**ТИТУЛЬНЫЙ ЛИСТ**

**ЛИСТ КУРСОВОГО ЗАДАНИЯ**

Содержание

[Введение 5](#_Toc12803)

[1 Анализ предметной области и формирование требований к проектируемому программному средству 7](#_Toc11716)

[1.1 Анализ предметной области 7](#_Toc31894)

[1.2 Обзор аналогов 8](#_Toc27405)

[1.3 Формирование требований к проектируемому программному средству 10](#_Toc27474)

[2 Анализ требований в программному средству и разработка функциональных требований 11](#_Toc12850)

[3 Проектирование программного средства 12](#_Toc18038)

[3.1 Описание используемых технологий 12](#_Toc20275)

[3.2 Среда разработки 12](#_Toc9170)

[3.3 Общая структура программы 13](#_Toc23334)

[3.4 Описание работы программы 13](#_Toc20612)

[4 Тестирование программного средства 16](#_Toc11660)

[5 Руководство пользователя программы 17](#_Toc29053)

[5.1 Запуск программы 17](#_Toc7483)

[5.2 Процесс рисования 17](#_Toc22846)

[5.3 Инструкции по управлению 18](#_Toc13740)

[5.4 Меню игры 18](#_Toc17520)

[Заключение 19](#_Toc27567)

[Список использованных источников 20](#_Toc31807)

[Приложение А 21](#_Toc29911)

Введение

Представление данных на мониторе компьютера в графическом виде впервые было реализовано в середине 50-х годов для больших ЭВМ, применявшихся в научных и военных исследованиях. С тех пор графический способ отображения данных стал неотъемлемой частью подавляющего числа компьютерных систем, в особенности персональных.

Компьютерная графика – это облать деятельности, изучающая методы и средства создания и обработки изображений с помощью программно-аппаратных вычислительных комплексов.

Она охватывает все виды и формы представления изображений, доступных для восприятия человеком либо на экране монитора, либо в виде копии на внешнем носителе (бумага, киноплёнка, ткань и прочее).

Без компьютерной графики невозможно представить себе не только компьютерный, но и обычный, вполне материальный мир. На сегодняшний день компьютеры и компьютерная графика неотъемлемая часть жизни современного общества. Например, медицина (компьютерная томография), научные исследования (визуализация строения вещества, векторных полей и других данных), моделирование тканей и одежды, опытно-конструкторские разработки, реклама, цветные журналы, спецэффекты в фильмах - всё это в той или иной мере имеет отношение к компьютерной графике. Поэтому созданы программы для создания и редактирования изображений – графические редакторы.

Целью данного курсового проекта является написание графического редактора с минимальным необходимым функционалом.

В настоящей пояснительной записке отражены следующие этапы написания курсового проекта

1. Анализ предметной области и формирование требований к проектируемому программному средству. Исследование существующих аналогов и формирование требований.
2. Анализ требований к программному средству с функциями графического редактора и разработка функциональных требований. На основе поставленных требований описывается функциональность разрабатываемого программного средства.
3. Проектирование программного средства. Определение архитектуры программного средства.
4. Конструирование программного средства. Описание модулей и компонентов программного средства, а также обоснование выбранных решений относительно функциональных требований.
5. Тестирование программного средства. Характеристика действий, выполненных для проведения тестирования программного средства.
6. Руководство пользователя программы. Включает в себя последовательности действий, выполнение которых пользователем приведёт к успешному решению определённых задач.

В качестве языка программирования был выбран язык С++ с использованием набора функций программирования интерфейсов WinApi[1], а средой разработки – Visual Studio 2019, которые изучаются в рамках предмета «Операционные системы и системное программирование»[2,3].

При выполнении всех этих этапов в итоге получится приложение графического редактора, полностью удовлетворяющее цели курсового проекта.

# Анализ предметной области и формирование требований к проектируемому программному средству

## Анализ предметной области

Графический редактор – программа, позволяющая создавать графические изображения или изменять уже имеющиеся. В настоящий момент, существует достаточно большое количество графических редакторов, как довольно известных, так и не очень распространенных. Изображения в графических редакторах хранятся по-разному.

Графические редакторы подразделяются на две категории: растровые и векторные. Они отличаются по способу, которым представляют графическую информацию.

Векторные изображения формируются из объектов (точка, линия, окружность и т. д.), которые хранятся в памяти компьютера в виде графических примитивов и описывающих их математических формул. Например, графический примитив точка задается своими координатами (X, У), линия – координатами начала (XI, Y1) и конца (Х2, Y2), окружность – координатами центра (X, У) и радиусом (R), прямоугольник – величиной сторон и координатами левого верхнего угла (XI, У1) и правого нижнего угла (Х2, У2) и т. д. Для каждого примитива назначается также цвет.

Растровое изображение хранится с помощью точек различного цвета (пикселей), которые образуют строки и столбцы. Любой пиксель имеет фиксированное положение и цвет. Хранение каждого пикселя требует некоторого количества бит информации, которое зависит от количества допустимых цветов в изображении.

Как следствие, способ представления растровых изображений совершенно отличен от векторных. Растровые изображения состоят из отдельных точек, называемых растром. Такое представление изображений существует не только в цифровом виде. Растровые изображения обеспечивают максимальную реалистичность, поскольку в цифровую форму переводится каждый мельчайший фрагмент оригинала. Такие изображения сохраняются в файлах гораздо большего объёма, чем векторные, поскольку в них запоминается информация о каждом пикселе изображения. Таким образом, качество растровых изображений зависит от их размера (числа пикселей по горизонтали и вертикали) и количества цветов, которые могут принимать пиксели. Как следствие того, что они состоят из пикселей фиксированного размера, свободное масштабирование без потери качества к ним не применимо. Эта особенность, а также сама структура растровых изображений несколько затрудняет их редактирование и обработку.

Разрабатываемое программное средство является растровым графическим редактором.

## Обзор аналогов

Существует немалое число реализаций графических редакторов с минимальным набором функций. Подобное программное средство может быть реализовано в браузере, для мобильных устройств, а также для ПК.

Наиболее известным графическим редактором для ПК является Paint. Microsoft Paint – многофункциональный, но в то же время довольно простой в использовании растровый графический редактор компании [Microsoft](https://ru.wikipedia.org/wiki/Microsoft" \o "Microsoft), входящий в состав всех операционных систем [Windows](https://ru.wikipedia.org/wiki/Windows" \o "Windows), начиная с первых версий. При открытии приложения пользователю становится доступен новый «холст», а так же панель инструментов.

Основные функции – настройка и использование кистей, заливки, рисование простых фигур, выделение и разворот/перемещение объектов. Доступна палитра настройки цветов. Пользователь имеет возможность создавать новые файлы, открывать и редактировать уже существующие, а также сохранять. Эти функции являются общими для всех графических редакторов.

Одной из наиболее заметных черт графического редактора Paint является его минималистичный интерфейс.

