Министерство науки и высшего образования Российской Федерации
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”»

27.12.22
отлично
фио

Выполнил:
студент группы 21ВВ1
Кирьянов В. Е.
Принял:
к.т.н., доцент Юрова О.В.

Пенза 2022

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 Кирьянову Владиславу Евгеньевичу Группа 21ВВ1

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 вычислений (на небольшом участке работы алгоритма);
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;
Результаты 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

Дата выдачи задания " 10 " сентября

Дата защиты проекта " ____ " _____

Руководитель Юрова О.В.

Задание получил " 10 " сентября 2022 г.

Студент Кирьянов Владислав Евгеньевич

Содержание

Реферат

Отчет 41 стр., 16 рисунков.

АЛГОРИТМ ГЕНЕРАЦИИ ЛАБИРИНТА, АЛГОРИТМ ПРОХОЖДЕНИЯ ЛАБИРИНТА.

Цель исследования – разработка игрового агента, способного проходить различные лабиринты и алгоритма генерации лабиринта.

В работе рассмотрен алгоритм генерации случайного лабиринта и алгоритм псевдо-обхода лабиринта игровым агентом.
При помощи последнего можно проверить достижимость выхода из лабиринта.

Введение

Алгоритм прохождения лабиринта

Алгоритм прохождения лабиринта игровым агентом можно назвать псевдо-обходом в глубину. В этом алгоритме тот же принцип, агент идёт по комнатам (вершинам) до тех пор, пока не упрётся в тупик, и если там нет выхода из лабиринта, он вернётся к ближайшему возможному пути и продолжит проходить его до тех пор, пока не найдёт выход. Однако, он не использует граф, то есть он не основан на обходе графа, он основан на обходе матрицы, представляющей лабиринт.

Алгоритм генерации лабиринта

Он схож с алгоритмом прохождения лабиринта, за исключением того, что агент-генератор не ищет выход из лабиринта, сгенерировав все возможные пути лабиринта, он возвращается в начальную позицию откуда началась генерация.

1. Постановка задачи

Исходный лабиринт в программе должен задаваться матрицей, причём при генерации данных должны быть предусмотрены граничные условия.

Как должна работать программа: пользователь выбирает «сгенерировать случайный лабиринт» или «загрузить его из файла», если он был сохранён ранее. После выбора «случайной генерации» пользователь задаёт количество вершин для задания параметров лабиринта. Далее пользователю должна предоставляться возможность протестировать прохождение лабиринта самостоятельно и возможность наблюдения за работой алгоритма псевдо-обхода лабиринта игровым агентом. Также пользователь может сохранять через меню, выбранный им лабиринт в файл. Необходимо предусмотреть различные исходы работы алгоритмов, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом №31.

2. Теоретическая часть задания

Так как принцип алгоритма обхода лабиринта похож на принцип обхода в глубину, далее будет описана теоретическая часть рекурсивного обхода в глубину.

Поскольку осуществляется обход каждого «соседа» каждого узла, игнорируя тех, которых посещали ранее, мы имеем время выполнения, равное O(V + E). V — общее количество вершин. E — общее количество граней (ребер).

V+E означает, что для каждой вершины мы оцениваем лишь примыкающие к ней грани. Каждая вершина имеет определенное количество граней и, в худшем случае, мы обойдем все вершины (O(V)) и исследуем все грани (O(E)). Мы имеем V вершин и E граней, поэтому получаем V+E.

Если использовать рекурсию для обхода каждой вершины, это означает, что используется стек (бесконечная рекурсия приводит к ошибке переполнения стека). Поэтому пространственная сложность составляет O(V).

Рассмотрим пример. Предположим, что у нас есть ориентированный граф, приведённый на рис.1.

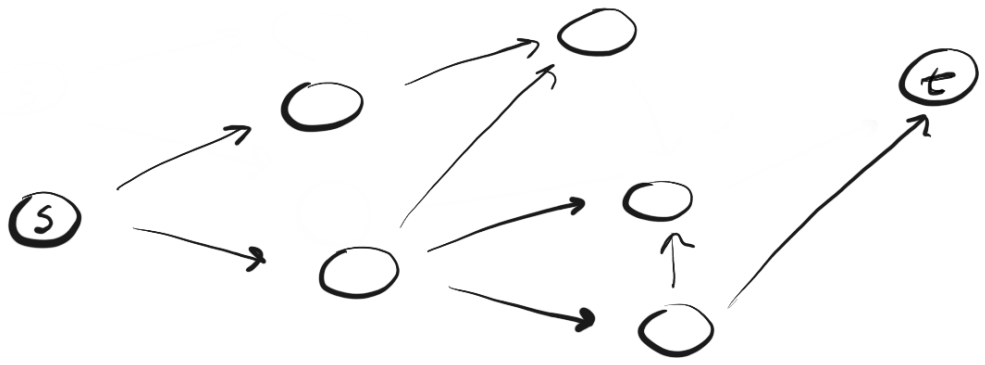

Рисунок 1 – ориентированный граф

Мы находимся в точке «s» и нам нужно найти вершину «t». Применяя DFS, мы исследуем один из возможных путей, двигаемся по нему до конца и, если не обнаружили t, возвращаемся и исследуем другой путь. Здесь мы двигаемся по пути (p1) к ближайшей вершине и видим, что это не конец пути. Поэтому мы переходим к следующей вершине. Мы достигли конца p1, но не

нашли t, поэтому возвращаемся в s и двигаемся по второму пути. Достигнув ближайшей к точке «s» вершины пути «p2» мы видим три возможных направления для дальнейшего движения. Поскольку вершину, венчающую первое направление, мы уже посещали, то двигаемся по второму. Мы вновь достигли конца пути, но не нашли t, поэтому возвращаемся назад. Следуем по третьему пути и, наконец, достигаем искомой вершины «t», это приведено на рис.2.

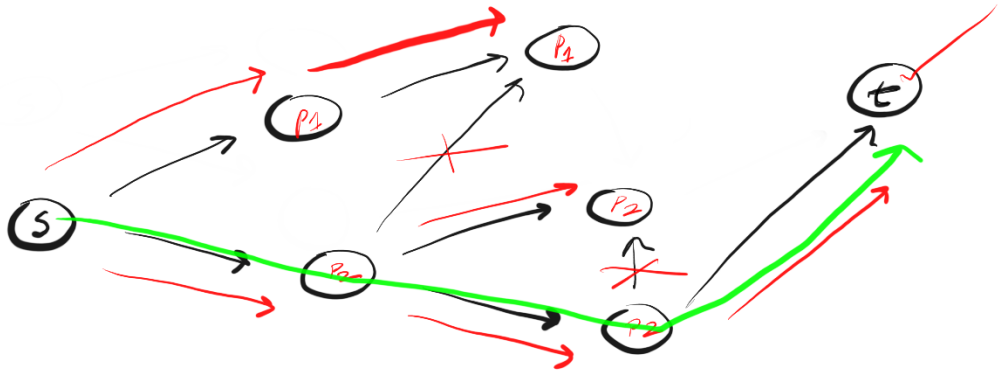

Рисунок 2 - ориентированный граф

Так работает DFS. Двигаемся по определенному пути до конца. Если конец пути — это искомая вершина, мы закончили. Если нет, возвращаемся назад и двигаемся по другому пути до тех пор, пока не исследуем все варианты. Следуем этому алгоритму применительно к каждой посещенной вершине.

Необходимость многократного повторения процедуры указывает на необходимость использования рекурсии для реализации алгоритма. Однако в данной программе реализуется немного изменённый обход в глубину, не рекурсивного типа.

3. Описание алгоритма программы

Для программной реализации алгоритма понадобиться один динамический двумерный массив: двумерный массив int maze – в нём хранится исходная матрица лабиринта, в последующем для вывода в консоль этот массив преобразовывается в вектор.

Главная функция *main* представляет собой меню выбора, в котором вызываются все остальные функции: функция *generator* генерирует случайный массив-лабиринт, который сохраняется в файл, к этой функции привязаны ещё две: *genVhod* и *genVihod*, они отвечают за случайное создание входа и выхода из лабиринта; функция *save* сохраняет случайную матрицу-лабиринт в файл; функция *open* позволяет использовать ранее сохранённый в файл лабиринт; функция *single* позволяет протестировать созданный лабиринт самостоятельно; функция *bot* позволяет наглядно убедиться в работоспособности игрового агента, который использует алгоритм обхода в глубину.

