Práctica 4: Simulador de Tráfico

Fecha de entrega: 19 de Marzo de 2018, a las 9:00

Objetivo: Programación Orientada a Objetos, Genéricos y Colecciones.

Introducción

Una *simulación* permite ejecutar un modelo en un ordenador para poder observar su comportamiento y aplicar este comportamiento a la vida real. Las prácticas del segundo cuatrimestre consistirán en construir un simulador de tráfico, que modelará *vehículos*, *carreteras* y *cruces*.

En esta práctica construiremos el simulador utilizando entrada/salida estándar (ficheros y/o consola). El simulador contendrá una colección de *objetos simulados* (vehículos y carreteras conectadas a través de cruces); otra colección de *eventos* a ejecutar; y un *contador de tiempo* que se incrementará en cada paso de la simulación. Un paso de la simulación consiste en realizar las siguientes operaciones:

1. *Procesar los eventos*. En particular estos eventos pueden añadir y/o alterar el estado de los objetos simulados;

2. *Avanzar* el estado actual de los objetos simulados atendiendo a su comportamiento. Por ejemplo, los vehículos avanzarán en su carretera correspondiente mientras dicha carretera finalice en un cruce con un semáforo en verde. Si el semáforo está en rojo, ningún vehículo de esa carretera podrá avanzar.

3. *Mostrar* el estado actual de los objetos simulados.

Los eventos se leen de un fichero de texto antes de que la simulación comience. Una vez leídos, se inicia la simulación, que se ejecutará un número determinado de pasos y, en cada paso, se mostrará el estado de la simulación, bien en la consola o en un fichero de texto.

Para facilitar la realización de la práctica, la dividiremos en dos partes. En la primera parte implementaremos un *simulador básico*, donde las carreteras, cruces y vehículos

tendrán un comportamiento simple. En la segunda parte añadiremos nuevos tipos de carreteras, vehículos y cruces. Antes de pasar a la segunda parte asegúrate de que la primera parte funciona correctamente.

Parte I: Simulador Básico

En esta parte tendremos únicamente tres tipos de objetos simulados:

- **Vehículos**, que viajan a través de carreteras y que se pueden averiar ocasionalmente. Cada vehículo tendrá un itinerario, que serán todos los cruces por los que tiene que pasar.
- **Carreteras de dirección única**, a través de las cuales viajan los vehículos. Cada carretera tendrá un cruce desde el que parte, y un cruce al que llega.
- **Cruces**, que conectan unas carreteras con otras, y que organizan el tráfico a través de semáforos. Cada cruce tendrá asociada una colección de carreteras que llegan a él, a las que denominaremos *carreteras entrantes*. **Desde un cruce sólo se puede llegar directamente a otro cruce a través de una única carretera.**

Cada objeto simulado tendrá un *identificador* único. A continuación pasamos a detallar las características concretas de cada uno de los objetos de la simulación.

Vehículos

Los vehículos tienen una velocidad máxima *velMaxima*, una velocidad actual *velActual*, la *carretera* en la que se encuentran, una *localización* en la carretera (distancia desde el comienzo de la carretera, que será la localización 0), y un *itinerario*, que consistirá en una colección de cruces a través de los cuales el vehículo debe viajar. Además los vehículos se pueden averiar, entrando en un estado de *avería*. Mientras un vehículo está averiado no puede circular, lo que provoca que la carretera en la que se encuentra tenga atasco, y que por tanto los vehículos situados detrás de él circulen más despacio. Un vehículo averiado se reparará después de un número determinado de pasos de simulación. Los vehículos ofrecen, entre otras, las siguientes operaciones:

- **avanza**. Cuando se le pide a un vehículo que avance, lo primero que hace el vehículo es comprobar si está averiado o no. Para ello comprueba si el contador **tiempoAveria** es positivo, y en tal caso, decrementa **tiempoAveria** en 1, y el vehículo no se mueve. Si el vehículo no está averiado (**tiempoAveria** es 0), entonces avanza su localización de acuerdo con su velocidad actual. La nueva localización será igual a la localización anterior más la velocidad actual. Si la nueva localización es igual o mayor que la longitud de la carretera por la que viaja, entonces pondremos su localización igual a la longitud de la carretera, y el vehículo entrará a la cola del correspondiente cruce (todas las carreteras tienen un cruce inicial y final). Los vehículos que esperan en la cola de un cruce no pueden avanzar, y permanecen en la cola del cruce hasta que el cruce determine que el vehículo debe moverse a la siguiente carretera, invocando al método **moverASiguienteCarretera** de vehículo.
- **moverASiguienteCarretera**. Solicita al vehículo que se mueva a la siguiente carretera. Para ello el vehículo sale de su carretera actual, y entra en la siguiente carretera que esté en su itinerario, en la localización 0. Observa que la primera vez que se

