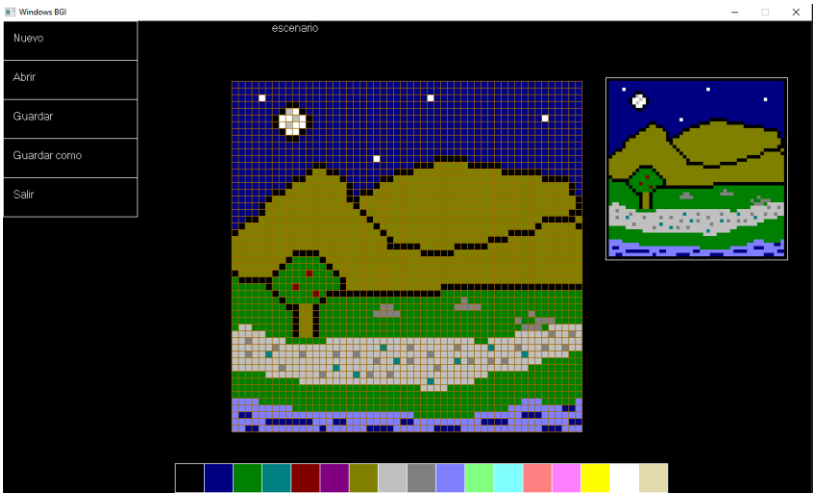
UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ



EDITOR DE SPRITES “EDITOR.C”

ALUMNO: MENDOZA RODRÍGUEZ FERNANDO

CLAVE: 285476

MATERIA: LENGUAJES DE PROGRAMACIÓN

PROFESOR: JOSÉ IGNACIO NÚÑEZ VARELA

ÍNDICE

Página

- **Introducción 1**
- **Objetivo 1**
- **Desarrollo del proyecto 2**
  - **Archivos 2**
    - **Headers 2**
    - **main.c 2**
    - **screen.h/screen.c 3**
    - **boton.h/boton.c 4**
    - **grid.h/grid.c 5**
    - **text.h/text.c 6**
    - **preview.h/preview.c 6**
    - **colors.h/colors.c 7**
    - **file.h/file.c 7**
  - **Modo gráfico 7**
    - **WinBGIm 7**
  - **Interfaz 8**
  - **Almacenamiento 9**
- **Problemas/pruebas 10**
- **Comentarios finales 11**
- **Manual de usuario 12**
  - **Crear un nuevo dibujo 12**
  - **Abrir un dibujo 16**

INTRODUCCIÓN

El editor de sprites es un programa hecho en el lenguaje de programación C con el paradigma estructurado, su función consiste en poder crear, guardar y editar dibujos estilo sprite (hechos con mapas de bits) por medio de una malla aprovechando la funcionalidad del modo gráfico, para posteriormente generar archivos binarios, los cuales contienen la información necesaria para ser interpretados con ayuda de la biblioteca WinBGIm.

El proyecto hace uso de estructuras para lograr una mejor implementación y facilitar el trabajo, todas las estructuras se utilizan con fines de control o almacenamiento.

OBJETIVO

El objetivo del proyecto puede ser visto de manera genérica como crear un editor de imágenes para diseñar los sprites que se deseen.

Específicamente este editor fue hecho ya que es necesario para crear los sprites utilizados en el proyecto del videojuego para la materia Estructuras de datos II.

DESARROLLO DEL PROYECTO

En esta sección se explicará cómo está estructurado el editor, de esta manera será fácil comprender a grandes rasgos como es su funcionamiento interno.

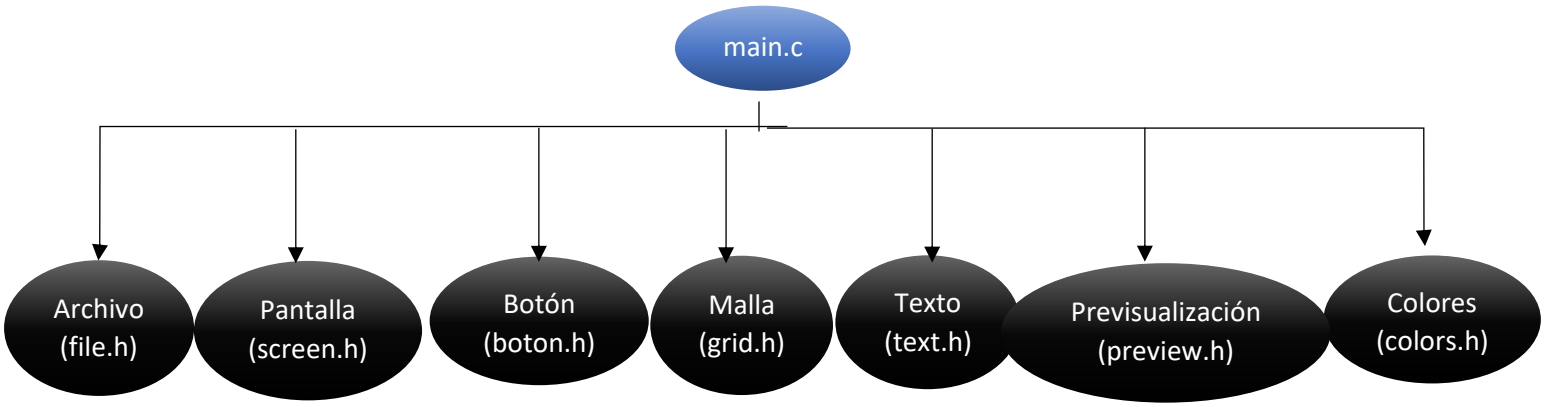
ARCHIVOS

Headers

Los archivos donde se declaran las funciones y estructuras son documentos de texto con terminación .h, se les conoce como headers y contienen dicha parte del código fuente.

Junto a cada archivo de tipo header se tiene que incluir un archivo de tipo .c el cual es el código fuente (source code) y contiene la definición de lo declarado en el header.

