ESCUELA TÉCNICA SUPERIOR
DE
INGENIERA INFORMÁTICA

UNIVERSIDAD DE MÁLAGA

Programación en Visual Studio.NET bajo C# de Aplicaciones Gráficas

---

Málaga, Febrero de 2002

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERO TÉCNICO EN INFORMÁTICA DE SISTEMAS

PROGRAMACIÓN EN VISUAL STUDIO.NET BAJO C# DE APLICACIONES GRÁFICAS

Realizado por

RICARDO VILLA BRIEVA

Dirigido por

FRANCISCO R. VILLATORO MACHUCA

Departamento

DEPARTAMENTO DE LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

MÁLAGA, FEBRERO 2002

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERO EN INFORMÁTICA

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente Dº;/Dª._____

Secretario Dº;/Dª._____

Vocal Dº;/Dª. _____

para juzgar el proyecto Fin de Carrera titulado:

Programación en Visual Studio.NET bajo C# de Aplicaciones Gráficas

del alumno Dº;/Dª. **Ricardo Villa Brieva**

dirigido por Dº;/Dª. **Francisco R. Villatoro Machuca**

ACORDÓ POR _____ OTORGAR LA CALIFICACIÓN DE

_____

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL TRIBUNAL, LA PRESENTE DILIGENCIA.

Málaga, a _____de _____de _____

El Presidente El Secretario El Vocal

Fdo: _____ Fdo:_____ Fdo:_____

ÍNDICE GENERAL

CAPÍTULO 1

INTRODUCCIÓN

La programación de aplicaciones gráficas en entornos de ventanas de tipo Windows mediante la programación orientada a eventos se ha facilitado con la aparición de lenguajes y entornos de programación visual. Estas herramientas facilitan el desarrollo de programas en entorno Windows y, como están construidas sobre un lenguaje de programación de propósito general, permite desarrollar aplicaciones muy complejas. Por ejemplo, Delphi basado en Object-Pascal [1], Visual Basic basado en el Microsoft Basic, Visual C++ basado en el Borland C++, etc.

Recientemente, Microsoft ha introducido el entorno de programación visual denominado Visual Studio.NET [2,3], que es independiente de un lenguaje de propósito general y que permite integrar programas realizados en diferentes lenguajes, como por ejemplo, C# [4,5], C++ [9], Basic, etc. Además ha introducido el lenguaje C# que está diseñado para aprovechar todas las posibilidades de .NET

1.1. OBJETIVOS DEL PROYECTO

Para el desarrollo de programas gráficos interactivos en dos dimensiones, como programas de dibujo tipo PaintBrush de Windows, se han venido utilizando bibliotecas de rutinas gráficas como GKS o PHIGS, o las bibliotecas de rutinas gráficas provistas por los desarrolladores de compiladores de lenguajes de programación de propósito general, como las rutinas de Borland C++ para desarrollo de aplicaciones Windows en un entorno orientado a objetos [9].

El objetivo de este proyecto fin de carrera es estudiar las ventajas e inconvenientes de la programación en el entorno Visual Studio.NET para el desarrollo de programas de aplicaciones gráficas interactivos en dos dimensiones utilizando una filosofía de programación orientada a eventos. Para ello, se han estudiado las técnicas de programación visual para el desarrollo de aplicaciones interactivas en entorno Windows, y se ha concretado en uno de los lenguajes más recientes, C# (léase C sostenido o *C sharp*), desarrollado por Microsoft e introducido al mismo tiempo que Visual Studio.

Como ejemplo de aplicación y puesta en práctica de las técnicas de programación visual estudiadas se ha desarrollado una aplicación para el dibujo gráfico basada en píxeles (*raster graphics*), que permitirá el dibujo de múltiples primitivas (líneas, círculos, arcos, polígonos,...), la gestión de sus atributos (grosor, color,...), el rellenado de regiones de píxeles y sus atributos, diferentes tipos de pinceles de dibujo, etc., así como una serie de funciones de entrada/salida de ficheros de imágenes, posibilidad de deshacer (*undo*) de operaciones, etc [6,7]. La aplicación a desarrollar será similar (en cuanto a funcionalidad y presentación) al programa Paint de Windows.

1.2. ESTRUCTURA DE LA MEMORIA

Vamos a detallar el contenido de los sucesivos capítulos que va a contener esta memoria, para facilitar la utilización y comprensión del mismo por parte del lector.

El capítulo segundo trata sobre el estudio de la programación en C#. Se mostrarán las ideas principales de este nuevo lenguaje de programación, así como sus principales características. Mostraremos cuales son las nuevas ideas que aporta al entorno de los desarrolladores y explicaremos por qué surge realmente la necesidad de crear un nuevo lenguaje de programación. A su vez, también estudiaremos el entorno de programación Visual Studio.NET. Observaremos cuales son sus características más importante y, dado el amplio número de áreas en las que se utiliza, nos centraremos en la programación visual (*Windows Forms*).

En el capítulo tercero analizaremos los algoritmos gráficos que hemos realizado para desarrollar la aplicación. Explicaremos por qué hemos decidido implementar los algoritmos gráficos, qué es lo que éstos realizan y veremos algunos códigos para que el lector entienda completamente lo que realiza y cómo lo realiza.

En el capítulo cuarto comentaremos todos los aspectos de la programación de nuestra aplicación gráfica con detenimiento, concentrándonos en los aspectos más dificultosos tales como la realización del zoom y la lupa o la recuperación de imágenes. Todos estos aspectos y muchos más pueden resultar un poco engorrosos y de seguro que con la lectura de este capítulo se le facilitará estos y otros muchos aspectos en torno a la programación gráfica en 2D bajo entorno de ventanas.

El capítulo quinto está dedicado a exponer las conclusiones que me han surgido al finalizar el programa, así como comentar los aspectos y problemas que me han surgido durante el transcurso del mismo y plantear posibles mejoras a la aplicación desarrollada. Dentro de estas mejoras he planteado la construcción de un nuevo proyecto teniendo como

base el actual y que contemplaría las mejoras que se comentarán junto con otras funciones y capacidades.

En el Apéndice A vamos a realizar un estudio de la estructura general de la programación interactiva. Para el desarrollo de nuestra aplicación de gráficos en 2D es necesario tener en cuenta los llamados factores humanos o "*look and feel*" y la facilidad de aprendizaje y uso. Estos factores son tan importantes como la corrección y comprensión funcional de la herramienta a desarrollar. Con relación a la aplicación a desarrollar se estudiarán los factores más importantes necesarios para obtener una buena interacción hombre-máquina en un entorno de ventanas. Así también profundizaremos en los eventos que es necesario programar en nuestra aplicación para llegar al resultado final. Es necesario tener claro la función de cada uno para así poder lograr una interacción más buena.

El Apéndice B está dedicado a comentar solamente ciertos aspectos sobre el contenido del CD-ROM que acompaña a la memoria. Es un simple apéndice acerca de la instalación, ejecución de mi programa y contenido de los ficheros del proyecto junto con su función en el mismo.

CAPÍTULO 2

.NET Y C#

En este capítulo trataremos el estudio de la programación en el nuevo lenguaje C# [4,5]. Explicaremos en que consiste la plataforma .NET y, con ello, la herramienta de desarrollo Visual Studio.NET [2,3]. Se mostrará las ideas principales del nuevo lenguaje de programación C#, así como sus principales características. Indicaremos cuales son las nuevas ideas que aporta al entorno de los desarrolladores y explicaremos por qué surge realmente la necesidad de crear un nuevo lenguaje de programación.

2.1. MICROSOFT.NET

.NET no es más que la nueva generación de productos orientados al mundo de Internet y al de los dispositivos móviles. .NET es anunciado por Microsoft como una nueva era en el desarrollo de aplicaciones tanto en entorno Windows, como en el resto de entornos que la soporten en un futuro, y la presenten como el estándar para Internet y todos los dispositivos que a ella se conecten.

Las características que presenta la propuesta del gigante de la informática son realmente prometedoras y, si el mercado las acepta, podría finalmente convertirse en el soporte definitivo para una Internet más homogénea e interactiva. En este sentido, el monopolio estadounidense va a invertir en los próximos dos años tres mil millones de dólares americanos para promover actividades I+D en empresas con el objetivo de fomentar el uso y difusión de .NET tanto a nivel de usuario como a nivel de desarrollo de software.

2.1.1. LA WEB PROGRAMABLE

Con .NET Gates persigue llevar a buen término dos ideas que han estado siempre presentes en la política comercial de su empresa. En primer lugar, Microsoft en todos lados, y en segundo, concebir una red con interconexión total desde cualquier lugar y mediante cualquier dispositivo. Con este objetivo en .NET Internet aparece como la base de un sistema operativo distribuido sobre el cual se ejecutarán aplicaciones que estarán preparadas para relacionarse entre sí de manera transparente.

En la actualidad las aplicaciones orientadas a Internet son como islas digitales que en contadas ocasiones tienen contacto o relación con el archipiélago digital que las rodea. Lo sitios Web en muchas ocasiones presentan la información en un formato estático, nada uniforme cuando proviene de distintas fuentes y difícilmente configurable y modificable por el usuario.

Con .NET la Web estará compuesta por una serie de servicios distribuidos, desarrollados por distintas empresas incluida la propia Microsoft, que colaborarán entre si para proporcionar a los navegantes una información consistente, uniforme, configurable y de fácil manejo. La programación del futuro se hará sobre un gran sistema operativo que residirá en Internet de forma que la información y las aplicaciones, servicios en este caso, ya no estarán en nuestro PC, sino en la Red.

2.1.2. CARACTERÍSTICAS DE MICROSOFT.NET

2.1.2.1. UTILIZACIÓN DE XML

La información procedente de distintos servicios suele tener formatos totalmente distintos de forma que el intercambio de información entre ellos supone un serio problema. Para salvar este contratiempo en .NET se utiliza el lenguaje XML (Extensible Markup Language). Toda la información que tenga que fluir de un servicio a otro será descrita con XML con el fin de establecer la correspondencia entre el formato de la información de cada servicio y los demás. Además, la comunicación entre servicios se realizará utilizando el protocolo SOAP (Simple Object Access Protocol) que, como es de esperar, también está basado en XML y que permite invocar métodos, servicios, componentes y objetos de otros servicios remotos.

2.1.2.2. NUEVAS INTERFACES DE USUARIO

.NET nos ofrecerá una nueva generación de interfaces de usuario más intuitivas y fáciles de manejar. Con el fin de conseguir estos objetivos Microsoft también se ha basado en la tecnología XML y nos proporciona dos elementos destacables:

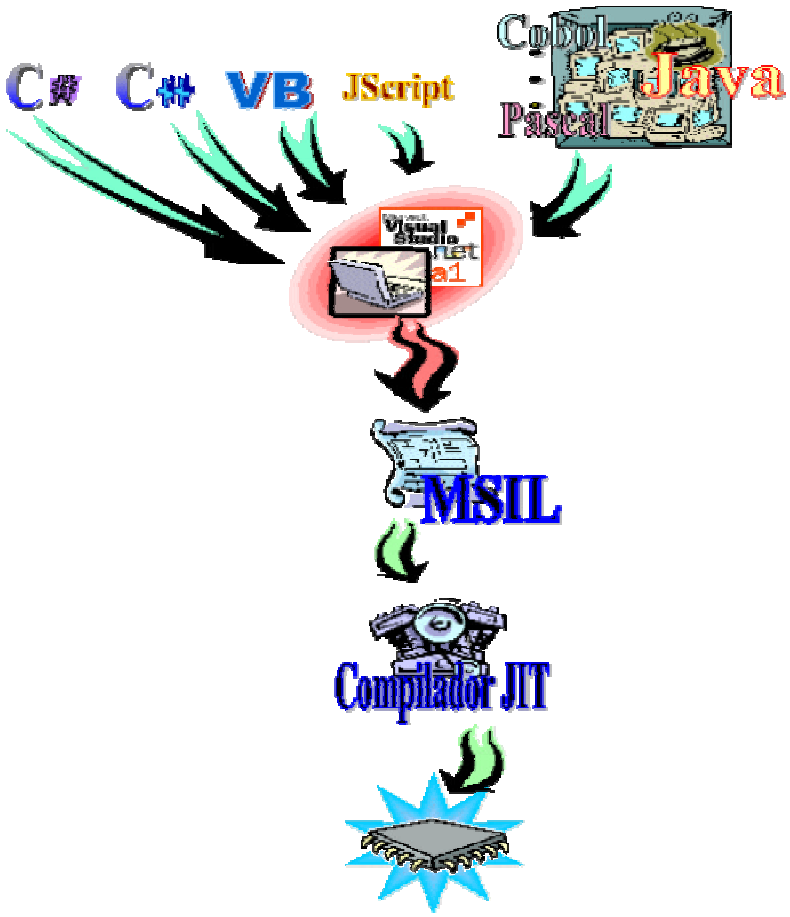
Por un lado, nos van a proporciona una interfaz natural de usuario que consistirá en una interfaz que facilitará la interacción hombre-máquina proporcionando capacidades de interacción a través de la voz, vista, escritura a mano y lenguaje natural. Por otro lado, está el llamado “Lienzo Universal”. Como ya hemos dicho, está basado también en XML, y

permitirá tener en una misma ventana integrado el navegador, las aplicaciones de correo, las de tratamiento de documentos, etc., todo ello con un formato homogéneo y totalmente configurable. Su aspecto está basado en tecnología de *skins*, es decir, pieles intercambiables como las que se utilizan en programas como Winamp que permiten cambiar por completo el aspecto de la ventana. Este lienzo universal hará posible, por ejemplo, que nuestra agenda se actualice con la hora del partido de nuestro equipo favorito sin tener que visitar su página web todas las semanas y cambiarla nosotros a mano.

2.1.2.3. CARACTERÍSTICAS TÉCNICAS DE .NET EN EL DESARROLLO DE APLICACIONES

En esta sección trataremos de las características relacionadas con el desarrollo de software. Con este objetivo, Microsoft ha construido una nueva herramienta para el desarrollo rápido de aplicaciones y servicios web llamada Visual Studio.NET que aún está en versión beta. En esta nueva versión del Visual Studio se incluyen los lenguajes Visual Basic, JScript, C++ y un nuevo lenguaje cuyo objetivo es sustituir a Java llamado C# sobre el que hablaremos en el apartado 2.2. Además, hay trabajando un conjunto de universidades y empresas privadas para incluir en .NET nuevos lenguajes de programación entre los que se encuentran APL, CAML, Cobol, Haskell, Mercury, MI, Oyeron, Oz, Pascal, Perl, Python, Écheme, Smaltalk e incluso Java en un proyecto liderado por la empresa Racional.

Lo más atractivo quizás de este nuevo entorno de desarrollo es la capacidad de construir por ejemplo, parte de nuestra aplicación en Visual Basic y el resto en C++, o incluso aprovechar una aplicación antigua sin tener que preocuparnos por la integración, con lo que facilitamos en gran medida la reutilización de código.



Para conseguir la integración, Microsoft se ha basado en la misma idea que utiliza Java para conseguir la portabilidad. La idea es construir una máquina virtual de forma que todo el código que escribamos se traduzca a un lenguaje intermedio común para todos los lenguajes, en este caso MSIL (*Microsoft Intermediate Lenguage*) y en el caso de Java los bytecodes, que será el que la máquina virtual ejecutará. El principal problema de Java es la eficiencia y este está derivado sin duda de la utilización de una máquina virtual. En este sentido Microsoft anuncia que la perdida de eficiencia en .NET va a ser sólo de un 10% aproximadamente, pero aún es pronto para hablar de eficiencia y tendremos que esperar a la versión definitiva del Visual Studio.NET para observar estos resultados. Para realizar esta afirmación, Microsoft se apoya en varios aspectos:

1. Por un lado, MSIL es de más alto nivel que otros lenguajes máquinas: trata directamente con objetos y tiene instrucciones para crearlos e inicializarlos

directamente, puede llamar a métodos virtuales sobre objetos y manipular elementos de tablas directamente. Incluso tiene instrucciones para el manejo de excepciones. De esta forma que la máquina virtual debe ser más eficiente que la JVM (*Java Virtual Machine*).

2. Por otro lado, los compiladores utilizados son JIT (*Just In Time*) que producen, o producirán, código nativo optimizado para el microprocesador que estemos usando, no compatible con toda la familia x86 como hacen otros JITs, de modo que al usar todas las capacidades de nuestro micro (registros, instrucciones propias del micro que usemos, etc.) hacemos que el código se ejecute más rápido. En .NET hay tres tipos de compiladores JIT, los dos primeros no permiten generar código nativo sin optimizar y optimizado. Estos JITs compilan a código nativo bajo demanda y una vez una parte ya está compilada, se guarda para utilizarla posteriormente evitando tener que volver a traducirla. El tercer tipo es un compilador de los de toda la vida que compila a nativo todo el código y guarda el resultado en el disco.

2.1.2.4. EL NUCLEO DE .NET. EL COMMON LANGUAGE RUNTIME (CLR)

Para construir una máquina virtual común a todos los lenguajes primero hay que definir una serie de características comunes a todos ellos. Este conjunto de características comunes se engloban en el llamado *Common Language Runtime (CLR)*. A continuación veremos algunas de las características que contempla el CLR:

- Es Orientado a objetos, no podría ser de otro modo. Soporta herencia (simple), polimorfismo, manejo de excepciones, etc.
- Posee un conjunto de tipos común estandarizado, el "*Common Type System*" *(CTS)*. Esto elimina la ambigüedad en el sentido de que un entero se represente con 16 bits o 32, o que se soporte o no la herencia múltiple.
- Soporta metainformación para todos los tipos en tiempo de ejecución.
- Posee un conjunto de clases pertenecientes a .NET que encapsulan la mayoría de la funcionalidad del API win32 y otras tecnologías, como XML, etc.

- Proporciona opciones avanzadas para depurar aplicaciones. Proporciona un depurador que actúa entre lenguajes distintos, que nos permite recorrer la pila, etc.
- Proporciona ejecución virtual de código y manejo de memoria automático.
- Posee un recolector de basura.
- Proporciona traductores de lenguaje intermedio a código nativo.
- Y tiene un modelo de seguridad declarativo.

El código que cumple con las restricciones del CLR se llama código "manejado", "*managed code*" en inglés. Todos los lenguajes que incluye .NET producen código manejado menos el C++ que posee características que se salen de las especificaciones, por ejemplo, el C++ permite la herencia múltiple que en el CLR no se permite. Para solucionar este problema se utilizan las llamadas extensiones de manejo (Manager Extensión) que hacen que el código que escribamos en C++ se ajuste a CLR.

Al tener un runtime común para todos los lenguajes, si escribimos dos clases con la misma funcionalidad, una de ellas codificada en Visual Basic y la otra codificada en C#, su traducción a lenguaje intermedio, es decir, a MSIL, será exactamente la misma.

2.1.2.5. EL NUEVO MODELO DE COMPONENTES: COM+

COM es el modelo de componentes de Microsoft y en el mundo Windows es, sin duda alguna, el más utilizado ya que permite interconectar componentes software y aplicaciones de una manera "simple", segura y homogénea. COM+ es la nueva versión de COM que Microsoft incluye en .NET y que pretende solucionar alguno de los problemas que presenta el antiguo COM.

Por supuesto, es posible usar COM en lenguajes como C, C++ o incluso Java. El problema es la gran complejidad de hacerlo. Cualquier programador con cierta experiencia en estas tecnologías sabe lo engorroso y propenso a errores que es desarrollar programas COM. En COM+ se han eliminado las tediosas interfaces que tenía COM, eliminando de este modo elementos como: IUnknows, MIDL, AddRef, Release, HRESULT, etc., se ha eliminado el tener que tratar con el Registro de Windows, los errores a partir de ahora se

tratarán con excepciones que se pueden propagar incluso entre los distintos lenguajes de programación, etc.

