Programación I - Segundo Semestre 2020
Trabajo Práctico: Boss Rabbit Rabber

El *Concejo Orgánico Normativo y Ético para la Juventud Organizada (CONEJO)*, en el marco de su campaña de seguridad vial, nos solicitó el desarrollo de un video juego para ayudar a que niños y niñas tomen conciencia acerca de la seguridad vial y del respeto a las normas de tránsito. Para ello, nos encargaron un video juego similar al clásico Frogger, pero con algunas diferencias que mencionamos a continuación.

El Juego: Boss Rabbit Rabber

El conejo Boss Rabbit que vive en la luna, bajó al planeta tierra, por este motivo no sabe cuáles son las normas de tránsito, y por ende, no sabe cursar la calle.

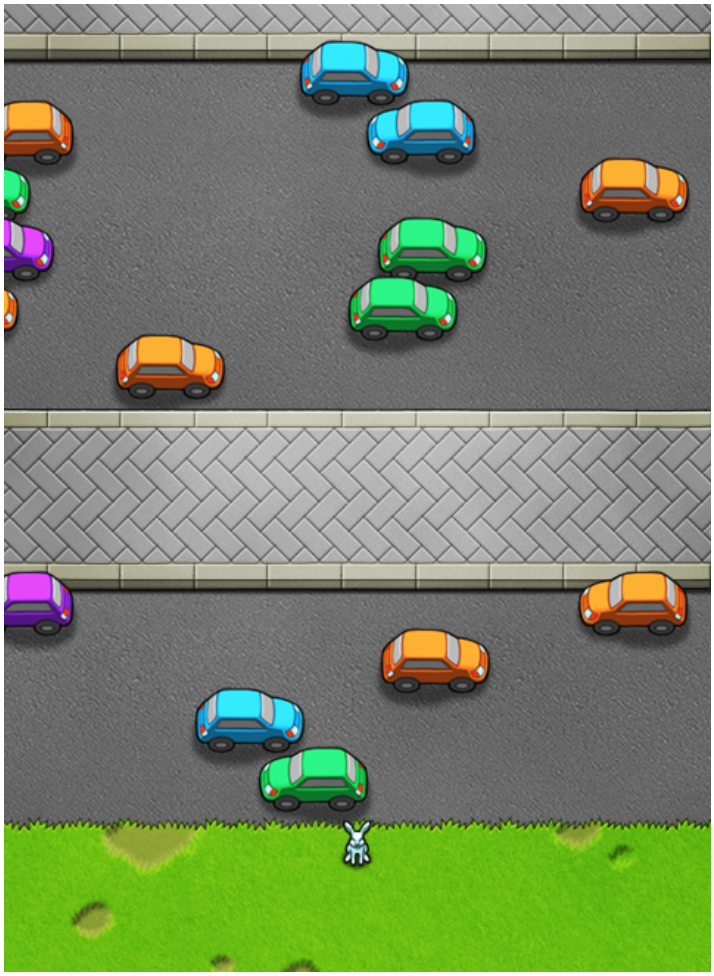

Figura 1: Ejemplo del juego.

El objetivo del juego es hacer que el conejito pueda cruzar las calles sin que lo atropelle ningún vehículo (ver Figura 1). Si un vehículo atropella al conejito, perdemos el juego.

Cabe aclarar, que en nuestro juego los vehículos y las calles se desplazan constantemente hacia abajo de la pantalla, creando la ilusión de "avance" del juego. Es decir, el conejo debe saltar hacia adelante para que no desaparezca por el borde inferior de la pantalla (cuando esto último sucede se pierde el juego).

Requerimientos obligatorios

1. Al iniciar el juego, el conejo Boss Rabbit debe aparecer centrado con respecto a los bordes izquierdo y derecho, y de la mitad hacia abajo de la pantalla (ver Figura 1).

2. Si se presiona una vez la fecha hacia arriba, el conejo debe saltar una vez, es decir, un solo salto cada vez que se presiona la fecha (no se permite que el conejo pueda saltar constantemente mientras se mantenga presionada la tecla).
   Si se presiona la fecha izquierda o derecha, el conejo debe desplazarse saltando hacia la dirección correspondiente, también de un solo salto a la vez.
   El conejo no puede saltar en otras direcciones.