Este es un esquema con los archivos header utilizados en el proyecto:



main.c

```
int main()
```

- Aquí son declaradas e inicializadas todas las variables de tipo estructura necesarias para el programa (botones y malla).
- Se inicializa la ventana.
- Son llamadas las funciones de las pantallas (dibujado e interacción tanto de la pantalla principal como de la de dibujo, contenidas en screen.h).

screen.h/screen.c

```
void drawMainScreen (TButton bN,TButton bO,TButton bC);
```

- Recibe todos los botones de la pantalla principal (Nuevo, Abrir y Cerrar) y manda llamar a la función de dibujado para cada uno de ellos (en botón.h).
- Dibuja el nombre del editor

```
void drawDrawScreen(TButton bS,TButton bN2,TButton
bC2,TButton bSA,TButton bO2,GRID gr);
```

- Recibe los botones que estarán en la pantalla de dibujo (Guardar, Nuevo, Cerrar, Guardar como, Abrir) y la malla, después manda a llamar la función para dibujar cada uno de los botones (boton.h), la previsualización (contenido y contorno dentro de preview.h) y la malla (grid.h), también se manda llamar a la función para dibujar la barra de colores (colors.h).

```
int useMainScreen(TButton bN,TButton bO,TButton bC,TButton
iS,GRID*gr);
```

- Recibe los tres botones mencionados anteriormente y la malla.
- Dentro de un while se checan los botones, para esto se manda llamar en un condicional la función para verificar que las coordenadas del mouse estén dentro de las del botón (boton.h).
- Al presionar Nuevo se mandará llamar a una función para pedir las dimensiones de la malla (grid.h).
- Al presionar Abrir se mandará llamar una función que pedirá el nombre de un archivo para leer una malla (grid.h).
- Con cerrar solo saldrá del editor.

```
void useDrawScreen(GRID *gr,TButton bS,TButton bN2,TButton
bC2,TButton bSA,TButton bO2,int c_Size,TButton iS);
```

- Al igual que la función anterior recibe los botones correspondientes y la malla además del tamaño de un color en la barra de colores, checa todos los botones cuando se da un clic izquierdo, así como también la malla y la barra de colores.
- Los botones “Nuevo”, “Abrir” y “Cerrar” hacen lo mismo, solo que antes se llamará una función la cual pregunta si se desea salir sin guardar (text.h).
- El botón “Guardar” mandará llamar una función para guardar (file.h) y si es la primera vez se pedirá el nombre del archivo y lo creará, de lo contrario solo sobrescribirá el archivo.
- El botón “Guardar como” pedirá el nombre del archivo y de no existir lo creará (file.h).

- Si se da clic en la barra de colores se seleccionará el color de la posición del cursor.
- Si se da clic en la maya se llamará una función que buscará el cuadro presionado y modificará el valor en la matriz (grid.h).

boton.h/boton.c

```
typedef struct
{
    int x_ini,y_ini,height,width;
    char label[20];
}TButton;
```

- “x_ini” y “y_ini” para el punto inicial.
- “height” y “width” para las dimensiones.
- Cadena “label” para el nombre.

```
void drawButton(TButton bt);
```

- Recibe la estructura de un botón y lo dibuja con las dimensiones especificadas, de color negro y el mensaje encima.

```
void createButton(TButton *bt,int x, int y,int w,int h,char
label[20]);
```

- Recibe las características de un botón y una estructura de botón por referencia.
- Copia los valores recibidos a los campos de la estructura.

```
int verifyCordinates (int xi,int yi,int xf,int yf);
```

- Recibe el punto inicial y las dimensiones de un botón.
- Verifica que la posición del ratón este dentro de las coordenadas del botón, regresa 1 si se cumple y 0 si no.

```
void createButtons (TButton*bN, TButton*bO, TButton*bC,
TButton*bS, TButton*bN2, TButton*bC2, TButton*bSA,
TButton*bO2, TButton*iS);
```

- Solo es una función que recibe los apuntadores de los botones que se desean crear, llama la función createButton para cada uno.

grid.h/grid.c

```
typedef struct
{
    char name[25];
    int row,col;
    int cellsize;
    int pixSize;
    int**grid;
}GRID;
```

- “name” es una cadena para el nombre del dibujo.
- “row” y “col” son las dimensiones de la malla que contendrá el mapa de bits.
- “cellsize” es el tamaño de dibujado de un cuadro en la malla.
- “pixSize” es el tamaño de dibujado de un cuadro en la previsualización.
- “grid” es un apuntador a apuntadores, nos permitirá asignar los espacios de memoria necesarios para almacenar la información de las dimensiones pedidas.

```
void getSettings(GRID*gr,TButton iS);
```

- Recibe un apuntador a la malla a modificar y una estructura de tipo botón (la cual se usará como si fuera una pantalla sobre la cual imprimir mensajes).
- Se escanean los valores llamando a la función intextxy (en text.h).

```
void initGrid(GRID*gr, int size,int pSize);
```

- Recibe la malla a modificar, el tamaño con el cual será dibujados los cuadros de la malla y de la previsualización.
- Asigna el espacio para la memoria (dinámico).
- Inicializa los valores que contiene todos los espacios

```
void freeGrid(GRID*gr);
```

- Libera los apuntadores de la maya (matriz) para poder usar de nuevo la estructura

```
void drawGrid(GRID gr);
```

- Recibe la maya y la dibuja en pantalla partiendo desde el centro y utilizando el tamaño especificado para cada cuadro

```
int openGrid(TButton iS,GRID*gr,int oA);
```

- Recibe el botón que será ocupado como pantalla, la estructura malla para guardar la información y el modo de apertura, 0 si se acaba de abrir el programa y 1 si ya estaba ocupada la estructura porque se necesita liberarla).
- Pide el nombre del archivo y llama a la función para leerlo (file.h).
- Regresa 1 si se pudo leer y 0 si no.

```
void verifyGrid(GRID*gr,int color);
```

