ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

SIMUCOM: Simulador de un Conmutador Multietapa

AUTOR: Gregorio Cañavate Cruzado
DIRECTOR: Francesc Burrull i Mestres

Enero / 2004

Autor	Gregorio Cañavate Cruzado
Director(es)	Francesc Burrull i Mestres
Título del PFC	SIMUCOM: Simulador de un Conmutador Multietapa
Descriptores	Simulador, Conmutación Espacial

Resúmen

Un elemento esencial en las redes de conmutación de circuitos son los conmutadores propiamente dichos. Como complemento a su estudio se pueden utilizar simuladores didácticos, que permiten una mejor comprensión de los conmutadores, así como la obtención de datos estadísticos.

En este proyecto se ha realizado un simulador de un conmutador multietapa, herramienta que servirá para la docencia en la Escuela Técnica Superior de Ingeniería en Telecomunicaciones (E.T.S.I.T.) de la Universidad Politécnica de Cartagena (U.P.C.T.) disponiendo además del código fuente de la herramienta.

Titulación	Ingeniero Técnico de Telecomunicación
Intensificación	Telemática
Departamento	Departamento de Tecnologías de la información y las comunicaciones
Fecha de Presentación	Enero - 2004

Índice General

Capítulo 1
Introducción

Un elemento esencial en las redes de conmutación de circuitos son los conmutadores propiamente dichos. Como complemento a su estudio se pueden utilizar simuladores didácticos, que permiten una mejor comprensión de los conmutadores, así como la obtención de datos estadísticos.

En este proyecto se ha realizado un simulador de un conmutador multietapa, herramienta que servirá para la docencia en la Escuela Técnica Superior de Ingeniería en Telecomunicaciones (E.T.S.I.T.) de la Universidad Politécnica de Cartagena (U.P.C.T.) disponiendo además del código fuente de la herramienta.

Capítulo 2
Objetivos

Nuestro objetivo es programar un simulador didáctico de un conmutador espacial multietapa, de características similares a uno ya realizado (el Matswit), con el fin de introducir algunas mejoras en el mismo y disponer del código fuente para posibles modificaciones.

Estas mejoras consistirán en:

- Modernización de la interfaz gráfica, haciendo más agradable el entorno visual al alumno.
- Facilitar las comprensión de los algoritmos de distribución del Matswit.
- Corregir ciertos aspectos teóricos para facilitarle la comprensión de los conceptos que se intentan enseñar con este simulador.
- Integración de diversos apartados del programa inicial en uno solo, consiguiendo 2 modos: Uno simple, con una duración de las llamadas infinita y otro continuo con llamadas de duración finita.
- Dejar el código abierto a posibles mejoras futuras.
- Hacer que el programa sea “portable”, que podamos ejecutarlo en cualquier sistema operativo: Windows, LINUX, ...

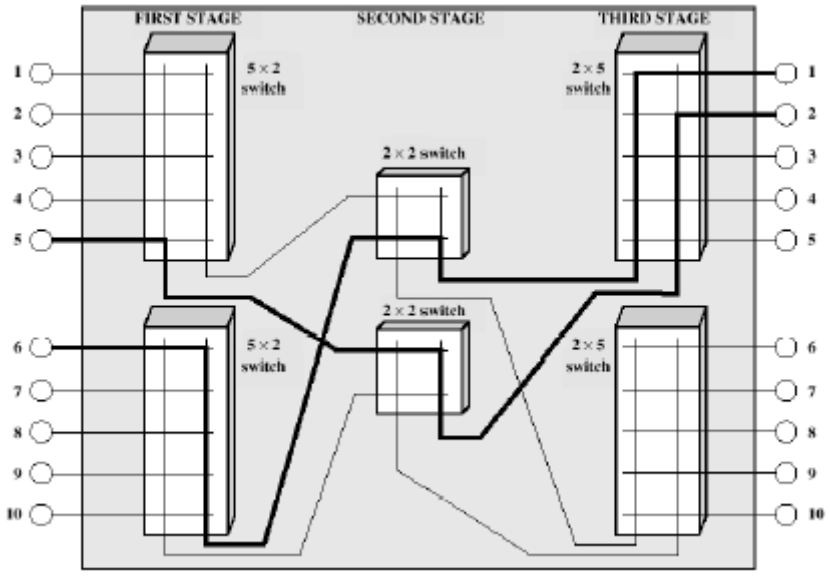

Figura 1: Ejemplo de conmutador espacial multietapa

Capítulo 3
Análisis y diseño

3.1 Ingeniería inversa

Nuestro objetivo principal es realizar un programa que se comporte de manera parecida a otro ya hecho, por lo que debemos recurrir a la ingeniería inversa.

La ingeniería inversa consiste en desmontar un objeto para ver cómo funciona y, de ese modo, duplicar o mejorar el mismo. En el ámbito informático, existen dos maneras de llevarla a cabo:

Empleando métodos de “caja negra”, que consiste en observar desde fuera el comportamiento de un programa al que se somete a una serie de casos de uso. Esto es factible en programas pequeños.

El otro consistiría en descompilar un programa de modo que se obtenga el código fuente del mismo a partir del código que ejecutamos en nuestro ordenador. Esto sería posible si el código se hubiera dejado a nuestro alcance por el programador. Es la única manera con programas grandes, como sistemas operativos.

En nuestro caso hemos considerado que lo más apropiado era utilizar el método de “caja negra”, tanto por no tratarse de un programa grande, como por no necesitar obtener un “calco” del programa original, como por no disponer de la información de depuración.

Nos disponemos a comentar las principales características del programa del que partimos (el Matswit):

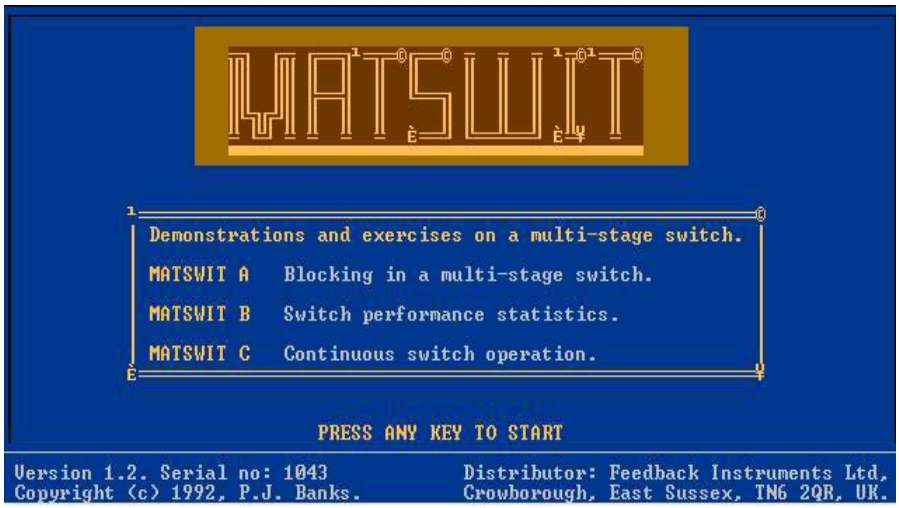

Figura 2: Matswit – Menú de inicio

El conmutador de este programa se compone de 16 entradas y 16 salidas. Internamente consta de 3 etapas: Matrices de entrada, matrices intermedias o de distribución y matrices de salida.

El simulador tiene tres opciones: Opción A, opción B y opción C.

3.2 Opción A

La opción A nos muestra el camino que sigue la "llamada" desde una entrada aleatoria hasta otra salida también aleatoria. Ese camino o matriz de distribución (intermedia), se decide en base a tres algoritmos diferentes. La principal función de esta opción A es mostrar el comportamiento de estos algoritmos.

La llamada será de duración infinita, con lo que una vez que se establezca no se terminará hasta que finalice el programa.

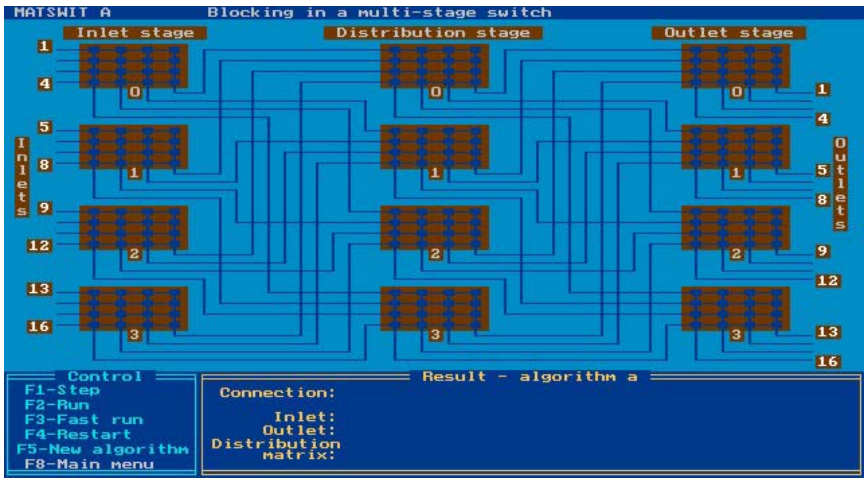

Figura 3: Matswit - Opción A

En la etapa de entrada se tiene 4 matrices crossbar 4X4, esto es, con 4 entradas y 4 salidas. Lo mismo pasa con la etapa intermedia y lo mismo con la etapa de salida.

Los tres algoritmos para la elección de la matriz de distribución son:

1. **Algoritmo A:**

En primer lugar, intenta conectar la entrada y la salida por la primera matriz intermedia (la matriz 0), luego irá variando en una unidad la matriz por la que quiere conectar, si el camino está ocupado y no se puede conectar por esa matriz, pasará a la siguiente.

Al llegar a la última (la 3) la siguiente matriz volverá a ser la primera (la 0). Si no se pudo por ninguna de las matrices, la conexión se habrá bloqueado.

Este Algoritmo se denomina de rotación.

2. **Algoritmo B:**

Siempre se intenta pasar por la matriz intermedia 0, si no se es posible y esa matriz está ocupada, se intentará con la siguiente y así ...

Si se llegara a tercera matriz y no se pudiera conectar, estaríamos en una situación de bloqueo, no habríamos podido conectar por ninguna.

Por su comportamiento se denomina algoritmo secuencial.

3. **Algoritmo C:**

Método aleatorio. Escoge la matriz de distribución al azar.

Si no se pudo conectar por ninguna de las matrices (4 intentos fallidos), la conexión se bloquearía y la ejecución de la secuencia habría terminado.

Las posibilidades de ejecutar el programa son:

- Step, en la que se avanza en la ejecución estableciendo conexiones de una en una, paso a paso.
- Run, en la que se ejecuta el simulador hasta que se bloquee o se hayan realizado las 16 conexiones sin problemas.
- Fast Run, es igual que Run pero no dibuja las conexiones, solo se reflejan en el cuadro inferior.

Las conexiones realizadas se muestran en una tabla que consta de las entradas, salidas, y matrices de distribución de cada una de las 16 posibles conexiones.

3.3 Opción B

La Opción B realiza un número de ejecuciones determinado el proceso de la Opción A. Se reflejan los resultados finales en un cuadro de estadísticas.

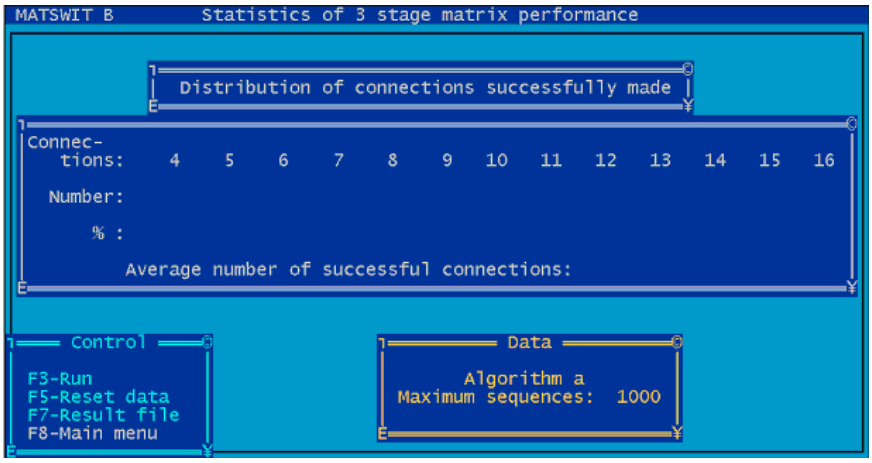

Figura 4: Matswit - Opción B

Cada vez que se realiza una ejecución se pueden conseguir de 4 a 16 conexiones. Así, el cuadro contiene el número de ejecuciones que se han conseguido realizar un número de conexiones (de 4 a 16) de manera satisfactoria, así como el porcentaje en que se produjeron cada una y la media total de conexiones hechas por simulación.

También se nos da la posibilidad de guardar los resultados en ficheros.

Nótese que la opción B simula la misma situación que la opción A, repetida tantas veces como se indique. Por tanto se justifica uno de los objetivos del proyecto, que es integrar estas 2 primeras opciones en una presentación común.

3.4 Opción C

En esta opción las llamadas son de duración finita y en régimen continuo, por lo que en esta opción habrá situaciones de conexión, desconexión y bloqueo.

Un bloqueo no supone la terminación de la simulación, tan solo se habrá perdido esa conexión y el programa seguirá su curso.

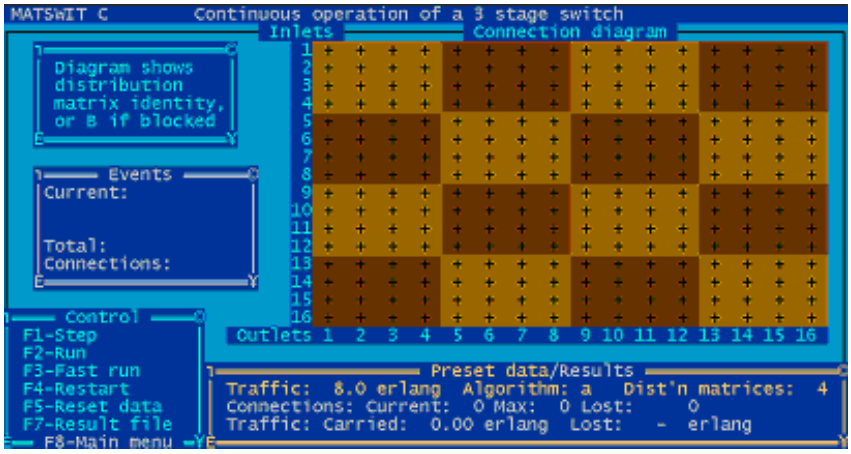

Figura 5: Matswit - Opción C

Esto se nos muestra es un diagrama de conexiones que está formado por una matriz con 16 filas representando a las entradas y 16 columnas representando a las salidas.

En él se ve la entrada, la salida y el número de matriz intermedia por la que se ha conectado. Esta matriz se verá reflejada dentro de la celda que resulta de la intersección de la fila y columna. Como en las opciones anteriores, se puede escoger entre 3 algoritmos de selección de matriz intermedia. El programa mostrará una B dentro de la celda si ha habido un bloqueo. En caso de desconexión mostrará de forma parpadeante la conexión a desconectar.

Por lo tanto por cada evento se tiene una celda, del que se sabe su fila (entrada) y columna (salida). Dentro de esa celda está el estado de dicho evento y su matriz de distribución en caso de que no se haya bloqueado.

En esta opción se tiene la posibilidad de aumentar las matrices intermedias hasta siete.

Uno de los datos que el programa nos pide es lo que el Matswit llama Tráfico Ofrecido (más tarde veremos que este no es tal). Debido a esto se obtienen nuevos valores tales como Tráfico Cursado, Tráfico Perdido, conexiones en curso, perdidas y máximas conexiones que ha habido en curso.

La distintas posibilidades de ejecución son:

- Step: Se podrá ir paso a paso en la ejecución, se mostrará en cada turno si ha habido conexión, desconexión o bloqueo.
- Run: Empezar la simulación y finalizarla cuando se produzca un evento de bloqueo.
- Fast Run: Finalizar la simulación transcurrido un número determinado de eventos.

También se ofrece la posibilidad de guardar los resultados obtenidos en archivos.

Capítulo 4
Ámbito de aplicación

4.1 Elección del lenguaje de desarrollo

En primer lugar para escoger el lenguaje de desarrollo elegido se tomaron en cuenta los siguientes lenguajes:

C:

C es un lenguaje creado a principios de los años 70 para escribir el sistema operativo Unix. Es el más utilizado para crear sistemas operativos y aplicaciones de sistema.

El lenguaje C se clasifica como un lenguaje compilado. En lenguajes compilados se convierte el código fuente a código objeto (compilación), y éste a código ejecutable (enlazado). Este es el caso del lenguaje C.
Podemos decir que el lenguaje C es un lenguaje de nivel medio, ya que combina elementos de lenguaje de alto nivel con la funcionalidad del lenguaje ensamblador. Es un lenguaje estructurado, ya que permite crear procedimientos en bloques dentro de otros procedimientos. Hay que destacar que el C es un lenguaje portable en el sentido de que permite utilizar el mismo código en diferentes equipos y sistemas informáticos: el lenguaje es independiente de la arquitectura de cualquier máquina en particular.

Es muy eficiente, pero limita mucho nuestras posibilidades al no ser orientado a objetos y no tener una “portabilidad total” , no podemos ejecutar el programa en distintas plataformas sin necesidad de recompilar el código.

C++:

Se crea a mediados de los 80, con el fin del paso de la programación tradicional (C) a estilos de abstracción de datos y orientación a objetos. Conserva características del C y añade nuevos conceptos como el de “Clase”, por lo que añade objetos a C.

Conserva la eficiencia, portabilidad, y disponibilidad de C.

Java:

Lenguaje de propósito general, orientado a Objetos. Su sintaxis inspirada en la de C/C++. Se trata de un lenguaje multiplataforma: Los programas Java se ejecutan sin variación (sin recompilar) en cualquier plataforma soportada (Windows, UNIX, Mac...). Es un lenguaje compilado e interpretado: El compilador produce un código intermedio independiente del sistema denominado `bytecode`. El intérprete a código máquina (dependiente de la plataforma) se llama Java Virtual Machine (JVM) y debe ser instalada en nuestro ordenador. Además es gratuito: Creado por SUN Microsystems, que distribuye gratuitamente el producto base, denominado JDK (Java Development Kit) o actualmente J2SE (Java 2 Standard Edition). La API distribuida con el J2SE es muy amplia y tiene su código fuente disponible. También nos da

bastantes posibilidades a la hora de manejar entornos gráficos, muy necesarios a priori para nuestro programa.

Lenguaje de desarrollo elegido

C se descartó por no ser orientado a objetos y no tener una “portabilidad total”.

Se ha decidido por Java sobre C++ debido a la portabilidad de Java, que no depende de la plataforma usada, pudiendo usar las clases ya compiladas tanto en UNIX como Windows. Fue decisivo también el alto nivel de la programación orientada a objetos propio de Java.

4.2 Elección del entorno de desarrollo

Una vez escogido JAVA, para la elección del entorno de programación se tuvieron en cuenta los siguientes:

Kawa:

Entorno de múltiples ventanas. Se parte de un proyecto, lleva dos módulos, uno con soporte HTML y otro para Java. Se pueden visualizar las clases, variables... Sencillo y potente.

Netbeans:

El Entorno de Desarrollo Integrado (IDE) NetBeans es un entorno de programación para varios lenguajes, incluyendo a Java y C++. Este desarrollo es de fuente abierto, es decir, se proporciona el código fuente del entorno para que se pueda modificar de acuerdo a ciertos parámetros de licencia.

Destacamos este entorno por su fácil manejo y por su muy buen depurador. Es muy útil para la programación de la interfaz gráfica. Nos ahorra tiempo en escribir código redundante ya que solo necesitamos pinchar los componentes que necesitamos y arrastrarlos con el ratón.

Un aspecto negativo de este entorno es la gran cantidad de recursos que consume.

JBuilder:

Diseñador visual para Enterprise JavaBeans 2.0 para desarrollo rápido de aplicaciones Java 2.

Distribución hacia los servidores de aplicación líderes, incluyendo Borland® Enterprise Server, BEA® WebLogic,® IBM® WebSphere,® e iPlanet Application Server.

Asistentes, herramientas y componentes para simplificar el desarrollo y la distribución de aplicaciones de bases de datos.

Desarrollo y distribución de aplicaciones Web con JSP y servlets.

Visualización de código UML.

Reconstructores de código y probadores de unidades.

Integración con sistemas líderes en el control de versiones.

Herramientas XML para publicación e integración de datos de negocios en dispositivos cruzados.

Entorno elegido

Finalmente se decidió escoger por el Netbeans para realizar la parte más sencilla de la interfaz gráfica debido a la facilidad con la que podemos implementarla con este programa. El Kawa se utilizó para el resto de la programación, debido a su facilidad de uso y su potencia. Para resolver problemas durante la programación se usó el depurador del Netbeans por las muchas posibilidades que ofrece, tales como la visualización de variables durante el proceso de depuración.

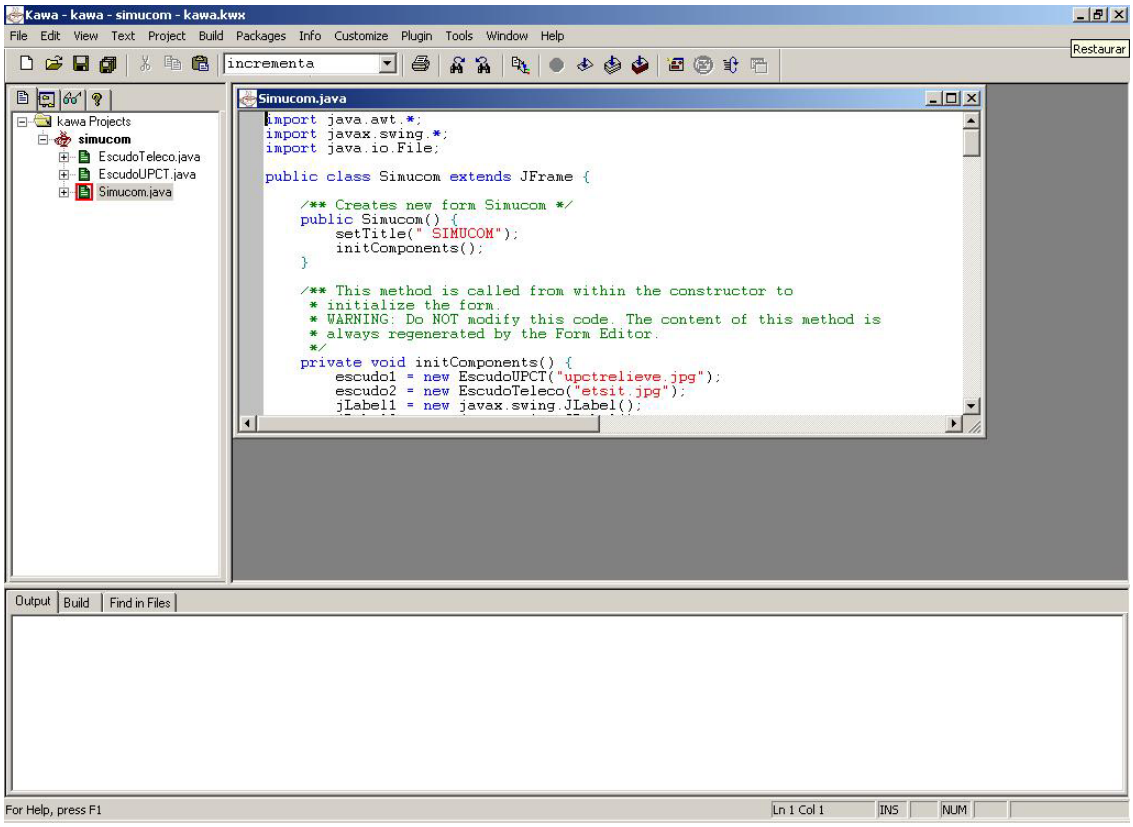

Figura 6: Kawa

Capítulo 5
Implementación de la aplicación

5.1 Descripción

Se desarrolla una herramienta de simulación de un conmutador espacial multietapa.

Empezaremos ofreciendo un menú de inicio para poder seleccionar una de las dos opciones que vamos a desarrollar.

Figura 7: Menú de inicio del SIMUCOM

La clase que implementa este menú de inicio se llama `InicioCE`. Se trata de una clase que extiende de JFrame y que contiene 2 botones. Cada uno de estos botones tiene asociado un método que ejecuta el comando que abre cada una de las 2 opciones de nuestro simulador. Por ejemplo, para el método del botón que ejecuta la opción A tendríamos:

```
Process p = r.exec ("java opcionA");
```

Las dos opciones son:

5.2 Opción A

Con 4 matrices crossbar 4X4 tanto en la entrada como en la salida, como en la etapa intermedia. Podremos ver el comportamiento de tres algoritmos de distribución mediante unas líneas que irán mostrando el camino elegido para realizar la conexión. Este camino también se refleja en una tabla que anota el número de conexión, la entrada, la salida y la matriz de distribución.

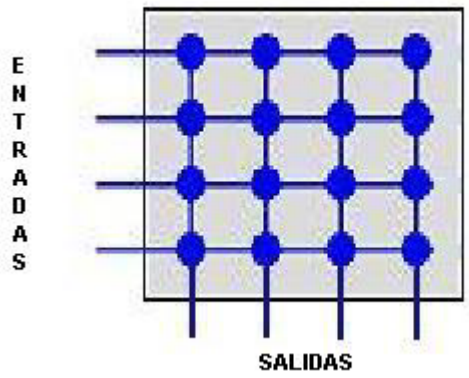

Figura 8: Matriz crossbar 4X4

Las matrices crossbar no se van a representar de la forma tradicional, esto es, con las líneas de entrada y salidas cruzadas formando una matriz de tantas filas como entradas y tantas columnas como salidas y con un punto en cada intersección de fila con columna. La forma que se representará será en forma rectangular con las entradas a un lado y las salidas en el lado opuesto. Esto se ha decidido así para facilitar la visualización del camino seguido por la conexión. De este modo se esconden los detalles internos de las matrices crossbar y aumenta la facilidad de seguimiento del encaminamiento de los algoritmos, alcanzando uno de los objetivos que se plantearon en un principio.

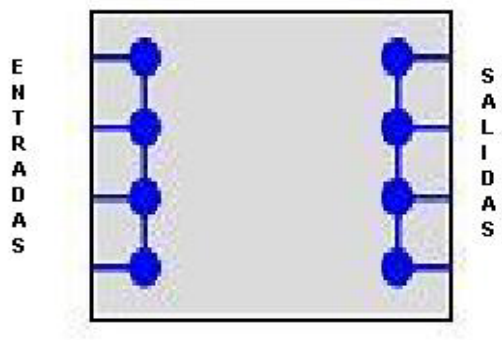

Figura 9: Matriz crossbar 4X4 usada en la implementación

Paneles:

Una serie de paneles muestran los siguientes datos estadísticos:

- El número de veces que se han realizado determinadas conexiones (de 4 a 16).
- El porcentaje en que se produjo cada una.
- La media total de conexiones hechas por simulación.

El programa contendrá una serie de botones, cajas de texto y pestañas que se enumeran y describen a continuación:

Botones:

- Paso adelante: Muestra, cada vez que se pulsa, el camino que ha seguido la conexión desde su entrada hasta la salida. Va rellenando en la tabla la conexión, entrada, salida y matriz de distribución correspondientes.
- Ejecución: Muestra todas las conexiones hechas en esa ejecución tanto en el dibujo como en la tabla.
- Nueva ejecución: Empieza una nueva ejecución, dispuesta para ser pintada y mostrada. Añade a las estadísticas la ejecución anterior.
- Reiniciar: Limpia las tablas y los dibujos.
- 1000 ejecuciones: Realiza seguidas 1000 ejecuciones, para añadir a las estadísticas (ver Caja de Texto "número de ejecuciones").
- Paso atrás: Es complementario a Paso adelante. Muestra las conexiones anteriores, borrando la actual.

Pestaña Algoritmo:

Sirve para escoger el algoritmo de encaminamiento (A, B o C).

Caja de Texto "número de ejecuciones":

Podemos modificar el número de ejecuciones seguidas a realizar mediante el botón de 1000 ejecuciones. La etiqueta del botón se ajusta al nuevo valor.

Comentaremos las distintas clases con sus métodos que se han programado para implementar este apartado. Después de la descripción de cada clase y sus métodos representaremos un diagrama UML de esa clase y diagramas de flujo de los métodos que consideremos necesario.

Entrada:

En su constructor hay que pasarle un entero que llamaremos `id`, que representa el número de entrada. Con este `id`, obteniendo su residuo de la división entre 4 (id % 4) obtenemos un valor que guardaremos en la variable entera "bloque", que representa el bloque de entrada.

Métodos:

- public int getId(): Devuelve `id`.
- public int getBloque(): Devuelve `bloque`.

Entrada
-id: int -bloque: int
+Entrada(int id) +getId(): int +getBloque: int

Figura 10: Diagrama UML de la clase Entrada de la opción A

Salida:

En su constructor hay que pasarle un entero que llamaremos `id`, que representa el número de salida. Con este `id`, obteniendo su residuo de la división entre 4 (`id % 4`) obtenemos un valor que guardaremos en la variable entera `bloque`, que representa el bloque de salida.

Métodos:

- `public int getId():`
  Devuelve id.
- `public int getBloque():`
  Devuelve bloque.

Salida
-id: int -bloque: int
+Salida(int id) +getId(): int +getBloque: int

Figura 11: Diagrama UML de la clase Salida de la opción A

Conexiones:

Hay que pasarle como parámetros un objeto del tipo entrada y otro del tipo salida. También lleva una variable entera: `MI` que sirve para indicar la matriz de distribución por la que pasa esa conexión. Esta clase es la representación de una conexión (entrada, salida y matriz intermedia).

Métodos:

- `public void setMI(int MI):`
  Asigna a la variable de clase `MI` el valor introducido en la función.

```
public void setMI(int MI)
{
    this.MI = MI;
}
```

- `public Entrada getEntrada():`
  Devuelve el objeto del tipo `Entrada: in`.

- `public Salida getSalida():`
  Devuelve el objeto del tipo `Salida: out`.

- `public int getMI():`
  Devuelve `MI`.

Conexiones
-in: Entrada -out: Salida -MI: int
+Conexiones(Entrada in, Salida out) +setMI(int MI) +getEntrada(): Entrada +getSalida(): Salida +getMI(): int

Figura 12: Diagrama UML de la clase Conexiones de la opción A

MatrizIntermedia:

Como parámetro en el constructor hay que pasarle un entero que llamamos `id`.

Las variables de clase son: el entero `id` que representa el número de matriz y los objetos de la clase `Vector`: `in`, `out` . Estos objetos contienen las matrices de entrada y salida que han conectado por la matriz de distribución a la que representa nuestra clase.

Métodos:

- `introduceBloqueIn:`

  Añade al vector de entradas `in` un bloque de entradas.

```
public void introduceBloqueIn(int bloque)
{
in.add(new Integer(bloque));
}
```

- `introduceBloqueOut:`

  Añade al vector de salidas `out` un bloque de salidas.

```
public void introduceBloqueOut(int bloque)
{
out.add(new Integer(bloque));
}
```

- `utilizaIn:`

  Indica si el valor introducido estaba ya almacenado en el vector de entradas devolviendo `true`. Devuelve `false` si se puede utilizar, no se utilizó una misma matriz de entrada 2 veces para nuestra matriz intermedia.

```
public boolean utilizaIn(int valor)
{
      for(int i=0;i<in.size();i++)
      {
      int e = ((Integer)in.elementAt(i)).intValue();
             if (valor ==
             ((Integer)in.elementAt(i)).intValue())
            {
            return true;
            }
      }
      return false;
}
```

- `utilizaOut`:

Indica si el valor introducido estaba ya almacenado en el vector de salidas devolviendo `true`. Devuelve `false` si se puede utilizar, no se utilizó una misma matriz de salida 2 veces para nuestra matriz intermedia.

```
public boolean utilizaOut(int valor
{
for(int i=0;i<out.size();i++)
{
int s = ((Integer)out.elementAt(i)).intValue();
       if (valor ==
       ((Integer)out.elementAt(i)).intValue())
       {
       return true;
       }
}
return false;
}
```

MatrizIntermedia
-in: Vector -out: Vector -id: int
+MatrizIntermedia (int id) +introduceBloqueIn(int bloque) +introduceBloqueOut(int bloque) +utilizaIn(int valor): boolean +utilizaOut(int valor): boolean

Figura 13: Diagrama UML de la clase MatrizIntermedia de la opción A

Algoritmo:

En el constructor de esta clase creamos un objeto del tipo Vector, que se llama conexiones y que contendrá objetos del tipo `Conexiones`. También se crean cuatro objetos del tipo `MatrizIntermedia`: a, b, c, d.

```
public Algoritmo()
{
conexiones = new Vector();
a = new MatrizIntermedia(0);
b = new MatrizIntermedia(1);
c = new MatrizIntermedia(2);
d = new MatrizIntermedia(3);
}
```

Métodos:

1. `public void generaConexiones():`

Este método rellena el vector conexiones con objetos del tipo `Conexiones` que constarán de entradas y salidas del 1 al 16, distribuidas al azar y procurando que no se repitan ninguna entrada y ninguna salida.

Para distribuir al azar las conexiones del vector llamamos al método `desordenar()`.

2. `private void desordenar():`

Este método coge un elemento del vector al azar, llamando a la función `Math.random()`, creando un número aleatorio entre 0 y 1 que multiplicándolo por el número de elementos del vector y redondeando indica un elemento del vector. Este elemento se añade al final del vector y se borra de la posición original. Esta operación se realiza 50 veces.

Por decirlo de alguna manera con este método barajamos el vector de conexiones.

3. `public int resuelveAlgoritmoA():`

Este método implementa el algoritmo A o de rotación.

Recorremos el vector conexiones buscando una matriz de entrada que coincida con alguna de las anteriores. En ese caso comprobamos que el bloque de entrada esté en la matriz intermedia que deberíamos escoger según nuestro algoritmo A. En caso afirmativo aumentamos una variable entera llamada `ok1` y hacemos que otra variable llamada `puntero` apunte a la siguiente matriz intermedia ya que nuestro algoritmo es el A o de rotación.

Para saber la matriz intermedia que estamos analizando haremos `puntero % 4`, con lo que la variable entera puntero se incrementa pasando de la matriz 3 a la 0 al aumentarla.

Con las salidas repetimos las mismas operaciones pero aumentando la variable `ok2`.

Si una de estas variables (`ok1` u `ok2`) son iguales a 4 o su la suma de ambos supera o iguala a 4, se habrá producido un bloqueo. Esto es así porque dada una matriz intermedia el algoritmo empieza comprobando si es posible

conectar la matriz de entrada a alguna matriz intermedia. Cada vez que no se consigue conexión se incrementa `ok1`. Una vez conseguida la conexión (suponiendo que se haya conseguido) se continua por la siguiente matriz intermedia a la utilizada con las matrices de salida, esta vez incrementando `ok2`.

Al final de cada una de las iteraciones que hemos hecho para comprobar que podemos conectar nuestra entrada y salida por alguna matriz intermedia (hemos comprobado que no hay bloqueo) añadimos la entrada y la salida a los vectores `in` y `out` de la Matriz de distribución escogida. En la conexión actual de nuestro vector de conexiones le asignamos a la variable "BloqueIntermedio" de esa conexión el valor de la matriz intermedia actual con el método `SetMI(puntero%4)` de la clase Conexiones. Incrementamos la variable puntero para que apunte a la siguiente matriz.

En caso de bloqueo llamamos al método `private void DepurarConexiones(int indiceMax)`.

4. `public int resuelveAlgoritmoB()`:

Este método implementa el algoritmo B o secuencial.

La descripción es similar a la del al método `resuelveAlgoritmoA()`, pero teniendo en cuenta que las matriz intermedia por la que vamos a intentar conectar es la 0, por lo que cada vez que hagamos una conexión(no ha habido bloqueo) pondremos el puntero a 0.

5. `public int resuelveAlgoritmoC()`:

Este método implementa el algoritmo C o aleatorio.

La descripción es similar a la del al método `resuelveAlgoritmoA()`, pero teniendo en cuenta que las matriz intermedia por la que vamos a intentar conectar es una al azar, por lo que cada vez que hagamos una conexión (no ha habido bloqueo) le daremos a puntero un valor al azar entre 1 y 16.

```
puntero = ((int) (Math.random() * 16)) + 1 ;
```

Simplificando el código en java se obtienen los 3 diagramas de flujo de los 3 últimos métodos, consiguiendo de esta manera mayor claridad.

