|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_***ИУК «Информатика и управление»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** \_\_***ИУК6 «Защита информации»***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**«**Исследование алгоритмов обхода лабиринта***.*»**

**Вариант 18**

**ДИСЦИПЛИНА: «Алгоритмы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК6-51 | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Щербаков В.И )  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Празян К.А )  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга , 2021

Постановка задачи

Дано: лабиринт в виде цифр

4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

4 10 2 2 4 2 2 2 2 2 4 2 2 2 2 2 2 2 4 4

4 4 4 2 4 2 4 2 4 2 4 2 4 4 4 4 4 2 4 4

4 2 4 2 4 2 4 2 4 2 4 2 2 2 2 2 4 2 4 4

4 2 4 2 4 4 4 2 4 2 4 4 4 4 4 4 4 2 4 4

4 2 4 2 2 2 2 2 4 2 4 2 2 2 4 2 2 2 4 4

4 2 4 4 4 4 4 4 4 2 4 2 4 2 4 2 4 2 4 4

4 2 2 2 2 2 4 2 2 2 4 2 4 2 2 2 4 2 4 4

4 2 4 4 4 4 4 2 4 4 4 2 4 4 4 4 4 4 4 4

4 2 4 2 2 2 2 2 4 2 2 2 4 2 2 2 2 2 4 4

4 2 4 2 4 4 4 4 4 2 4 4 4 2 4 4 4 2 4 4

4 2 4 2 2 2 4 2 4 2 2 2 2 2 2 2 4 2 4 4

4 2 4 4 4 2 4 2 4 2 4 4 4 4 4 4 4 2 4 4

4 2 2 2 4 2 4 2 2 2 2 2 4 2 2 2 4 2 4 4

4 2 4 4 4 2 4 4 4 4 4 4 4 2 4 2 4 2 4 4

4 2 2 2 4 2 4 2 2 2 4 2 2 2 4 2 2 2 4 4

4 2 4 2 4 2 4 2 4 2 4 2 4 4 4 4 4 2 4 4

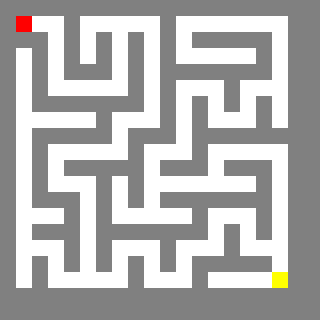
4 2 4 2 2 2 2 2 4 2 2 2 4 2 2 2 2 9 4 4

4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

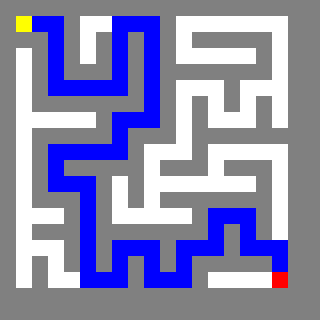
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

цифрой 4 обозначена стена  
цифрой 2 обозначено пустое пространство  
цифра 10 - стартовая позиция  
цифра 9 - точка выхода.

Цель: выработать практические навыки составления и реализации  
алгоритмов обхода лабиринта.



Пустая карта лабиринта.



Путь на карте лабиринта.

Алгоритм работы программы:

1)мы открываем файл .txt с лабиринтом и читаем данные с файла в двумерный массив.

2)мы передаем функции волнового алгоритма этот массив, переделывая сам массив так что бы в числовом виде остались только 0 и -1, запомнив при этом координаты начальной и конечной точек.

3) мы стартуя из конечной точки начинаем обходить лабиринт и при этом увеличиваем на каждом успешном шаге пути счётчик до тех пор пока не упремся в стену или не дойдём до нужной точки. После того как мы доходим до конечной точки мы помечаем эту точку и идём по пути уменьшения счётчика до тех пор пока счётчик не станет равным начальному значению и после этого мы получаем весь путь из начала в конец.

4)как только получили весь путь, мы изменяем изначальный массив так, чтобы отметить путь.

5)мы увеличиваем массив для более детального отображения в формате картинки.

6)мы рисуем получившуюся карту лабиринта без пути и с ним, попиксельно задавая для каждой цифры свой цвет.

Код на с++

#include "Bmp.h"

using namespace std;

void SearchWay(int\*\* map,int size)

{

int x = 0;

int y = 0;

int x\_to = 0;

int y\_to = 0;

int\*\*\* wave\_matrix = new int\*\* [size];

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

wave\_matrix[i] = new int\* [size];

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

wave\_matrix[i][j] = new int[3];

}

}

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

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

if (map[i][j] == 9) {

x = i;

y = j;

map[i][j] = 2;

}

if (map[i][j] == 10) {

x\_to = i;

y\_to = j;

map[i][j] = 2;

}

}

}

int step;

bool added = true, result = true;

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

{

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

{

if (map[i][j]!= 2)

{

wave\_matrix[i][j][0] = -2;// занято

}

else

{

wave\_matrix[i][j][0] = -1;// Мы еще нигде не были

}

}

}

wave\_matrix[x\_to][y\_to][0] = 0;//стартуем от финиша

step = 0; // Изначально мы сделали ноль шагов

// Пока вершины добаляются и мы не дошли до старта

while (added && wave\_matrix[x][y][0] == -1)

