1. Refuerzo de Java

Anexo V. Programación básica de juegos 2D

Programación de Servicios y Procesos

Nacho Iborra
Arturo Bernal
Álvaro Pérez
Javier Carrasco


Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Tabla de contenido

1. Programación básica de juegos 2D

Si bien Java FX se puede utilizar para el renderizado 3D, existen muchos buenos motores 3D que son mucho más avanzados y fáciles de usar que JavaFX. Sin embargo, dibujar y crear juegos simples en 2D no es muy complicado con JavaFX. Aquí, en la última parte de la unidad, se hará una breve introducción sobre cómo crear juegos con JavaFX.

En primer lugar, se creará una nueva aplicación JavaFX (no es una aplicación FXML esta vez), llamada *FX2DGame*. Una vez creado, puedes eliminar el fichero FXML y su controlador, no harán falta. Renombra la clase *HelloApplication.java* por *FX2DGame.java* y vacía el método *start()*.

1.1. Representar en un lienzo (*canvas*)

En Java FX, el objeto ***Canvas***[1] es un área rectangular en la que se pueden dibujar formas, textos o imágenes (*sprites*). Para dibujar en un *canvas* se utilizará la clase ***GraphicsContext***[2]. Puedes dibujar una forma o un texto definiendo su relleno y color de trazo (utilizando un objeto derivado de la subclase *Paint*[3], como *Color*[4]). También puedes aplicar un efecto mediante un objeto derivado de la clase *Effect*[5]. Con la clase *Image*[6], puedes cargar una imagen desde un archivo y dibujarla en el lienzo.

Comienza añadiendo el siguiente código al método *start()*. Estas líneas inicializan un lienzo de 400px de ancho y 200px de alto, y dibujarán un texto y formas. También dibuja una imagen llamada *image.png* ubicada en la carpeta raíz del proyecto. Descarga y copia allí cualquier imagen para probarlo:

```
@Override
public void start(Stage stage) throws IOException {
    StackPane root = new StackPane();
    Scene scene = new Scene(root, 400, 200);

    Canvas canvas = new Canvas(400, 200);
    root.getChildren().add(canvas);
    GraphicsContext gc = canvas.getGraphicsContext2D();

    // Effect (shadow) for drawing a rectangle
    DropShadow drop = new DropShadow(10, Color.GREY);
    drop.setOffsetX(4);
    drop.setOffsetY(6);

    // Draw a rectangle
    gc.setFill(Color.LIGHTSKYBLUE);
    gc.setEffect(drop);
    gc.fillRoundRect(10, 10, 380, 60, 10, 10);

    // Draw a text with a stroke
    gc.setEffect(null); // Unset the effect
```

1 Canvas (https://docs.oracle.com/javase/8/docs/api/java/awt/Canvas.html)
2 GraphicsContext (https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/GraphicsContext.html)
3 Paint (https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/Paint.html)
4 Color (https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/Color.html)
5 Effect (https://docs.oracle.com/javase/8/javafx/api/javafx/scene/effect/Effect.html)
6 Image (https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/Image.html)

```
    gc.setFill(Color.BLUE);
    gc.setStroke(Color.BLACK);
    gc.setLineWidth(2);
    Font myFont = Font.font("Times New Roman", FontWeight.BOLD, 36);
    gc.setFont(myFont);
    gc.fillText("Hello, World!", 60, 50);
    gc.strokeText("Hello, World!", 60, 50);
    // Draw an image
    Image img;
    try {
        img = new Image(Files.newInputStream(Paths.get("image.png")));
        gc.drawImage(img, 200, 100);
    } catch (IOException e) {
        e.printStackTrace();
        return;
    }

    stage.setTitle("2D Game Example");
    stage.setScene(scene);
    stage.show();
}
```

1.2. El bucle del juego

Si quieres animar cosas, deberás actualizar periódicamente la pantalla (*scene*). Para hacer eso, es necesario crear un bucle que continúe hasta que termine la animación (que puede ser cuando termina el juego). Por lo general, cuando comienzas a hacer tus primeros juegos (sin usar un motor de juego), suele hacerse manualmente. JavaFX no es un motor de juego, pero tiene algunas clases auxiliares como ***AnimationTimer***[7] para hacer animaciones o bucles de juegos. Tiene un método abstracto, ***handle()***, que tendrás que implementar ampliando una clase o usando una clase anónima.

