Федеральное государственное автономное  
образовательное учреждение  
высшего образования  
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий  
институт  
Кафедра «Информатика»  
кафедра

**КУРСОВАЯ РАБОТА**

Разработка программной среды для создания интерактивных приложений  
тема

Руководитель                   А. С. Черниговский

подпись, дата инициалы, фамилия

Студент КИ17-17Б, 031722834  В. И. Кулиненко

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2020

**ВВЕДЕНИЕ**

Компьютер прошёл долгий путь к современному потребителю. Когда-то бывший инструментом учёных и военных, сегодня он стал для многих скорее чем-то вроде игрушки. В обыденной жизни люди используют компьютеры больше для просмотра фильмов, прослушивания музыки и компьютерных игр. Но есть и другая сторона использования компьютеров. В противовес обыденному потреблению, можно противопоставить творчество. Да, компьютеры во всю используются художниками для рисования, а писатели ещё со времён монохромный терминалов печатают на них книги. Но мы можем использовать компьютеры для чего-то более технического, свойственного нашей специализации.

Искусство - понятие достаточно пространное и обширное в людских кругах. Кто-то считает, что под этим термином стоят произведения давно покойных классиков и не более того. Кто-то - что к нему не относятся продукты, сделанные с целью получения выгоды, при этом тут же отметается огромный культурный пласт фильмов и музыки. Якобы искусство должно рождаться лишь из одной идеи и одной лишь этой идеей жить. Мы же считаем, что искусство – это всё, что вызывает у нас хоть какие-то эмоции.

Пусть у человека нет твёрдой руки художника или таланта писателя, но, если он хочет отойти от привычной культуры потребления и начать творить, он всегда может сделать это на компьютере при помощи кода. Ведь искусство можно творить, условно говоря, не только кистью или пером, но и кодом. Программным кодом, который будет говорить компьютеру, что тому делать, что показывать. Казалось бы, о каком искусстве может быть речь при всей строгости и детерминированности кода, но сами компьютеры целиком построены на математике, а математика воистину красива. Никто из познавших великолепие множества Мандельброта и видевших всё многообразие клеточных автоматов вроде игры «Жизнь» Конвея не скажет, что программирование – не искусство.

В современном мире компьютер есть у каждого. Как у каждого есть возможность общаться с ним посредством программирования. Тем не менее, это не является лёгкой задачей. Если у человека нет усидчивости, действительно сильного желания корпеть над процессом, он может отложить это дело в долгий ящик.

В данном проекте мы хотели бы облегчить людям процесс вхождения в область компьютерной графики, предоставить программную среду с набором заранее заложенного функционала для создания интерактивных приложений творческой направленности, но не ограничиваемыми ей.

В качестве объекта исследования мы обозначаем область компьютерной графики. Мы хотим изучить то, как современные компьютеры работают с ней, и как мы можем её использовать. Предметом исследования является разработка программной среды для создания интерактивных приложений, где все желающие смогут творить не кистью, но кодом.

Далее в работе планируется провести обзор предметной области, поиск и анализ аналогов, сбор требований, и, наконец, начать разработку.

# Обзор теоретической области проекта

Перед началом процесса разработки нам нужно разобраться в предметной области проекта, то есть провести деятельность, направленную на выявление реальных потребностей заказчика, а также на выяснение смысла высказанных требований. Как разработчики, мы должны научиться:

- понимать язык, на котором говорят заказчики;

- выявить цели их деятельности;

- определить набор решаемых ими задач;

- определить набор сущностей, с которыми приходится иметь дело при решении этих задач.

## Анализ предметной области

В данном разделе мы планируем рассмотреть нашу предметную область. Если описать проект в общих словах, то его основная цель – это упростить программистам доступ к мультимедийным возможностям их операционной системы. Под упрощенным мы понимаем то, что всю низкоуровневую работу берёт на себя наш продукт в лице разрабатываемой программной среды, а пользователям остаётся только вызвать нужную функцию или метод.

Для того чтобы «взять на себя» эту часть функционала, нам нужно изучить соответствующую теорию, чем мы и займёмся далее. В данный момент затрагивается только мультимедийная составляющая проекта, а потому, теорию можно поделить на два раздела: графику и звук.

С точки зрения пользователя компьютера, вся графика на мониторе – это кучка пикселей. С точки зрения компьютера, графика – это набор математических функций (так называемая векторная графика), в основе своей голая теория. Но, чтобы показать её пользователю, графика посредством монитора должна быть выведена в реальный мир. Аналоговые средства, вроде некоторых моделей осциллографов, имеющих иные принципы работы, здесь не рассматриваются. Хотя это и является только базовой частью, основой, именно работу над отображением пикселей на экран мы ставим в основу нашего проекта.

Итак, содержимое экрана можно представить как двумерный массив пикселей. С точки зрения программы пиксель (то есть его цвет) однозначно определяется одним целым 32-битным числом. Если мы используем (а мы используем) модель RGBA (что расшифровывается как red, green, blue и alpha), то каждые 8 бит этого числа выделяются соответственно на красную, зелёную, синюю составляющую цвета. Последние 8 бит соответствуют уровню прозрачности, чем меньше число, тем прозрачней будет пиксель. При выборе такой модели на каждый компонент мы имеем 256 возможных состояния. Например, цвет в RGBA «255,255,0,255» будет соответствовать смеси красного и зелёного цветов, то есть жёлтому.

Теперь, когда мы разобрались с самой основой, пойдём чуть дальше. Как уже говорилось внутри компьютера графика хранится, как набор математических функций. Например, так хранятся шрифты для текста. Как бы мы ни увеличивали шрифт в каком-нибудь текстовом процессоре, например в «Microsoft Word», он всегда будет выглядеть безупречно, вы с трудом разглядите там сетку пикселей. Всё потому, что перед тем, как вывести текст на экран, компьютер работает над внутренним представлением шрифта (векторной графикой), а потом уже показывает человеку в понятном для него виде. Операция по выводу графики на экран называется растеризацией.

Растеризация – это вывод изображения на растер (то есть сетку пикселей) экрана. Когда векторная математика работает с идеализированными представлениями примитивов, вроде прямой или кривой Безье, которые легко описываются уравнениями, наш мир и мы сами в основе своей не такие. Экраны наших компьютеров состоят из квадратных пикселей. Мы можем это увидеть, если приглядимся, - любая прямая (если она не строго горизонтальна или вертикальна), будет выглядеть зубчатой. Для исправления такого эффекта существует технология сглаживания (то есть «anti-aliasing*»*), где соседним пикселям задаётся полупрозрачный оттенок, что придаёт примитивам ощущение гладкости, но в данной работе подобные подходы не рассматриваются.

Итак, когда мы работаем с графикой, перед выводом на монитор её нужна растеризовать, в результате чего мы полуим массив пикселей. Следующая задача, это вывести эти пиксели на экран. Как и любое взаимодействие с аппаратной частью, подобное дело не обходится без участия операционной системы. В нашем случае, так как в качестве операционной системы мы используем Windows, это набор библиотек и интерфейсов «Windows API» или кратко «WinAPI». WinAPI представляет функционал по работе с графическим интерфейсом программ, устройствами вывода видео и звука, а также считыванию ввода с устройств, вроде клавиатуры и мыши. В общем, всё то, что понадобится в нашем проекте. Стоит упомянуть, что через WinAPI также настраивается привязка набора интерфейсов OpenGL для работы с видеокартой, что используется в нашем проекте.