En .NET la utilización de COM+ es muy simple ya que todos los objetos son componentes COM+ por defecto. Por otra parte, utilizar dentro de nuestras aplicaciones .NET herramientas como MS Word, PowerPoint o Corel Draw es un juego de niños ya que la integración entre .NET y éstas es perfecta.

El COM en .NET se queda obsoleto, lo que quiere decir que .NET no está construido sobre COM. Según dice Microsoft esto no se ha llevado a cabo elevando el nivel de abstracción y dejando el manejo de estos detalles al framework, sino que en esta plataforma simplemente estos conceptos ya no existen.

Una de las afirmaciones más impactantes y atractivas a la vez es que Microsoft anuncia que se ha terminado el llamado incluso por ellos mismos “infierno de las DLLs” (*DLL hell*). Con esto se refieren a que se acabaron los problemas de instalación/desinstalación que tantos quebraderos de cabeza nos dan a los usuarios de la plataforma Windows. Un ejemplo de este tipo de problemas puede ser que el Microsoft Word deje de funcionar correctamente cuando instalemos la nueva impresora que acabamos de comprar. Nuestra impresora trae una librería (DLL) para su interfaz y al sobrescribir la nueva versión a la antigua el Word deja de funcionar correctamente. Con el fin de evitar este tipo de problemas, Microsoft proporciona un nuevo “motor de instalación” y una nueva política de gestión de versiones de las DLLs configurable por el propio desarrollador que según ellos solucionará estos problemas de una vez por todas. Ahora los componentes .NET no están referenciados en el registro de Windows, por lo que instalar una aplicación sólo requiere copiarla y desinstalarla sólo requiere borrar el directorio.

Con respecto a la reutilización en la que tanto redunda Microsoft, si sólo utilizamos el nuevo modelo COM+ ¿Qué pasa entonces con todo el software que está ya construido usando COM?. Para poder utilizar COM desde .NET hay que ayudarse de una herramienta

que proporciona el SDK que dado un objeto COM nos construye un wrapper o envoltorio a COM+ para hacer su utilización transparente. La construcción de este envoltorio es en principio sencilla, sólo hay que ejecutar el comando que lo construye. Pero en algunas ocasiones no es posible construirlo de manera automática y es aquí donde hay que tratar con COM y establecer el mapeo entro uno y otro de manera manual. Además, también podemos hacer envoltorio en el otro sentido, es decir de COM+ a COM, para desde aplicaciones antiguas hacer uso de los componente COM+ construidos sobre .NET.

2.1.2.6. EL FUTURO

La Internet de .NET será algo más interactivo en la que todas nuestras necesidades informáticas quedarán cubiertas a través de ella. En la Internet del futuro no dependeremos, como en la actualidad, de que cierta información esté en el PC de la oficina o en el PC de casa, la información estará en Internet y no tendremos que realizar la frustrante tarea de por ejemplo, aprender a manejar distintos gestores de correo según donde nos encontremos.

Para conseguir esto, Microsoft nos ofrecerá una serie de servicios que estarán disponibles para los usuarios, entre los que podemos encontrar: Un servicio para identificarnos que utiliza Microsoft Passport ya disponible, un servicio de notificación y mensajería que integrará correo, fax, mensajes de voz, etc., un servicio de personalización con el que indicaremos reglas para tratar las notificaciones, mensajes que nos lleguen, tratamiento de nuestros datos, etc., un servicio de almacén de datos en XML con la flexibilidad que esto proporciona, un servicio de calendario para indicar cuando estamos disponibles y para quien, un servicio de directorio de búsqueda, etc.

Para el mundo de la empresa también dilucida un futuro prometedor en el que se abre un amplio campo para el desarrollo de aplicaciones de nueva generación y para reutilización y/o ampliación de aplicaciones ya construidas. En uno de los ámbitos que sin duda tendrá mayor auge es en el del comercio electrónico donde la Web basada en servicios supondrá sin duda toda una revolución.

2.2. C#

C# (pronunciado en inglés "*C Sharp*", en español "C Sostenido", y más popularmente como "C Almohadilla") es el nuevo lenguaje diseñado por Microsoft para su plataforma .NET. En concreto, ha sido diseñado por Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Aunque en realidad es posible escribir código para la plataforma .NET en muchos otros lenguajes, como Visual Basic.NET o JScript.NET, C# es el único que ha sido diseñado específicamente para ser utilizado en esta plataforma, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes. Por esta razón, Microsoft suele referirse a C# como el lenguaje nativo de .NET, y de hecho gran parte de la librería de clases base de .NET ha sido escrito en este lenguaje.

C# es un lenguaje orientado a objetos sencillo, moderno, amigable, intuitivo y fácilmente legible que ha sido diseñado por Microsoft con el ambicioso objetivo de recoger las mejores características de muchos otros lenguajes, fundamentalmente Visual Basic, Java y C++, y combinarlas en uno sólo en el que se unan la alta productividad y facilidad de aprendizaje de Visual Basic con la potencia de C++.

Aunque actualmente no se consiguen prestaciones en velocidad tan altas como usando el C++ tradicional, Microsoft asegura que en futuras versiones se irá aumentando la eficiencia del lenguaje, gracias fundamentalmente a la posibilidad de generar dinámicamente código optimizado para aprovechar las características de la máquina sobre la que se ejecute el programa.

Quizás el más directo competidor de C# es Java, lenguaje con el que guarda un enorme parecido en sus sintaxis y características. En este aspecto, es importante señalar que C# incorpora muchos elementos de los que Java carece (sistema de tipos homogéneo, propiedades, indexadotes, tablas multidimensionales, operadores redefinibles, etc.) y que

según los benchmarks realizados la velocidad de ejecución del código escrito en C# es ligeramente superior a su respectiva versión en Java.

A continuación se recoge de manera resumida las principales características de C#. No vamos a detallar completamente todas las características del lenguaje, sino que nuestra intención es mostrar una visión general del lenguaje:

- Dispone de todas las características propias de cualquier lenguaje orientado a objetos: encapsulación, herencia y polimorfismo.
- Ofrece un modelo de programación orientada a objetos homogéneo, en el que todo el código se escribe dentro de clases y todos los tipos de datos, incluso los básicos, son clases de que heredan de *System.Object* (por lo que los métodos definidos en ésta son comunes a todos los tipos del lenguaje).
- Permite definir estructuras, que son clases un tanto especiales; sus objetos se almacenan en pila, por lo que se trabaja con ellos directamente y no referencias al montículo, lo que permite accederlos más rápido. Sin embargo, esta mayor eficiencia en sus accesos tiene también sus inconvenientes, fundamentalmente que el tiempo necesario para pasarlas como parámetros a métodos es mayor (hay que copiar su valor completo y no sólo una referencia) y no admiten herencia (aunque sí implementación de interfaces).
- Es un lenguaje fuertemente tipado, lo que significa que controla todas las conversiones entre tipos se realicen de forma compatible, lo que asegura que nunca se acceda fuera del espacio de memoria ocupado por un objeto. Así se evitan frecuentes errores de programación y se consigue que los programas no puedan poner en peligro la integridad de otras aplicaciones.
- Tiene a su disposición un recolector de basura que libera al programador de la tarea de tener que eliminar las referencias a objetos que dejen de ser útiles, encargándose de ello éste y evitándose así que se agote la memoria porque al programador se le olvide liberar objetos inútiles o que se produzcan errores porque el programador libere áreas de memoria ya liberadas y reasignadas.

- Incluye soporte nativo para eventos y delegados. Los delegados son similares a los punteros a funciones de otros lenguajes como C++ aunque más cercanos a la orientación a objetos, y los eventos son mecanismos mediante los cuales los objetos pueden notificar de la ocurrencia de sucesos. Los eventos suelen usarse en combinación con los delegados para el diseño de interfaces gráficas de usuario, con lo que se proporciona al programador un mecanismo cómodo para escribir códigos de respuesta a los diferentes eventos que puedan surgir a lo largo de la ejecución de la aplicación (pulsación de un botón, modificación de un texto, etc.).
- Incorpora propiedades, que son un mecanismo que permite el acceso controlado a miembros de una clase tal y como si de campos públicos se tratasen. Gracias a ellas se evita la pérdida de legibilidad que en otros lenguajes causa la utilización de métodos *Set()* y *Get()* pero se mantienen todas las ventajas de un acceso controlado por estos proporcionada.
- Permite la definición del significado de los operadores básicos del lenguaje (+, -, *, &, ==, etc.) para nuestros propios tipos de datos, lo que facilita enormemente tanto la legibilidad de las aplicaciones como el esfuerzo necesario para escribirlas. Es más, se puede incluso definir el significado del operador [ ] en cualquier clase, lo que permite acceder a sus objetos tal y como si fuesen tablas. A la definición de éste último operador se le denomina indicador, y es especialmente útil a la hora de escribir o trabajar con colecciones de objetos.
- Admite unos elementos llamados atributos que no son miembros de las clases sino información sobre éstas que podemos incluir en su declaración. Por ejemplo, indican si un miembro de una clase ha de aparecer en la ventana de propiedades de Visual Studio.NET, cuáles son los valores admitidos para cada miembro en ésta, etc.

CAPÍTULO 3

ALGORITMOS GRÁFICOS

En el capítulo tercero, que ahora comienza, vamos a ver los algoritmos para realizar las principales operaciones sobre gráficos [6,7]. En primer lugar, explicaremos los motivos que nos llevaron a implementar las rutinas básicas y a continuación veremos los códigos de éstos. Los algoritmos, tal y como están implementados en la aplicación, son de fácil entendimiento por lo que consideramos que no necesitan demasiadas explicaciones ni ejemplos para entenderlos. Se estudiarán las técnicas de dibujo gráfico basada en píxeles *(raster graphics)*, que permitirá el dibujo de múltiples primitivas (líneas, círculos,...), la gestión de sus atributos (grosor, color,...), el rellenado de regiones de píxeles y sus atributos,...

3.1. CONSIDERACIONES INICIALES

El entorno de programación Visual Studio.NET posee varios espacios de nombres destinados al trabajo sobre gráficos. Estos espacios de nombres incluyen muchas clases de objetos para trabajar con imágenes dependiendo del tipo de éstas y sus funciones.

Para realizar la aplicación Sketcher se tuvo que hacer un pequeño estudio para analizar cuál era la clase de objeto más idónea a las características que requería la aplicación. A primera vista y por su definición, la clase de objeto Graphics (*System.Drawing.Graphics*) era la ideal ya que poseía entre sus métodos todas las funciones gráficas deseadas y muchas más. Con ella, no se requería implementar los algoritmos gráficos porque ya venían incluidos entre sus métodos y, por tanto, la aplicación sería óptima y muy funcional. El principal problemas es que el PictureBox que empleamos para mostrar la imagen en la interfaz tiene su propiedad Image, es decir, el objeto que contiene la imagen que queremos mostrar, de un tipo que es incompatible con la clase de objeto Graphics. Podríamos utilizar la clase Graphics para trabajar en un nivel superior, es decir, como una fina capa por encima que realizara las operaciones gráficas, pero esto tendría un gran inconveniente a la hora de salvar las imágenes resultantes ya que al estar la imagen inicial por un lado y las operaciones gráficas en un nivel superior, no hay manera de guardar los resultados en un único fichero ya que no podemos mezclar ambas clases de objetos.

Es por este motivo, principalmente, por el que la elección fue finalmente realizar la aplicación con la clase de objeto Bitmap (*System.Drawing.Bitmap*). Esta clase es la que trabaja con imágenes en formato bitmap, jpeg, etc. Aunque es más difícil trabajar ya que no posee demasiados métodos para dibujar sobre la imagen, es la que nos permite modificarla directamente y, por ende, guardarla con los cambios introducidos. Esta clase trabaja con la imagen como un array bidimensional de píxeles, sea cual sea su formato.

En los siguientes cuadros se muestran todos los métodos y propiedades que posee la clase de objeto Bitmap

MÉTODOS DE LA CLASE DE OBJETO BITMAP	
Clone	***CreateObjRef***
Dispose	***Equals***
GetBounds	***GetEncoderParameterList***
GetFrameCount	***GetHashCode***
GetLifetimeService	***GetPixel***
GetPropertyItem	***GetThumbnailImage***
GetType	***InitializaLifetimeService***
LockBits	***MakeTransparent***
RemovePropertyItem	***RotateFlip***
Save	***SaveAdd***
SelectActiveFrame	***SetPixel***
SetPropertyItem	***SetResolution***
ToString	***UnlockBits***

PROPIEDADES DE LA CLASE DE OBJETO BITMAP	
Flags	***FrameDimensionList***
Height	***HorizontalResolution***
Palette	***PhysicalDimension***
PixelFormat	***PropertyIdList***
PropertyItems	***RawFormat***
Size	***VerticalResolution***
Width	

De los métodos que implementa la clase Bitmap hay que destacar *SetPixel*, que es el encargado de asignar un color determinado a un píxel del objeto. Su estructura es la siguiente:

```
void Bitmap.SetPixel (int x, int y, System.Drawing.Color color);
```

También hay que mencionar el método *GetPixel* que es el que devuelve el color correspondiente a una posición especificada del Bitmap.

```
System.Drawing.Color  Bitmap.GetPixel (int x, int y);
```

Con estos métodos, principalmente con el primero, nos basamos para realizar los algoritmos que realizan los dibujos sobre las imágenes. En la siguiente sección, se mostrarán los mencionados algoritmos. Estos algoritmos trabajan con píxeles para realizar los dibujos requeridos. Es decir, vamos asignando píxeles con el color predefinido para realizar el dibujo seleccionado. Los dibujos se realizan directamente sobre el mapa de bits y, por tanto, cuando guardamos éste, guardamos ya los cambios realizados.

3.2. ALGORITMOS GRÁFICOS

A continuación, se explicarán los algoritmos que realizan las diversas operaciones. Son operaciones triviales sobre gráficos y la mayoría de sus códigos se pueden encontrar en algunos libros especializados en el tema. Además, son de fácil entendimiento y, es por ello, por lo que no se van a explicar en demasía; y sólo comentaremos su función y la cabecera.

3.2.1. FUNCIÓN PUNTO

Esta función dibuja un punto en la imagen cuando se le pasa la coordenada en la que se quiere dibujar. Se basa básicamente en el método SetPixel, aunque dependiendo de la variable grosor (“Fino”, “Medio”. “Grande”), dibuja un punto de un único píxel, de cinco píxeles de diámetro o de diez píxeles de diámetro. Es la referencia para las demás funciones ya que éstas no llamaran al método SetPixel sino que se aprovecharán de la función Punto.

```
private void Punto (int x, int y)
```

3.2.2. FUNCIÓN LINEA

Esta función dibuja una línea recta que empieza en el punto x (pasado como coordenadas xa y xb) y acaba en el punto y (pasado como coordenadas ya y yb).

```
private void Linea (int xa, int xb, int ya, int yb)
```

Vamos a mostrar el código de esta función para mostrar lo que realmente realiza la función en la aplicación. Las demás funciones trabajan de manera análoga aunque realizando la función que le corresponde.

```
private void Linea (int xa, int ya, int xb, int yb)
{
     int dx = xb - xa, dy = yb - ya, steps, k;
     float xIncrement, yIncrement, x = xa, y = ya;

     if (Abs(dx) > Abs(dy)) steps = Abs(dx);
     else steps = Abs(dy);
     xIncrement = dx / (float) steps;
     yIncrement = dy / (float) steps;

     Punto(ROUND(x), ROUND(y));
     for (k=0; k<steps; k++)
     {
          x += xIncrement;
          y += yIncrement;
          Punto(ROUND(x), ROUND(y));
     }
}
```

El código anterior realiza el dibujo de la función línea entre dos puntos ya que realiza pequeños incrementos de la variable x según crezca la variable y. Va calculando esos incrementos y va dibujando los puntos en las coordenadas calculadas.

Esta función además de Punto, como hemos comentado anteriormente, emplea funciones como Abs, que calcula el valor absoluto, y ROUND, que redondea al entero más próximo un real al que se le ha sumado 0.5. Se debe destacar más que las operaciones realizadas para calcular la línea, como se llama a la función Punto, que es la verdadera encargada de dibujar sobre la imagen.

3.2.3. FUNCIÓN RECTÁNGULO SIN RELLENO

Esta función dibuja un rectángulo cuyo extremo superior izquierdo es el punto x (pasado como coordenadas xa y xb) y su extremo inferior derecho es el punto y (pasado como coordenadas ya y yb)

private void RectanguloSin (int xa, int xb, int ya, int yb)

3.2.4. FUNCIÓN RECTÁNGULO CON RELLENO

Esta función dibuja un rectángulo relleno del color seleccionado cuyo extremo superior izquierdo es el punto x (pasado como coordenadas xa y xb) y su extremo inferior derecho es el punto y (pasado como coordenadas ya y yb)

private void RectanguloCon (int xa, int xb, int ya, int yb)

Este algoritmo no es como el anterior que se basaba en la función Linea, sino que se encarga de recorrer con un bucle toda la superficie.

3.2.5. FUNCIÓN CÍRCULO SIN RELLENO

Esta función dibuja un círculo cuyo centro es pasado por las coordenadas xCentro e yCentro y su radio por la coordenada del mismo nombre.

private void CirculoSin (int xCentro, int yCentro, int radio)

Esta función se apoya en la función circlePlotPoints que se encarga de dibuja la misma coordenada en los cuatro cuadrantes.

3.2.6. FUNCIÓN CÍRCULO CON RELLENO

Esta función dibuja un círculo relleno del color seleccionado cuyo centro es pasado por las coordenadas xCentro e yCentro y su radio por la coordenada que lleva el mismo nombre.

private void CirculoCon (int xCentro, int yCentro, int radio)

Esta función se apoya en la función circlePlotPointsCon que se encarga de dibujar líneas entre cuadrantes dos a dos dada una coordenada.

3.2.7. FUNCIÓN ELIPSE

Esta función dibuja una elipse cuyo centro es pasado por las coordenadas xCentro e yCentro , su radio respecto al eje X por la coordenada Rx y su radio respecto al eje Y por la coordenada Ry.

private void Elipse (int xCentro, int yCentro, int Rx, int Ry)

Esta función se apoya en la función ellipsePlotPoints que se encarga de dibuja la misma coordenada en los cuatro cuadrantes según el radio correspondiente

3.2.8. FUNCIÓN RELLENO CON COLOR

Esta función rellena el área delimitada por otras áreas de colores diferentes del punto que se le pasa como coordenada. El color que se emplea para rellenar es el seleccionado previamente.

private void RellenoColor (int x, int y)

Esta función realiza un relleno de cuatro puntos ya que se basa en el punto superior, el inferior, el de la derecha y el de la izquierda para el relleno. Si quisiéramos hacer un relleno de ocho puntos deberíamos incluirle los puntos de las esquinas superior izquierda y derecha, y las esquinas inferior izquierda y derecha. Además, como el código es recursivo y realiza muchas llamadas, hemos incluido una clase de objeto Stack (que implementa una pila) para almacenar sólo los parámetros que nos hace falta y así evitar que se desborde realmente la pila de memoria.

```
void RellenoColor(int x, int y)
{
     int puntox,puntoy;
     System.Drawing.Color viejoColor;

     viejoColor = mapabits.GetPixel(x,y);
     if ((color.A != viejoColor.A)||(color.B !=
     viejoColor.B)||(color.G != viejoColor.G)||(color.R !=
     viejoColor.R))
     {
          Stack pila = new Stack();
          pila.Push(x);
          pila.Push(y);
          while (pila.Count != 0)
          {
               puntoy = (int)pila.Pop();
               puntox = (int)pila.Pop();
               if ((puntox>=0) && (puntoy>=0) &&
               (puntox<=mapabits.Width-1) &&
               (puntoy<=mapabits.Height -1))
               {
                    if (mapabits.GetPixel(puntox, puntoy) ==
                    viejoColor)
```

```
                    {
                        mapabits.SetPixel(puntox, puntoy,
                        color);
                        pila.Push(puntox-1);
                        pila.Push(puntoy);
                        pila.Push(puntox+1);
                        pila.Push(puntoy);
                        pila.Push(puntox);
                        pila.Push(puntoy+1);
                        pila.Push(puntox);
                        pila.Push(puntoy-1);
                    }
                }
            }
        pila.Clear();
        }
}
```

Esta función realiza un código recursivo, cuando se le da un punto, dibuja el color seleccionado sobre ese punto y mete en la pila los cuatro puntos que quiere dibujar. El programa va sacando coordenadas de la pila, dibujando el consiguiente punto e insertando los cuatro puntos siguientes, si puede. La función acaba cuando no tiene más elementos que sacar de la pila.

