МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

**(национальный исследовательский университет)**

Институт № 7 «Интеллектуальные и робототехнические системы»

Кафедра 703

**Курсовая работа по дисциплине**

**«Технологии разработки программного обеспечения»**

Работу выполнил

студент группы М70-407С-21

Полушкин Фёдор Юрьевич\_\_\_\_\_\_\_\_\_\_\_\_

Работу проверил

Старший преподаватель кафедры 703

Барчев Николай Борисович\_\_\_\_\_\_\_\_\_\_\_\_

Дата сдачи отчета

\_\_\_\_\_\_\_\_\_\_\_ 2025 года

Москва

2025

**Задание:**

Описать класс, реализующий стек. Написать программу, использующую этот класс для отыскания прохода по лабиринту. Лабиринт представляется в виде матрицы, состоящей из квадратов. Каждый квадрат либо открыт, либо закрыт. Вход в закрытый квадрат запрещен. Если квадрат открыт, то вход в него возможен со стороны, но не с угла. Каждый квадрат определяется его координатами в матрице. После отыскания прохода программа печатает найденный путь в виде координат квадратов.  
  
*Дополнительное задание*: реализовать генерацию лабиринта заданного размера с заданным параметром генерации (seed)

Оглавление

[Псевдокод алгоритмов решения задач. 3](#_Toc154411348)

[Сведения о программной реализации. 9](#_Toc154411349)

[I Сведения о выбранном языке программирования 9](#_Toc154411350)

[II Описание входных и выходных данных 9](#_Toc154411351)

[III Описание спроектированных программных единиц 9](#_Toc154411352)

[IV Инструкция пользователя 14](#_Toc154411353)

[Листинг программы 19](#_Toc154411354)

[Тестирование 32](#_Toc154411355)

Псевдокод алгоритмов решения задач.

**Алгоритм поиска в глубину**

**Функция DFS**  
*Входные параметры:(путь\_к\_карте, тип\_обхода)*

карта, стартовая\_позиция := загрузить\_и\_распарсить\_карту(путь\_к\_карте)

стек := новый пустой стек

стек.вставить(стартовая\_позиция)

решение\_найдено := ЛОЖЬ

**вызвать** *DFS\_Шаг(карта, стек, решение\_найдено, тип\_обхода)*

если решение\_найдено: вернуть стек (он содержит путь к цели)

иначе:

"Решение не найдено"

**Функция DFS\_Шаг**  
*Входные параметры:(карта, стек, решение\_найдено, тип\_обхода)*

направления := определить\_порядок\_обхода(тип\_обхода)

для каждого направления в направления:

**если** решение\_найдено:

продолжить

следующая\_позиция := рассчитать\_координаты  
 (текущая\_позиция\_в\_стеке,   
 направление)

**если** (следующая\_позиция внутри границ карты) и   
 (не содержится в стеке):

элемент := карта[следующая\_позиция]

**если** элемент == ДОСТУПНО:

стек.вставить(следующая\_позиция)

**вызвать** *DFS\_Шаг(карта, стек, решение\_найдено, тип\_обхода)*

**иначе если** элемент == КОНЕЦ:

стек.вставить(следующая\_позиция)

решение\_найдено := ИСТИНА

**если** решение\_найдено:

выйти из функции

удалить\_последний\_элемент\_из\_стека (откат)

**Алгоритм генерации случайной карты**

**Функция Сгенерировать\_Случайную\_Карту***Входные данные (seed, размер\_карты)*

создать пустую карту заданного размера и заполнить её ДОСТУПНЫМИ ячейками

**вызвать** *Деление(карта, размер\_карты, seed)*

установить стартовую точку в (0, 0)

установить конечную точку в (последняя\_строка, последний\_столбец)

вернуть карту и координаты старта

**Функция Деление***Входные данные (карта, размер\_карты, seed)*

если ширина ≤ 2 или высота ≤ 2: **выход** (слишком мала для деления)

инициализировать генератор случайных чисел с seed

выбрать случайную позицию горизонтальной стены (исключая края)

выбрать случайную позицию вертикальной стены (исключая края)

создать горизонтальную стену на выбранной строке, за исключением 1–2   
 "дверей" (отверстий)

создать вертикальную стену на выбранном столбце, за исключением 1–2   
 "дверей"

Разбить карту на 4 подкарты:

левый верх

левый низ

правый верх

правый низ

Для каждой подкарты:

рекурсивно **вызвать** *Деление(подкарта, её размер, seed)*

Объединить все 4 подкарты обратно в основную карту

Обеспечить наличие проходов между регионами (открытые ячейки рядом со   
 стенами)

**Функция Разделить\_На\_Подкарты***Входные данные (карта, размер\_карты, позиция\_гор\_стены, позиция\_верт\_стены)*

создать пустые 4 подкарты соответствующего размера

скопировать данные из основной карты в соответствующие подкарты в   
 зависимости от координат

для каждой непустой подкарты: **вызвать** *Деление(подкарта, её\_размер, seed)*

скопировать подкарты обратно в основную карту

при необходимости открыть дополнительные проходы в местах пересечения   
 (см. соседние ячейки)

**Описание реализации пользовательских типов**

**Класс Динамический\_Массив<Тип>:**

Переменные:

массив: массив из Тип

текущая\_длина: число элементов в массиве

выделенный\_размер: сколько элементов может уместиться без   
 перераспределения

Конструктор():

выделить память под 1 элемент

текущая\_длина = 0

выделенный\_размер = 1

Конструктор(n):

выделить память под n элементов

текущая\_длина = 0

выделенный\_размер = n

Метод resize(n):

если текущая\_длина > n:

выбросить исключение (нельзя уменьшать до меньше текущей длины)

создать новый массив размера n

скопировать все существующие элементы в новый массив

удалить старый массив

присвоить новый массив в переменную массив

обновить выделенный\_размер

Метод empty():

вернуть (текущая\_длина == 0)

Метод size():

вернуть текущая\_длина

Метод contain(элемент):

для каждого i от 0 до текущая\_длина - 1:

если массив[i] == элемент:

вернуть true

вернуть false

Метод invert():

создать новый массив той же длины

для каждого i от 0 до текущая\_длина - 1:

новый\_массив[текущая\_длина - 1 - i] = массив[i]

удалить старый массив

присвоить новый\_массив в переменную массив

Метод clear():

установить текущая\_длина = 0

Конструктор копирования(другой\_массив):

выделить память под размер другого массива

скопировать все элементы из другого массива

Оператор присваивания(другой\_массив):

если текущий и другой\_массив — не один и тот же:

удалить текущий массив

выделить новый массив нужного размера

скопировать все элементы

**Класс Стек<Тип> наследует Динамический\_Массив<Тип>:**

Конструктор():

вызвать конструктор базового класса (на 1 элемент)

Конструктор(n):

вызвать конструктор базового класса (на n элементов)

Метод push(новый\_элемент):

если текущая\_длина >= выделенный\_размер:

удвоить размер массива

записать новый\_элемент в массив[текущая\_длина]

увеличить текущая\_длина

Метод pop():

если стек пуст:

выбросить исключение

уменьшить текущая\_длина на 1

вернуть массив[текущая\_длина]

Метод top():

если стек пуст:

выбросить исключение

вернуть массив[текущая\_длина - 1]

Сведения о программной реализации.

**I Сведения о выбранном языке программирования**

Для разработки был применен язык ISO C++20 в среде разработки Microsoft Visual Studio

**II Описание входных и выходных данных**

• Входными данными для программы является карта “лабиринта” заданного формата, которая может быть введена пользователем в консоли, сгенерирована при помощи встроенных средств программы или представлена в виде внешнего файла.

• Выходными данными для программы является вектор решения нахождения пути для заданной карты, или уведомление об отсутствии решения.

**III Описание спроектированных программных единиц**

Реализация классов динамического массива (суперкласс) и стек (класс-потомок):

**Динамический массив (суперкласс)**

template <typename dyn\_array\_type>

class TDynamic\_array {

protected:

dyn\_array\_type\* body; // Dynamic array contents

size\_t body\_size; // The maximum number of elements that can be in   
 // the current memory allocated for the dynamic   
 // array

size\_t body\_len; // Number of items in the dynamic array

public:

// Creating a dynamic array with allocated memory for 1 element

TDynamic\_array();

// Creating a dynamic array with allocated memory for n-elements

TDynamic\_array(size\_t n);

virtual ~TDynamic\_array();

// Changing the allocated memory for the stack

void resize(size\_t n;

// Checking for emptiness

bool empty() const;

// Current stack size (number of items)

size\_t size() const;

// Checking for the content of an element

bool contain(const dyn\_array\_type& el) const;

// Inverting data

void invert();

// Clear data

void clear();

// Copy-constructor

TDynamic\_array(const TDynamic\_array& other);

// Assignment

TDynamic\_array& operator=(const TDynamic\_array& other);

};

**Стек (класс-потомок)**

template <typename stack\_type>

class TStack : public TDynamic\_array<stack\_type> {

public:

TStack() : TDynamic\_array<stack\_type>();

// Creating a stack with allocated memory for n-elements

TStack(size\_t n) : TDynamic\_array<stack\_type>(n);

// Removing the top element of the stack

stack\_type pop() ;

// Adding an item to the top of the stack

void push(const stack\_type& new\_el);

