ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Пиксельная графика – это одно из направлений цифрового искусства. Отличительные черты таких работ – малое разрешение изображения и ограниченное количество цветов в палитре. В игровой индустрии этот стиль пользуется популярностью у небольших команд-разработчиков и используется для оформления визуального дизайна игры. В интернете существует множество источников, где публикуются как статические, так и анимированные изображения, что свидетельствует о популярности данного направления. Пиксельная графика, как стиль компьютерного искусства, весьма проста. Изображение формируется из вручную выставленных пикселей, что не требует освоения сложных инструментов. Поэтому для создания шедевров достаточно простого растрового графического редактора. Однако в простых редакторах отсутствует функция пикселизации исходных изображений, что является особенно актуальным для новичков, осваивающих этот стиль. Целью нашей работы является создание простого графического редактора с функцией пикселизации импортированных изображений. Под простотой понимается упрощённый процесс рисования – попиксельное составление изображения без использования сложных инструментов внутри редактора.

1 Анализ конкурентов

Приложение	Наш проект	Piskel	Grafx	Pixel Studio
Платформа	.NET, Mono	Web	Windows, Mac, Linux	Android
Расширенные возможности рисования (заливка, фигуры и т.д.)	Нет	Да	Да	Да
Особенности	Встроенный пикселизатор, позволяющий визуально снизить разрешение импортированных изображений	Не требует регистрации; Позволяет создавать GIF-анимации, регулируя скорость обновления кадров	Позволяет писать расширения для редактора на языке Lua	Позволяет создавать GIF-анимации; Поддерживает слои

Существует гораздо больше редакторов, поэтому среди многих нами были выбраны отдельные представители для каждой из основных платформ (desktop, web, mobile).

2 Целевая аудитория конечного продукта

Потребность в нашей программе может возникнуть у совершенно разных пользователей. В первую очередь это профессиональные pixel-art художники. Непрофессионалы могут оценить встроенный в программу пикселизатор и возможность преобразить, к примеру, пикселизованную фотографию.

3 Пример использования продукта

В современной культуре можно встретить множество работ, выполненных в pixel-art стиле, поэтому существует огромное количество примеров задач, когда может потребоваться соответствующий графический редактор. Но мы приведём пример задачи, для решения которой подходит именно наш редактор.

Представим себе, что имеется некий художник, состоящий в команде разработчиков видеоигры. Его задачей является создание набора текстур реально существующей местности, поэтому текстуры нужно рисовать либо с натуры, либо по фотографии. Визуальный дизайн игры стилизован под графику игр 90-х. Задача художника существенно упрощается, если с помощью нашего редактора он предварительно пикселизует фотографии, а после подкорректировать их ручным редактированием. Эти манипуляции займут гораздо меньше времени, чем рисование текстур с нуля.

4 Практическая часть

4.1 Анализ путей реализации

Web-версия

Плюсы:

- Не требует обновлений
- В большинстве случаев – платформонезависимость

Минусы:

- Зависимость от Интернет-соединения и его качества

Мобильная версия

Плюсы:

- Возможность пользоваться приложением в пути
- Возможность использовать стилус, не приобретая дополнительно графический планшет

Минусы:

- Неудобство в использовании сенсорного экрана без стилуса
- В большинстве случаев размер экрана устройства не удобен для рисования

Desktop-версия

Плюсы:

- Не зависит от Интернет-соединения

Минусы:

- Требуется портирование под каждую операционную систему (зависит он конкретной реализации)

Наш выбор – desktop-приложение на программной платформе .NET. Такая комбинация позволяет охватить классическую аппаратную платформу и большое количество операционных систем для неё.

4.2 Функционал законченного приложения

Конечный результат – это кроссплатформенное desktop-приложение на платформе .NET, реализующее возможности попиксельного рисования, импортирования и экспортирования изображения в форматах PNG, JPEG и GIF, и позволяющее также пикселизовать импортированное изображение.