3.2.9. FUNCIÓN RELLENO CON IMAGEN

Esta función rellena con otras imágenes predefinidas el área delimitada por otras áreas de colores diferentes del punto que se le pasa como coordenada. Las bases que se emplean para el relleno son ladrillo, madera, nieve, rayas horizontales y rayas verticales.

private void RellenoImagen (int x, int y)

Las imágenes que sirven como relleno se crean al elegir la opción. El método que se emplea para el relleno es el del módulo 8 aunque, como las dimensiones son de 16x16 píxeles, empleamos módulo 16. El código es muy parecido al anterior, excepto en la forma de realizar el relleno, que se realiza por la técnica que hemos nombrado anteriormente.

```
void RellenoImagen(int x, int y, System.Drawing.Bitmap
imagenRelleno)
{
     int puntox,puntoy;
     int distanciax, distanciay;
     System.Drawing.Color colorImagen, viejoColor;

     viejoColor = mapabits.GetPixel(x,y);
     Stack pila = new Stack();
     pila.Push(x);
     pila.Push(y);
     while (pila.Count != 0)
     {
          puntoy = (int)pila.Pop();
          puntox = (int)pila.Pop();
          if
          ((puntox>=0)&&(puntoy>=0)&&(puntox<=mapabits.Width-
          1)&&(puntoy<=mapabits.Height -1))
          {
               if (mapabits.GetPixel(puntox, puntoy) ==
               viejoColor)
               {
                    distanciax = Abs(puntox - x);
                    distanciay = Abs(puntoy - y);
                    colorImagen =
                    imagenRelleno.GetPixel(distanciax%imagenR
```

```
                    elleno.Size.Width,distanciay%imagenRellen
                    o.Size.Height);
                    if (colorImagen != viejoColor)
                    {
                         mapabits.SetPixel(puntox,puntoy,colo
                         rImagen);
                         pila.Push(puntox-1);
                         pila.Push(puntoy);
                         pila.Push(puntox+1);
                         pila.Push(puntoy);

                         pila.Push(puntox);
                         pila.Push(puntoy+1);

                         pila.Push(puntox);
                         pila.Push(puntoy-1);
                    }
               }
          }
     }
     pila.Clear();
}
```

3.2.10. FUNCIÓN GOMA

La función Goma realiza una función muy similar a la función Punto, su única diferencia es que intenta simular a una goma y por eso su figura es rectangular, en lugar de circular como es la función Punto.

private void Goma (int x, int y)

CAPÍTULO 4

IMPLEMENTACIÓN DE LA APLICACIÓN GRÁFICA

En el capítulo cuarto voy a abordar los aspectos más profundos acerca de la creación de la aplicación Sketcher. En primer lugar, decir que este capítulo, a pesar de que incluya un poco de código en Visual C# para explicar las ideas, no es necesario saber programar en Visual C#; sólo basta con saber programar en cualquier lenguaje de programación visual bajo Windows, pues argumentaré mis códigos con ideas genéricas acerca de lo que hago y por que lo hago. En segundo lugar, no voy a hacer un estudio detallado línea por línea del código de mi programa pues gran parte de él es de fácil comprensión para aquellos que tengan nociones de programación en Visual C#, o simplemente, en programación visual, y les bastará con irse al código fuente. Es por ende, por lo que sólo abordaré las partes más interesantes y particulares de mi programa.

4.1. OBJETOS USADOS EN LA PROGRAMACIÓN DEL PROGRAMA SKETCHER

Aunque en principio sería posible desarrollar una aplicación de ventanas en C# con un simple editor de texto y utilizando las clases de los espacios de nombres adecuados (principalmente *System.WinForms*), esto no es lo que en la práctica se hace, ya que ello implicaría invertir mucho tiempo en la escritura del código encargado de generar la interfaz de nuestra aplicación, tiempo que podríamos estar aprovechando para centrarnos en resolver los problemas relativos a su lógica y no a su aspecto. Por esta razón, vamos a utilizar la herramienta Visual Studio.NET.

Visual Studio.NET, como entorno visual que es, permite diseñar la interfaz de la aplicación de manera visual, sin más que arrastrar con el ratón los elementos que necesitemos (botones, lista de selección, etc.) sobre las posiciones adecuadas en la ventana de nuestra aplicación (*Drag&Drop*). También incluye otras facilidades para el desarrollo, como una ventana de propiedades desde la que se puede modificar los valores de las propiedades de cada objeto sin tener que escribir código, un depurador de código gráfico, un editor de códigos inteligente que puede detectar nuestros errores de sintaxis instantáneamente, etc.

4.1.1. DISEÑO DE LA INTERFAZ DE LA APLICACIÓN SKETCHER

A continuación vamos a detallar cada uno de los controles que componen la interfaz del programa Sketcher, junto con la función que va a desempeñar en el mismo. Todos los controles gráficos en una aplicación de ventanas se seleccionan desde la etiqueta

Windows Forms, del Cuadro de Herramientas; y están en el espacio de nombre *System.Windows.Form.*

FormPrincipal

- imagen. Instancia de la clase *PictureBox*. Es el cuadro donde se muestra la imagen.
- barraHerramientas. Instancia de la clase *ToolBar*. Como su propio nombre indica es la Barra de Herramientas, aunque hay que ir completándola con sus elementos.
- barraEstado. Instancia de la clase *StatusBar*. Es una barra donde se muestra las coordenadas de la imagen.
- menuPrincipal. Instancia de la clase *MainMenu*. Es el Menú Principal que tiene la aplicación.
- barraH. Instancia de la clase *HScrollBar*. Es la barra de desplazamiento horizontal. Aparece cuando la dimensión horizontal que se muestra de la imagen es superior a la dimensión horizontal que muestra el cuadro de imagen.
- barraV. Instancia de la clase *VScrollBar*. Es la barra de desplazamiento vertical. Aparece cuando la dimensión vertical que se muestra de la imagen es superior a la dimensión vertical que muestra el cuadro de imagen.
- dlAbrirArchivo. Instancia de la clase *OpenFileDialog*. Carga un cuadro de diálogo para elegir el archivo que se quiere mostrar en el cuadro de imagen.
- dlGuardarArchivo. Instancia de la clase *SaveFileDialog*. Carga un cuadro de diálogo para salvar en un archivo el cuadro de imagen.
- dlImprimir. Instancia de la clase *PrintDialog*. Carga un cuadro de diálogo para imprimir el cuadro de imagen.

- pdImpresor. Instancia de la clase *PrintDocument*. Pasa un documento seleccionado a la impresora.

- imagenColor. Instancia de la clase *PictureBox*. Es una caja donde se carga el color que se haya seleccionado para que lo pueda ver el usuario.

- listaImagenes. Instancia de la clase *ImageList*. Es una colección de imágenes que después se utilizan como imágenes de los botones de la Barra de Herramientas.

- menuArchivo. Instancia de la clase *MenuItem.* Es una división del Menú Principal. Contiene todas aquellas opciones relacionadas con los ficheros.

- menuNuevo. Instancia de la clase *MenuItem.* Incluido en menuArchivo. Carga un formulario para seleccionar las dimensiones y el color de la imagen nueva que se quiere mostrar en el cuadro de imagen.

- menuAbrir. Instancia de la clase *MenuItem.* Incluido en menuArchivo. Carga un cuadro de diálogo para seleccionar el fichero que continene la imagen que se quiere mostrar en el cuadro de imagen.

- menuReabrir. Instancia de la clase *MenuItem.* Incluido en menuArchivo. Vuelve a cargar el último fichero que se cargó.

- menuGuardar. Instancia de la clase *MenuItem.* Incluido en menuArchivo. Guarda la imagen que actualmente contiene el cuadro de imagen en el fichero seleccionado anteriormente.

- menuGuardarComo. Instancia de la clase *MenuItem.* Incluido en menuArchivo. Carga un cuadro de diálogo para seleccionar un fichero donde almacenar la imagen que contiene el cuadro de imagen.

- menuImprimir. Instancia de la clase *MenuItem.* Incluido en menuArchivo. Carga un cuadro de diálogo para imprimir la imagen que contiene el cuadro de imagen.

- menuCerrar. Instancia de la clase *MenuItem.* Incluido en menuArchivo. Cierra la aplicación completamente sin guardar nada que no haya sido guardado anteriormente.

- menuEdicion. Instancia de la clase *MenuItem.* Es una división del Menú Principal. Contiene todas las opciones relacionadas con la edición de la imagen.

- menuVer. Instancia de la clase *MenuItem.* Es una división del Menú Principal. Contiene todas aquellas opciones relacionadas con las distintas formas de visionar la imagen.

- menuHerramientas. Instancia de la clase *MenuItem.* Es una división del Menú Principal. Contiene todas las opciones relacionadas con las diversas operaciones que se permiten realizar a la imagen.

- menuColores. Instancia de la clase *MenuItem.* Es una división del Menú Principal. Contiene los colores que se permiten utilizar en las operaciones que realizamos a la imagen.

- menuAcercaDe. Instancia de la clase *MenuItem.* Es una división del Menú Principal. Carga un formulario donde se muestra la realización y la dirección de la aplicación.

- menuZoom. Instancia de la clase *MenuItem.* Incluido en menuVer. Posee diversas opciones para elegir los distintos niveles de zoom.

- menuZoomx1. Instancia de la clase *MenuItem.* Incluido en menuZoom. Selecciona el modo de ver zoomx1. Se muestra las dimensiones de la imagen completando totalmente el cuadro de imagen.

- menuZoomx2. Instancia de la clase *MenuItem.* Incluido en menuZoom. Selecciona el modo de ver zoomx2. Se muestra las dimensiones de la imagen doblando las dimensiones del cuadro de imagen.

- menuZoomx3. Instancia de la clase *MenuItem.* Incluido en menuZoom. Selecciona el modo de ver zoomx3. Se muestra las dimensiones de la imagen triplicando las dimensiones del cuadro de imagen.

- menuZoomx4. Instancia de la clase *MenuItem.* Incluido en menuZoom. Selecciona el modo de ver zoomx4. Se muestra las dimensiones de la imagen cuadriplicando las dimensiones del cuadro de imagen.

- menuBlanco. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color blanco como el deseado para realizar las diversas operaciones sobre la imagen.

- menuAmarillo. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color amarillo como el deseado para realizar las diversas operaciones sobre la imagen.

- menuNaranja. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color naranja como el deseado para realizar las diversas operaciones sobre la imagen.

- menuVerde. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color verde como el deseado para realizar las diversas operaciones sobre la imagen.

- menuVerdeOscuro. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color verde oscuro como el deseado para realizar las diversas operaciones sobre la imagen.

- menuCeleste. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color celeste como el deseado para realizar las diversas operaciones sobre la imagen.

- menuAzul. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color azul como el deseado para realizar las diversas operaciones sobre la imagen.

- menuAzulOscuro. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color azul oscuro como el deseado para realizar las diversas operaciones sobre la imagen.

- menuRosa. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color rosa como el deseado para realizar las diversas operaciones sobre la imagen.

- menuRojo. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color rojo como el deseado para realizar las diversas operaciones sobre la imagen.

- menuFucsia. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color fucsia como el deseado para realizar las diversas operaciones sobre la imagen.

- menuGris. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color gris como el deseado para realizar las diversas operaciones sobre la imagen.

- menuGrisOscuro. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color gris oscuro como el deseado para realizar las diversas operaciones sobre la imagen.

- menuMarron. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color marrón como el deseado para realizar las diversas operaciones sobre la imagen.

- menuVioleta. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color violeta como el deseado para realizar las diversas operaciones sobre la imagen.

- menuNegro. Instancia de la clase *MenuItem.* Incluido en menuColores. Selecciona el color negro como el deseado para realizar las diversas operaciones sobre la imagen.

- menuLupa. Instancia de la clase *MenuItem.* Incluido en menuVer. Posee diversas opciones para elegir los distintos niveles de zoom.

- menuLupaAumento. Instancia de la clase *MenuItem.* Incluido en menuLupa. Aumenta en una unidad el nivel del zoom que se esté utilizando.

- menuLupaDecremento. Instancia de la clase *MenuItem.* Decrementa en una unidad el nivel del zoom que se esté utilizando.

- menuDeshacer. Instancia de la clase *MenuItem.* Incluido en menuEdición. Borra la última operación (hasta un máximo de cinco) que se haya realizado sobre el cuadro de imagen.

- menuGrosor. Instancia de la clase *MenuItem.* Incluido en menuEdición. Posee diversas opciones para elegir el tamaño de las operaciones que se realizan sobre la imagen.

- menuFino. Instancia de la clase *MenuItem.* Incluido en menuGrosor. Emplea un único píxel como unidad de dibujo sobre la imagen.

- menuMedio. Instancia de la clase *MenuItem.* Incluido en menuGrosor. Emplea cinco píxeles como unidad de dibujo sobre la imagen.

- menuGrande. Instancia de la clase *MenuItem.* Incluido en menuGrosor. Emplea diez píxeles como unidad de dibujo sobre la imagen.

- menuLapiz. Instancia de la clase *MenuItem.* Incluido en menuHerramientas. Permite utilizar un lápiz sobre el cuadro de imagen.

- menuLineas. Instancia de la clase *MenuItem.* Incluido en menuHerramientas. Permite dibujar líneas sobre el cuadro de imagen.

- menuPolilineas. Instancia de la clase *MenuItem.* Incluido en menuHerramientas. Permite dibujar polilíneas sobre el cuadro de imagen.

- menuRectangulo. Instancia de la clase *MenuItem.* Incluido en menuHerramientas. Permite seleccionar varias formas de dibujar un rectángulo en el cuadro de imagen.

- menuRectangulosSinRelleno. Instancia de la clase *MenuItem.* Incluido en menuRectangulo. Permite dibujar un rectángulo sin relleno sobre el cuadro de imagen.

- menuRectangulosConRelleno. Instancia de la clase *MenuItem.* Incluido en menuRectangulo. Permite dibujar un rectángulo relleno del mismo color sobre el cuadro de imagen.

- menuCirculo. Instancia de la clase *MenuItem.* Incluido en menuHerramientas. Permite seleccionar varias formas de dibujar un círculo en el cuadro de imagen.

- menuCirculosSinRelleno. Instancia de la clase *MenuItem.* Incluido en menuCirculo. Permite dibujar un círculo sin relleno sobre el cuadro de imagen.

- menuCirculosConRelleno. Instancia de la clase *MenuItem.* Incluido en menuCirculo. Permite dibujar un círculo relleno del mismo color sobre el cuadro de imagen.

- menuElipse. Instancia de la clase *MenuItem.* Incluido en menuHerramientas. Permite dibujar una elipse sin relleno en el cuadro de imagen.

- menuRelleno. Instancia de la clase *MenuItem.* Incluido en menuHerramientas. Contiene diversas opciones sobre dibujos para rellenar el cuadro de imagen.

- menuRellenoColor. Instancia de la clase *MenuItem.* Incluido en menuRelleno. Permite rellenar un plano cerrado con un color previamente seleccionado.

- menuRellenoLadrillo. Instancia de la clase *MenuItem.* Incluido en menuRelleno. Permite rellenar un plano cerrado con un dibujo de ladrillos.

- menuRellenoNieve. Instancia de la clase *MenuItem.* Incluido en menuRelleno. Permite rellenar un plano cerrado con un dibujo de nieve.

- menuRellenoMadera. Instancia de la clase *MenuItem.* Incluido en menuRelleno. Permite rellenar un plano cerrado con un dibujo de madera.

- menuRellenoRayasHorizontales. Instancia de la clase *MenuItem.* Incluido en menuRelleno. Permite rellenar un plano cerrado con un dibujo de rayas horizontales.

- menuRellenoRayasVerticales. Instancia de la clase *MenuItem.* Incluido en menuRelleno. Permite rellenar un plano cerrado con un dibujo de rayas verticales.

- menuGoma. Instancia de la clase *MenuItem.* Incluido en menuHerramientas. Permite utilizar una goma sobre el cuadro de imagen.

- menuArtistico. Instancia de la clase *MenuItem.* Incluido en menuHerramientas. Contiene opciones sobre tipos de dibujos artísticos.

- menuLineasArtisticas. Instancia de la clase *MenuItem.* Incluido en menuArtistico. Dibuja un conjunto de líneas circulares.

- menuRectangulosArtisticos. Instancia de la clase *MenuItem.* Incluido en menuArtístico. Dibuja un conjunto de rectángulos sin relleno.

- menuCirculosArtisticos. Instancia de la clase *MenuItem.* Incluido en menuArtisitco. Dibuja un conjunto de círculos sin relleno.

- menuRotar. Instancia de la clase *MenuItem.* Incluido en menuEdicion. Posee opciones para rotar la imagen del cuadro de imagen.

- menuRotar90. Instancia de la clase *MenuItem.* Incluido en menuRotar. Permite rotar la imagen del cuadro de imagen 90 grados.

- menuRotar180. Instancia de la clase *MenuItem.* Incluido en menuRotar. Permite rotar la imagen del cuadro de imagen 180 grados.

- menuRotar270. Instancia de la clase *MenuItem.* Incluido en menuRotar. Permite rotar la imagen del cuadro de imagen 270 grados.

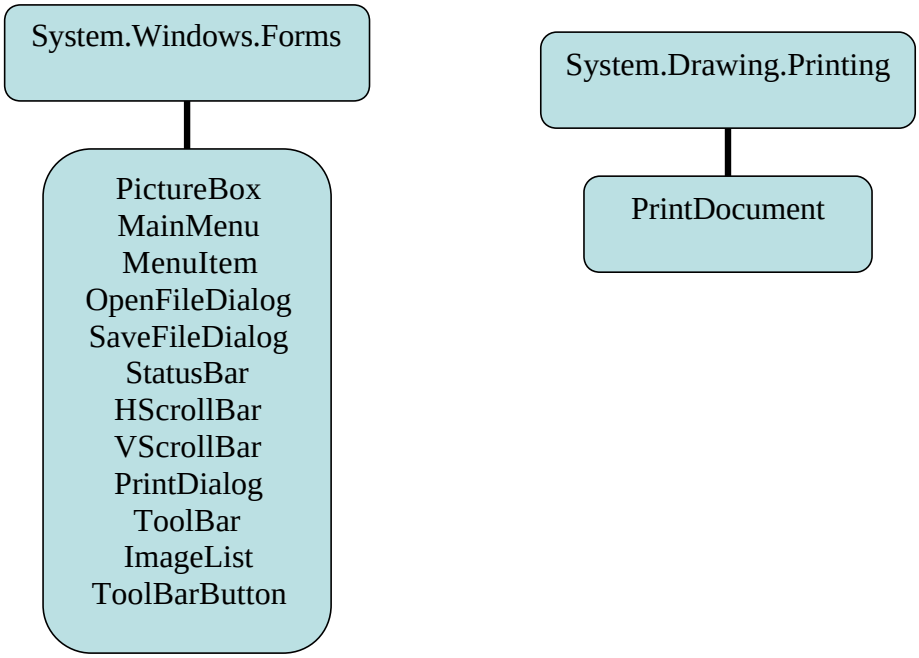
- menuInvertir. Instancia de la clase *MenuItem.* Incluido en menuEdicion. Posee opciones para invertir la imagen del cuadro de imagen.

- menuInvertirX. Instancia de la clase *MenuItem.* Incluido en menuInvertir. Permite invertir la imagen del cuadro de imagen respecto a las coordenadas de X.

