МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ
УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им. В.Ф. Уткина»

Кафедра САПР ВС

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине

«Автоматизация конструкторского и технологического проектирования»

по теме:

«Алгоритм Акерса»

Выполнил студент группы 846
Шмаков А.Ю.

Руководитель работы
к.т.н., доц. Сапрыкин А.Н.

Рязань 2022

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Рязанский государственный радиотехнический университет им. В.Ф. Уткина»

Факультет вычислительной техники

КУРСОВОЙ ПРОЕКТ

по курсу «Автоматизация конструкторского и технологического проектирования»

Студент(ы) Шмаков Андрей Юрьевич группа 846

Тема: Алгоритм Акерса

Срок представления работы к защите: «05» апреля 2022 г.

Исходные данные для проектирования: Персональный компьютер Intel; операционная система: Windows XP/Windows 7/ Windows 8/ Linux;

язык/среда программирования: Java/C#.

Требования к работе:

1. Листинг программы должен содержать поясняющие комментарии на русском языке.
2. Пояснительная записка представляется как в печатном, так и в электронном виде в соответствующем разделе личного портфолио студента на портале edu.rsreu.ru.
3. Программный модуль на портале edu.rsreu.ru как в виде скомпилированного EXE – файла, запускающего без предварительной инсталляции, так и исходных файлов.

Содержание пояснительной записки:

Бланк задания

Введение

1. Практическая и математическая постановка задачи
2. Анализ существующих алгоритмов решения задачи
3. Описание разрабатываемого алгоритма, его укрупненная схема
4. Развернутая блок-схема алгоритма
5. Руководство пользователя
6. Решение контрольного примера
7. Перечень идентификаторов, используемых при решении программы

Заключение

Список литературы

Приложение (листинг программного кода)

Задание принято к исполнению: Шмаков А.Ю.
_____ «10» февраля 2022 г.

Руководитель работы: доцент Сапрыкин А.Н.
_____ «10» февраля 2022 г.

Оглавление

Введение

Задача трассировки соединений отдельных компонентов возникает на последних этапах проектирования электронных средств (ЭС) и является одной из наиболее сложных задач конструкторского проектирования.

Сложность трассировки печатных соединений, составляющих значительную долю в общем объеме соединений, связана с постоянным увеличением функциональной насыщенности электронных узлов при одновременном снижении их габаритов. Это вызывает нехватку площади для размещения соединений, а также большое количество пересечений, что приводит к резкому увеличению числа проводящих слоев и, естественно, отражается на таких параметрах ЭС, как надежность, габариты, стоимость.

При трассировке печатных соединений «ручным» способом невозможно быстро просмотреть различные варианты и выбрать наиболее приемлемый для конкретного узла вследствие зависимости действий на последующих шагах трассировки от проведенных ранее соединений. Поэтому на разработку эффективных алгоритмов автоматической трассировки соединений направлены в настоящее время основные усилия в области автоматизации конструкторского проектирования.

1. Практическая и математическая постановка задачи

С практической точки зрения постановка задачи выглядит следующим образом: плата представляется в виде дискретного рабочего поля(ДПР) с ячейками, количество которых равно MxN, где M, N – размеры поля для трассировки. Каждый элемент ДПР отражает состояние ячейки – свободна, является начальной точкой трассировки, является конечной точкой трассировки, занята препятствием, является частью пути. Требуется построить кратчайший путь из начальной ячейки в конечную, при этом путь может включать только элементы, которые являются свободными.

Математическая постановка задачи выглядит следующим образом: необходимо найти путь из точки A в точку B, путем минимизации целевой функции весов ячеек, в зависимости от установленных заранее критериев построения пути.

2. Анализ существующих алгоритмов.

С математической точки зрения трассировка — наисложнейшая задача выбора из огромного числа вариантов оптимального решения. Существует большое количество алгоритмов трассировки печатной платы, их все можно разделить на 3 вида:

1)Волновые алгоритмы

2)Ортогональные алгоритмы

3)Эвристические алгоритмы

2.1 Алгоритм Акерса

В данном курсовом проекте будет рассмотрен один из волновых алгоритмов трассировки – алгоритм Акерса, который является улучшением алгоритма Ли.

Наиболее экономичный способ кодирования состояний ячеек коммутационного поля предложен Акерсом. При распространении волны ячейки поля получают отметки в соответствии с базовой последовательностью 1, 1, 2, 2, 1, 1, 2, 2, … . Данная последовательность характерна тем, что в ней любой член имеет разных соседей слева и справа. Вначале все незанятые ячейки, соседние с ячейкой-источником, помечаются 1, затем все ячейки фронта помечаются так же 1. Далее отметка 2 присваивается ячейкам фронта и т. д.

Для построения пути в общем случае необходимо найти требуемую под-последовательность отметок базовой последовательности. Это легко может быть сделано по отметке ячейки-цели и отметке соседней с ней ячейки. В нашем случае цель достигнута первой 1, поэтому надо искать ячейку с отметкой 2. Их две: внизу и слева. Воспользуемся приоритетом. Вначале просматриваем ячейку снизу от цели. Поскольку ее вес равен 2, путь строим в нее. В ней вновь анализируем соседние ячейки в порядке приоритета. Ищем ячейку с весом 2. Это опять нижняя ячейка. Далее надо аналогично искать ячейку с весом 1.

В методе Акерса ячейка поля может находиться в следующих состояниях: пустая, занятая, иметь отметку 1 или 2. Таким образом, на каждую ячейку поля достаточно всего два двоичных разряда памяти.

2.2 Алгоритм Рабина

Пусть требуется найти путь минимальной длины между ячейками A и B; C_i – некоторая ячейка ДРП, включаемая в очередной фронт волны. Значение весовой функции P_i, приписываемое Ci -й ячейке фронта, определяется по формуле:

$$P_i = L(i,A) + \eta(i,B) \quad (1)$$

где $L(i,A)$ – длина кратчайшего пути из A в ячейку C_i , а

$$\eta(i,B) = |x_i - x_B| + |y_i - y_B| \quad (2)$$

расстояние между ячейками Ci и B в ортогональной метрике.

Нетрудно видеть, что $\eta(i,B)$ является нижней оценкой длины пути из Ci в B, полученной в предположении отсутствия препятствий на этом пути. Отсюда выражение (1) в целом представляет собой нижнюю оценку длины любого пути из A в B, проходящего через ячейку Ci . Согласно методу ветвей и границ оптимальное решение следует искать в подмножестве решений, имеющем наилучшую оценку. В соответствии с этим в сформированном очередном фронте волны выделяется подмножество ячеек с минимальной оценкой (1), а распространение волны на следующем шаге осуществляется из ячеек этого подмножества.

2.3 Лучевой алгоритм

Основная идея алгоритма, предложенного Л. Б. Абрайтисом, заключается в исследовании поля для определения пути между ячейками A и B по некоторым заранее заданным направлениям, подобным лучам. Это позволяет сократить число просматриваемых алгоритмом ячеек, а, следовательно, и время на анализ и кодировку их состояний, однако снижает вероятность нахождения пути сложной конфигурации и усложняет учет конструктивных требований к технологии печатной платы.