// Viewing the top element of the stack

stack\_type top() const;

};

**В программе были объявлены следующие типы:**

namespace MapSymbolsMask {

char unav\_all = '|';

char unav\_hor = '\_';

char unav\_ver = '|';

char av = '.';

char ed = 'F';

char st = 'S';

char error\_data = 'e';

};

struct node {

size\_t x = std::string::npos;

size\_t y = std::string::npos;

node();

node(size\_t set\_x, size\_t set\_y) : x(set\_x), y(set\_y) {};

void set\_cords(size\_t set\_x, size\_t set\_y);

void operator=(const node& other);

bool operator==(const node& other);

void operator++(void);

};

enum class el\_type {

START = (int)'S', AVAILABLE = (int)'.', UNAVAILABLE = (int)'X',   
 END = (int)'O'

};

using map\_type = std::vector<std::vector<el\_type>>;

using map\_line = std::vector<el\_type>;

const std::unordered\_set<el\_type> availible\_el = { START , AVAILABLE ,   
 UNAVAILABLE , END };

enum menu\_point {

zero = 0x30, one, two, three, four, five, six, manual = 0x6d,   
 print = 0x70, randmap = 0x72

};

enum colors

{

green = 10, purple = 13, blue = 11, red = 12, yellow = 14, white = 15

};

enum class sides {

left, down, right, up, no\_side

};

enum class type\_procedure {

clockwise, counterclockwise, random

};

В программе были реализованы следующие программные единицы:

|  |  |  |  |
| --- | --- | --- | --- |
| название программной единицы | входные параметры | назначение программной единицы | возвращаемое значение |
| Программные единицы, реализованные на языке C++ Стандарта ISO C++20 | | | |
| inline std::pair<map\_type,node> map\_parsing (std::istream& map\_stream) | **std::istream& map\_stream**: ссылка на поток, содержащий информацию о карте | Приведение информации о карте в вид, программно-читаемый и применимый для дальнейшей работы в ПО | std::pair<map\_type,node>: Пара значений 1. Двумерный массив представления карты 2. Пользовательская структура представления точки в двумерном пространстве |
| inline std::pair<map\_type, node> rand\_map\_generate(const unsigned int seed, const std::pair<size\_t, size\_t>& map\_size = { 10, 10 }) | **const unsigned int seed**: значение для генерации случайной карты **const std::pair<size\_t, size\_t>& map\_size** = { 10, 10 }: размер случайно-генерируемой карты, задаваемый парой значений (первое - размер по Y; второе - размер по X) | Создание случайной карты, с гарантированной возможностью нахождения пути | std::pair<map\_type,node>: Пара значений 1. Двумерный массив представления карты 2. Пользовательская структура представления точки в двумерном пространстве |
| static inline void division\_step(map\_type& cur\_map, const std::pair<size\_t, size\_t>& map\_size, const unsigned int seed) | **map\_type& cur\_map**: обрабатываемая часть карты const unsigned int seed: значение для генерации случайной карты **const std::pair<size\_t, size\_t>& map\_size** = { 10, 10 }: размер случайно-генерируемой карты, задаваемый парой значений (первое - размер по Y; второе - размер по X) | Разбиение карты на четыре элемента путем создания вертикальной и горизонтальной стенки с гарантией наличия пути в каждую доступную точку карты | void (программная единица модифицирует входной параметр cur\_map) |
| TStack<node> dfs(std::filesystem::path map\_path, type\_procedure proc\_type = clockwise) | **std::filesystem::path map\_path**: путь до файла, содержащего информацию о карте **type\_procedure proc\_type** = clockwise: выбор направлений обхода при работе алгоритма | Нахождения пути от точки начала до точки завершения при наличии пути (выброс исключения об отсутствии доступного пути, если нет решения) | TStack<node>: стек, содержащий вектор решения |
| TStack<node> dfs(std::pair<map\_type, node> map\_area, type\_procedure proc\_type = clockwise) | **std::pair<map\_type,node>**: Пара значений 1. Двумерный массив представления карты 2. Пользовательская структура представления точки в двумерном пространстве **type\_procedure proc\_type** = clockwise: выбор направлений обхода при работе алгоритма | Нахождения пути от точки начала до точки завершения при наличии пути (выброс исключения об отсутствии доступного пути, если нет решения) | TStack<node>: стек, содержащий вектор решения |
| static void dfs\_step(const map\_type& cur\_map, TStack<node>& solution, bool& solution\_find, type\_procedure proc\_type) | **const map\_type& cur\_map**: карта, в рамках которой происходит поиск   **TStack<node>& solution**: вектор решения **bool& solution\_find**: флаг выхода из рекурсии **type\_procedure proc\_type**: выбор направлений обхода при работе алгоритма | Шаг алгоритма поиска в глубину в одну сторону до нахождения препятствия | void (программная единица модифицирует входной параметр solution и флаг вызода из рекурсии solution\_find) |
| void menu() | **void** | Отображает главное меню программы и обрабатывает выбор пользователя. | void |
| static void print\_map(map\_type& current\_map, TStack<node> sln, std::ostream& out = std::cout) | **map\_type& current\_map**: карта для отображения **TStack<node> sln**: стек решения **std::ostream& out**: поток вывода (по умолчанию std::cout) | Выводит карту и путь решения на экран или в файл. | void |
| static void exit\_message(bool& exit\_flag) | **bool& exit\_flag**: флаг завершения программы | Выводит сообщение о завершении и изменяет флаг выхода. | void |
| static void incorrect\_data\_message(const char err[]) | **const char err[]**: сообщение об ошибке | Отображает сообщение о некорректных данных. | void |
| static void incorrect\_file\_message() | **void** | Выводит сообщение об ошибке при работе с файлом. | void |
| static void print\_manual() | **void** | Выводит краткое руководство по использованию программы. | void |
| static bool check\_file\_name(const std::string& file\_name) | **const std::string& file\_name**: имя файла для проверки | Проверяет корректность имени файла. | bool: true, если имя корректно |
| static bool check\_file\_exist(std::string& file\_name) | **std::string& file\_name**: имя файла | Проверяет существование файла. | bool: true, если файл существует |
| static void set\_color(std::string, int, std::ostream& out = std::cout) | **std::string**: строка для вывода **int**: цвет **std::ostream& out**: поток вывода (по умолчанию std::cout) | Выводит текст в указанном цвете в консоль или файл. | void |
| static std::filesystem::path find\_file\_menu(const std::filesystem::path& directory) | **const std::filesystem::path& directory**: путь к папке | Предлагает пользователю выбрать файл из заданной директории. | std::filesystem::path: путь к выбранному файлу |
| static int file\_open\_menu(std::string filename) | **std::string filename**: имя файла | Открывает файл и возвращает код результата. | int: код результата |
| static void map\_load\_tf(map\_type& exemple) | **map\_type& exemple**: карта, в которую производится загрузка | Загружает карту из текстового файла. | void |
| static void sln\_load\_tf(TStack<node>& sln) | **TStack<node>& sln**: стек, в который загружается путь | Загружает путь решения из текстового файла. | void |
| static void random\_map\_generate\_menu(std::pair<map\_type&, node&> &start\_info) | **std::pair<map\_type&, node&> &start\_info**: ссылка на пару карта-старт | Интерфейс генерации случайной карты через меню. | void |

**IV Инструкция пользователя**

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

*Рис. 1 – Встречающее окно программы*

В программе доступны следующие функции:

1. Найти решение для карты, отображенной вверху консоли под текстом “current map of area” (для действия необходимо ввести 1)
2. Ввести карту, используя консоль (для действия необходимо ввести 2)
3. Ввести карту, находящуюся в файле (для действия необходимо ввести 3)
4. Сохранить текущую карту в файл (для действия необходимо ввести 4)
5. Очистить текущую карту (для действия необходимо ввести 5)
6. Поменять выбранный порядок обхода (для действия необходимо ввести 6)
7. Создать случайную карту (для действия необходимо ввести r)
8. Записать вектор решения в файл (для действия необходимо ввести p)
9. Открыть мануал программы ((для действия необходимо ввести m)

**При выборе “нахождения решения”:**

Изображение выглядит как снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

*Рис. 2 – Нахождение решения и его вывод в консоль*

**При выборе “ввода карты из консоли”:**

Появляется возможность вводить карту используя следующие обозначения:  
X – стена/преграда  
. – доступная для прохождения “местность”  
O – Out(выход)/конченая точка маршрута  
S – Start(начало)/стартовая точка маршрута

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

*Рис. 3 – Ввод карты в консоль*

Введенный маршрут должен заканчиваться сломов “/end” для подтверждения выхода из режима ввода

Введенная карта должна содержать ровно одну точку начала и ровно одну точку окончания

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

*Рис. 4 – Сообщение о невозможности ввода двух точек начала/окончания маршрута*

**При выборе ввода информации из файла**

**Изображение выглядит как снимок экрана, текст, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

*Рис. 5 – Ввод пути до файла, если нажать enter, выведутся все файлы в директории программы*

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

*Рис. 6 – Отображение всех файлов в директории программы*

**Сохранение в файл**

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

*Рис. 7 – Меню ввода файла, для сохранения*

При попытке сохранения в существующий файл программа уведомляет пользователя и просит подтверждение