- Recibe la malla y el color seleccionado.
- Si se hace clic dentro de la malla busca en que cuadro fue y lo cambia de color a la vez que cambia el valor de la matriz.

text.h/text.c

```
void intextxy (int x, int y, char text[]);
```

- Recibe como parámetros las coordenadas de donde se comenzará a capturar el texto y una cadena para guardar el resultado.
- Sirve para capturar texto carácter por carácter y se guarda en la cadena.
- Dibuja la animación de un cursor y espera a que se introduzca texto.

```
int verifyExit(TButton iS,GRID*gr);
```

- Recibe un botón para usarlo de pantalla, se imprime la pregunta sobre él y se pide una respuesta (si/no)

preview.h/preview.c

```
void drawSprite(GRID gr);
```

- Recibe la malla y dibuja el Sprite en la zona de previsualización (definida internamente).

```
void drawPreviewContour(GRID gr);
```

- Recibe la malla y dibuja el contorno de la previsualización.

```
void updatePreview(GRID gr,int r, int c,int color);
```

- Recibe la malla y la posición (renglón, columna) para modificar el color que se haya modificado en la malla.

colors.h/colors.c

```
void drawColors(int c_Size);
```

- Recibe el tamaño de celda de cada color para la barra y los dibuja en la parte inferior.

file.h/file.c

```
void guardaArchivo(GRID gr,char name[],int sM);
```

- Recibe la estructura de tipo malla, una cadena con el nombre de la malla y una bandera que indica si tiene nombre o no, en caso de no tenerlo copiará la cadena recibida en la cadena de la estructura.
- Guarda uno por uno los elementos contenidos en ella en un archivo con el mismo nombre del sprite, los datos de la matriz son guardados al final ya que es dinámico y al no saber su tamaño es más conveniente.

```
int leeArchivo(GRID*gr,char name[],int oA));
```

- Recibe la estructura de tipo malla en donde se guardarán los datos leídos, un nombre para poder abrir el archivo y una bandera que indica si hay un sprite ocupando la estructura, de ser así se libera el arreglo dinámico de la estructura para poder reusarlo con un tamaño diferente.
- Se copian los elementos uno por uno a la estructura de la malla, lo cual e útil porque así obtenemos primero las dimensiones de la malla, con lo que se vuelve a asignar el espacio para la matriz antes de leer sus datos.
- Si encuentra el archivo regresa un 1, de lo contrario regresa un 0.

MODO GRÁFICO

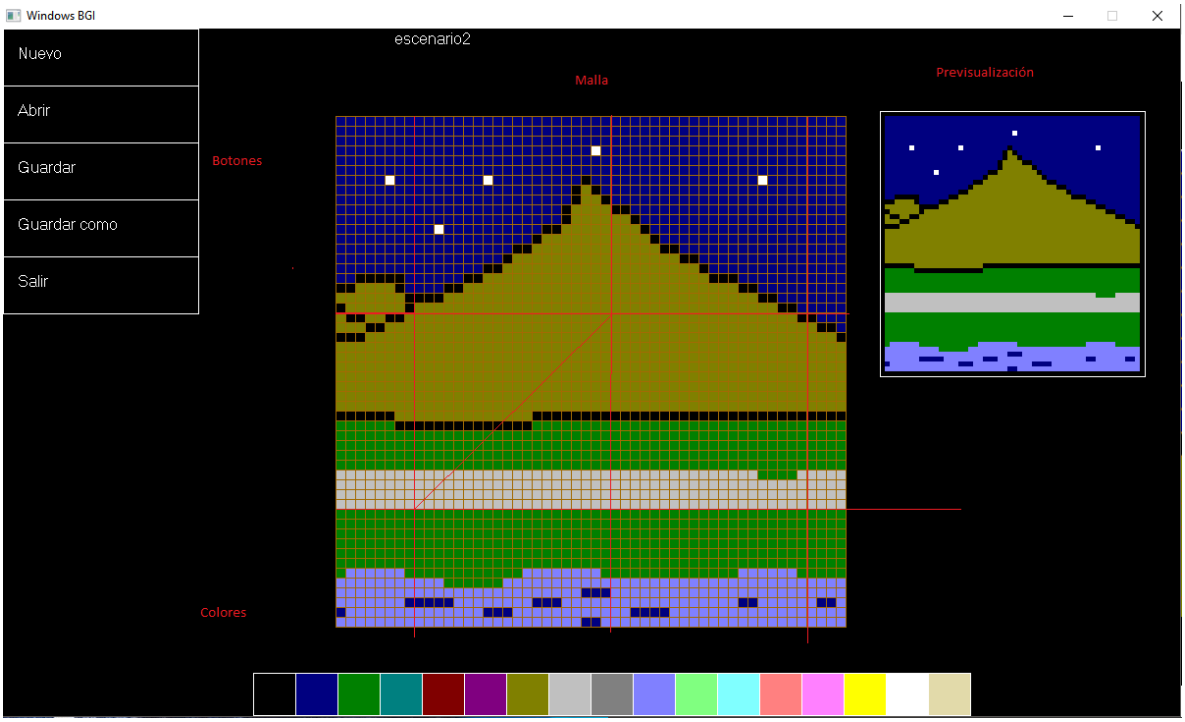
Usualmente el lenguaje c es usado en modo texto (una pantalla negra de consola de comandos en donde se despliega todo el texto que indica el programa y se capturan las entradas), sin embargo, con ayuda de bibliotecas se puede implementar el uso de gráficos (dibujado de figuras y colores).

WinBGIm

Esta biblioteca es de vital importancia para la creación del proyecto, en ella están contenidas todas las funciones usadas dentro del proyecto para crear gráficos, como los botones la malla, el texto o la ventana de previsualización.

