APAC 2. Abejas y osos

Programació de Serveis i Processos

CFGS Desenvolupament d'Aplicacions Multiplataforma

IES Jaume II El Just
Tavernes de la Valldigna
Departament d'Informàtica
Curs 2022-23

Contenidos

1 Abejas y osos

Vamos a crear un pequeño videojuego en JavaFX con **threads** y objetos compartidos, para abordar un problema de tipo productor-consumidor.

Nuestra aplicación dispondrá de dos elementos principales: abejas y osos, y un objeto compartido: un panal. La idea es que las abejas producirán miel en el panal, mientras que los osos la consumirán.

Además, dispondremos de otro tipo de consumidor: el jugador, un apicultor que será controlado por el usuario mediante el teclado y que tendrá por objetivo recoger tantas porciones de miel como pueda, antes de que se las coman los osos.

La aplicación se distribuye como paquete Gradle, y se invocará de la siguiente forma:

```
$ gradle run --args “nº_abejas nº_osos capacidad_panal”
```

Por ejemplo, si hacemos:

```
$ gradle run --args “5 5 3”
```

Tendremos 5 abejas productoras, 5 osos, y una capacidad en el panal de 5 unidades de miel. Podemos ver el resultado en la siguiente imagen:

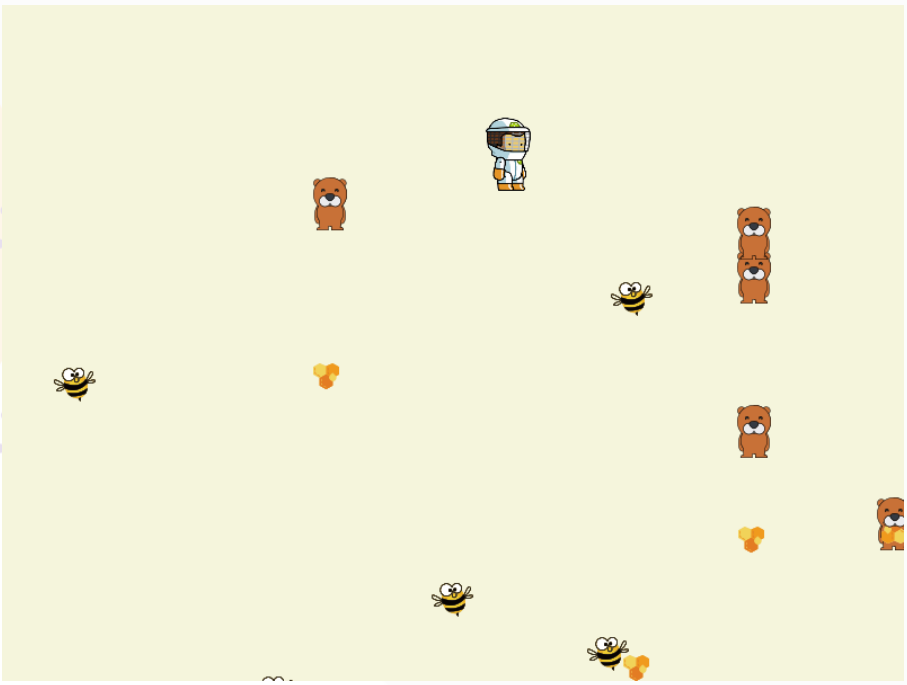

Figura 1: Ejemplo de aplicación

Cómo veis, el panal no es una ubicación fija concreta, sino que se encuentra distribuido por la pantalla, donde las abejas van soltando porciones de miel, y no podrá haber simultáneamente más porciones de miel en pantalla que las indicadas por la capacidad del panal.

Veamos algunos detalles de implementación

1.1 El fichero App.java

En este fichero se os proporciona la estructura principal de la aplicación gráfica.

En él encontramos la clase App, que deriva de la clase `Application` proporcionada por JavaFX.