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

*Рис. 8 – Попытка сохранения в существующий файл*

**Очистка карты**

**Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.**

*Рис. 9 – Очистка карты*

**Выбор направления обхода DFS**

При нажатии 6 порядок обхода меняется в зацикленном порядке (clockwise – по часовой стрелке, counterclockwise – против часовой стрелки, random – в случайном порядке)

**Генерация случайной карты**

При генерации пользователь может ввести два параметра:

1. Размер карты (два числа, определяющие размер введенной карты)
2. Seed (значения, влияющее на вид генерации карты)

Так же пользователь может не вводить указанные значения (пропускать при помощи enter), тогда в размер будет установлено 10X10, а seed будет зависеть от времени, выставленного в системе

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

*Рис. 10 – Значение генерации по умолчанию*

Изображение выглядит как снимок экрана, текст, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

*Рис. 11 – Пользовательские значения генерации*

**Печать вектора решения**

Вектор решения будет сохранен в пользовательский файл в меню, аналогичном меню сохранения карты

**Мануал программы**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

*Рис. 12 – Мануал программы*

Листинг программы

*map\_parsing.hpp*

#pragma once  
#include **<iostream>**#include **<vector>**#include **<unordered\_set>**#include **<filesystem>**#include **<fstream>**#include **<ctime>  
  
namespace** MapSymbolsMask {  
 **char** unav\_all = **'|'**;  
 **char** unav\_hor = **'\_'**;  
 **char** unav\_ver = **'|'**;  
 **char** av = **'.'**;  
 **char** ed = **'F'**;  
 **char** st = **'S'**;  
 **char** error\_data = **'e'**;  
};  
  
*// A point in 2D space***struct** node {  
 size\_t x = std::string::npos;  
 size\_t y = std::string::npos;  
 node() {};  
 node(size\_t set\_x, size\_t set\_y) : x(set\_x), y(set\_y) {};  
 **void** set\_cords(size\_t set\_x, size\_t set\_y) {  
 **this**->x = set\_x;   
 **this**->y = set\_y;  
 }  
  
 **void operator**=(**const** node& other) {  
 **this**->x = other.x;  
 **this**->y = other.y;  
 }  
  
 **bool operator**==(**const** node& other) {  
 **return** (**this**->x == other.x && **this**->y == other.y);  
 }  
 **void operator**++(**void**) {  
 **this**->x++;  
 **this**->y++;  
 }  
};  
*// Map body element***enum class** el\_type {  
 ***START*** = (**int**)**'S'**, ***AVAILABLE*** = (**int**)**'.'**, ***UNAVAILABLE*** = (**int**)**'X'**, ***END*** = (**int**)**'O'**};  
**using enum** el\_type;  
**using** map\_type = std::vector<std::vector<el\_type>>;  
**using** map\_line = std::vector<el\_type>;  
**const** std::unordered\_set<el\_type> availible\_el = **{ *START*** , ***AVAILABLE*** , ***UNAVAILABLE*** , ***END* }**;  
  
*/// <summary>  
/// Description: parsing a stream to get a map of available movements  
/// </summary>  
/// <param name="map\_stream">stream with the map of area</param>  
/// <returns>map of available movements and starting point</returns>***inline** std::pair<map\_type,node> map\_parsing (std::istream& map\_stream) {  
 node start\_node;  
 map\_type map\_array;  
  
 size\_t cur\_y = 0; *// The current Y coordinate* size\_t start\_counter = 0; *// Counter for the number of starts encountered on the map* size\_t end\_counter = 0; *// Counter for the number of completed route points encountered* **bool** end\_flag = **false**;  
 **if** (**not** map\_stream) **throw "File dont exist"**;  
 **while** (**not** map\_stream.eof() **and not** end\_flag) {  
 std::string curr\_line; *// The map string received from the stream* map\_line cur\_map\_line; *// A map line converted to the map\_line type* std::getline(map\_stream, curr\_line);  
 **if** (curr\_line != **"/end"**) {  
 size\_t cur\_x = 0; *// The current X coordinate* **for** (cur\_x; cur\_x < curr\_line.length(); ++cur\_x) {  
 *// Converting a char element to a matrix elements* el\_type map\_element;  
 **if** (curr\_line[cur\_x] == MapSymbolsMask::unav\_all **or** curr\_line[cur\_x] == MapSymbolsMask::unav\_hor **or** curr\_line[cur\_x] == MapSymbolsMask::unav\_ver) {  
 map\_element = ***UNAVAILABLE***;  
 }  
 **else if** (curr\_line[cur\_x] == MapSymbolsMask::av) {  
 map\_element = ***AVAILABLE***;  
 }  
 **else if** (curr\_line[cur\_x] == MapSymbolsMask::st) {  
 map\_element = ***START***;  
 }  
 **else if** (curr\_line[cur\_x] == MapSymbolsMask::st) {  
 map\_element = ***END***;  
 }  
 **else** {  
 map\_element = (el\_type)MapSymbolsMask::error\_data;  
 }  
   
 *// Checking for the content of valid characters only* **if** (**not** availible\_el.contains(map\_element)) **throw "Map isn't correct"**;  
  
 **if** (map\_element == ***START***) { *// Find out the coordinates of the beginning of the path* start\_counter++;  
 start\_node.set\_cords(cur\_x, cur\_y);  
 }  
 **if** (map\_element == ***END***) {  
 end\_counter++;  
 }  
 cur\_map\_line.push\_back(map\_element);  
 }  
 *// The map should not contain more or less than one beginning and end of the route.* **if** (start\_counter > 1 || end\_counter > 1)  
 **throw "the map does not have the possibility of finding a deterministic route"**;  
 map\_array.push\_back(cur\_map\_line);  
 cur\_y++;  
 }  
 **else** {  
 end\_flag = **true**;  
 }  
 }  
 **if** (start\_counter != 1 || end\_counter != 1)  
 **throw "the map does not have the possibility of finding a deterministic route"**;  
 **return** { map\_array, start\_node }; *// return the map and the starting point*}  
  
**static inline void** division\_step(map\_type& cur\_map, **const** std::pair<size\_t, size\_t>& map\_size, **const unsigned int** seed) {  
 **auto** gen\_sub\_map = [seed](map\_type& cur\_map, **const** std::pair<size\_t, size\_t>& map\_size,   
 size\_t hor\_wall\_pos, size\_t ver\_wall\_pos) {  
 size\_t x\_size = map\_size.second;  
 size\_t y\_size = map\_size.first;  
  
 map\_type sub\_map1;  
 map\_type sub\_map2;  
 map\_type sub\_map3;  
 map\_type sub\_map4;  
 *// left-upper corner* **if** (hor\_wall\_pos < INT32\_MAX **and** ver\_wall\_pos < INT32\_MAX) {  
 sub\_map1.resize(hor\_wall\_pos, map\_line(ver\_wall\_pos));  
 }  
 *// left-lower corner* **if** ((y\_size - hor\_wall\_pos - 1) < INT32\_MAX **and** ver\_wall\_pos < INT32\_MAX) {  
 sub\_map2.resize((**int**)y\_size - (**int**)hor\_wall\_pos - 1, map\_line(ver\_wall\_pos));  
 }  
 *// right-upper corner* **if** (hor\_wall\_pos < INT32\_MAX **and** (x\_size - ver\_wall\_pos - 1) < INT32\_MAX) {  
 sub\_map3.resize(hor\_wall\_pos, map\_line(x\_size - ver\_wall\_pos - 1));  
 }  
 *// right-lower corner* **if** ((y\_size - hor\_wall\_pos - 1) < INT32\_MAX **and** (x\_size - ver\_wall\_pos - 1) < INT32\_MAX) {  
 sub\_map4.resize(y\_size - hor\_wall\_pos - 1, map\_line((**int**)x\_size - (**int**)ver\_wall\_pos - 1));  
 }  
  
 **for** (size\_t x = 0; x < x\_size; ++x) {  
 **for** (size\_t y = 0; y < y\_size; ++y) {  
 **if** (x < ver\_wall\_pos **and** y < hor\_wall\_pos)  
 sub\_map1[y][x] = cur\_map[y][x];  
 **else if** (x < ver\_wall\_pos **and** y > hor\_wall\_pos)  
 sub\_map2[y - hor\_wall\_pos - 1][x] = cur\_map[y][x];  
 **else if** (x > ver\_wall\_pos **and** y < hor\_wall\_pos)  
 sub\_map3[y][x - ver\_wall\_pos - 1] = cur\_map[y][x];  
 **else if** (x > ver\_wall\_pos **and** y > hor\_wall\_pos)  
 sub\_map4[y-hor\_wall\_pos - 1][x-ver\_wall\_pos - 1] = cur\_map[y][x];  
 }  
 }  
  
 **if** (**not** sub\_map1.empty()) {  
 division\_step(sub\_map1, { sub\_map1.size(), sub\_map1[0].size() }, seed);  
 }  
 **if** (**not** sub\_map2.empty()) {  
 division\_step(sub\_map2, { sub\_map2.size(), sub\_map2[0].size() }, seed);  
 }  
 **if** (**not** sub\_map3.empty()) {  
 division\_step(sub\_map3, { sub\_map3.size(), sub\_map3[0].size() }, seed);  
 }  
 **if** (**not** sub\_map4.empty()) {  
 division\_step(sub\_map4, { sub\_map4.size(), sub\_map4[0].size() }, seed);  
 }  
  
