Programación I - Primer semestre de 2019
Trabajo Práctico: La torre de los magos

Amenaza en el mundo mágico

Recibimos una lechuza con noticias del mundo mágico! Aparentemente hay una invasión de monstruos venenosos y todos los magos están luchando por defenderse. Según las noticias, por el momento están teletransportando a los monstruos a una torre mágica que tiene la particularidad de que si caes en el agujero que está en el sector más bajo de la torre, vuelves a aparecer en la parte superior. Así los monstruos no pueden escapar (por ahora) y les da tiempo para intentar terminar con esa amenaza.

Los magos descubrieron un hechizo congelante que les permite derrotar a un monstruo primero congelandolo y luego golpeandolo con la varita (u otra cosa que tengan a mano). El problema es que los monstruos son muy venenosos; tanto que si alguien toca un monstruo no congelado muere inmediatamente.

Los magos quieren entrar en la torre mágica para acabar con los monstruos aunque saben que están arriesgando la vida. Por lo cual han solicitado a la materia de Programación I desarrollar un simulador para poder entrenar a los valientes magos antes del combate, siguiendo una serie de condiciones que se describen a continuación.

El simulador: Combate en la torre mágica

La aplicación a desarrollar debe simular el combate en la torre mágica. En la pantalla deben verse vigas horizontales a diferentes alturas que simulan los pisos de la torre. La longitud de las vigas no debe ocupar todo del eje horizontal de la pantalla y debe ser posible caer desde una viga superior a cada viga del escenario. La viga superior debe estar centrada, en el siguiente nivel hacia abajo debe haber dos vigas, una deberá tocar la pared izquierda y la otra deberá tocar la pared derecha, las siguientes dos vigas estarán centradas y en el nivel inferior habrá nuevamente dos vigas pegadas a las paredes y dejando un agujero en el piso (ver Figura 1).

Inicialmente en la viga superior se debe encontrar el mago y en las vigas inferiores se ubicarán los monstruos.

Durante la simulación, los monstruos se moverán avanzando por las vigas y cayendo en linea recta al llegar al final de una viga para seguir moviendose por la siguiente. Al chocar con una pared cambiarán de dirección. Al caer en el agujero del piso deberán aparecer en la parte superior de la pantalla.

El mago también se moverá horizontalmente, caerá en linea recta cuando llegue al final de una viga, cambiará de dirección al llegar a una pared y aparecerá en la parte superior de la pantalla luego de caer por el pozo del piso. Además cada vez que el mago cae de un viga superior a otra inferior lanza un hechizo congelante. Este hechizo avanzará en la dirección

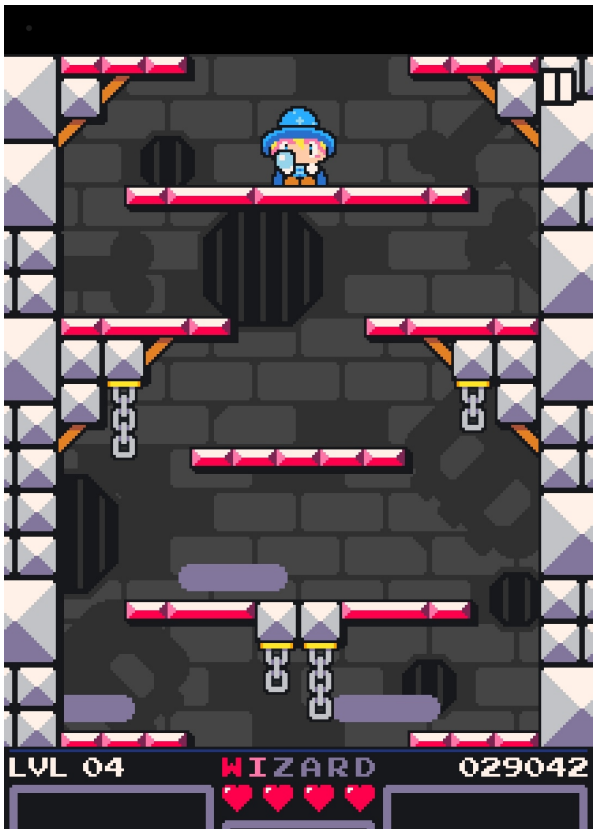

Figura 1: Arte conceptual ejemplo del simulador.

en la que se esté moviendo el mago al momento de terminar de caer. Si llega a golpear a un monstruo este quedará congelado, si no encuentra enemigos el hechizo desaparecerá al llegar a la pared.

Cuando un monstruo recibe un hechizo congelante, queda congelado por un tiempo (y por lo tanto, deja de moverse). Si en ese tiempo el mago lo toca, el monstruo congelado empieza a rodar por los pisos hasta llegar al agujero del piso donde desaparece para nunca más regresar. Si mientras va rodando se encuentra con otro monstruo, también lo arrastra hacia el fin. En caso de que el mago no llegue a golpear a un monstruo congelado en un tiempo determinado, éste se recupera y vuelve a moverse.

