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 «Разработка игрового агента для игры "Лабиринт"»

Выполнил

студент группы 21ВВ2

Шилов Алексей

Принял:

д.т.н. Митрохин М.А.

Пенза 2023

Содержание

[Реферат 5](#_Toc133554719)

[Введение 6](#_Toc133554720)

[Постановка задачи 7](#_Toc133554721)

[Описание алгоритма программы 8](#_Toc133554722)

[Описание программы 9](#_Toc133554723)

[Тестирование 15](#_Toc133554724)

[Заключение 18](#_Toc133554725)

[Список литературы 19](#_Toc133554726)

[Приложение А 20](#_Toc133554727)

# **Реферат**

Отчёт содержит 24 страницы, 3 рисунка 1 таблицу.

ГРАФ, ПОИСК В ГЛУБИНУ, ЛАБИРИНТ, ИГРОВОЙ АГЕНТ.

Цель исследования – разработка программы, способной генерировать лабиринты заданных пользователем размеров и осуществлять поиск выхода методом поиска в глубину.

В работе рассмотрены правила поиска в глубину, метод генерации идеальных лабиринтов заданных размеров. Реализован поиск выхода из лабиринта методом поиска в глубину.

# **Введение**

Агентом обычно является персонаж игры, но это может быть и машина, робот или даже нечто более абстрактное – целая группа сущностей, страна или цивилизация. В любом случае это объект, следящий за своим окружением, принимающий на основании него решения и действующий в соответствии с этими решениями.

Лабиринт – структура (обычно в двухмерном или трёхмерном пространстве), состоящая из запутанных путей к выходу (и/или путей, ведущих в тупик).

Поиск в глубину (англ. Depth-first search, DFS) – один из методов обхода графа. Алгоритм осуществляется с некоторой вершины вдоль ребер графа, до попадания в конец ребра. После этого нужно возвращаться назад вдоль пройденного пути, пока не будет обнаружена вершина, у которой есть еще не посещенная смежная вершина, и затем двигаться в направлении не посещённой вершины. Алгоритм продолжает свою работу пока не будут пройдены все вершины, либо не будет достигнута нужная вершина графа.

Поиск в ширину осуществляется с некоторой вершины и после посещения первой вершины, посещаются все соседние с ней вершины. Потом посещаются все вершины, находящиеся на расстоянии двух ребер от начальной. При каждом новом шаге посещаются вершины, расстояние от которых до начальной на единицу больше предыдущего.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, язык программирования – C++.

Целью данной курсовой работы является разработка программы для генерации лабиринта и обхода методом поиска в глубину. на языке C++.

# **Постановка задачи**

Требуется разработать программу, которая будет генерировать лабиринт, и которая будет иметь встроенный агент для прохождения полученного лабиринта.

Программа должна позволять пользователю задавать размеры генерируемого 2-мерного лабиринта. После ввода размеров программа должна выводить лабиринт, не содержащий петель и замкнутых цепей, все точки которого будут достижимыми. Из каждой точки должен существовать ровно один путь к любой другой точке. Лабиринт должен иметь только одно решение.

Далее игровой агент должен проходить сгенерированный лабиринт методом поиска в глубину.

Программа должна работать без ошибок и сбоев.

Устройство ввода – клавиатура и мышь.

# **Описание алгоритма программы**

Для алгоритма генерации лабиринта необходим двумерный массив maze (int). Данный массив будет хранить всю необходимую информацию о лабиринте.

Весь массив изначально заполнен «0». Далее задаётся начальная точка s с которой начнётся генерация путей в лабиринте, ей присваивается значение «1». Далее циклом случайным образом выбирается 1 из 4 направлений (Вверх, вниз, влево, вправо), в котором 2-м точкам s+1 и s+2 присваивается значение «1». Далее от точки s+2 снова выбирается направление и ещё 2-м точкам задаётся значение «1». В случае если ни по одному направлению нет возможности проложить путь, то цикл прерывается. Случайным образом выбирается новая точка со значением «0» и цикл возобновляется. Алгоритм продолжает работу до полного заполнения лабиринта. В конце генерации выбираются начальная и конечная (присваивается значение «2») точки.

Для алгоритма поиска пути дополнительных массивов не требуется. Поиск в глубину реализовывается рекурсивным способом. Алгоритм проверяет значение текущей точки s(x, y), при значении «2» алгоритм заканчивает работу, лабиринт считается пройденным, при значении «0» текущая ветвь рекурсии обрывается, при значении «1» алгоритм отмечает точку как посещённую (заменяет значение точка на «5») и создаёт 4 рекурсивных вызова для точек s(x+1, y), s(x, y+1), s(x-1, y), s(x, y-1).

