МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 03 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ**

по дисциплине «Защита информации и надежность информационных систем»

Тема ««Реализация стеганографического метода LSB \*.gif»

Исполнитель

|  |  |  |
| --- | --- | --- |
| cтудент 3 курса группы 2    Руководитель |  | \_\_\_\_\_\_\_\_\_\_\_\_ В.Д. Ильинов  подпись, дата |
| ассистент |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.В. Копыток  подпись, дата |
| Курсовой проект защищен с оценкой |  | \_\_\_\_\_\_\_\_\_\_\_\_ |
| Руководитель \_\_\_\_\_\_\_\_\_\_ |  | Д.В. Копыток |

подпись

Минск 2021

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПЕУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра информационных систем и технологий

Утверждаю

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Смелов

подпись инициалы и фамилия

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г.

ЗАДАНИЕ

**к курсовому проектированию по дисциплине**«Защита информации и надежность информационных систем»

Специальность: Информационные системы и технологии

Группа: 2

Студент: Ильинов Владислав Дмитриевич

**Тема:** Реализация стеганографического метода LSB \*.gif

**1. Срок сдачи студентом законченной работы**: «05» мая 2021г.

**2. Исходные данные к проекту:**

**2.1**. Функционально должны быть выполнены следующие задачи:

* + Реализация програмного средства для сокрытия информации методом LSB в \*.gif.
  + Составить руководство пользователя.

**2.2. Требования:**

* + Необходимо провести аналитический обзор литературы по теме проекта
  + Необходимо описать сферу применимости метода
  + Программное средство может быть разработано на любом языке
  + Архитектура приложения выбирается разработчиком
  + Листинги проекта должны содержать комментарии

**3. Содержание расчетно-пояснительной записки:**

* + Введение
  + Постановка задачи
  + Описание метода
  + Описание программного средства
  + Тестирование программного средства
  + Руководство пользователя
  + Заключение
  + Список используемых источников
  + Приложения

**4. Форма представления выполненного курсового проекта:**

* + Теоретическая часть курсового проекта должна быть представлена в формате MS Word.
  + Оформление записки должно быть согласно правилам.
  + Необходимые схемы, диаграммы и рисунки допускается делать в MS Office Visio или копии экрана (интерфейс).
  + Полные листинги проекта представляются в приложении.
  + К записке необходимо приложить CD (DVD), который должен содержать: пояснительную записку, листинги и файлы базы данных.

Календарный план

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Введение | 17.02.2021 |  |
| 2 | Аналитический обзор литературы по теме проекта | 24.02.2021 |  |
| 3 | Разработка метода | 10.03.2021 |  |
| 4 | Разработка прототипа программного средства | 17.03.2021 |  |
| 5 | Разработка программного средства | 24.03.2021 |  |
| 6 | Тестирование программного средства | 07.04.2021 |  |
| 7 | Написание руководства пользователя | 14.04.2021 |  |
| 8 | Оформление пояснительной записки | 28.04.2021 |  |
| 9 | Сдача проекта | 05.05.2021 |  |

**5. Дата выдачи задания «17» февраля 2021г.**

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Д. В. Копыток*

(подпись)

Задание принял к исполнению \_

(дата и подпись студента)

Оглавление

[Введение 5](#_Toc72434713)

[1. Теоретическое введение 6](#_Toc72434714)

[2. Проектирование 8](#_Toc72434715)

[2.1. Постановка задачи 8](#_Toc72434716)

[2.2. Алгоритмы решения 8](#_Toc72434717)

[3. Реализация программного средства 9](#_Toc72434718)

[3.1. Архитектура программного средства 9](#_Toc72434719)

[3.2. Реализация 9](#_Toc72434720)

[3.2.1. Метод LSB 9](#_Toc72434721)

[3.2.2. Метод расширения палитры 13](#_Toc72434722)

[3.2.3. Вспомогательные функции 15](#_Toc72434723)

[3.3. Реализация графического интерфейса 17](#_Toc72434724)

[4. Тестирование 21](#_Toc72434725)

[5. Руководство по использованию 24](#_Toc72434726)

[Заключение 29](#_Toc72434727)

[Список используемых источников 30](#_Toc72434728)

[Приложение А 31](#_Toc72434729)

[Приложение Б 34](#_Toc72434730)

# Введение

Проблема защиты информации от несанкционированного доступа возникла еще в древние времена, и с тех пор выделилось два основных направления решения этой проблемы, которые существуют и сегодня: криптография и стеганография. Задачей криптографии является скрытие информации, содержащейся в сообщении, за счет его шифрования, а стеганография (пер. с греч, "тайнопись") – это наука о скрытой передаче информации путём сохранения в тайне самого факта передачи. Главная задача сделать так, чтобы человек не подозревал, что внутри передаваемой информации, внешне не представляющей абсолютно никакой ценности, содержится секретная информация. Тем самым стеганография позволяет передавать важную информацию через открытые каналы, скрывая сам факт её передачи. В отличие от криптографии, которая скрывает содержимое секретного сообщения, стеганография скрывает само его существование. Стеганографию обычно используют совместно с методами криптографии, таким образом, дополняя её.

Компьютерная стеганография позволяет срыть информацию внутри медиафайлов, то есть файлов, которые содержат изображение, звук или текст. Такие файлы называются стеганографическими контейнерами (файлами-контейнерами), а скрытые сообщения – стеганосообщениями. Современный прогресс в области глобальных компьютерных сетей и средств мультимедиа привел к разработке новых методов, предназначенных для обеспечения безопасности передачи данных по каналам телекоммуникаций и использования их в необъявленных целях. Эти методы, учитывая естественные неточности устройств оцифровки и избыточность аналогового видео или аудио сигнала, позволяют скрывать сообщения в компьютерных файлах (контейнерах)

Целью данного курсового проекта является разработка программного средства, позволяющего на стороне отправителя встроить передаваемое сообщение в GIF-контейнер одним из стеганографических методов: LSB и метода расширения палитры – и извлечь сообщение на стороне получателя.

# Теоретическое введение

Стеганография – это наука о скрытой передаче информации путём сохранения в тайне самого факта передачи. Главная задача сделать так, чтобы человек не подозревал, что внутри передаваемой информации, не представляющей внешне абсолютно никакой ценности, содержится скрытая ценная информация. Тем самым стеганография позволяет передавать секретную информацию через открытые каналы, скрывая сам факт её передачи. Наиболее распространенный класс контейнеров — это мультимедийные файлы. Большие объемы таких файлов-контейнеров позволяют упаковывать в них значительные по размеру сообщения, а разнообразные, постоянно совершенствующиеся форматы и стандарты обусловили появление множества стеганографических алгоритмов. В простых некоммерческих программах для стеганографии в качестве контейнера нередко используют области графических файлов, изменение которых не влияет на изображение. Скрываемая информация может размещаться и после окончания данных изображения, и между отдельными картинками одного файла, и в полях-комментариях, которые игнорируются при прорисовке. Такие стеганограммы легко детектируются, поэтому обычно они рассчитаны на то, что специально их никто искать не будет.

Выделяют несколько направлений стеганографии:

* классическая стреганография;
* цифровая стеганография;
* компьютерная стеганография [1].

GIF (англ. Graphics Interchange Format — формат для обмена изображениями) — формат хранения графических изображений, способен хранить сжатые данные без потери качества в формате до 256 цветов. Файлы формата GIF имеют блочную структуру. Данные блоки всегда имеют фиксированную длину (либо она зависит от некоторых флагов), так что ошибиться в том, где какой блок находится, практически невозможно. Структура простейшего неанимированного GIF-изображения формата GIF89a изображена на рисунке ниже 1.1 [2].

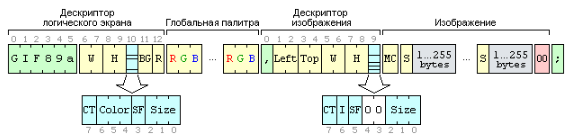


Рисунок 1.1 – Структура GIF

Блок глобальной палитры содержит параметры CT и Size. CT указывает на наличие глобальной палитры. Если этот флаг установлен, то сразу после дескриптора логического экрана должна начинаться глобальная палитра. Size — размер палитры и число цветов картинки. Параметр Size имеет значение от 0 до 7. Таким образом, количество цветов равно 2 в степени значения Size.

Метод LSB — распространенный метод стеганографии. Он заключается в замене последних значащих бит в контейнере на биты скрываемого сообщения. В программе будут использоваться в рамках этого метода два последних бита в байтах глобальной палитры. Это означает, что для 24-битного изображения, где цвет палитры представляет собой три байта для красного, синего, и зеленого цветов, после внедрения сообщения в него, каждая составляющая цвета изменится максимум на 3/255 градации. Такое изменение, во-первых, будет незаметно или труднозаметно для человеческого глаза, а во-вторых, не будет различимо на низкокачественных устройствах вывода информации. Количество информации будет напрямую зависеть от размера палитры изображения. Поскольку максимальный размер палитры 256 цветов и, если записывать по два бита сообщения в составляющую каждого цвета, то максимальная длина сообщения (при максимальной палитре в изображении) составляет 192 байта. После внедрения сообщения в изображение, размер файла не изменяется [3].

Метод расширения палитры, работающий только для структуры GIF, будет наиболее эффективен в изображениях с палитрой небольших размеров. Суть его состоит в том, что он увеличивает размер палитры, тем самым дав дополнительное пространство для записи необходимых байт на месте байт цветов. Если учесть, что минимальный размер палитры составляет 2 цвета (6 байт), то максимальный размер внедряемого сообщения может быть 256×3–6=762 байт. Палитра из N цветов определяется как список пар индексов (i, ci), который определяет соответствие между индексом i и его вектором цветности ci. В изображении каждому пикселю присваивается индекс в палитре. Так как цвета в палитре не всегда упорядочены, то скрываемую информацию можно кодировать последовательностью хранения цветов в палитре. Существует N! различных способов перестановки N-цветной палитры, что вполне достаточно для сокрытия небольшого сообщения. Недостаток — низкая криптозащищенность, прочесть внедренное сообщение можно при помощи любого текстового редактора, если сообщение не подвергалось шифрованию [4].

# Проектирование

## Постановка задачи

Функционально должны быть выполнены следующие задачи:

* разработать алгоритм для сокрытия и извлечения сообщения из GIF-контейнера методом LSB;
* разработать алгоритм для сокрытия и извлечения сообщения из GIF-контейнера методом расширения палитры;
* создать программное средство, реализующее разработанные алгоритмы;
* составить руководство по использованию программное средства.

## Алгоритмы решения

При разработке приложения были использованы технологии:

* интегрированная среда разработки Microsoft Visual Studio 2019;
* WPF (Windows Presentation Foundation).

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Silverlight. Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня.

Windows Presentation Foundation — система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML. WPF представляет собой обширный API-интерфейс для создания настольных графических программ имеющих насыщенный дизайн и интерактивность. В отличие от устаревшей технологии Windows Forms, WPF включает новую модель построения пользовательских приложений (в основе WPF лежит мощная инфраструктура, основанная на DirectX). Это означает возможность применения развитых графических эффектов, не платя за это производительностью, как это было в Windows Forms. Фактически даже становятся доступными такие расширенные средства, как поддержка видеофайлов и трехмерное содержимое.

# Реализация программного средства

## Архитектура программного средства

Программное средство написано на языке C# и представлено одним проектом, структура которого представлена на рисунке 3.1.

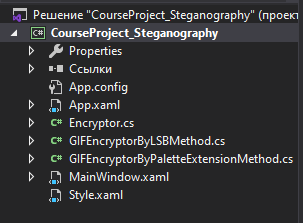


Рисунок 3.1 – Структура проекта

Описание каждого элемента в структуре проекта приведено в таблице 3.1.

Таблица 3.1 – Описание структуры проекта

|  |  |
| --- | --- |
| Properties | Свойства проекта, содержит информацию о сборке, используемых ресурсах и настройках |
| Ссылки | Перечень сборок, используемых в проекте |
| App.config | Файл с параметрами проекта |
| App.xaml | Класс Application |
| Encryptor.cs | Класс со вспомогательными методами |
| GIFEncryptorByLSBMethod.cs | Класс метода LSB |
| GIFEncryptorByPaletteExtensionMethod.cs | Класс метода расширения палитры |
| MainWindow.xaml | Главное окно |
| Style.xaml | Файл с описанием дополнительных стилей |

В данной структуре видно, что основные действия выполняются в 3 классах.