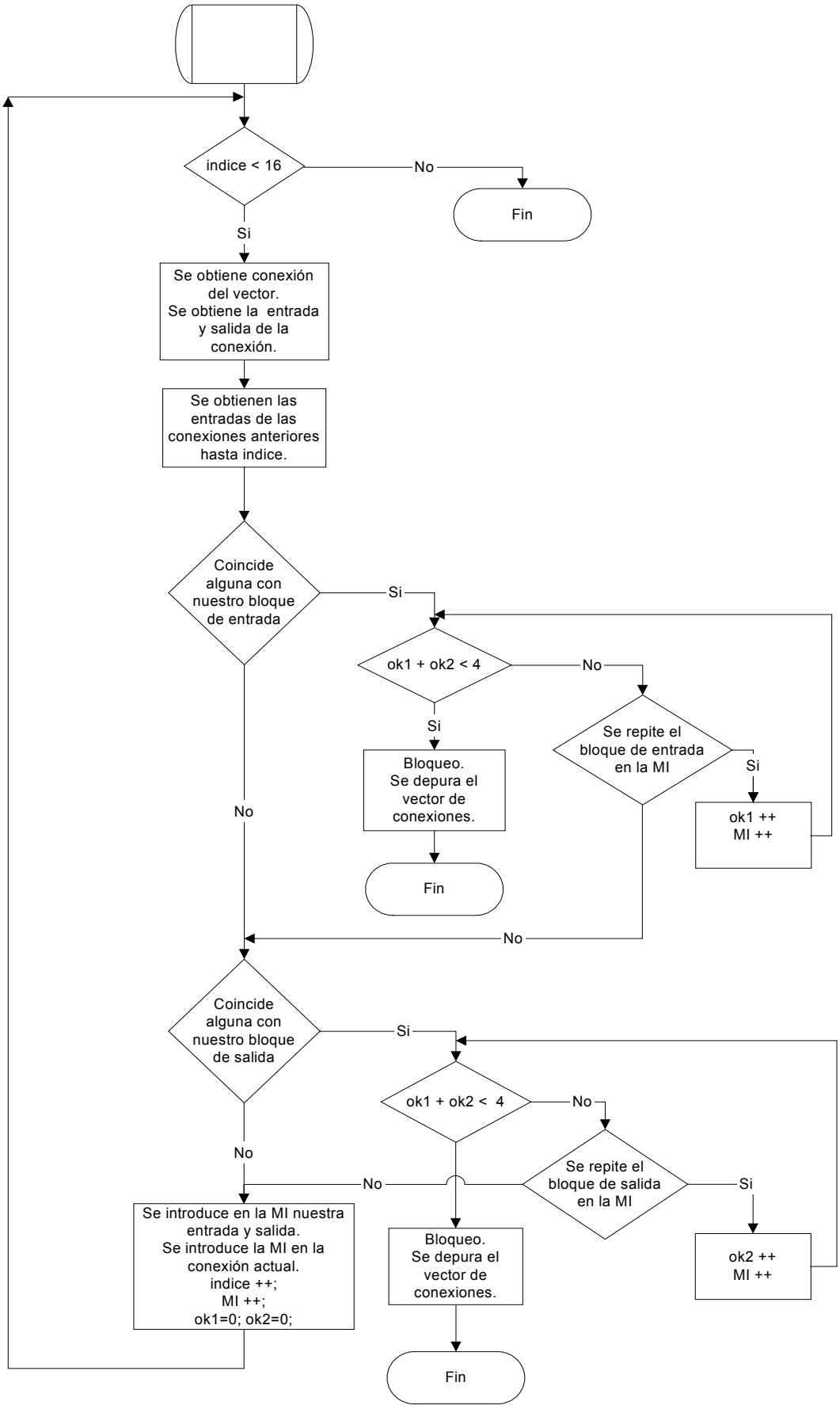

Figura 14: Diagrama de flujo del método `resuelveAlgoritmoA()`

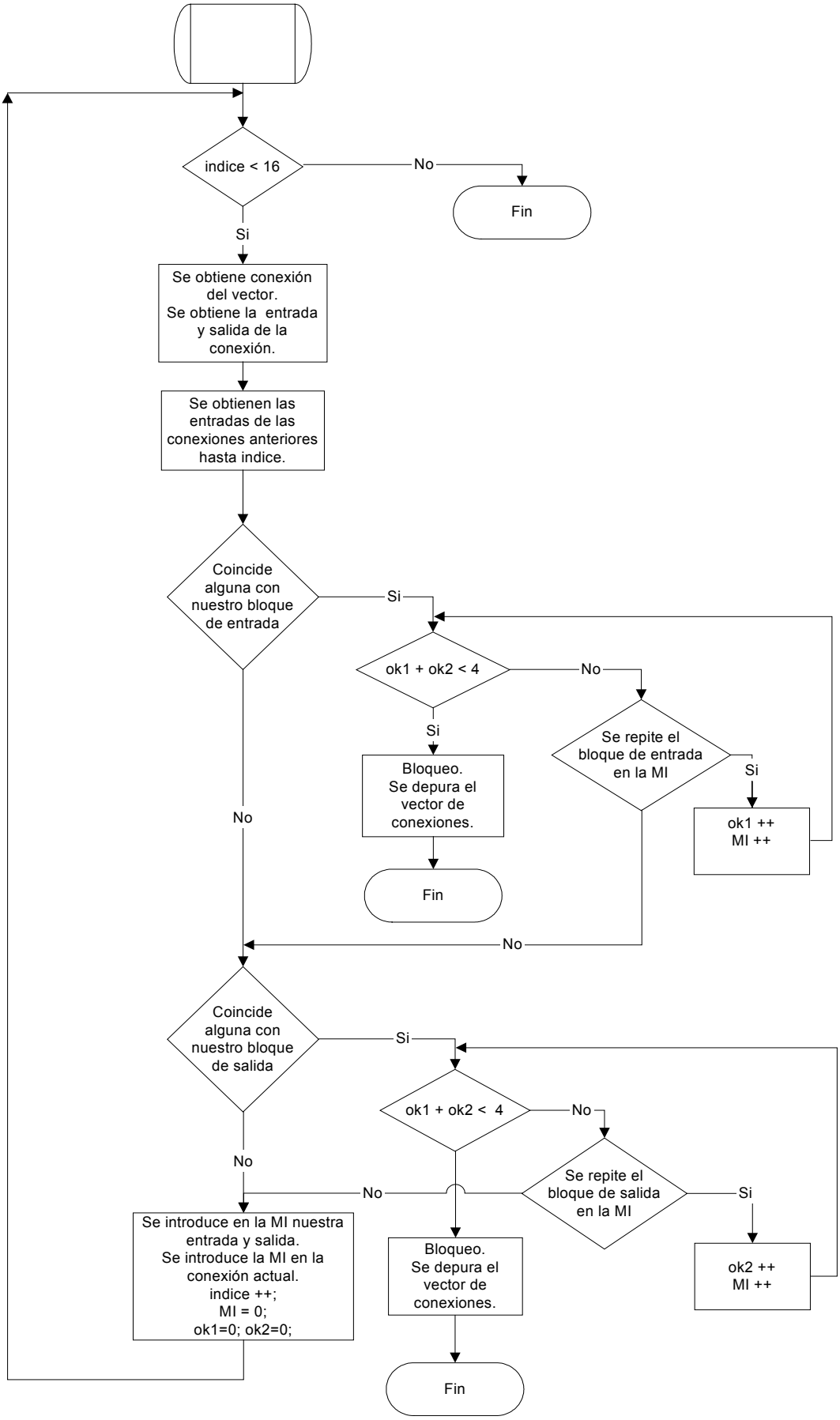

Figura 15: Diagrama de flujo de `resuelveAlgoritmoB()`

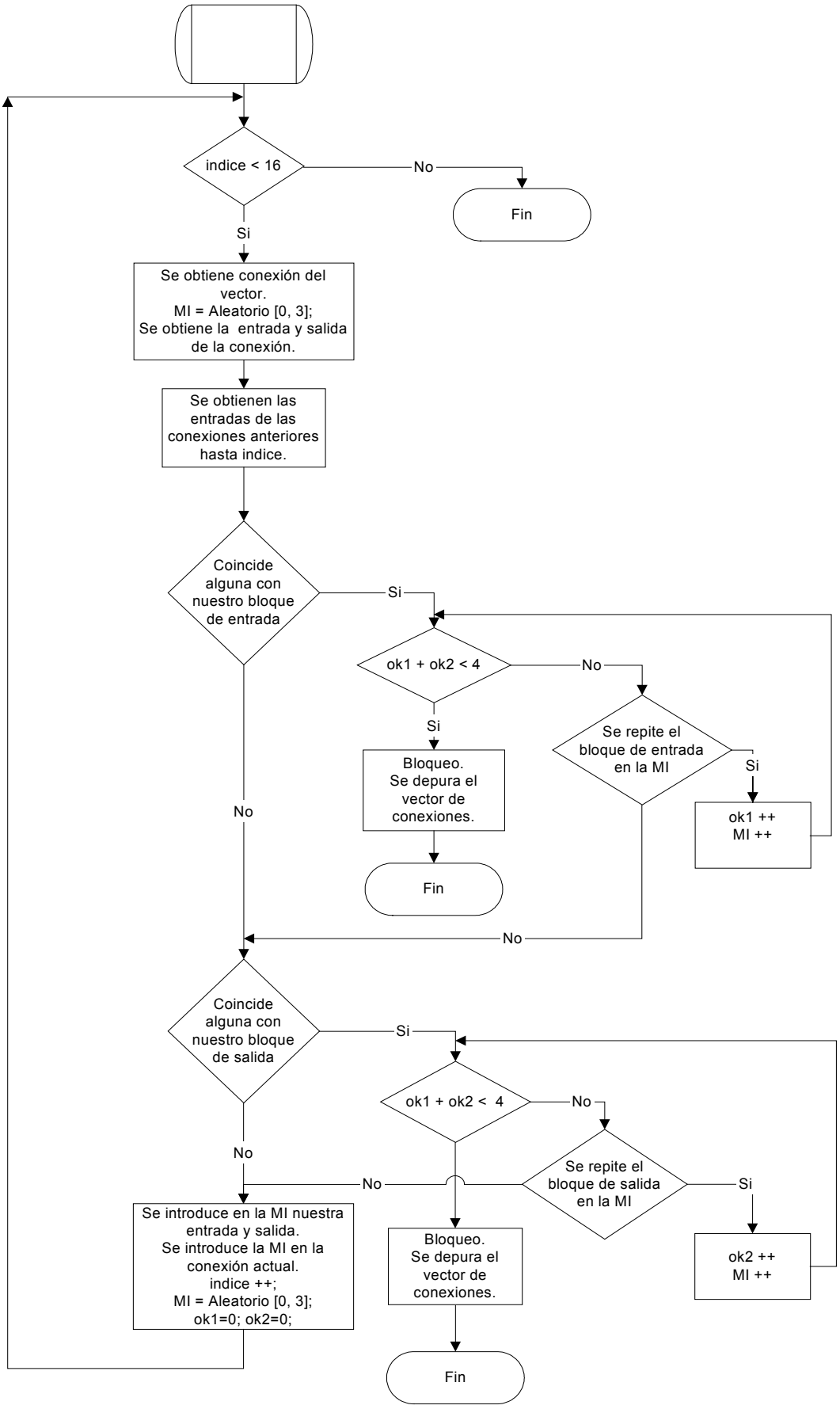

Figura 16: Diagrama de flujo de `resuelveAlgoritmoC()`

La simplificación principal en los diagramas esconde la necesidad de crear dos variables para saber si hemos analizado los bloques de entrada y los de salida: `BloqueinVisto`, `BloqueoutVisto`. En el caso de haber analizado los de salida y no los de entrada nos veremos en la necesidad de volver a analizar los de entrada después de los de salida.

6. `private void seleccionaMI(int valor, int entrada, int salida)`:

   Según la variable `valor`, añadiremos en un matriz intermedia u otra, los valores de entrada y salida a los vectores `in` y `out`.

Se llama a los métodos `introduceBloqueIn(entrada)` y `introduceBloqueOut(salida)` de la clase MatrizIntermedia.

7. `private boolean compruebaMI(int valor, int modulo)`:

   Según `valor` llamaremos en una matriz intermedia u otra a `utilizaIn(modulo)`.

8. `private boolean compruebaMO(int valor, int modulo)`:

   Según `valor` llamaremos en una matriz intermedia u otra a `utilizaOut(modulo)`.

9. `public Vector getConexiones()`:

   Devuelve el vector de conexiones.

10. `public void AlgoritmoEscogido(int Algoritm)`:

    Llamará según el valor de la variable `Algoritm` a los métodos:

    `resuelveAlgoritmoA(),resuelveAlgoritmoB(), resuelveAlgoritmoC()`.

11. `private void DepurarConexiones(int indiceMax)`:

    A este método lo llamamos en caso de que haya habido bloqueo y se encarga de borrar del vector conexiones las conexiones posteriores al bloqueo.

Algoritmo
+conexiones: Vector -a: MatrizIntermedia -b: MatrizIntermedia -c: MatrizIntermedia -d: MatrizIntermedia -BloqueinVisto: boolean -BloqueoutVisto: boolean
+Algoritmo() +generaConexiones() -desordenar() +resuelveAlgoritmoA(): int +resuelveAlgoritmoB(): int +resuelveAlgoritmoC(): int -seleccionaMI(int valor, int entrada, int salida) -compruebaMI(int valor, int modulo): boolean -compruebaMO(int valor, int modulo): boolean +getConexiones(): Vector +AlgoritmoEscogido(int Algoritm) -DepurarConexiones(int indiceMax)

Figura 17: Diagrama UML de la clase Algoritmo de la opción A

Imagen

Esta clase extiende de `JPanel`, por lo que implementa un panel sobre el que pondremos un dibujo proveniente de un fichero. El dibujo representa al conmutador espacial de 3 etapas, con sus 4 matrices crossbar en la entrada, salida y etapa intermedia. Sobre este `JPanel` se irá dibujando y repintando las líneas que irán recorriendo las matrices desde su entrada hasta su salida, pasando por la etapa central o de distribución.

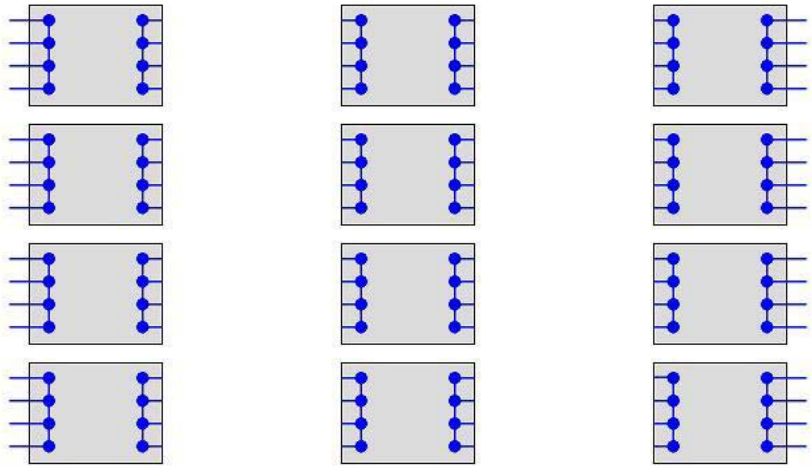

Figura 18: Imagen que contiene el fichero sincrossbar.jpg

Métodos:

- `public static BufferedImage getBufferedImage (String fichero,Component c ):`

Método para crear un objeto del tipo `Image` a partir de un fichero, que se guarda en un buffer.

- `public static boolean cargaImagen (Image imagen, Component c ):`

Este método toma un objeto `Image` asociado a un fichero y espera hasta que la carga se haya completado.

Nota: Este es el uso más simple de `MediaTracker`, si se van a cargar varias imágenes, no debe usarse este método en sucesivas llamadas, sino que es mucho mejor el uso de un `array` de imágenes.

- `public void dibuja(Graphics2D g2):`

Este método se encarga de dibujar las líneas que se van a ir pintando en la ejecución del programa. Se vale del método `PintaLinea(g2)`.

- `public void PintaLinea(Graphics2D g2):`

Según el valor de la entrada, la salida, el bloque de entrada, el de salida y la matriz de distribución pinta en azul las líneas que forman el camino tomado por las conexiones a pintar.

- `public void NumerosFondo(Graphics2D g2):`

Pinta los números del 1 al 16 que representan las entradas y salidas. Estos números se quedan permanentemente en el panel y sobre el dibujo.

También escribe sobre cada etapa de distribución su nombre.

- `public void LineasFondo(Graphics2D g2):`

Dibuja permanentemente sobre la imagen todas las líneas que unen a las etapas de distribución en un color gris claro.

- `public void setConexiones(Vector vectorconexiones):`

Se le asigna a la variable `con` de nuestra clase la variable `vectorconexiones.`

- `public void setContador(int cont):`

Se le asigna a la variable `contador` de nuestra clase, la variable `cont.`

- `public void paintComponent( Graphics g ):`

Éste método sirve para pintar los objetos que añadimos al dibujo, tales como el recuadro que lo enmarca, los números de fondo, las líneas de fondo, y el nombre de las etapas de distribución.

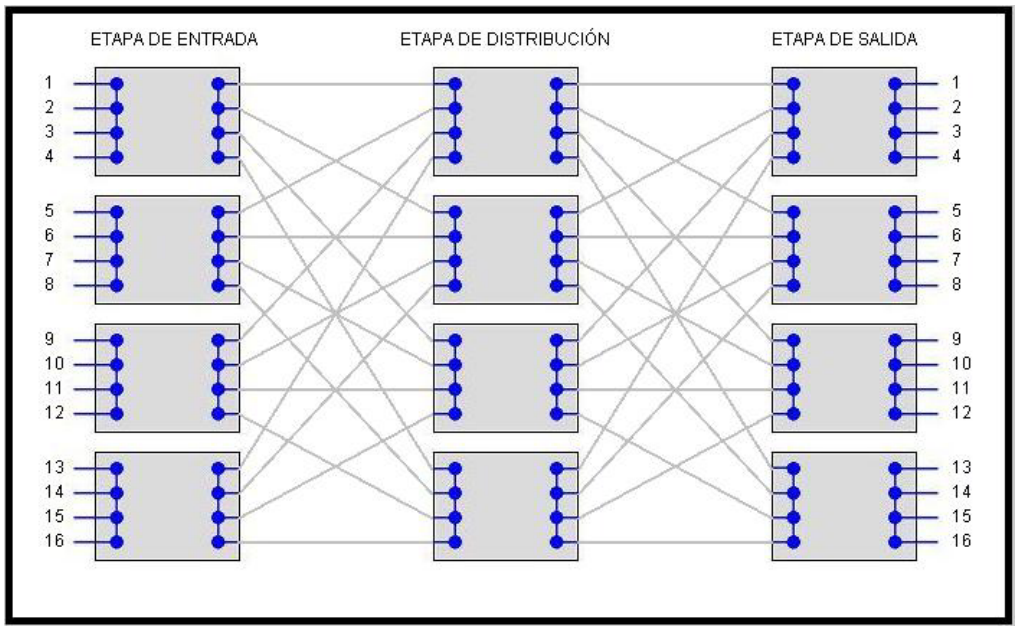

Figura 19: Dibujo proveniente del archivo modificado al pasar por el método `paintComponent( Graphics g )`

Imagen
-imagen1: TexturePaint -rectangulo: Rectangle -con: Vector -contador: int -ruta: String
+Imagen() static getBufferedImage(String fichero,Component c): BufferedImage static cargaImagen(Image imagen,Component c): boolean +paintComponent(Graphics g) +dibuja(Graphics2D g2) -NumerosFondo(Graphics2D g2) -LineasFondo(Graphics2D g2) +PintaLinea(Graphics2D g2) +setConexiones(Vector vectorconexiones) +setContador(int cont)

Figura 20: Diagrama UML de la clase Imagen de la opción A

OpcionA:

Clase principal de nuestro programa que extiende de `JDialog`. En ella se encuentra el método `main`.

En esta clase vamos a definir e inicializar todos los objetos del entorno gráfico tales como botones, cajas de texto, pestañas, etiquetas, tablas y un objeto de la clase `Imagen`.

Muchos de estos objetos estarán escuchando eventos. Vamos a citar los métodos que estos objetos proporcionan para que se ejecuten cuando se recibe el evento:

- `jMenuItem1ActionPerformed`:

  Se ejecuta al seleccionar la opción A del menú Algoritmo. Llama al método `Reiniciar()` que pone a 0 todas las variables, limpia la tabla, los paneles de estadísticas y la imagen. Finalmente deja seleccionado el algoritmo A como algoritmo de encaminamiento.

- `jMenuItem2ActionPerformed`:

  Se ejecuta al seleccionar la opción B del menú Algoritmo. Llama al método `Reiniciar()` que pone a 0 todas las variables, limpia la tabla, los paneles de estadísticas y la imagen. Finalmente deja seleccionado el algoritmo B como algoritmo de encaminamiento.

- `jMenuItem3ActionPerformed`:

Se ejecuta al selecionar la opción C del menú Algoritmo. Llama al método `Reiniciar()` que pone a 0 todas las variables, limpia la tabla, los paneles de estadísticas y la imagen. Finalmente deja seleccionado el algoritmo C como algoritmo de encaminamiento.

- `jButton1ActionPerformed`:

Se ejecuta al pinchar en el botón “Paso Adelante”. Según pinchamos en el botón se va incrementando un contador que tendrá un tope en el tamaño del vector de conexiones. Este contador hace que se pinten en el dibujo el numero de conexiones que indica. Se va escribiendo también en la tabla las conexiones realizadas.

- `jButton2ActionPerformed`:

Se ejecuta al pinchar en el botón “Ejecución”. Pinta en el dibujo y escribe en la tabla todas las conexiones hechas en esa ejecución.

- `jButton3ActionPerformed`:

Se ejecuta al pinchar en el botón “Nueva Ejecución”. Añade a las etiquetas de estadísticas el numero de conexiones realizadas en la ejecución en curso. Limpia la tabla de conexiones y la imagen. Por último deja resuelta una nueva ejecución.

- `jButton4ActionPerformed`:

Se ejecuta al pinchar en el botón “Reiniciar”. Llama al método `Reiniciar()`, por tanto la única función de este botón es limpiar la imagen, la tabla de conexiones, devolver a las variables a su estado por defecto.

- `jButton5ActionPerformed`:

Se ejecuta al pinchar en el botón “1000 Ejecuciones”. Este método realiza tantas ejecuciones como se haya indicado (1000 por defecto). Estas ejecuciones se verán reflejadas en las etiquetas de estadísticas. Limpia la tabla de conexiones y la imagen.

- `jButton6ActionPerformed`:

Se ejecuta al pinchar en el botón “Paso Atrás”. Conforme pinchamos en este botón se irá decrementando la variable `Contador1` que se incrementaba al pulsar el botón “Paso Adelante”. El mínimo valor que podrá tomar será 0. Por lo tanto, con este botón se repinta la imagen con el número nuevo de conexiones. También va borrando la última fila escrita de la tabla de conexiones.

- `jTextField1ActionPerformed`:

Se ejecuta al introducir caracteres en el campo de texto JTextField1.

Al introducir un número entero menor de 10.000 , puede determinar el número de ejecuciones que se podrán hacer al pulsar el botón de "1000 Ejecuciones". El texto del botón cambiará según el valor que se introduzca. Si se introduce un número mayor o igual a 10.000 o caracteres que no sean números se obtiene un mensaje de aviso en forma de un `JDialog`.

- `jPanelMousePressed`:

La imagen está escuchando hasta que se produce un clic de ratón en ella. Cuando esto sucede se repinta la imagen con todas las líneas que se habían dibujado en ella.

- `windowActivate`:

El `JDialog opcionA` está escuchando hasta que se activa la ventana, esto es que pasa de un segundo plano a un primero. Cuando este evento pasa se repinta la imagen con todas las líneas que se habían dibujado en ella.

- `windowDeactivate`:

El `JDialog opcionA` está escuchando hasta que se desactiva la ventana, esto es que pasa de un primer plano a un segundo. Cuando este evento pasa se repinta la imagen con todas las líneas que se habían dibujado en ella.

- `exitForm(java.awt.event.WindowEvent evt)`:

Cierra la ventana.

Otros métodos de esta clase:

- `pantallaCompleta(JDialog jf)`:

Con este método se logra que el `JDialog opcionA` se sitúe ocupando el tamaño de la pantalla. La configuración de la pantalla debe estar a 1024 por 768 píxeles para que la visualización de la `opcionA` sea completa.

- `private void setConexiones(Vector vc)`:

Pasa el vector conexiones al objeto de la clase `Imagen`.

- `private void rellenaTabla()`:

Método que rellena toda la tabla de conexiones con el contenido del vector de conexiones.

- `private void limpiaTabla()`:

Se encarga de borrar la tabla de conexiones.

- `private void limpiaEstadisticas()`:

  Borra el texto de las etiquetas que se encargan de reflejar las estadísticas.

- `private void CalculaMedia(float ejecuciones)`:

  Calcula la media de conexiones de las ejecuciones totales que se han realizado. Usa para ello el array `estadisticas`. Escribe el resultado en la etiqueta correspondiente a la media de conexiones (`JLabelMedia`).

- `private void escogeAlgoritmo()`:

  Método para decirle al objeto de la clase `Algoritmo` el algoritmo que se ha elegido.

Para la clase opcionA no se ha representado su diagrama UML por no considerarlo apropiado, debido a las numerosas variables de clase como botones, etiquetas, etc. que contiene y que no son determinantes a la hora de la comprensión del programa.

Dialogo:

Clase que extiende de `JDialog` creada para que se muestre cada vez que haya que avisar al usuario del programa de cualquier eventualidad, por ejemplo de que introduzca datos correctos en las cajas de texto.

Métodos:

- `public void CentraDialogo()`:

  Sirve para situar y dar tamaño al Dialogo.

- `public void mensaje(String ms)`:

  Introduce el mensaje que para cada caso mas conveniente en una etiqueta dentro de `Dialogo`.

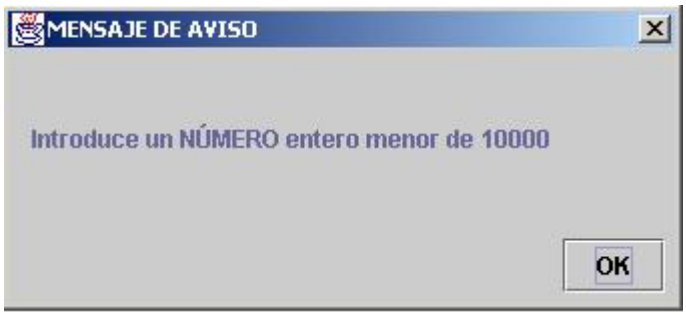

Figura 21: Ejemplo de Dialogo

Mostramos ahora la opción A ejecutada, pero sin realizar operación alguna:

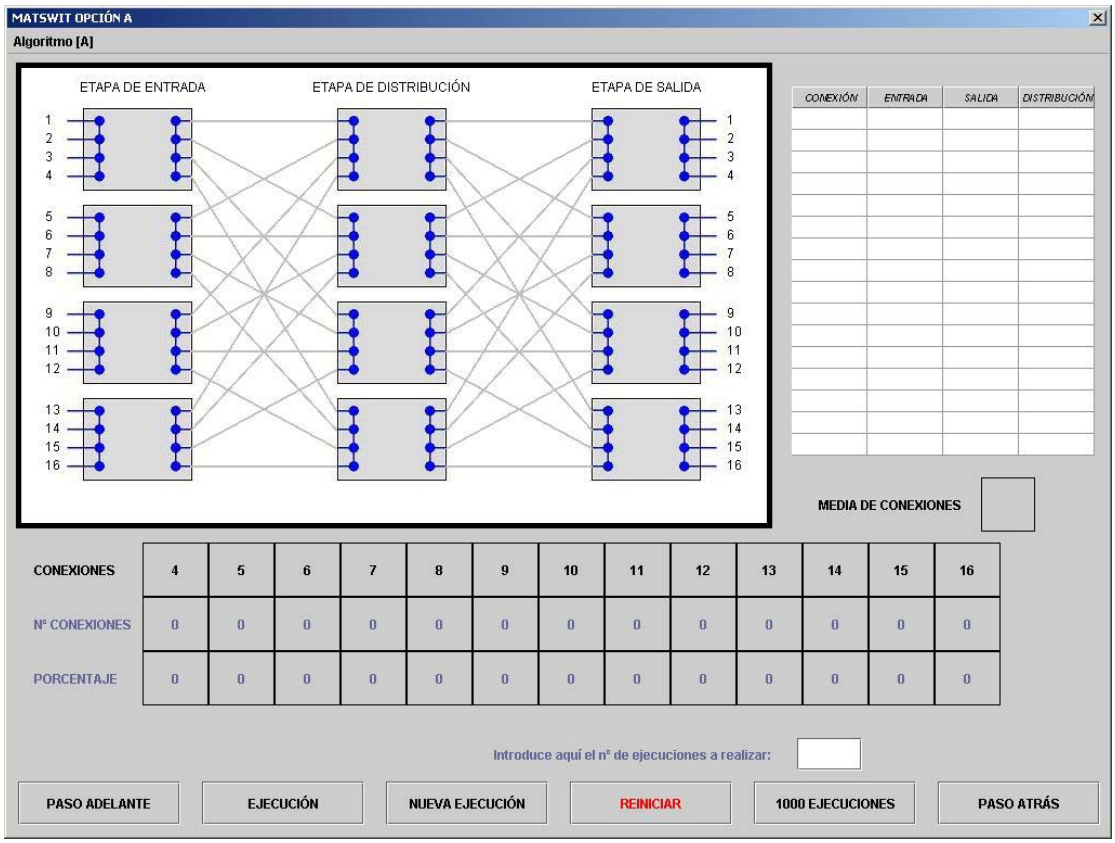

Figura 22: Opción A ejecutada

5.3 Opción B

En esta opción se simula un conmutador espacial de 3 etapas en régimen continuo. Las llamadas serán de duración finita. Se tienen por tanto situaciones de conexión, desconexión y de bloqueo. Una situación de bloqueo no hace que la ejecución del programa se pare, solamente que se pierda esa conexión.

Las matrices intermedias pueden variar de 4 hasta 7. Por lo que las matrices de crossbar de entrada y salida se verán modificadas para cada caso.

Las tres etapas consisten en cuatro matrices crossbar 4X(4, 5, 6, 7) en la entrada y cuatro matrices crossbar (4, 5, 6, 7)X4 en la salida. En la etapa intermedia podrá contener de cuatro a siete matrices crossbar de 4X4.

Las llamadas se irán conectando y desconectando de forma aleatoria.

La probabilidad de que ocurra uno u otro evento irá en base a un factor de actividad a introducir.

Se muestra un diagrama de conexiones consistente en 16 filas y 16 columnas representando a las entradas y salidas respectivamente, en las casillas del diagrama representamos la matriz de distribución de la conexión con su número correspondiente en color negro si hay conexión, en rojo si se desconecta y con una B si hay bloqueo.

Este diagrama representa las conexiones que hay en curso en ese momento.

Por lo tanto cada conexión queda representada mirando el número de la fila y columna, que estarán pintados en rojo en ese instante, para saber la entrada y salida. Dentro del diagrama miraremos para saber el número de matriz de distribución (la primera es la 0, la segunda la 1 y así...) o una B en el caso de que la conexión no se haya podido llevar a cabo.

Dispondremos de las siguientes pestañas:

- Algoritmo: Para introducir el algoritmo de encaminamiento: A, B o C.
- Matrices Intermedias: Para asignar al conmutador el número de matrices de distribución. De cuatro a siete.

Se pueden usar los siguientes botones:

- Paso Adelante: Realiza un evento de conexión, desconexión o bloqueo, según se determine.
- Ejecución: Realiza eventos hasta que haya un bloqueo. En caso de que no lo haya en un número máximo de ejecuciones , se parará al llegar a ese número indicándolo mediante un mensaje de aviso.
- 1000 ejecuciones: Producirá tantos eventos como le hayamos indicado, por defecto 1000.
- Reiniciar: Reinicia todos los valores (los pone al valor que tenían a iniciar el programa). También limpia el diagrama de conexiones.

Se dispone de dos cajas de texto para introducir valores:

- El factor de actividad. Valor comprendido entre 0.1 y 15.9 que va a determinar la probabilidad de que se intente conectar o desconectar.
- Número de ejecuciones a realizar. Determina el máximo número de eventos que se realizarán al pulsar los botones de “ejecución” o “1000 ejecuciones”. El valor introducido se verá reflejado en este último botón. El límite son 10.000 ejecuciones.

En la opción B se incorporan dos novedades importantes respecto a la opción A, que son la posibilidad de aumentar las matrices intermedias hasta siete (nótese que la representación matricial del conmutador es muy adecuada para esta característica) y

la contemplación de que la llamada se desconectará trascurrido un tiempo (régimen continuo). Uno de los datos que el programa nos pide es lo que el Matswit llama Tráfico Ofrecido (más tarde se verá que este no es tal). Debido a esto se obtienen nuevos valores tales como Tráfico Cursado, Tráfico Perdido, conexiones en curso, conexiones perdidas y máximas conexiones que ha habido en curso.

Por el uso de la representación matricial del conmutador la representación de las conexiones es simbólica en lugar de gráfica.

Para programar este apartado nos hemos valido de las siguientes clases:

Entrada:

En su constructor hay que pasarle un entero que se llama `id`, que representa el número de entrada. Con este `id`, obteniendo su residuo de la división entre 4 (`id % 4`) se tiene un valor que se guarda en la variable entera `bloque`, que representa el bloque de entrada.

Métodos:

- `public int getId():`

  Devuelve `id`.

- `public int getBloque():`

  Devuelve `bloque`.

Entrada
-id: int -bloque: int
+Entrada(int id) +getId(): int +getBloque: int

Figura 23: Diagrama UML de la clase Entrada de la opción B

Salida:

En su constructor hay que pasarle un entero que se llamará `id`, que representa el número de salida. Con este `id`, obteniendo su residuo de la división entre 4 (id % 4) se tiene un valor que guardaremos en la variable entera `bloque`, que representa el bloque de salida.

Métodos:

- `public int getId():`

  Devuelve `id`.

- `public int getBloque():`

Devuelve `bloque`.

Salida
-id: int -bloque: int
+Salida(int id) +getId(): int +getBloque: int

Figura 24: Diagrama UML de la clase Salida de la opción B

Conexiones:

Variables de clase: `in` del tipo `Entrada`, `out` del tipo `Salida`, `conectado` y `bloqueado` del tipo `boolean` y `MI` del tipo entero.

Esta clase es la representación de una conexión, con su entrada, su salida, su matriz intermedia (`MI`) y variables que indican si la conexión esta conectada o bloqueada.

Definimos dos constructores:

En el primero hay que pasarle como parámetros: Un objeto del tipo `Entrada` y otro del tipo `Salida`.

En el segundo constructor los parámetros a introducir son:

- Un objeto del tipo conexiones
- Variable del tipo `boolean` que llamamos `conectado` indican si la conexión está conectada.
- Variable del tipo `boolean` que llamamos `bloqueado` indican si la conexión está bloqueada.
- Una variable entera llamada `MI` que indica la Matriz intermedia por la que se realizó la conexión.

```
    public Conexiones (Conexiones c, boolean conectado, boolean
bloqueado, int MI)
     {
     this.in = c.getEntrada();
     this.out = c.getSalida();
     this.conectado = conectado;
     this.bloqueado = bloqueado;
     this.MI = MI;
```

```
}
```

Este segundo constructor tiene como objeto el crear una nueva conexión que tenga la entrada y salida de la conexión que le pasamos como parámetro. De esta manera esta nueva conexión no apuntará a la antigua (podremos modificarla sin modificar la antigua) y se le especificarán los valores de las variables `conectado`, `bloqueado` y `MI`. Estas variables se les asignarán a las variables de clase con el mismo nombre.

Esta clase es más complicada que su correspondiente a la opción A. Se ha reutilizado añadiendo nuevas variables, constructores y métodos para realizar la clase conexión de la opción B.

Ya hemos visto el nuevo constructor, estos son los métodos:

- `public void setMI(int MI):`

  Asigna a la variable de clase `MI` el valor introducido en la función.

```
public void setMI(int BloqueIntermedio)
{
this.MI = MI;
}
```

- `public Entrada getEntrada():`

  Devuelve el objeto del tipo `Entrada in`.

- `public Salida getSalida():`

  Devuelve el objeto del tipo `Salida out`.

- `public int getMI():`

  Devuelve `MI`.

- `public void Conectar():`

  Pone la variable `conectado` a `true`.

- `public void Desconectar`():

  Pone la variable conectado a `false`.

- `public boolean getConectado():`

  Obtenemos la variable `conectado`.

- `public void Bloquear():`

  Pone la variable `bloqueado` a `true`.

- `public void Desbloquear():`

  Pone la variable `bloqueado` a `false`.

- `public boolean getBloqueado():`

  Obtenemos la variable `bloqueado`.

Conexiones
-in: Entrada -out: Salida -conectado: boolean -bloqueado: boolean -MI: int
+Conexiones(Entrada in, Salida out) + Conexiones(Conexiones c, boolean conectado, boolean bloqueado, int MI) +setMI(int MI) +getEntrada(): Entrada +getSalida(): Salida +getMI(): int +Conectar() +Desconectar() +getConectado(): boolean +Bloquear() +Desbloquear() + getBloqueado: boolean

Figura 25: Diagrama UML de la clase Conexiones de la opción B

MatrizIntermedia:

Como parámetro en el constructor hay que pasarle un entero que se llama `id`.

Las variables de clase son: el entero `id` que representa el número de matriz y los objetos de la clase `Vector`: `in`, `out`. Estos objetos contienen las matrices de

entrada y salida que han conectado por la matriz de distribución a la que representa nuestra clase.

A la hora programar esta clase se ha reutilizado la clase del mismo nombre de la opción A, aunque debemos añadir nuevos métodos.

Antiguos métodos:

- `introduceBloqueIn:`

  Añade al vector de entradas `in` un bloque de entradas.

```
public void introduceBloqueIn(int bloque)
{
in.add(new Integer(bloque));
}
```

- `introduceBloqueOut:`

  Añade al vector de salidas `out` un bloque de salidas.

```
public void introduceBloqueOut(int bloque)
{
out.add(new Integer(bloque));
}
```

- `utilizaIn:`

  Indica si el valor que le pasamos como parámetro estaba ya almacenado en el vector de entradas de nuestra matriz intermedia, devolviendo true en caso afirmativo. Devuelve false en el caso de que se pueda utilizar dicha matriz intermedia, esto significa que, no se utilizó una misma matriz de entrada 2 veces para nuestra matriz intermedia.

```
public boolean utilizaIn(int valor)
{
for(int i=0;i<in.size();i++)
{
int e = ((Integer)in.elementAt(i)).intValue();
      if (valor == ((Integer)in.elementAt(i)).intValue())
      {
      return true;
      }
```

```
}
 return false;
}
```

- `utilizaOut:`

Indica si el valor que le pasamos como parámetro estaba ya almacenado en el vector de salidas de nuestra matriz intermedia, devolviendo true en caso afirmativo. Devuelve false en el caso de que se pueda utilizar dicha matriz intermedia, esto significa que, no se utilizó una misma matriz de entrada 2 veces para nuestra matriz intermedia.

```
public boolean utilizaOut(int valor
{
for(int i=0;i<out.size();i++)
{
int s = ((Integer)out.elementAt(i)).intValue();
      if (valor == ((Integer)out.elementAt(i)).intValue())
      {
      return true;
      }
}
return false;
 }
```

Nuevos métodos:

- `public void borraBloqueIn(int ent):`

Borra el bloque de entrada del vector de entradas de la matriz intermedia. Esto se puede hacer debido a que en este vector de entradas, en teoría y debido a al clase `Algoritmo2` no va a haber mas de un tipo de bloque de entrada. Por eso al borrarlo del vector se está seguro de que es ese bloque el que se quiere borrar.

Este método se debe a que el programa se va a contemplar la opción de desconexión.

- `public void borraBloqueOut(int sal):`

Borra el bloque de salida del vector de salidas de la matriz intermedia. Esto se puede hacer debido a que en este vector de salidas, en teoría y debido a al clase `Algoritmo2` no va a haber mas de un tipo de bloque de salida. Por eso

al borrarlo del vector se está seguro de que es ese bloque el que se quiere borrar.

Este método se debe a que en el programa se va a contemplar la opción de desconexión.

- `public int getInOutSize():`

  Devuelve el tamaño de los vectores de entradas y salidas, tamaños que se comprueban, son los mismos.

MatrizIntermedia
-in: Vector -out: Vector -id: int
+MatrizIntermedia (int id) +introduceBloqueIn(int bloque) +introduceBloqueOut(int bloque) +borraBloqueIn(int ent) +borraBloqueOut(int sal) +utilizaIn(int valor): boolean +utilizaOut(int valor): boolean

Figura 26: Diagrama UML de la clase MatrizIntermedia de la opción B

Algoritmo2:

En el constructor se observa que a los objetos de esta clase se les pasará como parámetro un número entero. Este número entero simboliza el número de matrices intermedias que queremos que lleve nuestro objeto. Este número de matrices intermedias irá de cuatro a siete.

También se crean dos objetos de la clase predefinida `Vector` que se llamarán `conexiones` y `eventos`. En el vector conexiones se guardan los objetos de tipo C`onexiones` con los que trabajará nuestro programa. El vector eventos llevará objetos del tipo conexiones que hayan sufrido una conexión, una desconexión o un bloqueo. Durante la ejecución del programa para luego ir mostrando este vector en el diagrama de conexiones.

Métodos:

1- `public void generaConexion():`

Se va añadiendo al vector de conexiones nuevas conexiones con entrada secuencial (1, 2, 3, ..., hasta 16) y salida aleatoria (de 1 a 16 sin que se repita ninguna salida).

2- `private void desordenar():`

Este método coge un elemento del vector al azar, llamando a la función `Math.random()`, creando un número aleatorio entre 0 y 1 que multiplicándolo por el número de elementos del vector y redondeando indica un elemento del vector. Este elemento se añade al final del vector y se borra de la posición original. Esta operación se realiza 50 veces.

Por decirlo de alguna manera con este método barajamos el vector de conexiones.

3- `private void SetSalida():`

En este método se buscan en el vector `conexiones` las que tengan la variable `conectado` a `false`. De estas conexiones se extraen las entradas y salidas y se añaden por separado a dos nuevos vectores (`entradas` y `salidas`). Se desordenan estos vectores. Se cambian las conexiones con la variable `conectado` a `false` por otras nuevas creadas a partir de los 2 vectores desordenados.

4- `private void desordenarEnSal():`

Desordena los vectores `entradas` y `salidas` con el fin de que luego tengamos nuevas conexiones que intentar conectar.

5- `public Vector getVector():`

Devuelve el vector `conexiones`.

6- `public int resuelveAlgoritmos():`

Se definen dos variables que van a representar a las probabilidades de conexión y a la de desconexión. Estas probabilidades se determinan mediante la variable `TO` que representa al "factor de actividad" del programa y mediante la suposición de que la llamada dura por media dos minutos.

Se recorre el vector conexiones buscando alguna que no esté conectada. Cuando la encuentre, mediante la probabilidad de conexión, se decidirá si se debe intentar la conexión. Si es así de llamará al método `AlgoritmoEscogido` que intentará conectar mediante el algoritmo que se esté usando.

Si la conexión está conectada se mira la probabilidad de desconexión y la compramos con un número aleatorio para desconectarla.

Cuando se desconecte o bloquee una conexión (en el caso de no poder conectar) se renovarán las conexiones que no estén conectadas.

Cuando se llegue al final del vector conexiones (tamaño 16) se volverá a analizarlo desde el principio y se incrementará la variable `tiempo` que simboliza las ranuras. Todo esto se realiza hasta que se obtenga una conexión, desconexión o bloqueo.

