2 像素画板

Requirement

1. 相关功能的使用效果情况。
2. 解释代码逻辑和自己使用的算法思路。

创建一个256 × 256 画幅的像素画板

功能要求

应用程序应当至少包含以下几个具体功能

（1） **画板**：一个256 × 256 像素画幅的画板

（2） **调色板和画笔**：鼠标左键点选相应的前景颜色。

（3） **画笔工具**：鼠标左键点击实现在画板上相应的像素实现对相应像素上色。

（4） **油漆桶工具**：使用前景色填充颜色相近的区域（实现对于连续区域内实现一次性上色）。

（5） **橡皮擦**：将相应的像素的颜色擦除。

（6） **撤销**：撤销上一步操作。

提示

1. 对于本题中所需要的图形部分，可以采用EasyX库实现，EasyX库是针对C/C++免费绘图库，支持VC6-

VC2022，blog.csdn.net

简单易用，学习成本极低，直接查阅手册进行函数调用即可

EasyX手册地址如下：EasyX 文档函数说明

(请注意！创建.c文件调用graphics.h会出现报错，请创建.cpp文件)

头文件的调用

2. 调色板选色、油漆桶、橡皮擦等工具的切换可以不考虑将其设计为一个按钮对象，

只需要规划相应按钮范围，判断鼠标点击处的位置坐标即可以实现，

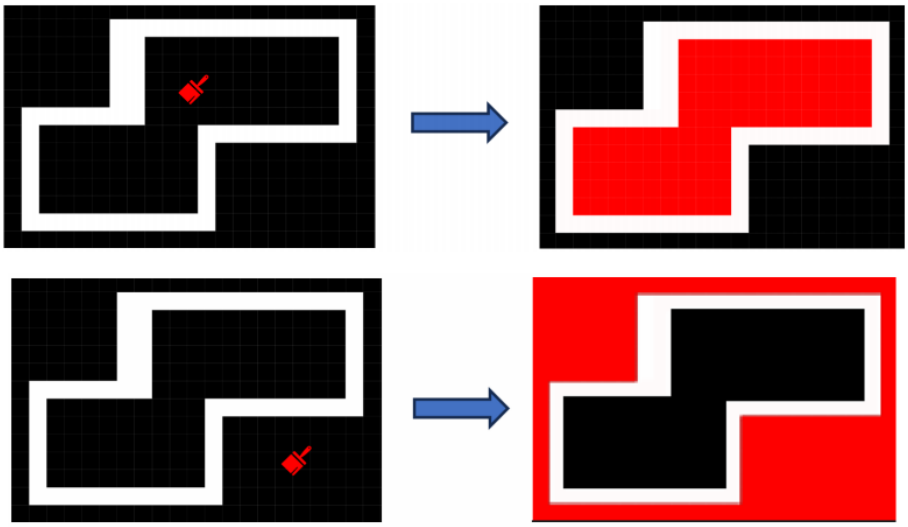
同理对于画板的操作也可以参考这种方式，鼠标事件的参考资料

- 可以查看微软开发文档Windows 和消息Win32 apps | Microsoft Learn。
- https://learn.microsoft.com/zh-cn/windows/win32/api/_winmsg/

