Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

|  |  |
| --- | --- |
| Факультет | Комп’ютерної інженерії та управління |
|  | (повна назва) |

|  |  |
| --- | --- |
| Кафедра | Електронних обчислювальних машин |
|  | (повна назва) |

**КУРСОВА РОБОТА**

|  |
| --- |
|  |
| (рівень вищої освіти) |

|  |  |
| --- | --- |
| з дисципліни: | Системне програмування |
|  |  |
| на тему: | Розробка програмного забезпечення з використанням |
|  | функцій WinAPI: «Генерація та знаходження виходу із |
|  | лабіринту» |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Виконав: студент | | | 2 | курсу, групи | | | КІУКІ 18-4 |
| спеціальності | | 123 «Комп’ютерна інженерія» | | | | | |
|  | | (код і повна назва спеціальності) | | | | | |
| Освітньо-професійної програми: «Комп’ютерна інженерія» | | | | | | | |
| (повна назва освітньої програми) | | | | | | | |
|  |  | | | |  | Кравченко Н. С. | |
|  | (підпис) | | | |  | (прізвище, ініціали) | |
|  |  | | | |  |  | |
|  |  | | | |  |  | |
|  |  | | | |  |  | |
| Голова комісії: |  | | | |  | проф. Волк М.О. | |
| Члени комісії: | (підпис) | | | |  | (посада, прізвище, ініціали)  доц. Філімончук Т.В. | |
|  | (підпис) | | | |  | (посада, прізвище, ініціали)  ст. викл. Мартовицький В.О. | |
|  | (підпис) | | | |  | (посада, прізвище, ініціали) | |

2020 р.

Харківський національний університет радіоелектроніки

(найменування вищого навчального закладу)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кафедра: | ЕОМ | | | | |
| Дисципліна: | Системне програмування | | | | |
| Спеціальність: | Комп’ютерна інженерія | | | | |
| Курс: | 2 | Група: | КІУКІ-18-4 | Семестр: | 4 |

**З А В Д А Н Н Я**

на курсовий проект (роботу) студента

Кравченко Назар Сергійович

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. Тема роботи | Розробка програмного забезпечення з використанням функції | | | |
| WinAPI: «Генерація та знаходження виходу із лабіринту» | | | | |
|  | | | | |
| 2. Строк здачі студентом закінченої роботи | | | | 07.06.20 |
| 3. Вихідні дані до роботи | | Демонстрація алгоритмів генерцації та знаходження | | |
| виходу із лабіринту. Наявність зручного користувацького інтерфейсу та можливість | | | | |
| зберігання результатів роботи програми у файл для подальшого використання. | | | | |
|  | | | | |
| 4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): | | | | |
| 1. Постановка задачі | | | | |
| 1. Теоретична частина | | | | |
| 1. Програмна реалізація | | | | |
| 1. Інструкція користувача | | | | |
|  | | | | |
|  | | | | |
| 5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): | | | | |
| Лістинг програми | | |  | |
|  | | |  | |
|  | | |  | |
| 6. Дата видачі завдання: 20.02.2020 | | | | |
|  | | | | |

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Найменування етапів**  **курсової роботи** | **Строк виконання**  **етапів роботи** | **Примітка** |
| **1.** | Ознайомлення з літературними | 22.02 – 05.03 |  |
|  | матеріалами, аналіз та вибір методу |  |  |
|  | виконання поставленого завдання |  |  |
| **2.** | Розробка алгоритмів рішення, вибір | 06.03 – 14.03 |  |
|  | системних засобів, вирішення задачі |  |  |
|  | курсового проектування |  |  |
|  |  |  |  |
| **3.** | Проектування додатку | 15.03 – 22.04 |  |
|  |  |  |  |
| **4.** | Налагодження та тестування програми | 23.04 – 11.05 |  |
|  |  |  |  |
| **5.** | Оформлення пояснювальної записки | 12.05 – 05.06 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | | |  |  | |
|  | | | (підпис) |  | |
|  | | |  |  |  |
| Керівник | | | (підпис) |  | (прізвище, ім’я, по батькові) |
| « 20 » | 02 | 2020 | |  | |

# РЕФЕРАТ

Записка пояснювальна до курсової роботи: 24 с., 14 рис. 1 таблиця, 4 розділи, 3 додатки.

Мета роботи — розробка додатка для демонстрації алгоритмів генерцації та знаходження виходу із лабіринту.

Метод вирішення задачі — для генераці лабіринту використаний алгоритм «recurcive backtracking», для знаходження виходу — алгоритм «recurcive solving».

Розроблено додаток, за допомогою якого, можна генерувати і вирішувати лабіринти.