Работа алгоритма заключается в следующем. Задается число лучей, распространяемых из ячейки А и В, а также порядок присвоения путевых координат. Обычно число лучей для каждой из ячеек (источников) принимают одинаковым (часто равным двум). Лучи A^1,A^2 ,…, A^m и B^1 , B^2 , …, B^m считаются одноименными, если они распространяются из одноименных источников А или В. Лучи A^i и B^i являются разноименными по отношению друг к другу. Распространение лучей происходит одновременно из обоих источников до встречи двух разноименных лучей в некоторой точке С.

Путь проводится из ячейки С, в которой встретились лучи по путевым координатам, и проходит через ячейки, по которым распространялись лучи.

При распространении луча может возникнуть ситуация, когда две соседние ячейки будут заняты. В этом случае луч считается заблокированным и его распространение прекращается.

2.4 Алгоритм слежения за целью

Так же, как и алгоритм Рабина, алгоритм слежения за целью обеспечивает отыскание пути между двумя ячейками ДРП всегда, когда он существует. В алгоритме в качестве значения весовой функции для ячейки Ci на этапе распространения волны используется выражение (2).

Таким образом, распространение волны происходит из тех же ячеек очередного фронта, которые ближе расположены к ячейке – цели (без учета возможных препятствий). При этом, очевидно, сокращается зона поиска, но из-за приближенного вычисления нижней оценки для подмножества путей, ведущих из А в В через ячейку Ci , возможна потеря глобального минимума.

Кроме того, на этапе «проведения пути» иногда возникает неоднозначность, связанная с отсутствием монотонного изменения оценок формируемых фронтов. Поэтому при восстановлении трассы используют обычно только путевые координаты («приоритетные направления»), что приводит к сложным конфигурациям соединений.

2.5 Кодирование ячеек по mod 3

Наиболее эффективным способом кодирования состояния ячеек ДРП является метод путевых координат по модулю 3. Он базируется на том, что Pk−1 ≠Pk ≠Pk+1. Ячейкам, включённым в последующие фронты, можно присваивать не сами веса, а их значения по модулю 3 (1,2,3,1,2,3…). Тогда количество разрядов на кодирование состояний ячеек будет равно N=log2 5=3. Проведение пути заключается в отслеживании отметок (1,2,3). Если ячейка имеет несколько соседних ячеек с одинаковыми метками, то используется правило приоритетных направлений.

Таким образом, кодирование ячеек по модулю 3, по сравнению с алгоритмом Ли, позволяет снизить затраты памяти и сделать их независимыми от максимально возможной длины пути

2.6 Трассировка соединений с использованием каналов

Особый тип алгоритмов предназначен для трассировки соединений на плоскости без пересечений. Они используются при проектировании однослойного монтажа при наличии в устройстве разнотипных по форме и размерам элементов.

В связи с тем, что ни один из известных алгоритмов не гарантирует полной трассировки при автоматизированном проектировании, считается целесообразным, чтобы в развитых системах автоматизированного конструкторского проектирования было несколько различных программ трассировки и имелась возможность их совместного использования при решении одной задачи. Оставшиеся непроложенными после трассировки соединения дорабатываются конструкторами вручную или в диалоговом режиме взаимодействия с ЭВМ.

Следует отметить, что при использовании САПР время проектирования топологии устройств, содержащих 20 ... 30 ИС, сокращается приблизительно на порядок, а для БИС практически нет другого способа достижения высокого качества проекта и его документации. Однако большим недостатком

современных систем автоматизированного конструкторского проектирования является необходимость огромного объема исходной информации, которая может готовиться порядка нескольких недель для БИС с уже разработанной логической схемой. Для преодоления этого недостатка необходимы создание САПР, соединяющих все этапы проектирования РЭА и имеющих интегрированную базу данных, содержащую инвариантную информацию, пополняемую в процессе разработки, а также стандартизация наиболее рациональных схемотехнических и конструктивных решений.

2.7 Сравнение алгоритмов трассировки

Алгоритм	Реализация	Быстродействие	Затратность ресурсов	Сложность
Лучевой Алгоритм	Простая	Низкое	Высокая	$O(n^2)$
Алгоритм Рабина	Средняя	Высокое	Средняя	$O(n^2)$
Алгоритм слежения за целью	Средняя	Высокое	Низкая	$O(n^2)$
mod 3	Простая	Среднее	Низкая	$O(n^2)$
Алгоритм Акерса	Простая	Высокое	Низкая	$O(n^2)$

3. Описание разрабатываемого алгоритма, его укрупненная схема

Укрупненная схема алгоритма представлена на рисунке 3.1

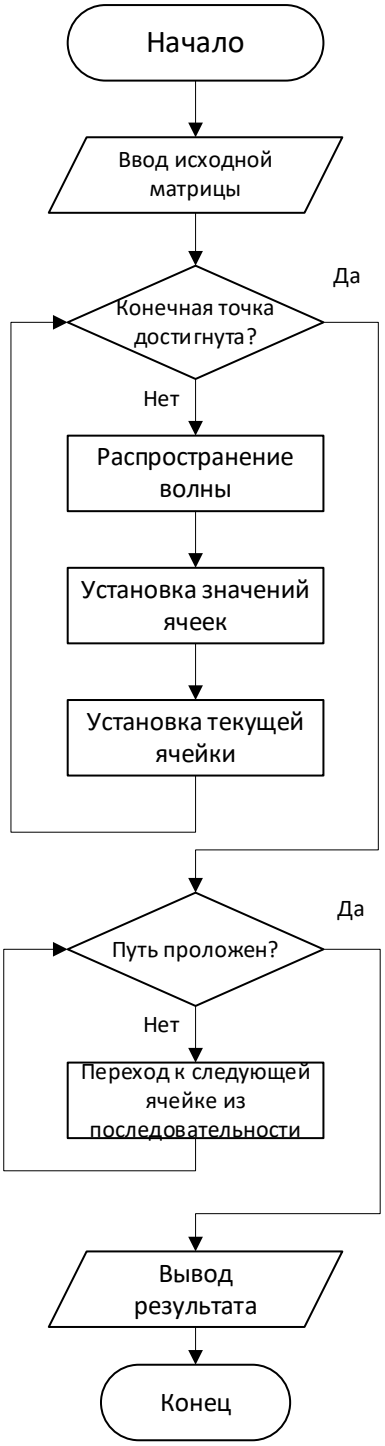

Рисунок 3.1 – «Укрупненная блок-схема алгоритма»

4. Развернутая блок-схема алгоритма.

Развернутая блок-схема алгоритма на языке программирования C# представлена на рисунках 4.1 - 4.8.

Функция BuildingField строит пустое поле (рисунок 4.1).