El siguiente diagrama de flujo clarifica el método `resuelveAlgoritmos():`

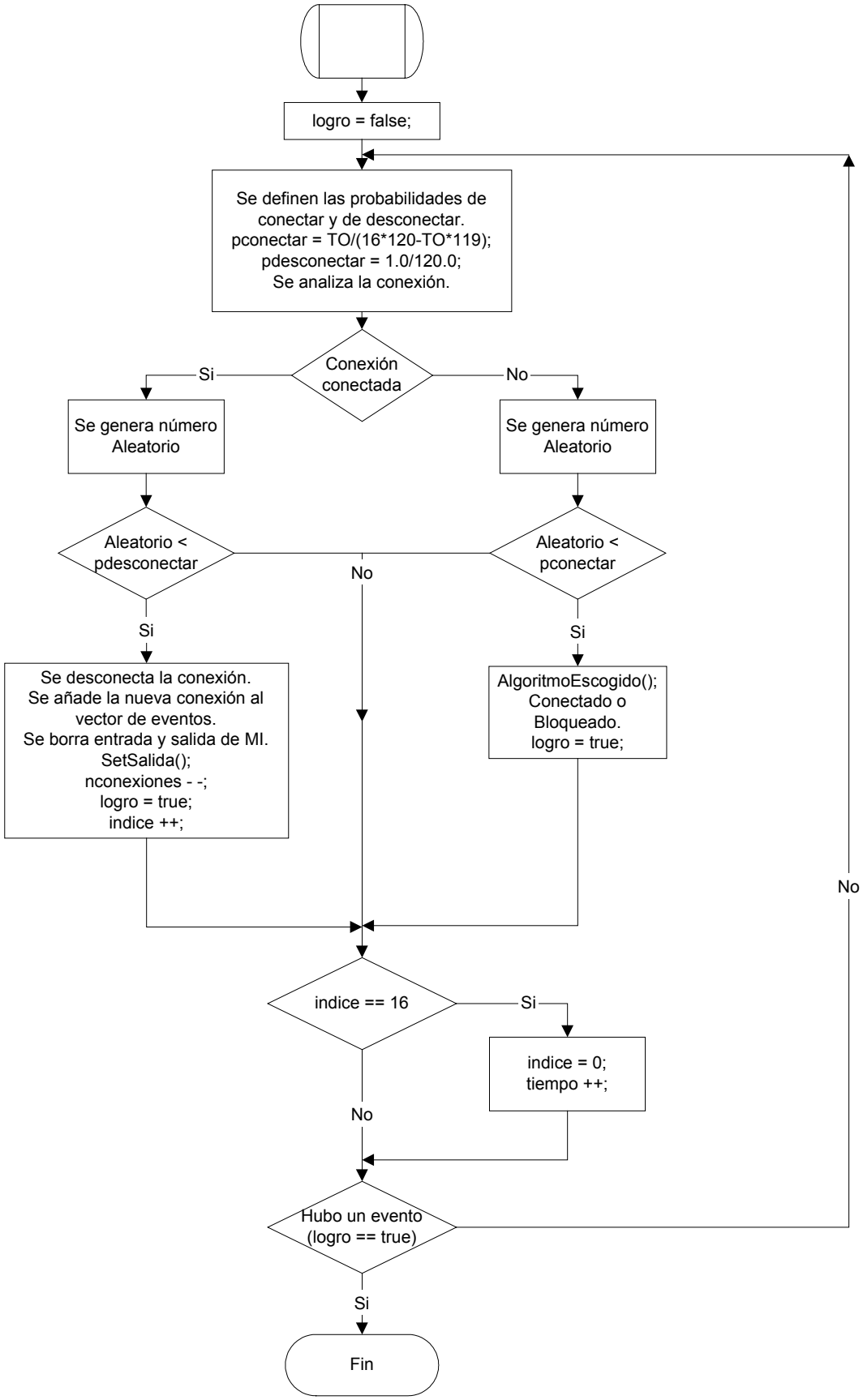

Figura 27: Diagrama de flujo del método `resuelveAlgoritmos()`

7- `private int AlgoritmoA():`

Este es el método que se utiliza para intentar conectar cuando hemos elegido en nuestro programa el algoritmo de encaminamiento A o de rotación.

Se mira la conexión situada en la posición que corresponda del vector de conexiones y se obtiene su bloque de entrada y de salida.

Se recorre el vector conexiones buscando una matriz de entrada que coincida con alguna de las anteriores. En ese caso se comprueba que el bloque de entrada esté en la matriz intermedia que se debe escoger según nuestro algoritmo A. En caso afirmativo se aumenta una variable entera llamada `ok1` y se hace que otra variable llamada `puntero` apunte a la siguiente matriz intermedia ya que el algoritmo es el A o de rotación.

La variable que indica el número de matrices intermedias que hay es `nmi`. Para saber la matriz intermedia que se está analizando se hace `puntero % nmi`, con lo que la variable entera puntero se incrementa pasando de la matriz `nmi-1` (`nmi` de 4 a 7) a la 0 al aumentarla.

Con las salidas se repiten las mismas operaciones pero aumentando la variable `ok2`.

Si alguna de estas variables (`ok1 u ok2`) es igual a `nmi` o su suma supera o iguala a `nmi`, se habrá producido un bloqueo. Se pone la conexión actual como desconectada.

```
con.Desconectar();
```

Se crea una nueva conexión con las mismas entrada y salida (pasándole como parámetro del constructor la conexión actual), poniendo la variable `conectado` a `false` y `bloqueado` a `true`. Se le indica también la matriz intermedia por la que habría pasado.

```
Conexiones conevt = new Conexiones(con, false, true, puntero
% nmi);
```

Se añade esta conexión al vector de eventos.

```
eventos.addElement(conevt);
```

Por último se dejan las conexiones que estén en estado de desconexión o bloqueo las cuales se renovarán para poder luego intentar nuevas conexiones.

```
SetSalida();
```

Al final de cada una de las iteraciones que se han hecho para comprobar que se puede conectar la entrada y salida por alguna matriz intermedia (se ha comprobado que no hay bloqueo):

Se pone la conexión actual como conectada.

```
con.Conectar();
```

Se le asigna la matriz de distribución por la que ha conectado.

```
con.setMI(puntero % nmi);
```

Se crea una nueva conexión con las variables `conectado` a true y `bloqueado` a false. Se le asigna la matriz intermedia por la que se realiza dicha conexión. La añadimos al vector de eventos.

```
Conexiones conevt = new Conexiones(con, true, false,
puntero % nmi);
eventos.addElement(conevt);
```

Se añade la entrada y la salida a los vectores `in` y `out` de la Matriz de distribución escogida.

```
seleccionaMI(puntero % nmi, bloqueIn, bloqueOut);
```

Se incrementa la variable puntero para que apunte a la siguiente matriz. Se incrementa la variable `nconexiones`, para llevar la cuenta de las conexiones realizadas.

Se inicializan las variables `ok1` y `ok2`.

8- `public int AlgoritmoB():`

Esta método implementa el algoritmo B o secuencial.

La descripción es similar a la del método `AlgoritmoA`(), pero teniendo en cuenta que las matriz intermedia por la que se va a intentar conectar es la 0, por lo que cada vez que se haga una conexión (no ha habido bloqueo) pondremos el puntero a 0.

9- `public int AlgoritmoC():`

Esta método implementa el algoritmo C o aleatorio.

La descripción es similar a la del método `AlgoritmoA()`, pero teniendo en cuenta que las matriz intermedia por la que se va a intentar conectar es una al azar, por lo que cada vez que se haga una conexión (no ha habido bloqueo) se le da a la

variable `puntero` un valor al azar entre 1 y 4 por `nmi` que representa al número de matrices intermedias de la simulación.

```
puntero = ((int) (Math.random() * 4 * nmi)) + 1;
```

Se muestran a continuación los diagramas de flujo de los métodos correspondientes a los tres algoritmos:

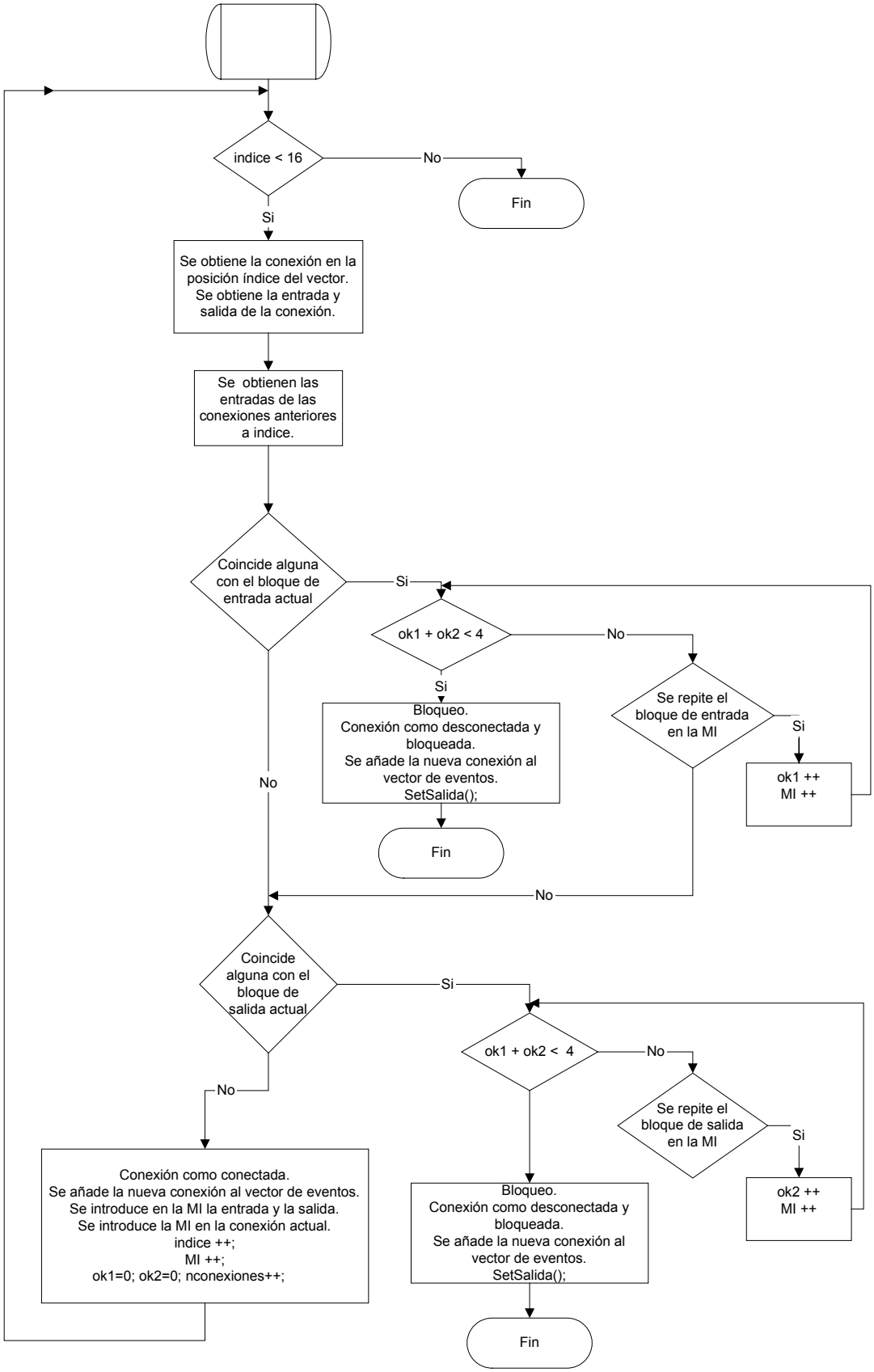

Figura 28: Diagrama de flujo del método `AlgoritmoA()`

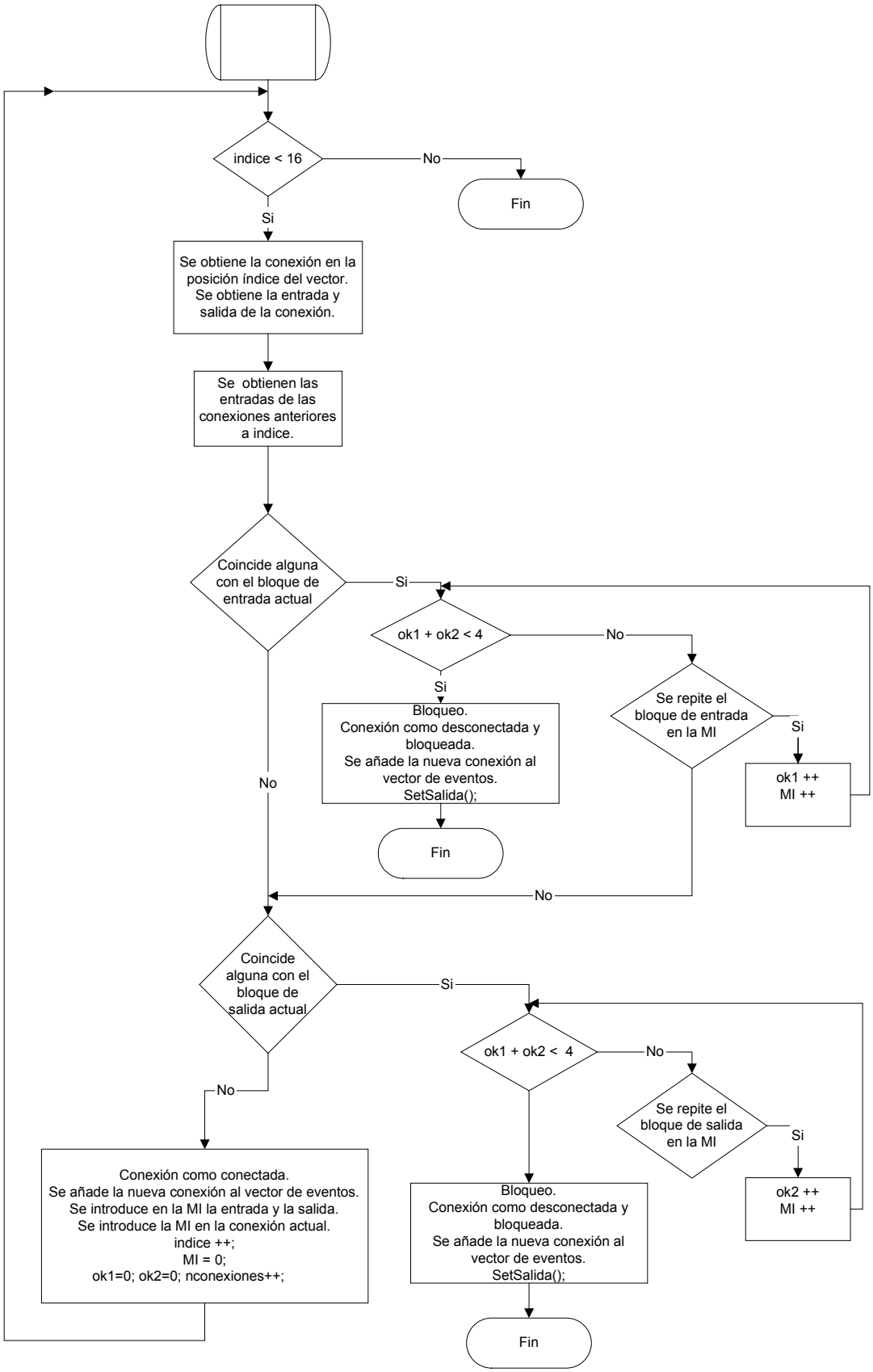

Figura 29: Diagrama de flujo del método `AlgoritmoB()`

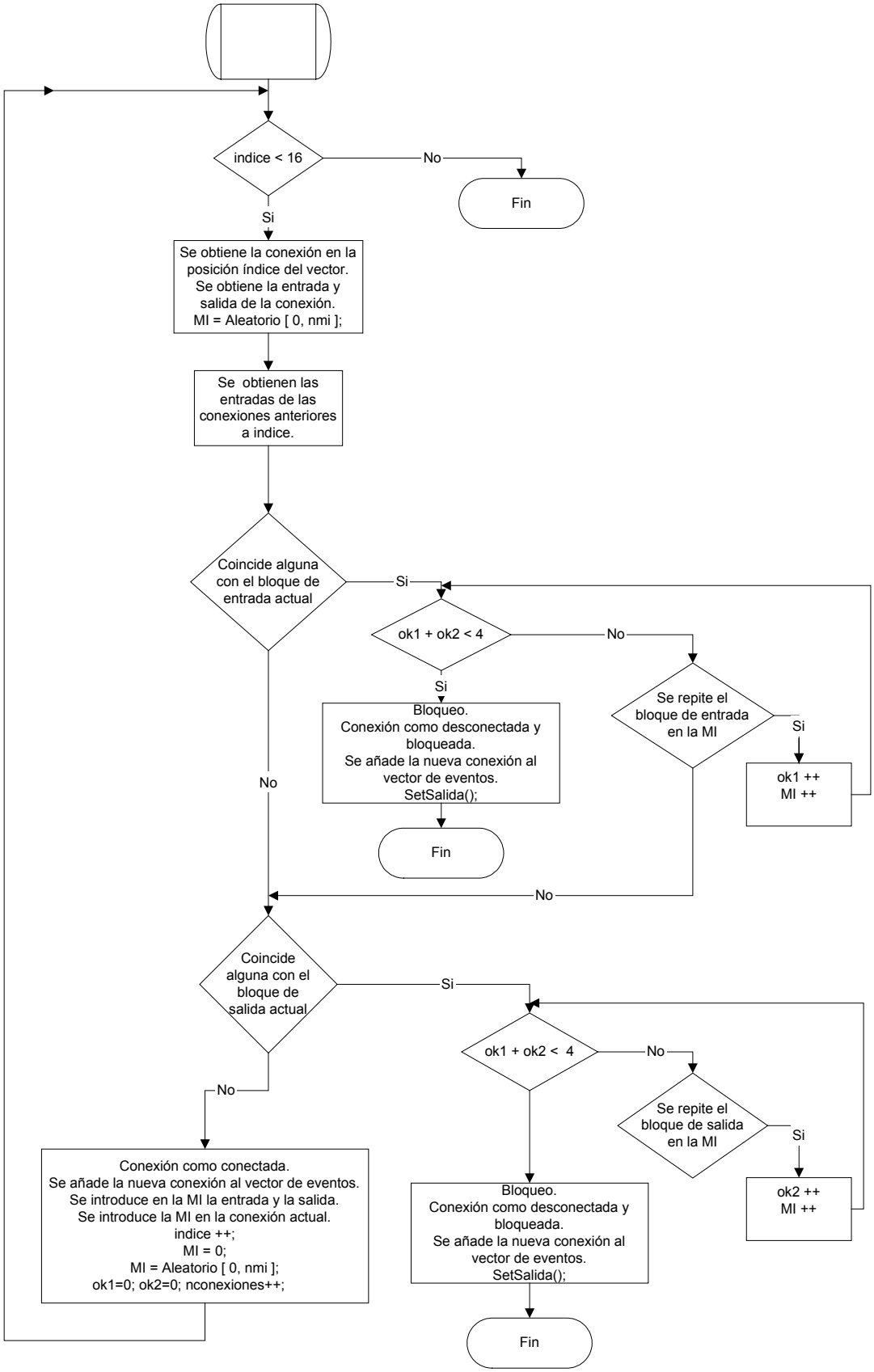

Figura 30: Diagrama de flujo del método `AlgoritmoC()`

10- `private void seleccionaMI(int valor, int entrada, int salida):`

Según la variable valor, se añade a un matriz intermedia u otra, los valores de entrada y salida a los vectores `in` y `out`.

Se llama a los métodos `introduceBloqueIn(entrada)` y `introduceBloqueOut(salida)` de la clase `MatrizIntermedia`.

11- `private void borraInOut(int valor, int ent, int sal):`

Según la variable valor, se borran en un matriz intermedia u otra, los valores de entrada y salida en los vectores `in` y `out`.

Se llama a los métodos `borraBloqueIn(entrada)` y `borraBloqueIOut(salida)` de la clase `MatrizIntermedia`.

Se llama a este método cuando se desconecte una conexión.

12- `private boolean compruebaMI(int valor, int modulo):`

Según la variable `valor` se llamará en una matriz intermedia u otra a `utilizaIn(modulo).`

13- `private boolean compruebaMO(int valor, int modulo):`

Según la variable `valor` se llamará en una matriz intermedia u otra a `utilizaOut(modulo).`

14- `public void setTO(double sto):`

Se cambia la variable `TO` desde la clase `opcionC`.

15- `public void setAlgoritm(int a):`

Determina el algoritmo de distribución a escoger.

16- `public Vector getEventos():`

Devuelve el vector de eventos.

17- `public int getTiempo():`

Devuelve la variable tiempo que simboliza las ranuras.

Algoritmo2
-conexiones: Vector -entradas: Vector -salidas: Vector +eventos: Vector -a: MatrizIntermedia -b: MatrizIntermedia -c: MatrizIntermedia -d: MatrizIntermedia -nmi: int -puntero: int -indice: int -Algoritm: int -BloqueinVisto: boolean -BloqueoutVisto: boolean -tiempo: int -nconexiones: int -logro: boolean -TO: double=8
+Algoritmo2(nmi) - desordenarEnSal() +generaConexion() -desordenar() - SetSalida +getVector(): Vector +resuelveAlgoritmos(): int -AlgoritmoA(): int -AlgoritmoB(): int -AlgoritmoC(): int -seleccionaMI(int valor, int entrada, int salida) -borraInOut(int valor, int ent, int sal) -compruebaMI(int valor, int modulo): boolean -compruebaMO(int valor, int modulo): boolean +getConexiones(): Vector +AlgoritmoEscogido() +setAlgoritm(int a) +setTO(double sto)

+getEventos: Vector +getTiempo: int

Figura 31: Diagrama UML de la clase Algoritmo2 de la opción B

Paneles:

Clase que extiende de `JPanel` y que implementa el diagrama de conexiones del programa.

Sus variables de clase son:

- `etiq`: Una array de 2 dimensiones de `Labels` de 16 por 16. Las filas corresponderán a la entrada de la conexión y las columnas a la salida.
- `EntradaAnterior`: Entero que guarda el número de la entrada de la conexión anterior.
- `SalidaAnterior`: Entero que guarda el número de la salida de la conexión anterior.
- `Fuedesconectado`: Del tipo `boolean` que indicará si hubo desconexión o bloqueo en la ejecución anterior.

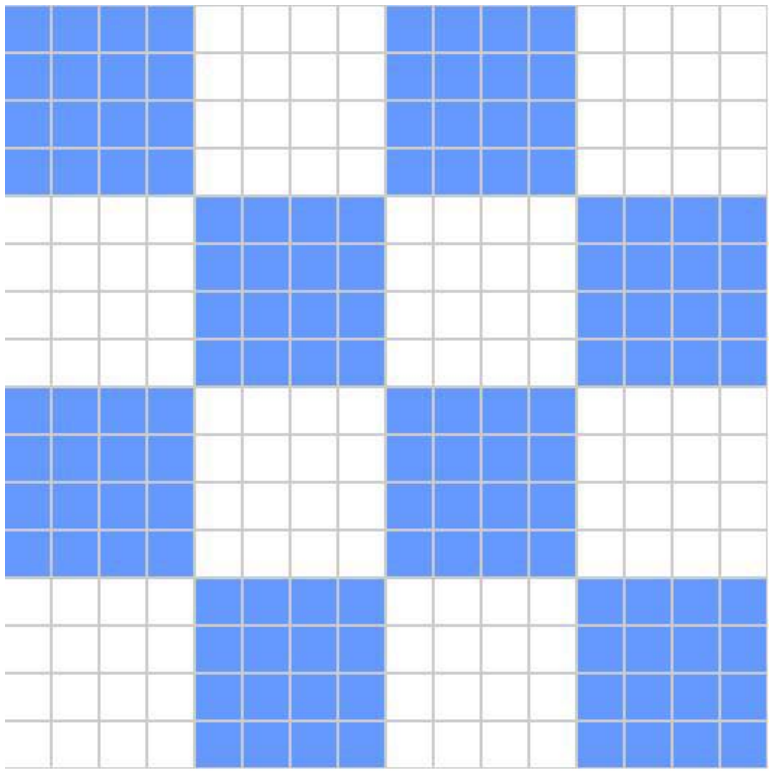

Figura 32: Panel que representa al diagrama de conexiones

Métodos:

- `setConexion(Conexiones con):`

  Del objeto del tipo conexiones que se le pasa obtenemos su entrada y salida, con esto se sabe la fila y columna del `array` de etiquetas que representa al diagrama.

  Si hubo desconexión o bloqueo en el evento anterior, se borra lo que había en su etiqueta correspondiente (tenemos su entrada y salida almacenadas) y según haya conexión, desconexión o bloqueo en la actual se hará:

  Conexión: Pintar de negro el número de matriz intermedia de la conexión en la etiqueta que le corresponde.

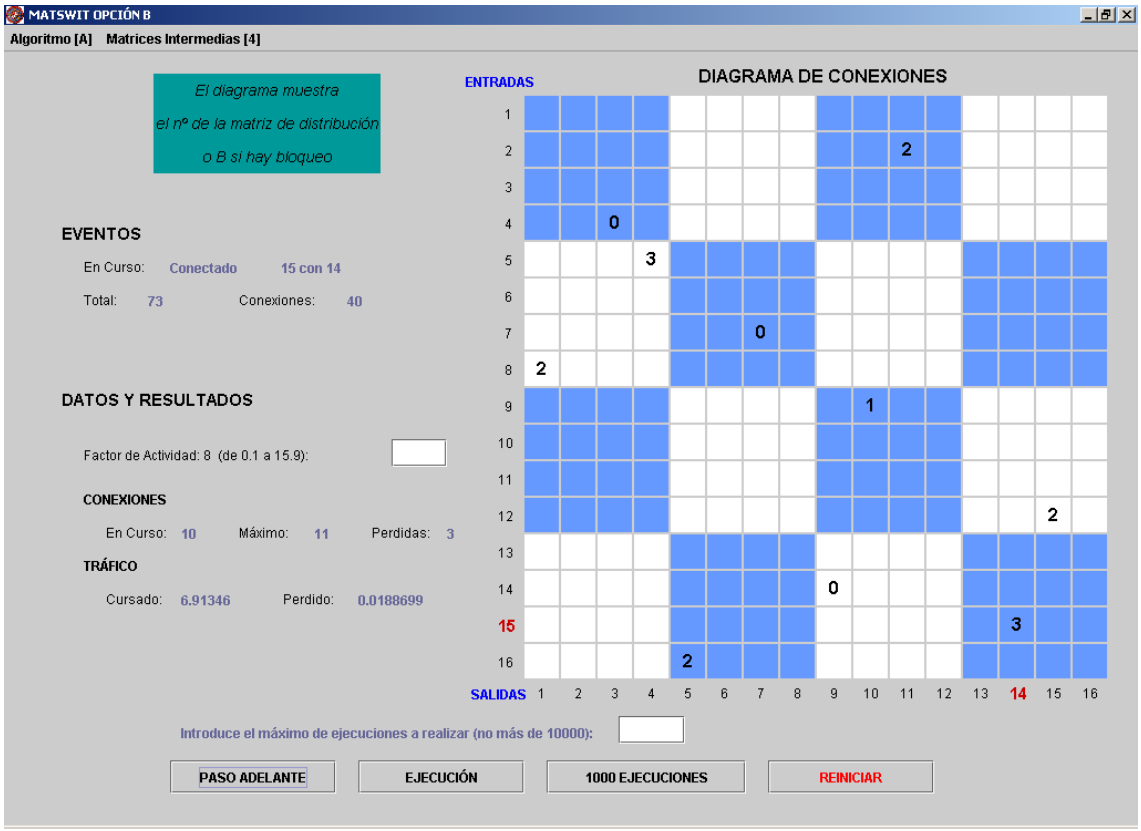

Figura 33: Situación de conexión

Desconexión: Pintar de rojo el número de matriz intermedia de la desconexión en la etiqueta que le corresponde.

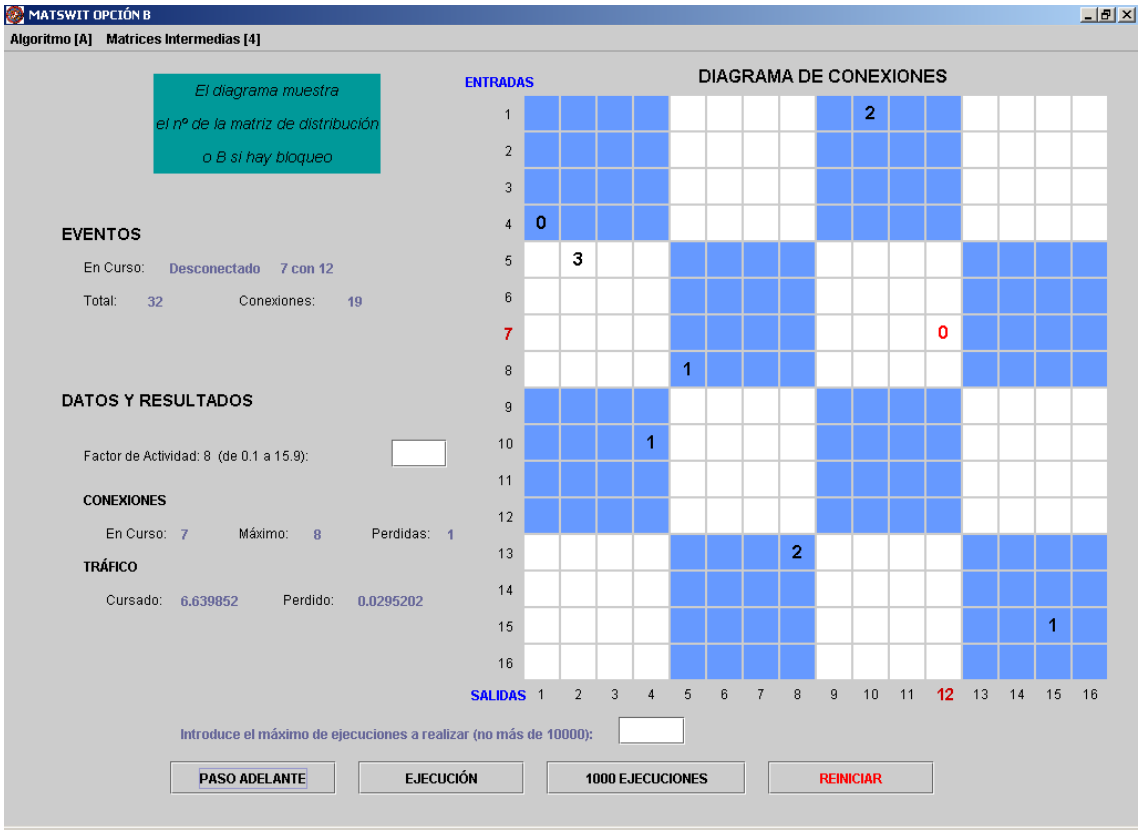

Figura 34: Situación de desconexión

Bloqueo: Pintar una B roja en la etiqueta que le correspondería si hubiera conexión.

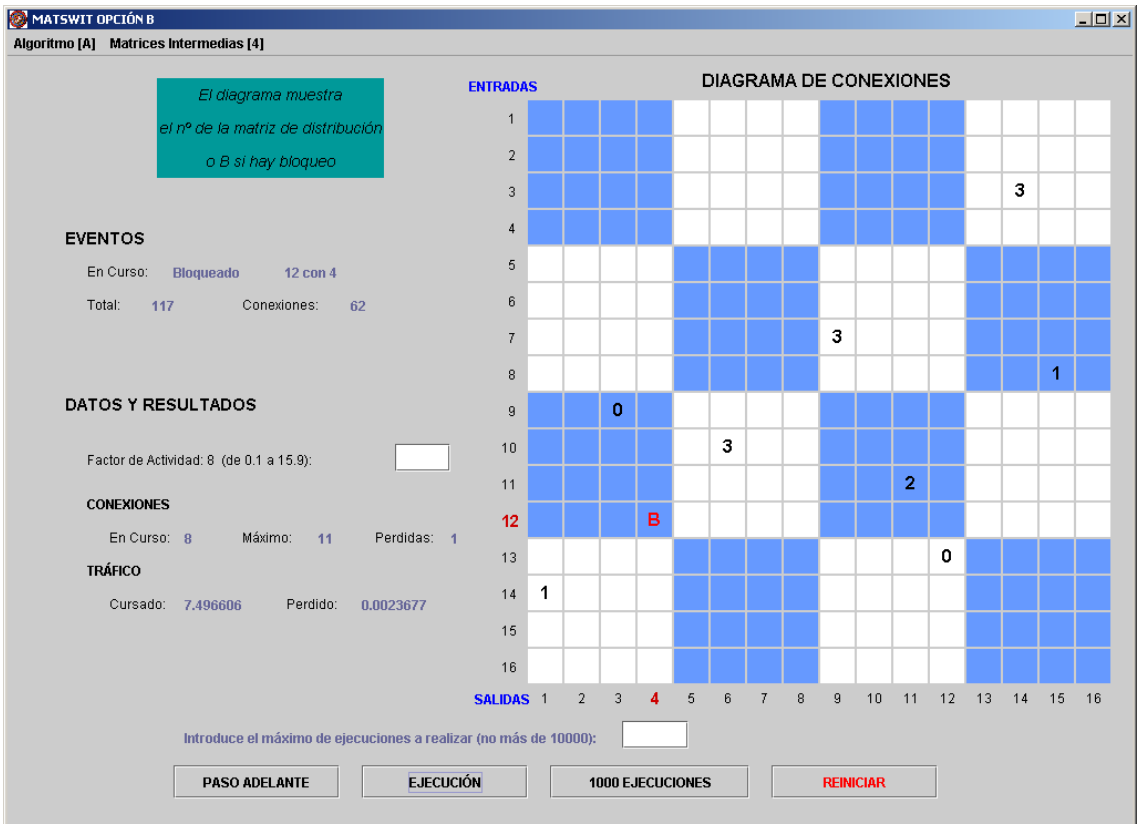

Figura 35: Situación de bloqueo

Al final se guarda la entrada y salida de esta conexión en las variables de `entradaAnterior` y `salidaAnterior`.

- `Limpiar()`:

Borra el contenido de todas las etiquetas del diagrama de conexiones.

Paneles
Etiq: Label[ ][ ] -entradaAnterior: int -salidaAnterior: int -fuedesconectado: boolean
+Paneles() +setConexion(Conexiones con) + Limpiar()

Figura 36: Diagrama UML de la clase Paneles de la opción B

OpcionC:

Clase principal de nuestro programa que extiende de `JFrame`. En ella se encuentra el método `main`.

En esta clase vamos a definir e inicializar todos los objetos del entorno gráfico tales como botones, cajas de texto, pestañas, etiquetas, tablas y un objeto de la clase Paneles.

Muchos de estos objetos estarán escuchando eventos. Se citan los métodos que estos objetos proporcionan para que se ejecuten cuando se recibe el evento:

- `jMenuItem1ActionPerformed`:

  Se ejecuta al seleccionar la opción A del menú Algoritmo. Llama al método `Reiniciar()` que les devuelve el valor inicial a todas las variables, limpia el diagrama de conexiones y las etiquetas de estadísticas. Finalmente deja seleccionado el algoritmo A como algoritmo de encaminamiento.

- `jMenuItem2ActionPerformed`:

  Se ejecuta al seleccionar la opción B del menú Algoritmo. Llama al método `Reiniciar()`. Finalmente deja seleccionado el algoritmo B como algoritmo de encaminamiento.

- `jMenuItem3ActionPerformed`:

  Se ejecuta al seleccionar la opción C del menú Algoritmo. Llama al método `Reiniciar()`. Finalmente deja seleccionado el algoritmo C como algoritmo de encaminamiento.

- `jMenuItem4ActionPerformed`:

  Se ejecuta al seleccionar la opción 4 del menú Matrices Intermedias. Llama al método `Reiniciar()`. Este método principalmente pone la variable `MI` a 4, especificando que en la simulación se tendrán cuatro matrices intermedias.

- `jMenuItem5ActionPerformed`:

  Se ejecuta al seleccionar la opción 5 del menú Matrices Intermedias. Llama al método `Reiniciar()`. Este método principalmente pone la variable `MI` a 5, especificando que en la simulación se tendrán cinco matrices intermedias.

- `jMenuItem6ActionPerformed`:

  Se ejecuta al seleccionar la opción 6 del menú Matrices Intermedias. Llama al método `Reiniciar()`. Este método principalmente pone la variable `MI` a 6, especificando que en la simulación se tendrán seis matrices intermedias.

- `jMenuItem7ActionPerformed`:

Se ejecuta al seleccionar la opción 7 del menú Matrices Intermedias. Llama al método `Reiniciar()`. Este método principalmente pone la variable `MI` a 7, especificando que en la simulación se tendrán siete matrices intermedias.

- `jTextField1ActionPerformed`:

Se ejecutará cuando introduzcamos algo en la caja de texto `JTexField1` correspondiente al "factor de actividad".

Se llama al método `Reiniciar()`.

Se debe introducir en la caja de texto un número entre 0,1 y 15,9. Este valor se guardará en la variable `TO`, que representa al factor de actividad del programa. En el caso de que nuestro número tenga más de tres decimales se ignorarán los siguientes decimales.

Si se introducen caracteres no apropiados, como letras u otros, o un número fuera del rango saltará un mensaje de aviso.

- `JTextField2ActionPerformed`:

Se ejecutará cuando introduzcamos algo en la caja de texto `JTextField2`.

Llamamos al método `Reiniciar()`.

Se debe introducir un valor entero entre 0 y 10.000, este valor será el número de ejecuciones que se podrán realizar seguidas cuando se pulse el botón de 1000 ejecuciones, cambiando por tanto el texto del botón al introducir estos datos. También sirve como límite para el botón Ejecución, en el caso de que no se encuentre un bloqueo en ese número de ejecuciones.

Tanto si se introduce un número con decimales, caracteres inapropiados o un número mayor de 10.000 saltará un mensaje de aviso.

- `jButton1ActionPerformed`:

Método que se ejecuta al pulsar el botón Paso Adelante. Se llama al método `paso()`, de esta misma clase.

- `jButton2ActionPerformed`:

Método que se ejecuta al pulsar el botón Ejecución. Con este botón se realizan ejecuciones hasta que se produzca un bloqueo o se hayan superado las máximas ejecuciones permitidas seguidas. Se llamará a método `paso()` hasta que la conexión esté bloqueada. En el caso de que se superen superado las máximas ejecuciones permitidas seguidas, se mostrará un mensaje de aviso.

- `jButton3ActionPerformed`:

Se ejecuta al pulsar el botón 1000 ejecuciones. Llama al método `paso()` tantas veces como le hayamos indicado en el campo de texto `JTextField2`, 1000 por defecto.

- `jButton4ActionPerformed`:

  Se ejecuta al pulsar el botón Reiniciar. Llama al método `Reiniciar()`.

Otros métodos de esta clase:

- `private void Reiniciar()`:

Devuelve el valor inicial a todas las variables, limpia el diagrama de conexiones y las etiquetas de estadísticas.

- `private void pantallaCompleta(JFrame jf)`:

Con este método se consigue que el `JFrame` se sitúe ocupando toda la pantalla. La configuración de la pantalla debe estar a 1024 por 768 píxeles para que la visualización de la opción A sea completa.

- `private void escogeAlgoritmo()`:

Se llama a métodos de la clase `Algoritmo2` para cambiar el tipo de algoritmo de encaminamiento a utilizar en esa clase, según la variable `Algoritmo`.

- `private void paso()`:

Llama al método de la clase algoritmo `resuelveAlgoritmos()` que se ejecutará hasta obtener un evento, ya sea conexión, desconexión o bloqueo.

Estos eventos se van obteniendo de la cima del vector de eventos. Se obtiene cada vez que se ejecute este método un objeto del tipo conexión que es analizado. Son extraídas su entrada, salida, matriz de distribución y se determina, mediante las variables `conectado` y `bloqueado`, si hubo conexión, desconexión o se bloqueó. Se escribe en cada paso, en las etiquetas de eventos, la entrada la salida y el estado del evento (conectado, bloqueado o desconectado).

También se va contando con unas variables el número de veces que se ha producido un tipo de evento. Con estas variables se obtienen datos como el máximo número de conexiones que ha habido en curso o las que hay actualmente. Con la variable ranuras, obtenida de la clase `Algoritmo2` en la que se llama `tiempo`, se pueden calcular una serie de datos estadísticos como Tráfico Perdido o Tráfico Cursado.

En esta clase también se pintan de rojo las etiquetas que corresponden a la numeración de entradas y salidas. En cada ejecución se cambia a rojo el color de la etiqueta de entrada y de salida.