# **Описание программы**

Для написания программы был использован язык C++.

Проект был создан в виде оконного приложения, для этого была подключена библиотека SFML – свободная кроссплатформенная мультимедийная библиотека и содержит ряд модулей для простого программирования игр и мультимедиа приложений.

Программа является многомодульной, т.к. содержит несколько функций: deadend, mazemake, Visit, main.

Работа программы начинается с запроса размеров будущего лабиринта, после ввода проводится проверка на нечётность введённых значений, т.к. для правильной работы алгоритма генерации требуются именно нечётные размеры массива. Далее т.к. лабиринт будет окружён стенами для сохранения требуемых размеров значения высоты и ширины лабиринта увеличиваются на 2. После создаётся двумерный массив для хранения лабиринта.

cout << "Input height and width (only odd sizes):" << "\n";

cin >> height;

cin >> width;

if (height % 2 == 0) {

height++;

}

if (width % 2 == 0) {

width++;

}

system("cls");

height += 2;

width += 2;

int\*\* maze = new int\* [height];

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

maze[i] = new int[width];

Далее вызывается функция mazemake, которая сгенерирует лабиринт.

В начале она заполняет весь массив нулями. Далее в переменных x и y указываются координаты стартовой позиции для генерации и создаётся счётчик со стартовым значением 0, далее запускается цикл while. Внутри этого цикла сначала по заданным координатам устанавливается значение «1» и значение счётчика увеличивается на 1. Далее идёт бесконечный цикл, который обрывается пи захождении в тупик. В нём в переменную c заносится случайное значение в диапазоне от 0 до 3 включительно. В зависимости от заданного числа выполняется проводится проверка на наличие в выбранном направлении границы лабиринта, если границы нет и проверяются значения следующих 2-х точек. При значении «0» в них заносится значение «1» и в переменные x и y заносятся координаты последней точки, тем самым прокладывая путь. Далее проводится проверка на тупик, для этого вызывается функция deadend, если проверка не пройдена, то цикл прерывается и производится «выход из тупика». Выбирается случайная точка со значением «0».

void mazemake(int\*\* maze, int height, int width) {

int x, y, c, count;

bool b;

for (int i = 0; i < height; i++) // Заполнение стенами

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

maze[i][j] = 0;

x = 1; y = 1; count = 0; // Стартовая точка и счетчик

while (count < 10000) {

maze[y][x] = 1;

count++;

while (1) { // Бесконечный цикл, прерывается только тупиком

c = rand() % 4;

switch (c) {

case 0: if (y != 1)

if (maze[y - 2][x] == 0) { // Вверх

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

maze[y - 2][x] = 1;

y -= 2;

}

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

if (maze[y + 2][x] == 0) { // Вниз

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

maze[y + 2][x] = 1;

y += 2;

}

case 2: if (x != 1)

if (maze[y][x - 2] == 0) { // Налево

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

maze[y][x - 2] = 1;

x -= 2;

}

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

if (maze[y][x + 2] == 0) { // Направо

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

maze[y][x + 2] = 1;

x += 2;

}

}

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

break;

}

if (deadend(x, y, maze, height, width)) // Выход из тупика

do {

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

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

} while (maze[y][x] != 1);

}

}

Для проверки на тупик вызывается функция deadend. В начале функции объявляется переменная «a» со значением «0». Далее поочерёдно проверяется окружение точки если точка граничит со внешней стеной, то значение «a» увеличивается на 1, если по выбранному направлению есть точка со значением «1» значение переменной «a» также повышается на 1. Если по окончанию проверки переменная содержит значение «4», то точка считается тупиковой.

bool deadend(int x, int y, int\*\* maze, int height, int width) {

int a = 0;

if (x != 1) {

if (maze[y][x - 2] == 1)

a += 1;

}

else a += 1;

if (y != 1) {

if (maze[y - 2][x] == 1)

a += 1;

}

else a += 1;

if (x != width - 2) {

if (maze[y][x + 2] == 1)

a += 1;

}

else a += 1;

if (y != height - 2) {

if (maze[y + 2][x] == 1)

a += 1;

}

else a += 1;

if (a == 4)

return 1;

else

return 0;

}

После генерации устанавливаются стартовые координаты и финишная точка. Лабиринт выводится на экран, для этого выводится окно и загружаются необходимые текстуры.