4.3 Этапы разработки

Для успешного выполнения проекта нужно определить задачи, входящие в процесс его реализации (см. рис. 1). Приложение состоит из двух основных компонентов: редактор и модуль пикселизации. Логично будет вести параллельную разработку этих двух частей, объединив их на последнем этапе в одно приложение. Для каждой части определены свои основные задачи.

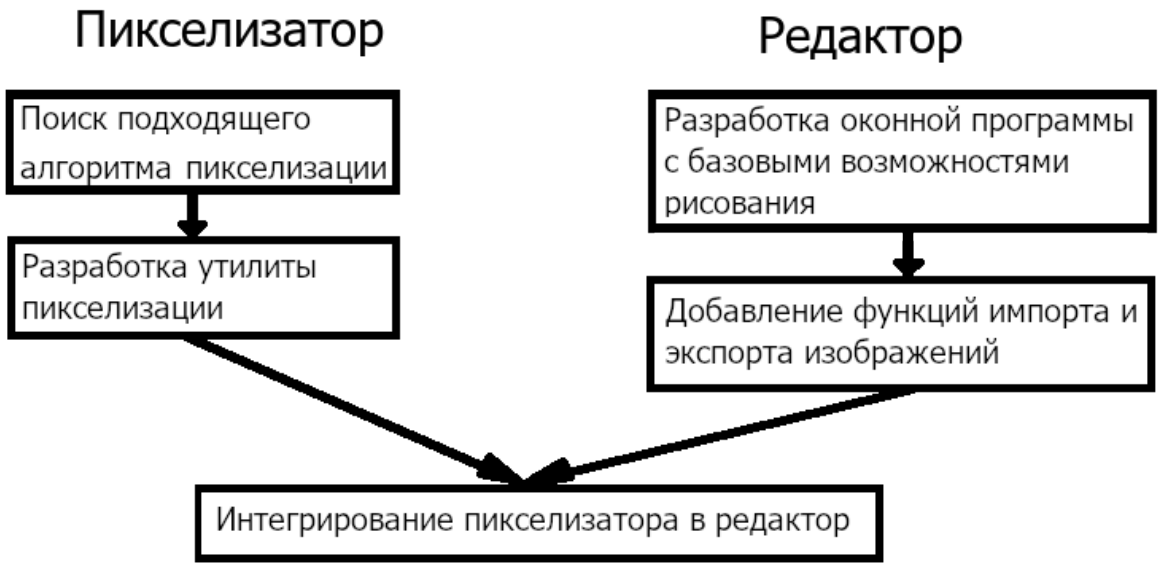

Рис. 1 – карта задач проекта

4.4 Обзор инструментария

Платформа .NET

Виртуальная машина *– программное обеспечение, эмулирующее архитектуру реально существующей или вымышленной аппаратной системы.*

Программная платформа .NET состоит из большого количества компонентов, позволяющих разрабатывать программы и целые программные продукты широкого спектра применения. Основной её компонент – виртуальная машина, которая является средой исполнения всех программ, разработанных под эту платформу. Это означает, что однажды написанная программа будет исполняться на любой платформе, если для неё есть реализация .NET. Главным примеров является Mono – открытая и кроссплатформенная реализация .NET. Благодаря Mono нашу программу можно запускать на широком спектре операционный систем: Windows, macOS и различные вариации BSD систем и Linux-дистрибутивов.

Однако выбор этой платформы не обусловлен тем, что приложения для неё являются кроссплатформенными. В самом деле, “кроссплатформенность” таких программ заканчивается там, где заканчивается поддержка Mono на конкретных операционных системах. В этом плане действительно кроссплатформенными можно назвать программы, написанные на языке Си, т.к. компиляторы для него имеются для всех существующих платформ. Таким образом, программная платформа выбрана нами исключительно на основании “привязанного” к ней языка программирования.

