5 Interfaces de usuario

Unos de los fines de la aplicación es almacenar correctamente los datos referentes a la red vial. Para ello se usan las estructuras de bases de datos del sistema.

Pero no menos importantes son los interfaces de usuario. Los interfaces de usuario presentan, en un entorno gráfico amigable para el mismo, la información que éste solicita. Asímismo, permite que introduzca nuevos datos o modifique los ya existentes sin necesidad de tener un gran conocimiento de la arquitectura de almacenamiento de datos.

Esto implica que, por ejemplo, cuando un usuario borra un nodo, el sistema se encarga de borrar no sólo el nodo, sino además todos aquellos tramos que tuviesen a este nodo como nodo de origen o destino y todos los giros que se produjesen en este nodo. Todo esto de forma completamente transparente al usuario.

5.1 Descripción de interfaces

El interfaz gráfico de usuario de la aplicación se ha dividido en varios interfaces. El primero de ellos es el interfaz de representación gráfica. Este interfaz se usa para representar la red vial, por lo que se usará también dentro de los demás interfaces, como podrá verse a continuación.

Otro interfaz es el interfaz de nodos. Una vez seleccionado un nodo sobre la red vial, se accede a este interfaz para visualizar la información sobre ese nodo y los giros que se producen en él. Por supuesto que es posible modificar la información ya existente o añadir nueva.

El interfaz de tramos ofrece la misma funcionalidad que la nodos pero sobre los tramos de la red vial, los puntos de medida y los distintos carriles que tiene cada tramo.

Finalmente, el interfaz de líneas de autobús permite acceder a la información sobre el recorrido y paradas de las distintas líneas de autobús e información horaria sobre las mismas.

En las secciones siguientes se explica cada uno de los interfaces con más detalle

5.1.1 Interfaz de representación gráfica

Realmente no existe un interfaz de usuario único en el que realizar el zoom y el windowing, ya que cualquiera de estas operaciones se van a poder realizar sobre cualquier imagen que aparezca en pantalla.

En concreto, se puede realizar zoom y windowing sobre la imagen de la red vial que aparece nada más cargarla, cuando aún no se ha seleccionado ningún interfaz en concreto, como se muestra en la figura 5.1.1.1.

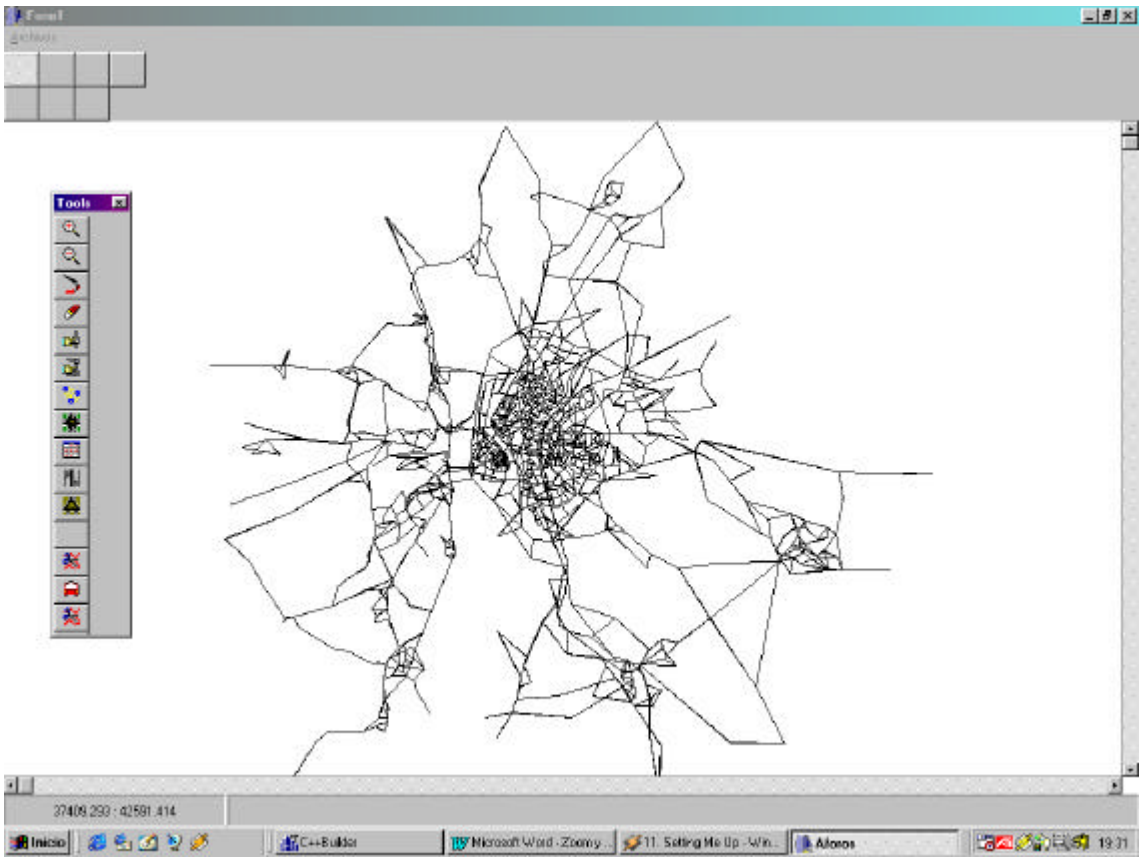


Figura 5.1.1.1: Imagen de la red vial

Como puede apreciarse en la figura 5.1.1.1, en la barra de herramientas, que es el formulario cuyo nombre es *Tools*, los dos primeros botones son sendas lupas, una con el signo + y la otra con el signo -. La lupa con el signo más es el botón que debe usarse para realizar el zoom, y la que aparece con el signo menos es la que corresponde al windowing (la acción contraria al zoom). Los demás botones se usan para cargar otros interfaces de usuario.

En el interfaz de usuario de líneas de bus, donde se permite añadir tramos a las líneas de bus, modificar los horarios, etc... también aparece una imagen de la red vial, más pequeña que la principal que se genera al cargar la información de la red vial, pero sobre la que se puede hacer zoom y windowing aparte de las funciones específicas que tiene la imagen para ese interfaz en concreto. En realidad en este interfaz aparecen dos imágenes, una de ellas muestra la zona de red vial seleccionada y la otra muestra una panorámica global de la red vial indicando sobre qué zona se está realizando el zoom o el windowing. Este interfaz se muestra en la figura 5.1.1.2.