 **for** (size\_t x = 0; x < x\_size; ++x) {  
 **for** (size\_t y = 0; y < y\_size; ++y) {  
 **if** (x < ver\_wall\_pos **and** y < hor\_wall\_pos)  
 cur\_map[y][x] = sub\_map1[y][x];  
 **if** (x == ver\_wall\_pos - 1 **and** (cur\_map[y][x + 1] == ***AVAILABLE* and** cur\_map[y][x] == ***UNAVAILABLE***))  
 cur\_map[y][x] = ***AVAILABLE***;  
 **if** (y == hor\_wall\_pos - 1 **and** (cur\_map[y+1][x] == ***AVAILABLE* and** cur\_map[y][x] == ***UNAVAILABLE***))  
 cur\_map[y][x] = ***AVAILABLE***;  
 **else if** (x < ver\_wall\_pos **and** y > hor\_wall\_pos)  
 cur\_map[y][x] = sub\_map2[y - hor\_wall\_pos - 1][x];  
 **if** (x == ver\_wall\_pos - 1 **and** (cur\_map[y][x + 1] == ***AVAILABLE* and** cur\_map[y][x] == ***UNAVAILABLE***))  
 cur\_map[y][x] = ***AVAILABLE***;  
 **if** (y == hor\_wall\_pos + 1 **and** (cur\_map[y - 1][x] == ***AVAILABLE* and** cur\_map[y][x] == ***UNAVAILABLE***))  
 cur\_map[y][x] = ***AVAILABLE***;  
 **else if** (x > ver\_wall\_pos **and** y < hor\_wall\_pos)  
 cur\_map[y][x] = sub\_map3[y][x - ver\_wall\_pos - 1];  
 **if** (x == ver\_wall\_pos + 1 **and** (cur\_map[y][x - 1] == ***AVAILABLE* and** cur\_map[y][x] == ***UNAVAILABLE***))  
 cur\_map[y][x] = ***AVAILABLE***;  
 **if** (y == hor\_wall\_pos - 1 **and** (cur\_map[y + 1][x] == ***AVAILABLE* and** cur\_map[y][x] == ***UNAVAILABLE***))  
 cur\_map[y][x] = ***AVAILABLE***;  
 **else if** (x > ver\_wall\_pos **and** y > hor\_wall\_pos)  
 cur\_map[y][x] = sub\_map4[y - hor\_wall\_pos - 1][x - ver\_wall\_pos - 1];  
 **if** (x == ver\_wall\_pos + 1 **and** (cur\_map[y][x - 1] == ***AVAILABLE* and** cur\_map[y][x] == ***UNAVAILABLE***))  
 cur\_map[y][x] = ***AVAILABLE***;  
 **if** (y == hor\_wall\_pos + 1 **and** (cur\_map[y - 1][x] == ***AVAILABLE* and** cur\_map[y][x] == ***UNAVAILABLE***))  
 cur\_map[y][x] = ***AVAILABLE***;  
 }  
 }  
 };  
  
 srand(seed);  
 **bool** is\_second\_door\_horizontal = rand()%2;  
 **bool** generate\_hor\_wall = map\_size.first > 2;  
 **bool** generate\_ver\_wall = map\_size.second > 2;  
 **if** (**not** generate\_hor\_wall || **not** generate\_ver\_wall) **return**;  
  
   
 size\_t hor\_wall\_pos = (generate\_hor\_wall) ? (rand() % (map\_size.first-2)) + 1 : 0;  
 size\_t ver\_wall\_pos = (generate\_ver\_wall) ? (rand() % (map\_size.second-2)) + 1 : 0;  
 **if** (generate\_hor\_wall) { *// Create horizontal wall and door* **for** (size\_t x = 0; x < map\_size.second; ++x) {  
 cur\_map[hor\_wall\_pos][x] = ***UNAVAILABLE***;  
 }  
 size\_t hor\_door\_pos1 = (rand() % (ver\_wall\_pos));  
 size\_t hor\_door\_pos2 = ((rand() % (map\_size.second - ver\_wall\_pos-1)) + ver\_wall\_pos) + 1;  
 **if** (**not** is\_second\_door\_horizontal) {  
 **if** (rand() % 2 == 0) {  
 cur\_map[hor\_wall\_pos][hor\_door\_pos1] = ***AVAILABLE***;  
 }  
 **else** {  
 cur\_map[hor\_wall\_pos][hor\_door\_pos2] = ***AVAILABLE***;  
 }  
 }  
 **else** {  
 cur\_map[hor\_wall\_pos][hor\_door\_pos1] = ***AVAILABLE***;  
 cur\_map[hor\_wall\_pos][hor\_door\_pos2] = ***AVAILABLE***;  
 }  
 }  
  
 **if** (generate\_ver\_wall) { *// Create vertical wall and door* size\_t ver\_door\_pos1 = (rand() % (hor\_wall\_pos));  
 size\_t ver\_door\_pos2 = (rand() % (map\_size.first - hor\_wall\_pos-1)) + hor\_wall\_pos + 1;  
 **for** (size\_t y = 0; y < map\_size.first; ++y) {  
 cur\_map[y][ver\_wall\_pos] = ***UNAVAILABLE***;  
 }  
 **if** (is\_second\_door\_horizontal) {  
 **if** (rand() % 2 == 0) {  
 cur\_map[ver\_door\_pos1][ver\_wall\_pos] = ***AVAILABLE***;  
 }  
 **else** {  
 cur\_map[ver\_door\_pos2][ver\_wall\_pos] = ***AVAILABLE***;  
 }  
 }  
 **else** {  
 cur\_map[ver\_door\_pos1][ver\_wall\_pos] = ***AVAILABLE***;  
 cur\_map[ver\_door\_pos2][ver\_wall\_pos] = ***AVAILABLE***;  
 }  
 }  
 gen\_sub\_map(cur\_map, { cur\_map.size(), cur\_map[0].size() }, hor\_wall\_pos, ver\_wall\_pos);  
}  
  
**inline** std::pair<map\_type, node> rand\_map\_generate(**const unsigned int** seed, **const** std::pair<size\_t, size\_t>& map\_size = { 10, 10 }) {  
 map\_type random\_map;  
 **for** (size\_t x = 0; x < map\_size.second; ++x) {  
 map\_line base\_line;  
 **for** (size\_t y = 0; y < map\_size.first; ++y) {  
 base\_line.push\_back(***AVAILABLE***);  
 }  
 random\_map.push\_back(base\_line);  
 }  
 division\_step(random\_map, { random\_map.size(), random\_map[0].size() }, seed);  
   
 random\_map[0][0] = ***START***;  
 random\_map[map\_size.first-1][map\_size.second-1] = ***END***;  
 **return** { random\_map, { 0,0 } };  
}

*user\_stack.hpp*

#pragma once  
  
**template** <**typename** dyn\_array\_type>  
**class** TDynamic\_array {  
**protected**:  
 dyn\_array\_type\* body; *// Dynamic array contents* size\_t body\_size; *// The maximum number of elements that can be in the current memory allocated for the dynamic array* size\_t body\_len; *// Number of items in the dynamic array***public**:  
 *// Creating a dynamic array with allocated memory for 1 element* TDynamic\_array() {   
 **this**->body = **new** dyn\_array\_type[1];  
 **this**->body\_len = 0;  
 **this**->body\_size = 1;  
 };  
  
 *// Creating a dynamic array with allocated memory for n-elements* TDynamic\_array(size\_t n) {  
 **this**->body = **new** dyn\_array\_type[n];  
 **this**->body\_len = 0;  
 **this**->body\_size = n;  
 };  
  
 **virtual** ~TDynamic\_array() {  
 **delete**[] **this**->body;  
 }  
  
 *// Changing the allocated memory for the stack* **void** resize(size\_t n) {  
 **if** (**this**->body\_len > n) { *// When trying to make a size reduction, we throw an exception* **throw "Resize error: an attempt to reduce the size"**;  
 }  
  
 dyn\_array\_type\* temp\_body = **new** dyn\_array\_type[n];  
 **for** (size\_t i = 0; i < **this**->body\_len; ++i) {  
 temp\_body[i] = body[i];  
 }  
 **delete**[] body; *// Deleting the old array bound to the stack body* body = temp\_body; *// Reconnecting to the memory that the temporary pointer points to* **this**->body\_size = n;  
 };  
  
 *// Checking for emptiness* **bool** empty() **const** {  
 **return not**((**bool**)**this**->body\_len); *// If the stack size is not zero, then it is not empty* }  
  
 *// Current stack size (number of items)* size\_t size() **const** {  
 **return this**->body\_len;  
 }  
  