El siguiente ejemplo dibuja la misma imagen utilizada en el ejemplo anterior de forma continua moviéndose horizontalmente.

```
@Override
public void start(Stage stage) throws Exception {
    StackPane root = new StackPane();
    Scene scene = new Scene(root, 400, 200);
```

7 AnimationTimer (https://docs.oracle.com/javase/8/javafx/api/javafx/animation/AnimationTimer.html)

```
    Canvas canvas = new Canvas(400, 200);
    root.getChildren().add(canvas);
    GraphicsContext gc = canvas.getGraphicsContext2D();

    Image img;
    try {
        img = new Image(Files.newInputStream(Paths.get("image.png")));
    } catch (IOException e) {
        e.printStackTrace();
        return;
    }

    // Game loop
    final long startNanoTime = System.nanoTime();

    new AnimationTimer() {
        public void handle(long currentNanoTime) {
            // Seconds in total from the beginning
            double t = (currentNanoTime - startNanoTime) / 1000000000.0;
            double x = Math.floor(t * 100) % 350; // Every 10ms move 1px
            gc.setFill(Color.WHITE);
            gc.fillRect(0, 0, 400, 200);
            gc.drawImage(img, x, 100);
        }
    }.start();

    stage.setTitle("2D Game Example");
    stage.setScene(scene);
    stage.show();
}
```

El método *handle()* de cada *AnimationTimer* creado se llamará aproximadamente 60 veces por segundo (60fps) de forma predeterminada. En este ejemplo, la imagen se mueve continuamente hacia la derecha y, cuando llega al final del *canvas* (más o menos), comienza de nuevo.

1.3. Eventos e interacción

En primer lugar, van a gestionarse los eventos de teclado. Por lo general, se tendrá que crear la escena para poder escuchar eventos (teclado, ratón, etc) registrándolos usando los métodos adecuados. Al registrar que teclas que se presionan, generalmente se quiere registrar todas las que se están pulsando al mismo tiempo. Para mantener esa pista, se utilizará una colección *Set* (no hay que repetir claves) para almacenar códigos de tecla.

Este ejemplo mueve la imagen hacia arriba, abajo, izquierda o derecha dependiendo de la(s) tecla(s) que se estén presionando, se comprobarán las teclas de cursor (las flechas).

```
private Set<KeyCode> activeKeys;

@Override
public void start(Stage stage) throws Exception {
    StackPane root = new StackPane();
    Scene scene = new Scene(root, 400, 200);

    Canvas canvas = new Canvas(400, 200);
    root.getChildren().add(canvas);
    GraphicsContext gc = canvas.getGraphicsContext2D();
```

```
    Image img;
    try {
        img = new Image(Files.newInputStream(Paths.get("image.png")));
    } catch (IOException e) {
        e.printStackTrace();
        return;
    }

    activeKeys = new HashSet<>();
    scene.setOnKeyPressed(e -> activeKeys.add(e.getCode()));
    scene.setOnKeyReleased(e -> activeKeys.remove(e.getCode()));

    // Game loop
    final long startNanoTime = System.nanoTime();
    new AnimationTimer() {

        private int x = 200 , y = 100;

        public void handle(long currentNanoTime) {

            if (activeKeys.contains(KeyCode.RIGHT))
                x++;
            if (activeKeys.contains(KeyCode.LEFT))
                x--;
            if (activeKeys.contains(KeyCode.UP))
                y--;
            if (activeKeys.contains(KeyCode.DOWN))
                y++;

            gc.setFill(Color.WHITE);
            gc.fillRect(0, 0, 400, 200);
            gc.drawImage(img, x, y);
        }
    }.start();

    stage.setTitle("2D Game Example");
    stage.setScene(scene);
    stage.show();
}
```

Si quieres manejar eventos del ratón, se podría hacer algo como esto dentro del método *start()* (habría que aumentar el alcance de las variables X e Y):

```
scene.setOnMouseClicked(e -> {
    x = (int) e.getX() - 25; // Image is 50x50
    y = (int) e.getY() - 25;
});
```

Estas líneas simplemente cambian la posición de la imagen a la nueva posición pulsada con el ratón.

1.4. *Sprites* y detección de colisiones

En los juegos, un *Sprite* representa una entidad visual en la pantalla. Suele tener métodos para actualizar la posición, dibujar la imagen que contiene, sacar en pantalla el rectángulo que representa el área que ocupa para calcular las colisiones, comprobar si choca con otro *sprite*, y muchos más…

Debería crearse una clase llamada *Sprite* que tenga todo lo necesario para utilizar *sprites* genéricos. Después, se crearán las clases derivadas necesarias para los elementos del juego (o, a veces, utilizar composición y hacer que la clase, por ejemplo *Enemy*, almacene un objeto *Sprite* en lugar de que heredar de él).

```
public class Sprite {
    private double x, y, width, height;
    private Image img;

    public Sprite(double x, double y, double height, double width, Image img) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.img = img;
    }

    public void draw(GraphicsContext gc) {
        gc.drawImage(img, x, y);
    }

    public Rectangle2D getBoundary() {
        return new Rectangle2D(x, y, width, height);
    }

    public boolean intersects(Sprite s) {
        return s.getBoundary().intersects(this.getBoundary());
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }
}
```

El siguiente ejemplo dibujará algunas *"lambdas"* en la pantalla del juego, y se podrá mover el logo de Java para atraparlas. Tan pronto como se capturen todas las lambdas de la pantalla, el juego terminará.