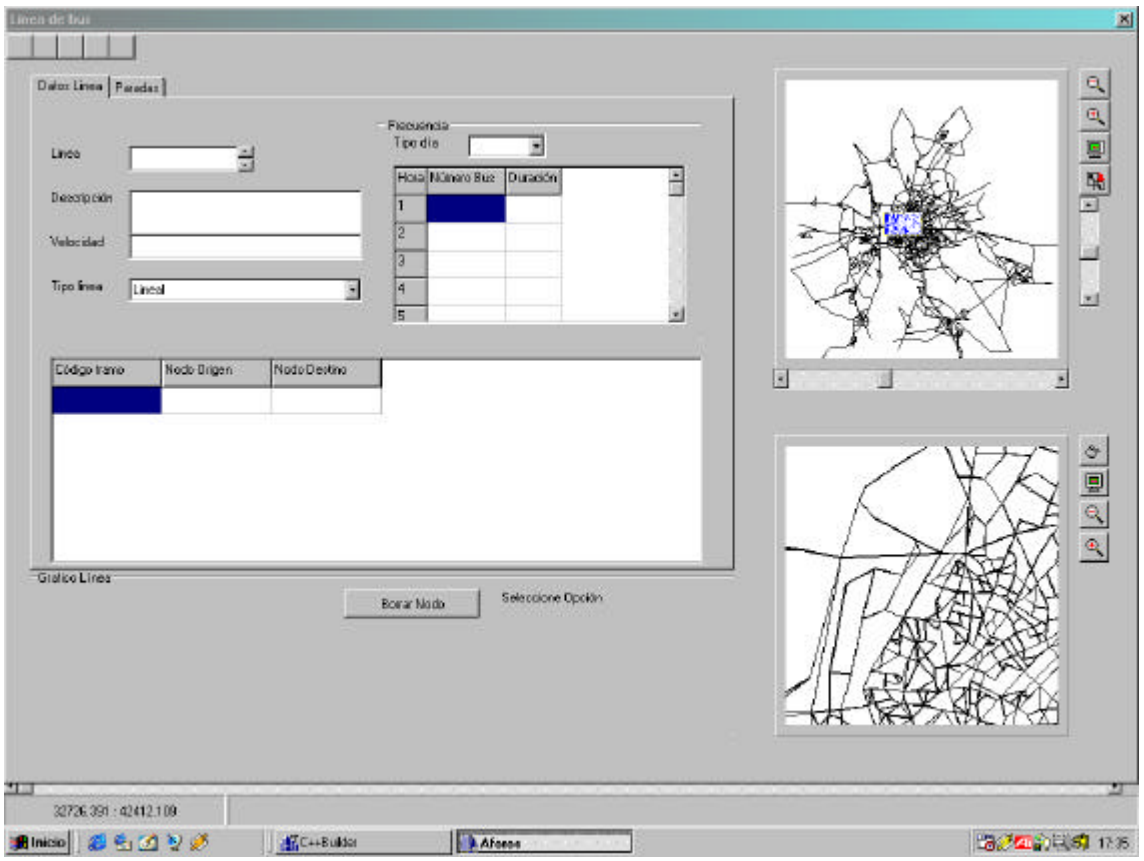


Figura 5.1.1.2: Imágenes de la red vial en el interfaz de líneas de bus

En esta figura se puede apreciar como la imagen que aparece en la esquina superior derecha contiene una panorámica global de red vial, y que aparece marcada con un rectángulo de color la porción de la red vial que se está representando en la otra imagen, la que aparece en la esquina inferior derecha.

Puede observarse como en los botones que aparecen a la derecha de cada una de las imágenes siempre hay uno con el icono de una lupa con el símbolo + y otra lupa con el símbolo -. Estos botones son los que proporcionan las funciones de zoom y windowing, respectivamente, en cada una de las imágenes.

En casi todos los interfaces que incluye la aplicación aparecen estas dos imágenes en la parte derecha del mismo. En todos ellos los botones de zoom y windowing están representados por los mismos iconos y tienen por supuesto la misma función.

Aparte de los botones de zoom y windowing existe siempre otro botón cuya función es volver al estado inicial, es decir, muestra toda la red vial en la pantalla. A esta situación se le ha llamado estado inicial porque es como aparece la imagen si no se ha realizado ninguna operación de windowing.

Cuando se trate en profundidad cada uno de los distintos interfaces se detallará las función del botón que resta por mencionar, ya que en este caso la función depende del interfaz en que se esté.

El procedimiento para realizar el zoom y el windowing es idéntico, aunque las consecuencias para la imagen que se muestra sean distintas, independentemente del

interfaz en que nos encontremos. En primer lugar hay que seleccionar la función que se desee pulsando el botoón correspondiente: lupa con símbolo + para el zoom y lupa con símbolo – para el windowing.

Una vez seleccionada la función deseada, hay que seleccionar la porción de red vial que se quiere ampliar en el caso del zoom, o una porción tal que el centro de la misma sea el centro de la próxima porción de red vial a representar y cuya área proporcione el factor de reducción. En la explicación teórica de los procedimientos de zoom y windowing se detallan las relaciones matemáticas que se emplean en cada uno de los casos.

La porción de red vial que se seleccione siempre será rectangular. Para seleccionar dicho rectángulo, se debe pulsar el botón derecho del ratón sobre una de las esquinas del rectángulo a seleccionar. A continuación, sin soltar el botón del ratón, se debe arrastar el puntero del mismo hasta la esquina contraria del rectángulo y soltar el botón que se mantenía pulsado.

Esto quiere decir es que si después de pinchar con el ratón sobre la imagen se desplaza el mismo hacia la derecha y hacia abajo, el punto seleccionado anteriormente se convertirá en la esquina superior izquierda del rectángulo, y el punto donde se libere el botón del ratón se convertirá en la esquina inferior derecha del rectángulo. Por lo tanto, el movimiento del ratón tras la selección del primer vértice del rectángulo determina qué esquina del mismo pasa a ser dicho punto.

Mientras se arrastra el ratón con el botón pulsado, el área seleccionada aparece coloreada en pantalla, para que se tenga siempre noción de la porción de red vial que se está representando. En la figura 5.1.1.3 aparece la imagen tras haber seleccionado la porción de red vial sobre la que se aplicará el zoom o el windowing y sin haber liberado aún el botón del ratón.