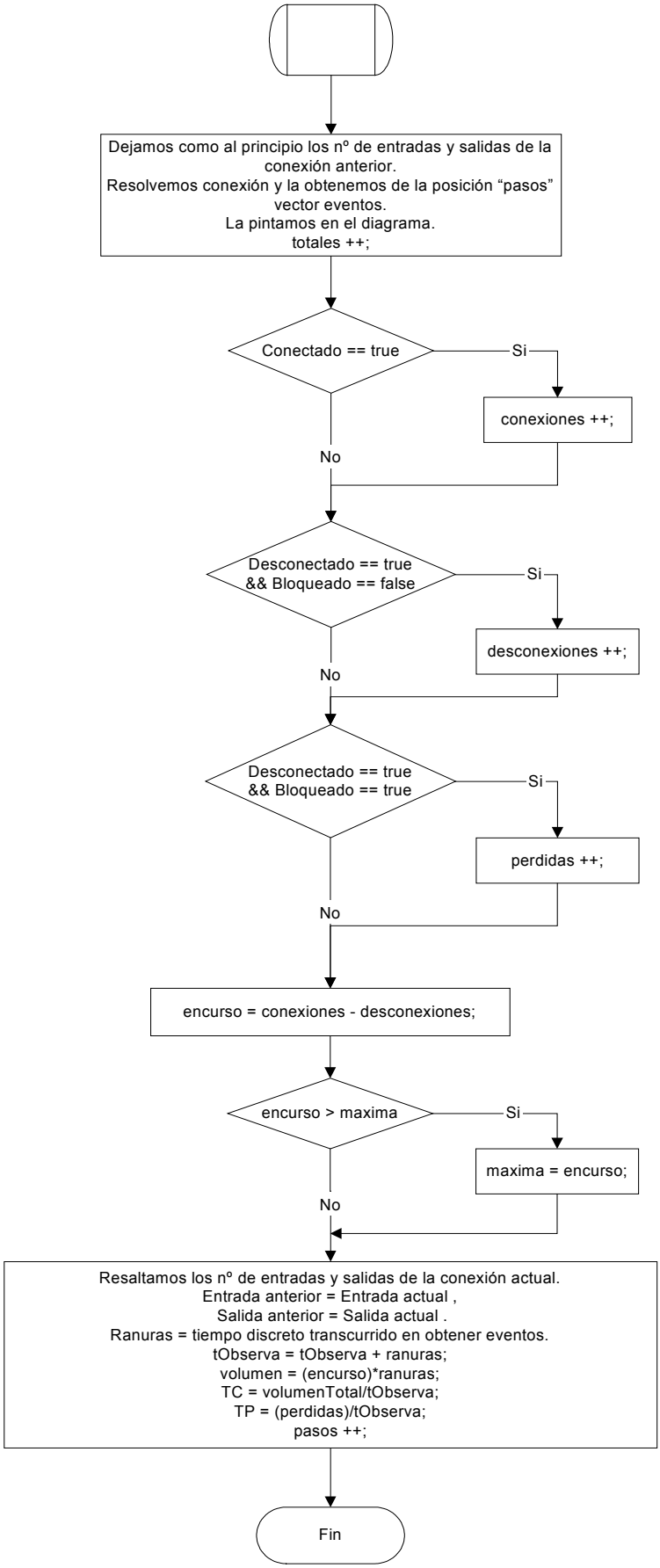

Figura 37: Diagrama de flujo del método `paso()`

- `private double acortaNumero(double T):`

  Método para acortar un número, evitando así números con muchos decimales, le quitamos precisión al número pero podemos escribirlo sin problemas en etiquetas y similares.

- `private String precisa(String s):`

  Método para evitar números del orden 10^{-6} o menores y poder mostrarlos escritos sin el símbolo E-n, siendo n el orden de la exponencial. Por ejemplo, el número 1.222222E-6 después de pasar por este método sería 0.00000122.

Para ésta clase no se ha representado su diagrama UML por no considerarlo apropiado, debido a las numerosas variables de clase como botones, etiquetas, etc. que contiene y que no son determinantes a la hora de la comprensión del programa.

Dialogo:

Clase que extiende de `JDialog` creada para que se muestre cada vez que haya que avisar al usuario del programa de cualquier eventualidad, por ejemplo de que introduzca datos correctos en las cajas de texto.

- `public void CentraDialogo():`

  Sirve para situar y dar tamaño al Dialogo.

- `public void mensaje(String ms):`

  Introduce el mensaje que para cada caso mas conveniente en una etiqueta dentro de `Dialogo`.

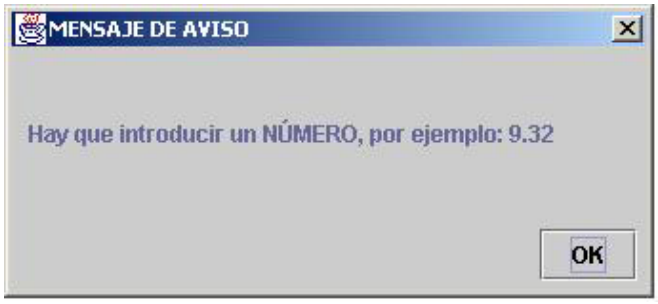

Figura 38: Ejemplo de Dialogo para la opción B

Capítulo 6
Uso académico de la aplicación

6.1 Teoría de conmutación

En este apartado se comentará brevemente una serie de conceptos básicos de conmutación antes de describir los posibles usos académicos de nuestro programa.

Conmutación: Por conmutación de Circuitos se entiende la técnica que permite que dos terminales emisor y receptor se comuniquen a través de un circuito único y específico, establecido para tal propósito antes del inicio de la misma y liberando una vez que ha terminado, quedando en este caso a disposición de otros usuarios para su Utilización.

La conmutación está asociada principalmente a una central telefónica y consta de dos partes básicas:

1) El establecimiento, mantenimiento y liberación de la comunicación (procesamiento de la llamada) coordinados por el control.

2) El establecimiento de la vía física por la cual se produce la comunicación realizada por la red de conexión.

Los Tipos de conmutación se pueden dividir de acuerdo a:

Según su operación:

- Manual y automática.
- Analógica y Digital.
- Espacial y Temporal.

Según el tipo de servicio:

- De circuitos.
- De paquetes.

Según su utilización:

- Privadas.
- Públicas.

El tipo de conmutación que ocupa al simulador a implementar es de circuitos y espacial.

Conmutación de circuitos:
Es una técnica en la que los equipos que se comunican entre sí utilizan un canal físico dedicado extremo a extremo, que se mantiene durante el tiempo de duración de la llamada o por el periodo de contratación.

Conmutación espacial:
Método de conmutación de circuitos, en el que cada conexión que pasa por el circuito sigue una vía separada.

6.2 Objetivos académicos de la aplicación

El objetivo de la herramienta de simulación es que el alumno entienda que es un conmutador espacial multietapa y que ventajas e inconvenientes presenta respecto al conmutador de matriz cuadrada (crossbar) de una sola etapa.

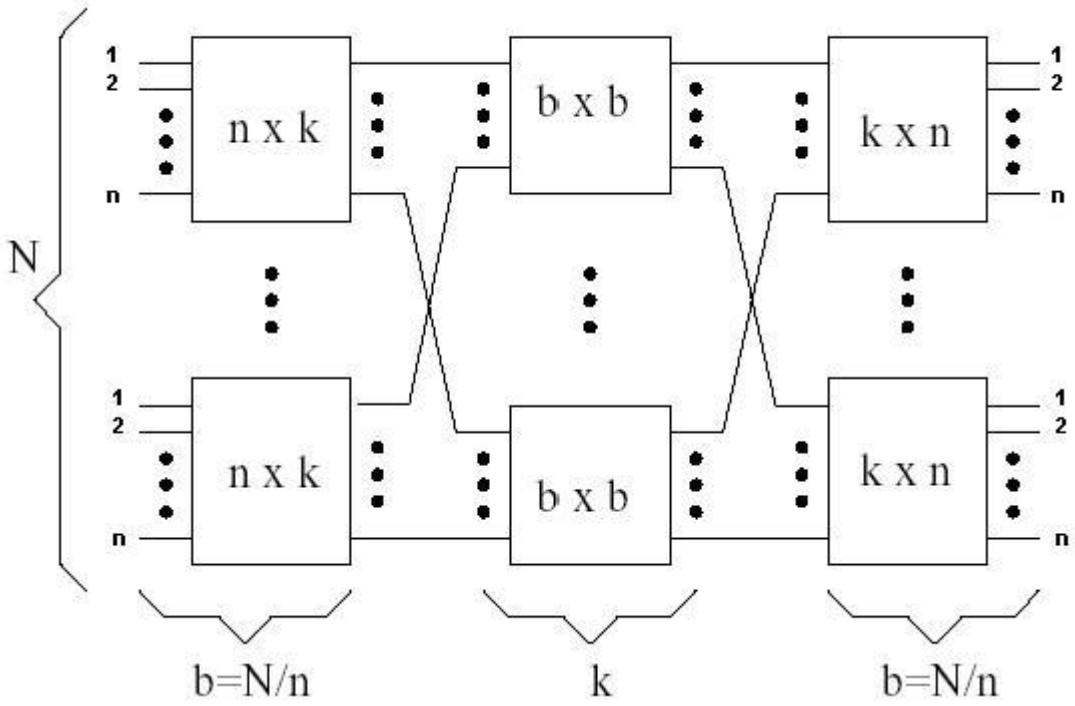

Figura 39: Conmutador Multietapa

6.3 Conmutador de matriz crossbar y conmutador multietapa

El conmutador de matriz cuadrada o crossbar es la solución más simple al problema de interconectar N entradas con N salidas.

Esta solución presenta accesibilidad total, ya que desde cualquier entrada se puede seleccionar cualquier salida, y no presenta bloqueo interno, ya que cada par entrada-salida dispone de su punto de cruce. Por otra parte, el número de puntos de cruce de un conmutador crossbar de N entradas y N salidas es de N X N, es decir, el número de puntos de cruce crece en proporción al cuadrado del número de entradas

(por ejemplo N = 1000, 10^6 puntos de cruce), alcanzando para valores elevados de N soluciones demasiado costosas o no implementables.

Aparece entonces la necesidad de nuevas arquitecturas que permitan mantener las características del crossbar disminuyendo el número de puntos de cruce. Un primer intento seria usar 2 conmutadores de 1000 entradas y 100 salidas, disminuyendo el número de puntos de cruce a $2*10^5$.

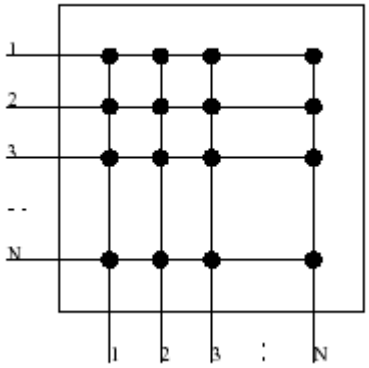

Figura 40: Conmutador de matriz cuadrada o crossbar

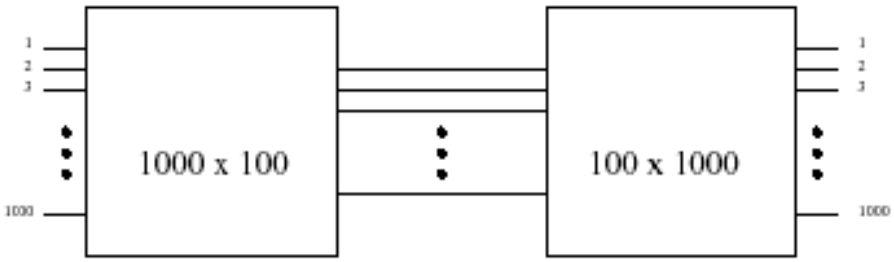

Figura 41: Dos conmutadores crossbar 1000x100 y 100x1000 implementando un conmutador 1000x1000

El problema de esta solución es que, a pesar de presentar accesibilidad total, presenta bloqueo en sentido estricto (imposible establecer 101 enlaces simultáneos).

Otra posibilidad sería desdoblar el crossbar en varios, por ejemplo. 10 unidades de crossbars de 100 X 10. Ahora el número de puntos de cruce ha disminuido a 2 X (100X10) X 10 = 2 X 10^4, pero aparece un nuevo problema: además de bloqueo interno se ha perdido la accesibilidad total, ya no existe interconexión posible entre distintas unidades.

Finalmente, la solución a utilizar es la red multietapa, que representa accesibilidad completa. Este tipo de red, en función del número de etapas intermedias y del tamaño de las etapas presentará o no bloqueo y mejorará o no el número de puntos de cruce.

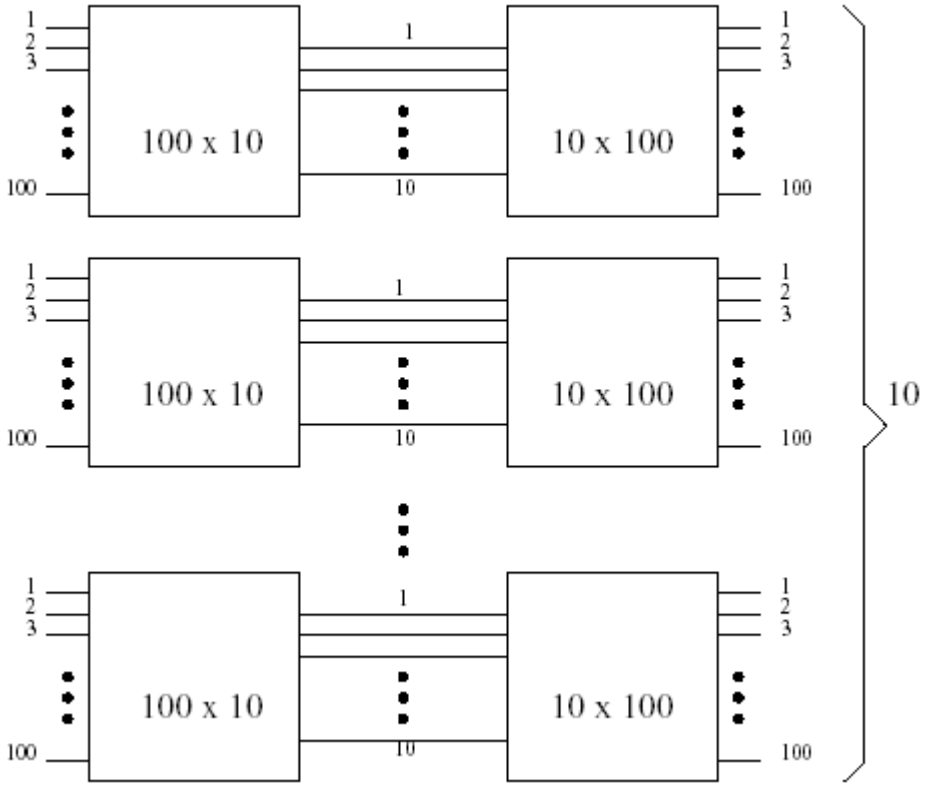

Figura 42: Etapas de 100*10 implementando un conmutador 1000X1000

Para comparar el conmutador multietapa con el crossbar hay que estudiar el número de puntos de cruce que presenta el conmutador multietapa. Siendo b = N/n el número de etapas tanto de entradas, de tamaño n x k, como de salida, con un tamaño de k x n, y k el número de etapas intermedia, de tamaño b x b, el número de puntos de cruce M viene dado por $M = 2 \times (n \times k) \times N/n + (N/n)^2 \times k$.

6.4 Condición de Clos

Es la condición para la que una red multietapa no presenta bloqueo interno en sentido estricto. Si se tiene que en una etapa de entrada de la red, de tamaño n x k, se han realizado todas las conexiones excepto una, se puede solicitar la conexión por n -1. Análogamente, también se puede tener una etapa de salida con n -1 conexiones realizadas. Por tanto, en el peor caso posible, que es que ninguna de las n – 1 conexiones de la matriz de entrada sea con alguna de las n – 1 líneas conectadas de la matriz salida, se ocuparán 2 x (n - 1) etapas de distribución.

Entonces, para que no haya posibilidad de bloqueo, el número de etapas de distribución debería ser mayor o igual a 2 x (n -1) + 1= 2n –1, esto es la condición de Clos. Se prescinde de la demostración matemática y solo se comenta de forma intuitiva.

Observar que si tenemos 2n-1 etapas de distribución con la situación anterior. Es decir 2 x (n-1) ocupadas, sólo quedaría una libre, y el encaminamiento, por tanto, se producirá a través suyo, ya que no queda ningún otro camino libre.

En el programa se tiene 4 matrices de entrada 4XN y 4 matrices de salida NX4. Para que se cumpla la condición de Clos deberemos tener 2 x 4 – 1 = 7 etapas intermedias. En la opción B se puede escoger el número de matrices intermedias, de 4 a 7, si se escoge 7 matrices se puede observar que no habrá bloqueo en ningún caso.

6.5 Tráfico en redes

Cuando se habla de tráfico, de manera intuitiva uno puede hacerse una idea bastante cercana al concepto matemático del tráfico. Supóngase que se dispone de un enlace entre 2 puntos. Dicho enlace, si siempre estuviese ocupado, se dirá que transporta tráfico unitario, mientras que si nunca se ocupara, transportaría tráfico cero. Ahora supóngase que el enlace en cuestión a veces está ocupado y a veces no: el tráfico que transportará estará situado entre 0 y 1, independientemente del modo en que haya estado ocupado o libre el enlace. Esta relación de ocupación de un enlace, o medida del tráfico transportado por este enlace, se medirá en Erlangs, siendo en este simple caso el tráfico mínimo de 0 Erlangs y el máximo de 1 Erlang. En general (M enlaces), una medida de tráfico en Erlangs nos indicará el número medio de enlaces ocupados (esta medida será ≥ M).

Cuando una cierta cantidad de tráfico es ofrecido a una red (Tráfico Ofrecido, TO), ésta puede cursarlo de manera parcial (Tráfico Cursado, TC) y rechazar el resto (Tráfico Perdido, TP). Estos valores irán relacionados entre sí por la probabilidad de bloqueo de la red (Pb). Así se tiene TP = TO x Pb y TC = TO x (1 - Pb).

Para nuestra opción B se ha supuesto que las llamadas por media duran dos minutos (120 segundos). El tiempo en el simulador es discreto, va representado por ranuras o slots, cada ranura representará un segundo de tiempo real por lo que una llamada por media ocupará 120 slots.

La probabilidad de colgar será el inverso de la duración media de la llamada (q_1 = 1/120).

La probabilidad de que se produzca una llamada vendrá definida por:

$$p_2 = \frac{a \times q_1}{1 - a + a \times q_1}$$

Esto viene de la definición del factor de actividad (a):

$$a = \frac{1/\mu}{1/\mu + 1/\lambda} = \frac{1/q_1}{1/q_1 + q_2/p_2}$$

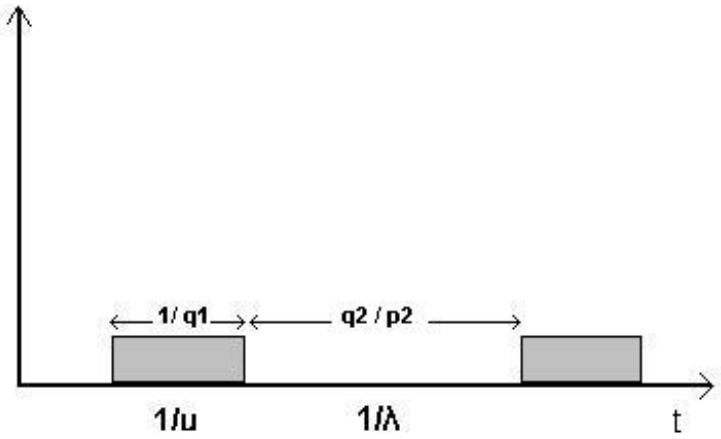

Figura 43: Diagrama que muestra el tiempo medio de servicio (1/μ) y tiempo medio entre llegadas (1/λ).

Como se ha definido:

q_1 = $1/120$ como la probabilidad de colgar, esto es el inverso de la duración media de una llamada en slots, se tiene que:

$$p_2 = \frac{a}{120 - 119 \times a}$$

Finalmente:

$$p_2 = \frac{1}{120 \times 16} \times \frac{a}{1 - a \times \frac{119}{120 \times 16}}$$

La representación del tiempo utilizada es, como se ha comentado con anterioridad, discreta, representada por ranuras. Las variables aleatorias se han tratado siguiendo una distribución uniforme. La forma correcta sería haber hecho una distribución exponencial o de Poisson, pero se puede demostrar matemáticamente que la distribución de Poisson coincide con la binomial para un número de experimentos muy alto y probabilidad muy baja. Así, tomando las ranuras lo suficientemente pequeñas (cada una de un segundo) y eventos de probabilidades muy bajas el programa se comporta como si se hubiera programado con la distribución de Poisson.

6.6 Trabajo a realizar por el alumno

Seleccionando la opción A del simulador:

- Para la red de conexión que se representa responder a las siguientes preguntas: ¿Tiene accesibilidad completa? ¿Porqué? ¿Cuántos puntos de cruce requiere y qué porcentaje de ahorro representa respecto a una matriz cuadrada 16x16?

- Describir el modo de operación de los distintos algoritmos. Etiquetar cada uno de dichos algoritmos mediante una de las siguientes palabras: rotante, aleatorio o secuencial.

- Seleccionar uno de los algoritmos, por ejemplo el a, y explicar mediante un ejemplo porqué se introducen situaciones de bloqueo interno. Para cada algoritmo intentar 10 secuencias de forma rápida y observar cuántas han tenido éxito. Comparar los resultados obtenidos. ¿Qué algoritmo prevé mejor desde el punto de vista de bloqueo interno?

- Para cada algoritmo realiza dos pruebas rápidas de 1000 ejecuciones y obtener en cada prueba el número medio de conexiones realizadas y el porcentaje de ejecuciones con éxito. Hallar los valores medios de dichos parámetros y comparar el comportamiento de los diferentes algoritmos.

- ¿Cuál es el número mínimo de conexiones que puede realizar cada algoritmo sin tener bloqueo interno?

Seleccionando la opción B del simulador:

- Explicar mediante un ejemplo sencillo cómo se representan las conexiones en el diagrama mostrado.

- Pulsando el botón de ejecución, obtener una situación de bloqueo. A la vista de las matrices de distribución conectadas a las correspondientes matrices de entrada y de salida de la conexión bloqueada ¿cómo se refleja dicha situación de bloqueo en el diagrama de conexiones?

- Mediante una ejecución rápida de 5000 eventos establecer la relación entre el tráfico cursado (TC), el tráfico rechazado (TP), las conexiones hechas (Cm) y los intentos de conexión rechazados (Cl).

Seleccionando la opción B del simulador con un factor de actividad de 15 Erlangs, responder a las siguientes preguntas:

- ¿Cuál es el número mínimo de matrices de distribución para no tener bloqueo interno en sentido estricto?¿Depende dicho resultado del algoritmo seleccionado? Comprobar ambas respuestas mediante simulación.
- ¿Cuántos puntos de cruce se requieren para garantizar la condición de no bloqueo en sentido estricto? Comparar este valor con el número de puntos de cruce que precisa el crossbar 16x16. A la vista del resultado anterior, ¿en qué condiciones es ventajoso diseñar una red de Clos?

Capítulo 7
Conclusiones y líneas futuras

7.1 Conclusiones

Observando la aplicación finalizada se pueden obtener una serie de conclusiones.

La conclusión principal es que se ha conseguido programar un simulador didáctico de un conmutador espacial multietapa, que engloba dos simuladores: Uno en modo simple, en el que la duración de las llamadas es infinita y otro en modo continuo, con la duración de las llamadas finita.

En ambos casos se ha conseguido una modernización de la interfaz gráfica haciendo más intuitivo y agradable el entorno visual al alumno.

Como consecuencia de haber cambiado la forma de representar las matrices crossbar en la opción A se ha facilitado la comprensión de los algoritmos de distribución de la aplicación.

Se corrigieron ciertos aspectos teóricos para facilitar la comprensión de los conceptos que se intentan enseñar con este simulador.

Se integraron las opciones A y B del “Matswit” en una sola del “SIMUCOM”, la opción A. Estas dos opciones eran de modo simple, con la duración de llamadas infinita, por lo que se ha integrado los experimentos realizados con la opción A del “Matswit” y los experimentos realizados con la opción B del mismo en una sola ventana con una mayor comprensión de los experimentos en modo simple.

El código fuente se ha dejado disponible, quedando así el simulador abierto a posibles modificaciones y mejoras.

Al haber programado la aplicación en Java, se ha conseguido que sea “portable”, por lo que podemos ejecutarlo en cualquier sistema operativo (Windows, LINUX, Machintosh, Solaris, ...) donde Sun Microsystems haya desarrollado máquina virtual Java.

7.2 Líneas futuras

La especificación de este Proyecto Final de Carrera consistió en una serie de objetivos claramente definidos y que se han conseguido en su totalidad. El definir unas líneas futuras para él va a depender de las necesidades didácticas que se puedan ir encontrando durante la utilización de la herramienta a lo largo de los cursos académicos. Se tiene que producir un proceso de retroalimentación en el que los usuarios (alumnos y profesores), dependiendo de hipotéticas necesidades futuras que el simulador pudiera ofrecer. Esta retroalimentación permitiría modificaciones en el programa para ajustarlo a necesidades futuras.

Anexos

Anexo A
Manual de usuario

A.1 Introducción

La aplicación ha sido programada en el lenguaje de programación Java por lo que se debe tener instalado el JRE (Java Runtime Environment) para su ejecución. JRE es el entorno mínimo para ejecutar programas Java 2. Incluye la JVM (Java Virtual Machine) y la API. Está incluida en el J2SE (Java 2 Standard Edition) aunque puede descargarse e instalarse separadamente. En aquellos sistemas donde se vayan a ejecutar programas Java, pero no compilarlos, el JRE es suficiente. Si nuestro propósito es modificar el código de la aplicación y luego compilarlo, se debe instalar J2SE.

Se puede descargar todo este software de http://java.sun.com/j2se.

La aplicación se compone de 3 ficheros con extensión “jar” : Uno para la opción A (opcionA.jar) , otro para la opción B (opcionB.jar) y otro para el menú de inicio que abre estas 2 opciones (SIMUCOM.jar).

En el mismo directorio donde se instalen estos ficheros se debe grabar la carpeta “Imágenes” que contiene los ficheros:

- “sincrossbar.jpg” con la imagen que usa la opción A para representar el conmutador.
- “etsit.jpg” y “upctrelieve.jpg” con los escudos de la E.T.S.I.T. y de la Universidad Politécnica de Cartagena respectivamente cuyo propósito es el de dar un aspecto corporativo a la aplicación.

Para ejecutar la aplicación se debe ejecutar el fichero SIMUCOM.jar, mediante la sentencia:

```
java –jar SIMUCOM.jar
```

Para la correcta visualización del programa, la configuración de área de pantalla debe ser de 1024 por 768 píxeles.

A.2 Menú de Inicio

Este menú de inicio es una interfaz gráfica sencilla, desde la cual se pueden llamar a los dos diferentes programas implementados.

La ventana del menú de inicio tiene este aspecto:

Figura 44: Menú de inicio del SIMUCOM

Mediante esta botonera se ejecutan cada una de las aplicaciones que componen el simulador.

A.3 Opción A

Este simulador es un simulador de un conmutador multietapa en modo simple, esto es, que la duración de las llamadas es infinita. No se termina la llamada durante la ejecución.

El simulador muestra el encaminamiento de llamadas, las cuales son elegidas al azar, dentro de nuestro conmutador de 3 etapas.

Aspecto de la opción A iniciada:

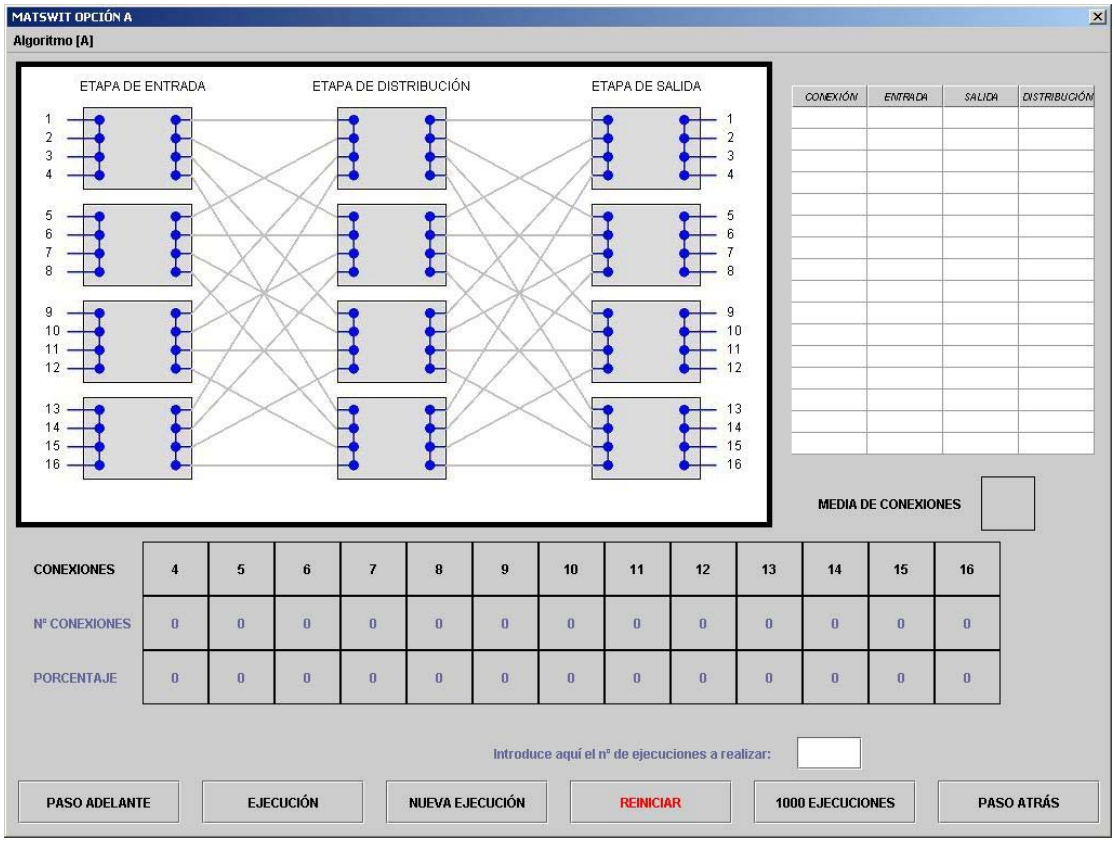

Figura 45: Opción A del SIMUCOM

Los elementos de la opción A son:

- **Dibujo del conmutador**: Por el que se van a ver las conexiones dibujadas, mostrando su encaminamiento.

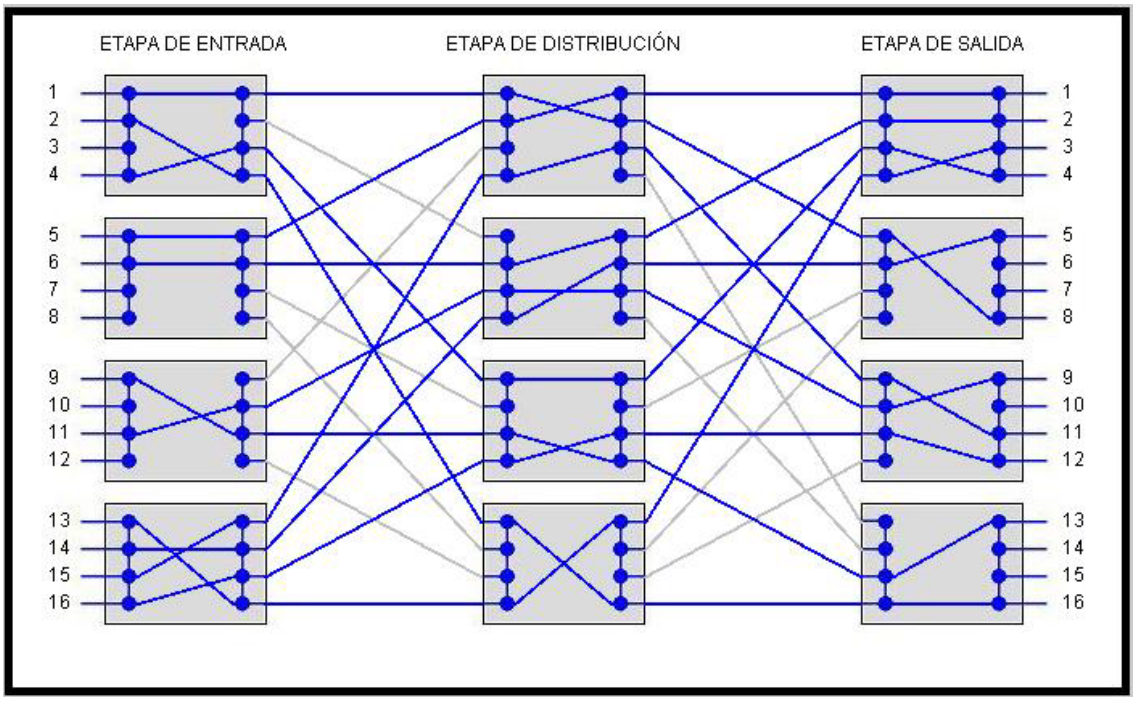

Figura 46 : Dibujo del conmutador de la opción A

- **Tabla de conexiones**: Se representa en esta tabla las conexiones realizadas. Consta de 4 columnas (Conexión, Entrada, Salida y Distribución) y 16 filas correspondientes a las 16 posibles conexiones que se pueden realizar.

  La tabla de conexiones es esta:

CONEXIÓN	ENTRADA	SALIDA	DISTRIBUCIÓN
1	15	11	0
2	11	9	1
3	9	13	2
4	13	3	3
5	1	8	0
6	16	12	2
7	5	1	0
8	6	2	1
9	14	5	1
10	4	4	2
11	2	16	3
12	3	6	Bloqueo

Figura 47: Tabla de conexiones de la opción A

- **Menú de algoritmos:** Con este menú se puede escoger entre los 3 distintos algoritmos de encaminamiento a través de nuestro conmutador.

  El menú de Algoritmos tiene este aspecto:

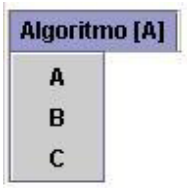

Figura 48: Menú de algoritmos de la opción A

Los botones de la opción A son:

- **Paso adelante**: Muestra, cada vez que se pulsa, el camino que ha seguido la conexión desde su entrada hasta la salida. Va rellenando en la tabla la conexión, entrada, salida y matriz de distribución correspondientes.
  El aspecto del botón “Paso adelante” es:

Figura 49: Botón Paso adelante de la opción A

- **Ejecución**: Muestra todas las conexiones hechas en esa ejecución tanto en el dibujo como en la tabla.
  El aspecto del botón “ejecución” es:

Figura 50: Botón Ejecución de la opción A

- **Nueva ejecución**: Empieza una nueva ejecución, dispuesta para ser pintada y mostrada. Añade a las estadísticas la ejecución anterior.
  El aspecto del botón "Nueva ejecución" es:

Figura 51: Botón Nueva ejecución de la opción A

- **Reiniciar**: Limpia las tablas y los dibujos.
  El aspecto del botón "Reiniciar" es:

Figura 52: Botón Reiniciar de la opción A

- **1000 ejecuciones**: Realiza seguidas 1000 ejecuciones que se añadirán a las estadísticas (ver Caja de Texto "número de ejecuciones").
  El aspecto del botón "1000 ejecuciones" es:

Figura 53: 1000 ejecuciones de la opción A

- **Paso atrás**: Es complementario a Paso adelante. Muestra las conexiones anteriores, borrando la actual.
  El aspecto del botón "Paso atrás" es:

Figura 54: Botón Paso atrás de la opción A

- **Caja de Texto "número de ejecuciones"**: Podemos modificar el número de ejecuciones seguidas a realizar mediante el botón de 1000 ejecuciones. La etiqueta del botón se ajusta al nuevo valor.

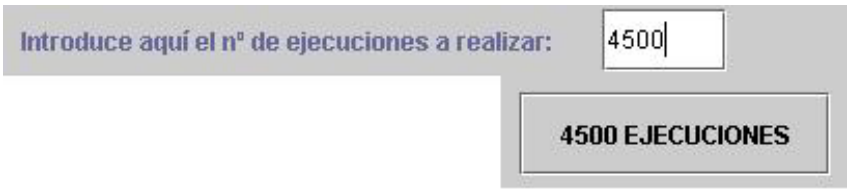

Figura 55: Caja de Texto "número de ejecuciones" y el botón modificado por el nuevo número de ejecuciones introducido

- **Tabla de estadísticas:** Tabla que indica las veces que se produjo en cada ejecución un número de conexiones, así como el porcentaje de cada una.

El aspecto de la Tabla de estadísticas es:

CONEXIONES	4	5	6	7	8	9	10	11	12	13	14	15	16
Nº CONEXIONES	4	8	15	15	42	47	68	45	50	42	20	0	44
PORCENTAJE	1.0	2.0	3.750	3.750	10.5	11.75	17.0	11.25	12.5	10.5	5.0	0.0	11.0

Figura 56: Tabla de estadísticas de la opción A

- **Media de conexiones:** Refleja la media total de conexiones hechas por simulación.

Figura 57: Media de conexiones de la opción A

A.4 Opción B

Este simulador es un simulador de un conmutador multietapa en modo continuo, esto es, que la duración de las llamadas es finita.

Aspecto de la opción B iniciada:

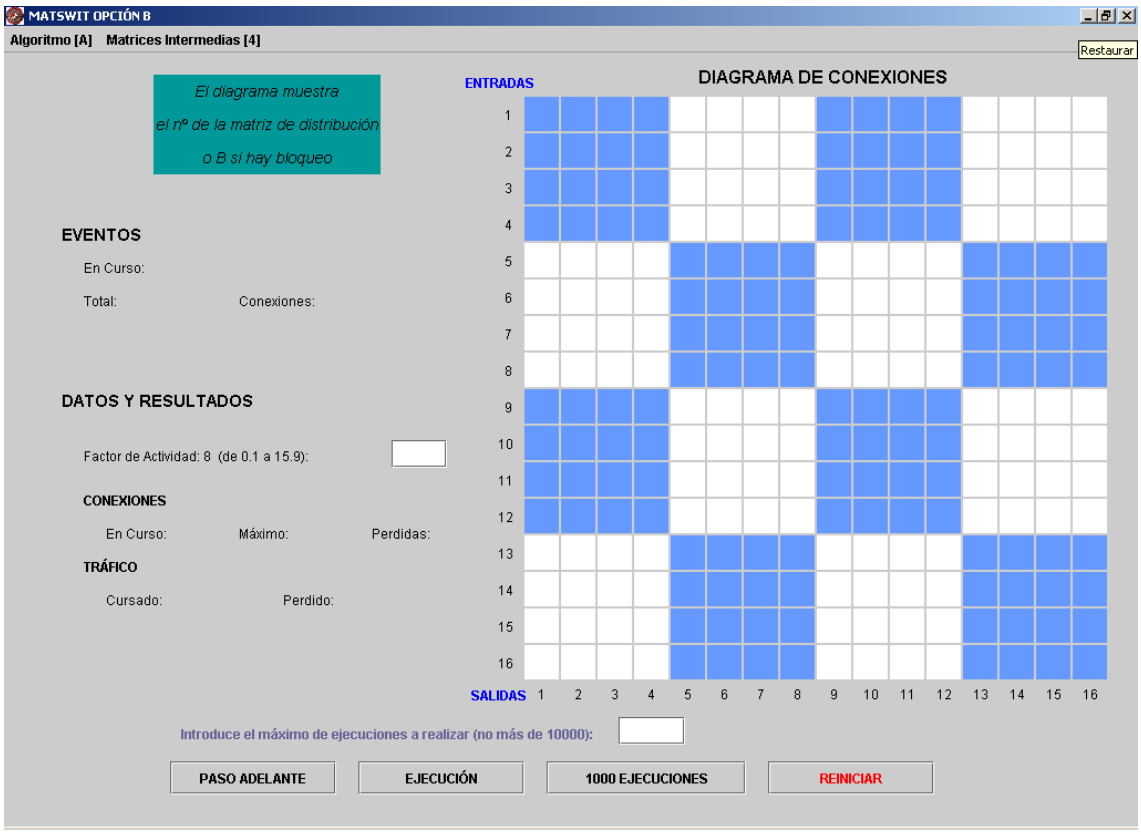

Figura 58: Opción B iniciada

Los elementos de la opción B son:

- **Diagrama de conexiones**:

El simulador muestra mediante el diagrama de conexiones los eventos de conexión, desconexión y bloqueo.

La conexión se representa mediante su entrada, su salida (ambas en rojo) y el número de la matriz intermedia en negro.

La desconexión se representa mediante su entrada, su salida (ambas en rojo) y el número de la matriz intermedia en rojo.

El bloqueo se representa mediante su entrada, su salida (ambas en rojo) y una B roja.

El diagrama de conexiones tiene este aspecto:

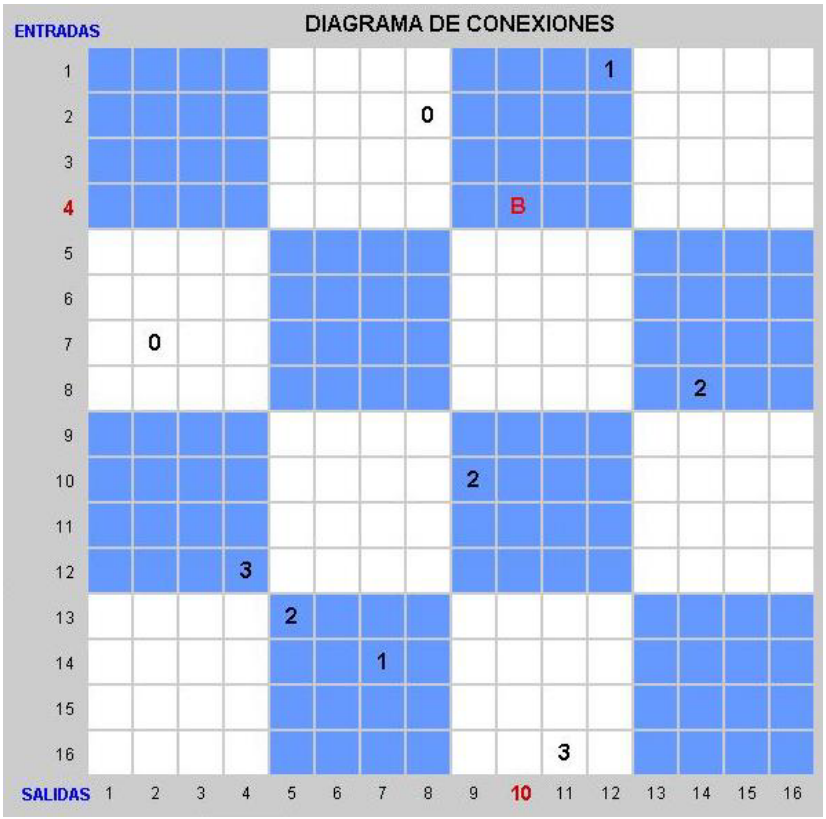

Figura 59: Diagrama de conexiones de la opción B

- **Etiquetas de eventos**: Estas etiquetas muestran el estado de la conexión en curso, su entrada y su salida. Se muestra el número de eventos realizados y el total de eventos que han sido una conexión.