A continuación, se creará una instancia de algunos *Sprites* y se comprobará si hay colisiones, incrementando la puntuación y eliminándolos de la pantalla cuando el *sprite* principal choque con ellos. Incluso cuando se pretende hacer un juego simple, el resultado debería ser mucho más complejo y mejor organizado en clases que ahora, recuerda que esto es solo un ejemplo.

```
@Override
public void start(Stage stage) throws Exception {
    StackPane root = new StackPane();
```

```
    Scene scene = new Scene(root, 400, 200);

    Canvas canvas = new Canvas(400, 200);
    root.getChildren().add(canvas);
    GraphicsContext gc = canvas.getGraphicsContext2D();

    Sprite img;
    Image lambda;
    try {
        img = new Sprite(200, 100, 50, 50,
                new Image(Files.newInputStream(Paths.get("image.png"))));
        lambda = new Image(Files.newInputStream(Paths.get("lambda.png")));
    } catch (IOException e) {
        e.printStackTrace();
        return;
    }

    activeKeys = new HashSet<>();
    scene.setOnKeyPressed(e -> activeKeys.add(e.getCode()));
    scene.setOnKeyReleased(e -> activeKeys.remove(e.getCode()));

    scene.setOnMouseClicked(e -> {
        img.setX((int) e.getX() - 25); // Image is 50x50
        img.setY((int) e.getY() - 25);
    });

    // Random lambda sprites
    Random rand = new Random(System.nanoTime());
    List<Sprite> lambdas = new ArrayList<>();
    for (int i = 0; i < 5; i++) {
        lambdas.add(new Sprite(rand.nextInt(350), rand.nextInt(150),
                50, 50, lambda));
    }

    // Game loop
    new AnimationTimer() {
        public void handle(long currentNanoTime) {
            if (activeKeys.contains(KeyCode.RIGHT))
                img.setX(img.getX() + 1);
            if (activeKeys.contains(KeyCode.LEFT))
                img.setX(img.getX() - 1);
            if (activeKeys.contains(KeyCode.UP))
                img.setY(img.getY() - 1);
            if (activeKeys.contains(KeyCode.DOWN))
                img.setY(img.getY() + 1);

            // collision detection
            Iterator<Sprite> lambdasIter = lambdas.iterator();
            while (lambdasIter.hasNext()) {
                Sprite lambda = lambdasIter.next();
                if (lambda.intersects(img)) {
                    lambdasIter.remove();
                }
            }

            gc.setFill(Color.WHITE);
            gc.fillRect(0, 0, 400, 200);
            for (Sprite lambda : lambdas)
                lambda.draw(gc);
            img.draw(gc);
            if (lambdas.isEmpty())
                this.stop();
        }
    }.start();
```

```
    stage.setTitle("2D Game Example");
    stage.setScene(scene);
    stage.show();
}
```

1.4.1. Animando *sprites*

También pueden manejarse hojas de *sprites* (*sprite sheets*) con JavaFX y animar un *sprite* eligiendo qué *sprite* de la hoja se dibujará la siguiente vez. Existen varios enfoques para conseguirlo, pero quizás el más simple sea elegir una versión sobrecargada del método *GraphicsContext.drawImage*, el cual toma 9 parámetros:

- La imagen a dibujar.
- Las coordenadas X e Y del rectángulo que ocupará la imagen.
- El ancho y alto del rectángulo de la imagen.
- Las coordenadas X e Y en las que se colocará el *sprite* en pantalla.
- El ancho y alto que se dibujará en pantalla (normalmente estos valores serán los mismos que se especifiquen para el ancho y alto del rectángulo de la imagen).

Por ejemplo, si quieres dibujar una cuarta parte de las imágenes utilizadas en el ejemplo anterior, se podría hacer algo como esto (recuerda que son imágenes de 50x50).

```
gc.drawImage(img, 0, 0, 25, 25, x, y, 25, 25);
```

1.4.2. Escalar imágenes

JavaFX también permite escalar las imágenes que se utilicen en las aplicaciones (aunque no es recomendable, ya que suelen utilizan imágenes cuyo tamaño es demasiado grande para la aplicación). Para hacer esto, se crea la imagen con un constructor que especifique el ancho y alto deseados de la imagen, y dos parámetros booleanos adicionales para establecer si se quiere preservar la relación de aspecto, y si se desea aplicar un algoritmo de escalado suave (*smooth*).

```
Image img = new Image(Files.newInputStream(Paths.get("image.png")), 30, 30, false, false);
```