## Реализация

### Метод LSB

Алгоритмы внедрения и извлечения сообщения методом LSB описаны в классе GIFEncryptorByLSBMethod, структура которого представлена на рисунке 3.2. В поле checkSequence хранится проверочная последовательность, с помощью которой обнаруживается сообщение в контейнере. В полях firstLBit и secondLBit хранятся первый и второй незначимые биты. Метод Encrypt отвечает за внедрение сообщения в GIF-контейнер, метод Decrypt – за извлечение сообщения из контейнера.

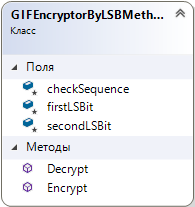


Рисунок 3.2 – Структура класса GIFEncryptorByLSBMethod

Метод Encrypt (string input, string text) принимает в качестве параметров путь к контейнеру и сообщение в виде строки.

Для работы с GIF-файлом его необходимо преобразовать в байты. Код записи сообщения представлен ниже на рисунке 3.3.



Рисунок 3.3 – Преобразование файла в байты

Первым делом мы узнаем размер палитры контейнера. Размер, согласно структуре GIF, заключен в первых трех битах десятого байта файла. Это можно наблюдать в коде, который представлен ниже на рисунке 3.4.

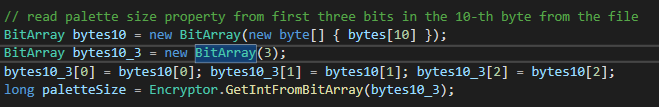


Рисунок 3.4 – Чтение размера палитры контейнера

Зная размер палитры, можно рассчитать длину сообщения как ¾ от количества цветов в файле. Длина текста сообщения будет равна длине сообщения с вычетом двух символом: один – для проверки, второй – хранящий длину текса. Запись сообщения начинается с тринадцатого байта контейнера: сначала записывается проверочная последовательность, затем длина текста сообщения, наконец само сообщение. Код расчёта длины сообщения представлен на рисунке 3.5.

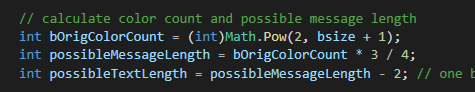


Рисунок 3.5 – Расчёт размера сообщения

Код записи сообщения представлен на рисунке 3.6.

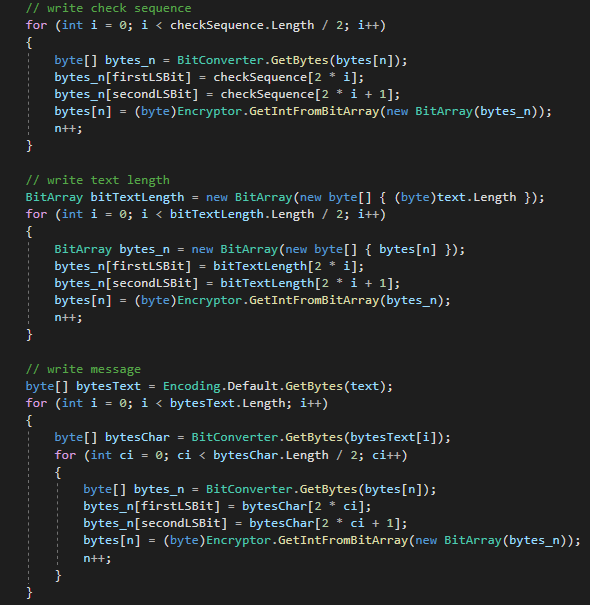


Рисунок 3.6 – Запись сообщения в контейнер

Также необходимо проверить соответствие необходимому в приложении формату. Код проверки представлен на рисунке 3.7.

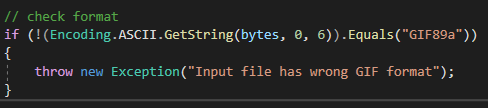


Рисунок 3.7 – Проверка формата сообщения

Метод Decrypt (string input) принимает в качестве параметра путь к файлу, хранящему сообщение. Код метода представлен на рисунке 3.8.

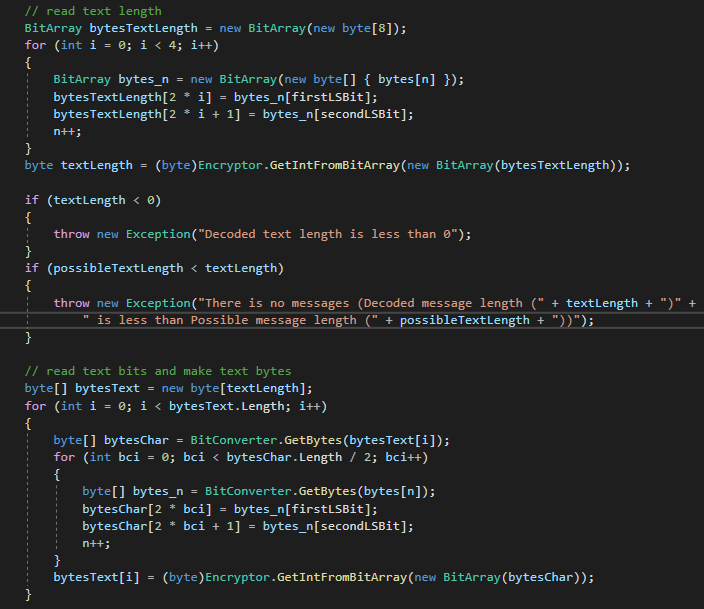


Рисунок 3.8 – Считывание сообщения из контейнера

Файл переводится в байты, затем рассчитывается размер палитры и длина сообщения. Что представлено ниже на рисунке 3.9.

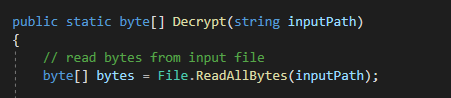


Рисунок 3.9 – Побайтовое чтение сообщения

Также необходимо рассчитать размер палитры и из него рассчитать размер сообщения, которое необходимо извлечь. Что представлено ниже на рисунке 3.10.

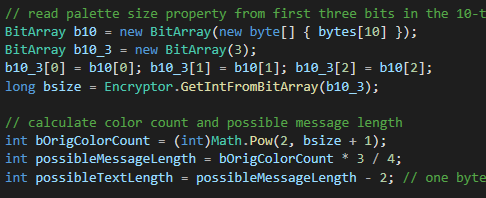


Рисунок 3.10 – Расчёт размера сообщения

После чего считывается проверочная последовательность, которая сравнивается с заданной проверочной последовательностью битов. В случае равенства проверочной последовательности в контейнере находится сообщение. Затем считывается размер передаваемого сообщения и его текст.

Листинги всего кода представлены в Приложении A.

### Метод расширения палитры

Алгоритмы внедрения и извлечения сообщения методом расширения палитры описаны в классе GIFEncryptorByPaletteExtensionMethod, структура которого представлена на рисунке 3.11. В поле checkSequence хранится проверочная последовательность, с помощью которой впоследствии обнаруживается сообщение в контейнере. Метод Encrypt отвечает за внедрение сообщения в GIF-контейнер, метод Decrypt – за извлечение сообщения из контейнера.

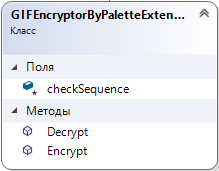


Рисунок 3.11 – Структура класса GIFEncryptorByLSBMethod

Метод Encrypt(string input, string text) принимает в качестве параметров путь к контейнеру и сообщение в виде строки. GIF-файл переводится в байты, рассчитывается размер палитры и количество цветов. Стоит отметить, что размер палитры должен быть меньше 7, иначе количество цветов будет уже максимальным и расширять будет некуда. Затем рассчитывается новый размер палитры так, чтобы можно было вместить все сообщение. Размер сообщения в битах делится на две части. В контейнер записывается проверочная последовательность, две части длины сообщения и само сообщение. Код записи сообщения представлен на рисунке 3.12.



Рисунок 3.12 – Запись сообщения в контейнер

Метод Decrypt(string input) принимает в качестве параметра путь к файлу, хранящему сообщение. Файл переводится в байты, рассчитывается размер палитры и длина сообщения. Затем считывается проверочная последовательность, которая сравнивается с заданной проверочной последовательностью. В случае равенства проверочной последовательности в контейнере находится сообщение. Две части длины сообщения соединяются в одно значение. После чего считывается текст. Код считывания сообщения представлен на рисунке 3.13.



Рисунок 3.13 – Считывание сообщения

Листинги всего кода представлены в Приложении B.

### Вспомогательные функции

Также в проекте имеется вспомогательный класс, в котором находятся методы, которые используются в двух методах стеганографии, поэтому они вынесены в отдельный класс. В классе Encryptor находятся некоторые вспомогательные функции. Структура класса представлена на рисунке 3.14.

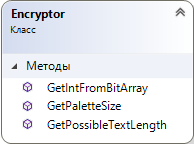


Рисунок 3.14 – Структура класса Encryptor

Метод GetIntFromBitArray используется для перевода бинарного массива значений в десятичное число. Что представлено на рисунке 3.15.

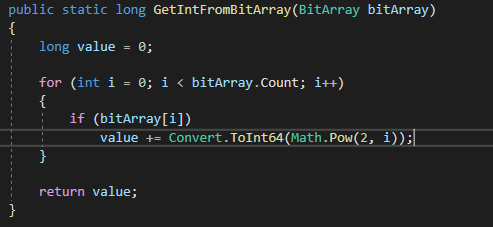


Рисунок 3.15 – Метод GetIntFromBitArray

Метод GetPaletteSize возвращает размер палитры. Данный метод читает необходимый файл и переводит его в байты. Далее производится проверка на соответствие заданного формата файла. Если формат файла не соответствует необходимому, то обрабатывается исключение. Далее в десятом байте байтового представления файла проверяются третий бит сообщения. Из него считывается размер палитры. Код данного метода можно увидеть на рисунке 3.16.

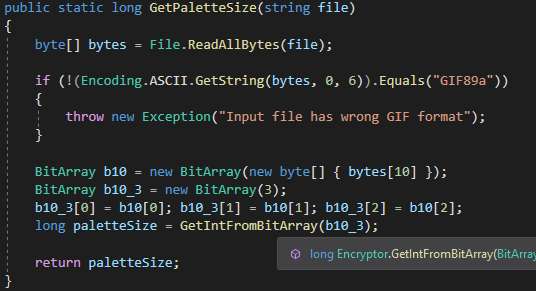


Рисунок 3.16 – Метод GetPaletteSize

Метод GetPossibleTextLength возвращает возможную длину сообщения.

В данном методе файл переводится в битовое представление. После чего проверяется формат файл на соответствие заданному. Далее необходимо вычислить размер палитры, для дальнейшего расчёта сообщение, которое будет определено по палитре цветов. Код данного метода можно увидеть ниже на рисунке 3.17.

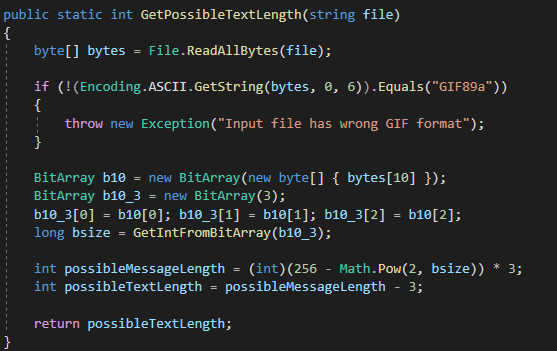


Рисунок 3.17 – Метод GetPossibleTextLength

В итоге данный класс обрабатывает методы, которые необходимы для работы двух методов стеганографии. Это можно заметить выше в коде данных методов.

Листинги кода представлены в Приложении C.

## Реализация графического интерфейса

Для создания графического интерфейса и возможности дополнительного шифрования данных, была выбрана технология C# WPF. В данном приложении будет использоваться 2 окна. Одно для сокрытия информации в контейнере, а другое будет выполнять роль клиента и извлекать сообщение из контейнера. На главном окне приложения можно будет увидеть контейнер для сокрытия сообщения. А также будет возможность выбрать пути к контейнеру и к текстовому файлу. Если же сообщение окажется не большим, то его можно будет ввести в поле приложения.

Создадим основное окно программы. Рисунок 3.18.

