Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №2

на тему

РАСШИРЕННОЕ ИСПОЛЬЗОВАНИЕ ОКОННОГО ИНТЕРФЕЙСА WIN32 И GDI. ФОРМИРОВАНИЕ СЛОЖНЫХ ИЗОБРАЖЕНИЙ, СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ, ОБРАБОТКА РАЗЛИЧНЫХ СООБЩЕНИЙ, МЕХАНИЗМ ПЕРЕХВАТА СООБЩЕНИЙ(WINHOOK).

Студент: гр.153502 Матвеев Н.С.

Руководитель: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

1. Формулировка задачи 3

2. Теоретические сведения 4

3. Описание функций программы 5

Список использованных источников 7

Приложение А (обязательное) Исходный код программы 8

**1 ФОРМУЛИРОВКА ЗАДАЧИ**

В качестве задачи необходимо реализовать оконное приложение для работы с фигурами с использованием *GDI*+. Управление приложением должно осуществляться с помощью клавиш клавиатуры и элементов управления. В приложении должен использоваться механизм перехвата сообщений.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Использование оконного интерфейса *Win32* и *GDI* (*Graphics* *Device* *Interface*) позволяет разработчикам создавать графические приложения для операционных систем *Windows*. Оконный интерфейс *Win32* предоставляет инструменты для создания оконных приложений с графическим пользовательским интерфейсом. *GDI*, с другой стороны, предоставляет набор функций и методов для рисования и работы с графическими объектами.

Функция *FillRect* в *Win32* *API* предназначена для заполнения прямоугольной области в контексте устройства (*device* *context*) заданным цветом или кистью. Она позволяет быстро и просто устанавливать цвет фона для окон, элементов управления и других графических объектов. *FillRect* принимает параметры, включая дескриптор контекста устройства, прямоугольник, который нужно заполнить, и кисть или цвет, которыми будет произведено заполнение. Эта функция широко используется для управления цветами и областями рисования в графических приложениях под *Windows*, включая создание фона, заливку и другие операции рисования на экране.

**3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

В программе предусмотрены следующие функции:

* выбор фигуры и переключение режима с помощью элементов управления *Combo* *Box* (см. рисунок 1);
* изменение цвета фигур с помощью элементов управления Button (см. рисунок 2).

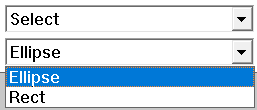


Рисунок 1 – Использование элементов управления *Combo* *Box*

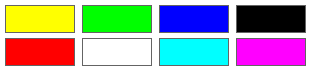


Рисунок 2 – Использование элементов управления *Button*

Главное окно программы состоит из двух дочерних окон:

* окно элементов управления (см. рисунок 3);
* окно сцены (см. рисунок 4).



Рисунок 3 – Дочернее окно для отображения элементов управления

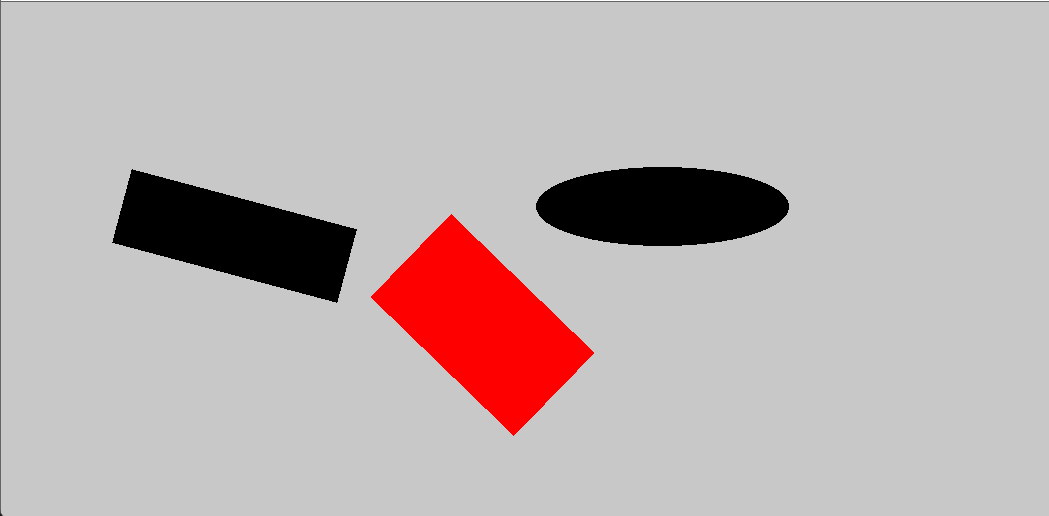


Рисунок 4 – Дочернее окно для отображения сцены

Для связи дочерних окон используется механизм перехвата сообщений.

Общий графический интерфейс программы представлен на рисунке 5.

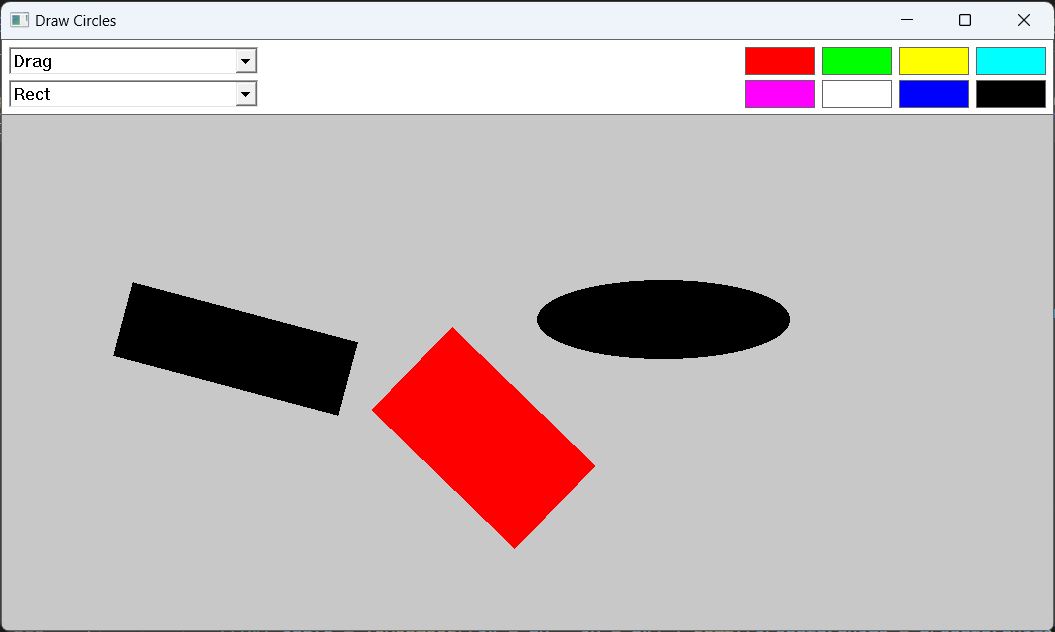


Рисунок 5 – Общий графический интерфейс программы

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Исходный код программы**

**Файл BaseFigure.h**

*#pragma once*

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include <gdiplus.h>*

*using namespace Gdiplus;*

*class BaseFigure*

*{*

*private:*

*static const Color DEFAULT\_BORDER\_COLOR;*

*public:*

*BaseFigure(Color color, Color borderColor = DEFAULT\_BORDER\_COLOR, Matrix\* matrix = new Matrix());*