Язык программирования C#

C# является одним из языков, предназначенных для разработки приложений на платформе .NET. Для реализации проекта нашего типа выбор языка программирования не является существенным и обусловлен скоростью разработки и удобством отладки на нём [1]. Хотя споры и исследования о скорости разработки на различных языках ведутся давно, тем не менее, разработка на C# является наиболее комфортным условием для нашей команды.

Интегрированная среда разработки Microsoft Visual Studio

Интегрированная среда разработки *(англ. Integrated development environment - IDE) – это комплекс программных средств для разработки программного обеспечения.*

Файл исходного кода *(англ. source code) – текстовый файл, содержимое которого представляет программу на каком-либо языке программирования и необходимо для процесса сборки программы.*

Сборка *– процесс получения представления программы в наборе машинных инструкций путём обработки исходных кодов и других входных данных.*

При разработке на C#, как и на любом другом языке программирования, на выходе, до этапа сборки, получается некоторое количество исходных файлов, которые по своей сути являются текстовыми данными. Фактически разработка может вестись в любом текстовом редакторе. Но особенности языка C# таковы, что без автоматических подсказок редактора процесс написание кода будет сопровождаться регулярными поисками в документации стандартной

библиотеки языка. Поэтому, без IDE вести продуктивную разработку на C# практически невозможно. На выбор предоставляется 3 основных IDE для языка C#: Microsoft Visual Studio, IntelliJ IDEA и Monodevelop. По сути, все они предоставляют один и тот же функционал с несущественными различиями. Наш выбор – Microsoft Visual Studio как де-факто стандарт для разработки в ОС Windows (см. рис. 2).

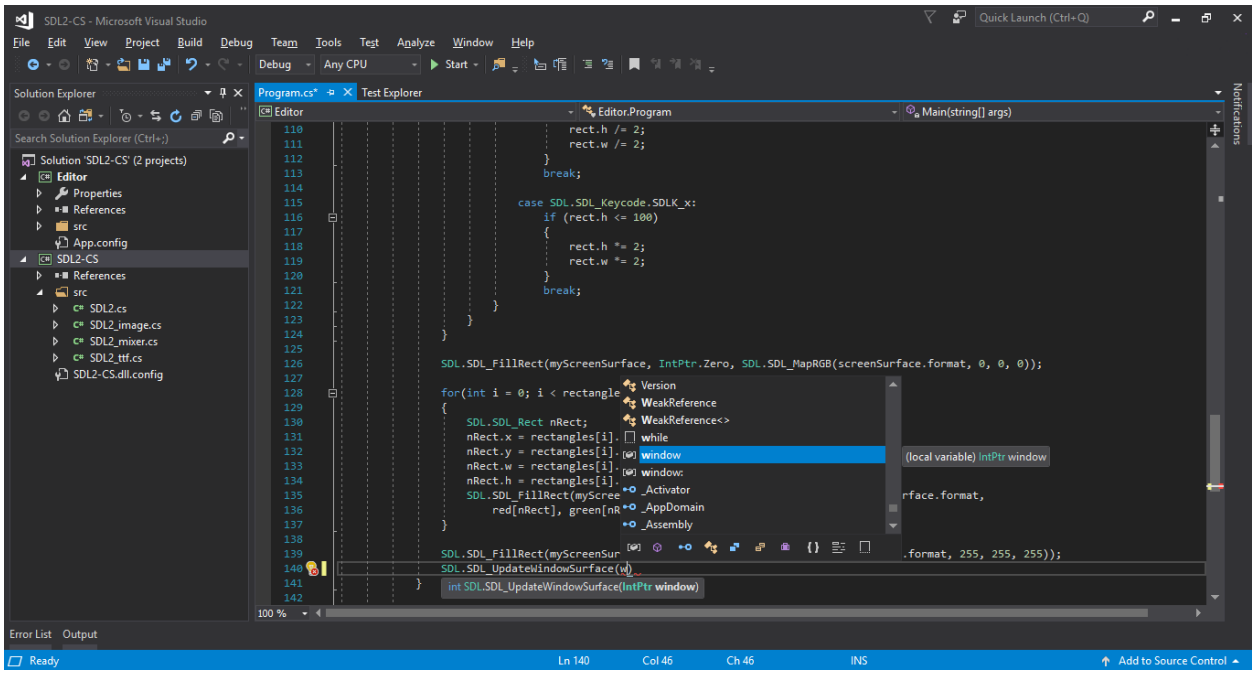

Рис. 2 - процесс написания кода в Visual Studio.