realiza esta operación sobre un vehículo, no existe ninguna carretera saliente, puesto que no ha entrado todavía en ninguna carretera. Además si el vehículo sale de la última carretera que existe en su itinerario, eso significa que ha llegado a su destino, y que no hay carretera entrante. Por tanto debemos marcar que el estado del vehículo es *haLlegado = true*.

- **setTiempoAveria**. Pone el estado del vehículo en modo avería durante n pasos. Si el vehículo ya estuviera averiado, entonces acumula n al tiempo de avería actual (contador **tiempoAveria**).
- **setVelocidadActual**. Pone la velocidad actual del vehículo a la que se indica por parámetro. Este método será utilizado por las carreteras para fijar la velocidad de los vehículos. En caso de que la velocidad que se desea poner exceda la velocidad máxima del vehículo, entonces la velocidad actual se pondrá igual a la velocidad máxima.
- **generaInforme**. Devuelve, en forma de String, el estado de un vehículo en formato textual. Para más información, ir a la sección 2.5. Por ejemplo:

```
[vehicle_report]
id = v1
time = 5
speed = 20
kilometrage = 60
faulty = 0
location = (r1,30)
```

representa un vehículo donde `id` es el identificador del vehículo, `time` es el paso de simulación en el cual el informe se ha generado, `speed` es la velocidad actual del vehículo, `kilometrage` es la distancia total recorrida por el vehículo, `faulty` es 0 si el vehículo no está averiado, o en caso de estarlo, es el tiempo que queda para que sea reparado y `location` es la localización del coche, que está compuesta por el identificador de la carretera y su localización en ella. En caso de que el coche ya haya llegado a su destino, entonces `location` será igual a `arrived`.

Observa que la velocidad de un vehículo debe ser 0 si el vehículo está averiado, está esperando en un cruce, o ya ha llegado a su destino.

Carreteras

Las carreteras tienen una *longitud*, un límite de velocidad *maxVel* y una *lista de vehículos* que están actualmente en ella, ordenados por su localización (la localización 0 ocupa la última posición). Además puede ocurrir que haya varios vehículos en la misma localización, en cuyo caso debe preservarse el orden de llegada a la lista, para que el vehículo que llegue el primero, abandone la carretera el primero. Las carreteras contienen las siguientes operaciones:

- **entraVehiculo**. Añade un vehículo a la lista de vehículos de la carretera. Este método lo invocan los vehículos cuando necesitan entrar a la carretera.
- **saleVehiculo**. Elimina un vehículo de la carretera. Este método lo invocan los vehículos cuando abandonan una carretera.

- **avanza**. Este método es invocado por el simulador para dar un paso en el estado de la carretera, y en particular para decir a cada coche de esa carretera que avance. Tras esto la carretera debe:

  1. Calcular su velocidad base (**velocidadBase**) utilizando la fórmula mín(m, $\frac{m}{\text{máx}(n,1)}$ +1), donde m es la velocidad máxima de la carretera y n es el número de vehículos en la carretera. **La división utilizada es la división entera**.
  2. Para cada vehículo:
     a) Se pone su velocidad a **velocidadBase**/**factorReduccion**, donde **factorReduccion** es una función del número de vehículos averiados (considerados obstáculos) que se encuentran delante[1] de este vehículo antes de que ningún vehículo haya avanzado. El factor de reducción (**factorReduccion**) es 1 por defecto si no hay obstáculos y 2 en otro caso.
     b) Se pide al vehículo que avance invocando a su método **avanza**.

