Министерство науки и высшего 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 поиска путей в лабиринте”

Выполнил:

Студент группы 22ВВВ1

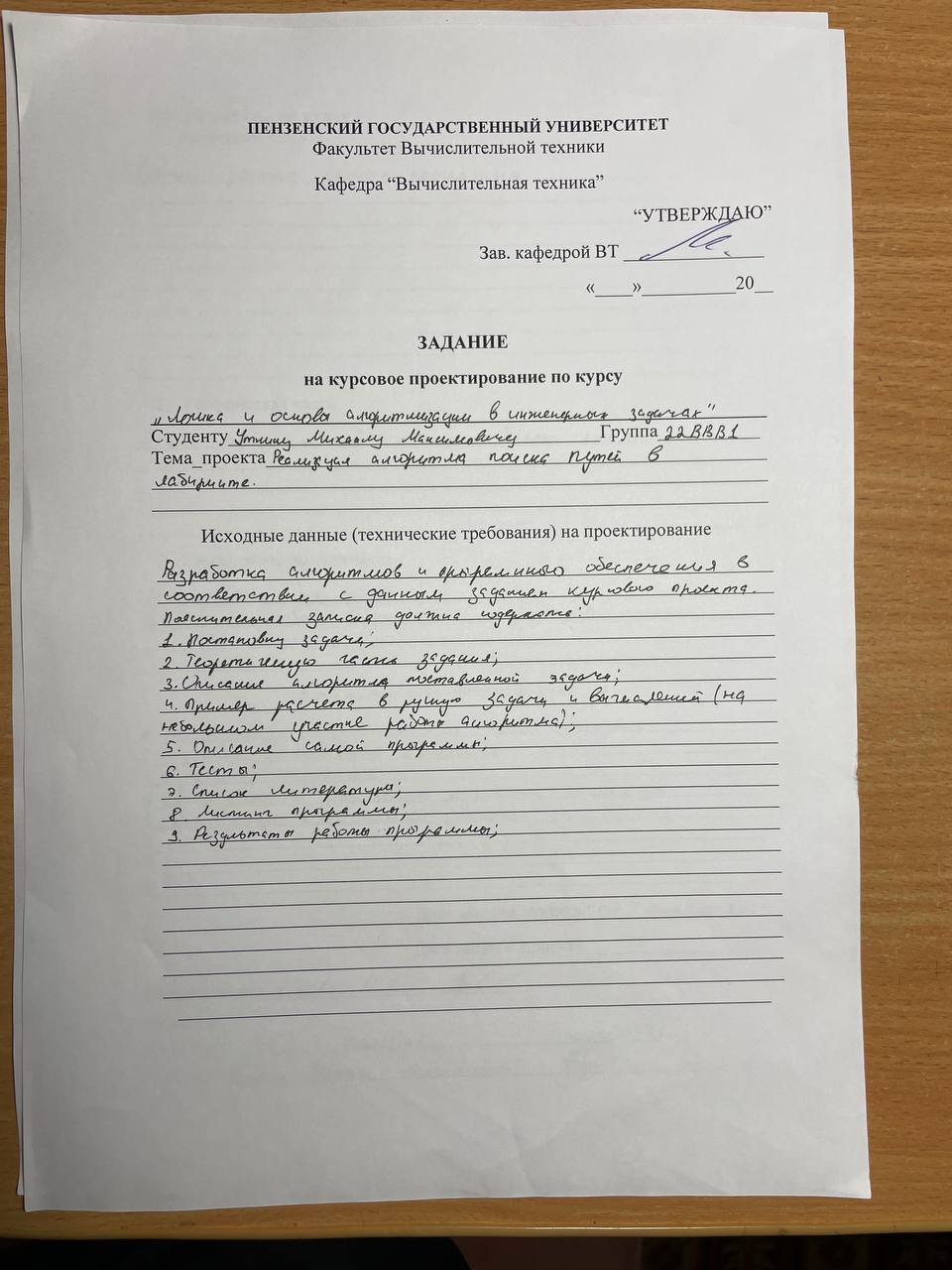
Уткин М.М.

