#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>

#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)

static int COLOR = WHITE;

int selectedIndex = -1;

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);

}

}

// 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 draw() const {

if (isPressed) {

setfillcolor(RED);

}

else {

setfillcolor(BLUE);

}

setlinecolor(YELLOW);

fillrectangle(left, top, right, bottom);

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 {

return x >= left && x <= right && y >= top && y <= bottom;

}

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 copyButton(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 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

};

Button\* colourbuttons[] = {

&BlackButton,

&RedButton,

&GreenButton,

&BlueButton,

&MagentaButton,

&BrownButton,

&LightGrayButton,

&DarkGrayButton,

&LightBlueButton,

&LightGreenButton,

&LightCyanButton,

&LightRedButton,

&LightMagentaButton,

&CyanButton,

&YellowButton,

&WhiteButton

};

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;

}

void drawButton() {

BeginBatchDraw();

drawCircleButton.draw();

drawRectButton.draw();

drawZhexianButton.draw();

drawDuoButton.draw();

drawTuoYuanButton.draw();

selectShapeButton.draw();

zoomButton.draw();

copyButton.draw();

deleteButton.draw();

fillButton.draw();

layerEditButton.draw();

insertImageButton.draw();

changeLineStyleButton.draw();

modifyLineWidthButton.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();

}

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;

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->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;

}

// 绘制图片

void draw() const override {

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

}

// 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, int radius) : 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);

}

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, POINT bottomRight) : 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);

}

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 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);

}

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, POINT bottomRight) : 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);

}

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);

}

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;

// 绘制所有的图形

void DrawAllShapes() {

BeginBatchDraw();

cleardevice();

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();

setlinecolor(YELLOW);

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

settextcolor(YELLOW);

//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;

}

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

settextcolor(YELLOW); // 或者选择合适的颜色

// 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;

}

int main() {

// 初始化图形窗口

initgraph(1600, 900);

setbkmode(TRANSPARENT);

cleardevice();

drawButton();

drawCircleButton.press();

WhiteButton.press();

while (true) {

// 获取鼠标消息

MOUSEMSG msg = GetMouseMsg();

drawButton();

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

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

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

{

pressButtom(&modifyLineWidthButton);

continue;

}

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

pressButtom(&layerEditButton);

continue;

}

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)) {

pressButtom(&drawCircleButton);

continue;

}

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

pressButtom(&drawRectButton);

continue;

}

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

pressButtom(&drawZhexianButton);

continue;

}

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

pressButtom(&drawDuoButton);

continue;

}

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

pressButtom(&selectShapeButton);

continue;

}

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

pressButtom(&drawTuoYuanButton);

continue;

}

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

pressButtom(&zoomButton);

continue;

}

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

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 (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 (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;

}