Библиотека SDL

GUI *(Graphical User Interface) – пользовательский интерфейс, построенный на основе интуитивно понятных графических элементов управления (кнопки, полосы прокрутки и т.д.).*

Библиотека *– сборник подпрограмм, используемых при разработке программного обеспечения*

SDL – кроссплатформенная мультимедийная библиотека, реализующая единый программный интерфейс к графической подсистеме, звуковым устройствам и средствам ввода для множества платформ [2]. Для нашего

проекта важной является исключительно графическая составляющая. С помощью этой библиотеки реализована область рисования.

Библиотека написана на языке программирования Си, но существует множество библиотек-обёрток (от англ. wrapper), обеспечивающих работоспособность библиотеки SDL в каком-либо языке, в котором прямой вызов функций этой библиотеки затруднителен или невозможен.

4.5 Макет

Процесс создания макета не имеет каких-либо особенностей. По сути, задача творческая.

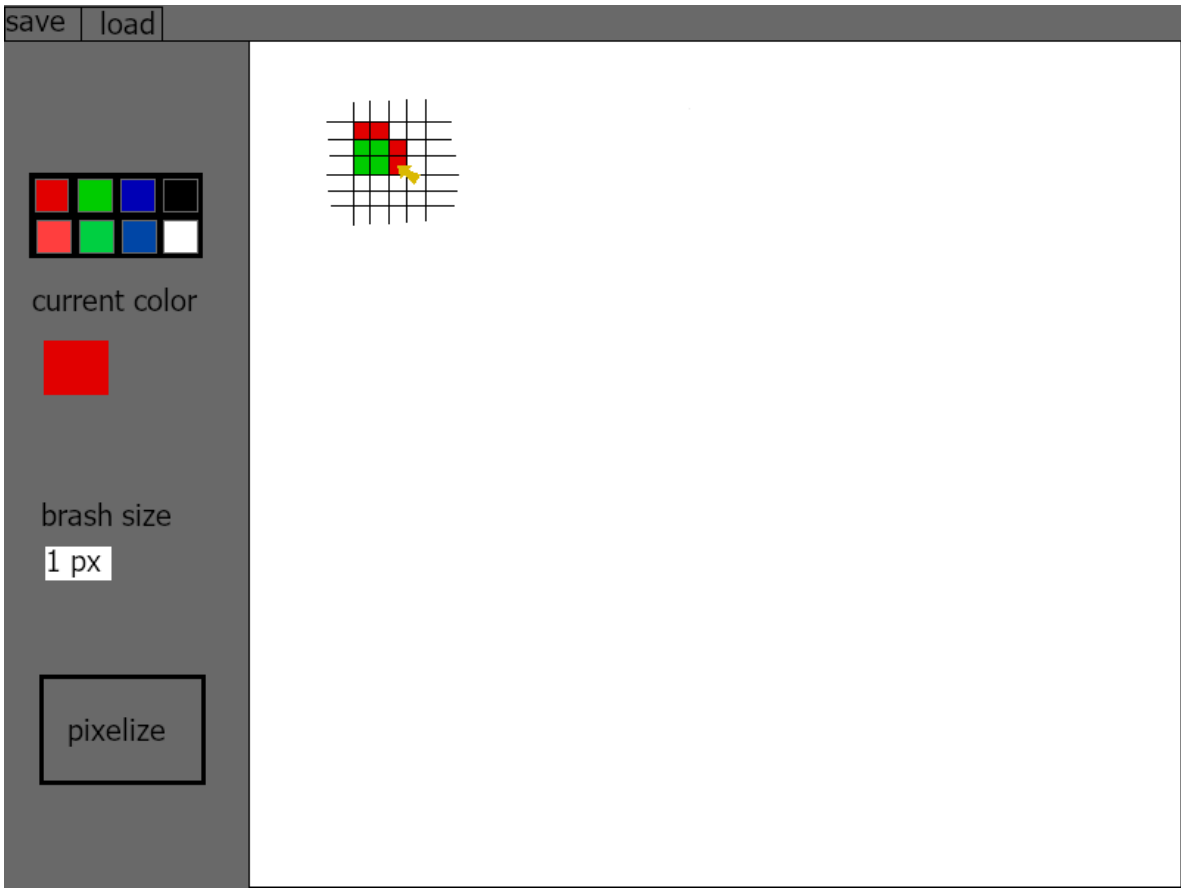

Рис. 3

Верхняя панель: кнопки сохранения и загрузки изображения. Боковая панель: палитра, текущий цвет, размер кисти, кнопка пикселизации. Область для рисования покрыта сеткой для разграничения пикселей (см. рис.3).

4.6 Прототип

Прототип не реализовывает весь функционал, представленный в макете (см. рис. 4).

Рис. 4 – поле для рисования. Курсор представлен белым пикселем.

Имеется поддержка красного, зелёного и синего цветов. Кисть перемещается клавишами на клавиатуре. Сохранение файла происходит автоматически при закрытии программы.

Обзор реализации прототипа