Функция main позволяет перемещаться по меню, вначале она имеет два пункта: Start и Exit, названия говорят сами за себя. При нажатии на Start консоль обновляется и появляется следующий уровень меню, который имеет пункты: Random generation maze – при нажатии вызывает функцию генерации случайного лабиринта, Generating from a file – из файла берётся уже сгенерированный ранее лабиринт, если таковой есть, Save in a file – сохраняет случайно сгенерированный лабиринт в файл, Back – возвращает на уровень меню ниже. При нажатии на 1-ый пункт предлагается ввести размер лабиринта (с условиями), после ввода и подтверждения, открывается следующий 3-ий уровень меню. При нажатии 2-ого пункта сразу открывается 3-ий уровень меню, в котором есть пункты: Single mod – запуск тестирования лабиринта для его самостоятельного прохождения, тоже самое, что и игра – лабиринт, Agent – запускает “бота”, который обходит граф с задаваемой скоростью.

Ниже приведено краткое описание оставшихся функций:

generator

1.1.Объявление переменных входа, выхода лабиринта и его размера.

1.2.Ввод через консоль размерности массива.

1.3.Создание двумерной динамической матрицы.

1.4.Заполнение матрицы: стенки – единицами, комнаты – нулями.

1.5.Вызов функции создания входа лабиринта.

1.5.1.Создание входа с края случайной комнаты.

1.6.Ввод и подсчёт переменной кол-ва комнат.

1.7.Создание списка со случайными числами от 0 до 3 количественно равными кол-ву комнат random[x].

1.8.Инициализация переменных положения в матрице str и stl.

1.9.Пока не кончатся random[i]:

1.9.1.Switch(random):

1.9.1.1.Каждый кейс соответствует стороне, которая выбирается случайно и в которую можно пойти и сломать стенку.

1.9.1.2.Если позиция тупиковая, возвращаемся назад до тех пор пока не найдём ближайшую не посещённую комнату.

1.9.1.3.При полном прохождении и возвращении в начальную позицию выходим из цикла.

1.10.Вызов функции создания выхода из лабиринта

1.10.1Случайно генерируем выход.

1.11.Записывем полученный лабиринт и переменные размера, входа и выхода в файл gen.txt.

single

1.1.Вытаскиваем из файла gen.txt матрицу-лабиринт и все переменные.

1.2.Из int[] преобразовываем массив в vector <char>.

1.3.Определяем точку входа лабиринта.

1.4.Заменяем цифирное представление матрицы-лабиринта на представление в ACSII кодировке.

1.5.Ставим положение игрока и выводим лабиринт.

1.6.Пока не достигнем выхода:

1.6.1.При нажатии на определённую стрелку идём в соответствующую сторону, передвигаемся по лабиринту.

1.6.1.1.При каждом изменении выводим лабиринт в консоль.

1.7.Выводим статистику игрока: время, размер лабиринта, количество нажатий.

Ниже представлен частичный псевдокод функции игрового агента - *bot*.

bot

1.1.Обьявляются переменные размера, входа и выхода лабиринта: n, vhod, vihod.

1.2.Вытаскиваем сохранённые значения переменных n, vhod, vihod и матрицу-лабиринт из файла gen.txt

1.2.1.Создаётся двумерный динамический массив maze[][] и в него записывается лабиринт.

1.3.Преобразовываем maze[][] в вектор с типом char.

1.4.Задаётся начальное положение агента - инициализируется переменная строки лабиринта str = vhod и переменная столбца stl = 0.

1.5.Матричные значения заменяются на символы из кодировки ASCII: стенки, равные 1 заменяются на ■, клетки 0 на пустоту, агент задаётся символом ¤.

1.6.На экран консоли выводится лабиринт со стартовым положением агента.

1.7.Пока агент не находится на клетке выхода:

1.7.1.Пока, хотя бы с одной стороны есть куда идти:

1.7.1.1. Идём в любую свободную сторону, приоритетом направления является правая сторона, после верхняя, затем нижняя и наконец левая.

1.7.1.2.Если текущее положение является выходом то выходим из цикла.

1.7.1.3Если в выбранную соответствующую сторону можно пойти, то смещаем положение агента на 1 в ту сторону, переносим символ агента, помечаем пройденную клетку ромбиком и обновляем лабиринт в консоли.

1.7.2. Если текущее положение является выходом, то выходим из цикла.

1.7.3.Если не в одну из сторон нельзя пойти:

1.7.3.1.Пока, рядом с агентом нет ни одной не посещённой клетки:

1.7.3.1.1.Проверка каждой стороны на наличие метки ромбика.

1.7.3.1.2.При наличии ромбика в соответствующей стороне смещаем туда агента на 1, пройденную клетку заменяем на новую метку черви и обновляем консоль.

1.8.Выводится время прохождения агента и завершается функция.

4. Описание программы

Для написания данной программы использован язык программирования C++. Язык программирования C++ - 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: main, generator, genVhod, genVihod, save, bot, open, single.

Выше в описании функции main, можно понять, что может сделать пользователь, и что от него требуется.

Когда пользователь генерирует случайный лабиринт, то он создаётся и записывается в файл.

При выборе пользователем прохождения лабиринта вручную, пользователь может использовать клавиши стрелок для перемещения по лабиринту, к исполнению приводится следующая часть кода:

```
while (TRUE)
	{
		if (GetAsyncKeyState(VK_UP)) //Если нажали стрелку вверх
		{
			keybd_event(VK_UP, 0, KEYEVENTF_KEYUP, 0);//Отжимаем кнопку
			if (maze2[str - 1][stl] == 32) { //Если сверху пусто
				maze2[str - 1][stl] = 253; //Ставим туда игрока
				maze2[str][stl] = '  '; //Убираем текущую его позицию
				str--; //Меняем позицию в матрице
				system("cls"); //Очищаем консоль
				for (int i = 0; i < n; i++)
				{
					for (int j = 0; j < m; j++)
						cout << maze2[i][j] << " ";
					cout << endl;
				}                                  //Выводим обновлённый лабиринт
				click++; //Переменная считывающая кол-во кликов для статистики
			}
		}
		if (GetAsyncKeyState(VK_DOWN)) //Если нажали стрелку вниз
		{
			keybd_event(VK_DOWN, 0, KEYEVENTF_KEYUP, 0);//Отжимаем кнопку
			if (maze2[str + 1][stl] == 32) { //если снизу пусто
				maze2[str + 1][stl] = 253;
				maze2[str][stl] = '  ';
				str++;
				system("cls");
				for (int i = 0; i < n; i++)
				{
```

```
                        for (int j = 0; j < m; j++)
                                cout << maze2[i][j] << " ";
                        cout << endl;
                }
                click++;
        }
    }
    if (GetAsyncKeyState(VK_RIGHT)) //Если нажали стрелку вправо
    {
        keybd_event(VK_RIGHT, 0, KEYEVENTF_KEYUP, 0);//Отжимаем кнопку
        if (maze2[str][stl + 1] == 32) { //если справа пусто
            maze2[str][stl + 1] = 253;
            maze2[str][stl] = '  ';
            stl++;
            system("cls");
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < m; j++)
                        cout << maze2[i][j] << " ";
                cout << endl;
            }
            if (maze2[str][stl] == maze2[vihod][n-1]) {
                break;
            }
            click++;
        }
    }
    if (GetAsyncKeyState(VK_LEFT)) //Если нажали стрелку влево
    {
        keybd_event(VK_LEFT, 0, KEYEVENTF_KEYUP, 0);//Отжимаем кнопку
        if (maze2[vhod][0] != maze2[str][stl]) { //если слева пусто
            if (maze2[str][stl - 1] == 32) {
                maze2[str][stl - 1] = 253;
                maze2[str][stl] = '  ';
                stl--;
                system("cls");
                for (int i = 0; i < n; i++)
                {
                    for (int j = 0; j < m; j++)
                            cout << maze2[i][j] << " ";
                    cout << endl;
                }
                click++;
            }
        }
    }
}
```