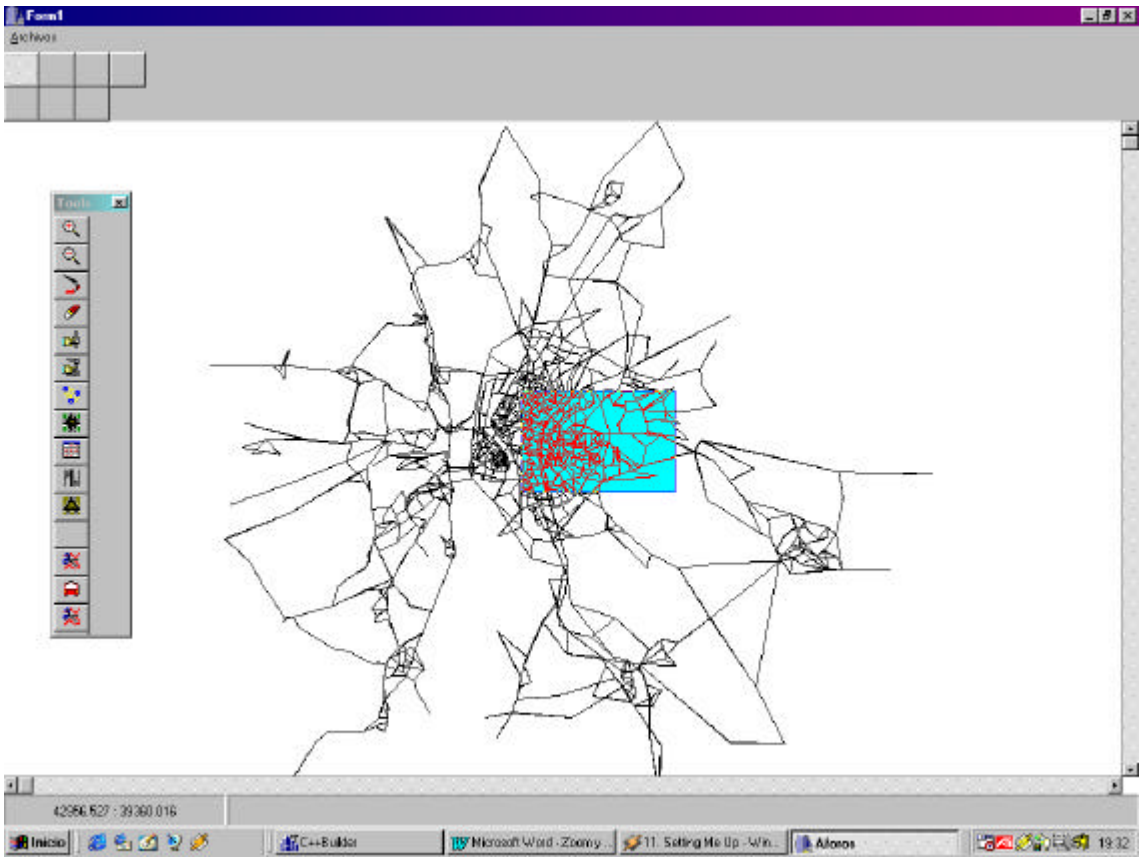


Figura 5.1.1.3: Zona de red vial seleccionada para zoom o windowing

Una vez que se libera el botón del ratón se ejecuta el procedimiento de zoom y windowing.

Estructura de clases necesarias para la representación de imágenes

El diagrama de clases UML ya se mostró en la sección correspondiente. En esta sección se detallan las operaciones más importantes que posee la clase con el fin de definir la implementación de cada uno de los casos de uso de este interfaz.

Es preciso advertir que, dada la gran cantidad de métodos y propiedades de cada clase, sólo se comentarán las propiedades más importantes. Hay ciertos métodos de algunas clases que no tienen uso en este interfaz, pero se comentarán, aunque en menor detalle, porque serán usados por otros interfaces.

La función de la clase *Contexto* es la representación gráfica de la información que contiene la clase *Mapa*. Hay que tener en cuenta que la clase *Limite* pertenece a la clase *Mapa* y la estrecha colaboración entre la clase *Zoom* y *Mapa*

La clase *Contexto* tiene como fin crear la interfaz gráfica necesaria para poder representar la información almacenada en la clase *Mapa* y con las condiciones de contorno también almacenadas en dicha clase. La relación entre ambas clases es extremadamente fuerte, ya que una representa gráficamente la información que la otra le proporciona y con cierto formato también impuesto. Por ello la relación entre estas

clases, como se aprecia en el diagrama de clases, no es una agregación, sino una composición.

La mayoría de las propiedades e incluso métodos de todas las clases que se usan para representar objetos gráficos en pantalla son similares. Por otra parte, no se pretende hacer una descripción exhaustiva de la librería de clases visuales de Borland, por lo que para las clases pertenecientes a dicha librería sólo se hará un breve comentario y los métodos o propiedades interesantes se irán viendo sobre la marcha.

Clase TScrollBar

Pertenece a la VCL de Borland. Sirve para representar barras de desplazamiento.

Clase TBevel

Pertenece a la VCL de Borland. Sirve para crear un pequeño pedestal sobre el que luego se representará la imagen.

Clase TImage

Pertenece a la VCL de Borland. Es la clase que se necesita para poder representar imágenes en un formulario.

Clase TList

Pertenece a la VCL de Borland. Esta clase mantiene una lista de punteros a objetos. Se usa para implementar listas enlazadas sin tener que preocuparse por los punteros. Entre sus propiedades más importantes están:

- *int Count*

Indica el número de objetos que hay en la lista.

- *void * Items[int j]*

Es el puntero el elemento de la lista j-ésimo de la lista, teniendo en cuenta que el primer elemento tiene como índice cero.

Y sus métodos:

- *Add (void * Item)*

Añade un nuevo elemento a la lista

- *Delete(int index)*

Borra el elemento de la lista con el índice *index*.

- *Clear (void)*

Borra todos los elementos de la lista

- *Insert (int index, void Item)*

Añade un elemento a la lista pero en la posición indicada por *index* y desplazando una posición el elemento que anteriormente ocupase ese lugar y todos los posteriores.

- *Pack (void)*

Elimina todos los elementos cuyo valor sea NULL y reordena la lista con los elementos distintos de NULL.

- *Remove (void * Item)*

Elimina el elemento *Item*.

A continuación se detallan las clases que se han desarrollado. En cada clase no aparecen los métodos irrelevantes para este interfaz. En la mayoría de las clases, sobre todo si son muy extensas, se han omitido también los métodos cuya única función es actualizar o devolver valores de las propiedades.

Clase Limite

```
class Limite
{
 float lx1;
 float lx2;
 float ly1;
 float ly2;
 public:
  Limite();
  Limite (float X1,float X2,float Y1,float Y2);
  void Actualiza_Limite (float X1,float X2,float Y1,float Y2);
  float Centro_x ();
  float Centro_y ();
  float X1() {return lx1;}
  float X2() {return lx2;}
  float Y1() {return ly1;}
  float Y2() {return ly2;}
  Limite operator= (Limite lim);
  bool operator!= (Limite lim1);
  bool operator== (Limite lim1);
};
```

Las propiedades *lx1, lx2, ly1* y *ly2* definen el rectángulo que determina la porción de red vial que se visualiza en pantalla. Estos valores están expresados en variables globales o absolutas (no relativas o locales). La pareja de valores *(lx1, ly1)* son las coordenadas del vértice superior izquierdo del rectángulo y la pareja *(lx2, ly2)* son las coordenadas del vértice inferior derecho.

El método *Limite* es el constructor, que está sobrecargado. Cuando se utiliza el constructor *Limite (float X1,float X2,float Y1,float Y2)* se asignan estos valores a las coordenadas, es decir, lo que se hace es:

```
lx1=X1;
lx2=X2;
ly1=Y1;
ly2=Y2;
```

Cuando se usa el constructor *Limite()* se asigna el valor cero a las cuatro coordenadas.

El método *void Actualiza_Limite (float X1,float X2,float Y1,float Y2)* asigna los valores que se le pasan como parámetros a las propiedades *lx1, lx2, ly1* y *ly2* análogamente al constructor que recibe parámetros.

Los métodos *Centro_x* y *Centro_y* devuelven la media de *lx1, lx2* y *ly1, ly2* respectivamente.

Los métodos *X1, X2, Y1* e *Y2* simplemente devuelven los valores de *lx1, lx2, ly1* y *ly2*, respectivamente.

Los tres métodos restantes son sobrecargas para los operadores =,== y !=.

La sobrecarga para el operador= simplemente copia el valor de las cuatro propiedades *lx1, lx2, ly1* y *ly2* del objeto que se le pasa como parámetro.