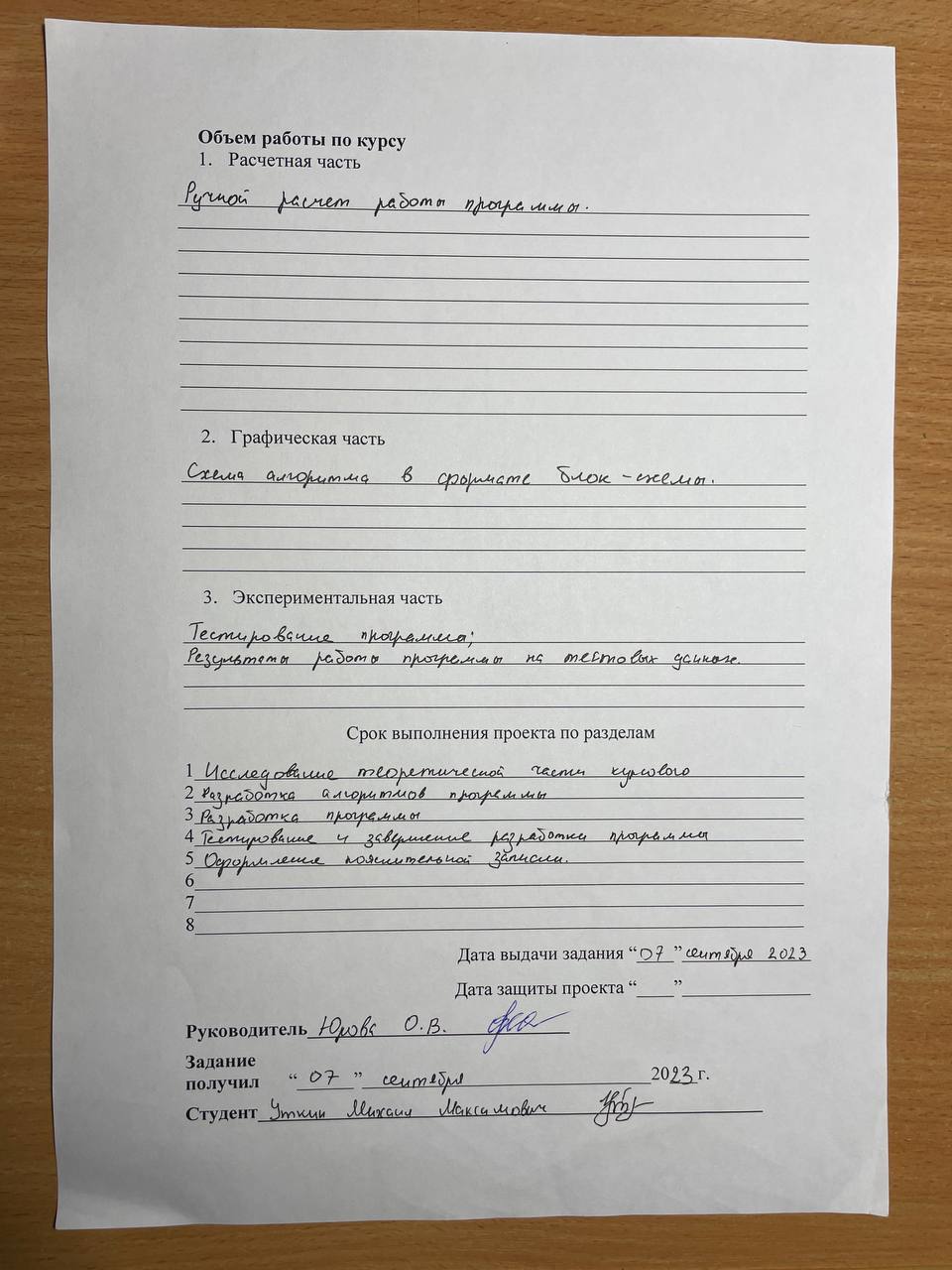
Принял:

К.т.н., доцент

Юрова О.В.

Пенза 2023

****

****

# Содержание

[Содержание 4](#_Toc154432850)

[Реферат 5](#_Toc154432851)

[Введение 6](#_Toc154432852)

[1. Постановка задачи 7](#_Toc154432853)

[2. Теоретическая основа задания 8](#_Toc154432854)

[3. Описание программного алгоритма 9](#_Toc154432855)

[Описание программы 11](#_Toc154432856)

[4. Тестирование 16](#_Toc154432857)

[5. Ручной расчёт задачи 19](#_Toc154432858)

[Заключение 21](#_Toc154432859)

[Список литературы 22](#_Toc154432860)

[Приложение А. 23](#_Toc154432861)

# Реферат

Отчет 27 стр, 9 рисунков.

Цель исследования – разработка программы для поиска путей в лабиринте.

