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 №1

на тему

ОСНОВЫ ПРОГРАММИРОВАНИЯ В WIN 32 API. ОКОННОЕ ПРИЛОЖЕНИЕ WIN 32 С МИНИМАЛЬНОЙ ДОСТАТОЧНОЙ ФУНКЦИОНАЛЬНОСТЬЮ. ОБРАБОТКА ОСНОВНЫХ ОКОННЫХ СООБЩЕНИЙ.

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**

Целью выполнения лабораторной работы является создание оконного приложения на *Win32 API*, обладающее минимальным функционалом, позволяющим отработать базовые навыки написания программы на *Win32 API*, таких как обработка оконных сообщений.

В качестве задачи необходимо реализовать оконное приложение, которое позволяет пользователю рисовать и редактировать графические фигуры (круги, прямоугольники) с помощью мыши и клавиш клавиатуры.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Win32 API (также называемый Windows API) — это исходная платформа для собственных Windows-приложений на языке C/C++, которым требуется прямой доступ к Windows и оборудованию. Он обеспечивает первоклассный опыт разработки без зависимости от управляемой среды выполнения, такой как .NET и WinRT (для приложений UWP для Windows 10). Благодаря этому API Win32 стает оптимальной платформой для приложений, которым требуется самый высокий уровень производительности и прямой доступ к системному оборудованию [1].

Приложение графического пользовательского интерфейса должно реагировать на события от пользователя и операционной системы. Эти события могут возникать в любое время во время работы программы, практически в любом порядке. Для решения этой проблемы Windows использует модель передачи сообщений. Операционная система взаимодействует с окном приложения, передавая ему сообщения [2].

**3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

Программа представляет собой простой графический редактор, позволяющий взаимодействовать с геометрическими фигурами.

Предусмотрены следующие функции:

1. Выбор фигуры. Пользователь может выбрать фигуру используя клавиши клавиатуры (*F6* – овал, *F7* – прямоугольник), при этом должен быть активирован режим добавления фигуры.
2. Добавление фигуры. Клавиша *F1* переводит программу в режим добавления фигуры. В этом режиме пользователь может «рисовать» фигуры с зажатой левой клавишей мыши.
3. Выделение фигуры. Для того чтобы выделить фигуру пользователь должен перевести программу в режим выделения фигуры с помощью клавиши *F2,* после чего навести курсор мыши на фигуру и нажать левую клавишу.
4. Масштабирование фигуры. Изменить размер фигуры можно передвигая курсор с зажатой левой клавишей мыши, при этом программа должна находиться в режиме масштабирования. Для перехода в режим масштабирования пользователь должен нажать клавишу *F3*.
5. Вращение фигуры. Работает аналогично масштабированию. Для перехода в режим вращения используется клавиша *F4*.
6. Перемещение фигуры. Работает аналогично вращению и масштабированию. Перехода в режим перемещения осуществляется клавишей F5.
7. Удаление фигуры. Удаление производится нажатием клавиши *DEL* в режиме выделения.

Получение данных с клавиатуры реализовано с помощью обработки сообщения *WM\_KEYDOWN* и таблиц ускорителей. Для отображения фигур используется *Direct2D*.

Графический интерфейс программы представлен на рисунке 1.

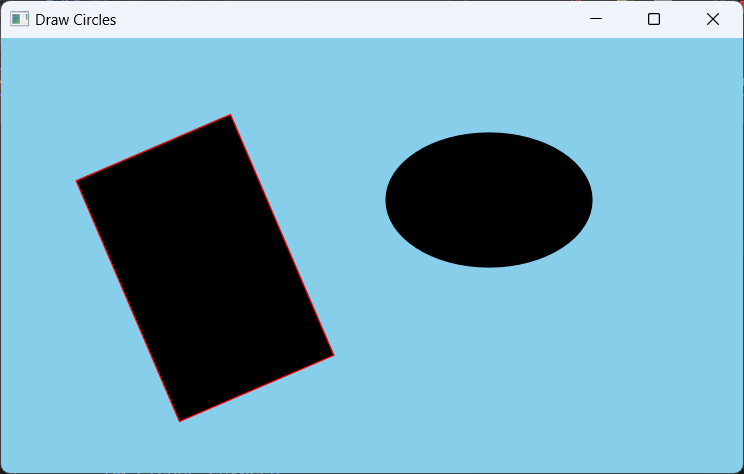


Рисунок 1 – Пример графического интерфейса программы

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Начало работы с классическими приложениями для Windows, которые используют API Win32 [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/desktop-programming.