3. 油漆桶工具的实现，要求使用所选的前景色实现对连续的区域内一次性上色，具体效果参考如下图。

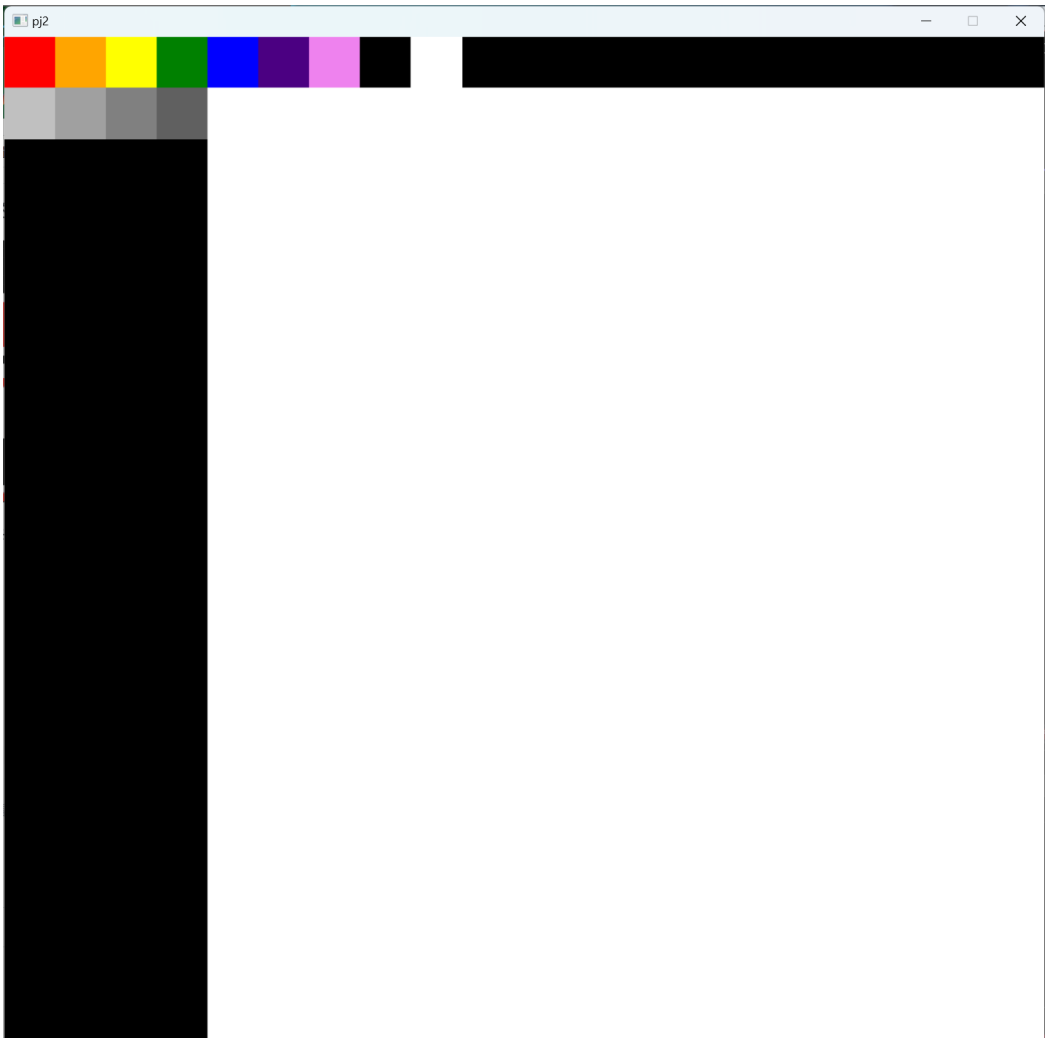
算法提示：鼠标所处区域的像素点开始，向外开始搜索连续且相同颜色的像素块，直到遇到不同颜色的像素块位置

