Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проектированию

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма раскрашивания графов»

Выполнила:

студентка группы 20ВВ4

Кожевникова А.В.

Приняла:

Юрова О. В.

Пенза 2021

**Содержание**

Реферат 5

Введение 6

Постановка задачи 7

1. Теоретическая часть 8

1.1. Некоторые теоремы, относящиеся к хроматическим числам 8

1.2. Точные алгоритмы раскраски 8

2. Описание алгоритма программы 11

3. Описание программы 13

4. Тестирование 16

5. Ручной расчёт задачи 21

6. Заключение 24

Список литературы 25

Приложение А. Листинг программы. 26

# Реферат

Отчет 35 стр., 20 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМЫ РАСКРАСКИ, ХРОМАТИЧЕСКОЕ ЧИСЛО, ГИПОТЕЗА ЧЕТЫРЕХ КРАСОК, ТЕОРЕМА О ПЯТИ КРАСКАХ

Цель исследования: создание программы для решения одной из задач теории графов – построение правильной раскраски.

В работе рассмотрены алгоритмы нахождения хроматического числа графа (построения раскраски) графа. Приведено описание точных и приближенных алгоритмов. Установлено, что, когда размеры графа слишком велики, получение оптимальной раскраски точными методами, затруднительно. Поэтому в работе использован последовательный алгоритм, основанный на упорядочивании множества вершин.

**ВВЕДЕНИЕ**

В теории графов раскраска графов является частным случаем разметки графов. При раскраске элементам графа ставятся в соответствие метки с учетом определенных ограничений; эти метки традиционно называются «цветами». В простейшем случае такой способ окраски вершин графа, при котором любым двум смежным вершинам соответствуют разные цвета, называется раскраской вершин. Аналогично раскраска ребер присваивает цвет каждому ребру так, чтобы любые два смежных ребра имели разные цвета. Наконец, раскраска областей планарного графа назначает цвет каждой области, так, что каждые две области, имеющие общую границу, не могут иметь одинаковый цвет. Раскраска вершин — главная задача раскраски графов, все остальные задачи в этой области могут быть сведены к ней.

Граф  называют *r*-хроматическим, если его вершины могут быть раскрашены с использованием *r* цветов так, что не найдется двух смежных вершин одного цвета. Наименьшее число *r*, такое, что граф *G* является *r*-хроматическим, называется хроматическим числом графа *G* и обозначается . Задача нахождения хроматического числа графа называется задачей о раскраске графа. Соответствующая этому числу раскраска вершин разбивает множество вершин графа на *r* подмножеств, каждое из которых содержит вершины одного цвета. Эти множества являются независимыми, поскольку в пределах одного множества нет двух смежных вершин.

Существует много алгоритмов раскрашивания графов, позволяющих находить хорошие приближения для хроматического числа графа в тех случаях, когда размеры графа слишком велики и получение оптимальной раскраски точными методами, затруднительно. В работе рассмотрен алгоритм “жадной раскраски” графа.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, язык программирования С++ / CLI.

Целью данной курсовой работы по дисциплине «Логика и основы алгоритмизации в инженерных задачах» является создание программы для решения одной из задач теории графов – раскрасить вершины графа минимальным количеством цветов.

# ПОСТАНОВКА ЗАДАЧИ

Задан граф *G*(*V*, *E*), где *V* - множество вершин; *E* - множество ребер. Раскрасить вершины графа красками так, чтобы ни одно его ребро не имело соцветных концов. Предусмотреть графическое представление исходного графа и цветовое выделение его вершин на экране.

Исходный граф в программе должен задаваться матрицей смежности. Пользователь задает количество вершин и может либо самостоятельно заполнить матрицу, либо дать программе самостоятельно сгенерировать. После обработки этих данных на экран должен выводиться граф с раскрашенными и пронумерованными вершинами. Устройство ввода - клавиатура.

# Теоретическая часть

Разнообразные задачи, возникающие при планировании производства, составлении графиков осмотра, хранении и транспортировке товаров и т. д., могут быть представлены часто как задачи теории графов, тесно связанные с так называемой «задачей раскраски». Графы, рассматриваемые далее, являются неориентированными и не имеют петель.

Раскраской вершин графа G=(X,U) называется разбиение множества вершин X на L непересекающихся классов (подмножеств) таких, что внутри каждого подмножества Xi не должно содержаться смежных вершин. Если каждому подмножеству Xi поставить в соответствие определенный цвет, то вершины внутри этого подмножества можно окрасить в один цвет, а вершины другого подмножества Xj - в другой цвет, и так далее, до раскраски всех подмножеств.

Задача нахождения хроматического числа произвольного графа явилась предметом многих исследований в конце XIX и в текущем столетии. Сейчас по этому вопросу известно большое количество интересных результатов. Мы вводим только такие понятия, которые нужны для построения алгоритмов решения задачи о раскраске графа.

## Некоторые теоремы, относящиеся к хроматическим числам

*Гипотеза четырех красок*

Возникновение гипотезы четырех красок исторически связано с раскрашиванием географических карт. Если имеется карта с изображением нескольких стран, то граничащие страны необходимо раскрасить в разные цвета. Если каждой стране сопоставить вершину графа, и соединить вершины ребром в случае, если страны граничат, то мы получим планарный граф

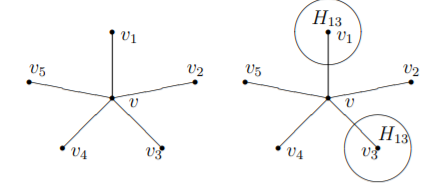
*Теорема о пяти красках*

Вершины любого планарного графа можно покрасить в пять цветов так, чтобы любые две смежные вершины были разных цветов (данный способ покраски в математике называют правильным), или, что то же самое, хроматическое число планарного графа не больше 5.

Доказательство: Пусть G – планарный граф с p вершинами, и что все планарные графы с p − 1 вершинами 5-раскрашиваемы. Без потери общности можно считать G простым графом, тогда (по следствию из формулы Эйлера) он содержит вершину v, степень которой не больше пяти. Удалим эту вершину вместе с инцидентными ей ребрами, получим планарный граф с p − 1 вершинами, который 5-раскрашиваем по индуктивному предположению.

Если d(v) < 5, то вершину v можно окрасить в любой цвет, не участвующий в окраске смежных вершин (которых не более четырех), и в этом случае доказательство закончено. Поэтому предположим, что d(v) = 5, и что смежные с v вершины расположены вокруг v по часовой стрелке. Если какие-либо из смежных вершин окрашены в одинаковый цвет, то вершину v можно окрасить в цвет, не задействованный в окраске смежных вершин.

Итак, мы пришли к случаю, когда все вершины v1, . . . , v5 окрашены в разные цвета, пусть это будут c1, . . . , c5 соответственно. Определим подграф Hij графа G0 как правильный подграф, содержащий все вершины, окрашенные в цвета ci и cj.



Вершины v1 и v3 лежат в разных компонентах связности графа H13. Тогда перекрашиваем вершины, вошедшие в ту же компоненту, что и v1: вершины, окрашенные в цвет c1, перекрашиваем в цвет c3, а вершины, окрашенные в цвет c3, перекрашиваем в цвет c1. Теперь вершина v1 окрашена в цвет c3, как и v3, а это означает, что вершину v можно окрасить в цвет c1.

## Алгоритмы раскраски

Существует большое количество эвристических алгоритмов раскраски графа. Большую практическую ценность на данный момент представляют следующие:

1. *Жадный алгоритм раскраски (Greedy-Colour):*

 Для упорядочения вершин для жадного алгоритма выбирают вершину *v* с минимальной степенью, упорядочивают остальные вершины, а *v* помещают в конец списка. Если любой подграф графа *G* содержит вершины со степенью, не превосходящей *d*, то алгоритм жадной раскраски для такого порядка вершин использует максимум *d* + 1 цветов. Наименьшее из *d* обычно называется вырожденностью графа.

Для графа с максимальной степенью Δ любой жадный алгоритм использует максимум Δ + 1 цветов. Теорема Брукса утверждает, что за исключением двух исключений для раскраски необходимо не более Δ цветов. Одно из доказательств теоремы Брукса использует упорядочение вершин, при котором первые две вершины смежны конечной вершине, но не смежны между собой. Такая последовательность имеет для каждой вершины по меньшей мере одну предшествующую вершину, принадлежащую окрестности. Для последовательности вершин с такими свойствами жадный алгоритм использует максимум Δ цветов.

Шаг 1. Упорядочить вершины по не возрастанию степени;

Шаг 2. Окрасить первую вершину в цвет *r*=1;

Шаг 3. Рассмотрим выбранную в данный момент вершину и раскрасьте ее цветом с наименьшим номером, который не использовался ни в каких ранее окрашенных вершинах, прилегающих к ней. Если все ранее использованные цвета отображаются на вершинах, смежных с v, назначьте ему новый цвет.

Шаг 4. Пока не раскрашены оставшиеся вершины повторяем первые три шага.

1. *Раскраска с обменом цветами (Colour-with-Interchange)*,
2. *Последовательная раскраска графа без упорядочивания его вершин (Random-SequentialColour)*,
3. *Последовательная раскраска графа начиная с вершин максимальных степеней (Smallest-Last-Colour),*
4. *Раскраска с обменом цветами без упорядочивания вершин графа (Random-Sequential-Interchange-Colour),*
5. *Раскраска с обменом цветами с упорядочиванием вершин графа по убыванию их степеней (LargestFirst-Interchange-Colour)*,
6. *Последовательная раскраска с динамическим упорядочиванием вершин графа (Saturation-Colour)*.

Необходимо отметить, что все эти алгоритмы представляют собой последовательность простых алгоритмических действий, условных переходов и циклов без использования каких-либо битовых или математических операций.

# Описание алгоритма программы

Алгоритм раскраски графа

Раскрасить граф можно используя следующий алгоритм:

1 Найти в графе максимальное независимое множество вершин.

2 Раскрасить найденное множество в один цвет.

3 Удалить найденные вершины из графа.

4 Если остались не раскрашенные вершины, то повторять п. 1…3.

Для программной реализации алгоритма необходимо описать следующие переменные:

Int\*\* a – матрица смежности графа представленная в виде двумерного динамического массива;

Int n\_a – количество вершин;

Int width – ширина вершины;

Int height – высота вершины;

Int BoxHeight – высота блока, в котором отображается граф;

Int BoxWidth – ширина блока, в котором отображается граф;

Int\* x\_first\_node – массив с координатами X вершин графа;

Int\* y\_first\_node - массив с координатами Y вершин графа;

int\* colorR = new int[n\_a], \* colorG = new int[n\_a], \* colorB = new int[n\_a] – массивы содержащие числовые коды для раскрашивания вершин;

std::vector<std::vector<int>> AdjList – вектор из векторов, содержащий список смежности;

Graphics^ g1 – класс, который рисует граф;

Pen^ BluePen – объект для рисования линий и геометрических фигур;

SolidBrush^ drawBrush – объект для заполнения цветом геометрической фигуры.

В начале выполняется чтение размерности графа и помещение числа вершин и рёбер в переменную n\_a. Для массивов a, colorR, colorG, colorB, x\_first\_node, y\_first\_node выделяется память на n\_a элементов и в конце очищается методом delete. Далее из таблицы формируется матрица смежности a и выводится на экран.