В работе рассмотрена реализация алгоритма ли для поиска путей в лабиринте. Установлено, что с помощью данного алгоритма можно найти кротчайший путь в лабиринте.

**Введение**

Алгоритм волновой трассировки, также известный как волновой алгоритм или алгоритм Ли, представляет собой метод поиска кратчайшего пути на планарном графе. Он относится к алгоритмам, использующим методы поиска в ширину. Для разработки данной программы была выбрана среда Microsoft Visual Studio 2022, а язык программирования - C++.

Целью данной курсовой работы является создание программы на языке C++, который широко применяется в различных областях. Именно с использованием этого языка в данном курсовом проекте будет реализован алгоритм поиска кратчайшего пути в лабиринте.

1. **Постановка задачи**

Цель проекта заключается в разработке программы, способной определить маршрут внутри лабиринта. Исходный лабиринт в программе должен быть представлен матрицей, при этом при генерации данных следует учесть граничные условия. После обработки этой информации на экране должны отображаться сам лабиринт и найденный маршрут. Необходимо предусмотреть разнообразные сценарии поиска, чтобы программа корректно функционировала и не выдавала ошибок.

1. **Теоретическая основа задания**

Алгоритм волновой трассировки функционирует на дискретном рабочем поле (ДРП), представляющем собой замкнутую линию фигуры, не обязательно прямоугольной, разделенную на прямоугольные или квадратные ячейки. Эти ячейки подразделяются на категории: "проходимые" (свободные для прохода) и "непроходимые" (препятствия). Кроме того, определяются стартовая и финишная ячейки, указывающие начальную и конечную точки маршрута.

Цель алгоритма заключается в поиске кратчайшего пути от стартовой ячейки к финишной. В случае отсутствия пути программа должна выдавать соответствующее сообщение о непроходимости.

Работа алгоритма включает три основных этапа: инициализацию, распространение волны и восстановление пути. На этапе инициализации строится образ множества ячеек рабочего поля, каждой из которых присваиваются атрибуты проходимости/непроходимости, а также запоминаются стартовая и финишная ячейки.

Далее, начиная с стартовой ячейки, алгоритм распространяет волну, порождая шаги в соседние ячейки, с учетом их проходимости и отсутствия меток от предыдущих шагов.

В конечном итоге, после завершения распространения волны, происходит восстановление кратчайшего пути от финишной ячейки до стартовой.

1. **Описание программного алгоритма**

Для реализации алгоритма был использован массив map(), предназначенный для хранения лабиринта. С помощью функции mazemake() производится генерация лабиринта.

Исходно массив лабиринта заполняется стенками. Затем, начиная с ячейки (1; 1), создаются проходы. После этого начинается поиск пути в лабиринте от ячейки (1; 1) к ячейке (N-1; N-1), где N - размер лабиринта. Поиск осуществляется путем маркировки каждой ячейки номером и использования вектора для отслеживания координат проходов.

Алгоритм начинает с ячейки (1; 1), помечая ее номером 0. С использованием вектора определяются проходы рядом с текущей ячейкой, и их координаты добавляются в вектор, маркируя их номером на единицу больше предыдущего. Этот процесс повторяется, пока не достигнута ячейка (N-1; N-1). Затем восстанавливается путь от финишной ячейки до стартовой, выводя его на экран.

Ниже представлен псевдокод части программы, ответственной за поиск кратчайшего пути в лабиринте.

1. Записываем координаты ячейки (1; 1) в вектор oldWave.
2. Устанавливаем nstep = 0.
3. Присваиваем ячейке матрицы (1; 1) значение 0.
4. Создаем массив dx с значениями { 0, 1, 0, -1 } и массив dy с значениями { -1, 0, 1, 0 }.
5. Пока вектор oldWave не пуст:
   1. Увеличиваем nstep.
   2. Очищаем вектор wave.
   3. Для каждого элемента i в векторе oldWave:
      1. Для каждого значения d от 0 до 3:
      2. Устанавливаем nx как первую координату вектора + dx[d].
      3. Устанавливаем ny как вторую координату вектора + dy[d].
      4. Если в ячейке (nx, ny) есть проход:
      5. Заносим координаты (nx, ny) в вектор wave.
      6. Присваиваем значение ячейке nstep.
      7. Если nx = N – 2 и ny = N – 2, то выходим из всех циклов.
   4. Присваиваем oldWave = wave.
6. Устанавливаем x = N – 2 и y = N – 2.
7. Очищаем вектор wave.
8. Заносим в вектор wave координаты x, y.
9. Пока значение в ячейке не равно 0:
   1. Для каждого значения d от 0 до 3:
   2. Устанавливаем nx как первую координату вектора + dx[d].
   3. Устанавливаем ny как вторую координату вектора + dy[d].
   4. Если в ячейке (x, y) значение на единицу меньше, чем в ячейке (nx, ny):
   5. Устанавливаем x = nx.
   6. Устанавливаем y = ny.
   7. Заносим координаты (x, y) в вектор wave.
10. Выводим путь на экран.

