Tecnología de la Programación II

Presentación de la Práctica 3: Mejora de la GUI Utilizando Threads

(Basado en la práctica de Samir Genaim)

Ana M. González de Miguel (ISIA, UCM)

Índice

1. Introducción

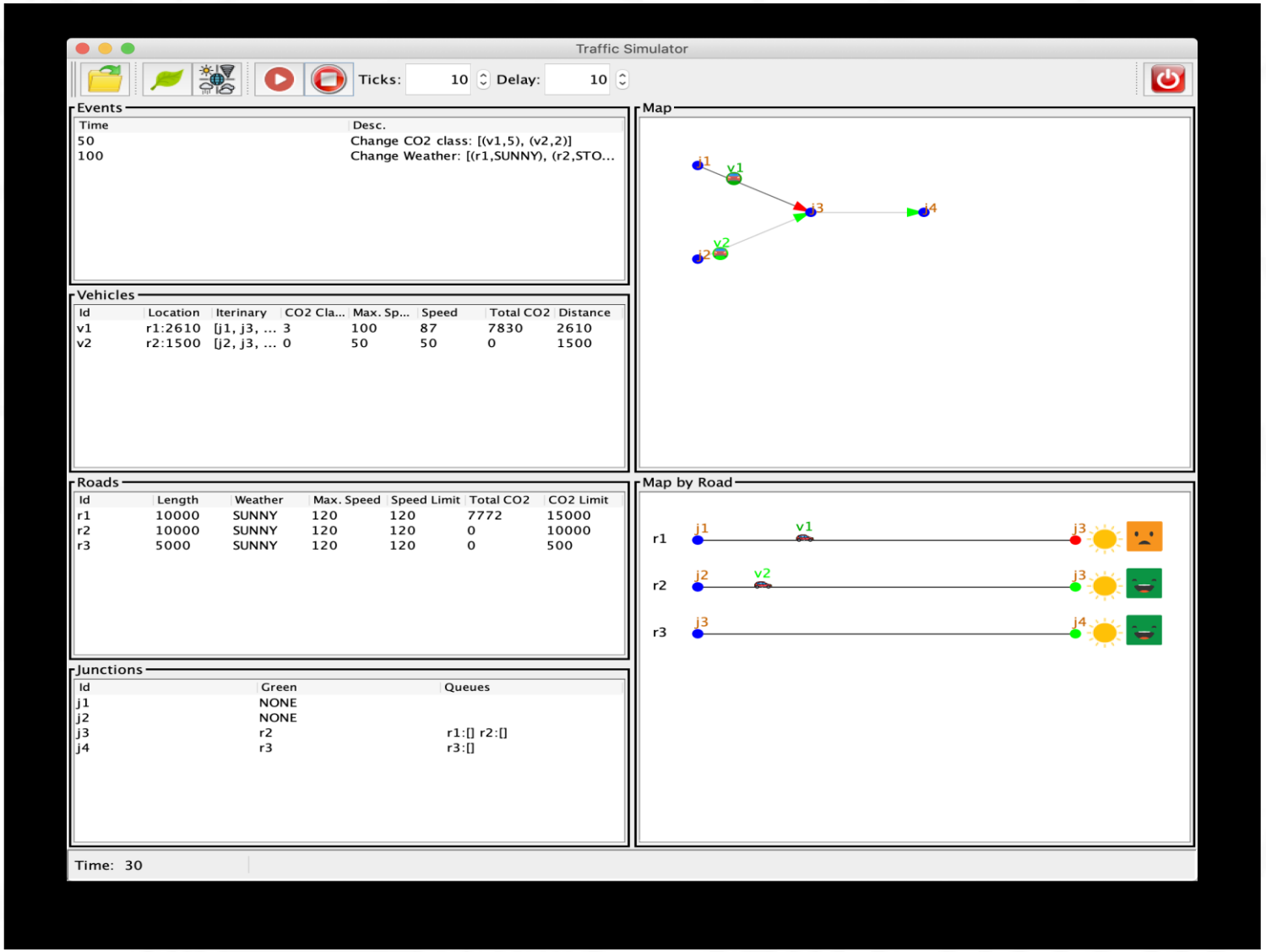
- ✓ La práctica 3 está especialmente dedicada a la **utilización de threads.**
- ✓ Hemos estimado 1 día para su realización.
- ✓ Este ejercicio no hay que entregarlo y por lo tanto no puntúa.

- ✓ En la práctica 2, hemos descrito dos enfoques para implementar la funcionalidad de los botones **run** y **stop**.
  - En el primero, utilizamos la cola de eventos de Swing para realizar la llamada recursiva a *run_sim*, y de este modo, entre una llamada y otra a *_ctrl.run(1),* Swing puede actualizar la vista y manejar las interacciones con el usuario.
  - En el segundo, sugerimos cambiar el método *run_sim* para simplemente llamar a *_ctrl.run (n),* en cuyo caso la vista permanece bloqueada mientras el simulador se está ejecutando y solo vemos el resultado final.
- ✓ Aunque hemos logrado un comportamiento razonable con el primer enfoque, teniendo en cuenta la capacidad de respuesta, podemos mejorarlo si utilizamos **programación multihilo**, que es lo que haremos en esta práctica.

