Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системное программирование (СП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ПРОГРАММНОЕ СРЕДСТВО «СВЕТОСКОП» ДЛЯ РИСОВАНИЯ ПОВЕРХ ЭКРАНА

БГУИР КП 1-40 01 01 106 ПЗ

Студент Гатльский Д. В.
Руководитель Деменковец Д. В.

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разработка программных средств для визуализации и аннотации является востребованной задачей, так как такие инструменты находят широкое применение в проектировании, обучении и профессиональной деятельности. В этом контексте важно создать эффективное приложение, которое позволит пользователям рисовать поверх экрана и сохранять свои изображения для дальнейшей работы.

Данная курсовая работа посвящена разработке программного средства "Светоскоп", предназначенного для создания временных и постоянных визуальных аннотаций на экране. Этот инструмент решает актуальные задачи, такие как дорисовка элементов поверх готовых проектов (например, чертежей), создание каркасов для будущих проектов (обведение силуэтов на изображениях), а также нанесение вспомогательных символов для облегчения работы с другими приложениями.

Цель настоящей курсовой работы — разработка программного средства "Светоскоп", которое предоставляет пользователям интуитивно понятный интерфейс для рисования на экране, отображая нарисованное поверх всех окон и сохраняя результаты в формате PNG. Программа обеспечит возможность рисовать, редактировать и сохранять графические элементы без прерывания основной работы на компьютере.

Основные функциональные возможности включают: Рисование поверх всех окон и приложений. Сохранение нарисованных элементов для дальнейшей работы. Экспорт изображений в формат PNG. Возможность использовать программу для создания временных пометок, аннотаций и визуального каркаса. Интеграцию в рабочий процесс для повышения производительности и удобства проектирования.

"Светоскоп" ориентирован на широкий круг пользователей: инженеров, дизайнеров, студентов и специалистов, работающих с графикой, предоставляя эффективное решение для визуальной аннотации и поддержки творческого процесса в различных областях — от проектирования до обучения и профессиональной работы.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Анализ литературных источников

Overlay-приложения представляют собой класс программного обеспечения, предназначенный для создания и отображения графических элементов поверх других окон и приложений операционной системы. Такие приложения активно используются в различных областях, включая проектирование, образование, создание презентаций, а также для выполнения повседневных задач. В основе многих overlay-приложений лежат технологии графического рендеринга и работы с оконными атрибутами операционных систем, таких как Windows API.

Одной из популярных технологий для создания оверлеев является использование расширенного стиля окна, который позволяет создавать прозрачные и полупрозрачные окна. Такие окна могут перекрывать другие приложения и оставаться видимыми без нарушения их функциональности.

Для обработки ввода в overlay-приложениях применяются функции Windows API, такие как глобальные хуки мыши и клавиатуры. Эти хуки позволяют перехватывать события ввода, даже если окно overlay-приложения не является активным. Однако установка таких хуков требует осторожности, так как неправильное использование может привести к конфликтам с другими приложениями или блокировке ввода пользователя.

Многие overlay-приложения используют формат PNG (Portable Network Graphics) для сохранения изображений. Формат PNG был разработан в 1996 году как замена GIF и поддерживает сжатие без потерь, а также прозрачность через альфа-канал. Это делает его идеальным для сохранения графических аннотаций и рисунков с высокой точностью и качеством. В отличие от формата JPEG, который применяет сжатие с потерями, PNG позволяет сохранять четкие линии и текст, что особенно важно для технических и образовательных иллюстраций.

Таким образом, overlay-приложения являются важным инструментом для пользователей, которым необходимо быстро и эффективно вносить визуальные правки и аннотации. Современные технологии графического рендеринга и обработки окон позволяют создавать гибкие решения для самых разных задач — от образовательных до профессиональных.

1.2 Обзор аналогов

Существует ряд программных средств, предназначенных для написания музыки, каждое из которых имеет свои особенности и преимущества. Проведем обзор нескольких популярных аналогов "Светоскоп".

1.2.1 «Epic Pen»

Epic Pen — это простое и удобное приложение для рисования поверх экрана. Оно позволяет делать заметки в реальном времени на любом окне, будь то рабочий стол, веб-страница или программа. Основные функции включают рисование линий, выделение областей, стирание и возможность делать скриншоты с аннотациями. Epic Pen поддерживает различные цвета и размеры кистей. Интерфейс приложения представлен на рисунке 1.1.

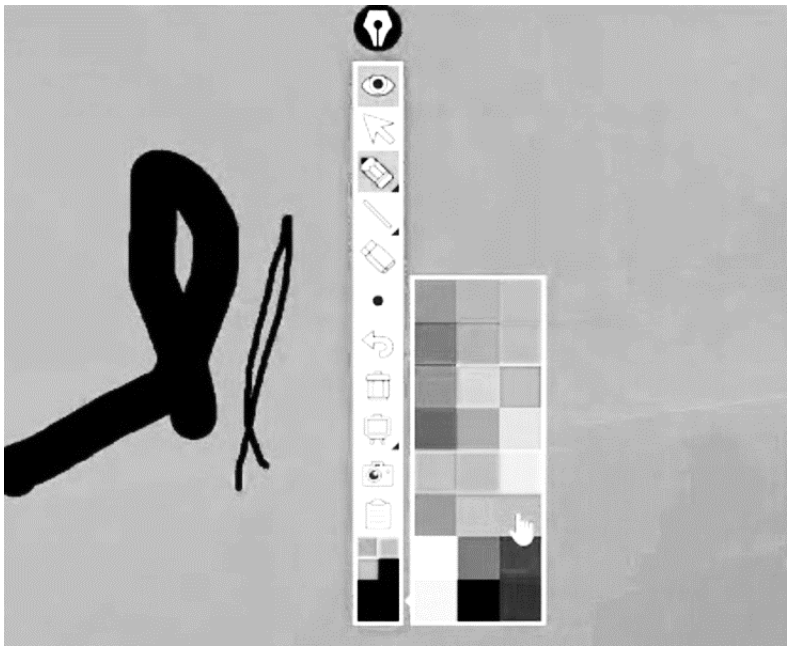

Рисунок 1.1 – Интерфейс Epic Pen

1.2.2 « Microsoft Snipping Tool / Snip & Sketch »

Microsoft Snipping Tool / Snip & Sketch — Snipping Tool (или его обновлённая версия Snip & Sketch) является встроенным приложением Windows для создания скриншотов с возможностью быстрого редактирования. Пользователь может делать скриншоты всего экрана, выбранного окна или произвольной области, а затем добавлять аннотации, такие как линии, текст и выделение. Интерфейс приложения представлен на рисунке 1.2.

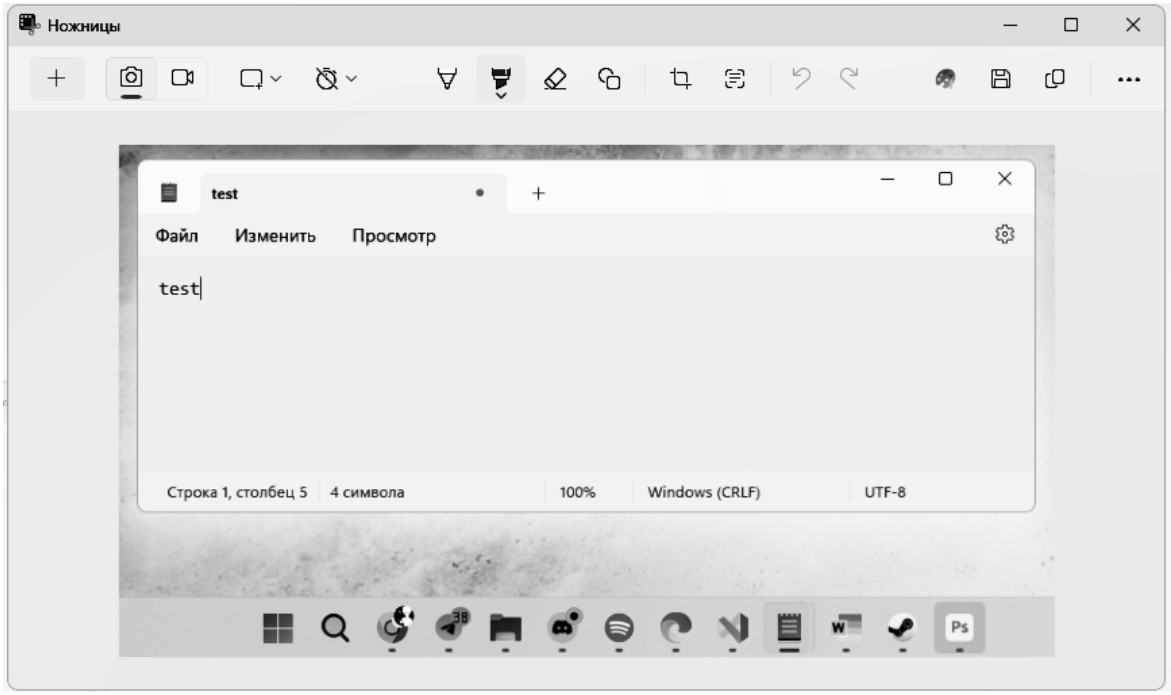

Рисунок 1.2 – Интерфейс Microsoft Snipping Tool / Snip & Sketch

1.2.3 «ZoomIt»

ZoomIt — приложение позволяет увеличивать определенные участки экрана и делать аннотации в реальном времени. Это полезный инструмент для презентаций и демонстраций, когда требуется акцентировать внимание на конкретных элементах.

В программе ZoomIt реализованы инструменты увеличения экрана, Возможность рисования поверх увеличенной области, Таймер для автоматических презентаций, Набор базовых инструментов для рисования. Интерфейс ZoomIt представлен на рисунке 1.3.

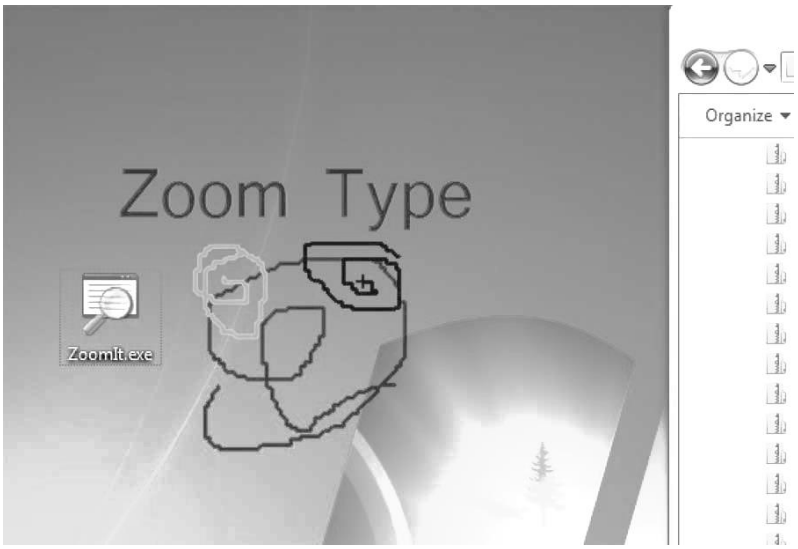

Рисунок 1.3 – Интерфейс NanoStudio

Следует отметить, что программное средство «Светоскоп» занимает свою нишу среди аналогичных решений, предлагая пользователям удобный и интуитивно понятный инструмент для создания аннотаций на экране. Основное внимание уделено функционалу рисования и сохранения изображений в формате PNG, а также возможности отображения нарисованных элементов поверх других окон даже после прекращения работы с системой. При этом включение сложных функций, характерных для профессиональных программ для экранных аннотаций, не рассматривалось как целесообразное на данном этапе разработки.