- **generaInforme**. Devuelve en un String el estado de la carretera en formato textual. Para más detalles consultar la sección 2.5. Por ejemplo:

  ```
  [road_report]
  id = r3
  time = 4
  state = (v2,80),(v3,67)
  ```

  es la forma textual de una carretera donde `id` es el identificador de la carretera, `time` es el paso en el cual se ha generado el informe y `state` es una lista con los identificadores y posiciones de todos los vehículos de la carretera, en el mismo orden en el que están almacenados en la lista de vehículos.

Cuando todos los vehículos han avanzado, si varios están en la misma localización, sus órdenes relativos en la lista de vehículos debe ser el del orden de llegada. Recuerda que la lista tiene que estar ordenada por las localizaciones de los vehículos, siendo la localización 0 la que ocupa la última posición.

Cruces

Los cruces son los encargados de administrar las carreteras que entran al cruce, utilizando semáforos. Los semáforos pueden estar en verde, permitiendo a los vehículos entrar al cruce y salir por la carretera correspondiente de acuerdo a su itinerario, o pueden estar rojos, impidiendo que los vehículos circulen. En cada paso, **habrá únicamente un semáforo en verde**. Desde un cruce sólo se puede llegar directamente a otro cruce a través de una única carretera. Los cruces contienen las siguientes operaciones:

- **entraVehiculo**. Método invocado por los vehículos para entrar en la cola de una carretera entrante al cruce cuando llegan al cruce. Los vehículos en la cola del cruce están ordenados según su orden de llegada (el primero que llega al cruce es el primero que sale).

[1] Un vehículo A está delante de un vehículo B si la localización de A es estrictamente mayor que la localización de B.

- **avanza**. La operación de avanzar en un cruce consiste en:

  1. Preguntar al primer vehículo de la cola (y **solo** al primer vehículo en caso de que haya) asociada a la carretera entrante con el semáforo en verde, que avance a su siguiente carretera de acuerdo con su itinerario. Si el avance es posible, el vehículo se elimina de la cola.
  2. Actualizar el semáforo. Por defecto, un semáforo estará en verde solamente durante un paso. Transcurrido el paso se pondrá en rojo, y se pasará cíclicamente a la siguiente carretera entrante para poner su semáforo en verde, preservando el orden en el cual las carreteras se añadieron al cruce.

  Al comienzo de la simulación, antes de que se ejecute ninguna llamada a avanza, todos los semáforos estarán en rojo. La primera carretera entrante que pondrá su semáforo en verde será la primera que se añadió al cruce.

- **generaInforme**. Devuelve en forma de String el estado del cruce en formato textual. Para más detalles consultar la sección 2.5. Como ejemplo:

  ```
  [junction_report]
  id = j2
  time = 5
  queues = (r1,red,[]),(r2,green,[v3,v2,v5]),(r3,red,[v1,v4])
  ```

  donde `id` es el idenficador del cruce, `time` es el paso en el cual se ha generado el informe y, `queues` es la lista con el estado de todas las carreteras entrantes al cruce, en el mismo orden en que fueron añadidas. Cada cola se representa con el identificador de la carretera entrante, el estado de su semáforo asociado (verde o rojo) y la lista de vehículos en la cola.

Eventos

Los eventos permiten la interacción con el simulador, añadiendo vehículos, carreteras y cruces, y provocando que algunos coches se averíen. Cada evento tiene un *tiempo* asociado, en el cual debe ejecutarse. En su representación textual, la clave `time` será opcional. Si no aparece significa que dicho tiempo es 0, y que el evento debe ser ejecutado al inicio de la simulación. En cada paso t, el simulador ejecuta todos los eventos cuyo tiempo asociado es t, en el orden en que fueron añadidos a la cola de eventos. Para ejecutar los eventos, el simulador invocará a la operación **ejecuta** de los eventos.

Los eventos se leen de un fichero de texto, y tienen el formato que se describe a continuación.

Formato INI

Los ficheros que contienen los eventos utilizan el antiguo formato **INI** popularizado por su uso en Microsoft Windows hasta la llegada de los registros en Windows NT. Ver `https://en.wikipedia.org/wiki/INI_file` para más detalles. Este formato será también el utilizado para generar los informes por parte de los objetos de la simulación. En dicho formato, puede haber múltiples secciones, cada una de ellas con una etiqueta asociada. Dentro de cada sección, puede haber diversas líneas, cada línea con una clave textual y un valor asociado.