При использовании функции игрового агента, принцип обхода которого, похож на принцип генерации лабиринта, пользователь задаёт скорость агента и наблюдает как он проходит лабиринт. Часть кода этой функции:

```
while (TRUE)
	{
		while ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl] == 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) //пока есть свободные клетки
		{ 	//хотя бы одна с 4 сторон
			if (maze2[str][stl] == maze2[vihod][n - 1]) { break; } //Если текущая позиция является выходом из лабиринта
			if (maze2[str][stl + 1] == 32) { //Если справа свободно
				maze2[str][stl + 1] = 253; //Ставим туда агента
				maze2[str][stl] = 4; //ставим метку на текущую позицию
				stl++; //Меняем позицию в матрице
				system("cls"); //Очищаем консоль
				for (int i = 0; i < n; i++)
				{
					for (int j = 0; j < m; j++)
						cout << maze2[i][j] << " ";
					cout << endl;
				} //Выводим обновлённый лабиринт
				Sleep(zader); //Задаваемая задержка для агента
			}
			if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
			if (maze2[str - 1][stl] == 32) { //Если сверху свободно
			maze2[str - 1][stl] = 253;
				maze2[str][stl] = 4;
				str--;
				system("cls");
				for (int i = 0; i < n; i++)
				{
					for (int j = 0; j < m; j++)
						cout << maze2[i][j] << " ";
					cout << endl;

				}
				Sleep(zader);
			}
			if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
			if (maze2[str + 1][stl] == 32) { //Если снизу свободно
				maze2[str + 1][stl] = 253;
				maze2[str][stl] = 4;
				str++;
				system("cls");
				for (int i = 0; i < n; i++)
				{
					for (int j = 0; j < m; j++)
						cout << maze2[i][j] << " ";
					cout << endl;
				}
				Sleep(zader);
			}
			if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
			if (maze2[vhod][0] != maze2[str][stl]) { //Если слева свободно
				if (maze2[str][stl - 1] == 32) {
					maze2[str][stl - 1] = 253;
					maze2[str][stl] = 4;
					stl--;
					system("cls");
```

```
                                        for (int i = 0; i < n; i++)
                                        {
                                                for (int j = 0; j < m; j++)
                                                        cout << maze2[i][j] << " ";
                                                cout << endl;
                                        }
                                }
                                Sleep(zader);
                        }
                }
                        if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                        if ((maze2[str - 1][stl] != 32) && (maze2[str + 1][stl] != 32) &&
(maze2[str][stl - 1] != 32) && (maze2[str][stl + 1] != 32)) { //Если не куда идти
                                while (TRUE) {
                                        if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                                        if ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl]
== 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) { break; }
//если хотя бы в одну сторону можно пойти
                                        if (maze2[str][stl + 1] == 4) { //если справа метка 4
                                                maze2[str][stl + 1] = 253; //Ставим туда агента
                                                maze2[str][stl] = 3;
//ставим новую метку на текущую позицию
                                                stl++;
                                                system("cls");
                                                for (int i = 0; i < n; i++)
                                                {
                                                        for (int j = 0; j < m; j++)
                                                                cout << maze2[i][j] << " ";
                                                        cout << endl;
                                                }
                                                Sleep(zader);
                                        }
                                        if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                                        if ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl]
== 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) { break; }
                                        if (maze2[str - 1][stl] == 4) { //если сверху метка 4
                                                maze2[str - 1][stl] = 253;
                                                maze2[str][stl] = 3;
                                                str--;
                                                system("cls");
                                                for (int i = 0; i < n; i++)
                                                {
                                                        for (int j = 0; j < m; j++)
                                                                cout << maze2[i][j] << " ";
                                                        cout << endl;
                                                }
                                                Sleep(zader);
                                        }
                                        if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                                        if ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl]
== 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) { break; }
                                        if (maze2[str + 1][stl] == 4) { //если снизу метка 4
                                        maze2[str + 1][stl] = 253;
                                                maze2[str][stl] = 3;
                                                str++;
                                                system("cls");
                                                for (int i = 0; i < n; i++)
                                                {
                                                        for (int j = 0; j < m; j++)
                                                                cout << maze2[i][j] << " ";
                                                        cout << endl;
                                                }
                                                Sleep(zader);
                                        }
```

```
                                    if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                                    if ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl]
== 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) { break; }
                                    if (maze2[str][stl - 1] == 4) { //если слева метка 4
                                          maze2[str][stl - 1] = 253;
                                          maze2[str][stl] = 3;
                                          stl--;
                                          system("cls");
                                          for (int i = 0; i < n; i++)
                                          {
                                                for (int j = 0; j < m; j++)
                                                      cout << maze2[i][j] << " ";
                                                cout << endl;
                                          }
                                          Sleep(zader);
                                    }
                                    if ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl]
== 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) { break; }
                              }
                        }
                  if (maze2[str][stl] == maze2[vihod][n - 1]) {
                        int endTime = clock();
                        double time = (endTime - startTime) / 1000;//время пройденное агентом
                        break;
                  }
            }
```

5. Тестирование

Среда разработки Microsoft Visual Studio 2019 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки и после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом и обращением к данным, изменением формата используемых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вызове функций без предварительного создания графа.

Тестирование:

Вначале появляется стартовое меню (Рисунок 3).

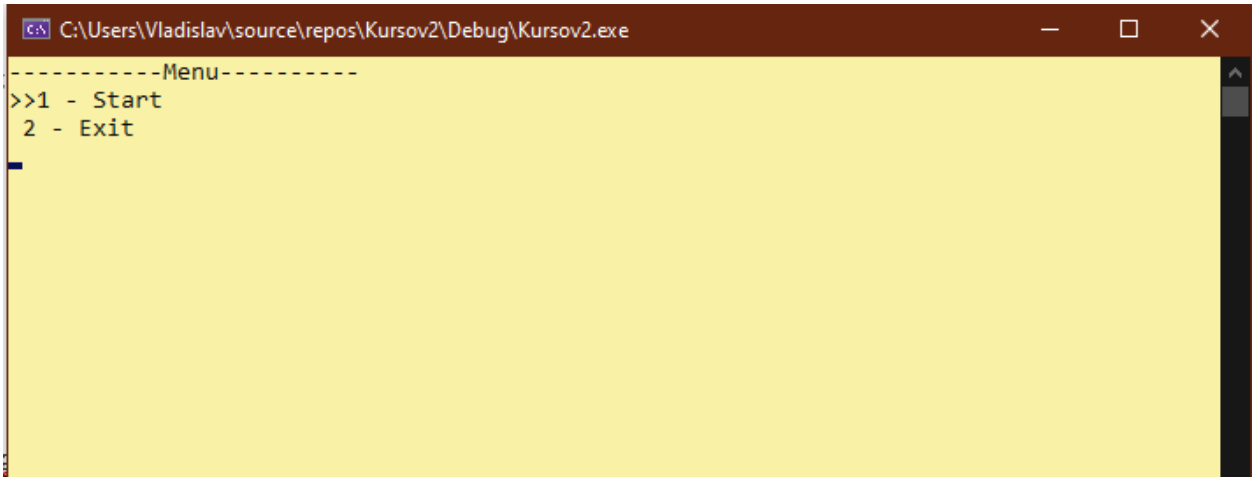

Рисунок 3

Далее при нажатии на Start появляется меню выбора (Рисунок 4).

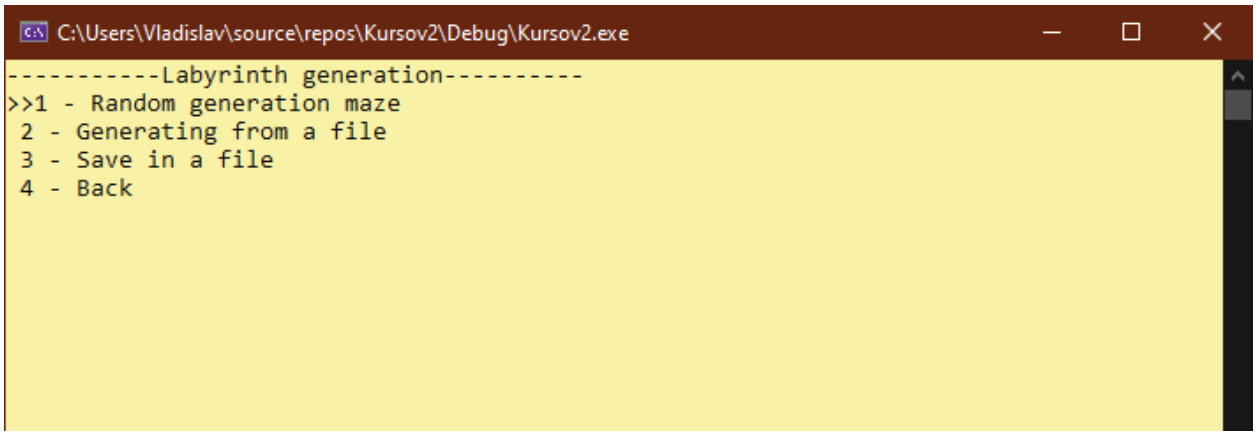

Рисунок 4

При выборе случайной генерации лабиринта поступит запрос на введение размерности лабиринта, введём 9 (Рисунки 5, 6)