Матрица смежности конвертируется в список смежности функцией std::vector<std::vector<int>> convert. Инициализирую unordered\_map <int, int> result, который хранит цвета всех вершин графа. Затем запускается цикл прохода по списку смежности AdjList. Проверяю цвета соседних вершин и сохраняю их в наборе. В этом же цикле присваиваю первой вершине цвет равный 1 и прохожу по соседним с раскрашенной вершинами.

Записываю цвета в result. Данные операции повторяются ровно столько сколько вершин у графа.

Выводим получившийся граф в виде:

Пронумерованный круг с определённым цветом и линиями до инцидентных вершин.

Полный код программы приведён в приложении А.

# Описание программы

Для написания данной программы использован язык программирования С++/CLI и среда Visual Studio.

С++ – это язык программирования общего назначения, хорошо известный своей эффективностью и переносимостью. Во многих случаях программы, написанные на С++, сравнимы по скорости с программами, написанными на языке ассемблера. При этом они имеют лучшую наглядность, и их более просто сопровождать.

Проект был создан в виде приложения Windows Form.

Рассмотрим событийные функции программы “Раскраска графа”, обрабатывающие элементы:

private: System::Void button\_create\_Click(System::Object^ sender, System::EventArgs^ e) – создание пустой матрицы смежности;

private: System::Void button\_generate\_Click(System::Object^ sender, System::EventArgs^ e) – заполнение матрицы смежности;

private: System::Void btn\_clear\_field\_Click(System::Object^ sender, System::EventArgs^ e) – очистка поля с графом;

private: System::Void btn\_generate\_graph\_Click(System::Object^ sender, System::EventArgs^ e) – создание раскрашенного графа и вывод его на экран;

private: System::Void btn\_return\_to\_main\_page\_Click(System::Object^ sender, System::EventArgs^ e) – возврат на главную страницу;

private: System::Void numericUpDown1\_ValueChanged(System::Object^ sender, System::EventArgs^ e) – изменение числа вершин графа;

private: System::Void pictureBox1\_ShowGraph\_Click(System::Object^ sender, System::EventArgs^ e) – элемент, отображающий граф;

System::Void Реализацияраскрашиванияграфа::UI\_graph::keep\_data\_btn\_Click(System::Object^ sender, System::EventArgs^ e) – считывание матрицы из таблицы.

Далее приведены функции программы “Раскраска графа”:

private: void ShowA() - отображение матрицы смежности;

private: void GraphColoring(std::vector<std::vector<int>> AdjList, Graphics^ g1,int\* y\_first\_node,int\* x\_first\_node) - раскраска графа;

std::vector<std::vector<int>> convert(std::vector<std::vector<int>> a) - конвертация матрицы смежности в список смежности;

После запуска программы на экране появляется главное окно программы (рис. 1).

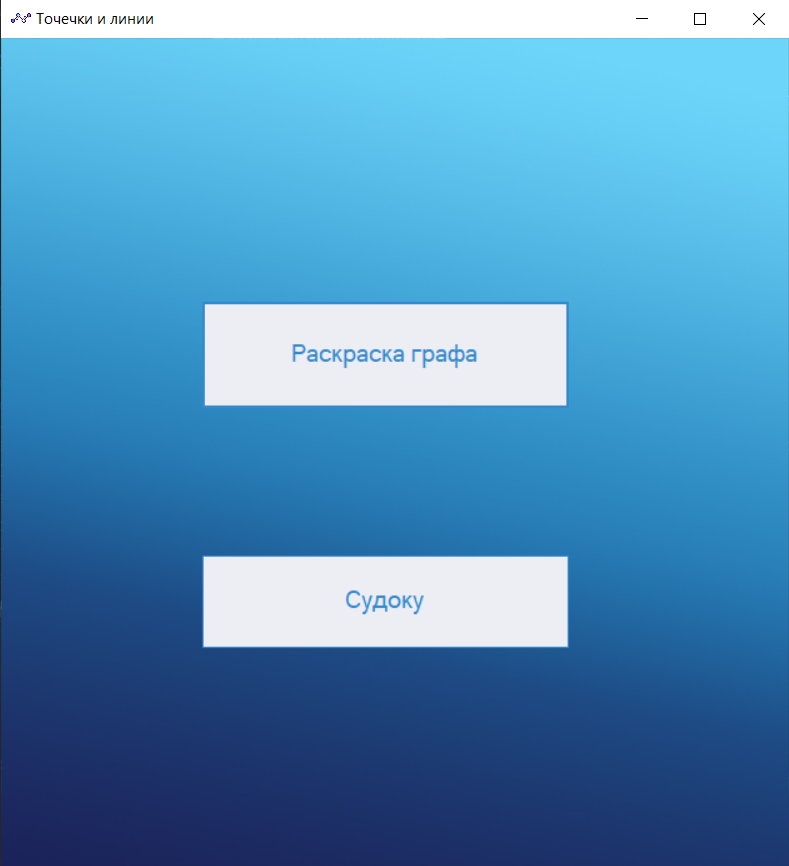


Рисунок 1 – главное окно программы

Пользователь может либо раскрасить граф, представленный матрицей, либо поиграть в игру под названием “Судоку”, которая является практическим применением алгоритма “раскраски графа”.

Если пользователь выбрал “Раскраску графа”, то перед ним появляется следующее окно (рис 2):

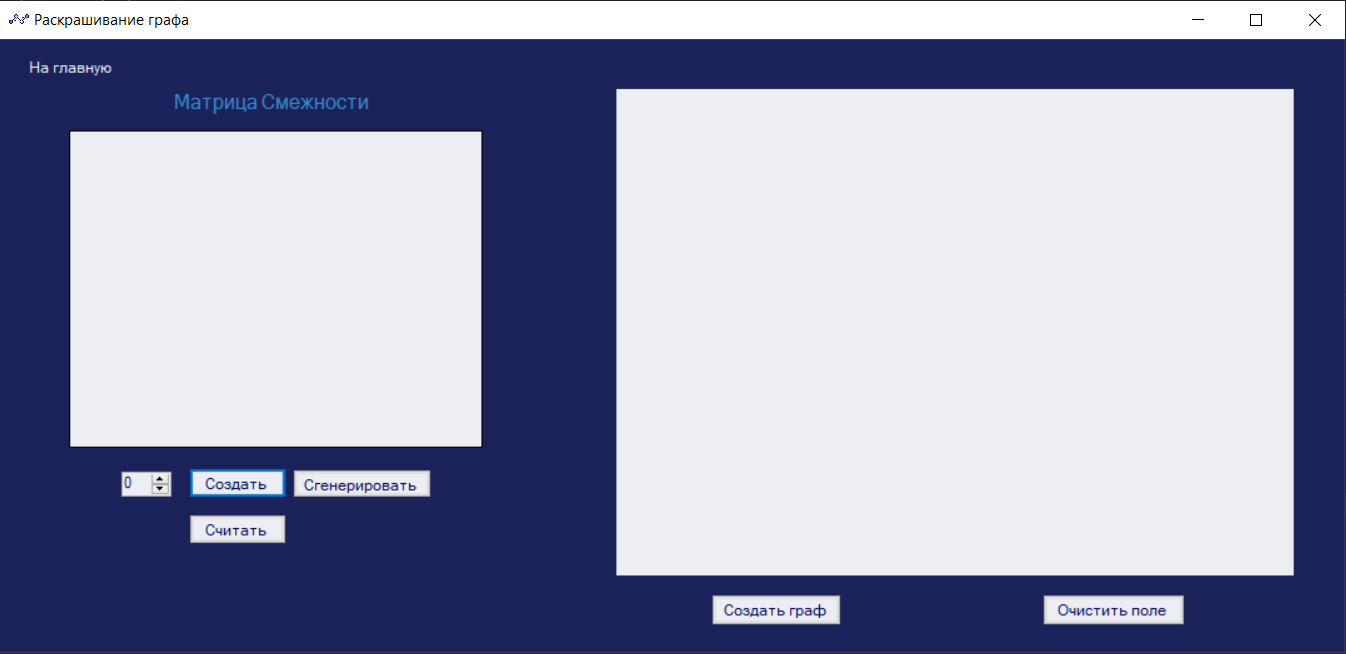


Рисунок 2 – окно с осуществлением алгоритма раскраски графа

Работа с программой производится в следующей последовательности:

1. Создание графа.

- Создаётся пустая матрица смежности с определённым количеством вершин. Пользователь самостоятельно заполняет её единицами или нулями. После нажимает на кнопку “Считать” и данные записываются в матрицу.

- Также пользователь может нажать на кнопку “Сгенерировать”, и программа самостоятельно введёт числа (единицы и нули).

2. Заполнение графа определёнными цветами.

Для построения и раскраски графа необходимо нажать на кнопку “Создать граф”. Результат выводится в виде изображения графа с раскрашенными и пронумерованными вершинами.

Если пользователь выбрал игру “Судоку”, то перед ним появляется следующее окно (рис 3):

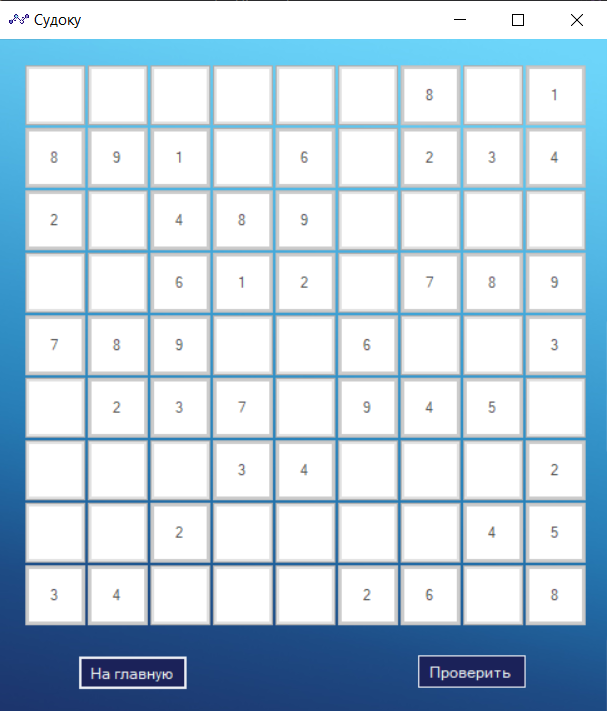


Рисунок 3 – Окно с игрой “Судоку”

Работа с программой производится в следующей последовательности:

1. Заполнение таблички с цифрами.

- Когда пользователь нажимает на пустую ячейку, появляются цифры от 1 до 9.

1. Проверка.

- Если пользователь уверен в правильности заполнения таблички, то он может нажать на кнопку “Проверить” и программа выведет либо окно о том, что всё верно заполнено (рис 4), либо что есть ошибка (рис 5).

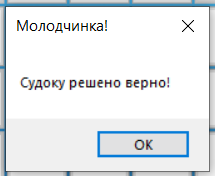


Рисунок 4.

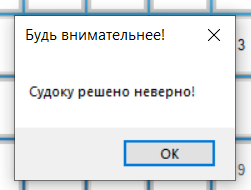


Рисунок 5 – окно ошибки.

# Тестирование

В данном разделе приведен результат тестирования программы при вводе пользователем различных графов.