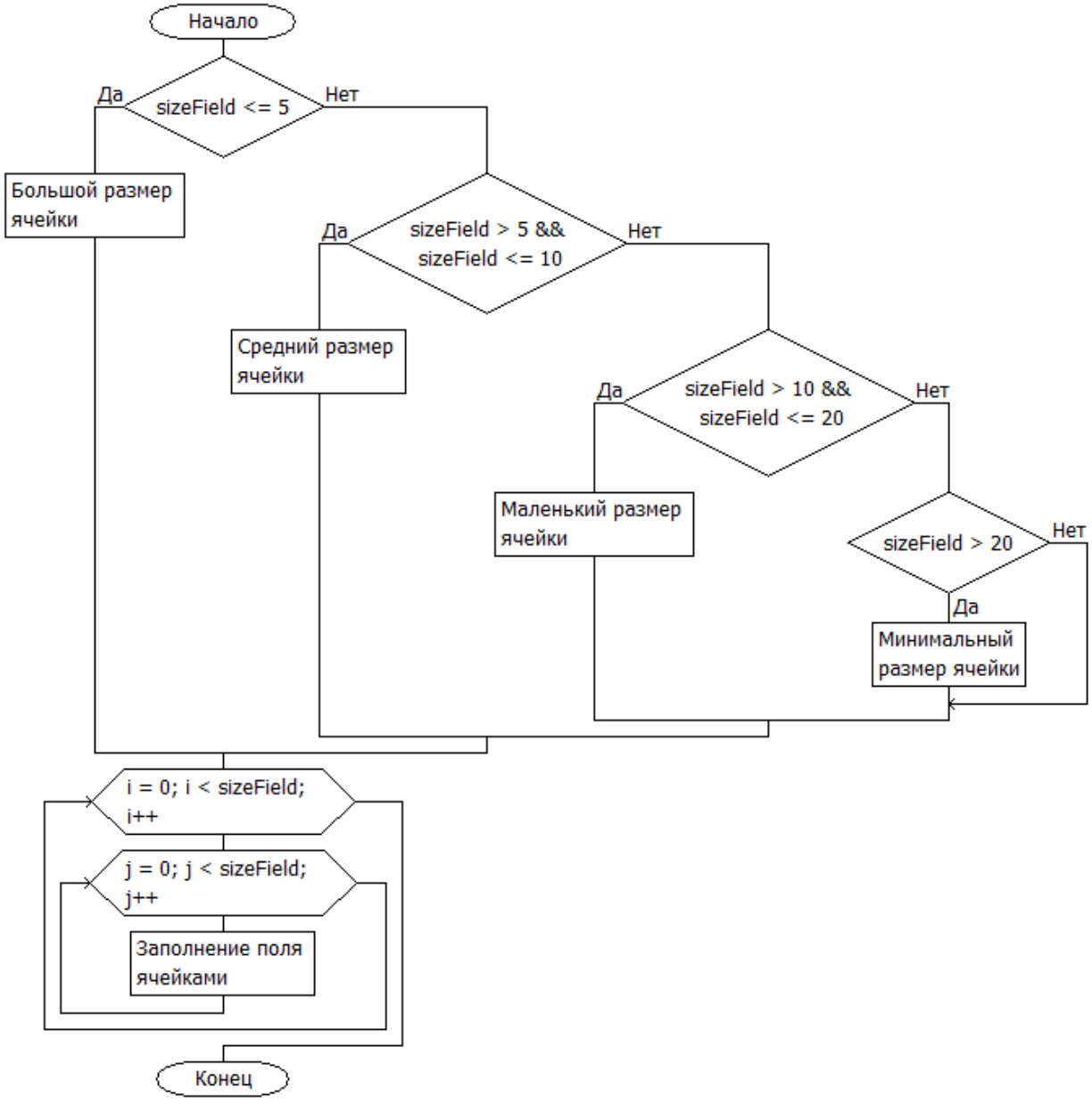

Рисунок 4.1 – «Блок-схема функции BuildingField»

Функция BuildingWindow задает необходимые параметры размера окна приложения (рисунок 4.2).

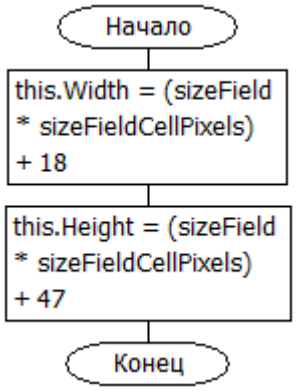

Рисунок 4.2 – «Блок-схема функции BuildingWindow»

Функция CheckCell (рисунок 4.3) проверяет возможность движения в следующую ячейку при распространении волны и восстановлении пути.

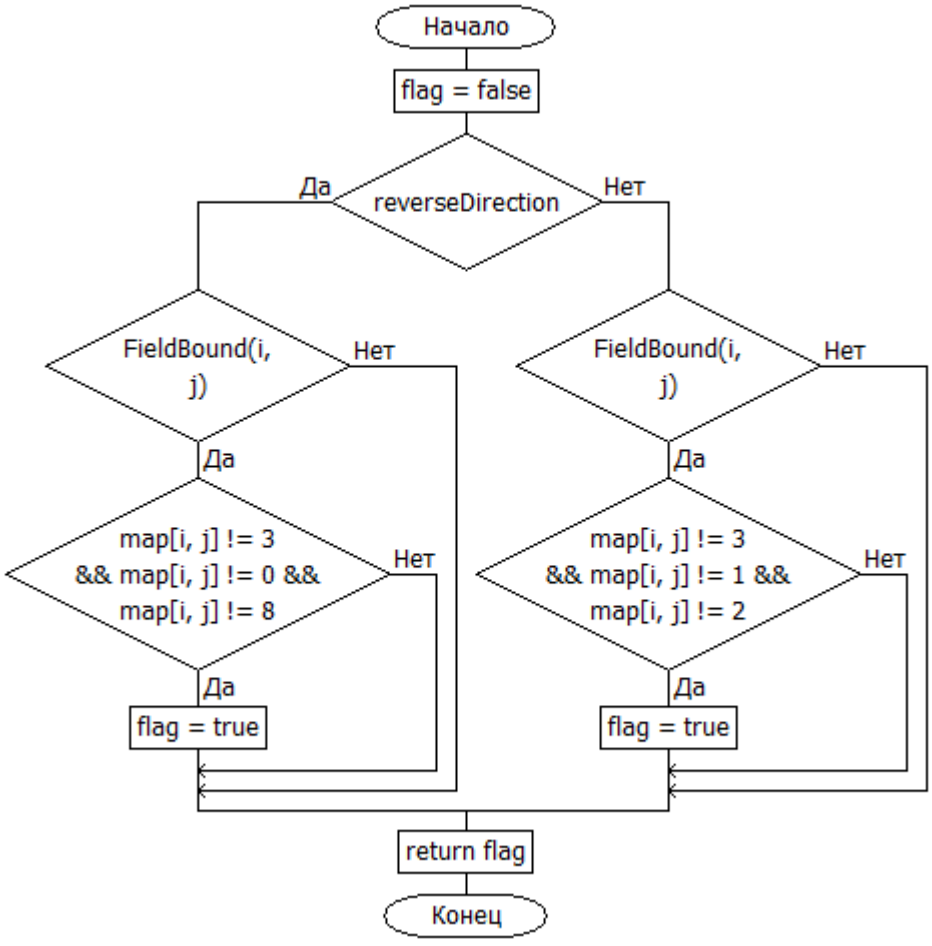

Рисунок 4.3 – «Блок-схема метода CheckCell»

Функция FieldBound проверяет не выходит ли рассматриваемая ячейка за границы поля (рисунок 4.4).