1.3 Постановка задачи

В рамках данной курсовой работы планируется разработать программное средство “Светоскоп” для рисования поверх экрана. Средство будет реализовано на языке C++ с использованием WinAPI и библиотеки для отрисовки “ DirectX 9”.

Основной целью данной работы является создание удобного и функционального инструмента для базового рисования поверх экрана, позволяющего пользователям быстро и просто создавать аннотации для дальнейшей работы.

Для реализации данной задачи необходимо выполнить следующие этапы:

– разработать интерфейс для взаимодействия с пользователем;

– реализовать функционал орисования на экране;

– обеспечить возможность изменения параметров рисования;

– реализовать возможность изменения параметров приложения;

– обеспечить сохранение созданных набросков в формате PNG с сохранением изображения позади наброска, и отдельно наброска.

Результатом работы должно стать готовое программное средство для создания набросков на экране, которое предоставит пользователю возможность рисовать аннотации поверх экрана, настраивать режим отображения и изменять параметры кисти, а также экспортировать наброски в удобном формате для дальнейшего использования.

.

2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Структура программы

При разработке программного средства будут использованы все стандарты разработки под платформу Windows.

2.2 Проектирование интерфейса программного средства

При разработке интерфейса музыкального секвенсора было принято решение использовать подходы, реализованные в программе ZoomIt, чтобы обеспечить удобство и интуитивность взаимодействия с пользователем. Главное окно программы будет служить центральным элементом управления всеми функциями, рисование кистью, настройку цвета, изменение размера кисти и сохранение результатов работы.

2.2.1 Главное окно программы

В связи с особенностями разрабатываемого рпограммного средства главное окно в основе своей будет прозрачным, однако, некоторая информация должна выводится на этом окне в виде текстовых сообщений. Пример представлен на рисунке 2.1.

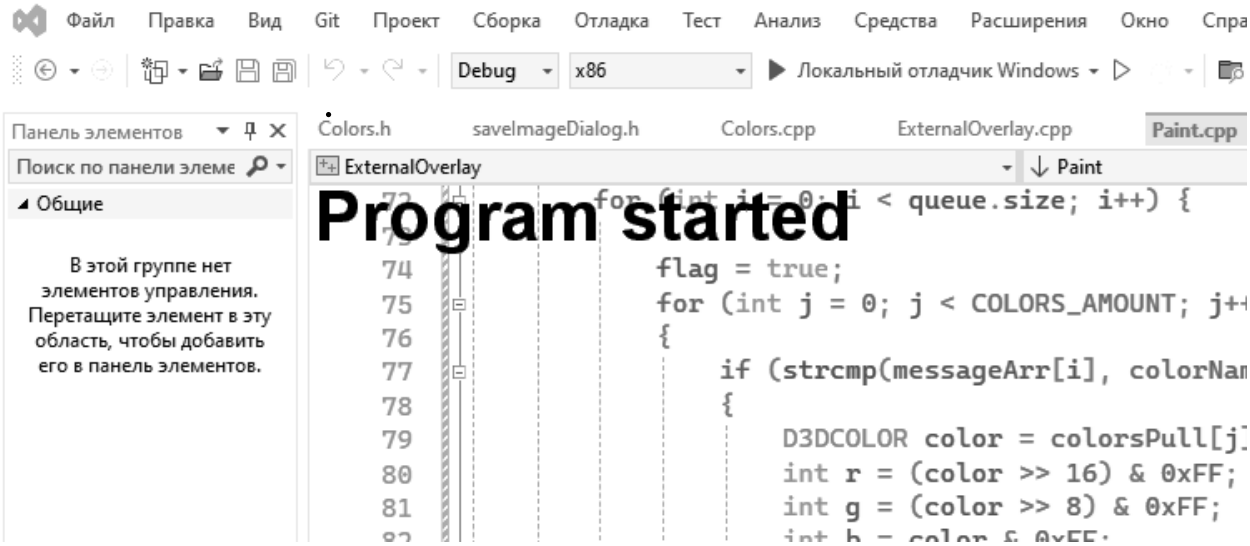

Рисунок 2.1 – Главное окно

2.2.2 Уведомления о настройке параметров отображения и работы

При изменении параметров приложения, например переход в режим рисования, изменения параметров кисти и другие, эти уведомления отображаются в левом верхнем углу экрана и отображаются на протяжении

заданного периода времени, после чего удаляются. Может быть отображено сразу несколько уведомлений. Если пользователь меняет цвет кисти то текст уведомления будет выведен новым цветом кисти. Эти уведомления позволяют пользователю получить информацию об успешности своих операций, и делают интерфейс программы отзывчивым. Пример представлен на рисунке 2.2.

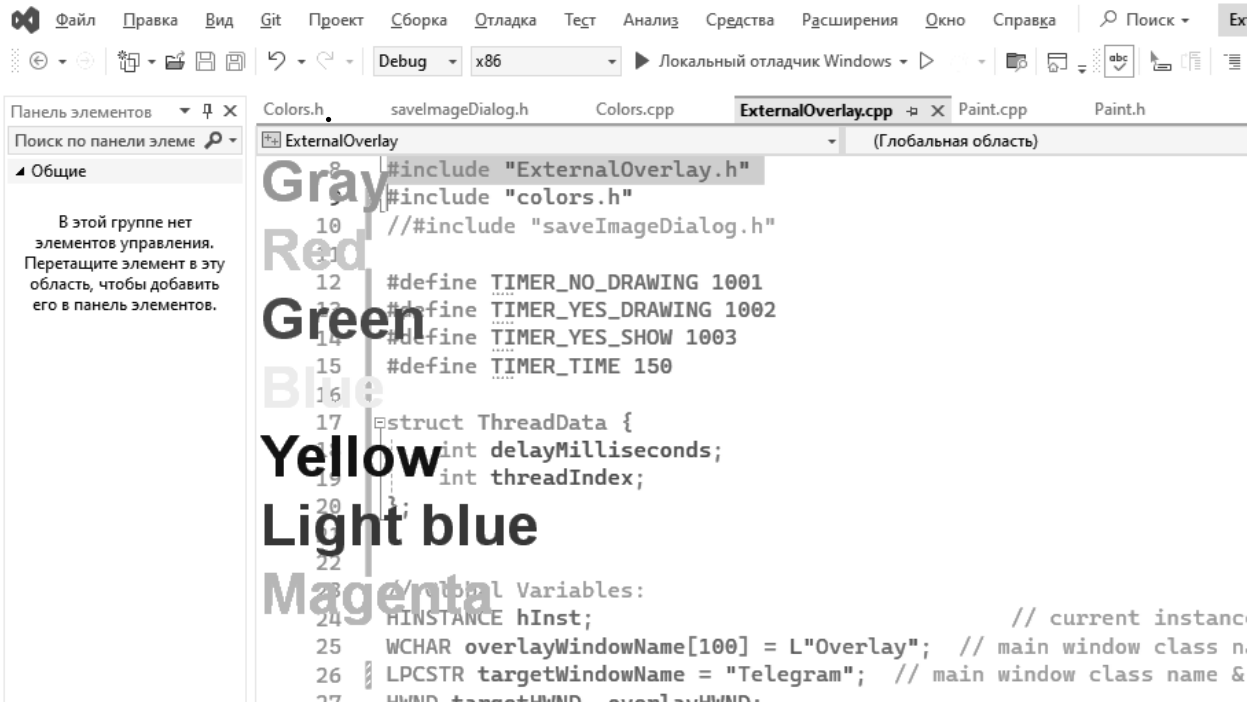

Рисунок 2.2 – Уведомления об изменении настроек

2.2.3 Сохранение наброска

При сохранении наброска, пользователю предоставляется выбор сохранить набросок отдельно от фона или вместе с фоном, однако из-за особенностей приложения затруднительно создать новое окно, в связи с чем окно выбора тоже рисуется поверх всего, после чего пользователь может выбрать опцию для сохранения. Такая реализация позволяет дать пользователю привычный интерфейс, даже при программных ограничениях.

2.3 Проектирование функционала программного средства

Программное средство «Светоскоп» для рисования поверх экрана э должно предоставлять пользователю такой минимальный функционал как:

- Отрисовка поверх других приложений;
- Обеспечение бесперебойной работы других приложений в системе;
- Изменение параметров кисти;
- Отрисовка уведомлений об изменении параметров;
- сохранение наброска в файле формата PNG с фоном и без него;

– завершение работы программы.

2.3.1 Отрисовка поверх других приложений

Для корректрой отрисовки и постоянной видимости оверлея, окно приложения должно быть всегда расположено поверх других окон в системе.

2.3.2 Обеспечение бесперебойной работы других приложений в системе

В связи с использованием перехвата сообщений от клавиатуры и мыши, а так же расположением окна приложения, приложение должно передавать сообщения о взаимодействии целевому окну и не передавать если сообщение отправлено оверлею. Для этого должна быть прописана логика обработки сообщений.

2.3.3 Изменение параметров кисти

Программа должна предусматривать возможность изменения параметров кисти. Для этого должна быть предусмотрена сочетания клавишь для изменения размера кисти и ее радиуса.

2.3.4 Отрисовка уведомлений об изменении параметров

В программе должны быть предусмотрены функции для отображения уведомлений, исчезающих со временем. Это обеспечит отзывчивость интерфейса для пользователя.

2.3.5 сохранение наброска в файле формата PNG с фоном и без него

Программа должна предоставлять возможность сохранить набросок в файл формата PNG. Для этого должны быть предусмотрены функции выбора способа сохранения и сохранения в файл.

2.3.6 Завершение работы программы

Программа должна поддерживать возможность корректного завершения работы с помощью сочетания клавишь закрывающего окно.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Обобщенный алгоритм работы ПС

Блок-схема обобщённого алгоритма работы программного средства представлена в Приложении А, Рисунок 1.

3.2 Разработка алгоритмов

3.2.1 Алгоритм отрисовки наброска

Для отрисовки изспользуются средства DirectX 9. Блок-схема алгоритма представлена на рисунке 3.1.

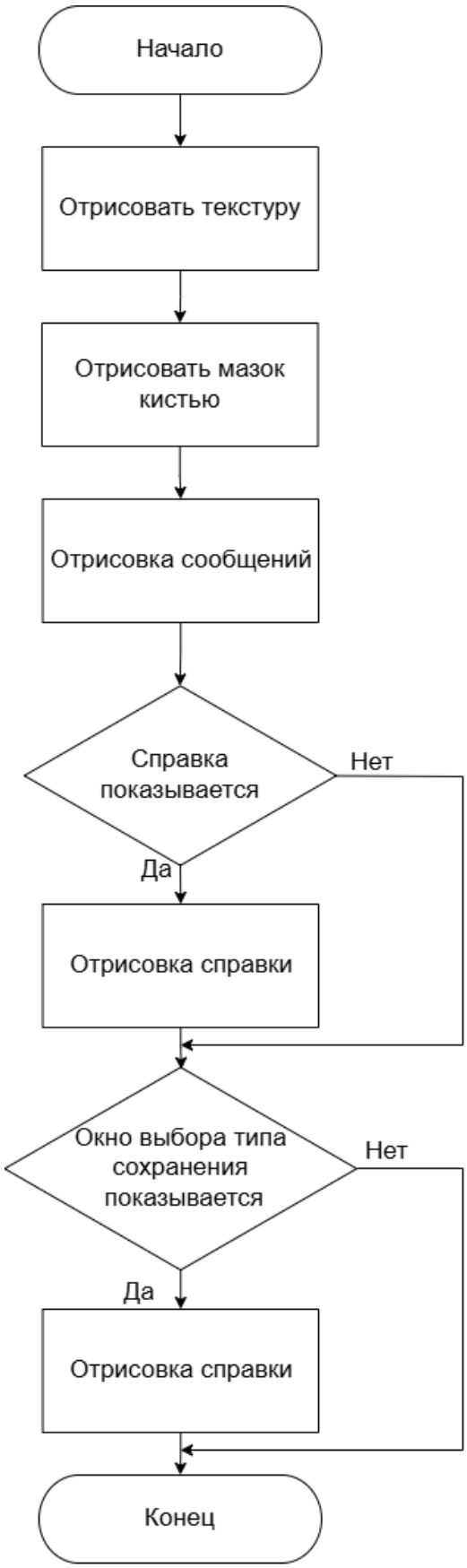

Рисунок 3.1 – Блок-схема алгоритма отрисовки наброска

3.2.2 Алгоритм Брезинхема

Для создания линий между точками, полученными от движения мыши необходимо дорисовывать точки при помощи алгоритма Брезинхема. Блок-схема алгоритма представлена на рисунке 3.2.