Полный код программы представлен в Приложении А.

## Описание программы

Для написания данной программы использован язык программирования С++. Язык программирования С++ - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: deadend, endend, mazemake, drawing, main.

Программа выводит на экран сгенерированный лабиринт и его кратчайший путь. Сгенерированный лабиринт сохраняется в файле. Для генерации лабиринта используется функция mazemake(). Сначала она заполняет лабиринт стенками. Затем начинается цикл, в котором с помощью функции ended() проверяется, полностью ли лабиринт заполнен проходами. Если это так, программа выходит из цикла. Если лабиринт не полностью заполнен, продолжается цикл, в котором создаются проходы в лабиринте, выбирая случайное направление. Затем с помощью функции deadend() проверяется, находимся ли мы в тупике, и, если да, выходим из него.

После генерации лабиринта начинается поиск кратчайшего пути в нем. Ячейка (1; 1) помечается номером 0. Затем, используя вектор, находятся проходы рядом с текущей ячейкой, и их номер маркируется на единицу больше предыдущего. Координаты этих проходов заносятся в вектор. Этот процесс повторяется, пока не достигнута ячейка (N-1; N-1). Затем вектор очищается, и начиная с ячейки (N-1; N-1), заносится ячейка, значение которой на единицу меньше, чем у исходной, пока не достигнута ячейка с номером 1. Затем маршрут выводится на экран.

bool deadend(int x, int y, int maze[][99], int height, int width) { //Таким образом, функция deadend позволяет определить, является ли заданная позиция в лабиринте тупиковой.

int a = 0;

if (x != 1) {

if (maze[y][x - 2] == pass)

a += 1;

}

else

a += 1;

if (y != 1) {

if (maze[y - 2][x] == pass)

a += 1;

}

else

a += 1;

if (x != width - 2) {

if (maze[y][x + 2] == pass)

a += 1;

}

else

a += 1;

if (y != height - 2) {

if (maze[y + 2][x] == pass)

a += 1;

}

else

a += 1;

return a == 4;

}

bool ended(int maze[][99], int height, int width) { //Таким образом, функция ended проверяет, завершено ли создание лабиринта, основываясь на том, что все внутренние клетки созданы (не являются стенами).%

bool b = true;

for (int i = 1; i < (height - 1); i += 2)

for (int j = 1; j < (width - 1); j += 2)

if (maze[i][j] == wall)

b = false;

return b;

}

void mazemake(int maze[][99], int height, int width) {

int x, y, c, a;

bool b;

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

for (int j = 0; j < width; j++)

maze[i][j] = wall;

x = 3; y = 3; a = 0;

while (1)

{

a++;

if (a % 100 == 0)

if (ended(maze, height, width))

break;

maze[y][x] = pass;

while (1) {

c = rand() % 4;

switch (c) {

case 0: if (y != 1)

if (maze[y - 2][x] == wall) {

maze[y - 1][x] = pass;

maze[y - 2][x] = pass;

y -= 2;

}

break;

case 1: if (y != height - 2)

if (maze[y + 2][x] == wall) {

maze[y + 1][x] = pass;

maze[y + 2][x] = pass;

y += 2;

}

break;

case 2: if (x != 1)

if (maze[y][x - 2] == wall) {

maze[y][x - 1] = pass;

maze[y][x - 2] = pass;

x -= 2;

}

break;

case 3: if (x != width - 2)

if (maze[y][x + 2] == wall) {

maze[y][x + 1] = pass;

maze[y][x + 2] = pass;

x += 2;

}

break;

}

if (deadend(x, y, maze, height, width))

break;

}

if (deadend(x, y, maze, height, width))

do {

x = 2 \* (rand() % ((width - 1) / 2)) + 1;

y = 2 \* (rand() % ((height - 1) / 2)) + 1;

} while (maze[y][x] != pass);