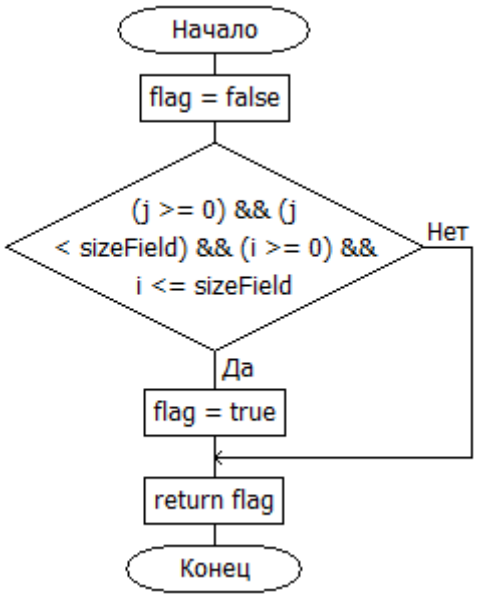

Рисунок 4.4 – «Блок-схема функции FieldBound»

Функция ProcessCmdKey отслеживает нажатие клавиши для начала работы алгоритма. (рисунок 4.5).

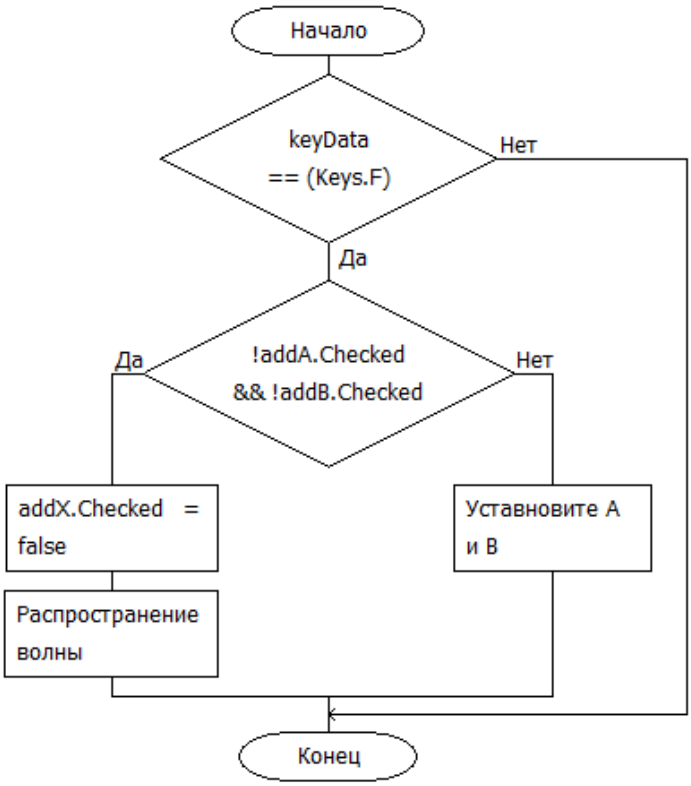

Рисунок 4.5 – «Блок-схема функции ProcessCmdKey»

Функция MarkingField отслеживает нажатие пользователем на ячейку и устанавливает соответствующие значения в ячейки рисунок (4.6).

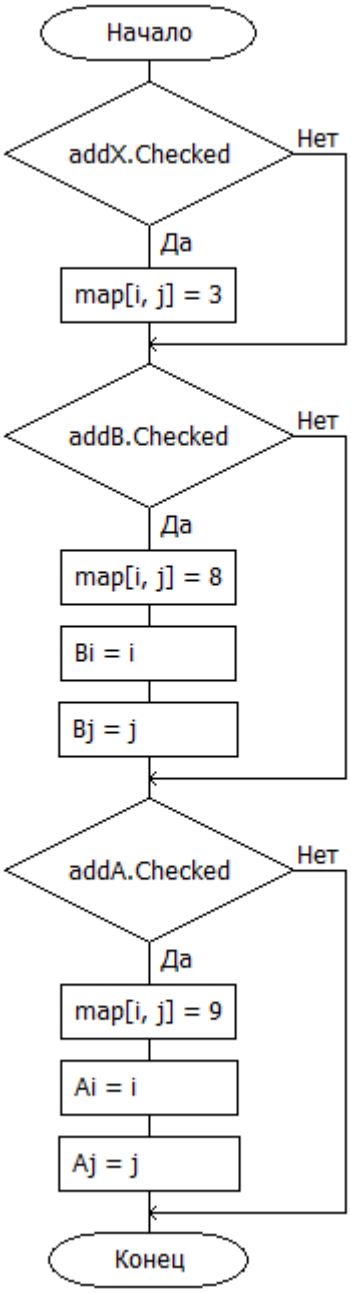

Рисунок 4.6 – «Блок-схема функции MarkingField»

Функция RestoringPath восстанавливает пусть рисунок 4.7

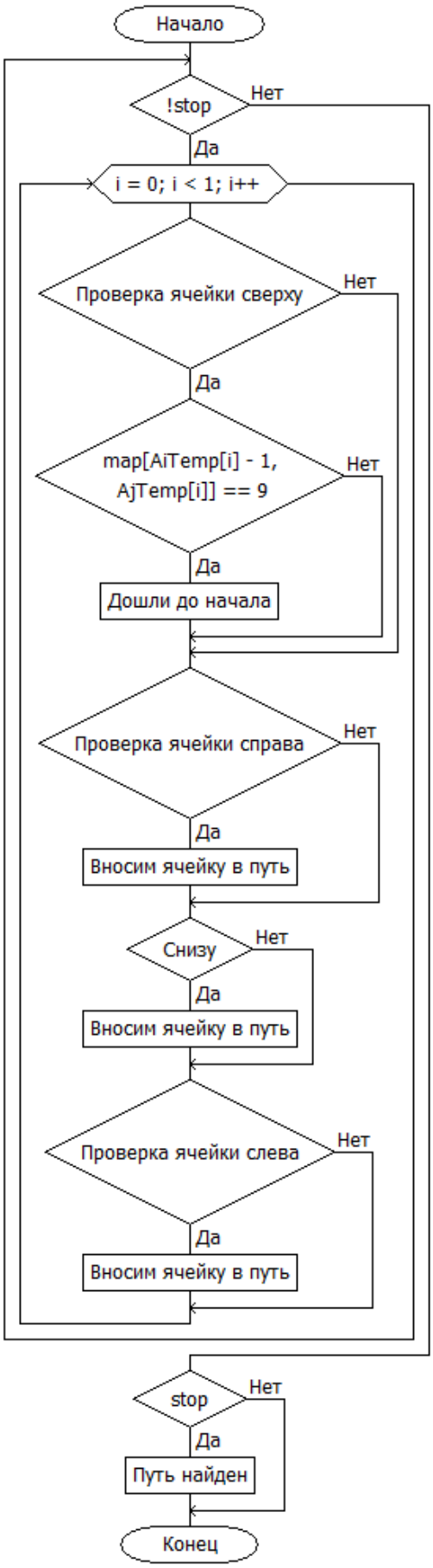

Рисунок 4.7 – «Блок-схема функции RestoringPath»

