


Министерство науки и высшего образования РФ
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по курсу «Логика и основы алгоритмизации в инженерных задачах»
на тему «Разработка игрового агента для игры «Лабиринт» »

ОАП.
21.12.23


Выполнил ст. гр. 22ВВВ2:

Китаев Я. Е.

Приняли:

Митрохин М.А.

Акифьев И.В

Пенза 2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ _____

«____» _____ 20__

ЗАДАНИЕ

на курсовое проектирование по курсу

«Логика и основы алгоритмизации в инженерных задачах»
Студенту Китаву Ярошаву Евгеньевичу Группа 2213332
Тема проекта Разработка игрового агента для игры
«Лабиринт»

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения
в соответствии с данным заданием курсового
проекта.

Пояснительная записка должна содержать:

1. Постановку задачи;
2. Теоретическую часть задания;
3. Описание алгоритма поставленной задачи;
4. Пример ручного расчёта задачи и
вычисления;
5. Описание самой программы;
6. Тесты;
7. Список литературы;
8. Листинг программы;
9. Результаты работы программы;

Объем работы по курсу

1. Расчетная часть

Ручной расчёт работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схем

3. Экспериментальная часть

Тестирование программы;
Результат работы программы на тестовых
данных

Срок выполнения проекта по разделам

- 1 Исследование теоретической части курсового
- 2 Разработка алгоритмов программы
- 3 Разработка программы
- 4 Тестирование и завершение разработки программы
- 5 Оформление пояснительной записки
- 6
- 7
- 8

Дата выдачи задания "28" 09.10.23

Дата защиты проекта " " "

Руководитель _____

Задание получил "28" сентября 2023 г.

Студент Китаев Арсав Евгеньевич

Содержание

Введение	5
1 Постановка задачи	7
2 Выбор решения	8
3 Описание алгоритма программы.....	9
3.1Алгоритм создания лабиринта	9
3.2 Алгоритм поиска пути в лабиринте	11
4 Описание программы	14
5 Отладка	29
6 Тестирование.....	31
Заключение.....	34
Список используемых источников	35
Приложение А (Схема взаимодействия функций).....	36
Приложение Б (Листинги программы).....	37

Введение

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

Под лабиринтом у древних греков и римлян подразумевалось более или менее обширное пространство, состоящее из многочисленных залов, камер, дворов и переходов, расположенных по сложному и запутанному плану, с целью запутать и не дать выхода несведущему в плане лабиринта человеку. В широком смысле слова лабиринт может представлять тупиковую ситуацию или дело, из которого очень сложно найти выход.

Игровой агент для лабиринта - это программа или алгоритм, который способен найти выход из лабиринта. Игровые агенты для лабиринтов могут быть использованы в различных областях, таких как в компьютерных играх, робототехнике. Эти агенты могут использовать различные методы и стратегии для поиска оптимального пути через лабиринт, такие как алгоритмы поиска в глубину, алгоритмы поиска в ширину, машинное обучение.

В данной курсовой работе представлена генерация простого лабиринта, путь к выходу из которого находит игровой агент.

Лабиринт- это программа, которая предназначена для развлечения пользователя, поэтому её интерфейс, в данной курсовой работе, выполнен в игровом стиле с сочетанием позитивных красок, которые обостряют эмоциональную часть игрового процесса.

В курсовой работе присутствует два варианта работы программы:

1. «Автоматический» - программа показывает путь к выходу из лабиринта
2. «Игра» - пользователю предлагается посоревноваться с программой в скорости поиска выхода из лабиринта.

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

1 Постановка задачи

Необходимо разработать программу для генерации лабиринта размер которого будет вводиться с клавиатуры.

Для этого следует продумать логику создания лабиринта любого размера.

Необходимо разработать игрового агента для прохождения лабиринта.

Для этого:

- Выбрать способ поиска пути к выходу из лабиринта
- Реализовать программу на основе выбранного способа

Необходимо создать интуитивно понятный интерфейс

Для этого:

- научиться работать с графической библиотекой;
- подобрать цвета и шрифты;
- продумать меню и возможность перезапуска программы;

Обязательные требования к программе (по заданию):

- текстовое или графическое меню;
- возможность задания пользователем размера графа (множества);
- возможность выбора автоматического (случайного) или ручного (с клавиатуры или из файла) задания графа (элементов множества);
- возможность сохранения результатов работы программы;

В случае победы пользователя предусмотреть запись его имени и времени прохождения лабиринта в таблицу победителей и вывод её на экран

2Выбор решения

При запуске программы пользователь должен ввести с клавиатуры высоту и ширину лабиринта. После этого на экран выводится случайно сгенерированный лабиринт с заданными размерами.

Далее на экране появляется меню с выбором. При наведении мышкой на надпись, та обводится в овал:

1. «Игра» - пользователь может посоревноваться в скорости поиска выхода из лабиринта с программой
 - 1.1.Игрок вводит скорость с которой бот будет идти до выхода по заранее найденному пути.
 - 1.2. Бот (представленный синим квадратом) начинает движение с заданной скоростью.
 - 1.3. В это время игрок (представленный зелёнымквадратом) тоже начинает поиск пути до выхода, передвигаясь по лабиринту путём нажатия клавиш со стрелочками.
 - 1.4. Если бот находит выход первым, то на экране появляется окно с надписью «Проигрыш»
 - 1.5.Если игрок находит выход первым, то на экране появляется окно с надписью «!Победа!», после чего пользователь вводит своё имя в таблицу победителей. Таблица выводится на экран.
2. «Автоматически » - программа выводит на экран путь к выходу.

3 Описание алгоритма программы

3.1 Алгоритм создания лабиринта

Для лабиринта нам понадобится массив, который будет содержать информацию о проходах и стенах. Размер массива вводится с клавиатуры, далее он заполняется «0» - стена.

Суть алгоритма заключается в том, что программа при помощи функции `rand()` генерирует случайное число от 0 до 3, где:

- 0 - от местоположения вверх
- 1 - от местоположения вниз
- 2 - от местоположения налево
- 3 - от местоположения направо

Начальная точка генерации $y = 3$, $x = 3$, где y - высота - ширина.

После выбора направления движения бота идёт проверка может ли он туда пойти, если да, то бот прорывает 2 клетки в заданном направлении.

После этого запускается функция `deadend()`, которая проверяет, является ли текущая позиция в лабиринте тупиковой. Она проверяет, есть ли проходы вокруг текущей позиции. Если все четыре прохода заблокированы стенами, функция возвращает «1», что означает, что текущая позиция является тупиковой.

Если функция `deadend()`, вернула «0», то программа продолжает создание пути с текущей позиции.

Если функция `deadend()`, вернула «1», то алгоритм выбирает новую случайную стартовую позицию и продолжает создавать проходы.

Алгоритм продолжает создание лабиринта, пока $(a < 10000)$.

Вход: `maze` - массив в котором будет создан лабиринт, `height` - высота лабиринта, `width` - ширина лабиринта.

Выход: тупик или не тупик.

1. Функция `deadend()`

- 1.1. $a = 0$
- 1.2. ЕСЛИ $x \neq 1$

```

1.3. ТО
1.4. ЕСЛИ maze[y][x - 2] == пустой клетке
1.5. ТО
1.6. a + 1
1.7. ИНАЧЕ
1.8. a + 1
1.9. ЕСЛИ y != 1
1.10. ТО
1.11. ЕСЛИ maze[y - 2][x] == пустой клетке
1.12. ТО
1.13. a + 1
1.14. ИНАЧЕ
1.15. a + 1
1.16. ЕСЛИ x != ширина - 2
1.17. ТО
1.18. ЕСЛИ maze[y][x + 2] == пустой клетке
1.19. ТО
1.20. a + 1
1.21. ИНАЧЕ
1.22. a + 1
1.23. ЕСЛИ y != высота - 2
1.24. ТО
1.25. ЕСЛИ maze[y + 2][x] == пустой клетке
1.26. ТО
1.27. a + 1
1.28. ИНАЧЕ
1.29. a + 1
1.30. ЕСЛИ a = 4
1.31. ТО
1.32. ВОЗВРАЩАЕТ 1
1.33. ИНАЧЕ
1.34. ВОЗВРАЩАЕТ 0
1.35.

```

Вход: maze – массив в котором будет создан лабиринт, height – высота лабиринта, width – ширина лабиринта.

Выход: сгенерированный лабиринт.