  Estas son las etiquetas de eventos:

Figura 60: Etiquetas de eventos de la opción B

- **Menú de Algoritmos**: Con este menú se puede escoger entre los 3 distintos algoritmos de encaminamiento a través de nuestro conmutador.

  El menú de Algoritmos tiene este aspecto:

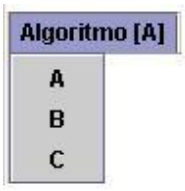

Figura 61: Menú de Algoritmos de la opción B

- **Menú de Matrices Intermedias**: Con este menú se puede escoger el número de matrices intermedias (4, 5, 6 o 7).

El menú de Matrices intermedias tiene este aspecto:

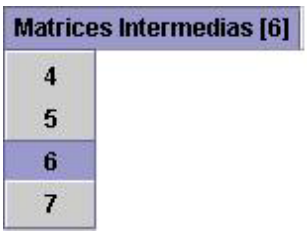

Figura 62: Menú de Matrices intermedias de la opción B

- **Datos y resultados**: Compuesto una caja de texto y de etiquetas.
  En la caja de texto se debe introducir el factor de actividad del programa. Este debe ser un número entre 0,1 y 15,9.
  En las etiquetas se tienen dos apartados:
  Etiquetas de conexiones, en las que se muestran las conexiones en curso, el máximo de conexiones que se han tenido en curso y las conexiones perdidas.
  Etiquetas de tráfico, en las que se muestra el Tráfico Cursado y el Tráfico Perdido

Los datos y resultados se muestran así:

Figura 63: Datos y resultados de la opción B

El programa consta de los siguientes botones:

- **Paso adelante**: Ejecuta, cada vez que se pulsa un nuevo evento.

  El aspecto del botón “Paso adelante” es:

Figura 64: Botón Paso adelante de la opción B

- **Ejecución**: Realiza eventos hasta que se produzca alguno de bloqueo.

  El aspecto del botón “ejecución” es:

Figura 65: Botón Ejecución de la opción B

- **1000 ejecuciones**: Realiza seguidos 1000 eventos que se verán reflejados en las "eventos" y "resultados" (Ver Caja de Texto "número de ejecuciones").

  El aspecto del botón "1000 ejecuciones" es:

Figura 66: Botón 1000 ejecuciones de la opción B

- **Reiniciar**: Limpia el diagrama de conexiones, los datos, los resultados, los eventos y retorna a su valor inicial al algoritmo elegido (A) y el número de matrices intermedias (4).

  El aspecto del botón "Reiniciar" es:

Figura 67: Botón Reiniciar de la opción B

Anexo B
Código Fuente de la aplicación

Código Fuente de la aplicación

B.1 Menú de Inicio

Archivo: C:\Documents and Settings\Gregorio\Escritorio\menu inicio\Simucom.java

```
/*
* Simucom.java
*/

import java.awt.*;
import javax.swing.*;
import java.io.File;

public class Simucom extends JFrame {

    /** Creates new form Simucom */
    public Simucom() {
        setTitle(" SIMUCOM");
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    private void initComponents() {
        escudo1 = new EscudoUPCT("upctrelieve.jpg");
        escudo2 = new EscudoTeleco("etsit.jpg");
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jLabel4 = new javax.swing.JLabel();
        label1 = new java.awt.Label();
        ImageIcon logo = new ImageIcon ("Imagenes" + File.separator + "upctrelieve.jpg");
        setIconImage(logo.getImage());


        getContentPane().setLayout(null);

        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });


        getContentPane().add(escudo1);
        escudo1.setBounds(0, 0, 130, 130);

        getContentPane().add(escudo2);
        escudo2.setBounds(130, 0, 130, 130);

        jLabel1.setText("             UNIVERSIDAD POLIT\u00c9CNICA DE CARTAGENA");
        jLabel1.setToolTipText("null");
        jLabel1.setBackground(new java.awt.Color(204, 204, 204));
        getContentPane().add(jLabel1);
        jLabel1.setBounds(260, 0, 340, 40);

        jLabel2.setText("               E.T.S de Ingenier\u00eda de Telecomunicaci\u00f3n  ");
        getContentPane().add(jLabel2);
        jLabel2.setBounds(260, 40, 340, 40);

        jLabel3.setText("                        \u00c1rea de Ingenier\u00eda Telem\u00e1tica");
        getContentPane().add(jLabel3);
        jLabel3.setBounds(261, 87, 340, 40);

        jButton1.setText("  OPCI\u00d3N A: Conmutador multietapa en modo simple");
                jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        getContentPane().add(jButton1);
        jButton1.setBounds(0, 220, 600, 30);

        jButton2.setText("      OPCI\u00d3N B: Conmutador multietapa en modo continuo");
                jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        getContentPane().add(jButton2);
        jButton2.setBounds(0, 250, 600, 27);
```

```
        jLabel4.setText("  DEMOSTRACI\u00d3N Y EJERCICIOS DE UN CONMUTADOR MULTIETAPA");
        jLabel4.setForeground(new java.awt.Color(51, 0, 255));
        jLabel4.setFont(new java.awt.Font("Dialog", 1, 14));
        getContentPane().add(jLabel4);
        jLabel4.setBounds(60, 170, 480, 30);

        label1.setFont(new java.awt.Font("Arial", 3, 12));
        label1.setBackground(new java.awt.Color(160, 167, 164));
        label1.setForeground(new java.awt.Color(51, 0, 51));
        label1.setText("Autor:  Gregorio Ca\u00f1avate Cruzado");
        label1.setAlignment(java.awt.Label.CENTER);
        getContentPane().add(label1);
        label1.setBounds(0, 330, 600, 60);

        pack();
    }

        private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
        {
                try
                {
                        // Obtienes un Runtime
                        Runtime r = Runtime.getRuntime();

                        // Le mandas ejecutar el comando
                        Process p = r.exec ("java -jar opcionA.jar"); /* No redirijas la salida
a un fichero, no hace falta.*/
                        System.exit(0);
                }
                catch(Exception e)
                {
                }

    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
    {

            try
            {

                // Obtienes un Runtime
                Runtime r = Runtime.getRuntime();

                // Le mandas ejecutar el comando
                Process p = r.exec ("java -jar opcionB.jar");
                System.exit(0);
            }
            catch(Exception e)
            {
            }


    }


    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt) {
        System.exit(0);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        Simucom = new Simucom();
                Simucom.setSize(608, 410);
                Simucom.setResizable(false);
                Simucom.show();
    }


    // Variables declaration - do not modify
    private EscudoTeleco escudo2;
    private EscudoUPCT escudo1;
    private javax.swing.JButton jButton2;
    private java.awt.Label label1;
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel1;
```

```
        private static Simucom Simucom;
    // End of variables declaration

}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\menu inicio\EscudoUPCT.java

```java
/*
* EscudoUPCT.java
*/
import java.awt.*;
import javax.swing.*;
import java.io.File;

public class EscudoUPCT extends Canvas
  {
    Image img;

    EscudoUPCT(String fichero)
    {
      img = getToolkit().getImage("Imagenes" + File.separator + fichero);
    }

        public void paint(Graphics g)
    {
      // Tamaño ajustable al tamaño del canvas
      g.drawImage(img, 0, 0,
      this.getSize().width , this.getSize().height, this);
    }

  }
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\menu inicio\EscudoTeleco.java

```java
/*
* EscudoTeleco.java
*/
import java.awt.*;
import javax.swing.*;
import java.io.File;

public class EscudoTeleco extends Canvas
  {
    Image img;

    EscudoTeleco(String fichero)
    {
          img = getToolkit().getImage("Imagenes" + File.separator + fichero);
    }


    public void paint(Graphics g)
    {
      // Tamaño ajustable al tamaño del canvas
      g.drawImage(img, 0, 0,
      this.getSize().width , this.getSize().height, this);
    }


  }
```

Código Fuente de la aplicación

B.2 Opción A

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionA\Entrada.java

```java
/*
 * Entrada.java
 */

public class Entrada
{
	private int id, bloque;

	/*
	 * Definición: Constructor
	 *
	 * Parámetros de entrada: int id: número de entrada
	 *
	 * Devuelve: -
	*/
	public Entrada(int id)
	{
		this.id = id;
		bloque = (id - 1) / 4;
	}

	/*
	 * Definición: Devuelve la entrada
	 *
	 * Parámetros de entrada: -
	 *
	 * Devuelve: int id: la entrada
	*/
	public int getId()
	{
		return id;
	}

	/*
	 * Definición: Devuelve el bloque de entrada
	 *
	 * Parámetros de entrada: -
	 *
	 * Devuelve: int bloque: bloque de entrada
	*/
	public int getBloque()//devuelve el bloque de entrada
	{
		return bloque;
	}

}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionA\Salida.java

```java
/*
* Salida.java
*/
public class Salida
{
	private int id, bloque;

	/*
	 * Definición: Constructor
	 *
	 * Parámetros de entrada: int id: número de salida
	 *
	 * Devuelve: -
	*/
	public Salida(int id)
	{
		this.id = id;
		bloque = (id - 1) / 4;
	}

	 /*
	 * Definición: Devuelve la salida
	 *
	 * Parámetros de entrada: -
	 *
	 * Devuelve: int id: la salida
	*/public int getId()
	{
		return id;
	}

	/*
	 * Definición: Devuelve el bloque de salida
	 *
	 * Parámetros de entrada: -
	 *
	 * Devuelve: int bloque: bloque de salida
	*/
	public int getBloque()
	{
		return bloque;
	}
}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionA\Conexiones.java

```
/*
 * Conexiones.java
 */

public class Conexiones
{
	private Entrada in;
	private Salida out;
	private int MI;/*matriz intermedia por la que pasa la conexión*/

	/*
	 * Definición: Constructor
	 *
	 * Parámetros de entrada:
	 * Entrada in: contiene el número de entrada
	 * Salida out: contiene el número de salida
	 * Devuelve: -
	*/
	public Conexiones(Entrada in, Salida out)
	{
		this.in = in;
		this.out = out;
	}

	/*
	 * Definición: Introduce la MI
	 *
	 * Parámetros de entrada: int MI
	 *
	 * Devuelve: -
	*/
	public void setMI(int MI)
	{
		this.MI = MI;
	}

	/*
	 * Definición: Obtiene el objeto de entrada
	 *
	 * Parámetros de entrada: -
	 *
	 * Devuelve: Entrada in.
	*/
	public Entrada getEntrada()//
	{
		return in;
	}

	/*
	 * Definición: Obtiene el objeto de salida
	 *
	 * Parámetros de entrada: -
	 *
	 * Devuelve: Salida out.
	*/
	public Salida getSalida()
	{
		return out;
	}

	/*
	 * Definición: Obtiene la MI
	 *
	 * Parámetros de entrada: -
	 *
	 * Devuelve: int MI
	*/
	public int getMI()//obtenemos la MI
	{
		return MI;
	}

}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionA\MatrizIntermedia.java

```
/*
 * MatrizIntermedia.java
 */
import java.util.Vector;

public class MatrizIntermedia {
  private int id;
  private Vector in, out;

        /*
         * Definición: Constructor
         *
         * Parámetros de entrada: int id: la MI
         *
         * Devuelve: -
        */
    public MatrizIntermedia(int id) {
    in = new Vector();
    out = new Vector();
    this.id = id;
  }

        /*
         * Definición: Añadimos el bloque de entrada al vector entradas
         *
         * Parámetros de entrada: int bloque: Matriz de entrada
         *
         * Devuelve: -
        */
   public void introduceBloqueIn(int bloque) {
    in.add(new Integer(bloque));
  }

        /*
         * Definición: Añadimos el bloque de salida al vector entrasalidadas
         *
         * Parámetros de entrada: int bloque: Matriz de salida
         *
         * Devuelve: -
        */
   public void introduceBloqueOut(int bloque) {
    out.add(new Integer(bloque));
  }

        /*
         * Definición: // indica si el valor introducido estaba ya almacenado en el vector
         * de entradas, devuelve false si se puede utilizar (no se utilizó la MI
         * por un mismo bloque 2 veces)
         *
         * Parámetros de entrada: int valor: bloque de entrada a comprobar
         *
         * Devuelve: boolean
        */
  public boolean utilizaIn(int valor)
  {
    for (int i = 0; i < in.size(); i++) {
      int e = ( (Integer) in.elementAt(i)).intValue();
      if (valor == ( (Integer) in.elementAt(i)).intValue()) {
        return true;
      }
    }
    return false;
  }

        /*
         * Definición: // indica si el valor introducido estaba ya almacenado en el vector
         * de salidas, devuelve false si se puede utilizar (no se utilizó la MI
         * por un mismo bloque 2 veces)
         *
         * Parámetros de entrada: int valor: bloque de salida a comprobar
         *
         * Devuelve: boolean
        */
  public boolean utilizaOut(int valor)
  {
    for (int i = 0; i < out.size(); i++) {
      int s = ( (Integer) out.elementAt(i)).intValue();
      if (valor == ( (Integer) out.elementAt(i)).intValue()) {
        return true;
      }
    }
    return false;
  }
```

```
}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionA\Algoritmo.java

```
/*
 * Algoritmo.java
 */

import java.util.Vector;

public class Algoritmo
{
	public  Vector conexiones;/*queremos leer esta variable cuando dibujemos*/
	private MatrizIntermedia a, b, c, d;/*las 4 matrices intermedias de nuestro programa*/
	private boolean BloqueinVisto = false; /*Con esta variable nos aseguramos que se mire el
bloque de entrada de cada M Intermedia*/
	private boolean BloqueoutVisto = false; /*Con esta variable nos aseguramos que se mire
el bloque de entrada de cada M Intermedia*/

	/*
	 * Constructor:  se crea  un objeto del tipo Vector,
	 * que se llama conexiones y que contendrá objetos del tipo Conexiones.
	 * También creamos 4 objetos del tipo MatrizIntermedia: a, b, c, d.
	 *
	 * Parámetros de entrada: -
	 *
	 */

	public Algoritmo()
	{
	conexiones = new Vector();
	a = new MatrizIntermedia(0);
	b = new MatrizIntermedia(1);
	c = new MatrizIntermedia(2);
	d = new MatrizIntermedia(3);
	}

	/*
	 * Definición: Rellena el vector conexiones con objetos del tipo Conexiones
	 * que constarán de entradas y salidas del 1 al 16, distribuidas al azar y
	 * procurando que nos se repitan ninguna entrada y ninguna salida.
	 * Parámetros de entrada: -
	 *
	 * Devuelve: -
	 */
	public void generaConexiones()/*este método crea un vector de Conexiones con entradas y
salidas del 1 al 16 //desordenadas, procurando que nos se repitan tanto entradas como salidas*/
	{
		conexiones.removeAllElements();/*inicializamos el vector*/
		int con = 0;

		while (con != 16)
		{
		int numero = ((int) (Math.random() * 16)) + 1;
		int sumador = 0;
		for(int a=0;a<conexiones.size();a++)
		{
			if (numero !=
(((Conexiones)conexiones.elementAt(a)).getSalida()).getId())
			{
				sumador ++;
			}
			else
			{
			break;
			}
		}

		if (sumador == conexiones.size())
		{
			conexiones.addElement(new Conexiones(new Entrada(con + 1), new
Salida(numero)));
			con ++;
		}
		}
		desordenar();

	}

	/*
	 * Definición: Desordena el vector de conexiones.
	 *
	 * Parámetros de entrada: -
	 *
	 * Devuelve: -
	*/
	private void desordenar()
```

```
	{
		for(int a=0; a<50; a++)
		{
		int numero = ((int) (Math.random() * 16));
		Conexiones con = (Conexiones) conexiones.elementAt(numero);
		conexiones.add(con);
		conexiones.remove(numero);
		}
	}


	/*
	 * Definición: Algoritmo de Rotación intenta primero por la MI 0, luego la 1, ...
	 * Cuando no pueda conectar por una MI lo intenta con la siguiente. Se conectará o
bloqueará.
	 * Parámetros de entrada: -
	 *
	 * Devuelve: int
	*/
	public int resuelveAlgoritmoA()
	{
		int puntero = 0;
		int indice = 0;
		int ok1=0,ok2=0;
	while (indice < 16)
		{
		/*cogemos elemento del vector*/
		Conexiones con = (Conexiones) conexiones.elementAt(indice);

		/*miramos en que bloque de entrada esta*/
		int bloqueIn = (con.getEntrada()).getBloque();

		/*miramos en que bloque de salida le toca*/
		int bloqueOut = (con.getSalida()).getBloque();

		/*miramos en que bloque intermedio le toca (puntero % 4)*/

				for (int a=0;a<indice;a++)
				{

					if (bloqueIn ==
(((Conexiones)conexiones.elementAt(a)).getEntrada()).getBloque())//comprobamos si coincide algun
bloque de entrada de los anteriores con el actual*/
					{
						while ((ok1 + ok2) < 4)
						{
							if (compruebaMI(puntero % 4,
bloqueIn))/*miramos que el bloque de entrada pertenece ala MI actual
							{
							/*No se puede por este bloque de entrada
ir por la MI puntero%4*/
							ok1 ++;
							puntero ++;
							a = 0;/*con a=0 comparamos el nuevo
puntero con la E desde el principio*/
							BloqueinVisto = true;
							}
							else
							{
								break;
							}

						}
					if ((ok1==4)||((ok1 + ok2) >=4))// bloqueo
						{
							DepurarConexiones(indice);
							return -1;
						}
					}

					if (bloqueOut ==
(((Conexiones)conexiones.elementAt(a)).getSalida()).getBloque())
					{

						while ((ok1 + ok2) < 4)
						{
							if (compruebaMO(puntero % 4, bloqueOut))
							{
							/*No se puede por este bloque de salida
ir por la MI puntero%4*/
							ok2 ++;
							puntero ++;
							a =0;
```

```
                                                        BloqueoutVisto = true;
                                                        }
                                                        else
                                                        {
                                                                break;
                                                        }
                                                }/*al salir de este bucle while tenemos ok con
un valor, si es =4 tendremos un bloqueo*/
                                                if ((ok2==4)||((ok1 + ok2) >=4))  //bloqueo
                                                {
                                                        DepurarConexiones(indice);
                                                        return -1;
                                                }

                                        } //if (bloqueOut)

                                        if ((BloqueinVisto == false)  && (BloqueoutVisto ==
true))
                                        /*con esto hacemos que si hemos incrementado el ptero y
no ha mirado la entrada que la mire*/
                                        {
                                                if (bloqueIn ==
(((Conexiones)conexiones.elementAt(a)).getEntrada()).getBloque())
                                                {
                                                        while ((ok1 + ok2) < 4)
                                                        {
                                                                if (compruebaMI(puntero % 4,
;rueueou))
                                                                {
                                                                /*No se puede por este bloque de
entrada ir por la MI puntero%4*/
                                                                ok1 ++;
                                                                puntero ++;
                                                                a = 0;
                                                                BloqueinVisto = true;
                                                                }
                                                                else
                                                                {
                                                                        break;
                                                                }
                                                        }

                                                if ((ok1==4)||((ok1 + ok2) >=4))//bloqueo
                                                        {
                                                                DepurarConexiones(indice);
                                                                return -1;
                                                        }
                                                }
                                        }/*fin if (bloquevisto == false)*/

                                } /*for (int a=0;a<indice;a++)*/
                                /*introducimos en una MI la entrada y salida asignada*/
                                seleccionaMI(puntero % 4, bloqueIn, bloqueOut);
                                con.setMI(puntero % 4);/*añadimos la MI q corresponde a la
conexion actual*/
                                indice ++;
                puntero ++;/*algoritmo rotacion, aumentamos MI*/
                ok1=0;
                ok2=0;
                BloqueinVisto = false;
                                BloqueoutVisto = false;

                } //while (indice<16)
                return 0;
        }

        /*
         * Definición: Algoritmo secuencial, se intenta conectar siempre por la matriz 0.
         * Cuando no pueda conectar por una MI lo intenta con la siguiente. Se conectará o
bloqueará.
         * Parámetros de entrada: -
         *
         * Devuelve: int
        */

        public int resuelveAlgoritmoB()
        {
                int puntero = 0;
                int indice = 0;
                int ok1=0,ok2=0;
        while (indice < 16)
                {
                /*cogemos elemento del vector*/
```

```
                Conexiones con = (Conexiones) conexiones.elementAt(indice);

                /*miramos en que bloque de entrada esta*/
                int bloqueIn = (con.getEntrada()).getBloque();

                /*miramos que bloquee de salida le toca*/
                int bloqueOut = (con.getSalida()).getBloque();

                /*miramos en que bloque intermedio le toca (puntero%4)*/


                                for (int a=0;a<indice;a++)
                                {

                                        if (bloqueIn ==
(((Conexiones)conexiones.elementAt(a)).getEntrada()).getBloque())
                                        {
                                                while ((ok1 + ok2) < 4)
                                                {
                                                        if (compruebaMI(puntero % 4, bloqueIn))
                                                        {
                                                        */No se puede por este bloque de entrada
//
ir por la MI puntero%4*/
                                                        ok1 ++;
                                                        puntero ++;
                                                        a = 0;  /*con a=0 comparamos el nuevo
puntero con la E desde el principio*/
                                                        BloqueinVisto = true;
                                                        }
                                                        else
                                                        {
                                                                break;
                                                        }

                                                }
                                                if ((ok1==4)||((ok1 + ok2) >=4))//bloqueo
                                                {
                                                        DepurarConexiones(indice);
                                                        return -1;
                                                }
                                        }

                                        if (bloqueOut ==
(((Conexiones)conexiones.elementAt(a)).getSalida()).getBloque())
                                        {

                                                while ((ok1 + ok2) < 4)
                                                {
                                                        if (compruebaMO(puntero % 4, bloqueOut))
                                                        {
                                                        /*No se puede por este bloque de salida
//
ir por la MI puntero%4*/
                                                        ok2 ++;
                                                        puntero ++;
                                                        a = 0;  /*con a=0 comparamos el nuevo
puntero con la S desde el principio*/
                                                        BloqueoutVisto = true;
                                                        }
                                                        else
                                                        {
                                                                break;
                                                        }
                                                }/*al salir de este bucle while tenemos ok con
un valor, si es =4 tendremos un bloqueo*/
                                                if ((ok2==4)||((ok1 + ok2) >=4))  // bloqueo
                                                {
                                                        DepurarConexiones(indice);
                                                        return -1;
                                                }

                                        } //if (bloqueOut)

                                        if ((BloqueinVisto == false)&&(BloqueoutVisto == true))
                                        /*con esto hacemos que si hemos incrementado el ptero y
no ha mirado la entrada que la mire*/
                                        {
                                                if (bloqueIn ==
(((Conexiones)conexiones.elementAt(a)).getEntrada()).getBloque())
                                                {
                                                        while ((ok1 + ok2) < 4)
                                                        {
                                                                if (compruebaMI(puntero % 4,
bloqueIn))
```

```
                                                                {
{*                                                              /*No se puede por este bloque de
entrada ir por la MI puntero%4*/
                                                                ok1 ++;
                                                                puntero ++;
                                                                a = 0;
                                                                BloqueinVisto = true;
                                                                }
                                                                else
                                                                {
                                                                        break;
                                                                }

                                                        }
                                                if ((ok1==4)||((ok1 + ok2) >=4))//bloqueo
                                                        {
                                                                DepurarConexiones(indice);
                                                                return -1;
                                                        }
                                                }
                                        }/*fin if (bloquevisto == false)*/
                                } /*for (int a=0;a<indice;a++)*/
                                /*introducimos en una MI la entrada y salida asignada*/
                                seleccionaMI(puntero % 4, bloqueIn, bloqueOut);
                                con.setMI(puntero % 4);/*añadimos la MI q corresponde a la
conexion actual*/
                                indice ++;
                puntero = 0;//Algoritmo secuencial
                ok1=0;
                ok2=0;
                BloqueinVisto = false;
                                BloqueoutVisto = false;

                } //while (indice<16)
                return 0;
        }

        /*
         * Definición: Algoritmo aleatorio.
         * Cuando no pueda conectar por una MI lo intenta con la siguiente. Se conectará o
bloqueará.
         * Parámetros de entrada: -
         *
         * Devuelve: Int
        */
        public int resuelveAlgoritmoC()
        {
                int puntero = 0;
                int indice = 0;
                int ok1=0,ok2=0;
        while (indice < 16)
                {
                //cogemos elemento del vector
                Conexiones con = (Conexiones) conexiones.elementAt(indice);

                //miramos en que bloque de entrada esta
                int bloqueIn = (con.getEntrada()).getBloque();

                //miramos que bloque de salida le toca
                int bloqueOut = (con.getSalida()).getBloque();

                puntero = ((int) (Math.random() * 16)) + 1;

                /*miramos en que bloquee intermedio le toca (puntero %4)*/


                                for (int a=0;a<indice;a++)
                                {

                                        if (bloqueIn ==
(((Conexiones)conexiones.elementAt(a)).getEntrada()).getBloque())
                                        {
                                                while ((ok1 + ok2) < 4)
                                                {
                                                        if (compruebaMI(puntero % 4, bloqueIn))
                                                        {
                                                        /*No se puede por este bloque de entrada
ir por la MI puntero%4*/
                                                        ok1 ++;
                                                        puntero ++;
                                                        a =0;
                                                        BloqueinVisto = true;
                                                        }
```

```
                                                        else
                                                        {
                                                                break;
                                                        }

                                                }
                                        if ((ok1==4)||((ok1 + ok2) >=4))//bloqueo
                                                {
                                                        DepurarConexiones(indice);
                                                        return -1;
                                                }
                                        }

                                        if (bloqueOut ==
(((Conexiones)conexiones.elementAt(a)).getSalida()).getBloque())
                                        {

                                                while ((ok1 + ok2) < 4)
                                                {
                                                        if (compruebaMO(puntero % 4, bloqueOut))
                                                        {
//                                                      /*No se puede por este bloque de salida
elsek                   b
                                                        ok2 ++;
                                                        puntero ++;
                                                        a =0;
                                                        BloqueoutVisto = true;
                                                        }
                                                        else
                                                        {
                                                                break;
                                                        }
                                                }/*al salir de este bucle while tenemos ok con
un valor, si es =4 tendremos un bloqueo*/
                                                if ((ok2==4)||((ok1 + ok2) >=4)) // bloqueo
                                                {
                                                        DepurarConexiones(indice);
                                                        return -1;
                                                }

                                        } //if (bloqueOut)

                                        if ((BloqueinVisto == false)&&(BloqueoutVisto == true))
                                        /*con esto hacemos que si hemos incrementado el ptero y
no ha mirado la entrada que la mire*/
                                        {
                                                if (bloqueIn ==
(((Conexiones)conexiones.elementAt(a)).getEntrada()).getBloque())
                                                {
                                                        while ((ok1 + ok2) < 4)
                                                        {
                                                                if (compruebaMI(puntero % 4,
bloqueIn))

                                                                {
//                                                              /*No se puede por este bloque de
entrada ir por la MI puntero%4 */
                                                                ok1 ++;
                                                                puntero ++;
                                                                a = 0;
                                                                BloqueinVisto = true;
                                                                }
                                                                else
                                                                {
                                                                        break;
                                                                }

                                                        }
                                                if ((ok1==4)||((ok1 + ok2) >=4))
                                                        {
                                                                DepurarConexiones(indice);
                                                                return -1;
                                                        }
                                                }
                                        }//fin if (bloquevisto == false)

                                } //for (int a=0;a<indice;a++)
                                /*introducimos en una MI la entrada y salida asignada*/
                                seleccionaMI(puntero % 4, bloqueIn, bloqueOut);
                                con.setMI(puntero % 4);/*añadimos la MI q corresponde a la
conexion actual*/
                                indice ++;
                        puntero = ((int) (Math.random() * 16)) + 1;//Algoritmo aleatorio
                        ok1=0;
```

```
                ok2=0;
                BloqueinVisto = false;
                BloqueoutVisto = false;

                } //while (indice<16)
                return 0;
        }

        /*
         * Definición: Introduce en una MI la entrada y salida asignada
         *
         * Parámetros de entrada:
         * int valor: MI
         * int entrada: bloque entrada
         * int salida: bloque salida
         * Devuelve: -
        */
        private void seleccionaMI(int valor, int entrada, int salida)
        {
                switch (valor)
                {
                case 0:
                        a.introduceBloqueIn(entrada);
                        a.introduceBloqueOut(salida);
                        break;

                case 1:
                        b.introduceBloqueIn(entrada);
                        b.introduceBloqueOut(salida);
                        break;

                case 2:
                        c.introduceBloqueIn(entrada);
                        c.introduceBloqueOut(salida);
                        break;

                case 3:
                        d.introduceBloqueIn(entrada);
                        d.introduceBloqueOut(salida);
                        break;
                }
        }

        /*
         * Definición: Comprueba si esta la entrada modulo en la MI valor
         *
         * Parámetros de entrada:
         * int valor : MI
         * int modulo : entrada
         * Devuelve: boolean
        */
        private boolean compruebaMI(int valor, int modulo)
        {
                switch (valor)
                {
                case 0:
                        return a.utilizaIn(modulo);

                case 1:
                        return b.utilizaIn(modulo);

                case 2:
                        return c.utilizaIn(modulo);

                case 3:
                        return d.utilizaIn(modulo);

                default:
                        return false;
                }
        }

        /*
         * Definición: Comprueba si esta la salida modulo en la MI valor
         *
         * Parámetros de entrada:
         * int valor : MI
         * int modulo : salida
         * Devuelve: boolean
        */
        private boolean compruebaMO(int valor, int modulo)
        {
                switch (valor)
```

```
                {
                case 0:
                        return a.utilizaOut(modulo);

                case 1:
                        return b.utilizaOut(modulo);

                case 2:
                        return c.utilizaOut(modulo);

                case 3:
                        return d.utilizaOut(modulo);

                default:
                        return false;
                }
        }

        /*
         * Definición: Devuelve el vector de conexiones
         *
         * Parámetros de entrada: -
         *
         * Devuelve: vector de conexiones
        */
        public Vector getConexiones()
        {
                return conexiones;
        }

        /*
         * Definición: Escogemos algoritmo
         *
         * Parámetros de entrada: int Algoritm
         *
         * Devuelve: -
        */
        public void AlgoritmoEscogido(int Algoritm)
        {
                switch (Algoritm){
                case 0:
                        {
                                resuelveAlgoritmoA();
                                break;
                        }
                case 1:
                        {
                                resuelveAlgoritmoB();
                                break;
                        }
                case 2:
                        {
                                resuelveAlgoritmoC();
                                break;
                        }
                        default: System.out.println("Error:");
                }
        }

        /*
         * Definición: Con este metodo queremos que el vector conexiones solo
         * tenga las conexiones que han sido realizadas
         *
         * Parámetros de entrada:
         *int indiceMax: corresponde a la conexion bloqueada
         *
         * Devuelve: -
        */
        private void DepurarConexiones(int indiceMax)
        {
                for (int i = indiceMax + 1; i< 16; i++)
                {
                conexiones.removeElementAt(indiceMax + 1);/*indiceMax corresponde a la conexion
bloqueada, con lo q la estamos respetando*/
                }
                /*conexiones.removeRange(indiceMax, 16);//borramos desde...
hasta...(incluidos)*/
        }

}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionA\Imagen.java

```
/*
 * Imagen.java
 *
 * Created on 12 de mayo de 2003, 11:20
 */

/**
 *
 * @author  Gregorio
 Con esta clase añado a un JPanel una imagen(fichero jpg)
 y la encuadro en un rectángulo
 */
//package opcionA;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;
import java.util.Vector;
import java.io.File;

public class Imagen extends JPanel {
  private TexturePaint imagen1; //una imagen dentro de un rectangulo o figura geom
  private Rectangle rectangulo;
  private Vector con;
  private int contador;
  private String ruta;

        /*
         * Definición: Constructor
         *
         * Parámetros de entrada:
         * String:ruta del directorio donde se encuentra la imagen
         * BufferedImage imagen
         * Rectangle rectangulo: para recuadrar imagen
         * TexturePaint imagen1
         * Devuelve: -
        */
     public Imagen() {
    ruta = new String("Imagenes" + File.separator + "sincrossbar.jpg");
    BufferedImage imagen = getBufferedImage(ruta, this);
    rectangulo = //Recuadro para encuadrar el dibujo
        new Rectangle(10, 10, imagen.getWidth(), imagen.getHeight());
    imagen1 = new TexturePaint(imagen, rectangulo);
  }


        /*
         * Definición: Método para crear un objeto Image a partir de un fichero, que
         * se guarda en un buffer
         *
         * Parámetros de entrada:
         * String fichero
         * Component c
         * Devuelve: -
        */
     public static BufferedImage getBufferedImage(
      String fichero, Component c) {
    Image imagen = c.getToolkit().getImage(fichero);

    cargaImagen(imagen, c);
    BufferedImage buffImagen =
        new BufferedImage(imagen.getWidth(c), imagen.getHeight(c),
                          BufferedImage.TYPE_INT_RGB);
    Graphics2D g2 = buffImagen.createGraphics();
    g2.drawImage(imagen, 0, 0, c);

    return (buffImagen);
  }


        /*
         * Definición: Este método toma un objeto Image asociado a un fichero y espera
         * hasta que la carga se haya completado. Éste es el uso más simple
         * de MediaTracker, si se van a cargar varias imágenes, no debe
         * usarse este método en sucesivas llamadas, sino que es mucho
         * mejor el uso de un array de imágenes
         *
         * Parámetros de entrada:
         * Image imagen
         * Component c
         * Devuelve: boolean
```

```
        */
     public static boolean cargaImagen(Image imagen, Component c) {
    MediaTracker tracker = new MediaTracker(c);
    tracker.addImage(imagen, 0);
    try {
      tracker.waitForAll();
    }
    catch (InterruptedException e) {
      e.printStackTrace();
    }
    return (!tracker.isErrorAny());
  }

        /*
         * Definición: Dibuja la imagen básica
         *
         * Parámetros de entrada: Graphics g
         *
         * Devuelve: -
        */
   public void paintComponent(Graphics g) {
    super.paintComponent(g); //Graphics2D extiende d Graphics
    Graphics2D g2 = (Graphics2D) g;
    g2.setPaint(Color.blue);
    g2.setStroke(new BasicStroke(5));
    g2.setPaint(imagen1);
    g2.fill(rectangulo);
    g2.setPaint(Color.black);
    g2.draw(rectangulo);
    g2.setStroke(new BasicStroke(1.8f));
    LineasFondo(g2);
    g2.setPaint(Color.black);
    NumerosFondo(g2);
  }

        /*
         * Definición: Va dibujando las líneas
         *
         * Parámetros de entrada: Graphics2D g2
         *
         * Devuelve: -
        */
    public void dibuja(Graphics2D g2) {
    g2.setStroke(new BasicStroke(1.8f)); //grosor de linea 1.8
    if (contador == 16) {//solo para que pinte todo si no hay bloqueo,
        // esto se puede hacer por q nunca se bloquea en el 16
      PintaLinea(g2);
    }

    if (contador == con.size()) {
      //para que no pinte la conexion bloqueada
      contador--;
      PintaLinea(g2);
    }

    if (contador < con.size()) {
      PintaLinea(g2);
    }
    else {
      /*No hay mas conexiones hechas*/
    }
  }

        /*
         * Definición: Dibuja los números de fondo
         *
         * Parámetros de entrada: Graphics2D g2
         *
         * Devuelve: -
        */
    private void NumerosFondo(Graphics2D g2) {
    //números de entrada
    g2.drawString("1", 35, 65);
    g2.drawString("2", 35, 82);
    g2.drawString("3", 35, 99);
    g2.drawString("4", 35, 116);
    g2.drawString("5", 35, 154);
    g2.drawString("6", 35, 171);
    g2.drawString("7", 35, 188);
    g2.drawString("8", 35, 205);
    g2.drawString("9", 35, 243);
    g2.drawString("10", 35, 260);
    g2.drawString("11", 35, 277);
```

```
    g2.drawString("12", 35, 294);
    g2.drawString("13", 35, 332);
    g2.drawString("14", 35, 349);
    g2.drawString("15", 35, 366);
    g2.drawString("16", 35, 383);
    //números de salida
    g2.drawString("1", 665, 65);
    g2.drawString("2", 665, 82);
    g2.drawString("3", 665, 99);
    g2.drawString("4", 665, 116);
    g2.drawString("5", 665, 154);
    g2.drawString("6", 665, 171);
    g2.drawString("7", 665, 188);
    g2.drawString("8", 665, 205);
    g2.drawString("9", 665, 243);
    g2.drawString("10", 665, 260);
    g2.drawString("11", 665, 277);
    g2.drawString("12", 665, 294);
    g2.drawString("13", 665, 332);
    g2.drawString("14", 665, 349);
    g2.drawString("15", 665, 366);
    g2.drawString("16", 665, 383);

    g2.drawString("ETAPA DE ENTRADA", 67, 35);
    g2.drawString("ETAPA DE DISTRIBUCIÓN", 282, 35);
    g2.drawString("ETAPA DE SALIDA", 540, 35);
  }

  //dibujamos las lineas grises de fondo
        /*
         * Definición: Dibuja las líneas de fondo
         *
         * Parámetros de entrada: Graphics2D g2
         *
         * Devuelve: -
        */
    private void LineasFondo(Graphics2D g2) {
    g2.setPaint(Color.lightGray);

    g2.draw(new Line2D.Double(170, 62, 304, 62));
    g2.draw(new Line2D.Double(170, 79, 304, 151));
    g2.draw(new Line2D.Double(170, 96, 304, 240));
    g2.draw(new Line2D.Double(170, 113, 304, 329));

    g2.draw(new Line2D.Double(170, 151, 304, 79));
    g2.draw(new Line2D.Double(170, 168, 304, 168));
    g2.draw(new Line2D.Double(170, 185, 304, 257));
    g2.draw(new Line2D.Double(170, 202, 304, 346));

    g2.draw(new Line2D.Double(170, 240, 304, 96));
    g2.draw(new Line2D.Double(170, 257, 304, 185));
    g2.draw(new Line2D.Double(170, 274, 304, 274));
    g2.draw(new Line2D.Double(170, 291, 304, 363));

    g2.draw(new Line2D.Double(170, 329, 304, 113));
    g2.draw(new Line2D.Double(170, 346, 304, 202));
    g2.draw(new Line2D.Double(170, 363, 304, 291));
    g2.draw(new Line2D.Double(170, 380, 304, 380));

    g2.draw(new Line2D.Double(406, 62, 540, 62));
    g2.draw(new Line2D.Double(406, 151, 540, 79));
    g2.draw(new Line2D.Double(406, 240, 540, 96));
    g2.draw(new Line2D.Double(406, 329, 540, 113));

    g2.draw(new Line2D.Double(406, 79, 540, 151));
    g2.draw(new Line2D.Double(406, 168, 540, 168));
    g2.draw(new Line2D.Double(406, 257, 540, 185));
    g2.draw(new Line2D.Double(406, 346, 540, 202));

    g2.draw(new Line2D.Double(406, 96, 540, 240));
    g2.draw(new Line2D.Double(406, 185, 540, 257));
    g2.draw(new Line2D.Double(406, 274, 540, 274));
    g2.draw(new Line2D.Double(406, 363, 540, 291));

    g2.draw(new Line2D.Double(406, 113, 540, 329));
    g2.draw(new Line2D.Double(406, 202, 540, 346));
    g2.draw(new Line2D.Double(406, 291, 540, 363));
    g2.draw(new Line2D.Double(406, 380, 540, 380));
  }

//para pintar la lineas que representan las conexiones
        /*
         * Definición: Pinta las líneas conectadas
```