Последним пунктом в нашем списке стоит работа со звука. Звук, как известно, представляет собой волну. В компьютерном представлении звук — это набор семплов, то есть дискретных значений, полученных в результате импульсно-кодовой модуляции аналогового сигнала. Наиболее распространенный формат хранения такого «сырого» звука (то есть без сжатия и предобработки) – это формат «wav». Файлы данного формата (как и любого другого) имеют характерный заголовок, где хранится информация вроде идентификатора контейнера («RIFF» для wav) и самого формата, а также размер файла. Кроме этого, в заголовке для этого формата определены два куска (с англ. «chunk») данных, определённых маркерами «fmt» и «data». Маркер «fmt» обозначает кусок заголовка, где определены характеристики данного аудиофайла, вроде количества каналов (моно, стерео и т.д.), частоты дискретизации (44100 в Audio CD), количества бит в семпле (8, 16, 24 или 32). Второй кусок данных обозначается маркером «data», как раз после него следуют сами семплы, хранящие нужную нам аудио-составляющую файла.

Стоит отметить, что, согласно стандарту, кусок «data» не должен обязательно следовать сразу же за куском «fmt», между ними может быть пробел, заполненный иными данными. Подобные вещи важно учитывать при ручном чтении файла (то есть без использования сторонних библиотек), чем мы и занимаемся.

За вывод аудио опять-таки отвечает WinAPI. Набор библиотек берёт на себя процедуру поиска нужного аудио устройства для вывода и всей работы с ним. Набор функций принимает данные в «сыром» виде (то есть из файла «wav») и отправляет их на аудио устройство вывода. Важно отметить, что WinAPI не даёт функционал по одновременному воспроизведению нескольких аудио фрагментов (то есть смешиванию «mix»), эта задача возлагается на разработчиков данного проекта.

Так нами был проведён анализ предметной области. В результате чего мы стали лучше разбираться в данной области и понимать требуемый функционал. Далее мы хотим провести поиск и анализ аналогов нашего проекта.

# Анализ проекта

Первым необходимым шагом при начале анализа будущего проекта идёт обследование рынка на наличие конкурентов. Затем мы должны точно определить требования к приложению и обозначить его конкретную функциональность. На основе этого следует подобрать наиболее подходящие средства разработки и составить план разработки проекта.

## Обзор существующих аналогов

В результате процесса поиска аналогов были обнаружены несколько продуктов, похожих по функциональности на наш проект.

Simple DirectMedia Library (SDL) – кроссплатформенная библиотека для разработки программ, представляющая собой надстройку над аппаратным обеспечением для удобной работы с графикой, звуком, вводом, сетью, потоками и т.д. Библиотека работает с графическими интерфейсами OpenGL, Vulkan и Direct3D. Первая версия вышла уже достаточно давно, в 1998 году. Библиотека написана на C, но также имеет «привязки» для других языков программирования. За годы своего существования библиотека хорошо себя показала, и была использована в сотнях проектах игровой или просто графической направленности.

Löve – игровой движок с открытым исходным кодом. Написан на языке C++. Базируется на библиотеке SDL, но при этом добавляет ещё один уровень абстракции: код пишется на языке Lua и интерпретируется в отдельной программе, что позволяет в коде Lua вызывать функции, написанные с использованием SDL. Благодаря использованию относительно простого для освоения языка Lua, Löve является одним из самых популярных средств для быстрого прототипирования интерактивных приложений (игр).

Существуют и другие аналоги вроде SFML и Allegro, но все они достаточно схожи по функционалу.

Проанализировав аналоги, мы принимаем тот факт, что не в силах конкурировать с подобными продуктами, столь хорошо зарекомендовавшими себя за продолжительное время своего существования. Тем не менее, важной целью (и задачей) нашего проекта, как и любого другого, является получение знаний. И нам, вместе с получением этих бесценных знаний, хотелось бы также воспользоваться опытом наших коллег для улучшения нашего будущего проекта.

## Спецификация требований разрабатываемого программного решения

Перед началом разработки проекта мы должны четко сформулировать и описать все требования к итоговому продукту.

Проект «Разработка программной среды для создания интерактивных приложений» нацелен на создание комплекса программных средств, в котором пользователи смогут создавать свои интерактивные приложения на языке Lua, используя интерпретатор и редактор кода, также являющиеся частью данного проекта.

Библиотека реализуется на языке C++ с использованием средств, предоставляемых операционной системой Windows - WinAPI и набором программных интерфейсов для видеокарты - OpenGL. Для привязки Lua используется официальный Lua C API, а для графического интерфейса редактора – набор инструментов QT.

Продукт разрабатывается только под семейство операционных систем Windows.

## Функциональные требования

В результате сбора требований был сформирован полный список требований к разрабатываемой программе. Библиотека должна содержать в себе следующий функционал.

Функции отрисовки графики:

* 1. Вывод текста на экран.
  2. Отрисовка линий.
  3. Отрисовка окружностей.
  4. Отрисовка прямоугольников.
  5. Отрисовка треугольников.
  6. Закрашивание отдельных пикселей.
  7. Вывод на экран изображений в формате «bmp».
  8. Вывод на экран только части изображения.
  9. Закрашивание всего экрана.
  10. Масштабирование изображений.
  11. Интерполяция методом ближайших соседей.
  12. Билинейная интерполяция.
  13. Бикубическая интерполяция.
  14. Поддержка прозрачных изображений.

Функции считывания ввода:

* 1. Определение нажатия кнопок клавиатуры.
  2. Определение нажатия кнопок мыши.
  3. Определение местоположения курсора мыши.

Функции воспроизведения аудиофайлов:

* 1. Загрузка звуковых файлов в формате «wav».
  2. Одновременное воспроизведение нескольких аудиофайлов.

Реализация математических векторов и работа с ними:

1. Сложение и вычитание векторов.
2. Скалярное и векторное произведения.
3. Поворот вектора на угол.
4. Нормализация вектора.
5. Получение длины векторов.

К функциональным требованиям для интерпретатора Lua относится только наличие «привязок» и доступность всех функции библиотеки на C++ через код на Lua.

Следующие функциональные требования относятся к редактору кода:

1. Возможность сохранения и загрузки файлов с кодом на Lua.
2. Возможность запуска и завершения интерпретатора из редактора.
3. Поддержка горячих клавиш для процедур, описанных в 1 и 2.
4. Поддержка нескольких вкладок.
5. Изменение визуальной темы редактора.
6. Поддержка символов-юникод в окне редактора.
7. Подсветка кода.
8. Вывод стандартного потока вывода в отдельное окно.
9. Возможность отправки строки в стандартный поток ввода.

## Нефункциональные требования

От проекта требуется реализация следующих требований:

1. Программа не должна аварийно завершаться в процессе работы.
2. Программа должна корректно обрабатывать ошибки.
3. Программа должна показывать информативные сообщения об ошибках.
4. Обновление содержимого экрана должно происходить не реже 60 раз в секунду.