El proyecto es un poco complejo como se puede observar, es por eso que se recomienda realizar un diseño (previo al código) el cuál ayudará a organizar las funciones y estructuras, en este caso la separación se realiza con elementos del editor como referencia.

INTERFAZ



La interfaz está compuesta por una malla, una barra de colores, una zona de previsualización y botones, visualmente solo es necesario hacer clic en alguno de los colores para seleccionarlo y pintar en algún cuadro dentro de la malla, sin embargo este proceso es más complejo que solo eso, la malla en realidad está guardada en un arreglo bidimensional dinámico, cada posición del arreglo contiene la información de uno de los colores, en este caso se utilizan los colores de la biblioteca WINBGI la cual contiene los colores que se ven en la imagen de 0 a 15 empezando por el negro, entonces una vez que se hace clic en un cuadro se cambia tanto visualmente como lógicamente modificando el valor al del color seleccionado.

Por otra parte, los botones no están realmente ahí, en realidad lo que se ve en pantalla solo es un dibujo, dentro del programa también se verifica si la posición del mouse está dentro de las coordenadas de un botón (solo cuando se hace clic).

Los colores y la previsualización son meramente visuales y como se mencionó anteriormente son dibujados a partir de la información que se guarda en la matriz dinámica.

ALMACENAMIENTO

Primero se crea un arreglo dinámico (el apuntador necesario se encuentra dentro de la estructura de la malla) con las dimensiones que se le piden al usuario, al inicializarlo se asigna en todas las posiciones un valor que no está dentro del rango de 0 a 15 (porque ese rango son los valores asignados a los colores de WINBGI, de esa manera sabemos que las celdas de la malla no contienen ningún color), posteriormente conforme se modifica el color de una celda en la función correspondiente a la pantalla de edición el valor de ese color es modificado en la matriz, cuando se guarda la información de esa matriz es guardada en un archivo binario, habiendo guardado antes la demás información del sprite (número de renglones y columnas, nombre, tamaños de dibujado) ya que será necesaria para reconstruir el arreglo dinámico al abrir el archivo y leer la información que contiene.

PROBLEMAS/ PRUEBAS

MENÚ

Se presentó el problema de que al hacer clic en cualquiera de los botones en algunas ocasiones lo detectaba y entraba a los procesos correspondientes, pero en otras simplemente no hacía nada, el acomodo de la lógica de control era el siguiente:

```
while(1)
{
      if(ismouseclick)
      {
            if(verifyCordinates)
            {
            }
            …
            clearmouseclic
      }
}
```

Se supone que checaba por un tiempo indefinido si se hacía clic, una vez que pasara esto verificaba cuál de los botones había sido presionado y dependiendo de eso se llamarían a las funciones necesarias, como no funcionó

En el menú principal se optó por checar el clic en cada uno de los condicionales para los botones, además en el while se introdujo una bandera (1 o 0) que indica si el menú debe continuar o no (solo cuando ocurrió un error y no se pudo ejecutar la opción seleccionada), ya que usualmente solo sería un clic el mouse se limpia cuando se salga del menú (el ciclo), en caso de ocurrir algún error se limpiará el mouse dentro del mismo condicional del botón

COMENTARIOS FINALES

Este proyecto me ayudo a aprender acerca de asignación de memoria dinámica, separación de código con el uso de headers, uso de archivos binarios, uso de la biblioteca WinBGIm e implementación de interfaces gráficas en Windows por medio del lenguaje C.

Aún hay cosas que se podrían pulir y mejorar, pienso que hay funciones que se tienen la posibilidad de ser mejoradas para un funcionamiento más optimo además de mejor pensado y estructurado, el proyecto es fácil de poder adaptar a un cambio ya que está programado de manera modular, podría modificarse una función y el programa seguiría funcionando, pero con el cambio implementado, en otras palabras, tiene cierto grado de escalabilidad.

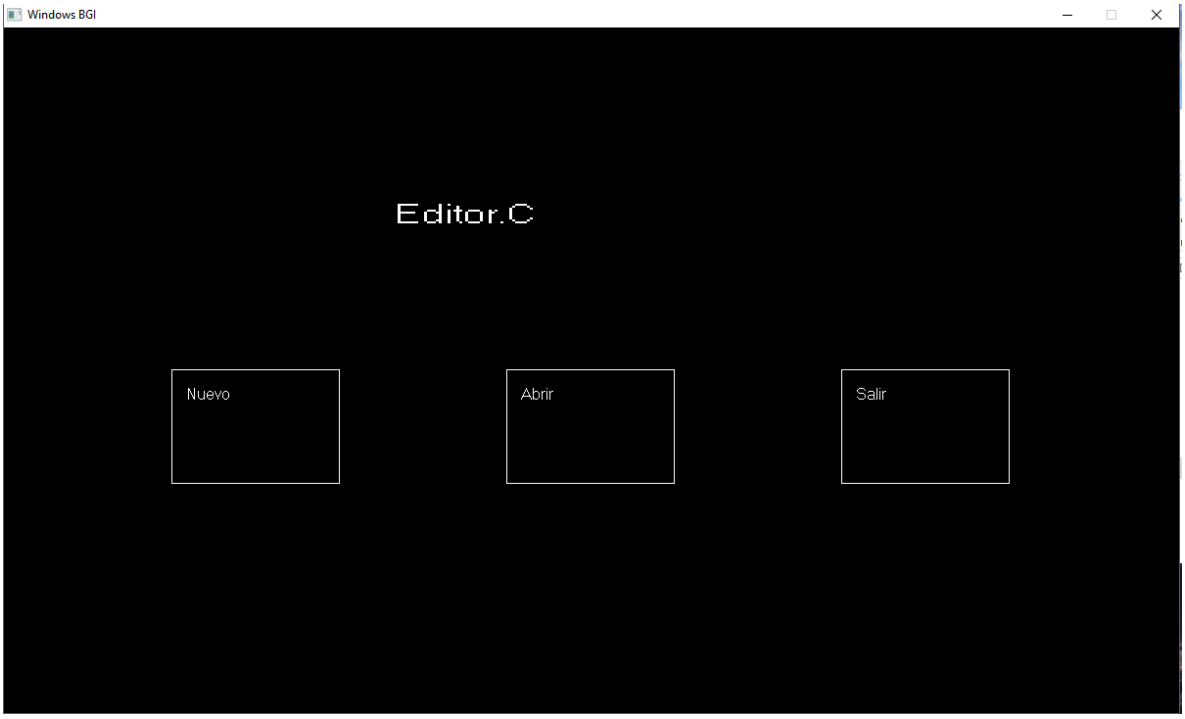
REFERENCIAS

University of Colorado, ”Borland Graphics Interface (BGI) for Windows”, http://www.cs.colorado.edu/~main/cs1300/doc/bgi/bgi.html, 2004