Функция WavePropagation распространяет волну (рисунок 4.8)

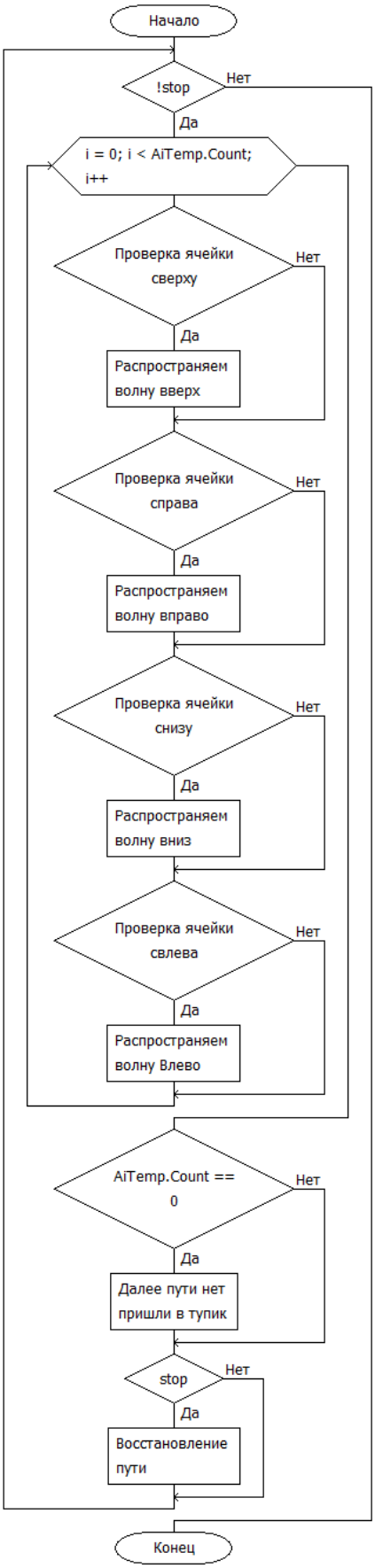

Рисунок 4.8 – «Блок-схема функции WavePropagation»

5 Руководство пользователя

5.1 Введение

5.1.1 Область применения

Программа предназначена для построения оптимального пути из начальной точки в конечную в ДПР размерностью n на n.

5.1.2 Краткое описание возможностей

Приложение позволяет выполнять следующие действия:

- Добавлять начальную и конечную ячейку маршрута
- Добавлять запрещенные для маршрута ячейки
- Просматривать распространение волны
- Оценивать кратчайший путь и количество пройденных ячеек
- Сохранять полученные результаты в текстовый файл и повторно загружать их в программу

5.1.3 Уровень подготовки пользователя

Для работы с приложением пользователь должен владеть клавиатурой и компьютерной мышью.

5.2 Назначение и условие применения

5.2.1 Виды деятельности, функции, программные и аппаратные требования к системе.

Приложение предназначено для автоматизации получения кратчайшего пути из начальной точки ДПР в конечную.

Для запуска приложения необходим персональный компьютер с 64-х разрядный процессором, имеющий менее 1Гб ОЗУ, а также быть под управлением ОС Windows не ниже Windows 7.

5.3 Подготовка к работе

5.3.1 Состав дистрибутива

В состав дистрибутива входит клиентская часть приложения Windows.

5.3.2 Запуск системы

1. Для запуска приложения, откройте папку, в которой находится программа, и запустите файл .exe.

5.3.3 Проверка работоспособности

Приложение работает корректно, если в результате действий пользователя, изложенных в п.п.4.3.2, на экране отобразилось главное окно клиентского приложения без выдачи пользователю сообщений о сбоях.

5.4 Описание операций

1. Отрисовка поля

   - Условия выполнения операции следующие: приложение запущено, успешно функционирует, не выполняется операций блокирующих доступ к основному меню.

   - Подготовительные действия отсутствуют.

   - Основные действия: нажать на кнопку *построить поле*

2. Поставить начальную ячейку

   - Условия выполнения операции следующие: приложение запущено, успешно функционирует, не выполняется операций блокирующих доступ к основному меню.

   - Подготовительные действия отсутствуют.

   - Основные действия: нажать на любую свободную ячейку.

   - Заключительные действия не требуются.

3. Поставить конечную ячейку

   - Условия выполнения операции следующие: приложение запущено, успешно функционирует, не выполняется операций блокирующих доступ к основному меню.

   - Подготовительные действия отсутствуют.

   - Основные действия: нажать на любую свободную ячейку.

   - Заключительные действия не требуются.

4. Поставить запрещенные ячейки

   – Условия выполнения операции следующие: приложение запущено, успешно функционирует, не выполняется операций блокирующих доступ к основному меню.

   – Подготовительные действия отсутствуют.

   – Основные действия: нажать на любую свободную ячейку.

   – Заключительные действия не требуются.

5. Задать начальную вершину и рассчитать

   – Условия выполнения операции следующие: приложение запущено, успешно функционирует, не выполняется операций блокирующих доступ к основному меню, нажата клавиша F.

   – Подготовительные действия отсутствуют.

   – Основные действия: нажать клавишу F

   – Заключительные действия: открытое окно с результатом.

5.5 Аварийные ситуации

При неверных действиях пользователей, неверных или недопустимых входных значениях, приложение выдает пользователю соответствующие сообщения и возвращается в рабочее состояние.

6. Решение контрольного примера

6.1 Пример 1

Этот пример будет состоять из матрицы размерностью 5x5. На рисунке 6.1 показаны исходные данные.

2	1			
2↑	2↑		B	
1↑	2↑		1↑	
1↑→	1↑→	2→	2↑→	1
A ↑→	1 ↑→	1→	2↑→	2

Рисунок 6.1 – «Контрольный пример 1»

Длина пути = 7

На рисунке 6.1 показан результат работы программы.