2. Utilizando Hilos en Java

- ✓ Cambia el **panel de control** para incluir un nuevo *JSpinner Delay* (con un valor mínimo de 0, valor máximo 1000 y tamaño de paso 1) y una etiqueta correspondiente; consulte la Figura 1. Su valor representa un retardo entre pasos de simulación consecutivos, ya que ahora la ejecución será más rápida.
- ✓ Cambia el método *run_sim* para incluir un segundo parámetro *delay* de tipo *long*, y a continuación cambia su cuerpo a algo parecido al siguiente pseudocódigo:

```
while ( n>0 && (the current thread has not been intereptred) ) {
   // 1. execute the simulator one step, i.e., call method
   // _ctrl.run(1) and handle exceptions if any
   // 2. sleep the current thread for 'delay' milliseconds
   n--;
```


Traffic Simulator
Ticks: 10
Delay: 10
Events
Time
Desc.
50
Change CO2 class: [(v1,5), (v2,2)]
100
Change Weather: [(r1,SUNNY), (r2,STO...
Map
j1
v1
j3
j4
v2
j2
Vehicles
Id
Location
Iterinary
CO2 Cla...
Max. Sp...
Speed
Total CO2
Distance
v1
r1:2610
[j1, j3, ...
3
100
87
7830
2610
v2
r2:1500
[j2, j3, ...
0
50
50
0
1500
Roads
Id
Length
Weather
Max. Speed
Speed Limit
Total CO2
CO2 Limit
r1
10000
SUNNY
120
120
7772
15000
r2
10000
SUNNY
120
120
0
10000
r3
5000
SUNNY
120
120
0
500
Map by Road
r1
r2
r3
Junctions
Id
Green
Queues
j1
NONE
j2
NONE
j3
r2
r1:[] r2:[]
j4
r3
r3:[]
Time: 30

- ✓ El primer paso se hace en un try catch de *Exception*. Si hay excepción se usa *JOptionPane*.
- ✓ El segundo paso se hace en un try catch de *InterruptedException*. Si hay excepción se hace *Thread.currentThread().interrupt().*
- ✓ Este bucle ejecuta *n* pasos del simulador, pero se detiene si el hilo correspondiente ha sido interrumpido. En el paso 1, ejecuta el simulador un solo paso y captura cualquier excepción lanzada por el simulador/controlador, preséntala al usuario utilizando un cuadro de diálogo y sale del método *run_sim* inmediatamente (como en la práctica 2).

- ✓ En el paso 2, usa *Thread.sleep* para dormir el hilo actual durante *delay* milisegundos. Recuerda que si un hilo se interrumpe mientras duerme, el flag de interrupción no se establece a true , sino que se lanza una excepción. Por lo tanto, en tal caso, se debe interrumpir nuevamente el hilo actual al capturar la excepción correspondiente para salir del bucle (o simplemente salir del método con *return*).
- ✓ A continuación, cambia la funcionalidad de los botones **run** y **stop** para ejecutar *run_sim* en un nuevo hilo de la siguiente manera:
  - Añade en la clase *ControlPanel* un nuevo atributo llamado *_thread* del tipo **java.util.Thread**, y hazlo **volatile** ya que será modificado desde distintos hilos.

- Cuando se haga clic en **run** , desactiva todos los botones salvo **stop** y crea un nuevo hilo (asigna esa referencia a _*thread*) que hará lo siguiente:
  - (1) llama a *run_sim* con el número de pasos y el *delay* especificados en los correspondientes componentes JSpinner;
  - (2) habilita todos los botones, es decir, cuando termine la llamada a *run_sim*.
- No olvidéis lanzar el thread con el método *start*.
- Cuando se haga clic en **stop**, si hay un hilo ejecutándose, es decir, si _*thread* es distinto de null, entonces interrúmpelo para salir del bucle while y de este modo termina el hilo.

- ✓ Observar que la misma funcionalidad se puede implementar utilizando el atributo _*stopped*, que hemos utilizado en la práctica 2, en lugar de la interrupción de hilos.
- ✓ En tal caso, se debe declarar _*stopped* como **volátil**. Sin embargo, queremos que practiques las interrupciones de hilos y, atributo _*stopped* (elimina este campo de la clase *ControlPanel*).
- ✓ Cambia los métodos de observador, en todas las clases de la vista, de modo que cada vez que se modifique un campo o un componente Swing, se realice utilizando *SwingUtilities.invokeLater*. Esto es necesario ya que los métodos de observador ahora se ejecutan en un hilo distinto al hilo de Swing. Lo mismo se debe hacer cuando mostramos mensajes de error en el método *run_sim*

3. Opcional

- ✓ Implementa la funcionalidad descrita en la sección anterior utilizando un **Swing worker** en lugar de crear un nuevo hilo cada vez que se haga clic en el botón **run**.
- ✓ Declara el worker como atributo de la clase *ControlPanel* con *SwingWorker<Void,Void>*
- ✓ Cuando se pulsa el botón **run**, se deshabilitan los botones menos el **stop**, se crea el worker y en su método *doInBackground* se hace lo mismo que con el thread, teniendo en cuenta que para ver si ha sido interrumpido se usa el método *isCancelled*. Cuando se captura excepción *InterruptedException* no se hace nada.
- ✓ En el método *done* del worker se llama a *enableToolBar* para volver a habilitar todos los botones menos el **stop**.

- ✓ No olvidéis lanzar el worker con el método *execute*.
- ✓ Cuando se pulsa al botón **stop**, simplemente se cancela el worker con el método *cancel* si este es distinto de null.