Programación I - Segundo Semestre 2024
Trabajo Práctico: Al rescate de los gnomos

En el mundo mágico, unos gnomos habitaban tranquilamente en islas flotantes en el cielo. Su vida pacífica se ve interrumpida cuando un día son atacados por el clan de las tortugas con caparazones de acero.

Las tortugas caen desde el cielo y al llegar a una isla se quedan en ella moviendose de un lado a otro y eliminando a los gnomos con los que se encuentren, pues son muy venenosas. Afortunadamente, hay un campo de fuerza que protege la casa de los gnomos, por lo que las tortugas no pueden caer en esa isla.

Los gnomos, que no son muy inteligentes, salen de su casa situada en la isla más alta y huyen de la invasión pero van cayendo de una isla a otra y pueden caer al vacío si llegan a las islas inferiores.

Ante esta amenaza, los gnomos deciden contratar los servicios de Pep, un caballero medieval conocido por su gran defensa contra los ataques enemigos y el rescate de prisioneros.

El objetivo de este trabajo práctico es crear un simulador en el cual Pep vaya rescatando a los gnomos antes de que caigan al vacío o sean destruidos por las tortugas de caparazones con púas.

Figura 1: Ejemplo del juego.

El Juego: Al rescate de los gnomos

Requerimientos obligatorios

1. Al comenzar el juego, deben mostrarse entre 4 y 6 filas de islas flotantes y la isla con la casa de los gnomos en el sector central superior de la pantalla. Las dos filas superiores de islas flotantes deben ocupar solo un sector de la pantalla (aproximadamente la mitad)

y deben estar intercaladas con sectores vacíos. Las islas en las filas inferiores también deben intercalarse con espacios vacíos ocupando todo el ancho de la pantalla. Pep debe estar ubicado sobre una de las islas en la fila inferior. (Ver Figura 1).

2. Mientras esté sobre las islas flotantes, Pep puede moverse usando las teclas izquierda y derecha, o las teclas 'a' y 'd', y puede saltar con la tecla flecha arriba o 'w'. Si no se presiona ninguna tecla, Pep debe permanecer parado sobre la isla. Si Pep no está sobre una isla flotante o no está saltando, debe caer hasta hacer colisión con una isla o caer al precipicio.

3. Pep tiene el poder del sol en sus manos, lo que le permite arrojar pequeñas bolas de fuego que van a ras del piso para acabar con sus enemigos. Para lanzar un disparo se debe presionar la tecla 'c' o el botón izquierdo del mouse, la bola de fuego se moverá hacia la dirección donde se movió Pep la última vez.

4. Los gnomos salen de su casa ubicada en la isla más alta y se mueven hacia la izquierda o derecha. Cuando caen a una isla de un nivel más bajo, cambian su dirección de manera aleatoria. Debe haber entre 2 y 4 gnomos en pantalla, reponiéndose cada cierto tiempo ya que serán rescatados por Pep, aniquilados por las tortugas, o caerán al precipicio.

5. Un gnomo es rescatado cuando colisiona con Pep. Pero sólo podrá rescatarlo en las islas inferiores (en las primeras dos filas desde abajo hacia arriba). Si un gnomo cae al precipicio o es colisionado por una tortuga, se considera un gnomo perdido. En ambos casos, se debe eliminar la instancia del gnomo.

6. Las tortugas aparecen cayendo desde el borde superior de la pantalla en una posición aleatoria pero no sobre la isla de la casa de los gnomos. Cuando una tortuga llega a una isla flotante empieza a moverse de izquierda a derecha (o al revés) hasta el borde de la isla, luego debe cambiar de dirección para permanecer en la misma. Si una tortuga hace coalisión con Pep o con los gnomos los aniquila ya que son muy venenosas.

7. El juego se gana cuando Pep haya rescatado una cantidad determinada de gnomos a elección del grupo. El juego se pierde cuando se pierde una cierta cantidad de gnomos, o bien cuando Pep cae al precipicio o es aniquilado por las tortugas.

   Aclaramos que la destrucción o desaparición de un objeto de la pantalla, implica eliminar explícitamente el objeto (debe ser nulo), no vale únicamente ocultar la imagen o posicionarlo fuera de la pantalla.

8. Durante todo el juego deberá mostrarse en algún lugar de la pantalla: El tiempo transcurrido del juego, la cantidad de gnomos rescatados y perdidos, y la cantidad de enemigos eliminados.

9. 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:

- Disparos de las tortugas: Las tortugas arrojan bombas que se mueven a ras del suelo hasta que hagan colisión con Pep, los gnomos o salgan de la pantalla. Esas bomban solo pueden ser destruidas por los disparos de fuego.
- Movimiento de las islas: Cada una de las filas de islas se desplaza siempre en la misma dirección (izquierda o derecha). Las islas superiores cambiarán de dirección al llegar al borde y las islas inferiores irán saliendo de la pantalla e irán apareciendo otras desde los bordes opuestos de la pantalla.
- Escudo: Pep tiene un escudo que puede resistir hasta 3 bombas. Puede activar el escudo presionando la tecla flecha abajo o 's'. Cada vez que se usa el escudo, este pierde fuerza y Pep puede ser derrotado después de recibir tres bombas.
- Gnomos con Ítems Especiales: Al hacer colisión con un gnomo, éste le puede dar (o quitar) a Pep un escudo extra o inmortalidad por unos segundos, siempre y cuando no caiga al precipicio
- Nave de Combate Aérea: En la parte inferior de la pantalla debe estar la nave de rescate de Pep para evitar que los gnomos caigan al precipicio. Manteniendo presionado el botón derecho del mouse, se puede mover la nave hacia los lados. Pep también puede usarla para no caer al precipicio.
- Niveles: La posibilidad de comenzar un nuevo nivel después de jugar determinada cantidad de tiempo o después de determinada cantidad de gnomos rescatados, e incrementar la dificultad y/o velocidad de cada 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, "Al rescate de los gnomos - 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.

`boolean seLevanto(char t)`
↪ Indica si la tecla `t` fue levantada en este instante de tiempo (es decir, estaba presionada en la última llamada a `tick()`, pero no en ésta).

`boolean sePresionoBoton(int bot), boolean estaPresionado(int bot), boolean seLevantoBoton(int bot)`
↪ Similar a los métodos anteriores pero respecto a los botones del mouse. Debe usarse como parámetro `bot` los valores `entorno.BOTON_IZQUIERDO`, `entorno.BOTON_CENTRAL` o `entorno.BOTON_DERECHO`.

`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)`, `sePresiono(char t)` y `seLevanto(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)`.

Para más especificaciones del proyecto base ver el documento Especificaciones del Entorno en el moodle.