Рисунок 3.2 – Блок-схема алгоритма Брезинхема

3.2.3 Алгоритм сохранения наброска в файл без фона

Для сохранения мы сначала сохраняем последний мазок в текстуру, а потом сохраняем ее в файл. Блок-схема алгоритма представлена в рисунке 3.3.

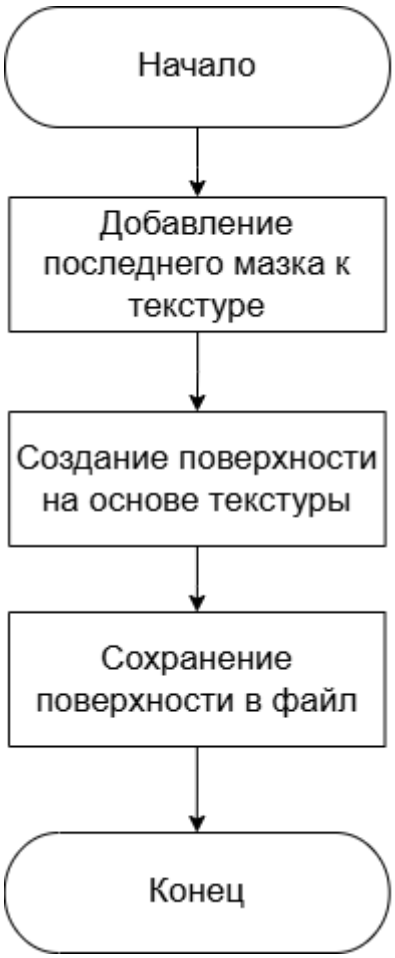

Рисунок 3.3 – Блок-схема алгоритма создания паттерна для ударных инструменов

3.2.4 Добавление уведомления

Для добавления уведомления необходимо записать его в очередь сообщений, а после запустить поток с таймером, который по окончании при помощи механизмов синхронизации безопасно удалит уведомление из очереди, это необходимо для поддрежки нескольких сообщений одновременно. Блок-схема алгоритма представлена на рисунке 3.4.

Рисунок 3.4 – Блок-схема алгоритма добавления уведомления

3.2.5 Завершение работы программы

Для корректного завершения работы программы необходимо освобождать используемые ресурсы. Блок-схема алгоритма представлена на рисунке 3.5.

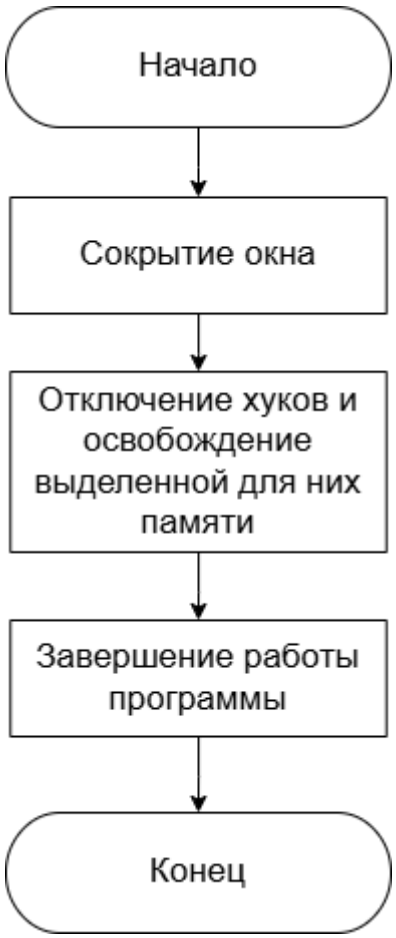

Рисунок 3.5 – Блок-схема алгоритма завершения работы программы

3.3 Работа с WinApi

Ключевой технологией при разработке данного программного средства является WinApi. Из него были использованный функции, позволяющие перехватывать сообщения клавиатуры и мыши, механизм создания потоков и их синхронизации, функции для получения изображения экрана. Также средствами WinApi реализуется основа приложения – создание окна которое находится поверх других и при этом является прозрачным, не препятсвующем работе других окон.

4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА

4.1 Проектирование основных методов программного средства

Основные методы, использованные в разработке, перечислены в таблицах 4.1 – 4.3.

Таблица 4.1 – Основные методы модуля Paint.h

Имя метода	Описание	Заголовок метода	Имя параметра	Назначение параметра
Paint	Конструктор для инициализации объекта Paint с указанием окна и размеров холста	Paint(HWND hWnd, int width, int height)	hWnd width height	Дескриптор окна Ширина холста. Высота холста.
render	Отображает содержимое холста	int render()	-	-
addBrushStroke	Добавляет мазок кисти на холст	void addBrushStroke(POINT point)	point	Координаты для мазка кисти
setBrushRadius	Устанавливает радиус кисти	void setBrushRadius(int radius)	radius	Новый радиус кисти
resetLastPoint	Сбрасывает последнюю точку мазка	void resetLastPoint()	-	-
clearCanvas	Очищает холст	void clearCanvas()	-	-
addMessage	Добавляет сообщение в очередь	void addMessage(const char* string)	string	Текст сообщения
stopMessage	Останавливает отображение сообщения	void stopMessage()	-	-
setBrushColor	Устанавливает цвет кисти	void setBrushColor(int ind)	ind	Индекс цвета кисти
changeEraser	Переключает режим между ластиком и кистью	void changeEraser()	-	-

Таблица 4.1 – Продолжение

saveScreenshotToFile	Сохраняет текущий набросок в файл с фоном	void saveScreenshotToFile(HBITMAP hBitmap, const std::string& filename)	hBitmap filename	Хендл битмапа для снимка окна Имя файла для сохранения
savePaintToFile	Сохраняет текущий набросок в файл без фона	void savePaintToFile(const std::string& filename)	filename	Имя файла для сохранения
showSaveDialog	Показывает или скрывает диалог сохранения файла	void showSaveDialog(BOOL isShown)	isShown	Флаг для отображения диалога
endOfBrushStroke	Завершает текущий мазок кисти	void endOfBrushStroke()	-	-
addMessageBrushInfo	Добавляет информацию о кисти в сообщения	void addMessageBrushInfo(int ind)	ind	Индекс цвета кисти
changeShowHelp	Переключает отображение справки.	void changeShowHelp()	-	-

Таблица 4.2 – Основные методы модуля ExternalOverlay.h

Имя метода	Описание	Заголовок метода	Имя параметра	Назначение параметра
WndProc	Обрабатывает сообщения для главного окна (отрисовка, завершение и таймеры).	LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM)	Hwnd message wParam lParam	Дескриптор окна Код сообщения Дополнительные данные Дополнительные данные

Таблица 4.2 – Продолжение

registerClass	Регистрирует оконный класс для приложения.	ATOM registerClass(HINSTANCE hInstance)	hInstance	Дескриптор текущего экземпляра приложения.
InitInstance	Инициализирует экземпляр приложения и создает главное окно.	BOOL InitInstance(HINSTANCE, int)	hInstance	Дескриптор текущего экземпляра приложения
DelayedFunction	Выполняет отложенную операцию после задержки	DWORD WINAPI DelayedFunction(LPVOID lpParameter)	lpParameter	Указатель на структуру с параметрами задержки и индекса потока.
ExecuteWithDelay	Создает поток для выполнения функции с задержкой	void ExecuteWithDelay(int delayMilliseconds, int threadIndex)	delayMilliseconds threadIndex	Задержка в миллисекундах Индекс потока
CaptureScreen	Захватывает текущий экран или его часть	HBITMAP CaptureScreen(int x, int y, int width, int height)	x y width height	Абсцисса верхнего левого угла Ордината верхнего левого угла Ширина Высота
MouseProc	Обрабатывает события мыши от хука	RESULT CALLBACK MouseProc(int nCode, WPARAM wParam, LPARAM lParam)	nCode wParam lParam	Код состояния Дополнительные данные Дополнительные данные
KeyboardProc	Обрабатывает события клавиатуры от хука	LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam)	nCode wParam lParam	Код состояния Дополнительные данные Дополнительные данные

Таблица 4.2 – Продолжение

Имя метода	Описание	Заголовок метода	Имя параметра	Назначение параметра
Unhook	Отменяет хуки мыши и клавиатуры.	void Unhook()	-	-
SetHooks	Устанавливает хуки для мыши и клавиатуры.	void SetHooks()	-	-
wWinMain	Точка входа для приложения. Инициализирует классы, окна и запускает основной цикл сообщений	int APIENTRY wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow)	hInstance hPrevInstance lpCmdLine nCmdShow	Дескриптор Текущего приложения. Дескриптор предыдущего приложения Параметры командной строки Флаг способа открытия окна
onKeyDown	Обработчик нажатия клавиши на клавиатуре	void OnKeyDown(WPARAM wParam)	wParam	Код нажатой клавиши
onKeyUp	Обработчик отпускания клавиши на клавиатуре	void OnKeyUp(WPARAM wParam)	wParam	Код отпущенной клавиши

Таблица 4.3 – Основные методы модуля StringQueue.h

Имя метода	Описание	Заголовок метода	Имя параметра	Назначение параметра
initQueue	Инициализирует очередь	void initQueue(StringQueue* queue)	queue	Указатель на очередь для инициализации
isEmpty	Проверяет, пуста ли очередь	int isEmpty(StringQueue* queue)LPARAM lParam)	queue	Указатель на очередь

Таблица 4.3 – Продолжение

Имя метода	Описание	Заголовок метода	Имя параметра	Назначение параметра
isFull	Проверяет, полна ли очередь	int isFull(StringQueue* queue)	queue	Указатель на очередь
enqueue	Добавляет строку в конец очереди	void enqueue(StringQueue* queue, const char* str)	queue str	Указатель на очередь Строка, которая будет добавлена в очередь
dequeue	Удаляет и возвращает строку из начала очереди	char* dequeue(StringQueue* queue)	queue str	Указатель на очередь Строка, которая будет удалена из очереди
queueToArray	Конвертирует содержимое очереди в массив строк	char** queueToArray(StringQueue* queue)	queue	Указатель на очередь
freeQueue	Освобождает память, выделенную для всех строк в очереди.	void freeQueue(StringQueue* queue)	queue	Указатель на очередь

5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

5.1 Тестирование и проверка работоспособности программного средства

Этап тестирования играет ключевую роль в цикле разработки, учитывая неизбежность ошибок при написании программного кода. Основной задачей является выявление отличий в функционировании программы от установленных требований и последующее устранение этих разногласий с целью повышения качества программного продукта. В данном случае акцент был сделан на функциональном тестировании программы, направленном на проверку ее соответствия установленным спецификациям.

Функциональные тесты, проведенные над программой, представлены в таблице 5.1.

