Programación I - Primer Semestre 2025
Trabajo Práctico: El camino de Gondolf

En lo profundo de las catacumbas, el anciano mago Gondolf perdió su camino intentando encontrar una pieza de joyería con forma circular y propiedades mágicas. Viendo esta oportunidad, su enemigo, el mago Suramun comienza a enviar un sin fin de sus lacayos murciélagos para intentar derrotarlo.

El objetivo de este trabajo práctico es desarrollar un video juego en el cual contolamos al mago Gondolf. En el juego se busca sobrevivir al ataque de los murciélagos y derrotarlos usando distintos hechizos.

Figure 1: Ejemplo del juego.

El Juego: El camino de Gondolf

Requerimientos obligatorios

1. Pantalla y menú de juego: Se deberá ver en todo momento una sección de la pantalla en la cual transcurre el juego y otra sección donde se encuentra un menú con botones e información. Al iniciar el juego, el mago Gondolf debe verse en el centro de la pantalla de juego (ver Figura 1).

2. Mago Gondolf: El personaje deberá poder moverse hacia arriba, abajo, izquierda, derecha, con las teclas 'w', 'a', 's' y 'd' respectivamente, o con las flechas del teclado. Si no se presiona ninguna tecla el personaje deberá quedarse quieto. El mago no deberá salirse de la pantalla en ningún momento, esto también incluye a la sección donde se coloque el menú, el personaje no deberá caminar sobre, o ocultarse debajo del menú.

3. En el escenario de juego hay rocas que interfieran con el paso de Gondolf. Estos obstaculos no podrán ser eliminados. Las rocas no detienen a los enemigos, ya que los murciélagos pueden volar sobre ellas. La cantidad de obstaculos queda a criterio del grupo, se recomienda agregar 5 rocas.

4. Los murciélagos enemigos de Gondolf deben aparecer desde lugares aleatorios fuera de la pantalla (desde los cuatro bordes) (ver Figura 2) y deben perseguirlo en todo momento. La cantidad de enemigos en pantalla queda a criterio de cada grupo pero debe cumplirse dos conceptos. Por un lado tiene que haber una cierta cantidad total de enemigos a derrotar (por ejemplo 50), y por otro lado tiene que existir una cantida máxima de enemigos vivos en pantalla (por ejemplo 10), a medida que estos 10 vayan muriendo, deben crearse nuevos enemigos para suplantarlos.

Figure 2: En verde las zonas de aparición de los murciélagos.

5. Si un murgiélago hace contacto con el jugador, deberá quitarle un poco de sus puntos de vida y luego desaparecer. Cuando el mago se queda sin energía de vida se da por perdido el juego.

6. Como todo buen mago, Gondolf puede realizar hechizos con su energía mágica para atacar a los murciélagos. Debe haber por lo menos 2 tipos de hechizos disponibles en el menú. Los hechizos deberán tener su propio nombre, área de efecto y costo de energía mágica. Uno de los hechizos deberá tener costo 0 para que siempre exista una opción de ataque. Para usar un hechizo primero se debe seleccionar su botón correspondiente posicionando el mouse sobre él y presionando el botón izquierdo del mouse (click). Al presionar un botón éste debe quedar seleccionado hasta que el hechizo que le corresponde sea lanzado o hasta que se seleccione otro hechizo (Ver Figura 3).

Figure 3: Si se selecciona un botón se des-selecciona el otro.

7. Para lanzar el hechizo seleccionado debe posicionarse el mouse sobre la pantalla de juego (no sobre el menú) y presionar el botón izquierdo el mouse. Una vez que un hechizo sea lanzado, su botón asociado deberá des-seleccionarse. Una vez lanzado un hechizo, todos los enemigos que estén en contacto con su área de efecto deberán ser eliminados (si el grupo lo prefiere, puede determinar un nivel de daño que cada hechizo aplicará a

los enemigos alcanzados en el área). Así mismo si el hechizo tiene un costo de energía magica para el jugador, éste deberá consumirse.

8. Los enemigos que sean eliminados (por hacer sido atacados con hechizos o bien por hacer contacto con el mago) deberán volverse null, no vale únicamente "ocultar" las imágenes del juego.

9. Si Gondolf elimina a todos los murciélagos el juego termina y se da por ganado.

10. Durante todo el juego deberá mostrarse en un sector del menú: la cantidad de enemigos eliminados, la energía mágica que le queda al personaje y sus puntos de vida. Además puede mostrarse otra información que el grupo considere necesaria.

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:

- Oleadas de enemigos: El objetivo del juego será sobrevivir a sucesivas oleadas de enemigos cada vez más dificiles, por ejemplo aumentando la cantidad de enemigos en pantalla, su velocidad o la cantidad de daño. Es decir, el juego deberá poder ganarse (al superar una cantidad suficiente de oleadas) o perderse (el personaje se queda sin vida).

- Recompensas: cuando se llegue a cierta cantidad de enemigos eliminados (o al superar una oleada), presentar en una pantalla de espera recompensas para elegir: incremento de puntos de vida o de magia, incremento de velocidad de movimiento, etc. Mientras se muestre esa pantalla, el juego quedará en pausa hasta que se decida continuar (presionando una tecla o botón).

- Más tipos de hechizos: Diseñar otros tipos de hechizos que tengan sus propios costos, áreas y efectos. Por ejemplo hechizo de barrera que no permita a los enemigos acercarse durante cierto tiempo, un hechizo que ralentice a los enemigos, etc.

- Items: Al morir los enemigos deberán tener una chance de soltar una poción beneficiosa para el mago (ver Figura 4). Cuando el jugador haga contacto con una poción en el suelo, la misma deberá restaurarle una cierta cantidad de recursos que le corresponda (vida o magia) y desaparecer del juego (volverse null). Debe ser posible tener varias pociones en el suelo al mismo tiempo.

Figure 4: Ejemplo de poción de vida.

- Tipos de enemigos: Podrán diseñarse otros tipos de enemigos, por ejemplo: enemigos que lanzan proyectiles a distancia, enemigos que ralentizen al jugador, enemigos que curen a sus compañeros, etc.
- Jefe: Después de un cierto número de oleadas podrá diseñarse un jefe el cual aparecerá solo contra el jugador (sin otros murciélagos), este deberá tener ataques a distancia y ataques en área.

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 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 **cuatro** personas.

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

Cada grupo deberá entregar tanto el informe como el código fuente del trabajo práctico mediante Git.

Cada grupo deberá crear un repositorio Git: asegurarse de que el repositorio sea privado y que el nombre del proyecto de Git tenga los apellidos en minúsculas de los tres integrantes, separados con guiones, seguidos del string `-tp-p1`.
Por ejemplo, `amaya-benelli-castro-paz-tp-p1`. Consultar el username de cada docente y agregar a los docentes como `maintainer`'s.

Fecha de entrega: 3 de junio de 2025 hasta las 17:59 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 metodos propios de cada grupo
    // ...

    public Juego() {
        // Inicializa el objeto entorno
        entorno = new Entorno(this, "El camino de Gondolf - 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 sá 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)`.