}

}

Функция генерации лабиринта

map[1][1] = pass;

vector<pair<int, int>> oldWave;

oldWave.push\_back(pair<int, int>(1, 1));

int nstep = 0;

map[1][1] = nstep;

const int dx[] = { 0, 1, 0, -1 };

const int dy[] = { -1, 0, 1, 0 };

while (!oldWave.empty())

{

++nstep;

vector<pair<int, int>> wave;

for (const auto& i : oldWave)

{

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

{

int nx = i.first + dx[d];

int ny = i.second + dy[d];

if (nx >= 0 && nx < map\_size && ny >= 0 && ny < map\_size && map[nx][ny] == pass)

{

wave.push\_back(pair<int, int>(nx, ny));

map[nx][ny] = nstep;

if (nx == map\_size - 2 && ny == map\_size - 2)

{

oldWave.clear();

break;

}

}

}

}

oldWave = wave;

}

int x = map\_size - 2;

int y = map\_size - 2;

vector<pair<int, int>> wave;

wave.push\_back(pair<int, int>(x, y));

while (map[x][y] != 0)

{

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

{

int nx = x + dx[d];

int ny = y + dy[d];

if (nx >= 0 && nx < map\_size && ny >= 0 && ny < map\_size && map[x][y] - 1 == map[nx][ny])

{

x = nx;

y = ny;

wave.push\_back(pair<int, int>(x, y));

}

}

}

for (const auto& i : wave)

{

map[i.first][i.second] = way;

}

drawing(map, map\_size);

getchar(); getchar();

}



Рисунок 1 - Выведенный на экран лабиринт

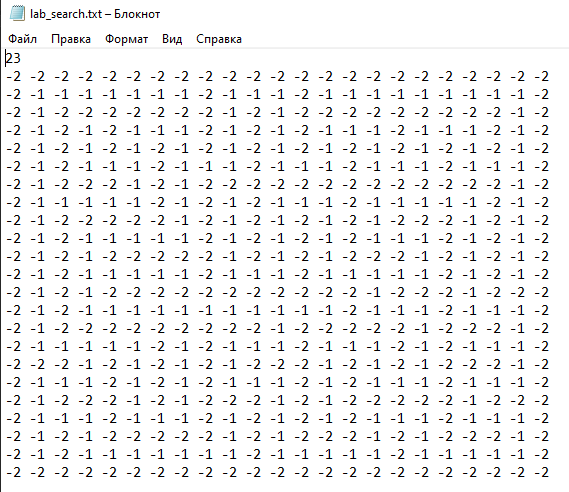


Рисунок 2 - Cохранённый в файле лабиринт

# Тестирование

Среда разработки Microsoft Visual Studio 2022 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций. Чтобы изменить размеры лабиринта мы должны поменять значение глобальной переменной N. Задали размеры лабиринта – N =11.

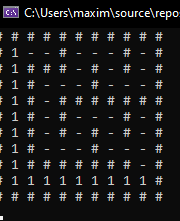


Рисунок 3- Результат тестирования при размерах лабиринта 11x11

На рисунке 3 продемонстрирован результат тестирования при размерах лабиринта 11x11. Наш лабиринт записался в файл.

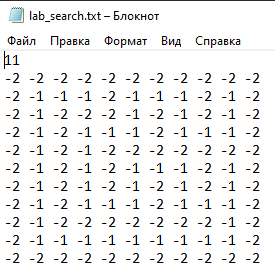


Рисунок 4 - Лабиринт, записанный в файл

Задали размеры лабиринта – N = 23.

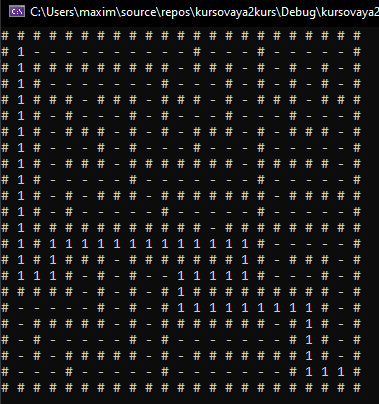


Рисунок 5 - результат тестирования при размерах лабиринта 23x23

На рисунке 5 продемонстрирован результат тестирования при размерах лабиринта 23x23. Наш лабиринт записался в файл.