La sobrecarga del operador== devuelve *true* si las cuatro propiedades *lx1, lx2, ly1* y *ly2* son idénticas a las correspondientes del objeto que se le pasa como parámetro y *false* en caso contrario.

La sobrecarga del operador != devuelve *true* si alguna de las cuatro propiedades *lx1, lx2, ly1* y *ly2* es distinta a las correspondientes del objeto que se le pasa como parámetro, y *false* en caso contrario.

La función de esta clase es básicamente almacenar la porción de red vial que debe representarse en pantalla.

Clase zoom

```
class Zoom
{
 int Xz1;
 int Xz2;
 int Yz1;
 int Yz2;
 bool Activado_caja;
 public:
  void Actualiza_caja (int X1,int Y1,int X2,int Y2);
  void Activar_caja() { Activado_caja=true;}
  void Desactivar_caja() {Activado_caja=false;}
```

```
  bool Estado() { return Activado_caja;}
  void Realizar_zoom (Contexto *origen,Contexto *destino);
  void Realizar_Windowing (Contexto *origen,Contexto *destino);
  int X1() { return Xz1;}
  int X2() { return Xz2;}
  int Y1() { return Yz1;}
  int Y2() { return Yz2;}
};
```

Las propiedades *Xz1, Xz2, Yz1* e *Yz2* definen los límites del rectángulo seleccionado por el usuario para realizar el zoom o el windowing. La propiedad *Activado_caja* indica si el modo zoom o windowing está activado o no.

Como ya es habitual, el rectángulo queda definido por su vértice superior izquierdo e inferior derecho. La pareja de coordenadas *(Xz1, Yz1)* define el vértice superior izquierdo y la pareja *(Xz2, Yz2)* el vértice inferior derecho.

Las funciones miembro *Activar_caja* y *Desactivar_caja* tienen como propósito activar y desactivar el indicador *Activado_caja.* La función *Estado* devuelve el indicador *Activado_caja.*

Las funciones *X1, X2, Y1* e *Y2* devuelven cada una de las coordenadas de los vértices que delimitan el área seleccionada para zoom o windowing.

La función *Realizar_zoom* lleva a cabo todas las operaciones necesarias para calcular la porción de red vial que debe verse tras realizar el zoom. A continuación actualiza los valores de las variables de entorno y dibuja la nueva imagen que se representa en pantalla.

La función *Realizar_windowing* proporciona idéntica funcionalidad pero en este caso se ocupa de la operación de windowing.

Por lo tanto la clase zoom es la que se ocupa de implementar las funciones de zoom y windowing. Necesita colaborar estrechamente con la clase *Mapa* porque, como se verá posteriormente, la clase *Mapa* contiene la información relativa a los nodos y tramos y por tanto necesaria para representar la red vial.

Clases Lista_inf<Punto> y Lista_inf<Tramo>

Estas clases son plantillas del tipo *Lista_inf*, es decir, son listas de objetos. En el primer caso se tiene una lista de objetos de la clase *Punto* y en el segundo la lista es de objetos de la clase *Tramo*.

Estas listas contienen la información relativa a los nodos y tramos. Para saber exactamente qué información aporta cada una de ellas no hay mas que ver la definición de las clases *Punto* y *Tramo*.

```
class Tramo
{
  unsigned int cod_linea;
```

```
  unsigned int cod_punto_origen;
  unsigned int cod_punto_destino;
  unsigned int numero_carriles;
  float X1,Y1;
  float X2,Y2;
  float longitud;
  unsigned char carriles_derecha;
  unsigned char carriles_izquierda;
public:
// Métodos de la clase
};
```

La propiedad *cod_linea* es un entero sin signo único para cada tramo. Sería la clave principal (primary key) de la tabla de tramos de la base de datos.

Las propiedades *cod_punto_origen* y *cod_punto_destino* son los códigos de los nodos de origen y destino del tramo. Se recuerda que cada tramo identifica a un solo sentido de los dos posibles que puede tener una vía. Estas propiedades son claves externas, ya que se corresponden con la clave principal de la tabla de nodos en la base de datos.

Las propiedades *X1*,*Y1*,*X2* e *Y2* son las coordenadas de los nodos origen y destino del tramo. Las coordenadas del nodo origen son (*X1*,*Y1*) y las del nodo destino son (*X2*,*Y2*).

La propiedad *longitud* indica la longitud del tramo, y las propiedades *carriles_derecha* y *carriles_izquierda* indican el número de carriles que adicionales que tiene el tramo.

```
class Punto
{
unsigned int cod_nodo;
 float X;
 float Y;
 unsigned int tipo;
 public:
// Métodos de la clase
};
```

En esta clase hay un identificador único del nodo: *cod_nodo*.

Las propiedades *X* e *Y* son las coordenadas globales del punto. La propiedad *tipo* proporciona información sobre el tipo de nodo.

Llegados a este punto quedan por aún por ver las dos clases más extensas, *Mapa* y *Contexto.* Una vez que se expongan sus propiedades y métodos y se complete así la introducción a cada una de las clases se tratará en detalle el proceso de carga de información de una red vial y su representación y la realización de zoom y windowing con lo que quedará completa la descripción del interfaz gráfico de la aplicación.

Por supuesto que estas mismas clases se usan en otros interfaces tanto para representar información como para interaccionar con el usuario, pero el principio de

funcionamiento será muy similar y en cualquier caso se detallará dentro del interfaz concreto.

Clase Mapa

```
class Mapa
{
 unsigned int cod_mapa;
Limite limite;
float desplazamiento_x;
 float desplazamiento_y;
float entorno;
char nombre_mapa[255];
Lista_inf<Punto> Lista_nodos;
Lista_inf<Tramo> Lista_tramos;
TList *Lista_tramos_nodo;
public:
void Grabar_tramos (char *nombre);
void Grabar_nodos (char *nombre);
void Grabar_mapa (char *nombre_nodo,char *nombre_tramos);
void Leer_nodos(char *nombre_fichero);
void Leer_tramos(char *nombre_fichero);
void Lectura_tramos (char *fichero);
void Lectura_nodos (char *fichero,char *fich_conv,unsigned int numero_centroides);
void Obtener_variable_entorno(float ancho,float alto);
void Pintar_nodos (float Alto,TCanvas *);
void Pinta_tramo (float x1,float y1,float x2,float y2,float Alto,TCanvas *Canvas,int
cod_tramo);
void Pintar_tramos (float Alto,TCanvas *);
void Pinta_punto_seleccionado (Punto p,float Alto,TCanvas *Canvas);
void Add_elemento (Punto p);
void Add_elemento (Tramo l);
void Centrar_imagen (TCanvas *canvas,float Alto,float Ancho);
void Pinta_punto (Punto p,float Alto,TCanvas *Canvas);
void Actualiza_conexiones();
void Prueba_tramo_nodo_cr (unsigned int codigo_nodo);
};
```

La clase *Mapa* incluye una propiedad, que es un objeto de la clase *Limite.* Teniendo en cuenta que la clase *Mapa* es la encargada de almacenar toda la información para representar la red vial, tanto la información de tramos y nodos como la información relativa al entorno, era inevitable que esto ocurriese.