Таблица 5.1 – Результаты функционального тестирования

Номер теста	Тестируемая функциональность	Последовательность действий	Ожидаемый результат	Полученный результат
1	Запуск программы	1) Запуск программы	Отображение уведомления о запуске программы.	Тест пройден, рисунок 5.1
2	Переход в режим рисования и обратно	1) Нажать на кнопки ctrl + shift + w дважды	Отображение уведомлений о переходе в режим рисования и обратно	Тест пройден, рисунок 5.2
3	Смена цветов	1) Нажать на кнопки ctrl + shift + left/right 2) Нажать на кнопки ctrl + shift + up/down	Появление уведомлений об изменении цвета кисти, причем цвет сообщения должен соответсвовать цвету и сообщений о новом размере кисти	Тест пройден, рисунок 5.3
4	Рисование	1)перейти в режим отрисовки 2)зажать левую кнопку мыши и поводить курсором по экрану	Появление следов от кисти, соответвующих размеру и цвету ксити	Тест пройден, рисунок 5.4

Продолжение таблицы 5.1

Номер теста	Тестируемая функциональность	Последовательность действий	Ожидаемый результат	Полученный результат
5	Вывод справки	1)Нажать на кнопки ctrl + shift + H	Вывод списка сочетаний клавишь для работы с программой	Тест пройден
6	Вывод информации о ксити	1)Нажать на кнопки ctrl + shift + I	Вывод уведомления о размере и цвете кисти	Тест пройден.
6	Сохранение в PNG-файл	1)Нажать на кнопки ctrl + shift + C 2)Выбрать способ сохраения	Появление окна с выбором вида сохранения, после выбора появится файл с сохраненным изображением	Тест пройден. рисунок 5.5
7	Изменение режима отображения	1)Нарисовать изображение 2)Нажать на кнопки ctrl + shift + Q 3) Нажать на кнопки ctrl + shift + Q	После 2 действия рисунок должен пропасть, после 3 действия он снова появится вместе с сообщением о смене режима	Тест пройден
8	Завершение работы программы	1) Нажать на кнопки ctrl + shift + X	Конец работы программы.	Тест пройден.

5.2 Анализ полученных результатов

В результате тестирования предложенных алгоритмов были получены результаты, представленнные на рисунках 5.1 – 5.5.

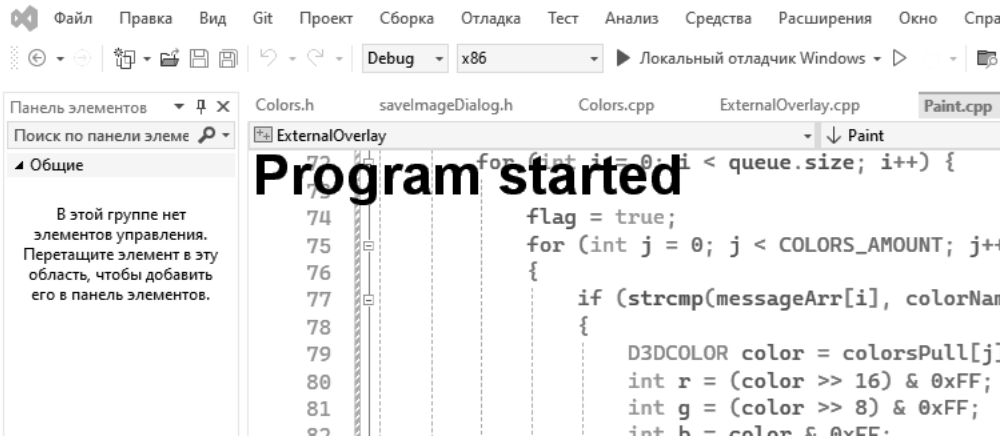

Рисунок 5.1 – Отображение окна программы

Рисунок 5.2 – Уведомления о переходе в режим рисования и обратно

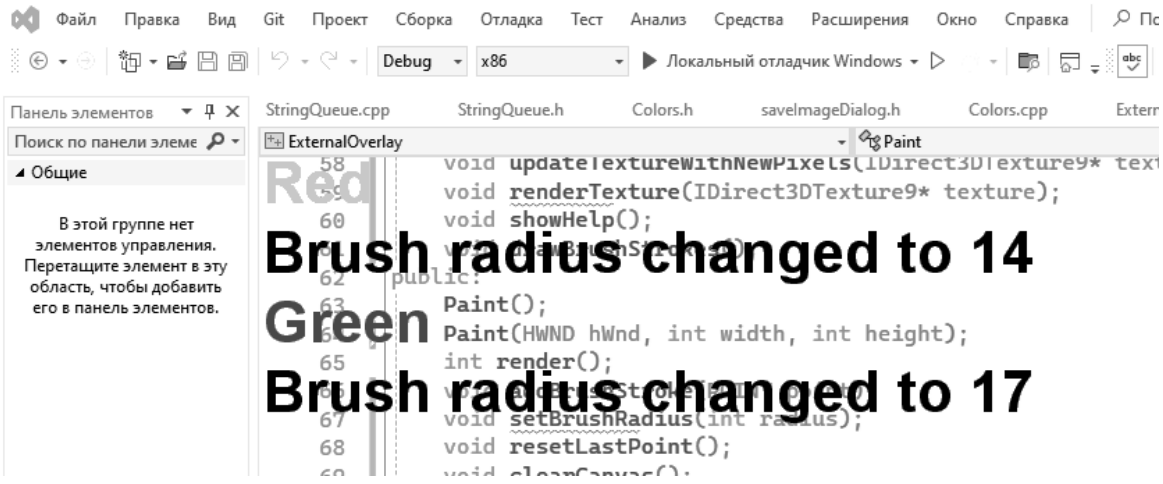

Рисунок 5.3 –уведомления об изменении цвета кисти, причем цвет сообщения должен соответсвовать цвету и сообщений о новом размере кисти

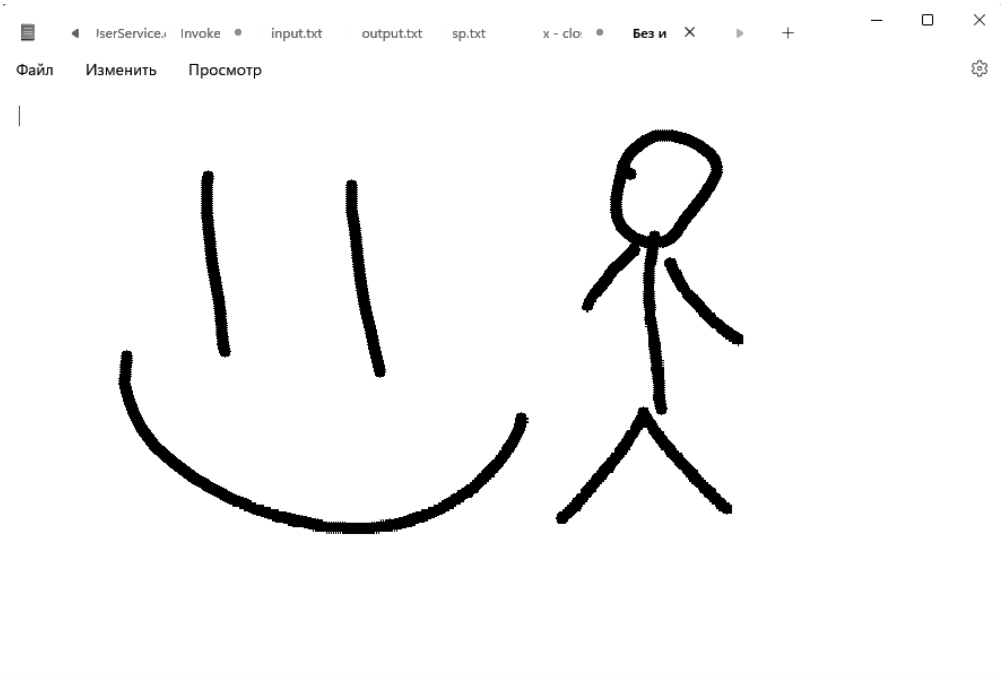

Рисунок 5.4 – Набросок созданный кситью

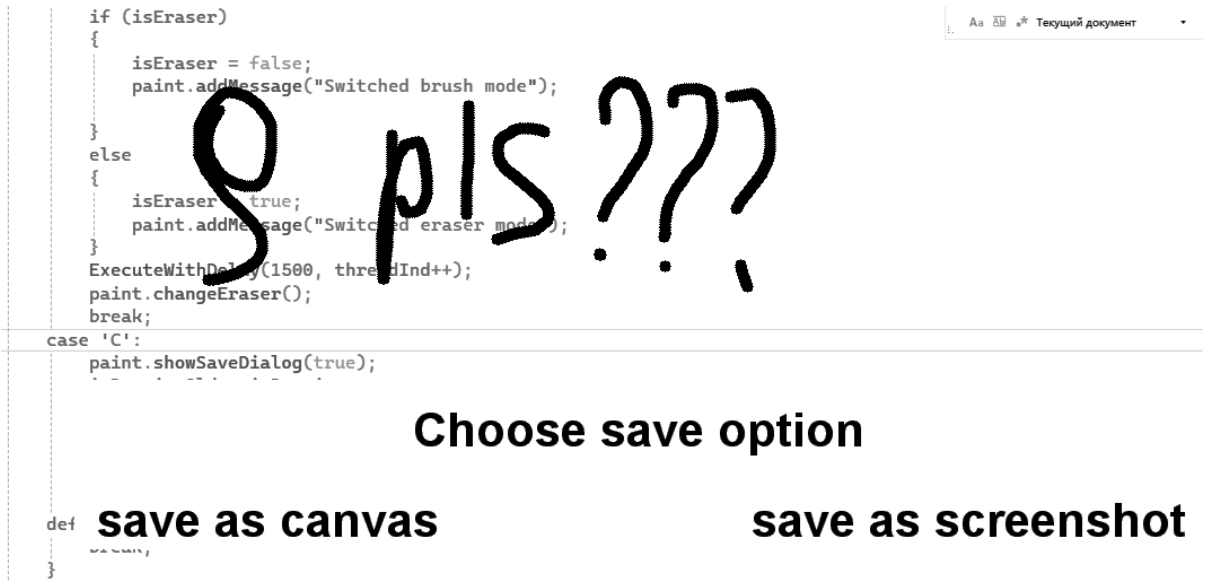

Рисунок 5.5 – Окно выбора сопособа сохранения

В результате тестирования программного средства, все тесты прошли успешно, подтверждая стабильность и соответствие функциональности приложения поставленным требованиям.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа Music Sequencer предназначена для создания визуальных аннотаций, которые будут отображаться поверх других окон системы. Для начала работы необходимо запустить приложение, дважды щелкнув по его ярлыку на рабочем столе или в меню «Пуск». После успешного запуска на экране отображается сообщение о запуске.

Для начала работы и знакомства с функционалом нажмите комбинацию клавиш ctrl, shift, H, что выведет на экран список сочетаний клавишь для работы с программой.

Для начала рисования, перейдите в соответсвующий режим, после чего при мощи левой кнопки мыши вы можете начать создавать набросок. После окончания каждого мазка, он добавляется к текстуре всех мазков для оптимизации хранения и отрисовки наброска.

Для изменения параметров кисти используйте соответсвующие сочетания клавиш, измененные парамтеры кисти будут отображены в левом верхнем углу экрана, если размер кисти будет слишком мал, то он автоматически приравняется наименьшему значению, цаета кисти изменяются циклически.

Для корректировки наброска вы можете перейти в режим стерки и убрать часть наброска, если необходимо начать набросок сначала, то соответсвующим сочетанием клавиш мы можете сбросить холст.Так же вы можете на время скрыть холст, в то время, пока холст скрыт, рисование будет недоступно.

Для сохранения нажмите нужное сочетание клавиш, после чего в открывшемся окне выберите способ сохранения, после сохранения появится соответсвующее уведомление.