## Требования к интерфейсам

Взаимодействие с библиотекой на C++ происходит посредством подключения заголовочного файла, переопределения основного класса «GameEngine» и вызова соответствующих функций, предоставляемых нашим продуктом.

Взаимодействие с интерпретатором происходит посредством передачи программе через командную строку файла с исходным кодом на языке программирования Lua. Ошибки, встречающиеся во время выполнения, автоматически выводятся в стандартный поток вывода «stdout» на консоль.

Взаимодействие с редактором начинается с запуска исполняемого файла. Редактирование и исполнение кода внутри редактора происходит посредством использования предоставляемого графического интерфейса пользователя.

## Выбор средств разработки программного обеспечения

Одним из важнейших шагов перед началом разработки проекта является выбор средств, которые будут использоваться во время процесса разработки.

В качестве языка разработки библиотеки был выбран C++. Данный выбор обусловлен в целом двумя факторами. Первый – это язык, на котором можно работать напрямую с такими низкоуровневыми средствами, как WinAPI и OpenGL, необходимыми для осуществления данного проекта. Второй фактор – продолжительный опыт работы на этом языке участниками команды.

В качестве скриптового языка для интерпретирующей составляющей был выбран Lua. Связано это с легкостью его использования и подключения с помощью официального C API к коду на языке C или C++. Lua разрабатывался специально, как язык для внедрения в приложения, например, для поддержки модификаций без нужды влезать в исходный код на основном языке. Кроме того, что, по сути, самое важное, язык обладает рядом преимуществ для пользователей.

Lua имеет низкий порог вхождения. Из-за своей интерпретируемой сущности, ошибки не приводят к краху программы. Например, если обратиться к несуществующей переменной, то мы всего лишь получим значение nil, когда в языках, подобных C, программа даже не будет компилироваться. Также использование Lua не вызовет никаких затруднений у знакомых с программированием людей, так как он поддерживает несколько популярных парадигм программирования вроде функциональной (функции – объекты первого класса, т.е. мы можем передавать их в качестве параметров другим функциям) и объектно-ориентированной, в наличии имеется даже перегрузка операторов.

В качестве инструмента для разработки графического интерфейса редактора был выбран набор средств Qt, включая саму среду разработки Qt Creator. Данный выбор обусловлен тем, что Qt в целом популярный выбор для задач подобного рода, имеет хорошую документацию с описанием всех возможностей и встроенную в среду Qt Creator, а также обладает средством визуального проектирования интерфейса - Qt Designer (входит в состав Qt Creator). Кроме того, Qt поддерживает косметическую настройку внешнего вида интерфейса посредством таблиц стилей (аналогично CSS, используемому в web-разработке).

Как основная среда разработки (IDE) для библиотеки и интерпретатора была выбрана Microsoft Visual Studio Community 2019. Члены команды разработки проекта имеют большой опыт использования данной среды. Но, кроме этого, для выбора есть ещё несколько причин:

1. В состав IDE входит удобная система дополнения кода – IntelliSense. Она анализирует контекст и код, написанный вами ранее, и на основе тысяч проектов с GitHub предлагает вам в первую очередь наиболее вероятные дополнения. Например, при работе с классом vector из стандартной библиотеки на первом месте будут методы получения размера коллекции («size») и вставки новых элементов («push\_back»). Подобные выдачи помечаются звёздочкой.

2. Полезный отладчик с поддержкой отладки многопоточных программ и возможностью получить подробный отчёт об использованной динамической памяти (сколько и в какой месте программы память была взята/освобождена). Также в наличии умные точки останова, которые срабатывают только при выполнении определённых условий (например, можно приостановить выполнение, когда переменная i в цикле будет равна значению 5 и т.д.).

3. Кроме того, Visual Studio имеет набор функций косметического характера, вроде встроенных визуальных тем, утилиты для создания диаграммы классов и поддержки множества различных плагинов, скачиваемых и устанавливаемых из встроенного магазина «Visual Studio Marketplace».

На этом заканчивается наш выбор средств разработки. Пора приступать непосредственно к началу процессу проектирования.

## Модель бизнес-процессов

Следующим шагом будет построить модель бизнес-процессов проекта. В силу направленности проекта, наличия в команде лишь одного человека и отсутствия бизнеса как такого строить графическую схему бизнес-процессов не имеет смысла. Данный проект включает в себя только непосредственную линейную разработку программы.

В качестве модели жизненного цикла программного проекта была выбрана каскадная модель, которая предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке, а переход на следующий этап означает полное завершение работ на предыдущем этапе.

В соответствии с собранными требованиями программного средства был выявлен перечень этапов и работ (таблица 1), которые должны быть выполнены для успешного завершения разработки программного продукта.

Таблица 1 – Перечень этапов и работ

|  |  |  |
| --- | --- | --- |
| Наименование работы | Сроки | Ответственный |
| Изучение принципов работы с компьютерной графикой | 01.09.2019 – 30.10.2019 | Кулиненко В. И. |
| Изучение WinApi и средств, предоставляемых операционной системой Windows | 01.09.2019 – 30.09.2019 | Кулиненко В. И. |
| Изучение языка Lua | 30.09.2019 – 29.10.2019 | Кулиненко В. И. |
| Исследование аналогов проекта | 01.11.2019 – 29.11.2019 | Кулиненко В. И. |
| Проектирование программного интерфейса библиотеки | 30.11.2019 – 14.12.2019 | Кулиненко В. И. |
| Проектирование графического интерфейса редактора | 30.11.2019 – 19.12.2019 | Кулиненко В. И. |
| Разработка функционального прототипа библиотеки | 05.12.2019 – 31.12.2019 | Кулиненко В. И. |
| Разработка полноценной библиотеки | 01.01.2020 – 14.02.2020 | Кулиненко В. И. |
| Разработка оболочки Lua | 14.02.2020 – 04.03.2020 | Кулиненко В. И. |
| Разработка графической интерфейса редактора | 04.03.2020 – 28.03.2020 | Кулиненко В. И. |
| Предварительное тестирование | 15.03.2020 – 29.03.2020 | Кулиненко В. И. |
| Полноценное тестирование | 01.04.2020 – 30.04.2020 | Кулиненко В. И. |
| Исправление мелких ошибок и недочётов | 30.04.2020 – 14.05.2020 | Кулиненко В. И. |
| Оценка полученных результатов | 14.05.2020 – 31.05.2020 | Кулиненко В. И. |

На основе выявленного перечня этапов и их сроков была построена диаграмма Ганта (рисунок 1).