2. Функция mazemake ()

```

2.1. Заполняем массив maze стенами
2.2. x = 3
2.3. y = 3
2.4. a = 0
2.5. ПОКА a < 10000
2.6. maze[y][x] = пустой клетке
2.7. a+1
2.8. ПОКА 1
2.9. c = случайное число от 0 до 4
2.10. ЕСЛИ c = 0
2.11. ТО

```

```

2.12.  ЕСЛИ y != 1
2.13.  ТО
2.14.  ЕСЛИ maze[y - 2][x] == wall
2.15.  ТО
2.16.  maze[y - 1][x] = пустой клетке
2.17.  maze[y - 2][x] = пустой клетке
2.18.  y - 2
2.19.  ЕСЛИ c = 1
2.20.  ТО
2.21.  ЕСЛИ y != высота - 2
2.22.  ТО
2.23.  ЕСЛИ maze[y + 2][x] == стена
2.24.  ТО
2.25.  maze[y + 1][x] = пустой клетке
2.26.  maze[y + 2][x] = пустой клетке
2.27.  y += 2
2.28.  ЕСЛИ c = 2
2.29.  ТО
2.30.  ЕСЛИ x != 1
2.31.  ТО
2.32.  ЕСЛИ maze[y][x - 2] == стена
2.33.  ТО
2.34.  maze[y][x - 1] = пустой клетке
2.35.  maze[y][x - 2] = пустой клетке
2.36.  x -= 2;
2.37.  ЕСЛИ c = 3
2.38.  ТО
2.39.  ЕСЛИ x != ширина - 2
2.40.  ТО
2.41.  ЕСЛИ maze[y][x + 2] == стена
2.42.  ТО
2.43.  maze[y][x + 1] = пустой клетке
2.44.  maze[y][x + 2] = пустой клетке x += 2
2.45.  ЕСЛИ функция deadend(maze, height, width) вернула 1)
2.46.  ТО
2.47.  Выйти из цикла
2.48.  ЕСЛИ функция deadend(maze, height, width) вернула 1)
2.49.  ТО {
2.50.  ДЕЛАТЬ
2.51.  x = 2 * (rand() % ((ширина - 1) / 2)) + 1;
2.52.  y = 2 * (rand() % ((высота - 1) / 2)) + 1;
2.53.  ПОКА maze[y][x] != пустой клетке

```

3.2 Алгоритм поиска пути в лабиринте

Поиск пути в лабиринте происходит при помощи обхода в глубину.

Функция `poisk_puti` принимает в качестве аргументов двумерный массив `maze`, представляющий лабиринт, а также его высоту и ширину.

Алгоритм начинает поиск пути из точки (1, 1) и продолжает до достижения точки (height - 2, width - 2).

Алгоритм продолжает движение вверх, вниз, налево или направо, пока не достигнет развилки или тупика.

Если достигнута развилка, то в массиве maze соответствующая ячейка помечается как razvil, после чего алгоритм продолжает движение.

Если достигнут тупик, то вызывается функция vozvrat(), которая отвечает за возврат от тупика до развилки. Также функция vozvrat(), помечает все ячейки от тупика до развилки как ошибочными.

Вход: maze – массив в котором будет создан лабиринт, height – высота лабиринта, width – ширина лабиринта.

Выход: бот возвращается к развилке

3. Алгоритм vozvrat() {

3.1. **ПОКА** maze[y][x] != развилка

3.2. **ЕСЛИ** maze[y - 1][x] == путь **ИЛИ** maze[y - 1][x] == развилка

3.3. **ТО** {

3.4. maze[y][x] = неверный путь

3.5. y-1

3.6. **ИНАЧЕЕСЛИ** maze[y + 1][x] == путь **ИЛИ** maze[y + 1][x] == развилка

3.7. **ТО**

3.8. maze[y][x] = неверный путь

3.9. y+1

3.10. **ИНАЧЕЕСЛИ** maze[y][x - 1] == путь **ИЛИ** maze[y][x - 1] == развилка

3.11. **ТО**

3.12. maze[y][x] = неверный путь

3.13. x-1

3.14. **ИНАЧЕЕСЛИ** maze[y][x + 1] == путь **ИЛИ** maze[y][x + 1] == развилка

3.15. **ТО**

3.16. maze[y][x] = неверный путь

3.17. x+1

Вход: maze – массив в котором будет создан лабиринт, height – высота лабиринта, width – ширина лабиринта.

Выход: путь от входа к выходу.

4. Алгоритм poisk_puti()

4.1. razvilka = 0

4.2. save_y = 0

4.3. save_x = 0


```

4.4. x = 1
4.5. y = 1
4.6. ПОКА(y != (высота - 2)) ИЛИ (x != (ширина - 2))
4.7. save_y = 0
4.8. save_x = 0
4.9. razvilka = 0
4.10. ЕСЛИ maze[y - 1][x] == свободная ячейка
4.11. ТО
4.12.     razvilka++
4.13.     save_y = -1
4.14.     save_x = 0
4.15. ЕСЛИ maze[y + 1][x] == свободная ячейка
4.16. ТО
4.17.     razvilka++
4.18.     save_y = 1
4.19.     save_x = 0
4.20. ЕСЛИ maze[y][x - 1] == свободная ячейка
4.21. ТО
4.22.     razvilka++
4.23.     save_x = -1
4.24.     save_y = 0
4.25. ЕСЛИ maze[y][x + 1] == свободная ячейка
4.26. ТО
4.27.     razvilka++
4.28.     save_x = 1
4.29.     save_y = 0
4.30. ЕСЛИ с позиции бота только один проход
4.31. ТО
4.32.     maze[y][x] = путь
4.33.     x += save_x
4.34.     y += save_y
4.35. ИНАЧЕ ЕСЛИ с позиции бота > 1 прохода
4.36. ТО
4.37.     maze[y][x] = 3 (развилка)
4.38.     x += save_x
4.39.     y += save_y
4.40. ИНАЧЕ
4.41.     maze[y][x] = путь
4.42. ВЫЗОВ ФУНКЦИИ vozvrat(maze, height, width)
4.43.     maze[height - 2][width - 2] = путь
4.44.     maze[height - 2][width - 1] = путь

```

4 Описание программы

Для написания данной программы использован язык программирования

Си++. Язык программирования Си - универсальный язык программирования.

Данная программа является многомодульной, поскольку состоит из нескольких функций: `input_size`, `deadend`, `mazemake`, `visual`, `vnes_lider`, `vozvrat`, `poisk_puti`, `П_proxod`, `play`, `input_time_П`, `mod_selectino`, `return_menu`.

Функции: `input_size`, `visual`, `vnes_lider`, `play`, `input_time_П`, `mod_selectino`, `return_menu` данные функции задают цвета, шрифты, размеры текста, выводят данные в окно `windows`, всё это осуществляется при помощи возможностей графической библиотеки SFML.

В начале пользователь вводит размеры лабиринта: высоту и ширину.

Ввод осуществляется при помощи функции `input_size`. Также функция считывает нажатую клавишу, если эта цифра, то она записывается в массив, после чего выводится в окно `windows` (рисунок 1).

Вход: `height` – высота лабиринта, `width` – ширина лабиринта.

Выход: высота и ширина лабиринта.

```
void input_size(int* height, int* width) {
    RenderWindow window(VideoMode(800, 400), "Labyrinth");
    char size[4];
    bool vibor = TRUE;
    int i = -1, enter = 0;

    window.clear(Color::White);
    text.setString(L"Введите размеры:");
    text.setPosition(40, 90);
    text.setCharacterSize(60);
    text.setFillColor(Color::Red);
    window.draw(text);

    text.setString(L"Высота: ");
    text.setPosition(40, 200);
    text.setCharacterSize(60);
    text.setFillColor(Color::Blue);
    window.draw(text);
    window.display();
}
```

```

while (window.isOpen() && enter != 2)
{
    Event event;
    while (window.pollEvent(event) && enter != 2)
    {
        if (event.type == sf::Event::Closed || enter == 2)
        window.close();
    }
    if (Keyboard::isKeyPressed(Keyboard::Enter)) {
        int m = 1;
        if (vibor) {
            for (int j = i; j >= 0; j--) {
                switch (j) {
case(0):
                    (*height) += (int(size[j]) - 48) * m;
                    size[j] = NULL;
                    break;
case(1):
                    (*height) += (int(size[j]) - 48) * m;
                    m *= 10;
                    size[j] = NULL;
                    break;
case(2):
                    (*height) += (int(size[j]) - 48) * m;
                    m *= 10;
                    size[j] = NULL;
                    break;
                default:
                    cout<< "Ошибка";
                }
            }
            if ((*height) % 2 == 0) {
                (*height)++;
            }
        }
        vibor = FALSE;
        enter++;
        i = -1;
        Sleep(100);
    }
    else {
        for (int j = i; j >= 0; j--) {
            switch (j) {
case(0):
                    (*width) += (int(size[j]) - 48) * m;
                    break;
case(1):
                    (*width) += (int(size[j]) - 48) * m;
                    m *= 10;
                    break;
case(2):
                    (*width) += (int(size[j]) - 48) * m;
                    m *= 10;
                    break;
                default:
                    cout<< "Ошибка";
                }
            }
            if ((*width) % 2 == 0) {
                (*width)++;
            }
        }
        enter++;
    }
    Sleep(100);
}