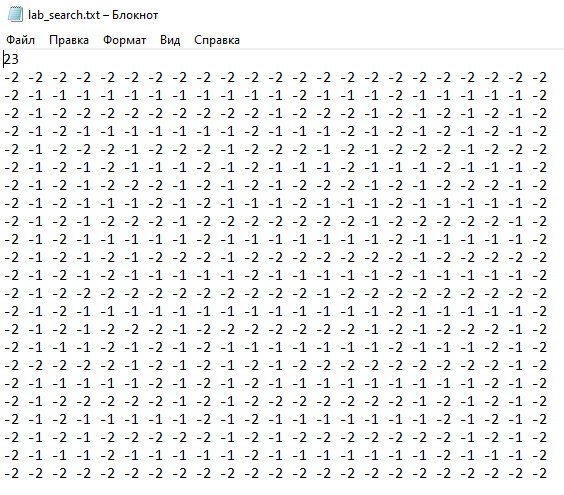


Рисунок 6 - Лабиринт 23х23, записанный в файл

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Запуск окна | + |
| Генерирование лабиринта | Вывод сгенерированного лабиринта на экран | + |
| Поиск пути | Вывод на экран кротчайшего пути в лабиринте | + |

Таблица 1 – Описание поведения программы при тестировании

1. **Ручной расчёт задачи**

Проведем проверку программы посредством ручных вычислений на примере лабиринта с размерами 11x11.

Начинаем из ячейки с координатами (1; 1). Помечаем эту ячейку нулём, проверяем есть ли рядом с ней свободные пути. В нашем случае есть свободный путь вниз и вправо. Помечаем их единицами. Далее просматриваем ячейки помеченные единицами и смотрим, есть ли рядом с ними свободные пути если есть, то помечаем свободные пути двойками и аналогично продолжаем проверять. Пока не дойдём до ячейки (10;10).

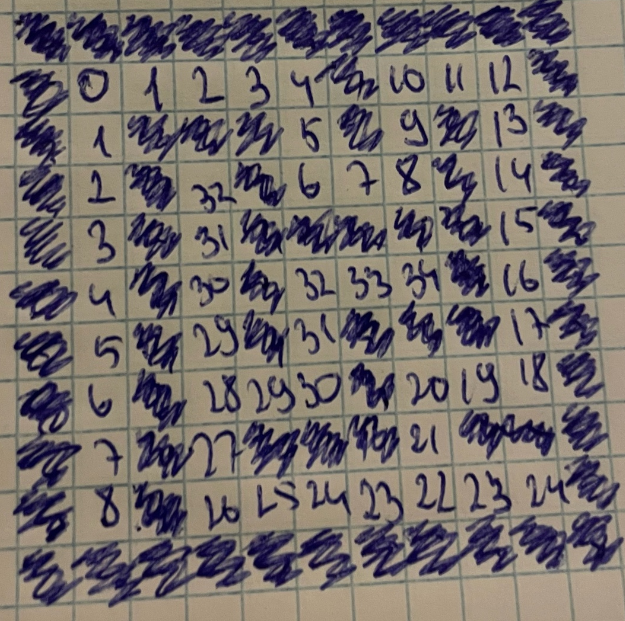


Рисунок 7- ручной расчёт путей в лабиринте

После того, как мы дошли до ячейки (11; 11), мы должны начиная с этой ячейки искать поблизости ячейку со значение на единицу меньше и так пока не дойдём до нуля, это будет наш кротчайший путь.