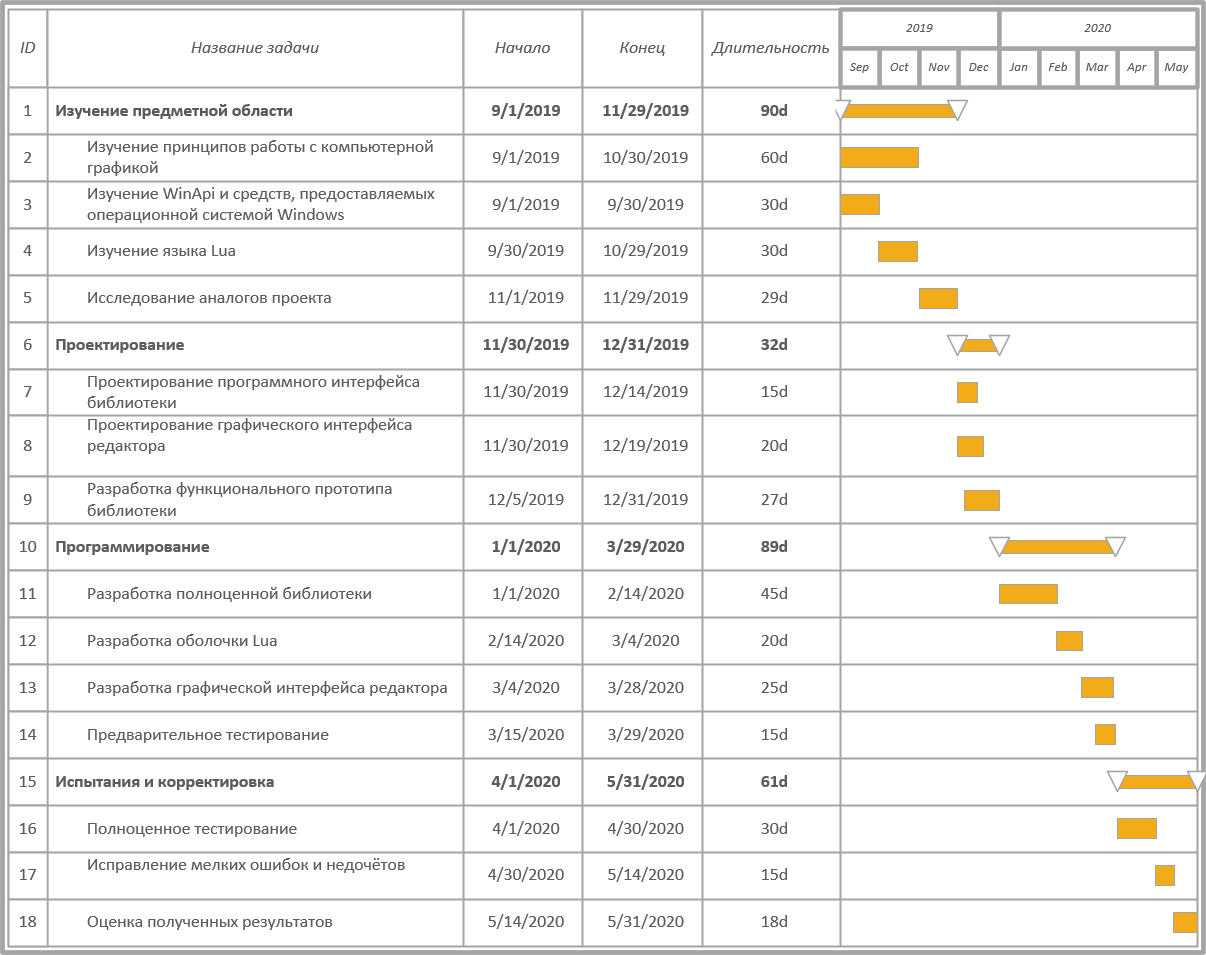


Рисунок 1 – Диаграмма Ганта с перечнем этапов и работ проекта

# Разработка программного обеспечения

Разработка программного обеспечения является важнейшей частью всего жизненного цикла проекта. Однако немедленное начало написания кода сразу после изучения технического задания является серьёзной ошибкой, так как до непосредственного начала разработки кода разработчик должен организовать корректную структуру программы и разбить программный комплекс на модули. Уже после, учитывая разработанную структуру, мы можем последовательно приступать к разработке модулей. Рассмотрим полученные архитектором модули программы и приступим к их реализации.

## Обзор архитектуры проекта

Следующий шаг после – обзор архитектуры, разрабатываемой программной среды. Сперва опишем структуру в общем виде и определим все задействованные модули. Диаграмма на рисунке 2 иллюстрирует архитектуру модулей нашего проекта.

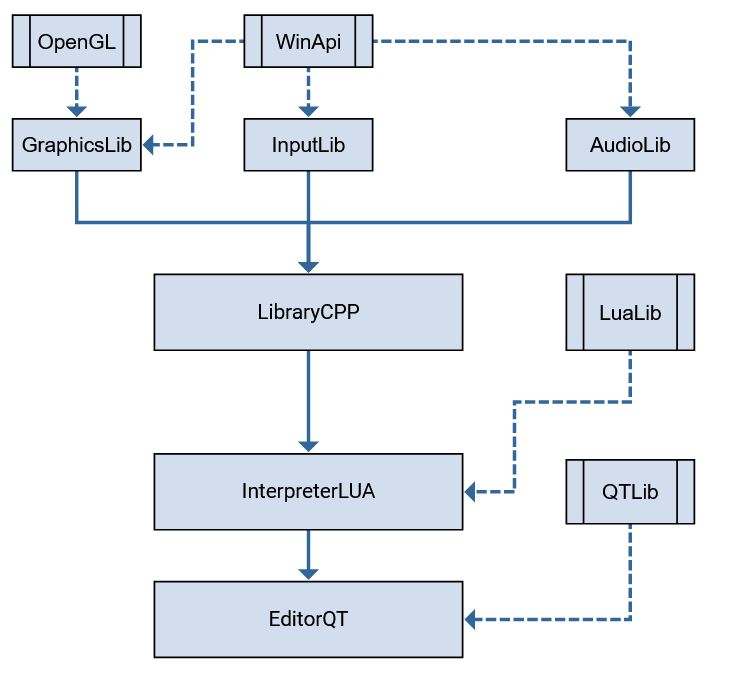
****

Рисунок 2 – Диаграмма программной архитектуры системы

Каждый блок обозначает программный модуль. Модули с двумя вертикальными чертами по краям, присоединяемые пунктирными линиями, не являются непосредственно частью процесса разработки нашего проекта, но необходимы в качестве зависимостей для основных модулей.

Кратко рассмотрим модули-зависимости:

OpenGL – это интерфейс для взаимодействия с функциями отрисовки примитивов видеокарты либо процессора. В целом, наравне с WinAPI, отвечает за вывод графической информации на окно программы.

WinAPI - набор библиотек и интерфейсов, предоставляемых операционной системой Windows. В их число входит управление окнами, обработка событий (вроде нажатия клавиш и закрытия окон), низкоуровневый функционал для воспроизведения звука.

LuaLib - интерфейс взаимодействия с функциями интерпретатора Lua из языков C/C++.

QTLib - набор библиотек для создания графического интерфейса для программ.

## Основные модули

Рассмотрим теперь модули, непосредственно разрабатываемые в рамках данного проекта.

**Модуль GraphicsLib**

Данный модуль отвечает за отрисовку на экран информации любого характера: прямые, треугольники, просто пиксели, текст, отдельные изображения. Также за масштабирование изображений и некоторые математические функции, вроде операций над векторами. В качестве зависимостей он задействует OpenGL для функций отрисовка и WinAPI для обновления содержимого окна.