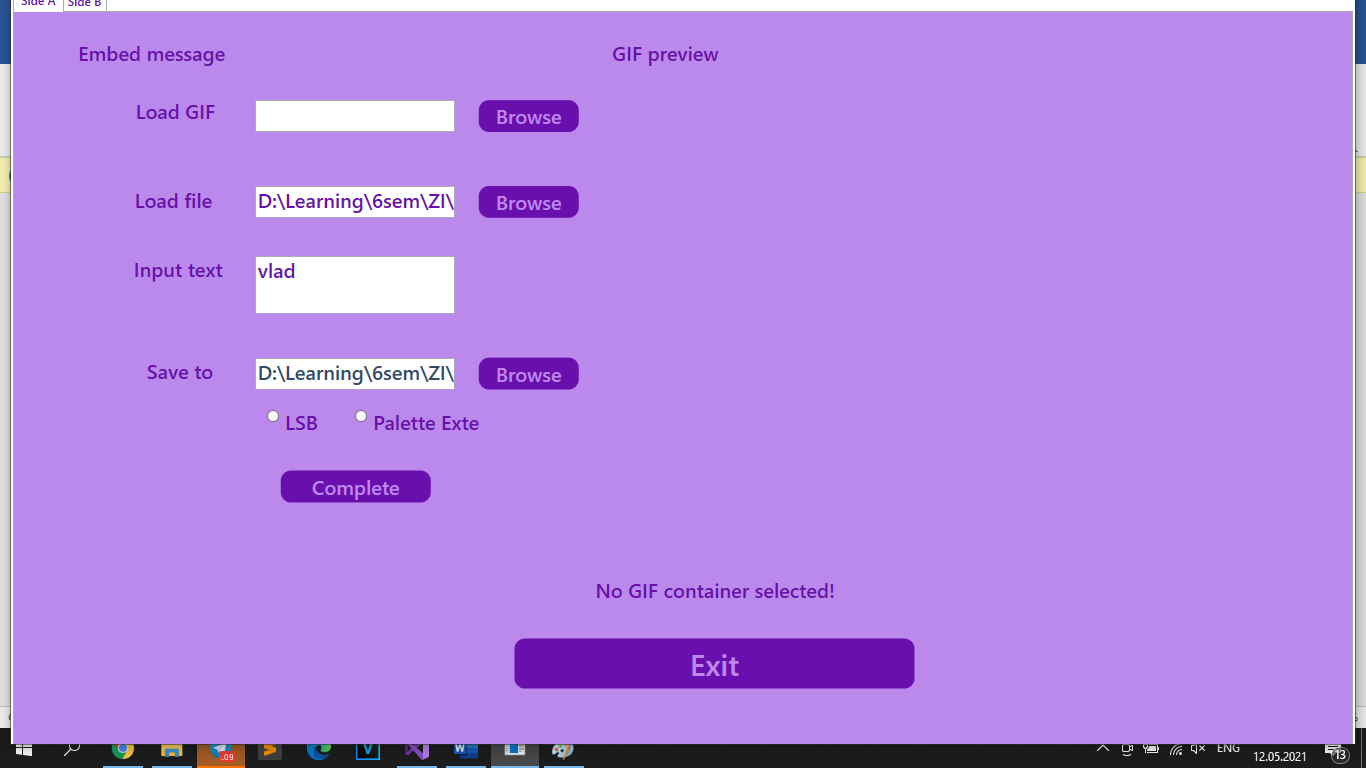


Рисунок 3.18 – Главное окно приложения

В главном окне есть три кнопки для открытия файлов. Первая кнопка необходима для указания пути к файлу контейнеру. Вторая кнопка для указания пути к текстовому файлу. И третья кнопка для сохранения файла с сокрытой информацией. Также можно увидеть две радиокнопки, которые позволяют выбрать метод сокрытия информации. В поле для ввода текста можно ввести сообщение для сокрытия. Внизу данного окна находится кнопка для закрытия приложения и информационное поле, в котором выводятся ошибки или оповещение о выполнении.

Окно получателя сообщения представлено на рисунке 3.19.

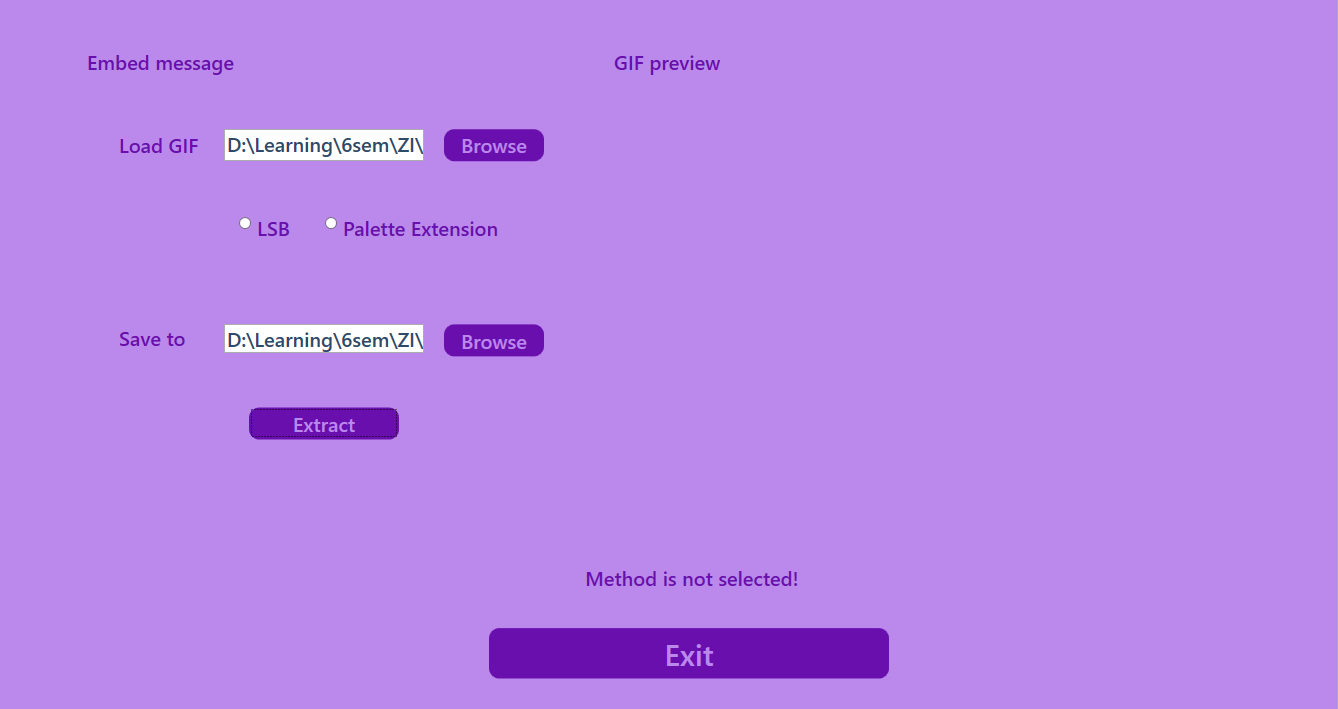


Рисунок 3.19 – Окно получателя сообщения

Теперь реализуем обработчики события для кнопок.

Функция BtnLoadGifCommandBinding\_B\_Executed вызывает диалоговое окно, которое позволяет выбрать файл формата .gif, что представлено на рисунке 3.20.

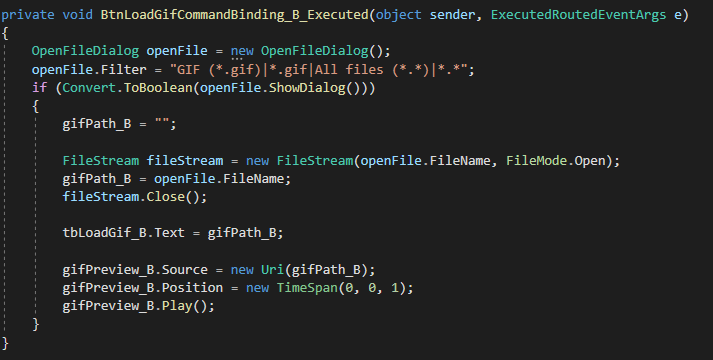


Рисунок 3.20 – Функция BtnLoadGifCommandBinding\_B\_Executed

Функция BtnLoadFileCommandBinding\_Executed вызывает диалоговое окно, которое позволяет выбрать файл формата .txt, что представлено на рисунке 3.21.

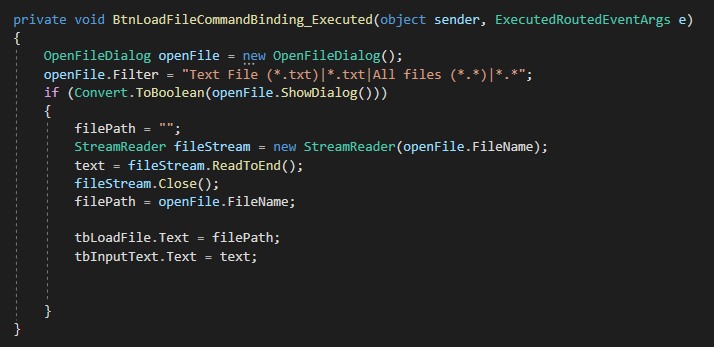


Рисунок 3.21 – Функция BtnLoadFileCommandBinding\_Executed

Функция btnEmbed\_Click является обработчиком клика для кнопки «Complete». При нажатии на данную кнопку обрабатывается ряд исключения, что можно увидеть на рисунке 3.22.

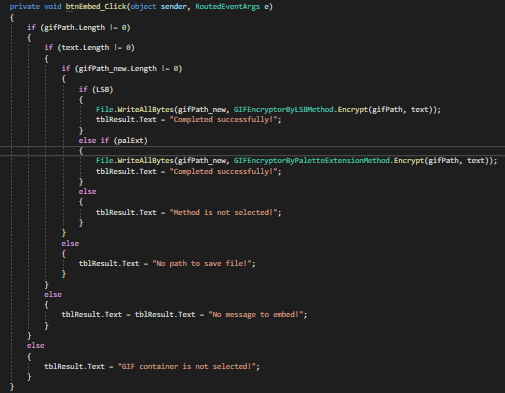


Рисунок 3.22 – Функция btnEmbed\_Click

Сначала происходит инициализация диалогового окна для сохранения нового файла, затем проверка вводимых полей на наличие значения. При отсутствии корректного значения программы выводит соответствующее сообщение и завершает выполнении метода.

Далее функция получает информацию из поля введённого сообщения или из файла путь на который был введёт в окне выше и записывает его.

После программа вызывает модуль и ожидает завершение его работы.

Если при осаждении сообщение произошла ошибка, то программа выведет соответствующее сообщение. В случае успеха, программа предложит выбрать место для сохранения нового файла. Осаждение файла в контейнер можно увидеть в представлении графических изображений в правой части приложения.

# Тестирование

Разработанное программное средстве предусматривает совершение пользователем некоторых ошибок. Сообщения об ошибках выводятся в нижней части окна приложения (рисунок 4.1).