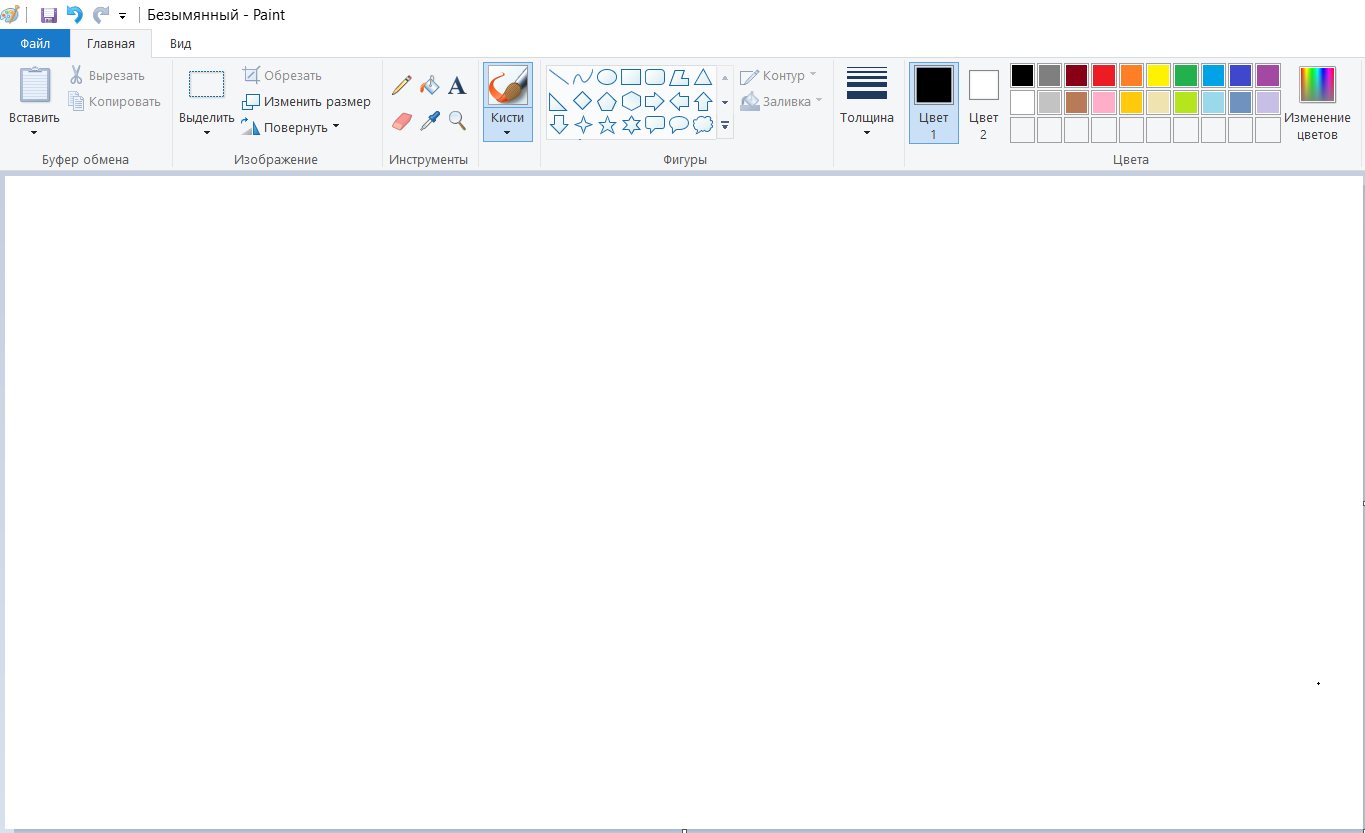


Рисунок 1.2.1 – Интерфейс графического редактора Paint

С развитием мобильных устройств, были созданы аналоги и для телефонов. Их отличительной особенностью является упрощённый интерфейс, что позволяет управлять редактором без использования стилуса. Примером подобных редакторов может являться ibisPaint X. Приложению доступны все основные функции графических редакторов, упомянутые в предыдущем пункте. Стоит отметить, что мобильные аналоги чаще используются для редактирования фотографий, чем для создания новых изображений.

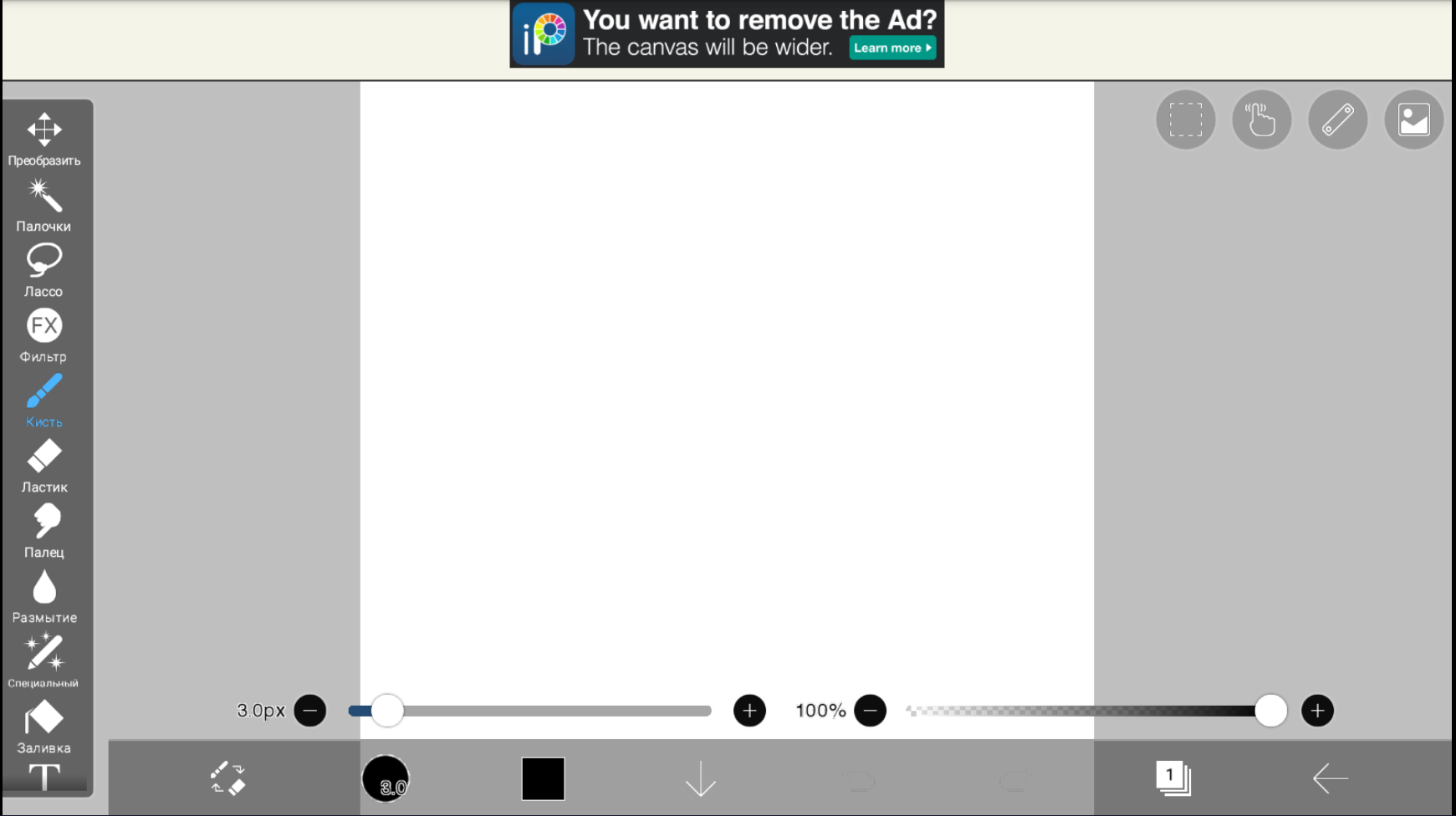


Рисунок 1.2.2 – Интерфейс графического редактора ibisPaint X

Стоит упомянуть и браузерные версии графических редакторов. Большиство из них созданы с развлекательной целью, однако обладают всеми основными функциями. К этой категории можно отнести браузерный редактор Vector Paint. Эти программы не требуют установки на ПК пользователя и доступны с любого устройства.

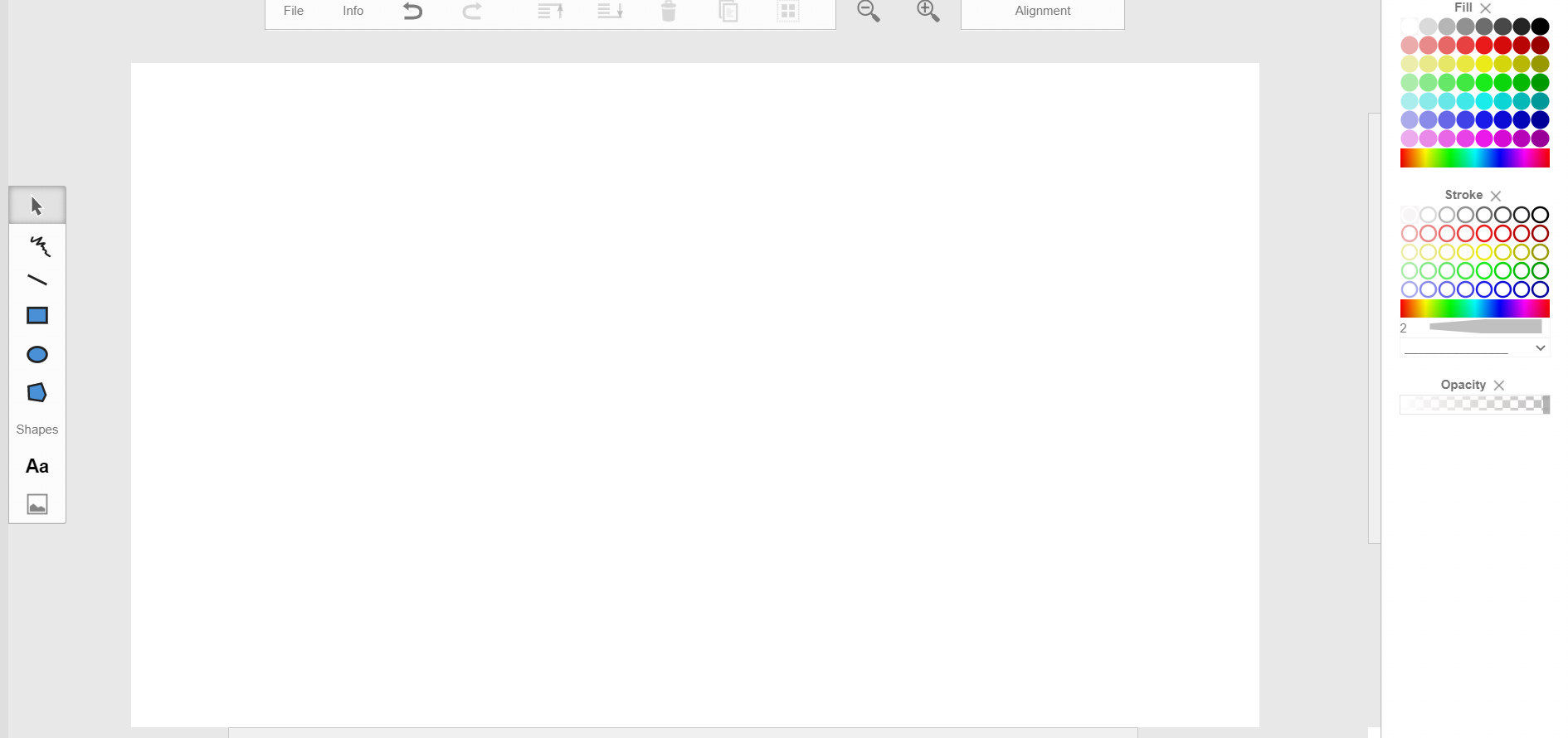


Рисунок 1.2.3 – Интерфейс браузерного графического редактор Vector Paint

## Формирование требований к проектируемому программному средству

Таким образом, при проектировании программного средства были учтены возможности аналогов, их основной функционал, недостатки и преимущества. В соответствии с проведенным анализом были сформулированы задачи, касающиеся функциональности и технических особенностей проекта. Интерфейс программы должен быть простым, но предоставляющим весь необходимый функционал. Цвета в разработке препочтительно использовать светлые.