[2] Сообщения окна (начало работы с Win32 и C++) [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/window-messages.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Исходный код программы**

**Файл BaseFigure.h**

*#pragma once*

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*class BaseFigure*

*{*

*private:*

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

*public:*

*BaseFigure(D2D1\_COLOR\_F color, D2D1\_COLOR\_F borderColor = DEFAULT\_BORDER\_COLOR, D2D1::Matrix3x2F matrix = D2D1::Matrix3x2F::Identity());*

*void Translate(D2D1\_SIZE\_F size);*

*void Rotate(FLOAT angle, D2D1\_POINT\_2F center);*

*void Scale(D2D1\_SIZE\_F size, D2D1\_POINT\_2F center);*

*void RevertTransform();*

*void SaveTransform();*

*void SetColor(D2D1\_COLOR\_F color) { this->color = color; }*

*void SetBorderColor(D2D1\_COLOR\_F borderColor) { this->borderColor = borderColor; }*

*void SetMatrix(D2D1::Matrix3x2F matrix) { this->matrix = matrix; }*

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

*D2D1\_COLOR\_F GetBorderColor() { return borderColor; }*

*D2D1::Matrix3x2F GetMatrix() { return matrix; }*

*virtual void Draw(ID2D1RenderTarget\* pRT, ID2D1SolidColorBrush\* pBrush) = 0;*

*virtual D2D1\_POINT\_2F GetCenter() = 0;*

*virtual void PlaceIn(D2D1\_RECT\_F rect) = 0;*

*virtual BOOL HitTest(D2D1\_POINT\_2F hitPoint) = 0;*

*protected:*

*D2D1\_COLOR\_F color;*

*D2D1\_COLOR\_F borderColor;*

*D2D1::Matrix3x2F matrix;*

*D2D1::Matrix3x2F lastMatrix;*

*};*

*Файл BaseFigure.cpp*

*#include "BaseFigure.h"*

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

*BaseFigure::BaseFigure(D2D1\_COLOR\_F color, D2D1\_COLOR\_F borderColor, D2D1::Matrix3x2F matrix) :*

*color(color), borderColor(borderColor), matrix(matrix)*

*{*

*SaveTransform();*

*}*

*void BaseFigure::Translate(D2D1\_SIZE\_F size)*

*{*

*lastMatrix = matrix;*

*matrix = lastMatrix \* D2D1::Matrix3x2F::Translation(size);*

*}*

*void BaseFigure::Rotate(FLOAT angle, D2D1\_POINT\_2F center)*

*{*

*matrix = lastMatrix = matrix;*

*matrix = lastMatrix \* D2D1::Matrix3x2F::Rotation(angle, center);*

*}*

*void BaseFigure::Scale(D2D1\_SIZE\_F size, D2D1\_POINT\_2F center)*

*{*

*lastMatrix = matrix;*

*matrix = lastMatrix \* D2D1::Matrix3x2F::Scale(size, center);*

*}*

*void BaseFigure::RevertTransform()*

*{*

*matrix = lastMatrix;*

*}*

*void BaseFigure::SaveTransform()*

*{*

*lastMatrix = matrix;*

*}*

**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 D2D1\_COLOR\_F DEFAULT\_BORDER\_COLOR;*

*public:*

*EllipseFigure(D2D1\_ELLIPSE ellipse, D2D1\_COLOR\_F color, D2D1\_COLOR\_F borderColor = DEFAULT\_BORDER\_COLOR, D2D1::Matrix3x2F matrix = D2D1::Matrix3x2F::Identity());*