RenderWindow window(VideoMode(width \* TileSize, height \* TileSize), "Maze!");

Texture t;

t.loadFromFile("Paint/Bplat.png");

Sprite plat(t);

int x = 1, y = 1; //Cтартовые координаты

int newx = 0, newy = 0;

maze[height - 1][width - 2] = 2; //Спавн финиша

Texture w;

w.loadFromFile("Paint/win.png");

Sprite win(w);

win.setPosition(20, 140);

Texture p;

p.loadFromFile("Paint/pause.png");

Sprite pause(p);

pause.setPosition(80, 140);

Далее по нажатию клавиши «Пробел» запускается игровой агент.

Event event;

while (window.pollEvent(event))

{

if (event.type == Event::Closed)

window.close();

if (event.key.code == Keyboard::Space) Visit(maze, 1, 1, plat, window);

}

Агент является рекурсивным. Сначала проверяется значение Win, пока установлено значение «false» алгоритм проверяет значение указанной точки, для которой он был вызван. Если заданные координаты выходят за пределы лабиринта, то дальнейшие действия не производятся, и текущая ветвь ротации обрывается. Далее если значение точки равно «2», то устанавливается значение «5» (точка становится посещённой), в переменную Win заносится «true», алгоритм завершает работу и лабиринт считается пройденным. Далее если значение точки отличается от «1», то текущая ветвь рекурсии обрывается, иначе устанавливается значение «5». Далее программа заново отрисовывает изменения в лабиринте и создаёт рекурсивные вызовы для соседних точек.

void Visit(int\*\* maze, int x, int y, Sprite plat, RenderWindow& window)

{

if (!Win) {

if (x < 1 || x >= width || y < 1 || y >= height) return;

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

maze[y][x] = 5;

Win = true;

sound.win.play();

exit;

}

if (maze[y][x] != 1) return;

maze[y][x] = 5;

window.clear(Color::White);

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

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

if (maze[i][j] == 0)

plat.setTextureRect(IntRect(0, 0, TileSize, TileSize));

if (maze[i][j] == 5)

plat.setTextureRect(IntRect(TileSize, 0, TileSize, TileSize));

if (maze[i][j] == 2)

plat.setTextureRect(IntRect(TileSize \* 2, 0, TileSize, TileSize));

if (maze[i][j] == 1)

continue;

plat.setPosition(j \* TileSize - OffsetX, i \* TileSize - OffsetY);

window.draw(plat);

}

}

window.display();

Sleep(200);

Visit(maze, x + 1, y, plat, window);

Visit(maze, x, y + 1, plat, window);

Visit(maze, x - 1, y, plat, window);

Visit(maze, x, y - 1, plat, window);

}

}

После успешного прохождения лабиринта программа выводит сообщение о победе и по нажатию на клавишу «Escape» завершает работу.

# **Тестирование**

Отладка и тестирование программного продукта это один из самых важных этапов разработки. Проведение тестирований и отладки программного продукта позволяет выявить и исключить большое количество ошибок и недоработок. Без этого этапа не представляется возможным написание законченного, правильно работающего приложения.

Отладка – этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки.

Тестирование – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением.

Тестирование и отладка программы производились в процессе написания программы и после завершения разработки. В ходе тестирования были выявлены проблемы и ошибки программы, связанные со вводом данных, отображением лабиринта, работой алгоритма игрового агента и последующего отображения результатов. При дальнейшей отладке данные ошибки были устранены.

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод консольного окна с запросом ввести размеры лабиринта | Верно |
| Ввод размеров данных | Очистка окна консоли, открытие графического окна со сгенерированным лабиринтом заданного размера | Верно |
| Нажатие клавиши «Пробел» | Запуск игрового агента | Верно |
| Работа игрового агента | Последовательный проход ветвей лабиринта в поиске финишной клетки и остановка алгоритма по нахождении финиша | Верно |
| Нажатие клавиши «Escape» | Закрытие программы | Верно |