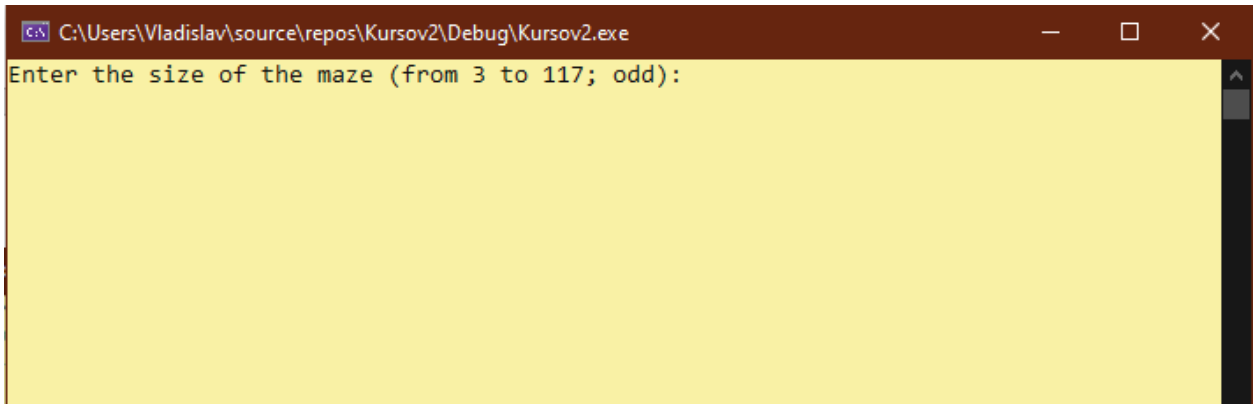

Рисунок 5

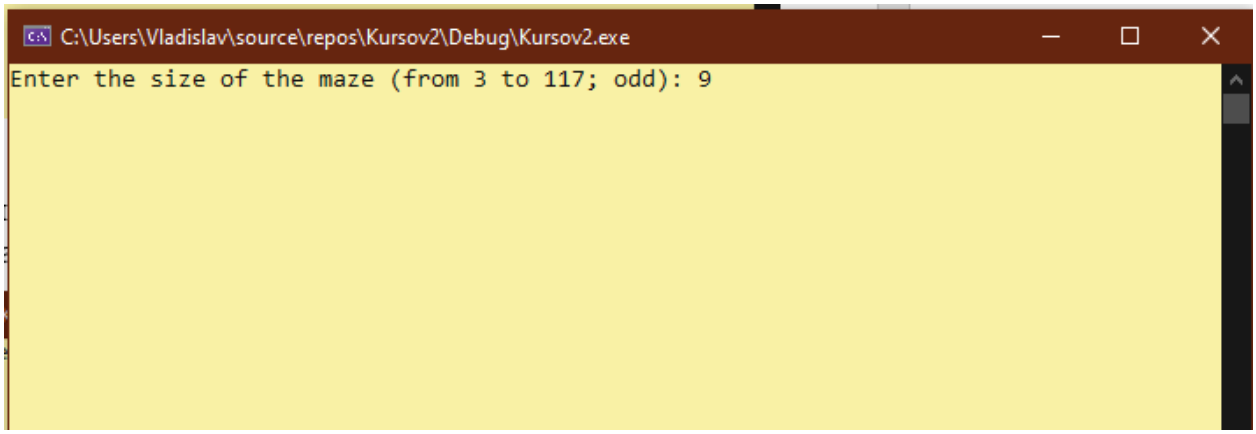

Рисунок 6

При подтверждении выводит выбор действий (Рисунок 7).

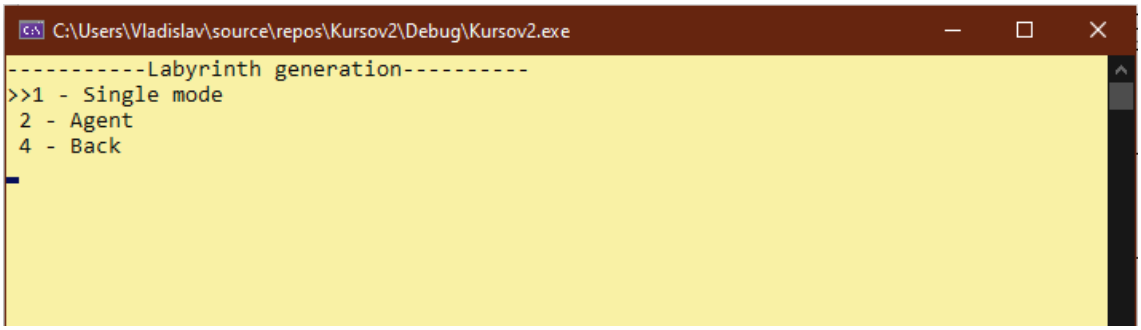

Рисунок 7

При выборе одиночного режима выводится лабиринт, который проходится клавишами на клавиатуре (Рисунок 8).

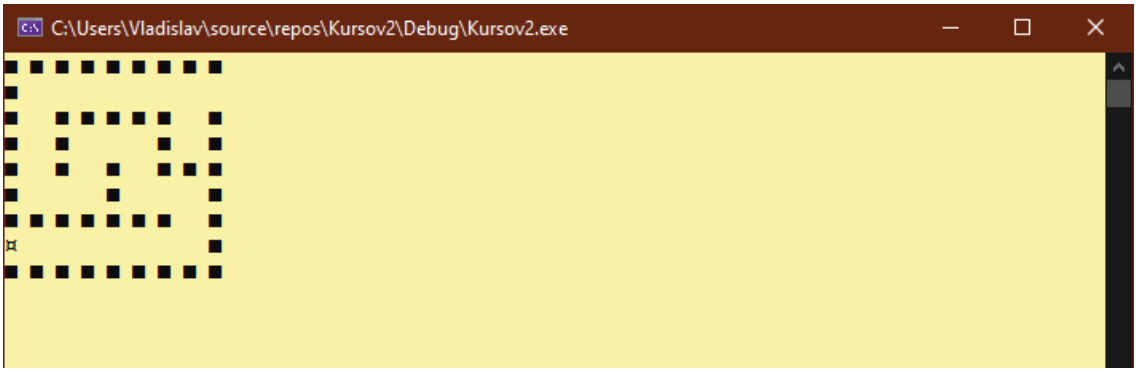

Рисунок 8

При прохождении лабиринта выводит окно и выводит в консоль статистику игрока (Рисунки 9, 10)

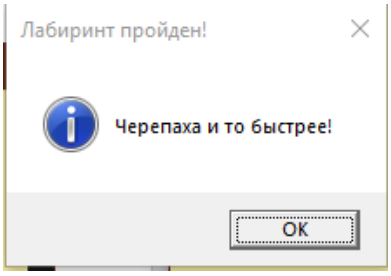

Рисунок 9

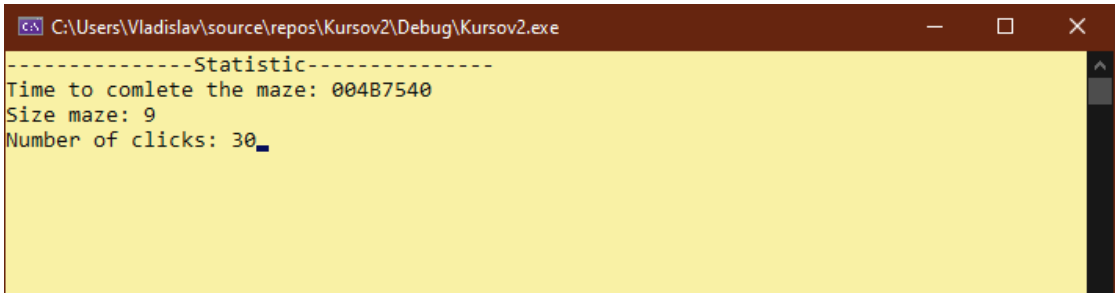

Рисунок 10

При выборе режима агента поступит запрос на введение его скорости (задержки при прохождении) (Рисунок 11).