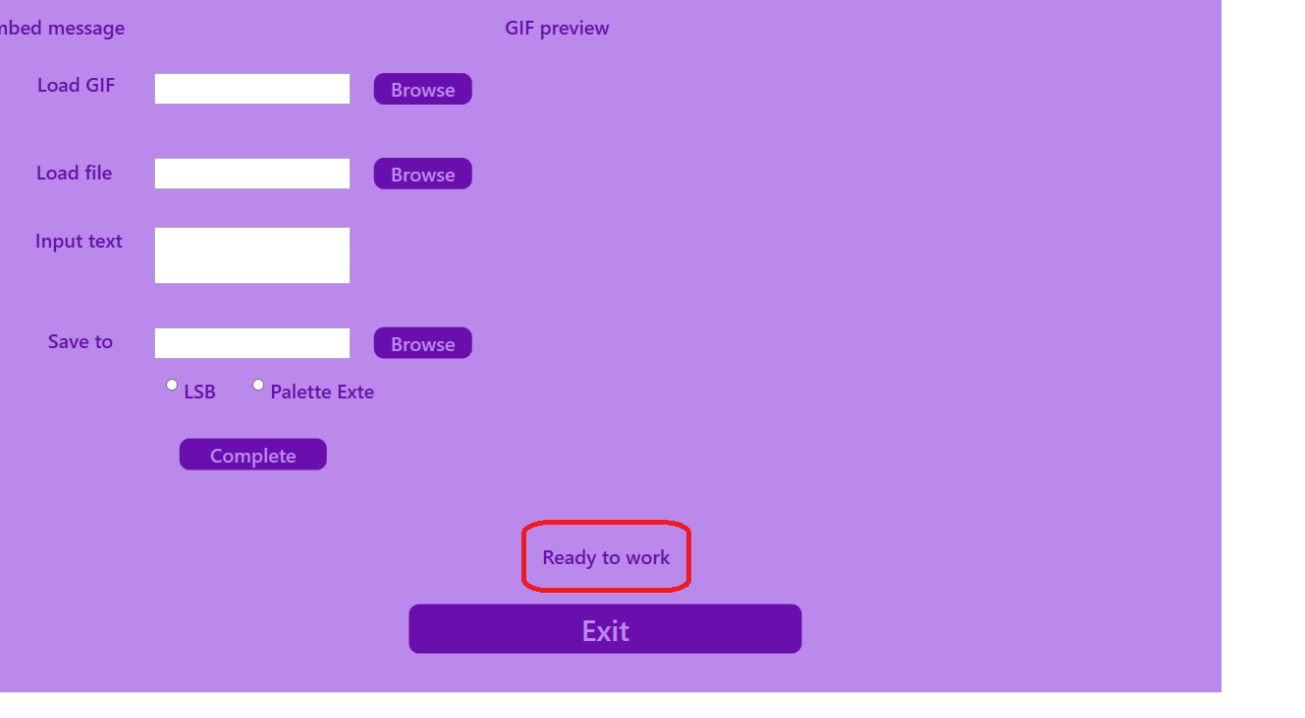


Рисунок 4.1 – Место для сведений об ошибках

На стороне отправителя, если пользователь не укажет GIF-контейнер, сообщение или путь для сохранения файла, выведется сообщение «GIF container is not selected!», «No path to save file!» соответственно (рисунок 4.2).

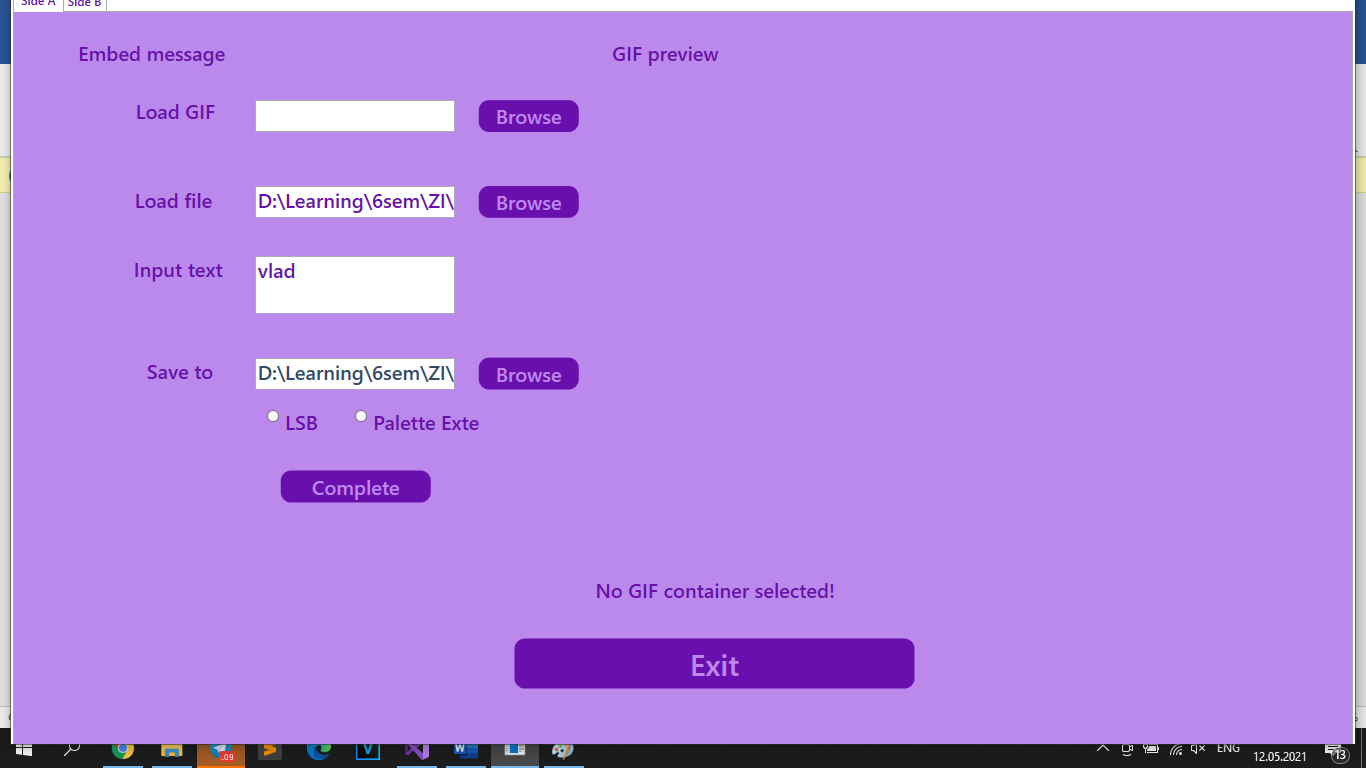


Рисунок 4.2 – Не выбран GIF-контейнер

Также, если не будет выбран ни один из методов для записи сообщения в контейнер, будет выведено сообщение «No method selected!» (рисунок 4.3).

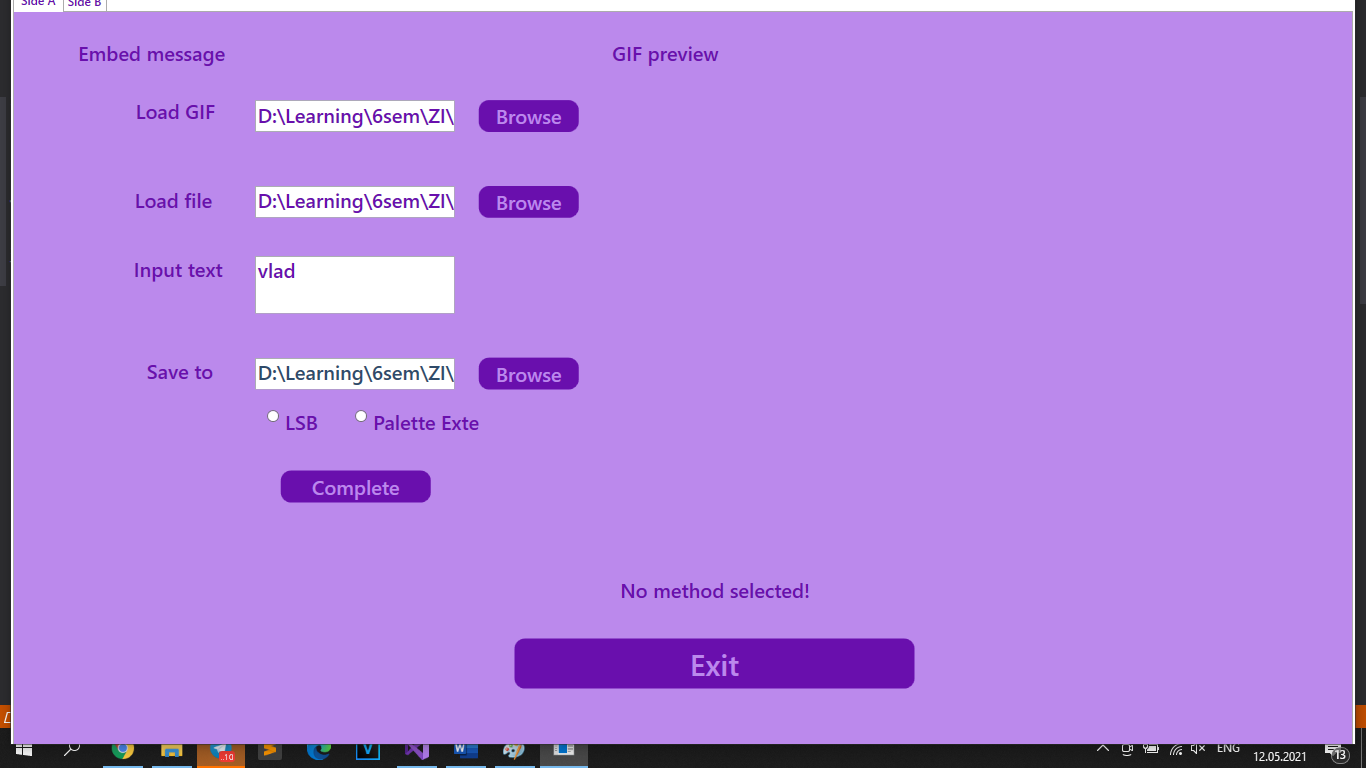


Рисунок 4.3 – Не выбран ни один метод

На стороне получателя также предусмотрены ошибки, связанные с не указанием контейнера с сообщением или файла для считывания сообщения. При совершении ошибок выводятся сообщения «No path to save message!» (рисунок 4.4).

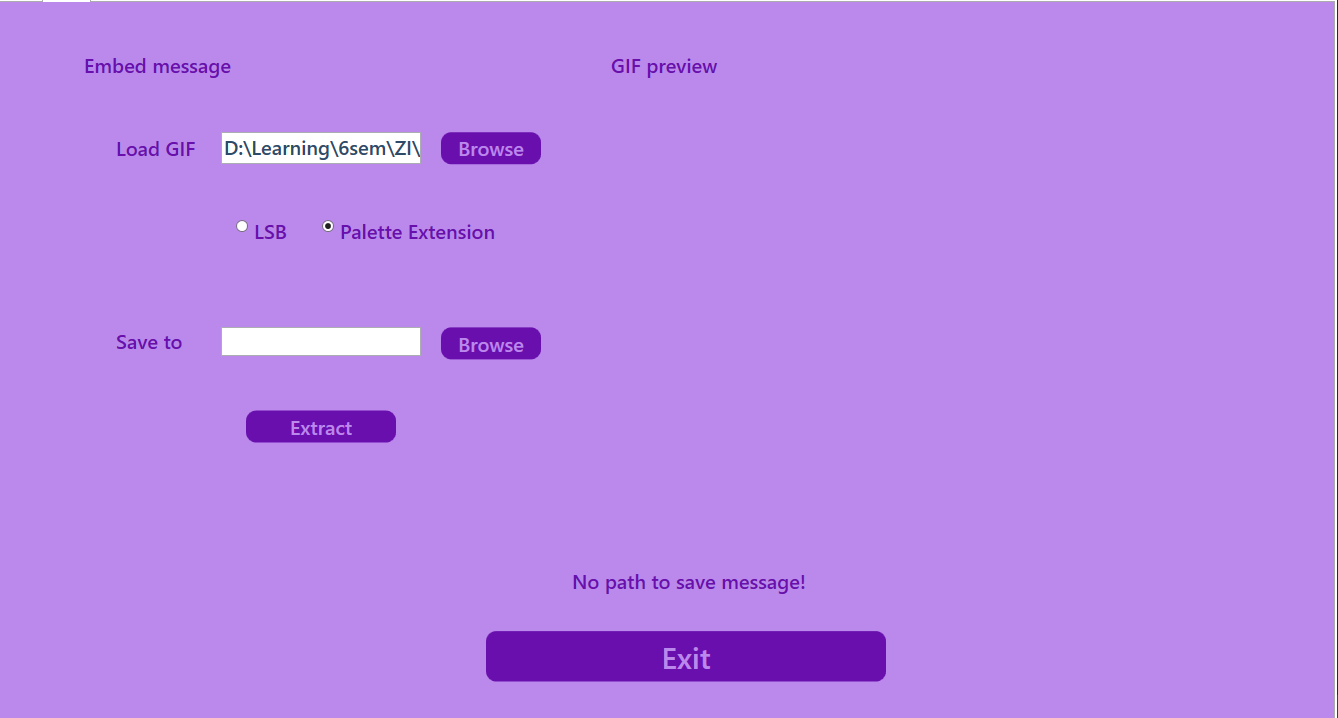


Рисунок 4.4 – Не выбран файл для сохранения сообщения

Также, если не будет выбран ни один из методов для считывания сообщения из контейнера, будет выведено сообщение «No method selected!» (рисунок 4.5).