# Анализ требований в программному средству и разработка функциональных требований

Назначение программного средства «Графический редактор MiniSimplePaint» заключается в предоставлении пользователю возможности создания и редакирования изображений.

Исходя из поставленной задачи и сформулированных требований, разрабатываемый программный продукт должен выполнять следующие функции:

* использование и настройка инструмента «Кисть»;
* использование инструмента «Текст»;
* использование инструментов «Линия», «Ломаная линия», «Многоугольник»;
* использование инструментов «Прямоугольник», «Эллипс»;
* заливка фигуры определённым цветом;
* сохранение файла;
* открытие уже существующего файла;
* вывод информации о программе.

Схема UseCase функционала программы:

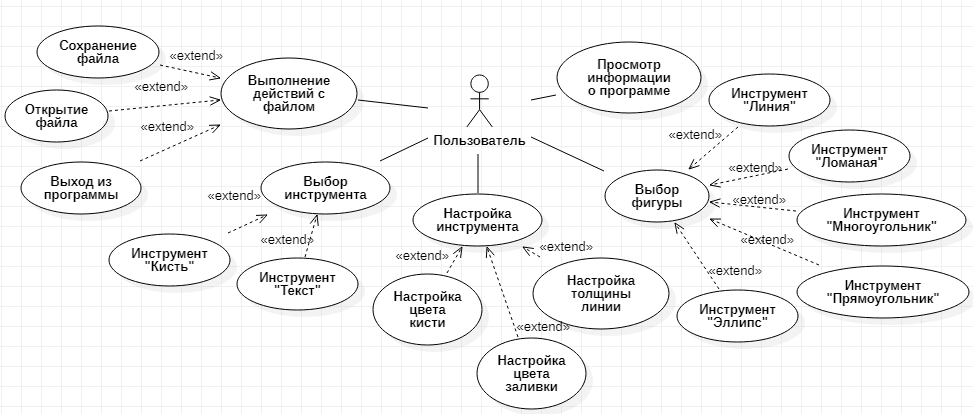


Рисунок 2.1 – UseCase-диаграмма

# Проектирование программного средства

## Описание используемых технологий

Операционная система Windows является самой массовой и доступной операционной системой, которые существуют на данный момент. В связи с этим приоритетной платформой для разработки можно считать именно её.

Данный курсовой проект предусматривает написание простейшего windows-приложения, используя набор базовых функций WinApi.

Windows API (англ. application programming interfaces) – общее наименование целого набора базовых функций интерфейсов программирования приложений операционных систем семейств Windows и Windows NT корпорации «Майкрософт». Является самым прямым способом взаимодействия приложений с Windows. Для создания программ, использующих Windows API, «Майкрософт» выпускает SDK, который называется Platform SDK и содержит документацию, набор библиотек, утилиты других инструментальных средств.[4]

Windows API был изначально спроектирован для использования в программах, написанных на языке C (или C++). Работа через Windows API – это наиболее близкий к системе способ взаимодействия с ней из прикладных программ.

В настоящее время язык С++ является ведущим языком программирования в различных средах и операционных системах. Программу на языке С++ можно написать и cкомпилировать в таких операционных системах как MS DOS, WINDOWS, UNIX, MACINTOSH OS.

Язык С++ является наследником языка С, который также используется в наши дни. Не смотря на множество добавлений в С++ по сравнению с С, в компиляторах С++ можно откомпилировать программу С, так как в С++ используются те же правила и функции, что и в языке С. Главным отличием между языками является то, что язык С – процедурный, а С++ – объектно-ориентированный, но также может использоваться, как процедурный.

Для программирования под каждую ОС используются различные компиляторы и линкеры, в которых могут быть свои правила, однако, все компиляторы должны придерживаться общего стандарта С++

Не смотря на появление конкурирующих языков, таких как: JAVA, Python, C#, С++ не теряет популярности и продолжает использоваться.

## Среда разработки

Несмотря на то, что любое приложение с использованием языка С++ можно написать в текстовом редакторе и скомпилировать, этот метод очень трудный, особенно при работе с использованием системных ресурсов, окон, графики. Также возникает проблема при использовании нескольких файлов, когда нужно постоянно переключаться между ними. Для упрощения жизни программистов были созданы IDE (Integrated Development Environment — интегрированная среда разработки).

Для написания игрового приложения была использована среда разработки Microsoft Visual Studio. Это интегрированная среда разработки с широкими возможностями для создания приложений для Windows, Android и iOS, а также современных веб-приложений и облачных служб. Данная среда помогает создавать приложения, обладающие инновационным интерфейсом и функциональными возможностями. Эта интегрированная среда упрощает выполнение сложных задач, что позволит вам уделять больше времени и сил на достижение своих непосредственных целей.

## Общая структура программы

Данное программное средство предоставляет пользователю возможность создавать простые графические формы. Суть программы сводится к написанию компонентов, которые будут реагировать на действия пользователя, а в соответствии с этими действиями выдавать результат, который отразиться на внешнем виде программы.

## Описание работы программы

Основными классами, обеспечивающими корректную работу программы, являются классы: DrawingProcess, DrawingObject. Так же следует указать класс LineObject как пример класса, позволяющего реализовать конкретную фигуру.

Основная задача класса DrawingProcess – настраивать процесс отрисовки объектов, выбор рисуемой фигуры и переключение между состояниями «рисования» и «ожидания».

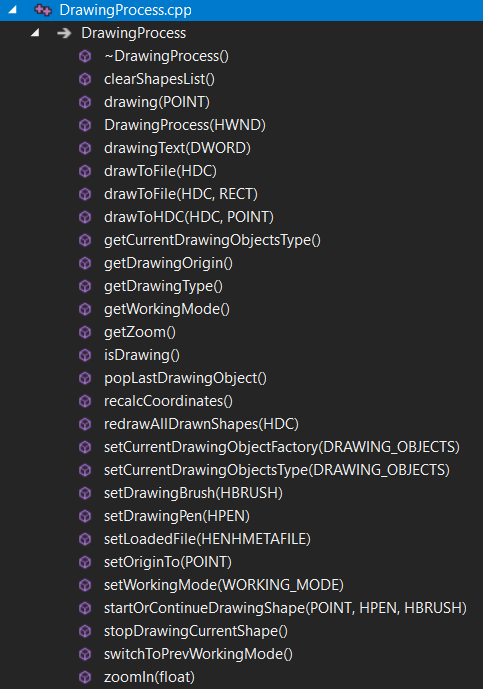


Рисунок 3.4.1 – Диаграмма класса DrawingProcess

Основная задача класса DrawingObject – отрисовка объекта выбранного пользователем типа, основанная на данных о координатах объекта.

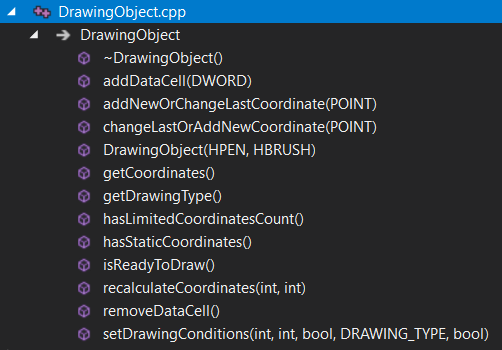


Рисунок 3.4.2 – Диаграмма класса DrawingObject

Класс LineObject не играет ключевую роль для разработки программного средства, однако даёт представление о том, как выполняется непосредственно отрисовка объекта выбранного типа - в данном случае использование пользователем инструмента «Линия».

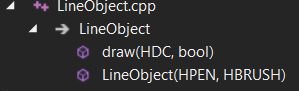


Рисунок 3.4.3 – Диаграмма класса GridPosition

Функции, позволяющие пользователю корректно взаимодействовать с интерфейсом и обрабатывающие ввод, приведены в файле Main.cpp; упомянутые выше классы реализуют внутреннюю логику программного средства.

# Тестирование программного средства

Осуществлялось функциональное тестирование. Отчет о проведенном тестировании представлен в таблице 4.1.

Таблица 4.1 – Тестирование программного средства

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестируемая функция | Ожидаемый результат | Полученный результат |
| 1 | Открытие существующего файла | Файл открывается | Соответствует ожидаемому |
| 2 | Рисование произвольной формы с помощью элемента «Кисть» | Произвольная форма нарисована | Соответствует ожидаемому |
| 3 | Изменение цвета кисти и заливки | Цвет изменён | Соответствует ожидаемому |
| 4 | Изменение толщины линии | Толщина линии изменена | Соответствует ожидаемому |
| 5 | Использование инструмента «Текст» | Вводимый текст отображается на холсте | Соответствует ожидаемому |
| 6 | Использование инструмента «Линия» | Линия заданного цвета с заданной толщиной линии отрисована по заданным координатам | Соответствует ожидаемому |
| 7 | Использование инструмента «Эллипс» | Эллипс заданного цвета с заданной толщиной линии отрисован по заданным координатам | Соответствует ожидаемому |
| 8 | Сохранение файла | Файл сохраняется | Соответствует ожидаемому |
| 9 | Проверка совместимости с оригинальным приложением Paint | Файл, созданный в тестируемом программном средстве, открывается в Paint | Соответствует ожидаемому |

В ходе итогового тестирования не было выявлено каких-либо ошибок либо некорректной работы программы.