 *// Checking for the content of an element* **bool** contain(**const** dyn\_array\_type& el) **const** {  
 **for** (size\_t i = 0; i < **this**->body\_len; ++i) {  
 **if** (el == **this**->body[i]) **return true**;  
 }  
 **return false**;  
 }  
 *// Inverting data* **void** invert() {  
 dyn\_array\_type\* inverted\_array = **new** dyn\_array\_type[**this**->body\_len];  
 **for** (size\_t i = 0; i < **this**->body\_len; ++i) {  
 inverted\_array[**this**->body\_len - i - 1] = **this**->body[i];  
 }  
 dyn\_array\_type\* tmp\_ptr = **this**->body;  
 **this**->body = inverted\_array;  
 **delete**[] tmp\_ptr;  
 }  
 *// Clear data* **void** clear() {  
 **this**->body\_len = 0;  
 }  
 *// Copy-constructor* TDynamic\_array(**const** TDynamic\_array& other) {  
 **this**->body\_size = other.body\_size;  
 **this**->body\_len = other.body\_len;  
 **this**->body = **new** dyn\_array\_type[body\_size];  
 **for** (size\_t i = 0; i < body\_len; ++i)  
 **this**->body[i] = other.body[i];  
 }  
 *// Assignment* TDynamic\_array& **operator**=(**const** TDynamic\_array& other) {  
 **if** (**this** != &other) {  
 **delete**[] body;  
 **this**->body\_size = other.body\_size;  
 **this**->body\_len = other.body\_len;  
 **this**->body = **new** dyn\_array\_type[body\_size];  
 **for** (size\_t i = 0; i < body\_len; ++i)  
 **this**->body[i] = other.body[i];  
 }  
 **return** \***this**;  
 }  
};

**template** <**typename** stack\_type>  
**class** TStack : **public** TDynamic\_array<stack\_type> {  
**public**:  
 TStack() : TDynamic\_array<stack\_type>() {};  
 *// Creating a stack with allocated memory for n-elements* TStack(size\_t n) : TDynamic\_array<stack\_type>(n) {};  
 *// Removing the top element of the stack* stack\_type pop() {  
 **if** (**this**->body\_len == 0) **throw "Size error: an attempt to delete[] an item for an empty stack"**;  
 **this**->body\_len--;  
 **return this**->body[**this**->body\_len];  
 }  
 *// Adding an item to the top of the stack* **void** push(**const** stack\_type& new\_el) {  
 **if** (**this**->body\_size < **this**->body\_len+1) {  
 **this**->resize(**this**->body\_len \* 2);  
 }  
 **this**->body[**this**->body\_len++] = new\_el;  
 }  
 *// Viewing the top element of the stack* stack\_type top() **const** {  
 **if** (**this**->body\_len == 0) **throw "Size error: an attempt to get an item for an empty stack"**;  
 **return this**->body[**this**->body\_len-1]; *// Return of the top element of the stack* }  
};

*dfs.hpp*

#pragma once  
#include **"user\_stack.hpp"**#include **"map\_parsing.hpp"**#include **<ctime>***// The sides of bypassing the DFS algorithm***enum class** sides {  
 ***left***, ***down***, ***right***, ***up***, ***no\_side***};  
**using enum** sides;  
*// Type of porecudure (clockwise, counterclockwise, random)***enum class** type\_procedure {  
 ***clockwise***, ***counterclockwise***, ***random***};  
**using enum** type\_procedure;  
**constexpr** size\_t crawl\_procedure\_len = 3; *// max element of crawl\_procedure array index  
  
/// <summary>  
/// Description: The step of the DFS algorithm (depths-first search).  
/// The crawl procedure is randomized  
/// </summary>  
/// <param name="cur\_map">a map containing the available movements</param>  
/// <param name="solution">current solution</param>  
/// <param name="solution\_find">a flag certifying that a solution has been found</param>  
/// <param name="proc\_type">type of porecudure (clockwise, counterclockwise, random)</param>***static void** dfs\_step(**const** map\_type& cur\_map, TStack<node>& solution, **bool**& solution\_find, type\_procedure proc\_type) {  
 std::vector<sides> crawl\_procedure = **{ *up*** , ***down*** , ***left*** , ***right* }**;  
  
#pragma region Rules of procedure  
 **if** (proc\_type == ***random***) { *// Random order* srand((**unsigned int**)time(NULL));  
 **for** (size\_t index = 0; index < crawl\_procedure\_len; ++index) {  
 size\_t random\_el\_index = rand() % (crawl\_procedure\_len + 1);  
 sides temp\_side = crawl\_procedure[random\_el\_index];  
 crawl\_procedure[random\_el\_index] = crawl\_procedure[index];  
 crawl\_procedure[index] = temp\_side;  
 }  
 }  
 **else if** (proc\_type == ***counterclockwise***) { *// Counterclockwise order, started by left direction* crawl\_procedure = **{ *left*** , ***up***, ***right*** , ***down* }**;  
 }  
 **else if** (proc\_type == ***clockwise***) { *// Clockwise order, started by right direction* crawl\_procedure = **{ *right*** , ***down***, ***left*** , ***up* }**;  
 }  
#pragma endregion  
  
 **for** (**const auto**& side : crawl\_procedure) {  
 **if** (solution\_find == **true**) **continue**;  
  
 *// Calculating the next position on the map* node next\_step;  
 **switch** (side) {  
 **case *left***:  
 next\_step.set\_cords(solution.top().x - 1, solution.top().y);  
 **break**;  
 **case *down***:  
 next\_step.set\_cords(solution.top().x, solution.top().y - 1);  
 **break**;  
 **case *right***:  
 next\_step.set\_cords(solution.top().x + 1, solution.top().y);  
 **break**;  
 **case *up***:  
 next\_step.set\_cords(solution.top().x, solution.top().y + 1);  
 **break**;  
 **default**:  
 **break**;  
 }  
  
 *// Checking the validity of the new coordinate* **if** (next\_step.y < cur\_map.size() && next\_step.x < cur\_map[next\_step.y].size()  
 **and not** (solution.contain(next\_step))) {  
 el\_type next\_el = cur\_map[next\_step.y][next\_step.x];  
 **if** (next\_el == ***AVAILABLE***) { *// If available, continue the crawl* solution.push(next\_step);  
 dfs\_step(cur\_map, solution, solution\_find, proc\_type);  
 }  
 **else if** (next\_el == ***END***) { *// At the end, we expand the recursion* solution.push(next\_step);  
 solution\_find = **true**;  
 }  
 }  
  
 **if** (solution\_find) **return**; *// Recursion reversal condition* }  
 *// Removing an item from the solution list if further depth-first traversal is not possible* solution.pop();  
 **return**;  
}  
  
*/// <summary>  
/// Description: Depth-first search algorithm for a map from a file (DFS)  
/// </summary>  
/// <param name="map\_path">the path to the file containing the map</param>  
/// <param name="proc\_type">type of porecudure (clockwise, counterclockwise, random)</param>  
/// <returns>solving the problem of finding a path</returns>*TStack<node> dfs(std::filesystem::path map\_path, type\_procedure proc\_type = ***clockwise***) {  
 *// Getting a map from a file* std::ifstream map\_stream(map\_path);  
 **if** (**not** map\_stream.is\_open()) **throw "File is unavailible"**;  
 std::pair<map\_type, node> map\_parsing\_res = map\_parsing(map\_stream);  
 map\_type cur\_map = map\_parsing\_res.first;  
 node start\_node = map\_parsing\_res.second;  
  
 *// Finding a solution* TStack<node> solution(cur\_map.size());  
 solution.push(start\_node);  
 **bool** solution\_find = **false**;  
 dfs\_step(cur\_map, solution, solution\_find, proc\_type);  
  
 *// If there is no solution, we throw an exception.* **if** (**not** solution\_find) **throw "Solution not found"**;   
 **else return** solution;  
}  
  
*/// <summary>  
/// Description: Depth-first search Algorithm for a given Map (DFS)  
/// </summary>  
/// <param name="map\_area">map containing the available movements</param>  
/// <param name="proc\_type">type of porecudure (clockwise, counterclockwise, random)</param>  
/// <returns>solving the problem of finding a path</returns>*TStack<node> dfs(std::pair<map\_type, node> map\_area, type\_procedure proc\_type = ***clockwise***) {  
 map\_type cur\_map = map\_area.first;  
 node start\_node = map\_area.second;  
  
 TStack<node> solution(cur\_map.size());  
 solution.push(start\_node);  
 **bool** solution\_find = **false**;  
 dfs\_step(cur\_map, solution, solution\_find, proc\_type);  
  
 **if** (**not** solution\_find) **throw "Solution not found"**;  
 **else return** solution;  
}

*interface.hpp*

#include **<iostream>**#include **<ctime>**#include **<conio.h>**#include **<Windows.h>**#include **<random>**#include **"dfs.hpp"  
  
constexpr short unsigned int** bigM\_ASCII = 0x4d;  
**constexpr short unsigned int** bigP\_ASCII = 0x50;  
**constexpr short unsigned int** bigR\_ASCII = 0x52;  
**constexpr short unsigned int** skip\_file = 2;  
#pragma region Signatures  
**enum** menu\_point {  
 ***zero*** = 0x30, ***one***, ***two***, ***three***, ***four***, ***five***, ***six***, ***manual*** = 0x6d, ***print*** = 0x70, ***randmap*** = 0x72  
};  
**using enum** menu\_point;  
  
**enum** colors  
{  
 ***green*** = 10, ***purple*** = 13, ***blue*** = 11, ***red*** = 12, ***yellow*** = 14, ***white*** = 15  
};  
  