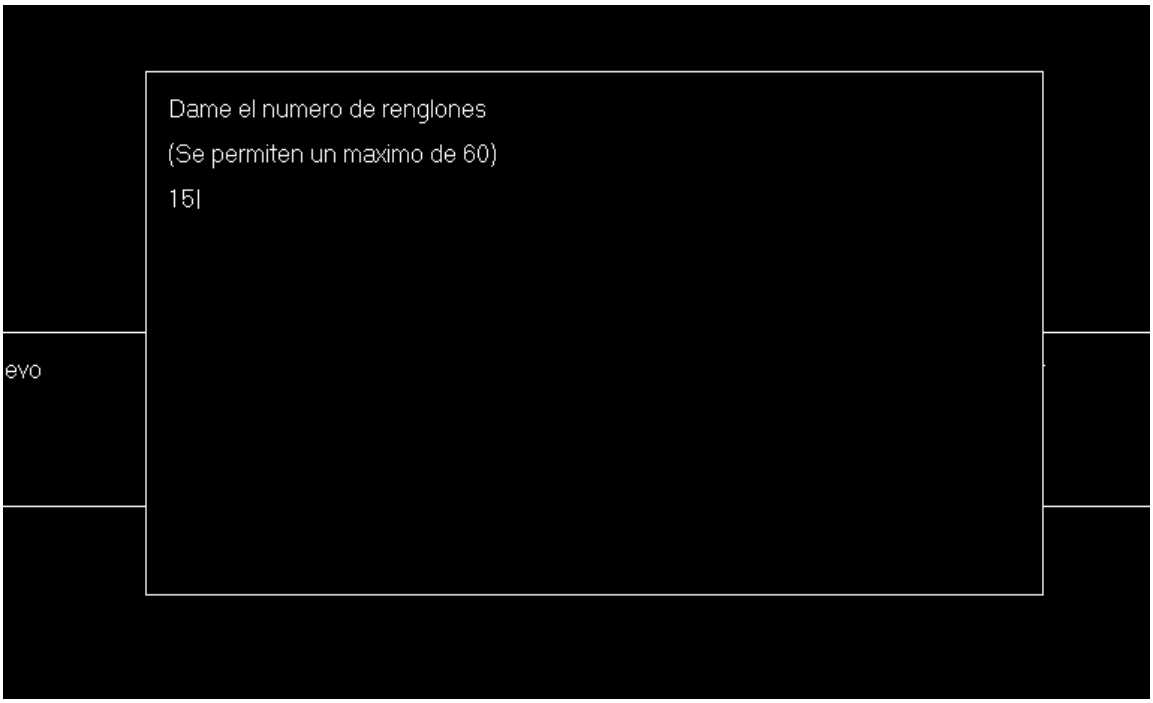
MANUAL DE USUARIO

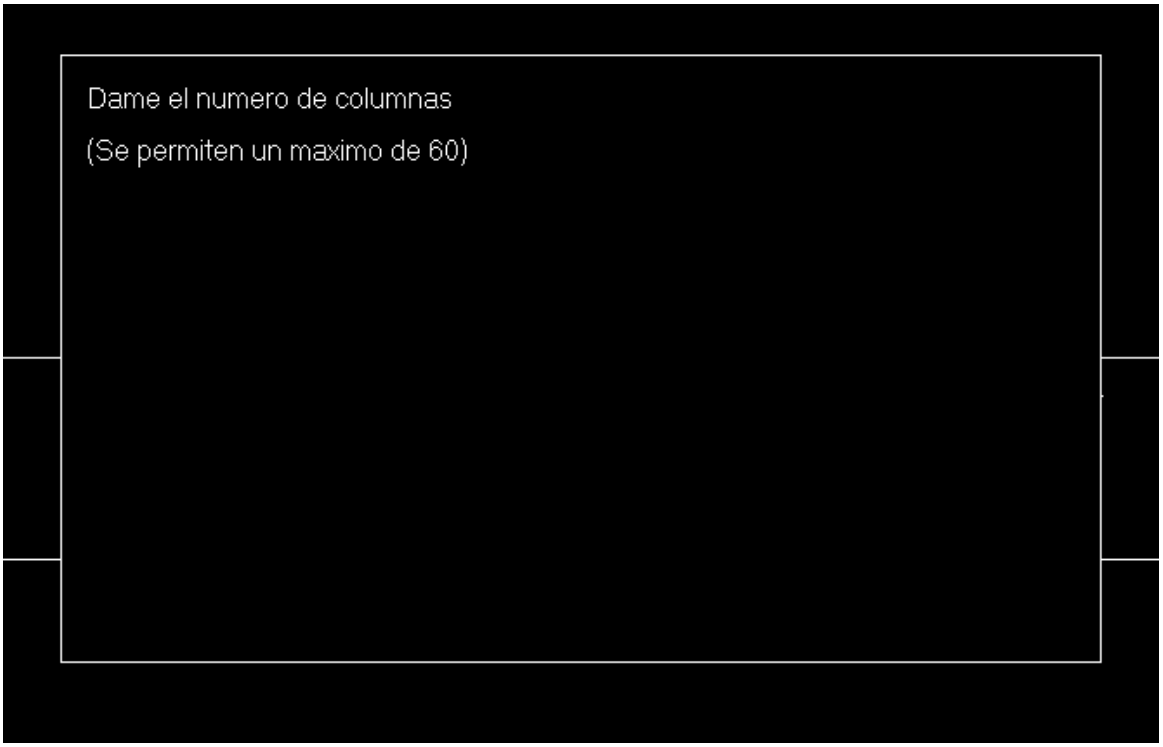
En este documento aprenderás como usar el editor de sprites, así podrás crear tus propias imágenes y tal vez usarlas en un proyecto en donde puedas hacer uso de archivos binarios.