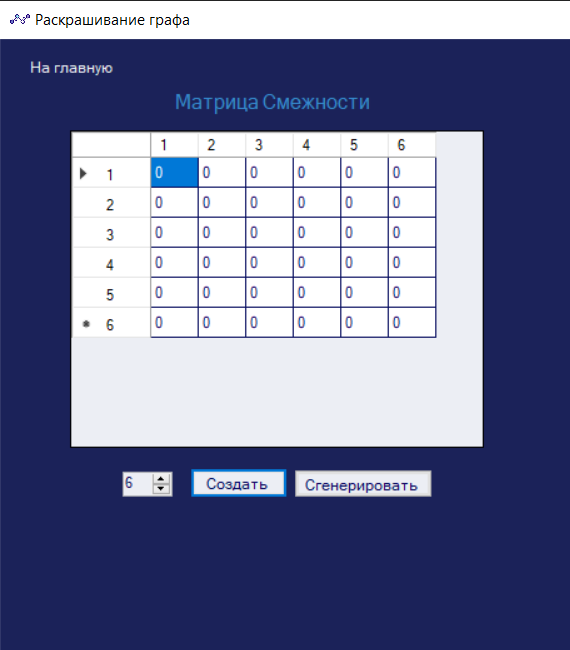


Рисунок 5 – изменение числа вершин.

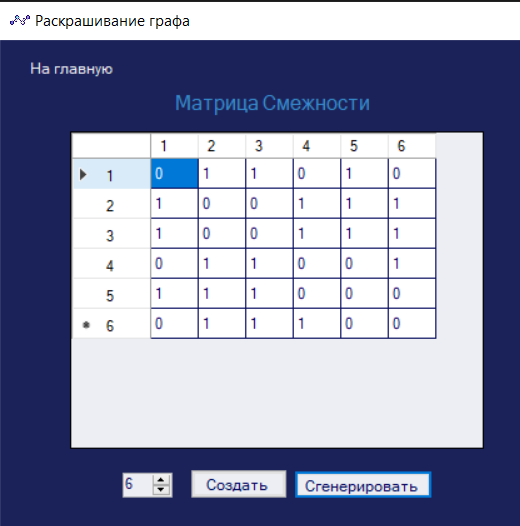


Рисунок 6 – случайное заполнение матрицы смежности.



Рисунок 7 – Заполнение матрицы с клавиатуры

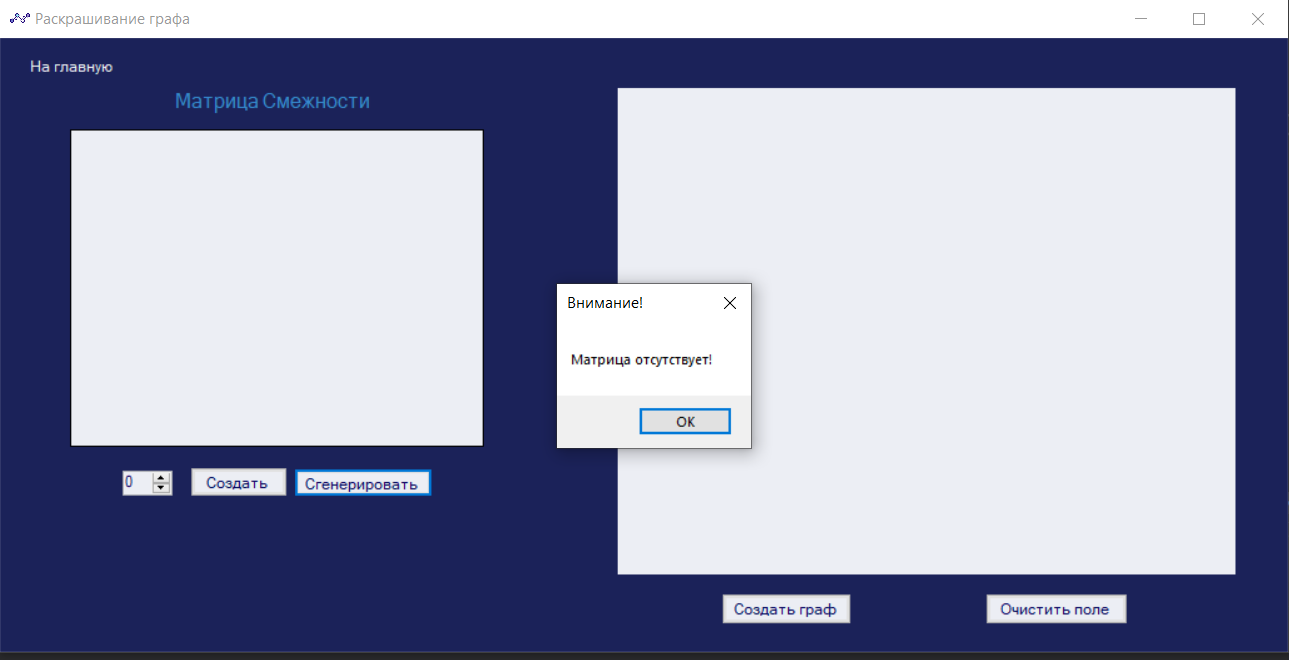


Рисунок 8 – Попытка заполнения несозданной матрицы.

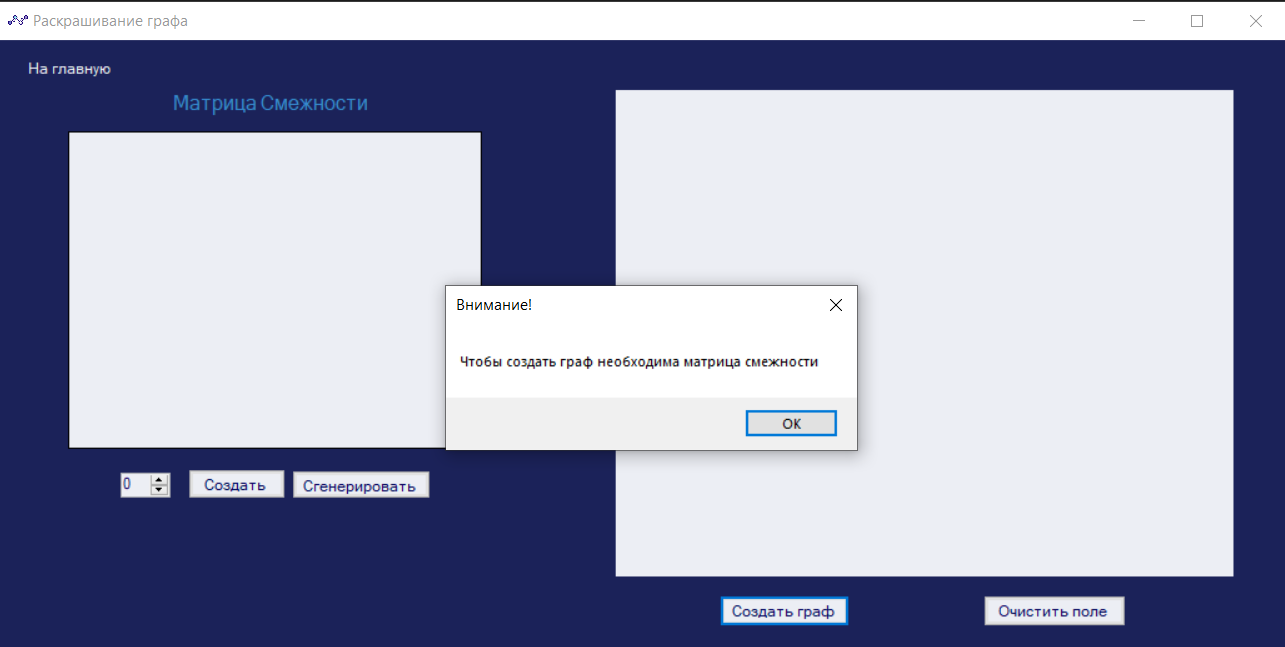


Рисунок 9 – Попытка создать граф когда матрица не создана.