（注意画板边缘的判断），对所有相同颜色的像素块实现一次性上色。



4. 撤销功能的实现可以参考栈，对上一次修改情况入栈，撤销时出栈，只需要记录一次操作即可。

一、 相关功能的使用效果情况。

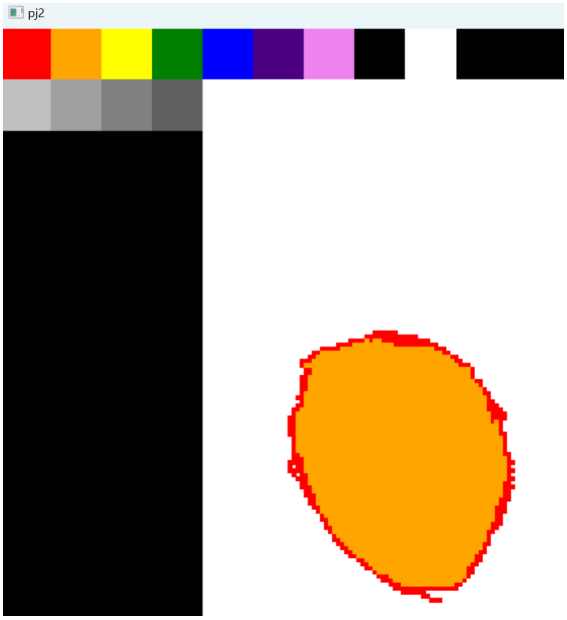


最上面是颜色画板，左键取色

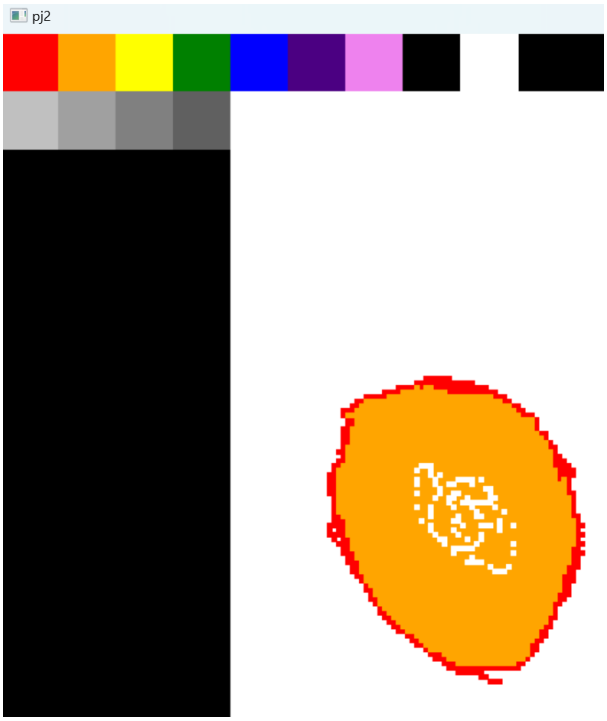
第二排从左到右四种不同颜色的灰色分别对应功能，左键选取功能

【画笔】【油漆桶】【橡皮擦】【撤销】

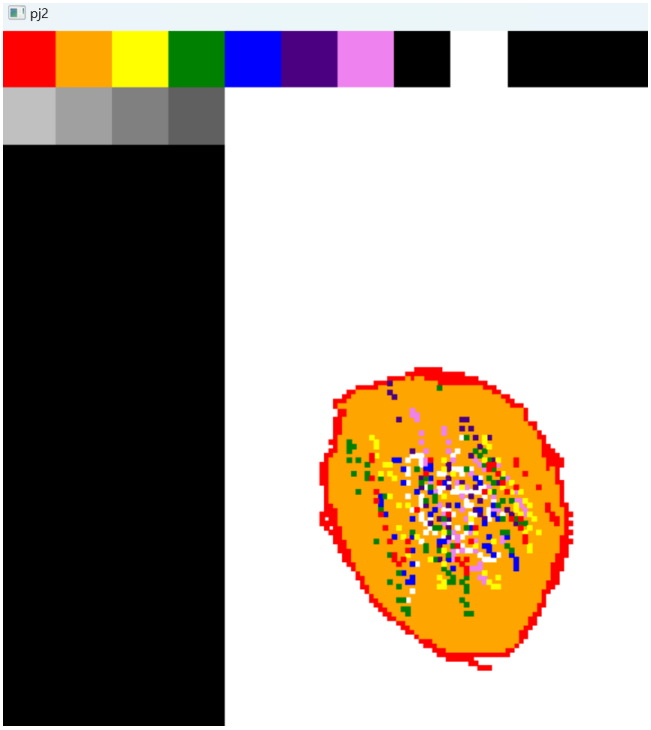
画笔+油漆桶使用效果



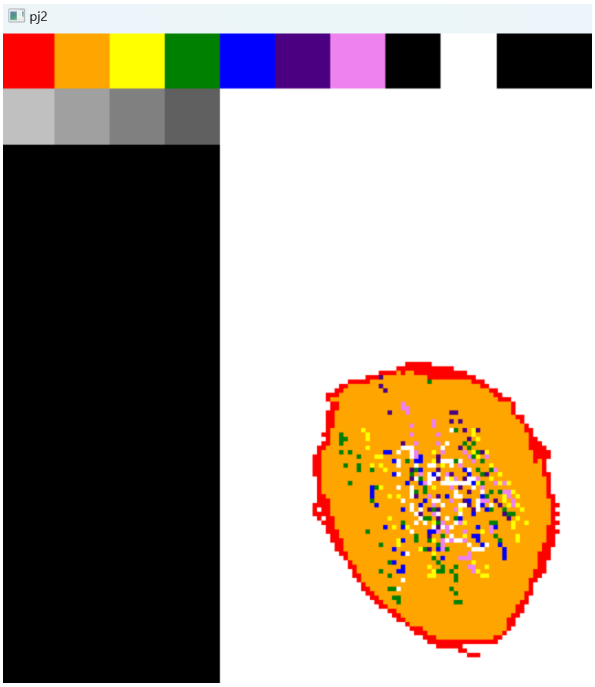
再加橡皮擦使用效果



每种颜色都用上



使用撤销功能



二、解释代码逻辑和自己使用的算法思路。

（1）画板

代码逻辑：定义了一个固定尺寸为256x256逻辑像素的画板，通过二维数组 `canvas` 记录每个像素的颜色信息。

使用 `drawCanvas` 函数遍历 `canvas` 数组，在屏幕上放大绘制每个像素块，形成视觉上的画板效果。

同时，通过 `rectangle` 函数绘制画板边框，明确了画板的边界。

（2）调色板和画笔

代码逻辑：提供一组预设颜色的调色板，用户通过鼠标左键点击选择颜色作为画笔的前景色。

在 `drawPalette` 函数中定义了多种颜色并使用 `setfillcolor` 和 `bar` 函数绘制色块。

当用户点击调色板上某个颜色区域时，根据点击位置计算颜色索引并更新 `currentColor` 。

（3）画笔工具

代码逻辑：允许用户在画板上指定位置，通过鼠标左键点击上色。

在主循环中监听鼠标左键按下事件，计算鼠标点击位置对应画板上的逻辑像素坐标，

如果该位置有效，则根据当前画笔颜色更新像素颜色，并调用 `drawPixel` 即时显示。

（4）油漆桶工具

代码逻辑：选择一个像素后，用当前前景色填充与其颜色相同的相邻像素区域。

算法思路：实现了一个简单的广度优先搜索算法 `bucketFill` 。

当选择油漆桶工具并在画板上点击时，从点击点开始，

检查周围像素颜色是否与点击点相同且不等于当前前景色，

如果是，则将其颜色更改为当前前景色并加入队列继续搜索。

（5）橡皮擦

代码逻辑：擦除画板上选定像素的颜色，一般替换为背景色（本例中为白色）。

橡皮擦功能作为工具之一，在鼠标点击或拖动时，如果当前工具是橡皮擦，

则将对应逻辑像素的颜色设置为白色，并通过 `drawPixel` 更新显示。

（6）撤销

代码逻辑：提供撤销上一步操作的能力，方便用户修正错误。

使用 `std::stack` 维护一个历史状态栈 `history` 。

每次在画板上执行绘图操作之前，都会将当前画板状态保存到栈中。

当用户请求撤销时，从栈顶弹出最近的状态并恢复到画板上，

通过遍历栈顶状态数组更新 `canvas` ，然后重绘整个画板。

代码

```
1 #include <graphics.h>
```

```cpp
#include <conio.h>
#include <stdio.h>
#include <stack>
#include <vector>
#include <queue>
#include <utility>
#include <string>

// 定义常量
const int WINDOW_WIDTH = 1024; // 窗口宽度
const int WINDOW_HEIGHT = 1074; // 窗口高度（包含调色板和工具区）
const int CANVAS_SIZE = 256; // 逻辑画板尺寸
const int PIXEL_SIZE = 4; // 每个像素块的显示尺寸
const int PALETTE_SIZE = 50; // 调色板尺寸
const int TOOL_SIZE = 50; // 工具按钮尺寸
//通过定义像素块大小为4，画布大小为1024，实现1024/4=256的画布边长


// 定义不同程度的灰色
const COLORREF GRAY_COLORS[] = {
    RGB(192, 192, 192), // 浅灰色
    RGB(160, 160, 160), // 中灰色
    RGB(128, 128, 128), // 深灰色
    RGB(96, 96, 96)     // 更深的灰色
};


// 记录每个像素的颜色
COLORREF canvas[CANVAS_SIZE][CANVAS_SIZE];
COLORREF currentColor = BLACK; // 当前选择的颜色

// 操作栈用于撤销功能
std::stack<std::vector<std::vector<COLORREF>>> history;

// 工具类型枚举
enum Tool { PEN, BUCKET, ERASER };
Tool currentTool = PEN; // 当前选择的工具

// 保存当前画布状态
void saveState() {
    std::vector<std::vector<COLORREF>> state(CANVAS_SIZE, std::vector<COLORREF>(CANVAS_SIZE));
    for (int i = 0; i < CANVAS_SIZE; ++i) {
        for (int j = 0; j < CANVAS_SIZE; ++j) {