**Модуль InputLib**

Данный модуль отвечает за считывание пользовательского ввода. В задачу входит фиксация нажатия кнопок на мыши и клавиатуре, положение курсора на экране и так далее. В качестве зависимостей используется WinAPI, поскольку взаимодействие с железом (мышью и клавиатурой) происходит только посредством функций операционной системы.

**Модуль AudioLib**

Данный модуль отвечает за загрузку и воспроизведение звуковых файлов формата «wav» без сжатия. Опять же в качестве зависимостей WinAPI, так как взаимодействие с средством вывода аудио возможно только посредством функций операционной системы.

**Модуль LibraryCPP**

Данный модуль представляет собой программную связь трёх модулей выше в единое целое, готовое к использованию в качестве библиотеки для программ на C++.

**Модуль InterpreterLUA**

Данный модуль отвечает за корректную интерпретацию кода Lua с задействованием функций, реализованных на языке C++ в модуле LibraryCPP. Представляет собой готовый исполняемый файл, принимаемый файл с Lua кодом через командную строку. В качестве зависимостей официальная библиотека Lua C API. Она обеспечивает взаимодействие с виртуальной машиной Lua, выполнение функций C из Lua и вызов функций Lua из C.

**Модуль EditorQT**

Данный модуль представляет собой редактор с графическим интерфейсом с использованием набора библиотек Qt. В число функций входит сохранение и загрузка кода, исполнение кода (с передачей файла с кодом в модуль InterpreterLUA), получение вывода из потока «stdout» приложения и отправка ввода в поток «stdin». А также набор стандартных горячих клавиш в виде копирования и вставки кода. В редакторе есть подсветка кода, индикатор номера для каждой строки и поддержка двух тем (светлая и темная).

Закончив с поверхностным рассмотрением архитектуры модулей перейдём к более точному её представлению – в виде диаграмм классов UML.

## Объектно-ориентированные модели

Перед непосредственным началом процесса кодирования была разработана диаграмма классов, представляющая классы и их связи, планируемые к реализации. Далее приведено краткое описание нотации UML.

На диаграмме полые стрелки обозначают отношение наследования, когда дочерний класс (источник стрелки) наследует свойства и методы родительского. Ещё один тип стрелок – композиция. Отношение изображается закрашенным ромбом и обозначает, что приёмник (тот, на чьей стороне ромб) состоит из экземпляров класса источника и управляет жизненным циклом этого объекта, то есть отвечает за его создание и уничтожение.

Так как большинство классов проекта содержат десятки свойств и методов, в данной работе было решено представить сокращенную версию диаграммы, изобразив только имена самих классов и характер их взаимоотношений. Диаграмма классов проекта представлена на рисунке 3.

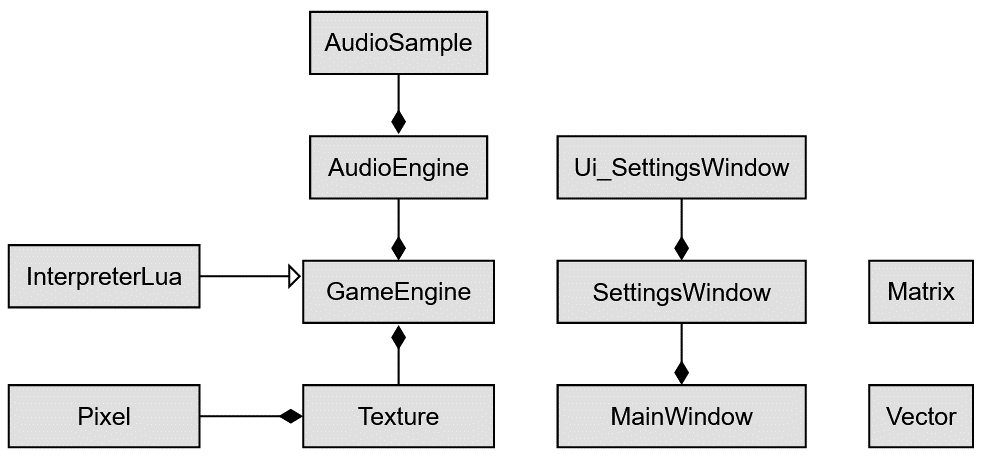


Рисунок 3 – Диаграмма классов проекта

Опишем функции и задачи каждого класса:

1. AudioSample – представляет собой готовый аудио фрагмент и содержит в себе всю информацию о файле. Отвечает за загрузку аудио из файлов формата wav при частоте дискретизации 44100 Гц и глубине дискретизации – 16 бит.
2. AudioEngine – отвечает за хранения, воспроизведение и корректное смешивание аудио фрагментов (экземпляров класса AudioSample).
3. Texture – изображение, по сути, массив пикселей (класс Pixel). Отвечает за модификацию массива, масштабирование изображения и загрузку его из файла «bmp».
4. Pixel – пиксель, или цвет, что то же самое. Содержит цвет в виде четырёх компонент RGBA: красная, зелёная, синяя и непрозрачность.
5. GameEngine – содержит функции по работе с текстурами, аудио, считыванием ввода. Отвечает за отрисовку примитивов и так далее.
6. InterpreterLua – интерпретатор языка Lua. Содержит «привязки» для основной библиотеки из класса GameEngine на этот язык.
7. Vector и Matrix – классы, отвечающие за реализацию векторов и матриц, а также функций по работе с ними: сложение, вычитание, умножение и так далее.
8. Ui\_SettingsWindow – класс, содержащий в себе элементы интерфейса и сгенерированный автоматически средством прототипирования графического интерфейса – Qt Designer.
9. SettingsWindow – класс, отвечающий за окно с настройками редактора, то есть за сохранение пути до исполняемого файла интерпретатора и выбор визуальной темы интерфейса.
10. MainWindow – класс, отвечающий за главное окно редактора. В нём обрабатываются горячие клавиши, открываются и редактируются файлы с кодом, запускается интерпретатор и так далее.

В данном разделе было сформировано представление о программной структуре проекта и, как итог, построена диаграмма классов.

Закончив с обзором архитектуры, перейдём к стадии моделирования пользовательского интерфейса.

## Моделирование пользовательского интерфейса

Следующим важным шагом идёт разработка пользовательского интерфейса. Пользователь будет взаимодействовать с программной средой посредством редактора, для которого мы создаём прототип интерфейса. Работа выполняется в среде Qt Designer.

Интерфейс состоит из двух окон. В основном окне пользователь открывает и работает с файлами с кодом, запускает интерпретатор, смотрит вывод из стандартного потока вывода в специальном окне и т.д. Во втором окне задаётся путь до интерпретатора и устанавливается визуальная тема: тёмная или светлая. Прототип обоих окон с текстовыми пояснениями демонстрируется на рисунках 4 и 5. Содержимое меню «File» показано на рисунке 6.