*void Translate(PointF size);*

*void Rotate(FLOAT angle, PointF center);*

*void Scale(FLOAT size);*

*void RevertTransform();*

*void SaveTransform();*

*void SetColor(Color color) { this->color = color; }*

*void SetBorderColor(Color borderColor) { this->borderColor = borderColor; }*

*void SetMatrix(Matrix\* matrix) { delete this->matrix; this->matrix = matrix; }*

*Color GetColor() { return color; }*

*Color GetBorderColor() { return borderColor; }*

*Matrix\* GetMatrix() { return matrix->Clone(); }*

*virtual void Draw(Graphics\* graphics) = 0;*

*virtual PointF\* GetCenter() = 0;*

*virtual void PlaceIn(RectF rect) = 0;*

*virtual BOOL HitTest(PointF hitPoint) = 0;*

*protected:*

*Pen\* pen;*

*Color color;*

*Color borderColor;*

*Matrix\* matrix;*

*Matrix\* lastMatrix;*

*};*

**Файл BaseFigure.cpp**

*#include "BaseFigure.h"*

*const Color BaseFigure::DEFAULT\_BORDER\_COLOR = Color(0, 0, 0);*

*BaseFigure::BaseFigure(Color color, Color borderColor, Matrix\* matrix) :*

*color(color), borderColor(borderColor), matrix(matrix), lastMatrix(NULL), pen(new Pen(borderColor, 3.0F))*

*{*

*SaveTransform();*

*}*

*void BaseFigure::Translate(PointF size)*

*{*

*delete lastMatrix;*

*lastMatrix = matrix->Clone();*

*matrix->Translate(size.X, size.Y, MatrixOrderAppend);*

*}*

*void BaseFigure::Rotate(FLOAT angle, PointF center)*

*{*

*delete lastMatrix;*

*lastMatrix = matrix->Clone();*

*matrix->RotateAt((REAL)angle, center, MatrixOrderAppend);*

*}*

*void BaseFigure::Scale(FLOAT size)*

*{*

*delete lastMatrix;*

*lastMatrix = matrix->Clone();*

*PointF\* center = GetCenter();*

*matrix->Translate(-center->X, -center->Y, MatrixOrderAppend);*

*matrix->Scale(size, size, MatrixOrderAppend);*

*matrix->Translate(center->X, center->Y, MatrixOrderAppend);*

*}*

*void BaseFigure::RevertTransform()*

*{*

*delete matrix;*

*matrix = lastMatrix->Clone();*

*}*

*void BaseFigure::SaveTransform()*

*{*

*delete lastMatrix;*

*lastMatrix = matrix->Clone();*

*}*

**Файл BaseWindow.h**

*#pragma once*

*template <class DERIVED\_TYPE>*

*class BaseWindow*

*{*

*public:*

*const PCWSTR CLASS\_NAME;*

*static LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)*

*{*

*DERIVED\_TYPE \*pThis = NULL;*

*if (uMsg == WM\_NCCREATE)*

*{*

*CREATESTRUCT\* pCreate = (CREATESTRUCT\*)lParam;*

*pThis = (DERIVED\_TYPE\*)pCreate->lpCreateParams;*

*SetWindowLongPtr(hwnd, GWLP\_USERDATA, (LONG\_PTR)pThis);*

*pThis->m\_hwnd = hwnd;*

*}*

*else*

*{*

*pThis = (DERIVED\_TYPE\*)GetWindowLongPtr(hwnd, GWLP\_USERDATA);*

*}*

*if (pThis)*

*{*

*return pThis->HandleMessage(uMsg, wParam, lParam);*

*}*

*else*

*{*

*return DefWindowProc(hwnd, uMsg, wParam, lParam);*

*}*

*}*

*BaseWindow(PCWSTR CLASS\_NAME) : CLASS\_NAME(CLASS\_NAME), m\_hwnd(NULL) { }*

*BOOL Create(*

*PCWSTR lpWindowName,*

*DWORD dwStyle,*

*HWND hWndParent = 0,*

*DWORD dwExStyle = 0,*

*int x = CW\_USEDEFAULT,*

*int y = CW\_USEDEFAULT,*

*int nWidth = CW\_USEDEFAULT,*

*int nHeight = CW\_USEDEFAULT,*

*HMENU hMenu = 0*

*)*

*{*

*WNDCLASS wc = {};*

*wc.lpfnWndProc = DERIVED\_TYPE::WindowProc;*

*wc.hInstance = GetModuleHandle(NULL);*

*wc.lpszClassName = CLASS\_NAME;*

*RegisterClass(&wc);*

*m\_hwnd = CreateWindowEx(*

*dwExStyle, CLASS\_NAME, lpWindowName, dwStyle, x, y,*

*nWidth, nHeight, hWndParent, hMenu, GetModuleHandle(NULL), this*

*);*

*return (m\_hwnd ? TRUE : FALSE);*

*}*

*HWND Window() const { return m\_hwnd; }*

*protected:*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) = 0;*

*HWND m\_hwnd;*

*};*

**Файл DPIScale.h**

*#pragma once*

*#include <d2d1.h>*

*class DPIScale*

*{*

*static float scaleX;*

*static float scaleY;*

*public:*

*static void Initialize()*

*{*

*FLOAT dpi = GetDpiForSystem();*

*scaleX = dpi / 96.0f;*

*scaleY = dpi / 96.0f;*

*}*

*template <typename T>*

*static float PixelsToDipsX(T x)*

*{*

*return static\_cast<float>(x) / scaleX;*

*}*

*template <typename T>*

*static float PixelsToDipsY(T y)*

*{*

*return static\_cast<float>(y) / scaleY;*

*}*

*template <typename T>*

*static T DipXToPixels(float x)*

*{*

*return static\_cast<T>(x) \* scaleX;*

*}*

*template <typename T>*

*static T DipYToPixels(float y)*

*{*

*return static\_cast<T>(y) \* scaleY;*

*}*

*};*

**Файл EllipseFigure.h**

*#pragma once*

*#include "BaseFigure.h"*

*class EllipseFigure : public BaseFigure*

*{*

*private:*

*static const Color DEFAULT\_BORDER\_COLOR;*

*public:*

*EllipseFigure(RectF Rect, Color color, Color borderColor = DEFAULT\_BORDER\_COLOR, Matrix\* matrix = new Matrix());*