```
         *
         * Parámetros de entrada: Graphics2D g2
         *
         * Devuelve: -
         */
    public void PintaLinea(Graphics2D g2) {
    for (int i = 0; i < contador; i++) {
      Conexiones conx = (Conexiones) con.elementAt(i);
      int IdIn = (conx.getEntrada()).getId();
      int bloqueIn = (conx.getEntrada()).getBloque();
      int IdOut = (conx.getSalida()).getId();
      int bloqueOut = (conx.getSalida()).getBloque();
      int Matriz = conx.getMI();
      g2.setPaint(Color.blue);

      switch (bloqueIn) {
        case 0: {
          g2.draw(new Line2D.Double(170, 62 + 17 * Matriz, 304,
                                    62 + 89 * Matriz));
          g2.draw(new Line2D.Double(90, 62 + 17 * ( (IdIn - 1) % 4), 150,
                                    62 + 17 * Matriz));
          break;
        }

        case 1: {
          g2.draw(new Line2D.Double(170, 151 + 17 * Matriz, 304,
                                    79 + 89 * Matriz));
          g2.draw(new Line2D.Double(90, 151 + 17 * ( (IdIn - 1) % 4), 150,
                                    151 + 17 * Matriz));
          break;
        }

        case 2: {
          g2.draw(new Line2D.Double(170, 240 + 17 * Matriz, 304,
                                    96 + 89 * Matriz));
          g2.draw(new Line2D.Double(90, 240 + 17 * ( (IdIn - 1) % 4), 150,
                                    240 + 17 * Matriz));
          break;
        }

        case 3: {
          g2.draw(new Line2D.Double(170, 329 + 17 * Matriz, 304,
                                    113 + 89 * Matriz));
          g2.draw(new Line2D.Double(90, 329 + 17 * ( (IdIn - 1) % 4), 150,
                                    329 + 17 * Matriz));
          break;
        }

        default:
          System.out.println("Error:");
      }

      switch (Matriz) {
        case 0: {
          g2.draw(new Line2D.Double(325, 62 + 17 * bloqueIn, 385,
                                    62 + 17 * bloqueOut));
          break;
        }

        case 1: {
          g2.draw(new Line2D.Double(325, 151 + 17 * bloqueIn, 385,
                                    151 + 17 * bloqueOut));
          break;
        }

        case 2: {
          g2.draw(new Line2D.Double(325, 240 + 17 * bloqueIn, 385,
                                    240 + 17 * bloqueOut));
          break;
        }

        case 3: {
          g2.draw(new Line2D.Double(325, 329 + 17 * bloqueIn, 385,
                                    329 + 17 * bloqueOut));
          break;
        }

        default:
          System.out.println("Error:");
      }

      switch (bloqueOut) {
        case 0: {
```

```
          g2.draw(new Line2D.Double(406, 62 + 89 * Matriz, 540,
                                    62 + 17 * Matriz));
          g2.draw(new Line2D.Double(560, 62 + 17 * Matriz, 620,
                                    62 + 17 * ( (IdOut - 1) % 4)));
          break;
        }

        case 1: {
          g2.draw(new Line2D.Double(406, 79 + 89 * Matriz, 540,
                                    151 + 17 * Matriz));
          g2.draw(new Line2D.Double(560, 151 + 17 * Matriz, 620,
                                    151 + 17 * ( (IdOut - 1) % 4)));
          break;
        }

        case 2: {
          g2.draw(new Line2D.Double(406, 96 + 89 * Matriz, 540,
                                    240 + 17 * Matriz));
          g2.draw(new Line2D.Double(560, 240 + 17 * Matriz, 620,
                                    240 + 17 * ( (IdOut - 1) % 4)));
          break;
        }

        case 3: {
          g2.draw(new Line2D.Double(406, 113 + 89 * Matriz, 540,
                                    329 + 17 * Matriz));
          g2.draw(new Line2D.Double(560, 329 + 17 * Matriz, 620,
                                    329 + 17 * ( (IdOut - 1) % 4)));
          break;
        }

        default:
          System.out.println("Error:");

      } //end switch
    } // end for

  }

        /*
         * Definición: Introduce el vector de conexiones
         *
         * Parámetros de entrada: Vector vectorconexiones
         *
         * Devuelve: -
         */
    public void setConexiones(Vector vectorconexiones) {
    con = vectorconexiones;
  }

        /*
         * Definición: Nos pasa el contador
         *
         * Parámetros de entrada: int id: número de entrada
         *
         * Devuelve: -
         */
    public void setContador(int cont) {
    contador = cont;
  }

}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionA\opcionA.java

```
/*
* opcionA.java;
*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.JFrame.*;
import java.awt.image.*;
import java.util.Vector;
import java.awt.geom.*;
import java.lang.String;

public class opcionA
    extends javax.swing.JDialog {

  public opcionA() {

    setResizable(false);
    setTitle(" MATSWIT OPCIÓN A");
    initComponents();
  }

  private void initComponents() {
    jButton1 = new JButton();
    jButton2 = new JButton();
    jButton3 = new JButton();
    jButton4 = new JButton();
    jButton5 = new JButton();
    jButton6 = new JButton();
    jTable1 = new JTable();
    jMenuBar1 = new JMenuBar();
    jMenu1 = new JMenu();
    jMenuItem1 = new JMenuItem();
    jMenuItem2 = new JMenuItem();
    jMenuItem3 = new JMenuItem();

    JLabel1 = new JLabel("CONEXIONES");
    JLabel2 = new JLabel("N° CONEXIONES");
    JLabel3 = new JLabel("PORCENTAJE");
    JLabel4 = new JLabel("MEDIA DE CONEXIONES");
    JLabel5 = new JLabel("Introduce aquí el n° de ejecuciones a realizar:");
    JLabel1.setForeground(Color.black);
    JLabel4.setForeground(Color.black);
    JLabel4.setBackground(Color.blue);
    JLabelMedia = new JLabel();
    JLabelMedia.setForeground(Color.black);
    JLabelMedia.setBorder(BorderFactory.createLineBorder(Color.black));
    JTextField1 = new JTextField();

    img = new Imagen(); //Container, hereda de jpanel
    g = img.getGraphics();
    setVisible(true);
    getContentPane().setLayout(null);

    addWindowListener(new java.awt.event.WindowAdapter() {
      public void windowClosing(java.awt.event.WindowEvent evt) {
        exitForm(evt);
      }

      public void windowActivated(java.awt.event.WindowEvent evt) {
/*                                System.out.println("WindowListener: Ventana Activada");*/
        windowActivate();
      }

      public void windowDeactivated(java.awt.event.WindowEvent evt) {
/*                                System.out.println("WindowListener: Ventana desactivada");*/
        windowDeactivate();
      }

    });

    img.addMouseListener(new java.awt.event.MouseAdapter() {
      public void mousePressed(java.awt.event.MouseEvent evt) {
        jPanelMousePressed(evt);
      }
    });

    img.setBounds(0, 0, 707, 437);
    getContentPane().add(img);

    jButton1.setText("PASO ADELANTE");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
```

```
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
      }
    });
    jButton1.setBounds(10, 670, 150, 40); //(int x, int y, int width, int height)
    getContentPane().add(jButton1);

    jButton2.setText("EJECUCIÓN");
    jButton2.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton2ActionPerformed(evt);
      }
    });

    jButton2.setBounds(180, 670, 150, 40);
    getContentPane().add(jButton2);

    jButton3.setText("NUEVA EJECUCIÓN");
    jButton3.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton3ActionPerformed(evt);
      }
    });

    jButton3.setBounds(350, 670, 150, 40);
    getContentPane().add(jButton3);

    jButton4.setText("REINICIAR");
    jButton4.setForeground(Color.red);
    jButton4.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton4ActionPerformed(evt);
      }
    });

    jButton4.setBounds(520, 670, 150, 40);
    getContentPane().add(jButton4);

    jButton5.setText(NumeroSecuencias + " EJECUCIONES");
    jButton5.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton5ActionPerformed(evt);
      }
    });

    jButton5.setBounds(690, 670, 150, 40);
    getContentPane().add(jButton5);

    jButton6.setText("PASO ATRÁS");
    jButton6.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton6ActionPerformed(evt);
      }
    });

    jButton6.setBounds(860, 670, 150, 40);
    getContentPane().add(jButton6);

    JTextField1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        JTextField1ActionPerformed(evt);
      }
    });

    JTextField1.setBounds(730, 630, 60, 30);
    getContentPane().add(JTextField1);

    jTable1.setToolTipText("");
    jTable1.setModel(new javax.swing.table.DefaultTableModel(
        new Object[][] { {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
        , {null, null, null, null}
```

```
        , {null, null, null, null}
        , {null, null, null, null}
    }
        ,
        new String[] { //Cabeceras de las columnas de la tabla
        "CONEXIÓN", "ENTRADA", "SALIDA", "DISTRIBUCIÓN"
    }
        ) {
      boolean[] canEdit = new boolean[] {
          false, false, false, false
      };

      public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit[columnIndex];
      }
    });
    jTable1.setRowHeight(20);
    jTable1.setName("null");
    jTable1.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
    jTable1.setRowSelectionAllowed(false);
    jTable1.setBounds(725, 50, 280, 320);
    getContentPane().add(jTable1); //(int x,int y,int ancho, int largo)
    jTable1.getTableHeader().setFont(new Font("Dialog", 2, 9));
    jTable1.getTableHeader().setBounds(725, 30, 280, 20);
    getContentPane().add(jTable1.getTableHeader());
    jTable1.getTableHeader().setReorderingAllowed(false);
    jTable1.getTableHeader().setResizingAllowed(false);

    //jpanel para estadisticas

    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(etiq.length, etiq[0].length));
    for (int i = 0; i < etiq.length; i++) {
      for (int j = 0; j < etiq[0].length; j++) {
        etiq[i][j].setHorizontalAlignment(JLabel.CENTER);
        if ( (i == 1) || (i == 2))
          etiq[i][j].setBackground(Color.red);
        else {
          etiq[i][j].setBackground(Color.blue);
          etiq[i][j].setForeground(Color.black);
        }
        etiq[i][j].setBorder(BorderFactory.createLineBorder(Color.black));
        panel.add(etiq[i][j]);
      }
    }
    panel.setBounds(125, 450, 800, 150);
    getContentPane().add(panel);
    JLabel1.setBounds(25, 450, 100, 50);
}ull     , { de las().add(JLabel1);
    JLabel2.setBounds(25, 500, 100, 50);
    getContentPane().add(JLabel2);
    JLabel3.setBounds(25, 550, 100, 50);
    getContentPane().add(JLabel3);
    JLabel4.setBounds(750, 390, 150, 50);
    getContentPane().add(JLabel4);
    JLabelMedia.setBounds(900, 390, 50, 50);
    getContentPane().add(JLabelMedia);
    JLabel5.setBounds(450, 630, 350, 30);
    getContentPane().add(JLabel5);

    //fin jpanel para estadisticas

    jMenu1.setText("Algoritmo [A]");
    jMenuItem1.setText("A");
    jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItem1ActionPerformed(evt);
      }
    });

    jMenu1.add(jMenuItem1);
    jMenuItem2.setText("B");
    jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItem2ActionPerformed(evt);
      }
    });

    jMenu1.add(jMenuItem2);
    jMenuItem3.setText("C");
    jMenuItem3.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItem3ActionPerformed(evt);
```

```
      }
    });

    jMenu1.add(jMenuItem3);
    jMenuBar1.add(jMenu1);
    setJMenuBar(jMenuBar1);

    pack();
  }
        /*
         * Definición: Deja seleccionado el algoritmo C como algoritmo de encaminamiento.
        */
  private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
    jMenu1.setText("Algoritmo [C]");
    Algoritmo = 2;
    Reiniciar();

  }
        /*
         * Definición: Deja seleccionado el algoritmo B como algoritmo de encaminamiento.
        */
  private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
    jMenu1.setText("Algoritmo [B]");
    Algoritmo = 1;
    Reiniciar();

  }
        /*
         * Definición: Deja seleccionado el algoritmo A como algoritmo de encaminamiento.
        */
  private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
    jMenu1.setText("Algoritmo [A]");
    Algoritmo = 0;
    Reiniciar();
  }
        /*
         * Definición: Se ejecuta al pinchar en el botón "Paso Atrás".
        */
  private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
      if (Contador1 > 0) {
      img.dibuja(g2);
      Contador1--; //decrementamos lineas a dibujar
      img.setContador(Contador1);
      g = img.getGraphics();
      g2 = (Graphics2D) g;
      img.update(g2);
      img.dibuja(g2);

      TamañoVector = Contador1;
//              rellena Tabla
      for (int i = 0; i <= TamañoVector - 1; i++) {
        Conexiones conx = (Conexiones) con.elementAt(i);
        int IdIn = (conx.getEntrada()).getId();
        int bloqueIn = (conx.getEntrada()).getBloque();
        int IdOut = (conx.getSalida()).getId();
        int bloqueOut = (conx.getSalida()).getBloque();
        int Matriz = conx.getMI();
        int fila = 0;
        int columna = 0;

        int b = i + 1;
        jTable1.setValueAt(null, b, columna);
        jTable1.setValueAt(null, b, columna + 1);
        jTable1.setValueAt(null, b, columna + 2);
        jTable1.setValueAt(null, b, columna + 3);

        String indice = Integer.toString(i + 1);
        String idEntrada = Integer.toString(IdIn);
        String idSalida = Integer.toString(IdOut);
        String matriz = Integer.toString(Matriz);
        jTable1.setValueAt(indice, i, columna);
        jTable1.setValueAt(idEntrada, i, columna + 1);
        jTable1.setValueAt(idSalida, i, columna + 2);
        jTable1.setValueAt(matriz, i, columna + 3);
      }

    }
    else { /*optamos por no avisar que ya no quedan conexiones,
//              por no ser muy necesario y poder perjudicar a la gráfica
//              aviso = new Frame();
//              d = new DIALOGO(aviso, true) ;
//              d.mensaje("   No hay conexiones.");
//              d.CentraDialogo();*/
```

```
    }
  }

        /*
         * Definición: Se ejecuta al pinchar el botón EJECUTAR n SECUENCIAS, 1000 por defecto
        */
  private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
      for (int i = 0; i < NumeroSecuencias; i++) {
      Algor = new Algoritmo(); // nueva tirada
      Algor.generaConexiones();
      escogeAlgoritmo();
      con = Algor.getConexiones();
      OA.setConexiones(con);

      TamañoVector = (Algor.getConexiones()).size();
      int Conexiones;
      if (TamañoVector < 16)
        Conexiones = TamañoVector - 1;
      else {
        Conexiones = TamañoVector; // no se bloquea en la 16
      }

      estadisticas[Conexiones - 4]++;
      ejecuciones++;
    }

    //obtenemos estadisticas
    for (int a = 0; a < estadisticas.length; a++) {
      int npor = estadisticas[a];
      float porciento = (npor / ejecuciones) * 100;
      String Sporciento = Float.toString(porciento);
      String Snpor = Integer.toString(npor);
      if (Sporciento.length() > 4) {
        Sporciento = Sporciento.substring(0, 5);
      }
      etiq[1][a].setText(Snpor);
      etiq[2][a].setText(Sporciento);
    }
    CalculaMedia(ejecuciones);

    g = img.getGraphics(); //limpiamos el dibujo
    g2 = (Graphics2D) g;
    img.update(g2);
    limpiaTabla();

  }

        /*
         * Definición: Se ejecuta al pinchar el botón Reiniciar
        */
  private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
     Reiniciar();
  }

        /*
         * Definición: Se ejecuta al pinchar el botón Nueva Ejecución
         * introducimos la ejecucion el el cuadro de estadisticas Y creamos una nueva
        */
  private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    g = img.getGraphics(); //limpiamos el dibujo
    g2 = (Graphics2D) g;
    img.update(g2);

    limpiaTabla(); //limpiamos la tabla

    TamañoVector = (Algor.getConexiones()).size();
    int Conexiones;
    if (TamañoVector < 16)
      Conexiones = TamañoVector - 1;
    else {
      Conexiones = TamañoVector; // no se bloquea en la 16
    }

    /*las conexiones van de la 4 a la 16, ya que 4 es el minimo de conexiones que se podrán
realizar en una ejecucion*/
    estadisticas[Conexiones - 4]++;
    //cuadro de conexiones
    int ncon;
    ncon = Integer.parseInt(etiq[1][Conexiones - 4].getText());
    ncon++;
    String sncon = Integer.toString(ncon);
    etiq[1][Conexiones - 4].setText(sncon);
    ncon = Integer.parseInt(sncon);
```

```
    float porciento, npor;
    ejecuciones++;
    for (int c = 0; c < 13; c++) {
      npor = Float.parseFloat(etiq[1][c].getText());
      porciento = (npor / ejecuciones) * 100;
      String Sporciento = Float.toString(porciento);
      /*si el numero tiene mas de 3 decimales, nos quedamos con lo 3 1°*/
      if (Sporciento.length() > 4) {
        Sporciento = Sporciento.substring(0, 5);
      }
      etiq[2][c].setText(Sporciento);
    }
    Algor = new Algoritmo(); //nueva ejecución
    Algor.generaConexiones();
    escogeAlgoritmo();
    con = Algor.getConexiones();
    OA.setConexiones(con);
    Contador1 = 0;
    //Metodo para calcular la media de conexiones realizadas
    CalculaMedia(ejecuciones);
  }

        /*
         * Definición: Se ejecuta al pinchar el botón Ejecución
         */
  private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    //dibujamos
    Contador1 = (Algor.getConexiones()).size();
    TamañoVector = Contador1;
    img.setContador(TamañoVector);
    g = img.getGraphics();
    g2 = (Graphics2D) g;
    img.dibuja(g2);

    //rellena tabla
    rellenaTabla();
  }

        /*
         * Definición: Se ejecuta al pinchar el botón Paso Adelante
         */
  private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    TamañoVector = (Algor.getConexiones()).size();

    if (Contador1 < TamañoVector) {
      //rellena tabla
      Conexiones conx = (Conexiones) con.elementAt(Contador1);
      int IdIn = (conx.getEntrada()).getId();
      int bloqueIn = (conx.getEntrada()).getBloque();
      int IdOut = (conx.getSalida()).getId();
      int bloqueOut = (conx.getSalida()).getBloque();
      int Matriz = conx.getMI();
      int fila = 0;
      int columna = 0;
      String indice = Integer.toString(Contador1 + 1);
      String idEntrada = Integer.toString(IdIn);
      String idSalida = Integer.toString(IdOut);
      String matriz = Integer.toString(Matriz);
      jTable1.setValueAt(indice, Contador1, columna);
      jTable1.setValueAt(idEntrada, Contador1, columna + 1);
      jTable1.setValueAt(idSalida, Contador1, columna + 2);
      jTable1.setValueAt(matriz, Contador1, columna + 3);

      //para el ultimo elemento del vector si no es el 16:
      if ( (Contador1 == TamañoVector - 1) && (TamañoVector != 16)) /*esto se puede hacer xq
      no se va a bloquear en la conexion 16(nunca se haran 15 conexiones)*/
      {
        jTable1.setValueAt(indice, Contador1, columna);
        jTable1.setValueAt(idEntrada, Contador1, columna + 1);
        jTable1.setValueAt(idSalida, Contador1, columna + 2);
        jTable1.setValueAt("Bloqueo", Contador1, columna + 3);
      }

    }
    /*Contador1 es mayor que el tamaño del vector, se ha llegado al final de la ejecución*/
    else {
      if (Contador1 == 16) {
        TamañoVector = 17; //para el caso de no bloqueo
      }
      aviso = new Frame();
      d = new DIALOGO(aviso, true);
      d.mensaje("    No hay más conexiones, has llegado a la última: " +
```

```
                (TamañoVector - 1));
      d.CentraDialogo();
    }

    //dibuja lineas
    if (Contador1 < (Algor.getConexiones()).size()) {
      Contador1++; //pinta y avanza contador1
      img.setContador(Contador1);
      g = img.getGraphics();
      g2 = (Graphics2D) g;
      img.dibuja(g2);
    }
    else { //pinta pero no incrementes contador1 que has llegado al final
      img.setContador(Contador1);
      g = img.getGraphics();
      g2 = (Graphics2D) g;
      img.dibuja(g2);
    }

  }

        /*
         * Definición: Caja de texto para introducir el número de ejecuciones
        */
  private void JTextField1ActionPerformed(java.awt.event.ActionEvent evt) {
    String Seje;
    Seje = JTextField1.getText();
    int eje = 1000;
    try {
      eje = Integer.parseInt(Seje);
    }
    catch (NumberFormatException nfe) {
      aviso = new Frame();
      d = new DIALOGO(aviso, true);
      d.mensaje("    Introduce un NÚMERO entero menor de 10000");
      d.CentraDialogo();
    }
    if ( (eje < 10000) && (eje > 0)) {
      jButton5.setText(eje + " EJECUCIONES");
      NumeroSecuencias = eje;
    }
    else {
      //No introduzcas mas de 10000 ejecuciones
      aviso = new Frame();
      d = new DIALOGO(aviso, true);
      d.mensaje("    Introduce un NÚMERO entero menor de 10000");
      d.CentraDialogo();
    }

  }

        /*
         * Definición: Repintara la imagen si pulsamos en la imagen
        */
  private void jPanelMousePressed(java.awt.event.MouseEvent evt) {
    TamañoVector = Contador1;
/*para el caso de que se hayan realizado las 16 conexiones sin bloqueo, queremos que repinte
todo*/
    if (TamañoVector == 16) {
      img.setContador(TamañoVector);
    }
    g = img.getGraphics();
    g2 = (Graphics2D) g;
    img.update(g2);
    img.dibuja(g2);
  }

  /** Exit the Application */
  private void exitForm(java.awt.event.WindowEvent evt) {
    try {
      // Obtienes un Runtime
      Runtime r = Runtime.getRuntime();

      // Le mandas ejecutar el comando
      Process p = r.exec("java -jar SIMUCOM.jar"); /* No redirijas la salida a un fichero, no
hace falta.*/
      System.exit(0);
    }
    catch (Exception e) {
    }

    System.exit(0);
  }
```

```
        /*
         * Definición: Para cuando de active la ventana, repinta la imagen
        */
  private void windowActivate() {
    TamañoVector = Contador1;
/*para el caso de que se hayan realizado las 16 conexiones sin bloqueo, queremos que repinte
todo*/
    if (TamañoVector == 16) {
      img.setContador(TamañoVector);
    }
    g = img.getGraphics();
    g2 = (Graphics2D) g;
    img.update(g2);
    img.dibuja(g2);
  }

        /*
         * Definición: Para cuando de desactive la ventana, repinta la imagen
        */
  private void windowDeactivate() {
    TamañoVector = Contador1;
    if (TamañoVector == 16) {
      img.setContador(TamañoVector);
    }
    g = img.getGraphics();
    g2 = (Graphics2D) g;
    img.update(g2);
    img.dibuja(g2);
  }

        /*
         * Definición: Situamos el JDialog en el centro de la pantalla y ocupandola toda(1024 x
768 píxeles)
         *
         * Parámetros de entrada: JDialog jf
         *
         * Devuelve: -
        */
  private void pantallaCompleta(JDialog jf)
    {
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = jf.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    jf.setSize(screenSize.width, screenSize.height);
    jf.setVisible(true);
  }

        /*
         * Definición: Le pasa al objeto imagen el vector conexiones
         *
         * Parámetros de entrada: Vector vc: será el de conexiones
         *
         * Devuelve: -
        */
  private void setConexiones(Vector vc) {
    img.setConexiones(vc);
  }

        /*
         * Definición: Método para la rellenar la tabla de conexiones
         *
         * Parámetros de entrada: -
         *
         * Devuelve: -
        */
  private void rellenaTabla()
  {
    for (int i = 0; i <= TamañoVector - 1; i++) {
      Conexiones conx = (Conexiones) con.elementAt(i);
      int IdIn = (conx.getEntrada()).getId();
      int bloqueIn = (conx.getEntrada()).getBloque();
      int IdOut = (conx.getSalida()).getId();
      int bloqueOut = (conx.getSalida()).getBloque();
      int Matriz = conx.getMI();
      int fila = 0;
      int columna = 0;
      String indice = Integer.toString(i + 1);
```

```
      String idEntrada = Integer.toString(IdIn);
      String idSalida = Integer.toString(IdOut);
      String matriz = Integer.toString(Matriz);
      jTable1.setValueAt(indice, i, columna);
      jTable1.setValueAt(idEntrada, i, columna + 1);
      jTable1.setValueAt(idSalida, i, columna + 2);
);dInringate1.setValueAt(matriz, i, columna + 3);

      //para el ultimo elemento del vector si no es el 16:
      if ( (i == TamañoVector - 1) && (TamañoVector != 16)) //esto se puede hacer xq
      //no se va a bloquear en la conexion 16(nunca se haran 15 conexiones)
      {
        jTable1.setValueAt(indice, i, columna);
        jTable1.setValueAt(idEntrada, i, columna + 1);
        jTable1.setValueAt(idSalida, i, columna + 2);
        jTable1.setValueAt("Bloqueo", i, columna + 3);
      }
    }

  }

       /*
        * Definición:   Limpia tabla de conexiones
        *
        * Parámetros de entrada: -
        *
        * Devuelve: -
        */
  private void limpiaTabla()
  {
    for (int f = 0; f < 16; f++) {
      for (int c = 0; c < 4; c++) {
        jTable1.setValueAt(null, f, c);
      }
    }
  }

       /*
        * Definición: Limpia cuadros de estadísticas
        *
        * Parámetros de entrada: -
        *
        * Devuelve: -
        */
  private void limpiaEstadisticas()
  {
    for (int f = 1; f < 3; f++) {
      for (int c = 0; c < 13; c++) {
        etiq[f][c].setText("0"); //jlabels
      }
    }

    JLabelMedia.setText("  ");
    for (int i = 0; i < estadisticas.length; i++) {
      estadisticas[i] = 0;
    }
  }
       /*
        * Definición: Método para calcular la media de conexiones
        *
        * Parámetros de entrada: float ejecuciones
        *
        * Devuelve: -
        */
  private void CalculaMedia(float ejecuciones)
  {
    float NConexiones = 0, MediaConexiones = 0;
    for (int i = 0; i < estadisticas.length; i++) {
      NConexiones = NConexiones + estadisticas[i] * (i + 4);
    }
    MediaConexiones = NConexiones / ejecuciones;
    String SMedia = Float.toString(MediaConexiones);
    if (SMedia.length() > 4) {
      SMedia = SMedia.substring(0, 5);
    }
    JLabelMedia.setText("  " + SMedia);
  }

       /*
        * Definición: Le pasa a la clase algoritmo el algoritmo escogido
        *
        * Parámetros de entrada: -
        *
```

```
         * Devuelve: -
        */
  private void escogeAlgoritmo() {
    Algor.AlgoritmoEscogido(Algoritmo);

  }

        /*
         * Definición: Limpia la imagen,la tabla y las etiquetas de estadísticas.
         *
         * Parámetros de entrada: -
         *
         * Devuelve: -
        */
  private void: -Devuelv() {
    Contador1 = 0;
    ejecuciones = 0;
    Algor = new Algoritmo();
    Algor.generaConexiones();
    escogeAlgoritmo();
    con = Algor.getConexiones();
    OA.setConexiones(con);
    g = img.getGraphics(); //limpiamos el dibujo
    g2 = (Graphics2D) g;
    img.update(g2);
    limpiaTabla(); //limpiamos la tabla
    limpiaEstadisticas(); // limpia jlabel estadisticas
  }

  /**
   * @param args the command line arguments
   */
  public static void main(String args[]) {

    Algor = new Algoritmo();
    Algor.generaConexiones();
    Algor.AlgoritmoEscogido(Algoritmo);
    OA = new opcionA();
    con = Algor.getConexiones();
    OA.setConexiones(con);
    OA.pantallaCompleta(OA);
    OA.show();
  }

  // Variables declaration - do not modify
  private javax.swing.JTable jTable1;
  private Imagen img;
  private javax.swing.JButton jButton6;
  private javax.swing.JButton jButton5;
  private javax.swing.JButton jButton4;
  private javax.swing.JButton jButton3;
  private javax.swing.JButton jButton2;
  private javax.swing.JButton jButton1;
  private javax.swing.JMenuItem jMenuItem3;
  private javax.swing.JMenuItem jMenuItem2;
  private javax.swing.JMenuItem jMenuItem1;
  private javax.swing.JMenuBar jMenuBar1;
  private javax.swing.JMenu jMenu1;
  private JTextField JTextField1;

  private JLabel JLabel1, JLabel2, JLabel3, JLabel4, JLabel5, JLabelMedia;
  //JLabels para el cuadro de conexiones
  private static JLabel etiq[][] = {
      {
      new JLabel("4"), new JLabel("5"), new JLabel("6"), new JLabel("7"),
      new JLabel("8"), new JLabel("9"), new JLabel("10"), new JLabel("11"),
      new JLabel("12"), new JLabel("13"), new JLabel("14"), new JLabel("15"),
      new JLabel("16")}
      , {
      new JLabel("0"), new JLabel("0"), new JLabel("0"), new JLabel("0"),
      new JLabel("0"), new JLabel("0"), new JLabel("0"), new JLabel("0"),
      new JLabel("0"), new JLabel("0"), new JLabel("0"), new JLabel("0"),
      new JLabel("0")}
      , {
      new JLabel("0"), new JLabel("0"), new JLabel("0"), new JLabel("0"),
      new JLabel("0"), new JLabel("0"), new JLabel("0"), new JLabel("0"),
      new JLabel("0"), new JLabel("0"), new JLabel("0"), new JLabel("0"),
      new JLabel("0")}
      ,
  };

  private static int estadisticas[] = new int[13];
  private static int Contador1 = 0, NumeroSecuencias = 1000;
```

```
  private static int TamañoVector; /*el tamaño del vector conexiones*/
  private static int Algoritmo = 0; /*El numero Algoritmo indica el tipo de alg escogido(0:A,
1:B, 2:C)*/
  private static float ejecuciones = 0, cuentaEjecuciones = 0;
  private static Algoritmo Algor;
  private static Vector con;
  private static Graphics g;
  private static Graphics2D g2;
  private static opcionA OA;
  private static DIALOGO d;
  private static Frame aviso;
  // End of variables declaration
}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionA\DIALOGO.java

```
/*
 * DIALOGO.java
 */
import javax.swing.*;
import java.awt.*;


public class DIALOGO extends JDialog {
    /** A return status code - returned if Cancel button has been pressed */
    public static final int RET_CANCEL = 0;
    /** A return status code - returned if OK button has been pressed */
    public static final int RET_OK = 1;

        /*
         * Definición: Constructor
         *
         * Parámetros de entrada:
         * Frame parent
         * boolean modal
         * Devuelve: vector de conexiones
        */
    public DIALOGO(java.awt.Frame parent, boolean modal) {
        super(parent, modal);//constructor de la superclase
        initComponents();
    }

    /** @return the return status of this dialog - one of RET_OK or RET_CANCEL */
    public int getReturnStatus() {
        return returnStatus;
    }

        /*
         * Definición: Inicia los componentes del Dialogo
        */
    private void initComponents() {
        buttonPanel = new JPanel();
        okButton = new JButton();
        jLabel1 = new JLabel();

        setTitle("MENSAJE DE AVISO");
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                closeDialog(evt);
            }
        });

        buttonPanel.setLayout(new java.awt.FlowLayout(java.awt.FlowLayout.RIGHT));

        okButton.setText("OK");
        okButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                okButtonActionPerformed(evt);
            }
        });

        buttonPanel.add(okButton);

        getContentPane().add(buttonPanel, java.awt.BorderLayout.SOUTH);

        jLabel1.setText("");
        getContentPane().add(jLabel1, java.awt.BorderLayout.CENTER);

        pack();
    }

        /*
         * Definición: Cierra el dialogo cuando pulsas el botón OK
        */
    private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
        doClose(RET_OK);
    }

    /*
     * Definición: Cierra el dialogo
     */
    private void closeDialog(java.awt.event.WindowEvent evt) {
        doClose(RET_CANCEL);
    }

    private void doClose(int retStatus) {
        returnStatus = retStatus;
        setVisible(false);
        dispose();
```

```
    }

        /*
         * Definición: Introduce un mensaje en la etiqueta
         *
         * Parámetros de entrada: String ms: mensaje de aviso
         *
         * Devuelve: -
        */
        public void mensaje(String ms)
        {
                jLabel1.setText(ms);
        }

        /*
         * Definición: Situa el Dialogo
        */

        public void CentraDialogo()
        {
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = this.getSize();
        if (frameSize.height > screenSize.height) {
          frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
          frameSize.width = screenSize.width;
        }
        this.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
        this.setSize(screenSize.width/3,screenSize.height/5);
        this.setVisible(true);
        }


    /**
     * @param args the command line arguments
     */

    // Variables declaration - do not modify
    private JPanel buttonPanel;
    private JButton okButton;
    private JLabel jLabel1;
    // End of variables declaration

    private int returnStatus = RET_CANCEL;
}
```

Código Fuente de la aplicación

B.3 Opción B

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionB\Entrada.java

```java
/*
 * Entrada.java
 */

public class Entrada
{
	private int id, bloque;

	/*
	 * Definición: Constructor
	 *
	 * Parámetros de entrada: int id: número de entrada
	 *
	 * Devuelve: -
	*/
	public Entrada(int id)
	{
		this.id = id;
		bloque = (id - 1) / 4;
	}

	/* Definición: Devuelve la entrada
	 *
	 * Parámetros de entrada: -
	 *
	 * Devuelve: int id: la entrada
	*/
	public int getId()
	{
		return id;
	}

	/*
	 * Definición: Devuelve el bloque de entrada
	 *
	 * Parámetros de entrada: -
	 *
	 * Devuelve: int bloque: bloque de entrada
	*/
	public int getBloque()//devuelve el bloque de entrada
	{
		return bloque;
	}

}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionB\Salida.java

```java
/*
*       Salida.java
*/

public class Salida
{
        private int id, bloque;

        /* Definición:  Constructor
         *
         * Parámetros de entrada: int id
         *
         * Devuelve: -
        */
        public Salida(int id)
        {
                this.id = id;
                bloque = (id - 1) / 4;
        }

        /* Definición: Devuelve la salida
         *
         * Parámetros de entrada:
         *
         * Devuelve: int id
        */
        public int getId()
        {
                return id;
        }

        /* Definición: Devuelve el bloque de salida
         *
         * Parámetros de entrada:
         *
         * Devuelve: int bloque
        */
        public int getBloque()
        {
                return bloque;
        }
}
```

```
/*
 * Conexiones.java
 */
public class Conexiones
{
	private Entrada in;/*vectores para trabajar con las entradas y*/
	private Salida out; /*salidas por separado*/
	/*variables que indican el estado de la conexión conectado (conectado = true),
	 desconectado (desconectado = true)  bloqueado bloqueado = true*/
	private boolean conectado = false;
	private boolean bloqueado = false;
	private int MI;/*la matriz intermedia por la que esta conectada actualmente*/

	/*
	 * Definición: Constructor
	 * Parámetros de entrada:
	 * Entrada in: objeto con la entrada
	 * Salida out: objeto con la salida
	 * Devuelve: -
	 */
	public Conexiones(Entrada in, Salida out)
	{
		this.in = in;
		this.out = out;
	}

	/*
	 * Definición: Constructor, sirve para crear una nueva conexion apartir de otra
	 * Parámetros de entrada:
	 * Conexiones c: conexión apartir de la que queremos crear otra
	 * boolean conectado: Le especificamos el valor de esta variable de clase
	 * boolean bloqueado: Le especificamos el valor de esta variable de clase
	 * int MI: Le especificamos el valor de esta variable de clase
	 * Devuelve: -
	 */
	public Conexiones(Conexiones c, boolean conectado, boolean bloqueado, int MI)
	{
		this.in = c.getEntrada();
		this.out = c.getSalida();
		this.conectado = conectado;
		this.bloqueado = bloqueado;
		this.MI = MI;
	}

	/*
	 * Definición: Devuelve la entrada
	 * Parámetros de entrada: -
	 *
	 * Devuelve: Entrada in
	 */
	public Entrada getEntrada()
	{
		return in;
	}

	/*
	 * Definición: Devuelve la salida
	 * Parámetros de entrada: -
	 *
	 * Devuelve: Salida out
	 */
	public Salida getSalida()
	{
		return out;
	}

	/*
	 * Definición: Se le especifica por que MI conectó
	 * Parámetros de entrada: int MI
	 *
	 * Devuelve: -
	 */
	public void setMI(int MI)
	{
		this.MI = MI;
	}

	/*
	 * Definición: Devuelve por que MI conectó
	 * Parámetros de entrada: -
	 *
	 * Devuelve: int MI
	 */
```

```
	public int getMI()
	{
		return MI;
	}

	/*
	 * Definición: Se pone la conexión como conectada
	 * Parámetros de entrada: -
	 *
	 * Devuelve: -
	 */
	public void Conectar()
	{
		conectado = true;
	}

	/*
	 * Definición: Se pone la conexión como desconectada
	 * Parámetros de entrada: -
	 *
	 * Devuelve: -
	 */
	public void Desconectar()
	{
		conectado = false;
	}

	/*
	 * Definición: Dice si la conexión esta conectada o no
	 * Parámetros de entrada: -
	 *
	 * Devuelve: boolean conectado
	 */
	public boolean getConectado()
	{
		return conectado;
	}

	/*
	 * Definición: Se pone la conexión como bloqueada
	 * Parámetros de entrada: -
	 *
	 * Devuelve: -
	 */
	public void Bloquear()
	{
		bloqueado = true;
	}

	/*
	 * Definición: //Se pone la conexión como desbloqueada
	 * Parámetros de entrada: -
	 *
	 * Devuelve: -
	 */
	public void Desbloquear()
	{
		bloqueado = false;
	}

	/*
	 * Definición: Dice si la conexión esta bloqueada o no
	 * Parámetros de entrada: -
	 *
	 * Devuelve: boolean bloqueado
	 */
	public boolean getBloqueado()
	{
		return bloqueado;
	}

}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionB\MatrizIntermedia.java

```
/*
 * MatrizIntermedia.java
 */

import java.util.Vector;

public class MatrizIntermedia {
  private int id;
  private Vector in, out;

        /* Definición: Constructor
         *
         * Parámetros de entrada: int id
         *
         * Devuelve: -
        */
  public MatrizIntermedia(int id) {
    in = new Vector();
    out = new Vector();
    this.id = id;
  }

        /* Definición: Se añade el bloque de entrada al vector entradas
         *
         * Parámetros de entrada: int bloque: bloque entrada
         *
         * Devuelve: -
        */
  public void introduceBloqueIn(int bloque) {
    in.add(new Integer(bloque));
  }

        /* Definición: Se añade el bloque de salida al vector salidas
         *
         * Parámetros de entrada: int bloque: bloque salida
         *
         * Devuelve: -
        */
  public void introduceBloqueOut(int bloque) {
    out.add(new Integer(bloque));
  }

        /* Definición: Borra el bloque de entrada del vector entradas
         * este metodo es para el caso de desconexión
         *
         * Parámetros de entrada: int ent: el bloque de entrada que queremos borrar
         *
         * Devuelve: -
        */
  public void borraBloqueIn(int ent) {
    int i;
    for (i = 0; i < in.size(); i++) {
      if ( ( (Integer) in.elementAt(i)).intValue() == ent) {
        in.remove(i);
        break;
      }
    }
  }

        /* Definición: Borra el bloque de salida del vector salidas
         * este metodo es para el caso de desconexión
         *
         * Parámetros de entrada: int sal: el bloque de salida que queremos borrar
         *
         * Devuelve: -
        */
  public void borraBloqueOut(int sal) { //borramos el bloque de salida del vector salidas
    //este metodo es para el caso de desconexion
    int i;
    for (i = 0; i < out.size(); i++) {
      if ( ( (Integer) out.elementAt(i)).intValue() == sal) {
        out.remove(i);
        break;
      }
    }
  }

        /* Definición: Devuelve el tamaño del vector de entradas y el de salidas
         *
         * Parámetros de entrada: -
         *
         * Devuelve: int :tamaño vector
        */
```

```
  public int getInOutSize() {
    if (in.size() == out.size()) {
      return out.size(); ////el tamaño del vector in y el de out debe ser igual
    }
    else {
      return -1;
    }
  }

        /* Definición: Indica si el valor introducido estaba
         * ya almacenado en el vector de entradas, devuelve false
         * si se puede utilizar (no se utilizó la MI por un mismo bloque 2 veces)
         *
         * Parámetros de entrada: -
         *
         * Devuelve: boolean
        */
  public boolean utilizaIn(int valor)
  {
    for (int i = 0; i < in.size(); i++) {
      int e = ( (Integer) in.elementAt(i)).intValue();
      if (valor == e) {
        return true;
      }
    }
    return false;
  }

        /* Definición: Indica si el valor introducido estaba
         * ya almacenado en el vector de salida, devuelve false
         * si se puede utilizar (no se utilizó la MI por un mismo bloque 2 veces)
         *
         * Parámetros de entrada: -
         *
         * Devuelve: boolean
        */
  public boolean utilizaOut(int valor)
  {
    for (int i = 0; i < out.size(); i++) {
      int s = ( (Integer) out.elementAt(i)).intValue();
      if (valor == s) {
        return true;
      }
    }
    return false;
  }