Програму складено мовою C++ у середовищі програмування Visual C++.

ЛАБІРИНТ, ГЕНЕРАЦІЯ, РЕКУРСІЯ, ФУНКЦІЇ, ДИНАМІЧНИЙ МАСИВ, ГЕНЕРАЦІЯ ЛАБІРИНТУ, ЗНАХОДЖЕННЯ ВИХОДУ, АЛГОРИТМ, РОБОТА З ФАЙЛАМИ.

ЗМІСТ

[Вступ……………………………………………………………………...………. 6](#_Toc11706395)

[1 Постановка задачі.………………………………………………………………7](#_Toc11706396)

[2 Теоретична частина………………………………………….………………….8](#_Toc11706397)

[2.1 Основні теоретичні відомості..……….……………...…….……….8](#_Toc11706398)

[2.2 Опис алгоритму генерації……………………….………….………8](#_Toc11706399)

[3 Програмна реалізація……………………...………………….….......................10](#_Toc11706400)

[3.1 Опис структури програми…………………………………………..10](#_Toc11706401)

[3.2 Використані бібліотеки С++………………………………………..10](#_Toc11706402)

[3.3 Розроблені функції…………………………………………………..11](#_Toc11706403)

[3.4 Опис розроблених функцій…………………………………………11](#_Toc11706404)

[4 Інструкція користувача…………………………………………………...........13](#_Toc11706405)

[4.1 Запуск програми…………………………………………..…......….13](#_Toc11706406)

[4.2 Початок виконання програми..…………………………….............13](#_Toc11706407)

[4.3 Завершення виконання программи………………………...............16](#_Toc11706408)

[Висновки……………………………………………………………..…………....17](#_Toc11706409)

[Перелік посилань………………………………………………..…......................18](#_Toc11706410)

[Додаток А. Текст програми.………………………………………….……...…...19](#_Toc11706411)

[Додаток Б. Можливі результати роботи програми……...………..…….............19](#_Toc11706412)

[Додаток В. Блок-схема алгоритму генерації лабіринту...……………...….…...24](#_Toc11706413)

# ВСТУП

C++ — універсальна мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом у 1979 році та названа «Сі з класами». Пізніше Страуструп перейменував мову у C++. Мова C++була розвинута з мови програмування C і за дуже малими виключеннями зберігає мову програмування C як підмножину.

Метою даної роботи є розробка додатка, для демонстрації алгоритмів генерцації та знаходження виходу із лабіринту.

Лабіринти мають досить давну історію. У древніх греків і римлян це поняття означало простір, що складається з численних залів, камер і переходів, розташованих згідно із складним планом. У стародавньому Єгипті лабіринти застосовували для того, щоб захистити фараонів від зовнішнього світу, для забезпечення безпеки. Таким чином вони могли керувати країною з центру лабірінту без будь – якого страху.

Однак, у наш час для побудови лабіринтів люди використовують не каміння у поєднанні з гігантькими за площею території, а спеціальні комп’ютерні алгоритми генерації.

Таких алгоритмів налічується просто безліч, і кожен з них має свої переваги та недоліки. Проте, в даній роботі буде детально розглянуто алгоритм під назвою «recursive backtracking» та наведено приклад його роботи.

# 1 ПОСТАНОВКА ЗАДАЧІ

Завдання курсовї роботи:

* Розробка алгоритму генерації лабіринту ;
* Розробка алгоритму знаходження виходу із лабіринту;
* Демонстрація виходу для користовуча;
* Забезпечення консольного користувацього інтерфейсу.
* Обробка виключень
* Запис лабіринту у файл

Програма реалізована на мові програмування С++ в середовищі Microsoft Visual Studio.

# 2 ТЕОРЕТИЧНА ЧАСТИНА

## 2.1 Основні теоретичні відомості

Лабіринт - структура (зазвичай в двомірному просторі), що складається із заплутаних шляхів до виходу.

В цілому, існує близько семи відмінних критеріїв, що розділяють всі лабіринти на окремі групи, одинадцяти способів генерції та тринадцяти методів знаходження шляху із лабіринту.

В цій роботі лабіринт задається у вигляді матриці, тобто двумірного символьного массиву, що складається із елементів WALL('@'), FREE(' ') або VISITEDPOINT('.'). Кожен елемент масиву визначає поточний стан клітини:

WALL('@') — символ для позначення "стін", тобто елементів, по яким "прохід" заборонений;

FREE (' ') — сивол, що означає прохід, тобто порожні місця, по яким користувач має змогу "пройти";

VISITEDPOINT('.') — цей символ означає, що користувач вже "був" в цьому елементі , тому "проходити" там знову заборонено. Також для коректної роботи програми потрібно задати координати точок початку та призначення. Користувач повинен сам вибрати розмір лабіринту.