Esta clase ya se os da prácticamente implementada, y encontraréis comentarios explicativos a lo largo del código. Cómo podréis comprobar, se definen los siguientes atributos:

- `W` y `H`, con el alto y el ancho de la ventana,
- `num_abejas`, `tamanyo_panal`, `num_osos`: con el número de abejas, osos y el tamaño máximo para el panal,
- `ListaAbejas`, `ListaOsos`, `ListaThreadsAbejas`, `ListaThreadsOsos`: con los vectores de objetos productores y consumidores y sus vectores de hilos asociados (si lo deseáis podéis modificar la representación como listas).
- `player` y `threadPlayer`, con el objeto jugador y el thread asociado, y
- `panal`, que es el objeto compartido de tipo Miel.

Por otro lado, esta clase contiene los métodos:

- `init`: Para inicializar todos los objetos de la clase se hace uso de este método, en lugar de utilizar el constructor. **Aquí tendréis que realizar algunas modificaciones para crear los diferentes objetos e hilos.**
- `start`: Se encarga de arrancar la aplicación, generando todos los recursos gráficos necesarios: gestionar la pantalla, preparar el canvas sobre el que se dibujará, y gestionar los tiempos. Este método es el que implementa el bucle del juego, y periódicamente, invocará los métodos correspondientes de los diferentes objetos productores y consumidores para dibujarse sobre un contexto gráfico. **Nuestra tarea en este método solo consistirá en eliminar los comentarios que invocan estos métodos cuando tengamos inicializados los vectores de los diferentes objetos.** · `main`: Es el método que pone en marcha la aplicación. Comprueba que le pasamos tres argumentos, e invoca el método `launch` del ciclo de vida de la aplicación JavaFX, iniciando este. Posteriormente, en el ciclo de vida ya se invocan los métodos `init` y `start`. Recuerda que el ciclo de vida en una aplicación JavaFX es el siguiente:

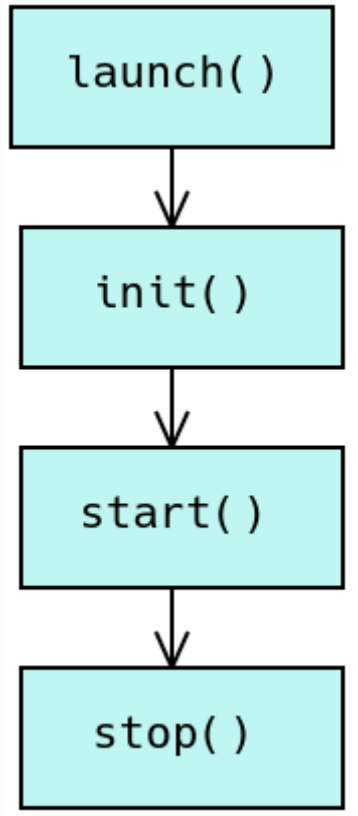

Figura 2: Ciclo de via de una aplicación JavaFX

1.2 El fichero Abeja.java y la clase Abeja

La clase Abeja implementará el productor de nuestro problema. Se proporciona una clase Abeja que habrá que hacer `Runnable`, y que tendrá los siguientes atributos:

- `private Integer x=100,y=100`: La posición x e y de la abeja (se tendrá que modificar en el constructor).
- `private Integer incx=3, incy=3;`: El número de píxeles (en horizontal y en vertical, respectivamente) que se incrementará en principio la posición de la abeja. Cómo veremos, esta será la base, y después el incremento puede ser aleatorio.
- `private Integer screenX, screenY;`: Son las dimensiones de la pantalla, que tendremos que pasarle cuando la generamos, para saber cuándo la abeja llega al final de la pantalla.
- `private Miel panal`: Será una referencia al objeto compartido, que también tendremos que inicializar en el constructor.

También se os proporciona el método `drawMe`, que dado un contexto gráfico proporcionado por la clase App y asociado al Canvas o lienzo, se dibuja en su posición.