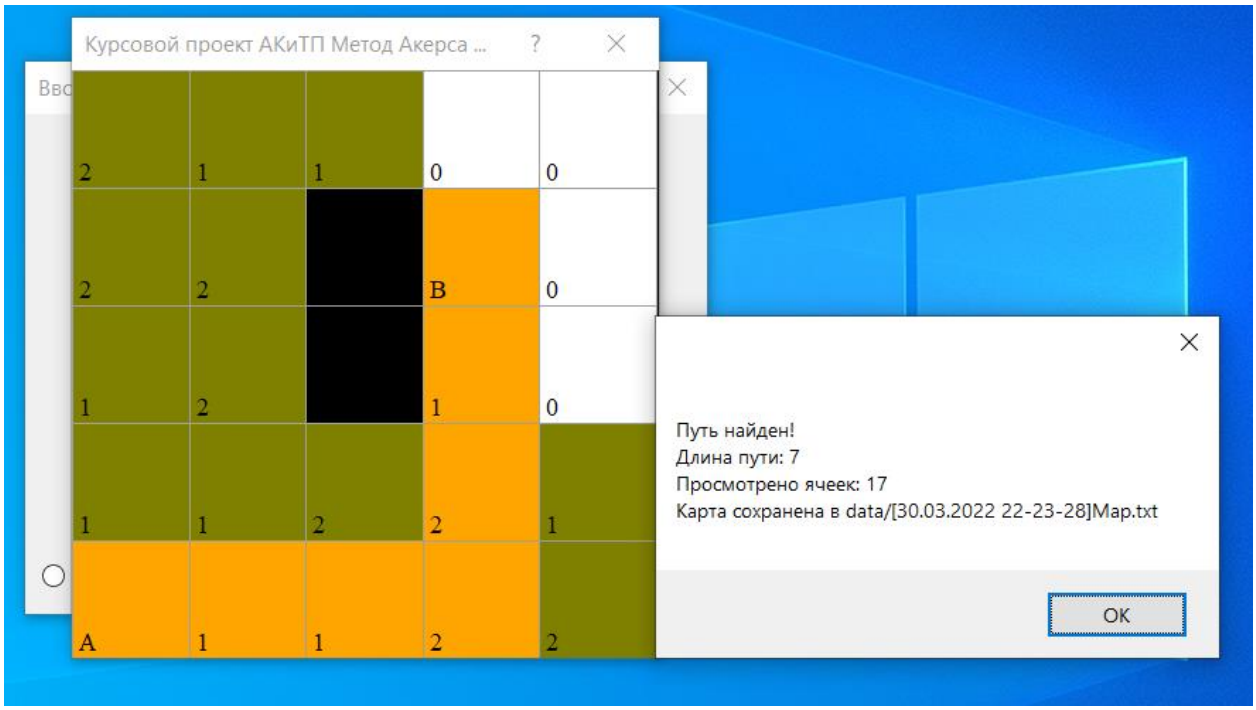

Рисунок 6.2 – «Результат работы программы»

Как видно из результатов решения контрольного примера вручную и с помощью программы, результаты сходятся. Следовательно, можно сделать вывод – программа работает верно.

7. Перечень идентификаторов, используемых при решении программы

string sizeFieldCell; // Размерность ячейки поля

const int HEIGHT_CELLS = 23; Исходная(минимальная) высота ячейки поля

const int CELL_HEIGHT_INC = 20; // Прирощение высоты ячейки поля

int sizeFieldCellPixels; // Размерность ячейки в пикселях

public static int sizeField; // Размер поля, задает пользователь

public int Ai, Aj, Bi, Bj; // Соответствующие координаты точки A(начала) и B(конца)

public int[,] map = new int[1000, 1000]; // Карта поля

private List<int> AiTemp = new(); // Список временных координат строк

private List<int> AjTemp = new();// Список временных координат столбцов

private List<int> BiTemp = new(); // Список временных координат строк

private List<int> BjTemp = new(); // Список временных координат столбцов

readonly int[] posl = { 1, 1, 2, 2 }; // Последовательность распространения волны

int totalCellsViewed = 0; // Всего просмотренных ячеек

int pathLength = 1; // Длина пути

Заключение

В ходе выполнения курсового проекта был изучен принцип работы алгоритма Акерса для трассировки печатных плат и написана программа на языке программирования C# для реализации этого алгоритма. Помимо этого, были получены новые навыки программирования на C# и использования технологии .NET.

Список литературы

1. Сапрыкин А.Н. Алгоритмические методы автоматизации конструирования электронных средств: учебное пособие – Рязань: ИП Коняхин А.В. (Book Jet), 2021. – 116 с.

2. Горин В.С. Исследование алгоритмов трассировки печатных соединений. Методические указания к лабораторным работам. РГРТУ 2006 – 16с.

3. Селютин В. А. Машинное конструирование электронных устройств. - М.: «Сов. радио», 1977. - 384с.