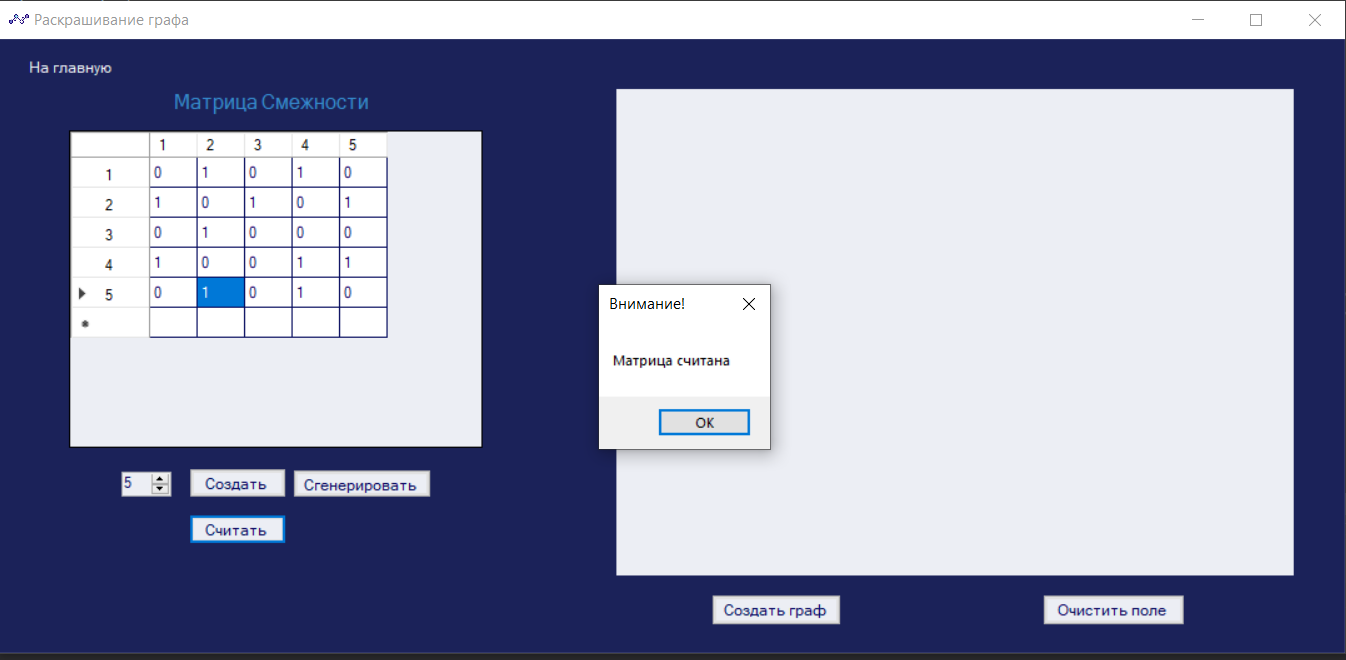


Рисунок 10 – окно об успешном считывании данных из матрицы

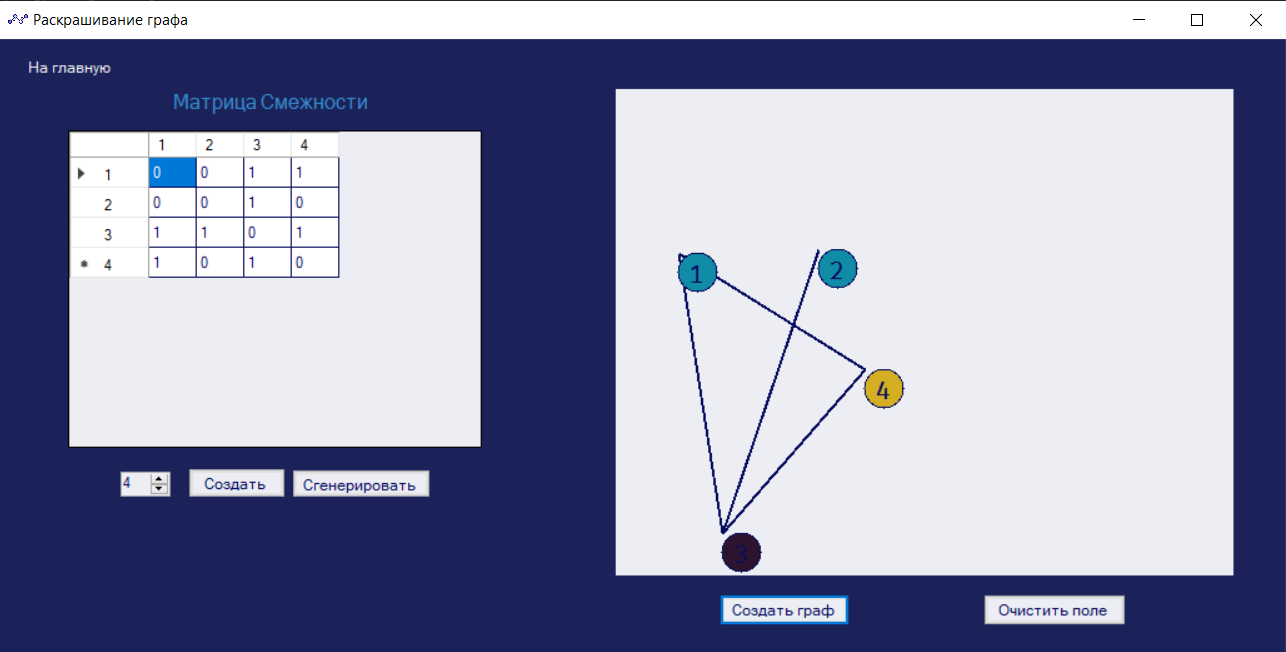


Рисунок 11 – Раскраска графа из 4х вершин.

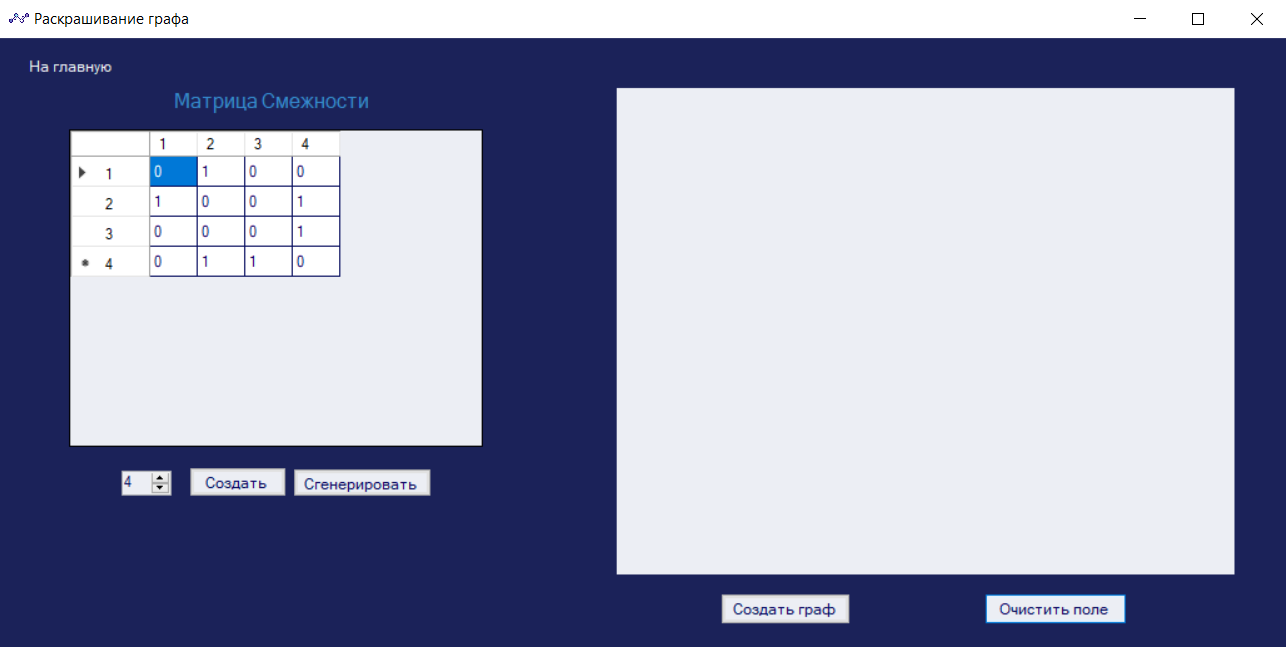


Рисунок 12 – Очистка поля с графом.

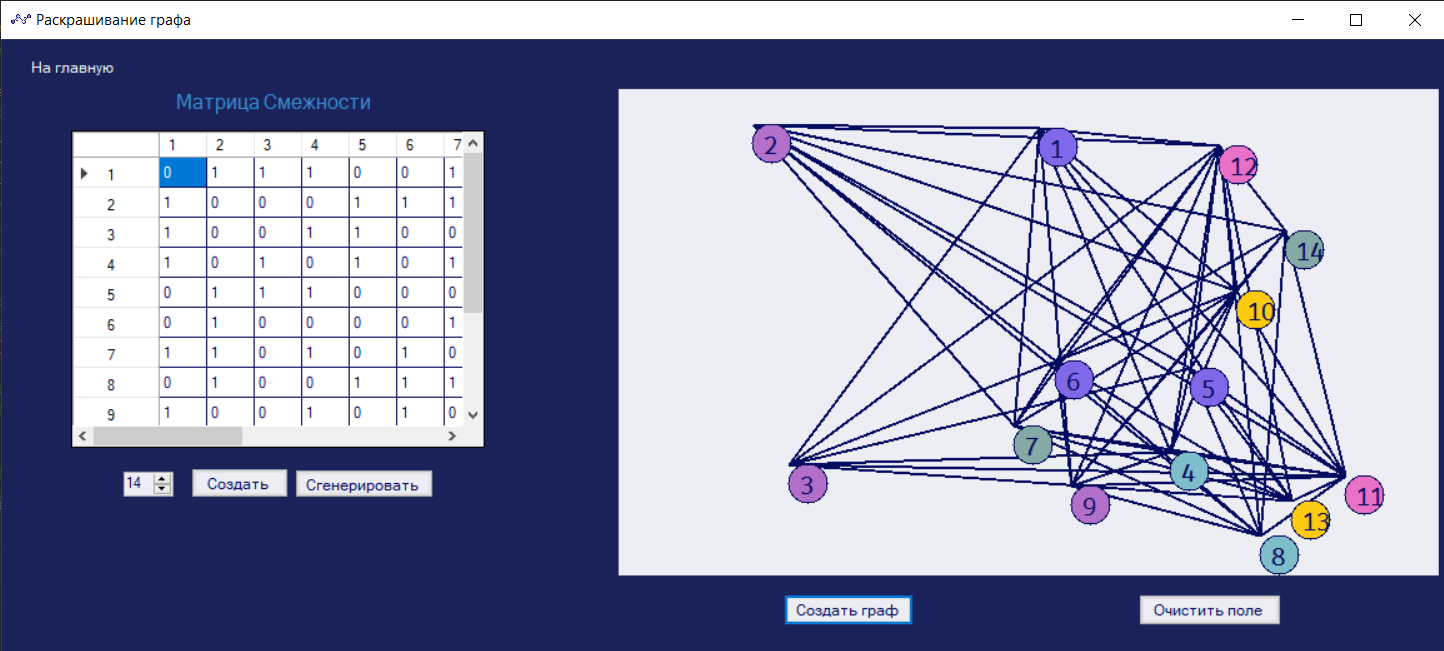


Рисунок 13 – Раскраска графа из 14 вершин

1. Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод окна программы | Верно |
| Изменение числа ребер | Изменение числа строк таблицы | Верно |
| Формирование случайного графа | Заполнение таблицы случайными значениями | Верно |
| Ввод графа с клавиатуры | Заполнение таблицы значениями, вводимыми пользователем | Верно |
| Пустая или слишком маленькая матрица смежности | Вывод сообщение об ошибке | Верно |
| Раскраска графа из 4 вершин | Вывод графа с раскрашенными вершинами | Верно |
| Раскраска произвольного графа из 14 вершин | Вывод графа с раскрашенными вершинами | Верно |
| Очищение поля с графом | Очищение поля с графом и его удаление из памяти устройства | Верно |

В результате тестирования было выявлено, что программа успешно выполняет свои функции.

# Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 7 вершинами.

Пример:

Раскрасить граф

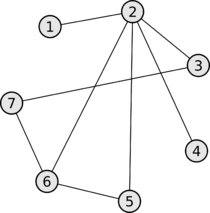
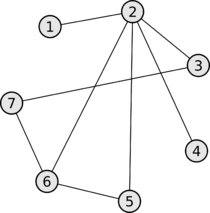


Рисунок 14 – Исходный граф.

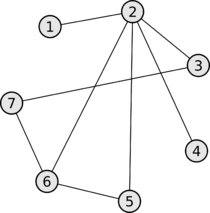
Рассматриваем вершины в порядке не возрастания их номера. Раскрашиваем первую вершину в цвет №1.



1

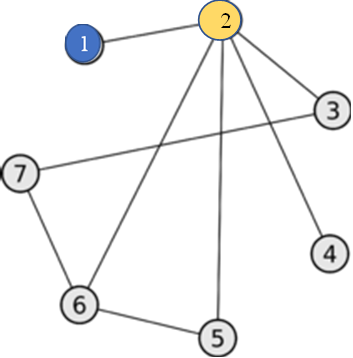
Рисунок 15 – Шаг 1.

Рассматриваем вершины в порядке не возрастания их номера. Следующая вершина связана с первой, поэтому закрашиваем её цветом №2.



2

Рисунок 16 – Шаг 2

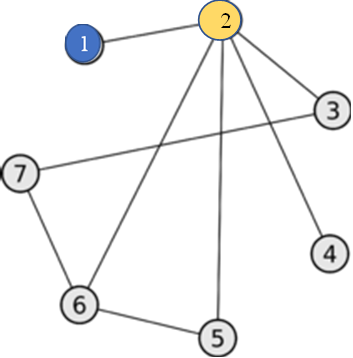
Рассматриваем вершины в порядке не возрастания их номера. Следующая вершина связана со второй, но не связана с первой, поэтому закрашиваем её цветом №1. Четвёртая вершина связана со второй, но не связана с третье, закрашиваем её цветом №1.

3

4

Рисунок 17 – Шаг 3

Рассматриваем вершины в порядке не возрастания их номера. Вершина номер пять не связана с четвёртой, но связана со второй, поэтому закрашиваем цветом №1. Следующая связана с пятой и второй, поэтому заполняем цветом №3



3

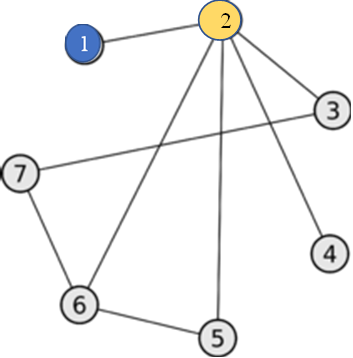
4

5

6

Рисунок 18 – Шаг 4

Рассматриваем вершины в порядке не возрастания их номера. Вершина номер семь связана с шестой и третьей, но не связана со второй, поэтому закрашиваем её цветом №2.