Junto con el enunciado de la práctica, se proporcionará un paquete que puede parsear y generar ficheros y secciones en formato **INI**. Este paquete permite además comparar dos ficheros o secciones para comprobar su equivalencia. Debes usar esta funcionalidad para asegurarte de que la salida de tu práctica es equivalente a la salida esperada que te proporcionaremos. Para las secciones en formato **INI** adoptaremos los siguientes convenios:

- Los *identificadores de los objetos* de la simulación deben ser strings compuestos únicamente de letras, números y subrayados ("_").
- Los *identificadores de las listas* deben contener únicamente identificadores válidos separados por comas, sin espacios en blanco.

Tienes que comprobar, antes de ejecutar un evento, que su identificador es válido. Finalmente ten en cuenta que el parser que suministramos tiene además una funcionalidad extra (no presente en el formato **INI** estándar) que permite ignorar secciones. Para ello basta añadir el símbolo "!" delante de la etiqueta de la sección. Por ejemplo, el parser ignorará la sección:

```
[!make_vehicle_faulty]
time = 2
vehicles = v1
duration = 3
```

Esta funcionalidad es muy útil para experimentar con el código. En lugar de eliminar la sección entera, se puede simplemente comentar. A continuación detallamos el formato **INI** de los distintos objetos de la simulación.

Evento Nuevo Cruce

Añade un nuevo cruce a la simulación, con los siguientes parámetros:

```
[new_junction]
time = <NONEG-INTEGER>
id = <JUNC-ID>
```

Evento Nueva Carretera

Añade una nueva carretera al simulador, con los siguientes parámetros:

```
[new_road]
time = <NONEG-INTEGER>
id = <ROAD-ID>
src = <JUNC-ID>
dest = <JUNC-ID>
max_speed = <POSITIVE-INTEGER>
length = <POSITIVE-INTEGER>
```

Evento Nuevo Vehículo

Añade un nuevo vehículo a la simulación. El itinerario es una lista de identificadores de cruces, y **debe contener al menos dos**.

```
[new_vehicle]
time = <NONEG-INTEGER>
id = <VEHICLE-ID>
max_speed = <POSITIVE-INTEGER>
itinerary = <JUNC-ID>,<JUNC-ID>(,<JUNC-ID>)*
```

Evento Vehículo Averiado

Provoca que todos los vehículos que aparecen en la lista de identificadores del evento `vehicles` pasen a estar averiados durante el tiempo especificado en el campo `duration`.

```
[make_vehicle_faulty]
time = <NONEG-INTEGER>
vehicles = <VEHICLE-ID>(,<VEHICLE-ID>)*
duration = <POSITIVE-INTEGER>
```

El Simulador

El simulador contiene una *lista de eventos*, una *estructura* para almacenar los objetos simulados (vehículos, cruces y carreteras) y un *contador de tiempo* (inicialmente a 0). El simulador debe ofrecer las siguientes operaciones:

- **insertaEvento**. Añade un evento a la lista de eventos. Los eventos tienen que estar ordenados por su atributo *tiempo*, y en caso de tiempos iguales, por su orden de llegada. Al añadir el evento debe comprobarse que su tiempo de ejecución asociado debe ser mayor o igual que el contador de tiempo del simulador.

- **ejecuta**. Recibe como parámetros el número de pasos de la simulación (*pasosSimulacion*), y el fichero de salida donde escribir los informes (del tipo `OutputStream`). El fichero de salida puede ser `System.out` para escribir en la salida estándar, o `new FileOutputStream(filename)` para escribir en un fichero. El pseudo-código asociado a este método es el siguiente:

  ```
  limiteTiempo = this.contadorTiempo + pasosSimulacion - 1;
  while (this.contadorTiempo <= limiteTiempo) {
    // 1. ejecutar los eventos correspondientes a ese tiempo
    // 2. invocar al método avanzar de las carreteras
    // 3. invocar al método avanzar de los cruces
    // 4. this.contadorTiempo++;
    // 5. esciribir un informe en OutputStream
    //    en caso de que no sea null
  }
  ```

  El informe se genera simplemente llamando al método **generaInforme** de cada uno de los objetos simulados en el siguiente orden: Primero se muestra la información de los cruces, después de las carreteras y finalmente de los vehículos. En cada caso, el informe del objeto simulado se generará en el orden en el cual fueron añadidos al simulador.