{

added = false;// Пока что ничего не добавили

step++;// Увеличиваем число шагов

for (int i = 0; i < size; i++)// Пробегаем по всей карте

{

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

{

// Если (i, j) была добавлена на предыдущем шаге

// Пробегаем по всем четырем сторонам

if (wave\_matrix[i][j][0] == step - 1)

{

int \_i, \_j;

\_i = i + 1; \_j = j;

// Если не вышли за пределы карты - обрабатываем

if (\_i >= 0 && \_j >= 0 && \_i < size && \_j < size)

{

// Если (\_i, \_j) уже добавлено или непроходимо, то не обрабатываем

if (wave\_matrix[\_i][\_j][0] == -1 && wave\_matrix[\_i][\_j][0] != -2)

{

wave\_matrix[\_i][\_j][0] = step; // Добавляем

wave\_matrix[\_i][\_j][1] = i;

wave\_matrix[\_i][\_j][2] = j;

added = true; // Что-то добавили

}

}

\_i = i - 1; \_j = j;

// Если не вышли за пределы карты - обрабатываем

if (\_i >= 0 && \_j >= 0 && \_i < size && \_j < size)

{

// Если (\_i, \_j) уже добавлено или непроходимо, то не обрабатываем

if (wave\_matrix[\_i][\_j][0] == -1 && wave\_matrix[\_i][\_j][0] != -2)

{

wave\_matrix[\_i][\_j][0] = step; // Добавляем

wave\_matrix[\_i][\_j][1] = i;

wave\_matrix[\_i][\_j][2] = j;

added = true; // Что-то добавили

}

}

\_i = i; \_j = j + 1;

// Если не вышли за пределы карты - обрабатываем

if (\_i >= 0 && \_j >= 0 && \_i < size && \_j < size)

{

// Если (\_i, \_j) уже добавлено или непроходимо, то не обрабатываем

if (wave\_matrix[\_i][\_j][0] == -1 && wave\_matrix[\_i][\_j][0] != -2)

{

wave\_matrix[\_i][\_j][0] = step; // Добавляем

wave\_matrix[\_i][\_j][1] = i;

wave\_matrix[\_i][\_j][2] = j;

added = true; // Что-то добавили

}

}

\_i = i; \_j = j - 1;

// Если не вышли за пределы карты - обрабатываем

if (\_i >= 0 && \_j >= 0 && \_i < size && \_j < size)

{

// Если (\_i, \_j) уже добавлено или непроходимо, то не обрабатываем

if (wave\_matrix[\_i][\_j][0] == -1 && wave\_matrix[\_i][\_j][0] != -2)

{

wave\_matrix[\_i][\_j][0] = step; // Добавляем

wave\_matrix[\_i][\_j][1] = i;

wave\_matrix[\_i][\_j][2] = j;

added = true; // Что-то добавили

}

}

}

}

}

}

if (wave\_matrix[x][y][0] == -1)

{

result = false; // то пути не существует

}

if (result)

{

int \_i = x, \_j = y;

while (wave\_matrix[\_i][\_j][0] != 0)

{

map[\_i][\_j] = 3;

// тут записываем в \_i \_J значение клеток пути для изначального массива и шагаем дальше к началу

int li = wave\_matrix[\_i][\_j][1];

int lj = wave\_matrix[\_i][\_j][2];

\_i = li; \_j = lj;

}

}

map[x][y] = 10;

map[x\_to][y\_to] = 9;

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

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

delete[] wave\_matrix[i][j];

}

delete[] wave\_matrix[i];

}

delete[] wave\_matrix;

}

void read\_txt(int\*\* map, int size) { //cout << "path to txt file"<<endl;string enter;cin >> enter;string line;

ifstream f;

f.open("18.txt");

if (f.is\_open())

{

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

{

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

{

f >> map[i][j];

}

}

f.close();

}

else {

cout << "eror" << endl;

}

}

void resize\_and\_write\_BMP(int\*\* map, int size, int b) {

BMP newBmp(b \* size, b \* size);

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

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

int x = trunc(i / b);

int y = trunc(j / b);

if (map[x][y] == 2) {

newBmp.set\_pixel(i, j, 255, 255, 255, 255);

}

if (map[x][y] == 3) {

newBmp.set\_pixel(i, j, 255, 0, 0, 255);

}

if (map[x][y] == 4) {

newBmp.set\_pixel(i, j, 128, 128, 128, 255);

}

if (map[x][y] == 10) {

newBmp.set\_pixel(i, j, 0, 0, 255, 255);

}

if (map[x][y] == 9) {

newBmp.set\_pixel(i, j, 0, 255, 255, 255);

}

}

}

newBmp.write("labyrinth\_with\_way2.bmp");

}

int main() {

int size = 20;

int b = 16;

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

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

map[i] = new int[size];

}

read\_txt(map, size);

SearchWay(map,size);

resize\_and\_write\_BMP(map, size, b);

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

delete[] map[i];

}

delete[] map;

return 0;

}

Вывод: в данной лабораторной работе я изучил как работает волновой алгоритм нашёл “библиотеку” для работы с бмп файлами для с++ по адресу https://github.com/sol-prog/cpp-bmp-images/blob/master/BMP.h, с помощью которой я научился рисовать попиксельно лабиринт, а так же научился с помощью волнового алгоритма находить путь в лабиринте.