*RectF GetRect() { return rect; }*

*virtual void Draw(Graphics\* graphics) override;*

*virtual PointF\* GetCenter() override;*

*virtual void PlaceIn(RectF rect) override;*

*virtual BOOL HitTest(PointF hitPoint) override;*

*protected:*

*RectF rect;*

*};*

**Файл EllipseFigure.cpp**

*#include "EllipseFigure.h"*

*const Color EllipseFigure::DEFAULT\_BORDER\_COLOR = Color(0, 0, 0);*

*EllipseFigure::EllipseFigure(RectF rect, Color color, Color borderColor, Matrix\* matrix) :*

*BaseFigure(color, borderColor, matrix), rect(rect)*

*{*

*}*

*void EllipseFigure::Draw(Graphics\* graphics)*

*{*

*graphics->SetTransform(matrix);*

*pen->SetColor(borderColor);*

*graphics->DrawEllipse(pen, rect);*

*pen->SetColor(color);*

*graphics->FillEllipse(pen->GetBrush(), rect);*

*graphics->ResetTransform();*

*}*

*PointF\* EllipseFigure::GetCenter()*

*{*

*PointF\* result = new PointF((rect.GetRight() + rect.GetLeft()) / 2, (rect.GetTop() + rect.GetBottom()) / 2);*

*matrix->TransformPoints(result);*

*return result;*

*}*

*void EllipseFigure::PlaceIn(RectF rect)*

*{*

*matrix->Reset();*

*lastMatrix->Reset();*

*this->rect = rect;*

*}*

*BOOL EllipseFigure::HitTest(PointF hitPoint)*

*{*

*Matrix\* invertedMatrix = matrix->Clone();*

*invertedMatrix->Invert();*

*invertedMatrix->TransformPoints(&hitPoint);*

*const float a = rect.Width / 2;*

*const float b = rect.Height / 2;*

*const float x1 = hitPoint.X - (rect.GetLeft() + rect.GetRight()) / 2;*

*const float y1 = hitPoint.Y - (rect.GetTop() + rect.GetBottom()) / 2;*

*const float d = ((x1 \* x1) / (a \* a)) + ((y1 \* y1) / (b \* b));*

*return d <= 1.0f;*

*}*

**Файл GraphicsScene.h**

*#pragma once*

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include <memory>*

*#include <list>*

*#include "settings.h"*

*#include "BaseFigure.h"*

*#include "EllipseFigure.h"*

*#include "RectFigure.h"*

*#include "BaseWindow.h"*

*#include "DPIScale.h"*

*#include "resource.h"*

*class GraphicsScene : public BaseWindow<GraphicsScene>*

*{*

*private:*

*static const PCWSTR DEFAULT\_CLASS\_NAME;*

*static const Color DEFAULT\_BORDER\_COLOR;*

*static const Color DEFAULT\_SELECTION\_COLOR;*

*static const float DEFAULT\_FIGURE\_SIZE;*

*public:*

*GraphicsScene(Mode\* mode = NULL, Figure\* figure = NULL, Color\* color = NULL, PCWSTR CLASS\_NAME = DEFAULT\_CLASS\_NAME, Color borderColor = DEFAULT\_BORDER\_COLOR, Color selectionColor=DEFAULT\_SELECTION\_COLOR);*

*std::shared\_ptr<BaseFigure> Selection();*

*void ClearSelection();*

*BOOL Select(PointF hitPoint);*

*void InsertFigure(float dipX, float dipY);*

*void ColorChanged();*

*//void Resize();*

*void OnPaint();*

*void OnLButtonDown(int pixelX, int pixelY, DWORD flags);*

*void OnLButtonUp();*

*void OnMouseMove(int pixelX, int pixelY, DWORD flags);*

*void OnKeyDown(UINT vkey);*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) override;*

*protected:*

*Color borderColor;*

*Color selectionColor;*

*Mode\* mode;*

*Figure\* figure;*

*Color\* color;*

*PointF ptMouse;*

*std::list<std::shared\_ptr<BaseFigure>> figures;*

*std::list<std::shared\_ptr<BaseFigure>>::iterator selection;*

*bool tracking;*

*TRACKMOUSEEVENT trackingStruct;*

*};*

**Файл GraphicsScene.cpp**

*#include "GraphicsScene.h"*

*#include "helper\_functions.h"*

*const PCWSTR GraphicsScene::DEFAULT\_CLASS\_NAME = L"Graphics";*

*const Color GraphicsScene::DEFAULT\_BORDER\_COLOR = Color(0, 0, 0);*

*const Color GraphicsScene::DEFAULT\_SELECTION\_COLOR = Color(255, 0, 0);*

*const float GraphicsScene::DEFAULT\_FIGURE\_SIZE = 2.0F;*

*GraphicsScene::GraphicsScene(Mode\* mode, Figure\* figure, Color\* color, PCWSTR CLASS\_NAME, Color borderColor, Color selectionColor) :*

*BaseWindow<GraphicsScene>(CLASS\_NAME), mode(mode), figure(figure), color(color),*

*borderColor(borderColor), selectionColor(selectionColor), ptMouse(), selection(figures.end()),*

*tracking(false), trackingStruct{ sizeof(trackingStruct), NULL, NULL, NULL }*

*{*

*}*

*std::shared\_ptr<BaseFigure> GraphicsScene::Selection()*

*{*

*if (selection == figures.end())*

*{*

*return nullptr;*

*}*

*else*

*{*

*return (\*selection);*

*}*

*}*

*void GraphicsScene::ClearSelection()*

*{*

*if (Selection())*

*{*

*Selection()->SetBorderColor(DEFAULT\_BORDER\_COLOR);*

*}*

*selection = figures.end();*

*}*

*BOOL GraphicsScene::Select(PointF hitPoint)*

*{*

*ClearSelection();*

*for (auto i = figures.rbegin(); i != figures.rend(); ++i)*

*{*

*if ((\*i)->HitTest(hitPoint))*

*{*

*auto tmp = \*i;*

*\*i = figures.back();*

*figures.back() = tmp;*

*selection = --figures.end();*

*Selection()->SetBorderColor(DEFAULT\_SELECTION\_COLOR);*

*return TRUE;*

*}*

*}*

*return FALSE;*

*}*

*void GraphicsScene::InsertFigure(float dipX, float dipY)*

*{*

*ClearSelection();*

*switch (\*figure)*

*{*

*case Figure::Ellipse:*

*{*

*ptMouse = PointF(dipX, dipY);*

*RectF rect = RectF(dipX, dipY, DEFAULT\_FIGURE\_SIZE, DEFAULT\_FIGURE\_SIZE);*

*selection = figures.insert(*

*figures.end(),*

*std::shared\_ptr<BaseFigure>(new EllipseFigure(rect, \*color, DEFAULT\_SELECTION\_COLOR)));*