Для завершения работы программы пользователь нажимает комбинацию клавиш ctrl, shift, X. Это действие приводит к завершению работы приложения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано программное средство «Светоскоп», предназначенное для создания рисунков поверх экрана. Данное программное средство предоставляет базовый функционал для Создания набросков, которые будут отображаться над другими окнами системы. Созданные наброски могут быть сохранены в формате «PNG», что обеспечивает их совместимость с фоторедакторами.

Основной функционал «Светоскоп» включает возможность задания размера и цвета кисти, что позволяет адаптировать набросок к разным фоновым изображения и разнообразить его для наглядности. Также пользователям предоставляется функция стерки для иправления ошибок, и функция сброса холста для начала наброска заново.

Программное средство использует «DirectX 9» для отрисовки наброска. Этот инструмент позволяет не только отрисовывать движения кисти во время нанесения мазка, но и предоставляет способ оптимизации отрисовки при помощи использования текстур. В связи с особенностями окна приложения, сообщения оно получает не через стандартный обработчик, а при помощи хуков клавиатуры и мыши, что так же позволяет отделять сообщения адресованные программному средсву от сообщений для других окон системы.

Перспективы развития программного средства включают увеличение количемво базовых цветов и добавление возможости самостоятельно задавать цвет и размер кисти, расширение базовых инструментов например печатью произвольного текста, линий, примитивов и добавление возможности заливки. Данные усовершенствования позволят сделать «Светоскоп» более универсальным инструментом, подходящим как для начинающих, так и для более опытных пользователей. На текущем этапе разработка успешно решает поставленные задачи, предоставляя удобное средство для базового создания экранных аннотаций.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] DirectX 9 — это набор API, разработанных для решения задач, связанных с программированием под Microsoft Windows [Электронный ресурс]. – Режим доступа: http://www.directxtutorial.com Дата доступа 20.10.2024

[2] WinAPI - набор базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/learn-to-program-for-windows Дата доступа: 21.11.2024

[3] СТП 01–2017, Стандарт предприятия, дипломные проекты, общие требования [Электронный ресурс]. – Электронные данные. – Режим доступа: https://library.bsuir.by/m/12_101945_1_141950.pdf Дата доступа: 29.11.2024

ПРИЛОЖЕНИЕ А
(обязательное)
Блок-схема алгоритма работы программы (к пункту 3.1)