CREAR UN NUEVO DIBUJO



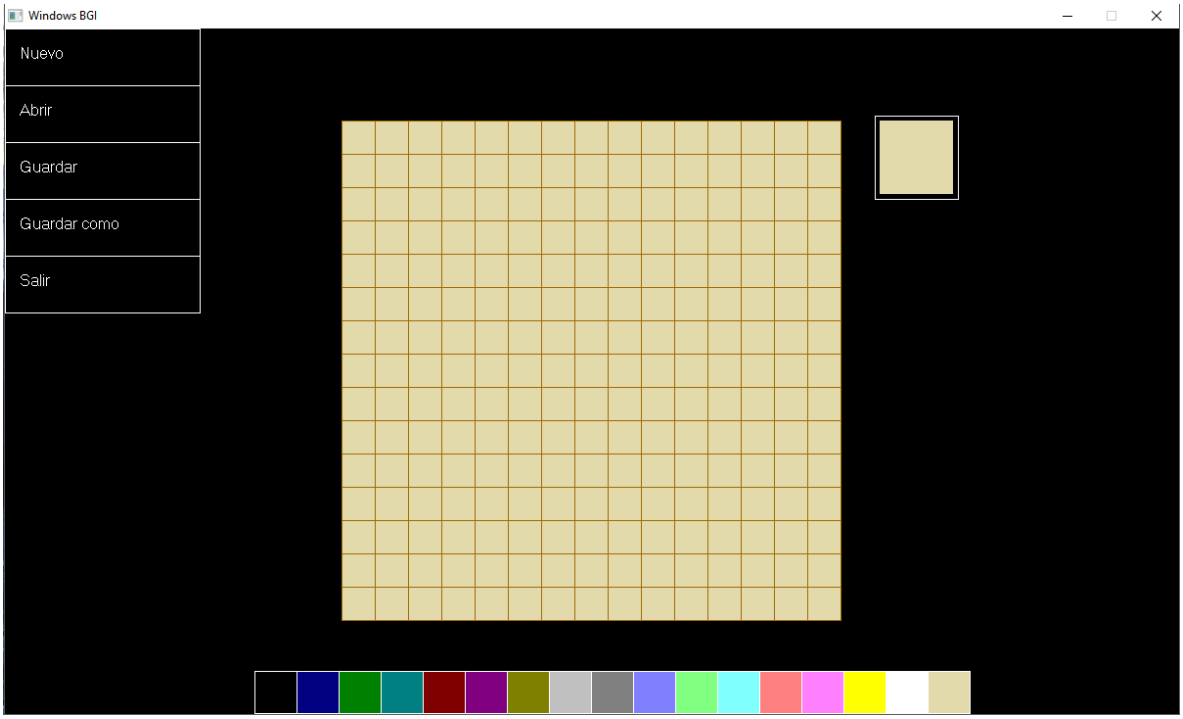
Para comenzar, en la pantalla principal tenemos que hacer el clásico clic izquierdo en el botón nuevo, después de lo cual saldrá una mini pantalla, la cual nos pide el número de renglones y después de columnas, introduciremos cada una seguida de la tecla intro.





Para ejemplificar se introdujeron 15 renglones y 15 columnas.

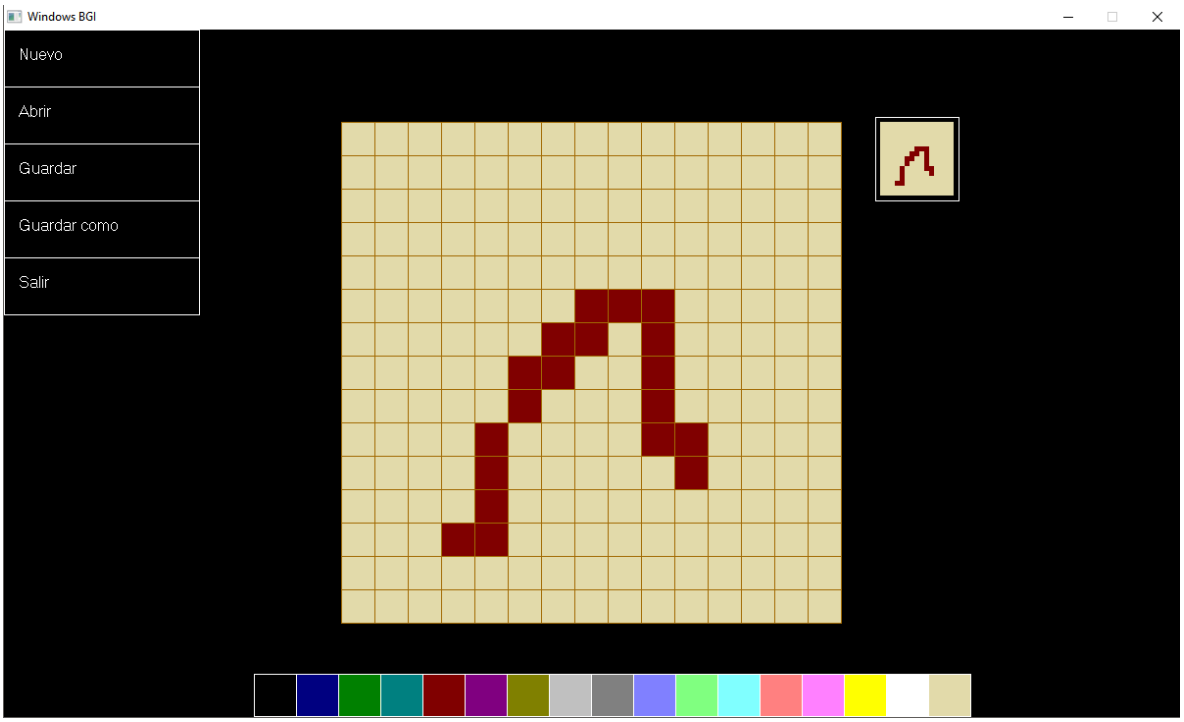
Después aparecerá en pantalla una barra de colores, botones de opciones una zona de previsualización y la malla con las dimensiones especificadas anteriormente