*break;*

*}*

*case Figure::Rect:*

*{*

*ptMouse = PointF(dipX, dipY);*

*RectF rect = RectF(dipX, dipY, DEFAULT\_FIGURE\_SIZE, DEFAULT\_FIGURE\_SIZE);*

*selection = figures.insert(*

*figures.end(),*

*std::shared\_ptr<BaseFigure>(new RectFigure(rect, \*color, DEFAULT\_SELECTION\_COLOR)));*

*}*

*}*

*}*

*void GraphicsScene::ColorChanged()*

*{*

*if ((\*mode == Mode::SelectMode) && Selection())*

*Selection()->SetColor(\*color);*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*

*/\*void GraphicsScene::Resize()*

*{*

*if (pRenderTarget != NULL)*

*{*

*RECT rc;*

*GetClientRect(m\_hwnd, &rc);*

*D2D1\_SIZE\_U size = D2D1::SizeU(rc.right, rc.bottom);*

*pRenderTarget->Resize(size);*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*

*}\*/*

*void GraphicsScene::OnPaint()*

*{*

*PAINTSTRUCT ps;*

*HDC hdc = BeginPaint(m\_hwnd, &ps);*

*POINT lpSize = { ps.rcPaint.right - ps.rcPaint.left, ps.rcPaint.bottom - ps.rcPaint.top };*

*POINT pSize = lpSize;*

*LPtoDP(hdc, &pSize, 1);*

*HDC hdcBits = CreateCompatibleDC(hdc);*

*HBITMAP bitMap = CreateCompatibleBitmap(hdc, pSize.x, pSize.y);*

*SelectObject(hdcBits, bitMap);*

*Graphics graphics(hdcBits);*

*graphics.Clear(Color(200, 200, 200));*

*for (auto i = figures.begin(); i != figures.end(); ++i)*

*{*

*(\*i)->Draw(&graphics);*

*}*

*BitBlt(hdc, ps.rcPaint.left, ps.rcPaint.top, lpSize.x, lpSize.y, hdcBits, 0, 0, SRCCOPY);*

*EndPaint(m\_hwnd, &ps);*

*DeleteDC(hdc);*

*DeleteDC(hdcBits);*

*DeleteObject(bitMap);*

*/\*HRESULT hr = CreateGraphicsResources();*

*if (SUCCEEDED(hr))*

*{*

*PAINTSTRUCT ps;*

*BeginPaint(m\_hwnd, &ps);*

*pRenderTarget->BeginDraw();*

*pRenderTarget->Clear(D2D1::ColorF(D2D1::ColorF::SkyBlue));*

*for (auto i = figures.begin(); i != figures.end(); ++i)*

*{*

*(\*i)->Draw(pRenderTarget, pBrush);*

*}*

*hr = pRenderTarget->EndDraw();*

*if (FAILED(hr) || hr == D2DERR\_RECREATE\_TARGET)*

*{*

*DiscardGraphicsResources();*

*}*

*EndPaint(m\_hwnd, &ps);*

*}\*/*

*}*

*void GraphicsScene::OnLButtonDown(int pixelX, int pixelY, DWORD flags)*

*{*

*const float dipX = pixelX;// DPIScale::PixelsToDipsX(pixelX);*

*const float dipY = pixelY;// DPIScale::PixelsToDipsY(pixelY);*

*POINT pt = { pixelX, pixelY };*

*ptMouse = { dipX, dipY };*

*if (DragDetect(m\_hwnd, pt))*

*{*

*SetCapture(m\_hwnd);*

*switch (\*mode)*

*{*

*case Mode::DrawMode:*

*InsertFigure(dipX, dipY);*

*break;*

*}*

*}*

*else*

*{*

*if (\*mode == Mode::SelectMode)*

*{*

*Select(ptMouse);*

*}*

*}*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*

*void GraphicsScene::OnLButtonUp()*

*{*

*if (Selection())*

*{*

*Selection()->SaveTransform();*

*}*

*ReleaseCapture();*

*}*

*void GraphicsScene::OnMouseMove(int pixelX, int pixelY, DWORD flags)*

*{*

*const float dipX = pixelX;//DPIScale::PixelsToDipsX(pixelX);*

*const float dipY = pixelY;//DPIScale::PixelsToDipsY(pixelY);*

*if ((flags & MK\_LBUTTON) && Selection())*

*{*

*switch (\*mode)*

*{*

*case Mode::DrawMode:*

*{*

*float left;*

*float width;*

*float top;*

*float height;*

*if (ptMouse.X > dipX)*

*{*

*left = dipX;*

*width = ptMouse.X - dipX;*

*}*

*else*

*{*

*left = ptMouse.X;*

*width = dipX - ptMouse.X;*

*}*

*if (ptMouse.Y > dipY)*

*{*

*top = dipY;*

*height = ptMouse.Y - dipY;*

*}*

*else*

*{*

*top = ptMouse.Y;*

*height = dipY - ptMouse.Y;*

*}*

*Selection()->PlaceIn(RectF(left, top, width, height));*

*break;*

*}*

*case Mode::DragMode:*

*{*

*Selection()->Translate({ dipX - ptMouse.X, dipY - ptMouse.Y });*

*ptMouse = { dipX, dipY };*

*break;*

*}*

*{*

*case Mode::ScaleMode:*

*Selection()->RevertTransform();*

*PointF\* center = Selection()->GetCenter();*

*PointF size = { abs((dipX - center->X) / (ptMouse.X - center->X)), abs((dipY - center->Y) / (ptMouse.Y - center->Y)) };*