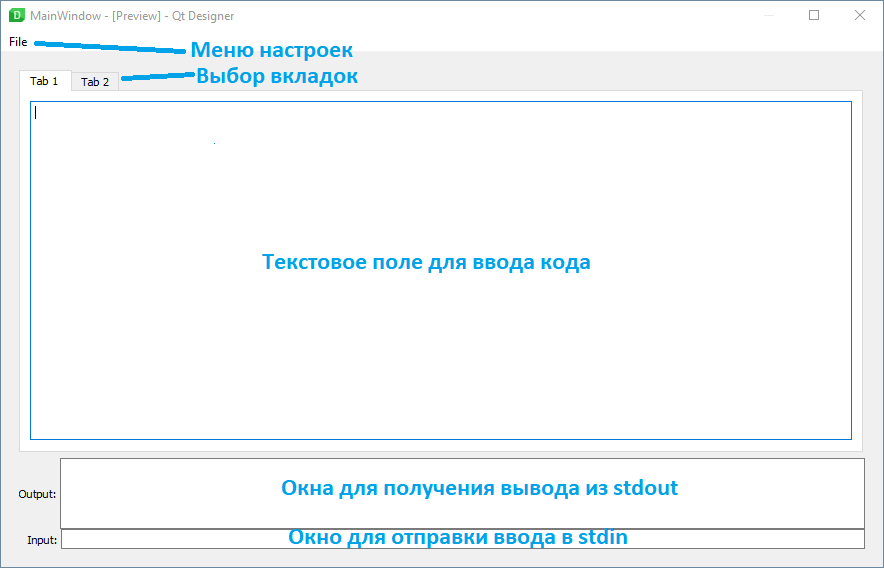


Рисунок 4 – Прототип основного окна редактора

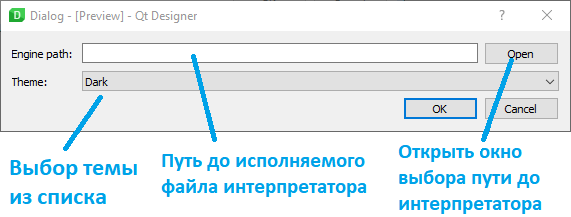


Рисунок 5 – Прототип окна с настройками редактора

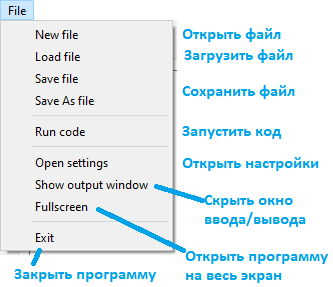


Рисунок 6 – Меню «File»

В результате данной работы был спроектирован окончательный вид графического интерфейса пользователя.

## Функциональное моделирование

В рамках проекта были задействованы некоторые алгоритмы, часть из них будет описана далее.

## Алгоритм Брезенхема для рисования прямой

Любое изображение на экране монитора представляет собой сетку пикселей. Чтобы что-либо нарисовать, нужно закрасить пиксели. Очевидно, что в современных (и не только) графических приложениях отрисовка происходит очень часто. Считается, что для плавности, изображение и анимация должны обновляться на экране как минимум 60 раз в секунду. Не трудно догадаться, что в таком случае процедуры, отвечающие за эту отрисовку, должны быть предельно эффективны.

Наиболее часто встречающаяся процедура и лежащая в основе всех примитивов – это отрисовка точки (одного пикселя). Но существует другая процедура, без которой не обходится ни один примитив – прямая. В простейшем описании прямая — это просто череда пикселей от точки начала до точки конца. Однако эта задача не так проста, как может показаться. Если мы рисуем не строго вертикальную или горизонтальную прямую, то в дело вступают операции с плавающей точкой, которые приносят с собой более высокую вычислительную сложность (относительно целочисленной арифметики) и эффект накопительной ошибки, связанный с тем, что в памяти компьютера все числа хранятся в двоичной системе счисления, в которой представление дробного числа обычно имеет определённую погрешность.

Итак, важным свойством алгоритмов отрисовки должна быть эффективность, скорость, производительность. Поставленную проблему решает так называемый алгоритм Брезенхема, придуманный ещё в 60-х годах прошлого века самим Брезенхемом. Отличительным свойством данного алгоритма является то, что среди математических операций он требует только целочисленное сложение и побитовый сдвиг, которые компьютеры выполняют очень быстро.

Рассмотрим классическое уравнение прямой: «y = kx + b» или, раскрыв угловой коэффициент: «y = (y1-y0)/(x1-x0)\*x + y0». Перебирая «x» в интервале от «x0» до «x1» мы получим корректную прямую (или, вернее, отрезок). Но в данном примере имеется затруднение, мы используем дроби и операцию деления, что будет относительно медленно выполняться на компьютерах.

Заметим, что переменная «y» (что очевидно) всегда должна изменяться не более, чем на один пиксель (если мы имеем дело с квадратной сеткой). Для дальнейшей работы введём такое понятие, как ошибка. Ошибка – это разница между текущей координатой «y» и теоретической, то есть той, что даёт нам математическая формула. Так как сетка пикселей квадратная, эта ошибка будет равна нулю только в случае строго вертикальной или горизонтальной прямой. Основная идея состоит в том, чтобы менять координату «y» на один пиксель только тогда, когда ошибка превышает единицу.

В алгоритме на каждой итерации мы будем увеличивать значение ошибки на коэффициент угла наклона. Превышение ошибки единицы будет значить, что пора увеличивать «y» на 1. Остаётся решить последнюю задачу – избавиться от операции деления.

Отличительной идеей, заложенной в алгоритм Брезенхема, является то, что коэффициент угла наклона можно представить не одним дробным числом, а как частное двух целых. Используя эту идею, мы можем просто умножить коэффициент угла наклона на изменение «x» (dx) и, в соответствии с этим, скорректировать саму ошибку в выражениях.

Остаётся последний момент. Как и описывалось самим Брезенхемом в его оригинальной статье, алгоритм симметричен. Мы прорабатываем его только для одного октанта (одна восьмая координатной сетки), а остальные получается изменением некоторых заданных конфигураций. Например, в некоторых случаях мы меняем местами точку начала прямой и точку конца.

Всё вышеперечисленное учтено в реализации алгоритма Брезенхема на языке Lua (листинге 1)

Листинг 1 – Алгоритм Брезенхема на языке Lua

function draw\_line(x0, y0, x1, y1)

local steep = false

if math.abs(y1 - y0) > math.abs(x1 - x0) then

x0,y0=y0,x0

x1,y1=y1,x1

steep = true

end

if x0>x1 then

x0,x1=x1,x0

y0,y1=y1,y0

end

local dy = math.abs(y1-y0)

local dx = (x1-x0)

local dir\_y = (y1-y0) > 0 and 1 or -1

local error = 0

local y = y0

for x=x0,x1,1 do

if not steep then

draw\_pixel(x, y)

else

draw\_pixel(y, x)

end

error = error + dy

if error > dx then

y = y + dir\_y

error = error - dx

end

end

end

## Алгоритмы интерполяции

Один из самых распространенных типов операций над изображениями – это масштабирование, то есть увеличение или уменьшение их размера. В данной части работы мы рассмотрим три алгоритма интерполяции, реализованных в нашем проекте.