3. El conejo constantemente está moviendose hacia abajo, de manera que hay que saltar hacia adelante para mantenerse en la pantalla.
   El conejo no puede salirse de los límites izquierdo, derecho, ni superior de la pantalla.
   Si el conejo sale del límite inferior de la pantalla porque se quedó parado sin avanzar, se da por perdido el juego.

4. En pantalla deberán verse una o dos calles, a criterio de cada grupo. Debe haber un espacio de separación entre las calles.
   Las calles deben moverse constantemente hacia abajo, para que el juego pueda "avanzar". Cuando una calle sale del límite inferior de la pantalla debe reaparecer en el borde superior.

5. Cada calle, debe tener entre cuatro y seis manos de vehículos, con direcciones alternadas, es decir, una mano con vehículos que vayan hacia la izquierda, la siguiente mano con vehículos que vayan hacia la derecha, la próxima mano va hacia la izquierda, y así siguiendo (ver Figura 1 donde hay dos calles: una con 6 manos de vehículos y otra con 4).

6. La cantidad de vehículos por mano queda a criterio de cada grupo, aunque se sugiere que sean entre tres y cinco por mano. Además, tener cuidado de que los vehículos no queden demasiado juntos de manera que sea difícil saltarlos ni demasiado separados de manera que no presente ninguna dificultad. Los vehículos no se pueden superponer entre ellos.

7. Todos los vehículos de una misma mano se desplazan juntos constantemente hacia la misma dirección, y con la misma velocidad.

   Los vehículos de distintas manos deben tener distintas velocidades.

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

8. Si un vehículo choca al conejo, perdemos el juego.

9. El gran maestro Kame Sennin le enseñó al conejo Boss Rabbit su técnica secreta, el *Kamehameha*. Cuando el conejo está apurado por cruzar la calle puede usar el Kamehameha para eliminar los vehículos de su paso.

   Cuando se presione la barra espaciadora, el conejo debe lanzar un Kamehameha. El Kamehameha se dezplaza hacia arriba y, si hace contacto con un vehículo, lo destruye y el vehículo no debe mostrarse más en pantalla. El Kamehameha que destruya a un vehículo también debe desaparecer de la pantalla. Aclaramos que tanto el vehículo como el Kamehameha deben ser eliminados, no vale únicamente "ocultar" las imágenes.

10. Cuando el conejo haya eliminado algún vehículo, luego de determinada cantidad de tiempo, a criterio de cada grupo, deben aparecer nuevos vehículos para que la calle no se quede sin vehículos.

11. Durante todo el juego, deberá mostrarse en algún lugar de la pantalla el puntaje acumulando, y la cantidad de saltos hacia adelante que hayamos dado. Los saltos laterales no deben contarse.
    Por cada vehículo que el conejo elimine, se incrementa en 5 su puntaje.

12. 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:

- Rayos conversores de zanahorias: que el conejo pueda convertir en zanahorias a los vehículos y se pueda comer las zanahorias sumando puntos por cada zanahoria que coma.
- Obstáculos: Colocar obstáculos que se interpongan en el camino del conejo y que no lo dejen avanzar, de modo que el conejo tenga que moverse hacia los lados izquierdo y/o derecho hasta rodear al obstáculo y poder pasar por el costado del mismo.
- Trenes: Agregar vías de tren y que cada determinada cantidad de tiempo pase un tren de modo que el conejito tenga que esperar hasta que pase el tren para no ser atropellado por el mismo.
- Niveles: La posibilidad de comenzar un nuevo nivel después de jugar determinada cantidad de tiempo o de puntaje o saltos acumulados, e incrementar la dificultad y/o velocidad de cada nivel.
- Ganar el juego: Al pasar cierto tiempo, conseguir una cierta cantidad de saltos o de puntaje acumulado, que el juego se dé por ganado.

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, `moyano-gutierrez-castro-tp-p1`.

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

Consultar la modalidad de entrega de cada comision (mail, drive, git, etc).

- 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, `moyano-gutierrez-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.

Apendice: 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, "Boss Rabbit Rubber - 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)`.