Саме з цим масивом працює програма.

## 2.2 Опис алгоритму генерації

Позначимо FREE(‘ ’) як прохід і WALL(‘@’) як стіну. Початкові умови —двомірний символьний масив, заповнений елементами WALL(‘@’). Розмірність масиву - будь-які непарні числа більше 5.

Вважаємо що лівий верхній кут стіни має координати (1; 1). Координати «землерийки» завжди парні, пересувається вона, відповідно, тільки стрибками довжиною в два елементи масиву.

Алгоритм:

1. Обираємо точку приземлення, якій присвоюється значення FREE(‘ ’);
2. Випадково вибираємо один з напрямків - верх, низ, ліво, право;
3. Якщо після стрибка в обраному напрямку ми опинимося у зовнішній стіні або в проході, то повернутися до попереднього пункту. Інакше - стрибаємо в зазначеному напрямку. Надаємо значення FREE(‘ ’) клітці, в яку приземлилися і через яку перестрибнули;
4. Якщо після приземлення ми не можемо зробити стрибок (попадання в глухий кут), то повертаємося назад по рекурсії. Інакше - переходимо до другого пункту;
5. Повторюємо пункт вище до тих пір, поки не будуть оброблені всі можливі ходи «землерийки». Інакше — кінець алгоритму.

Блок-схема алгоритму приведена в додатку В.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ

## 3.1 Опис структури програми

Програму було реалізовано за допомогою мови програмування С++. Більшу частину коду розподілено на структурні елменти – функції. Це допомагає зробити програму зручною для читання початкового коду робить можливим використання цих частин програми в інших проектах, тобто реалізація стає більш універсальною.

Лабіринт був побудований на основі двумірного динамічного массиву, що значно підвищує читальність коду і робить обробку лабіринту доволі зручною для програміста.

## 3.2 Використані бібліотеки С++

* Windоws.h - це специфічний для Windows файл заголовок для мови програмування C++, який містить декларації для всіх функцій Windows API, всі загальні макроси, що використовуються програмістами Windows, і всі типи даних, що використовуються різними функціями і підсистемами.
* iostream - [бібліотека](https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%B1%D0%BB%D1%96%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) і відповідний [заголовний файл](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB), які використовується для організації введення-виведення в мові програмування [C++](https://uk.wikipedia.org/wiki/C%2B%2B).
* fstream - заголовки зі стандартної бібліотеки C ++, що включає набір класів, методів і функцій, які надають інтерфейс для читання/запису даних з/в файл.
* ctime - заголовки стандартної бібліотеки мови програмування C++, що містить типи і функції для роботи з датою і часом.
* string - клас з методами і змінними для організації роботи з рядками в мові програмування C ++.

## 3.3 Розроблені функції

Всі розроблені функції представлені у таблиці 3.1.

Таблиця 3.1 — Основні цілі всіх розроблених функції програми

|  |  |
| --- | --- |
| Функція | Призначення |
| void fillMaze | Заповнення масива «стінами» |
| void createMaze | Створення лабіринту |
| bool findWay | Знаходження виходу |
| void output | Меню вибору способу показу лабіринта |
| void printMaze | Вивід лабіринта в консоль |
| void printMaze | Вивід в файл |
| void enterSizes | Ввод ширини та висоти і перевірка їх на валідність |
| bool checkSizes | Перевірки ширини в висоти |
| bool inLimits | Перевірка індексів на коректність |
| void badInput | Виправлення некоректного вводу |
| void setColour | Встановлення кольору консолі |
| void oneMore | Запит користувача почати спочатку |