La clase Main

Recibirás, junto con la práctica, un esqueleto de la clase Main, que utiliza una librería externa para simplificar el parseo de las opciones de la línea de comandos. La clase Main debe procesar las siguientes líneas de comandos:

```
usage: es.ucm.fdi.launcher.Main [-h] [-i <arg>] [-o <arg>] [-t <arg>]
 -h,--help           Muestra la ayuda.
 -i,--input <arg>    Fichero de entrada de eventos.
 -o,--output <arg>   Fichero de salida, donde se escriben los informes.
 -t,--ticks <arg>    Pasos que ejecuta el simulador en su bucle principal
                     (el valor por defecto es 10).
```

Por ejemplo, podríamos ejecutar:

```
java Main -i eventsfile.ini -o output.ini -t 100
```

que lee los eventos del fichero `eventsfile.ini`, ejecuta la simulación durante 100 pasos y escribe el informe en el fichero de texto `output.ini`. Otro ejemplo sería:

```
java Main -i eventsfile.ini -t 100
```

que hace lo mismo que el ejemplo anterior, pero escribe los informes en la salida estándar.

Parte II: Objetos Avanzados

Esta segunda parte introduce nuevas clases de vehículos, carreteras y cruces que debes añadir al simulador sin romper la funcionalidad de la Parte I. Los eventos asociados a los objetos avanzados son similares a los eventos de los que heredan, extendidos con una nueva clave `type` que identifica la variante del evento que debe crearse. Por otro lado, dado que algunos objetos avanzados contienen información adicional, sus eventos asociados podrán tener nuevos campos.

De forma similar, los informes asociados a los objetos avanzados son idénticos a sus correspondientes objetos básicos (aquellos de los que heredan), pero deben incluir información sobre su atributo `type` y posiblemente información adicional específica para cada variante particular.

Nuevos Tipos de Vehículos

Aparecen dos nuevos tipos de vehículos: *Coches* y *Bicicletas*. Pasamos a describir con detalle cada uno de ellos.

Coches

Un coche es un vehículo que aleatoriamente puede averiarse con una cierta probabilidad. Cuando un coche ejecuta su operación **avanza**, primero comprueba si las siguientes condiciones se cumplen:

- El coche no está averiado.

- El coche ha viajado al menos **resistenciaKm** kilómetros desde la última vez que estuvo averiado. Ten en cuenta que esta distancia no es el total de la distancia recorrida y por lo tanto debes mantener almacenada esta información.
- Se genera un número aleatorio real x entre 0 (incluido) y 1 (excluido) y se comprueba si este número es menor que la probabilidad de avería (**probabilidadDeAveria**).

Si todas las condiciones son ciertas, entonces el coche pone su contador de avería (**tiempoAveria**) a un número entero aleatorio entre 1 y **duracionMaximaAveria** (ambos inclusive) y entonces ejecuta la operación estándar de avanzar (**avanza**). Observa que **setTiempoAveria** debe funcionar exactamente igual que en la parte I. Los coches se añaden a la simulación vía el evento:

```
[new_vehicle]
time = <NONEG-INTEGER>
id = <VEHICLE-ID>
itinerary = <JUNC-ID>,<JUNC-ID>(,<JUNC-ID>)*
max_speed = <POSITIVE-INTEGER>
type = car
resistance = <POSITIVE-INTEGER>
fault_probability = <NONEG-DOUBLE>
max_fault_duration = <POSITIVE-INTEGER>
seed = <POSITIVE-LONG>
```

La entrada `type = car` es sólo válida para esta clase de vehículos y debe ser incluida en su informe. El campo `fault_probability` es un número real entre 0 y 1, ambos inclusive. Finalmente, la clave `seed`, que explicaremos más tarde, es opcional y su valor por defecto es `System.currentTimeMillis()`.

Puesto que esta clase de vehículos tiene un comportamiento aleatorio, es difícil comprobar si la salida producida por el programa, sobre los ejemplos suministrados, encaja con los resultados esperados (que también se adjuntarán con el enunciado). Para solucionar este problema vamos a utilizar un *generador de números aleatorios* con una *semilla* de la siguiente forma:

- Esta clase de vehículos recibirá una semilla, `semilla` (un número de tipo `long`), como parámetro en su constructora, y almacenará `new Random(semilla)` en un atributo, por ejemplo, **numAleatorio**;
- Usaremos **numAleatorio.nextDouble()** para generar un número aleatorio x entre 0 y 1;
- Usaremos **numAleatorio.nextInt(duracionMaximaAveria) + 1** para generar un entero entre 1 y **duracionMaximaAveria**, ambos inclusive.