El objetivo del usuario será derrotar a todos los monstruos usando hechizos congelantes y luego haciéndolos rodar. Cuando nuestro mago logre derrorar a todos, se dará por finalizada la simulación. Si el mago llega a tocar a un monstruo no congelado, la simulación también será finalizada.

Requerimientos obligatorios

Son requerimientos obligatorios los siguientes:

1. Inicialmente se debe mostrar el mago se ubica en la viga superior y 4 monstros en las vigas inferiores centradas, uno en cada extremo de las vigas. Las vigas deben estar distribuidas tal como se explicó en la sección anterior.

2. Durante la simulación, los monstruos se moverán avanzando por las vigas y cayendo en linea recta al llegar al final. Al chocar con una pared deberán cambiar de dirección. Al caer en el agujero del piso deberán aparecer en la parte superior de la pantalla.

3. En todo momento el usuario deberá poder elegir la derección en la que se dezplazará el mago presionando las flechas izquierda y derecha. El Mago también se moverá horizontalmente sobre las vigas y caerá en linea recta al final, cambiará de dirección automáticamente al llegar a una pared y al caer por el agujero del piso aparecerá en la parte superior de la pantalla. Además cuando el mago termina de caer y llega a una viga deberá lanzar un hechizo congelante en la dirección en la que se esté moviendo en ese momento. Si el hechizo llega toca a un monstruo, éste quedará congelado y el hechizo desaparece. Si no encuentra enemigos en el camino, el hechizo desaparecerá al llegar a una pared.

4. Cuando un monstruo recibe un hechizo congelante deja de moverse por un tiempo. Si en ese tiempo el mago lo toca, el nonstruo congelado empieza a rodar por las vigas hasta caer al agujero del piso donde desaparece. Si mientras va rodando se encuentra con otro monstruo, lo arrastra con él. En caso de que el mago no llegue a tiempo el monstruo se recupera y vuelve a moverse.

5. Si en algún momento el mago toca un monstruo no congelado, la simulación termina y se considera perdida. Por el contrario, si el mago logra derrotar a todos los monstruos, la simulación se considera terminada exitosamente.

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 (y levantar así la calificación obtenida). Sugerimos a continuación algunas ideas:

- *Disparo especial*: Agregar al mago una habilidad especial, una barra de energía que cuando se llena, habilita al usuario disparar con la barra espaciadora un disparo especial (hechizos congelantes en 8 direcciones.). Este es un requerimiento deseable.
- *Sistema de puntos*: Contabilizar puntajes para poder competir con otros jugadores. Por ejemplo, otorgar puntos al derrotar a un enemigo, más puntos si un enemigo cae rodando con otro, agregar aleatoriamente items con premios y que al recogerlos suman puntos, etc.
- Agregar aleatoriamente *items especiales* distribuidos en el escenario de juego. Por ejemplo: Disparo especial, vidas extra, trampa de fuego (que el mago tenga que esperar un tiempo para pasar por ahi), o un nuevo enemigo.
- *Distintos enemigos*: Crear varios tipos de monstruos que se diferencien por ejemplo en el tamaño o en la velocidad al moverse.

- *Muchos niveles*: La posibilidad de comenzar un nuevo nivel después de ganar la simulación. Incluso podrá incrementar la dificultad a cada nivel.

Sobre el informe a desarrollar y las condiciones de entrega

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

- Una carátula del documento con los nombres, números de legajo y dirección de e-mail de los integrantes del grupo.
- Una introducción describiendo el trabajo realizado.
- Una explicación de cada clase implementada describiendo las variables de instancia y dando una breve descripción de cada método. Se pide además el invariante de representación de cada clase (salvo por la clase `Juego`).
- Un resumen de los problemas encontrados y de las decisiones de implementación que les permitieron resolverlos.
- Opcionalmente un diagrama de las clases utilizadas y las relaciones entre ellas.
- Un apéndice con el código impreso (se pueden omitir las partes del código que se consideren no relevantes para el trabajo).

Tanto este informe como el código fuente del trabajo práctico deben ser enviados a los docentes de la materia. El informe además deberá ser entregado en versión impresa. El trabajo práctico se puede hacer en grupos de hasta tres personas.

Fecha de entrega: Verificar en la página de la comisión correspondiente.

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;
```

[1] **Importante:** las palabras clave "`extends InterfaceJuego`" en la definición de la clase son fundamentales para el buen funcionamiento del juego.

```
    // Variables y métodos propios de cada grupo


    // ...


    Juego()
    {
        // Inicializa el objeto entorno
        entorno = new Entorno(this, "Torre mágica - Grupo X - v0.01", 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.
- 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` fué 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` en el código fuente, como por ejemplo las flechas de dirección del teclado o las teclas ALT, CTRL, SHIFT, etc. 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 CTRL
`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)`.