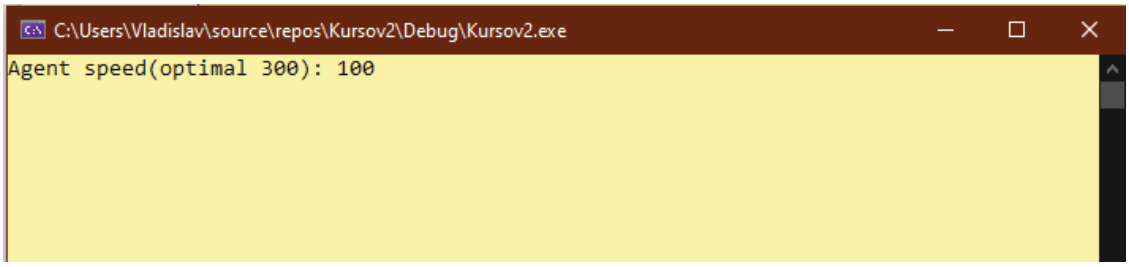

Рисунок 11

Далее агент самостоятельно обойдёт лабиринт до тех пор, пока не найдёт выход из него (Рисунок 12).

Рисунок 12

При нажатии на 2-ом уровне меню на генерацию из файла, лабиринт создастся из файла и откроется меню 3-его уровня, для дальнейших действий пользователя.

При нажатии на сохранение файла, сгенерированный случайный лабиринт сохранится в файл и появится окно, как на рисунке 13.

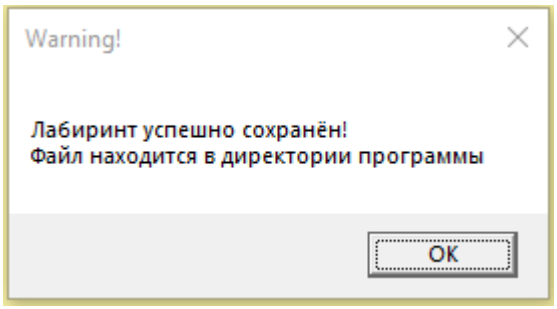

Рисунок 13 - окно

Таблица 1 – Описание поведения программы при тестировании.

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод пользователю списка с функциями программы	Верно
Меню широкого спектра пользования	Вывод пользователю предложений выбора функций	
Задание размера лабиринта при его случайной генерации	Вывод пользователю предложения задать размер лабиринта числом	Верно
Тестирование на вывод лабиринта	Генерация и вывод лабиринта на экран	Верно
Вызов функции заполнения лабиринта из файла без задания размера графа.	Вывод пользователю предупреждения об успешности вызова данной функции	Верно
Вызов функции алгоритма прохождения лабиринта самостоятельно	Вывод на экран лабиринта и предоставление возможности пользователю его пройти	Верно
Вызов агента для прохождения лабиринта	Пользователь наблюдает за прохождением лабиринта игровым агентом	Верно
Сохранение в файл лабиринта	Запись лабиринта в файл	Верно
Меню с возможностью возвращения на предыдущий уровень меню	Пользователь может возвращаться на предыдущий уровень меню и повторять операции сколько угодно раз	Верно

6. Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере лабиринта с размерностью 7 (Рисунок 14). Определим как будет игровой агент проходить сгенерированный лабиринт.

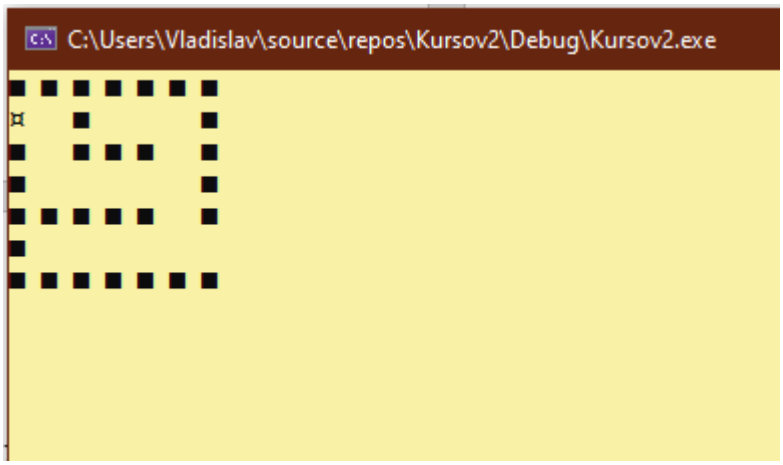

Рисунок 14

Лабиринт основан на координатах, поэтому определим начальную точку откуда будет двигаться игровой агент (1;0). За левую координату взято положение агента на строчке (1), за правую (0), потому что сточки и столбцы идут от 0 до 6 (в нашем случае). Агент в первую очередь «увидит», что справа от него есть проход, поэтому он сдвинется туда, столбец + 1: (1;1).

Далее он «увидит», что снизу свободно и спуститься на 2 клетки: (3;1). Далее он пойдёт вправо на координаты (3;5) (Рисунок 15).

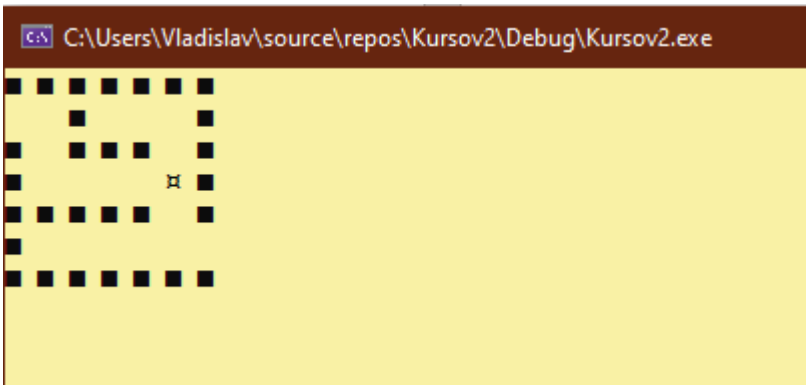

Рисунок 15

По алгоритму, агент сначала пойдёт вверх на 2 клетки и влево до конца, он будет в тупике и вернётся в те же координаты: (3;5). Далее он спуститься вниз на 2 клетки (5;5) и пойдёт влево до конца, в тупик, и так же

вернётся в эти координаты, и завершит своё прохождение лабиринта (Рисунок 16 – Игровой агент).

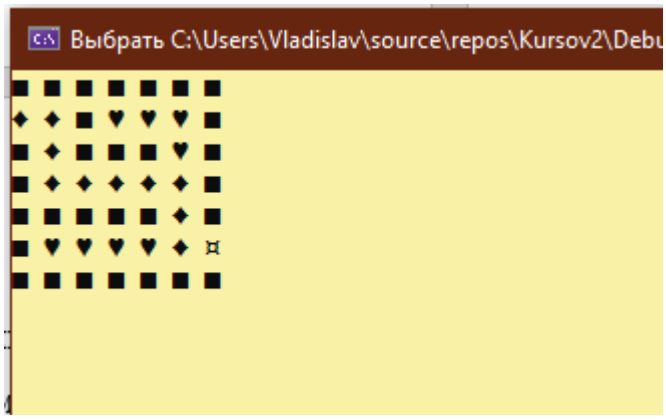

Рис.16-Игровой агент

Таким образом игровой агент прошёл весь лабиринт от (1;0) до (5;6).

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм генерации лабиринта и алгоритм прохождения лабиринта игровым в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания обхода в глубину, и динамических структур данных. Приобретены навыки по осуществлению алгоритма генерации случайных лабиринтов и алгоритма прохождения лабиринта игровым агентом. Углублены знания языка программирования C++.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Код написан максимально компактно и интерфейс удобен для использования. Полученные знания можно применять в дальнейшей деятельности.

Список литературы

1. Герберт Шилдт. C++. Базовый курс. 2014 г.
2. В. Л. Дольников О. П. Якимова. Основные алгоритмы на графах. 2011 г.
3. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.

Приложение А.

Листинг программы.