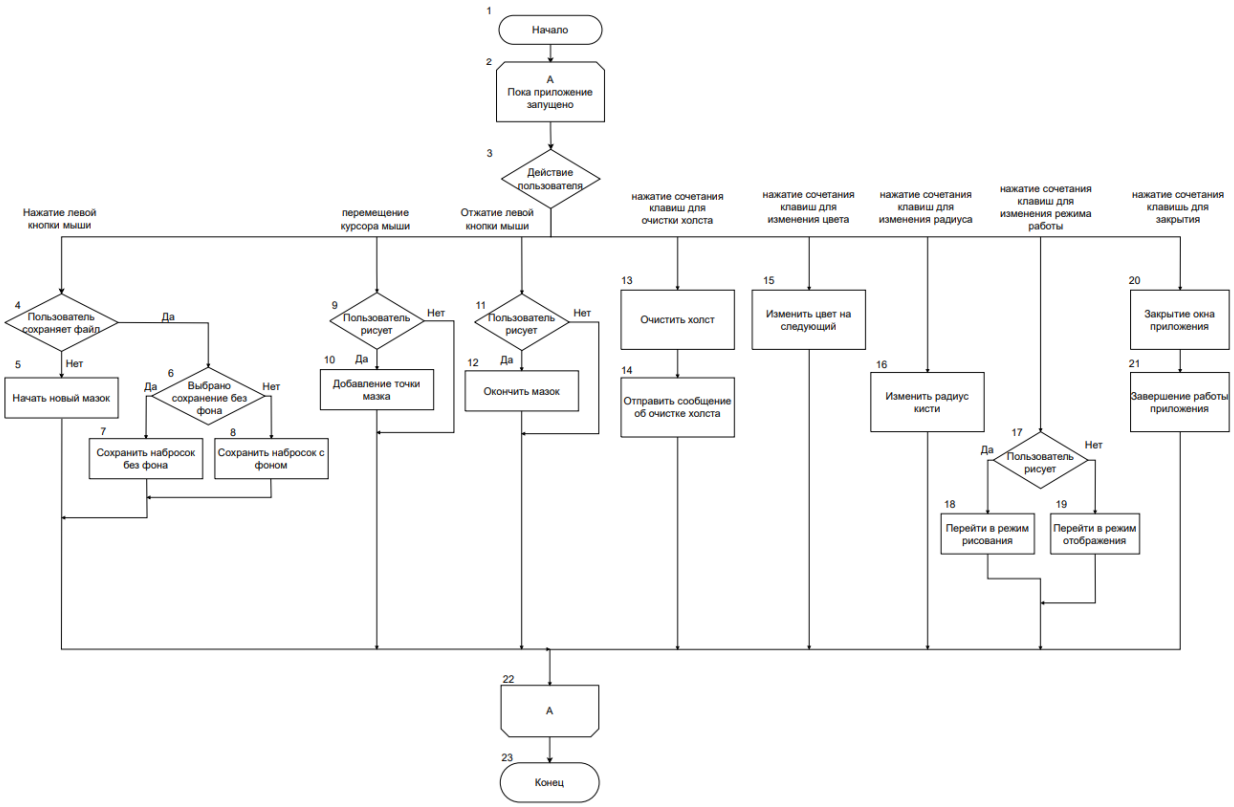

Рисунок А.1 – Блок-схема программы

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг кода (к пункту 4)

```
#pragma once

#include "resource.h"
#include "Paint.h"
#pragma once
#include <Dwmapi.h>
#include <windows.h>
#include <vector>
#include "framework.h"
#include "ExternalOverlay.h"
#include "colors.h"
//#include "saveImageDialog.h"

#define TIMER_NO_DRAWING 1001
#define TIMER_YES_DRAWING 1002
#define TIMER_YES_SHOW 1003
#define TIMER_TIME 150

struct ThreadData {
    int delayMilliseconds;
    int threadIndex;
};


// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR overlayWindowName[100] = L"Overlay";  // main window class name & The title bar text
LPCSTR targetWindowName = "Telegram";  // main window class name & The title bar text
HWND targetHWND, overlayHWND;
int width, height;
Paint paint;
int brushRadius = 5;
int threadInd = 1;
int colorInd = 0;
BOOL isShown = true;
BOOL isDrawing = false;
BOOL isDrawingOld = false;
BOOL isChoosingFile = false;
```

```
BOOL isEraser = false;
BOOL DrawFlagChange = false;
HANDLE hMutex;
// Forward declarations of functions included in this code module:
ATOM             registerClass(HINSTANCE hInstance);
BOOL             InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
// delayed func (notification stop)
DWORD WINAPI DelayedFunction(LPVOID lpParameter) {
    ThreadData* data = (ThreadData*)lpParameter;


    Sleep(data->delayMilliseconds);  // Засыпаем на указанное количество
миллисекунд
   WaitForSingleObject(hMutex, INFINITE);
   paint.stopMessage();
   ReleaseMutex(hMutex);
   delete data;
   InvalidateRect(overlayHWND, NULL, FALSE);
    return 0;
}
// start thred with deleyed task
void ExecuteWithDelay(int delayMilliseconds,int threadIndex) {
    ThreadData* data = new ThreadData{ delayMilliseconds, threadIndex };
    // Создаём поток для выполнения функции с задержкой
    CreateThread(
        NULL,               // Атрибуты безопасности (NULL — по умолчанию)
        0,                  // Размер стека (0 — по умолчанию)
        DelayedFunction,      // Указатель на функцию потока
        data,   // Параметр, передаваемый в функцию потока
        0,                  // Флаги создания (0 — запускаем сразу)
        NULL                // Указатель для получения идентификатора потока
(NULL — не требуется)
    );
}
//screen capture for saving
HBITMAP CaptureScreen(int x, int y, int width, int height) {
    HDC hScreenDC = GetDC(nullptr);
    HDC hMemoryDC = CreateCompatibleDC(hScreenDC);

    HBITMAP hBitmap = CreateCompatibleBitmap(hScreenDC, width, height);
    SelectObject(hMemoryDC, hBitmap);

    BitBlt(hMemoryDC, 0, 0, width, height, hScreenDC, x, y, SRCCOPY);
```

```
    // Освобождаем ресурсы
    DeleteDC(hMemoryDC);
    ReleaseDC(nullptr, hScreenDC);

    return hBitmap;
}


// win hooks start

HHOOK mouseHook;
HHOOK keyboardHook;
BOOL isMouseDrawing = false;
std::vector<HWND> targetWindows;

void Unhook() {
    if (mouseHook)
        UnhookWindowsHookEx(mouseHook);
    if (keyboardHook)
        UnhookWindowsHookEx(keyboardHook);
}
// Функция обработки мышиных событий
LRESULT CALLBACK MouseProc(int nCode, WPARAM wParam, LPARAM
lParam) {
    HBITMAP tmp;
    if (isDrawing) {
        static POINT lastPoint;
        if (nCode >= 0) {


            MSLLHOOKSTRUCT* mouseInfo = (MSLLHOOKSTRUCT*)lParam;
            // Получаем позицию мыши
            int mouseX = mouseInfo->pt.x;
            int mouseY = mouseInfo->pt.y;

            switch (wParam) {
            case WM_LBUTTONDOWN:
                if (!isChoosingFile) {
                    isMouseDrawing = true;
                    lastPoint.x = mouseX;
                    lastPoint.y = mouseY;
                    paint.addBrushStroke(lastPoint);
                }
```

```
            else
            {
               if (mouseX > 500 && mouseX < 1060 && mouseY > 565 &&
mouseY < 615)//canvas
               {

                  isDrawing = isDrawingOld;
                  isChoosingFile = false;
                  paint.showSaveDialog(false);
                  paint.addMessage("Canvas saved");
                  ExecuteWithDelay(1500, threadInd++);
                  InvalidateRect(overlayHWND, NULL, TRUE);  // Принудительно
перерисовываем окно
                  UpdateWindow(overlayHWND);
                  paint.savePaintToFile("E:\\5Sem\\Canvas.png");
               }
               if (mouseX > 1060 && mouseX < 1460 && mouseY > 565 &&
mouseY < 615)//screenshot
               {
                  isDrawing = isDrawingOld;
                  isChoosingFile = false;
                  paint.showSaveDialog(false);
                  paint.addMessage("Screenshot saved");
                  ExecuteWithDelay(1500, threadInd++);
                  InvalidateRect(overlayHWND, NULL, TRUE);  // Принудительно
перерисовываем окно
                  UpdateWindow(overlayHWND);
                  tmp = CaptureScreen(0, 0, width, height);
                  paint.saveScreenshotToFile(tmp,
"E:\\5Sem\\ScreenshotWithCanvas.png");
               }
               return 1;
            }
            // Левая кнопка мыши нажата
            break;
         case WM_LBUTTONUP:
            if (isMouseDrawing) {
               isMouseDrawing = false;
               paint.resetLastPoint();

               paint.endOfBrushStroke();
            }
            //Левая кнопка мыши отпущена
            break;
```

```
            case WM_MOUSEMOVE:
                if (isMouseDrawing) {  // Перемещение мыши
                    POINT currentPoint;
                    currentPoint.x = mouseX;
                    currentPoint.y = mouseY;

                    paint.addBrushStroke(currentPoint);  // Добавляем точку для кисти
при движении
                    InvalidateRect(overlayHWND, NULL, FALSE);   // Перерисовываем
окно|

                }
                return CallNextHookEx(mouseHook, nCode, wParam, lParam);
                break;

            default:
                break;
            }
        }
        return 1;
    }
    else
    return CallNextHookEx(mouseHook, nCode, wParam, lParam);

}

// Функция обработки клавиатурных событий
LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam,
LPARAM lParam) {
    char buffer[100];

    if (nCode >= 0) {

        bool ctrlPressed = GetAsyncKeyState(VK_CONTROL) & 0x8000;
        bool shiftPressed = GetAsyncKeyState(VK_SHIFT) & 0x8000;
        KBDLLHOOKSTRUCT* keyInfo = (KBDLLHOOKSTRUCT*)lParam;
        // Определяем код нажатой клавиши
        DWORD vkCode = keyInfo->vkCode;

        switch (wParam) {
        case WM_KEYDOWN:
            if (ctrlPressed && shiftPressed) {
                switch (vkCode)
                {
```

```
case 'X':
    SendMessage(overlayHWND, WM_SETREDRAW, FALSE, 0);
    Unhook();
    PostQuitMessage(0);
    return 1;
    break;
case 'D':

    paint.clearCanvas();
    paint.addMessage("Canvas cleared");
    ExecuteWithDelay(1500, threadInd++);

    //очистка нарисованного
    break;
case 'H':
    //помощь
    paint.changeShowHelp();
    InvalidateRect(overlayHWND, NULL, TRUE);
    break;
case 'I':
    //конфигурация
    paint.addMessageBrushInfo(colorInd);
    ExecuteWithDelay(1500, threadInd++);

    break;
case VK_RIGHT://цвета
    colorInd = (colorInd + 1) % COLORS_AMOUNT;

    paint.setBrushColor(colorInd);

    paint.addMessage(colorNamesPull[colorInd]);
    ExecuteWithDelay(1500, threadInd++);
    break;
case VK_LEFT:
    if (colorInd == 0)
        colorInd = COLORS_AMOUNT;
    colorInd--;
    paint.setBrushColor(colorInd);
    paint.addMessage(colorNamesPull[colorInd]);
    ExecuteWithDelay(1500, threadInd++);
    break;
case VK_UP://размер
    brushRadius += 3;
    paint.setBrushRadius(brushRadius);
```

```
            sprintf_s(buffer, "Brush radius changed to %d", brushRadius);
            paint.addMessage(buffer);
            ExecuteWithDelay(1500, threadInd++);
            break;
        case VK_DOWN:
            brushRadius -= 3;
            if (brushRadius < 1)
                brushRadius = 2;
            paint.setBrushRadius(brushRadius);

            sprintf_s(buffer, "Brush radius changed to %d", brushRadius);
            paint.addMessage(buffer);
            ExecuteWithDelay(1500, threadInd++);
            break;
        case 'Q':// показать или нет нарисованное
            if (isShown)
            {
                isShown = false;
                if (isDrawing)
                {
                    isDrawing = false;
                    DrawFlagChange = true;
                }
                SendMessage(overlayHWND, WM_SETREDRAW, FALSE, 0);
            }
            else
            {
                paint.addMessage("Switched to show Drawing mode");
                //SetTimer(overlayHWND, TIMER_YES_SHOW, TIMER_TIME,
NULL);
                ExecuteWithDelay(1500, threadInd++);
                if (DrawFlagChange)
                {

                    DrawFlagChange = false;
                    isDrawing = true;
                }
                isShown = true;
                SendMessage(overlayHWND, WM_SETREDRAW, TRUE, 0);
                InvalidateRect(overlayHWND, NULL, TRUE);  // Принудительно
перерисовываем окно
                UpdateWindow(overlayHWND);
            }
```

```
            break;
        case 'W'://режим рисования да/нет
            if (isDrawing)
            {
                paint.addMessage("Switched to no Drawing mode");
                ExecuteWithDelay(1500, threadInd++);
                isDrawing = false;
            }
            else
            {
                paint.addMessage("Switched to Drawing mode");
                ExecuteWithDelay(1500, threadInd++);
                isDrawing = true;
            }
            break;
        case 'E':
            if (isEraser)
            {
                isEraser = false;
                paint.addMessage("Switched brush mode");

            }
            else
            {
                isEraser = true;
                paint.addMessage("Switched eraser mode");
            }
            ExecuteWithDelay(1500, threadInd++);
            paint.changeEraser();
            break;
        case 'C':
            paint.showSaveDialog(true);
            isDrawingOld = isDrawing;
            isChoosingFile = true;
            InvalidateRect(overlayHWND, NULL, TRUE);  // Принудительно
перерисовываем окно
            UpdateWindow(overlayHWND);

            break;
        default:
            break;
        }

    }
```

```
            break;
        default:
            break;
        }
    }
    if (isDrawing)
        return 1;
    else
        return CallNextHookEx(keyboardHook, nCode, wParam, lParam);

}

void SetHooks() {
    mouseHook = SetWindowsHookEx(WH_MOUSE_LL, MouseProc, NULL, 0);
    if (!mouseHook)
        MessageBoxW(0,L"Не вышло установить mousehook",L"Ошибка",0);
    keyboardHook = SetWindowsHookEx(WH_KEYBOARD_LL, KeyboardProc,
NULL, 0);
    if (!keyboardHook)
        MessageBoxW(0, L"Не вышло установить keyboardHook", L"Ошибка", 0);
}


// win hooks end


int APIENTRY wWinMain(_In_ HINSTANCE hInstance,_In_opt_ HINSTANCE
hPrevInstance,_In_ LPWSTR    lpCmdLine, _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    registerClass(hInstance);

     width = GetSystemMetrics(SM_CXSCREEN); // Ширина экрана
     height = GetSystemMetrics(SM_CYSCREEN); // Высота экрана

     //устанавливаем хуки
     SetHooks();
     // создаем мьютекс для доступа к очереди paint
     hMutex = CreateMutex(NULL, FALSE, NULL);

    // Perform application initialization:
```

```
    if (!InitInstance(hInstance, SW_SHOW)){
        return FALSE;
    }

    paint = Paint(overlayHWND,width,height);
    paint.setBrushRadius(brushRadius);

    paint.addMessage("Program started");
    ExecuteWithDelay(3000, threadInd++);
    MSG msg;

    // Main message loop:
    while (GetMessage(&msg, nullptr, 0, 0)){
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return (int) msg.wParam;
}



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
ATOM registerClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = 0;
    wcex.hCursor        = LoadCursor(nullptr, IDC_CROSS);
    wcex.hbrBackground  = CreateSolidBrush(RGB(0, 0, 0));
    wcex.lpszMenuName   = overlayWindowName;
    wcex.lpszClassName  = overlayWindowName;
    wcex.hIconSm        = 0;
```

```
   return RegisterClassExW(&wcex);
}

//
//   FUNCTION: InitInstance(HINSTANCE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  hInst = hInstance; // Store instance handle in our global variable

  overlayHWND = CreateWindowExW(WS_EX_TOPMOST |
WS_EX_TRANSPARENT | WS_EX_LAYERED, overlayWindowName,
overlayWindowName, WS_POPUP,
     1, 1, width, height, nullptr, nullptr, hInstance, nullptr);

  if (!overlayHWND){
    return FALSE;
  }
  SetLayeredWindowAttributes(overlayHWND, RGB(0, 0, 0), 0,
LWA_COLORKEY);

  ShowWindow(overlayHWND, nCmdShow);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE: Processes messages for the main window.
//
//  WM_COMMAND  - process the application menu
//  WM_PAINT    - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
//
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam){
   switch (message){
   case WM_PAINT:
      paint.render();
      break;
   case WM_DESTROY:
      Unhook();
      PostQuitMessage(0);
      break;
   default:
      return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}
#pragma once

#include <string> //save error
#include <Windows.h>
#include <vector>
#include <d3d9.h>
#include <d3dx9.h>
#include "StringQueue.h"
#include <gdiplus.h>
#include "colors.h"
#include <gdiplus.h>
#include <iostream>
#pragma comment(lib, "gdiplus.lib")

#pragma comment(lib, "d3d9.lib")
#pragma comment(lib, "d3dx9.lib")

#include <DxErr.h> //get error from error code
#pragma comment(lib, "dxerr.lib")
#pragma comment(lib, "legacy_stdio_definitions.lib")



class Paint{
private:
        struct CUSTOMVERTEX {
                FLOAT x, y, z, rhw;
                D3DCOLOR color;
        };
```

```
        std::vector<POINT> brushStrokes;//мазки кистью
        std::vector<CUSTOMVERTEX> pixels;
        std::vector<CUSTOMVERTEX> pixelsOld;
        IDirect3DTexture9* texture;
        std::vector<std::pair<POINT, POINT>> lines;//линии между мазками
        IDirect3D9Ex* d3dObject = NULL; //used to create device
        IDirect3DDevice9Ex* d3dDevice = NULL; //contains functions like begin
and end scene
        D3DPRESENT_PARAMETERS d3dparams; //parameters for creating
device
        ID3DXFont* d3dFont = 0; // font used when displaying text
        HWND targetWnd;
        POINT lastPoint;
        D3DCOLOR brushColor;
        bool hasLastPoint = false;
        bool isEraser = false;
        bool saveFileChoose = false;
        BOOL isShowHelp = false;
        D3DCOLOR oldBrushColor;
        StringQueue queue;
        int width;
        int height;
        int brushRadius = 5;
        int d3D9Init(HWND hWnd);
        void drawText(char* String, int x, int y, int a, int r, int g, int b);
        void addCircle(POINT center);
        void drawPixels();
        void BresenhamLine(POINT p1, POINT p2);
        void drawFilledRect(int x, int y, int width, int height, D3DCOLOR color);
        void drawSaveFileDialog();
        void updateTextureWithNewPixels(IDirect3DTexture9* texture, const
std::vector<CUSTOMVERTEX>& newPixels);
        void renderTexture(IDirect3DTexture9* texture);
        void showHelp();
        void drawBrushStrokes();
public:
        Paint();
        Paint(HWND hWnd, int width, int height);
        int render();
        void addBrushStroke(POINT point);
        void setBrushRadius(int radius);
        void resetLastPoint();
        void clearCanvas();
        void addMessage(const char* string);
```

```
	void stopMessage();
	void setBrushColor(int ind);
	void changeEraser();
	void savePaintToFile(const std::string& filename);
	void saveScreenshotToFile(HBITMAP hBitmap,const std::string&
filename);
	void showSaveDialog(BOOL isShown);
	void endOfBrushStroke();
	void addMessageBrushInfo(int ind);
	void changeShowHelp();
};

#pragma once
#include "Paint.h"


int Paint::d3D9Init(HWND hWnd){

	if (FAILED(Direct3DCreate9Ex(D3D_SDK_VERSION, &d3dObject))){
		exit(1);
	}

	ZeroMemory(&d3dparams, sizeof(d3dparams));

	d3dparams.BackBufferWidth = width;
	d3dparams.BackBufferHeight = height;
	d3dparams.Windowed = TRUE;
	d3dparams.SwapEffect = D3DSWAPEFFECT_DISCARD;
	d3dparams.hDeviceWindow = hWnd;
	d3dparams.MultiSampleQuality = D3DMULTISAMPLE_NONE;
	d3dparams.BackBufferFormat = D3DFMT_A8R8G8B8;
	d3dparams.EnableAutoDepthStencil = TRUE;
	d3dparams.AutoDepthStencilFormat = D3DFMT_D16;

	HRESULT res = d3dObject->CreateDeviceEx(D3DADAPTER_DEFAULT,
D3DDEVTYPE_HAL,                                    hWnd,
D3DCREATE_HARDWARE_VERTEXPROCESSING,    &d3dparams,    0,
&d3dDevice);

	if (FAILED(res)){
		std::wstring ws(DXGetErrorString(res));
		std::string str(ws.begin(), ws.end());
		std::wstring ws2(DXGetErrorDescription(res));
		std::string str2(ws2.begin(), ws2.end());
```

```
            std::string error = "Error: " + str + " error description: " + str2;
            exit(1);
        }
        D3DXCreateFont(d3dDevice, 50, 0, FW_BOLD, 1, false, DEFAULT_CHARSET, OUT_DEVICE_PRECIS, ANTIALIASED_QUALITY, DEFAULT_PITCH, L"Comic Sans", &d3dFont);



        return 0;

}

Paint::Paint() {};

Paint::Paint(HWND hWnd, int width, int height){
        this->width = width;
        this->height = height;
        initQueue(&(this->queue));
        this->brushColor = D3DCOLOR_XRGB(255, 255, 255);

        d3D9Init(hWnd);
        HRESULT hr = d3dDevice->CreateTexture(
            width, height, 1, D3DUSAGE_DYNAMIC, D3DFMT_A8R8G8B8, D3DPOOL_DEFAULT, &texture, NULL);

}

int Paint::render()
{
        if (d3dDevice == nullptr)
            return 1;
        d3dDevice->Clear(0, 0, D3DCLEAR_TARGET, 0, 1.0f, 0);
        d3dDevice->BeginScene();


        renderTexture(texture);
        drawBrushStrokes();  // Отрисовка мазков кисти


        BOOL flag = true;
        char** messageArr = queueToArray(&queue);

        if (messageArr) {
```