- menuInvertirY. Instancia de la clase *MenuItem.* Incluido en menuInvertir. Permite invertir la imagen del cuadro de imagen respecto a las coordenadas de Y.

- menuTamañoReal. Instancia de la clase *MenuItem.* Incluido en menuVer. Muestra otro formulario donde se ve la imagen en el tamaño real en píxeles que tiene, sin tener en cuenta el cuadro de imagen.

- menuItem1, menuItem2, menuItem7, menuItem8, menuItem9. Pertenecen a la clase *MenuItem.* Sirven de separadores entre las diversas opciones del Menú Principal.



Toda la funcionalidad de Sketcher se puede encontrar en su Menú Principal. En su Barra de Herramientas, para mayor facilidad en su uso, se han incluido botones que realizan las operaciones más frecuentes. A continuación, enumeraré todos estos botones que pertenecen a la clase *ToolBarButton*. Por el nombre del botón puede saberse con que opción del Menú Principal está relacionado. La imagen que aparece en el botón debe haberse introducido antes en ListaImagenes, relacionar la barra de Herramientas con ListaImagenes a través de la propiedad *ImageList* y seleccionar la imagen desde la propiedad *ImageIndex* de cada botón.

botonNuevo, botonAbrir, botonGuardar, botonImprimir, botonLupaAumento, botonLupaDecremento, botonFino, botonMedio, botonGrande, botonColorBlanco, botonColorAmarillo, botonColorVerde, botonColorVerdeOscuro, botonColorCeleste, botonColorAzul, botonColorAzulOscuro, botonColorRosa, botonColorRojo, botonColorFucsia, botonColorVioleta, botonColorGris, BotonColorGrisOscuro, botonColorMarron, botonColorNegro, botonLapiz, botonLineas, botonPolilineas, BotonRectanguloSinRelleno, botonRectanguloConRelleno, botonCirculoSinRelleno, botonCirculoConRelleno, botonElipse, botonGoma, botonRelleno, botonSeparador1, botonSeparador2, ... , botonSeparador29.

FormNuevo

- etiquetaAncho. Instancia de la clase *Label*. Es una etiqueta informativa para indicar lo que tiene que completar en numericoAncho.
- etiquetaAlto. Instancia de la clase *Label*. Es una etiqueta informativa para indicar lo que tiene que completar en numericoAlto.
- etiquetaColor. Instancia de la clase *Label*. Es una etiqueta informativa para indicar lo que tiene que seleccionar en listaChequeada.

- numericoAncho. Instancia de la clase *NumericUpDown*. Es un cuadro de escritura-selección para indicar el número de píxeles de ancho que desea que tenga el nuevo dibujo que desea cargar.

- numericoAlto. Instancia de la clase *NumericUpDown*. Es un cuadro de escritura-selección para indicar el número de píxeles de alto que desea que tenga el nuevo dibujo que desea cargar.

- listaChequeada. Instancia de la clase *ListBox*. Es un cuadro para seleccionar el color de fondo que desea que aparezca en el nuevo dibujo. Por defecto, la selección es el color blanco.

- AceptarNuevo. Instancia de la clase *Button*. Es un botón para indicar que se ha seleccionado todo correctamente y que se quiere proceder a la carga en el cuadro de imagen de la nueva imagen con las características seleccionadas.

FormTamañoReal

- imagenReal. Instancia de la clase *PictureBox*. Es el cuadro de imagen que posee la imagen con el tamaño correspondiente al número de píxeles que realmente tiene la imagen.

FormAcercaDe

- etiquetaRealizado. Instancia de la clase *Label*. Es una etiqueta informativa sobre quien ha sido el encargado de realizar este PFC.

- etiquetaDirigido. Instancia de la clase *Label*. Es una etiqueta informativa sobre quien ha sido el director de este PFC.

- label1. Instancia de la clase *Label* Es una etiqueta informativa.

- boton1. Instancia de la clase *Button*. Es un botón para cerrar ese formulario.

4.1.2. OTRAS CLASES DE OBJETOS EMPLEADAS

Ya hemos comentado las clases de objetos y los objetos que hemos utilizado para hacer la interfaz de la aplicación Sketcher. Pero, también, hemos utilizado otras clases de objetos para la realización de la aplicación; aunque estas influyen únicamente en la parte lógica, y no en el aspecto del programa.

En un principio, debo comentar los tipos predefinidos por C#. Debo comentar que incluye un tipo booleano (*bool*) que modifica la forma de trabajar de la sentencia if respecto a C o C++, y que el tipo cadena de caracteres (*string*) tiene el operador == sobrecargado para poder compararlas.

Además de todo esto, posee una enorme cantidad de clases de objetos repartidos en también muchos espacios de nombres. Posee la clase de objeto Color *(System.Drawing.Color)* instanciada en los objetos color, viejoColor, etc. que se han utilizado para trabajar con los diversos colores que hacía falta en cada momento. También se ha empleado una clase de objeto muy útil para el relleno que es la clase Stack *(System.Collections.Stack)* instanciada en el objeto pila. Pero la clase de objetos más útil en esta aplicación es la clase de objeto Bitmap *(System.Drawing.Bitmap)* instanciada en mapabits, aux y en el array deshacer.

4.2. FUNCIONAMIENTO GENERAL DEL PROGRAMA

Para empezar en el programa Sketcher, nos encontramos con la variable seleccion de tipo string definida por mí. Esta cadena contiene el nombre del estado en el que nos queremos encontrar en cada momento, es decir, que cuando queremos realizar cierta operación, cargamos el nombre de ésta en esa cadena para que sepamos lo que queremos hacer en cada momento.

POSIBLES CADENAS DE CARACTERES QUE CONTIENE SELECCIÓN
Nada
LupaAumento
LupaDecremento
Lineas
Lineas Continuas
Lapiz
Rectangulos Sin Relleno
Rectangulos Con Relleno
Circulos Sin Relleno
Circulos Con Relleno
Elipses
Relleno Color
Relleno Ladrillo
Relleno Nieve
Relleno Madera
Relleno Rayas Horizontales
Relleno Rayas Verticales
Goma
Lineas Artisticas
Rectangulos Artisticos
Circulos Artisticos

Cada vez que pulsemos el botón de una herramienta almacenaremos su valor correspondiente en la variable seleccion, para que nuestro programa sepa que hacer cuando llegue el momento. Pero ¿qué ocurre cuando habiendo el usuario seleccionado una herramienta intenta usarla sobre la imagen? ¿Cómo actúa realmente el programa? Para ello ha sido necesario programar los eventos de imagen*(PictureBox)*, *MouseDown*, *MouseMove y MouseUp*. Cuando se produce un evento *MouseDown*, este trae como parámetros las coordenadas del punto donde se produjeron; con ellas y consultando el valor de la variable seleccion podemos empezar a tratarla. Según el tipo de operación que sea, empezaremos a

tratarla cuando se haya pulsado un botón del ratón, cuando esté en movimiento o cuando se haya soltado el botón del ratón. Veamos por ejemplo que seleccion es Lineas, es decir, la herramienta escogida es la de dibujar líneas. Al producirse este evento comprobamos mediante un anidamiento de sentencias de selección *IF* el contenido de la variable seleccion, entra por la rama correspondiente, y almacena los valores de las coordenadas reales (calculadas mediante la función CalcularCoordenadas) en unas variables globales llamadas yrealinicio y xrealinicio. También pone a Verdadero la variable global pulsado que se usa para saber después si se había pulsado el botón del ratón o no.

Sucesivamente a este evento, suponemos que se sucede el evento *MouseMove* que pasa como parámetros entre otras cosas las coordenadas por donde pasa el ratón. En este caso, debemos comprobar si el botón se había pulsado anteriormente (mediante la variable pulsado). En caso de que si se hubiera pulsado, debemos borrar la anterior línea que se dibujara con este evento y dibujar la nueva con las nuevas coordenadas que se la ha pasado para que parezca que la línea se está moviendo con el movimiento del ratón. Lo que estamos haciendo simplemente, cada vez que se produce este evento, es borrar la línea anterior y pintar una nueva. Para ello, tenemos dos variables de bitmaps (*System.Drawing.Bitmap*) que son aux y mapabits. La variable mapabits contiene la imagen que se muestra en cada momento y aux contiene la imagen, pero sin añadirle la última operación que está realizando en ese momento. Lo único que hacemos es copiar la imagen de aux en mapabits y dibujar la línea en cada movimiento.

Por último, en el evento *MouseUp* comprobamos si pulsado es Verdadero, y en caso que lo sea lo ponemos a Falso, borramos la línea del movimiento anterior y pintamos la definitiva. Metemos la imagen en aux (porque es la que queremos realizar) y actualizamos el array deshacer para tener esa opción en el futuro. En otra sección más adelante, explicaré el funcionamiento de dicho *array* para recuperar imágenes. Este mismo caso de la línea, se puede aplicar para la mayoría de herramientas tales como el rectángulo, la elipse, etc., pues únicamente almacenamos sus coordenadas de inicio y final en cada evento.

4.3. EL ZOOM Y LA LUPA

En este apartado vamos a tratar como realizar de una forma fácil y rápida tanto la ampliación como la disminución del tamaño en pantalla de la imagen con el fin de poder trabajar mejor con dicha imagen.

Todos los objetos de la clase *PictureBox* poseen una propiedad llamada *SizeMode*, en la cual se puede elegir el tipo *StretchImage*. Si se elige esa propiedad, hace que al cargar una imagen, ésta se ajuste a las dimensiones del objeto, y si no se elige, solo se muestra el trozo de la imagen que quepa en dicho objeto. Veamos un ejemplo de ambas situaciones:

Imagen sin la propiedad StretchImage

Imagen con la propiedad StretchImage

Nuestra intención es mostrar inicialmente la imagen ocupando toda la dimensión del cuadro de imagen, por tanto, lo que hacemos es mantener la propiedad *StretchImage* como la opción elegida permanentemente de la propiedad *SizeMode*. Cuando queremos mostrar ciertas partes concretas de la imagen, lo que hacemos es recortar esa parte y mostrarla en el cuadro de imagen. Es de este modo como conseguimos el efecto de zoom y lupa.

Mirando el código fuente podemos ver que el zoom y la lupa se realizan sobre la variable mapabits y que la variable imagen es un cuadro donde se ve lo que hemos seleccionado. Por tanto, tenemos unas variables que según el nivel del zoom o la lupa nos muestran más o menos área de la imagen en el cuadro de imagen. El zoom siempre muestra, al principio, su referencia desde la esquina superior izquierda; mientras que la lupa tiene su referencia de centro como el punto donde se ha pulsado el ratón. A todo ello, para el perfecto funcionamiento de estas herramientas, debemos comprobar como se desplazan las barras de desplazamiento. Estas aparecen cuando se activan el zoom o la lupa en niveles de x2 o superior. Esta tienen cual es el desplazamiento para cada nivel y simplemente va cambiando la referencia con el desplazamiento de la barra. Veamos dos ejemplos de zoom y lupa cuando pasan al nivel x2:

ZOOM

```
private void menuZoomx2_Click(object sender, System.EventArgs
e)
{
     zoom = 2;
     seleccion = "Nada";
     // Permito y elimino opciones
     menuLupaAumento.Enabled = true;
     menuLupaDecremento.Enabled = true;
     botonLupaAumento.Enabled = true;
     botonLupaDecremento.Enabled = true;
```

```
        // Inicializo las barras
        barraH.Visible = true;
        barraV.Visible = true;
        iniciozoomalto = 0;
        iniciozoomancho = 0;
        zoomalto = mapabits.Height/zoom;
        zoomancho = mapabits.Width/zoom;
        Rectangle Zoom = new
        Rectangle(iniciozoomancho,iniciozoomalto,zoomancho,zooma
        lto);
        imagen.Image =
        mapabits.Clone(Zoom,System.Drawing.Imaging.PixelFormat.D
        ontCare);
        // El desplazamiento largo va a ser en dos partes
        barraH.LargeChange = (barraH.Maximum -
        barraH.Minimum)/zoom;
        barraV.LargeChange = (barraV.Maximum -
        barraV.Minimum)/zoom;
        // Iniciamos las barras de desplazamiento
        barraH.Value = 0;
        barraV.Value = 0;
}
```

LUPA AUMENTO

```
private void LupaAumento(int x,int y)
{
     if (zoom ==1)
     {
          zoom = 2;
          // Permito y elimino opciones
          menuLupaAumento.Enabled = true;
          menuLupaDecremento.Enabled = true;
          botonLupaAumento.Enabled = true;
          botonLupaDecremento.Enabled = true;
          // Inicializo las barras
          barraH.Visible = true;
          barraV.Visible = true;
          zoomalto = mapabits.Height/zoom;
          zoomancho = mapabits.Width/zoom;
          iniciozoomalto = (y - (zoomalto/2));
          iniciozoomancho = (x - (zoomancho/2));
          // Controlo que los indices no se salgan del bitmap
          if (iniciozoomalto < 0) iniciozoomalto = 0;
          if (iniciozoomancho < 0) iniciozoomancho = 0;
          if (iniciozoomancho + zoomancho > mapabits.Width)
          iniciozoomancho = mapabits.Width - zoomancho;
          if (iniciozoomalto + zoomalto > mapabits.Height)
          iniciozoomalto = mapabits.Height - zoomalto;
          // Recorto el zoom
          Rectangle Zoom = new
          Rectangle(iniciozoomancho,iniciozoomalto,zoomancho,
          zoomalto);
```

```
            imagen.Image =
            mapabits.Clone(Zoom,System.Drawing.Imaging.PixelFor
            mat.DontCare);
            // El desplazamiento largo va a ser en dos partes
            barraH.LargeChange = (barraH.Maximum -
            barraH.Minimum)/zoom;
            barraV.LargeChange = (barraV.Maximum -
            barraV.Minimum)/zoom;
            barraH.Value = iniciozoomancho;
            barraV.Value = iniciozoomalto;
        }
        ...
}
```

BARRA DESPLAZAMIENTO

```
private void barraH_Scroll(object sender,
System.Windows.Forms.ScrollEventArgs e)
{
     iniciozoomancho = e.NewValue;
     if (e.NewValue + zoomancho > mapabits.Width) zoomancho =
     mapabits.Width - e.NewValue;
     Rectangle Zoom = new
     Rectangle(iniciozoomancho,iniciozoomalto,zoomancho,zooma
     lto);
     imagen.Image =
     mapabits.Clone(Zoom,System.Drawing.Imaging.PixelFormat.D
     ontCare);
}
```

Como pudimos observar en el ejemplo anterior de las dos imágenes, una con la propiedad *StretchImage* activa y la otra no; el sólo cambio de esta propiedad me ocasiona el reajuste del tamaño de la imagen en pantalla. Se debe notar que realmente el tamaño no ha cambiado, lo que cambia es el número de píxeles por pulgada, pero no el número de píxeles de ancho y alto de la imagen. Esto quiere decir que una vez que queramos realizar alguna modificación sobre la imagen ampliada o disminuida deberemos hacer un reajuste de las coordenadas del ratón que me suministra los distintos eventos a las nuevas. Por ello, tendremos que tener en cuenta que estas proporciones luego debemos usarlas para el traslado de coordenadas de un sistema a otro, mediante la famosa regla de tres (semejanza de triángulos) sería una cosa así: si la coordenada 33 de abcisa corresponde a mi imagen con una anchura de 850, la coordenada Z corresponderá para una anchura de 800, con los que nos quedaría, Z = (33 * 800) / 850 = 31,085. Como vemos en este caso deberemos trabajar con decimales, cosa que no es posible pues no podemos tratar coordenadas no enteras, lo que nos obliga al redondeo y la consecuente pérdida de exactitud.

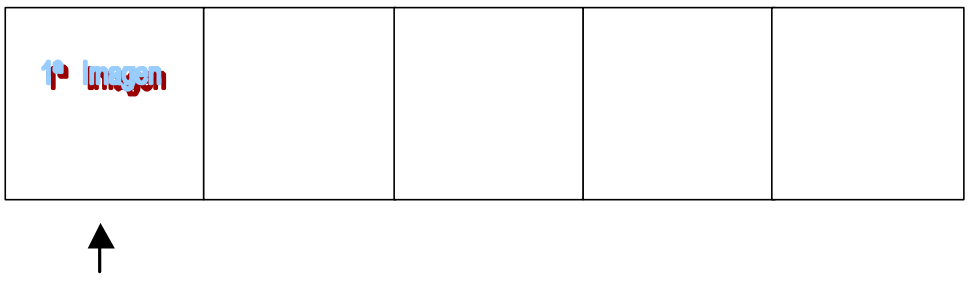
4.4. RECUPERACIÓN DE CAMBIOS PRODUCIDOS EN LA IMAGEN

En el apartado que ahora comenzamos, vamos a estudiar la forma de Sketcher de gestionar la recuperación de cambios producidos sobre la imagen. Esto puede ser producido por la circunstancia de que el usuario en un momento dado realice una operación sobre la imagen y una vez observados los resultados se arrepienta de haberla hecho y decida volver al punto anterior en que se encontraba la imagen.

Para este fin vamos a usar una cola de imágenes (*array* de la clase de objetos Bitmap, deshacer, con un índice, indicedeshacer) para la imagen abierta por nuestro programa. Y a dicha cola la gestionaremos como tal, es decir, la primera imagen en entrar será la primera imagen en salir. En total la cola tendrá cabida para cinco imágenes con lo que quiere decir que podremos recuperar hasta cinco errores consecutivos. Cada vez que recuperamos una imagen de la cola viene a ser como dar un paso hacia atrás.

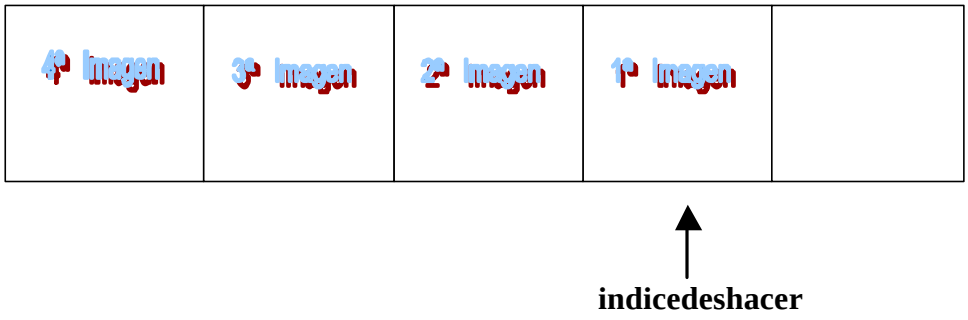
El contador indicedeshacer guarda el número de elementos que hay almacenado en el array para controlar los límites de este y saber en cada momento cuántas imágenes tenemos guardadas. Siempre incluimos la última imagen por la cabeza de la cola y actualizamos el array. Por tanto, cuando queremos realizar la operación deshacer, recuperamos la imagen de la posición 0 y desplazamos los elementos del array hacia la izquierda. Vamos a ver con un gráfico esta idea para aclararlo mejor:

En primer lugar, se introduce una imagen

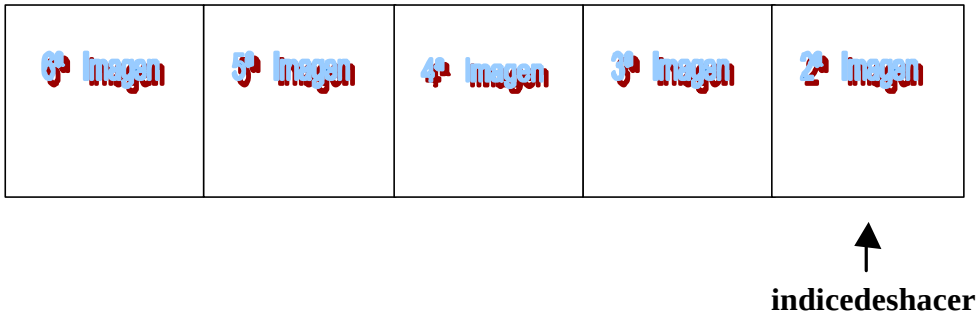


Indicedeshacer

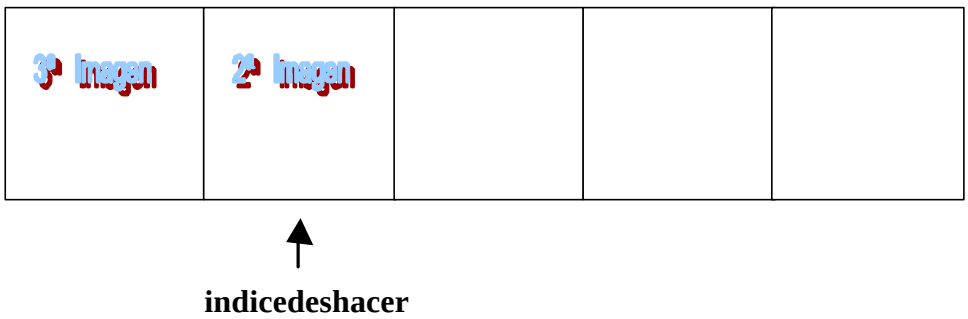
Si el usuario no recupera nada se irá introduciendo imágenes sucesivamente



Una vez que llega a la quinta imagen almacenada y necesita guardar otra, seguirá su orden y saldrán las que entraron en primer lugar.