Для понимания необходимо знать, что библиотека SDL предоставляет несколько абстрактных объектов, композиция которых позволяет реализовывать множество различных мультимедийных возможностей. Например, для нашего прототипа необходимы SDL_Window (окно), SDL_Surface (плоскость) и SDL_Rect (прямоугольник). SDL_Window, по сути, представляет из себя саму оконную программу. Для SDL_Surface можно устанавливать размеры и привязывать к SDL_Window. В нашем случае SDL_Surface представляет собой плоскость для рисования, поэтому имеет размеры окна. SDL_Rect используется для представления пикселей на плоскости для рисования. Изначально плоскость содержит только один SDL_Rect, являющийся курсором, который можно перемещать. При нажатии определённых клавиш создаётся ещё один SDL_Rect, имеющий те же координаты, те же параметры высоты и ширины, что и курсор. Цвет нового прямоугольника определяется нажатой клавишей. Ниже приведён фрагмент кода, позволяющий при нажатии клавиши “1” создать на месте курсора новый пиксель зелёного цвета (см. рис. 5).

```
case SDL.SDL_Keycode.SDLK_1:
    SDL.SDL_Rect newRect;
    newRect.x = rect.x;
    newRect.y = rect.y;
    newRect.h = rect.h;
    newRect.w = rect.w;
    rectangles.Add(newRect);
    red[newRect] = 255;
    green[newRect] = 0;
    blue[newRect] = 0;
    break;
```

Рис. 5 - создание нового пикселя.

Из фрагмента кода можно увидеть, что при каждом нажатии кнопки для рисования данные помещаются в 4 объекта-коллекции, а именно: лист rectangles и 3 словаря – red, green и blue (см. рис. 6).

```
var rectangles = new List<SDL.SDL_Rect>();
var red = new Dictionary<SDL.SDL_Rect, byte>();
var green = new Dictionary<SDL.SDL_Rect, byte>();
var blue = new Dictionary<SDL.SDL_Rect, byte>();
```

Рис. 6 - инициализация объектов-коллекций.