Приложение (Листинг программного кода)

```
namespace KursShmakovAKITP
{
    public partial class Form1 : Form
    {
        string sizeFieldCell; // Размерность ячейки поля
        const int HEIGHT_CELLS = 23; // Исходная(минимальная) высота ячейки поля
        const int CELL_HEIGHT_INC = 20; // Прирощение высоты ячейки поля
        int sizeFieldCellPixels; // Размерность ячейки в пикселях
        public static int sizeField; // Размер поля, задает пользователь
        public int Ai, Aj, Bi, Bj; // Соответствующие координаты точки A(начала) и
B(конца)
        public int[,] map = new int[1000, 1000]; // Карта поля
        private List<int> AiTemp = new(); // Список временных координат строк
        private List<int> AjTemp = new();// Список временных координат столбцов
        private List<int> BiTemp = new(); // Список временных координат строк
        private List<int> BjTemp = new(); // Список временных координат столбцов
        readonly int[] posl = { 1, 1, 2, 2 }; // Последовательность распространения
волны
        int totalCellsViewed = 0; // Всего просмотренных ячеек
        int pathLength = 1; // Длина пути

        // Элементы управления для выбора размещения требуемого элемента на поле
        RadioButton addA = new RadioButton();
        RadioButton addB = new RadioButton();
        RadioButton addX = new RadioButton();

        /*
         * Задание параметров окна в зависимости от количества ячеек
        */
        private void BuildingWindow() {
            // Установка размера программы
            this.Width = (sizeField * sizeFieldCellPixels) + 18;
            this.Height = (sizeField * sizeFieldCellPixels) + 47;
        }

        /*
         * Построение поля
       */
        private void BuildingField() {
            // Настройки поля
            Field.RowCount = sizeField;
```

```
            Field.ColumnCount = sizeField;
            Field.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCells;
            Field.DefaultCellStyle.WrapMode = DataGridViewTriState.True;

            // Выбор размера ячейки в зависимости от их количества
            if(sizeField <= 5){
                sizeFieldCell = "\n\n\n";
            }
            else if (sizeField > 5 && sizeField <= 10) {
                sizeFieldCell = "\n\n";
            }
            else if (sizeField > 10 && sizeField <= 20) {
                sizeFieldCell = "\n";
            }
            else if(sizeField > 20) {
                sizeFieldCell = "";
            }

            // Рассчитываем значение размера ячеек
            sizeFieldCellPixels    =    (HEIGHT_CELLS    +    CELL_HEIGHT_INC    *
sizeFieldCell.Length);

            // Заполняем поле пустыми ячейками
            for (int i = 0; i < sizeField; i++)
                for (int j = 0; j < sizeField; j++)
                    Field.Rows[i].Cells[j].Value  = sizeFieldCell;
        }
        public Form1()
        {
            InitializeComponent();
            BuildingField();
            BuildingWindow();
            Field.DefaultCellStyle.SelectionBackColor = Color.White;
        }

        /*
         * Обработка событий нажатия на клавиши для выбора действия
         */
        protected override bool ProcessCmdKey(ref Message msg, Keys keyData)
        {
            if (keyData == (Keys.F))
            {
                if(!addA.Checked && !addB.Checked)
```

```
                {
                    addX.Checked = false;
                    Field.DefaultCellStyle.SelectionBackColor = Color.Transparent;
                    WavePropagation();

                } else
                {
                    MessageBox.Show("Установите начальную точку (A) и конечную точку
(B)!");
                }
                return true;
            }
            return base.ProcessCmdKey(ref msg, keyData);
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Разрешаем ставить начальную ячейку самой первой
            addA.Checked = true;
        }

        /*
         * Событие нажатия мышкой на ячейку
         */
        private void MarkingField(object sender, DataGridViewCellEventArgs e)
        {
            int i = e.RowIndex;
            int j = e.ColumnIndex;

            if (addX.Checked)
            {
                Field.DefaultCellStyle.SelectionBackColor = Color.Black;
                Field.Rows[i].Cells[j].Style.BackColor = Color.Black;
                Field.Rows[i].Cells[j].Selected = false;
                map[i, j] = 3;
            }
            if (addB.Checked)
            {
                Field.DefaultCellStyle.SelectionBackColor = Color.Red;
                Field.Rows[i].Cells[j].Style.BackColor = Color.Red;
                Field.Rows[i].Cells[j].Selected = false;
                Field.DefaultCellStyle.SelectionBackColor = Color.Black;
                addB.Checked = false;
```

```
                    addX.Checked = true;
                    map[i, j] = 8;
                    Bi = i;
                    Bj = j;

                }
                if (addA.Checked)
                {
                    Field.Rows[i].Cells[j].Style.BackColor = Color.Green;
                    Field.DefaultCellStyle.SelectionBackColor = Color.Green;
                    Field.Rows[i].Cells[j].Selected = false;
                    Field.DefaultCellStyle.SelectionBackColor = Color.Red;
                    addA.Checked = false;
                    addB.Checked = true;
                    map[i, j] = 9;
                    Ai = i;
                    Aj = j;
                }
            }

            /*
             * Распространение волны
             * Волна распространяется по 2 шага, 11,22...
             */
            public void WavePropagation()
            {
                int waveSequenceCounter = 0; // Счетчик последовательности
распространения волны
                bool stop = false; // Флаг остановки распространения волны
                // Начальные координаты распространения волны ячейки A
                AiTemp.Add(Ai);
                AjTemp.Add(Aj);

                // Пока не распространим волну до конца или не дойдем до тупика
                while (!stop)
                {
                    // Распространение одной волны
                    for (int i = 0; i < AiTemp.Count; i++)
                    {
                        // Шаг Вверх
                        if (CheckCell(AiTemp[i] - 1, AjTemp[i], false))
                        {
                            // Если подошли к финишу заканчиваем распространение
```

```
                    if (map[AiTemp[i] - 1, AjTemp[i]] == 8)
                    {
                        stop = true;
                        break;
                    }
                    // Присваиваем ячейке номер в соответствии с последовательностью
                    map[AiTemp[i] - 1, AjTemp[i]] = posl[waveSequenceCounter];
                    // Добавляем пройденную ячейку во временный список, для распространения от нее волны
                    // на следующем шаге
                    BiTemp.Add(AiTemp[i] - 1);
                    BjTemp.Add(AjTemp[i]);
                    // Подсчитываем количество пройденных ячеек
                    totalCellsViewed++;
                }
                // Шаг Вправо
                if (CheckCell(AiTemp[i], AjTemp[i] + 1, false))
                {
                    if (map[AiTemp[i], AjTemp[i] + 1] == 8)
                    {
                        stop = true;
                        break;
                    }
                    map[AiTemp[i], AjTemp[i] + 1] = posl[waveSequenceCounter];
                    BiTemp.Add(AiTemp[i]);
                    BjTemp.Add(AjTemp[i] + 1);
                    totalCellsViewed++;
                }
                // Шаг Вниз
                if (CheckCell(AiTemp[i] + 1, AjTemp[i], false))
                {
                    if (map[AiTemp[i] + 1, AjTemp[i]] == 8)
                    {
                        stop = true;
                        break;
                    }
                    map[AiTemp[i] + 1, AjTemp[i]] = posl[waveSequenceCounter];
                    BiTemp.Add(AiTemp[i] + 1);
                    BjTemp.Add(AjTemp[i]);
                    totalCellsViewed++;
                }
                // Шаг Влево
```

```
                        if (CheckCell(AiTemp[i], AjTemp[i] - 1, false))
                        {
                            if (map[AiTemp[i], AjTemp[i] - 1] == 8)
                            {
                                stop = true;
                                break;
                            }
                            map[AiTemp[i], AjTemp[i] - 1] = posl[waveSequenceCounter];
                            BiTemp.Add(AiTemp[i]);
                            BjTemp.Add(AjTemp[i] - 1);
                            totalCellsViewed++;
                        }
                    }
                    // Если шаги некуда больше сделать, значит попали в тупик
                    if (AiTemp.Count == 0) {
                        MessageBox.Show("Тупик!");
                        break;
                    }
                    //  Меняем списки временных координат распространения волны пройденной, на новую
                    FlippingLists(AiTemp, AjTemp, BiTemp, BjTemp);

                    // Если успешно распространили волну и дошли до финишной ячейки запускаем функцию восстановления пути
                    if (stop)
                    {
                        AiTemp.Clear();
                        AjTemp.Clear();
                        RestoringPath(waveSequenceCounter);
                    }

                    waveSequenceCounter++;    //    Увеличиваем    значение    индекса последовательности распространения волны