Si necesita deshacer varios cambios consecutivos, por ejemplo, tres a partir de la posición anterior quedaría:



Como puede comprobarse en los ejemplos, su funcionamiento es sencillo. La implementación ya se ha comentado con anterioridad. Cada vez que se ha completado una operación, el estado anterior de la imagen se guarda en la cola deshacer. A continuación, introduzco el código para realizar la operación deshacer, es decir, para recuperar una imagen de la cola de mapa de bits deshacer.

```
private void menuDeshacer_Click(object sender,
System.EventArgs e)
{
     int i;    // Indice que se utiliza para recorrer el
array deshacer

     try
     {
          Rectangle Todo = new
          Rectangle(0,0,deshacer[0].Width,deshacer[0].Height)
          ;
          mapabits =
          deshacer[0].Clone(Todo,System.Drawing.Imaging.Pixel
          Format.DontCare);
          aux =
          deshacer[0].Clone(Todo,System.Drawing.Imaging.Pixel
          Format.DontCare);
          Rectangle Zoom = new
          Rectangle(iniciozoomancho,iniciozoomalto,zoomancho,
          zoomalto);
          imagen.Image =
          mapabits.Clone(Zoom,System.Drawing.Imaging.PixelFor
          mat.DontCare);
          indicedeshacer--;
          if (indicedeshacer == -1)
```

```
            {
                menuDeshacer.Text = "Deshacer";
                menuDeshacer.Enabled = false;
            }
            else
            {
                menuDeshacer.Text =
                "Deshacer("+(indicedeshacer+1).ToString()+")";
                for(i=0; i<=indicedeshacer; i++)
                {
                    Rectangle Todo2 = new
                    Rectangle(0,0,deshacer[i+1].Width,deshace
                    r[i+1].Height);
                    deshacer[i] =
                    deshacer[i+1].Clone(Todo2,System.Drawing.
                    Imaging.PixelFormat.DontCare);
                }
            }
        }
        catch
        {
            // Inicializamos las variables para deshacer
            indicedeshacer = -1;
            menuDeshacer.Text = "Deshacer";
            menuDeshacer.Enabled = false;
            MessageBox.Show("Se ha producido un error por la
            capacidad de la memoria","");
        }
}
```

CAPÍTULO 5

CONCLUSIONES

Llegados a esta sección tenemos que tener claro las posibilidades del programa Sketcher, así como la forma de implementación del mismo, pues en este capítulo vamos a estudiar posibles mejoras que se le pueden aplicar al mismo, así también daremos la idea de cómo realizaremos su implementación. Junto con todo esto y antes de pasar a estas cuestiones haré una exposición de los principales problemas que me he encontrado en la realización del programa.

5.1. PRINCIPALES PROBLEMAS ENCONTRADOS EN EL DESARROLLO DE LA APLICACIÓN

En un principio, el principal escollo que surgió era el estudiar un nuevo lenguaje de programación como era C#. Sin embargo, en Internet se puede encontrar muchísima información sobre este lenguaje. Además, ya existen varios libros sobre éste. Por el contrario, la información sobre Visual Studio.NET es escasa. Al ser una aplicación que se encuentra todavía en fase beta, es decir, no está completamente finalizada, no existen publicaciones que traten esta plataforma. En Internet se puede encontrar información sobre sus especificaciones y funciones, pero no existen manuales que informen al usuario sobre todos los recursos que posee la plataforma y, en especial, sobre los *Windows Forms*. Todo esto, unido a la poca experiencia previa que tenía con la programación visual, hizo que la fase de búsqueda de información fuese una tarea dura.

Una vez metidos en el diseño de la aplicación, el problema que causó más dificultades era el mencionado en capítulos anteriores, la elección del objeto que contendría la imagen y sobre el cuál realizáramos los dibujos. Era una elección complicada ya que una mala selección haría que más adelante nos encontrásemos con objetivos que eran imposibles de realizar. Teníamos dos opciones, el objeto Graphics, que implementaba ya muchas herramientas o el objeto Bitmap, que trataba la imagen como un simple mapa de bits. Al final, nos decidimos por la segunda opción ya que aunque aumentaba un poco la dificultad (la necesidad de implementar los algoritmos gráficos), nos daba la seguridad de poder realizar todo lo que nos propusimos en un principio.

Estos dos han sidos los dos principales problemas que hemos encontrado en la programación de la aplicación. Ha habido muchos más problemas pero de menor importancia, y provocados mayoritariamente por la falta de información, como han podido ser: la realización del zoom y su coordinación con las barras de desplazamientos, la realización de algunos algoritmos gráficos como el relleno con imágenes, etc.

5.2. CONCLUSIONES

Una vez finalizada la aplicación hemos de observar la gran cantidad de posibilidades que nos ofrece la programación en C#, y aún más, con la plataforma Visual Studio.NET. Hemos de resaltar la gran cantidad de clases de objetos que traen ya sus librerías, y sólo hablando de los Windows Forms y sus clase de objetos orientados a gráficos; ya que Visual Studio.NET es realmente interesante para otros tipos de aplicaciones, sobre todo, la creación de Servicios Webs y todo lo relacionado con Internet.

Aunque la elección del lenguaje no ha sido especialmente significativa ni ha mejorado en demasía su realización, si debemos comentar la facilidad de su aprendizaje ya que es muy parecido a otros ya existentes, y su beneficio con algunas de sus características como el recolector de basura, que hace olvidarte de recuperar el espacio de los objetos que ya no necesitas, o su uniformidad de tipo, que hace al lenguaje mucho más seguro e intuitivo.

En contra también debemos comentar la mala gestión de memoria, como casi todo lo relacionado con Microsoft, que realiza la aplicación ya que cuando trabajamos con imágenes de gran resolución, las operaciones se ralentizan demasiado y consume muchos recursos de memoria para una aplicación que no debería hacerlo tanto. Es decir, el programa es demasiado lento por la mala gestión de memoria que realiza. Por último, mencionar la cantidad de memoria que necesita Visual Studio.NET para ejecutarse ya que requiere para un funcionamiento más o menos correcto un mínimo de 256 MB de memoria RAM y, aunque hoy en día, esta cantidad es bastante asequible, no deja de ser exagerada para cualquier plataforma.

5.3. POSIBLES MEJORAS DEL PROGRAMA

Mucho se podría escribir acerca de este tema, sólo basta coger cualquier herramienta de retoque fotográfico y observar el sinfín de utilidades y posibilidades que te ofrece para el retoque. En un principio mi proyecto no tenía intención de orientarlo hacia ese campo pero es innegable que cualquier aplicación que se precia debe permitir al usuario por lo menos algunas posibilidades de filtrado y ajuste de colores.

Pero lo que se me ocurre plantear y que puede ser estudiado para un posible proyecto inminente a partir del mío, es sectorizar un poco el aspecto de la recuperación de los cambios sufridos en la imagen y explico por qué. La razón es que de la forma en que yo lo desarrollo, he de tener cinco imágenes más cargadas en memoria donde se van a grabar las distintas situaciones por donde ha ido pasando la imagen. Esto conlleva un gran derroche de recursos de memoria, y más con imágenes de gran resolución. Es por ello, por lo que sería más factible buscar otro método. La vectorización supondría que almacenaríamos en una pila las acciones que hemos efectuado sobre la imagen de forma que a partir de esta información podamos reconstruir la imagen anterior. Esta información podría venir dada por un registro variante donde uno de sus campos fuera el tipo de herramienta aplicada, y a partir de este campo pues condicionara el resto de los campos. Por ejemplo, para una herramienta como el Rectángulo Sin Relleno, deberíamos guardar: en primer lugar, en el campo herramienta debería haber una referencia al herramienta Rectángulo Sin Relleno, después como información para recuperar la situación anterior al campo deberíamos guardar el color del rectángulo, las coordenadas de origen y final del mismo, el grosor del trazado de la línea, el tipo de relleno y el Zoom que posee la imagen en ese momento.

Por supuesto que en cuanto a las posibilidades de retoque se refiere, se podría construir una biblioteca de procedimientos de filtros bastante completo que podría ocupar la mayor parte del proyecto. Y si esto no es suficiente podría incorporar como hacen herramientas, como el PhotoShop, la posibilidad de trabajar en una misma imagen con varias capas aumentando el número de posibilidades que podría adquirir el programa.

Para concluir con un último apunte, una posibilidad más sería la de incorporar al número de formatos de compresión de imágenes que reconoce el programa algunos muy popularizados tales como los GIF.

APÉNDICE A

MANUAL DE USUARIO

En este apéndice se va a aclarar que función tiene el programa desarrollado (Sketcher), así como nos adentraremos en todos los aspectos de su utilización. Se trata de una aplicación en C# para el dibujo gráfico en píxeles (*raster graphics*), que permitirá el dibujo de múltiples primitivas (líneas, círculos, polígonos,...), la gestión de sus atributos (grosor, color,...), el rellenado de regiones de píxeles y sus atributos, filtrado de trozos de imágenes o la imagen completa, etc., así como una serie de funciones de entrada/salida de fichero de imágenes.

Este capítulo se desarrolla la interacción del programa Sketcher con su posible usuario, destacando su facilidad de manejo, su alto grado de interacción, el gran número de posibilidades al nivel de herramientas de dibujo y sobre todo los resultados tan profesionales que podemos obtener en ella, comparables incluso con los que conseguiríamos con cualquier otra herramienta del mismo estilo como el PaintBrush o cualquier editor de imágenes. Este capítulo se dedicará a mostrar el funcionamiento del programa Sketcher, junto con todas las posibilidades que nos ofrece.

A.1. ESTRUCTURA GENERAL DEL PROGRAMA

Para empezar a describir su estructura, debemos observar que el programa tiene cuatro partes bien diferenciadas:

- **Menú Principal**: esta parte está compuesta por una serie de opciones que nos permitirán usar todas y cada una de las posibilidades de nuestra aplicación, comprende desde abrir una imagen desde un fichero hasta obtener información del programa que se está usando.
- **Barra de Herramientas**: el hecho de usar el menú cada vez que queramos realizar una acción dentro de nuestro programa, resulta un poco tedioso por el hecho de tener que pinchar con el ratón en la opción principal del menú correspondiente, esperar que se despliegue, y luego seleccionar dentro de dicho menú desplegado. Todo esto unido además a la necesidad por parte del usuario de conocer la ubicación dentro del menú de las distintas opciones, para poder acceder a ella con mayor rapidez, hace de uso necesario la barra de herramientas. Dicha Barra de Herramientas sólo es un panel que se sitúa en la parte superior del programa, y que contiene acceso directo a las funciones de uso más frecuentes en nuestra aplicación, y todo ello acompañado de un botón con un dibujo significativo de para qué se utiliza. El objeto de la misma es permitir al usuario que aprenda a manejarlo rápidamente, y que aprecie la facilidad de su uso junto con la comodidad de tener todas las opciones de uso más común al alcance de un simple clic del ratón.
- **Barra de Estado**: es un pequeño panel situado en la esquina inferior derecha que muestra indicaciones tanto de las herramientas que se han seleccionado en un momento determinado como de la situación del ratón sobre la imagen.
- Por último entre la Barra de Herramientas y la Barra de Estado nos encontramos con un gran espacio, al principio vacío cuya función es albergar las futuras imágenes que generemos o bien que extraigamos de algún fichero. Decir que nuestro programa sólo permite tratar una imagen al mismo tiempo, por lo que se superpondrán una sobre otra perdiéndose la información que no se haya guardado de la anterior imagen.

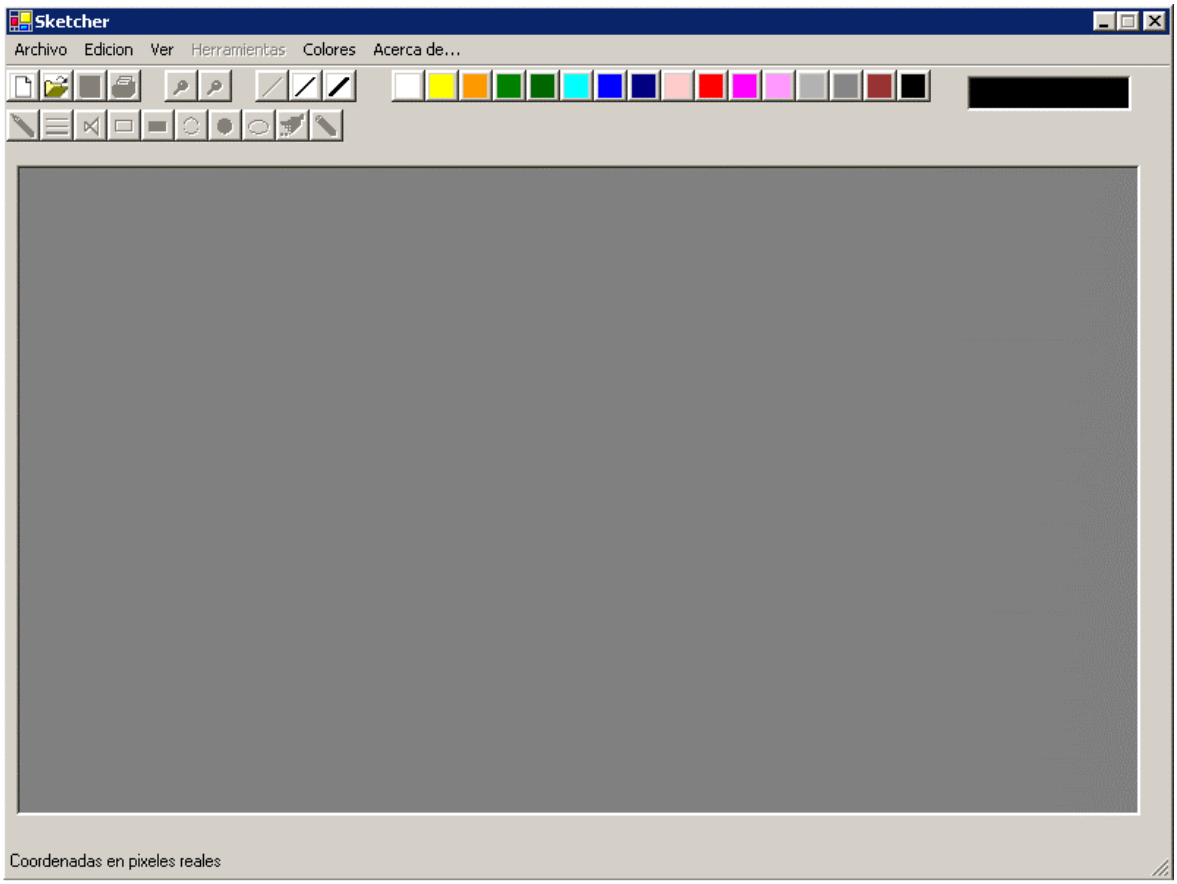


Figura 1. El programa Sketcher en ejecución

A.2. OPCIONES DEL MENÚ PRINCIPAL

Como hemos dicho anteriormente el Menú Principal es la puerta hacia todas y cada una de las opciones que nos proporciona nuestro programa. Aparece en la parte superior justo encima de la barra de herramientas. En la siguiente imagen podemos ver dicho menú:

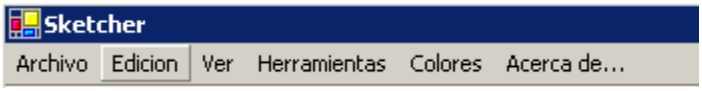


Figura 2. Menú Principal de Sketcher

Como se aprecia en la Figura 2 posee seis opciones principales que al picar sobre ellas se nos despliega a su vez otro menú, pero esta vez verticalmente. A continuación, vamos a pasar a describir todas y cada una de las distintas opciones del Menú Principal como son: Archivo, Edición, Ver, Herramientas, Colores y Acerca de...

A.2.1. OPCIONES DEL MENÚ PRINCIPAL: ARCHIVO

Como se aprecia en la siguiente figura (Figura 3), esta opción posee las características típicas necesarias para el acceso a imágenes almacenadas en ficheros, creación de nuevas imágenes, impresión, finalización del programa, etc.

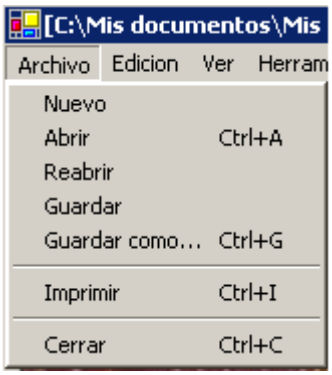


Figura 3. Opciones de Archivo

Pasamos a comentar una a una las opciones de Archivo:

- Nuevo: mediante esta opción se nos permite crear una imagen nueva con las dimensiones de anchura y altura en píxeles que deseemos. Para ello al pinchar en Nuevo, se nos aparece un panel como el que aparece a continuación donde debemos introducir los datos. Una vez introducidos pulsaremos el botón Aceptar para comunicarle a nuestro programa que puede pasar a crear la nueva imagen. El programa analiza los datos introducidos, verifica que son correctos, y la crea. En caso de que alguno de los campos anchura o altura estuviera sin rellenar no facilitándole las dimensiones, o bien introdujéramos dimensiones demasiado grandes, lanzaría un mensaje de error indicándonos el error. Si no se selecciona ningún color del cuadro de colores, se toma por defecto el blanco.

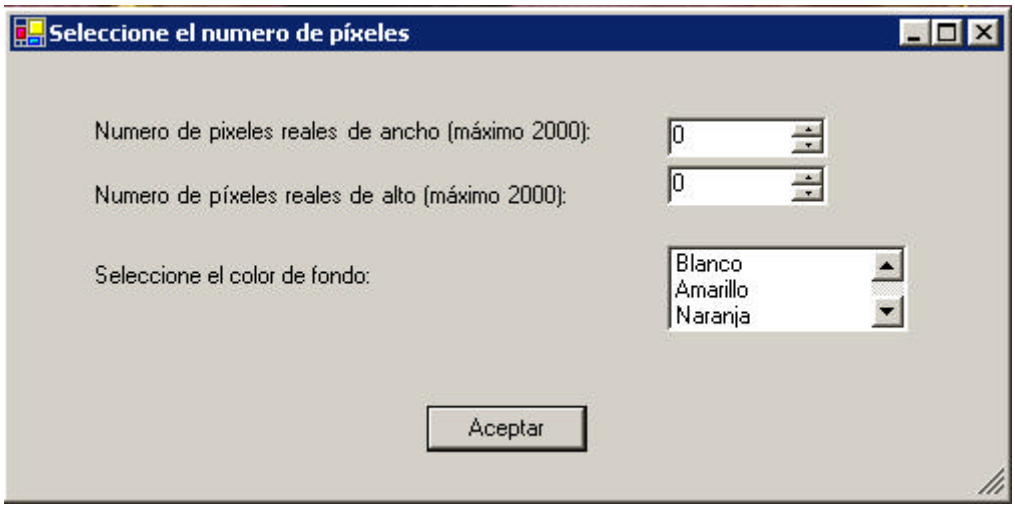


Figura 4. Panel para crear una nueva imagen

- Abrir: mediante esta opción se nos permite abrir una imagen anteriormente salvada u otra imagen cualquiera almacenada en un archivo. Los formatos que reconoce Sketcher son JPG (formato de compresión de imágenes JPEG) y BMP (formato de mapa de bits). Cuando se hace clic en esta acción nos aparece un cuadro de diálogo

que nos permite examinar todo los archivos del disco duro, para poder seleccionar aquel que queramos abrir.