# Руководство пользователя программы

## Запуск программы

Для открытия приложения необходимо запустить файл MiniSimplePaint.exe. Далее появится окно c игрой, представленное на рисунке 5.1.

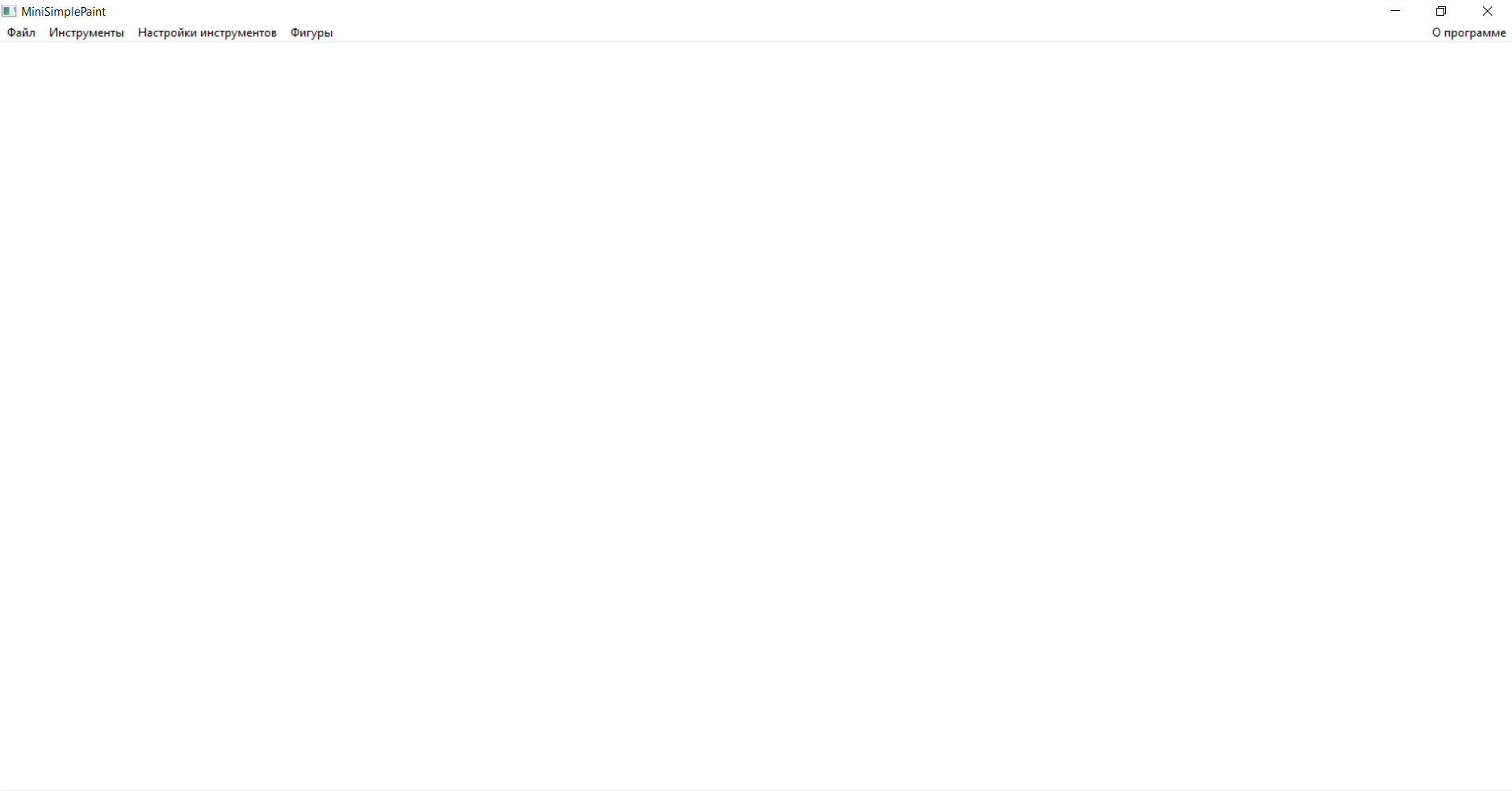


Рисунок 5.1 – Окно программы

## Процесс рисования

Сверху расположена панель инструментов, белое поле доступно для рисования. На рисунке 5.2 приведён примен использования различных инструментов.

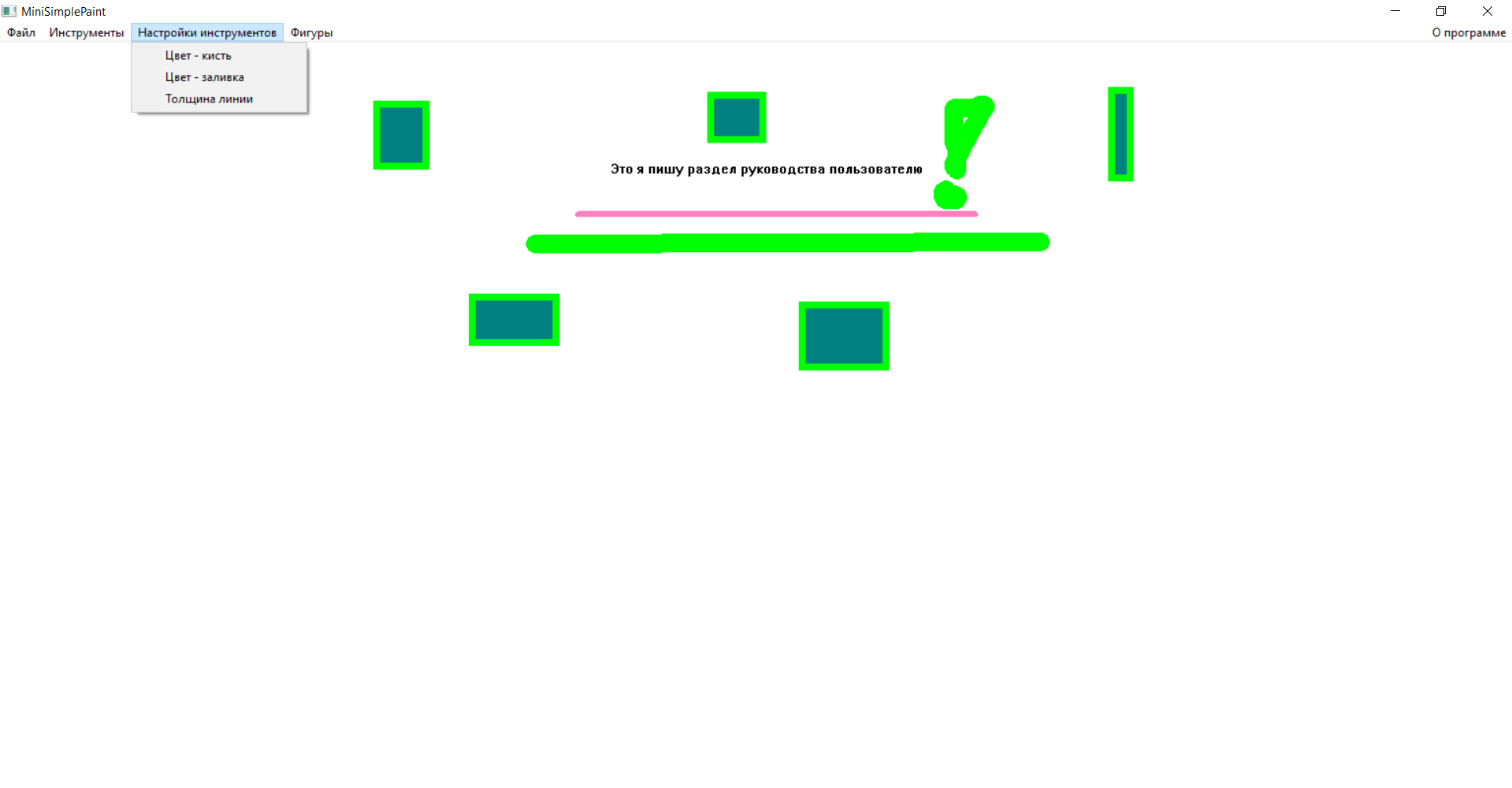


Рисунок 5.2 – Окно программы после использования инструментов

## Инструкции по управлению

Панели инструментов находятся в верхней части окна и реализованы с помощью выпадающего при нажатии меню. Вкладки «Инструменты» и «Фигуры» содержат средства для создания простых графических форм, вкладка «Настройка инструментов» позволяет изменять толщину линий, цвет линий и заливки. Вкладка «Файл» даёт доступ с открытию и сохранению файлом, включая возможость использования горячих клавиш.

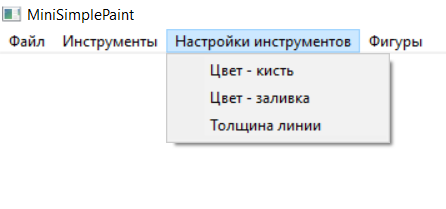


Рисунок 5.3 – Окно программы с открытой панелью настройки инструментов

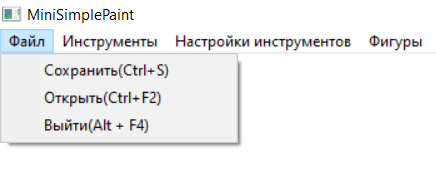


Рисунок 5.4 – Окно программы с открытой панелью работы с файлами

## Меню игры

В программе присутствует вкладка «О программе», содержащая сведения о годе создания программы и авторстве.