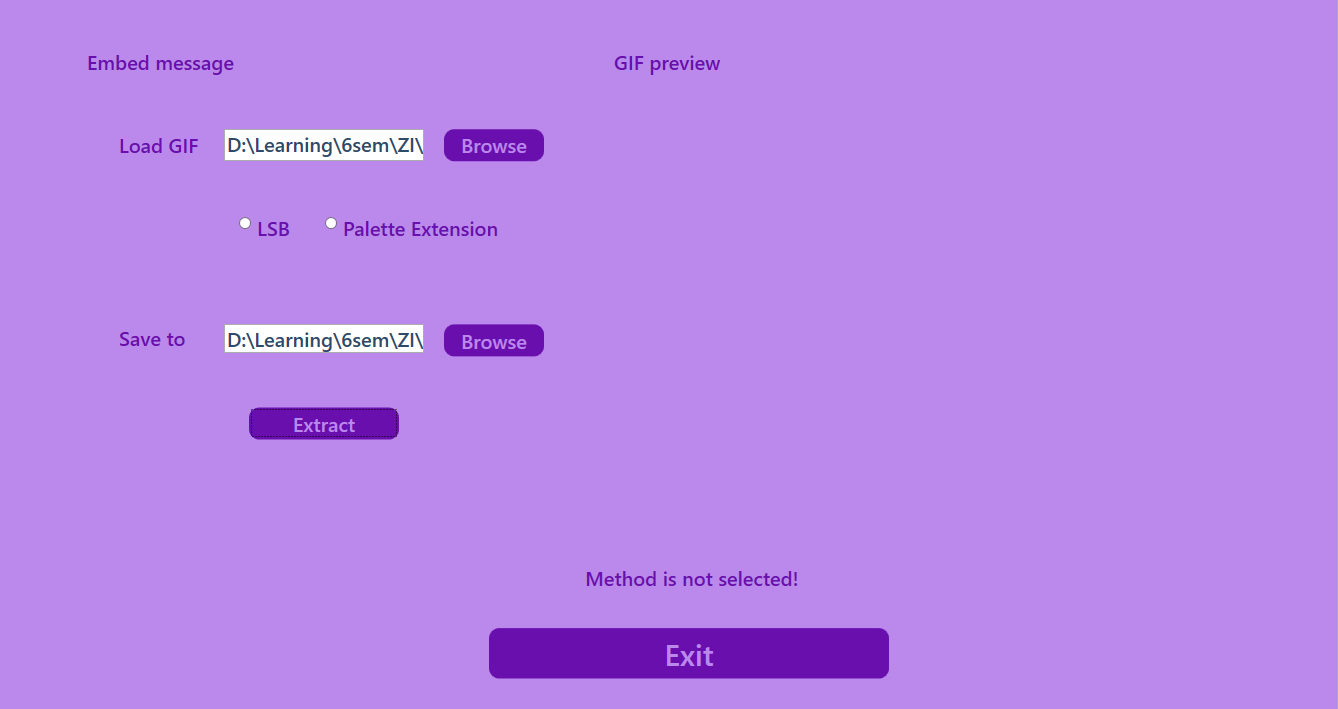


Рисунок 4.5 – Не выбран ни один метод

Все описанные выше возможные исключения были обработаны для правильного функционирования приложения, а также информирования пользователя в случае допущения ошибки. Это необходимо для того, чтобы не возникало исключительных ситуация в приложении во время выполнения.

# Руководство по использованию

Внешний вид приложения при запуске изображен на рисунке 5.1.

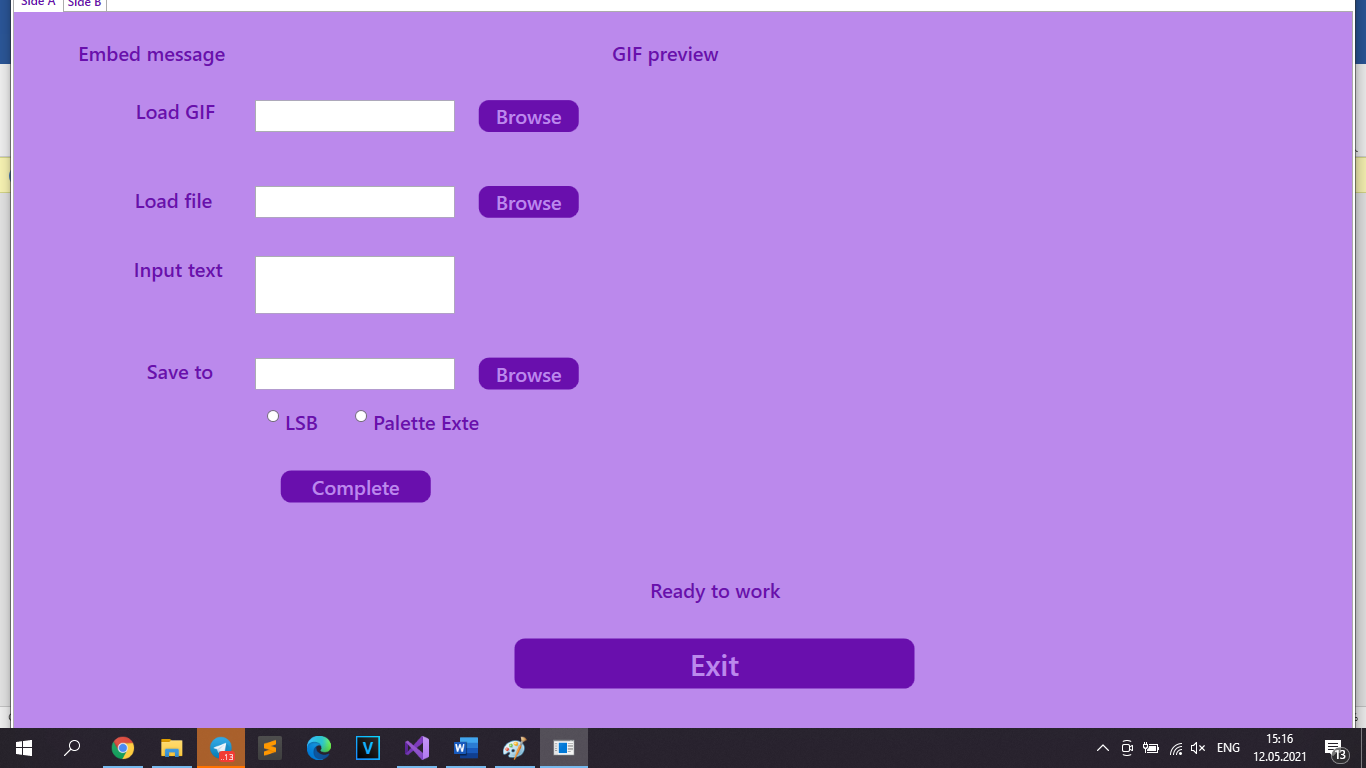


Рисунок 5.1 – Начальный вид приложения

Для приближения приложения к реальности в верхней части содержатся две вкладки Side A и Side B, соответствующие стороне отправителя и стороне получателя соответственно.

Пользователь работает с частью Embed message. Для начала необходимо загрузить GIF-контейнер, нажав на Browse около поля Load GIF (рисунок 5.2).

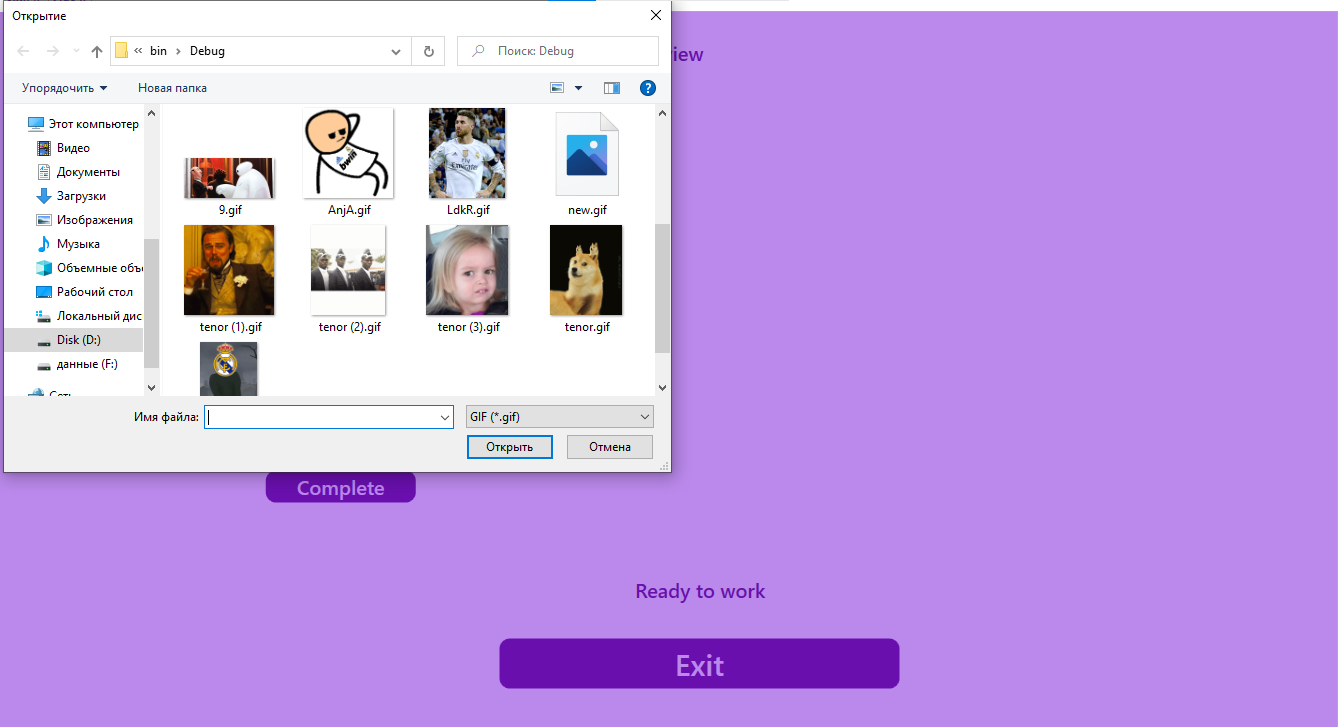


Рисунок 5.2 – Загрузка контейнера

После загрузки контейнера выбранный GIF-файл отобразится в части GIF preview, что изображено на рисунке ниже (рисунок 5.3).

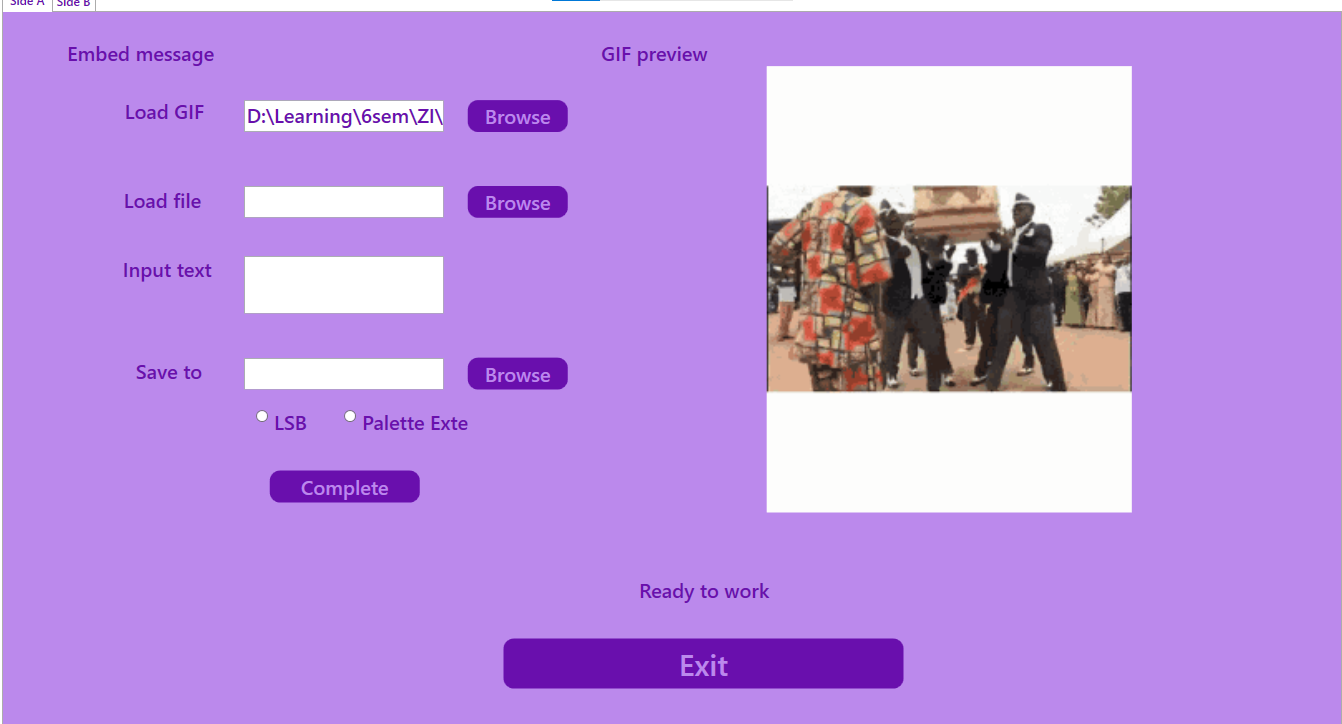


Рисунок 5.3 – Загруженный GIF-файл

Затем необходимо загрузить сообщение из файла, нажав на кнопку Browse около поля Load file, или ввести сообщение вручную в поле Input text (рисунок 5.4).