Лист rectangles содержит в себе все пиксели изображения. Программа исполняется в бесконечном цикле (до тех пор, пока пользователь не закроет окно программы), поэтому на каждом шаге этого цикла программа отрисовывает всё содержимое окна. Редактируемое изображение отрисовывается попиксельно, т.е. необходим цикл по коллекции, на каждой итерации которого отрисовывается конкретный пиксель (см. рис. 7).

```
for(int i = 0; i < rectangles.Count; ++i)
{
    SDL.SDL_Rect nRect;
    nRect.x = rectangles[i].x;
    nRect.y = rectangles[i].y;
    nRect.w = rectangles[i].w;
    nRect.h = rectangles[i].h;
    SDL.SDL_FillRect(myScreenSurface, ref nRect, SDL.SDL_MapRGB(screenSurface.format,
        red[nRect], green[nRect], blue[nRect]));
}
```

Рис. 7 - функция SDL_FillRect() помещает пиксель на плоскость редактируемого изображения.

Словари red, green и blue привязывают каждому пикселю свой цвет для красного, зелёного и синего значений.

4.7 Пикселизация изображения

Одна из функций, необходимых к реализации – это возможность преобразовать исходное изображение в его пикселизованный вид. Для этого был выбран алгоритм Брезенхэма [3]. Алгоритм состоит из следующих этапов [4]:

1) Исходное изображение делится на прямоугольники (квадраты), тем самым создавая матрицу (рис. 8).

```
for (int x = 0; x < source.Width; x += this.BlockSize)
{
    for (int y = 0; y < source.Height; y += this.BlockSize)
    {
        var sums = new Sums();

        for (int xx = 0; xx < this.BlockSize; ++xx)
        {
            for (int yy = 0; yy < this.BlockSize; ++yy)
```

Рис. 8

2) В матрице среди всех пикселей находят “средний” цвет, затем весь сектор заменяется на данную цветовую гамму (рис. 9).

```
var average = Color.FromArgb(
    sums.A / sums.T,
    sums.R / sums.T,
    sums.G / sums.T,
    sums.B / sums.T);
```

Рис. 9

После окрашивания всех секторов на выходе мы получаем пикселизованное изображение.

4.8 Конечная реализация

Финальный результат представляет из себя двухоконное приложение (см. рис. 10), т.к. представилось невозможным объединить в одном окне SDL-контекст и элементы графического интерфейса. Одно окно является областью для рисования, а другое – панелью инструментов. Нам не удалось реализовать задачу полностью: область рисования имеет ограниченное разрешение 800x600 пикселей; отсутствует сетка разграничения пиксельных сегментов; любое изображение растягивается или сжимается до размеров 800x600 пикселей, т.к. не реализован вертикально-горизонтальный скроллинг; отсутствует альфа-канал – любое созданное изображение будет иметь чёрный фон, а не прозрачный.