*Selection()->Scale(size.X > size.Y ? size.X : size.Y);*

*delete center;*

*break;*

*}*

*case Mode::RotateMode:*

*{*

*PointF\* center = Selection()->GetCenter();*

*FLOAT ax = ptMouse.X - center->X;*

*FLOAT ay = ptMouse.Y - center->Y;*

*FLOAT bx = dipX - center->X;*

*FLOAT by = dipY - center->Y;*

*FLOAT aLengthSquare = ax \* ax + ay \* ay;*

*FLOAT bLengthSquare = bx \* bx + by \* by;*

*FLOAT angle = ToDegrees((ax \* by - ay \* bx) / sqrtf(aLengthSquare \* bLengthSquare));*

*Selection()->Rotate(angle, \*center);*

*ptMouse = { dipX, dipY };*

*delete center;*

*break;*

*}*

*}*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*

*}*

*void GraphicsScene::OnKeyDown(UINT vkey)*

*{*

*switch (vkey)*

*{*

*case VK\_DELETE:*

*if ((\*mode == Mode::SelectMode) && Selection())*

*{*

*figures.erase(selection);*

*selection = figures.end();*

*ClearSelection();*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*

*break;*

*}*

*}*

*LRESULT GraphicsScene::HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam)*

*{*

*switch (uMsg)*

*{*

*case WM\_CREATE:*

*if (!GetParent(m\_hwnd))*

*{*

*DPIScale::Initialize();*

*}*

*trackingStruct.hwndTrack = m\_hwnd;*

*return 0;*

*case WM\_DESTROY:*

*if (!GetParent(m\_hwnd))*

*{*

*PostQuitMessage(0);*

*}*

*return 0;*

*case WM\_PAINT:*

*OnPaint();*

*return 0;*

*case WM\_SIZE:*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*//Resize();*

*return 0;*

*case WM\_LBUTTONDOWN:*

*OnLButtonDown(GET\_X\_LPARAM(lParam), GET\_Y\_LPARAM(lParam), (DWORD)wParam);*

*return 0;*

*case WM\_LBUTTONUP:*

*OnLButtonUp();*

*return 0;*

*case WM\_MOUSEMOVE:*

*if (tracking)*

*{*

*OnMouseMove(GET\_X\_LPARAM(lParam), GET\_Y\_LPARAM(lParam), (DWORD)wParam);*

*}*

*else*

*{*

*trackingStruct.dwFlags = TME\_HOVER | TME\_LEAVE;*

*TrackMouseEvent(&trackingStruct);*

*tracking = true;*

*}*

*return 0;*

*case WM\_MOUSEHOVER:*

*SetFocus(m\_hwnd);*

*trackingStruct.dwFlags = TME\_LEAVE;*

*TrackMouseEvent(&trackingStruct);*

*return 0;*

*case WM\_MOUSELEAVE:*

*tracking = false;*

*return 0;*

*case WM\_KEYDOWN:*

*OnKeyDown((UINT)wParam);*

*return 0;*

*case WM\_COMMAND:*

*if (~GetKeyState(VK\_LBUTTON) & 0x8000)*

*{*

*HWND parentWND = GetParent(m\_hwnd);*

*if (!parentWND)*

*{*

*parentWND = m\_hwnd;*

*}*

*switch (LOWORD(wParam))*

*{*

*case ID\_DRAW\_MODE:*

*\*mode = Mode::DrawMode;*

*PostMessage(m\_hwnd, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*case ID\_SELECT\_MODE:*

*\*mode = Mode::SelectMode;*

*PostMessage(m\_hwnd, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*case ID\_DRAG\_MODE:*

*\*mode = Mode::DragMode;*

*PostMessage(m\_hwnd, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*case ID\_SCALE\_MODE:*

*\*mode = Mode::ScaleMode;*

*PostMessage(m\_hwnd, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*case ID\_ROTATE\_MODE:*

*\*mode = Mode::RotateMode;*

*PostMessage(m\_hwnd, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*case ID\_ELLIPSE:*

*if (\*mode == Mode::DrawMode)*

*{*

*\*figure = Figure::Ellipse;*

*PostMessage(m\_hwnd, WM\_FIGURE\_CHANGED, NULL, NULL);*

*}*

*return 0;*

*case ID\_RECT:*

*if (\*mode == Mode::DrawMode)*

*{*

*\*figure = Figure::Rect;*

*PostMessage(m\_hwnd, WM\_FIGURE\_CHANGED, NULL, NULL);*

*}*

*return 0;*

*}*

*}*

*break;*

*case WM\_MODE\_CHANGED:*

*return 0;*

*case WM\_FIGURE\_CHANGED:*

*return 0;*

*case WM\_COLOR\_CHANGED:*

*ColorChanged();*

*return 0;*

*}*

*return DefWindowProc(m\_hwnd, uMsg, wParam, lParam);*

*}*

**Файл helper\_functions.h**

*#pragma once*

*const double PI = 3.14;*

*template <class T> void SafeRelease(T\*\* ppT)*

*{*

*if (\*ppT)*

*{*

*(\*ppT)->Release();*

*\*ppT = NULL;*

*}*

*}*

*double ToDegrees(double rad);*

*Файл helper\_functions.cpp*

*#pragma once*

*const double PI = 3.14;*

*template <class T> void SafeRelease(T\*\* ppT)*

*{*

*if (\*ppT)*

*{*

*(\*ppT)->Release();*

*\*ppT = NULL;*

*}*

*}*

*double ToDegrees(double rad);*

**Файл input.rc**

*//<Snippetinput\_rc>*

*#include "resource.h"*

*IDR\_ACCEL1 ACCELERATORS*

*{*

*0x70, ID\_DRAW\_MODE, VIRTKEY // F1*

*0x71, ID\_SELECT\_MODE, VIRTKEY // F2*

*0x72, ID\_SCALE\_MODE, VIRTKEY*

*0X73, ID\_ROTATE\_MODE, VIRTKEY*

*0x74, ID\_DRAG\_MODE, VIRTKEY*

*}*

*IDR\_ACCEL2 ACCELERATORS*

*{*

*0x75, ID\_ELLIPSE, VIRTKEY*

*0x76, ID\_RECT, VIRTKEY*

*}*

*//</Snippetinput\_rc>*

**Файл main.cpp**

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include <list>*

*#include <memory>*

*using namespace std;*

*#pragma comment(lib, "d2d1")*

*#include "BaseWindow.h"*

*#include "EllipseFigure.h"*

*#include "resource.h"*

*#include "GraphicsScene.h"*

*#include "SceneControl.h"*

*#include "MainWindow.h"*

*int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE, PWSTR, int nCmdShow)*