```

```

    }
    if ((event.type == Event::TextEntered&& (event.key.code == '0' || event.key.code == '1' || event.key.code == '2' ||
event.key.code == '3' || event.key.code == '4' || event.key.code == '5' || event.key.code == '6' || event.key.code == '7' ||
event.key.code == '8' || event.key.code == '9')) || Keyboard::isKeyPressed(Keyboard::LShift) || i == -1) {

window.clear(Color::White);
text.setString(L"Введите размеры:");
text.setPosition(40, 90);
text.setCharacterSize(60);
text.setFillColor(Color::Red);
window.draw(text);
    if (vibor) {
text.setString(L"Высота: ");
text.setPosition(40, 200);
text.setCharacterSize(60);
text.setFillColor(Color::Blue);
window.draw(text);
    }
    else {
text.setString(L"Ширина:");
text.setPosition(40, 200);
text.setCharacterSize(60);
text.setFillColor(Color::Blue);
window.draw(text);
    }
text.setCharacterSize(70);
text.setFillColor(Color::Green);
    if (event.type == Event::TextEntered&& (event.key.code == '0' || event.key.code == '1' || event.key.code == '2' ||
event.key.code == '3' || event.key.code == '4' || event.key.code == '5' || event.key.code == '6' || event.key.code == '7' ||
event.key.code == '8' || event.key.code == '9')) {
i++;
        size[i] = event.key.code;
        for (int g = 0; g <= i; g++) {
text.setString(size[g]);
text.setPosition(330 + g * 50, 190);
window.draw(text);
        }
    }
    if (Keyboard::isKeyPressed(Keyboard::LShift)) {
        size[i] = NULL;
i--;
        for (int g = 0; g <= i; g++) {
text.setString(size[g]);
text.setPosition(330 + g * 50, 190);
window.draw(text);
        }
    }
window.display();
Sleep(200);

    }
}
}
}

```

ВВЕДИТЕ РАЗМЕРЫ :

ВЫСОТА : 21

Рисунок 1 – Ввод размеров

На следующем шаге на экране появляется сгенерированный лабиринт. Выводом лабиринта на экран занимается функция `visual`. При помощи возможностей графической библиотеки SFML осуществляется вывод лабиринта в окно `windows`(рисунки 2).

Вход: `height` – высота лабиринта, `width` – ширина лабиринта, `maze` – массив с лабиринтом, `window`–окно `windows`.

Выход: отображает лабиринт в окне `windows`.

```
Voidvisual(int** maze, intheight, intwidth, RenderWindow* window, boolavto)
{
    wid = (width * 23) / 15;
    hei = (height * 25) / 10;

    RectangleShapewal(Vector2f(20, 20));
    RectangleShapeway(Vector2f(20, 20));
    RectangleShapespace(Vector2f(20, 20));
    RectangleShapeplay(Vector2f(20, 20));
    RectangleShapeIII(Vector2f(20, 20));

    way.setFillColor(Color::Red);
    wal.setFillColor(Color::Black);
    space.setFillColor(Color::White);
    play.setFillColor(Color::Green);
    III.setFillColor(Color::Blue);
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++) {
            switch (maze[i][j])
            {
                case (wall):
                    wal.setPosition(wid + j * 20, hei + i * 20);
                    (*window).draw(wal);
                    break;
                case (pass):
                    space.setPosition(wid + j * 20, hei + i * 20);
                    (*window).draw(space);
                    break;
                case (error):
                    space.setPosition(wid + j * 20, hei + i * 20);
                    (*window).draw(space);
                    break;
                case (razvil):
                    if (avto) {
                        way.setPosition(wid + j * 20, hei + i * 20);
                        (*window).draw(way);
                    }
                    else {
                        space.setPosition(wid + j * 20, hei + i * 20);
                        (*window).draw(space);
                    }
                    break;
                case put:
                    if (avto) {
                        way.setPosition(wid + j * 20, hei + i * 20);
                        (*window).draw(way);
                    }
                    else {

```

```

space.setPosition(wid + j * 20, hei + i * 20);
(*window).draw(space);
}
    break;
    case (player):
play.setPosition(wid + j * 20, hei + i * 20);
    (*window).draw(play);
    break;
    case (II):
III.setPosition(wid + j * 20, hei + i * 20);
(*window).draw(III);
break;
    }
    }
}
}
}

```

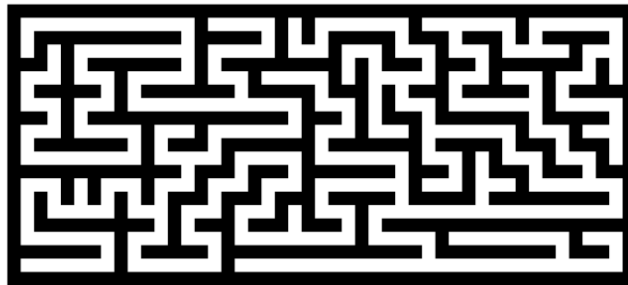


Рисунок 2 – Вывод лабиринта

Далее с помощью функции `mod_selection` пользователю предоставляется выбор (рисунок 3) :

Выход: выбор режима работы программы.

```

bool mod_selectino() {

RenderWindowwindow(VideoMode(600, 500), "Labirint");
RectangleShape rectangle1(Vector2f(400, 100));
RectangleShape rectangle2(Vector2f(560, 100));

    rectangle1.setFillColor(Color::White);
    rectangle1.setPosition(100, 150);
    rectangle1.setOutlineThickness(5);
    rectangle1.setOutlineColor(Color::White);

    rectangle2.setFillColor(Color::White);
    rectangle2.setPosition(20, 300);
    rectangle2.setOutlineThickness(5);
    rectangle2.setOutlineColor(Color::White);

    while (window.isOpen())
    {
window.clear(Color::White);
text.setString(L"Играть");
text.setPosition(150, 150);
text.setCharacterSize(70);
text.setFillColor(Color::Blue);

window.draw(rectangle1);
window.draw(text);

```

```

text.setString(L"Автоматически");
text.setPosition(30, 300);
text.setCharacterSize(60);
text.setFillColor(Color::Red);

window.draw(rectangle2);
window.draw(text);

window.display();

    Vector2i pos = Mouse::getPosition(window);
    Event event;
    while (window.pollEvent(event))
    {

if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboard::Escape))
    {
window.close();
    }
    if (rectangle1.getGlobalBounds().contains(pos.x, pos.y)) {
        rectangle1.setOutlineColor(Color::Black);
        if (event.key.code == Mouse::Left) {
input_time_II(&window);
            return TRUE;
        }
    }
    else {
        rectangle1.setOutlineColor(Color::White);
    }
    if (rectangle2.getGlobalBounds().contains(pos.x, pos.y)) {
        rectangle2.setOutlineColor(Color::Black);
        if (event.key.code == Mouse::Left) { return FALSE; }
    }
    else {
        rectangle2.setOutlineColor(Color::White);
    }
    }
}
}

```



АВТОМАТИЧЕСКИ

Рисунок 3 – Меню

1. «ИГРАТЬ» . Запускается игра. Суть игры заключается в том, что пользователь должен прийти к выходу быстрее чем бот.

1.1. В начале игрок вводит скорость движения бота по лабиринту (рисунок 4).

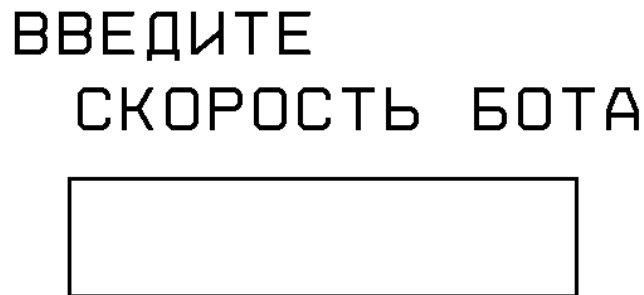


Рисунок 4 –Ввод скорости движения бота

Ввод осуществляется при помощи функции `input_time_II`. Программа считывает нажатую клавишу, если эта цифра, «.» или «,», то она записывается в массив, после чего выводится на экран.

Вход: window–окно windows.

Выход: введена скорость движения бота.

```
void input_time_II(RenderWindow* window) {  
  
    char size[7];  
    int i = -1;  
    bool vibor = TRUE;  
    RectangleShape rectangle1(Vector2f(400, 90));  
    rectangle1.setFillColor(Color::White);  
    rectangle1.setPosition(100, 250);  
    rectangle1.setOutlineThickness(3);  
    rectangle1.setOutlineColor(Color::Black);  
  
    (*window).clear(Color::White);  
    text.setString(L"Введите ");  
    text.setPosition(50, 100);  
    text.setCharacterSize(50);  
    text.setFillColor(Color::Red);  
    (*window).draw(text);  
    text.setString(L"скоростьбота");  
    text.setPosition(100, 160);  
    (*window).draw(text);  
    (*window).draw(rectangle1);  
    (*window).display();  
  
    while ((*window).isOpen())  
    {  
  
        Event event;  
        while ((*window).pollEvent(event))  
        {  
  
            if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboard::Escape))
```



```

        {
            (*window).close();}
    }
    if (Keyboard::isKeyPressed(Keyboard::Enter)) {
        int m = 10;
        double n = 0.1;

        for (int j = 0; j <= i; j++) {
            if (size[j] == '.' || size[j] == ',') { vibor = FALSE; }
            if (vibor) {
                time_II = (time_II * m) + (int(size[j]) - 48);
            }
            if (!vibor && size[j] != '.' && size[j] != ',') {
                time_II += (int(size[j]) - 48) * n;
                n *= 0.1;
            }
        }

        Sleep(100);
        (*window).close();
    }
    if ((event.type == Event::TextEntered && (event.key.code == '0' || event.key.code == '1' || event.key.code == '2' ||
event.key.code == '3' || event.key.code == '4' || event.key.code == '5' || event.key.code == '6' || event.key.code == '7' ||
event.key.code == '8' || event.key.code == '9' || event.key.code == '.' || event.key.code == ',')) ||
Keyboard::isKeyPressed(Keyboard::LShift)) {

        (*window).clear(Color::White);
        text.setString(L"Введите ");
        text.setPosition(50, 100);
        text.setCharacterSize(50);
        text.setFillColor(Color::Red);
        (*window).draw(text);
        text.setString(L"скорость бота");
        text.setPosition(100, 160);
        (*window).draw(text);
        (*window).draw(rectangle1);
        text.setCharacterSize(60);
        text.setFillColor(Color::Green);
        if (event.type == Event::TextEntered && (event.key.code == '0' || event.key.code == '1' || event.key.code == '2' ||
event.key.code == '3' || event.key.code == '4' || event.key.code == '5' || event.key.code == '6' || event.key.code == '7' ||
event.key.code == '8' || event.key.code == '9' || event.key.code == '.' || event.key.code == ',')) {
            i++;
            size[i] = event.key.code;
            for (int g = 0; g <= i; g++) {
                text.setString(size[g]);
                text.setPosition(200 + g * 35, 250);
                (*window).draw(text);
            }
        }
        if (Keyboard::isKeyPressed(Keyboard::LShift)) {

            size[i] = NULL;
            i--;
            for (int g = 0; g <= i; g++) {
                text.setString(size[g]);
                text.setPosition(200 + g * 35, 250);
                (*window).draw(text);
            }
        }
        (*window).display();
        Sleep(200);}
    }
}