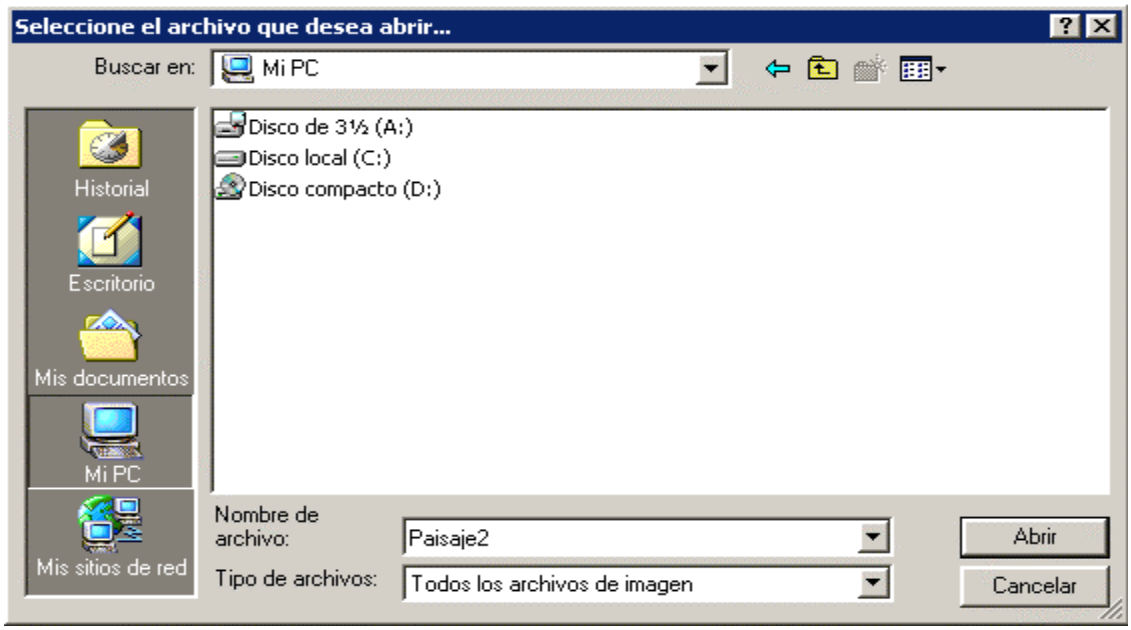


Figura 5. Cuadro de diálogo para abrir una imagen de un fichero

- Reabrir: esta opción abre de nuevo el fichero que actualmente esté utilizando. Sólo se puede elegir imágenes que estén salvadas en ficheros. Se utiliza si se quiere empezar de nuevo por los cambios que tiene guardado.

- Guardar: esta opción viene a ser más o menos la contraria de Abrir. Con ella podemos almacenar en un fichero una imagen que estemos en ese momento trabajando con ella. Además, debemos resaltar que esta acción actúa de dos formas distintas dependiendo si la imagen posee un nombre establecido por el usuario, es decir, es una imagen procedente de un fichero o bien es una imagen nueva pero que ha sido anteriormente grabado con un nombre. En este caso, la grabación es automática con el mismo nombre que el usuario ha establecido y sobre el fichero de la imagen, sobrescribiéndola. Por el contrario, si la imagen es nueva y no ha sido salvada anteriormente con ningún nombre, es en ese momento cuando aparecerá el

cuadro de diálogo anteriormente comentado donde le suministraremos el nombre, el sitio donde grabarla, además del formato.

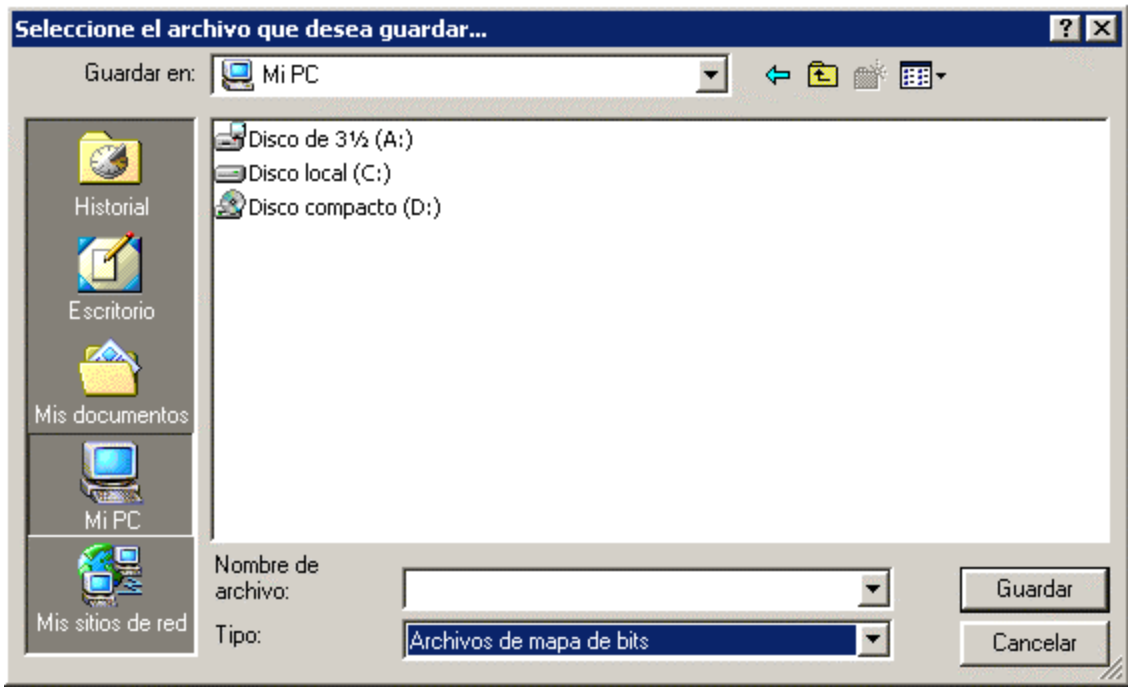


Figura 6. Cuadro de diálogo de Guardar

- Guardar como: esta opción es parecida a la anterior en cuanto trae como objeto el almacenamiento en un fichero de la imagen, pero difiere en que esta siempre aparecerá el cuadro de diálogo anterior para preguntar el nombre del fichero con el que deseamos almacenarlo.

- Imprimir: con esta opción podremos sacar una imagen por impresora. Para ello aparecerá en la pantalla un cuadro de diálogo que nos permitirá seleccionar entre otras cosas la impresora por la que queremos imprimir si hubieras varias seleccionada o el número de copias que deseamos realizar.

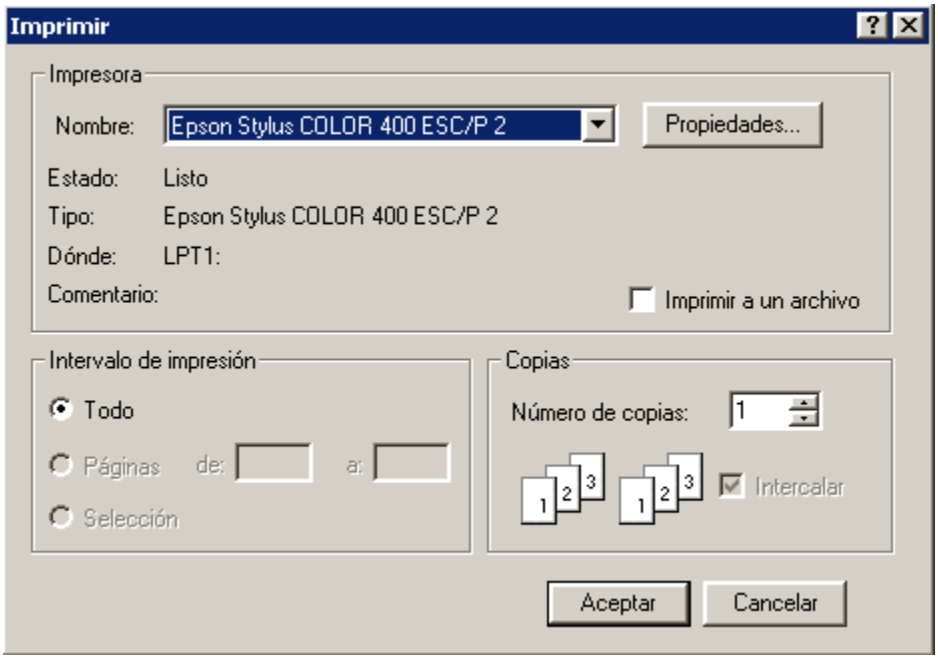


Figura 7. Cuadro de diálogo de Imprimir

- Cerrar: con esta acción da por concluida la sesión con el programa Sketcher, cerrando la imagen que estuviera abierta en ese momento, junto con el programa principal.

A.2.2. OPCIONES DEL MENÚ PRINCIPAL: EDICIÓN

Como se aprecia en la siguiente figura, esta opción contienen todos los aspectos relacionados con la edición de la imagen, tales como el grosor de la herramienta de dibujo, o la rotación o inversión respecto a X o Y de éstas. También, incluye la opción Deshacer que se utiliza para recuperar pasos anteriores de la imagen.

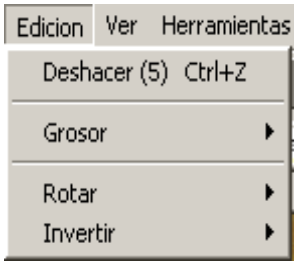


Figura 8. Opciones de Edición

A continuación pasaremos a comentar una a una las opciones de dicho menú:

- Deshacer: este programa permite corregir una acción una vez realizada volviendo a la posición anterior de realizarla. En realidad el número de cambios que permite corregir consecutivos es cinco. Pulsando sucesivamente esta acción iremos deshaciendo según creamos oportuno. Otra manera de poder acceder a este método sin necesidad de pinchar sobre Editar, es mediante la combinación de teclas CTRL+Z.

- Grosor: esta opción se utiliza para seleccionar la anchura de nuestra herramienta de dibujo. En realidad, la anchura es el número de píxeles que tenemos por referencia para dibujar. Hay tres opciones: Fino, que solo dibuja un píxel, Medio, que dibuja una círculo de 5 píxeles de diámetro y Grande, que dibuja una circunferencia de 10 píxeles de diámetro.

- Rotar: como indica su nombre, esta opción rota la imagen. Se puede elegir cuantos grados se desea que se rote la imagen: 90º, 180º o 270º. Algunas de estas rotaciones no son posibles ya que sus dimensiones son difícilmente ajustables a las del cuadro de imagen. En este caso, se muestra un mensaje de error y se realiza la única rotación posible. Veremos un ejemplo con las tres posibles rotaciones:

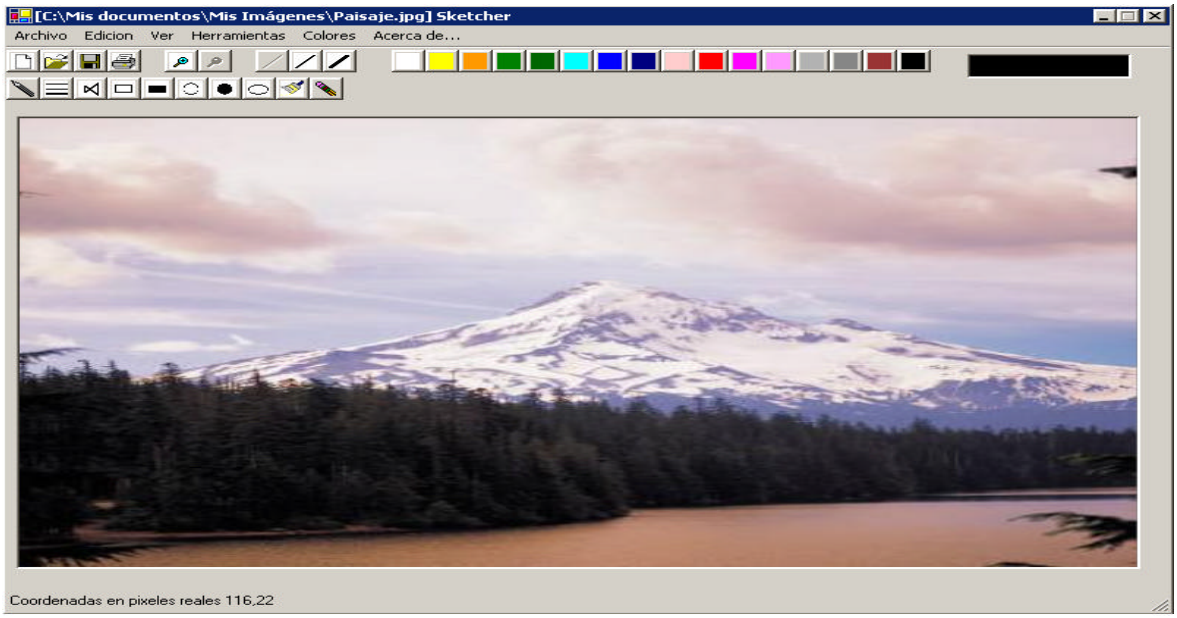


Figura 9. Imagen inicial sin utilizar la función Rotar

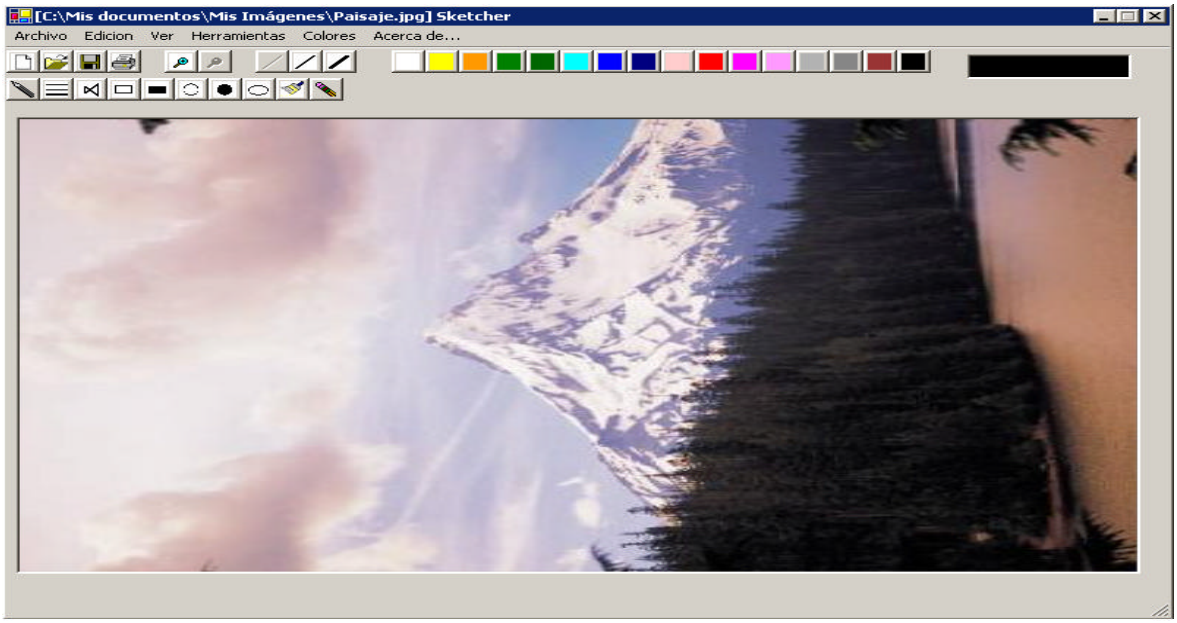


Figura 10. Imagen al utilizar la función Rotar 90º

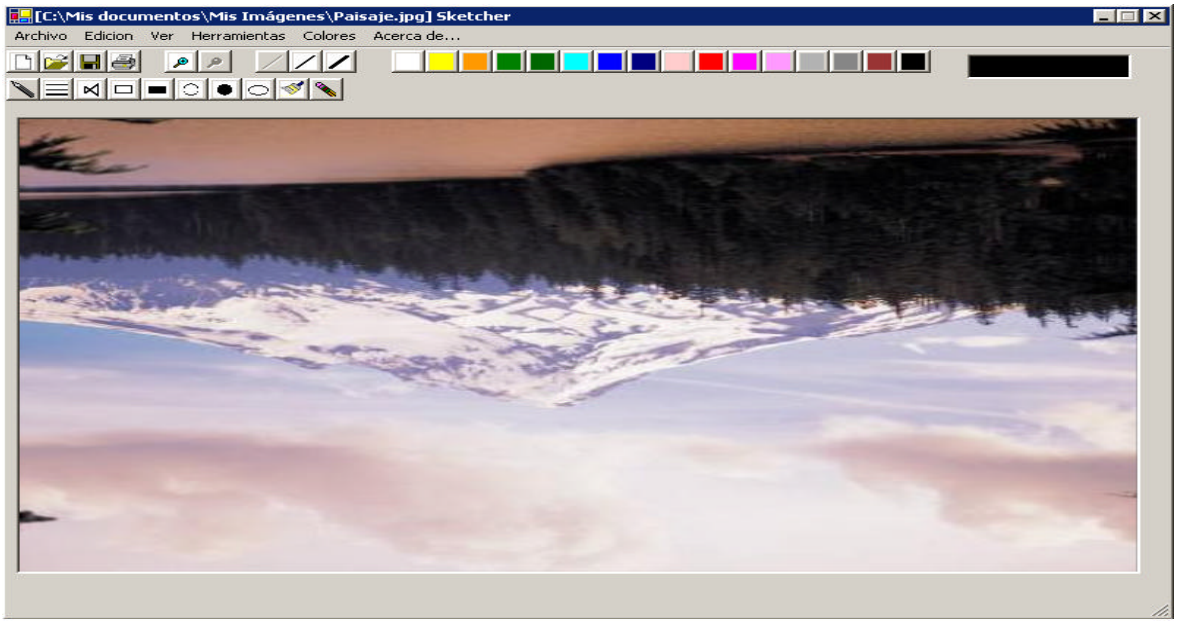


Figura 11. Imagen al utilizar la función Rotar 180º

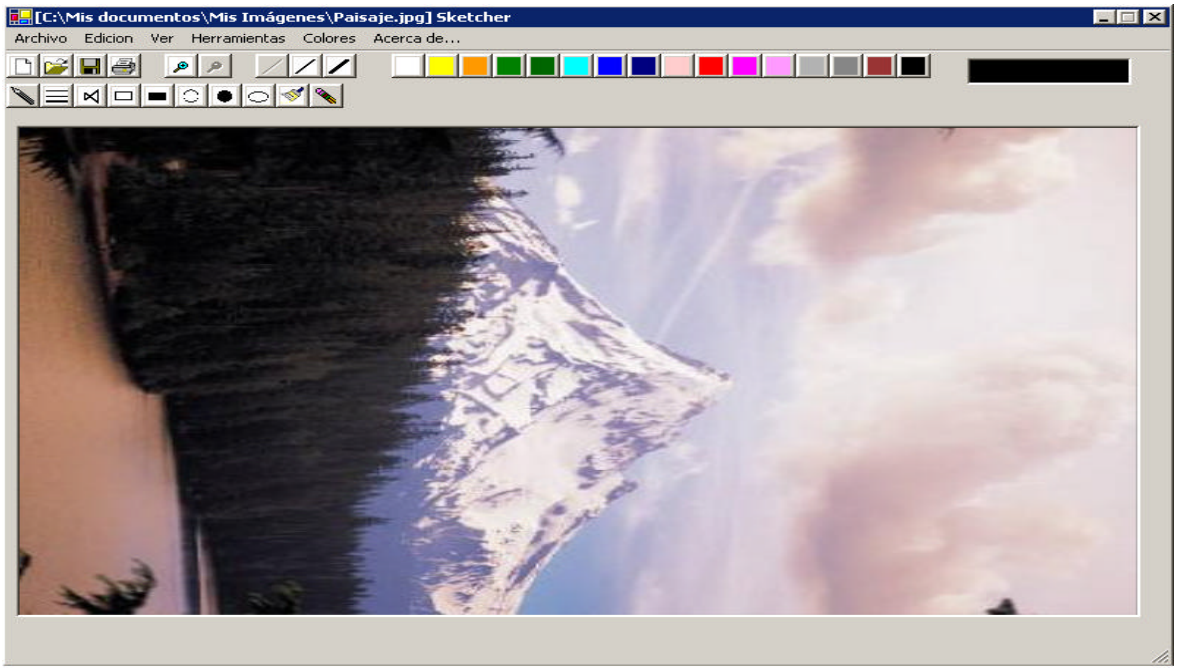


Figura 12. Imagen al utilizar la función Rotar 270º

- Invertir: esta operación invierte la imagen sobre sus respectivos ejes. Se puede elegir invertir respecto al eje X o respecto al eje Y. En el siguiente ejemplo veremos que es lo que realiza realmente la inversión respecto el eje Y.