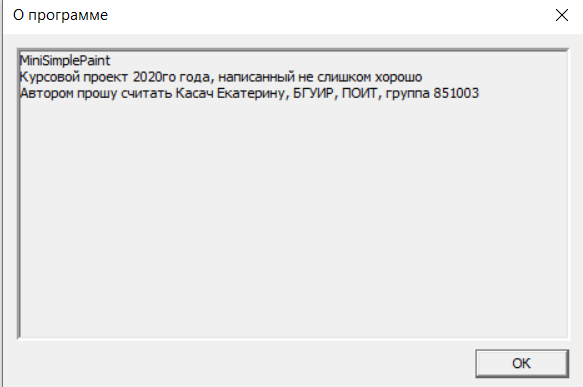


Рисунок 5.4 – Окно «О программе»

Заключение

В результате выполнения данной курсового проекта было сконструировано и разработано программное средство «MiniSimplePaint», обаладающее функционалом простого графического редактора. Разработанное игровое приложение в полной мере соответствует условиям, сформулированным в курсовом проекте, работает корректно и без сбоев.

В ходе разработки приложения были использованы набор функций WinApi и язык программирования C++ на базе IDE Visual Studio 2019.

Работа была разделена на этапы, такие, как анализ прототипов и постановка требований к проектируемому программному средству, разработка алгоритма и его схемы, конструирование программного средства, отладка и тестирование. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее программное средство.

Список использованных источников

[1] Щупак Ю. А. Win 32 API. Разработка приложений для Windows – СПб.: Питер, 2008.

[2] Microsoft C/C++ Documentation [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/en-us/cpp/?view=msvc-160> – Дата доступа: 13.012.2020.

[3] VisualStudio Documentation [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/en-us/visualstudio/?view=vs-2019> – Дата доступа: 13.12.2020.

[4] Win32 Api. Эффективная разработка приложений. / Ю. Щупак. – Питер, 2007.

[5] Создание эффективных WIN32-приложений с учетом специфики 64-разрядной версии Windows./ Джеффри Рихтер. – 4-е издание. – Питер, 2008.

Приложение А

(исходный код программы)

**DrawingProcess.cpp**

#include "DrawingProcess.h"

#include <Windows.h>

DrawingProcess::DrawingProcess(HWND hWnd)

{

this->hWnd = hWnd;

origin.x = 0;

origin.y = 0;

prevOrigin.x = 0;

prevOrigin.y = 0;

workingMode = DRAWING;

zoom = 1;

loadedFile = NULL;

currentDrawingObjectType = MULTI\_LINE;

drawingObjectFactory = new MultiLineObjectFactory();

}

DrawingProcess::~DrawingProcess()

{

int shapesCount = shapes.size();

for (int i = 0; i < shapesCount; i++)

{

delete(shapes[shapesCount-1]);

shapes.pop\_back();

shapesCount--;

}

DeleteEnhMetaFile(loadedFile);

}

void DrawingProcess::setDrawingPen(HPEN hPen)

{

this->drawingPen = hPen;

}

void DrawingProcess::setDrawingBrush(HBRUSH hBrush)

{

this->drawingBrush = hBrush;

}

void DrawingProcess::setCurrentDrawingObjectsType(DRAWING\_OBJECTS currentDrawingObjectType)

{

if (currentDrawingObjectType != this>currentDrawingObjectType)

{

this->currentDrawingObjectType = currentDrawingObjectType;

setCurrentDrawingObjectFactory(currentDrawingObjectType);

}

}

DRAWING\_OBJECTS DrawingProcess::getCurrentDrawingObjectsType()

{

return currentDrawingObjectType;

}

float DrawingProcess::getZoom()

{

return zoom;

}

void DrawingProcess::setCurrentDrawingObjectFactory(DRAWING\_OBJECTS drawingObjectType)

{

delete(drawingObjectFactory);

switch (drawingObjectType)

{

case LINE:

{

drawingObjectFactory = new LineObjectFactory();

}

break;

case BROKEN\_LINE:

{

drawingObjectFactory = new BrokenLineObjectFactory();

}

break;

case MULTI\_LINE:

{

drawingObjectFactory = new MultiLineObjectFactory();

}

break;

case RECTANGLE:

{

drawingObjectFactory = new RectangleObjectFactory();

}

break;

case POLYGON:

{

drawingObjectFactory = new PolygonObjectFactory();

}

break;

case ELLIPSE:

{

drawingObjectFactory = new EllipseObjectFactory();

}

break;

case TEXT:

{

drawingObjectFactory = new TextObjectFactory();

}

break;

default:

break;

}

}

DrawingObject\* DrawingProcess::popLastDrawingObject()

{

currentShape = NULL;

int shapesCount = shapes.size();

if (shapesCount != 0)

{

DrawingObject\* lastObject = shapes[shapesCount - 1];

shapes.pop\_back();

return lastObject;

}

return NULL;

}

void DrawingProcess::redrawAllDrawnShapes(HDC hdc)

{

int shapesCount = shapes.size();

for (int i = 0; i < shapesCount; i++)

{

shapes[i]->draw(hdc, false);

}

}

void DrawingProcess::stopDrawingCurrentShape()

{

if (currentShape->isReadyToDraw())

{

shapes.push\_back(currentShape);

}

InvalidateRect(hWnd, NULL, true);

isDrawingNow = false;

}

DRAWING\_TYPE DrawingProcess::getDrawingType()

{

return currentShape->getDrawingType();

}

void DrawingProcess::clearShapesList()

{

int shapesCount = shapes.size();

DrawingObject \*drawingObj;

for (int i = 0; i < shapesCount; i++)

{

drawingObj = popLastDrawingObject();

delete(drawingObj);

}

}

void DrawingProcess::setLoadedFile(HENHMETAFILE file)

{

DeleteEnhMetaFile(loadedFile);

this->loadedFile = file;

int shapesCount = shapes.size();

if (shapesCount != 0)

{

clearShapesList();

}

}

void DrawingProcess::setOriginTo(POINT origin)

{

prevOrigin = this->origin;

this->origin = origin;

}

POINT DrawingProcess::getDrawingOrigin()

{

return origin;

}

void DrawingProcess::startOrContinueDrawingShape(POINT currentLocation, HPEN hPen, HBRUSH hBrush)

{

currentLocation.x = currentLocation.x / zoom;

currentLocation.y = currentLocation.y / zoom;

if (isDrawingNow == false)

{

if (workingMode == SELECTING\_AREA)

{

this->currentShape = drawingObjectFactory->newDrawingObject(hPen, (HBRUSH)GetStockObject(HOLLOW\_BRUSH));

}

else

{

this->currentShape = drawingObjectFactory->newDrawingObject(hPen, hBrush);

}

isDrawingNow = true;

}

currentShape->addNewOrChangeLastCoordinate(currentLocation);

if (currentShape->isReadyToDraw() && currentShape->hasLimitedCoordinatesCount())

{

shapes.push\_back(currentShape);

isDrawingNow = false;

}

}

void DrawingProcess::recalcCoordinates()

{

int shapesCount = shapes.size();

int deltaX = origin.x - prevOrigin.x;

int deltaY = origin.y - prevOrigin.y;

for (int i = 0; i < shapesCount; i++)

{

shapes[i]->recalculateCoordinates(deltaX, deltaY);

}

if (shapesCount != 0)

{

if ((shapes[shapesCount - 1] != currentShape) && (currentShape != NULL))

{

currentShape->recalculateCoordinates(deltaX, deltaY);

}

}

}

void DrawingProcess::drawToFile(HDC fileContext)

{

RECT windowRect;

GetClientRect(hWnd, &windowRect);

FillRect(fileContext, &windowRect, CreateSolidBrush(RGB(255, 255, 255)));

redrawAllDrawnShapes(fileContext);

}

void DrawingProcess::drawToFile(HDC fileContext, RECT rectToDraw)

{

FillRect(fileContext, &rectToDraw, CreateSolidBrush(RGB(255, 255, 255)));

redrawAllDrawnShapes(fileContext);

}

void DrawingProcess::drawToHDC(HDC hdc, POINT currentLocation)

{

redrawAllDrawnShapes(hdc);

if (isDrawingNow == true)

{

if (currentShape->hasStaticCoordinates() == false)

{

if (currentShape->getDrawingType() == SEPARATED)

{

currentShape->changeLastOrAddNewCoordinate(currentLocation);

}

else

{

currentShape->addNewOrChangeLastCoordinate(currentLocation);

}

}

currentShape->draw(hdc, true);

}

}

void DrawingProcess::zoomIn(float zoom)

{

if (((this->zoom \* zoom) <= 64) && ((this->zoom \* zoom) >= 0.25))

{

this->zoom \*= zoom;

InvalidateRect(hWnd, NULL, true);

}

}

void DrawingProcess::drawing(POINT currentLocation)

{

currentLocation.x = currentLocation.x / zoom;

currentLocation.y = currentLocation.y / zoom;

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

//init buffer

RECT windowRect;

GetClientRect(hWnd, &windowRect);

//

windowRect.right = windowRect.right / zoom;

windowRect.bottom = windowRect.bottom / zoom;

//

int windowWidth = windowRect.right - windowRect.left;//!

int windowHeight = windowRect.bottom - windowRect.top;//!