Cuando testeemos nuestro programa, utilizaremos la misma semilla que la utilizada para generar la salida de los ejemplos suministrados. De esta manera ambos ficheros deben coincidir.

Bicicletas

Las bicicletas se averían menos ya que su velocidad es más baja. Por lo tanto su método **setTiempoAveria** no se aplicará a las bicicletas que viajen a una velocidad menor o igual

que la mitad de su velocidad máxima. Las bicicletas se añaden a la simulación vía el siguiente evento:

```
[new_vehicle]
time = <NONEG-INTEGER>
id = <VEHICLE-ID>
itinerary = <JUNC-ID>,<JUNC-ID>(,<JUNC-ID>)*
max_speed = <POSITIVE-INTEGER>
type = bike
```

La entrada `type = bike` es sólo válida para esta clase de vehículos y por lo tanto debe ser incluida en su informe.

Carreteras

Tendremos dos tipos de carreteras: Autopistas y Caminos.

Autopistas

Las autopistas contendrán varios carriles para soportar un nivel de tráfico mayor que las carreteras convencionales. Por lo tanto los vehículos podrán circular a mayor velocidad incluso aunque haya varios vehículos averiados. Su factor de reducción **factorReduccion** es 1 siempre que haya mas carriles que vehículos averiados (i.e., **numCarriles >numObstaculos**) y 2 en otro caso.

La velocidad base (**velocidadBase**) de una autopista también considera el número de carriles y se define como mín$(m, \frac{m*l}{\text{máx}(n,1)} + 1)$, donde m es la velocidad máxima, n es el número de vehículos en la carretera y l es el número de carriles. La división es entera. Las autopistas se añaden usando el siguiente evento:

```
[new_road]
time = <NONEG-INTEGER>
id = <ROAD-ID>
src = <JUNC-ID>
dest = <JUNC-ID>
max_speed = <POSITIVE-INTEGER>
length = <POSITIVE-INTEGER>
type = lanes
lanes = <POSITIVE-INTEGER>
```

donde `type = lanes` es una entrada válida sólo para este tipo de carreteras y por tanto debe ser incluida en los informes.

Caminos

Por los caminos es más difícil conducir que por las carreteras convencionales en el caso de que haya vehículos averiados. Su factor de reducción (**factorReduccion**) será 1 más el número de coches averiados. Su velocidad base (**velocidadBase**) se define como la velocidad máxima de la carretera. Los caminos se añaden utilizando el siguiente evento:

```
[new_road]
time = <NONEG-INTEGER>
```

```
id = <ROAD-ID>
src = <JUNC-ID>
dest = <JUNC-ID>
max_speed = <POSITIVE-INTEGER>
length = <POSITIVE-INTEGER>
type = dirt
```

donde la entrada `type = dirt` es una entrada válida sólo para esta clase de objetos y por lo tanto debe incluirse en sus informes.

Cruces

Existen dos nuevas clases de cruces: Circulares (Round-robin) y Congestionados (Most-crowed).

Cruces Circulares

En estos cruces la duración de los semáforos en verde es mayor. En lugar de ir cambiando el semáforo en cada paso de la simulación como se hace en los cruces convencionales, en este tipo de cruces el semáforo estará en verde durante un intervalo de tiempo [**minValorIntervalo**,**maxValorIntervalo**]. Estos cruces se representan en modo textual como:

```
[new_junction]
time = <NONEG-INTEGER>
id = <JUNC-ID>
type = rr
max_time_slice = <POSITIVE-INTEGER>
min_time_slice = <POSITIVE-INTEGER>
```

Al comienzo de la simulación el intervalo de tiempo (**intervaloDeTiempo**) de cada carretera entrante se pone a **maxValorIntervalo** y su correspondiente **unidadesDeTiempoUsadas** se pone a 0. Ten en cuenta que al comienzo de la simulación, antes de ejecutar ninguna llamada a **avanza**, todos los semáforos están en rojo.