La propiedad *cod_mapa* es el código del mapa.

La propiedad *nombre_mapa* contiene el nombre del mapa.

La propiedad *limite* es donde se almacena qué porción de red vial se representa en pantalla.

Las propiedades *desplazamiento_x* y *desplazamiento_y* se corresponden con la variable de entorno desplazamiento. Como ya se detalló en la parte teórica, estas variables sirven para saber el desplazamiento que hay que sumar a las coordenadas globales de los nodos para que la red vial aparezca representada en el centro de la imagen.

La propiedad *entorno* se corresponde con la variable de entorno llamada asímismo entorno. También se ha comentado ya anteriormente que esta variable era una constante de proporcionalidad entre la porción de red vial a representar y el tamaño de la imagen.

La propiedad *Lista_nodos* es un objeto de la clase *Lista_inf<Punto>*. En efecto, es la lista que contiene toda la información necesaria para la representación de la red vial. Análogamente la propiedad *Lista_tramos* es la lista que contiene toda la información sobre los tramos.

La propiedad *Lista_tramos_nodo* es una lista enlazada, pero no del tipo *Lista_inf*, sino del tipo que incluye Borland, *TList*. Esta lista contiene, a su vez, dos listas para cada nodo. En una de estas listas se almacenan todos los tramos que tienen ese nodo como origen y en la otra se almacenan todos los tramos que tienen ese nodo como destino.

El método *Leer_nodos* lee la información de los nodos de un fichero binario y guarda toda la información en la lista *Lista_nodos*. Además calcula el valor inicial que se le debe asignar al objeto *limite* y se lo asigna.

El método *Leer_tramos* lee la información de los tramos y guarda los datos en la lista *Lista_tramos*.

Los métodos *Add_elemento*, que están sobrecargados, añaden nodos a la lista adecuada según el objeto que se les pase como argumento. Es usado, por ejemplo, por *Leer_nodos* y por *Leer_tramos*.

El método *Actualiza_conexiones* crea tantos *tramos_nodo* como nodos se tienen y los almacena en una lista. Todos estos nodos los lee de la lista *Lista_nodos* que ya debe estar creada.

Rellena la lista *Lista_tramos_nodo* de objetos de la clase *Tramos_nodo* con igual numero de elementos que la lista de nodos, *Lista_nodos*. La clase *Tramos_nodo* tiene un identificador y una lista origen y otra destino. Son listas de *unsigned int*, es decir, almacenan posiciones. Llama al método *Crear_lista* de la clase *Tramos_nodo*, cuya función es crear una lista de nodos origen y otra de nodos destino. Recorre la lista de tramos para obtener el nodo origen y destino de cada tramo y busca el índice de los nodos en la lista de nodos *Lista_nodos*. Ahora , en la lista *Lista_datos_tramos*, añade en los elementos que ocupan las mismas posiciones que los nodos el nodo origen y el destino para el tramo. En concreto, es una lista con los tramos que salen o van hacia ese nodo. Esta lista se usa en el interfaz de tramos. A continuación añade cada *Tramos_nodo* a la lista *Lista_tramos_nodo*.
Así se tiene en cada elemento de la lista *Lista_tramos_nodo* una lista con los tramos de los que es origen y otra lista con los tramos de los que es destino.

El método *Obtener_variable_entorno* calcula la variable de entorno *entorno* y la almacena en la propiedad *entorno*. Necesita como parámetro las dimensiones del objeto *TImage* sobre el que se va a representar la red vial, como se recordará de la explicación teórica.

El método *Centrar_imagen* calcula los valores necesarios de las variables *desplazamiento_x* y *desplazamiento_y* para representar correctamente centrada la red vial en la imagen.

Los métodos *Pintar_tramos* y *Pintar_nodos* son llamados desde el método *Pintar_red* de la clase *Contexto* para representar la red vial en la imagen.

El método al que se llama primero es *Pintar_tramos*. Este método obtiene las coordenadas de los nodos origen y destino de las propiedades *X1,Y1,X2* e *Y2* de la *Lista_tramos*. A estas coordenadas les suma el desplazamiento para pasarlas a la función *Pinta_tramo*. Esta función no es un método de ninguna clase. Su cometido es, en primer lugar, calcular el ángulo que forma el tramo con la horizontal. A continuación calcula los puntos de origen y destino de la línea que representará al tramo, como se explicó en la parte de teoría. Finalmente dibuja la línea que representará al tramo. Estas operaciones las realiza llamando a otras funciones que realizan estos cálculos. No hay que olvidar que los datos que recibe son en coordenadas globales, por lo que también hace la conversión a coordenadas relativas.

A continuación se llama al método *Pintar_nodos*. Este método lo que hace es leer la lista *Lista_nodos* para obtener la información de cada nodo y con esta información llama a la función *Pinta_punto.* Se hacen los cálculos para corregir el desplazamiento y convertir a coordenadas relativas. Una vez obtenidas se dibuja el rectángulo que representará el nodo y si el tamaño lo permite se escribirá dentro de dicho rectángulo el código del nodo.

Estos dos últimos métodos hacen cálculos y hacen uso de otras funciones, pero lo único que se hace es implementar los algoritmos que se detallaron en la sección donde se explica teóricamente el interfaz.

En esta clase hay muchísimos métodos, la mayoría de los cuales no se han comentado. Gran parte de estos métodos tienen como objetivo obtener información de las listas donde se almacena la información sobre tramos nodos y no tienen mayor interés, a pesar de que son absolutamente necesarios para el funcionamiento de la aplicación.

Clase Contexto

```
class Contexto
{
 TImage *imagen_min;
 TImage *imagen_aux;
 TImage *imagen;
 TBevel *borde;
 TScrollBar *horizontal;
 TScrollBar *vertical;
 bool Vis;
```

```
void Destruir_imagen (TImage **imagen);
 void Crea_imagen (TImage **imag,int X1,int Y1,int ancho,int alto,TWinControl *form);
 public:
      Mapa mapa;
      void Pintar_red ();
      void Creacion_borde(int X1,int Y1,int anch,int alt,TForm *form);
      void Pinta_punto_seleccionado (Punto p);
      void Pinta_rectangulo_frame (TCanvas *image,float x1,float x2,float y1,float y2);
      void Pinta_rectangulo (TCanvas *image,float x1,float x2,float y1,float y2,int modo);
      void Copiar_imagen_zoom_directo (float x1,float y1,float x2,float y2);
      void Copiar_imagen_zoom_inverso (float x1,float y1,float x2,float y2);
      void Copiar_imagenes_inverso (float ancho,float alto);
      void Copiar_imagenes_directo (float ancho,float alto);
      void Crea_imagen_principal (int X1,int Y1,int anch,int alt,TWinControl *form);
      void Crea_imagen_auxiliar (int X1,int Y1,int anch,int alt,TWinControl *form);
      void Actualizacion_evento_MouseDown (void __fastcall (__closure *funcion) (TObject *, TMouseButton , TShiftState , int, int ));
      void Actualizacion_evento_MouseUp (void __fastcall (__closure *funcion)(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y));
      void Actualizacion_evento_MouseMove (void __fastcall (__closure * funcion)(TObject *Sender, TShiftState Shift,int X,int Y));
      void __fastcall Seleccionar_giro (TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
      void __fastcall Seleccionar_tramo_final (TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
      void __fastcall Seleccionar_tramo_inicial (TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
      void __fastcall Seleccionar_nodo(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
      void __fastcall Boton_abajo_nulo(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
      void __fastcall Boton_arriba_nulo(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
      void __fastcall Desplazamiento_raton_imagen (TObject *Sender, TShiftState Shift,int X,int Y);
      void __fastcall Caja_zoom_inicio(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
      void __fastcall Caja_zoom_final(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
      void __fastcall Caja_zoom_desplazamiento (TObject *Sender, TShiftState Shift,int X,int Y);
      void __fastcall Caja_zoom_final_m(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
      void Pintar_zona_sel (float x1,float x2,float y1,float y2);
      void Pinta_rectangulo_min (Contexto *context);
      void Centrar_imagen (int x,int y);
      void Centrar_imagen (float x,float y);
  };
```