Рассмотрим первый алгоритм: метод ближайших соседей. В этом алгоритме на цвет пикселя могу повлиять только 4 соседа: верхний, нижний, правый, левый, и только, если они существуют (например, у пикселя на границе может не быть одного или двух соседей). Но из этих 4 возможных выбирается только один, ближайший. Его цвет мы берём для нового пикселя. В отличие от других алгоритмов такой подход предельно прост в реализации и требует минимум затрат со стороны компьютера. С точки зрения внешнего восприятия, его использование имеет одно преимущество – он сохраняет структуру пикселей. Он хорошо подходит в пиксель-арте, так как при масштабировании подобного изображения оно не потеряет своей структуру, и не будет размыто, как это происходит при других методах (рисунок 1).

Данный алгоритм очень прост в реализации, но в масштабируемом изображении легко заметить прежнюю сетку пикселей. Зачастую бывает нужно сгладить изображение, чем и занимаются остальные методы.

Метод билинейной интерполяции так же, как и метод ближайших соседей, берёт 4 соседних пикселя, но, когда последний просто выбирает ближайшего из них, данный алгоритм работает над значениями всех четырёх.

С точки зрения работы алгоритм выглядит следующим образом. Производится линейную интерполяция между двумя верхними пикселями и отдельно с двумя нижними пикселями, а затем просто интерполируется эти два последних результата интерполяций между собой. Линейная интерполяция достаточно дешёвая процедура.

Третий метод – бикубическая интерполяция. Он показывает визуально лучший результат, по сравнению с билинейной, но при этом требует значительно больше ресурсов и зачастую не стоит своего использования.

Основная идея работы в следующим. В расчёт берутся теперь не 4, а все 16 пикселей вокруг. В каждом ряду происходит кубическая интерполяция, то есть строится полином третьей степени и находится значение функции в позиции пикселя. Потом эти 4 значения также интерполируется кубически. В итоге мы получаем значение пикселя, аппроксимируемое сразу по 16 точкам.

Ниже можно увидеть результат работы всех трёх алгоритмов при сжатии и увеличении изображения в 15 раз и сравнить это с оригиналом. Легко можно заметить отличие метода ближайших соседей от билинейной интерполяции, но уже сложнее между билинейной и бикубической. При последнем цвета более насыщенны.

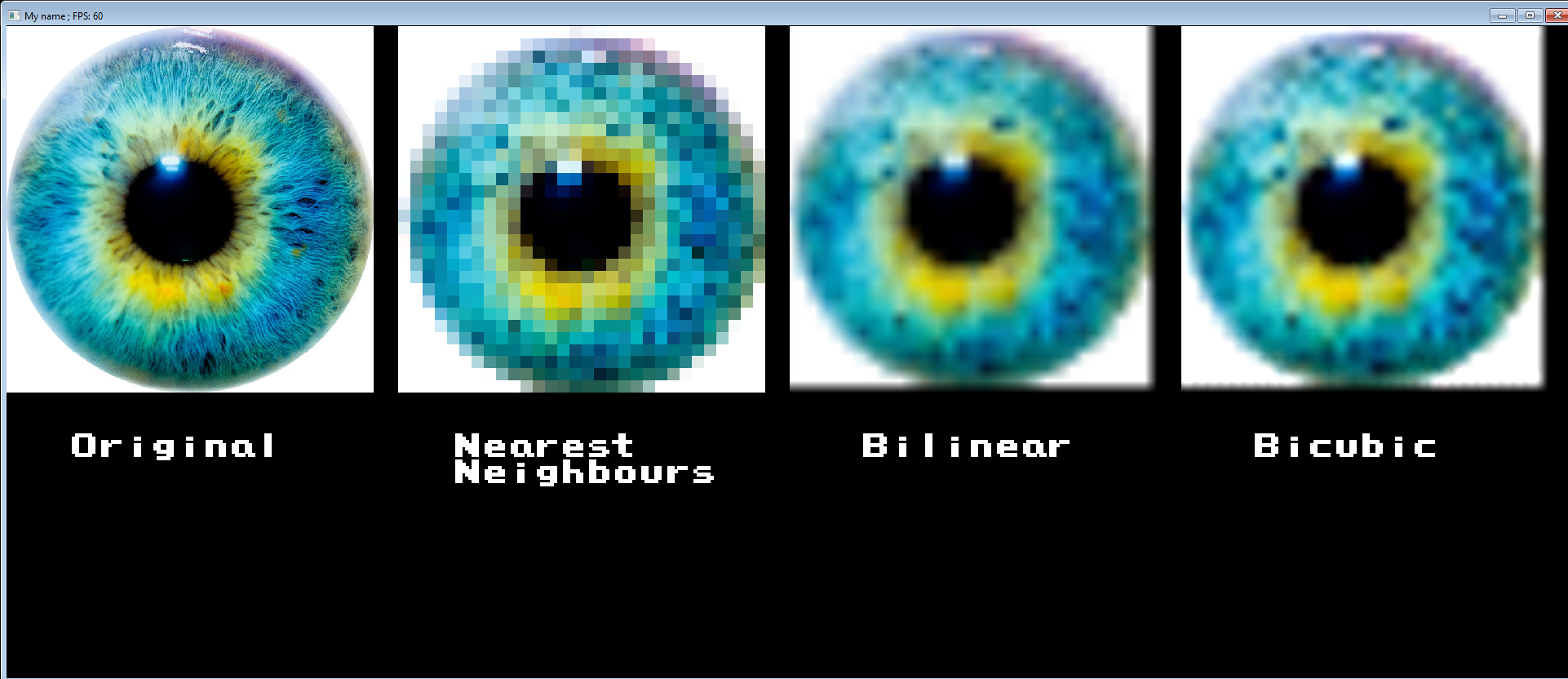


Рисунок 7 – Сравнение методов интерполяции после сжатия и увеличения изображения в 15 раз. Слева направо: оригинал, метод ближайших соседей, билинейная, бикубическая интерполяция.

## Описание разработанного программного продукта

По итогам разработки мы имеем проект, состоящий из трёх программных частей:

Библиотека на C++. Это вспомогательная библиотека по созданию интерактивных приложений под OS Windows. В её обязанности входит воспроизведение музыкальных файлов в формате «wav» (без сжатия), считывания ввода (кнопки клавиатуры и мыши, состояния нажата/отпущена/зажата, отслеживание координат курсора и т.д.), и, самое главное, отображение различных графических примитивов на экран монитора (прямые, прямоугольники, круги, текст, изображения в формате «bmp» и т.д.). В рамках библиотеки реализованы некоторый функционал по работе с математикой (например, векторы и матрицы). В основе работы стоят низкоуровневые «WinApi» и API «OpenGL». Библиотека может использоваться отдельно, однако в данном проекте она предназначена только как основа для интерпретатора.

Интерпретатор. Представляет собой интерпретатор и «обёртку» для языка Lua для функций из библиотеки. Может использоваться отдельно вместе с библиотекой. Представляет собой один исполняемый файл, который через командную строку получает путь до файла к исходному коду и исполняет его.

Редактор кода. Является несложным текстовым редактором с подсветкой кода. Есть поддержка горячих клавиш. Программа содержит меню с настройками (рисунок 1), где задаётся путь до исполняемого файла интерпретатора и выбирается графическая тема (тёмная или светлая, рисунок 2). В нижней части редакторе есть отдельная панель для получения/отправки информации из потоков stdout или stdin. То есть, если в программе будет строчка «print(123)», то при запуске программы число 123 попадёт в поток stdout и появится в соответствующем месте окна (рисунок 3).