}
```

```
/*
 * Algoritmo2.java
 */
import java.util.Vector;
import java.lang.Integer;
import java.lang.Math;


public class Algoritmo2
{
        private Vector conexiones;
        /*introducimos las 16 conexiones que se realizarian en principio,si
        no hubiera descox ni bloqueo. */
        private Vector entradas;
        /* vectores para manejar entradas y salidas por separado */
        private Vector salidas;
        private  Vector eventos;
        /*en este vector se va a guardar los eventos en el orden que se producen*/
        private MatrizIntermedia a, b, c, d, e, f, g;
        private int nmi;//Numero de Matrices Intermedias
        private int puntero =0 ;
        private int indice =0 ;
        private int Algoritm = 0;//el algoritmo escogido
        private boolean BloqueinVisto = false;
        /*Con esta variable nos aseguramos que se mire el bloque de entrada de cada M I*/
        private boolean BloqueoutVisto = false;
        /*Con esta variable nos aseguramos que se mire el bloque de entrada de cada M I*/
        private int tiempo = 0;/*variable que cuenta las ranuras que tarda cada evento*/
        private int maxEventos = 1;
        private int nconexiones;/*cuenta el número de conexiones realizadas con exito*/
        private boolean logro;
        private double TO = 8;

        /*
         * Definición: Constructor, crea distinto nº de MI según su parámetro
         * Parámetros de entrada: int nmi: numero de matrices intermedias
         *
         * Devuelve: -
         */
        public Algoritmo2 (int nmi)
        {
        this.nmi = nmi;
        conexiones = new Vector();
        eventos = new Vector();

                switch (nmi)
                {
                case 4:
                a = new MatrizIntermedia(0);
                b = new MatrizIntermedia(1);
                c = new MatrizIntermedia(2);
                d = new MatrizIntermedia(3);
                break;

                case 5:
                a = new MatrizIntermedia(0);
                b = new MatrizIntermedia(1);
                c = new MatrizIntermedia(2);
                d = new MatrizIntermedia(3);
                e = new MatrizIntermedia(4);
                break;

                case 6:
                a = new MatrizIntermedia(0);
                b = new MatrizIntermedia(1);
                c = new MatrizIntermedia(2);
                d = new MatrizIntermedia(3);
                e = new MatrizIntermedia(4);
                f = new MatrizIntermedia(5);
                break;

                case 7:
                a = new MatrizIntermedia(0);
                b = new MatrizIntermedia(1);
                c = new MatrizIntermedia(2);
                d = new MatrizIntermedia(3);
                e = new MatrizIntermedia(4);
                f = new MatrizIntermedia(5);
                g = new MatrizIntermedia(6);
                break;

                default: System.out.println("Error: Introduzca un nº entre 4 y 7 ");
                }
```

```
	}

	/*
	 * Definición: Desordenamos las conexiones para que las entradas no sean en orden
	 * Parámetros de entrada: -
	 *
	 * Devuelve: -
	 */
	private void desordenarEnSal()
	{
		for(int a=0; a<50; a++)//barajamos unas 50 veces
		{
		int numero = ((int) (Math.random() * (entradas.size()))); //cojo una conexion al
azar del vector "conexiones"
		Entrada en = (Entrada) entradas.elementAt(numero);
		entradas.add(en);		//añado al vector la entrada escogida al azar
		entradas.remove(numero);	//borro esa entrada "con" del vector, de esta
forma estara,
		}						//al final del vector solamente.
Tendremos al final 16 entradas

		for(int a=0; a<50; a++)//barajamos unas 50 veces
		{
		int numero = ((int) (Math.random() * (salidas.size()))); //cojo una salida al
azar del vector "conexiones"
		Salida sal = (Salida) salidas.elementAt(numero);
		salidas.add(sal);		//añado al vector la salida escogida al azar
		salidas.remove(numero); //borro esa salida "con" del vector, de esta forma
estara,
		}						//al final del vector solamente.
Tendremos al final 16 salidas

	}

	/*
	 * Definición: Se va añadiendo al vector de conexiones nuevas conexiones
	 * con entrada secuencial(1, 2, 3, ..., hasta16) y salida aleatoria
	 * (de 1 a 16 sin que se repita ninguna salida)
	 * Parámetros de entrada: -
	 *
	 * Devuelve: -
	 */
	public void generaConexion()
	{
		eventos.removeAllElements();
		conexiones.removeAllElements();

		int con = 0;
		int sumador = 0;
		int numerosal = 0;

			while (con != 16)
			{

				numerosal = ((int) (Math.random() * 16)) + 1;/*numero aleatorio
entre 0 y 16*/
				sumador = 0;
				for(int a=0; a<conexiones.size(); a++)
				{
					if (numerosal !=
(((Conexiones)conexiones.elementAt(a)).getSalida()).getId() )
					{
						sumador ++;
					}
					else
					{
					break;
					}
				}

				if (sumador == conexiones.size())
				/*asignamos  acada entrada secuencialmente una salida aleatoria,
fijandonos en no repetir salida*/
				{
					conexiones.addElement(new Conexiones(new Entrada(con +
1), new Salida(numerosal)));
					con ++;
				}
			}
			desordenar();
	}

	/*
```

```
         * Definición: Desordenar el vector conexiones
         * Parámetros de entrada: -
         *
         * Devuelve: -
         */
        private void desordenar()
        {
                for(int a=0; a<50; a++)
                {
                int numero = ((int) (Math.random() * 16));
                Conexiones con = (Conexiones) conexiones.elementAt(numero);
                conexiones.add(con);
                conexiones.remove(numero);
                }
        }

        /*
         * Definición: Las conexiones de vector conexiones que no estén conectadas se renuevan
         * Parámetros de entrada: -
         *
         * Devuelve: -
         */
        private void SetSalida()
        {
                entradas = new Vector();
                salidas = new Vector();
                int contador = 0;
                        for(int a=0; a<conexiones.size(); a++)
                        {
                                Conexiones cs = (Conexiones) conexiones.elementAt(a);
                                if(cs.getConectado() == false)/*vamos añadiendo a los vectores
entradas y salidas las conexiones que esten bloqueadas o desconectadas*/
                                {
                                        entradas.addElement(cs.getEntrada());
                                        salidas.addElement(cs.getSalida());
                                }
                        }
                        desordenarEnSal();/*obtenemos las entradas y salidas barajadas en ambos
vectores*/
                        for(int a=0; a<conexiones.size(); a++)
                        {
                                Conexiones cs = (Conexiones) conexiones.elementAt(a);
                                if(cs.getConectado() == false)/*sustituimos las entradas y
salidas por otras barajadas*/
                                {
                                        Conexiones nuevacon = new
Conexiones((Entrada)entradas.elementAt(contador), (Salida)salidas.elementAt(contador));
                                        conexiones.setElementAt(nuevacon, a);/*este metodo borra
la antigua conexion*/
                                        contador ++; /*cuenta las desconexiones y bloqueos*/
                                }
                        }
                        entradas.removeAllElements();
                        salidas.removeAllElements();

        }

        /*
         * Definición: Con este método se decide si se quiere desconectar o intentar conectar
         * Parámetros de entrada: -
         *
         * Devuelve: -
         */
        public int resuelveAlgoritmos()
        {
                logro = false;

                do{

                        double pconectar;
                        double pdesconectar;
                        pconectar= TO/(16*120-TO*119);/*llamadas de 2 min(con TO = 8 es
0.00826)*/
                        pdesconectar = 1.0/120.0; /*    0.008333*/
                        Conexiones conect = (Conexiones) conexiones.elementAt(indice);
                                if (conect.getConectado() == false)     /*esta desconectado*/
                                {
                                        double monedac = ((double) (Math.random()));
                                        if (monedac < pconectar)
                                        {
                                        AlgoritmoEscogido();
                                        logro = true;
                                        }// conecta
```

```
                                }
                                else //esta conectado
                                {
                                        double monedad = ((double) (Math.random()));
                                        if (monedad < pdesconectar)
                                        {
                                        conect.Desconectar();
                                        Conexiones desconect = new Conexiones(conect, false,
false, conect.getMI());/*creamos un nuevo objeto igual a conect pero que no apunta a él,
conectado = bloqueado = false*/
                                        eventos.addElement(desconect);
                                        borraInOut(conect.getMI(),
(conect.getEntrada()).getBloque(), (conect.getSalida()).getBloque());/*MI , ent, sal*/
                                        nconexiones --;
                                        SetSalida();/*le dejamos puesta a esa entrada del vector
conexiones una nueva salida*/
                                        logro = true;
                                        }//desconecta
                                }       //else esta conectado
                                indice ++;
                                if (indice==16)
                                {
                                        tiempo ++;//TIEMPO DE OBSERVACION, RANURAS
                                        indice = 0;
                                }

                        }   while (logro == false);

                        return 0;
        }   //resuelveAlgoritmo

        /*
         * Definición: //Algoritmo de Rotacion, intenta primero por la MI 0, luego la 1...
         * si no puedo por alguna lo intento con el siguiente.
         * Parámetros de entrada: -
         *
         * Devuelve: int
         */
        private int AlgoritmoA()
        {
                        int ok1=0,ok2=0; /*nos van a indicar el numero de veces seguidas que se
ha rechazado una MI
                        //por la entrada en caso de ok1 o por la salida en ok2
                        //cogemos elemento del vector
                        Conexiones con = (Conexiones) conexiones.elementAt(indice);

                        //miramos en que bloque de entrada esta actualmente*/
                        int bloqueIn = (con.getEntrada()).getBloque();

                        /*miramos que bloque de salida esta actualmente*/
                        int bloqueOut = (con.getSalida()).getBloque();

                        /*miramos en que bloque intermedio le toca (puntero%nmi)*/

                                        for (int a=0;a<conexiones.size();a++)
                                        {
                                                Conexiones cot = (Conexiones)
conexiones.elementAt(a);
                                                /*comparo el bloque entrada actual si coincide
con alguno de los  bloques de entrada   de "conexiones" */
                                                if ( (bloqueIn ==
(cot.getEntrada()).getBloque()) )
                                                {
                                                 while ((ok1 + ok2) < nmi)
                                                {
                                                        if (compruebaMI(puntero % nmi,
bloqueIn)) /*ahora hay que comprobar si el bloqueIn

se ha repetido en esa MI, si es true es que sí.*/


                                                        {
                                                        ok1 ++;
                                                        puntero ++;
                                                        a =0;/*con a=0 comparamos el
nuevo puntero con la Entr desde el principio*/

                                                        BloqueinVisto = true;
                                                        }
                                                        else
                                                        {
                                                                break;
                                                        }

                                                }
                                                if ((ok1==nmi)||((ok1 + ok2) >=
```

```
nmi))/*si supera o iguala el nmi es que hay Bloqueo*/
                                                        {
                                                                /*ponemos como desconectada la
conexión actual en el vector "conexiones"*/
                                                                con.Desconectar();
                                                                Conexiones conevt = new
Conexiones(con, false, true, puntero % nmi);/*conectado = false, bloqueado = true*/
eventos.addElement(conevt);/*Añadimos esta conexión al vector de eventos*/
                                                                SetSalida();/*le dejamos puesta
a la conexiones desconectadas una nueva conexion*/
                                                                return -1;
                                                        }
                                                }

                                                if ( (bloqueOut ==
(cot.getSalida()).getBloque()))
                                                {

                                                        while ((ok1 + ok2) < nmi)/*ahora hay que
comprobar si el bloqueOut
                                                        se ha repetido en esa MI, si es true es
que sí.*/
                                                        {
                                                                if (compruebaMO(puntero % nmi,
bloqueOut))

                                                                {
                                                                ok2 ++;
                                                                puntero ++;
                                                                a =0;
                                                                BloqueoutVisto = true;
                                                                }
                                                                else
                                                                {
                                                                        break;
                                                                }
                                                        }
                                                        if ((ok2==nmi)||((ok1 + ok2) >=nmi))
                                                        {       /*ponemos la conexión como
desconectada(desconexion o bloqueo)     */
                                                                con.Desconectar();
                                                                Conexiones  conevt = new
Conexiones(con, false, true, puntero % nmi);/*conectado = false, bloqueado = true*/
                                                                eventos.addElement(conevt);
                                                                SetSalida();/*le dejamos puesta
a la conexiones desconectadas una nueva conexion*/
                                                                return -1;
                                                        }

                                                } //if (bloqueOut)

                                                if ((BloqueinVisto == false)&&(BloqueoutVisto ==
true))
                                                /*con esto hacemos que si hemos incrementado el
ptero y no ha mirado la entrada que la mire*/
                                                {
                                                        if ( (bloqueIn ==
(cot.getEntrada()).getBloque()) )
                                                        {
                                                                while ((ok1 + ok2) < nmi)
                                                                {
                                                                        if (compruebaMI(puntero
% nmi, bloqueIn))

                                                                        {
                                                                        ok1 ++;
                                                                        puntero ++;
                                                                        a = 0;
                                                                        BloqueinVisto = true;
                                                                        }
                                                                        else
                                                                        {
                                                                                break;
                                                                        }
                                                                }

                                                        if ((ok1==nmi)||((ok1 + ok2) >=nmi))
                                                                {
                                                                        con.Desconectar();
                                                                        Conexiones  conevt = new
Conexiones(con, false, true, puntero % nmi);/*conectado = false, bloqueado = true */
eventos.addElement(conevt);
                                                                        SetSalida();/*le dejamos
```

```
puesta a la conexiones desconectadas y bloqueadas un nuevo objeto conexion*/
                                                                        return -1;
                                                                }
                                                        }
                                                }//fin if (bloquevisto == false)
                                        } //for (int a=0;a<indice;a++)
                                        con.Conectar();
                                        con.setMI(puntero % nmi);/*introducimos la MI escogida
en nuestra conexión*/
                                        Conexiones  conevt = new Conexiones(con, true, false,
puntero % nmi);/*conectado = true, bloqueado = false*/
                                        eventos.addElement(conevt);
                                        /* añadimos la entrada y la salida a los vectores "in" y
"out" de la Matriz de distribución escogida*/
                                        seleccionaMI(puntero % nmi, bloqueIn, bloqueOut);
                        puntero ++; // buscamos conectar por la MI siguiente
                        nconexiones++;
                        ok1=0;
                        ok2=0;
                        BloqueinVisto = false;
                        BloqueoutVisto = false;
                                        return 0;
        }

        /*
         * Definición: Algoritmo Secuencial se intentara siempre por la conexion 0,
         * si no se puede por una lo intento con la siguiente
         * Parámetros de entrada: -
         *
         * Devuelve: int
         */
        private int AlgoritmoB()
        {
                int ok1=0,ok2=0; /*nos van a indicar el numero de veces seguidas que se ha
rechazado una MI
                                por la entrada en caso de ok1 o por la salida en ok2*/
                /*cogemos elemento del vector*/
                Conexiones con = (Conexiones) conexiones.elementAt(indice);

                /*/miramos en que bloque de entrada esta actualmente*/
                int bloqueIn = (con.getEntrada()).getBloque();

                /*/miramos que bloque de salida esta actualmente*/
                int bloqueOut = (con.getSalida()).getBloque();

                /*/miramos en que bloque intermedio le toca (puntero%nmi)*/

                                for (int a=0;a<conexiones.size();a++)
                                {
                                        Conexiones cot = (Conexiones) conexiones.elementAt(a);
                                        /*/comparo el bloque entrada actual si coincide con
alguno de los anteriores*/
                                        if ( (bloqueIn == (cot.getEntrada()).getBloque()) )
                                        {
                                                 while ((ok1 + ok2) < nmi)
                                                {
                                                        if (compruebaMI(puntero % nmi,
bloqueIn)) /*/ahora hay q comprobar si el bloqueIn*/
                                                                                /*/se ha
repetido en esa Matriz Int, si es true es que si*/
                                                        {
                                                        ok1 ++;
                                                        puntero ++;
                                                        a =0;/*/con a=0 comparamos el nuevo
puntero con la Entrada desde el principio*/
                                                        BloqueinVisto = true;
                                                        }
                                                        else
                                                        {
                                                                break;
                                                        }

                                                }
                                                if ((ok1==nmi)||((ok1 + ok2) >= nmi))/*/si
supera o iguala el nmi es que hay Bloqueo*/
                                                {
                                                        con.Desconectar();/*/ponemos la conexión
como desconectada(desconexion o bloqueo)*/
                                                        Conexiones conevt = new Conexiones(con,
false, true, puntero % nmi);/*conectado = false, bloqueado = true*/
                                                        eventos.addElement(conevt);
                                                        SetSalida();/*/le dejamos puesta a la
conexiones desconectadas una nueva conexión*/
```

```
                                                        return -1;
                                                }
                                        }

                                        if ( (bloqueOut == (cot.getSalida()).getBloque()))
                                        {

                                                while ((ok1 + ok2) < nmi)
                                                {
                                                        if (compruebaMO(puntero % nmi,
bloqueOut))
/*                                              No se puede por este bloque de salida  bloqueOut
ir por esta MI  puntero % nmi*/
                                                        {
                                                        ok2 ++;
                                                        puntero ++;
                                                        a =0;
                                                        BloqueoutVisto = true;
                                                        }
                                                        else
                                                        {
                                                                break;
                                                        }
                                                }/*/al salir de este bucle while tenemos ok con
un valor, si es =4 tendremos un bloqueo*/
                                                if ((ok2==nmi)||((ok1 + ok2) >=nmi))
                                                {
                                                        con.Desconectar();/*/ponemos la conexión
como desconectada(desconexion o bloqueo)*/
                                                        Conexiones  conevt = new Conexiones(con,
false, true, puntero % nmi);/*/conectado = false, bloqueado = true */
                                                        eventos.addElement(conevt);
                                                        SetSalida();/*/le dejamos puesta a la
conexiones desconectadas una nueva conexion*/
                                                        return -1;
                                                }

                                        } //if (bloqueOut)

                                        if ((BloqueinVisto == false)&&(BloqueoutVisto == true))
/*/con esto hacemos que si hemos incrementado el ptero y no ha mirado la entrada que la mire*/
                                        {
                                                if ( (bloqueIn ==
(cot.getEntrada()).getBloque()))
                                                {
                                                        while ((ok1 + ok2) < nmi)
                                                        {
                                                                if (compruebaMI(puntero % nmi,
bloqueIn))

                                                                {
/*/                                             No se puede por este bloque de entrada  bloqueIn
ir por esta MI  puntero % nmi*/
                                                                ok1 ++;
                                                                puntero ++;
                                                                a = 0;
                                                                BloqueinVisto = true;
                                                                }
                                                                else
                                                                {
                                                                        break;
                                                                }
                                                        }

                                                if ((ok1==nmi)||((ok1 + ok2) >=nmi))
                                                        {
                                                                con.Desconectar();/*ponemos la
conexión como desconectada(desconexion o bloqueo)*/
                                                                Conexiones  conevt = new
Conexiones(con, false, true, puntero % nmi);/*/conectado = false, bloqueado = true */
                                                                eventos.addElement(conevt);
                                                                SetSalida();/*/le dejamos puesta
a la conexiones desconectadas una nueva conexion*/

                                                                return -1;
                                                        }
                                                }
                                        }       //fin if (bloquevisto == false)
                                } //for (int a=0;a<indice;a++)
                                con.Conectar();
                                Conexiones  conevt = new Conexiones(con, true, false, puntero %
nmi);/*/conectado = true, bloqueado = false */
                                eventos.addElement(conevt);/*/ añadimos al vector de eventos la
conexión*/
                                con.setMI(puntero % nmi);
```

```
                                seleccionaMI(puntero % nmi, bloqueIn, bloqueOut);
                        puntero = 0; // buscamos conectar por la MI 0
                        ok1=0;
                        ok2=0;
                            BloqueinVisto = false;
                        BloqueoutVisto = false;
                                return 0;
        }

        /*
         * Definición: Algoritmo Aleatorio, escoge siempre una MI al azar
         * Parámetros de entrada: -
         *
         * Devuelve: int
         */
        private int AlgoritmoC()
        {
                int ok1=0,ok2=0;
                puntero = ((int) (Math.random() * 4 * nmi)) + 1;

                //cogemos elemento del vector
                Conexiones con = (Conexiones) conexiones.elementAt(indice);

                /*/miramos en que bloque de entrada esta actualmente*/
                int bloqueIn = (con.getEntrada()).getBloque();

                /*/miramos que bloque de salida esta actualmente*/
                int bloqueOut = (con.getSalida()).getBloque();

                /*/miramos en que bloque intermedio le toca (puntero%nmi)*/
                for (int a=0;a<conexiones.size();a++)
                {
                        Conexiones cot = (Conexiones) conexiones.elementAt(a);
                        /*/comparo el bloque entrada actual si coincide con alguno de los
anteriores*/
                        if ( (bloqueIn == (cot.getEntrada()).getBloque()) )
                        {
                                 while ((ok1 + ok2) < nmi)
                                {
                                        if (compruebaMI(puntero % nmi, bloqueIn)) /*ahora hay
quq comprobar si el bloqueIn
                                        se ha repetido en esa Matriz Int, si es true es que sí*/
                                        {
/*/             No se puede por este bloque de entrada  bloqueIn  ir por esta MI  puntero %
nmi*/
                                        ok1 ++;
                                        puntero ++;
                                        a =0;/*/con a=0 comparamos el nuevo puntero con la Entr
desde el principio*/
                                        BloqueinVisto = true;
                                        }
                                        else
                                        {
                                                break;
                                        }

                                }
                                if ((ok1==nmi)||((ok1 + ok2) >= nmi))/*/si supera o iguala el
nmi es que hay Bloqueo*/
                                {
                                        con.Desconectar();/*ponemos la conexión como
desconectada(desconexión o bloqueo)*/
                                        Conexiones conevt = new Conexiones(con, false, true,
puntero % nmi);/*conectado = false, bloqueado = true*/
                                        eventos.addElement(conevt);
                                        SetSalida();/*le dejamos puesta a la conexiones
desconectadas una nueva conexión*/
                                        return -1;
                                }
                        }

                        if ( (bloqueOut == (cot.getSalida()).getBloque())
                        {

                                while ((ok1 + ok2) < nmi)
                                {
                                        if (compruebaMO(puntero % nmi, bloqueOut))
                                        {
/*      No se puede por este bloke de salida  bloqueOut ir por esta MI puntero%nmi);*/
                                        ok2 ++;
                                        puntero ++;
                                        a =0;
                                        BloqueoutVisto = true;
```

```
                                        }
                                        else
                                        {
                                                break;
                                        }
                                }/*al salir de este bucle while tenemos ok con un valor, si es
=4 tendremos un bloqueo*/
                                if ((ok2==nmi)||((ok1 + ok2) >=nmi))
                                {
                                        con.Desconectar();
                                        Conexiones  conevt = new Conexiones(con, false, true,
puntero % nmi);/*conectado = false, bloqueado = true */
                                        eventos.addElement(conevt);
                                        SetSalida();/*le dejamos puesta a la conexiones
desconectadas una nueva conexion*/
                                        return -1;
                                }

                        } //if (bloqueOut)
                        if ((BloqueinVisto == false)&&(BloqueoutVisto == true)) /*/con esto
hacemos que si hemos incrementado el ptero y no ha mirado la entrada que la mire*/
                        {
                                if ( (bloqueIn == (cot.getEntrada()).getBloque())  )
                                {
                                        while ((ok1 + ok2) < nmi)
                                        {
                                                if (compruebaMI(puntero % nmi, bloqueIn))
                                                {
/*"No se puede por este bloke de entrada  bloqueIn  ir por esta MI  puntero % 4*/
                                                ok1 ++;
                                                puntero ++;
                                                a = 0;
                                                BloqueinVisto = true;
                                                }
                                                else
                                                {
                                                        break;
                                                }
                                        }

                                if ((ok1==nmi)||((ok1 + ok2) >=nmi))
                                        {
                                                con.Desconectar();
                                                Conexiones  conevt = new Conexiones(con, false,
true, puntero % nmi);//conectado = false, bloqueado = true
                                                eventos.addElement(conevt);
                                                SetSalida();/*le dejamos puesta a la conexiones
desconectadas una nueva conexion*/
                                                return -1;
                                        }
                                }//fin if (bloquevisto == false)
                        }
                } //for (int a=0;a<indice;a++)
        con.Conectar();
        Conexiones  conevt = new Conexiones(con, true, false, puntero % nmi);/*conectado = true,
bloqueado = false */
        eventos.addElement(conevt);
        con.setMI(puntero % nmi);
        seleccionaMI(puntero % nmi, bloqueIn, bloqueOut);
        puntero = ((int) (Math.random() * 4 * nmi)) + 1; /*elegimos una MI al azar 4 matrices *
nmi cada MI*/
        ok1=0;
        ok2=0;
        BloqueinVisto = false;
        BloqueoutVisto = false;
        return 0;
        }

        /*
         * Definición: Introduce en una MI la entrada y salida asignada
         *
         * Parámetros de entrada:
         * int valor: MI
         * int entrada: bloque entrada
         * int salida: bloque salida
         * Devuelve: -
        */
        private void seleccionaMI(int valor, int entrada, int salida)
        {
                switch (valor)
                {
                case 0:
                        a.introduceBloqueIn(entrada);
```

```
                        a.introduceBloqueOut(salida);
                        break;

                case 1:
                        b.introduceBloqueIn(entrada);
                        b.introduceBloqueOut(salida);
                        break;

                case 2:
                        c.introduceBloqueIn(entrada);
                        c.introduceBloqueOut(salida);
                        break;

                case 3:
                        d.introduceBloqueIn(entrada);
                        d.introduceBloqueOut(salida);
                        break;

                case 4:
                        e.introduceBloqueIn(entrada);
                        e.introduceBloqueOut(salida);
                        break;

                case 5:
                        f.introduceBloqueIn(entrada);
                        f.introduceBloqueOut(salida);
                        break;

                case 6:
                        g.introduceBloqueIn(entrada);
                        g.introduceBloqueOut(salida);
                        break;
                }
        }

        /*
         * Definición: Borra de una MI la entrada y salida asignada
         *
         * Parámetros de entrada:
         * int valor: MI
         * int entrada: bloque entrada
         * int salida: bloque salida
         * Devuelve: -
        */
        private void borraInOut(int valor, int ent, int sal)
        {
                switch (valor)
                {
                case 0:
                        a.borraBloqueIn(ent);
                        a.borraBloqueOut(sal);
                        break;

                case 1:
                        b.borraBloqueIn(ent);
                        b.borraBloqueOut(sal);
                        break;

                case 2:
                        c.borraBloqueIn(ent);
                        c.borraBloqueOut(sal);
                        break;

                case 3:
                        d.borraBloqueIn(ent);
                        d.borraBloqueOut(sal);
                        break;

                case 4:
                        e.borraBloqueIn(ent);
                        e.borraBloqueOut(sal);
                        break;

                case 5:
                        f.borraBloqueIn(ent);
                        f.borraBloqueOut(sal);
                        break;

                case 6:
                        g.borraBloqueIn(ent);
                        g.borraBloqueOut(sal);
                        break;
                }
```

```
	}


	/*
	 * Definición: Comprueba si esta la entrada modulo en la MI valor
	 *
	 * Parámetros de entrada:
	 * int valor : MI
	 * int modulo : entrada
	 * Devuelve: boolean
	*/
	private boolean compruebaMI(int valor, int modulo)
	{
		switch (valor)
		{
		case 0:
			return a.utilizaIn(modulo);

		case 1:
			return b.utilizaIn(modulo);

		case 2:
			return c.utilizaIn(modulo);

		case 3:
			}* Def d.utilizaIn(modulo);

		case 4:
			return e.utilizaIn(modulo);

		case 5:
			return f.utilizaIn(modulo);

		case 6:
			return g.utilizaIn(modulo);

		default:
			return false;
		}
	}

	/*
	 * Definición: Comprueba si esta la salida modulo en la MI valor
	 *
	 * Parámetros de entrada:
	 * int valor : MI
	 * int modulo : salida
	 * Devuelve: boolean
	*/
	private boolean compruebaMO(int valor, int modulo)
	{
		switch (valor)
		{
		case 0:
			return a.utilizaOut(modulo);

		case 1:
			return b.utilizaOut(modulo);

		case 2:
			return c.utilizaOut(modulo);

		case 3:
			return d.utilizaOut(modulo);

		case 4:
			return e.utilizaOut(modulo);

		case 5:
			return f.utilizaOut(modulo);

		case 6:
			return g.utilizaOut(modulo);

		default:
			return false;
		}
	}

	/*
	 * Definición: Según la variable algortim escogemos un algoritmo u otro
	 * Parámetros de entrada: -
	 *
```

```
         * Devuelve: -
         */
        public void AlgoritmoEscogido()
        {
                switch (Algoritm){
                case 0:
                        {
                                AlgoritmoA();
                                break;
                        }
                case 1:
                        {
                                AlgoritmoB();
                                break;
                        }
                case 2:
                        {
                                AlgoritmoC();
                                break;
                        }
                        default: System.out.println("Error:");
                }
        }

        /*
         * Definición: Cambiamos desde opcionC la variable TO
         * Parámetros de entrada: double sto, nuevo TO
         *
         * Devuelve: -
         */
        public void setTO(double sto)
        {
           TO = sto;
        }

        /*
         * Definición: Cambiamos desde opcionC la variable algoritm
         * Parámetros de entrada: int a
         *
         * Devuelve:
         */
        public void setAlgoritm(int a)
        {
                if ((a>=0)&&(a<=2))
                Algoritm = a;
                else System.out.println("Error en setAlgoritm:");//esto no va pasar
        }

        /*
         * Definición: Devuelve el vector con los eventos
         * Parámetros de entrada: -
         *
         * Devuelve: Vector eventos
         */
        public Vector getEventos()
        {
                return eventos;
        }

        /*
         * Definición: Devuelve la duración en ranuras de un evento
         * Parámetros de entrada: -
         *
         * Devuelve: int tiempo
         */
        public int getTiempo()
        {
                return tiempo;
        }
}
```

```
/*
*       Paneles.java
*/

import java.awt.*;
import javax.swing.*;
import java.applet.*;
import java.awt.Font;

public class Paneles extends JPanel {

    Label [][] etiq =   {
    { new Label ("0"), new Label ("1"), new Label ("2"), new Label ("3"), new Label ("4"), new Label ("5"), new Label
("6"), new Label ("7"), new Label ("8"), new Label ("9"), new Label ("10"), new Label ("11"), new Label
("12"), new Label ("13"), new Label ("14"), new Label ("15"), new Label ("16"), new Label ("17") },
    { new Label ("18"), new Label ("19"), new Label ("20"), new Label ("21"), new Label ("22"), new Label ("23"), new Label
("24"), new Label ("25"), new Label ("26"), new Label ("27"), new Label ("28"), new Label ("29"), new Label
("30"), new Label ("31"), new Label ("32"), new Label ("33"), new Label ("34"), new Label ("35") },
    { new Label ("36"), new Label ("37"), new Label ("38"), new Label ("39"), new Label ("40"), new Label ("41"), new Label
("42"), new Label ("43"), new Label ("44"), new Label ("45"), new Label ("46"), new Label ("47"), new Label
("48"), new Label ("49"), new Label ("50"), new Label ("51"), new Label ("52"), new Label ("53") },
    { new Label ("54"), new Label ("55"), new Label ("56"), new Label ("57"), new Label ("58"), new Label ("59"), new Label
("60"), new Label ("61"), new Label ("62"), new Label ("63"), new Label ("64"), new Label ("65"), new Label
("66"), new Label ("67"), new Label ("68"), new Label ("69"), new Label ("70"), new Label ("71") },
    { new Label ("72"), new Label ("73"), new Label ("74"), new Label ("75"), new Label ("76"), new Label ("77"), new Label
("78"), new Label ("79"), new Label ("80"), new Label ("81"), new Label ("82"), new Label ("83"), new Label
("84"), new Label ("85"), new Label ("86"), new Label ("87"), new Label ("88"), new Label ("89") },
    { new Label ("90"), new Label ("91"), new Label ("92"), new Label ("93"), new Label ("94"), new Label ("95"), new Label
("96"), new Label ("97"), new Label ("98"), new Label ("99"), new Label ("100"), new Label ("101"), new Label
("102"), new Label ("103"), new Label ("104"), new Label ("105"), new Label ("106"), new Label ("107") },
    { new Label ("108"), new Label ("109"), new Label ("110"), new Label ("111"), new Label ("112"), new Label ("113"), new Label
("114"), new Label ("115"), new Label ("116"), new Label ("117"), new Label ("118"), new Label ("119"), new Label
("120"), new Label ("121"), new Label ("122"), new Label ("123"), new Label ("124"), new Label ("125") },
    { new Label ("126"), new Label ("127"), new Label ("128"), new Label ("129"), new Label ("130"), new Label ("131"), new Label
("132"), new Label ("133"), new Label ("134"), new Label ("135"), new Label ("136"), new Label ("137"), new Label
("138"), new Label ("139"), new Label ("140"), new Label ("141"), new Label ("142"), new Label ("143") },
    { new Label ("144"), new Label ("145"), new Label ("146"), new Label ("147"), new Label ("148"), new Label ("149"), new Label
("150"), new Label ("151"), new Label ("152"), new Label ("153"), new Label ("154"), new Label ("155"), new Label
("156"), new Label ("157"), new Label ("158"), new Label ("159"), new Label ("160"), new Label ("161") },
    { new Label ("162"), new Label ("163"), new Label ("164"), new Label ("165"), new Label ("166"), new Label ("167"), new Label
("168"), new Label ("169"), new Label ("170"), new Label ("171"), new Label ("172"), new Label ("173"), new Label
("174"), new Label ("175"), new Label ("176"), new Label ("177"), new Label ("178"), new Label ("179") },
    { new Label ("180"), new Label ("181"), new Label ("182"), new Label ("183"), new Label ("184"), new Label ("185"), new Label
("186"), new Label ("187"), new Label ("188"), new Label ("189"), new Label ("190"), new Label ("191"), new Label
("192"), new Label ("193"), new Label ("194"), new Label ("195"), new Label ("196"), new Label ("197") },
    { new Label ("198"), new Label ("199"), new Label ("200"), new Label ("201"), new Label ("202"), new Label ("203"), new Label
("204"), new Label ("205"), new Label ("206"), new Label ("207"), new Label ("208"), new Label ("209"), new Label
("210"), new Label ("211"), new Label ("212"), new Label ("213"), new Label ("214"), new Label ("215") },
    { new Label ("216"), new Label ("217"), new Label ("218"), new Label ("219"), new Label ("220"), new Label ("221"), new Label
("222"), new Label ("223"), new Label ("224"), new Label ("225"), new Label ("226"), new Label ("227"), new Label
("228"), new Label ("229"), new Label ("230"), new Label ("231"), new Label ("232"), new Label ("233") },
    { new Label ("234"), new Label ("235"), new Label ("236"), new Label ("237"), new Label ("238"), new Label ("239"), new Label
("240"), new Label ("241"), new Label ("242"), new Label ("243"), new Label ("244"), new Label ("245"), new Label
("246"), new Label ("247"), new Label ("248"), new Label ("249"), new Label ("250"), new Label ("251") },
    { new Label ("252"), new Label ("253"), new Label ("254"), new Label ("255"), new Label ("256"), new Label ("257"), new Label
("258"), new Label ("259"), new Label ("260"), new Label ("261"), new Label ("262"), new Label ("263"), new Label
("264"), new Label ("265"), new Label ("266"), new Label ("267"), new Label ("268"), new Label ("269") },
    { new Label ("270"), new Label ("271"), new Label ("272"), new Label ("273"), new Label ("274"), new Label ("275"), new Label
("276"), new Label ("277"), new Label ("278"), new Label ("279"), new Label ("280"), new Label ("281"), new Label
("282"), new Label ("283"), new Label ("284"), new Label ("285"), new Label ("286"), new Label ("287") },
    { new Label ("288"), new Label ("289"), new Label ("290"), new Label ("291"), new Label ("292"), new Label ("293"), new Label
("294"), new Label ("295"), new Label ("296"), new Label ("297"), new Label ("298"), new Label ("299"), new Label
("300"), new Label ("301"), new Label ("302"), new Label ("303"), new Label ("304"), new Label ("305") },
    { new Label ("306"), new Label ("307"), new Label ("308"), new Label ("309"), new Label ("310"), new Label ("311"), new Label
("312"), new Label ("313"), new Label ("314"), new Label ("315"), new Label ("316"), new Label ("317"), new Label
("318"), new Label ("319"), new Label ("320"), new Label ("321"), new Label ("322"), new Label ("323") },
    { new Label ("324"), new Label ("325"), new Label ("326"), new Label ("327"), new Label ("328"), new Label ("329"), new Label
("330"), new Label ("331"), new Label ("332"), new Label ("333"), new Label ("334"), new Label ("335"), new Label
("336"), new Label ("337"), new Label ("338"), new Label ("339"), new Label ("340"), new Label ("341") },
    { new Label ("342"), new Label ("343"), new Label ("344"), new Label ("345"), new Label ("346"), new Label ("347"), new Label
("348"), new Label ("349"), new Label ("350"), new Label ("351"), new Label ("352"), new Label ("353"), new Label
("354"), new Label ("355"), new Label ("356"), new Label ("357"), new Label ("358"), new Label ("359") },
    { new Label ("360"), new Label ("361"), new Label ("362"), new Label ("363"), new Label ("364"), new Label ("365"), new Label
("366"), new Label ("367"), new Label ("368"), new Label ("369"), new Label ("370"), new Label ("371"), new Label
("372"), new Label ("373"), new Label ("374"), new Label ("375"), new Label ("376"), new Label ("377") },
    { new Label ("378"), new Label ("379"), new Label ("380"), new Label ("381"), new Label ("382"), new Label ("383"), new Label
("384"), new Label ("385"), new Label ("386"), new Label ("387"), new Label ("388"), new Label ("389"), new Label
("390"), new Label ("391"), new Label ("392"), new Label ("393"), new Label ("394"), new Label ("395") },
    { new Label ("396"), new Label ("397"), new Label ("398"), new Label ("399"), new Label ("400"), new Label ("401"), new Label
("402"), new Label ("403"), new Label ("404"), new Label ("405"), new Label ("406"), new Label ("407"), new Label
("408"), new Label ("409"), new Label ("410"), new Label ("411"), new Label ("412"), new Label ("413") },
    { new Label ("414"), new Label ("415"), new Label ("416"), new Label ("417"), new Label ("418"), new Label ("419"), new Label
("420"), new Label ("421"), new Label ("422"), new Label ("423"), new Label ("424"), new Label ("425"), new Label
("426"), new Label ("427"), new Label ("428"), new Label ("429"), new Label ("430"), new Label ("431") },
    { new Label ("432"), new Label ("433"), new Label ("434"), new Label ("435"), new Label ("436"), new Label ("437"), new Label
("438"), new Label ("439"), new Label ("440"), new Label ("441"), new Label ("442"), new Label ("443"), new Label
("444"), new Label ("445"), new Label ("446"), new Label ("447"), new Label ("448"), new Label ("449") },
    { new Label ("450"), new Label ("451"), new Label ("452"), new Label ("453"), new Label ("454"), new Label ("455"), new Label
("456"), new Label ("457"), new Label ("458"), new Label ("459"), new Label ("460"), new Label ("461"), new Label
("462"), new Label ("463"), new Label ("464"), new Label ("465"), new Label ("466"), new Label ("467") },
    { new Label ("468"), new Label ("469"), new Label ("470"), new Label ("471"), new Label ("472"), new Label ("473"), new Label
("474"), new Label ("475"), new Label ("476"), new Label ("477"), new Label ("478"), new Label ("479"), new Label
("480"), new Label ("481"), new Label ("482"), new Label ("483"), new Label ("484"), new Label ("485") },
    { new Label ("486"), new Label ("487"), new Label ("488"), new Label ("489"), new Label ("490"), new Label ("491"), new Label
("492"), new Label ("493"), new Label ("494"), new Label ("495"), new Label ("496"), new Label ("497"), new Label
("498"), new Label ("499"), new Label ("500"), new Label ("501"), new Label ("502"), new Label ("503") },
    { new Label ("504"), new Label ("505"), new Label ("506"), new Label ("507"), new Label ("508"), new Label ("509"), new Label
("510"), new Label ("511"), new Label ("512"), new Label ("513"), new Label ("514"), new Label ("515"), new Label
("516"), new Label ("517"), new Label ("518"), new Label ("519"), new Label ("520"), new Label ("521") },
    { new Label ("522"), new Label ("523"), new Label ("524"), new Label ("525"), new Label ("526"), new Label ("527"), new Label
("528"), new Label ("529"), new Label ("530"), new Label ("531"), new Label ("532"), new Label ("533"), new Label
("534"), new Label ("535"), new Label ("536"), new Label ("537"), new Label ("538"), new Label ("539") },
    { new Label ("540"), new Label ("541"), new Label ("542"), new Label ("543"), new Label ("544"), new Label ("545"), new Label
("546"), new Label ("547"), new Label ("548"), new Label ("549"), new Label ("550"), new Label ("551"), new Label
("552"), new Label ("553"), new Label ("554"), new Label ("555"), new Label ("556"), new Label ("557") },
    { new Label ("558"), new Label ("559"), new Label ("560"), new Label ("561"), new Label ("562"), new Label ("563"), new Label
("564"), new Label ("565"), new Label ("566"), new Label ("567"), new Label ("568"), new Label ("569"), new Label
("570"), new Label ("571"), new Label ("572"), new Label ("573"), new Label ("574"), new Label ("575") },
    { new Label ("576"), new Label ("577"), new Label ("578"), new Label ("579"), new Label ("580"), new Label ("581"), new Label
("582"), new Label ("583"), new Label ("584"), new Label ("585"), new Label ("586"), new Label ("587"), new Label
("588"), new Label ("589"), new Label ("590"), new Label ("591"), new Label ("592"), new Label ("593") },
    { new Label ("594"), new Label ("595"), new Label ("596"), new Label ("597"), new Label ("598"), new Label ("599"), new Label
("600"), new Label ("601"), new Label ("602"), new Label ("603"), new Label ("604"), new Label ("605"), new Label
("606"), new Label ("607"), new Label ("608"), new Label ("609"), new Label ("610"), new Label ("611") },
    { new Label ("612"), new Label ("613"), new Label ("614"), new Label ("615"), new Label ("616"), new Label ("617"), new Label
("618"), new Label ("619"), new Label ("620"), new Label ("621"), new Label ("622"), new Label ("623"), new Label
("624"), new Label ("625"), new Label ("626"), new Label ("627"), new Label ("628"), new Label ("629") },
    { new Label ("630"), new Label ("631"), new Label ("632"), new Label ("633"), new Label ("634"), new Label ("635"), new Label
("636"), new Label ("637"), new Label ("638"), new Label ("639"), new Label ("640"), new Label ("641"), new Label
("642"), new Label ("643"), new Label ("644"), new Label ("645"), new Label ("646"), new Label ("647") },
    { new Label ("648"), new Label ("649"), new Label ("650"), new Label ("651"), new Label ("652"), new Label ("653"), new Label
("654"), new Label ("655"), new Label ("656"), new Label ("657"), new Label ("658"), new Label ("659"), new Label
("660"), new Label ("661"), new Label ("662"), new Label ("663"), new Label ("664"), new Label ("665") },
    { new Label ("666"), new Label ("667"), new Label ("668"), new Label ("669"), new Label ("670"), new Label ("671"), new Label
("672"), new Label ("673"), new Label ("674"), new Label ("675"), new Label ("676"), new Label ("677"), new Label
("678"), new Label ("679"), new Label ("680"), new Label ("681"), new Label ("682"), new Label ("683") },
    { new Label ("684"), new Label ("685"), new Label ("686"), new Label ("687"), new Label ("688"), new Label ("689"), new Label
("690"), new Label ("691"), new Label ("692"), new Label ("693"), new Label ("694"), new Label ("695"), new Label
("696"), new Label ("697"), new Label ("698"), new Label ("699"), new Label ("700"), new Label ("701") },
    { new Label ("702"), new Label ("703"), new Label ("704"), new Label ("705"), new Label ("706"), new Label ("707"), new Label
("708"), new Label ("709"), new Label ("710"), new Label ("711"), new Label ("712"), new Label ("713"), new Label
("714"), new Label ("715"), new Label ("716"), new Label ("717"), new Label ("718"), new Label ("719") },
    { new Label ("720"), new Label ("721"), new Label ("722"), new Label ("723"), new Label ("724"), new Label ("725"), new Label
("726"), new Label ("727"), new Label ("728"), new Label ("729"), new Label ("730"), new Label ("731"), new Label
("732"), new Label ("733"), new Label ("734"), new Label ("735"), new Label ("736"), new Label ("737") },
    { new Label ("738"), new Label ("739"), new Label ("740"), new Label ("741"), new Label ("742"), new Label ("743"), new Label
("744"), new Label ("745"), new Label ("746"), new Label ("747"), new Label ("748"), new Label ("749"), new Label
("750"), new Label ("751"), new Label ("752"), new Label ("753"), new Label ("754"), new Label ("755") },
    { new Label ("756"), new Label ("757"), new Label ("758"), new Label ("759"), new Label ("760"), new Label ("761"), new Label
("762"), new Label ("763"), new Label ("764"), new Label ("765"), new Label ("766"), new Label ("767"), new Label
("768"), new Label ("769"), new Label ("770"), new Label ("771"), new Label ("772"), new Label ("773") },
    { new Label ("774"), new Label ("775"), new Label ("776"), new Label ("777"), new Label ("778"), new Label ("779"), new Label
("780"), new Label ("781"), new Label ("782"), new Label ("783"), new Label ("784"), new Label ("785"), new Label
("786"), new Label ("787"), new Label ("788"), new Label ("789"), new Label ("790"), new Label ("791") },
    { new Label ("792"), new Label ("793"), new Label ("794"), new Label ("795"), new Label ("796"), new Label ("797"), new Label
("798"), new Label ("799"), new Label ("800"), new Label ("801"), new Label ("802"), new Label ("803"), new Label
("804"), new Label ("805"), new Label ("806"), new Label ("807"), new Label ("808"), new Label ("809") },
    { new Label ("810"), new Label ("811"), new Label ("812"), new Label ("813"), new Label ("814"), new Label ("815"), new Label
("816"), new Label ("817"), new Label ("818"), new Label ("819"), new Label ("820"), new Label ("821"), new Label
("822"), new Label ("823"), new Label ("824"), new Label ("825"), new Label ("826"), new Label ("827") },
    { new Label ("828"), new Label ("829"), new Label ("830"), new Label ("831"), new Label ("832"), new Label ("833"), new Label
("834"), new Label ("835"), new Label ("836"), new Label ("837"), new Label ("838"), new Label ("839"), new Label
("840"), new Label ("841"), new Label ("842"), new Label ("843"), new Label ("844"), new Label ("845") },
    { new Label ("846"), new Label ("847"), new Label ("848"), new Label ("849"), new Label ("850"), new Label ("851"), new Label
("852"), new Label ("853"), new Label ("854"), new Label ("855"), new Label ("856"), new Label ("857"), new Label
("858"), new Label ("859"), new Label ("860"), new Label ("861"), new Label ("862"), new Label ("863") },
    { new Label ("864"), new Label ("865"), new Label ("866"), new Label ("867"), new Label ("868"), new Label ("869"), new Label
("870"), new Label ("871"), new Label ("872"), new Label ("873"), new Label ("874"), new Label ("875"), new Label
("876"), new Label ("877"), new Label ("878"), new Label ("879"), new Label ("880"), new Label ("881") },
    { new Label ("882"), new Label ("883"), new Label ("884"), new Label ("885"), new Label ("886"), new Label ("887"), new Label
("888"), new Label ("889"), new Label ("890"), new Label ("891"), new Label ("892"), new Label ("893"), new Label
("894"), new Label ("895"), new Label ("896"), new Label ("897"), new Label ("898"), new Label ("899") },
    { new Label ("900"), new Label ("901"), new Label ("902"), new Label ("903"), new Label ("904"), new Label ("905"), new Label
("906"), new Label ("907"), new Label ("908"), new Label ("909"), new Label ("910"), new Label ("911"), new Label
("912"), new Label ("913"), new Label ("914"), new Label ("915"), new Label ("916"), new Label ("917") },
    { new Label ("918"), new Label ("919"), new Label ("920"), new Label ("921"), new Label ("922"), new Label ("923"), new Label
("924"), new Label ("925"), new Label ("926"), new Label ("927"), new Label ("928"), new Label ("929"), new Label
("930"), new Label ("931"), new Label ("932"), new Label ("933"), new Label ("934"), new Label ("935") },
    { new Label ("936"), new Label ("937"), new Label ("938"), new Label ("939"), new Label ("940"), new Label ("941"), new Label
("942"), new Label ("943"), new Label ("944"), new Label ("945"), new Label ("946"), new Label ("947"), new Label
("948"), new Label ("949"), new Label ("950"), new Label ("951"), new Label ("952"), new Label ("953") },
    { new Label ("954"), new Label ("955"), new Label ("956"), new Label ("957"), new Label ("958"), new Label ("959"), new Label
("960"), new Label ("961"), new Label ("962"), new Label ("963"), new Label ("964"), new Label ("965"), new Label
("966"), new Label ("967"), new Label ("968"), new Label ("969"), new Label ("970"), new Label ("971") },
    { new Label ("972"), new Label ("973"), new Label ("974"), new Label ("975"), new Label ("976"), new Label ("977"), new Label
("978"), new Label ("979"), new Label ("980"), new Label ("981"), new Label ("982"), new Label ("983"), new Label
("984"), new Label ("985"), new Label ("986"), new Label ("987"), new Label ("988"), new Label ("989") },
    { new Label ("990"), new Label ("991"), new Label ("992"), new Label ("993"), new Label ("994"), new Label ("995"), new Label
("996"), new Label ("997"), new Label ("998"), new Label ("999"), new Label ("1000"), new Label ("1001"), new Label
("1002"), new Label ("1003"), new Label ("1004"), new Label ("1005"), new Label ("1006"), new Label ("1007") },
    };
        private int entradaAnterior, salidaAnterior;
        private boolean fuedesconectado;