La propiedad *imagen* es un puntero a un objeto de la *TImage*, es decir, una imagen. La imagen a la que apunte este puntero será la imagen principal, es decir, en la que se representa la red vial después de cargar la información, como la que aparece en la figura 5.1.1.1.

La propiedad *imagen_min* se usará por la clase *Informacion_pantalla* en la representación de los interfaces de tramos, nodos y líneas de bus.

El método *Crea_imagen_principal* llama a *Crear_componente* para crear una imagen, de manera que el puntero a la clase *TImage imagen* contiene la dirección del objeto imagen creado.

La propiedad *imagen_aux* también es un puntero a un objeto de la clase *TImage*.

El método *Crea_imagen_auxiliar* llama a *Crear_componente* para crear otra imagen, de manera que el puntero *imagen_aux* contiene la dirección de la imagen creada. Pero se pone el atributo *Visible* de esta imagen a false porque, como se verá posteriormente, esta imagen se usa para otras funciones.

El método *Destruir_imagen* libera la memoria reservada por la imagen que se le pase como parámetro.

Los objetos *vertical* y *horizontal* son punteros a objetos barras de desplazamiento (*TScrollBar*). De nuevo en la clase aparecen los punteros porque los objetos se crearán en tiempo de ejecución mediante *Crear_componente.*

El método *Generar_scroll* llama a *Crear_componente* para crear dos barras de desplazamiento, una horizontal y otra vertical. Es sencillo imaginar que las propiedades *horizontal* y *vertical* contendrán las direcciones de estos objetos.

El método *Creacion_borde* llama a *Crear_componente* para crear un objeto de la clase *TBevel*. Se recuerda que *TBevel* es un objeto que representa algo así como un pedestal sobre el que luego se creará la imagen.

La propiedad pública *mapa* es un objeto de la clase *Mapa* que almacenará toda la información necesaria para que *Contexto* pueda representar la imagen.

Los métodos *Actualizacion_evento_MouseMove*, *Actualizacion_evento_MouseDown* y *Actualizacion_evento_MouseUp* asignan los métodos que reciben como parámetros a distintos eventos.
Lo que hace el método *Actualizacion_evento_MouseMove* es asociar al evento *imagen->onMouseMove* la función que se le pasa como parámetro. Los otros métodos son análogos pero con los eventos *imagen->onMouseDown* e *imagen->onMouseUp*, respectivamente.

Los métodos *Boton_abajo_nulo* y *Boton_arriba_nulo* no hacen nada, pero tiene su lógica. Se usan para asignar a eventos, de manera que lo que hacen es desactivarlos, es decir, cuando se produce ese evento no ocurre nada.

La clase *Contexto* es el interfaz gráfico para la representación de la red vial. Toda la información necesaria para la representación de la red vial se almacena en la clase *Mapa*. Con los datos de dicha clase *Contexto* proporciona un interfaz de usuario sencillo y funcional.

El método *Pintar_red* llama a los métodos necesarios del objeto *mapa* para representar la red vial en la imagen.

Los métodos *Copiar_imagen_zoom_directo, Copiar_imagen_zoom_directo, void Copiar_imagenes_inverso* y *void Copiar_imagenes_inverso* se usan para copiar imágenes y porciones de imágenes entre las distintas imágenes de la clase: *imagen*, *imagen_aux* e *imagen_min*.

La función *Pinta_punto_seleccionado* será usada por otros interfaces para representar en la imagen adecuada el punto seleccionado por el usuario de una forma fácilmente reconocible. El punto se representa como un círculo inscrito en un cuadrado.

Los métodos *Pinta_rectangulo* y *Pinta_rectangulo_frame* se usan para dibujar los rectángulos de color que posibilitan que el área seleccionada por el usuario aparezca resaltada.

El método *Seleccionar_giro* se usa en el interfaz de nodos, en la parte de giros. Es el método al que se debe llamar cuando el usuario pincha sobre la imagen para ver o modificar los giros que se producen en el nodo que ha seleccionado.

El método *Seleccionar_nodo* es al que se debe llamar cuando un usuario que está en el interfaz de nodos, en la página de nodos, decide seleccionar un nodo para ver o modificar la información relacionada con el mismo.

Los métodos *Pintar_zona_sel* y *Pinta_rectangulo_min* también son funciones auxiliares usadas para la representación de imágenes.

Creación del interfaz gráfico de usuario

Este conjunto de clases que se acaba de estudiar tiene como fin la representación gráfica de la red vial. Como ya es sabido, la representación de la red vial es necesaria en dos escenarios distintos. El primero de ellos es la representación gráfica de la red vial inmediatamente después de cargar la información de los nodos y tramos. Este interfaz presenta una imagen de la red vial sobre la que el usuario puede realizar funciones de zoom y windowing. También incluye una barra de herramientas en la que se puede seleccionar el interfaz con que se desea trabajar.

El segundo escenario aparece cuando en la barra de herramientas anterior se selecciona un interfaz concreto, como puede ser el de nodos, o el de tramos, o el de líneas de bus. En estos casos, en la parte derecha del interfaz aparecen dos imágenes en la que se representan distintas vistas de la red vial.

La implementación del segundo tipo de escenario se explicará más en profundidad cuando se esté tratando el interfaz en concreto. Esta sección se ocupa de describir el proceso de representación del primer tipo de escenario y las operaciones internas que

tienen lugar para poder ofrecer al usuario las funciones de zoom y windowing. En apartados anteriores se estudiaron estos aspectos desde un punto de vista más general, teóricamente, y en esta sección se entra más en profundidad en la implementación.

Sin más dilación, comienza el proceso de creación del interfaz.

Cuando se arranca el programa, se crean dos formularios. Uno de ellos tiene como nombre *Form1* y el otro *Form5*.

La primera función que se ejecuta al arrancar el programa es el método *FormCreate* del formulario *Form1*. Este método está asociado al evento que se produce cuando se crea un formulario, por eso es lo primero que se ejecuta.

Por tanto hay que analizar en primer lugar los pasos que se realizan en esa función.