                    // Если индекс = 4, значит одну волну распространили и начинаем заного
                    if (waveSequenceCounter == 4) {waveSequenceCounter = 0;}
                }
            }

            /*
             * Функция смены временных списков координат
             */
```

```
            public void FlippingLists(List<int> AiTemp, List<int> AjTemp, List<int>
BiTemp, List<int> BjTemp)
            {
                AiTemp.Clear();
                AjTemp.Clear();
                AiTemp.AddRange(BiTemp);
                AjTemp.AddRange(BjTemp);
                BiTemp.Clear();
                BjTemp.Clear();
            }

            /*
             * Функция восстановления пути в качестве параметра принимает значение
счетчика распространения волны
             * на котором остановилась последняя волна
             */
            public void RestoringPath(int waveSequenceCounter)
            {
                // Начальные координаты восстановления пути от финишной ячейки B
                AiTemp.Add(Bi);
                AjTemp.Add(Bj);
                bool stop = false; // Флаг остановки восстановления пути

                // Если волна закончила распространяться на первой единице [1]122, то
предыдущая ячейка начинается с [2]
                if (waveSequenceCounter == 0)
                {
                    waveSequenceCounter = 2;
                }
                // Если волна закончила распространяться на первой двойке 11[2]2, то
предыдущая ячейка начинается с [1]
                else if (waveSequenceCounter == 2)
                {
                    waveSequenceCounter = 0;
                }

                // Заполняем поле значениями в соответствии с ячейками
                // 0 - пустая 1 - первый фронт волны 2 - второй фронт волны 3 -
препядствие
                // 8 - Финиш 9 - Начало
                for (int i = 0; i < sizeField; ++i)
                {
                    for (int j = 0; j < sizeField; ++j)
```

```
        {
            Field.Rows[i].Cells[j].Value = sizeFieldCell + map[i, j];

            if (map[i, j] == 1)
            {
                Field.Rows[i].Cells[j].Style.BackColor = Color.Olive;
            }
            if (map[i, j] == 2)
            {
                Field.Rows[i].Cells[j].Style.BackColor = Color.Olive;
            }
            if (map[i, j] == 3)
            {
                Field.Rows[i].Cells[j].Style.BackColor = Color.Black;
            }
            if (map[i, j] == 8)
            {
                Field.Rows[i].Cells[j].Selected = false;
                Field.Rows[i].Cells[j].Value = sizeFieldCell + "B";
                Field.Rows[i].Cells[j].Style.BackColor = Color.Orange;
            }
            if (map[i, j] == 9)
            {
                Field.Rows[i].Cells[j].Value = sizeFieldCell + "A";
                Field.Rows[i].Cells[j].Style.BackColor = Color.Orange;
            }
        }
    }
    // Пока не дошли до начала
    while (!stop)
    {
        pathLength++; // Подсчитываем путь

        for (int i = 0; i < 1; i++)
        {
            // Проверяем ячейку сверху
            if (CheckCell(AiTemp[i] - 1, AjTemp[i], true))
            {
                // Уже дошли до начала останавливаемся
                if (map[AiTemp[i] - 1, AjTemp[i]] == 9)
                {
                    stop = true;
                    break;
```

```
                            }
                            // Сравниваем просматриваемую ячейку с заданной
последовательностью
                            if (map[AiTemp[i] - 1, AjTemp[i]] ==
posl[waveSequenceCounter])
                            {
                                Field.Rows[AiTemp[i] -
1].Cells[AjTemp[i]].Style.BackColor = Color.Orange;
                                BiTemp.Add(AiTemp[i] - 1);
                                BjTemp.Add(AjTemp[i]);

                                FlippingLists(AiTemp, AjTemp, BiTemp, BjTemp);
                                waveSequenceCounter++;    // Прибавляем счетчик
последовательности

                                // Счетчик последовательности в начало если всю прошли
                                if (waveSequenceCounter == 4)
                                {
                                    waveSequenceCounter = 0;
                                }
                                break;
                            }
                        }

                        // Проверяем ячейку справа
                        if (CheckCell(AiTemp[i], AjTemp[i] + 1, true))
                        {
                            if (map[AiTemp[i], AjTemp[i] + 1] == 9)
                            {
                                stop = true;
                                break;
                            }

                            if (map[AiTemp[i], AjTemp[i] + 1] ==
posl[waveSequenceCounter])
                            {
                                Field.Rows[AiTemp[i]].Cells[AjTemp[i] +
1].Style.BackColor = Color.Orange;
                                BiTemp.Add(AiTemp[i]);
                                BjTemp.Add(AjTemp[i] + 1);

                                FlippingLists(AiTemp, AjTemp, BiTemp, BjTemp);
                                waveSequenceCounter++;
```

```
                        if (waveSequenceCounter == 4)
                        {
                            waveSequenceCounter = 0;
                        }
                        break;
                    }
                }

                // Проверяем ячейку снизу
                if (CheckCell(AiTemp[i] + 1, AjTemp[i], true))
                {
                    if (map[AiTemp[i] + 1, AjTemp[i]] == 9)
                    {
                        stop = true;
                        break;
                    }

                    if (map[AiTemp[i] + 1, AjTemp[i]] == posl[waveSequenceCounter])
                    {
                        Field.Rows[AiTemp[i] + 1].Cells[AjTemp[i]].Style.BackColor = Color.Orange;
                        BiTemp.Add(AiTemp[i] + 1);
                        BjTemp.Add(AjTemp[i]);

                        FlippingLists(AiTemp, AjTemp, BiTemp, BjTemp);
                        waveSequenceCounter++;

                        if (waveSequenceCounter == 4)
                        {
                            waveSequenceCounter = 0;
                        }
                        break;
                    }
                }

                // Проверяем ячейку Слева
                if (CheckCell(AiTemp[i], AjTemp[i] - 1, true))
                {
                    if (map[AiTemp[i], AjTemp[i] - 1] == 9)
                    {
                        stop = true;
```

```
                                break;
                            }

                            if      (map[AiTemp[i],      AjTemp[i]      -      1]      ==
posl[waveSequenceCounter])
                            {
                                Field.Rows[AiTemp[i]].Cells[AjTemp[i]                -
1].Style.BackColor = Color.Orange;
                                BiTemp.Add(AiTemp[i]);
                                BjTemp.Add(AjTemp[i] - 1);

                                FlippingLists(AiTemp, AjTemp, BiTemp, BjTemp);
                                waveSequenceCounter++;

                                if (waveSequenceCounter == 4)
                                {
                                    waveSequenceCounter = 0;
                                }
                                break;
                            }
                        }
                    }
                }

                if(stop)
                {
                    MessageBox.Show("Путь найден!\nДлина пути: " + pathLength +
                        "\nПросмотрено ячеек: " + totalCellsViewed);
                }
            }

            private         void         Form1_HelpButtonClicked(object         sender,
System.ComponentModel.CancelEventArgs e)
            {
                MessageBox.Show("Для распространения волны нажмите F.\n" +
                    "Для построение нового поля закройте окно.");
            }

            /*
             * Функция проверки не выходит ли рассматриваемая ячейка за границы поля
             */
            public static bool FieldBound(int i, int j)
            {
```

```
            bool flag = false;
            if (j >= 0 && j < sizeField & i >= 0 && i < sizeField)
            {
                flag = true;
            }

            return flag;
        }
        /*
         * Функция проверки на возможность движения в следующую ячейку при распространении волны
         * и восстановлении пути (reverseDirection true).
         */
        public bool CheckCell(int i, int j, bool reverseDirection)
        {
            bool flag = false;

            if (reverseDirection)
            {
                if (FieldBound(i, j))
                {
                    // При восстановлении пути ячейка не должна быть стеной,пустой и конечной
                    if (map[i, j] != 3 && map[i, j] != 0 && map[i, j] != 8)
                    {
                        flag = true;
                    }
                }
            }
            else
            {
                if (FieldBound(i, j))
                {
                    // При распространении волны ячейка не должна быть стеной,началом и просмотренной
                    if (map[i, j] != 3 && map[i, j] != 9 && map[i, j] != 1
                           && map[i, j] != 2)
                    {
                        flag = true;
                    }
                }
            }
            return flag;
```

```
        }

        public void ZeroingValues()
        {
            totalCellsViewed = 0;
            pathLength = 1;
        }
    }
}
```