**void** menu();  
**static void** print\_map(map\_type& current\_map, TStack<node> sln, std::ostream& out = std::cout);  
**static void** exit\_message(**bool**& exit\_flag);  
**static void** incorrect\_data\_message(**const char** err[]);  
**static void** incorrect\_file\_message();  
**static void** print\_manual();  
**static bool** check\_file\_name(**const** std::string& file\_name);  
**static bool** check\_file\_exist(std::string& file\_name);  
**static void** set\_color(std::string, **int**, std::ostream& out = std::cout);  
**static** std::filesystem::path find\_file\_menu(**const** std::filesystem::path& directory);  
**static int** file\_open\_menu(std::string filename);  
**static void** map\_load\_tf(map\_type& exemple);  
**static void** sln\_load\_tf(TStack<node>& sln);  
**static void** random\_map\_generate\_menu(std::pair<map\_type&, node&> &start\_info);  
#pragma endregion  
  
*//--------------------------- functional program units ---------------------------------***void** menu() {  
 *// Function responsible for user interaction with the program* **bool** end\_programm = **false**;  
 map\_type current\_map = rand\_map\_generate((**unsigned int**)time(NULL)).first;  
 node curr\_start = { 0, 0 };  
  
 *// Variables for work with files* std::filesystem::path file\_path;   
 std::ifstream file\_stream;  
  
 std::pair<map\_type&, node&> start\_info = { current\_map , curr\_start };  
 type\_procedure proc\_type = ***clockwise***;  
 TStack<node> sln;  
  
 **while** (**not**(end\_programm)) {  
 system(**"cls"**);  
 set\_color(**"You are working with programm, that finding path from S to O, "** \  
 **"excluding X symbols.\n"**, colors::***white***);  
 print\_map(start\_info.first, sln);  
 set\_color(**"current type of procedure (clockwise/counterclockwise/random):\n"**, colors::***blue***);  
 **switch** (proc\_type)  
 {  
 **case** type\_procedure::***clockwise***:  
 std::cout << **"clockwise\n"**;  
 **break**;  
 **case** type\_procedure::***counterclockwise***:  
 std::cout << **"counterclockwise\n"**;  
 **break**;  
 **case** type\_procedure::***random***:  
 std::cout << **"random\n"**;  
 **break**;  
 **default**:  
 **break**;  
 }  
 std::cout << **"\n\n"**;  
  
 std::cout << **"To find solution: "**; set\_color(**"enter -- 1\n"**, colors::***green***);  
 std::cout << **"To enter map of area from console: "**; set\_color(**"enter -- 2\n"**, colors::***green***);  
 std::cout << **"To enter map of area from file: "**; set\_color(**"enter -- 3\n"**, colors::***green***);  
 std::cout << **"To copy map of area to file: "**; set\_color(**"enter -- 4\n"**, colors::***green***);  
 std::cout << **"To clear map of area: "**; set\_color(**"enter -- 5\n"**, colors::***green***);  
 std::cout << **"To change type\_procedure: "**; set\_color(**"enter -- 6\n\n"**, colors::***green***);  
 std::cout << **"To create random map: "**; set\_color(**"enter -- r\n"**, colors::***green***);  
 std::cout << **"To print solution vector: "**; set\_color(**"enter -- p\n"**, colors::***green***);  
 std::cout << **"To open manual: "**; set\_color(**"enter -- m\n"**, colors::***green***);  
 std::cout << **"To exit "**; set\_color(**"enter -- 0\n\n"**, colors::***green***);  
  
  
 **int** user\_enter;  
 **bool** exit\_flag = **false**;  
 **while** (!exit\_flag) {  
 user\_enter = \_getch();  
 *// if find 'M' or 'P' translate to lower register* user\_enter = (user\_enter == bigM\_ASCII) ? ***manual*** : user\_enter;  
 user\_enter = (user\_enter == bigP\_ASCII) ? ***print*** : user\_enter;  
 user\_enter = (user\_enter == bigR\_ASCII) ? ***randmap*** : user\_enter;  
 exit\_flag = **true**;  
 **switch** ((menu\_point)user\_enter) {  
 **case *one***:  
 system(**"cls"**);  
 **try** {  
 sln = dfs(start\_info, proc\_type);  
 sln.invert();  
 print\_map(start\_info.first, sln);  
 set\_color(**"\nVector of solution:\n"**, colors::***blue***);  
 **while** (**not** sln.empty())  
 {  
 std::cout << **"("** << sln.top().x << **","** << sln.top().y << **")\n"**;  
 sln.pop();  
 }  
 }  
 **catch** (**const char** e[]) {  
 std::cout << e;  
 }  
 \_getch();  
 **break**;  
 **case *two***:  
 system(**"cls"**);  
 std::cout << **"Enter the map (X = wall, . = path, O = end, S = start).\n"**;  
 set\_color(**"Type /end to finish input.\n"**, ***green***);  
 **try** {  
 start\_info = map\_parsing(std::cin);  
 }  
 **catch** (**const char**\* e) {  
 incorrect\_data\_message(e);  
 }  
 **break**;  
 **case *three***:  
 system(**"cls"**);  
 file\_path = find\_file\_menu(std::filesystem::current\_path());  
 **if** (file\_path != **""**) {  
 file\_stream.open(file\_path);  
 **try** {  
 start\_info = map\_parsing(file\_stream);  
 }  
 **catch** (**const char**\* e) {  
 incorrect\_data\_message(e);  
 }  
 file\_stream.close();  
 }  
 **break**;  
 **case *four***:  
 map\_load\_tf(start\_info.first);  
 **break**;  
 **case *five***:  
 start\_info.first = **{** {} **}**;  
 start\_info.second = {};  
 **break**;  
 **case *six***:  
 **if** (proc\_type == ***clockwise***) proc\_type = ***counterclockwise***;  
 **else if** (proc\_type == ***counterclockwise***) proc\_type = ***random***;  
 **else if** (proc\_type == ***random***) proc\_type = ***clockwise***;  
 **break**;  
 **case *randmap***:  
 random\_map\_generate\_menu(start\_info);  
 **break**;  
 **case *manual***:  
 print\_manual();  
 **break**;  
 **case *print***:  
 **try** {  
 sln = dfs(start\_info, proc\_type);  
 sln.invert();  
 sln\_load\_tf(sln);  
 }  
 **catch** (**const char** e[]) {  
 std::cout << e;  
 }  
 **break**;  
 **case *zero***: *// exit from func* exit\_message(end\_programm);  
 **break**;  
 **default**:  
 exit\_flag = **false**;  
 **break**; *// skip unregulated user commands* }  
 }  
 }  
}  
  
**static void** print\_map(map\_type& current\_map, TStack<node> sln, std::ostream& out) {  
 set\_color(**"Current map of area:\n"**, colors::***blue***);  
 **if** (current\_map[0].size() == NULL) {  
 out << **"CLEAR \n"**;  
 }  
 **else** {  
 **bool** is\_rectangle = **true**;  
 **for** (size\_t y = 1; y < current\_map.size(); ++y) {  
 **if** (current\_map[y].size() != current\_map[y - 1].size())  
 is\_rectangle = **false**;  
 }  
 **for** (size\_t y = 0; y < current\_map.size(); ++y) {  
 **for** (size\_t x = 0; x < current\_map[y].size(); ++x) {  
 std::string printing\_line = **" "**;  
 **switch** (current\_map[y][x]) {  
 **case *AVAILABLE***:  
 printing\_line[0] = MapSymbolsMask::av;  
 **break**;  
 **case *START***:  
 printing\_line[0] = MapSymbolsMask::st;  
 **break**;  
 **case *END***:  
 printing\_line[0] = MapSymbolsMask::ed;  
 **break**;  
 **case *UNAVAILABLE***:  
 **if** (**not** is\_rectangle) {  
 printing\_line[0] = MapSymbolsMask::unav\_all;  
 **break**;  
 }  
 **if** (x > 0 **and** x < current\_map[y].size() - 1) {  
 **if** (y == 0) {  
 **if** (current\_map[y][x + 1] == ***UNAVAILABLE***)  
 printing\_line[0] = MapSymbolsMask::unav\_all;  
 **else** printing\_line[0] = MapSymbolsMask::unav\_ver;  
 }  
  
 **else if** (y == current\_map.size() - 1) {  
 **if** (current\_map[y][x + 1] == ***UNAVAILABLE***)  
 printing\_line[0] = MapSymbolsMask::unav\_all;  
 **else** printing\_line[0] = MapSymbolsMask::unav\_ver;  
 }  
 }  
 **else** printing\_line[0] = MapSymbolsMask::unav\_ver;  
  
 **if** (y > 0) {  
 **if** (x == 0 **and** y != current\_map.size() - 1) {  
 **if** (current\_map[y + 1][x] == ***UNAVAILABLE***)  
 printing\_line[0] = MapSymbolsMask::unav\_all;  
 **else** printing\_line[0] = MapSymbolsMask::unav\_hor;  
 }  
  
 **else if** (x == current\_map[y].size() - 1 **and** y != current\_map.size() - 1) {  
 **if** (current\_map[y - 1][x] == ***UNAVAILABLE***)  
 printing\_line[0] = MapSymbolsMask::unav\_all;  
 **else** printing\_line[0] = MapSymbolsMask::unav\_hor;  
 }  
 }  
 **else** printing\_line[0] = MapSymbolsMask::unav\_hor;  
  