*void SetEllipse(D2D1\_ELLIPSE ellipse) { this->ellipse = ellipse; }*

*D2D1\_ELLIPSE GetEllipse() { return ellipse; }*

*virtual void Draw(ID2D1RenderTarget\* pRT, ID2D1SolidColorBrush\* pBrush) override;*

*virtual D2D1\_POINT\_2F GetCenter() override { return matrix.TransformPoint(ellipse.point); }*

*virtual void PlaceIn(D2D1\_RECT\_F rect) override;*

*virtual BOOL HitTest(D2D1\_POINT\_2F hitPoint) override;*

*protected:*

*D2D1\_ELLIPSE ellipse;*

*};*

**Файл EllipseFigure.cpp**

*#include "EllipseFigure.h"*

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

*EllipseFigure::EllipseFigure(D2D1\_ELLIPSE ellipse, D2D1\_COLOR\_F color, D2D1\_COLOR\_F borderColor, D2D1::Matrix3x2F matrix) :*

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

*{*

*}*

*void EllipseFigure::Draw(ID2D1RenderTarget\* pRT, ID2D1SolidColorBrush\* pBrush)*

*{*

*pRT->SetTransform(matrix);*

*pBrush->SetColor(color);*

*pRT->FillEllipse(ellipse, pBrush);*

*pBrush->SetColor(borderColor);*

*pRT->DrawEllipse(ellipse, pBrush, 1.0f);*

*pRT->SetTransform(D2D1::Matrix3x2F::Identity());*

*}*

*void EllipseFigure::PlaceIn(D2D1\_RECT\_F rect)*

*{*

*matrix = lastMatrix = D2D1::Matrix3x2F::Identity();*

*D2D1\_POINT\_2F center = D2D1::Point2F((rect.right + rect.left) / 2, (rect.bottom + rect.top) / 2);*

*FLOAT radiusX = (rect.right - rect.left) / 2;*

*FLOAT radiusY = (rect.bottom - rect.top) / 2;*

*ellipse = D2D1::Ellipse(center, radiusX, radiusY);*

*}*

*BOOL EllipseFigure::HitTest(D2D1\_POINT\_2F hitPoint)*

*{*

*D2D1::Matrix3x2F invertedMatrix = matrix;*

*invertedMatrix.Invert();*

*hitPoint = invertedMatrix.TransformPoint(hitPoint);*

*const float a = ellipse.radiusX;*

*const float b = ellipse.radiusY;*

*const float x1 = hitPoint.x - ellipse.point.x;*

*const float y1 = hitPoint.y - ellipse.point.y;*

*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 D2D1\_COLOR\_F DEFAULT\_BORDER\_COLOR;*

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

*static const float DEFAULT\_FIGURE\_SIZE;*

*public:*

*GraphicsScene(Mode\* mode = NULL, Figure\* figure = NULL, D2D1\_COLOR\_F\* color = NULL, ID2D1Factory\* pFactory = NULL,*

*PCWSTR CLASS\_NAME = DEFAULT\_CLASS\_NAME, D2D1\_COLOR\_F borderColor = DEFAULT\_BORDER\_COLOR, D2D1\_COLOR\_F selectionColor=DEFAULT\_SELECTION\_COLOR);*

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

*void ClearSelection();*

*BOOL Select(D2D1\_POINT\_2F hitPoint);*

*HRESULT CreateGraphicsResources();*

*void DiscardGraphicsResources();*

*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:*

*D2D1\_COLOR\_F borderColor;*

*D2D1\_COLOR\_F selectionColor;*

*ID2D1Factory\* pFactory;*

*Mode\* mode;*

*Figure\* figure;*

*D2D1\_COLOR\_F\* color;*

*ID2D1HwndRenderTarget\* pRenderTarget;*

*ID2D1SolidColorBrush\* pBrush;*

*D2D1\_POINT\_2F 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 D2D1\_COLOR\_F GraphicsScene::DEFAULT\_BORDER\_COLOR = D2D1::ColorF(D2D1::ColorF::Black);*