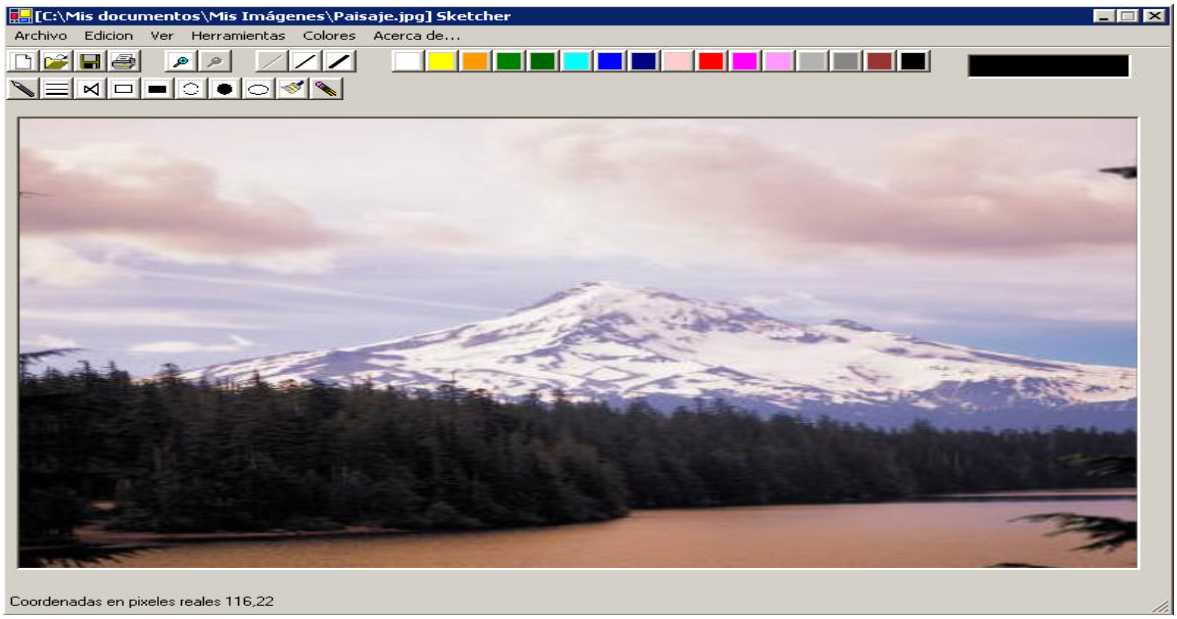


Figura 13. Imagen inicial sin utilizar la función Rotar

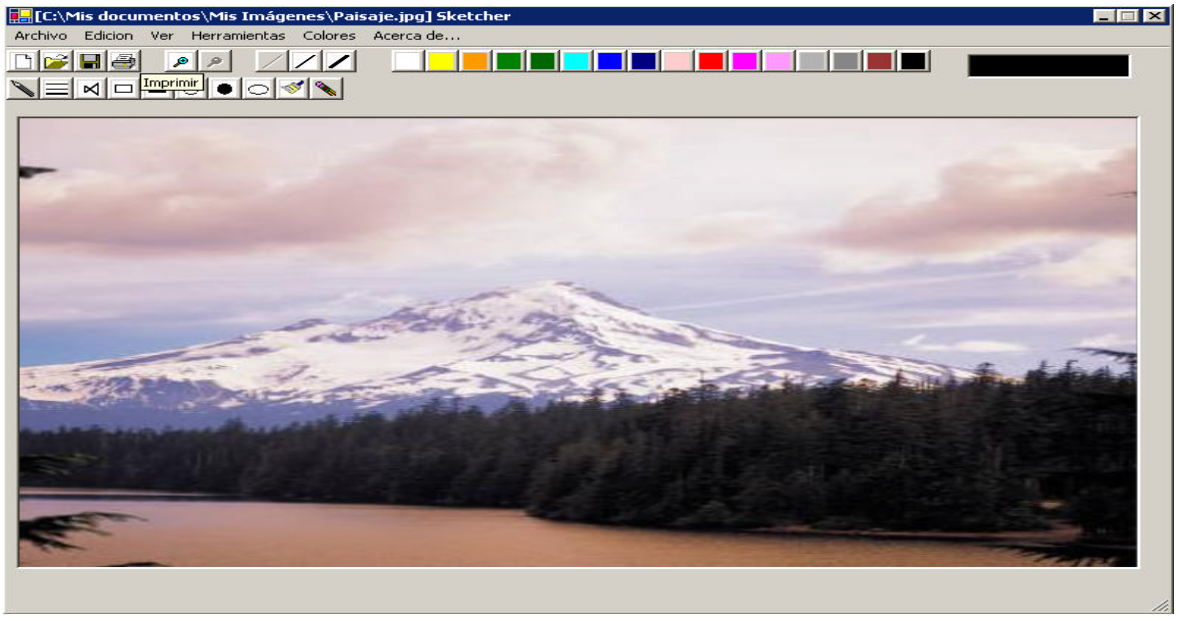


Figura 14. Imagen al utilizar la función Invertir respecto al eje Y

A.2.3. OPCIONES DEL MENÚ PRINCIPAL: VER

Esta opción del menú contiene las actividades relacionadas con las diferentes formas de visionar la imagen. Ahora, comentaremos las tres opciones del menú.

- Lupa: esta herramienta es la archiconocida de todas las aplicaciones al igual que Zoom. Nos permite ampliar la imagen en pantalla o disminuirla para que sea mejor apreciada y permitiéndonos el trabajar con ese tamaño como si fuera con el tamaño de la imagen original. Su funcionamiento es una vez seleccionada esta opción pinchar sobre la imagen para poner en funcionamiento el mecanismo. Según estemos aumentando o decrementando, el nivel del zoom de la imagen subirá o bajará un nivel, siendo el centro de la ampliación o disminución el lugar donde hemos realizado el clic del ratón.

- Zoom: al igual que la anterior, esta herramienta es la que utilizamos para controlar el nivel del zoom con el que queremos trabajar en cada momento. La única diferencia con el anterior, es que el Zoom no centra la imagen en un punto donde hayamos picado como la Lupa; sino que sube o baja un nivel el zoom y toma siempre como referencia inicial la esquina superior izquierda.

- Tamaño Real: como ya se explicó en el apartado 3 del capítulo 4, la aplicación Sketcher extiende la imagen de trabajo hasta rellenar toda su cuadro de imagen. Este acto lo realiza para utilizar las herramientas con mayor facilidad sobre la imagen, aunque se pierde mucha calidad cuando la imagen tiene poca resolución. Por tanto, esta es la herramienta que es capaz de mostrar la imagen en formato real, es decir, mostrando exactamente el número de píxeles que tiene y sus dimensiones.

A.2.4. OPCIONES DEL MENÚ PRINCIPAL: HERRAMIENTAS

Este menú va a contener las primitivas más importantes del programa para la creación de dibujos. La mayoría de estas primitivas se encuentran en la Barra de Herramientas accesibles con un solo clic de ratón en el botón correspondiente. Están realizadas con los algoritmos que explicamos ampliamente en el capítulo 3. Como se puede apreciar, posee una gran variedad de oportunidades semejantes a los de cualquier herramienta de diseño de imágenes. A continuación, y como en los apartados anteriores, pasamos a analizar una por una mencionadas primitivas.

Figura 15. Opciones de Herramientas

- Lápiz: esta es la típica herramienta de trazado de dibujo que se encuentra en cualquier editor de gráficos. Su funcionamiento consiste en pinchar en la zona de la imagen donde queramos comenzar a dibujar y a partir de ese momento si no soltamos el botón del ratón, en la imagen se irá dibujando el trozo por donde se mueva dicho ratón. Para concluir de trazar el dibujo o empezar en otro sitio simplemente soltando el botón del ratón, dejará de pintar. Además también contamos relacionada con esta herramienta con la posibilidad de aumentar o disminuir el grueso del trazado, es decir, el número de píxeles que coloreamos en la opción Grosos del menú Edición. Veamos un ejemplo de esta herramienta:

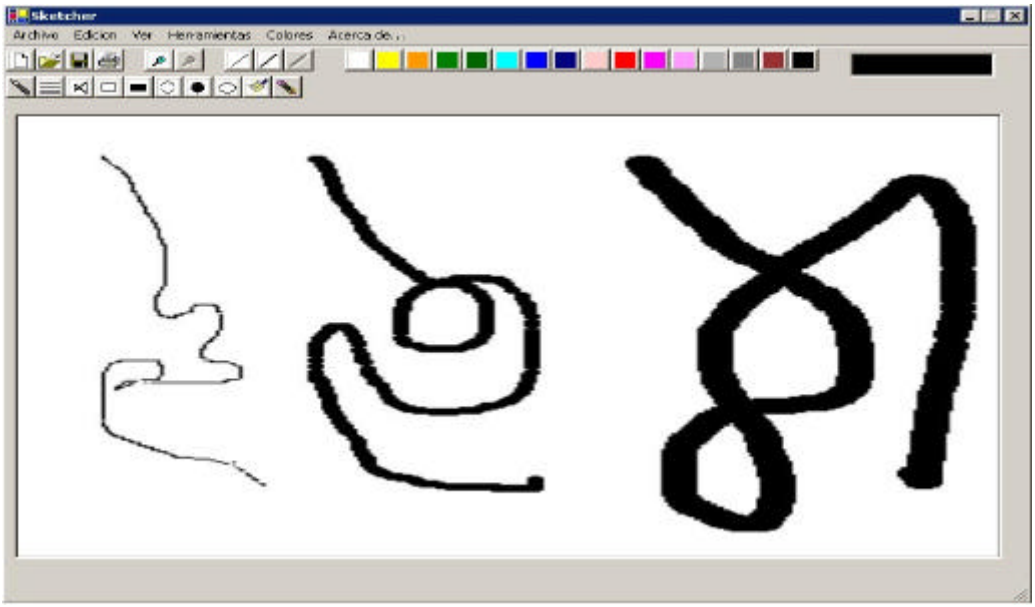


Figura 16. Ejemplo de la Herramienta Lápiz con distintos opciones de Grosor

- Líneas: mediante esta herramienta podemos dibujar líneas rectas. Simplemente pulsando en el comienzo de la línea sobre la imagen y sin soltar el ratón arrastrar hasta el final de la misma. Veamos un ejemplo de cómo quedaría en una imagen:

Figura 17. Ejemplo de la Herramienta Lineas con diversas opciones de Grosor

- Polilíneas: esta opción es semejante a la anterior con la diferencia que ésta encadena una serie de líneas consecutivas donde el comienzo de una es el final de otra, y este ciclo se acaba cuando se cambia de herramienta de dibujo. Su funcionamiento difiere un poco de la opción anterior. Ya que no debemos mantener el botón del ratón pulsado al desplazarnos. Para dibujar la primera línea debemos picar donde queremos que comience y donde queremos que acabe la línea recta; a continuación, ya sólo debemos picar donde queremos que acabe la siguiente recta ya que, como hemos comentado anteriormente, empieza al final de la línea que ha dibujado antes. Veamos un ejemplo de lo que hemos comentado:

Figura 18. Ejemplo de la herramienta Polilineas

- Rectángulos: este menú como se verá en la siguiente imagen tiene dos opciones: Rectángulos Sin Relleno y Rectángulos Con Relleno.

Figura 19. Opciones de Rectángulos

- Rectángulos Sin Relleno: esta herramienta como su nombre indica se utiliza para el dibujo de rectángulos no rellenos sobre la imagen. Su funcionamiento como el resto de las herramientas es: pulsar sobre el comienzo del rectángulo en la imagen y conforme arrastramos el ratón sobre la imagen sin dejar de pulsar el botón del ratón, se irá trazando un rectángulo automáticamente con el tamaño adecuado. Además, también va a influir en esta herramienta el tamaño de línea, aumentando o disminuyendo el grosor de trazado y el color seleccionado para los bordes de esa imagen, de forma que podemos variar dicho color. Veamos un ejemplo de esta herramienta:

Figura 20. Ejemplo de Rectángulos Sin Relleno

- Rectángulos Con Relleno: esta opción nos permite dibuja rectángulos pero con la particularidad de que su interior está relleno de un color seleccionado. Su funcionamiento como el resto de las herramientas es: pulsar sobre el comienzo del rectángulo en la imagen y conforme arrastramos el ratón sobre la imagen sin dejar de pulsar el botón del ratón, se irá trazando un rectángulo automáticamente con el tamaño adecuado. Además, también va a influir en esta herramienta el color seleccionado ya que será también con el que se realice el relleno. A continuación, vemos un ejemplo del uso de esta herramienta.

Figura 21. Ejemplo de Rectángulos Con Relleno

- Círculos: este menú como se verá en la siguiente imagen tiene dos opciones: Círculos Sin Relleno y Círculos Con Relleno.

Figura 22. Opciones de Círculos

- Círculos Sin Relleno: esta herramienta como su nombre indica se utiliza para el dibujo de círculos de medio punto no rellenos sobre la imagen. Su funcionamiento como el resto de las herramientas es: pulsar sobre el comienzo del rectángulo que contiene al círculo de medio punto en la imagen y conforme arrastramos el ratón sobre la imagen sin dejar de pulsar el botón del ratón, se irá trazando un círculo de medio punto automáticamente con el tamaño adecuado. Además, también va a influir en esta herramienta el tamaño de línea, aumentando o disminuyendo el grosor de trazado y el color seleccionado para los bordes de esa imagen, de forma que podemos variar dicho color. Veamos un ejemplo de esta herramienta:

Figura 23. Ejemplo de Círculos Sin Relleno

- Círculos Con Relleno: esta opción nos permite dibuja círculos de medio punto pero con la particularidad de que su interior está relleno un color seleccionado. Su funcionamiento como el resto de las herramientas es: pulsar sobre el comienzo del rectángulo que contiene al círculo de medio punto en la imagen y conforme arrastramos el ratón sobre la imagen sin dejar de pulsar el botón del ratón, se irá trazando un círculo de medio punto automáticamente con el tamaño adecuado. Además, también va a influir en esta herramienta el color seleccionado ya que será también con el que se realice el relleno. A continuación, vemos un ejemplo del uso de esta herramienta.

Figura 24. Ejemplo de Círculos Con Relleno

- Elipses: esta herramienta como su nombre indica se utiliza para el dibujo de elipses no rellenas sobre la imagen. Su funcionamiento como el resto de las herramientas es: pulsar sobre el comienzo del rectángulo que contiene la elipse en la imagen y conforme arrastramos el ratón sobre la imagen sin dejar de pulsar el botón del ratón, se irá trazando una elipse automáticamente con el tamaño adecuado. Además, también va a influir en esta herramienta el tamaño

de línea, aumentando o disminuyendo el grosor de trazado y el color seleccionado para los bordes de esa imagen, de forma que podemos variar dicho color. Veamos un ejemplo de esta herramienta:

Figura 25. Ejemplo de Elipse

- Relleno: esta opción es la que permite rellenar una superficie tanto con un color como con una imagen ya predefinida. Tiene varias opciones como veremos a continuación.

Figura 26. Opciones de Relleno

- Color: esta opción rellena una superficie de un determinado color por otro color previamente seleccionado. Para ello basta con seleccionar la opción adecuada y hacer clic en un área de la superficie que se quiere rellenar.

Figura 27. Ejemplo de Relleno Color

- Ladrillo: esta opción rellena una superficie de un determinado color por una imagen de ladrillos. Para ello basta con seleccionar la opción adecuada y hacer clic en un área de la superficie que se quiere rellenar.

Figura 28. Ejemplo de Relleno Ladrillo

- Madera: esta opción rellena una superficie de un determinado color por una imagen de madera. Para ello basta con seleccionar la opción adecuada y hacer clic en un área de la superficie que se quiere rellenar.

Figura 29. Ejemplo de Relleno Madera

- Nieve: esta opción rellena una superficie de un determinado color por una imagen de nieve. Para ello basta con seleccionar la opción adecuada y hacer clic en un área de la superficie que se quiere rellenar.

Figura 30. Ejemplo de Relleno Nieve

- Rayas Horizontales: esta opción rellena una superficie de un determinado color por una imagen de rayas horizontales. Para ello basta con seleccionar la opción adecuada y hacer clic en un área de la superficie que se quiere rellenar.

Figura 31. Ejemplo de Relleno Rayas Horizontales

- Rayas Verticales: esta opción rellena una superficie de un determinado color por una imagen de rayas verticales. Para ello basta con seleccionar la opción adecuada y hacer clic en un área de la superficie que se quiere rellenar.

Figura 32. Ejemplo de Relleno Rayas Verticales

- Goma: como su nombre indica esta opción es utilizada para el borrado de trozos de la imagen que deseemos prescindir de ellos. Su funcionamiento es una vez activada la opción pinchamos sobre un punto a borrar y sin soltar el botón del ratón lo desplazamos por la zona a eliminar. Por supuesto, podemos elegir tanto el tamaño de la goma como el color con el que desea borrar ya que entendemos que no siempre ha de borrar con blanco. Algún lector observador se ha podido dar cuenta que en realidad la goma no es más que el lápiz pero que su referencia básica en vez de ser un círculo es un cuadrado.

- Artístico: esta herramienta la he denominado así porque el resultado de su uso es para crear imágenes por así llamarlas, artísticas. Nos encontramos que dentro de esta opción tenemos a su vez tres posibilidades:

Figura 33. Opciones de Artístico

- Líneas Artísticas: creo que la mejor forma de explicar lo que hace esta herramienta, es viendo un ejemplo. Como se observará en él consiste en el dibujado de una serie de líneas partiendo todas de un origen común:

Figura 34. Ejemplo de Líneas Artísticas

- Rectángulos Artísticos: esta opción es igual que la anterior con la diferencia que en lugar de pintar líneas, dibuja rectángulos. Veamos otro ejemplo:

Figura 35. Ejemplo de Rectángulos Artísticos

- Círculos Artísticos: esta opción es exactamente igual a las dos anteriores, pero como indica el nombre con círculos.