 **if** ((x > 0 **and** x < current\_map[y].size() - 1) **and** (y > 0 **and** y < current\_map.size() - 1)) {  
 **if** ((current\_map[y + 1][x] == ***UNAVAILABLE* or** current\_map[y - 1][x] == ***UNAVAILABLE***) **and** (current\_map[y][x + 1] == ***UNAVAILABLE* or** current\_map[y][x - 1] == ***UNAVAILABLE***)) {  
 printing\_line[0] = MapSymbolsMask::unav\_all;  
 }  
 **else if** (current\_map[y + 1][x] == ***UNAVAILABLE* or** current\_map[y - 1][x] == ***UNAVAILABLE***) {  
 printing\_line[0] = MapSymbolsMask::unav\_ver;  
 }  
 **else if** (current\_map[y][x + 1] == ***UNAVAILABLE* or** current\_map[y][x - 1] == ***UNAVAILABLE***) {  
 printing\_line[0] = MapSymbolsMask::unav\_hor;  
 }  
 **else** {  
 printing\_line[0] = MapSymbolsMask::unav\_all;  
 }  
 }  
 }  
   
 **if** (sln.contain({ x, y })) {  
 set\_color(printing\_line, colors::***green***, out);  
 }  
 **else** {  
 set\_color(printing\_line, colors::***white***, out);  
 }  
 } out << **"\n"**;  
 }  
 }  
}  
  
**static void** set\_color(std::string text, **int** color, std::ostream& out) { *// выбор цвета (в значение color подставлять из перечисления serv\_colors)* SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), color);  
 out << text;  
 SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 15);  
}  
  
*//---------------------------- informational program units --------------------------------***static void** exit\_message(**bool**& exit\_flag) {  
 *// confirmation message before exiting the program* **bool** message\_reload = **true**;  
 **while** (message\_reload) {  
  
 system(**"cls"**);  
 std::cout << **"Are you shure, that you want exit?\n"**;  
 std::cout << **"Y/n\n"**;  
  
 **char** user\_answer = \_getch();  
 user\_answer = std::tolower(user\_answer);  
 **if** (user\_answer == **'y'**) {  
 exit\_flag = **true**;  
 message\_reload = **false**;  
 }  
 **else if**(user\_answer == **'n'**) {  
 exit\_flag = **false**;  
 message\_reload = **false**;  
 }  
 }  
}  
  
**static void** incorrect\_data\_message(**const char** err[]) {  
 *// Information message about incorrect input data* set\_color(**"you entered incorrect data\n"**, colors::***red***);  
 set\_color(err, colors::***red***);  
 \_getch();  
}  
  
**static void** incorrect\_file\_message() {  
 *// Information message about an incorrect file* std::cout << **"file contains incorrect data, please change file data.\n"**;  
 std::cout << **"to continue, enter any key "**;  
 \_getch();  
}  
  
**static void** print\_manual() {  
 system(**"cls"**);  
 set\_color(**"MANUAL FOR THE PATHFINDING PROGRAM\n"**, ***blue***); *// Section header* std::cout << **"1. Introduction\n"**;  
 std::cout << **" This program implements a pathfinding algorithm, where the goal is to find a way\n"  
 " from a start point 'S' to an end point 'O', while avoiding obstacles represented by 'X'.\n\n"**;  
  
 std::cout << **"2. Controls:\n"**;  
 std::cout << **" - You can navigate through the map and choose options by entering corresponding numbers.\n"**;  
 std::cout << **" - The pathfinding algorithm can be modified by choosing the procedure type (Clockwise, Counterclockwise, or Random).\n"**;  
 std::cout << **" - Maps can be manually entered through the console, loaded from a file, or saved to a file.\n\n"**;  
  
 std::cout << **"3. Menu Options:\n"**;  
 std::cout << **" 1. Find solution: Starts the pathfinding search using the current map.\n"**;  
 std::cout << **" 2. Input map from console: Allows you to manually input a new map.\n"**;  
 std::cout << **" 3. Load map from file: Opens an existing map from a file and loads it.\n"**;  
 std::cout << **" 4. Save map to file: Saves the current map to a file.\n"**;  
 std::cout << **" 5. Clear map: Clears the current map.\n"**;  
 std::cout << **" 6. Change procedure type: Allows you to change the pathfinding algorithm (Clockwise, Counterclockwise, or Random).\n"**;  
 std::cout << **" p. Print solution to file: Print solution for current map to a file.\n"**;  
 std::cout << **" m. Show manual: Displays this manual.\n"**;  
 std::cout << **" 0. Exit: Exits the program.\n\n"**;  
  
 std::cout << **"4. How to read the map:\n"**;  
 std::cout << **" - 'S' is the start point.\n"**;  
 std::cout << **" - 'O' is the endpoint.\n"**;  
 std::cout << **" - 'X' represents obstacles (impassable areas).\n"**;  
 std::cout << **" - '.' represents passable areas.\n\n"**;  
  
 set\_color(**"Press any key to return to the menu."**, ***green***);  
 \_getch(); *// Wait for user input before returning to the menu*}  
  
  
*//------------------------------- logical program units ----------------------------------***static bool** check\_file\_name(**const** std::string& file\_name) {  
 *// Check for invalid characters in the file name* std::vector<**char**> bad\_symbols = **{  
 '<'**,  
 **'>'**,  
 **':'**,  
 **'\"'**,  
 **'/'** ,  
 **'\\'**,  
 **'|'** ,  
 **'?'** ,  
 **'\*'** ,  
 **}**;  
 **if** (file\_name == **""**) **return false**;  
 **for** (**auto** sym : bad\_symbols) **if** (file\_name.find(sym) != -1) **return false**;  
 **return true**;  
}  
  
**static bool** check\_file\_exist(std::string& file\_name) {  
 *// This function converts the name to the full file path and returns the existence status* **if** (file\_name.find(**".txt"**) != file\_name.length() - 4 **or** file\_name.length() < 4) file\_name += **".txt"**; *// Add extension if not provided by the user* std::string f\_path = std::filesystem::current\_path().string();  
 f\_path += (**"\\"** + file\_name);  
 file\_name = f\_path;  
  
 **return** std::filesystem::exists(f\_path);  
}  
  
*//--------------------------------------- working with files -----------------------------------------***static bool** is\_number(**const** std::string& str) {  
 **return** !str.empty() && (str.find\_first\_not\_of(**"0123456789"**) == str.npos);  
}  
  
**static** std::string trim(**const** std::string& str) {  
 **auto** space = **' '**;  
 *// If the string does not contain spaces to remove* **if** ((str.find\_first\_not\_of(space) == str.npos)) {  
 **return** std::string();  
 }  
 **else** {  
 *// Trim spaces at the beginning and end* size\_t count = str.find\_last\_not\_of(space) - str.find\_first\_not\_of(space) + 1;  
 **return** std::string(str.substr(str.find\_first\_not\_of(space), count));  
 }  
}  
  
**static** std::filesystem::path correct\_ffn(**const** std::string& str) {  
 std::string nstr = trim(str);  
 **if** (**not** nstr.empty()) {  
 **if** (nstr.back() == **'\"' or** nstr.back() == **'\''**) nstr.pop\_back();  
 }  
 **if** (**not** nstr.empty()) {  
 **if** (nstr[0] == **'\"' or** nstr[0] == **'\''**) nstr = nstr.substr(1);  
 }  
 std::filesystem::path ffn = std::filesystem::path(nstr);  
 **if** (ffn.is\_absolute()) {  
 *// If nstr is an absolute file path* **return** std::filesystem::path(nstr).lexically\_normal();  
 }  
 **else if** (nstr.find\_first\_of(**"\\/:\*?\"<>|+ !"**) != nstr.npos  
 **or** nstr.empty()  
 **or** nstr.back() == **L'.'  
 or** nstr.length() > \_MAX\_PATH) {  
 *// If nstr is just a file name  
 // and the file name is incorrect* **return** std::filesystem::path();  
 }  
 **else** {  
 *// If nstr is a correct file name* **return** std::filesystem::path(nstr).lexically\_normal();  
 }  
}  
  
**static void** find\_files(std::string key, **const** std::filesystem::path& sPath, std::vector<std::filesystem::path>& files) **try** {  
 files.clear();  
 std::filesystem::path key\_path = correct\_ffn(key);  
 *// Search for the file by name* **if** (exists(key\_path) **and** key\_path.extension() == **".txt"**) {  
 files.push\_back(key\_path);  
 }  
 *// Search for files in the specified directory* **else if** (exists(key\_path) **and** is\_directory(key\_path)) {  
 **for** (**const auto**& var : std::filesystem::directory\_iterator(key\_path, std::filesystem::directory\_options::skip\_permission\_denied)) {  
 **if** (var.path().extension() == **".txt"**) {  
 *// Browse all available files with the file extension* files.push\_back(var.path());  
 }  
 }  
 }  
 *// Search for files in the default directory by substring* **else if** (exists(sPath) **and** is\_directory(sPath)) {  
 **for** (**const auto**& var : std::filesystem::directory\_iterator(sPath, std::filesystem::directory\_options::skip\_permission\_denied)) {  
 **if** (var.path().filename().string().find(key) != key.npos  
 **and** var.path().extension() == **".txt"**) {  
 *// Browse all available files with the file extension and keyword* files.push\_back(var.path());  
 }  
 }  
 }  
}  
**catch** (...) { *// Exception handling* files.clear();  
 std::cout << **"Something went wrong, please try again or change the directory\n\n"**;  
}  
  