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

                flag = true;
                for (int j = 0; j < COLORS_AMOUNT; j++)
                {
                    if (strcmp(messageArr[i], colorNamesPull[j]) == 0)
                    {
                        D3DCOLOR color = colorsPull[j];
                        int r = (color >> 16) & 0xFF;  // Красный
                        int g = (color >> 8) & 0xFF;   // Зеленый
                        int b = color & 0xFF;                  // Синий
                        drawText((char*)messageArr[i], width / 10, height
/ 10 + i * 50, 255, r, g, b);
                        flag = false;
                    }
                }
                if(flag)
                //drawText((char*)messageArr[i], width / 10, height / 10 + i *
50, 255, 171, 0, 240);
                drawText((char*)messageArr[i], width / 10, height / 10 + i * 50,
255, 255, 255, 255);
            }

            // Освобождение памяти массива
            freeArray(messageArr, queue.size);
        }

        if (isShowHelp)
        {
            showHelp();
        }
        if (saveFileChoose) {
            drawSaveFileDialog();
        }

        //drawSaveFileDialog();
        d3dDevice->EndScene();
        d3dDevice->PresentEx(0, 0, 0, 0, 0);



        return 0;
}
```

```
void Paint::drawPixels() {
	// Устанавливаем формат вершин
	d3dDevice->SetFVF(D3DFVF_XYZRHW | D3DFVF_DIFFUSE);

	// Рисуем все пиксели
	if (!pixels.empty()) {
		d3dDevice->DrawPrimitiveUP(D3DPT_POINTLIST, pixels.size(), pixels.data(), sizeof(CUSTOMVERTEX));
	}
}

void Paint::drawText(char* String, int x, int y, int a, int r, int g, int b)
{
	RECT FontPos;
	FontPos.left = x;
	FontPos.top = y;
	d3dFont->DrawTextA(0, String, strlen(String), &FontPos, DT_NOCLIP, D3DCOLOR_ARGB(a, r, g, b));
}

// Обновляем пиксели в текстуре
void Paint::updateTextureWithNewPixels(IDirect3DTexture9* texture, const std::vector<CUSTOMVERTEX>& newPixels) {
	// Получаем доступ к текстуре
	D3DLOCKED_RECT lockedRect;
	//HRESULT hr = texture->LockRect(0, &lockedRect, NULL, D3DLOCK_DISCARD); // D3DLOCK_DISCARD сбрасывает предыдущие данные
	HRESULT hr = texture->LockRect(0, &lockedRect, NULL, 0); // D3DLOCK_DISCARD сбрасывает предыдущие данные
	if (FAILED(hr)) {
		// Обработка ошибки
		return;
	}

	// Получаем указатель на данные текстуры (пиксели в текстуре)
	DWORD* pDest = (DWORD*)lockedRect.pBits;

	// Преобразуем данные из newPixels в данные для текстуры
	for (const auto& pixel : newPixels) {
		int x = static_cast<int>(pixel.x); // Позиция по X
		int y = static_cast<int>(pixel.y); // Позиция по Y
```

```
            // Проверяем, чтобы координаты не выходили за пределы текстуры
            if (x >= 0 && x < width && y >= 0 && y < height) {
                // Получаем цвет пикселя
                D3DCOLOR color = pixel.color;

                // Индекс пикселя в текстуре: y * ширина + x
                pDest[y * (lockedRect.Pitch / 4) + x] = color;
            }
        }

        // Завершаем обновление данных текстуры
        texture->UnlockRect(0);
}

void Paint::renderTexture(IDirect3DTexture9* texture) {
        // Устанавливаем текстуру
        d3dDevice->SetTexture(0, texture);

        // Прямоугольник, на который будем растягивать текстуру
        struct Vertex {
            float x, y, z, rhw;
            DWORD color;
            float u, v;
        };

        // Прямоугольник с текстурой
        Vertex vertices[] = {
            { 0.0f, 0.0f, 0.0f, 1.0f, 0xFFFFFFFF, 0.0f, 0.0f },
            { (float)width, 0.0f, 0.0f, 1.0f, 0xFFFFFFFF, 1.0f, 0.0f },
            { 0.0f, (float)height, 0.0f, 1.0f, 0xFFFFFFFF, 0.0f, 1.0f },
            { (float)width, (float)height, 0.0f, 1.0f, 0xFFFFFFFF, 1.0f, 1.0f }
        };

        // Настройка FVF и рисование
        d3dDevice->SetFVF(D3DFVF_XYZRHW | D3DFVF_DIFFUSE | D3DFVF_TEX1);
        d3dDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP, 2, vertices, sizeof(Vertex));

        d3dDevice->SetTexture(0, nullptr);
}

void Paint::drawFilledRect(int x, int y, int width, int height, D3DCOLOR color) {
```

```
        D3DRECT rect = { x, y, x + width, y + height };
        d3dDevice->Clear(1, &rect, D3DCLEAR_TARGET, color, 1.0f, 0);
}

void Paint::drawSaveFileDialog() {
        // Координаты и размеры прямоугольника
        int rectWidth = 1000;
        int rectHeight = 150;
        int rectX = (width - rectWidth) / 2;
        int rectY = (height - rectHeight) / 2;

        // Цвета для рисования
        D3DCOLOR black = D3DCOLOR_XRGB(1, 1, 1);
        D3DCOLOR white = D3DCOLOR_XRGB(255, 255, 255);

        // Нарисовать черный прямоугольник
        drawFilledRect(rectX, rectY, rectWidth, rectHeight, black);

        // Текст "Choose save option" по центру
        drawText((char*)"Choose save option", rectX + rectWidth / 2 - 200, rectY +
20, 255, 255, 255, 255);

        // Координаты для текста кнопок
        int buttonY = rectY + rectHeight - 50;

        // Текст "save as canvas" (слева)
        drawText((char*)"save as canvas", rectX + 20, buttonY, 255, 255, 255, 255);

        // Текст "save as screenshot" (справа)
        drawText((char*)"save as screenshot", rectX + rectWidth - 400, buttonY, 255,
255, 255, 255);
}

void Paint::BresenhamLine(POINT p1, POINT p2) {
        int dx = abs(p2.x - p1.x);  // Разница по X
        int dy = abs(p2.y - p1.y);  // Разница по Y
        int sx = (p1.x < p2.x) ? 1 : -1;  // Направление движения по X
        int sy = (p1.y < p2.y) ? 1 : -1;  // Направление движения по Y
        int err = dx - dy;  // Ошибка, инициализируем разницу между dx и dy

        int frequncy = brushRadius / 2;
        int pointInd = 0;
        while (true) {
                if ((pointInd % frequncy) == 0) {
```

```
                    pixels.push_back({ (float)p1.x, (float)p1.y, 0.0f, 1.0f,
brushColor });
                    int r = brushRadius;

                    for (int dx = -r; dx <= r; ++dx) {
                        for (int dy = -r; dy <= r; ++dy) {
                            if (dx * dx + dy * dy <= r * r) {
                                POINT p = { p1.x + dx, p1.y + dy };

                                pixels.push_back({ (float)p.x, (float)p.y,
0.0f, 1.0f, brushColor });

                            }
                        }
                    }
                    pointInd = 0;
                }
                pointInd++;
                // Если дошли до конечной точки, завершить
                if (p1.x == p2.x && p1.y == p2.y)
                    break;

                int e2 = err * 2;  // Удваиваем ошибку для вычислений

                if (e2 > -dy) {
                    err -= dy;
                    p1.x += sx;  // Шаг по X
                }

                if (e2 < dx) {
                    err += dx;
                    p1.y += sy;  // Шаг по Y
                }
            }


}

void Paint::addBrushStroke(POINT point) {
    // Добавляем заполненные точки в радиусе кисти
    int r = brushRadius;

    for (int dx = -r; dx <= r; ++dx) {
        for (int dy = -r; dy <= r; ++dy) {
```

```
                    if (dx * dx + dy * dy <= r * r) {
                        POINT p = { point.x + dx, point.y + dy };

                            pixels.push_back({ (float)p.x, (float)p.y, 0.0f, 1.0f,
brushColor });

                    }
                }
            }

            // Если есть предыдущая точка, добавляем линию между последней и
текущей точкой
            if (hasLastPoint) {
                BresenhamLine(lastPoint, point);
            }

            // Сохраняем текущую точку как последнюю
            lastPoint = point;
            hasLastPoint = true;
}

void Paint::setBrushRadius(int radius) {
            brushRadius = radius;
}

void Paint::clearCanvas() {
            freeQueue(&queue);
            initQueue(&queue);
            pixels.clear();
            d3dDevice->CreateTexture(
                width, height, 1, D3DUSAGE_DYNAMIC, D3DFMT_A8R8G8B8,
D3DPOOL_DEFAULT, &texture, NULL);
}

void Paint::drawBrushStrokes() {
            drawPixels();
}

void Paint::resetLastPoint() {
            hasLastPoint = false;
}

void Paint::addMessage(const char* sting) {
            enqueue(&queue, sting);
```

```
}

void Paint::stopMessage() {
	dequeue(&queue);
}

void Paint::setBrushColor(int ind)
{
	brushColor = colorsPull[ind];
}

void Paint::changeEraser() {
	if (isEraser)
	{
		brushColor = oldBrushColor;
		isEraser = false;
	}
	else
	{
		oldBrushColor = brushColor;
		brushColor = eraser;
		isEraser = true;
	}
}

void Paint::savePaintToFile(const std::string& filename) {

	  // Получаем поверхность из текстуры
	updateTextureWithNewPixels(texture, pixels);
	IDirect3DSurface9* surface = nullptr;
	HRESULT hr = texture->GetSurfaceLevel(0, &surface);

	if (FAILED(hr)) {
		MessageBoxA(nullptr, "Failed to get surface from texture!", "Error",
MB_OK | MB_ICONERROR);
		return;
	}

	// Сохраняем поверхность в файл формата PNG
	hr = D3DXSaveSurfaceToFileA(filename.c_str(), D3DXIFF_PNG, surface,
nullptr, nullptr);

	if (FAILED(hr)) {
```

```
            MessageBoxA(nullptr, "Failed to save image to file!", "Error", MB_OK | MB_ICONERROR);
        }


        // Освобождаем поверхность
        surface->Release();
}

bool MergeImages(const std::wstring& baseImagePath, const std::wstring& overlayImagePath, const std::wstring& outputPath) {

            using namespace Gdiplus;
            using namespace std;