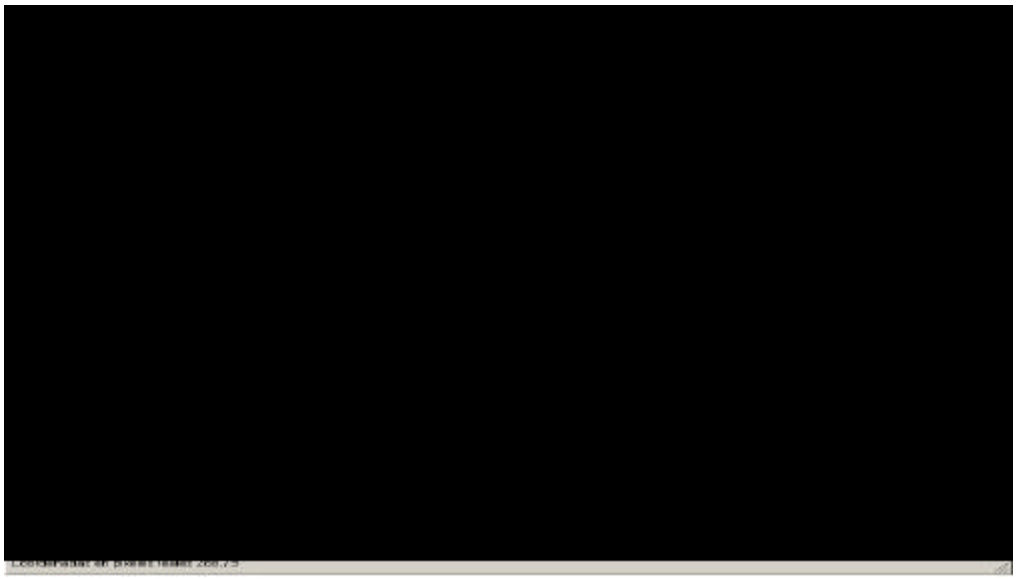


Figura 36. Ejemplo de Círculos Artísticos

A.2.5. OPCIONES DEL MENÚ PRINCIPAL: COLORES

Este menú va a contener todos los colores con los que podemos pintar en nuestra aplicación. Es muy importante ya que todas las opciones que hemos tenido para dibujar en el Menú Herramientas se pueden aplicar con colores diferentes, hasta Goma. Este menú está completamente en la Barra de Herramientas para que sea más intuitivo (porque se ve realmente el color) y rápido la elección de un color. Además, en la Barra de Herramientas se ve en cuadro donde se visualiza el color seleccionado en cada momento. Por defecto, está seleccionado el color Negro.

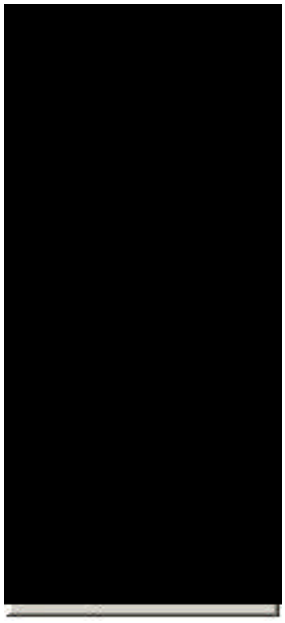


Figura 37. Opciones de Color

Figura 38. Zona donde están las opciones de color en la Barra de Herramientas

A.2.6. OPCIONES DEL MENÚ PRINCIPAL: ACERCA DE...

Este menú no posee ninguna opción y al pulsar sobre él, nos muestra un cuadro de diálogo como el que vemos a continuación:

Figura 39. Cuadro de diálogo de Acerca de...

A.3. OPCIONES DE LA BARRA DE HERRAMIENTAS

En todos los programas realizados para un entorno visual de ventanas suelen contar con un panel donde se encuentra una serie de botones y que se le denomina barra de herramientas. Esta barra se suele colocar en diferentes lugares de la ventana siendo el más habitual como en el caso de mi programa, en la parte superior de la ventana. En dicha barra encontramos aquellas herramientas de uso más habitual en nuestra aplicación y por ese motivo, deben situarse en un lugar más accesible que dentro de las opciones del menú principal. Sobre este tema es el que nos referíamos al principio del capítulo referente a conseguir una mayor interactividad con el usuario. Si además de esto, le añadimos que cada botón está acompañado de su correspondiente dibujo significativo de su función, junto con un pequeño cuadro que aparece al situarse encima del botón y que te muestra el nombre de la función describiéndote para que sirve dicha herramienta, todo esto hacen mucho más fácil el uso de la misma, así como para aquellos usuarios novatos, su aprendizaje. Veamos a continuación una figura con la Barra de Herramientas y posteriormente pasaremos a describir una por una todas las posibilidades de la misma:

Figura 40. Barra de Herramientas del programa Sketcher

Una aclaración antes de pasar a mencionar los distintos componentes de la Barra de Herramientas: la explicación de la función de cada botón es la misma que la correspondiente en el Menú Principal por lo que omitiré su explicación pero indicaré la correspondencia con el Menú Principal para aquellos que no lo hayan leído o no logren recordar todas sus particularidades.

- Botón Nuevo: menú Archivo.

- Botón Abrir: menú Archivo.

- Botón Guardar Como: menú Archivo.

- Botón Imprimir: menú Archivo:

- Botón Lupa Aumento: menú Ver, opción Lupa.

- Botón Lupa Decremento: menú Ver, opción Lupa.

- Botón Fino: menú Edición, opción Grosor.

- Botón Medio: menú Edición, opción Grosor.

- Botón Grande: menú Edición, opción Grosor.

- Botón de Colores: menú Colores.

- Botón Lápiz: menú Herramientas.

- Botón Líneas: menú Herramientas.

- Botón Polilíneas: menú Herramientas.

- Botón Rectángulo Sin Relleno: menú Herramientas, opción Rectángulos.

- Botón Rectángulo Con Relleno: menú Herramientas, opción Rectángulos.

- Botón Círculo Sin Relleno: menú Herramientas, opción Círculos.

- Botón Círculo Con Relleno: menú Herramientas, opción Círculos.

- Botón Elipses: menú Herramientas.

- Botón Relleno Color: menú Herramientas, opción Relleno.

- Botón Goma: menú Herramientas.

A.4. BARRA DE ESTADO Y ZONA DE IMÁGENES.

La Barra de Estado es una pequeña barra que aparece en la esquina inferior izquierda y que muestra tanto información sobre la herramienta seleccionada, como la posición por donde pasa el ratón respecto a las coordenadas reales de la imagen. La zona de imágenes es el cuadro central donde aparecen las imágenes que queremos editar. En este espacio se irán creando las imágenes que vayamos creando o abriendo.

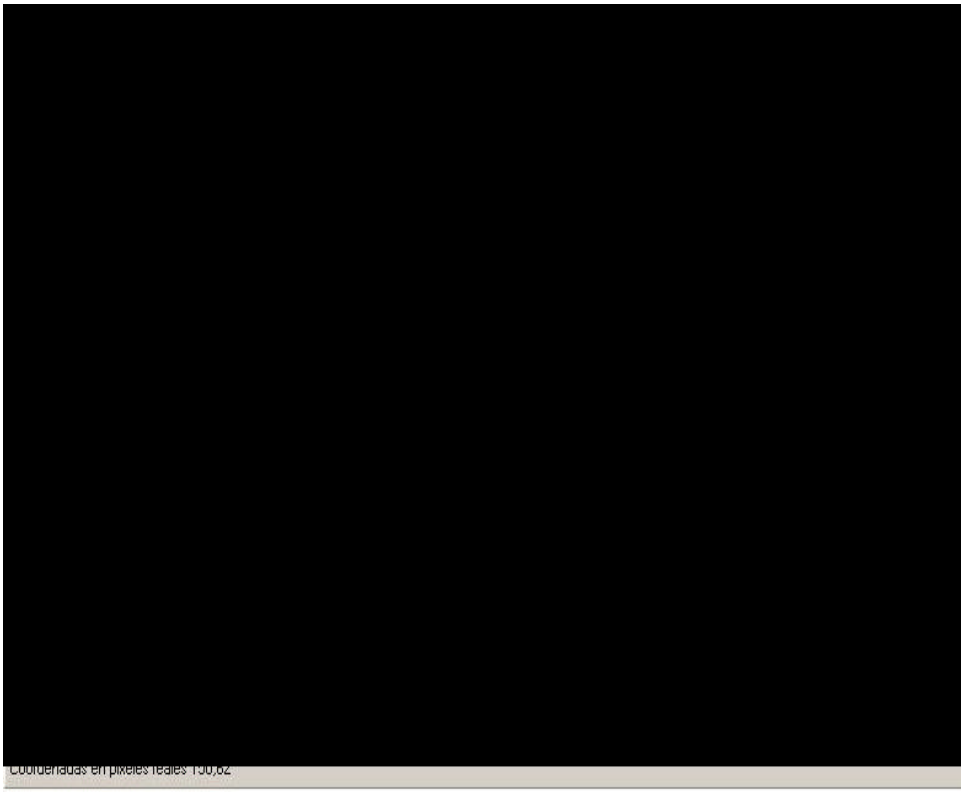


Figura 41. Barra de Estado y Zona de Imágenes

APÉNDICE B

CONTENIDO DEL CD-ROM

En este capítulo vamos a explicar el contenido de los ficheros que engloban el proyecto. Mostraremos inicialmente como está organizado el CD-ROM que se adjunta y explicaremos cuáles son los archivos principales y lo que realizan cada uno de ellos. Este apéndice será muy breve y su único objetivo es que se sepa utilizar el CD-ROM tanto para ver los códigos fuente del proyecto como para realizar la instalación de la aplicación Sketcher.

B.1. CONTENIDO Y FUNCIONES DEL CD-ROM

Cuando abrimos el contenido del CD-ROM desde Windows, vemos que está formado por tres carpetas: Ejecutable, Sketcher y Memorias.

Como indica su nombre, la carpeta Ejecutable contiene el fichero ejecutable que realiza la instalación de la aplicación Sketcher en el sistema. Este fichero se llama setup.exe y al hacer doble clic sobre éste, comienza la instalación del programa. En primer lugar, aparece una ventana sobre un fondo azul que indica en el subdirectorio que se va a instalar por defecto. Si se desea cambiar dicho subdirectorio, debe pulsar sobre el botón Examinar y seleccionar el subdirectorio deseado. Cuando se haya seleccionado el directorio donde se quiere instalar el programa, debe pulsarse el botón Comenzar para iniciar la instalación.

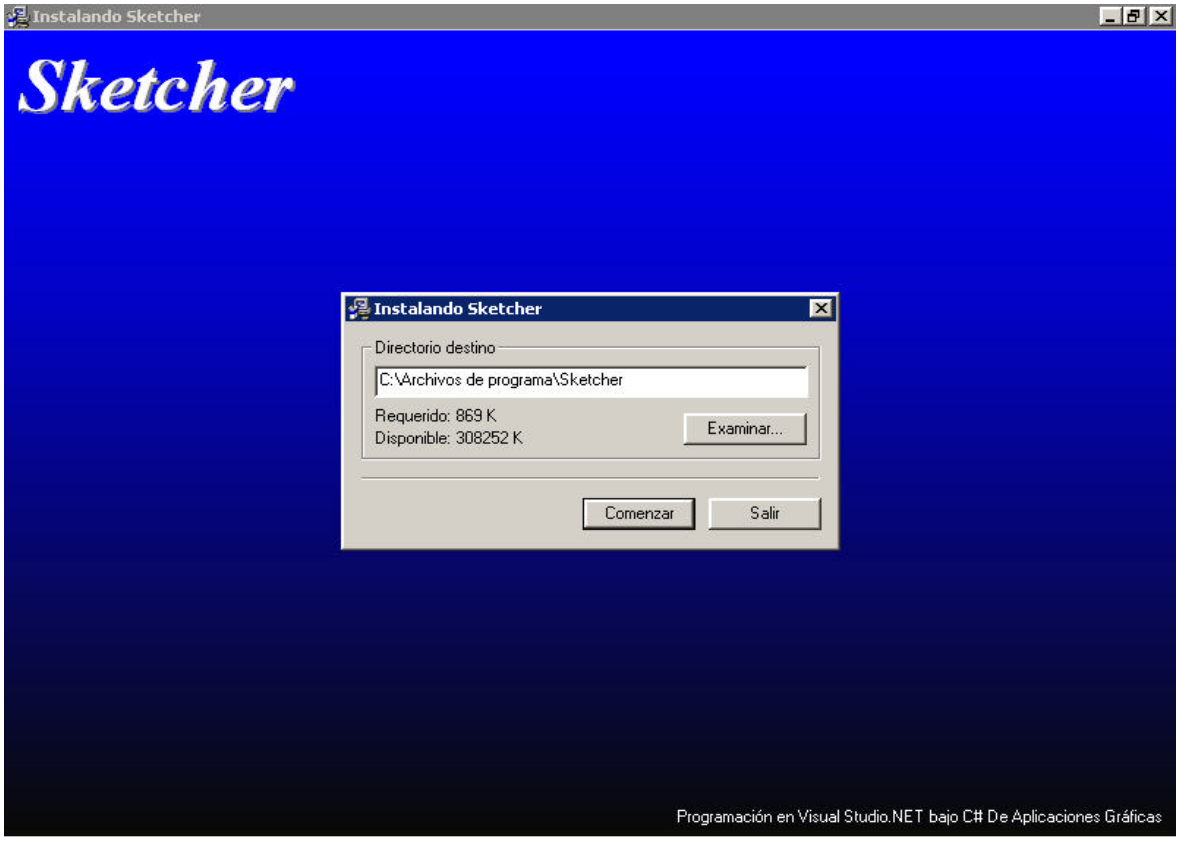


Figura 42. Menú Inicial en la instalación de la aplicación Sketcher

Cuando abrimos la carpeta Sketcher nos encontramos con los ficheros que contienen el código fuente de la aplicación.

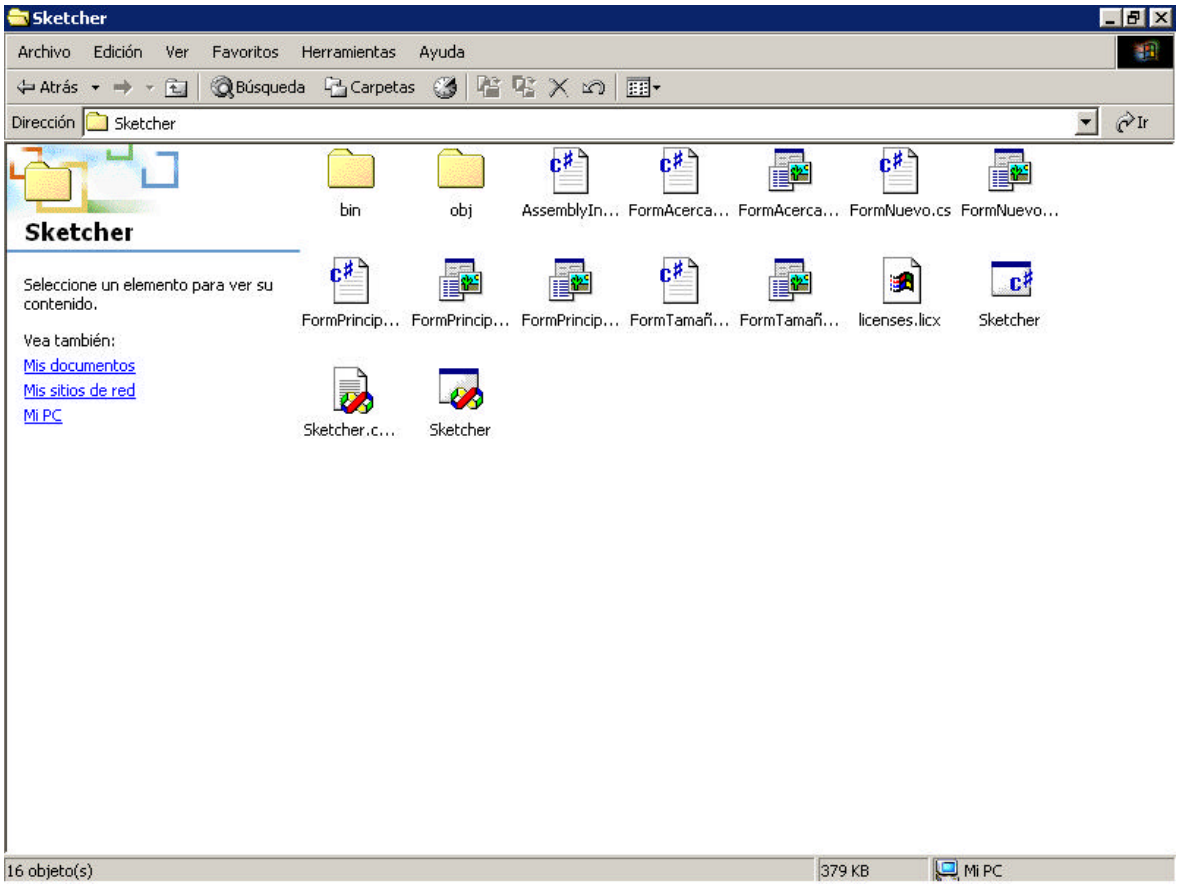


Figura 43. Contenido de la carpeta Sketcher del CD-ROM

Está compuesto inicialmente por dos carpetas que es donde se guarda los ejecutables que genera el compilador (bin y obj) y por catorce archivos de diverso contenido.

FICHEROS DE LA CARPETA SKETCHER (CÓDIGO FUENTE)
AssemblyInfo.cs
FormAcercaDe.cs
FormAcercaDe.resx
FormNuevo.cs
FormNuevo.resx
FormPrincipal.cs
FormPrincipal.resx
FormPrincipal.es.resx
FormTamañoReal.cs
FormTamañoReal.resx
licenses.licx
Sketcher.csproj
Sketcher.csproj (user)
Sketcher.sln

FormAcercaDe.cs, *FormNuevo.cs*, *FormTamañoReal.cs* y *FormPrincipal.cs* contiene los códigos fuentes en C# de cada uno de los formularios que componen la aplicación. Por el nombre, se puede intuir que es lo que realiza cada formulario. Los otros ficheros son los encargados de guardar información del proyecto Sketcher, opciones del ensamblado y de los diversos formularios brevemente comentados.

Finalmente, la carpeta Memorias contiene los ficheros que contienen la memoria tanto en formato .doc como en formato .pdf.

BIBLIOGRAFÍA

[1] Antonio Carlos Melgar Nieto, *"Programación Visual en Delphi de Aplicaciones Gráficas en 2D,"* proyecto fin de carrera de Ingeniería Informática dirigido por Francisco R. Villatoro, E.T.S. Ingenieros en Informática, Universidad de Málaga, 1998.

[2] "*Microsoft Visual Studio.NET beta 2 Guided Tour,"* 2001. Disponible vía web: http://msdn.microsoft.com/vstudio/nextgen/technology/beta2guidetour.exe

[3] *"Microsoft Visual Studio.NET Evaluation Guide,"* 2001. Disponible vía web: http://msdn.microsoft.com/vstudio/nextgen/technology/beta2evalguide.exe

[4] Christoph Wille, *"C#,"* Prentice Hall, 2001.

[5] José Antonio González Seco, *"El lenguaje de programación C#,"* 2001. Disponible vía web: http://www.galeon.com/josanguapo/librocsharp.zip

[6] Foley, J.D., A. van Dam, S.K. Feiner and J.F. "*Computer Graphics. Principles and Practice,"* Second Edition, Addison-Wesley, 1990.

[7] Hearn, D. and M.P. Baker, *"Gráficas por Computadora,"* Prentice-Hall, 1996.

[8] Foley, J.D. and A. van Dam and S.K. Feiner and J.F. Hughes and R.L. Phillips, *"Introducción a la Graficación por Computadora,"* Addison-Wesley Iberoamericana, 1996.

[9] Pappas, C.H. & W.H. Murray *"Manual de Borland C++ 4.0,"* Osborne/McGraw-Hill, 1994.