HDC mdc = CreateCompatibleDC(hdc);

HBITMAP mbmp = CreateCompatibleBitmap(hdc, windowWidth, windowHeight);

SelectObject(mdc, mbmp);

FillRect(mdc, &windowRect, CreateSolidBrush(RGB(255, 255, 255)));

//drawing logic

//redrawing loaded file

if (loadedFile != NULL)

{

windowRect.right \*= zoom;

windowRect.bottom \*= zoom;

windowRect.left += origin.x;

windowRect.right += origin.x;

windowRect.bottom += origin.y;

windowRect.top += origin.y;

PlayEnhMetaFile(mdc, loadedFile, &windowRect);

}

//recalc

if (workingMode == MOVING)

{

recalcCoordinates();

}

//drawing

drawToHDC(mdc, currentLocation);

//zoom logic

if (zoom != 1)

{

XFORM xForm;

xForm.eM11 = xForm.eM22 = zoom;

xForm.eM12 = xForm.eM21 = xForm.eDx = xForm.eDy = 0;

SetGraphicsMode(hdc, GM\_ADVANCED);

SetWorldTransform(hdc, &xForm);

}

//copying image from buffer to screen

BitBlt(hdc, 0, 0, windowWidth, windowHeight, mdc, 0, 0, SRCCOPY);

//free resources

DeleteObject(mbmp);

DeleteDC(mdc);

DeleteDC(hdc);

EndPaint(hWnd, &ps);

}

void DrawingProcess::drawingText(DWORD character)

{

if (character != 8)

{

currentShape->addDataCell(character);

}

else

{

currentShape->removeDataCell();

}

}

bool DrawingProcess::isDrawing()

{

return isDrawingNow;

}

void DrawingProcess::switchToPrevWorkingMode()

{

int workingModesHistorySize = workingModesHistory.size();

if (workingModesHistorySize > 0)

{

WORKING\_MODE prevWorkingMode = workingModesHistory[workingModesHistorySize - 1];

workingMode = prevWorkingMode;

workingModesHistory.pop\_back();

}

}

void DrawingProcess::setWorkingMode(WORKING\_MODE workingMode)

{

if (workingModesHistory.size() == 3)

{

workingModesHistory.pop\_back();

}

workingModesHistory.push\_back(this->workingMode);

this->workingMode = workingMode;

}

WORKING\_MODE DrawingProcess::getWorkingMode()

{

return workingMode;

}

**DrawingObject.cpp**

#include "DrawingObject.h"

DrawingObject::DrawingObject(HPEN hPen, HBRUSH hBrush)

{

this->hPen = hPen;

this->hBrush = hBrush;

}

DrawingObject::~DrawingObject()

{

DeleteObject(hPen);

DeleteObject(hBrush);

}

DRAWING\_TYPE DrawingObject::getDrawingType()

{

return drawingType;

}

bool DrawingObject::isReadyToDraw()

{

return readyToDraw;

}

bool DrawingObject::hasLimitedCoordinatesCount()

{

return limitedCoordinatesCount;

}

bool DrawingObject::hasStaticCoordinates()

{

return staticCoordinates;

}

void DrawingObject::setDrawingConditions(int minCoordinatesCount, int maxCoordinatesCount, bool limitedCoordinatesCount, DRAWING\_TYPE drawingType, bool staticCoordinates)

{

this->minCoordinatesCount = minCoordinatesCount;

this->maxCoordinatesCount = maxCoordinatesCount;

this->limitedCoordinatesCount = limitedCoordinatesCount;

this->drawingType = drawingType;

this->staticCoordinates = staticCoordinates;

}

void DrawingObject::recalculateCoordinates(int deltaX, int deltaY)

{

int coordinatesCount = coordinates.size();

for (int i = 0; i < coordinatesCount; i++)

{

coordinates[i].x += deltaX;

coordinates[i].y += deltaY;

}

}

void DrawingObject::addDataCell(DWORD data)

{

objectData += (TCHAR)data;

}

void DrawingObject::removeDataCell()

{

if (objectData.size() != 0)

{

objectData.pop\_back();

}

}

void DrawingObject::addNewOrChangeLastCoordinate(POINT newPoint)

{

if (readyToDraw && limitedCoordinatesCount)

{

coordinates.pop\_back();

coordinates.push\_back(newPoint);

}

else

{

coordinates.push\_back(newPoint);

if (readyToDraw == false)

{

if (limitedCoordinatesCount)

{

if (coordinates.size() == this->maxCoordinatesCount)

{

readyToDraw = true;

}

}

else

{

if (coordinates.size() >= this->minCoordinatesCount)

{

readyToDraw = true;

}

}

}

}

}

vector<POINT> DrawingObject::getCoordinates()

{

return coordinates;

}

void DrawingObject::changeLastOrAddNewCoordinate(POINT newPoint)

{

if (coordinates.size() > 1)

{

coordinates[coordinates.size() - 1] = newPoint;

}

else

{

addNewOrChangeLastCoordinate(newPoint);

}

}

**Main.cpp**

#define \_USE\_MATH\_DEFINES

#define STATUS\_BAR\_PARTS\_COUNT 4

#include <windows.h>

#include <tchar.h>

#include <xstring>

#include "resource.h"

#include <fstream>

#include <math.h>

#include <CommCtrl.h>

#include "DrawingProcess.h"

#include "RectangleObject.h"

#define FILE\_NOT\_SAVED\_TEXT \_T("Предупреждение: файл не сохранён.\nВыберите ваше действие.")

typedef std::basic\_string<TCHAR, std::char\_traits<TCHAR>, std::allocator<TCHAR>> String;

//global variables

TCHAR WinName[] = \_T("WinFrame");

BOOL sessionSaved = false;

OPENFILENAMEW openFileName;

CHOOSECOLOR chooseColor;

int penColor;

int brushColor;

int textColor;

int penThickness;

HPEN hPen;

HPEN hTextPen;

HBRUSH hBrush;

HWND aboutDialog = NULL;

HWND lineThicknessDialog = NULL;

HWND exitDialog = NULL;

HWND statusBar = NULL;

DrawingProcess \*drawingProcess;

DRAWING\_OBJECTS remShapeType; //type of object

HCURSOR crossCursor;

HCURSOR handCursor;

bool zoom = false;

//functions and procedures signatures

ATOM RegisterWindowClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProcMessages(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK AboutDialogProc(\_In\_ HWND hwndDlg, \_In\_ UINT uMsg, \_In\_ WPARAM wParam, \_In\_ LPARAM lParam);

INT\_PTR CALLBACK LineThicknessDialogProc(\_In\_ HWND hwndDlg, \_In\_ UINT uMsg, \_In\_ WPARAM wParam, \_In\_ LPARAM lParam);

INT\_PTR CALLBACK ExitDialogProc(\_In\_ HWND hwndDlg, \_In\_ UINT uMsg, \_In\_ WPARAM wParam, \_In\_ LPARAM lParam);

void ExitApplication(HWND hWnd);

HWND CreateTrackbar(HWND, UINT, UINT, UINT, UINT, UINT, UINT);

HWND CreateStatusBar(HWND, int, HINSTANCE, int);

int ShowWarning(LPCWSTR lpText, LPCWSTR lpCaption);

BOOL GetSaveFileName(LPWSTR \*fileName);

BOOL GetLoadFileName(LPWSTR \*fileName);

LPWSTR SaveFileDialog();

void SaveFile(HWND hWnd, LPWSTR fileName);

LPWSTR LoadFileDialog();

void LoadFile(LPWSTR fileName);

void InitOpenFileNameStructure(HWND hWnd);

void InitChooseColorStructure(HWND hWnd);

int ChooseColorProc();

void CreateDialogs(HWND hWnd);

void InitResources(HWND hWnd);

void FreeResources(HWND hWnd);

//handlers

void MenuClick(HWND hWnd, WORD menuItemID);

void LeftButtonDown(HWND hWnd, POINT mousePos);

void LeftButtonUp(HWND hWnd, POINT mousePos);

void MouseMove(HWND hWnd, POINT prevMousePos, POINT currentMousePos);

void MouseWheel(HWND hWnd, WPARAM wParam);

//void PrintSelectedRectToFile(HWND hWnd, LPWSTR fileName, RectangleObject \*selectedRect);

void ClosingApplication(HWND hWnd);

//application start point

int APIENTRY \_tWinMain(HINSTANCE This, HINSTANCE prev, LPTSTR cmd, int mode)

{

HACCEL hAccelTable;

MSG msg;

crossCursor = LoadCursor(NULL, IDC\_CROSS);

handCursor = LoadCursor(NULL, IDC\_HAND);

RegisterWindowClass(This);

InitInstance(This, mode);

SetCursor(crossCursor);

hAccelTable = LoadAccelerators(This, MAKEINTRESOURCE(IDR\_ACCELERATOR));

//main window messages processing cicle

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

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return 0;

}

//init and register main window class

ATOM RegisterWindowClass(HINSTANCE hInstance)

{

WNDCLASS wc;

wc.hInstance = hInstance;

wc.lpszClassName = WinName;

wc.lpfnWndProc = WndProcMessages;

wc.style = CS\_HREDRAW | CS\_VREDRAW;

wc.hIcon = LoadIcon(NULL, IDI\_WINLOGO);

wc.hCursor = NULL;

wc.lpszMenuName = NULL;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = NULL;

return RegisterClass(&wc);