3

4

5

6

7

Рисунок 19 – Шаг 5

Все вершины графа раскрашены, число цветов 3. Из получившего графа создаём матрицу смежности и записываем в программу для проверки.

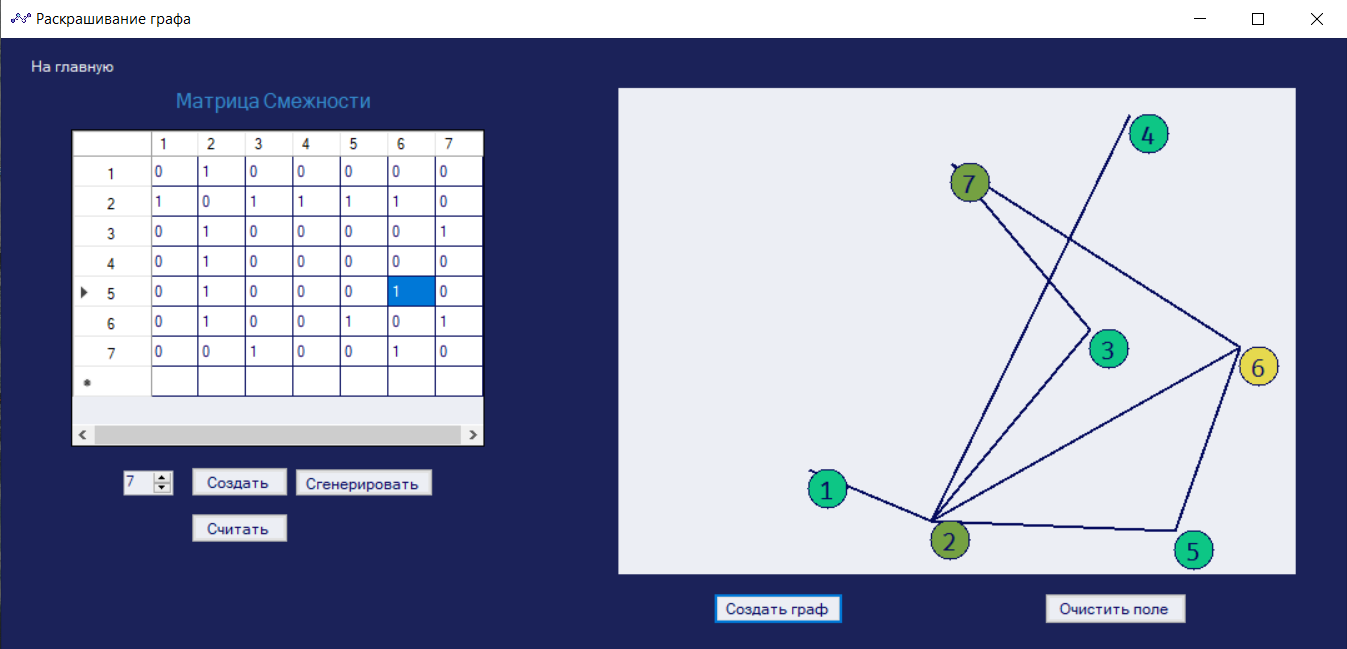


Рисунок 20 – Тестирование работы программы.

Исходя из полученных результатов, можно сделать вывод, что программа работает верно.

# Заключение

В результате выполнения работы была разработана программа, реализующая алгоритм раскраски графа в Microsoft Visual Studio 2019 на языке С++/ CLI.

В процессе выполнения курсовой работы была рассмотрена такая важная тема, как раскраска графов. Задачи подобного рода встречаются довольно часто в повседневной жизни – ярким примером того может служить раскраска географических карт, на которых не должно быть граничащих областей с одинаковыми цветами. Использование алгоритмов, дающих оптимальную раскраску минимальным количеством цветов, и их реализация на ЭВМ позволяет значительно сократить время (в сравнении с ручным счетом) на получение правильной раскраски.

Созданная программа позволяет выполнять следующие действия:

* вводить граф с клавиатуры матрицей смежности;
* генерировать случайный граф заданной полноты;
* играть в игру “Судоку”.

Программа имеет удобный оконный интерфейс, что позволяет наглядно выводить результат её работы.

Работа по созданию программы позволила приобрести практические навыки программирования прикладных задач теории графов. Одна из таких задач реализована в игре “Судоку”.

На основе контрольных примеров получились верные результаты, что позволяет сделать вывод о правильной реализации алгоритма программы.

# Список литературы

1. “Pro Visual C++/CLI and the .NET 2.0 Platform” Авторы: Stephen R.G. Fraser.
2. Документация Microsoft по Windows Form C++ <https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms?view=windowsdesktop-6.0>
3. “АЛГОРИТМ РАСКРАСКИ ГРАФА И ЕГО ПРИМЕНЕНИЕ В ОБЛАСТИ КОМПЬЮТЕРНЫХ СЕТЕЙ” - ФИЛОНЕНКО И.Н.1, ЧЕБОТАРЕВ М.В.1.
4. А.О. ПШЕНИЧНЫХ, С.Н. ГВОЗДЕВА, Э.И. ВАТУТИН -
5. “О ВЛИЯНИИ ПОРЯДКА РАССМОТРЕНИЯ ВЕРШИН ПРИ ПОИСКЕ РАСКРАСОК ГРАФОВ ОБЩЕГО ВИДА С ИСПОЛЬЗОВАНИЕМ ЖАДНОГО АЛГОРИТМА”.
6. Ф.Харари – “ТЕОРИЯ ГРАФОВ” М.: Мир, 1973, 300 стр.
7. “ПРИМЕНЕНИЕ ТЕОРИИ ГРАФОВ В ПРОГРАММИРОВАНИИ” -ЕВСТИГНЕЕВ ВЛАДИМИР АНАТОЛЬЕВИЧ.
8. “What Is C++\CLI?” - Vivek Ragunathan.
9. “ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C++ В СРЕДЕ VISUAL STUDIO CLR WINDOWS FORMS” - РЯЗАНОВА НАТАЛЬЯ, ТАССОВ КИРИЛЛ, ФИЛИППОВ МИХАИЛ.

# Приложение А. Листинг программы.

**Main\_page.h –** Заголовочный файл с элементами и функциями для реализации “Главного окна”.

#include "UI\_graph.h"

#include "Sudolu\_game.h"

namespace Реализацияраскрашиванияграфа {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace System::Drawing::Drawing2D;

/// <summary>

/// Сводка для Main\_page

/// </summary>

public ref class Main\_page : public System::Windows::Forms::Form

{

public:

Main\_page(void){

InitializeComponent();

}

protected:

~Main\_page()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::Button^ btn\_graph\_coloring;

private: System::Windows::Forms::Button^ btn\_sudoku;

protected:

protected:

private:

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

void InitializeComponent(void){

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(Main\_page::typeid));

this->btn\_graph\_coloring = (gcnew System::Windows::Forms::Button());

this->btn\_sudoku = (gcnew System::Windows::Forms::Button());

this->SuspendLayout();

//

// btn\_graph\_coloring

//

this->btn\_graph\_coloring->AutoEllipsis = true;

this->btn\_graph\_coloring->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)),

static\_cast<System::Int32>(static\_cast<System::Byte>(238)), static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->btn\_graph\_coloring->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->btn\_graph\_coloring->Font = (gcnew System::Drawing::Font(L"Dubai", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->btn\_graph\_coloring->ForeColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(53)),

static\_cast<System::Int32>(static\_cast<System::Byte>(136)), static\_cast<System::Int32>(static\_cast<System::Byte>(204)));

this->btn\_graph\_coloring->Location = System::Drawing::Point(161, 211);

this->btn\_graph\_coloring->Name = L"btn\_graph\_coloring";

this->btn\_graph\_coloring->Size = System::Drawing::Size(293, 85);

this->btn\_graph\_coloring->TabIndex = 1;

this->btn\_graph\_coloring->Text = L"Раскраска графа";

this->btn\_graph\_coloring->UseVisualStyleBackColor = false;

this->btn\_graph\_coloring->Click += gcnew System::EventHandler(this, &Main\_page::btn\_graph\_coloring\_Click);

//

// btn\_sudoku

//

this->btn\_sudoku->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)), static\_cast<System::Int32>(static\_cast<System::Byte>(238)),

static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->btn\_sudoku->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->btn\_sudoku->Font = (gcnew System::Drawing::Font(L"Dubai", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->btn\_sudoku->ForeColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(53)), static\_cast<System::Int32>(static\_cast<System::Byte>(136)),

static\_cast<System::Int32>(static\_cast<System::Byte>(204)));

this->btn\_sudoku->Location = System::Drawing::Point(161, 414);

this->btn\_sudoku->Name = L"btn\_sudoku";

this->btn\_sudoku->Size = System::Drawing::Size(293, 74);

this->btn\_sudoku->TabIndex = 2;

this->btn\_sudoku->Text = L"Судоку";

this->btn\_sudoku->UseVisualStyleBackColor = false;

this->btn\_sudoku->Click += gcnew System::EventHandler(this, &Main\_page::btn\_sudoku\_Click);

//

// Main\_page

//

this->AutoScaleDimensions = System::Drawing::SizeF(7, 22);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(27)), static\_cast<System::Int32>(static\_cast<System::Byte>(34)),

static\_cast<System::Int32>(static\_cast<System::Byte>(89)));

this->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"$this.BackgroundImage")));

this->ClientSize = System::Drawing::Size(630, 664);

this->Controls->Add(this->btn\_sudoku);

this->Controls->Add(this->btn\_graph\_coloring);

this->Font = (gcnew System::Drawing::Font(L"Dubai", 7.8F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->Icon = (cli::safe\_cast<System::Drawing::Icon^>(resources->GetObject(L"$this.Icon")));

this->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->Name = L"Main\_page";

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"Точечки и линии";

this->Load += gcnew System::EventHandler(this, &Main\_page::Main\_page\_Load);

this->ResumeLayout(false);

}

#pragma endregion

private: System::Void btn\_graph\_coloring\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void btn\_sudoku\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void Main\_page\_Load(System::Object^ sender, System::EventArgs^ e);

};

}

**Main\_page.cpp –** Реализация функций, представленных в предыдущем файле.

#include "Main\_page.h"

using namespace System;

using namespace System::Windows::Forms;

using namespace System::Drawing;

[STAThreadAttribute]

void main(array<String^>^ args) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Реализацияраскрашиванияграфа::Main\_page formMain;

Application::Run(% formMain);

}

System::Void Реализацияраскрашиванияграфа::Main\_page::btn\_graph\_coloring\_Click(System::Object^ sender, System::EventArgs^ e){

UI\_graph^ Graph = gcnew UI\_graph();//Создаём новый экземпляр

Graph->Show();//Открываем форму с раскраской графа

Main\_page::Hide();//закрывается предыдущая

return System::Void();

}

System::Void Реализацияраскрашиванияграфа::Main\_page::btn\_sudoku\_Click(System::Object^ sender, System::EventArgs^ e){

Sudolu\_game^ Sudoku = gcnew Sudolu\_game();

Sudoku->Show();

Main\_page::Hide();

return System::Void();

}

System::Void Реализацияраскрашиванияграфа::Main\_page::Main\_page\_Load(System::Object^ sender, System::EventArgs^ e){

return System::Void();

}

**UI\_grapg.h –** заголовочный файл с элементами и функциями для реализации “Раскраски графа”.

#include "Main\_page.h"