```

1.2. Далее в функции play начинается игра от левого верхнего угла (рисунок 5). Игрок передвигается по лабиринту при помощи стрелочек. Функция считывает, нажата ли стрелочка, если да, то проверяется может ли игрок пойти в ту сторону или нет.

Бот в свою очередь совершает один ход в заданное время.

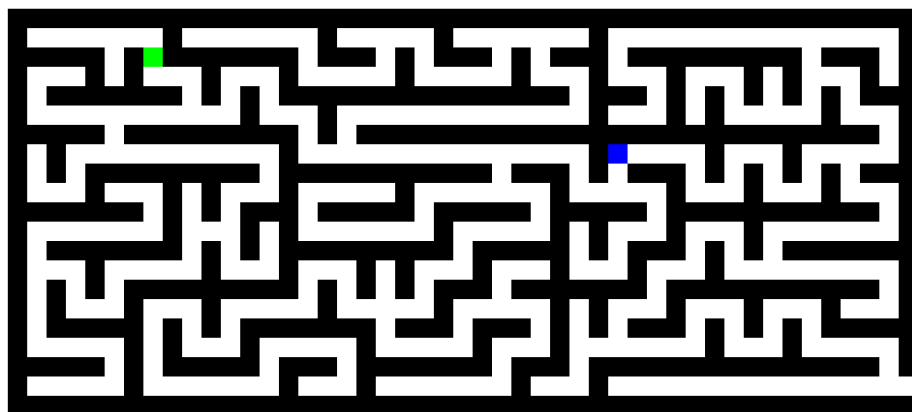


Рисунок 5 – Вывод лабиринта во время игры

Направление движения бота осуществляется при помощи функции `II_prohod`. Программа ищет в какой стороне от его позиции находится путь к выходу сгенерированный в алгоритме `poisk_puti`.

Вход: `height` – высота лабиринта, `width` – ширина лабиринта, `maze` – массив с лабиринтом, `window`–окно `windows`.

Выход: новоеместоположениебота.

```
void II_prohod(int** maze, RenderWindow* window, int height, int width) {
    stop = clock();
    if (((double)(stop - start) / CLOCKS_PER_SEC) >= time_II) {
        start = clock();
        if (maze[z][w + 1] == put || maze[z][w + 1] == razvil || maze[z][w + 1] == player || maze[z][w + 1] == II) // направо
        {
            maze[z][w] = error;
            w++;
            maze[z][w] = II;
        }

        if (maze[z - 1][w] == put || maze[z - 1][w] == razvil || maze[z - 1][w] == player || maze[z - 1][w] == II) // вверх
        {
            maze[z][w] = error;
            z--;

            maze[z][w] = II;
        }

        if (maze[z][w - 1] == put || maze[z][w - 1] == razvil || maze[z][w - 1] == player || maze[z][w - 1] == II) // налево
```

```

    {
        maze[z][w] = error;
        w--;

        maze[z][w] = II;
    }

    if (maze[z + 1][w] == put || maze[z + 1][w] == razvil || maze[z + 1][w] == player || maze[z + 1][w] == II) { // вниз

        maze[z][w] = error;
        z++;

        maze[z][w] = II;
    }
    (*window).clear(Color::White);

visual(maze, height, width, window, FALSE);

    (*window).display();
}
}
void play(int** maze, int height, int width, RenderWindow* window) {
    start = clock();
    start_pl = clock();

    RectangleShapeplay(Vector2f(20, 20));

    play.setFill(Color::Green);

    int x = 1, y = 1;
    bool win = TRUE;
    maze[y][x] = II;
    play.setPosition(wid + x * 20, hei + y * 20);

    while ((*window).isOpen() && win)
    {
        (*window).clear(Color::White);

visual(maze, height, width, window, FALSE);
        (*window).draw(play);
        (*window).display();
        II_proxod(maze, window, height, width);
        Event event;
        while ((*window).pollEvent(event))
        {
            II_proxod(maze, window, height, width);

            if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboard::Escape))
            {
                (*window).close();
            }
            if (Keyboard::isKeyPressed(Keyboard::Insert))
            {
                (*window).close();
            }
            return_menu();
        }
        if (Keyboard::isKeyPressed(Keyboard::Right)) {
            if (maze[y][x + 1] != wall) // направо
            {
                x++;
            }
            play.setPosition(wid + x * 20, hei + y * 20);
        }
    }
    if (Keyboard::isKeyPressed(Keyboard::Up)) {

```

```

        if (maze[y - 1][x] != wall) // вверх
        {
            y--;
play.setPosition(wid + x * 20, hei + y * 20);
        }
    }
    if (Keyboard::isKeyPressed(Keyboard::Left)) {
        if (maze[y][x - 1] != wall) // налево
        {
            x--;
play.setPosition(wid + x * 20, hei + y * 20);
        }
    }
    if (Keyboard::isKeyPressed(Keyboard::Down)) {
        if (maze[y + 1][x] != wall) { // вниз
            y++;
play.setPosition(wid + x * 20, hei + y * 20);
        }
    }
    (*window).clear(Color::White);

visual(maze, height, width, window, FALSE);
    (*window).draw(play);
    (*window).display();
    if (y == height - 2 && x == width - 2) {
end_pl = clock();
        (*window).close();
RenderWindowokno(VideoMode(400, 400), "Labirint");

        okno.clear(Color::White);
        text.setString(L"!Победа!");
        text.setPosition(40, 150);
        text.setCharacterSize(60);
        text.setFillColor(Color::Green);
        okno.draw(text);
        okno.display();
        win = FALSE;
        Sleep(3000);
        okno.close();

        vnes_lider(((int)(end_pl - start_pl) / CLOCKS_PER_SEC));
    }
    if (maze[height - 2][width - 2] == П) {
        (*window).close();
RenderWindowokno(VideoMode(400, 400), "Labirint");

        okno.clear(Color::White);
        text.setString(L"Проигрыш");
        text.setPosition(40, 150);
        text.setCharacterSize(60);
        text.setFillColor(Color::Red);
        okno.draw(text);
        okno.display();
        win = FALSE;
        Sleep(5000);
        okno.close();
        return_menu();
    }

}

}

```

1.3. В результате игры возможны 2 исхода:

1.3.1. Победил бот. На экран выводится надпись «Проигрыш» (рисунок 6).

ПРОИГРЫШ

Рисунок 6 – Вывод надписи «Проигрыш»

1.3.2. Победил игрок. На экран выводится надпись «Победа» (рисунок 7).

! ПОБЕДА !

Рисунок 7 – Вывод надписи «!Победа!»

После чего игрок вводит своё имя в таблицу победителей. После нажатия на клавишу «ENTER» на экран выведется таблица победителей (рисунок 8).

ПОБЕДИТЕЛИ :

YASYA 13

GERALD 6

Рисунок 8 – Вывод таблицы победителей

Ввод имени, запись его и времени прохождения лабиринта в файл, считывание таблицы победителей и вывод её на экран осуществляет функция `vnes_lider`.

Вход: `time` – время прохождения игроком лабиринта.