Файл Kursov2.cpp: функция main()

```
 #include "Windows.h"
#include <stdio.h>
#include <iostream>
#include <string>
#include "generator.h"
#include "single.h"
#include "save.h"
#include "open.h"
#include "bot.h"
using namespace std;

void menu1(int Item1)
{
    system("cls");
    printf("-----------Menu----------\n");
    printf("%s1 - Start\n", Item1 == 1 ? ">>" : " ");
    printf("%s2 - Exit\n", Item1 == 2 ? ">>" : " ");
}
void menu2(int Item2)
{
    system("cls");
    printf("-----------Labyrinth generation----------\n");
    printf("%s1 - Random generation maze\n", Item2 == 1 ? ">>" : " ");
    printf("%s2 - Generating from a file\n", Item2 == 2 ? ">>" : " ");
    printf("%s3 - Save in a file\n", Item2 == 3 ? ">>" : " ");
    printf("%s4 - Back\n", Item2 == 4 ? ">>" : " ");
}
void menu3(int Item3)
{
    system("cls");
    printf("-----------Labyrinth generation----------\n");
    printf("%s1 - Single mode\n", Item3 == 1 ? ">>" : " ");
    printf("%s2 - Agent\n", Item3 == 2 ? ">>" : " ");
    printf("%s4 - Back\n", Item3 == 3 ? ">>" : " ");
}
int main()
{
    system("color E0");//07
menu11:
    int Item1 = 1;
    int Last1 = 2;
    menu1(Item1);
    while (TRUE)
    {
        system("pause >> NUL");
        Sleep(300);
        if (GetAsyncKeyState(VK_UP))
        {
            keybd_event(VK_UP, 0, KEYEVENTF_KEYUP, 0);
            if (0 < Item1 - 1)
                Item1 = Item1 - 1;
            else
                Item1 = Last1;
            menu1(Item1);
        }
        if (GetAsyncKeyState(VK_DOWN))
```

```
        {
            keybd_event(VK_DOWN, 0, KEYEVENTF_KEYUP, 0);
            if (Item1 < Last1)
                Item1 = Item1 + 1;
            else
                Item1 = 1;
            menu1(Item1);
        }
        if (GetAsyncKeyState(VK_RETURN))
        {
            keybd_event(VK_DOWN, 0, KEYEVENTF_KEYUP, 0);
            menu1(Item1);
            switch (Item1)
            {
            case 1:
                Sleep(300);
                goto menu22;
                break;
            case 2:
                system("cls");
                exit(0);
            }
        }
    }
    return 0;
menu22:
    int Item2 = 1;
    int Last2 = 4;
    menu2(Item2);
    while (TRUE)
    {
        system("pause >> NUL");
        Sleep(300);
        if (GetAsyncKeyState(VK_UP))
        {
            keybd_event(VK_UP, 0, KEYEVENTF_KEYUP, 0);
            if (0 < Item2 - 1)
                Item2 = Item2 - 1;
            else
                Item2 = Last2;
            menu2(Item2);
        }
        if (GetAsyncKeyState(VK_DOWN))
        {
            keybd_event(VK_DOWN, 0, KEYEVENTF_KEYUP, 0);
            if (Item2 < Last2)
                Item2 = Item2 + 1;
            else
                Item2 = 1;
            menu2(Item2);
        }
        if (GetAsyncKeyState(VK_RETURN))
        {
            keybd_event(VK_DOWN, 0, KEYEVENTF_KEYUP, 0);
            switch (Item2)
            {
            case 1:
                generator();
                system("cls");
                Sleep(300);
                goto menu33;
            case 2:
                open();
                Sleep(300);
```

```
                goto menu33;
            case 3:
                save();
                MessageBox(0, L"Лабиринт успешно сохранён!\nФайл находится в директории
программы", L"Warning!", MB_OK);
                Sleep(300);
                break;
            case 4:
                Sleep(300);
                goto menu11;
            }
        }
    }
    return 0;
menu33:
    int Item3 = 1;
    int Last3 = 3;
    menu3(Item3);
    while (TRUE)
    {
        system("pause >> NUL");
        Sleep(300);
        if (GetAsyncKeyState(VK_UP))
        {
            keybd_event(VK_UP, 0, KEYEVENTF_KEYUP, 0);
            if (0 < Item3 - 1)
                Item3 = Item3 - 1;
            else
                Item3 = Last3;
            menu3(Item3);
        }
        if (GetAsyncKeyState(VK_DOWN))
        {
            keybd_event(VK_DOWN, 0, KEYEVENTF_KEYUP, 0);
            if (Item3 < Last3)
                Item3 = Item3 + 1;
            else
                Item3 = 1;
            menu3(Item3);
        }
        if (GetAsyncKeyState(VK_RETURN))
        {
            keybd_event(VK_DOWN, 0, KEYEVENTF_KEYUP, 0);
            menu3(Item3);
            switch (Item3)
            {
            case 1:
                single();
                system("cls");
                Sleep(300);
                goto menu33;
            case 2:
                bot();
                system("cls");
                Sleep(300);
                goto menu33;
            case 3:
                Sleep(300);
                goto menu22;
            }
        }
    }
    return 0;
}
```

Файл generator.cpp: функции: generator(), genVhod(), genVihod().

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include "TIME.H"
#include "windows.h"
#include <random>
#include <fstream>
FILE* WRITE;

using namespace std;
int vhod;
int vihod;
int** genVhod(int** maze, int n, int m) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dist(0, n - 2);
    vhod = dist(gen);
    if (vhod % 2 == 0) { vhod++; }
    maze[vhod][0] = 0;
    return maze;
}
int** genVihod(int** maze, int n, int m) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dist(0, m - 2);
    vihod = dist(gen);
    if (vihod % 2 == 0) { vihod++; }
    maze[vihod][m - 1] = 0;
    return maze;
}
void generator() {
    int n, m;
    hello:
    system("cls");
    cout << "Enter the size of the maze (from 3 to 117; odd): ";
    cin >> n;
    if ((n % 2 == 0) || (n <= 2) || (n >= 118)) { goto hello; }
    m = n;
    int** maze = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++)
    {
        maze[i] = (int*)malloc(m * sizeof(int));
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if ((i % 2 == 0) || (j % 2 == 0)) {
                maze[i][j] = 1;
            }
            if ((i % 2 != 0) && (j % 2 != 0)) {
                maze[i][j] = 0;
            }
        }
    }
    genVhod(maze, n, m);
    float x = (n / 2) * (n / 2); //кол-во нечетных 0 в псевдографе
    int* random = new int[x];
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dist(0, 3);
    for (int y = 0; y < x; y++) {
        int choice = dist(gen);
        random[y] = choice;
    }
```

```
cout << endl;
int y = 0;
int str = vhod;
int stl = 1;
maze[str][stl - 1] = 8;//vhod 8
maze[str][stl] = 6;
while (y < x) {
    switch (random[y]) {
    case 0:
        if (str - 2 >= 0) {//vverh
            if (maze[str - 2][stl] == 0) {
                maze[str - 1][stl] = 6;
                maze[str - 2][stl] = 6;
                str = str - 2;
                y++;
                break;
            }
        }
    case 1:
        if (str + 2 <= n - 2) {//vniz
            if (maze[str + 2][stl] == 0) {
                maze[str + 1][stl] = 6;
                maze[str + 2][stl] = 6;
                str = str + 2;
                y++;
                break;
            }
        }
    case 2:
        if (stl - 2 >= 0) {//vlevo
            if (maze[str][stl - 2] == 0) {
                maze[str][stl - 1] = 6;
                maze[str][stl - 2] = 6;
                stl = stl - 2;
                y++;
                break;
            }
        }
    case 3:
        if (stl + 2 <= n - 2) {//vpravo
            if (maze[str][stl + 2] == 0) {
                maze[str][stl + 1] = 6;
                maze[str][stl + 2] = 6;
                stl = stl + 2;
                y++;
                break;
            }
        }
    case 4:
        if (str - 2 >= 0) {//vverh
            if (maze[str - 2][stl] == 0) {
                maze[str - 1][stl] = 6;
                maze[str - 2][stl] = 6;
                str = str - 2;
                y++;
                break;
            }
        }
    case 5:
        if (str + 2 <= n - 2) {//vniz
            if (maze[str + 2][stl] == 0) {
                maze[str + 1][stl] = 6;
                maze[str + 2][stl] = 6;
                str = str + 2;
```