*{*

*MainWindow win = MainWindow();*

*//Mode mode = Mode::SelectMode;*

*//Figure figure = Figure::Ellipse;*

*//D2D1\_COLOR\_F color = D2D1::ColorF(D2D1::ColorF::Black);*

*//SceneControl win = SceneControl(&mode, &figure, &color);*

*//GraphicsScene win = GraphicsScene(&mode, &figure, &color);*

*if (!win.Create(L"Draw Circles", WS\_OVERLAPPEDWINDOW))*

*{*

*return 0;*

*}*

*HACCEL hAccel1 = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR\_ACCEL1));*

*HACCEL hAccel2 = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR\_ACCEL2));*

*ShowWindow(win.Window(), nCmdShow);*

*MSG msg;*

*while (GetMessage(&msg, NULL, 0, 0))*

*{*

*if (!TranslateAccelerator(GetFocus(), hAccel1, &msg) && !TranslateAccelerator(GetFocus(), hAccel2, &msg))*

*{*

*TranslateMessage(&msg);*

*DispatchMessage(&msg);*

*}*

*}*

*return 0;*

*}*

**Файл MainWindow.h**

*#pragma once*

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include "settings.h"*

*#include "BaseWindow.h"*

*#include "DPIScale.h"*

*#include "resource.h"*

*#include "GraphicsScene.h"*

*#include "SceneControl.h"*

*class MainWindow : public BaseWindow<MainWindow>*

*{*

*private:*

*static const PCWSTR DEFAULT\_CLASS\_NAME;*

*static const Mode DEFAULT\_MODE;*

*static const Figure DEFAULT\_FIGURE;*

*static const D2D1\_COLOR\_F DEFAULT\_COLOR;*

*public:*

*MainWindow(Mode mode = DEFAULT\_MODE, Figure figure = DEFAULT\_FIGURE, D2D1\_COLOR\_F color = DEFAULT\_COLOR, PCWSTR CLASS\_NAME = DEFAULT\_CLASS\_NAME);*

*~MainWindow();*

*Mode\* GetMode() { return &mode; }*

*Figure\* GetFigure() { return &figure; }*

*D2D1\_COLOR\_F\* GetColor() { return &color; }*

*ID2D1Factory\* GetFactory() { return pFactory; }*

*HWND GetScene() { return graphicsScene->Window(); }*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) override;*

*protected:*

*void CreateLayout();*

*void SetLayout();*

*Mode mode;*

*Figure figure;*

*D2D1\_COLOR\_F color;*

*ID2D1Factory\* pFactory;*

*SceneControl\* sceneControl;*

*GraphicsScene\* graphicsScene;*

*};*

**Файл MainWindow.cpp**

*#include "MainWindow.h"*

*#include "helper\_functions.h"*

*const PCWSTR MainWindow::DEFAULT\_CLASS\_NAME = L"Graphics";*

*const Mode MainWindow::DEFAULT\_MODE = Mode::SelectMode;*

*const Figure MainWindow::DEFAULT\_FIGURE = Figure::Ellipse;*

*const D2D1\_COLOR\_F MainWindow::DEFAULT\_COLOR = D2D1::ColorF(D2D1::ColorF::Black);*

*MainWindow::MainWindow(Mode mode, Figure figure, D2D1\_COLOR\_F color, PCWSTR CLASS\_NAME) :*

*BaseWindow<MainWindow>(CLASS\_NAME), mode(mode), figure(figure), color(color), sceneControl(NULL), graphicsScene(NULL)*

*{*

*}*

*MainWindow::~MainWindow()*

*{*

*if (sceneControl)*

*delete sceneControl;*

*if (graphicsScene)*

*delete graphicsScene;*

*}*

*LRESULT MainWindow::HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam)*

*{*

*switch (uMsg)*

*{*

*case WM\_CREATE:*

*if (FAILED(D2D1CreateFactory(*

*D2D1\_FACTORY\_TYPE\_SINGLE\_THREADED, &pFactory)))*

*{*

*return -1; // Fail CreateWindowEx.*

*}*

*DPIScale::Initialize();*

*CreateLayout();*

*return 0;*

*case WM\_SIZE:*

*SetLayout();*

*return 0;*

*case WM\_DESTROY:*

*SafeRelease(&pFactory);*

*PostQuitMessage(0);*

*return 0;*

*case WM\_MODE\_CHANGED:*

*PostMessage(sceneControl->Window(), uMsg, wParam, lParam);*

*PostMessage(graphicsScene->Window(), uMsg, wParam, lParam);*

*return 0;*

*case WM\_FIGURE\_CHANGED:*

*PostMessage(sceneControl->Window(), uMsg, wParam, lParam);*

*PostMessage(graphicsScene->Window(), uMsg, wParam, lParam);*

*return 0;*

*case WM\_COLOR\_CHANGED:*

*PostMessage(sceneControl->Window(), uMsg, wParam, lParam);*

*PostMessage(graphicsScene->Window(), uMsg, wParam, lParam);*

*return 0;*

*}*

*return DefWindowProc(m\_hwnd, uMsg, wParam, lParam);*

*}*

*void MainWindow::CreateLayout()*

*{*

*sceneControl = new SceneControl(&mode, &figure, &color);*

*sceneControl->Create(L"Scene control", WS\_CHILD | WS\_BORDER | WS\_VISIBLE, m\_hwnd);*

*graphicsScene = new GraphicsScene(&mode, &figure, &color, pFactory);*

*graphicsScene->Create(L"Scene", WS\_CHILD | WS\_BORDER | WS\_VISIBLE, m\_hwnd);*

*}*

*void MainWindow::SetLayout()*

*{*

*RECT rcClient;*

*GetClientRect(m\_hwnd, &rcClient);*

*MoveWindow(sceneControl->Window(),*

*rcClient.left,*

*rcClient.top,*

*rcClient.right,*

*sceneControl->GetRealWindowHeight(),*

*FALSE);*

*RECT rcControl;*

*GetClientRect(sceneControl->Window(), &rcControl);*

*MoveWindow(graphicsScene->Window(),*

*rcClient.left,*

*rcControl.bottom,*

*rcClient.right,*

*rcClient.bottom - rcControl.bottom,*

*FALSE);*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*

**Файл RectFigure.h**

*#pragma once*

*#include "BaseFigure.h"*

*class RectFigure : public BaseFigure*

*{*

*private:*

*static const Color DEFAULT\_BORDER\_COLOR;*

*public:*

*RectFigure(RectF rect, Color color, Color borderColor = DEFAULT\_BORDER\_COLOR, Matrix\* matrix = new Matrix());*

