntos.
- En la esquina superior izquierda de la pantalla, la cantidad de ninjas 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 deberá 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:

- Obstáculos: Que los ninjas puedan lanzar shurikens o dejar clavos en el piso, de manera que si Sakura toca estos obstáculos se pierda el juego.
- Items especiales: Agregar items especiales como monedas o regalos que aumenten el puntaje del juego.
- Florería: Sakura debe buscar el ikebana en la florería antes de llevarlo a la siguiente casa.
- Tipos de enemigos: Que aparezcan nuevos tipos de ninjas (más rápidos, más dificiles de eliminar, etc).
- Dos jugadores: Permitir el juego para dos jugadores de manera que compitan entre sí para entregar los pedidos.
- 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, que el juego se dé por ganado.

Informe solicitado

Además del código, la entrega deberá 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 como el código fuente del trabajo práctico. Hay dos opciones de entrega:

- Subir al Moodle el informe y el código del proyecto (exportado vía Eclipse como **archive file**).
- Subir al Moodle el link al 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 podrá incluirse directamente en el repositorio.

Fecha de entrega: Jueves 27 de mayo hasta las 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, "Sakura Ikebana Delivery - 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)`.