```
            y++;
            break;
        }
    }
case 6:
    if (stl - 2 >= 0) {//vlevo
        if (maze[str][stl - 2] == 0) {
            maze[str][stl - 1] = 6;
            maze[str][stl - 2] = 6;
            stl = stl - 2;
            y++;
            break;
        }
    }
case 7:
    while (TRUE) {
        if (maze[str][stl - 1] == 8) { break; }

        if (str - 2 >= 0) {
            if (maze[str - 2][stl] == 0) { break; }
        }
        if (str + 2 <= n - 2) {
            if (maze[str + 2][stl] == 0) { break; }
        }
        if (stl - 2 >= 0) {
            if (maze[str][stl - 2] == 0) { break; }
        }
        if (stl + 2 <= n - 2) {
            if (maze[str][stl + 2] == 0) { break; }
        }
        if (maze[str - 1][stl] == 6) {//vverh
            maze[str][stl] = 5;
            maze[str - 1][stl] = 5;
            str = str - 2;
        }
        if (str - 2 >= 0) {
            if (maze[str - 2][stl] == 0) { break; }
        }
        if (str + 2 <= n - 2) {
            if (maze[str + 2][stl] == 0) { break; }
        }
        if (stl - 2 >= 0) {
            if (maze[str][stl - 2] == 0) { break; }
        }
        if (stl + 2 <= n - 2) {
            if (maze[str][stl + 2] == 0) { break; }
        }
        if (maze[str + 1][stl] == 6) {//vniz
            maze[str][stl] = 5;
            maze[str + 1][stl] = 5;
            str = str + 2;
        }
        if (str - 2 >= 0) {
            if (maze[str - 2][stl] == 0) { break; }
        }
        if (str + 2 <= n - 2) {
            if (maze[str + 2][stl] == 0) { break; }
        }
        if (stl - 2 >= 0) {
            if (maze[str][stl - 2] == 0) { break; }
        }
        if (stl + 2 <= n - 2) {
            if (maze[str][stl + 2] == 0) { break; }
        }
```

```
                if (maze[str][stl - 1] == 6) {//vlevo
                    maze[str][stl] = 5;
                    maze[str][stl - 1] = 5;
                    stl = stl - 2;
                }
                if (str - 2 >= 0) {
                    if (maze[str - 2][stl] == 0) { break; }
                }
                if (str + 2 <= n - 2) {
                    if (maze[str + 2][stl] == 0) { break; }
                }
                if (stl - 2 >= 0) {
                    if (maze[str][stl - 2] == 0) { break; }
                }
                if (stl + 2 <= n - 2) {
                    if (maze[str][stl + 2] == 0) { break; }
                }
                if (maze[str][stl + 1] == 6) {//vpravo
                    maze[str][stl] = 5;
                    maze[str][stl + 1] = 5;
                    stl = stl + 2;
                }
            }
        }
        if (maze[str][stl - 1] == 8) { break; }
    }
    genVihod(maze, n, m);
    cout << endl;
    WRITE = fopen("gen.txt", "w");
    fprintf(WRITE, "%3d ", n);
    fprintf(WRITE, "%3d ", vhod);
    fprintf(WRITE, "%3d ", vihod);
    fprintf(WRITE, "\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            fprintf(WRITE, "%3d ",maze[i][j]);
        }
        fprintf(WRITE, "\n");
    }
    fclose(WRITE);
    for (int i = 0; i < n; i++) {
        delete[] maze[i];  // удаляем массив
    }
}
```

Файл open.cpp: функция open().

```
#include <fstream>
using namespace std;

void open() {
       int n = 0;
       int vhod = 0;
       int vihod = 0;

       ifstream fin("save.txt");
       fin >> n;
       fin >> vhod;
       fin >> vihod;

       int** maze = (int**)malloc(n * sizeof(int*));
       for (int i = 0; i < n; i++)
       {
              maze[i] = (int*)malloc(n * sizeof(int));
       }
```

```
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			fin >> maze[i][j];
		}
	}
	fin.close();



	ofstream fout("gen.txt");
	fout << n << " " << vhod << " " << vihod << "\n";
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			fout << maze[i][j] << " ";
		}
		fout << "\n";
	}
	fout.close();


}
```

Файл save.cpp: функция save().

```
#include <fstream>
using namespace std;
void save() {
	int n = 0;
	int vhod = 0;
	int vihod = 0;

	ifstream fin("gen.txt");
	fin >> n;
	fin >> vhod;
	fin >> vihod;

	int** maze = (int**)malloc(n * sizeof(int*));
	for (int i = 0; i < n; i++)
	{
		maze[i] = (int*)malloc(n * sizeof(int));
	}

	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			fin >> maze[i][j];
		}
	}
	fin.close();



	ofstream fout("save.txt");
	fout << n << " " << vhod << " " << vihod << "\n";
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			fout << maze[i][j] << " ";
		}
		fout << "\n";
	}
	fout.close();

}
```

Файл single.cpp: функция single().

```
//#define _CRT_SECURE_NO_WARNINGS
#include <vector>
#include <iostream>
#include "Windows.h"
#include <fstream>
#include <random>
#include <time.h>
//FILE* READ;
using namespace std;

void single() {
	int n = 0;
	int vhod = 0;
	int vihod = 0;
	ifstream fin("gen.txt");
	fin >> n;
	fin >> vhod;
	fin >> vihod;
	int** maze = (int**)malloc(n * sizeof(int*));
	for (int i = 0; i < n; i++)
	{
		maze[i] = (int*)malloc(n * sizeof(int));
	}

	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			fin >> maze[i][j];
		}
	}
	fin.close();
	int m = n;
    auto maze2 = vector <vector<char>>(n, vector<char>(n));
    for (int i = 0; i < n; i++) {
        vector<char> row;
        for (int j = 0; j < n; j++) {
            row.push_back(maze[i][j]);
        }
        maze2[i] = row;
    }
    int str = vhod;
    int stl = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (maze2[i][j] == 1)
                maze2[i][j] = 254;
            if (maze2[i][j] == 0)
                maze2[i][j] = '  ';
            if (maze2[i][j] == 5)
                maze2[i][j] = '  ';
            if (maze2[i][j] == 6)
                maze2[i][j] = '  ';
            if (maze2[i][j] == 8)
                maze2[i][j] = '  ';
        }
    }

    maze2[str][stl] = 253;
	system("cls");
    for (int i = 0; i < n; i++)
    {
		for (int j = 0; j < m; j++)
			cout << maze2[i][j] << " ";
```

```
		cout << endl;
}
	int click = 0;
	int startTime = clock();
	while (TRUE)
	{
		if (GetAsyncKeyState(VK_UP))
		{
			keybd_event(VK_UP, 0, KEYEVENTF_KEYUP, 0);//Отжимаем кнопку
			if (maze2[str - 1][stl] == 32) {
				maze2[str - 1][stl] = 253;
				maze2[str][stl] = '  ';
				str--;
				system("cls");
				for (int i = 0; i < n; i++)
				{
					for (int j = 0; j < m; j++)
						cout << maze2[i][j] << " ";
					cout << endl;
				}
				click++;
			}
		}
		if (GetAsyncKeyState(VK_DOWN))
		{
			keybd_event(VK_DOWN, 0, KEYEVENTF_KEYUP, 0);//Отжимаем кнопку
			if (maze2[str + 1][stl] == 32) {
				maze2[str + 1][stl] = 253;
				maze2[str][stl] = '  ';
				str++;
				system("cls");
				for (int i = 0; i < n; i++)
				{
					for (int j = 0; j < m; j++)
						cout << maze2[i][j] << " ";
					cout << endl;
				}
				click++;
			}
		}
		if (GetAsyncKeyState(VK_RIGHT))
		{
			keybd_event(VK_RIGHT, 0, KEYEVENTF_KEYUP, 0);//Отжимаем кнопку
			if (maze2[str][stl + 1] == 32) {
				maze2[str][stl + 1] = 253;
				maze2[str][stl] = '  ';
				stl++;
				system("cls");
				for (int i = 0; i < n; i++)
				{
					for (int j = 0; j < m; j++)
						cout << maze2[i][j] << " ";
					cout << endl;
				}
				if (maze2[str][stl] == maze2[vihod][n-1]) {
					int endTime = clock();
					double time = (endTime - startTime) / 100000000;
					break;
				}
				click++;
			}
		}
		if (GetAsyncKeyState(VK_LEFT))
		{
```