Cuando se le pide al cruce que actualice sus semáforos, primero comprueba si el cruce que tiene el semáforo en verde ha consumido su intervalo (i.e., **unidadesDeTiempoUsadas** es igual a **intervaloDeTiempo**). Si fuera así, entonces el cruce hace lo siguiente:

- Pone el semáforo verde a rojo;
- Actualiza el **intervaloDeTiempo** de la correspondiente carretera entrante como sigue:
  1. Si el intervalo de tiempo **ha sido completamente usado** (en cada paso de la simulación ha pasado un vehículo de esa carretera entrante), entonces **intervaloDeTiempo** se pone a mín(**intervaloDeTiempo** $+ 1$, **maxValorIntervalo**).
  2. Si el intervalo de tiempo **no se ha usado nada** (ningún vehículo ha pasado mientras el semáforo estaba en verde), entonces **intervaloDeTiempo** se pone a máx(**intervaloDeTiempo** $- 1$, **minValorIntervalo**).
  3. En otro caso no modificados **intervaloDeTiempo**.
- Pone **unidadesDeTiempoUsadas** a 0;

- Pone en verde el semáforo de la siguiente carretera entrante, en el orden en que fueron añadidas. Observa que cuando todos los semáforos están en rojo, la siguiente carretera entrante es la primera que se añadió al cruce.

Los vehículos de la cola se ordenan respetando el orden de llegada (el primero que entra es el primero que sale). Los informes para estos cruces contienen la misma información que para los cruces convencionales, pero ahora, para cada semáforo verde, mostramos también las unidades de tiempo restantes. Por ejemplo, para:

```
[junction_report]
id = j2
time = 5
type = rr
queues = (r1,red,[]),(r2,green:3,[v3,v2,v5]),(r3,red,[v1,v4])
```

se indica que la carretera `r2` tiene el semáforo en verde y todavía le quedan tres pasos más para cambiar a rojo.

Cruces Congestionados

Cuando un cruce está congestionado, la siguiente carretera entrante en poner su semáforo en verde es aquella que tiene más vehículos en su cola. En caso de que haya varias carreteras entrantes con la misma longitud en sus colas, entonces se tiene en cuenta el orden en que fueron añadidas a la simulación.

Al inicio de la simulación **intervaloDeTiempo** y **unidadesDeTiempoUsadas** son 0. Cuando se pide un a cruce congestionado que actualice su semáforo en verde, entonces primero comprueba si el cruce que actualmente tiene el semáforo en verde ha consumido su intervalo de tiempo (i.e., **unidadesDeTiempoUsadas** es igual a **intervaloDeTiempo**). Si este fuera el caso, entonces el cruce se comporta como sigue:

- Poner el semáforo verde a rojo;
- Se busca la carretera entrante (distinta a la actual) con más vehículos en la cola respetando el orden de llegada. Se pone su semáforo a verde y se le asigna un **intervaloDeTiempo** igual a máx$(\frac{n}{2}, 1)$, donde n es el número de vehículos en la cola de la carretera, y **unidadesDeTiempoUsadas** se pone a 0. La división es entera.

Los vehículos de la cola de las carreteras entrantes a estos cruces se ordenan de acuerdo al orden de llegada. Además al inicio de la simulación, antes de ejecutar ninguna llamada a **avanza**, todos los semáforos están en rojo.

Los eventos que definen los cruces congestionados tienen el siguiente formato:

```
[new_junction]
time = <NONEG-INTEGER>
id = <JUNC-ID>
type = mc
```

La entrada `type = mc` es única para esta clase de objetos y por tanto debe aparecer en sus informes.

Instrucciones para realizar la Práctica

- El manejo de errores, tales como añadir objetos de la simulación con el mismo identificador, referenciar objetos simulados que no existen, utilizar valores inválidos de parámetros, etc., deben ser controlados usando tratamiento de excepciones.
- Elige las colecciones que vas a utilizar teniendo en cuenta el uso que vas hacer de ellas.
- Se adjuntará, junto con el enunciado de la práctica, un conjunto de ficheros de eventos con la correspondiente salida. Debes comprobar que tu práctica, para los ficheros de entrada suministrados, genera la misma salida. Además se publicará el código asociado al parser de eventos así como un esqueleto de la clase **Main** que incluirá algunos comandos de ejecución.