En esta clase habrá que implementar:

- El método **constructor**, que tendrá que inicializar tanto las dimensiones de la pantalla que conoce la abeja como la posición inicial de esta, así como la referencia al objeto compartido panal.
- El **método que se invocará cuando se lance el objeto como hilo**, y que regirá el comportamiento de la abeja (ver a continuación).

1.3 ¿Cómo obtenemos una posición aleatoria en la pantalla?

Podemos utilizar el método `java.lang.Math.random()`, que obtiene un número aleatorio entre 0 y 1. Si queremos que este número esté entre dos valores concretos, la fórmula sería:

```
valor_inicial+(int)(java.lang.Math.random()*(valor_final-valor_inicial));
```

O bien, ***como alternativa***, para incluir el valor final como válido también podríamos utilziar:

```
(int)Math.floor(Math.random()*(valor_final-valor_inicial+1)+valor_inicial);
```

1.4 ¿Cuál será el comportamiento de la abeja?

La abeja irá volando por el escenario de manera infinita, en línea recta (incrementando su posición **x** e **y**), hasta llegar a un borde de la pantalla, momento en el cual cambiará de dirección. Este cambio de dirección se consigue modificando el incremento. Por ejemplo:

- Si volaba hacia la izquierda (por lo tanto, `incx` sería negativo) y llega al 0, cambiaríamos `incx` por un valor positivo.
- Si volaba hacia la derecha y llega al final, ahora cambiaremos `incx` a un valor negativo.
- Lo mismo aplicaríamos para los bordes superior e inferior e `incy`.
- Además, podemos añadir un cambio aleatorio cuando recalculamos `incx` e `incy` para que la dirección del vuelo no sea siempre en diagonal de *45°* (cosa que se produce cuando el incremento de `x` e `y` son iguales en valor absoluto). Para añadir este cambio, podemos hacer algo pareciendo a:

```
if (x>screenX) incx=-(3+(int)(java.lang.Math.random()*4));
```

Es decir, si la posición x supera el límite de la pantalla, ahora el incremento tendrá que ser negativo. Obtenemos un número aleatorio entre 3 y 7, de forma que la posición x ahora se decrementará cada vez entre 3 y 7 píxeles, hasta que llego a otro lado.

En este método también tendremos que producir de forma aleatoria miel en el objeto compartido, una vez lo tengamos creado, y siempre que quedo espacio en el panal. Para ello obtendremos un número aleatorio entre 0 y 1 y si este es mayor de 0.99 (probabilidad del 1%), invocaremos al método Produce correspondiente en el panal.

1.5 El fichero Miel.java y las clases Posicion y Miel

Este fichero define una clase Posicion, que representa un punto en la pantalla y la clase que implementará el objeto compartido `Miel`.

Esta clase `Miel` define una lista de elementos de tipos posición, que almacenará las diferentes unidades de miel que dejan las abejas. Además, también mantiene un entero, denominado cantidad que representará la cantidad máxima de unidades de miel que puede haber en el panal.

En esta clase tenemos implementados los siguientes métodos:

- método `getPositions`: Obtiene en forma de vector la lista de puntos que contienen miel.
- métodos `PanalDisponible` y `MielDisponible`, que indican respectivamente si queda lugar en el panal para ubicar miel (para que las abejas puedan producir), y si el panal tiene alguna porción de miel (para que los osos lo puedan consumir).

En esta clase habrá que implementar:

- El método `Produce(int x, int y)`: que, en caso de quedar sitio en el panal, añadirá un nuevo elemento de tipo Posición a este en la lista de posiciones. Esta posición será la propia de la abeja que lo ha invocado, y que nos la pasará en la propia invocación.
- El método `Recolecta(int x, int y)`: que, en caso de tener miel disponible, recolectará la porción de miel que hay en la posición *x,y*. Será el oso quién nos invocará este método cuando llegue a la posición. Para ello cual, habrá que recorrer toda la lista de posiciones y ver cuál es la que coincide con la posición *x,y* que nos han pasado. Dado que los movimientos no son de píxel en píxel, podemos dar por válido que esta posición sea próxima. Para tener en cuenta esto, podemos hacer algo como:

```
if ((x>=pos.getX()-5) && (x<=pos.getX()+5) &&
 (y>=pos.getY()-5) && (y<=pos.getY()+5)) ...
```

Siendo **x** e **y** las posiciones que nos han enviado, y `pos` una posición de las de la lista. En este caso, tenemos un margen de -5 a 5 píxeles de diferencia en *X* y *Y* para dar por válida la recolección.