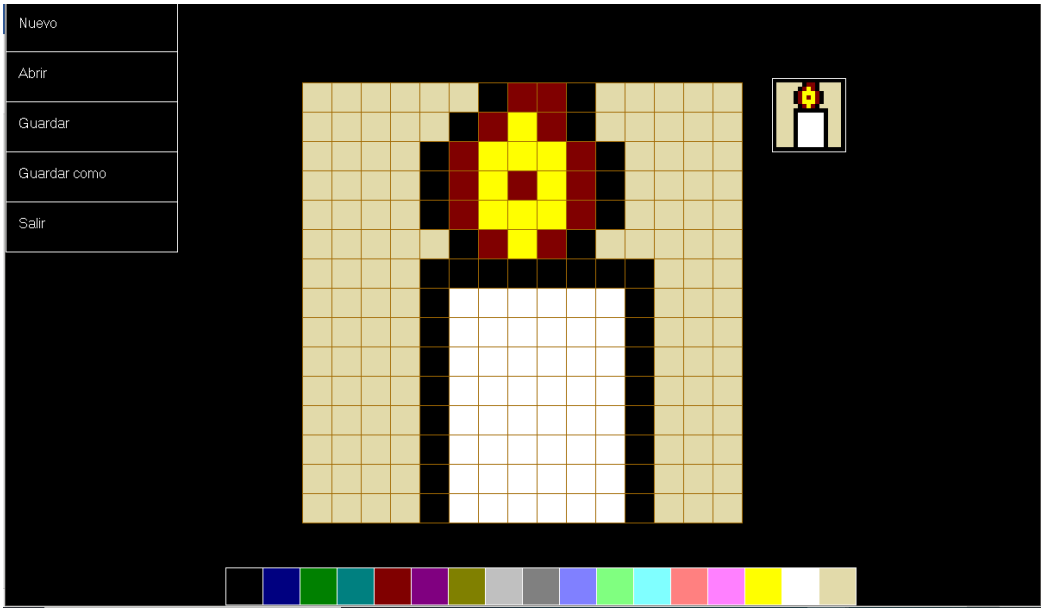
Barra de colores: Bastará con que hagas clic izquierdo sobre uno de los 16 colores (El ultimo color es una goma) para seleccionarlo.

Malla: Una vez seleccionado un color puedes cliquear cualquier celda de la malla, inmediatamente este tomará el color que seleccionaste, no es necesario que presiones uno por uno las celdas, también puedes arrastrar el mouse sin soltar el botón de clic y seguirá funcionando.

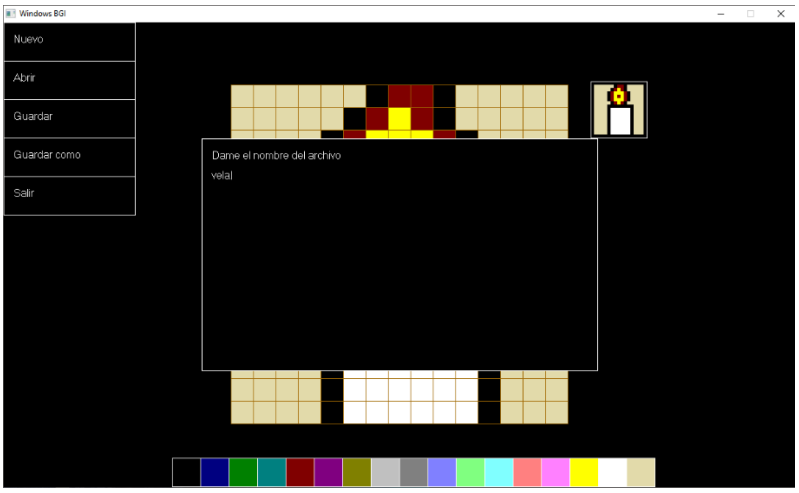
Previsualización: Una vez que una celda es “pintada” se podrá ver el efecto en la zona de previsualización, básicamente es la misma imagen que estamos haciendo en la malla, pero con dimensiones más pequeñas, es una muestra de cómo se verá el sprite fuera del editor (sin el dibujo de las celdas).



Un ejemplo de un dibujo hecho con el editor.



¡Con el editor de imágenes podrás realizar sprites realmente geniales!

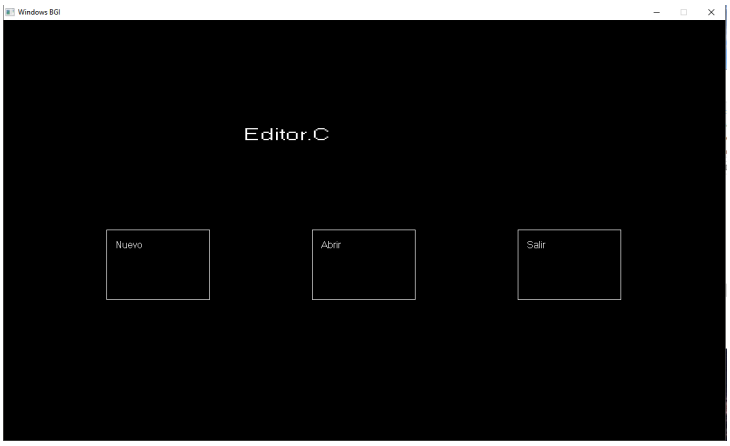


Guardar: Si damos clic sobre el botón guardar la primera vez saldrá el mismo recuadro de mensajes esperando que introduzcamos el nombre del sprite, si ya ha sido guardado antes ya tiene un nombre, por tanto, no es necesario pedirlo y solo lo guarda.