#include <vector>

#include <math.h>

#include <unordered\_map>

#include <set>

namespace Реализацияраскрашиванияграфа {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Collections::Generic;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

public ref class UI\_graph : public System::Windows::Forms::Form

{

public:

UI\_graph(void){

InitializeComponent();

}

protected:

~UI\_graph(){

if (components){

delete components;

}

}

protected:

private: System::Windows::Forms::Label^ label1;

private: static bool flag = false;

private: System::Windows::Forms::Button^ button\_generate;

private: System::Windows::Forms::Button^ button\_create;

private: System::Windows::Forms::Button^ btn\_generate\_graph;

private: System::Windows::Forms::Button^ btn\_clear\_field;

private: System::Windows::Forms::PictureBox^ pictureBox1\_ShowGraph;

private: System::Windows::Forms::DataGridView^ Adj\_matrix\_dataGridView1;

private: System::Windows::Forms::NumericUpDown^ numericUpDown1;

private: System::Windows::Forms::Button^ btn\_return\_to\_main\_page;

private: System::Windows::Forms::Button^ keep\_data\_btn;

protected:

private:

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

void InitializeComponent(void)

{

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(UI\_graph::typeid));

this->label1 = (gcnew System::Windows::Forms::Label());

this->button\_create = (gcnew System::Windows::Forms::Button());

this->button\_generate = (gcnew System::Windows::Forms::Button());

this->btn\_generate\_graph = (gcnew System::Windows::Forms::Button());

this->btn\_clear\_field = (gcnew System::Windows::Forms::Button());

this->pictureBox1\_ShowGraph = (gcnew System::Windows::Forms::PictureBox());

this->Adj\_matrix\_dataGridView1 = (gcnew System::Windows::Forms::DataGridView());

this->numericUpDown1 = (gcnew System::Windows::Forms::NumericUpDown());

this->btn\_return\_to\_main\_page = (gcnew System::Windows::Forms::Button());

this->keep\_data\_btn = (gcnew System::Windows::Forms::Button());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1\_ShowGraph))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->Adj\_matrix\_dataGridView1))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->BeginInit();

this->SuspendLayout();

//

// label1

//

this->label1->AutoSize = true;

this->label1->BackColor = System::Drawing::Color::Transparent;

this->label1->Font = (gcnew System::Drawing::Font(L"Dubai", 10.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->label1->ForeColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(53)), static\_cast<System::Int32>(static\_cast<System::Byte>(136)),

static\_cast<System::Int32>(static\_cast<System::Byte>(204)));

this->label1->Location = System::Drawing::Point(181, 49);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(219, 29);

this->label1->TabIndex = 2;

this->label1->Text = L"Матрица Смежности";

//

// button\_create

//

this->button\_create->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)), static\_cast<System::Int32>(static\_cast<System::Byte>(238)),

static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->button\_create->Cursor = System::Windows::Forms::Cursors::Hand;

this->button\_create->Font = (gcnew System::Drawing::Font(L"Dubai", 8));

this->button\_create->ForeColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(5)), static\_cast<System::Int32>(static\_cast<System::Byte>(11)),

static\_cast<System::Int32>(static\_cast<System::Byte>(94)));

this->button\_create->Location = System::Drawing::Point(203, 422);

this->button\_create->Name = L"button\_create";

this->button\_create->Size = System::Drawing::Size(104, 29);

this->button\_create->TabIndex = 3;

this->button\_create->Text = L"Создать";

this->button\_create->UseVisualStyleBackColor = false;

this->button\_create->Click += gcnew System::EventHandler(this, &UI\_graph::button\_create\_Click);

//

// button\_generate

//

this->button\_generate->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)), static\_cast<System::Int32>(static\_cast<System::Byte>(238)),

static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->button\_generate->Cursor = System::Windows::Forms::Cursors::Hand;

this->button\_generate->Font = (gcnew System::Drawing::Font(L"Dubai", 8));

this->button\_generate->ForeColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(5)), static\_cast<System::Int32>(static\_cast<System::Byte>(11)),

static\_cast<System::Int32>(static\_cast<System::Byte>(94)));

this->button\_generate->Location = System::Drawing::Point(313, 423);

this->button\_generate->Name = L"button\_generate";

this->button\_generate->Size = System::Drawing::Size(148, 28);

this->button\_generate->TabIndex = 4;

this->button\_generate->Text = L"Сгенерировать";

this->button\_generate->UseVisualStyleBackColor = false;

this->button\_generate->Click += gcnew System::EventHandler(this, &UI\_graph::button\_generate\_Click);

//

// btn\_generate\_graph

//

this->btn\_generate\_graph->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)),

static\_cast<System::Int32>(static\_cast<System::Byte>(238)), static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->btn\_generate\_graph->Cursor = System::Windows::Forms::Cursors::Hand;

this->btn\_generate\_graph->Font = (gcnew System::Drawing::Font(L"Dubai", 8));

this->btn\_generate\_graph->Location = System::Drawing::Point(760, 547);

this->btn\_generate\_graph->Name = L"btn\_generate\_graph";

this->btn\_generate\_graph->Size = System::Drawing::Size(139, 31);

this->btn\_generate\_graph->TabIndex = 6;

this->btn\_generate\_graph->Text = L"Создать граф";

this->btn\_generate\_graph->UseVisualStyleBackColor = false;

this->btn\_generate\_graph->Click += gcnew System::EventHandler(this, &UI\_graph::btn\_generate\_graph\_Click);

//

// btn\_clear\_field

//

this->btn\_clear\_field->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)), static\_cast<System::Int32>(static\_cast<System::Byte>(238)),

static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->btn\_clear\_field->Cursor = System::Windows::Forms::Cursors::Hand;

this->btn\_clear\_field->Font = (gcnew System::Drawing::Font(L"Dubai", 8));

this->btn\_clear\_field->Location = System::Drawing::Point(1113, 547);

this->btn\_clear\_field->Name = L"btn\_clear\_field";

this->btn\_clear\_field->Size = System::Drawing::Size(152, 31);

this->btn\_clear\_field->TabIndex = 7;

this->btn\_clear\_field->Text = L"Очистить поле";

this->btn\_clear\_field->UseVisualStyleBackColor = false;

this->btn\_clear\_field->Click += gcnew System::EventHandler(this, &UI\_graph::btn\_clear\_field\_Click);

//

// pictureBox1\_ShowGraph

//

this->pictureBox1\_ShowGraph->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Bottom)

| System::Windows::Forms::AnchorStyles::Left)

| System::Windows::Forms::AnchorStyles::Right));

this->pictureBox1\_ShowGraph->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)),

static\_cast<System::Int32>(static\_cast<System::Byte>(238)), static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->pictureBox1\_ShowGraph->Location = System::Drawing::Point(658, 49);

this->pictureBox1\_ShowGraph->Name = L"pictureBox1\_ShowGraph";

this->pictureBox1\_ShowGraph->Size = System::Drawing::Size(722, 479);

this->pictureBox1\_ShowGraph->SizeMode = System::Windows::Forms::PictureBoxSizeMode::StretchImage;

this->pictureBox1\_ShowGraph->TabIndex = 8;

this->pictureBox1\_ShowGraph->TabStop = false;

this->pictureBox1\_ShowGraph->Click += gcnew System::EventHandler(this, &UI\_graph::pictureBox1\_ShowGraph\_Click);

//

// Adj\_matrix\_dataGridView1

//

this->Adj\_matrix\_dataGridView1->AllowUserToOrderColumns = true;

this->Adj\_matrix\_dataGridView1->BackgroundColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)),

static\_cast<System::Int32>(static\_cast<System::Byte>(238)), static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->Adj\_matrix\_dataGridView1->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->Adj\_matrix\_dataGridView1->GridColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(5)),

static\_cast<System::Int32>(static\_cast<System::Byte>(11)), static\_cast<System::Int32>(static\_cast<System::Byte>(94)));

this->Adj\_matrix\_dataGridView1->Location = System::Drawing::Point(74, 90);

this->Adj\_matrix\_dataGridView1->Name = L"Adj\_matrix\_dataGridView1";

this->Adj\_matrix\_dataGridView1->RowHeadersWidth = 40;

this->Adj\_matrix\_dataGridView1->RowTemplate->Height = 24;

this->Adj\_matrix\_dataGridView1->Size = System::Drawing::Size(441, 312);

this->Adj\_matrix\_dataGridView1->TabIndex = 9;

this->Adj\_matrix\_dataGridView1->CellContentClick += gcnew System::Windows::Forms::DataGridViewCellEventHandler(this, &UI\_graph::Adj\_matrix\_dataGridView1\_CellContentClick);

//

// numericUpDown1

//

this->numericUpDown1->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)), static\_cast<System::Int32>(static\_cast<System::Byte>(238)),

static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->numericUpDown1->ForeColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(5)), static\_cast<System::Int32>(static\_cast<System::Byte>(11)),

static\_cast<System::Int32>(static\_cast<System::Byte>(94)));

this->numericUpDown1->Location = System::Drawing::Point(131, 426);

this->numericUpDown1->Name = L"numericUpDown1";

this->numericUpDown1->Size = System::Drawing::Size(53, 22);

this->numericUpDown1->TabIndex = 10;

this->numericUpDown1->ValueChanged += gcnew System::EventHandler(this, &UI\_graph::numericUpDown1\_ValueChanged);

//

// btn\_return\_to\_main\_page

//

this->btn\_return\_to\_main\_page->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(27)),

static\_cast<System::Int32>(static\_cast<System::Byte>(34)), static\_cast<System::Int32>(static\_cast<System::Byte>(89)));

this->btn\_return\_to\_main\_page->Cursor = System::Windows::Forms::Cursors::Hand;

this->btn\_return\_to\_main\_page->FlatAppearance->BorderSize = 0;

this->btn\_return\_to\_main\_page->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->btn\_return\_to\_main\_page->Font = (gcnew System::Drawing::Font(L"Dubai", 7.8F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->btn\_return\_to\_main\_page->ForeColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)),

static\_cast<System::Int32>(static\_cast<System::Byte>(238)), static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->btn\_return\_to\_main\_page->Location = System::Drawing::Point(12, 12);

this->btn\_return\_to\_main\_page->Name = L"btn\_return\_to\_main\_page";

this->btn\_return\_to\_main\_page->Size = System::Drawing::Size(131, 33);

this->btn\_return\_to\_main\_page->TabIndex = 11;

this->btn\_return\_to\_main\_page->Text = L"На главную";

this->btn\_return\_to\_main\_page->UseVisualStyleBackColor = false;

this->btn\_return\_to\_main\_page->Click += gcnew System::EventHandler(this, &UI\_graph::btn\_return\_to\_main\_page\_Click);

//

// keep\_data\_btn

//

this->keep\_data\_btn->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)), static\_cast<System::Int32>(static\_cast<System::Byte>(238)),

static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->keep\_data\_btn->Font = (gcnew System::Drawing::Font(L"Dubai", 7.8F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->keep\_data\_btn->Location = System::Drawing::Point(203, 468);

this->keep\_data\_btn->Name = L"keep\_data\_btn";

this->keep\_data\_btn->Size = System::Drawing::Size(104, 29);

this->keep\_data\_btn->TabIndex = 12;

this->keep\_data\_btn->Text = L"Считать";

this->keep\_data\_btn->UseVisualStyleBackColor = false;

this->keep\_data\_btn->Click += gcnew System::EventHandler(this, &UI\_graph::keep\_data\_btn\_Click);