Tras inicializar algunas variables globales, como por ejemplo, los colores que se usarán, se crean los siguientes objetos relacionados con estas clases: *contexto* de la clase *Contexto* y *zoom* de la clase *Zoom*.

En primer lugar se llama al método *Crear_borde* del objeto *contexto* para crear el "pedestal" sobre el que luego se crearán las imágenes.

A continuación se crea la imagen principal y la auxiliar llamando a los métodos *Crea_imagen_principal* y *Crea_imagen_auxiliar* del objeto contexto. También llama al método *Generar_scroll* de este mismo objeto para crear las barras de desplazamiento.

Y esta función ya no hace nada más.

Pero la acción no se detiene aquí. El formulario conocido por *Form1* tiene una serie de botones que tienen asignadas funciones para cuando sean pulsados.

Lo que ocurre si el usuario pulsa el primer botón que aparece en la barra de herramientas: en primer lugar se leen los dos ficheros necesarios, usando los métodos *Leer_nodos* y *Leer_tramos* del objeto mapa del objeto contexto. También se crea la lista *Lista_tramos_nodos* llamando al método *Actualiza_conexiones* del objeto *mapa* del objeto *contexto.*

Ahora se calcula la propiedad *entorno* usando el método *Obtener_variable_entorno* del objeto *mapa* del objeto *contexto.* También se calculan los valores de las propiedades *desplazamiento_x* y *desplazamiento_y* llamando al método *Centrar_imagen* del objeto *mapa*.

A continuación se llama a los métodos *Actualizacion_evento_MouseMove, Actualizacion_evento_MouseDown* y *Actualizacion_evento_MouseUp* del objeto contexto pasándoles como argumento el método *Desplazamiento_raton_imagen, Boton_abajo_nulo* y *Boton_arriba_nulo,* respectivamente, del mismo objeto, *contexto.*

Los métodos *Boton_abajo_nulo* y *Boton_arriba_nulo* no hacen nada, es decir, cuando el usuario pulse con el ratón sobre la imagen no pasará nada. Y el método *Desplazamiento_raton_imagen* lo que hace es imprimir continuamente el valor de las

coordenadas globales del punto sobre el que está el ratón en la barra de estado de la aplicación.

A continuación se llama a la función *Borra_imagen* para borrar la imagen actual y finalmente se llama al método *Pintar_red* del objeto contexto para representar la red vial en la imagen.

Funciones de zoom y windowing

Al comienzo de esta sección se comentó que al iniciarse la aplicación aparecen dos formularios, uno que ocupa todo el ancho de la pantalla, que se ha denominado *Form1*, y otro que es una barra de herramientas y que se ha llamado *Form5*.

El formulario *Form5* consta de una serie botones. Cada uno de estos botones tiene una función específica. Así, los dos primeros tienen como función activar el zoom y el windowing, respectivamente, y los demás se encargan de cargar otros interfaces de usuario.

En esta sección se detallará el funcionamiento de esos dos primeros botones.

En primer lugar se va a tratar el zoom. ¿Qué ocurre cuando se pulsa el botón de zoom?

Al pulsar este botón se llama a la función *Activacion_zoom*, pasándole como argumento el objeto *contexto* de la clase *Contexto,* es decir, el que se ha estado usando para representar la red vial.

La función *Activación_zoom* asigna ciertas funciones a ciertos eventos realizando para ello llamadas a los siguientes métodos del objeto *contexto*: *Actualizacion_evento_MouseMove, Actualizacion_evento_MouseUp* y *Actualizacion_evento_MouseDown*. A cada una de estas funciones se les ha pasado como parámetro un método del objeto contexto, en concreto, *Caja_zoom_desplazamiento, Caja_zoom_final* y *Caja_zoom_inicio,* respectivamente.

El método *Actualizacion_evento_MouseMove* asigna la función que recibe como parámetro al evento *onMouseMove* sobre el objeto *TImage imagen*. Análogamente, los métodos *Actualizacion_evento_MouseUp* y *Actualizacion_evento_MouseDown* asignan las funciones que reciben como parámetros a los eventos *onMouseUp* y *onMouseDown* del mismo objeto.

Por lo tanto, lo interesante es seguir la pista a estos métodos.

Caja_zoom_inicio es la función que se ejecuta cuando se pincha con el ratón sobre la imagen. Esta función pone a *true* la propiedad *Activado_caja* del objeto *zoom* y, usando el método *Actualiza_caja* de dicho objeto, pone como límites del zoom el punto donde se ha pinchado con el ratón. Es decir, el rectángulo que define el área que está siendo seleccionada por el usuario en estos instantes es en realidad un solo punto, el punto donde el usuario ha pulsado el botón del ratón.

El método *Caja_zoom_desplazamiento* se ejecuta mientras se está desplazando el ratón sobre la imagen. En primer lugar comprueba si el zoom está activado comprobando la

propiedad *Activado_caja* del objeto *zoom*. Si el zoom está activado, copia la porción de red vial limitada por el rectángulo que define el objeto *zoom* y que esté representada en el objeto *imagen_aux* a la porción del objeto *imagen* también limitada por el rectángulo definido por *zoom*. A continuación actualiza las coordenadas del rectángulo definido por *zoom* a las nuevas coordenadas, ya que el ratón ha sido desplazado por el usuario.

Seguidamente hace la operación inversa, es decir, copia la porción de red vial limitada por el rectángulo zoom representada en el objeto *imagen* al objeto *imagen_aux*. A continuación dibuja un rectángulo de color sobre la porción de red vial representada por el rectángulo del objeto *zoom* sobre el objeto *imagen*. Esto borra la porción de red vial representada en esa zona.

Finalmente, esté activada la propiedad de *zoom Activado_caja* o no, se usa el método del objeto *contexto Desplazamiento_raton_imagen* para imprimir las coordenadas globales del punto sobre el que está el ratón en ese instante en la barra de estado de la aplicación.