Guardar como: Si damos clic en este botón siempre pedirá el nombre del archivo y lo guardará como uno nuevo, en caso de darle el nombre de un archivo existente lo sobrescribirá.

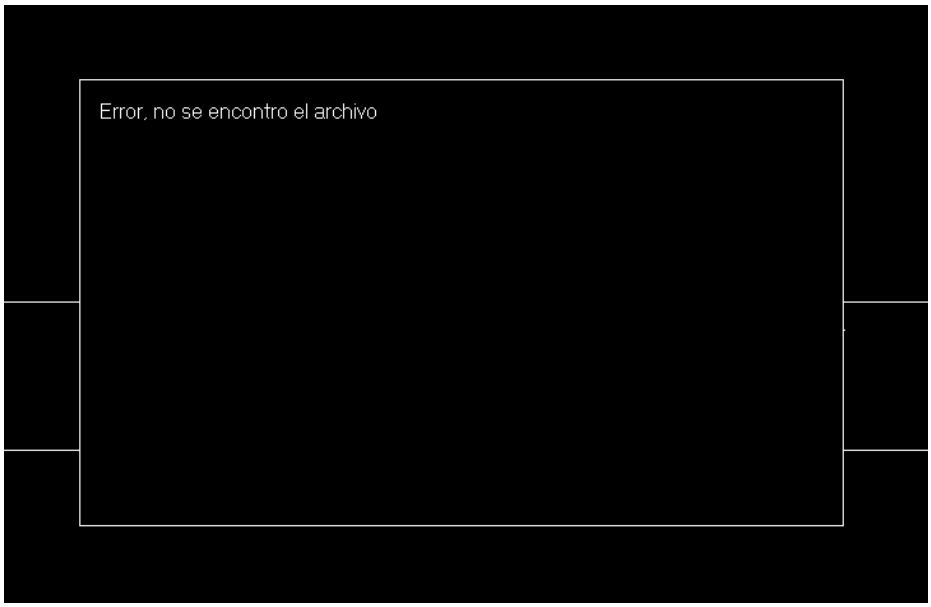
Salir: Se desplegará el recuadro de información preguntando si se quiere salir sin guardar, si contestamos que si entonces cerrará el editor.

ABRIR UN DIBUJO

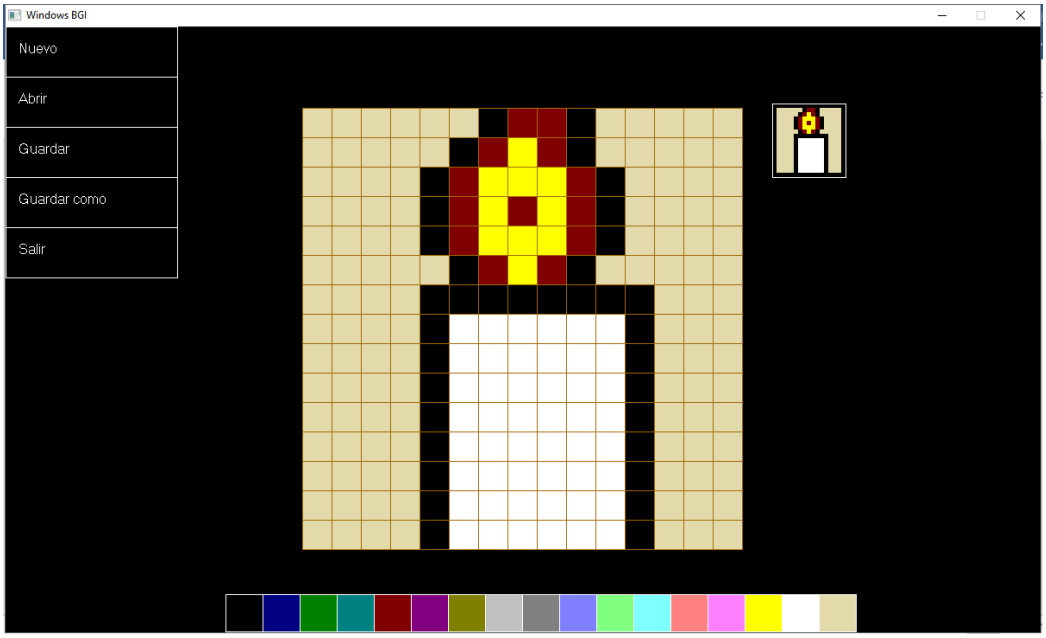


Después de haber guardado un archivo este podrá ser abierto por el editor, regresando a la pantalla principal, si damos clic en el botón abrir saldrá el recuadro de mensajes esperando a que introduzcamos el nombre del archivo que pretendemos abrir.

Después de insertar el nombre podría ser que el archivo en realidad no esté ahí, en tal caso aparecerá el mensaje en el recuadro de información.



Si se encontró el archivo nos enviará a la pantalla de dibujo con el archivo.



Nos encontraremos todo tal cual lo dejamos la última vez.

Nuevo: Al igual que salir preguntará si se desea salir del dibujo sin guardar, si se contesta que si pedirá nuevas dimensiones para un nuevo dibujo.

Abrir: Preguntará si se desea salir sin guardar y al contestar que si pedirá el nombre de un archivo (como en la pantalla principal).