De todos modos, si lo consideráis oportuno, podéis implementar vosotros el sistema de colisiones de la forma que deseéis.

1.6 El fichero Oso.java y la clase Oso

La clase `Oso` implementará el consumidor de nuestro problema. Se proporciona una clase `Oso` que habrá que hacer `Runnable`, y que tendrá los siguientes atributos:

- `private Integer x=100,y=100;` : La posición x e y del Oso (se tendrá que modificar en el constructor). `private Integer screenX, screenY`: Son las dimensiones de la pantalla, que tendremos que pasarle cuando la generamos.
- `private Miel panal;`: Será una referencia al objeto compartido, que también tendremos que inicializar en el constructor.
- `private Boolean isEating=false;`: Indica si el **Oso** está comiendo del panal en ese momento.

En esta clase habrá que implementar:

- El método **constructor**, que tendrá que inicializar tanto las dimensiones de la pantalla que conoce la abeja como la posición inicial de esta, así como la referencia al objeto compartido panal. · El **método que se invocará cuando se lance el objeto como hilo**, y que regirá el comportamiento del Oso.

1.7 Comportamiento del Oso

Lo osos tendrán que decidir, primeramente, -y si hay miel- qué porción de miel va a comerse. Para lo cual elegirá la que tiene más cerca (o si lo deseáis, una cualquiera). Para elegir la más próxima, habrá que recorrer la lista de posiciones del panal y calcular la diferencia entre la posición del oso y la de la porción de miel de la siguiente forma:

```
Distancia= valor_absoluto((pox_x_Oso-pos_x_miel)+(pox_y_Oso-pos_y_miel));
```

Una vez tengamos la posición más próxima, nos acercaremos a ella (modificando la posición del oso), primero en horizontal y después en vertical. Es decir, nos movemos a izquierda o derecha, según esté la miel que vamos a buscar, y cuando nos ubicamos arriba o bajo de ella, subiremos o bajaremos. Así, hasta que llegamos a la posición o a una posición próxima a la miel. Cuando llegamos, el oso podrá comerse la porción de miel, es decir, invocar el método `Recolecta`, enviándole su posición x e y. Mientras el Oso consume la miel, pasarán dos segundos, tiempo durante el cual, su estado habrá de indicar que está comiendo (`isRunning=false`), y pasar de nuevo a `true` pasados estos dos segundos.

1.8 El jugador

Se trata de una clase parecida al oso, con la diferencia que, en lugar de moverse para comer la porción de miel más próxima, será el usuario quién lo controle para ir a por una porción de miel.

El control del teclado ya lo proporciona la aplicación, y utiliza el método setDireccion del jugador para establecer la dirección como *UP*, *DOWN*, *LEFT* o *RIGHT*. Nuestra tarea será mover el jugador en el sentido que toque y encargarnos de la recolección de miel por su parte.

1.9 Posibles ampliaciones

Algunas ampliaciones que podéis hacer, además de personalizar lo que deseáis, pueden ser:

- Mantener un contador y mostrarlo por pantalla con las porciones de miel o puntuación que va consiguiendo el jugador (o si queréis también los osos).
- Hacer que cuando una abeja impacte con el jugador deje a ése “dormido” durante unos segundos.
- Incorporar un temporizador al juego, para que haga una cuenta atrás y el juego consista en obtener la mayor puntuación, por ejemplo, en un minuto. Tened cono cuenta que, en este caso, cuando se finalice, habrá que finalizar los threads.