            state[i][j] = canvas[i][j];
        }
    }
```

```
    history.push(state);
}

// 绘制调色板
void drawPalette() {
    // 定义调色板颜色
    const COLORREF colors[] = {
        RGB(255, 0, 0), // 红色
        RGB(255, 165, 0), // 橙色
        RGB(255, 255, 0), // 黄色
        RGB(0, 128, 0), // 绿色
        RGB(0, 0, 255), // 蓝色
        RGB(75, 0, 130), // 靛色
        RGB(238, 130, 238), // 紫色
        RGB(0, 0, 0), // 黑色
        RGB(255, 255, 255) // 白色
    };
    // 绘制调色板
    for (int i = 0; i < 9; ++i) {
        setfillcolor(colors[i]);
        bar(i * PALETTE_SIZE, 0, (i + 1) * PALETTE_SIZE, PALETTE_SIZE);
    }
}

// 绘制工具区
void drawTools() {
    // 定义工具按钮标签和颜色
    const char* toolNames[] = { "Pen", "Bucket", "Eraser", "Undo" };
    for (int i = 0; i < 4; ++i) {
        setfillcolor(GRAY_COLORS[i]);
        bar(i * TOOL_SIZE, PALETTE_SIZE, (i + 1) * TOOL_SIZE, PALETTE_SIZE +
TOOL_SIZE);
    }
    //for (int i = 0; i < 4; ++i) {
    //    setfillcolor(LIGHTGRAY);
    //    bar(i * TOOL_SIZE, PALETTE_SIZE, (i + 1) * TOOL_SIZE, PALETTE_SIZE +
TOOL_SIZE);
    //    settextcolor(DARKGRAY);
    //    //outtextxy(i * TOOL_SIZE + 10, PALETTE_SIZE + 15, toolNames[i]);
    //}
}

// 绘制画板
void drawCanvas() {
    // 绘制画板边框
    rectangle(TOOL_SIZE * 4, PALETTE_SIZE, TOOL_SIZE * 4 + CANVAS_SIZE *
PIXEL_SIZE, PALETTE_SIZE + CANVAS_SIZE * PIXEL_SIZE);
```

```
    // 遍历画板每个像素，放大绘制
    for (int i = 0; i < CANVAS_SIZE; ++i) {
        for (int j = 0; j < CANVAS_SIZE; ++j) {
            setfillcolor(canvas[i][j]);
            bar(TOOL_SIZE * 4 + i * PIXEL_SIZE, PALETTE_SIZE + j * PIXEL_SIZE,
TOOL_SIZE * 4 + (i + 1) * PIXEL_SIZE, PALETTE_SIZE + (j + 1) * PIXEL_SIZE);
        }
    }
}

// 绘制单个像素
void drawPixel(int x, int y) {
    setfillcolor(canvas[x][y]);
    bar(TOOL_SIZE * 4 + x * PIXEL_SIZE, PALETTE_SIZE + y * PIXEL_SIZE,
TOOL_SIZE * 4 + (x + 1) * PIXEL_SIZE, PALETTE_SIZE + (y + 1) * PIXEL_SIZE);
}

// 绘制工具和颜色状态
void drawStatus() {
    setfillcolor(currentColor);
    bar(0, WINDOW_HEIGHT - PALETTE_SIZE, PALETTE_SIZE, WINDOW_HEIGHT);
    settextcolor(DARKGRAY);
    setbkmode(TRANSPARENT);
    std::string toolText = "Tool: ";
    if (currentTool == PEN) toolText += "Pen";
    else if (currentTool == BUCKET) toolText += "Bucket";
    else if (currentTool == ERASER) toolText += "Eraser";
    //outtextxy(PALETTE_SIZE + 10, WINDOW_HEIGHT - PALETTE_SIZE + 10,
toolText.c_str());
}

// 油漆桶填充算法（使用广度优先搜索）
void bucketFill(int x, int y, COLORREF oldColor) {
    // 边界检查和颜色检查
    if (x < 0 || x >= CANVAS_SIZE || y < 0 || y >= CANVAS_SIZE || canvas[x][y]
!= oldColor || canvas[x][y] == currentColor) {
        return;
    }

    std::queue<std::pair<int, int>> q;
    q.push(std::make_pair(x, y));
    canvas[x][y] = currentColor;
    drawPixel(x, y); // 绘制填充的像素

    // 广度优先搜索
    while (!q.empty()) {
        std::pair<int, int> current = q.front();
```

```
        q.pop();
        int cx = current.first;
        int cy = current.second;

        // 检查上下左右四个方向
        std::vector<std::pair<int, int>> directions = {
            std::make_pair(1, 0), // 向右
            std::make_pair(-1, 0), // 向左
            std::make_pair(0, 1), // 向下
            std::make_pair(0, -1) // 向上
        };
        for (std::pair<int, int> direction : directions) {
            int nx = cx + direction.first;
            int ny = cy + direction.second;
            if (nx >= 0 && nx < CANVAS_SIZE && ny >= 0 && ny < CANVAS_SIZE &&
canvas[nx][ny] == oldColor) {
                canvas[nx][ny] = currentColor;
                drawPixel(nx, ny); // 绘制填充的像素
                q.push(std::make_pair(nx, ny));
            }
        }
    }
}

int main() {
    // 初始化图形窗口
    initgraph(WINDOW_WIDTH, WINDOW_HEIGHT);
    cleardevice();

    // 初始化画布为白色
    for (int i = 0; i < CANVAS_SIZE; ++i) {
        for (int j = 0; j < CANVAS_SIZE; ++j) {
            canvas[i][j] = WHITE;
        }
    }

    saveState(); // 保存初始状态

    drawPalette(); // 初始绘制调色板
    drawTools(); // 初始绘制工具区
    drawCanvas(); // 初始绘制画板
    drawStatus(); // 初始绘制当前工具和颜色状态

    while (true) {
        ExMessage msg = getmessage(EX_MOUSE | EX_KEY);

        if (msg.message == WM_LBUTTONDOWN) {
```

```
            int x = msg.x, y = msg.y;
            if (y < PALETTE_SIZE) {
                // 选择调色板颜色
                int colorIndex = x / PALETTE_SIZE;
                const COLORREF colors[] = {
                    RGB(255, 0, 0), // 红色
                    RGB(255, 165, 0), // 橙色
                    RGB(255, 255, 0), // 黄色
                    RGB(0, 128, 0), // 绿色
                    RGB(0, 0, 255), // 蓝色
                    RGB(75, 0, 130), // 靛色
                    RGB(238, 130, 238), // 紫色
                    RGB(0, 0, 0), // 黑色
                    RGB(255, 255, 255) // 白色
                };
                currentColor = colors[colorIndex];
                drawStatus(); // 更新状态显示
            }
            else if (y >= PALETTE_SIZE && y < PALETTE_SIZE + TOOL_SIZE) {
                // 选择工具
                int toolIndex = x / TOOL_SIZE;
                if (toolIndex == 0) currentTool = PEN;
                else if (toolIndex == 1) currentTool = BUCKET;
                else if (toolIndex == 2) currentTool = ERASER;
                else if (toolIndex == 3 && !history.empty()) {
                    // 撤销上一步操作
                    auto lastState = history.top();
                    history.pop();
                    for (int i = 0; i < CANVAS_SIZE; ++i) {
                        for (int j = 0; j < CANVAS_SIZE; ++j) {
                            canvas[i][j] = lastState[i][j];
                        }
                    }
                    drawCanvas(); // 重绘整个画板
                }
                drawStatus(); // 更新状态显示
            }
            else {
                // 在画板上绘制
                x = (x - TOOL_SIZE * 4) / PIXEL_SIZE;
                y = (y - PALETTE_SIZE) / PIXEL_SIZE;
                if (x < CANVAS_SIZE && y < CANVAS_SIZE && x >= 0 && y >= 0) {
                    saveState(); // 保存当前状态
                    if (currentTool == PEN) {
                        canvas[x][y] = currentColor;
                        drawPixel(x, y); // 绘制单个像素
                    }
```

```
                    else if (currentTool == BUCKET) {
                        COLORREF oldColor = canvas[x][y];
                        bucketFill(x, y, oldColor);
                    }
                    else if (currentTool == ERASER) {
                        canvas[x][y] = WHITE;
                        drawPixel(x, y); // 绘制单个像素
                    }
                }
            }
        }
        else if (msg.message == WM_MOUSEMOVE && (msg.lbutton & MK_LBUTTON)) {
            int x = msg.x, y = msg.y;
            x = (x - TOOL_SIZE * 4) / PIXEL_SIZE;
            y = (y - PALETTE_SIZE) / PIXEL_SIZE;
            if (x < CANVAS_SIZE && y < CANVAS_SIZE && x >= 0 && y >= 0) {
                if (currentTool == PEN) {
                    canvas[x][y] = currentColor;
                    drawPixel(x, y); // 绘制单个像素
                }
                else if (currentTool == ERASER) {
                    canvas[x][y] = WHITE;
                    drawPixel(x, y); // 绘制单个像素
                }
            }
        }
        FlushBatchDraw();
    }

    closegraph();
    return 0;
}
```