Programación I - Segundo Semestre 2019
Trabajo Práctico: Vegan Bird

El Juego: Vegan Bird

La Asociación Vegana Europea (AVE) nos contrató para desarrollar un video juego similar al Flappy Bird—pero vegano! (ver Figura 1).

En el juego, vemos un pájaro que se mantiene en vuelo cuando se presiona la flecha hacia arriba—si se deja de presionar la flecha, la gravedad hace que el pájaro vaya perdiendo altura.

A medida que el pájaro avanza se va encontrando con distintos obstáculos que tiene que evitar. Cabe aclarar, que en nuestro juego para simular el avance del pájaro, los obstáculos se desplazan hacia la izquierda de la pantalla, creando la ilusión de que el pájaro está avanzando. Es decir, el pájaro no avanza ni retrocede, sólo sube y baja, y son los obstáculos los que se mueven hacia la izquierda.

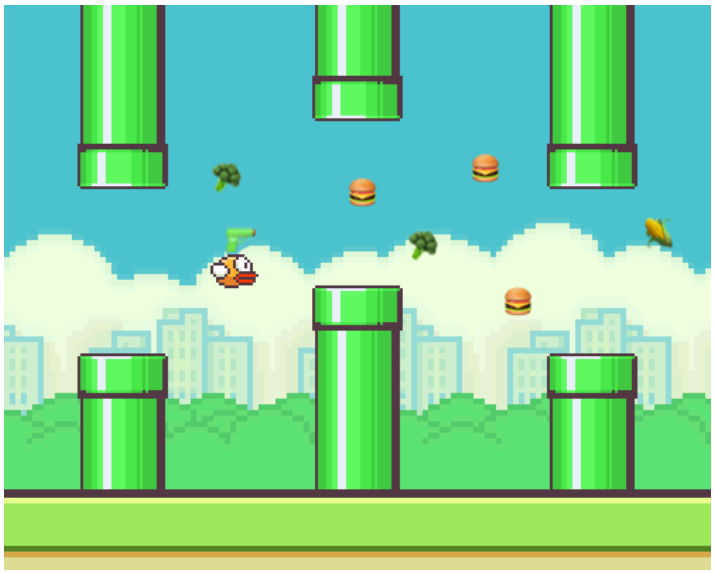

Figura 1: Ejemplo del juego.

Además, para hacer al juego vegano—a diferencia del Flappy Bird original—van apareciendo de forma aleatoria en la pantalla distintos alimentos:

- Si los alimentos son vegetales, el pájaro los puede comer, obteniendo así cinco puntos por vegetal consumido.
- Si los alimentos son hamburguesas (obviamente no veganas), el pájaro cuenta con un rayo conversor vegano que transforma a las hamburguesas en vegetales. Por cada hamburguesa transformada en vegetal, se obtienen tres puntos.

- Si el pájaro come una hamburguesa pierde cinco puntos.

El objetivo del juego es conseguir el mayor puntaje posible, evitando chocarse con los obstáculos. El juego termina cuando el pájaro choca contra uno de los obstáculos.

Requerimientos obligatorios

Los requerimientos obligatorios son los siguientes:

1. Al iniciar el juego debe aparecer un pájaro en la mitad en sentido vertical de la pantalla, y de la mitad hacia la izquierda en sentido horizontal de la pantalla (ver Figura 1).
2. También, al iniciar el juego, deberán aparecer obstáculos en la pantalla, con lugares libres que permitan al pájaro volar entre los obstáculos.

   La cantidad y distribución de los obstáculos queda a criterio de cada grupo, aunque se sugiere que no sean menos 3 en la parte superior y 3 en la parte inferior de la pantalla. Además, tener cuidado de que no quede una trayectoria libre demasiado ámplia.
3. Todos los obstáculos se desplazan juntos constantemente hacia la izquierda, y siempre con la misma velocidad.

   Cuando un obstáculo llega al borde izquierdo de la pantalla debe reaparecer en el borde derecho.
4. Durante el transcurso del juego, aleatoriamente, apareceran vegetales y hamburguesas en pantalla. Hay que tener cuidado de que no se superpongan ni con los obstáculos, ni con el pájaro, ni entre ellos.

   La forma en que los alimentos se mueven en pantalla queda a criterio de cada grupo—e.g. pueden flotar en el aire y volar, ó simplemente caerse lentamente debido a la fuerza de gravedad.
5. El pájaro sólo se mueve hacia arriba (cuando se presiona la flecha hacia arriba), y se cae automáticamente debido a la gravedad hacia abajo.
6. Si el pájaro toca un alimento, se lo come, y el alimento debe desaparecer de pantalla.
7. El pájaro lleva consigo en su lomo todo el tiempo un arma que dispara rayos conversores veganos.
8. La barra espaciadora producirá el disparo del rayo conversor vegano, el cual saldrá disparado en la dirección que apunte el arma. La dirección del arma se controla con la tecla `w` para girar en sentido antihorario, y con la tecla `s` para girar en sentido horario.

   Si el rayo toca una hamburguesa, esta debe transformarse en un vegetal. El rayo no afecta a ningún otro elemento del juego.
9. Durante todo el juego, deberá mostrarse en algún lugar de la pantalla el puntaje correspondiente obtenido: cinco puntos por vegetal consumido, tres puntos por hamburguesa transformada, y cinco puntos **menos** por hamburguesa consumida.

10. El pájaro no puede volar más allá del límite superior de la pantalla.
11. Si en algún momento el pájaro toca uno de los obstáculos ó se pasa del límite inferior de la pantalla, el juego se termina y se considera perdido.
12. El código del proyecto **deberá tener un buen diseño** de modo que cada objeto tenga **bien delimitadas sus responsabilidades**.

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:

- Obstáculos que se muevan verticalmente de manera que los huecos por donde pasa el pájaro se desplacen hacia arriba ó hacia abajo.
- Muchos niveles: La posibilidad de comenzar un nivel nuevo después de ganar el juego, e.g., al avanzar una cantidad determinada de obstáculos. Incluso se podría incrementar la dificultad y/ó velocidad de cada nivel.
- Efectos de sonido.

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:

- Un encabezado del documento con los nombres, números de legajo y dirección de email de cada integrante 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 de la clase `Juego`).
- Un resumen de los problemas encontrados y de las decisiones de implementación que les permitieron resolverlos.
- 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 debe hacer en grupos de **tres** personas.

Fecha de entrega: Verificar en el moodle 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;

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

    public Juego() {
        // Inicializa el objeto entorno
        entorno = new Entorno(this, "Arañas - 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.

[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` 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` 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)`.