*const D2D1\_COLOR\_F GraphicsScene::DEFAULT\_SELECTION\_COLOR = D2D1::ColorF(D2D1::ColorF::Red);*

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

*GraphicsScene::GraphicsScene(Mode\* mode, Figure\* figure, D2D1\_COLOR\_F\* color,*

*ID2D1Factory\* pFactory, PCWSTR CLASS\_NAME, D2D1\_COLOR\_F borderColor, D2D1\_COLOR\_F selectionColor) :*

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

*borderColor(borderColor), selectionColor(selectionColor), pRenderTarget(NULL), pBrush(NULL), ptMouse(D2D1::Point2F()), 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(D2D1\_POINT\_2F hitPoint)*

*{*

*ClearSelection();*

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

*{*

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

*{*

*selection = (++i).base();*

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

*return TRUE;*

*}*

*}*

*return FALSE;*

*}*

*HRESULT GraphicsScene::CreateGraphicsResources()*

*{*

*HRESULT hr = S\_OK;*

*if (pRenderTarget == NULL)*

*{*

*RECT rc;*

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

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

*hr = pFactory->CreateHwndRenderTarget(*

*D2D1::RenderTargetProperties(),*

*D2D1::HwndRenderTargetProperties(m\_hwnd, size),*

*&pRenderTarget);*

*if (SUCCEEDED(hr))*

*{*

*const D2D1\_COLOR\_F color = D2D1::ColorF(1.0f, 1.0f, 0);*

*hr = pRenderTarget->CreateSolidColorBrush(color, &pBrush);*

*}*

*}*

*return hr;*

*}*

*void GraphicsScene::DiscardGraphicsResources()*

*{*

*SafeRelease(&pRenderTarget);*

*SafeRelease(&pBrush);*

*}*

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

*{*

*ClearSelection();*

*switch (\*figure)*

*{*

*case Figure::Ellipse:*

*{*

*ptMouse = D2D1::Point2F(dipX, dipY);*

*D2D1\_ELLIPSE ellipse = D2D1::Ellipse(ptMouse, DEFAULT\_FIGURE\_SIZE, DEFAULT\_FIGURE\_SIZE);*

*selection = figures.insert(*

*figures.end(),*

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

*break;*

*}*

*case Figure::Rect:*

*{*

*ptMouse = D2D1::Point2F(dipX, dipY);*

*D2D1\_RECT\_F rect = D2D1::Rect(ptMouse.x - DEFAULT\_FIGURE\_SIZE, ptMouse.y - DEFAULT\_FIGURE\_SIZE, ptMouse.x + DEFAULT\_FIGURE\_SIZE, ptMouse.y + 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()*

*{*

*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 = DPIScale::PixelsToDipsX(pixelX);*

*const float dipY = 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 = DPIScale::PixelsToDipsX(pixelX);*

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

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

*{*

*switch (\*mode)*

*{*

*case Mode::DrawMode:*

*{*

*float left;*

*float right;*

*float top;*

*float bottom;*

*if (ptMouse.x > dipX)*

*{*

*left = dipX;*

*right = ptMouse.x;*

*}*

*else*

*{*

*left = ptMouse.x;*

*right = dipX;*

*}*

*if (ptMouse.y > dipY)*

*{*

*top = dipY;*

*bottom = ptMouse.y;*

*}*

*else*

*{*

*top = ptMouse.y;*

*bottom = dipY;*

*}*

*Selection()->PlaceIn(D2D1::Rect(left, top, right, bottom));*

*break;*

*}*

*case Mode::DragMode:*

*{*

*Selection()->Translate({ dipX - ptMouse.x, dipY - ptMouse.y });*

*ptMouse = { dipX, dipY };*

*break;*

*}*

*{*

*case Mode::ScaleMode:*

*Selection()->RevertTransform();*

*D2D1\_POINT\_2F center = Selection()->GetCenter();*

*D2D1\_SIZE\_F size = { abs((dipX - center.x) / (ptMouse.x - center.x)), abs((dipY - center.y) / (ptMouse.y - center.y)) };*