}

//creating and initialization main window

BOOL InitInstance(HINSTANCE hInstance, int mode)

{

HWND hWnd;

hWnd = CreateWindow

(

WinName,

\_T("MiniSimplePaint"),

WS\_OVERLAPPED | WS\_CAPTION | WS\_SYSMENU | WS\_THICKFRAME | WS\_MINIMIZEBOX | WS\_MAXIMIZEBOX,

CW\_USEDEFAULT,

CW\_USEDEFAULT,

800,

600,

HWND\_DESKTOP,

NULL,

hInstance,

NULL

);

if (!hWnd)

{

return false;

}

HMENU hMenu = LoadMenu(NULL, MAKEINTRESOURCE(IDR\_MENU));

SetMenu(hWnd, hMenu);

statusBar = CreateStatusBar(hWnd, NULL, hInstance, 10);

ShowWindow(hWnd, SW\_MAXIMIZE);

UpdateWindow(hWnd);

HDC hdc = GetDC(hWnd);

SetGraphicsMode(hdc, GM\_ADVANCED);

SetMapMode(hdc, MM\_LOENGLISH);

ReleaseDC(hWnd, hdc);

return true;

}

//main window messages processer

LRESULT CALLBACK WndProcMessages(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

WORD wParamLowWord = LOWORD(wParam);

WORD wParamHighWord = HIWORD(wParam);

WORD lParamLowWord = LOWORD(lParam);

WORD lParamHighWord = HIWORD(lParam);

PAINTSTRUCT ps;

static POINT mousePos;

switch (message)

{

//Window creating

case WM\_CREATE:

{

InitResources(hWnd);

}

break;

//mouse events

case WM\_RBUTTONDOWN://right mouse button down

{

if (drawingProcess->isDrawing())

{

drawingProcess->stopDrawingCurrentShape();

sessionSaved = false;

}

}

break;

case WM\_LBUTTONDOWN://left mouse button down

case WM\_LBUTTONUP:

{

mousePos.x = lParamLowWord;

mousePos.y = lParamHighWord;

if ((GetKeyState(VK\_LBUTTON) & 0x100) != 0)

{

LeftButtonDown(hWnd, mousePos);

}

else

{

LeftButtonUp(hWnd, mousePos);

}

}

break;

case WM\_MOUSEMOVE://mouse move

{

POINT currentMousePos;

currentMousePos.x = lParamLowWord;

currentMousePos.y = lParamHighWord;

MouseMove(hWnd, mousePos, currentMousePos);

mousePos = currentMousePos;

}

break;

case WM\_MBUTTONDOWN:

{

mousePos.x = lParamLowWord;

mousePos.y = lParamHighWord;

SetCursor(handCursor);

drawingProcess->setWorkingMode(MOVING);

}

break;

case WM\_MBUTTONUP:

{

SetCursor(crossCursor);

drawingProcess->switchToPrevWorkingMode();

sessionSaved = false;

}

break;

case WM\_MOUSEWHEEL:

{

MouseWheel(hWnd, wParam);

}

break;

//window repaint

case WM\_PAINT:

{

GetCursorPos(&mousePos);

ScreenToClient(hWnd, &mousePos);

drawingProcess->drawing(mousePos);

}

break;

//keyboard click

case WM\_CHAR:

{

if (drawingProcess->isDrawing() && (drawingProcess->getCurrentDrawingObjectsType() == TEXT))

{

if (wParam != 13)

{

drawingProcess->drawingText(wParam);

InvalidateRect(hWnd, NULL, true);

}

else

{

drawingProcess->stopDrawingCurrentShape();

}

}

}

break;

case WM\_SIZE:

{

int\* rightEdgeXCoordinates = (int\*)malloc(sizeof(int) \* STATUS\_BAR\_PARTS\_COUNT);

RECT windowRect;

GetClientRect(hWnd, &windowRect);

int statusBarPartWidth = (windowRect.right - windowRect.left) / STATUS\_BAR\_PARTS\_COUNT;

int prevRightEdgeXCoordinate = windowRect.left;

for (int i = 0; i < STATUS\_BAR\_PARTS\_COUNT; i++)

{

rightEdgeXCoordinates[i] = prevRightEdgeXCoordinate + statusBarPartWidth;

prevRightEdgeXCoordinate = rightEdgeXCoordinates[i];

}

SendMessage(statusBar, SB\_SETPARTS, (WPARAM)STATUS\_BAR\_PARTS\_COUNT, (LPARAM)rightEdgeXCoordinates);

SetWindowPos(statusBar, NULL, 0, 0, 0, 0, SWP\_SHOWWINDOW);

}

break;

//menu events

case WM\_COMMAND:

{

drawingProcess->setWorkingMode(DRAWING);

MenuClick(hWnd, wParamLowWord);

}

break;

case WM\_CLOSE:

{

ClosingApplication(hWnd);

}

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

void LeftButtonDown(HWND hWnd, POINT mousePos)

{

switch (drawingProcess->getWorkingMode())

{

case DRAWING:

{

drawingProcess->startOrContinueDrawingShape(mousePos, hPen, hBrush);

if (drawingProcess->isDrawing() == false)

{

sessionSaved = false;

}

}

break;

default:

break;

}

}

void LeftButtonUp(HWND hWnd, POINT mousePos)

{

switch (drawingProcess->getWorkingMode())

{

case DRAWING:

{

if (drawingProcess->isDrawing() && (drawingProcess->getDrawingType() == UNSEPARATED))

{

drawingProcess->stopDrawingCurrentShape();

}

}

break;

default:

break;

}

}

void MouseMove(HWND hWnd, POINT prevMousePos, POINT currentMousePos)

{

if (drawingProcess->isDrawing())

{

InvalidateRect(hWnd, NULL, true);

}

else

{

if (drawingProcess->getWorkingMode() == MOVING)

{

POINT prevOrigin = drawingProcess->getDrawingOrigin();

POINT newOrigin;

newOrigin.x = prevOrigin.x + (currentMousePos.x - prevMousePos.x);

newOrigin.y = prevOrigin.y + (currentMousePos.y - prevMousePos.y);

drawingProcess->setOriginTo(newOrigin);

InvalidateRect(hWnd, NULL, true);

}

}

}

void MouseWheel(HWND hWnd, WPARAM wParam)

{

WORD wParamLowWord = LOWORD(wParam);

if (wParamLowWord == MK\_CONTROL)

{

WORD wheelDestiny = HIWORD(wParam);

if (wheelDestiny > 65000)

{

drawingProcess->zoomIn(0.5);

}

else

{

drawingProcess->zoomIn(2);

}

}

}

void InitOpenFileNameStructure(HWND hWnd)

{

openFileName.lStructSize = sizeof(OPENFILENAME);

openFileName.hwndOwner = hWnd;

openFileName.lpstrFilter = \_T("Bitmap files(\*.bmp)\0\*.bmp\0JPEG files(\*.jpg)\0\*.jpg\0All files(\*.\*)\0\*.\*\0\0");

openFileName.lpstrFile = new TCHAR[256];

openFileName.lpstrFile[0] = '\0';

openFileName.nMaxFile = 256;

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

openFileName.lpstrDefExt = \_T("bmp");

}

void InitChooseColorStructure(HWND hWnd)

{

static COLORREF arCustClr[16];

static DWORD rgbCurrent;

ZeroMemory(&chooseColor, sizeof(CHOOSECOLOR));

chooseColor.lStructSize = sizeof(CHOOSECOLOR);

chooseColor.hwndOwner = hWnd;

chooseColor.lpCustColors = (LPDWORD)arCustClr;

chooseColor.rgbResult = rgbCurrent;

chooseColor.Flags = CC\_RGBINIT | CC\_PREVENTFULLOPEN;

}

//standard choose color dialog

int ChooseColorProc()

{

if (ChooseColor(&chooseColor) == true)

{

return chooseColor.rgbResult;

}

}

//create dialog windows

void CreateDialogs(HWND hWnd)

{

aboutDialog = CreateDialog(NULL, MAKEINTRESOURCE(IDD\_ABOUT\_DIALOG), hWnd, AboutDialogProc);

lineThicknessDialog = CreateDialog(NULL, MAKEINTRESOURCE(IDD\_LINE\_THICKNESS\_DIALOG), hWnd, LineThicknessDialogProc);

exitDialog = CreateDialog(NULL, MAKEINTRESOURCE(IDD\_EXIT\_DIALOG), hWnd, ExitDialogProc);

}

//initalization of all application resources

void InitResources(HWND hWnd)

{

InitOpenFileNameStructure(hWnd);

InitChooseColorStructure(hWnd);

CreateDialogs(hWnd);

drawingProcess = new DrawingProcess(hWnd);

penThickness = 0;

penColor = RGB(0, 0, 0);

brushColor = RGB(255, 255, 255);

hPen = CreatePen(PS\_SOLID, penThickness, penColor);

hBrush = CreateSolidBrush(brushColor);

}

//release all application resources

void FreeResources(HWND hWnd)

{

delete(drawingProcess);

free(openFileName.lpstrFile);

DestroyWindow(aboutDialog);

DestroyWindow(lineThicknessDialog);

DestroyWindow(exitDialog);

}

HWND CreateTrackbar(

HWND hWnd,

UINT minValue,

UINT maxValue,

UINT width,

UINT height,

UINT x,

UINT y

)

{

HWND trackBar;