```
                        keybd_event(VK_LEFT, 0, KEYEVENTF_KEYUP, 0);//Отжимаем кнопку
                        if (maze2[vhod][0] != maze2[str][stl]) {
                                if (maze2[str][stl - 1] == 32) {
                                        maze2[str][stl - 1] = 253;
                                        maze2[str][stl] = '  ';
                                        stl--;
                                        system("cls");
                                        for (int i = 0; i < n; i++)
                                        {
                                                for (int j = 0; j < m; j++)
                                                        cout << maze2[i][j] << " ";
                                                cout << endl;
                                        }
                                        click++;
                                }
                        }
                }
        }
        click++;
        for (int i = 0; i < n; i++) {
                delete[] maze[i];  // удаляем массив
        }
        system("pause >> NUL");
        system("cls");
        cout << "---------------Statistic---------------\n" << "Time to comlete the maze:
" << time << "\nSize maze: " << n << "\nNumber of clicks: " << click;
        random_device rd;
        mt19937 gen(rd());
        uniform_int_distribution<> dist(0, 2);
        int box = dist(gen);
        switch(box) {
        case 0:
                MessageBox(NULL, L"Ну ты и черепаха!", L"Лабиринт пройден!",
MB_ICONASTERISK | MB_OK);
                break;
        case 1:
                MessageBox(NULL, L"А быстрее нельзя?!", L"Лабиринт пройден!",
MB_ICONASTERISK | MB_OK);
                break;
        case 2:
                MessageBox(NULL, L"Черепаха и то быстрее!", L"Лабиринт пройден!",
MB_ICONASTERISK | MB_OK);
                break;
        }
        system("pause >> NUL");
}
```

Файл bot.cpp: функция bot().

```
#include <vector>
#include "Windows.h"
#include <fstream>
#include <time.h>
#include <iostream>
using namespace std;

void bot() {
        int zader;
        system("cls");
        cout << "Agent speed(optimal 300): ";
        cin >> zader;

        int n = 0;
        int vhod = 0;
        int vihod = 0;
```

```
ifstream fin("gen.txt");
fin >> n;
fin >> vhod;
fin >> vihod;

int** maze = (int**)malloc(n * sizeof(int*));
for (int i = 0; i < n; i++)
{
	maze[i] = (int*)malloc(n * sizeof(int));
}


for (int i = 0; i < n; i++) {
	for (int j = 0; j < n; j++) {
		fin >> maze[i][j];
	}
}
fin.close();

int m = n;

auto maze2 = vector <vector<char>>(n, vector<char>(n));
for (int i = 0; i < n; i++) {
	vector<char> row;
	for (int j = 0; j < n; j++) {
		row.push_back(maze[i][j]);
	}
	maze2[i] = row;
}

int str = vhod;
int stl = 0;

for (int i = 0; i < n; i++) {
	for (int j = 0; j < n; j++) {
		if (maze2[i][j] == 1)
			maze2[i][j] = 254;
		if (maze2[i][j] == 0)
			maze2[i][j] = '  ';
		if (maze2[i][j] == 5)
			maze2[i][j] = '  ';
		if (maze2[i][j] == 6)
			maze2[i][j] = '  ';
		if (maze2[i][j] == 8)
			maze2[i][j] = '  ';
	}
}

maze2[str][stl] = 4;
maze2[str][stl + 1] = 253;
stl++;
system("cls");
for (int i = 0; i < n; i++)
{
	for (int j = 0; j < m; j++)
		cout << maze2[i][j] << " ";
	cout << endl;
}

int startTime = clock();
while (TRUE)
{
```

```
                    while ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl] == 32) ||
(maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32))
                    {
                          if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                          if (maze2[str][stl + 1] == 32) {//vpravo
                                maze2[str][stl + 1] = 253;
                                maze2[str][stl] = 4;
                                stl++;
                                system("cls");
                                for (int i = 0; i < n; i++)
                                {
                                      for (int j = 0; j < m; j++)
                                            cout << maze2[i][j] << " ";
                                      cout << endl;
                                }
                                Sleep(zader);
                          }
                          if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                          if (maze2[str - 1][stl] == 32) {//vverh
                                maze2[str - 1][stl] = 253;
                                maze2[str][stl] = 4;
                                str--;
                                system("cls");
                                for (int i = 0; i < n; i++)
                                {
                                      for (int j = 0; j < m; j++)
                                            cout << maze2[i][j] << " ";
                                      cout << endl;

                                }
                                Sleep(zader);
                          }
                          if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                          if (maze2[str + 1][stl] == 32) {//vniz
                                maze2[str + 1][stl] = 253;
                                maze2[str][stl] = 4;
                                str++;
                                system("cls");
                                for (int i = 0; i < n; i++)
                                {
                                      for (int j = 0; j < m; j++)
                                            cout << maze2[i][j] << " ";
                                      cout << endl;
                                }
                                Sleep(zader);
                          }
                          if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                          if (maze2[vhod][0] != maze2[str][stl]) {//vlevo
                                if (maze2[str][stl - 1] == 32) {
                                      maze2[str][stl - 1] = 253;
                                      maze2[str][stl] = 4;
                                      stl--;
                                      system("cls");
                                      for (int i = 0; i < n; i++)
                                      {
                                            for (int j = 0; j < m; j++)
                                                  cout << maze2[i][j] << " ";
                                            cout << endl;
                                      }
                                }
                                Sleep(zader);
                          }
                    }
                    if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
```

```
                    if ((maze2[str - 1][stl] != 32) && (maze2[str + 1][stl] != 32) &&
(maze2[str][stl - 1] != 32) && (maze2[str][stl + 1] != 32)) {
                            while (TRUE) {
                                    if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                                    if ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl]
== 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) { break; }
                                    if (maze2[str][stl + 1] == 4) {//vpravo
                                            maze2[str][stl + 1] = 253;
                                            maze2[str][stl] = 3;
                                            stl++;
                                            system("cls");
                                            for (int i = 0; i < n; i++)
                                            {
                                                    for (int j = 0; j < m; j++)
                                                            cout << maze2[i][j] << " ";
                                                    cout << endl;
                                            }
                                            Sleep(zader);
                                    }
                                    if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                                    if ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl]
== 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) { break; }
                                    if (maze2[str - 1][stl] == 4) {//vverh
                                            maze2[str - 1][stl] = 253;
                                            maze2[str][stl] = 3;
                                            str--;
                                            system("cls");
                                            for (int i = 0; i < n; i++)
                                            {
                                                    for (int j = 0; j < m; j++)
                                                            cout << maze2[i][j] << " ";
                                                    cout << endl;
                                            }
                                            Sleep(zader);
                                    }
                                    if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                                    if ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl]
== 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) { break; }
                                    if (maze2[str + 1][stl] == 4) {//vniz
                                            maze2[str + 1][stl] = 253;
                                            maze2[str][stl] = 3;
                                            str++;
                                            system("cls");
                                            for (int i = 0; i < n; i++)
                                            {
                                                    for (int j = 0; j < m; j++)
                                                            cout << maze2[i][j] << " ";
                                                    cout << endl;
                                            }
                                            Sleep(zader);
                                    }
                                    if (maze2[str][stl] == maze2[vihod][n - 1]) { break; }
                                    if ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl]
== 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) { break; }
                                    if (maze2[str][stl - 1] == 4) {//vlevo
                                            maze2[str][stl - 1] = 253;
                                            maze2[str][stl] = 3;
                                            stl--;
                                            system("cls");
                                            for (int i = 0; i < n; i++)
                                            {
                                                    for (int j = 0; j < m; j++)
                                                            cout << maze2[i][j] << " ";
                                                    cout << endl;
```

```
                                        }
                                        Sleep(zader);
                                }
                                if ((maze2[str - 1][stl] == 32) || (maze2[str + 1][stl]
== 32) || (maze2[str][stl - 1] == 32) || (maze2[str][stl + 1] == 32)) { break; }
                                }
                        }
                if (maze2[str][stl] == maze2[vihod][n - 1]) {
                        int endTime = clock();
                        double time = (endTime - startTime) / 1000;
                        break;
                }
        }
        system("pause >> NUL");
        system("cls");
        cout << "---------------Statistic---------------\n" << "Time to comlete the maze:
" << time << "\nSize maze: " << n;
        system("pause >> NUL");
}
```