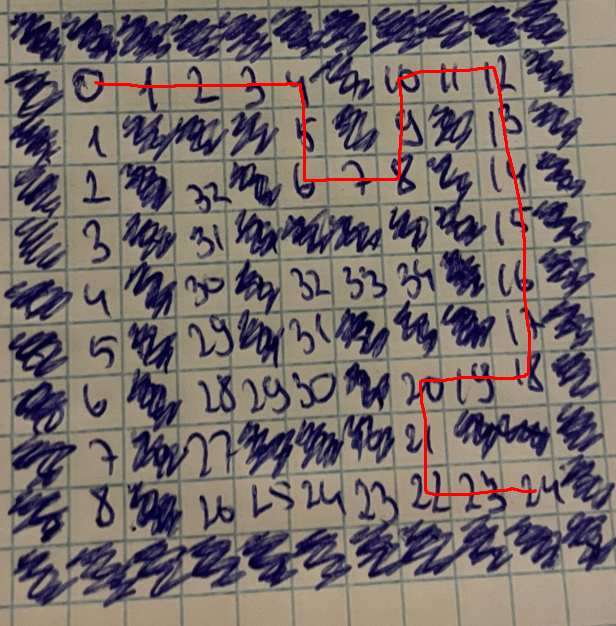


Рисунок 7 - ручной расчёт восстановление пути

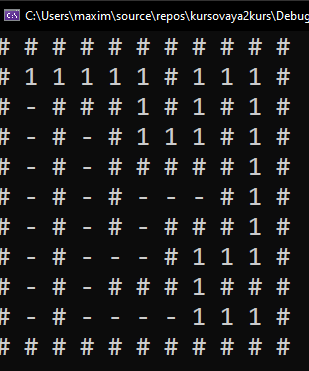


Рисунок 9 - результат работы программы

Сравнив результат работы программ и ручной расчёт, можно сделать вывод, что программа работает верно.

# Заключение

В заключение данного проекта была разработана программа, реализующая алгоритм поиска кратчайшего пути в лабиринте в среде Microsoft Visual Studio 2022 с использованием языка программирования C++. В ходе выполнения курсовой работы были приобретены навыки создания программ и освоены методы разработки алгоритма для генерации лабиринта, а также реализации алгоритма поиска кратчайшего пути.

Курсовая работа позволила углубить знания по языку программирования C++ и освоить особенности работы среды разработки Microsoft Visual Studio 2022. Однако, следует отметить, что недостатком программы является примитивный интерфейс. Это обусловлено отсутствием оконного интерфейса, что могло бы сделать программу более удобной в использовании. Несмотря на это, программа обладает достаточным функционалом для решения поставленных задач.

**Список литературы**

1. Седжвик Р., Уэйн К. "Алгоритмы. Построение и анализ." М.: Вильямс, 2018. - Рекомендуемая глава: "Графы и алгоритмы на графах."

2. Хоровиц Д. "Алгоритмы. Справочник с примерами на C++." Питер, 2017. - Особенно главы, посвященные алгоритмам поиска путей.

3. Кормен Т., Лейзерсон Ч., Ривест Р. "Алгоритмы: Построение и анализ." М.: МЦНМО, 2001. - Главы, связанные с алгоритмами поиска.

4. Шилдт Г. "Полный справочник по C++." Вильямс, 2006. - Разделы, охватывающие структуры данных и алгоритмы на C++.

5. Дейтел Х., Дейтел П. "Как программировать на C/C++." Питер, 2013. - Главы, касающиеся разработки алгоритмов на C++.

6. Лафоре Р. "Объектно-ориентированное программирование в C++." Питер, 2002. - Практические примеры реализации алгоритмов на C++.

7. Макконнелл С. "Совершенный код. Мастер-класс." Питер, 2014. - Рекомендации по эффективной реализации алгоритмов.

# Приложение А.

Листинг программы.

#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

#include <ctime>

#include <string>

#include <locale.h>

using namespace std;

const int way = -3, wall = -2, pass = -1;

bool deadend(int x, int y, int maze[][99], int height, int width) {

int a = 0;

if (x != 1) {

if (maze[y][x - 2] == pass)

a += 1;

}

else

a += 1;

if (y != 1) {

if (maze[y - 2][x] == pass)

a += 1;

}

else

a += 1;

if (x != width - 2) {

if (maze[y][x + 2] == pass)

a += 1;

}

else

a += 1;

if (y != height - 2) {

if (maze[y + 2][x] == pass)

a += 1;

}

else

a += 1;

return a == 4;

}

bool ended(int maze[][99], int height, int width) {

bool b = true;

for (int i = 1; i < (height - 1); i += 2)

for (int j = 1; j < (width - 1); j += 2)

if (maze[i][j] == wall)

b = false;

return b;

}

void mazemake(int maze[][99], int height, int width) {

int x, y, c, a;

bool b;

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

for (int j = 0; j < width; j++)

maze[i][j] = wall;

x = 3; y = 3; a = 0;

while (1)

{

a++;

if (a % 100 == 0)

if (ended(maze, height, width))

break;

maze[y][x] = pass;

while (1) {

c = rand() % 4;

switch (c) {

case 0: if (y != 1)

if (maze[y - 2][x] == wall) {

maze[y - 1][x] = pass;

maze[y - 2][x] = pass;

y -= 2;

}

break;

case 1: if (y != height - 2)

if (maze[y + 2][x] == wall) {

maze[y + 1][x] = pass;

maze[y + 2][x] = pass;

y += 2;

}

break;

case 2: if (x != 1)

if (maze[y][x - 2] == wall) {

maze[y][x - 1] = pass;

maze[y][x - 2] = pass;

x -= 2;