*RectF GetRect() { return rect; }*

*virtual void Draw(Graphics\* graphics) override;*

*virtual PointF\* GetCenter() override;*

*virtual void PlaceIn(RectF rect) override;*

*virtual BOOL HitTest(PointF hitPoint) override;*

*protected:*

*RectF rect;*

*};*

**Файл RectFigure.cpp**

*#include "RectFigure.h"*

*const Color RectFigure::DEFAULT\_BORDER\_COLOR = Color(0, 0, 0);*

*RectFigure::RectFigure(RectF rect, Color color, Color borderColor, Matrix\* matrix) :*

*BaseFigure(color, borderColor, matrix), rect(rect)*

*{*

*}*

*void RectFigure::Draw(Graphics\* graphics)*

*{*

*graphics->SetTransform(matrix);*

*pen->SetColor(borderColor);*

*graphics->DrawRectangle(pen, rect);*

*pen->SetColor(color);*

*graphics->FillRectangle(pen->GetBrush(), rect);*

*graphics->ResetTransform();*

*}*

*PointF\* RectFigure::GetCenter()*

*{*

*PointF\* result = new PointF((rect.GetRight() + rect.GetLeft()) / 2, (rect.GetTop() + rect.GetBottom()) / 2);*

*matrix->TransformPoints(result);*

*return result;*

*}*

*void RectFigure::PlaceIn(RectF rect)*

*{*

*matrix->Reset();*

*lastMatrix->Reset();*

*this->rect = rect;*

*}*

*BOOL RectFigure::HitTest(PointF hitPoint)*

*{*

*Matrix\* invertedMatrix = matrix->Clone();*

*invertedMatrix->Invert();*

*invertedMatrix->TransformPoints(&hitPoint);*

*if (hitPoint.X > rect.GetLeft() && hitPoint.X < rect.GetRight() && hitPoint.Y > rect.GetTop() && hitPoint.Y < rect.GetBottom())*

*return true;*

*return false;*

*}*

**Файл SceneControl.h**

*#pragma once*

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include <CommCtrl.h>*

*#include <unordered\_map>*

*#include "settings.h"*

*#include "BaseWindow.h"*

*#include "DPIScale.h"*

*#include "resource.h"*

*#include "BaseFigure.h"*

*class SceneControl : public BaseWindow<SceneControl>*

*{*

*private:*

*static const PCWSTR DEFAULT\_CLASS\_NAME;*

*static const float MARGIN\_X;*

*static const float MARGIN\_Y;*

*static const float DEFAULT\_PICKER\_WIDTH;*

*static const float DEFAULT\_WINDOW\_HEIGHT;*

*static const float DEFAULT\_BUTTON\_WIDTH;*

*static const wchar\_t\* const MODE\_NAMES[];*

*static const wchar\_t\* const FIGURE\_NAMES[];*

*static const COLORREF BUTTON\_COLORS[];*

*public:*

*SceneControl(Mode\* mode = NULL, Figure\* figure = NULL, Color\* color = NULL, PCWSTR CLASS\_NAME = DEFAULT\_CLASS\_NAME, float windowHeight = DEFAULT\_WINDOW\_HEIGHT, float buttonWidth = DEFAULT\_BUTTON\_WIDTH, float pickerWidth = DEFAULT\_PICKER\_WIDTH);*

*~SceneControl() { DeleteObject(brush); }*

*// HRESULT CreateGraphicsResources();*

*// void DiscardGraphicsResources();*

*float GetWindowHeight() { return windowHeight; }*

*int GetRealWindowHeight();*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) override;*

*protected:*

*void CreateLayout();*

*void SetLayout();*

*Mode\* mode;*

*Figure\* figure;*

*Color\* color;*

*float windowHeight;*

*float buttonWidth;*

*float pickerWidth;*

*std::unordered\_map<HWND, COLORREF> buttons;*

*HBRUSH brush;*

*HWND modePicker;*

*HWND figurePicker;*

*bool tracking;*

*TRACKMOUSEEVENT trackingStruct;*

*};*

**Файл SceneControl.cpp**

*#include "SceneControl.h"*

*#include "helper\_functions.h"*

*const PCWSTR SceneControl::DEFAULT\_CLASS\_NAME = L"Graphics";*

*const float SceneControl::MARGIN\_X = 6.0F;*

*const float SceneControl::MARGIN\_Y = 6.0F;*

*const float SceneControl::DEFAULT\_PICKER\_WIDTH = 200.0F;*

*const float SceneControl::DEFAULT\_WINDOW\_HEIGHT = 60.0F;*

*const float SceneControl::DEFAULT\_BUTTON\_WIDTH = 70.0F;*

*const wchar\_t\* const SceneControl::MODE\_NAMES[] = { // must follow the same order as enum*

*L"Draw",*

*L"Select",*

*L"Drag",*

*L"Scale",*

*L"Rotate"*

*};*

*const wchar\_t\* const SceneControl::FIGURE\_NAMES[] = { // must follow the same order as enum*

*L"Ellipse",*

*L"Rect"*

*};*

*const COLORREF SceneControl::BUTTON\_COLORS[] = {*

*0x00000000,*

*0x00FF0000,*

*0x0000FF00,*

*0x000000FF,*

*0x00FFFF00,*

*0x0000FFFF,*

*0x00FF00FF,*

*0x00FFFFFF*

*};*

*SceneControl::SceneControl(Mode\* mode, Figure\* figure, Color\* color, PCWSTR CLASS\_NAME, float windowHeight, float buttonWidth, float pickerWidth) :*

*BaseWindow<SceneControl>(CLASS\_NAME), mode(mode), figure(figure), color(color), modePicker(NULL), figurePicker(NULL), windowHeight(windowHeight), buttonWidth(buttonWidth), pickerWidth(pickerWidth), buttons(), brush(NULL), tracking(false), trackingStruct{ sizeof(trackingStruct), NULL, NULL, NULL }*

*{*

*}*

*LRESULT SceneControl::HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam)*

*{*

*switch (uMsg)*

*{*

*case WM\_CREATE:*

*{*

*if (!GetParent(m\_hwnd))*

*{*

*DPIScale::Initialize();*

*}*

*CreateLayout();*

*RECT rcWindow;*

*GetWindowRect(m\_hwnd, &rcWindow);*

*MoveWindow(m\_hwnd,*

*rcWindow.left,*

*rcWindow.top,*

*rcWindow.right - rcWindow.left,*

*GetRealWindowHeight(),*

*TRUE);*

*trackingStruct.hwndTrack = m\_hwnd;*

*return 0;*

*}*

*case WM\_SIZE:*

*SetLayout();*

*return 0;*

*case WM\_SIZING:*

*{*

*RECT\* dragRc = (RECT\*)lParam;*

*if (wParam == WMSZ\_BOTTOM || wParam == WMSZ\_BOTTOMLEFT || wParam == WMSZ\_BOTTOMRIGHT)*