Выход: имя победителя и время его прохождения заносятся в файл.

```
void vnes_lider(int time) {
```

```

RenderWindowwindow(VideoMode(600, 600), "Labirint");
char lider[10];
int i = -1;

window.clear(Color::White);
text.setString(L"Введите имя:");
text.setPosition(20, 20);
text.setCharacterSize(60);
text.setFillColor(Color::Red);
window.draw(text);
window.display();

text.setCharacterSize(80);
text.setFillColor(Color::Blue);
FILE* file;
file = fopen("D:/ЛОВЗ/Лабиринткурсач/Лидеры.txt", "a");
String string;
char mas[10];

while (window.isOpen())
{
    Event event;
    while (window.pollEvent(event))
    {
        if (event.type == sf::Event::Closed)
        {
            window.close();
        }
        if (Keyboard::isKeyPressed(Keyboard::Enter)) {
            for (int a = 0; a <= i; a++) {
                fprintf(file, "%c", lider[a]);
            }

            fprintf(file, "%c", ' ');
            fprintf(file, "%d", time);

            fprintf(file, "\n");
            fclose(file);
            window.clear(Color::White);
            text.setString(L"Победители : ");
            text.setPosition(20, 20);
            text.setCharacterSize(60);
            text.setFillColor(Color::Red);
            window.draw(text);
            file = fopen("D:/ЛОВЗ/Лабиринткурсач/Лидеры.txt", "r+");
            int peremeshenie_po_y = 0;
            while (fgets(mas, 10, file)) {

                text.setString(mas);
                text.setPosition(20, 100 + peremeshenie_po_y);
                text.setCharacterSize(40);
                text.setFillColor(Color::Blue);
                window.draw(text);
                peremeshenie_po_y += 50;
            }
            window.display();

            Sleep(5000);
            return_menu();
        }
        if ((event.type == Event::TextEntered || Keyboard::isKeyPressed(Keyboard::LShift))) {
            window.clear(Color::White);
            text.setString(L"Введите имя:");
            text.setPosition(20, 20);

```

2. «Автоматически» . При помощи функции `visual` происходит вывод пути к выходу из лабиринта на экран (рисунок 9).



На каждом этапе можно начать программу заново при помощи функции `return_menu`.

```
void return_menu() {
    bool mod;
    int height = 0, width = 0;

    input_size(&height, &width);

    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 * 23, height * 25), "Labirint");

    while (window.isOpen())
    {

        Event event;
        while (window.pollEvent(event))
        {
            if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboard::Escape))
            {
                window.close();
            }
        }

        window.clear(Color::White);

        visual(maze, height, width, &window, TRUE);

        window.display();

        Sleep(2000);

        mod = mod_selectino();
        if (mod) {
            poisk_puti(maze, height, width);
            play(maze, height, width, &window);
        }
        else {
            poisk_puti(maze, height, width);

            window.clear(Color::White);

            visual(maze, height, width, &window, TRUE);

            window.display();
            Sleep(5000);
            return_menu();
        }
    }
}
```

5 Отладка

В качестве среды разработки была выбрана программа Microsoft Visual Studio 2022. Программа обладает всеми необходимыми средствами для разработки и отладки программы. Для отладки использовались несколько возможностей Visual Studio:

1. Точки останова - это места в коде, которые позволяют остановить программу и включить отладку. Они позволяют изучить поведение кода и его функции, чтобы попытаться определить ошибку (рисунок 10-13).
2. Трассировка (trouseroute) — процедура получения информации о маршрутизаторах (узлах), через которые проходят пакеты к интересующему компьютеру. Позволяет обнаружить ошибки маршрутизации, а также то, к какому первичному провайдеру подключён хостинг-провайдер (рисунок 10-13) .

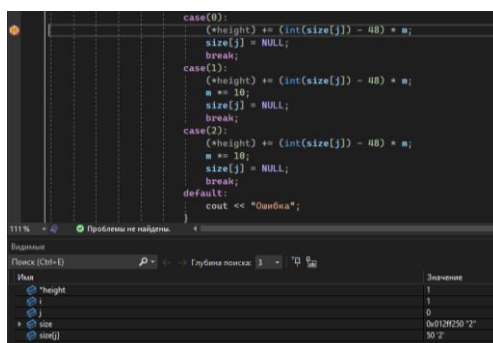


Рисунок 10 – Протокол трассировки программы

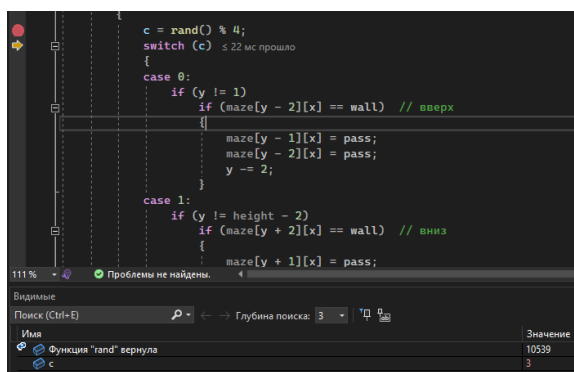


Рисунок 11 – Протокол трассировки программы

6 Тестирование

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы.

Далее будет продемонстрирован результат тестирования программы при вводе пользователем разных размеров лабиринта (рисунок 14-16).



Рисунок 14 – Тестирование при вводе размера лабиринта 5*5

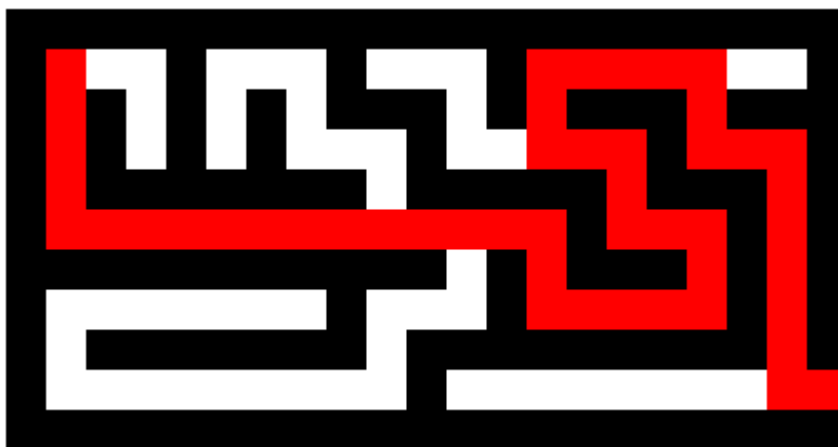


Рисунок 15 – Тестирование при вводе размера лабиринта 11*21

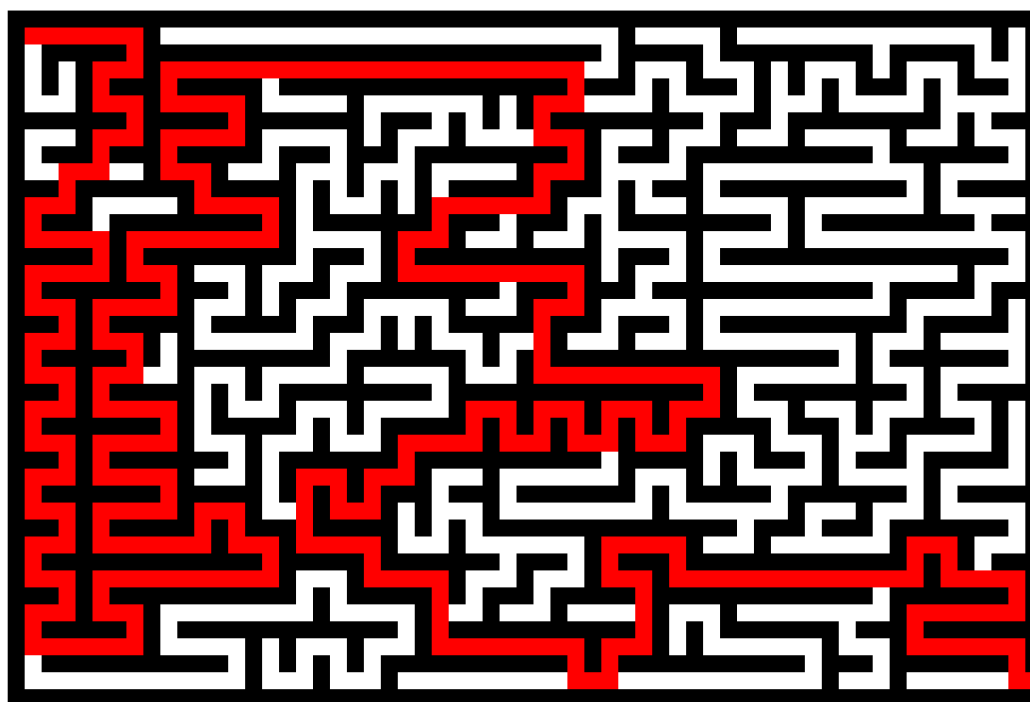


Рисунок 16 – Тестирование при вводе размера лабиринта 40*60

Таблица 1 – Описание поведения программы при тестировании

Описание теста	Ожидаемы результат	Полученный результат
Запуск программы	Вывод окна с вводом высоты и ширины	Верно
Вывод лабиринта	Вывод в окно сгенерированного лабиринта с заданными параметрами	Верно
Поиск пути	Вывод в окно правильного пути до выхода	Верно
Ввод скорости бота	Вывод окна с вводом скорости бота, корректная запись введённых данных в переменную	Верно

Продолжение таблицы 1

Описание теста	Ожидаемы результат	Полученный результат
Игра	Постоянное обновление лабиринта, корректное считывание стрелочек и движение игрока, правильная работа бота	Верно
Занесение имени и времени в таблицу лидеров	Правильный ввод имени, занесение в файл, вывод из файла таблицы лидеров	Верно

Заключение

При выполнении данной курсовой работы были получены навыки разработки игр. Была освоена работа с графической библиотекой SFML. Также были получены основные навыки отладки и тестирования программ и программирования в среде VisualStudio 2019 на языках Си/C++.