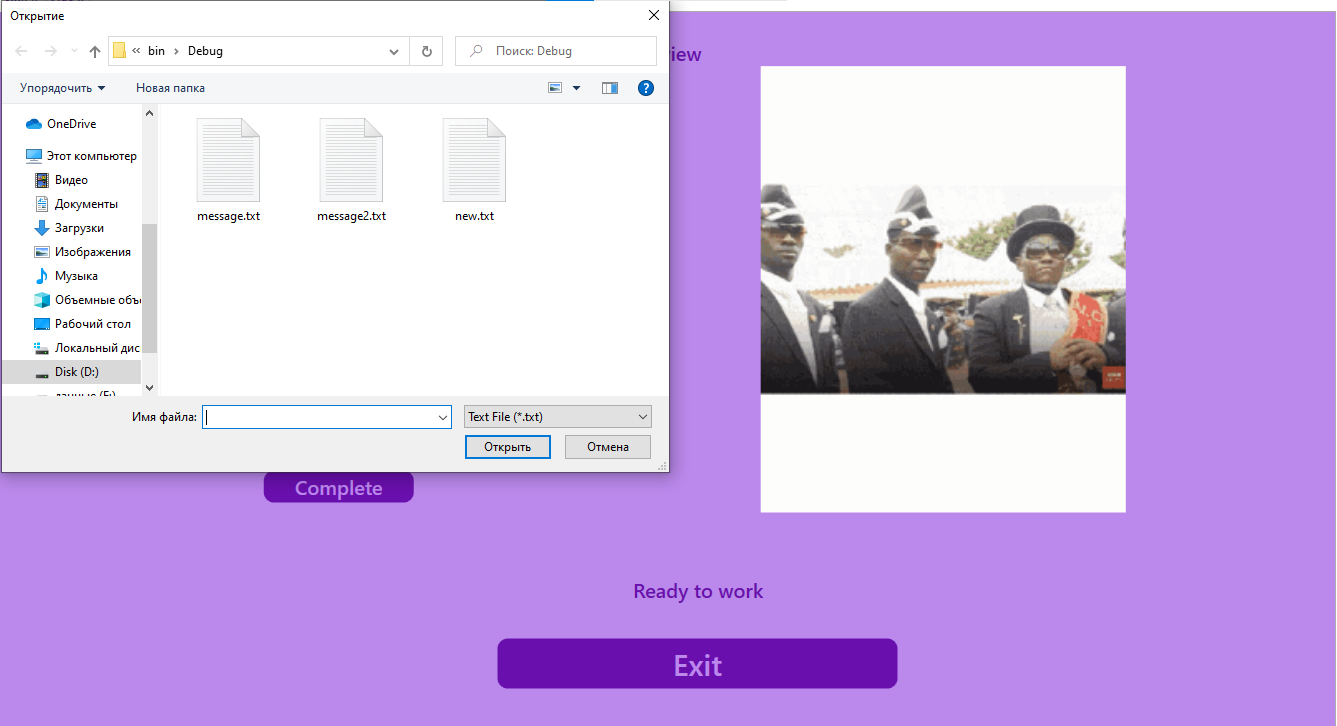


Рисунок 5.4 – Загрузка файла с сообщением

Затем необходимо выбрать путь, по которому будет сохранен GIF-файл с сообщением. Для этого нажимаем на Browse около поля Save file (рисунок 5.5).

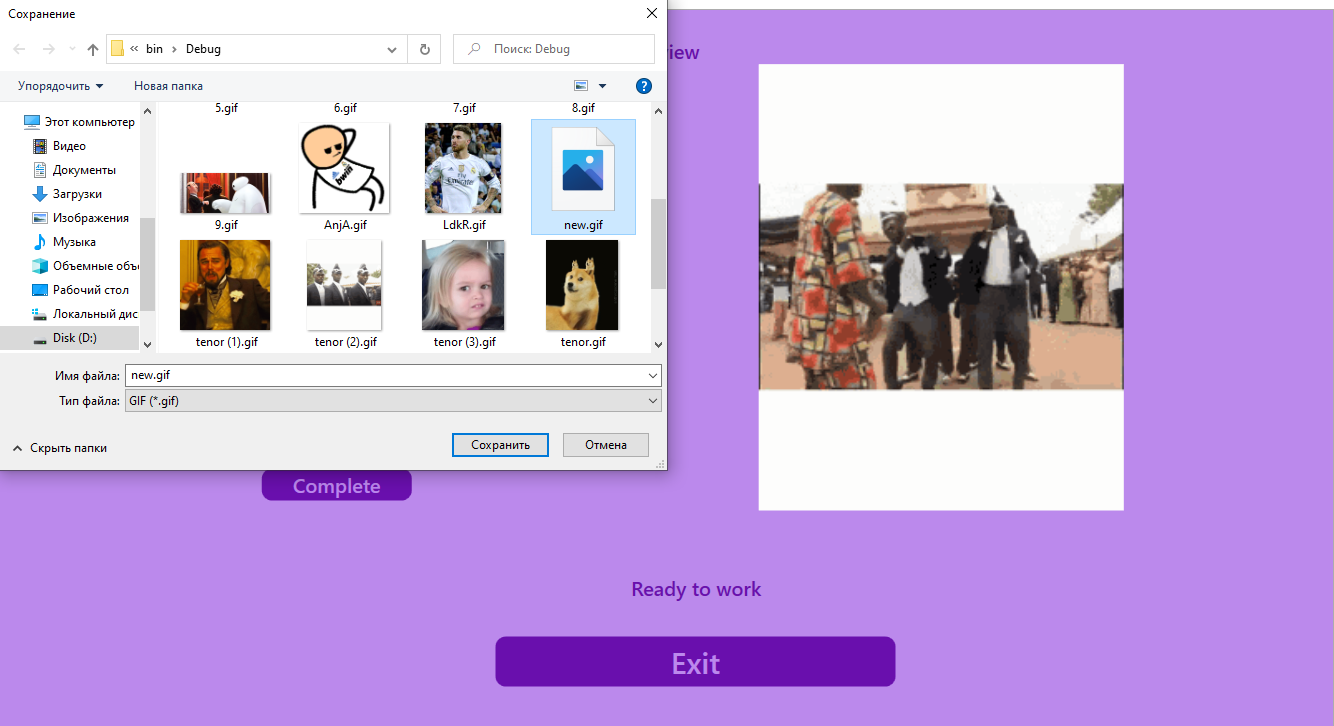


Рисунок 5.5 – Сохранение файла

Наконец, выбираем метод, которым будет записано сообщение, и нажимаем на кнопку Complete. В случае успешного выполнения внизу появится строка «Completed successfully» (рисунок 5.6).

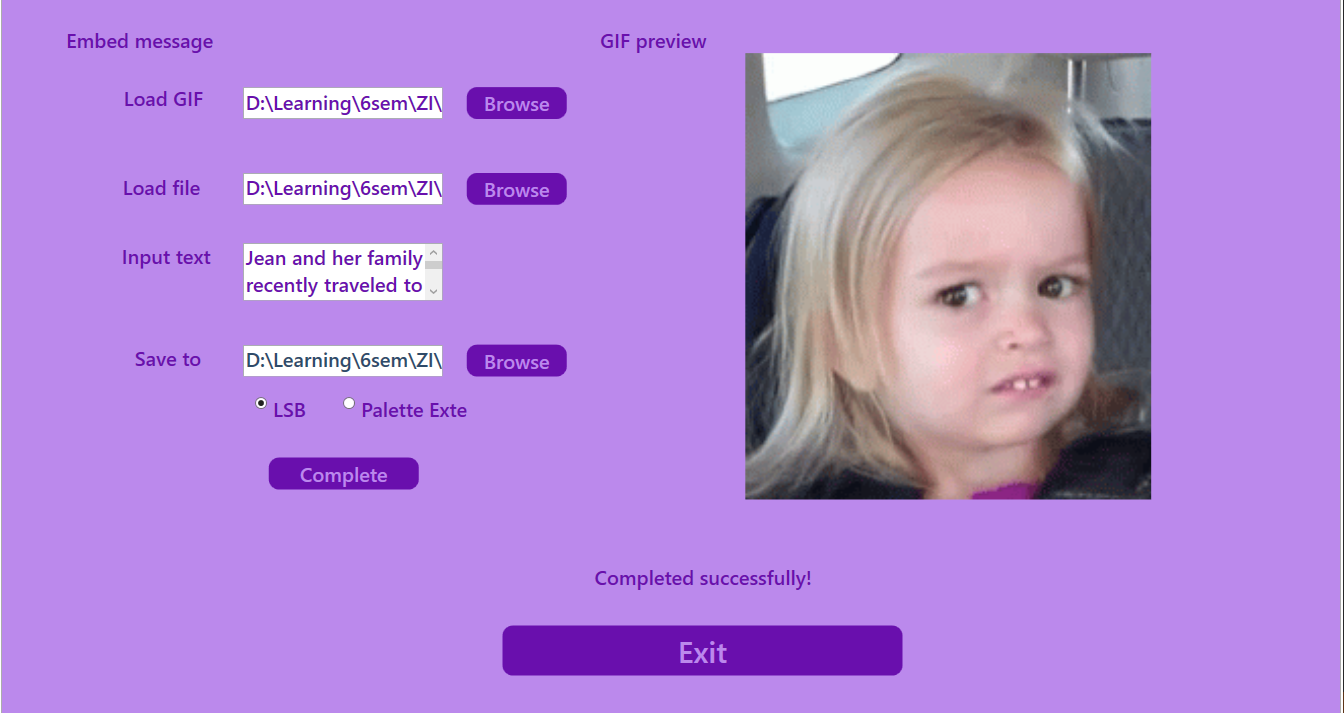


Рисунок 5.6 – Выполнение записи сообщения

Здесь работа на стороне отправителя заканчивается. Выйти из приложения можно, нажав на кнопку Exit внизу данного экрана.

Переходим на вторую вкладку Side B (рисунок 5.7).