*Selection()->Scale(size, Selection()->GetCenter());*

*break;*

*}*

*case Mode::RotateMode:*

*{*

*D2D1\_POINT\_2F 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, Selection()->GetCenter());*

*ptMouse = { dipX, dipY };*

*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))*

*{*

*if (FAILED(D2D1CreateFactory(*

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

*{*

*return -1; // Fail CreateWindowEx.*

*}*

*DPIScale::Initialize();*

*}*

*trackingStruct.hwndTrack = m\_hwnd;*

*return 0;*

*case WM\_DESTROY:*

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

*{*

*SafeRelease(&pFactory);*

*PostQuitMessage(0);*

*}*

*DiscardGraphicsResources();*

*return 0;*

*case WM\_PAINT:*

*OnPaint();*

*return 0;*

*case WM\_SIZE:*

*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(parentWND, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*case ID\_SELECT\_MODE:*

*\*mode = Mode::SelectMode;*

*PostMessage(parentWND, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*case ID\_DRAG\_MODE:*

*\*mode = Mode::DragMode;*

*PostMessage(parentWND, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*case ID\_SCALE\_MODE:*

*\*mode = Mode::ScaleMode;*

*PostMessage(parentWND, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*case ID\_ROTATE\_MODE:*

*\*mode = Mode::RotateMode;*

*PostMessage(parentWND, WM\_MODE\_CHANGED, NULL, NULL);*

*return 0;*

*case ID\_ELLIPSE:*

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

*{*

*\*figure = Figure::Ellipse;*

*PostMessage(parentWND, WM\_FIGURE\_CHANGED, NULL, NULL);*

*}*

*return 0;*

*case ID\_RECT:*

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

*{*

*\*figure = Figure::Rect;*

*PostMessage(parentWND, 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)*

*{*

*Mode mode = Mode::SelectMode;*

*Figure figure = Figure::Ellipse;*

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

*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 RectFigure.h**

*#pragma once*

*#include "BaseFigure.h"*

*class RectFigure : public BaseFigure*

*{*

*private:*

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

*public:*

*RectFigure(D2D1\_RECT\_F rect, D2D1\_COLOR\_F color, D2D1\_COLOR\_F borderColor = DEFAULT\_BORDER\_COLOR, D2D1::Matrix3x2F matrix = D2D1::Matrix3x2F::Identity());*

*void SetEllipse(D2D1\_RECT\_F rect) { this->rect = rect; }*

*D2D1\_RECT\_F GetRect() { return rect; }*

*virtual void Draw(ID2D1RenderTarget\* pRT, ID2D1SolidColorBrush\* pBrush) override;*

*virtual D2D1\_POINT\_2F GetCenter() override { return matrix.TransformPoint(D2D1::Point2F((rect.right + rect.left) / 2, (rect.top + rect.bottom) / 2)); }*

*virtual void PlaceIn(D2D1\_RECT\_F rect) override;*

*virtual BOOL HitTest(D2D1\_POINT\_2F hitPoint) override;*

*protected:*

*D2D1\_RECT\_F rect;*

*};*

**Файл RectFigure.cpp**

*#include "RectFigure.h"*

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

*RectFigure::RectFigure(D2D1\_RECT\_F rect, D2D1\_COLOR\_F color, D2D1\_COLOR\_F borderColor, D2D1::Matrix3x2F matrix) :*

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

*{*

*}*

*void RectFigure::Draw(ID2D1RenderTarget\* pRT, ID2D1SolidColorBrush\* pBrush)*

*{*

*pRT->SetTransform(matrix);*

*pBrush->SetColor(color);*

*pRT->FillRectangle(rect, pBrush);*

*pBrush->SetColor(borderColor);*

*pRT->DrawRectangle(rect, pBrush, 1.0f);*

*pRT->SetTransform(D2D1::Matrix3x2F::Identity());*

*}*

*void RectFigure::PlaceIn(D2D1\_RECT\_F rect)*

*{*

*matrix = lastMatrix = D2D1::Matrix3x2F::Identity();*