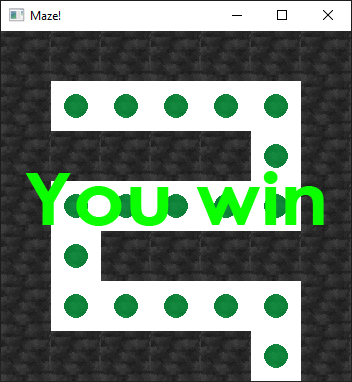


Рисунок 1 – Пройденный лабиринт размера 5 на 5

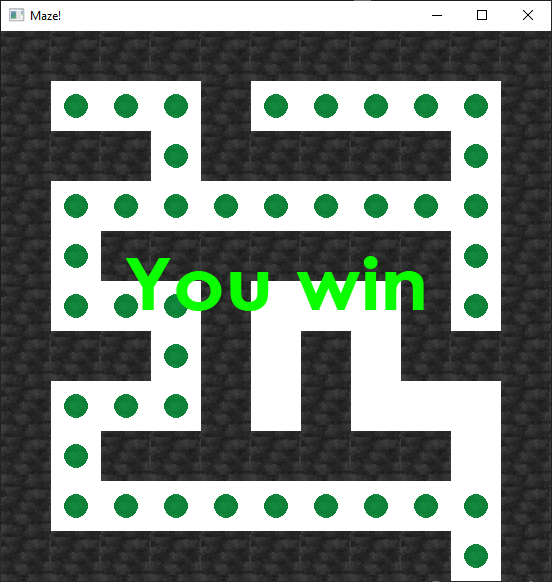


Рисунок 2 – Пройденный лабиринт размера 9 на 9

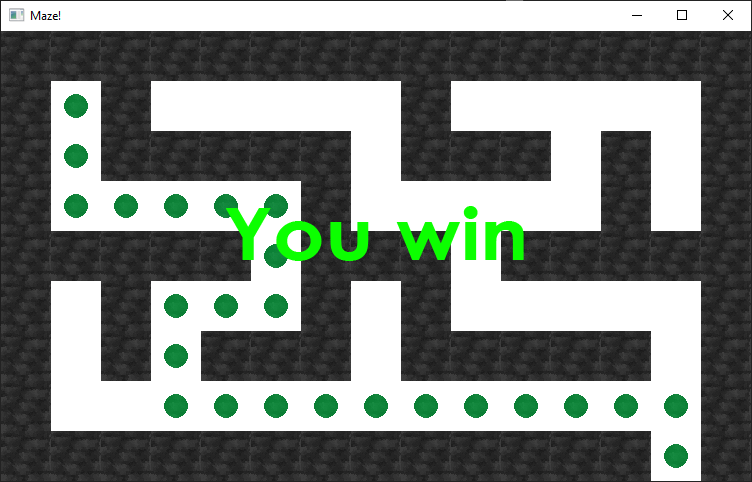


Рисунок 3 – Пройденный лабиринт размера 7 на 13

# **Заключение**

В процессе создания данного проекта разработана программа, реализующая корректную генерацию лабиринта различных размеров и содержащая игровой агент для прохождения сгенерированных лабиринтов методом поиска в глубину.

Для реализации программы использовалась среда Microsoft Visual Studio 2019 и язык программирования C++.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания и работы с графами. Приобретены навыки по осуществлению метода поиска в глубину.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

Программу можно доработать путём полного отказа от консоли, доработкой метода генерации лабиринта или добавлением новых.

Поставленные задачи выполнены в полном объёме.

# **Список литературы**

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и
2. анализ - М.: МЦНМО, 2001. - 960 с.
3. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
4. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
5. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
6. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009г.
7. 3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.

# **Приложение А**

**Листинг программы**

#include <SFML/Graphics.hpp>

#include <SFML/Audio.hpp>

#include <windows.h>

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace sf;

using namespace std;

const int TileSize = 50;

int height = 21;

int width = 21;

bool deadend(int, int, int\*\*, int, int); // Вспомогательная функция, определяет тупики

void mazemake(int\*\*, int, int); // Алгоритм генерации

//Проход = 1

//Стена = 0

//Финиш = 2

//Посещено = 5

bool deadend(int x, int y, int\*\* maze, int height, int width) {

int a = 0;

if (x != 1) {

if (maze[y][x - 2] == 1)

a += 1;

}

else a += 1;

if (y != 1) {

if (maze[y - 2][x] == 1)

a += 1;

}

else a += 1;

if (x != width - 2) {

if (maze[y][x + 2] == 1)

a += 1;

}

else a += 1;

if (y != height - 2) {

if (maze[y + 2][x] == 1)

a += 1;

}

else a += 1;

if (a == 4)

return 1;

else

return 0;