//

// UI\_graph

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->AutoSizeMode = System::Windows::Forms::AutoSizeMode::GrowAndShrink;

this->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(27)), static\_cast<System::Int32>(static\_cast<System::Byte>(34)),

static\_cast<System::Int32>(static\_cast<System::Byte>(89)));

this->BackgroundImageLayout = System::Windows::Forms::ImageLayout::None;

this->ClientSize = System::Drawing::Size(1436, 604);

this->Controls->Add(this->keep\_data\_btn);

this->Controls->Add(this->btn\_return\_to\_main\_page);

this->Controls->Add(this->numericUpDown1);

this->Controls->Add(this->Adj\_matrix\_dataGridView1);

this->Controls->Add(this->pictureBox1\_ShowGraph);

this->Controls->Add(this->btn\_clear\_field);

this->Controls->Add(this->btn\_generate\_graph);

this->Controls->Add(this->button\_generate);

this->Controls->Add(this->button\_create);

this->Controls->Add(this->label1);

this->ForeColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(5)), static\_cast<System::Int32>(static\_cast<System::Byte>(11)),

static\_cast<System::Int32>(static\_cast<System::Byte>(94)));

this->Icon = (cli::safe\_cast<System::Drawing::Icon^>(resources->GetObject(L"$this.Icon")));

this->Name = L"UI\_graph";

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"Раскрашивание графа";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1\_ShowGraph))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->Adj\_matrix\_dataGridView1))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private:

private: System::Void numericUpDown1\_ValueChanged(System::Object^ sender, System::EventArgs^ e);

private: System::Void Adj\_matrix\_dataGridView1\_CellContentClick(System::Object^ sender, System::Windows::Forms::DataGridViewCellEventArgs^ e);

private: System::Void button\_create\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void button\_generate\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void btn\_clear\_field\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void btn\_generate\_graph\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void pictureBox1\_ShowGraph\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void btn\_return\_to\_main\_page\_Click(System::Object^ sender, System::EventArgs^ e);

//отображение матрицы смежности:

private: void ShowA();

//раскраска графа:

private: void GraphColoring(std::vector<std::vector<int>> AdjList, Graphics^ g1,int\* y\_first\_node,int\* x\_first\_node);

private: System::Void keep\_data\_btn\_Click(System::Object^ sender, System::EventArgs^ e);

};

}

**UI\_graph.cpp –** Реализация функций, представленных в предыдущем файле.

#include "UI\_graph.h"

#include <ctime>

#include <random>

#include "Functions.h"

using namespace System;

using namespace System::Windows::Forms;

using namespace System::Drawing;

//данные

int\*\* a;

int n\_a = 0;

System::Void Реализацияраскрашиванияграфа::UI\_graph::button\_create\_Click(System::Object^ sender, System::EventArgs^ e){

//Размерность матрицы

n\_a = Convert::ToInt32(numericUpDown1->Value);

//Создадим матрицу

a = new int\* [n\_a];

for (int i = 0; i < n\_a; i++){

a[i] = new int[n\_a];

}

if (n\_a == 1) {

MessageBox::Show("Недостаточно информации для построения графа", "Внимание!");

return;

}

//заполним матрицу нулями

for (int i = 0; i < n\_a; i++){

for (int j = 0; j < n\_a; j++){

a[i][j] = 0;

}

}

//Создаём таблицу

Adj\_matrix\_dataGridView1->RowCount = n\_a;

Adj\_matrix\_dataGridView1->ColumnCount = n\_a;

//Выводим таблицу:

ShowA();

//Выравниваем таблицу:

Adj\_matrix\_dataGridView1->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);//ячейки

Adj\_matrix\_dataGridView1->AutoResizeColumns();//столбцы

return System::Void();

}

System::Void Реализацияраскрашиванияграфа::UI\_graph::button\_generate\_Click(System::Object^ sender, System::EventArgs^ e){

if (n\_a==0){

MessageBox::Show("Матрица отсутствует!", "Внимание!");

return;

}

if (flag == true){

flag = false;

//генерируем матрицу:

std::srand(time(NULL));

for (int i = 0; i < n\_a; i++){

for (int j = 0; j < n\_a; j++){

a[i][j] = rand() % 100;

if (i == j) {

a[i][j] = 0;

}

if (a[i][j] >= 0 && a[i][j] <= 30) {

a[i][j] = 0;

}

if (a[i][j] >= 31 && a[i][j] <= 100) {

a[i][j] = 1;

}

}

}

//Выводим таблицу:

ShowA();

//Выравниваем таблицу:

Adj\_matrix\_dataGridView1->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);//ячейки

Adj\_matrix\_dataGridView1->AutoResizeColumns();//столбцы

}

return System::Void();

}

System::Void Реализацияраскрашиванияграфа::UI\_graph::keep\_data\_btn\_Click(System::Object^ sender, System::EventArgs^ e){

flag = true;

for (int i = 0; i < n\_a; i++){

for (int j = 0; j < n\_a; j++){

a[i][j] = Convert::ToInt32(Adj\_matrix\_dataGridView1->Rows[i]->Cells[j]->FormattedValue->ToString());

}

}

//Выводим таблицу:

ShowA();

//Выравниваем таблицу:

Adj\_matrix\_dataGridView1->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);//ячейки

Adj\_matrix\_dataGridView1->AutoResizeColumns();//столбцы

MessageBox::Show("Матрица считана", "Внимание!");

return System::Void();

}

void Реализацияраскрашиванияграфа::UI\_graph::ShowA(){

//Вывод матрицы в таблицу

for (int i = 0; i < n\_a; i++){

for (int j = 0; j < n\_a; j++){

a[i][j] = a[j][i];

//Вывод номеров столбцов:

Adj\_matrix\_dataGridView1->Columns[j]->HeaderCell->Value = Convert::ToString(j + 1);

//Вывод номеров строк:

Adj\_matrix\_dataGridView1->Rows[i]->HeaderCell->Value = Convert::ToString(i + 1);

//Вывод значений матрицы

Adj\_matrix\_dataGridView1->Rows[i]->Cells[j]->Value = a[i][j];

}

}

}

int width = 30, height = 30, BoxHeight, BoxWidth;

int\* x\_first\_node = new int[n\_a], \* y\_first\_node = new int[n\_a];

void Реализацияраскрашиванияграфа::UI\_graph::GraphColoring(std::vector<std::vector<int>> AdjList, Graphics^ g1,int\* x\_first\_node,int\* y\_first\_node) {

///////////////////////////////////////////////////

//Ниже я создала три одномерных массива с тремя цветами и каждый массив заполняю случайными значениями от 2 до 256 чтобы потом использовать для расскраски вершин:

int\* colorR = new int[n\_a], \* colorG = new int[n\_a], \* colorB = new int[n\_a];

for (int i = 0; i < n\_a; i++){

colorR[i] = rand() % 257 + 2;

colorG[i] = rand() % 257 + 2;

colorB[i] = rand() % 257 + 2;

}

// keep track of the color assigned to each vertex

std::unordered\_map<int, int> result;

// assign a color to vertex one by one

for (int u = 0; u < n\_a; u++){

// set to store the color of adjacent vertices of `u`

std::set<int> assigned;

// check colors of adjacent vertices of `u` and store them in a set

for (int i : AdjList[u]){

if (result[i]) {

assigned.insert(result[i]);

}

}

// check for the first free color

int color = 1;

for (auto& c : assigned){

if (color != c) {

break;

}

color++;

}

// assign vertex `u` the first available color

result[u] = color;

}

SolidBrush^ drawBrush = gcnew SolidBrush(Color::FromArgb(5, 11, 94));

System::Drawing::Font^ FontForWin = gcnew System::Drawing::Font("Dubai", 16, FontStyle::Regular);

StringFormat^ drawFormat = gcnew StringFormat;

for (int v = 0; v < n\_a; v++) {

SolidBrush^ brush = gcnew SolidBrush(Color::FromArgb(colorR[result[v]], colorG[result[v]], colorB[result[v]]));

g1->FillEllipse(brush, x\_first\_node[v] + v \* 30, y\_first\_node[v], width, height);

String^ str\_v = Convert::ToString(v + 1);

g1->DrawString(str\_v, FontForWin, drawBrush, x\_first\_node[v] + v \* 30 + 5, y\_first\_node[v], drawFormat);

}

}

System::Void Реализацияраскрашиванияграфа::UI\_graph::btn\_generate\_graph\_Click(System::Object^ sender, System::EventArgs^ e){

if (n\_a == 0){

MessageBox::Show("Чтобы создать граф необходима матрица смежности", "Внимание!");

return;

}

Color color = Color::FromArgb(5, 11, 94);

TextFormatFlags flags = TextFormatFlags::Bottom | TextFormatFlags::EndEllipsis;

System::Drawing::Font^ FontForWin = gcnew System::Drawing::Font("Dubai", 16, FontStyle::Regular);

PictureBox^ box = pictureBox1\_ShowGraph;

Pen^ BluePen = gcnew Pen(Color::FromArgb(5, 11, 94), 2);

SolidBrush^ drawBrush = gcnew SolidBrush(Color::FromArgb(5, 11, 94));

Graphics^ g1 = pictureBox1\_ShowGraph->CreateGraphics();

Graphics^ line = pictureBox1\_ShowGraph->CreateGraphics();

StringFormat^ drawFormat = gcnew StringFormat;

BoxHeight = box->Height;

BoxWidth = box->Width;

//Конвертируем матрицу в список:

std::vector<std::vector<int>> arr;

arr.resize(n\_a);

for (int i = 0; i < n\_a; i++){

for (int j = 0; j < n\_a; j++){

arr[i].push\_back(a[i][j]);

}

}

std::vector<std::vector<int>> AdjList = convert(arr);

////////////////////////////////////////////////////////////////

srand(time(NULL));

for (int i = 0; i < n\_a; i++){

y\_first\_node[i] = rand() % (30 - BoxHeight);

x\_first\_node[i] = rand() % 350 + 10;

String^ str\_i = Convert::ToString(i + 1);

g1->DrawEllipse(BluePen, x\_first\_node[i] + i \* 30, y\_first\_node[i], width, height);

}

for (int i = 0; i < n\_a; ++i){

for (int j = 0; j < n\_a; ++j){

if (a[i][j] == 1){

line->DrawLine(BluePen, x\_first\_node[i] + i \* 30, y\_first\_node[i], x\_first\_node[j] + j \* 30, y\_first\_node[j]);

a[j][i]=0;

}

}

}

GraphColoring(AdjList, g1, x\_first\_node, y\_first\_node);

return System::Void();

}

System::Void Реализацияраскрашиванияграфа::UI\_graph::btn\_return\_to\_main\_page\_Click(System::Object^ sender, System::EventArgs^ e){

Main\_page^ mPage = gcnew Main\_page();//Создаём новый экземпляр

mPage->Show();//Открываем форму с главной страницей

UI\_graph::Hide();

return System::Void();

}

System::Void Реализацияраскрашиванияграфа::UI\_graph::btn\_clear\_field\_Click(System::Object^ sender, System::EventArgs^ e){

pictureBox1\_ShowGraph->Invalidate();

return System::Void();

}

**Sudolu\_game.h -** заголовочный файл, который хранит функции и объекты для реализации игры “Судоку”.

#include "Main\_page.h"

#include <map>

namespace Реализацияраскрашиванияграфа {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для Sudolu\_game