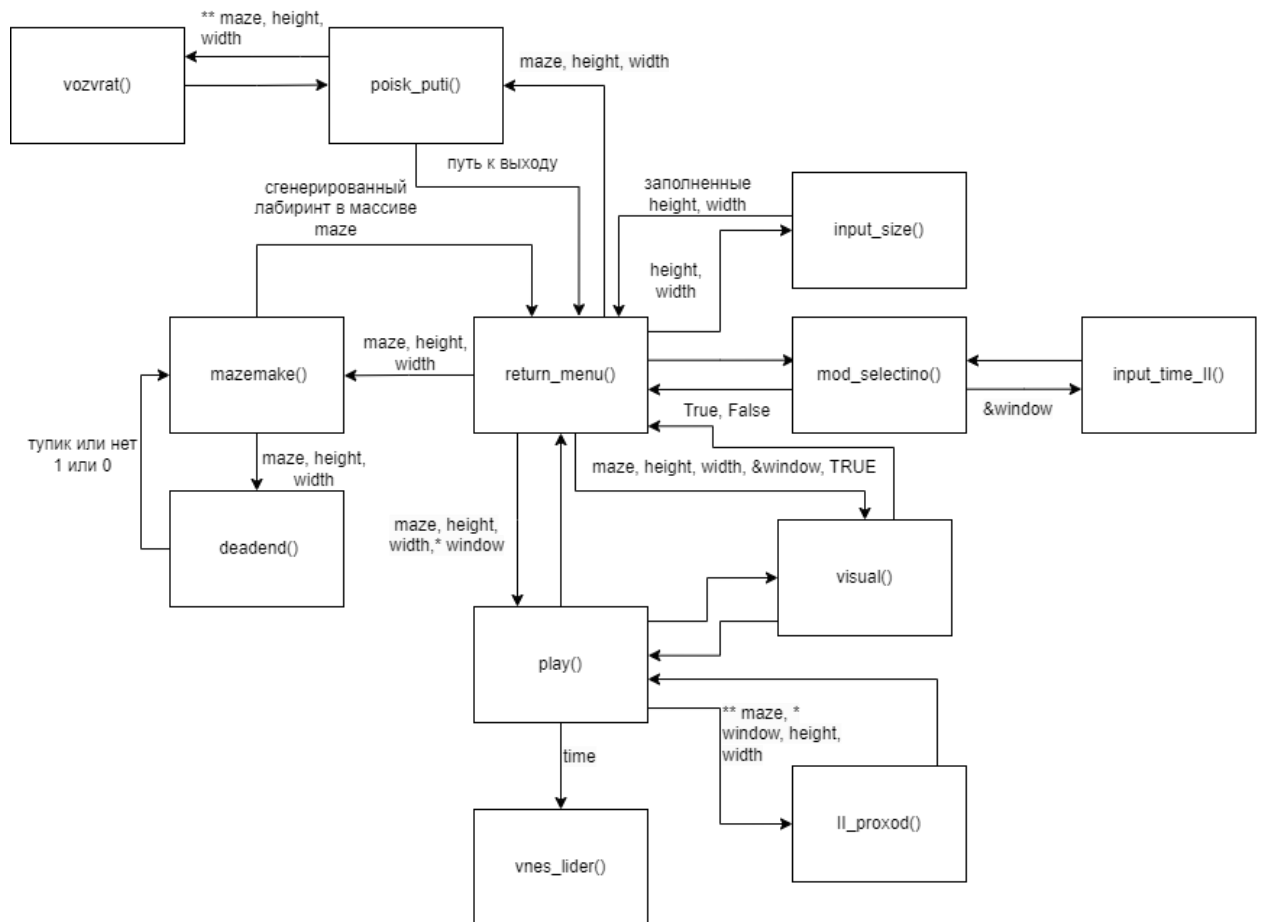
В рамках курсовой работы была разработана программа «поиск путей в лабиринте».

В будущем программу можно улучшить, добавив новые функции и улучшив графику.

Список используемых источников

1. Березин Б.И. Березин С.Б.. Начальный курс С и С++ 1996
2. Maxim Bardier SFML Blueprints 2015

Приложение А (Схема взаимодействия функций)



ПриложениеБ (Листинги программы)

```
#define _CRT_SECURE_NO_WARNINGS
#include <SFML/Graphics.hpp>
#include<windows.h>
#include <iostream>
#include <cstdlib>
#include <ctime>
#include<time.h>

using namespace std;
using namespace sf;

const int wall = 0, pass = 1, put = 2, razvil = 3, error = 4, player = 5, II = 6;
int x, y, w = 1, z = 1, wid, hei;
double time_II = 0;
clock_t start, stop, start_pl, end_pl;

Font font;
Text text;
void input_size(int* height, int* width);
bool deadend(int** maze, int height, int width);
void mazemake(int** maze, int height, int width);
void visual(int** maze, int height, int width, RenderWindow* window, bool avto);
void vnes_lider(int time);
void vozvrat(int** maze, int height, int width);
void poisk_puti(int** maze, int height, int width);
void II_proxod(int** maze, RenderWindow* window, int height, int width);
void play(int** maze, int height, int width, RenderWindow* window);
void input_time_II(RenderWindow* window);
bool mod_selectino();
void return_menu();
int main()
{
    srand(time(NULL));
    font.loadFromFile("D:/font/Disket-Mono-Regular.ttf");
    text.setFont(font);
    return_menu();
    return 0;
}
void input_size(int* height, int* width) {
```

```

RenderWindowwindow(VideoMode(800, 400), "Labirint");
    char size[4];
    bool vibor = TRUE;
    int i = -1, enter = 0;
window.clear(Color::White);
text.setString(L"Введите размеры:");
text.setPosition(40, 90);
text.setCharacterSize(60);
text.setFillColor(Color::Red);
window.draw(text);
text.setString(L"Высота: ");
text.setPosition(40, 200);
text.setCharacterSize(60);
text.setFillColor(Color::Blue);
window.draw(text);
window.display();
    while (window.isOpen() && enter != 2)
    {

        Event event;
        while (window.pollEvent(event) && enter != 2)
        {

            if (event.type == sf::Event::Closed || enter == 2)
            {
window.close();
            }
            if (Keyboard::isKeyPressed(Keyboard::Enter)) {
                int m = 1;
                if (vibor) {
                    for (int j = i; j >= 0; j--) {
                        switch (j) {
case(0):
                            (*height) += (int(size[j]) - 48) * m;
                            size[j] = NULL;
                            break;
case(1):
                            (*height) += (int(size[j]) - 48) * m;
                            m *= 10;
                            size[j] = NULL;
                            break;

```

```

case(2):
    (*height) += (int(size[j]) - 48) * m;
    m *= 10;
    size[j] = NULL;
    break;
default:
cout<< "Ошибка";
    }
    }
    if ((*height) % 2 == 0) {
        (*height)++;
    }
vibor = FALSE;
    enter++;

i = -1;
Sleep(100);

    }
    else {
        for (int j = i; j >= 0; j--) {
            switch (j) {
case(0):
                (*width) += (int(size[j]) - 48) * m;
                break;
case(1):
                (*width) += (int(size[j]) - 48) * m;
                m *= 10;
                break;
case(2):
                (*width) += (int(size[j]) - 48) * m;
                m *= 10;
                break;
default:
cout<< "Ошибка";
                }
            }
            if ((*width) % 2 == 0) {
                (*width)++;
            }
            enter++;

```

```

    }
    Sleep(100);
    }
    if ((event.type == Event::TextEntered&& (event.key.code == '0' || event.key.code == '1' || event.key.code ==
'2' || event.key.code == '3' || event.key.code == '4' || event.key.code == '5' || event.key.code == '6' || event.key.code ==
'7' || event.key.code == '8' || event.key.code == '9')) || Keyboard::isKeyPressed(Keyboard::LShift) || i == -1) {

window.clear(Color::White);
text.setString(L"Введитеразмеры:");
text.setPosition(40, 90);
text.setCharacterSize(60);
text.setFillColor(Color::Red);
window.draw(text);
    if (vibor) {
text.setString(L"Высота: ");
text.setPosition(40, 200);
text.setCharacterSize(60);
text.setFillColor(Color::Blue);
window.draw(text);
    }
    else {
text.setString(L"Ширина:");
text.setPosition(40, 200);
text.setCharacterSize(60);
text.setFillColor(Color::Blue);
window.draw(text);
    }

text.setCharacterSize(70);
text.setFillColor(Color::Green);
    if (event.type == Event::TextEntered&& (event.key.code == '0' || event.key.code == '1' || event.key.code
== '2' || event.key.code == '3' || event.key.code == '4' || event.key.code == '5' || event.key.code == '6' || event.key.code
== '7' || event.key.code == '8' || event.key.code == '9')) {
i++;
    size[i] = event.key.code;
    for (int g = 0; g <= i; g++) {
text.setString(size[g]);
text.setPosition(330 + g * 50, 190);
window.draw(text);
    }