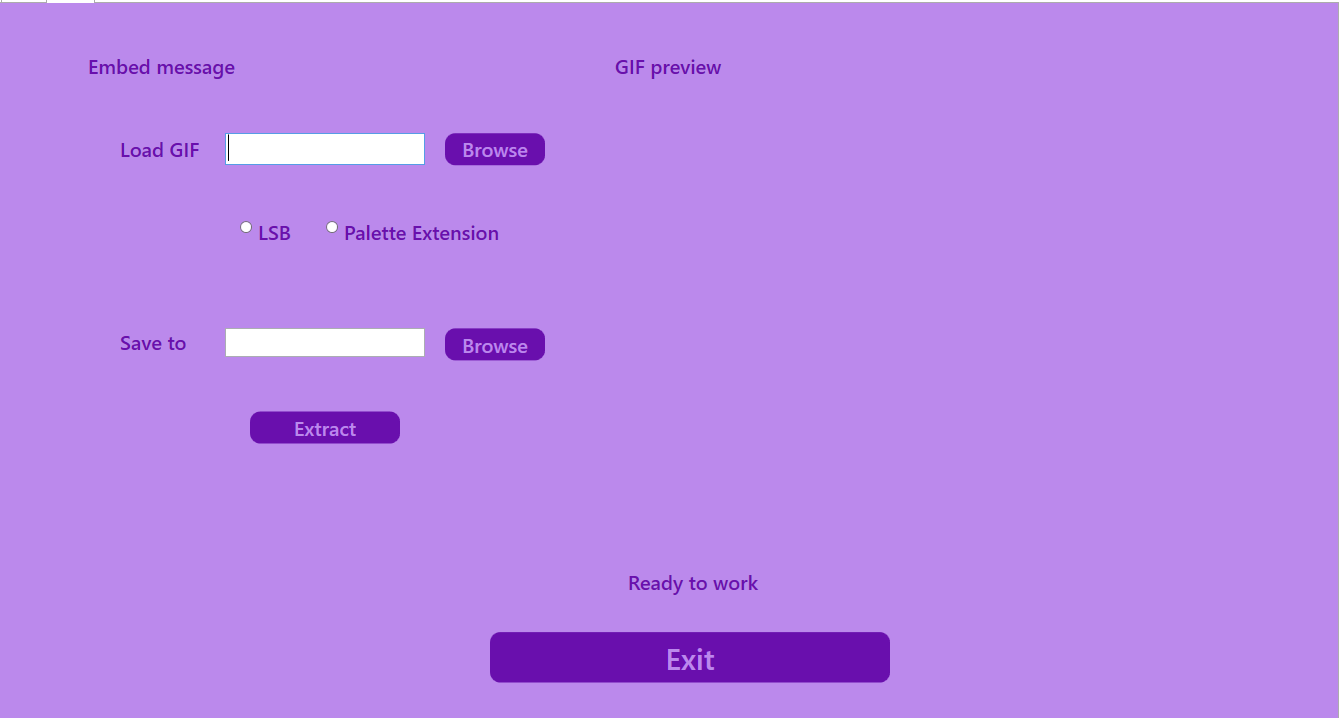


Рисунок 5.7 – Внешний вид вкладки Side B

Для чтения сообщения из контейнера сначала необходимо загрузить GIF-файл, нажав на кнопку Browse около поля Load GIF, и выбрать метод, которым сообщение было записано. После загрузки файл отобразится в части GIF preview (рисунок 5.8).

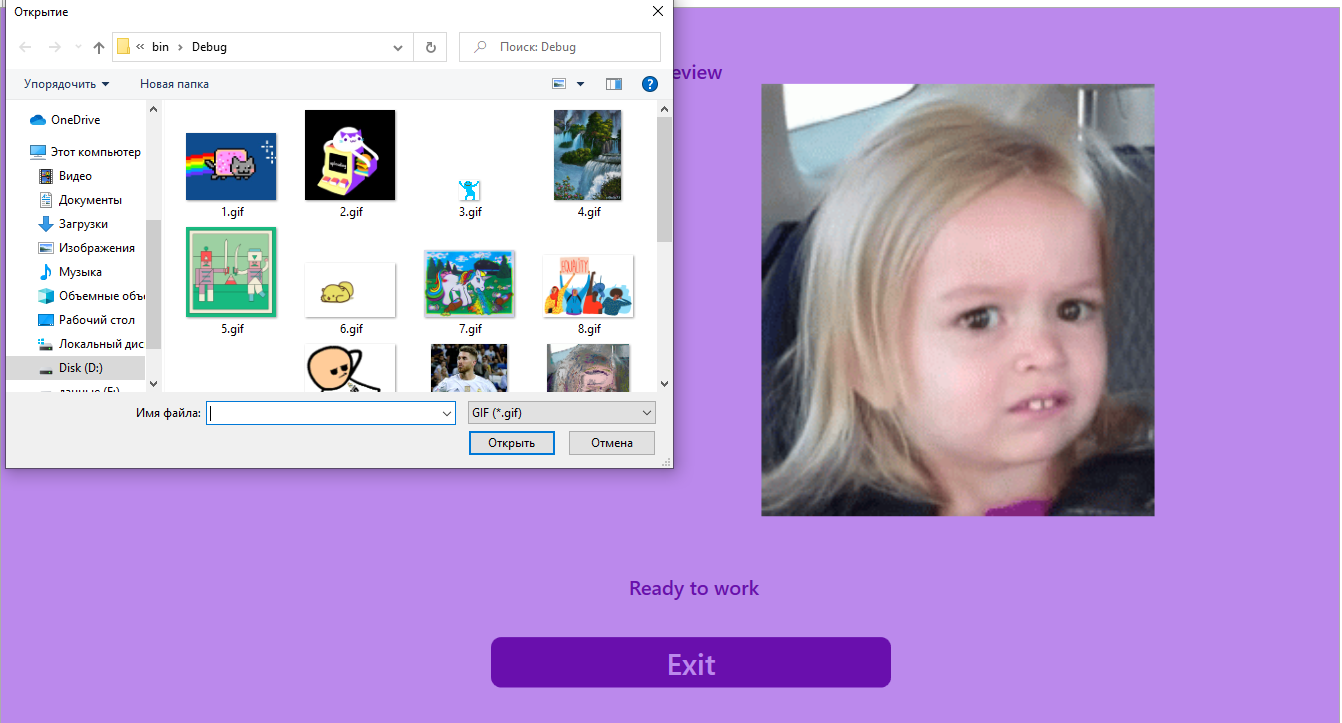


Рисунок 5.8 – Загрузка GIF-файла

Затем необходимо указать путь, по которому будет сохранено считанное сообщение. Для этого нажимаем на Browse около поля Extract to (рисунок 5.9).

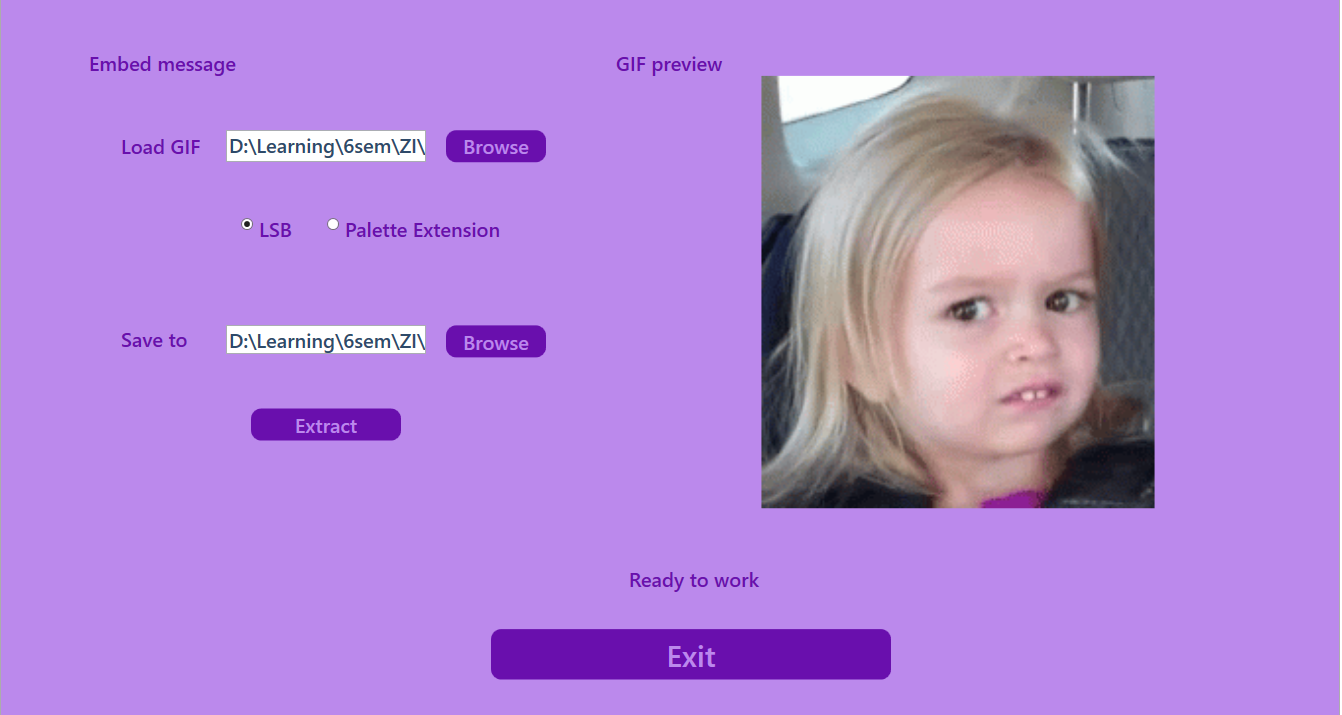


Рисунок 5.9 – Сохранение извлеченного сообщение

Наконец, нажимаем на кнопку Extract, и сообщение появится в указанному выше файле (рисунок 5.10).

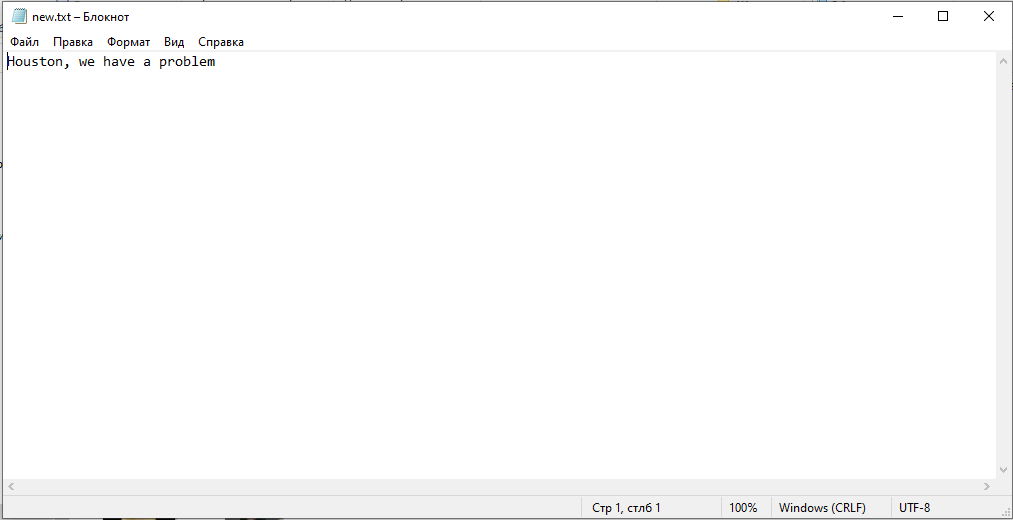


Рисунок 5.10 – Считывание сообщения

По окончании работы с программным средством для выхода из него необходимо нажать на кнопку Exit внизу экрана.

# Заключение

В ходе выполнения данного курсового проекта были выполнены все поставленные задачи:

* Разработан алгоритм для сокрытия и извлечения сообщения из GIF-контейнера методом LSB;
* Разработан алгоритм для сокрытия и извлечения сообщения из GIF-контейнера методом расширения палитры;
* Создано программное средство, реализующее разработанные алгоритмы;
* Составлено руководство по использованию программное средства.

Результатом работы является программное средство «StegoGIF». Разработанное программное средство реализует два стеганографический метода: LSB и метод расширения палитры – на стороне отправителя и получателя. Пользователю предоставляется возможность вписать сообщение в GIF-контейнер или извлечь сообщение одним из методов. Также при разработке были учтены и обработаны возможные ошибки при использовании программного средства.

При создании программного средства были использованы все указанные технологии, выполнены все функциональные требования, обработаны возможные ошибки при использовании.

Приложение является десктопным и разработано на языке C# с использованием технологии WPF. Данная технология была выбрана неслучайно, так как пользователь в первую очередь оценивает внешний вид приложения, поэтому необходимо было разработать привлекательный графический интерфейс, для создания которого на программной платформе .NET и подходит WPF [5].