/// </summary>

public ref class Sudolu\_game : public System::Windows::Forms::Form{

public:

Sudolu\_game(void){

InitializeComponent();

GenerateMap();

}

protected:

~Sudolu\_game(){

if (components)

{

delete components;

}

}

private: System::Windows::Forms::Button^ btn\_return\_to\_main\_page;

private: System::Windows::Forms::Button^ btn\_check\_answer;

protected:

protected:

private:

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

void InitializeComponent(void){

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(Sudolu\_game::typeid));

this->btn\_return\_to\_main\_page = (gcnew System::Windows::Forms::Button());

this->btn\_check\_answer = (gcnew System::Windows::Forms::Button());

this->SuspendLayout();

//

// btn\_return\_to\_main\_page

//

this->btn\_return\_to\_main\_page->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(27)),

static\_cast<System::Int32>(static\_cast<System::Byte>(34)), static\_cast<System::Int32>(static\_cast<System::Byte>(89)));

this->btn\_return\_to\_main\_page->Cursor = System::Windows::Forms::Cursors::Hand;

this->btn\_return\_to\_main\_page->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->btn\_return\_to\_main\_page->Font = (gcnew System::Drawing::Font(L"Dubai", 7.8F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->btn\_return\_to\_main\_page->ForeColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)),

static\_cast<System::Int32>(static\_cast<System::Byte>(238)), static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->btn\_return\_to\_main\_page->Location = System::Drawing::Point(86, 608);

this->btn\_return\_to\_main\_page->Name = L"btn\_return\_to\_main\_page";

this->btn\_return\_to\_main\_page->Size = System::Drawing::Size(114, 32);

this->btn\_return\_to\_main\_page->TabIndex = 0;

this->btn\_return\_to\_main\_page->Text = L"На главную";

this->btn\_return\_to\_main\_page->UseVisualStyleBackColor = false;

this->btn\_return\_to\_main\_page->Click += gcnew System::EventHandler(this, &Sudolu\_game::btn\_return\_to\_main\_page\_Click);

//

// btn\_check\_answer

//

this->btn\_check\_answer->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(27)), static\_cast<System::Int32>(static\_cast<System::Byte>(34)),

static\_cast<System::Int32>(static\_cast<System::Byte>(89)));

this->btn\_check\_answer->Cursor = System::Windows::Forms::Cursors::Hand;

this->btn\_check\_answer->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->btn\_check\_answer->Font = (gcnew System::Drawing::Font(L"Dubai", 7.8F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->btn\_check\_answer->ForeColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(236)),

static\_cast<System::Int32>(static\_cast<System::Byte>(238)), static\_cast<System::Int32>(static\_cast<System::Byte>(244)));

this->btn\_check\_answer->Location = System::Drawing::Point(447, 607);

this->btn\_check\_answer->Name = L"btn\_check\_answer";

this->btn\_check\_answer->Size = System::Drawing::Size(114, 32);

this->btn\_check\_answer->TabIndex = 1;

this->btn\_check\_answer->Text = L"Проверить";

this->btn\_check\_answer->UseVisualStyleBackColor = false;

this->btn\_check\_answer->Click += gcnew System::EventHandler(this, &Sudolu\_game::btn\_check\_answer\_Click);

//

// Sudolu\_game

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackColor = System::Drawing::Color::White;

this->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"$this.BackgroundImage")));

this->ClientSize = System::Drawing::Size(648, 664);

this->Controls->Add(this->btn\_check\_answer);

this->Controls->Add(this->btn\_return\_to\_main\_page);

this->Icon = (cli::safe\_cast<System::Drawing::Icon^>(resources->GetObject(L"$this.Icon")));

this->Name = L"Sudolu\_game";

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"Судоку";

this->ResumeLayout(false);

}

#pragma endregion

//дополнительные функции:

cli::array<System::Windows::Forms::Button^, 2>^ buttons = gcnew cli::array<System::Windows::Forms::Button^, 2>(9, 9);

private: void GenerateMap();

private: void CreateMap();

private: void ShuffleMap(int i);

private: void MatrixTransposition();

private: void SwapBlocksInColumn();

private: void SwapBlocksInRow();

private: void SwapRowsInBlock();

private: void SwapColumnsInBlock();

private: void HideSells();

private: void OnCellPressed(System::Object^ sender, System::EventArgs^ e);

private: System::Void btn\_check\_answer\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void btn\_return\_to\_main\_page\_Click(System::Object^ sender, System::EventArgs^ e);

};

}

**Sudolu\_game.cpp** – Файл с реализацией функций, представленном в предыдущем файле.

#include "Sudolu\_game.h"

const int n = 3, sizeButton = 50, n\_map=2;

int\*\* map;

void Реализацияраскрашиванияграфа::Sudolu\_game::GenerateMap(){

cli::array<System::Windows::Forms::Button^, 2>^ buttons = gcnew cli::array<System::Windows::Forms::Button^, 2>(9, 9);

map = new int\* [n \* n];

for (int i = 0; i < n \* n; i++){

map[i] = new int[n \* n];

}

for (int i = 0; i < n\*n; i++){

for (int j = 0; j < n\*n; j++)

map[i][j] = 0;

}

for (int i = 0; i < n \* n; i++){

for (int j = 0; j < n\*n; j++){

map[i][j] = (i \* n + i / n + j) % (n \* n) + 1;

buttons[i, j] = gcnew Button();

}

}

Random^ r = gcnew Random();

//количество итераций по перемешиванию кнопок:

for (int i = 0; i < 10; i++){

ShuffleMap(r->Next(0, 5));

}

CreateMap();

HideSells();

}

void Реализацияраскрашиванияграфа::Sudolu\_game::CreateMap(){

for (int i = 0; i < n \* n; i++){

for (int j = 0; j < n \* n; j++){

Button^ button = gcnew Button();

buttons[i,j] = button;

button->Size = System::Drawing::Size(sizeButton, sizeButton);

button->Text = Convert::ToString(map[i][j]);

button->Click += gcnew EventHandler(this,&Sudolu\_game::OnCellPressed);

button->Location = Drawing::Point(j \* sizeButton+20, i \* sizeButton+20);

this->Controls->Add(button);

}

}

}

void Реализацияраскрашиванияграфа::Sudolu\_game::HideSells() {

int N = 40; //количество скрываемых клеток

Random^ r = gcnew Random();

while (N > 0) {

for (int i = 0; i < n \* n; i++){

for (int j = 0; j < n \* n; j++){

//если текущая кнопка пустая

if (!String::IsNullOrWhiteSpace(buttons[i, j]->Text)) {

int a = r->Next(0, 3);

buttons[i, j]->Text = Convert::ToString(a) == "0" ? " " : buttons[i, j]->Text;

//Скрываем кнопки, которые не нужно менять:

buttons[i, j]->Enabled = a == 0 ? true : false;

if (a == 0)

N--;

if (N <= 0)

break;

}

}

if (N <= 0)

break;

}

}

}

void Реализацияраскрашиванияграфа::Sudolu\_game::ShuffleMap(int i) {

switch (i) {

case 0:

MatrixTransposition();

break;

case 1:

SwapRowsInBlock();

break;

case 2:

SwapColumnsInBlock();

break;

case 3:

SwapBlocksInRow();

break;

case 4:

SwapBlocksInColumn();

break;

default:

MatrixTransposition();

break;

}

}

void Реализацияраскрашиванияграфа::Sudolu\_game::SwapBlocksInColumn() {

Random^ r = gcnew Random();

auto block1 = r->Next(0, n);

auto block2 = r->Next(0, n);

while (block1 == block2) {

block2 = r->Next(0, n);

}

block1 \*= n;

block2 \*= n;

for (int i = 0; i < n \* n; i++){

auto k = block2;

for (int j = block1; j < block1 + n; j++){

auto temp = map[i][j];

map[i][j] = map[i][k];

map[i][k] = temp;

k++;

}

}

}

void Реализацияраскрашиванияграфа::Sudolu\_game::SwapBlocksInRow() {

Random^ r = gcnew Random();

auto block1 = r->Next(0, n);

auto block2 = r->Next(0, n);

while (block1 == block2) {

block2 = r->Next(0, n);

}

block1 \*= n;

block2 \*= n;

for (int i = 0; i < n\*n; i++){

auto k = block2;

for (int j = block1; j < block1 + n; j++){

auto temp = map[j][i];

map[j][i] = map[k][i];

map[k][i] = temp;

k++;

}

}

}

void Реализацияраскрашиванияграфа::Sudolu\_game::SwapRowsInBlock() {

Random^ r = gcnew Random();

auto block = r->Next(0, n);

auto row1 = r->Next(0, n);

auto line1 = block \* n + row1;

auto row2 = r->Next(0, n);

while (row1 == row2) {

row2 = r->Next(0, n);

}

auto line2 = block \* n + row2;

for (int i = 0; i < n\*n; i++) {

auto temp = map[line1][i];

map[line1][i] = map[line2][i];

map[line2][i] = temp;

}

}

void Реализацияраскрашиванияграфа::Sudolu\_game::SwapColumnsInBlock() {

Random^ r = gcnew Random();

auto block = r->Next(0, n);

auto row1 = r->Next(0, n);

auto line1 = block \* n + row1;

auto row2 = r->Next(0, n);

while (row1 == row2) {

row2 = r->Next(0, n);

}

auto line2 = block \* n + row2;

for (int i = 0; i < n\*n; i++) {

auto temp = map[i][line1];

map[i][line1] = map[i][line2];

map[i][line2] = temp;

}

}

void Реализацияраскрашиванияграфа::Sudolu\_game::MatrixTransposition() {

int\*\* tMap = new int\*[n \* n];

for (int i = 0; i < n \* n; i++){

tMap[i] = new int[n \* n];

}

for (int i = 0; i < n\*n; i++){

for (int j = 0; j < n\*n; j++){

tMap[i][j] = map[j][i];

}

}

map = tMap;

}

void Реализацияраскрашиванияграфа::Sudolu\_game::OnCellPressed(System::Object^ sender, System::EventArgs^ e) {

Button^ pressedButton = (Button^)sender;

String^ buttonText = pressedButton->Text;

if (String::IsNullOrWhiteSpace(buttonText)){

pressedButton->Text = "1";

}

else{

int num = 0;

num = Convert::ToInt16(buttonText);

num++;

if (num == 10) {

num = 1;

}

pressedButton->Text = Convert::ToString(num);

}

}

System::Void Реализацияраскрашиванияграфа::Sudolu\_game::btn\_check\_answer\_Click(System::Object^ sender, System::EventArgs^ e) {

for (int i = 0; i < n\*n; i++){

for (int j = 0; j < n\*n; j++){

auto btnText = buttons[i, j]->Text;

if (btnText != Convert::ToString(map[i][j])) {

MessageBox::Show("Судоку решено неверно!", "Будь внимательнее!");

return;

}

}

}

MessageBox::Show("Судоку решено верно!", "Молодчинка!");

for (int i = 0; i < n\*n; i++){

for (int j = 0; j < n\*n; j++){

this->Controls->Remove(buttons[i, j]);

}

}

GenerateMap();

}

System::Void Реализацияраскрашиванияграфа::Sudolu\_game::btn\_return\_to\_main\_page\_Click(System::Object^ sender, System::EventArgs^ e) {

Main\_page^ mPage = gcnew Main\_page();//Создаём новый экземпляр

mPage->Show();//Открываем форму с главной страницей

Sudolu\_game::Hide();

return System::Void();

}