}

void mazemake(int\*\* maze, int height, int width) {

int x, y, c, count;

bool b;

for (int i = 0; i < height; i++) // Заполнение стенами

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

maze[i][j] = 0;

x = 1; y = 1; count = 0; // Стартовая точка и счетчик

while (count < 10000) {

maze[y][x] = 1;

count++;

while (1) { // Бесконечный цикл, прерывается только тупиком

c = rand() % 4;

switch (c) {

case 0: if (y != 1)

if (maze[y - 2][x] == 0) { // Вверх

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

maze[y - 2][x] = 1;

y -= 2;

}

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

if (maze[y + 2][x] == 0) { // Вниз

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

maze[y + 2][x] = 1;

y += 2;

}

case 2: if (x != 1)

if (maze[y][x - 2] == 0) { // Налево

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

maze[y][x - 2] = 1;

x -= 2;

}

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

if (maze[y][x + 2] == 0) { // Направо

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

maze[y][x + 2] = 1;

x += 2;

}

}

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

break;

}

if (deadend(x, y, maze, height, width)) // Выход из тупика

do {

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

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

} while (maze[y][x] != 1);

}

}

float OffsetX = TileSize \* (0); //Установка поля зрения центром

float OffsetY = TileSize \* (0); //на стартовую позицию игрока

bool Win = false;

int pe = 2;

class setSound {

public:

SoundBuffer s1, s2, s3;

Sound time, win;

setSound() {

s3.loadFromFile("Sound/win.ogg");

time.setBuffer(s1);

win.setBuffer(s3);

time.setLoop(true);

}

};

setSound sound;

void Visit(int\*\* maze, int x, int y, Sprite plat, RenderWindow& window)

{

if (!Win) {

if (x < 1 || x >= width || y < 1 || y >= height) return;

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

maze[y][x] = 5;

Win = true;

sound.win.play();

exit;

}

if (maze[y][x] != 1) return;

maze[y][x] = 5;

window.clear(Color::White);

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

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

if (maze[i][j] == 0)

plat.setTextureRect(IntRect(0, 0, TileSize, TileSize));

if (maze[i][j] == 5)

plat.setTextureRect(IntRect(TileSize, 0, TileSize, TileSize));

if (maze[i][j] == 2)

plat.setTextureRect(IntRect(TileSize \* 2, 0, TileSize, TileSize));

if (maze[i][j] == 1)

continue;

plat.setPosition(j \* TileSize - OffsetX, i \* TileSize - OffsetY);

window.draw(plat);

}

}

window.display();

Sleep(200);

Visit(maze, x + 1, y, plat, window);

Visit(maze, x, y + 1, plat, window);

Visit(maze, x - 1, y, plat, window);

Visit(maze, x, y - 1, plat, window);

}

}

int main(int argc, char\* argv[]) {

srand((unsigned)time(NULL));

cout << "Input height and width (only odd sizes):" << "\n";

cin >> height;

cin >> width;

if (height % 2 == 0) {

height++;

}

if (width % 2 == 0) {

width++;

}

system("cls");

height += 2;

width += 2;

int\*\* maze = new int\* [height];

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

maze[i] = new int[width];

mazemake(maze, height, width);

RenderWindow window(VideoMode(width \* TileSize, height \* TileSize), "Maze!");

Texture t;

t.loadFromFile("Paint/Bplat.png");

Sprite plat(t);

int x = 1, y = 1; //Cтартовые координаты

int newx = 0, newy = 0;

maze[height - 1][width - 2] = 2; //Спавн финиша

Texture w;

w.loadFromFile("Paint/win.png");

Sprite win(w);

win.setPosition(20, 140);

while (window.isOpen())

{

Event event;

while (window.pollEvent(event))

{

if (event.type == Event::Closed)

window.close();

if (event.key.code == Keyboard::Space) Visit(maze, 1, 1, plat, window);

window.clear(Color::White);

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

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

if (maze[i][j] == 0)

plat.setTextureRect(IntRect(0, 0, TileSize, TileSize));

if (maze[i][j] == 5)

plat.setTextureRect(IntRect(TileSize, 0, TileSize, TileSize));

if (maze[i][j] == 2)

plat.setTextureRect(IntRect(TileSize \* 2, 0, TileSize, TileSize));

if (maze[i][j] == 1)

continue;

plat.setPosition(j \* TileSize - OffsetX, i \* TileSize - OffsetY);

window.draw(plat);

}

}

if (Win) {

window.draw(win);

if (event.key.code == Keyboard::Escape)

return 0;

}

window.display();

}

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

delete[] maze[i];

delete[] maze;

return 0;

}