**static** std::filesystem::path find\_file\_menu(**const** std::filesystem::path& directory) {  
 std::vector<std::filesystem::path> files;  
 std::string input;  
 size\_t input\_int = 0;  
  
 **do** {  
 std::cout << **"Enter the full or partial file name to search map of area.\n"  
 "To display all available files from the program folder, enter an empty line[Enter].\n"  
 "To exit input /exit\n\n"**;  
  
 std::getline(std::cin, input);  
 system(**"cls"**);  
  
 find\_files(input, directory, files);  
 input = trim(input);  
 **if** (input == **"/exit"**) **return** {};  
  
 **if** (files.size() == 0ull) {  
 std::cout << **"Files not found, please try again.\n\n"**;  
 }  
 **else** {  
 **do** {  
 std::cout << **"Select the file to search map of area.\n"**;  
 **for** (size\_t i = 0; i != files.size(); ++i) {  
 std::cout << i + 1 << **". "** << files[i].filename() << **'\n'**;  
 }  
 std::cout << **"Enter the file number or the /again command to repeat the file/files search.\n"  
 "To exit input /exit\n\n"**;  
  
 std::getline(std::cin, input);  
 system(**"cls"**);  
  
 **if** (is\_number(input)) {  
 input\_int = std::stoull(input);  
  
 *// The user entered the correct number from the list* **if** (0 < input\_int **and** input\_int <= files.size()) {  
 *// If the selected file is empty* **if** (std::filesystem::file\_size(files[input\_int - 1]) == 0) {  
 std::cout << **"The selected file is empty, try again.\n\n"**;  
 }  
 **else** {  
 **return** files[input\_int - 1];  
 }  
 }  
 **else** {  
 std::cout << **"Please select a file from the list.\n\n"**;  
 }  
 }  
 **else if** (input == **"/again"**) {  
 *// The user wants to search again* **break**;  
 }  
 **else if** (input == **"/exit"**) **return** {};  
 **else** {  
 std::cout << **"Incorrect input, please try again.\n\n"**;  
 }  
 } **while** (**true**);  
 }  
 } **while** (**true**);  
}  
  
**static int** file\_open\_menu(std::string filename) {  
 *// Menu for how to open a file to unload information* **int** op\_code = 0;  
 **if** (std::filesystem::exists(filename)) { *// If the file exists* set\_color(**"Attention! File already exists\n"**, colors::***red***);  
 std::cout << **"Remember, if you want to fill the file, the information in the console will be\n"**;  
 std::cout << **"overwritten by the information that will be saved in the file\n\n"**;  
  
 std::cout << **"If you want to recreate the file enter -- 1\n"**;  
 std::cout << **"Otherwise, enter -- 0\n"**;  
  
 **while** (op\_code == 0) { *// Until the user selects one of the options* **int** var = \_getch();  
 **switch** (var) {  
 **case *one***: *// Merge data from the console and file with priority for the console* op\_code = 1;  
 **break**;  
 **case *zero***: *// Recreate the file* op\_code = skip\_file;  
 **break**;  
 }  
 }  
 }  
 **return** op\_code;  
};  
  
**static void** map\_load\_tf(map\_type& cur\_map) {  
 *// Function responsible for unloading data into a file* **bool** need\_exit = **false**;  
 **while** (**not**(need\_exit)) {  
 std::string file\_name;  
 system(**"cls"**);  
 std::cout << **"Specify the name of the file where you are going to save the map\n"**;  
 std::cout << **"It will be located in the directories: "**;  
 set\_color(std::filesystem::current\_path().string() + **"\n"**, colors::blue);  
 set\_color(**"To go back to the menu -- enter 0\n\n"**, colors::***green***);  
 std::cout << **"Please enter the file name with file extension: "**;  
 std::getline(std::cin, file\_name); *// Get the file name* **if** (file\_name == **"0"**) **return**; *// Special exit code as indicated in the interaction window* trim(file\_name);  
  
 **if** (check\_file\_name(file\_name)) { *// If the file name is valid* check\_file\_exist(file\_name); *// Get the full name for the file* **int** op\_code = file\_open\_menu(file\_name); *// How do we want to open the file if it already exists (1 -- continue filling, 2 -- recreate)* **bool** need\_load = **false**;  
  
 std::ofstream file(file\_name, std::ios::out);  
  
 **if** (op\_code != skip\_file) {  
 print\_map(cur\_map, {}, file);  
 }  
 file.close();  
 need\_exit = **true**;  
 }  
 **else** { *// If the file name is invalid* system(**"cls"**);  
 std::cout << **"The entered file name is incorrect, make sure it does not contain the following characters:\n"  
 " < > : \" \\ / | ? \*\n"  
 "To continue, enter any key"**;  
 \_getch();  
 }  
 }  
}  
  
**static void** sln\_load\_tf(TStack<node>& sln) {  
 *// Function responsible for unloading data into a file* **bool** need\_exit = **false**;  
 **while** (**not**(need\_exit)) {  
 std::string file\_name;  
 system(**"cls"**);  
 std::cout << **"Specify the name of the file where you are going to save vector of solution\n"**;  
 std::cout << **"It will be located in the directories: "**;  
 set\_color(std::filesystem::current\_path().string() + **"\n"**, colors::blue);  
 set\_color(**"To go back to the menu -- enter 0\n\n"**, colors::***green***);  
 std::cout << **"Please enter the file name with file extension: "**;  
 std::getline(std::cin, file\_name); *// Get the file name* **if** (file\_name == **"0"**) **return**; *// Special exit code as indicated in the interaction window* trim(file\_name);  
  
 **if** (check\_file\_name(file\_name)) { *// If the file name is valid* check\_file\_exist(file\_name); *// Get the full name for the file* **int** op\_code = file\_open\_menu(file\_name); *// How do we want to open the file if it already exists (1 -- continue filling, 2 -- recreate)* **bool** need\_load = **false**;  
  
 std::ofstream file(file\_name, std::ios::out);  
  
 **if** (op\_code != skip\_file) {  
 set\_color(**"Vector of solution:\n"**, colors::***blue***, file);  
 **while** (**not** sln.empty())  
 {  
 file << **"("** << sln.top().x << **","** << sln.top().y << **")\n"**;  
 sln.pop();  
 }  
 }  
 file.close();  
 need\_exit = **true**;  
 }  
 **else** { *// If the file name is invalid* system(**"cls"**);  
 std::cout << **"The entered file name is incorrect, make sure it does not contain the following characters:\n"  
 " < > : \" \\ / | ? \*\n"  
 "To continue, enter any key"**;  
 \_getch();  
 }  
 }  
}  
  
**static void** random\_map\_generate\_menu(std::pair<map\_type&, node&>& start\_info) {  
 **int** N1 = 0, N2 = 0;  
 **unsigned int** seed = 0;  
 **bool** valid\_input = **false**;  
  
 **while** (!valid\_input) {  
 system(**"cls"**);  
 std::cout << **"Enter map size (N1 N2) or press Enter for default (10 10): "**;  
 std::string size\_input;  
 std::getline(std::cin, size\_input);  
  
 **if** (size\_input.empty()) {  
 N1 = 10;  
 N2 = 10;  
 std::cout << **"No input detected. Using default size: 10 x 10\n"**;  
 }  
 **else** {  
 std::stringstream ss(size\_input);  
 **if** (!(ss >> N1 >> N2) || N1 <= 0 || N2 <= 0) {  
 std::cout << **"Error: Please enter two positive integers separated by space.\n"**;  
 **continue**;  
 }  
 }  
  
 std::cout << **"Enter seed value or press Enter for default (time-seed): "**;  
 std::string seed\_input;  
 std::getline(std::cin, seed\_input);  
 std::stringstream seed\_ss(seed\_input);  
  
 **if** (seed\_input.empty()) {  
 std::random\_device rd;  
 seed = rd(); *// hardware-based random seed* std::cout << **"No input detected. Using random seed: "** << seed << **"\n"**;  
 valid\_input = **true**;  
 }  
 **else if** (seed\_ss >> seed) {  
 valid\_input = **true**;  
 }  
 **else** {  
 std::cout << **"Error: Seed must be a positive integer.\n"**;  
 *// Erase if error* std::cin.clear();  
 std::cin.ignore();  
 }  
 }  
 \_getch();  
 start\_info = rand\_map\_generate(seed, {N1, N2});  
}

***find\_path.cpp***

#include **<iostream>**#include **"dfs.hpp"**#include **"interface.hpp"  
  
int** main()  
{  
 menu();  
}

Тестирование

1. Нахождение пути для случайной карты с параметрами 10X10 и seed 805113345

Изображение выглядит как снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

1. Ввод невозможной карты

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

1. Ввод корректной карты

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

1. Ввод карты из файла

Изображение выглядит как снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

1. Нахождение пути в свободном поле 5X5 “по часовой стрелке”

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

1. Нахождение пути в свободном поле 5X5 “против часовой стрелке”

Изображение выглядит как снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

1. Нахождение пути в свободном поле 5X5 “случайное направление”

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

1. Отсутствие возможного решения

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

1. Сохранение карты в файл

Изображение выглядит как текст, снимок экрана, диаграмма, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

1. Сохранение результата в файл

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.