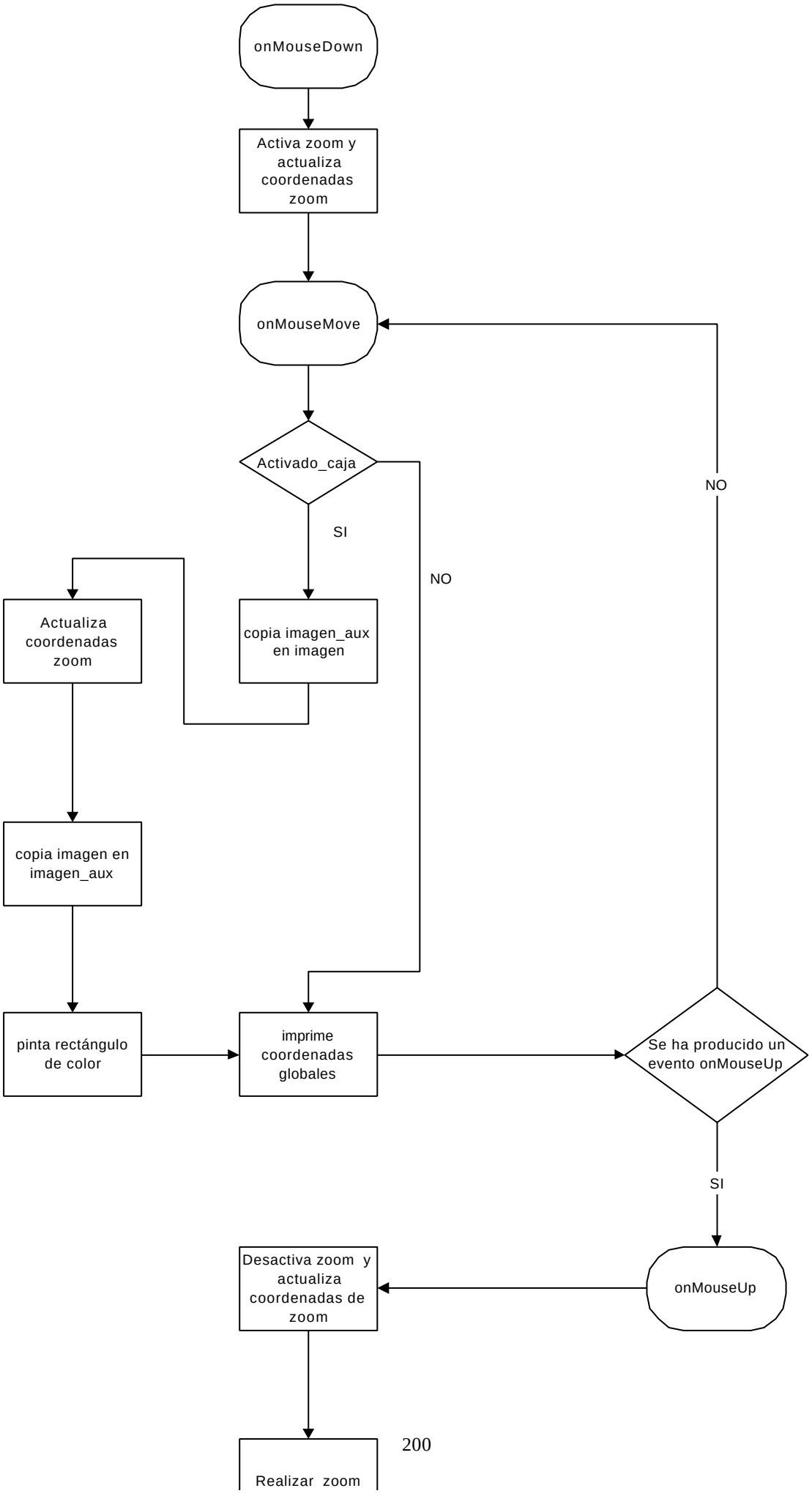
En principio las operaciones que se realizan mientras se está desplazando el ratón sobre la imagen parecen no tener mucho sentido, pero hay que tener en cuenta que este método se ejecutará continuamente mientras el usuario mantenga el botón del ratón pulsado, en este caso sí tienen sentido dichas operaciones.

Resta el método *Caja_zoom_final*. Éste llama a tres métodos del objeto zoom. El primero de ellos para que se actualice el valor del rectángulo que delimita el área sobre la que se hará el zoom. El segundo de ellos para desactivar la propiedad *Activado_caja* del objeto zoom. Esto tiene como fin que una vez liberado el botón el ratón, lo único que provoque el desplazamiento del ratón sobre la imagen sea la impresión de las coordenadas sobre la barra de estado de la aplicación.

Finalmente, el tercer método que se ejecuta es el método *Realizar_zoom* del objeto *zoom*, que realiza todos los cálculos necesarios para realizar el zoom, recalcula las variables de entorno y redibuja la porción de red vial que deba visualizarse.

El procedimiento para realizar el windowing es análogo con la única diferencia de que al evento *onMouseUp* se le asigna el método *Caja_zoom_final_m*. El código de este método es idéntico al del método *Caja_zoom_final* excepto por el hecho de que el tercer método al que se llama es *Realizar_windowing* en lugar de *Realizar_zoom*.

Con el objetivo de que se comprenda mejor el algoritmo que se sigue para realizar el zoom y el windowing se presenta el siguiente diagrama:


onMouseDown
Activa zoom y actualiza coordenadas zoom
onMouseMove
Activado_caja
SI
NO
NO
Actualiza coordenadas zoom
copia imagen_aux en imagen
copia imagen en imagen_aux
pinta rectángulo de color
imprime coordenadas globales
Se ha producido un evento onMouseUp
SI
Desactiva zoom y actualiza coordenadas de zoom
onMouseUp
Realizar zoom

Analizando el diagrama, puede verse hay zonas del mismo que se corresponden con los tres métodos que se usan para cada función, zoom o windowing.

En concreto, desde que se produce el evento *onMouseDown* hasta que llega el evento *onMouseMove* se corresponde con *Caja_zoom_inicio*. Desde que empiezan a producirse los eventos *onMouseMove* hasta que llega el evento *onMouseUp*, se corresponde con *Caja_zoom_desplazamiento*. Finalmente, a partir del evento *onMouseUp* se corresponde con *Caja_zoom_final* o *Caja_zoom_final_m*, dependiendo de si la función es zoom o windowing, ya que el diagrama es válido para los dos.

En primer lugar, cuando el usuario pincha con el ratón sobre la imagen se produce el evento *onMouseDown*. Esto activa el zoom (*Activado_caja* vale *true*) y inicializa el valor del rectángulo definido por las propiedades de *zoom*.

A partir de ahora comenzarán a producirse eventos *onMouseMove* provocados por el desplazamiento del ratón sobre la imagen.

Como previamente se ha producido un evento *onMouseDown*, la propiedad *Activado_caja* estará activada, por lo que se copia *imagen_aux* en *imagen*. La primera vez que ocurre esto no tiene mucho sentido, ya que *imagen_aux* en principio estará vacía. Enseguida se verá por qué se hace esto. Realmente no se copia *imagen_aux* en *imagen*, sino que se copia la porción de imagen limitada por *zoom* del objeto *imagen_aux* en la porción de imagen limitada por *zoom* en *imagen*.

A continuación se actualizan las coordenadas del rectángulo definido por *zoom*. Es necesario porque el ratón ha sido desplazado.

Seguidamente se copia *imagen* en *imagen_aux*. De nuevo sólo se copian los porciones limitadas por *zoom*.

Finalmente, la porción de *imagen* que se acaba de copiar en *imagen_aux* se pierde al dibujar sobre ella un rectángulo de color y se imprimen en la barra de estado las coordenadas globales del punto sobre el que está el ratón.

Ahora viene lo interesante, el ratón sigue moviéndose y se ejecuta de nuevo esta parte de código. Ahora se copia *imagen_aux* en *imagen*. El efecto es que se dibuja la imagen que había justo antes de dibujar el rectángulo de color sobre este mismo rectángulo de color, por lo que el usuario ve la misma imagen que había anteriormente pero sobre el rectángulo de color.

Y sigue el ciclo hasta que se el usuario libere el botón del ratón y se produzca un evento *onMouseUp*. En ese momento ya se tiene la porción de red vial sobre la que se va a realizar el zoom o el windowing y no hay mas que realizarlo tras desactivar la propiedad *Activado_caja* del objeto *zoom*.