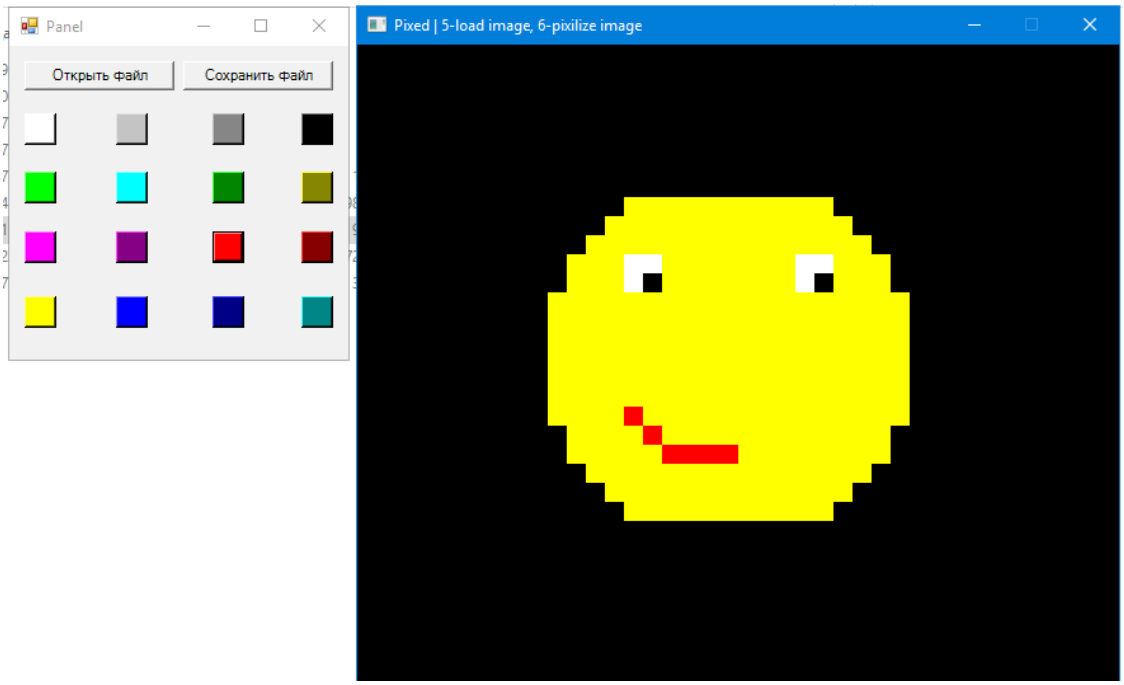

Рис.10

Панель инструментов содержит палитру с ограниченным набором цветов, кнопки вызова диалогового окна для импорта и экспорта изображения [5]. Вызов пикселизатора осуществляется отдельной клавишей на клавиатуре.

4.9 Примеры изображений

Приведём несколько изображений, созданных с помощью нашего проекта.

1)

Рис. 11 – Цветы в вазе.

2)

Рис. 12 – До пикселизации.

Рис. 13 – После пикселизации.

3)

Рис. 14 – Граффити.

ЗАКЛЮЧЕНИЕ

Нам не удалось реализовать весь запланированный функционал из-за неправильного распределения рабочего времени, ошибочной оценки своих возможностей, неоптимального выбора стека технологий и плохой кодовой базы приложения. Главной причиной всего указанного является отсутствие опыта в проектной деятельности и малый опыт разработки у каждого члена команды.

Выбор библиотеки SDL был ошибочен. Вся графическая составляющая могла бы быть реализована быстрее, удобнее и в одном оконном приложении с помощью библиотеки Windows Forms. Отсутствие должного проектирования архитектуры приложения не позволило представить в графическом интерфейсе вызов пикселизатора, импорт изображения также требует дополнительного нажатия клавиши на клавиатуре.

Редактор не подходит для профессионального использования, но всё ещё является подходящим выбором для начинающих осваивать пиксель-арт.

Что касается изученного в ходе работы, то теперь мы имеем представление о том, что разработка программ, утилизирующих какие-либо графические возможности компьютеров, требует предварительного опыта разработки. Также мы поняли, что требуется тратить больше времени на выбор и анализ технологий и средств разработки, а проектирование является обязательным этапом в разработке программного обеспечения. Анализ и проектирование – то, от чего зависит время выполнения и результат.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дают ли новые языки программирования прирост скорости разработки // Хабр URL: https://habr.com/ru/post/308550/ (дата обращения: 20.03.19).
2. Статьи по библиотеке SDL // Lazy Foo' Productions URL: http://lazyfoo.net/tutorials/SDL/index.php (дата обращения: 03.04.2019).
3. Bresenham's line algorithm // Wikipedia URL: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm (дата обращения: 10.04.2019).
4. Реализация фильтра пикселизации изображения на C# // Stack Overflow URL: https://stackoverflow.com/questions/1987054/whats-a-good-pixelation-algorithm-in-c-sharp-net (дата обращения: 10.04.2019).
5. Руководство по программированию в Windows Forms // METANIT.COM Сайт о программировании URL: http://metanit.com/sharp/windowsforms/ (дата обращения: 10.05.2019).