# Список используемых источников

1. Википедия [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/
2. Статья «Краткое описание формата GIF» [Электронный ресурс] – Режим доступа: http://home.onego.ru/~chiezo/gif.htm
3. Статья «Стеганография. Метод LSB» [Электронный ресурс] – Режим доступа: https://ghostbasenji.blogspot.com/2018/08/steganography-method-LSB.html
4. Статья «Метод расширения палитры» [Электронный ресурс] – Режим доступа: https://studbooks.net/2063267/informatika/metod\_rasshireniya\_palitry
5. Руководство по WPF [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/wpf/

# Приложение А

public class GIFEncryptorByLSBMethod

{

protected static byte[] checkSequence = new byte[] { 0, 1, 0, 1, 0, 1, 0, 1 };

protected static int firstLSBit = 0;

protected static int secondLSBit = 1;

public static byte[] Encrypt(string inputPath, string text)

{

// read bytes from input file

byte[] bytes = File.ReadAllBytes(inputPath);

// read palette size property from first three bits in the 10-th byte from the file

BitArray bytes10 = new BitArray(new byte[] { bytes[10] });

BitArray bytes10\_3 = new BitArray(3);

bytes10\_3[0] = bytes10[0]; bytes10\_3[1] = bytes10[1]; bytes10\_3[2] = bytes10[2];

long paletteSize = Encryptor.GetIntFromBitArray(bytes10\_3);

// calculate color count and possible message length

int bOrigColorCount = (int)Math.Pow(2, paletteSize + 1);

int possibleMessageLength = bOrigColorCount \* 3 / 4;

int possibleTextLength = possibleMessageLength - 2;

if (possibleTextLength < text.Length)

{

throw new Exception("Text is too big");

}

int n = 13;

// write check sequence

for (int i = 0; i < checkSequence.Length / 2; i++)

{

byte[] bytes\_n = BitConverter.GetBytes(bytes[n]);

bytes\_n[firstLSBit] = checkSequence[2 \* i];

bytes\_n[secondLSBit] = checkSequence[2 \* i + 1];

bytes[n] = (byte)Encryptor.GetIntFromBitArray(new BitArray(bytes\_n));

n++;

}

// write text length

BitArray bitTextLength = new BitArray(new byte[] { (byte)text.Length });

for (int i = 0; i < bitTextLength.Length / 2; i++)

{

BitArray bytes\_n = new BitArray(new byte[] { bytes[n] });

bytes\_n[firstLSBit] = bitTextLength[2 \* i];

bytes\_n[secondLSBit] = bitTextLength[2 \* i + 1];

bytes[n] = (byte)Encryptor.GetIntFromBitArray(bytes\_n);

n++;

}

// write message

byte[] bytesText = Encoding.Default.GetBytes(text);

for (int i = 0; i < bytesText.Length; i++)

{

byte[] bytesChar = BitConverter.GetBytes(bytesText[i]);

for (int ci = 0; ci < bytesChar.Length / 2; ci++)

{

byte[] bytes\_n = BitConverter.GetBytes(bytes[n]);

bytes\_n[firstLSBit] = bytesChar[2 \* ci];

bytes\_n[secondLSBit] = bytesChar[2 \* ci + 1];

bytes[n] = (byte)Encryptor.GetIntFromBitArray(new BitArray(bytes\_n));

n++;

}

}

return bytes;

}

public static byte[] Decrypt(string inputPath)

{

// read bytes from input file

byte[] bytes = File.ReadAllBytes(inputPath);

// read palette size property from first three bits in the 10-th byte from the file

BitArray b10 = new BitArray(new byte[] { bytes[10] });

BitArray b10\_3 = new BitArray(3);

b10\_3[0] = b10[0]; b10\_3[1] = b10[1]; b10\_3[2] = b10[2];

long bsize = Encryptor.GetIntFromBitArray(b10\_3);

// calculate color count and possible message length

int bOrigColorCount = (int)Math.Pow(2, bsize + 1);

int possibleMessageLength = bOrigColorCount \* 3 / 4;

int possibleTextLength = possibleMessageLength - 2; // one byte for check and one byte for message length

int n = 13;

// read check sequence

BitArray csBytes = new BitArray(checkSequence);

for (int i = 0; i < 4; i++)

{

BitArray bit = new BitArray(new byte[] { bytes[n] });

csBytes[2 \* i] = bit[firstLSBit];

csBytes[2 \* i + 1] = bit[secondLSBit];

n++;

}

byte cs = (byte)Encryptor.GetIntFromBitArray(csBytes);

if (cs != (byte)Encryptor.GetIntFromBitArray(new BitArray(checkSequence)))

{

return new byte[] { 0 };

}

// read text length

BitArray bytesTextLength = new BitArray(new byte[8]);

for (int i = 0; i < 4; i++)

{

BitArray bytes\_n = new BitArray(new byte[] { bytes[n] });

bytesTextLength[2 \* i] = bytes\_n[firstLSBit];

bytesTextLength[2 \* i + 1] = bytes\_n[secondLSBit];

n++;

}

byte textLength = (byte)Encryptor.GetIntFromBitArray(new BitArray(bytesTextLength));

if (textLength < 0)

{

throw new Exception("Decoded text length is less than 0");

}

if (possibleTextLength < textLength)

{

throw new Exception("There is no messages (Decoded message length (" + textLength + ")" +

" is less than Possible message length (" + possibleTextLength + "))");

}

// read text bits and make text bytes

byte[] bytesText = new byte[textLength];

for (int i = 0; i < bytesText.Length; i++)

{

byte[] bytesChar = BitConverter.GetBytes(bytesText[i]);

for (int bci = 0; bci < bytesChar.Length / 2; bci++)

{

byte[] bytes\_n = BitConverter.GetBytes(bytes[n]);

bytesChar[2 \* bci] = bytes\_n[firstLSBit];

bytesChar[2 \* bci + 1] = bytes\_n[secondLSBit];

n++;

}

bytesText[i] = (byte)Encryptor.GetIntFromBitArray(new BitArray(bytesChar));

}

return bytesText;

}

}

# Приложение Б

public class GIFEncryptorByPaletteExtensionMethod

{

static protected byte[] checkSequence = new byte[] { 0, 1, 0, 1, 0, 1, 0, 1 };

public static byte[] Encrypt(string input, string text)

{

// read bytes from file

byte[] bytes = File.ReadAllBytes(input);

// check format

if (!(Encoding.ASCII.GetString(bytes, 0, 6)).Equals("GIF89a"))

{

throw new Exception("Input file has wrong GIF format");

}

// read palette size property from first three bits in the 10-th byte from the file

BitArray bytes10 = new BitArray(new byte[] { bytes[10] });

BitArray bytes10\_3 = new BitArray(3);

bytes10\_3[0] = bytes10[0]; bytes10\_3[1] = bytes10[1]; bytes10\_3[2] = bytes10[2];

long paletteSize = Encryptor.GetIntFromBitArray(bytes10\_3);

// calculate original color count

int bOrigColorCount = (int)Math.Pow(2, paletteSize + 1);

// calculate new palette size to contain all message

long newPaletteSize = paletteSize;

int possibleMessageLength = (int)(Math.Pow(2, newPaletteSize + 1) - bOrigColorCount) \* 3;

int possibleTextLength = possibleMessageLength - 3;

while ((newPaletteSize < 7) && (possibleTextLength < text.Length))

{

newPaletteSize++;

possibleMessageLength = (int)(Math.Pow(2, newPaletteSize + 1) - bOrigColorCount) \* 3;

possibleTextLength = possibleMessageLength - 3;

}

if (possibleTextLength < text.Length)

{

throw new Exception("Text is too big. Max text lenght for this image is " + possibleTextLength);

}

// set new palette property to the 10-th byte in the file

BitArray newBsizeBits = new BitArray(new byte[] { (byte)newPaletteSize });

bytes10[0] = newBsizeBits[0];

bytes10[1] = newBsizeBits[1];

bytes10[2] = newBsizeBits[2];

bytes[10] = (byte)Encryptor.GetIntFromBitArray(bytes10);

// create message array

byte[] messageArray = new byte[possibleMessageLength];

// create bit array from text length value and divide it into two arrays

BitArray lengthBytes = new BitArray(new byte[] { (byte)text.Length });

BitArray lowTextLengthByte = new BitArray(4);

for (int i = 0; i < 4; i++)

{

lowTextLengthByte[i] = lengthBytes[i];

}

BitArray highTextLengthByte = new BitArray(4);

for (int i = 0; i < 4; i++)

{

highTextLengthByte[i] = lengthBytes[i + 4];

}

// write bytes of check sequence and of two parts of message length

messageArray[possibleMessageLength - 1] = (byte)Encryptor.GetIntFromBitArray(new BitArray(checkSequence));

messageArray[possibleMessageLength - 2] = (byte)Encryptor.GetIntFromBitArray(new BitArray(highTextLengthByte));

messageArray[possibleMessageLength - 3] = (byte)Encryptor.GetIntFromBitArray(new BitArray(lowTextLengthByte));

// write text bytes

byte[] textBytes = Encoding.Default.GetBytes(text);

for (int i = 0; i < text.Length; i++)

{

messageArray[messageArray.Length - i - 4] = textBytes[i];

}

// write output file

byte[] resultBytes = new byte[bytes.Length + messageArray.Length];

int j = 0;

for (int i = 0; i < 13 + 3 \* bOrigColorCount; i++)

{

resultBytes[j] = bytes[i];

j++;

}

for (int i=0; i<messageArray.Length; i++)

{

resultBytes[j] = messageArray[i];

j++;

}

for (int i = 13 + 3 \* bOrigColorCount; i < bytes.Length; i++)

{

resultBytes[j] = bytes[i];

j++;

}

return resultBytes;

}

public static byte[] Decrypt(string input)

{

// read bytes from input file

byte[] bytes = File.ReadAllBytes(input);

// read palette size property from first three bits in the 10-th byte from the file

BitArray b10 = new BitArray(new byte[] { bytes[10] });

BitArray b10\_3 = new BitArray(3);

b10\_3[0] = b10[0]; b10\_3[1] = b10[1]; b10\_3[2] = b10[2];

long bsize = Encryptor.GetIntFromBitArray(b10\_3);

// calculate color count and possible message length

int bOrigColorCount = (int)Math.Pow(2, bsize + 1);

int possibleMessageLength = (int)Math.Pow(2, bsize + 1) \* 3;

int possibleTextLength = possibleMessageLength - 3;

int n = 13;

// read check sequence

byte cs = bytes[n + bOrigColorCount \* 3 - 1];

if (cs != (byte)Encryptor.GetIntFromBitArray(new BitArray(checkSequence)))

{

return new byte[] { 0 };

}

// read two text length bytes and split them to one

BitArray highTextLengthByte = new BitArray(new byte[] { bytes[n + bOrigColorCount \* 3 - 2] });

BitArray lowTextLengthByte = new BitArray(new byte[] { bytes[n + bOrigColorCount \* 3 - 3] });

BitArray textLengthByte = new BitArray(8);

for (int i = 0; i < 4; i++)

{

textLengthByte[i] = lowTextLengthByte[i];

textLengthByte[i + 4] = highTextLengthByte[i];

}

int textLength = (int)Encryptor.GetIntFromBitArray(new BitArray(textLengthByte));

if (textLength < 0)

{

throw new Exception("Decoded text length is less than 0");

}

if (possibleTextLength < textLength)

{

throw new Exception("There is no encrypted message (Decoded message length (" + textLength + ") is less than Possible message length (" + possibleTextLength + "))");

}

// read text bytes

byte[] bt = new byte[textLength];

for (int i = 0; i < bt.Length; i++)

{

bt[i] = bytes[n + bOrigColorCount \* 3 - 1 - i - 3];

}

return bt;

}

}

**Приложение C**

public class Encryptor

{

public static int GetPossibleTextLength(string file)

{

byte[] bytes = File.ReadAllBytes(file);

if (!(Encoding.ASCII.GetString(bytes, 0, 6)).Equals("GIF89a"))

{

throw new Exception("Input file has wrong GIF format");

}

BitArray b10 = new BitArray(new byte[] { bytes[10] });

BitArray b10\_3 = new BitArray(3);

b10\_3[0] = b10[0]; b10\_3[1] = b10[1]; b10\_3[2] = b10[2];

long bsize = GetIntFromBitArray(b10\_3);

int possibleMessageLength = (int)(256 - Math.Pow(2, bsize)) \* 3;

int possibleTextLength = possibleMessageLength - 3;

return possibleTextLength;

}

public static long GetPaletteSize(string file)

{

byte[] bytes = File.ReadAllBytes(file);

BitArray b10 = new BitArray(new byte[] { bytes[10] });

BitArray b10\_3 = new BitArray(3);

b10\_3[0] = b10[0]; b10\_3[1] = b10[1]; b10\_3[2] = b10[2];

long paletteSize = GetIntFromBitArray(b10\_3);

return paletteSize;

}

public static long GetIntFromBitArray(BitArray bitArray)

{

long value = 0;

for (int i = 0; i < bitArray.Count; i++)

{

if (bitArray[i])

value += Convert.ToInt64(Math.Pow(2, i));

}

return value;

}

}