            ULONG_PTR gdiplusToken;
            GdiplusStartupInput gdiplusStartupInput;
            GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, nullptr);
            // Загружаем базовое изображение
            Image baseImage(baseImagePath.c_str());
            if (baseImage.GetLastStatus() != Ok) {
                wcerr << L"Failed to load base image!" << endl;
                return false;
            }

            // Загружаем изображение-оверлей
            Image overlayImage(overlayImagePath.c_str());
            if (overlayImage.GetLastStatus() != Ok) {
                wcerr << L"Failed to load overlay image!" << endl;
                return false;
            }

            // Создаём графический объект для базового изображения
            Bitmap resultBitmap(baseImage.GetWidth(), baseImage.GetHeight(), PixelFormat32bppARGB);
            Graphics graphics(&resultBitmap);

            // Рисуем базовое изображение
            graphics.DrawImage(&baseImage, 0, 0, baseImage.GetWidth(), baseImage.GetHeight());

            // Рисуем изображение-оверлей поверх базового изображения
            graphics.DrawImage(&overlayImage, 0, 0, overlayImage.GetWidth(), overlayImage.GetHeight());
```

```
            // Сохраняем итоговое изображение
            CLSID pngClsid;
            CLSIDFromString(L"{557CF406-1A04-11D3-9A73-
0000F81EF32E}", &pngClsid); // CLSID для формата PNG

            if (resultBitmap.Save(outputPath.c_str(), &pngClsid, nullptr) != Ok) {
                wcerr << L"Failed to save result image!" << endl;
                return false;
            }

            wcout << L"Image saved successfully to " << outputPath << endl;

            //GdiplusShutdown(gdiplusToken);

        return true;
}

void Paint::saveScreenshotToFile(HBITMAP hBitmap, const std::string& filename
) {
        if (!hBitmap) {
            MessageBoxA(nullptr, "Invalid HBITMAP!", "Error", MB_OK |
MB_ICONERROR);
            return;
        }

        // Получаем информацию о HBITMAP
        BITMAP bmp;
        GetObject(hBitmap, sizeof(BITMAP), &bmp);

        int width = bmp.bmWidth;
        int height = bmp.bmHeight;

        // Инициализируем Direct3D9
        IDirect3D9* d3d = Direct3DCreate9(D3D_SDK_VERSION);
        if (!d3d) {
            MessageBoxA(nullptr, "Failed to initialize Direct3D9!", "Error",
MB_OK | MB_ICONERROR);
            return;
        }

        // Параметры для создания временного устройства
        D3DPRESENT_PARAMETERS d3dpp = {};
        d3dpp.Windowed = TRUE;
```

```
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.hDeviceWindow = GetDesktopWindow(); // Используем десктопное окно

    // Создаём временное устройство
    IDirect3DDevice9* d3dDevice = nullptr;
    if (FAILED(d3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, d3dpp.hDeviceWindow,
        D3DCREATE_SOFTWARE_VERTEXPROCESSING, &d3dpp, &d3dDevice))) {
        MessageBoxA(nullptr, "Failed to create Direct3D device!", "Error", MB_OK | MB_ICONERROR);
        d3d->Release();
        return;
    }

    // Создаём поверхность для хранения данных изображения
    IDirect3DSurface9* surface = nullptr;
    if (FAILED(d3dDevice->CreateOffscreenPlainSurface(width, height, D3DFMT_A8R8G8B8,
        D3DPOOL_SYSTEMMEM, &surface, nullptr))) {
        MessageBoxA(nullptr, "Failed to create surface!", "Error", MB_OK | MB_ICONERROR);
        d3dDevice->Release();
        d3d->Release();
        return;
    }

    // Заполняем поверхность данными из HBITMAP
    HDC hdcSurface;
    if (FAILED(surface->GetDC(&hdcSurface))) {
        MessageBoxA(nullptr, "Failed to get surface DC!", "Error", MB_OK | MB_ICONERROR);
        surface->Release();
        d3dDevice->Release();
        d3d->Release();
        return;
    }

    HDC hdcBitmap = CreateCompatibleDC(nullptr);
    HBITMAP oldBitmap = (HBITMAP)SelectObject(hdcBitmap, hBitmap);

    BitBlt(hdcSurface, 0, 0, width, height, hdcBitmap, 0, 0, SRCCOPY);
```

```
	// Освобождаем DC
	SelectObject(hdcBitmap, oldBitmap);
	DeleteDC(hdcBitmap);
	surface->ReleaseDC(hdcSurface);

	// Сохраняем поверхность в файл PNG

	//D3DXSaveSurfaceToFileA(filename.c_str(), D3DXIFF_PNG, surface, nullptr, nullptr);

	if (FAILED(D3DXSaveSurfaceToFileA("base.png", D3DXIFF_PNG, surface, nullptr, nullptr))) {
		MessageBoxA(nullptr, "Failed to save surface to PNG!", "Error", MB_OK | MB_ICONERROR);
	}
	else {
		//MessageBoxA(nullptr, filename.c_str(), "Image saved to ", MB_OK);
	}

	// Освобождаем ресурсы
	surface->Release();
	d3dDevice->Release();
	d3d->Release();


	int size_needed = MultiByteToWideChar(CP_UTF8, 0, filename.c_str(), (int)filename.size(), nullptr, 0);
	std::wstring wstr(size_needed, 0);
	MultiByteToWideChar(CP_UTF8, 0, filename.c_str(), (int)filename.size(), &wstr[0], size_needed);

	savePaintToFile("overlay.png");
	MergeImages(L"base.png", L"overlay.png", wstr);
}

void Paint::showSaveDialog(BOOL isShown) {
	saveFileChoose = isShown;
}

void Paint::endOfBrushStroke() {
	updateTextureWithNewPixels(texture, pixels);
	pixels.clear();
}
```

```
void Paint::addMessageBrushInfo(int ind) {
	int size = snprintf(nullptr, 0, "Brush color - %s \n Brush radius - %d", colorNamesPull[ind], brushRadius) + 1;

	// Выделяем память для итоговой строки
	char* result = new char[size];

	// Формируем строку с использованием sprintf
	snprintf(result, size, "Brush color - %s \n Brush radius - %d", colorNamesPull[ind], brushRadius);
	addMessage(result);
}

void Paint::changeShowHelp()
{
	if (isShowHelp)
		isShowHelp = false;
	else
		isShowHelp = true;
}

void Paint::showHelp() {
	drawText((char*)"ctrl+shift + ...:\n \
X - close \n \
D - clear canvas \n \
H - show / unshow help \n \
I - info(configuration) \n \
right / left - change color \n \
up - ++ brush size \n \
down - -- brush size \n \
E - eraser \n \
S - save \n \
Q - show / unshow canvas \n \
W - paint mode", width / 10 * 7, height / 50, 255, 171, 0, 240);
}

#pragma once
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_QUEUE_SIZE 100
typedef struct {
```

```
    char* data[MAX_QUEUE_SIZE];
    int front;  // Указывает на начало очереди
    int rear;   // Указывает на конец очереди
    int size;   // Текущий размер очереди
} StringQueue;
void initQueue(StringQueue* queue);
int isEmpty(StringQueue* queue);
int isFull(StringQueue* queue);
void enqueue(StringQueue* queue, const char* str);
char* dequeue(StringQueue* queue);
char* peek(StringQueue* queue);
char** queueToArray(StringQueue* queue);
void freeArray(char** array, int size);
void freeQueue(StringQueue* queue);
#pragma once
#include "StringQueue.h"

#define MAX_QUEUE_SIZE 100  // Максимальный размер очереди

// Функция для инициализации очереди
void initQueue(StringQueue* queue) {
    queue->front = 0;
    queue->rear = 0;
    queue->size = 0;
}

// Проверка, пуста ли очередь
int isEmpty(StringQueue* queue) {
    return queue->size == 0;
}

// Проверка, полна ли очередь
int isFull(StringQueue* queue) {
    return queue->size == MAX_QUEUE_SIZE;
}

// Добавление элемента в очередь (enqueue)
void enqueue(StringQueue* queue, const char* str) {
    if (isFull(queue)) {
        printf("Ошибка: Очередь переполнена!\n");
        return;
    }
    queue->data[queue->rear] = _strdup(str);  // Копируем строку в очередь
    queue->rear = (queue->rear + 1) % MAX_QUEUE_SIZE;
```

```
    queue->size++;
}

// Удаление элемента из очереди (dequeue)
char* dequeue(StringQueue* queue) {
    if (isEmpty(queue)) {
        printf("Ошибка: Очередь пуста!\n");
        return NULL;
    }
    char* removed = queue->data[queue->front];
    queue->front = (queue->front + 1) % MAX_QUEUE_SIZE;
    queue->size--;
    return removed;
}

// Просмотр первого элемента (peek)
char* peek(StringQueue* queue) {
    if (isEmpty(queue)) {
        printf("Ошибка: Очередь пуста!\n");
        return NULL;
    }
    return queue->data[queue->front];
}

// Функция для перевода очереди в массив строк
char** queueToArray(StringQueue* queue) {
    if (isEmpty(queue)) {
        return NULL;
    }

    char** array = (char**)malloc(queue->size * sizeof(char*));
    if (!array) {
        printf("Ошибка выделения памяти!\n");
        return NULL;
    }

    int current = queue->front;
    for (int i = 0; i < queue->size; i++) {
        array[i] = _strdup(queue->data[current]);  // Копируем строку в массив
        current = (current + 1) % MAX_QUEUE_SIZE;
    }

    return array;
}
```

```
// Функция для освобождения памяти массива строк
void freeArray(char** array, int size) {
    for (int i = 0; i < size; i++) {
        free(array[i]);
    }
    free(array);
}

// Функция для освобождения памяти очереди
void freeQueue(StringQueue* queue) {
    while (!isEmpty(queue)) {
        free(dequeue(queue));
    }
}

#pragma once
#include <d3dx9.h>
#define COLORS_AMOUNT 12
extern D3DCOLOR colorsPull[COLORS_AMOUNT];

extern const char* colorNamesPull[COLORS_AMOUNT];

extern const D3DCOLOR eraser;
#include "colors.h"

// Определяем массив цветов
D3DCOLOR colorsPull[COLORS_AMOUNT] = {
    D3DCOLOR_XRGB(255, 255, 255), // Белый
    D3DCOLOR_XRGB(1, 1, 1),       // Черный
    D3DCOLOR_XRGB(128, 128, 128), // Серый

    D3DCOLOR_XRGB(255, 0, 0),     // Красный
    D3DCOLOR_XRGB(0, 255, 0),     // Зеленый
    D3DCOLOR_XRGB(0, 0, 255),     // Синий

    D3DCOLOR_XRGB(255, 255, 0),   // Желтый
    D3DCOLOR_XRGB(0, 255, 255),   // Голубой
    D3DCOLOR_XRGB(255, 0, 255),   // Пурпурный

    D3DCOLOR_XRGB(255, 165, 0),   // Оранжевый
    D3DCOLOR_XRGB(139, 69, 19),   // Коричневый
    D3DCOLOR_XRGB(255, 192, 203)  // Розовый
};
```

```
const D3DCOLOR eraser = D3DCOLOR_XRGB(0, 0, 0);

// Определяем массив с именами цветов
const char* colorNamesPull[COLORS_AMOUNT] = {
    "White",      // Белый
    "Black",      // Черный
    "Gray",       // Серый

    "Red",        // Красный
    "Green",      // Зеленый
    "Blue",       // Синий

    "Yellow",     // Желтый
    "Light blue", // Голубой
    "Magenta",    // Пурпурный

    "Orange",     // Оранжевый
    "Brown",      // Коричневый
    "Pink"        // Розовый
};
```