```

```

    }
    if (Keyboard::isKeyPressed(Keyboard::LShift)) {

        size[i] = NULL;
i--;

        for (int g = 0; g <= i; g++) {
text.setString(size[g]);
text.setPosition(330 + g * 50, 190);
window.draw(text);
        }
    }
window.display();
Sleep(200);
    }
}
}

bool deadend(int** maze, 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;
    if (a == 4)
        return 1;
    else
        return 0;
}

void mazemake(int** maze, int height, int width)
{
    int 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 (a < 10000)
    {
        maze[y][x] = pass;
        a++;
        while (1)
        {
            c = rand() % 4;
            switch (c)
            {
                case 0:
                    if (y != 1)
                        if (maze[y - 2][x] == wall) // вверх
                        {
                            maze[y - 1][x] = pass;
                            maze[y - 2][x] = pass;
                            y -= 2;
                        }
                case 1:
                    if (y != height - 2)

```

```

        if (maze[y + 2][x] == wall) // вниз
        {
maze[y + 1][x] = pass;
maze[y + 2][x] = pass;
        y += 2;
        }
case 2:
    if (x != 1)
        if (maze[y][x - 2] == wall) // налево
        {
            maze[y][x - 1] = pass;
            maze[y][x - 2] = pass;
            x -= 2;
        }
case 3:
    if (x != width - 2)
        if (maze[y][x + 2] == wall) // направо
        {
            maze[y][x + 1] = pass;
            maze[y][x + 2] = pass;
            x += 2;
        }
    }
    if (deadend(maze, height, width)) {
        break;
    }
}
if (deadend(maze, height, width))
do
{
    x = 2 * (rand() % ((width - 1) / 2)) + 1;
    y = 2 * (rand() % ((height - 1) / 2)) + 1;
} while (maze[y][x] != pass);
}
}
///-----|
/// изображение результата с помощью SFML графики.
///-----:
void visual(int** maze, int height, int width, RenderWindow* window, bool avto)
{
    wid = (width * 23) / 15;

```

```

hei = (height * 25) / 10;

RectangleShapewal(Vector2f(20, 20));
RectangleShapeway(Vector2f(20, 20));
RectangleShapespace(Vector2f(20, 20));
RectangleShapeplay(Vector2f(20, 20));
RectangleShapeIII(Vector2f(20, 20));

way.setFillColor(Color::Red);
wal.setFillColor(Color::Black);
space.setFillColor(Color::White);
play.setFillColor(Color::Green);
III.setFillColor(Color::Blue);
for (int i = 0; i < height; i++)
{
    for (int j = 0; j < width; j++) {
        switch (maze[i][j])
        {
            case (wall):
wal.setPosition(wid + j * 20, hei + i * 20);
                (*window).draw(wal);
                break;
            case (pass):
space.setPosition(wid + j * 20, hei + i * 20);
                (*window).draw(space);
                break;
            case (error):
space.setPosition(wid + j * 20, hei + i * 20);
                (*window).draw(space);
                break;
            case (razvil):
                if (avto) {
wal.setPosition(wid + j * 20, hei + i * 20);
                    (*window).draw(way);
                }
                else {
space.setPosition(wid + j * 20, hei + i * 20);
                    (*window).draw(space);
                }
                break;
            case put:

```

```

        if (avto) {
way.setPosition(wid + j * 20, hei + i * 20);
        (*window).draw(way);
        }
        else {
space.setPosition(wid + j * 20, hei + i * 20);
        (*window).draw(space);
        }
        break;
        case (player):
play.setPosition(wid + j * 20, hei + i * 20);
        (*window).draw(play);
        break;
        case (II):
III.setPosition(wid + j * 20, hei + i * 20);
        (*window).draw(III);
        break;
        }
    }
}
}
void vozvrat(int** maze, int height, int width) { // возвратоттупикадоразвилки

```

```

while (maze[y][x] != razvil) {
    if (maze[y - 1][x] == put || maze[y - 1][x] == razvil) // вверх
    {
        maze[y][x] = error;
        y--;
    }
    else if (maze[y + 1][x] == put || maze[y + 1][x] == razvil) { // вниз
        maze[y][x] = error;
        y++;
    }
    else if (maze[y][x - 1] == put || maze[y][x - 1] == razvil) // налево
    {
        maze[y][x] = error;
        x--;
    }
    else if (maze[y][x + 1] == put || maze[y][x + 1] == razvil) // направо
    {
        maze[y][x] = error;
    }
}

```

```

        x++;
    }
}
}

void poisk_puti(int** maze, int height, int width) {
    int razvilka = 0, save_y = 0, save_x = 0;
    x = 1;
    y = 1;
    while ((y != (height - 2)) || (x != (width - 2))) {
save_y = 0;
save_x = 0;
razvilka = 0;
        if (maze[y - 1][x] == pass) // вверх
        {
razvilka++;
save_y = -1;
save_x = 0;
        }
        if (maze[y + 1][x] == pass) { // вниз

razvilka++;
save_y = 1;
save_x = 0;
        }
        if (maze[y][x - 1] == pass) // налево
        {
razvilka++;
save_x = -1;
save_y = 0;
        }
        if (maze[y][x + 1] == pass) // направо
        {
razvilka++;
save_x = 1;
save_y = 0;
        }
        if (razvilka == 1) {
            maze[y][x] = put;
            x += save_x;
            y += save_y;
        }
    }
}

```

```

else if (razvilka > 1) {
    maze[y][x] = razvil;
    x += save_x;
    y += save_y;
}
else {
    maze[y][x] = put;
    vozvrat(maze, height, width);
}
}

maze[height - 2][width - 2] = put;
maze[height - 2][width - 1] = put;

}

void II_proxod(int** maze, RenderWindow* window, int height, int width) {
    stop = clock();
    if (((double)(stop - start) / CLOCKS_PER_SEC) >= time_II) {
        start = clock();
        if (maze[z][w + 1] == put || maze[z][w + 1] == razvil || maze[z][w + 1] == player || maze[z][w + 1] == II) //
направо
        {
            maze[z][w] = error;
            w++;
            maze[z][w] = II;
        }
        if (maze[z - 1][w] == put || maze[z - 1][w] == razvil || maze[z - 1][w] == player || maze[z - 1][w] == II) //
вверх
        {
            maze[z][w] = error;
            z--;
            maze[z][w] = II;
        }
        if (maze[z][w - 1] == put || maze[z][w - 1] == razvil || maze[z][w - 1] == player || maze[z][w - 1] == II) //
налево
        {
            maze[z][w] = error;
            w--;
            maze[z][w] = II;
        }
    }
}

```

```

        if (maze[z + 1][w] == put || maze[z + 1][w] == razvil || maze[z + 1][w] == player || maze[z + 1][w] == II) { //
ВНИЗ

            maze[z][w] = error;
            z++;
            maze[z][w] = II;
        }
        (*window).clear(Color::White);
visual(maze, height, width, window, FALSE);
        (*window).display();
    }
}

void play(int** maze, int height, int width, RenderWindow* window) {
    start = clock();
    start_pl = clock();

    RectangleShapeplay(Vector2f(20, 20));
    play.setFillColor(Color::Green);
    int x = 1, y = 1;
    bool win = TRUE;
    maze[y][x] = II;
    play.setPosition(wid + x * 20, hei + y * 20);

    while ((*window).isOpen() && win)
    {
        (*window).clear(Color::White);
visual(maze, height, width, window, FALSE);
        (*window).draw(play);
        (*window).display();

    II_proxod(maze, window, height, width);
        Event event;
        while ((*window).pollEvent(event))
        {
    II_proxod(maze, window, height, width);

            if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboard::Escape))
            {
                (*window).close();
            }
            if (Keyboard::isKeyPressed(Keyboard::Insert))

```

```

        {
            (*window).close();
return_menu();
        }
        if (Keyboard::isKeyPressed(Keyboard::Right)) {
            if (maze[y][x + 1] != wall) // направо
            {
                x++;
play.setPosition(wid + x * 20, hei + y * 20);
            }
        }
        if (Keyboard::isKeyPressed(Keyboard::Up)) {
            if (maze[y - 1][x] != wall) // вверх
            {
                y--;
play.setPosition(wid + x * 20, hei + y * 20);
            }
        }
        if (Keyboard::isKeyPressed(Keyboard::Left)) {
            if (maze[y][x - 1] != wall) // налево
            {
                x--;
play.setPosition(wid + x * 20, hei + y * 20);
            }
        }
        if (Keyboard::isKeyPressed(Keyboard::Down)) {
            if (maze[y + 1][x] != wall) { // вниз
                y++;
play.setPosition(wid + x * 20, hei + y * 20);
            }
        }
        (*window).clear(Color::White);

visual(maze, height, width, window, FALSE);
        (*window).draw(play);
        (*window).display();
    }
    if (y == height - 2 && x == width - 2) {
end_pl = clock();
        (*window).close();
RenderWindowokno(VideoMode(400, 400), "Labirint");

```



```

okno.clear(Color::White);
text.setString(L"!Победа!");
text.setPosition(40, 150);
text.setCharacterSize(60);
text.setFillColor(Color::Green);
okno.draw(text);
okno.display();
    win = FALSE;
Sleep(3000);
okno.close();