}

break;

case 3: if (x != width - 2)

if (maze[y][x + 2] == wall) {

maze[y][x + 1] = pass;

maze[y][x + 2] = pass;

x += 2;

}

break;

}

if (deadend(x, y, maze, height, width))

break;

}

if (deadend(x, y, maze, height, width))

do {

x = 2 \* (rand() % ((width - 1) / 2)) + 1;

y = 2 \* (rand() % ((height - 1) / 2)) + 1;

} while (maze[y][x] != pass);

}

}

void drawing(int map[][99], int size) {

system("cls");

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

{

for (int j = 0; j < size; j++)

{

if (map[i][j] == wall)

cout << "# ";

else if (map[i][j] == way)

cout << "1 ";

else

cout << "- ";

}

cout << '\n';

}

}

void saveToFile(const char\* filename, int map[][99], int size) {

ofstream file(filename, ios\_base::out);

if (file.is\_open()) {

file << size << '\n';

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

for (int j = 0; j < size; ++j) {

file << map[i][j] << " ";

}

file << '\n';

}

file.close();

cout << "Maze has been saved to " << filename << endl;

}

else {

cout << "Unable to open the file for writing." << endl;

}

}

void loadFromFile(const char\* filename, int map[][99], int& size) {

ifstream file(filename, ios\_base::in);

if (file.is\_open()) {

file >> size;

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

for (int j = 0; j < size; ++j) {

file >> map[i][j];

}

}

file.close();

cout << "Maze has been loaded from " << filename << endl;

}

else {

cout << "Unable to open the file for reading." << endl;

}

}

void start() {

int map\_size;

do {

cout << "Введите размер лабиринта - только нечетные числа (от 5 до 99): ";

cin >> map\_size;

if (cin.fail() || map\_size <= 4 || map\_size >= 100 || map\_size % 2 == 0) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный размер лабиринта! Повторите попытку." << endl;

}

else {

break;

}

} while (true);

int map[99][99];

mazemake(map, map\_size, map\_size);

// Запись лабиринта в файл перед поиском расстояния

saveToFile("lab\_search.txt", map, map\_size);

drawing(map, map\_size);

map[1][1] = pass;

vector<pair<int, int>> oldWave;

oldWave.push\_back(pair<int, int>(1, 1));

int nstep = 0;

map[1][1] = nstep;

const int dx[] = { 0, 1, 0, -1 };

const int dy[] = { -1, 0, 1, 0 };

while (!oldWave.empty()) {

++nstep;

vector<pair<int, int>> wave;

for (const auto& i : oldWave) {

for (int d = 0; d < 4; ++d) {

int nx = i.first + dx[d];

int ny = i.second + dy[d];

if (nx >= 0 && nx < map\_size && ny >= 0 && ny < map\_size && map[nx][ny] == pass) {

wave.push\_back(pair<int, int>(nx, ny));

map[nx][ny] = nstep;

if (nx == map\_size - 2 && ny == map\_size - 2) {

oldWave.clear();

break;

}

}

}

}

oldWave = wave;

}

int x = map\_size - 2;

int y = map\_size - 2;

vector<pair<int, int>> wave;

wave.push\_back(pair<int, int>(x, y));

while (map[x][y] != 0) {

for (int d = 0; d < 4; ++d) {

int nx = x + dx[d];

int ny = y + dy[d];

if (nx >= 0 && nx < map\_size && ny >= 0 && ny < map\_size && map[x][y] - 1 == map[nx][ny]) {

x = nx;

y = ny;

wave.push\_back(pair<int, int>(x, y));

}

}

}

for (const auto& i : wave) {

map[i.first][i.second] = way;

}

drawing(map, map\_size);

getchar();

getchar();

}

int main() {

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

cout << "\t" << "Курсовая работа" << endl << "Тема: реализация алгоритма поиска путей в лабиринте" << endl << "\n"

<< "Выполнил: ст. гр. 22ВВВ1" << endl << "Уткин М.М." << endl;

system("pause");

int choice;

do {

system("cls");

cout << "1)Поиск расстояния между вершинами." << endl;

cout << "2)Выход." << endl;

cout << "Выберите действие:";

cin >> choice;

switch (choice) {

case 1:

system("cls");

start();

break;

case 2:

system("cls");

break;

default:

cout << endl;

cout << "Неверный выбор! Повторите попытку.";

cout << endl;

break;

}

} while (choice != 2);

return 0;

}