*{*

*dragRc->bottom = dragRc->top + GetRealWindowHeight();*

*}*

*else if (wParam == WMSZ\_TOP || wParam == WMSZ\_TOPLEFT || wParam == WMSZ\_TOPRIGHT)*

*{*

*dragRc->top = dragRc->bottom - GetRealWindowHeight();*

*}*

*return TRUE;*

*}*

*case WM\_PAINT:*

*{*

*PAINTSTRUCT ps;*

*HDC hdc = BeginPaint(m\_hwnd, &ps);*

*FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));*

*EndPaint(m\_hwnd, &ps);*

*return 0;*

*}*

*case WM\_MOUSEMOVE:*

*if (!tracking)*

*{*

*trackingStruct.dwFlags = TME\_HOVER | TME\_LEAVE;*

*TrackMouseEvent(&trackingStruct);*

*tracking = true;*

*}*

*return 0;*

*case WM\_MOUSEHOVER:*

*SetFocus(m\_hwnd);*

*trackingStruct.dwFlags = TME\_LEAVE;*

*TrackMouseEvent(&trackingStruct);*

*return 0;*

*case WM\_MOUSELEAVE:*

*tracking = false;*

*return 0;*

*case WM\_COMMAND:*

*if (HIWORD(wParam) == CBN\_SELCHANGE)*

*{*

*HWND parentWND = GetParent(m\_hwnd);*

*if (!parentWND)*

*{*

*parentWND = m\_hwnd;*

*}*

*int ItemIndex = SendMessage((HWND)lParam, (UINT)CB\_GETCURSEL, (WPARAM)0, (LPARAM)0);*

*if ((HWND)lParam == modePicker)*

*{*

*\*mode = (Mode)ItemIndex;*

*PostMessage(m\_hwnd, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*}*

*else if ((HWND)lParam == figurePicker)*

*{*

*\*figure = (Figure)ItemIndex;*

*PostMessage(m\_hwnd, WM\_FIGURE\_CHANGED, NULL, NULL);*

*return 0;*

*}*

*}*

*else if (HIWORD(wParam) == BN\_CLICKED)*

*{*

*HWND parentWND = GetParent(m\_hwnd);*

*if (!parentWND)*

*{*

*parentWND = m\_hwnd;*

*}*

*COLORREF colorRef = buttons[(HWND)lParam];*

*\*color = Color(GetRValue(colorRef), GetGValue(colorRef), GetBValue(colorRef));*

*PostMessage(m\_hwnd, WM\_COLOR\_CHANGED, NULL, NULL);*

*return 0;*

*}*

*break;*

*case WM\_CTLCOLORBTN:*

*{*

*DeleteObject(brush);*

*brush = CreateSolidBrush(buttons[(HWND)lParam]);*

*return (LRESULT)brush;*

*}*

*case WM\_MODE\_CHANGED:*

*SendMessage(modePicker, CB\_SETCURSEL, (WPARAM)\*mode, 0);*

*return 0;*

*case WM\_FIGURE\_CHANGED:*

*SendMessage(figurePicker, CB\_SETCURSEL, (WPARAM)\*figure, 0);*

*return 0;*

*case WM\_COLOR\_CHANGED:*

*return 0;*

*}*

*return DefWindowProc(m\_hwnd, uMsg, wParam, lParam);*

*}*

*int SceneControl::GetRealWindowHeight()*

*{*

*RECT adjustedWindowHeightRect = { 0, 0, 0, DPIScale::DipYToPixels<int>(windowHeight) };*

*AdjustWindowRect(&adjustedWindowHeightRect, GetWindowStyle(m\_hwnd), (BOOL)GetMenu(m\_hwnd));*

*return adjustedWindowHeightRect.bottom - adjustedWindowHeightRect.top;*

*}*

*void SceneControl::CreateLayout()*

*{*

*modePicker = CreateWindowEx(0,*

*WC\_COMBOBOX,*

*NULL,*

*CBS\_DROPDOWNLIST | CBS\_HASSTRINGS | WS\_CHILD | WS\_OVERLAPPED | WS\_VISIBLE,*

*0, 0, 0, 0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*figurePicker = CreateWindowEx(0,*

*WC\_COMBOBOX,*

*NULL,*

*CBS\_DROPDOWNLIST | CBS\_HASSTRINGS | WS\_CHILD | WS\_OVERLAPPED | WS\_VISIBLE,*

*0, 0, 0, 0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*for (auto& i : MODE\_NAMES)*

*{*

*SendMessage(modePicker, CB\_ADDSTRING, 0, (LPARAM)i);*

*}*

*SendMessage(modePicker, CB\_SETCURSEL, (WPARAM)\*mode, 0);*

*for (auto& i : FIGURE\_NAMES)*

*{*

*SendMessage(figurePicker, CB\_ADDSTRING, 0, (LPARAM)i);*

*}*

*SendMessage(figurePicker, CB\_SETCURSEL, (WPARAM)\*figure, 0);*

*for (auto& i : BUTTON\_COLORS)*

*{*

*buttons[CreateWindow(L"BUTTON",*

*NULL,*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | BS\_OWNERDRAW | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL)] = i;*

*}*

*}*

*void SceneControl::SetLayout()*

*{*

*RECT rcClient;*

*GetClientRect(m\_hwnd, &rcClient);*

*int MARGIN\_XPix = DPIScale::DipXToPixels<int>(MARGIN\_X);*

*int MARGIN\_YPix = DPIScale::DipYToPixels<int>(MARGIN\_Y);*

*int pickerWidthPix = DPIScale::DipXToPixels<int>(pickerWidth);*

*MoveWindow(modePicker,*

*MARGIN\_XPix,*

*MARGIN\_YPix,*

*pickerWidthPix,*

*1000,*

*FALSE);*

*RECT rcMode;*

*GetClientRect(modePicker, &rcMode);*

*MoveWindow(figurePicker,*

*MARGIN\_XPix,*

*rcClient.bottom - MARGIN\_YPix - rcMode.bottom,*

*pickerWidthPix,*

*1000,*

*FALSE);*

*int i = 0;*

*for (auto& it : buttons)*

*{*

*MoveWindow(it.first,*

*rcClient.right - (MARGIN\_XPix + buttonWidth) \* (1 + i / 2),*

*MARGIN\_YPix \* (1 - i % 2) + (rcClient.bottom - MARGIN\_YPix - rcMode.bottom) \* (i % 2),*

*buttonWidth,*

*rcMode.bottom,*

*FALSE);*

*++i;*

*}*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*