```

```

vnes_lider(((int)(end_pl - start_pl) / CLOCKS_PER_SEC));
    }
    if (maze[height - 2][width - 2] == II) {
        (*window).close();
RenderWindowokno(VideoMode(400, 400), "Labirint");

```

```

okno.clear(Color::White);
text.setString(L"Проигрыш");
text.setPosition(40, 150);
text.setCharacterSize(60);
text.setFillColor(Color::Red);
okno.draw(text);
okno.display();
    win = FALSE;
Sleep(5000);
okno.close();
return_menu();
    }
}
}

```

```

void input_time_II(RenderWindow* window) {

    char size[7];
    int i = -1;
    bool vibor = TRUE;
RectangleShape rectangle1(Vector2f(400, 90));
    rectangle1.setFillColor(Color::White);
    rectangle1.setPosition(100, 250);

```

```

rectangle1.setOutlineThickness(3);
rectangle1.setOutlineColor(Color::Black);

(*window).clear(Color::White);
text.setString(L"Введите ");
text.setPosition(50, 100);
text.setCharacterSize(50);
text.setFillColor(Color::Red);
(*window).draw(text);
text.setString(L"скоростьбота");
text.setPosition(100, 160);
(*window).draw(text);
(*window).draw(rectangle1);
(*window).display();
while ((*window).isOpen())
{
    Event event;
    while ((*window).pollEvent(event))
    {

        if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboard::Escape))
        {
            (*window).close();
        }
    }
    if (Keyboard::isKeyPressed(Keyboard::Enter)) {
        int m = 10;
        double n = 0.1;

        for (int j = 0; j <= i; j++) {
            if (size[j] == '.' || size[j] == ',') { vibor = FALSE; }
            if (vibor) {
time_II = (time_II * m) + (int(size[j]) - 48);
            }
            if (!vibor && size[j] != '.' && size[j] != ',') {
time_II += (int(size[j]) - 48) * n;
                n *= 0.1;
            }
        }
    }
    Sleep(100);
    (*window).close();
}

```

```

    }

    if ((event.type == Event::TextEntered&& (event.key.code == '0' || event.key.code == '1' || event.key.code == '2'
|| event.key.code == '3' || event.key.code == '4' || event.key.code == '5' || event.key.code == '6' || event.key.code == '7'
|| event.key.code == '8' || event.key.code == '9' || event.key.code == '.' || event.key.code == ',')) ||
Keyboard::isKeyPressed(Keyboard::LShift)) {

        (*window).clear(Color::White);
text.setString(L"Введите ");
text.setPosition(50, 100);
text.setCharacterSize(50);
text.setFillColor(Color::Red);

        (*window).draw(text);
text.setString(L"скоростьбота");
text.setPosition(100, 160);

        (*window).draw(text);

        (*window).draw(rectangle1);
text.setCharacterSize(60);
text.setFillColor(Color::Green);

        if (event.type == Event::TextEntered&& (event.key.code == '0' || event.key.code == '1' || event.key.code ==
'2' || event.key.code == '3' || event.key.code == '4' || event.key.code == '5' || event.key.code == '6' || event.key.code ==
'7' || event.key.code == '8' || event.key.code == '9' || event.key.code == '.' || event.key.code == ',')) {
i++;

            size[i] = event.key.code;
            for (int g = 0; g <= i; g++) {
text.setString(size[g]);
text.setPosition(200 + g * 35, 250);

                (*window).draw(text);
            }
        }

        if (Keyboard::isKeyPressed(Keyboard::LShift)) {

            size[i] = NULL;
i--;

            for (int g = 0; g <= i; g++) {
text.setString(size[g]);
text.setPosition(200 + g * 35, 250);

                (*window).draw(text);
            }
        }

        (*window).display();
Sleep(200);

```

```

    }
}
}
bool mod_selectino() {
RenderWindowwindow(VideoMode(600, 500), "Labirint");
RectangleShape rectangle1(Vector2f(400, 100));
RectangleShape rectangle2(Vector2f(560, 100));

rectangle1.setFillColor(Color::White);
rectangle1.setPosition(100, 150);
rectangle1.setOutlineThickness(5);
rectangle1.setOutlineColor(Color::White);

rectangle2.setFillColor(Color::White);
rectangle2.setPosition(20, 300);
rectangle2.setOutlineThickness(5);
rectangle2.setOutlineColor(Color::White);

while (window.isOpen())
{
window.clear(Color::White);
text.setString(L"Играть");
text.setPosition(150, 150);
text.setCharacterSize(70);
text.setFillColor(Color::Blue);

window.draw(rectangle1);
window.draw(text);
text.setString(L"Автоматически");
text.setPosition(30, 300);
text.setCharacterSize(60);
text.setFillColor(Color::Red);
window.draw(rectangle2);
window.draw(text);
window.display();

Vector2i pos = Mouse::getPosition(window);
Event event;
while (window.pollEvent(event))
{
if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboard::Escape))
{

```

```

window.close();
    }
    if (rectangle1.getGlobalBounds().contains(pos.x, pos.y)) {
        rectangle1.setOutlineColor(Color::Black);
        if (event.key.code == Mouse::Left) {
input_time_II(&window);
            return TRUE;
        }
    }
    else {
        rectangle1.setOutlineColor(Color::White);
    }
    if (rectangle2.getGlobalBounds().contains(pos.x, pos.y)) {
        rectangle2.setOutlineColor(Color::Black);
        if (event.key.code == Mouse::Left) { return FALSE; }
    }
    else {
        rectangle2.setOutlineColor(Color::White);
    }
}
}
}

void return_menu() {
    bool mod;
    int height = 0, width = 0;
input_size(&height, &width);
    int** maze = new int* [height];
    for (int i = 0; i < height; i++) {
        maze[i] = new int[width];
    }
    mazemake(maze, height, width);
RenderWindowwindow(VideoMode(width * 23, height * 25), "Labirint");
    while (window.isOpen())
    {
        Event event;
        while (window.pollEvent(event))
        {
            if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboard::Escape))
            {
window.close();
            }

```

```

    }
    window.clear(Color::White);
    visual(maze, height, width, &window, TRUE);
    window.display();
    Sleep(2000);
    mod = mod_selectino();
    if (mod) {
    poisk_puti(maze, height, width);
    play(maze, height, width, &window);
    }
    else {
    poisk_puti(maze, height, width);
    window.clear(Color::White);
    visual(maze, height, width, &window, TRUE);
    window.display();
    Sleep(5000);
    return_menu();
    }
    }
}

void vnes_lider(int time) {

RenderWindowwindow(VideoMode(600, 600), "Labirint");
    char lider[10];
    int i = -1;

    window.clear(Color::White);
    text.setString(L"Введите имя:");
    text.setPosition(20, 20);
    text.setCharacterSize(60);
    text.setFillColor(Color::Red);
    window.draw(text);
    window.display();

    text.setCharacterSize(80);
    text.setFillColor(Color::Blue);
    FILE* file;
    file = fopen("D:/ЛОВЗ/Лабиринткурсач/Лидеры.txt", "a");
    String string;
    char mas[10];
    while (window.isOpen())

```

```

{
    Event event;
    while (window.pollEvent(event))
    {
        if (event.type == sf::Event::Closed)
        {
            window.close();
        }
        if (Keyboard::isKeyPressed(Keyboard::Enter)) {
            for (int a = 0; a <= i; a++) {
                fprintf(file, "%c", lider[a]);
            }

            fprintf(file, "%c", ' ');
            fprintf(file, "%d", time);

            fprintf(file, "\n");
            fclose(file);
            window.clear(Color::White);
            text.setString(L"Победители : ");
            text.setPosition(20, 20);
            text.setCharacterSize(60);
            text.setFillColor(Color::Red);
            window.draw(text);

            file = fopen("D:/ЛОВЗ/Лабиринткурсач/Лидеры.txt", "r+");
            int peremeshenie_po_y = 0;
            while (fgets(mas, 10, file)) {

                text.setString(mas);
                text.setPosition(20, 100 + peremeshenie_po_y);
                text.setCharacterSize(40);
                text.setFillColor(Color::Blue);
                window.draw(text);
                peremeshenie_po_y += 50;
            }
            window.display();
            Sleep(5000);
            return_menu();
        }
        if ((event.type == Event::TextEntered || Keyboard::isKeyPressed(Keyboard::LShift))) {
            window.clear(Color::White);

```

```

text.setString(L"Введите имя:");
text.setPosition(20, 20);
text.setCharacterSize(60);
text.setFillColor(Color::Red);
window.draw(text);
text.setCharacterSize(80);
text.setFillColor(Color::Blue);
    if (event.type == Event::TextEntered) {
i++;
lider[i] = event.key.code;
        for (int g = 0; g <= i; g++) {
text.setString(lider[g]);
text.setPosition(20 + g * 50, 200);
window.draw(text);
        }
    }
    if (Keyboard::isKeyPressed(Keyboard::LShift)) {

lider[i] = NULL;
i--;
printf("-%d", i);
        for (int g = 0; g <= i; g++) {
text.setString(lider[g]);
text.setPosition(20 + g * 50, 200);
window.draw(text);
        }
    }
window.display();
Sleep(200);
    }
}
}
}

```