

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 ¹signatura:

```
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, "Selva Mono Capuchino - 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: