#include <easyx.h>

#include <graphics.h>

#include <cmath>

#include <vector>

#include <memory>

#include <conio.h>

#include <string>

#include <codecvt>

#include <locale>

#include <sstream>

#include <algorithm>

#include <CommCtrl.h>

#include <CommDlg.h>

#include <Shlobj.h>

#include <tchar.h>

#include <fstream>

#include <iostream>

#include <map>

#include <functional>

#define drawCircleMode drawCircleButton.isPressed

#define drawRectMode drawRectButton.isPressed

#define drawZhexianMode drawZhexianButton.isPressed

#define drawDuoMode drawDuoButton.isPressed

#define drawTuoyuanMode drawTuoYuanButton.isPressed

#define selectMode selectShapeButton.isPressed

#define zoomMode zoomButton.isPressed

#define layerEditMode layereditButton.isPressed

#define modifyLineWidthMode modifyLineWidthButton.isPressed

#define canBeSelected (selectMode || zoomMode || layerEditMode || modifyLineWidthMode)

//全局设定

int THEME = 0; // 0为黑色主题，1为白色主题

bool FRESH\_WHILE\_MOVEMOUSE = false; // 鼠标移动时是否刷新画布

bool OPEN\_TIPS = true; // 是否打开提示

bool OPEN\_DOUBLE\_CLICK = false; // 是否双击

int SCREEN\_WIDTH = 1600;

int SCREEN\_HEIGHT = 900;

static int COLOR = WHITE;

int selectedIndex = -1;

std::string ws;

std::wstring colorToString(int color) {

switch (color) {

case BLACK:

return L"黑色";

case BLUE:

return L"蓝色";

case GREEN:

return L"绿色";

case CYAN:

return L"青色";

case RED:

return L"红色";

case MAGENTA:

return L"品红";

case BROWN:

return L"棕色";

case LIGHTGRAY:

return L"浅灰色";

case DARKGRAY:

return L"深灰色";

case LIGHTBLUE:

return L"浅蓝色";

case LIGHTGREEN:

return L"浅绿色";

case LIGHTCYAN:

return L"浅青色";

case LIGHTRED:

return L"浅红色";

case LIGHTMAGENTA:

return L"浅品红";

case YELLOW:

return L"黄色";

case WHITE:

return L"白色";

default:

return L"色值=" + std::to\_wstring(color);

}

}

void changeTheme()

{

if (THEME == 0)

{

THEME = 1;

setbkcolor(WHITE);

}

else

{

THEME = 0;

setbkcolor(BLACK);

}

}

// 按钮类

class Button {

public:

Button(int left, int top, int right, int bottom, const TCHAR\* text)

: left(left), top(top), right(right), bottom(bottom), text(text), isPressed(false) {}

void drawColorButtom(int color)

{

if (isPressed) {

setlinecolor(RED);

}

else

{

setlinecolor(WHITE);

}

setfillcolor(color);

fillrectangle(left, top, right, bottom);

if (isPressed) {

setlinecolor(RED);

line(left, top, right, bottom);

line(left, bottom, right, top);

}

}

void drawcopyButton(int left, int top, int right, int bottom) {

this->bottom = bottom;

this->left = left;

this->right = right;

this->top = top;

THEME == 0 ? setfillcolor(DARKGRAY) : setfillcolor(LIGHTGRAY);

fillrectangle(left, top, right, bottom);

THEME == 0 ? setlinecolor(YELLOW) : setlinecolor(BLACK);

rectangle(left, top, right, bottom);

settextstyle(21, 0, \_T("Arial"));

THEME == 0 ? settextcolor(YELLOW) : settextcolor(BLACK);

outtextxy((left + right) / 2 - 5 \* 4 + 3, top + 6, \_T("拷贝"));

THEME == 0 ? setlinecolor(YELLOW) : setlinecolor(WHITE);

}

void draw() const {

if (isPressed) {

if (THEME == 0)

setfillcolor(RED);

else {

setfillcolor(LIGHTRED);

}

}

else {

if (THEME == 0)

setfillcolor(BLUE);

else {

setfillcolor(LIGHTBLUE);

}

}

setlinecolor(YELLOW);

if (THEME == 0)

fillrectangle(left, top, right, bottom);

else

fillroundrect(left, top, right, bottom, 10, 10);

settextstyle(24, 0, \_T("Arial"));

settextcolor(WHITE);

outtextxy((left + right) / 2 - 5 \* sizeof(text), top + 13, text);

setlinecolor(WHITE);

}

bool isInside(int x, int y) const {

if(x >= left && x <= right && y >= top && y <= bottom){

setfillcolor(LIGHTRED);

fillrectangle(left, top, right, bottom);

Sleep(50);

return true;

}

else {

return false;

}

}

void press() {

isPressed = true;

}

void release() {

isPressed = false;

}

bool isPressed;

private:

int left, top, right, bottom;

const WCHAR\* text;

int color;

};

Button drawCircleButton(200, 0, 300, 50, \_T("圆"));

Button drawRectButton(300, 0, 400, 50, \_T("矩形"));

Button drawZhexianButton(400, 0, 500, 50, \_T("折线"));

Button drawDuoButton(500, 0, 600, 50, \_T("多边形"));

Button drawTuoYuanButton(600, 0, 700, 50, \_T("椭圆"));

Button selectShapeButton(700, 0, 800, 50, \_T("选择"));

Button zoomButton(800, 0, 900, 50, \_T("缩放"));

Button fillButton(900, 0, 1000, 50, \_T("填充"));

Button duplicateButton(1000, 0, 1100, 50, \_T("复制"));

Button deleteButton(1100, 0, 1200, 50, \_T("删除"));

Button changeLineStyleButton(1200, 0, 1300, 50, \_T("改变线型"));

Button modifyLineWidthButton(1300, 0, 1400, 50, \_T("修改线宽"));

Button insertImageButton(1400, 0, 1500, 50, \_T("置入图片"));

Button layereditButton(1500, 0, 1600, 50, \_T("图层编辑"));

Button theme(1500, 50, 1600, 100, L"画板颜色");

Button saveButton(1500, 100, 1600, 150, \_T("保存工程"));

Button loadButton(1500, 150, 1600, 200, \_T("读取工程"));

Button insertButton(1500, 200, 1600, 250, \_T("置入对象"));

Button copyButton(-100, -100, -25, -25, \_T("拷贝"));

Button WhiteButton(0, 0, 25, 25, \_T(""));

Button RedButton(25, 0, 50, 25, \_T(""));

Button GreenButton(50, 0, 75, 25, \_T(""));

Button BlueButton(75, 0, 100, 25, \_T(""));

Button MagentaButton(100, 0, 125, 25, \_T(""));

Button BrownButton(125, 0, 150, 25, \_T(""));

Button LightGrayButton(150, 0, 175, 25, \_T(""));

Button DarkGrayButton(175, 0, 200, 25, \_T(""));

Button LightBlueButton(0, 25, 25, 50, \_T(""));

Button LightGreenButton(25, 25, 50, 50, \_T(""));

Button LightCyanButton(50, 25, 75, 50, \_T(""));

Button LightRedButton(75, 25, 100, 50, \_T(""));

Button LightMagentaButton(100, 25, 125, 50, \_T(""));

Button CyanButton(125, 25, 150, 50, \_T(""));

Button YellowButton(150, 25, 175, 50, \_T(""));

Button BlackButton(175, 25, 200, 50, \_T(""));

// 定义一个按钮数组，包含所有需要分组的按钮

Button\* buttons[] = {

&drawCircleButton,

&drawRectButton,

&drawZhexianButton,

&drawDuoButton,

&drawTuoYuanButton,

&selectShapeButton,

&zoomButton,

&layereditButton,

&modifyLineWidthButton,

&theme

};

Button\* colourbuttons[] = {

&BlackButton,

&RedButton,

&GreenButton,

&BlueButton,

&MagentaButton,

&BrownButton,

&LightGrayButton,

&DarkGrayButton,

&LightBlueButton,

&LightGreenButton,

&LightCyanButton,

&LightRedButton,

&LightMagentaButton,

&CyanButton,

&YellowButton,

&WhiteButton

};

void CopyTextToClipboard(const std::string& text) {

if (!OpenClipboard(NULL)) {

return;

}

EmptyClipboard();

HGLOBAL hglb = GlobalAlloc(GMEM\_MOVEABLE, text.size() + 1);

if (hglb) {

memcpy(GlobalLock(hglb), text.c\_str(), text.size() + 1);

GlobalUnlock(hglb);

SetClipboardData(CF\_TEXT, hglb);

}

CloseClipboard();

}

LPCWSTR stringToLPCWSTR(std::string str)

{

size\_t size = str.length();

int wLen = ::MultiByteToWideChar(CP\_UTF8,

0,

str.c\_str(),

-1,

NULL,

0);

wchar\_t\* buffer = new wchar\_t[wLen + 1];

memset(buffer, 0, (wLen + 1) \* sizeof(wchar\_t));

MultiByteToWideChar(CP\_ACP, 0, str.c\_str(), size, (LPWSTR)buffer, wLen);

return buffer;

}

std::string wstring2string(std::wstring wstr)

{

std::string result;

//获取缓冲区大小，并申请空间，缓冲区大小事按字节计算的

int len = WideCharToMultiByte(CP\_ACP, 0, wstr.c\_str(), wstr.size(), NULL, 0, NULL, NULL);

char\* buffer = new char[len + 1];

//宽字节编码转换成多字节编码

WideCharToMultiByte(CP\_ACP, 0, wstr.c\_str(), wstr.size(), buffer, len, NULL, NULL);

buffer[len] = '\0';

//删除缓冲区并返回值

result.append(buffer);

delete[] buffer;

return result;

}

std::wstring string2wstring(std::string str)

{

std::wstring result;

//获取缓冲区大小，并申请空间，缓冲区大小按字符计算

int len = MultiByteToWideChar(CP\_ACP, 0, str.c\_str(), str.size(), NULL, 0);

TCHAR\* buffer = new TCHAR[len + 1];

//多字节编码转换成宽字节编码

MultiByteToWideChar(CP\_ACP, 0, str.c\_str(), str.size(), buffer, len);

buffer[len] = '\0'; //添加字符串结尾

//删除缓冲区并返回值

result.append(buffer);

delete[] buffer;

return result;

}

void drawButton() {

BeginBatchDraw();

theme.draw();

drawCircleButton.draw();

drawRectButton.draw();

drawZhexianButton.draw();

drawDuoButton.draw();

drawTuoYuanButton.draw();

selectShapeButton.draw();

zoomButton.draw();

duplicateButton.draw();

deleteButton.draw();

fillButton.draw();

layereditButton.draw();

insertImageButton.draw();

changeLineStyleButton.draw();

modifyLineWidthButton.draw();

saveButton.draw();

loadButton.draw();

insertButton.draw();

BlackButton.drawColorButtom(BLACK);

RedButton.drawColorButtom(RED);

GreenButton.drawColorButtom(GREEN);

BlueButton.drawColorButtom(BLUE);

MagentaButton.drawColorButtom(MAGENTA);

BrownButton.drawColorButtom(BROWN);

LightGrayButton.drawColorButtom(LIGHTGRAY);

DarkGrayButton.drawColorButtom(DARKGRAY);

LightBlueButton.drawColorButtom(LIGHTBLUE);

LightGreenButton.drawColorButtom(LIGHTGREEN);

LightCyanButton.drawColorButtom(LIGHTCYAN);

LightRedButton.drawColorButtom(LIGHTRED);

LightMagentaButton.drawColorButtom(LIGHTMAGENTA);

CyanButton.drawColorButtom(CYAN);

YellowButton.drawColorButtom(YELLOW);

WhiteButton.drawColorButtom(WHITE);

EndBatchDraw();

}

void DrawAllShapes();

class HintManager {

public:

void updateHints(const POINT& mousePos, const std::string& mode, std::string tips) {

this->mousePos = mousePos;

this->mode = mode;

this->tips = tips;

if (FRESH\_WHILE\_MOVEMOUSE)

{

DrawAllShapes();

}

}

void displayHints() const {

settextstyle(16, 0, \_T("Arial"));

settextcolor(LIGHTGRAY);

outtextxy(5, SCREEN\_HEIGHT - 60, stringToLPCWSTR(("点击位置: (" + std::to\_string(mousePos.x) + ", " + std::to\_string(mousePos.y) + ")")));

outtextxy(5, SCREEN\_HEIGHT - 40, stringToLPCWSTR(("模式: " + mode)));

outtextxy(5, SCREEN\_HEIGHT - 20, stringToLPCWSTR("提示: " + tips));

}

void updatePt(POINT pt) {

this->mousePos = pt;

if (FRESH\_WHILE\_MOVEMOUSE)

{

DrawAllShapes();

}

}

private:

POINT mousePos;

std::string mode;

std::string tips;

};

class Shape {

public:

virtual void draw() const = 0;

virtual RECT getBoundingBox() const = 0;

virtual void move(int dx, int dy) = 0;

virtual ~Shape() {}

virtual std::shared\_ptr<Shape> clone() const = 0;

virtual std::string generateLabel() const = 0;

virtual void parseInfoFromStream(std::stringstream& ss) = 0;

int color = COLOR;

int fillcolor = COLOR;

bool is\_fill = false;

int lineWidth = 1; // 默认线宽

int lineStyle = PS\_SOLID; // 默认线型，PS\_SOLID表示实线

virtual void changeLineStyle() {

if (lineStyle == 4)

{

lineStyle = 0;

}

else

{

lineStyle++;

}

}

virtual void addLineWidth() {

lineWidth++;

}

virtual void reduceLineWidth() {

if (lineWidth > 1) {

lineWidth--;

}

}

virtual void zoom(double factor, POINT center) = 0;

virtual void setColor(int co) {

this->color = co;

}

virtual std::wstring getInfo() const = 0;

void setFill(bool fill) {

this->fillcolor = COLOR;

this->is\_fill = fill;

}

virtual void setFillColour(int co) {

this->fillcolor = co;

}

virtual void changePoints(int x, int y) {

}

};

class Image : public Shape {

public:

Image() : Shape(), filename("") {}

// 从文件加载图片

bool loadImage(const std::string& filename) {

this->filename = filename;

loadimage(&m\_image, stringToLPCWSTR(filename));

if (&m\_image != NULL) {

width = m\_image.getwidth();

height = m\_image.getheight();

return true;

}

return false;

}

bool loadImageUsedInLoad(const std::string& filename) {

this->filename = filename;

loadimage(&m\_image, stringToLPCWSTR(filename), width, height);

if (&m\_image != NULL) {

return true;

}

return false;

}

// 绘制图片

void draw() const override {

putimage(topLeft.x, topLeft.y, width, height, &m\_image, 0, 0);

}

void parseInfoFromStream(std::stringstream& ss) {

std::string token;

ss >> token; // 忽略"左上点: "

ss >> token; // (151,358)

// 移除')'

token.pop\_back();

// 移除'('

token.erase(0, wcslen(L"("));

std::stringstream coordStream(token);

coordStream >> topLeft.x;

coordStream.ignore(1, L',');

coordStream >> topLeft.y;

ss >> ws >> token; // 宽度: 642

width = std::stoi(token);

ss >> ws >> token; // 高度: 303

height = std::stoi(token);

ss >> ws >> token; // 文件名: C:\Users\23258\Pictures\1.png

filename = token;

loadImageUsedInLoad(filename);

}

std::string generateLabel() const {

std::wstringstream ss;

ss << L"Image";

ss << L" 左上点: (" << topLeft.x << L"," << topLeft.y << L")";

ss << L" 宽度: " << width;

ss << L" 高度: " << height;

ss << L" 文件名: " << std::wstring\_convert<std::codecvt\_utf8<wchar\_t>>().from\_bytes(filename);

return wstring2string(ss.str());

}

// 获取边界框

RECT getBoundingBox() const override {

return { topLeft.x, topLeft.y, topLeft.x + width, topLeft.y + height };

}

// 移动图片

void move(int dx, int dy) override {

topLeft.x += dx;

topLeft.y += dy;

}

// 缩放图片

void zoom(double factor, POINT zoomCenter) override {

int dx = topLeft.x - zoomCenter.x;

int dy = topLeft.y - zoomCenter.y;

topLeft.x = static\_cast<int>(zoomCenter.x + dx \* factor);

topLeft.y = static\_cast<int>(zoomCenter.y + dy \* factor);

width = static\_cast<int>(width \* factor);

height = static\_cast<int>(height \* factor);

loadimage(&m\_image, stringToLPCWSTR(filename), width, height);

}

// 克隆图片

std::shared\_ptr<Shape> clone() const override {

return std::make\_shared<Image>(\*this);

}

// 获取图片信息

std::wstring getInfo() const override {

std::wstringstream ss;

ss << L"图片: \n左上点=(" << topLeft.x << L"," << topLeft.y << L")\n宽度=" << width << L"\n高度=" << height;

ss << L"\n文件名=" << std::wstring\_convert<std::codecvt\_utf8<wchar\_t>>().from\_bytes(filename);

return ss.str();

}

private:

std::string filename;

POINT topLeft = { 100,100 };

int width, height;

IMAGE m\_image;

};

// 派生类 Circle，表示圆形

class Circle : public Shape {

public:

Circle(POINT center = { 0,0 }, int radius = 0) : center(center), radius(radius) {}

void draw() const override {

setlinecolor(color);

setlinestyle(lineStyle, lineWidth);

if (is\_fill) {

setfillcolor(fillcolor);

solidcircle(center.x, center.y, radius);

circle(center.x, center.y, radius);

}

else {

circle(center.x, center.y, radius);

}

setlinecolor(WHITE);

setlinestyle(PS\_SOLID, 1);

}

void parseInfoFromStream(std::stringstream& ss) {

std::string token;

ss >> token; // 忽略"Circle"

//ss >> ws; // 吃掉空白

ss >> token; // 圆心: (251,308)

//token.erase(0, wcslen(L"圆心: ("));

// 移除')'

token.pop\_back();

// 移除'('

token.erase(0, wcslen(L"("));

std::stringstream coordStream(token);

coordStream >> center.x;

coordStream.ignore(1, L',');

coordStream >> center.y;

ss >> ws >> token; // 半径: 118

//token.erase(0, wcslen(L"半径: "));

radius = std::stoi(token);

ss >> ws >> token; // 线条颜色: 5636095

//token.erase(0, wcslen(L"线条颜色: "));

color = std::stoi(token);

ss >> ws >> token; // 填充状态: 1

//token.erase(0, wcslen(L"填充状态: "));

is\_fill = (token == "1");

ss >> ws >> token; // 填充颜色: 16733695

//token.erase(0, wcslen(L"填充颜色: "));

fillcolor = std::stoi(token);

ss >> ws >> token; // 线宽: 1

//token.erase(0, wcslen(L"线宽: "));

lineWidth = std::stoi(token);

ss >> ws >> token; // 线型: 0

//token.erase(0, wcslen(L"线型: "));

lineStyle = std::stoi(token);

}

std::string generateLabel() const {

std::wstringstream ss;

ss << L"Circle";

ss << L" 圆心: (" << center.x << L"," << center.y << L")";

ss << L" 半径: " << radius;

ss << L" 线条颜色: " << color;

ss << L" 填充状态: " << (int)is\_fill;

ss << L" 填充颜色: " << fillcolor;

ss << L" 线宽: " << lineWidth << L" 线型: " << lineStyle;

return wstring2string(ss.str());

}

RECT getBoundingBox() const override {

return { center.x - radius, center.y - radius, center.x + radius, center.y + radius };

}

void move(int dx, int dy) override {

center.x += dx;

center.y += dy;

}

void zoom(double factor, POINT zoomCenter) override {

int dx = center.x - zoomCenter.x;

int dy = center.y - zoomCenter.y;

center.x = static\_cast<int>(zoomCenter.x + dx \* factor);

center.y = static\_cast<int>(zoomCenter.y + dy \* factor);

radius = static\_cast<int>(radius \* factor);

}

std::wstring getInfo() const override {

std::wstringstream ss;

ss << L"圆: \n半径=" << radius << L"\n圆心=(" << center.x << L"," << center.y << L")";

ss << L"\n线条颜色=" << colorToString(color);

if (is\_fill) {

ss << L"\n填充颜色=" << colorToString(fillcolor);

}

return ss.str();

}

std::shared\_ptr<Shape> clone() const override {

return std::make\_shared<Circle>(\*this);

}

private:

POINT center;

int radius;

};

// 派生类 Rect，表示矩形

class Rect : public Shape {

public:

Rect(POINT topLeft = { 0,0 }, POINT bottomRight = { 0,0 }) : topLeft(topLeft), bottomRight(bottomRight) {}

void draw() const override {

setlinecolor(color);

setlinestyle(lineStyle, lineWidth);

if (is\_fill) {

setfillcolor(fillcolor);

solidrectangle(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y);

rectangle(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y);

}

else {

rectangle(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y);

}

setlinecolor(WHITE);

setlinestyle(PS\_SOLID, 1);

}

void parseInfoFromStream(std::stringstream& ss) {

std::string token;

ss >> token; // 忽略"Rect"

ss >> token; // 左上点: (463,188)

//token.erase(0, wcslen(L"左上点: ("));

token.pop\_back(); // 移除')'

token.erase(0, wcslen(L"(")); // 移除'('

std::stringstream coordStream(token);

coordStream >> topLeft.x;

coordStream.ignore(1, L',');

coordStream >> topLeft.y;

ss >> ws >> token; // 右下点: (684,474)

//token.erase(0, wcslen(L"右下点: ("));

token.pop\_back(); // 移除')'

token.erase(0, wcslen(L"(")); // 移除'('

coordStream.str(token);

coordStream.clear();

coordStream >> bottomRight.x;

coordStream.ignore(1, L',');

coordStream >> bottomRight.y;

ss >> ws >> token; // 线条颜色: 5636095

//token.erase(0, wcslen(L"线条颜色: "));

color = std::stoi(token);

ss >> ws >> token; // 填充状态: 0

//token.erase(0, wcslen(L"填充状态: "));

is\_fill = (token == "1");

ss >> ws >> token; // 填充颜色: 5636095

//token.erase(0, wcslen(L"填充颜色: "));

fillcolor = std::stoi(token);

ss >> ws >> token; // 线宽: 6

//token.erase(0, wcslen(L"线宽: "));

lineWidth = std::stoi(token);

ss >> ws >> token; // 线型: 0

//token.erase(0, wcslen(L"线型: "));

lineStyle = std::stoi(token);

}

std::string generateLabel() const {

std::wstringstream ss;

ss << L"Rect";

ss << L" 左上点: (" << topLeft.x << L"," << topLeft.y << L")";

ss << L" 右下点: (" << bottomRight.x << L"," << bottomRight.y << L")";

ss << L" 线条颜色: " << color;

ss << L" 填充状态: " << (int)is\_fill;

ss << L" 填充颜色: " << fillcolor;

ss << L" 线宽: " << lineWidth << L" 线型: " << lineStyle;

return wstring2string(ss.str());

}

RECT getBoundingBox() const override {

return { topLeft.x, topLeft.y, bottomRight.x, bottomRight.y };

}

void move(int dx, int dy) override {

topLeft.x += dx;

topLeft.y += dy;

bottomRight.x += dx;

bottomRight.y += dy;

}

void zoom(double factor, POINT zoomCenter) override {

int dx1 = topLeft.x - zoomCenter.x;

int dy1 = topLeft.y - zoomCenter.y;

int dx2 = bottomRight.x - zoomCenter.x;

int dy2 = bottomRight.y - zoomCenter.y;

topLeft.x = static\_cast<int>(zoomCenter.x + dx1 \* factor);

topLeft.y = static\_cast<int>(zoomCenter.y + dy1 \* factor);

bottomRight.x = static\_cast<int>(zoomCenter.x + dx2 \* factor);

bottomRight.y = static\_cast<int>(zoomCenter.y + dy2 \* factor);

}

std::shared\_ptr<Shape> clone() const override {

return std::make\_shared<Rect>(\*this);

}

std::wstring getInfo() const override {

std::wstringstream ss;

ss << L"矩形: \n左上点=(" << topLeft.x << L"," << topLeft.y << L")\n右下点=(" << bottomRight.x << L"," << bottomRight.y;

ss << L"\n线条颜色=" << colorToString(color);

if (is\_fill) {

ss << L"\n填充颜色=" << colorToString(fillcolor);

}

return ss.str();

}

void changePoints(int x, int y) {

{

bottomRight.x = x;

bottomRight.y = y;

}

}

private:

POINT topLeft;

POINT bottomRight;

};

// 派生类 Zhexian，表示折线

class Zhexian : public Shape {

public:

void addPoint(POINT pt) {

points.push\_back(pt);

}

void updateLastPoint(POINT pt) {

if (!points.empty()) {

points.back() = pt;

}

}

void parseInfoFromStream(std::stringstream& ss) {

std::string token;

ss >> token; // 忽略"Zhexian"

ss >> token; // 点数: 6

//token.erase(0, wcslen(L"点数: "));

int numPoints = std::stoi(token);

ss >> ws >> token; // 点集 {#(974,178)#(791,438)#(1084,446)#(1050,313)#(1237,319)#(1261,154)}

//token.erase(0, wcslen(L"点集 {#"));

token.pop\_back(); // 移除'}'

token.erase(0, wcslen(L"{#")); // 移除'{#'

std::stringstream pointStream(token);

std::string pointToken;

while (std::getline(pointStream, pointToken, '#')) {

if (!pointToken.empty()) {

pointToken.erase(0, wcslen(L"("));

pointToken.pop\_back(); // 移除')'

std::stringstream pointCoordStream(pointToken);

POINT pt;

pointCoordStream >> pt.x;

pointCoordStream.ignore(1, L',');

pointCoordStream >> pt.y;

points.push\_back(pt);

}

}

ss >> ws >> token; // 线条颜色: 16733695

//token.erase(0, wcslen(L"线条颜色: "));

color = std::stoi(token);

ss >> ws >> token; // 填充状态: 0

//token.erase(0, wcslen(L"填充状态: "));

is\_fill = (token == "1");

ss >> ws >> token; // 填充颜色: 16733695

//token.erase(0, wcslen(L"填充颜色: "));

fillcolor = std::stoi(token);

ss >> ws >> token; // 线宽: 1

//token.erase(0, wcslen(L"线宽: "));

lineWidth = std::stoi(token);

ss >> ws >> token; // 线型: 0

//token.erase(0, wcslen(L"线型: "));

lineStyle = std::stoi(token);

}

void draw() const override {

if (points.size() < 2) return;

if (is\_fill && points.size() > 2) {

setfillcolor(fillcolor);

solidpolygon(&points[0], points.size());

for (size\_t i = 0; i < points.size() - 1; ++i) {

setlinecolor(color);

setlinestyle(lineStyle, lineWidth);

line(points[i].x, points[i].y, points[i + 1].x, points[i + 1].y);

}

}

else {

for (size\_t i = 0; i < points.size() - 1; ++i) {

setlinecolor(color);

setlinestyle(lineStyle, lineWidth);

line(points[i].x, points[i].y, points[i + 1].x, points[i + 1].y);

}

}

setlinecolor(WHITE);

setlinestyle(PS\_SOLID, 1);

}

std::string generateLabel() const {

std::wstringstream ss;

ss << L"Zhexian";

ss << L" 点数: " << points.size();

ss << L" 点集 {";

for (const auto& pt : points) {

ss << L"#(" << pt.x << L"," << pt.y << L")";

}

ss << L"}";

ss << L" 线条颜色: " << color;

ss << L" 填充状态: " << (int)is\_fill;

ss << L" 填充颜色: " << fillcolor;

ss << L" 线宽: " << lineWidth << L" 线型: " << lineStyle;

return wstring2string(ss.str());

}

RECT getBoundingBox() const override {

if (points.empty()) return { 0, 0, 0, 0 };

int minX = points[0].x, maxX = points[0].x, minY = points[0].y, maxY = points[0].y;

for (const auto& pt : points) {

if (pt.x < minX) minX = pt.x;

if (pt.x > maxX) maxX = pt.x;

if (pt.y < minY) minY = pt.y;

if (pt.y > maxY) maxY = pt.y;

}

return { minX, minY, maxX, maxY };

}

void move(int dx, int dy) override {

for (auto& pt : points) {

pt.x += dx;

pt.y += dy;

}

}

void zoom(double factor, POINT zoomCenter) override {

for (auto& pt : points) {

int dx = pt.x - zoomCenter.x;

int dy = pt.y - zoomCenter.y;

pt.x = static\_cast<int>(zoomCenter.x + dx \* factor);

pt.y = static\_cast<int>(zoomCenter.y + dy \* factor);

}

}

std::shared\_ptr<Shape> clone() const override {

return std::make\_shared<Zhexian>(\*this);

}

std::wstring getInfo() const override {

std::wstringstream ss;

ss << L"折线:\n Points=\n";

//for (int i = 0; i < sizeof(points) / sizeof(points[0]); i++)

// {

// if (i > 0 && (points[i].x == points[i - 1].x) && (points[i].y == points[i - 1].y)) continue;//避免重复

// ss << L"(" << points[i].x << L"," << points[i].y << L")\n";

//}

for (const auto& pt : points) {

ss << L"(" << pt.x << L"," << pt.y << L")";

}

ss << L"\n线条颜色=" << colorToString(color);

if (is\_fill) {

ss << L"\n填充颜色=" << colorToString(fillcolor);

}

return ss.str();

}

void changePoints(int x, int y) {

int index = 0, dis = 1000000;

for (int i = 0; i < points.size(); i++) {

POINT& it = points[i];

if ((x - it.x) \* (x - it.x) + (y - it.y) \* (y - it.y) < dis) {

dis = (x - it.x) \* (x - it.x) + (y - it.y) \* (y - it.y);

index = i;

}

/\*if ((x - it.x) \* (x - it.x) + (y - it.y) \* (y - it.y) <= 1000) {

it.x = x;

it.y = y;

break;

}\*/

}

points[index].x = x;

points[index].y = y;

}

private:

std::vector<POINT> points;

};

class Tuoyuan : public Shape {

public:

Tuoyuan(POINT topLeft = { 0,0 }, POINT bottomRight = { 0,0 }) : topLeft(topLeft), bottomRight(bottomRight) {}

void draw() const override {

setlinecolor(color);

setlinestyle(lineStyle, lineWidth);

if (is\_fill) {

setfillcolor(fillcolor);

solidellipse(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y);

ellipse(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y);

}

else {

ellipse(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y);

}

setlinecolor(WHITE);

setlinestyle(PS\_SOLID, 1);

}

void parseInfoFromStream(std::stringstream& ss) {

std::string token;

ss >> token; // 忽略"Tuoyuan"

ss >> token; // 左上点: (862,555)

//token.erase(0, wcslen(L"左上点: ("));

token.pop\_back(); // 移除')'

token.erase(0, wcslen(L"(")); // 移除'('

std::stringstream coordStream(token);

coordStream >> topLeft.x;

coordStream.ignore(1, L',');

coordStream >> topLeft.y;

ss >> ws >> token; // 右下点: (1340,818)

token.erase(0, wcslen(L"("));

token.pop\_back(); // 移除')'

coordStream.str(token);

coordStream.clear();

coordStream >> bottomRight.x;

coordStream.ignore(1, L',');

coordStream >> bottomRight.y;

ss >> ws >> token; // 线条颜色: 21930

//token.erase(0, wcslen(L"线条颜色: "));

color = std::stoi(token);

ss >> ws >> token; // 填充状态: 0

//token.erase(0, wcslen(L"填充状态: "));

is\_fill = (token == "1");

ss >> ws >> token; // 填充颜色: 21930

//token.erase(0, wcslen(L"填充颜色: "));

fillcolor = std::stoi(token);

ss >> ws >> token; // 线宽: 1

//token.erase(0, wcslen(L"线宽: "));

lineWidth = std::stoi(token);

ss >> ws >> token; // 线型: 0

//token.erase(0, wcslen(L"线型: "));

lineStyle = std::stoi(token);

}

std::string generateLabel() const {

std::wstringstream ss;

ss << L"Tuoyuan";

ss << L" 左上点: (" << topLeft.x << L"," << topLeft.y << L")";

ss << L" 右下点: (" << bottomRight.x << L"," << bottomRight.y << L")";

ss << L" 线条颜色: " << color;

ss << L" 填充状态: " << (int)is\_fill;

ss << L" 填充颜色: " << fillcolor;

ss << L" 线宽: " << lineWidth << L" 线型: " << lineStyle;

return wstring2string(ss.str());

}

RECT getBoundingBox() const override {

return { topLeft.x, topLeft.y, bottomRight.x, bottomRight.y };

}

void move(int dx, int dy) override {

topLeft.x += dx;

topLeft.y += dy;

bottomRight.x += dx;

bottomRight.y += dy;

}

void zoom(double factor, POINT zoomCenter) override {

int dx1 = topLeft.x - zoomCenter.x;

int dy1 = topLeft.y - zoomCenter.y;

int dx2 = bottomRight.x - zoomCenter.x;

int dy2 = bottomRight.y - zoomCenter.y;

topLeft.x = static\_cast<int>(zoomCenter.x + dx1 \* factor);

topLeft.y = static\_cast<int>(zoomCenter.y + dy1 \* factor);

bottomRight.x = static\_cast<int>(zoomCenter.x + dx2 \* factor);

bottomRight.y = static\_cast<int>(zoomCenter.y + dy2 \* factor);

}

std::shared\_ptr<Shape> clone() const override {

return std::make\_shared<Tuoyuan>(\*this);

}

std::wstring getInfo() const override {

std::wstringstream ss;

ss << L"椭圆: \n左上点=(" << topLeft.x << L"," << topLeft.y << L")\n右下点=(" << bottomRight.x << L"," << bottomRight.y;

ss << L"\n线条颜色=" << colorToString(color);

if (is\_fill) {

ss << L"\n填充颜色=" << colorToString(fillcolor);

}

return ss.str();

}

void changePoints(int x, int y) {

{

bottomRight.x = x;

bottomRight.y = y;

}

}

private:

POINT topLeft;

POINT bottomRight;

};

// 派生类 Duo，表示多边形

class Duo : public Shape {

public:

void addPoint(POINT pt) {

points.push\_back(pt);

}

void updateLastPoint(POINT pt) {

if (!points.empty()) {

points.back() = pt;

}

}

void draw() const override {

if (points.size() < 3) return;

setlinecolor(color);

setlinestyle(lineStyle, lineWidth);

if (is\_fill) {

setfillcolor(fillcolor);

solidpolygon(&points[0], points.size());

polygon(&points[0], points.size());

}

else {

polygon(&points[0], points.size());

}

setlinecolor(WHITE);

setlinestyle(PS\_SOLID, 1);

}

void parseInfoFromStream(std::stringstream& ss) {

std::string token;

ss >> token; // 忽略"Duo"

ss >> token; // 点数: 4

//token.erase(0, wcslen(L"点数: "));

int numPoints = std::stoi(token);

ss >> ws >> token; // 点集 {#(216,569)#(170,756)#(574,808)#(450,621)}

//token.erase(0, wcslen(L"点集 {#"));

token.pop\_back(); // 移除'}'

token.erase(0, wcslen(L"{#")); // 移除'{#'

std::stringstream pointStream(token);

std::string pointToken;

while (std::getline(pointStream, pointToken, '#')) {

if (!pointToken.empty()) {

pointToken.erase(0, wcslen(L"("));

pointToken.pop\_back(); // 移除')'

std::stringstream pointCoordStream(pointToken);

POINT pt;

pointCoordStream >> pt.x;

pointCoordStream.ignore(1, L',');

pointCoordStream >> pt.y;

points.push\_back(pt);

}

}

ss >> ws >> token; // 线条颜色: 16733695

//token.erase(0, wcslen(L"线条颜色: "));

color = std::stoi(token);

ss >> ws >> token; // 填充状态: 1

//token.erase(0, wcslen(L"填充状态: "));

is\_fill = (token == "1");

ss >> ws >> token; // 填充颜色: 21930

//token.erase(0, wcslen(L"填充颜色: "));

fillcolor = std::stoi(token);

ss >> ws >> token; // 线宽: 7

//token.erase(0, wcslen(L"线宽: "));

lineWidth = std::stoi(token);

ss >> ws >> token; // 线型: 0

//token.erase(0, wcslen(L"线型: "));

lineStyle = std::stoi(token);

}

std::string generateLabel() const {

std::wstringstream ss;

ss << L"Duo";

ss << L" 点数: " << points.size();

ss << L" 点集 {";

for (const auto& pt : points) {

ss << L"#(" << pt.x << L"," << pt.y << L")";

}

ss << L"}";

ss << L" 线条颜色: " << color;

ss << L" 填充状态: " << (int)is\_fill;

ss << L" 填充颜色: " << fillcolor;

ss << L" 线宽: " << lineWidth << L" 线型: " << lineStyle;

return wstring2string(ss.str());

}

RECT getBoundingBox() const override {

if (points.empty()) return { 0, 0, 0, 0 };

int minX = points[0].x, maxX = points[0].x, minY = points[0].y, maxY = points[0].y;

for (const auto& pt : points) {

if (pt.x < minX) minX = pt.x;

if (pt.x > maxX) maxX = pt.x;

if (pt.y < minY) minY = pt.y;

if (pt.y > maxY) maxY = pt.y;

}

return { minX, minY, maxX, maxY };

}

void move(int dx, int dy) override {

for (auto& pt : points) {

pt.x += dx;

pt.y += dy;

}

}

void zoom(double factor, POINT zoomCenter) override {

for (auto& pt : points) {

int dx = pt.x - zoomCenter.x;

int dy = pt.y - zoomCenter.y;

pt.x = static\_cast<int>(zoomCenter.x + dx \* factor);

pt.y = static\_cast<int>(zoomCenter.y + dy \* factor);

}

}

void changePoints(int x, int y) {

int index = 0, dis = 1000000;

for (int i = 0; i < points.size(); i++) {

POINT& it = points[i];

if ((x - it.x) \* (x - it.x) + (y - it.y) \* (y - it.y) < dis) {

dis = (x - it.x) \* (x - it.x) + (y - it.y) \* (y - it.y);

index = i;

}

/\*if ((x - it.x) \* (x - it.x) + (y - it.y) \* (y - it.y) <= 1000) {

it.x = x;

it.y = y;

break;

}\*/

}

points[index].x = x;

points[index].y = y;

}

std::shared\_ptr<Shape> clone() const override {

return std::make\_shared<Duo>(\*this);

}

std::wstring getInfo() const override {

std::wstringstream ss;

ss << L"多边形: \nPoints=\n";

for (const auto& pt : points) {

ss << L"(" << pt.x << L"," << pt.y << L")";

}

ss << L"\n线条颜色=" << colorToString(color);

if (is\_fill) {

ss << L"\n填充颜色=" << colorToString(fillcolor);

}

return ss.str();

}

private:

std::vector<POINT> points;

};

// 全局变量

std::vector<std::shared\_ptr<Shape>> shapes;

std::shared\_ptr<Zhexian> currentZhexian;

std::shared\_ptr<Duo> currentDuo;

bool isDrawingCircle = false;

bool isDrawingRect = false;

bool isDrawingZhexian = false;

bool isDrawingDuo = false;

bool isDrawingTuoyuan = false;

bool isRDraging = false;

POINT startPoint;

POINT endPoint;

bool isDragging = false;

POINT lastMousePos;

HintManager hintManager;

// 绘制所有的图形

void DrawAllShapes() {

BeginBatchDraw();

cleardevice();

if (OPEN\_TIPS) hintManager.displayHints();

drawButton();

// 绘制所有形状

for (const auto& shape : shapes) {

shape->draw();

}

// 绘制当前绘制中的形状

if (isDrawingCircle) {

int radius = static\_cast<int>(std::sqrt(std::pow(endPoint.x - startPoint.x, 2) + std::pow(endPoint.y - startPoint.y, 2)));

Circle tempCircle(startPoint, radius);

tempCircle.draw();

}

else if (isDrawingRect) {

Rect tempRect(startPoint, endPoint);

tempRect.draw();

}

else if (isDrawingZhexian && currentZhexian) {

currentZhexian->draw();

}

else if (isDrawingDuo && currentDuo) {

currentDuo->draw();

}

else if (isDrawingTuoyuan) {

Tuoyuan tempTuoyuan(startPoint, endPoint);

tempTuoyuan.draw();

}

if (selectedIndex != -1) {

RECT bbox = shapes[selectedIndex]->getBoundingBox();

THEME == 0 ? setlinecolor(YELLOW) : setlinecolor(DARKGRAY);

rectangle(bbox.left - 5, bbox.top - 5, bbox.right + 5, bbox.bottom + 5);

setlinecolor(WHITE);

// 显示信息

std::wstring info = shapes[selectedIndex]->getInfo();

settextstyle(18, 0, \_T("Arial")); // 设置字体样式

THEME==0 ? settextcolor(YELLOW) : settextcolor(DARKGRAY);

//outtextxy(bbox.right - 200, bbox.bottom + 10, info.c\_str()); // 在外框右下角显示信息

int y = bbox.bottom + 10; // 初始y坐标

for (size\_t pos = 0; pos < info.length(); ) {

size\_t next\_pos = info.find(L'\n', pos);

if (next\_pos == std::wstring::npos) {

next\_pos = info.length();

}

outtextxy(bbox.left, y, info.substr(pos, next\_pos - pos).c\_str());

y += 20; // 调整y坐标以适应下一行

pos = next\_pos + 1;

}

copyButton.drawcopyButton(bbox.right - 40, bbox.bottom + 10, bbox.right, bbox.bottom + 40);

for (const auto& shape : shapes) {

// 如果图形被选中

if (selectedIndex != -1 && shape.get() == shapes[selectedIndex].get()) {

// 计算层级信息的字符串

std::wstring layerInfo = L"层级: " + std::to\_wstring(selectedIndex);

std::wstring lineInfo = L"线宽: " + std::to\_wstring(shape->lineWidth) + L" 线型: " + std::to\_wstring(shape->lineStyle);

// 设置字体和颜色

settextstyle(18, 0, \_T("Arial")); // 或者选择合适的字体大小和样式

THEME == 0 ? settextcolor(YELLOW) : settextcolor(DARKGRAY);

// 计算位置

int posX = bbox.left; // 距离图形右边界5个单位

int posY = bbox.top - 30; // 距离图形顶部5个单位

// 输出层级信息

outtextxy(posX, posY, layerInfo.c\_str());

posX = bbox.right - 100; // 距离图形右边界5个单位

posY = bbox.top - 30; // 距离图形顶部5个单位

outtextxy(posX, posY, lineInfo.c\_str());

}

}

}

// 绘制选中的形状的外框

EndBatchDraw();

}

// 定义一个函数来更新按钮组状态

void pressButtom(Button\* targetButton) {

for (auto button : buttons) {

if (button != targetButton) {

button->release(); // 如果不是目标按钮，释放它

}

else {

button->press(); // 如果是目标按钮，按下去

}

}

}

void pressColourButtom(Button\* targetButton) {

for (auto button : colourbuttons) {

if (button != targetButton) {

button->release(); // 如果不是目标按钮，释放它

}

else {

button->press(); // 如果是目标按钮，按下去

if (selectedIndex != -1) {

shapes[selectedIndex]->setColor(COLOR);

DrawAllShapes();

}

}

}

}

std::string convertTCharToString(TCHAR\* tcharStr)

{

std::string result;

// 如果项目使用Unicode，需要从宽字符字符串转换

std::wstring\_convert<std::codecvt\_utf8<wchar\_t>> myconv;

result = myconv.to\_bytes(tcharStr);

return result;

}

void TcharToChar(const TCHAR\* tchar, char\* \_char) {

int iLength;

iLength = WideCharToMultiByte(CP\_ACP, 0, tchar, -1, NULL, 0, NULL, NULL);

WideCharToMultiByte(CP\_ACP, 0, tchar, -1, \_char, iLength, NULL, NULL);

}

bool saveProject(const std::vector<std::shared\_ptr<Shape>>& shapes) {

OPENFILENAME ofn;

TCHAR szFileName[MAX\_PATH] = { 0 };

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = NULL;

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT | OFN\_HIDEREADONLY | OFN\_PATHMUSTEXIST | OFN\_EXPLORER | OFN\_ENABLESIZING;

ofn.lpstrFilter = \_T("Text文件 (\*.txt)\0\*.txt\0All Files (\*.\*)\0\*.\*\0");

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = \_T(".");

if (GetSaveFileName(&ofn)) {

std::ofstream file(ofn.lpstrFile);

if (!file) {

return false;

}

for (const auto& shape : shapes) {

file << (shape->generateLabel()) << "\n";

}

return true;

file.close();

}

return false;

}

int loadProject(const std::string& filePath, std::vector<std::shared\_ptr<Shape>>& shapes,bool needClear = true) {

int count = 0;

std::fstream file(filePath);

if (!file) {

std::wcerr << L"文件打开失败！" << std::endl;

return -1;

}

if(needClear) shapes.clear();

selectedIndex = -1;

std::string line;

std::map<std::string, std::function<std::shared\_ptr<Shape>()>> shapeCreators = {

{ "Circle", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Circle>(); } },

{ "Rect", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Rect>(); } },

{ "Zhexian", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Zhexian>(); } },

{ "Duo", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Duo>(); } },

{ "Tuoyuan", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Tuoyuan>(); } },

{ "Image", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Image>(); } }

};

while (std::getline(file, line)) {

std::stringstream ss(line);

std::string shapeType;

ss >> shapeType;

if (shapeCreators.find(shapeType) != shapeCreators.end()) {

std::shared\_ptr<Shape> shape = shapeCreators[shapeType]();

shape->parseInfoFromStream(ss);

shapes.push\_back(shape);

count++;

}

}

return count;

}

int main() {

// 初始化图形窗口

OPEN\_DOUBLE\_CLICK ? initgraph(1600, 900, EX\_SHOWCONSOLE | EX\_DBLCLKS) : initgraph(1600, 900, EX\_SHOWCONSOLE);

setbkmode(TRANSPARENT);

cleardevice();

drawButton();

if (OPEN\_TIPS) { hintManager.updateHints({ 0, 0 }, "绘圆", "左键确定圆心，拖拽确定半径"); hintManager.displayHints(); }

drawCircleButton.press();

WhiteButton.press();

while (true) {

// 获取鼠标消息

MOUSEMSG msg = GetMouseMsg();

drawButton();

POINT pt = { msg.x, msg.y };

if (THEME == 1)

{

setbkcolor(WHITE);

}

else

{

setbkcolor(BLACK);

}

if (OPEN\_TIPS) hintManager.updatePt(pt);

if (!canBeSelected) { selectedIndex = -1; }

if (OPEN\_DOUBLE\_CLICK && msg.uMsg == WM\_LBUTTONDBLCLK)

{

if (OPEN\_TIPS) hintManager.updateHints(pt, "选择图形", "左键点击选择图形，拖动移动，滚轮选择图形，右键修改图形参数");

pressButtom(&selectShapeButton);

for (size\_t i = 0; i < shapes.size(); ++i) {

RECT bbox = shapes[i]->getBoundingBox();

if (pt.x >= bbox.left && pt.x <= bbox.right &&

pt.y >= bbox.top && pt.y <= bbox.bottom) {

selectedIndex = static\_cast<int>(i);

break;

}

}

isDragging = true;

lastMousePos = pt;

DrawAllShapes();

}

switch (msg.uMsg) {

case WM\_LBUTTONDOWN:

if (insertImageButton.isInside(msg.x, msg.y)) {

// 打开文件对话框选择图片

OPENFILENAME ofn;

TCHAR szFile[260] = { 0 };

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = GetHWnd();

ofn.lpstrFile = szFile;

ofn.nMaxFile = sizeof(szFile);

ofn.lpstrFilter = \_T("图像文件(\*.bmp;\*.jpg;\*.png)\0\*.bmp;\*.jpg;\*.png\0");

ofn.nFilterIndex = 1;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

if (GetOpenFileName(&ofn)) {

// 成功选择了图片文件

// 创建Image对象并加载图片

std::string filePath = convertTCharToString(szFile);

std::shared\_ptr<Image> image = std::make\_shared<Image>();

if (image->loadImage(filePath)) {

// 图片加载成功，将其添加到shapes向量中

shapes.push\_back(image);

DrawAllShapes();

}

else {

// 显示错误消息

HWND hnd = GetHWnd();

MessageBox(hnd, \_T("无法加载图片"), \_T("错误"), MB\_OK);

}

}

}

else if (saveButton.isInside(msg.x, msg.y)) {

if (saveProject(shapes)) {

MessageBox(GetHWnd(), \_T("保存成功"), \_T("提示"), MB\_OK);

}

else {

MessageBox(GetHWnd(), \_T("保存失败"), \_T("错误"), MB\_OK);

}

}

else if (loadButton.isInside(msg.x, msg.y)) {

OPENFILENAME ofn;

TCHAR szFileName[MAX\_PATH] = { 0 };

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = NULL;

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST | OFN\_EXPLORER;

ofn.lpstrFilter = \_T("Text文件 (\*.txt)\0\*.txt\0All Files (\*.\*)\0\*.\*\0");

ofn.nFilterIndex = 1;

ofn.lpstrInitialDir = \_T(".");

if (GetOpenFileName(&ofn)) {

std::string filePath = convertTCharToString(szFileName);

loadProject(filePath, shapes);

DrawAllShapes();

}

}

else if (insertButton.isInside(msg.x, msg.y)) {

HANDLE hConsole = GetStdHandle(STD\_INPUT\_HANDLE);

AttachConsole(ATTACH\_PARENT\_PROCESS);

//SetConsoleActiveProcess(GetCurrentProcessId());

// 接收多行输入

std::cout << "输入对象标签:" << std::endl;

std::string input;

std::map<std::string, std::function<std::shared\_ptr<Shape>()>> shapeCreators = {

{ "Circle", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Circle>(); } },

{ "Rect", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Rect>(); } },

{ "Zhexian", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Zhexian>(); } },

{ "Duo", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Duo>(); } },

{ "Tuoyuan", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Tuoyuan>(); } },

{ "Image", []() -> std::shared\_ptr<Shape> { return std::make\_shared<Image>(); } }

};

int count = 0;

while (std::getline(std::cin, input)&&input!="end") {

std::stringstream ss(input);

std::string shapeType;

ss >> shapeType;

if (shapeCreators.find(shapeType) != shapeCreators.end()) {

std::shared\_ptr<Shape> shape = shapeCreators[shapeType]();

shape->parseInfoFromStream(ss);

shapes.push\_back(shape);

count++;

}

}

std::cout << "成功置入" << count << "个对象" << std::endl;

MessageBox(GetHWnd(), stringToLPCWSTR(("成功置入"+std::to\_string(count)+"个对象")), \_T("提示"), MB\_OK);

FreeConsole();

// 重新绘制所有形状

DrawAllShapes();

}

else if (modifyLineWidthButton.isInside(msg.x, msg.y))

{

pressButtom(&modifyLineWidthButton);

if (OPEN\_TIPS) hintManager.updateHints(pt, "修改线宽", "滚轮上下滚动增减线宽");

continue;

}

else if (layereditButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "图层编辑", "滚轮上下滚动改变层级");

pressButtom(&layereditButton);

continue;

}

else if (theme.isInside(msg.x, msg.y))

{

changeTheme();

DrawAllShapes();

continue;

}

else if (copyButton.isInside(msg.x, msg.y)) {

std::string label = shapes[selectedIndex]->generateLabel();

CopyTextToClipboard(label);

MessageBox(GetHWnd(), \_T("已将标签拷贝到剪贴板"), \_T("提示"), MB\_OK);

DrawAllShapes();

}

else if (BlackButton.isInside(msg.x, msg.y)) {

COLOR = BLACK;

pressColourButtom(&BlackButton);

continue;

}

else if (BlueButton.isInside(msg.x, msg.y)) {

COLOR = BLUE;

pressColourButtom(&BlueButton);

continue;

}

else if (GreenButton.isInside(msg.x, msg.y)) {

COLOR = GREEN;

pressColourButtom(&GreenButton);

continue;

}

else if (CyanButton.isInside(msg.x, msg.y)) {

COLOR = CYAN;

pressColourButtom(&CyanButton);

continue;

}

else if (RedButton.isInside(msg.x, msg.y)) {

COLOR = RED;

pressColourButtom(&RedButton);

continue;

}

else if (MagentaButton.isInside(msg.x, msg.y)) {

COLOR = MAGENTA;

pressColourButtom(&MagentaButton);

continue;

}

else if (BrownButton.isInside(msg.x, msg.y)) {

COLOR = BROWN;

pressColourButtom(&BrownButton);

continue;

}

else if (LightGrayButton.isInside(msg.x, msg.y)) {

COLOR = LIGHTGRAY;

pressColourButtom(&LightGrayButton);

continue;

}

else if (DarkGrayButton.isInside(msg.x, msg.y)) {

COLOR = DARKGRAY;

pressColourButtom(&DarkGrayButton);

continue;

}

else if (LightBlueButton.isInside(msg.x, msg.y)) {

COLOR = LIGHTBLUE;

pressColourButtom(&LightBlueButton);

continue;

}

else if (LightGreenButton.isInside(msg.x, msg.y)) {

COLOR = LIGHTGREEN;

pressColourButtom(&LightGreenButton);

continue;

}

else if (LightCyanButton.isInside(msg.x, msg.y)) {

COLOR = LIGHTCYAN;

pressColourButtom(&LightCyanButton);

continue;

}

else if (LightRedButton.isInside(msg.x, msg.y)) {

COLOR = LIGHTRED;

pressColourButtom(&LightRedButton);

continue;

}

else if (LightMagentaButton.isInside(msg.x, msg.y)) {

COLOR = LIGHTMAGENTA;

pressColourButtom(&LightMagentaButton);

continue;

}

else if (YellowButton.isInside(msg.x, msg.y)) {

COLOR = YELLOW;

pressColourButtom(&YellowButton);

continue;

}

else if (WhiteButton.isInside(msg.x, msg.y)) {

COLOR = WHITE;

pressColourButtom(&WhiteButton);

continue;

}

else if (drawCircleButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "绘制圆", "左键点击确定圆心，点击左键确定半径");

pressButtom(&drawCircleButton);

continue;

}

else if (drawRectButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "绘制矩形", "拖拽画矩形");

pressButtom(&drawRectButton);

continue;

}

else if (drawZhexianButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "绘制折线", "左键点击确定第一个点，点击左键确定下一个点，右键结束");

pressButtom(&drawZhexianButton);

continue;

}

else if (drawDuoButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "绘制多边形", "左键点击确定第一个点，点击左键确定下一个点，右键结束");

pressButtom(&drawDuoButton);

continue;

}

else if (selectShapeButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "选择图形", "左键点击选择图形，拖动移动，滚轮选择图形，右键修改图形参数");

pressButtom(&selectShapeButton);

continue;

}

else if (drawTuoYuanButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "绘制椭圆", "拖拽画椭圆");

pressButtom(&drawTuoYuanButton);

continue;

}

else if (zoomButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "缩放", "滚轮上下滚动缩放图形");

pressButtom(&zoomButton);

continue;

}

else if (duplicateButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "复制", "复制选中的图形");

if (selectedIndex == -1) {

HWND hnd = GetHWnd();

MessageBox(hnd, \_T("必须先选择一个图形"), \_T("提示"), MB\_OK);

continue;

}

if (selectedIndex != -1) {

// 复制选定的形状

std::shared\_ptr<Shape> copiedShape = shapes[selectedIndex]->clone(); // 假设Shape类有clone方法

// 将复制的形状移动到窗口左上角

copiedShape->move(10, 10); // 或者根据需要设置具体位置

// 添加复制的形状到shapes向量

shapes.push\_back(copiedShape);

// 更新selectedIndex以选中新复制的形状

selectedIndex = shapes.size() - 1;

// 重新绘制所有形状

DrawAllShapes();

}

}

else if (deleteButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "删除", "删除选中的图形");

if (selectedIndex == -1) {

HWND hnd = GetHWnd();

MessageBox(hnd, \_T("必须先选择一个图形"), \_T("提示"), MB\_OK);

continue;

}

if (selectedIndex != -1) {

// 删除选中的形状

shapes.erase(shapes.begin() + selectedIndex);

// 清除selectedIndex

selectedIndex--;

// 重新绘制所有形状

DrawAllShapes();

}

}

else if (fillButton.isInside(msg.x, msg.y)) {

if (OPEN\_TIPS) hintManager.updateHints(pt, "填充", "填充选中的图形");

if (selectedIndex != -1) {

shapes[selectedIndex]->setFill(!shapes[selectedIndex]->is\_fill);

shapes[selectedIndex]->setFillColour(COLOR);

DrawAllShapes();

}

else {

HWND hnd = GetHWnd();

MessageBox(hnd, \_T("必须先选择一个图形"), \_T("提示"), MB\_OK);

continue;

}

}

else if (changeLineStyleButton.isInside(msg.x, msg.y)) {

if (selectedIndex != -1) {

shapes[selectedIndex]->changeLineStyle();

DrawAllShapes();

}

else {

HWND hnd = GetHWnd();

MessageBox(hnd, \_T("必须先选择一个图形"), \_T("提示"), MB\_OK);

continue;

}

}

else if (canBeSelected) {

for (size\_t i = 0; i < shapes.size(); ++i) {

RECT bbox = shapes[i]->getBoundingBox();

if (pt.x >= bbox.left && pt.x <= bbox.right &&

pt.y >= bbox.top && pt.y <= bbox.bottom) {

selectedIndex = static\_cast<int>(i);

break;

}

}

isDragging = true;

lastMousePos = pt;

DrawAllShapes();

}

else if (drawCircleMode) {

// 左键按下，开始绘制圆

isDrawingCircle = true;

startPoint = pt;

endPoint = startPoint;

}

else if (drawRectMode) {

// 左键按下，开始绘制矩形

isDrawingRect = true;

startPoint = pt;

endPoint = startPoint;

}

else if (drawTuoyuanMode) {

startPoint = { msg.x, msg.y };

isDrawingTuoyuan = true;

}

else if (drawZhexianMode) {

// 左键按下，绘制折线

if (!isDrawingZhexian) {

isDrawingZhexian = true;

currentZhexian = std::make\_shared<Zhexian>();

currentZhexian->addPoint(pt);

}

else if (isDrawingZhexian && currentZhexian) {

//currentZhexian->addPoint(pt);

DrawAllShapes();

}

}

else if (drawDuoMode) {

// 左键按下，绘制多边形

if (!isDrawingDuo) {

isDrawingDuo = true;

currentDuo = std::make\_shared<Duo>();

currentDuo->addPoint(pt);

}

else if (isDrawingDuo && currentDuo) {

//currentDuo->addPoint(pt);

DrawAllShapes();

}

}

break;

case WM\_RBUTTONDOWN:

// 右键按下，结束折线绘制或多边形绘制

if (isDrawingZhexian && currentZhexian) {

shapes.push\_back(currentZhexian);

currentZhexian.reset();

isDrawingZhexian = false;

DrawAllShapes();

}

else if (isDrawingDuo && currentDuo) {

shapes.push\_back(currentDuo);

currentDuo.reset();

isDrawingDuo = false;

DrawAllShapes();

}

else if (canBeSelected) {

isRDraging = true;

}

break;

case WM\_MOUSEMOVE:

// 鼠标移动，更新终点并重绘

if (isDrawingCircle || isDrawingRect || isDrawingTuoyuan) {

endPoint = pt;

DrawAllShapes();

}

else if (isDrawingZhexian && currentZhexian) {

currentZhexian->updateLastPoint(pt);

DrawAllShapes();

}

else if (isDrawingDuo && currentDuo) {

currentDuo->updateLastPoint(pt);

DrawAllShapes();

}

else if (isDragging && selectedIndex != -1) {

// 移动选中的图形

int dx = pt.x - lastMousePos.x;

int dy = pt.y - lastMousePos.y;

shapes[selectedIndex]->move(dx, dy);

lastMousePos = pt;

DrawAllShapes();

}

else if (isRDraging && selectedIndex != -1) {

shapes[selectedIndex]->changePoints(msg.x, msg.y);

DrawAllShapes();

}

break;

case WM\_LBUTTONUP:

// 左键松开，保存图形并结束绘制

if (isDrawingCircle) {

isDrawingCircle = false;

endPoint = pt;

int radius = static\_cast<int>(std::sqrt(std::pow(endPoint.x - startPoint.x, 2) + std::pow(endPoint.y - startPoint.y, 2)));

shapes.push\_back(std::make\_shared<Circle>(startPoint, radius));

DrawAllShapes();

}

else if (isDrawingRect) {

isDrawingRect = false;

endPoint = pt;

shapes.push\_back(std::make\_shared<Rect>(startPoint, endPoint));

DrawAllShapes();

}

else if (drawZhexianMode) {

if (isDrawingZhexian && currentZhexian) {

currentZhexian->addPoint(pt);

DrawAllShapes();

}

}

else if (drawDuoMode) {

if (isDrawingDuo && currentDuo) {

currentDuo->addPoint(pt);

DrawAllShapes();

}

}

else if (isDrawingTuoyuan) {

shapes.push\_back(std::make\_shared<Tuoyuan>(startPoint, endPoint));

isDrawingTuoyuan = false;

}

else if (isDragging && selectedIndex != -1) {

// 停止拖动图形

isDragging = false;

}

break;

case WM\_KEYDOWN:

// 按下 ESC 键退出程序

if (msg.mkCtrl == 1) { // 检查是否按下了 Ctrl 键

if (isDrawingZhexian && currentZhexian) {

shapes.push\_back(currentZhexian);

}

if (isDrawingDuo && currentDuo) {

shapes.push\_back(currentDuo);

}

closegraph();

return 0;

}

break;

case WM\_MOUSEWHEEL:

if (selectMode) {

short temp = msg.wheel;

// 选择模式下，鼠标滚轮选择图形

if (temp == 120) {

if (!shapes.empty()) {

selectedIndex = (selectedIndex - 1 + shapes.size()) % shapes.size();

DrawAllShapes();

}

}

else {

if (!shapes.empty()) {

selectedIndex = (selectedIndex + 1) % shapes.size();

DrawAllShapes();

}

}

}

if (zoomMode && selectedIndex != -1) {

double factor = msg.wheel > 0 ? 1.1 : 0.9;

shapes[selectedIndex]->zoom(factor, pt);

DrawAllShapes();

}

if (layerEditMode && selectedIndex != -1) {

short temp = msg.wheel;

if (temp == 120) {

// 上移一层

if (selectedIndex > 0) {

std::swap(shapes[selectedIndex], shapes[selectedIndex - 1]);

selectedIndex--;

}

}

else {

// 下移一层

if (selectedIndex < shapes.size() - 1) {

std::swap(shapes[selectedIndex], shapes[selectedIndex + 1]);

selectedIndex++;

}

}

DrawAllShapes();

}

if (modifyLineWidthMode) {

if (selectedIndex == -1) {

HWND hnd = GetHWnd();

MessageBox(hnd, \_T("必须先选择一个图形"), \_T("提示"), MB\_OK);

}

else {

short temp = msg.wheel;

if (temp == 120) {

shapes[selectedIndex]->addLineWidth();

}

else {

shapes[selectedIndex]->reduceLineWidth();

}

DrawAllShapes();

}

}

break;

case WM\_MBUTTONDOWN:

// 中键按下，将选中的图形设为填充

if (selectMode && selectedIndex != -1) {

shapes[selectedIndex]->setFill(!shapes[selectedIndex]->is\_fill);

DrawAllShapes();

}

break;

case WM\_RBUTTONUP:

//右键弹起

if (isRDraging && selectedIndex != -1) {

isRDraging = false;

}

break;

}

}

closegraph();

return 0;

}