Редактор поддерживает следующий функционал:

- загрузка и сохранение файлов с кодом;

- исполнение кода прямо из редактора;

- разворачивание окна «во весь экран»;

- показать/скрыть окно с stdin/stdout;

- номера строк;

- несколько открытых вкладок с кодом;

- увеличение/уменьшение размера шрифта (CTRL+колёсико мыши);

- комментирование/раскомментирование выделенной области (CTRL+/).

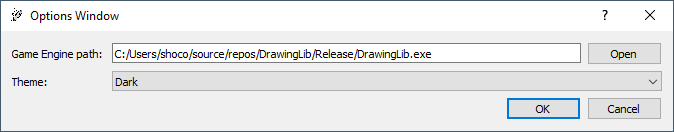


Рисунок 8 – Окно настроек

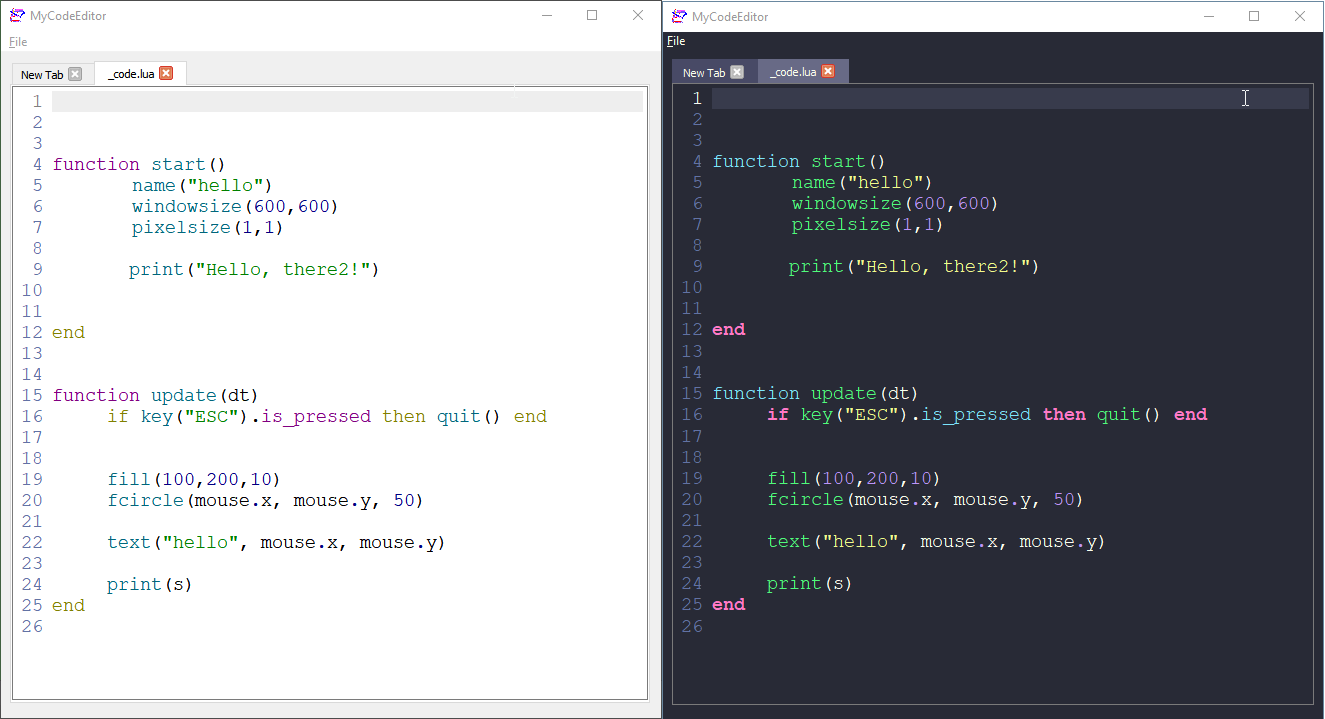


Рисунок 9 – Демонстрация двух тем интерфейса

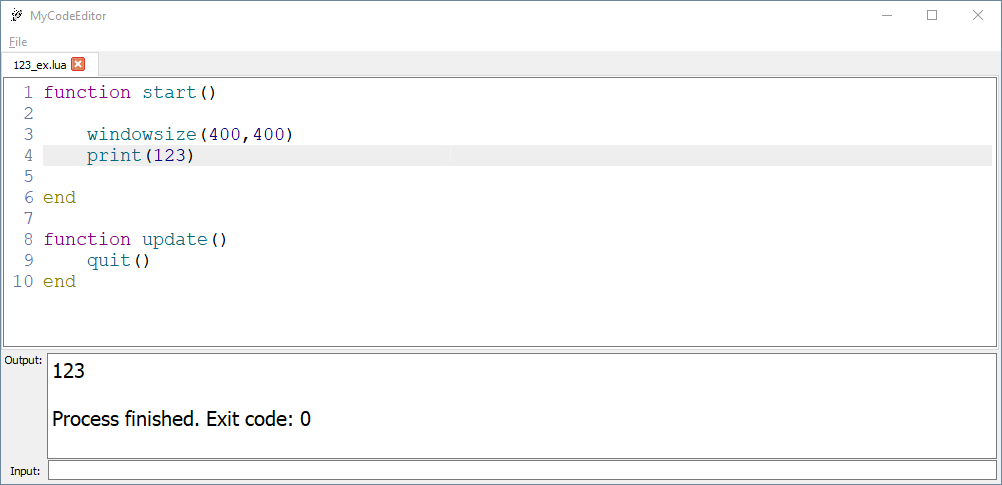


Рисунок 10 – отображение содержимого стандартного потока выхода (stdout) в нижней панели окна (output)

**ЗАКЛЮЧЕНИЕ**

В процессе разработки была создана программная среда по созданию интерактивных приложений, где был получен незаменимый опыт работы и выполнения всех задач в одиночку.

Погружение в предметную область компьютерной графики началось с разбора основных понятий в ней, что позволило начать её анализ. Проанализировав основные проблемы данной области, был составлен список требований и план их решения.

Далее был проведен анализ требований и на его основе было составлено техническое задание, подробно описывающее всю необходимую функциональность проекта. Мы провели сравнение с возможными аналогами и выявили их сильные и слабые стороны.

В начале разработки была составлена структура классов, а сама разработка была разбита на несколько модулей. Была реализована библиотека на C++, интерпретатор для неё на Lua и редактор кода с графическим интерфейсом пользователя. В процессе кодирования были реализованы алгоритм Брезенхема для рисования прямой и алгоритмы интерполяции: метод ближайших соседей, билинейная и бикубическая интерполяции. В завершение каждый модуль программной среды был тщательно протестирован и отлажен.

В итоге мы имеем программную среду для создания интерактивных приложений, в который любой желающий может творить или строить прототипы больших проектов. Несмотря на достаточно обширных функционал и выполнение всех требований, проекту есть куда расти, и мы планируем развивать его дальше. В планах улучшить производительность.