//here is creating trackbar component

trackBar = CreateWindowEx(

0,

TRACKBAR\_CLASS,

\_T("TrackBar Control"),

WS\_CHILD | WS\_VISIBLE | TBS\_AUTOTICKS | TBS\_BOTTOM | TBS\_HORZ | TBS\_TOOLTIPS,

x,

y,

width,

height,

hWnd,

NULL,

NULL,

NULL

);

//setting trackbar values range

SendMessage(trackBar, TBM\_SETRANGE, (WPARAM)true, (LPARAM)MAKELONG(minValue, maxValue));

//setting the default number of logical positions the trackbar's slider moves

SendMessage(trackBar, TBM\_SETPAGESIZE, 0, (LPARAM)5);

SetFocus(trackBar);

return trackBar;

}

HWND CreateStatusBar(

HWND hWnd,

int idStatus,

HINSTANCE hinst,

int cParts

)

{

HWND statusBar;

statusBar = CreateWindowEx(

0,

STATUSCLASSNAME,

NULL,

WS\_CHILD | WS\_VISIBLE,

0, 0, 0, 0,

hWnd,

(HMENU)idStatus,

hinst,

NULL

);

int\* rightEdgeXCoordinates = (int\*)malloc(sizeof(int) \* cParts);

RECT windowRect;

GetClientRect(hWnd, &windowRect);

int statusBarPartWidth = (windowRect.right - windowRect.left) / cParts;

int prevRightEdgeXCoordinate = windowRect.left;

for (int i = 0; i < cParts; i++)

{

rightEdgeXCoordinates[i] = prevRightEdgeXCoordinate + statusBarPartWidth;

prevRightEdgeXCoordinate = rightEdgeXCoordinates[i];

}

SendMessage(statusBar, SB\_SETPARTS, (WPARAM)cParts, (LPARAM)rightEdgeXCoordinates);

return statusBar;

}

//model dialog about messages processer

INT\_PTR CALLBACK AboutDialogProc(

\_In\_ HWND hwndDlg,

\_In\_ UINT uMsg,

\_In\_ WPARAM wParam,

\_In\_ LPARAM lParam

)

{

switch (uMsg)

{

case WM\_COMMAND:

{

if (LOWORD(wParam) != IDOK)

{

break;

}

}

case WM\_CLOSE:

{

ShowWindow(hwndDlg, SW\_HIDE);

}

break;

default:

return false;

}

return true;

}

//model dialog line thickness messages processer

INT\_PTR CALLBACK LineThicknessDialogProc(

\_In\_ HWND hwndDlg,

\_In\_ UINT uMsg,

\_In\_ WPARAM wParam,

\_In\_ LPARAM lParam

)

{

static HWND trackbar;

switch (uMsg)

{

case WM\_HSCROLL:

break;

case WM\_INITDIALOG:

{

RECT rect;

GetWindowRect(hwndDlg, &rect);

trackbar = CreateTrackbar(hwndDlg, 1, 30, rect.right - rect.left - 40, rect.bottom - rect.top - 100, 10, 10);

}

break;

case WM\_COMMAND:

{

if (LOWORD(wParam) != IDOK)

{

break;

}

penThickness = SendMessage(trackbar, TBM\_GETPOS, 0, 0);

hPen = CreatePen(PS\_SOLID, penThickness, penColor);

}

case WM\_CLOSE:

{

ShowWindow(hwndDlg, SW\_HIDE);

}

break;

default:

return false;

}

return true;

}

//exit dialog window messages processer

INT\_PTR CALLBACK ExitDialogProc(

\_In\_ HWND hwndDlg,

\_In\_ UINT uMsg,

\_In\_ WPARAM wParam,

\_In\_ LPARAM lParam

)

{

bool exit = false;

HWND hWnd = GetParent(hwndDlg);

switch (uMsg)

{

case WM\_COMMAND:

{

switch (LOWORD(wParam))

{

case IDC\_EXIT:

{

exit = true;

}

break;

case IDC\_CANCEL:

{

ShowWindow(hwndDlg, SW\_HIDE);

return 0;

}

break;

case IDC\_SAVE:

{

ShowWindow(hwndDlg, SW\_HIDE);

LPWSTR fileName = SaveFileDialog();

SaveFile(hWnd, fileName);

}

break;

default:

break;

}

}

case WM\_CLOSE:

{

ShowWindow(hwndDlg, SW\_HIDE);

}

default:

break;

}

//exit

if (exit == true)

{

ExitApplication(hWnd);

}

}

//menu click processer

void MenuClick(HWND hWnd, WORD menuItemID)

{

switch (menuItemID)

{

case ID\_LINE:

{

drawingProcess->setCurrentDrawingObjectsType(LINE);

}

break;

case ID\_BROKEN\_LINE:

{

drawingProcess->setCurrentDrawingObjectsType(BROKEN\_LINE);

}

break;

case ID\_MULTILINE:

{

drawingProcess->setCurrentDrawingObjectsType(MULTI\_LINE);

}

break;

case ID\_RECTANGLE:

{

drawingProcess->setCurrentDrawingObjectsType(RECTANGLE);

}

break;

case ID\_POLYGON:

{

drawingProcess->setCurrentDrawingObjectsType(POLYGON);

}

break;

case ID\_ELLIPSE:

{

drawingProcess->setCurrentDrawingObjectsType(ELLIPSE);

}

break;

case ID\_TEXT:

{

drawingProcess->setCurrentDrawingObjectsType(TEXT);

}

break;

case ID\_LOAD:

{

LPWSTR fileName = NULL;

fileName = LoadFileDialog();

if (fileName != NULL)

{

LoadFile(fileName);

}

InvalidateRect(hWnd, NULL, true);

}

break;

case ID\_SAVE:

{

LPWSTR fileName = NULL;

fileName = SaveFileDialog();

if (fileName != NULL)

{

SaveFile(hWnd, fileName);

}

}

break;

case ID\_EXIT:

{

ClosingApplication(hWnd);

}

break;

case ID\_LINE\_THICKNESS:

{

ShowWindow(lineThicknessDialog, SW\_SHOW);

}

break;

case ID\_ABOUT:

{

ShowWindow(aboutDialog, SW\_SHOW);

}

break;

case ID\_PEN\_COLOR:

{

penColor = ChooseColorProc();

hPen = CreatePen(PS\_SOLID, penThickness, penColor);

}

break;

case ID\_BRUSH\_COLOR:

{

brushColor = ChooseColorProc();

hBrush = CreateSolidBrush(brushColor);

}

break;

/\*case ID\_PRINT:

{

StartPrinting();

}\*/

break;

default:

break;

}

}

BOOL GetSaveFileName(LPWSTR \*fileName)

{

if (GetSaveFileNameW(&openFileName) == 0)

{

if (CommDlgExtendedError() == 0)

{

(\*fileName) = NULL;

return true;

}

return false;

}

(\*fileName) = openFileName.lpstrFile;

return true;

}

BOOL GetLoadFileName(LPWSTR \*fileName)

{

if (GetOpenFileNameW(&openFileName) == 0)

{

if (CommDlgExtendedError() == 0)

{

(\*fileName) = NULL;

return true;

}

return false;

}

(\*fileName) = openFileName.lpstrFile;

return true;

}

LPWSTR SaveFileDialog()

{

LPWSTR fileName = NULL;

while (!GetSaveFileName(&fileName))

{

int dialogResult = ShowWarning(\_T("Повторить попытку?"), \_T("Не удалось сохранить файл!"));

if (dialogResult == IDNO)

{

return NULL;

}

}

return fileName;

}

LPWSTR LoadFileDialog()

{

LPWSTR fileName = NULL;

while (!GetLoadFileName(&fileName))

{

int dialogResult = ShowWarning(\_T("Повторить попытку?"), \_T("Не удалось загрузить файл!"));

if (dialogResult == IDNO)

{

return NULL;

}

}

return fileName;

}

void LoadFile(LPWSTR fileName)

{

HENHMETAFILE hemf = GetEnhMetaFileW(fileName);

drawingProcess->setLoadedFile(hemf);

}

void SaveFile(HWND hWnd, LPWSTR fileName)

{

HENHMETAFILE hmf;

HDC hdc = GetDC(hWnd);

HDC hdcEMF;

hdcEMF = CreateEnhMetaFile(hdc, fileName, NULL, NULL);

drawingProcess->drawToFile(hdcEMF);

hmf = CloseEnhMetaFile(hdcEMF);

DeleteEnhMetaFile(hmf);

ReleaseDC(hWnd, hdc);

sessionSaved = true;

}

//warning dialog

int ShowWarning(LPCWSTR lpText, LPCWSTR lpCaption)

{

int dialogResult = MessageBox(NULL, lpText, lpCaption,

MB\_YESNO |

MB\_DEFBUTTON1 |

MB\_ICONWARNING |

MB\_DEFAULT\_DESKTOP\_ONLY

);

return dialogResult;

}

void ClosingApplication(HWND hWnd)

{

if (!sessionSaved /\*|| !sessionPrinted\*/)

{

HWND text = GetDlgItem(exitDialog, IDC\_EXIT\_STATIC);

if (!sessionSaved)

{

SetWindowText(text, FILE\_NOT\_SAVED\_TEXT);

}

ShowWindow(exitDialog, SW\_SHOW);

}

else

{

ExitApplication(hWnd);

}

}

void ExitApplication(HWND hWnd)

{

FreeResources(hWnd);

PostQuitMessage(0);

}