        /* Definición: Constructor
         *
         * Parámetros de entrada: -
         *
         * Devuelve:
        */
        public Paneles()
        {
                super();//llamamos al costructor de la clase JPanel
                GridLayout GridLayout1 = new GridLayout(etiq.length, etiq[0].length);
                GridLayout1.setHgap(2);/*/separamos los labels horizontalmente por una distancia
2*/
                GridLayout1.setVgap(2);/*/separamos los labels verticalmente por una distancia
2*/
                setLayout(GridLayout1);
                for (int i=0; i<etiq.length; i++)
                {
                                for (int j=0; j<etiq[0].length; j++)
                                {
```

```
                                        etiq[i][j].setBackground (Color.white);
                                        etiq[i][j].setAlignment(Label.CENTER);
                                        add(etiq[i][j]);
                                        if ( (i<=3) || ((i>7)&&(i<12)) )
                                        {
                                                if ( (j<=3)||((j>7)&&(j<12)) )
                                                {
                                                        etiq[i][j].setBackground (new
java.awt.Color(102, 153, 255));
                                                }
                                        }
                                        if ((i>3)&&(i<8) || ((i>11)&&(i<16)))
                                        {
                                                if ( ((j>3)&&(j<8)) || ((j>11)&&(j<16)) )
                                                {
                                                        etiq[i][j].setBackground (new
java.awt.Color(102, 153, 255));
                                                }
                                        }
                                }
                        }
                        Font f = new Font("Arial", Font.BOLD, 16);//15
                        setFont(f);
                }

                /* Definición:  Según la conexión pinta en el diagrama de conexiones.
                 *
                 * Parámetros de entrada: Conexiones con
                 *
                 * Devuelve:
                */
                public void setConexion(Conexiones con)
                {
                        int MI = con.getMI();
                        String SMI = Integer.toString(MI);
                        boolean b = con.getBloqueado();
                        boolean c = con.getConectado();
                        Entrada entrada = (Entrada)con.getEntrada();
                        Salida salida = (Salida)con.getSalida();
                        int e = entrada.getId();
                        int s = salida.getId();

                        if(fuedesconectado == true)
                        {
                        etiq[entradaAnterior][salidaAnterior].setText("");
                        }
                        if (c == true)
                        {
//                      Conexion
                        etiq[e - 1][s - 1].setForeground(Color.black);
                        etiq[e - 1][s - 1].setText(SMI);
                        fuedesconectado = false;
                        }
                        else
                        {
//                      Desconexion
                        etiq[e - 1][s - 1].setForeground(Color.red);
                        etiq[e - 1][s - 1].setText(SMI);
                        fuedesconectado = true;
                        }
                        if ( b == true )
                        {
//                      Bloqueado
                        etiq[e - 1][s - 1].setForeground(Color.red);
                        /*/no es realmente necesario, cogeria el clor rojo de la anterior condicion */
                        etiq[e - 1][s - 1].setText("B");
                        fuedesconectado = true;
                        }
                        entradaAnterior = e - 1;
                        salidaAnterior = s - 1;
                }

                /* Definición: Limpia el diagrama de conexiones
                 *
                 * Parámetros de entrada:
                 *
                 * Devuelve:
                */
                public void Limpiar()
                {
                        for (int i=0; i<etiq.length; i++)
                        {
                                        for (int j=0; j<etiq[0].length; j++)
```

```
                                        {
                                                etiq[i][j].setText("");
                                        }
                        }
                }

}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionB\opcionC.java

```
/*
 * opcionC.java
 *
 * Created on 17 de mayo de 2003, 18:47
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.JFrame.*;
import java.awt.image.*;
import java.util.Vector;
import java.io.File;


public class opcionC extends JFrame {

    /** Creates new form opcionC */
    public opcionC() {
        setTitle(" MATSWIT OPCIÓN B");
                eventos = new Vector();
        initComponents();
    }

        private void initComponents() {//GEN-BEGIN:initComponents

                jButton1 = new JButton();
        jButton2 = new JButton();
        jButton3 = new JButton();
        jButton4 = new JButton();

        jMenuBar1 = new JMenuBar();
        jMenu1 = new JMenu();
        jMenuItem1 = new JMenuItem();
        jMenuItem2 = new JMenuItem();
        jMenuItem3 = new JMenuItem();
        jMenu2 = new JMenu();
        jMenuItem4 = new JMenuItem();
        jMenuItem5 = new JMenuItem();
        jMenuItem6 = new JMenuItem();
        jMenuItem7 = new JMenuItem();

        Label01 = new Label("El diagrama muestra");
        Label02 = new Label("el nº de la matriz de distribución");
        Label03 = new Label("o B si hay bloqueo");

        Label1 = new Label();//eventos
        Label2 = new Label("En Curso:");
        JLabel3 = new JLabel();//meteremos el estado de la conexione en curso
        JLabel4 = new JLabel();// la e-s de la conexion en curso
        Label5 = new Label("Total:");
        JLabel6 = new JLabel();//meteremos el total
        Label7 = new Label("Conexiones:");
        JLabel8 = new JLabel();//meteremos las conectadas

        JTextField1 = new JTextField();//para meter TO
        JTextField2 = new JTextField();//para meter maximo de ejecuciones

        Label9 = new Label();
        Label10 = new Label("Factor de Actividad: "+TO+"  (de 0.1 a 15.9): ");
        Label11 = new Label();
        Label12 = new Label("En Curso:");
        JLabel13 = new JLabel();
        Label14 = new Label("Máximo:");
        JLabel15 = new JLabel();
        Label16 = new Label("Perdidas:");
        JLabel17 = new JLabel();

        Label18 = new Label();
        Label19 = new Label("Cursado:");
        JLabel20 = new JLabel();
        Label21 = new Label("Perdido:");
        JLabel22 = new JLabel();
        JLabel23 = new JLabel();
        Label24 = new Label("ENTRADAS");
        Label25 = new Label("SALIDAS");
        Label26 = new Label("DIAGRAMA DE CONEXIONES");

        ImageIcon logo = new ImageIcon ("Imagenes" + File.separator + "upctrelieve.jpg");
        setIconImage(logo.getImage());


        panel = new Paneles();
```

```
        getContentPane().setLayout(null);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        //jpanel
                panel.setBounds(470, 40, 528, 528);
        getContentPane().add(panel);//470
        int valor1 = 480;
        int valor2 = 450;
                for(int i =0; i<etiqS.length; i++)
                {
                if (i==9)
                {
                        valor1 = 477;
                        valor2 = 444;
                }
                etiqS[i].setBounds(valor1 + i*33, 570, 20, 20);
                getContentPane().add(etiqS[i]);//470
                etiqE[i].setBounds(valor2, 47 + i*33, 20, 20);
                getContentPane().add(etiqE[i]);//470
                }//texto de ayuda en el frame

        Font f1 = new Font("ITALIC", Font.ITALIC, 14);
);lletLayoutont.setFont(f1);
        Label02.setFont(f1);
        Label03.setFont(f1);

        Label01.setAlignment(Label.CENTER);
        Label02.setAlignment(Label.CENTER);
        Label03.setAlignment(Label.CENTER);

        Label01.setBackground(new java.awt.Color(0, 153, 153));
        Label02.setBackground(new java.awt.Color(0, 153, 153));
        Label03.setBackground(new java.awt.Color(0, 153, 153));

        Label01.setBounds(135, 20, 205, 30);
        getContentPane().add(Label01);
        Label02.setBounds(135, 50, 205, 30);
        getContentPane().add(Label02);
        Label03.setBounds(135, 80, 205, 30);
        getContentPane().add(Label03);
                //fin de texto ayuda

        Font f2 = new Font("Arial", Font.BOLD, 15);
        Label1.setFont(f2);
        Label1.setForeground(Color.black);
        Label1.setText("EVENTOS");
        Label2.setForeground(Color.black);
        Label5.setForeground(Color.black);
        Label7.setForeground(Color.black);

        Label1.setBounds(50, 150, 150, 30);
        getContentPane().add(Label1);
        Label2.setBounds(70, 180, 80, 30);
        getContentPane().add(Label2);
        JLabel3.setBounds(150, 180, 120, 30);
        getContentPane().add(JLabel3);//para el estado
        JLabel4.setBounds(250, 180, 80, 30);
        getContentPane().add(JLabel4);//para la e-s
        Label5.setBounds(70, 210, 60, 30);
        getContentPane().add(Label5);//total
        JLabel6.setBounds(130, 210, 50, 30);
        getContentPane().add(JLabel6);//metemos el total
        Label7.setBounds(210, 210, 80, 30);
        getContentPane().add(Label7);//conexiones
        JLabel8.setBounds(310, 210, 50, 30);
        getContentPane().add(JLabel8);//metemos las conectadas

        Label9.setFont(f2);
        Label9.setForeground(Color.black);
        Label9.setText("DATOS Y RESULTADOS");
        Label9.setBounds(50, 300, 250, 30);
        getContentPane().add(Label9);//DATOS Y RESULTADOS
        Label10.setBounds(70, 350, 260, 30);
        getContentPane().add(Label10);

        JTextField1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                JTextField1ActionPerformed(evt);
            }
```

```
        });

        JTextField1.setBounds(350, 350, 50, 25);
        getContentPane().add(JTextField1);

        Font f3 = new Font("Arial", Font.BOLD, 12);
        Label11.setFont(f3);
        Label11.setText("CONEXIONES");
        Label11.setBounds(70, 390, 80, 30);
        getContentPane().add(Label11);
        Label12.setBounds(90, 420, 60, 30);
        getContentPane().add(Label12);
        JLabel13.setBounds(160, 420, 30, 30);
        getContentPane().add(JLabel13);
        Label14.setBounds(210, 420, 60, 30);
        getContentPane().add(Label14);
        JLabel15.setBounds(280, 420, 30, 30);
        getContentPane().add(JLabel15);
        Label16.setBounds(330, 420, 60, 30);
        getContentPane().add(Label16);
        JLabel17.setBounds(400, 420, 30, 30);
        getContentPane().add(JLabel17);

        Label18.setFont(f3);
        Label18.setText("TRÁFICO");
        Label18.setBounds(70, 450, 80, 30);
        getContentPane().add(Label18);
        Label19.setBounds(90, 480, 60, 30);
        getContentPane().add(Label19);
        JLabel20.setBounds(160, 480, 90, 30);
        getContentPane().add(JLabel20);
        Label21.setBounds(250, 480, 60, 30);
        getContentPane().add(Label21);
        JLabel22.setBounds(320, 480, 100, 30);
        getContentPane().add(JLabel22);

        JLabel23.setText("Introduce el máximo de ejecuciones a realizar (no más de 10000):");
        JLabel23.setBounds(160, 600, 400, 30);
        getContentPane().add(JLabel23);

        JTextField2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                JTextField2ActionPerformed(evt);
            }
        });

        JTextField2.setBounds(555, 600, 60, 25);
        getContentPane().add(JTextField2);

        Label24.setFont(f3);
        Label25.setFont(f3);
        Label24.setForeground(Color.blue);
        Label25.setForeground(Color.blue);
        Font f4 = new Font("bold", Font.BOLD, 16);
        Label26.setFont(f4);
        Label26.setForeground(Color.black);
        Label25.setBounds(420, 573, 80, 15);
        getContentPane().add(Label25);
        Label24.setBounds(415, 20, 70, 15);
        getContentPane().add(Label24);
        Label26.setBounds(625, 10, 270, 25);
        getContentPane().add(Label26);

        jButton1.setText("PASO ADELANTE");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jButton1.setBounds(150, 640, 150, 30);
        getContentPane().add(jButton1);


        jButton2.setText("EJECUCIÓN");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        jButton2.setBounds(320, 640, 150, 30);
        getContentPane().add(jButton2);
```

```
        jButton3.setText(nejecuciones+" EJECUCIONES");
        jButton3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton3ActionPerformed(evt);
            }
        });

        jButton3.setBounds(490, 640, 180, 30);
        getContentPane().add(jButton3);

        jButton4.setText("REINICIAR");
        jButton4.setForeground (Color.red);
        jButton4.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton4ActionPerformed(evt);
            }
        });

        jButton4.setBounds(690, 640, 150, 30);
        getContentPane().add(jButton4);



        jMenu1.setText("Algoritmo [A]");
        jMenuItem1.setText("A");
        jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem1ActionPerformed(evt);
            }
        });

        jMenu1.add(jMenuItem1);
        jMenuItem2.setText("B");
        jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem2ActionPerformed(evt);
            }
        });

        jMenu1.add(jMenuItem2);
        jMenuItem3.setText("C");
        jMenuItem3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem3ActionPerformed(evt);
            }
        });

        jMenu1.add(jMenuItem3);
        jMenuBar1.add(jMenu1);
        setJMenuBar(jMenuBar1);

        jMenu2.setText("Matrices Intermedias [4]");
        jMenuItem4.setText("4");
        jMenuItem4.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem4ActionPerformed(evt);
            }
        });

        jMenu2.add(jMenuItem4);
        jMenuItem5.setText("5");
        jMenuItem5.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem5ActionPerformed(evt);
            }
        });

        jMenu2.add(jMenuItem5);
        jMenuItem6.setText("6");
        jMenuItem6.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem6ActionPerformed(evt);
            }
        });

        jMenu2.add(jMenuItem6);
        jMenuBar1.add(jMenu2);
        setJMenuBar(jMenuBar1);

        jMenuItem7.setText("7");
        jMenuItem7.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
                jMenuItem7ActionPerformed(evt);
            }
        });

        jMenu2.add(jMenuItem7);
        jMenuBar1.add(jMenu2);
        setJMenuBar(jMenuBar1);


        pack();
    }//GEN-END:initComponents

        /*
         * Definición: Deja seleccionadas 7 MI
        */
    private void jMenuItem7ActionPerformed(java.awt.event.ActionEvent evt) {
        jMenu2.setText("Matrices Intermedias [7]");
        MI = 7;
        Reiniciar();
    }

        /*
         * Definición: Deja seleccionadas 6 MI
        */
        private void jMenuItem6ActionPerformed(java.awt.event.ActionEvent evt) {
        jMenu2.setText("Matrices Intermedias [6]");
                MI = 6;
        Reiniciar();
    }

        /*
         * Definición: Deja seleccionadas 5 MI
        */
        private void jMenuItem5ActionPerformed(java.awt.event.ActionEvent evt) {
        jMenu2.setText("Matrices Intermedias [5]");
        MI = 5;
        Reiniciar();

    }

        /*
         * Definición: Deja seleccionadas 4 MI
        */
    private void jMenuItem4ActionPerformed(java.awt.event.ActionEvent evt) {
        jMenu2.setText("Matrices Intermedias [4]");
        MI = 4;
        Reiniciar();

    }
        /*
         * Definición: Deja seleccionado el algoritmo C como algoritmo de encaminamiento.
        */
        private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
                jMenu1.setText("Algoritmo [C]");
        Algoritmo = 2;
        Reiniciar();
    }

        /*
         * Definición: Deja seleccionado el algoritmo B como algoritmo de encaminamiento.
        */
    private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
        jMenu1.setText("Algoritmo [B]");
        Algoritmo = 1;
        Reiniciar();
    }

        /*
         * Definición: Deja seleccionado el algoritmo A como algoritmo de encaminamiento.
        */
    private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
        jMenu1.setText("Algoritmo [A]");
        Algoritmo = 0;
        Reiniciar();
    }

        /*
         * Definición: Campo de texto para introducir  El factor de actividad
        */
        private void JTextField1ActionPerformed(java.awt.event.ActionEvent evt){
                String STO;
                STO = JTextField1.getText();
                double TO = 8;
```

```
                if (STO.length()>4){STO = STO.substring(0,5);}
                //si el numero tiene mas de 3 decimales, nos quedamos con los 3 1°
                try     {
                TO = Double.parseDouble(STO);
                }catch(NumberFormatException nfe)
                {
                        aviso = new Frame();
                        d = new DIALOGO(aviso, true) ;
                        d.mensaje("   Hay que introducir un NÚMERO, por ejemplo: 9.32");
                        d.CentraDialogo();

                }

                if ((TO<=15.9)&&(TO>=0.1))
                {
                        Reiniciar();
                        //lo quitamos y solo cambiamos el TO sin borrar los datos actuales
                        Label10.setText("Factor de Actividad: "+ (TO) +"  (de 0.1 a 15.9): ");
                        Algor.setTO(TO);
                }
                else
                {
                aviso = new Frame();
                d = new DIALOGO(aviso, true) ;
                d.mensaje("   El F.A. debe estar entre 0.1 y 15.9");
                d.CentraDialogo();
                }

        }

        /*
         * Definición:
        */
        private void JTextField2ActionPerformed(java.awt.event.ActionEvent evt){
                String Sejecuciones;
                Sejecuciones = JTextField2.getText();
                int ejecuciones = 100;
                try
                {
                ejecuciones = Integer.parseInt(Sejecuciones);
                }catch (NumberFormatException nfe)
                {
                        aviso = new Frame();
                        d = new DIALOGO(aviso, true) ;
                        d.mensaje("   Hay que introducir un NÚMERO entero");
                        d.CentraDialogo();

                }

                if ((ejecuciones<=10000)&&(ejecuciones>0))
                {
                        Reiniciar();
                        jButton3.setText(ejecuciones +" EJECUCIONES");
                        nejecuciones = ejecuciones;
                }
                else
                {
                aviso = new Frame();
                d = new DIALOGO(aviso, true) ;
                d.mensaje("   Hay que introducir un número entero menor de 10.000");
                d.CentraDialogo();
                }

        }

        /*
         * Definición:
        */
    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
        //  Reiniciar
                Reiniciar();
    }

        /*
         * Definición:
        */
        private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        // Botón: 1000 ejecuciones
                int i=0;
                do
                {
                        paso();
                        i ++;
```

```
                }while(i<nejecuciones);
    }

        /*
         * Definición:
        */
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        // Botón: Ejecucion hasta bloqueo
                int i=0;
                do
                {
                        paso();
                        i++;
                }while((((Conexiones)eventos.elementAt(pasos - 1)).getBloqueado() == false) &&
(i<nejecuciones));
                if (i == nejecuciones )
                {
                        aviso = new Frame();
                        d = new DIALOGO(aviso, true);
                        d.mensaje("   No hubo bloqueo en: "+ nejecuciones +" ejecuciones.");
                        d.CentraDialogo();
                }
    }

        /*
         * Definición:
        */
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        //Botón: Paso adelante
                paso();
    }

    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt) {//GEN-FIRST:event_exitForm
    try
    {
        // Obtienes un Runtime
        Runtime r = Runtime.getRuntime();

        // Le mandas ejecutar el comando
        Process p = r.exec ("java -jar SIMUCOM.jar"); // Llamamos al menú de incio
        System.exit(0);
    }
    catch(Exception e)
    {
    }

                System.exit(0);

    }//GEN-LAST:event_exitForm

        /*
         * Definición: Situa nuestro JFrame en el centro dela pantalla
        y ocupando enteramente (1024 x 768 píxeles)
         * Parámetros de entrada: JFrame jf
         *
         * Devuelve: -
         */
    private void pantallaCompleta(JFrame jf)
    {
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = jf.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    jf.setSize(screenSize.width,screenSize.height);
    jf.setVisible(true);
    }

        /*
         * Definición: Le pasa al objeto de la clase Algoritmo2 el
         * algoritmo de encaminamiento elegido
         * Parámetros de entrada: -
         *
         * Devuelve: -
         */
    private void escogeAlgoritmo()
);eje
                Algor.setAlgoritm(Algoritmo);
        Algor.AlgoritmoEscogido();
```

```
    }

        /*
         * Definición: Realiza un evento
         * Parámetros de entrada: -
         *
         * Devuelve: -
         */
        private void paso()
        {
        etiqE[entradaAnt].setForeground(null);
        etiqS[salidaAnt].setForeground(null);
        etiqE[entradaAnt].setFont(null);
        etiqS[salidaAnt].setFont(null);

        Algor.resuelveAlgoritmos();//resolvemos y usamos

        eventos = Algor.getEventos();
        Conexiones evento = (Conexiones) eventos.elementAt(pasos);
        Entrada entrada = (Entrada)evento.getEntrada();
        Salida salida = (Salida)evento.getSalida();
        panel.setConexion(evento);//pintamos la conexion
        totales ++;
        ranuras = Algor.getTiempo();//tiempo que ha pasado desde el ultimo evento
        tObserva = tObserva + ranuras;//t total de observacion que vamos acumulando


                if ( ((Conexiones)eventos.elementAt(pasos)).getConectado() == true)
                {
                conexiones ++;
                JLabel3.setText("Conectado");
                }
                if ( (((Conexiones)eventos.elementAt(pasos)).getConectado() == false) &&
(((Conexiones)eventos.elementAt(pasos)).getBloqueado() == false))
                {
                JLabel3.setText("Desconectado");
                desconexiones++;
                }
                if ( (((Conexiones)eventos.elementAt(pasos)).getConectado() == false) &&
(((Conexiones)eventos.elementAt(pasos)).getBloqueado() == true))
                {
                 JLabel3.setText("Bloqueado");
                 perdidas ++;
                }
        int encurso = conexiones - desconexiones;
        JLabel4.setText(entrada.getId() +" con "+ salida.getId());
        JLabel6.setText(Integer.toString(totales));
        JLabel8.setText(Integer.toString(conexiones));
        JLabel13.setText(Integer.toString(encurso));
        if(encurso>maxima)
                maxima = encurso;
        JLabel15.setText(Integer.toString(maxima));
        JLabel17.setText(Integer.toString(perdidas));

        etiqE[entrada.getId()- 1].setForeground(new java.awt.Color(204, 0, 0));
        etiqS[salida.getId()- 1].setForeground(new java.awt.Color(204, 0, 0));
        Font fuente = new Font("Arial", Font.BOLD, 15);
        etiqE[entrada.getId()- 1].setFont(fuente);
        etiqS[salida.getId()- 1].setFont(fuente);
        entradaAnt = entrada.getId()- 1;
        salidaAnt = salida.getId()- 1;

        //obtendremos el TC y TP
        double volumen;
        volumen = (encurso)*ranuras;
        volumenTotal = volumenTotal + volumen;
        TC = volumenTotal/tObserva;//tráfico cursado
        TC = acortaNumero(TC);
        //evitamos que el nuemro sea muy largo, no mas de 8 decimales
        JLabel20.setText(Double.toString(TC));
        TP = (perdidas) * 120 /tObserva;

        String STP = precisa(Double.toString(TP));

        if (STP.length()>8){STP = STP.substring(0,9);}
        //si el numero tiene mas de 4 decimales, nos quedamos con lo 3 1º
        try     {
        TP = Double.parseDouble(STP);
        }catch(NumberFormatException nfe)
        {
//      TP No es un numero
        }
```

```
        JLabel22.setText(STP);
        pasos ++;
        }

        /*
         * Definición: Reinicia variable, limpia diagrama de conexiones y etiquetas
         * Parámetros de entrada: -
         *
         * Devuelve: -
         */
        private void Reiniciar()
        {
                etiqE[entradaAnt].setForeground(null);
                etiqS[salidaAnt].setForeground(null);
                etiqE[entradaAnt].setFont(null);
                etiqS[salidaAnt].setFont(null);

                JLabel3.setText("");
                JLabel4.setText("");
                JLabel6.setText("");
                JLabel8.setText("");
                JLabel13.setText("");
                JLabel15.setText("");
                JLabel17.setText("");
                JLabel20.setText("");
                JLabel22.setText("");
                pasos = 0;
                conexiones = 0;
                desconexiones = 0;
                totales = 0;
                perdidas = 0;
                maxima = 0;
                TC = 0;
                TP = 0;
                ts = 0;
                ranuras = 0;
                volumenTotal = 0;
                tObserva = 0;
                st = "";
                panel.Limpiar();
                eventos.clear();
                Algor  = new Algoritmo2(MI);
                Algor.setTO(TO);
                Label10.setText("Factor de Actividad: "+ (TO) +"  (de 0.1 a 15.9): ");
                JTextField1.setText("");
                Algor.generaConexion();//nueva conexion
                escogeAlgoritmo();

        }

        /*
         * Definición: Método para acortar un número
         * Parámetros de entrada: double TO
         *
         * Devuelve: double: el número modificado
         */
        private double acortaNumero(double T)
        {
                String ST = Double.toString(T);
                if (ST.length()>7){ST = ST.substring(0,8);}
                //si el numero tiene mas de  decimales, nos quedamos con los 3 1°
                try     {
                T = Double.parseDouble(ST);
                }catch(NumberFormatException nfe)
                {
//      Debe ser un munero
                        return 0;
                }
                return T;
        }

        /*
         * Definición: //metodo para evitar numeros del tipo 1.222222E-6 y
         * poder sacarlos como 0.00000122
         * Parámetros de entrada: -
         *
         * Devuelve: String modificado
         */
    private String precisa(String s)
    {
                String e = s.substring(s.length()- 3, s.length() - 2);
                String E = "E";
```

```
                if (e.compareTo(E)==0)
                {
                        String nd = s.substring(s.length() - 1);
                        //aqui tengo el nº de decimales
                        if (Double.parseDouble(nd)<6)
                        //al ser del orden de -6 lo ponemos a 0
                        {
                                st = "0.";
                                for(int i =0; i<Double.parseDouble(nd) - 1;i++)
                                {
                                st = st.concat("0");
                                }
                                st = st.concat(s.substring(0,1));
                                st = st.concat(s.substring(2,4));
                                //nos saltamos el punto
                                return st;
                        }
                        else
                        {
                                st = "0";
                                return st;
                        }
                }
                else
                {
                return s;//no hay que modificarla
                }
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        OC = new opcionC();
        OC.pantallaCompleta(OC);
        OC.show();
        Algor  = new Algoritmo2(MI);
        Algor.setTO(TO);
        Algor.generaConexion();/*/generamos al principio un vector de conexiones*/

    }


    // Variables declaration - do not modify//GEN-BEGIN:variables

        private JButton jButton4;
        private JButton jButton3;
        private JButton jButton2;
        private JButton jButton1;
        private JMenuItem jMenuItem7, jMenuItem6, jMenuItem5, jMenuItem4, jMenuItem3,
         jMenuItem2, jMenuItem1;
        private JMenuBar jMenuBar1;
        private JMenu jMenu1, jMenu2;
        private JLabel JLabel01, JLabel02, JLabel03, JLabel3, JLabel4,  JLabel6, JLabel8,
        JLabel13, JLabel15, JLabel17, JLabel20 ,JLabel22, JLabel23;
        private Label Label1, Label2, Label5 , Label7, Label9, Label10, Label11, Label12,
        Label14, Label16, Label18, Label19, Label21, Label24, Label25, Label26, Label01,
        Label02, Label03;
        private JTextField JTextField1, JTextField2;
        // End of variables declaration//GEN-END:variables
        private static opcionC OC;
        private static Paneles panel;
        private static DIALOGO d;
        private static Frame aviso;
        private static Algoritmo2 Algor;
        private static int Algoritmo, MI = 4, TO = 8;///El numero Algoritmo indica el tipo*/
        // de alg escogido(0:A, 1:B, 2:C)
        private  Vector eventos;
        /*en este vector se va a guardar el bloq, desbloq, conc, descnx de conexiones
         en el orden q se producen, q provienen de algoritmo2*/
        private int pasos = 0, totales, conexiones,desconexiones, perdidas, maxima,
        nejecuciones = 1000, entradaAnt, salidaAnt;
        private static Graphics g;
        private static Label [] etiqE =
        { new Label ("1"), new Label ("2"), new Label ("3"), new Label ("4"), new Label ("5"),
        new Label ("6"), new Label ("7"), new Label ("8"), new Label ("9"), new Label ("10"),
        new Label ("11"), new Label ("12"), new Label ("13"), new Label ("14"),
        new Label ("15"), new Label ("16") };
        private static Label [] etiqS =
        { new Label ("1"), new Label ("2"), new Label ("3"), new Label ("4"), new Label ("5"),
        new Label ("6"), new Label ("7"), new Label ("8"), new Label ("9"), new Label ("10"),
        new Label ("11"), new Label ("12"), new Label ("13"), new Label ("14"),
```

```
        new Label ("15"), new Label ("16") };
        private static double ranuras = 0, tObserva=0, volumenTotal=0, TC, TP, ts =0;
        private static String st;
}
```

Archivo: C:\Documents and Settings\Gregorio\Escritorio\opcionB\DIALOGO.java

```
/*
 * DIALOGO.java
 */
import javax.swing.*;
import java.awt.*;


public class DIALOGO extends JDialog {
    /** A return status code - returned if Cancel button has been pressed */
    public static final int RET_CANCEL = 0;
    /** A return status code - returned if OK button has been pressed */
    public static final int RET_OK = 1;

        /*
         * Definición: Constructor
         *
         * Parámetros de entrada:
         * Frame parent
         * boolean modal
         * Devuelve: vector de conexiones
        */
    public DIALOGO(java.awt.Frame parent, boolean modal) {
        super(parent, modal);//constructor de la superclase
        initComponents();
    }

    /** @return the return status of this dialog - one of RET_OK or RET_CANCEL */
    public int getReturnStatus() {
        return returnStatus;
    }

        /*
         * Definición: Inicia los componentes del Dialogo
        */
    private void initComponents() {
        buttonPanel = new JPanel();
        okButton = new JButton();
        jLabel1 = new JLabel();

        setTitle("MENSAJE DE AVISO");
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                closeDialog(evt);
            }
        });

        buttonPanel.setLayout(new java.awt.FlowLayout(java.awt.FlowLayout.RIGHT));

        okButton.setText("OK");
        okButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                okButtonActionPerformed(evt);
            }
        });

        buttonPanel.add(okButton);

        getContentPane().add(buttonPanel, java.awt.BorderLayout.SOUTH);

        jLabel1.setText("");
        getContentPane().add(jLabel1, java.awt.BorderLayout.CENTER);

        pack();
    }

        /*
         * Definición: Cierra el dialogo cuando pulsas el botón OK
        */
    private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
        doClose(RET_OK);
    }

    /*
     * Definición: Cierra el dialogo
     */
    private void closeDialog(java.awt.event.WindowEvent evt) {
        doClose(RET_CANCEL);
    }

    private void doClose(int retStatus) {
        returnStatus = retStatus;
        setVisible(false);
        dispose();
```

```
    }

        /*
         * Definición: Introduce un mensaje en la etiqueta
         *
         * Parámetros de entrada: String ms: mensaje de aviso
         *
         * Devuelve: -
        */
        public void mensaje(String ms)
        {
                jLabel1.setText(ms);
        }

        /*
         * Definición: Situa el Dialogo
        */
        public void CentraDialogo()
        {
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = this.getSize();
        if (frameSize.height > screenSize.height) {
          frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
          frameSize.width = screenSize.width;
        }
        this.setLocation((screenSize.width - frameSize.width) / 2, 3*(screenSize.height -
frameSize.height) / 4);
        this.setSize(screenSize.width/3,screenSize.height/5);
        this.setVisible(true);
        }


    /**
     * @param args the command line arguments
     */

    // Variables declaration - do not modify
    private JPanel buttonPanel;
    private JButton okButton;
    private JLabel jLabel1;
    // End of variables declaration

    private int returnStatus = RET_CANCEL;
}
```

Bibliografía

D. Bertsekas and R. Gallager, Data Networks ,Prentice Hall,New Jersey,second edition,1992. ISBN: 0132016745.

Broadband Integrated Networks. Mischa Schwartz. Prentice Hall, 1996, ISBN 0135192404.

Notas de clase de Conmutación, 2º curso Ingeniería Telemática.

Notas de clase de Programación, 2º curso Ingeniería Telemática.

Notas de clase de Redes y Servicios, 2º curso Ingeniería Telemática.

Manuales de Java:

- Java 1.2 al descubierto. Jamie Jaworski, ISBN: 848322061X.
- Como programar en Java. Harvey Deitel, ISBN: 9701700449.
- The JFC Swing Tutorial. Kathy Walrath y Mary Campione, ISBN: 0201433214.

Índice de figuras