## 3.4 Опис розроблених функцій

* void fillMaze(char\*\* maze) — функція приймає один аргумент — двумірний динамічний масив, що лежить в основі лабіринту. Функція заповнює весь двумірний масив «стінами» WALL('@'), тобто непрохідними елементами.
* void createMaze(char\*\* maze, int x, int y) — функція приймає 3 аргумента: двумірний динамічний масив, що лежить в основі лабіринту та координати (*x, y*).Створення лабіринту відбувається в наслідок рекурсивного проходження елементів масиву. Функція приймає початкові координати (*x, y*), записує в ці координати символ FREE (' ') і випадковим чином обирає наступний вектор руху. Якщо через два елемента массиву в цьому напрямку «стіна», то «руйнуємо» простір між нею і початковими координатами (*x, y*) та викликаємо рекурсивно функцію createMaze з координатами, що відрізняються на 2 від початкових в заданому напрямку.
* bool findWay(char\*\* maze, int x, int y) — функція приймає 3 аргументи: двумірний динамічний масив, що лежить в основі лабіринту та координати (*x, y*).Функція рекурсивно проходить через кожен елемент массива, спочатку намагаючись піти вниз, потім вправо, вліво і вверх. Рекурсія закінчується, коли функція досягає точки призначення, тобто знаходить вихід і повертає true, або ж вихід не знайдений і функція повертає значення false.
* void output(char\*\* maze) — функція приймає один аргумент — двумірний динамічний масив, що лежить в основі лабіринту. Вона розроблена з метою реалізації зручного меню вибору способу показу лабіринту.
* void printMaze(char\*\* maze) — функція приймає один аргумент — двумірний динамічний масив, що лежить в основі лабіринту. Розроблена для демонстрації лабіринта в консолі.
* void printMaze(char\*\* maze, string path) — функція приймає два аргументи — двумірний динамічний масив, що лежить в основі лабіринту, і назву файла, в який буде здійснюватися запис лабіринту.
* void enterSizes — функція не приймає аргументів (використовує глобальні змінні «WIDTH(ширину)» та «HEIGHT(висоту)». Розроблена для вводу ширини та висоти лабіринту.
* bool checkSizes(int width, int height) — функція приймає два аргументи: ширину і висоту лабіринту. Розроблена для перевірки ширини і висоти на правильність вводу.
* bool inLimits(int x, int y) — функція приймає два аргументи: індекс ширини і висоти. Розроблена для перевірки індексів на коректність вводу.
* void setColour(int text, int back) — функція приймає два аргументи: номера кольору тексту та фона. Служить для зміни кольорів консолі.
* void oneMore() — функція не приймає аргументів. Cлужить для запиту користувача почати виконання програми спочатку.

# 4 ІНСТРУКЦІЯ КОРИСТУВАЧА

## 4.1 Запуск програми

Після запуску програми буде запропоновано ввести ширину та висоту лабіринту. В програмі встановлені чіткі обмеження: значення ширини повинно бути в межах від 5 до 165, а висоти — від 5 до 41. Обидва значення мають бути непарними цілими числами. Лише за цих умов програма прийме ввід даних від користувача.

Оберемо ширину 51 і висоту 21. Діалогове вікно матиме наступний вигляд (рис.4.1):

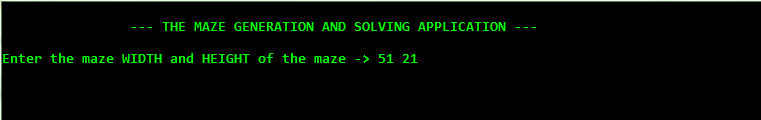


Рисунок 4.1 — Запуск програми

## 4.2 Початок виконання програми

Після вводу розміру лабіринту, програма сповіщує користувача про успішну генерацію лабіринту та пропонує меню для вибору способу виводу отриманої інформації або пропуску цього кроку (рис.4.2).

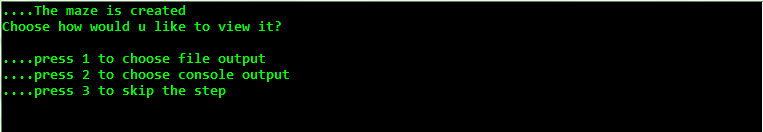


Рисунок 4.2 — Меню вибору способу демонстрації згенерованого лабіринту

Якщо обрати «1», програма запропонує ввести розташування та назву файла. Для прикладу створюю файл на робочому столі комп’ютера під назвою «generated\_maze.txt». Отже створений файл буде мати такий вигляд (рис.4.3):

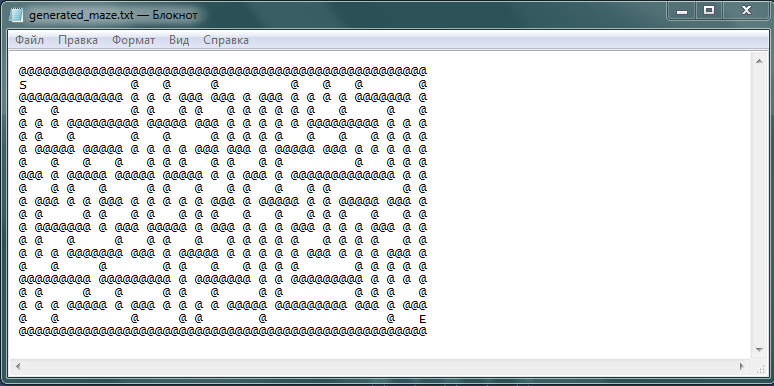


Рисунок 4.3 — Вивід згенерованого лабыринту в файл

Якщо обрати «2», програма виконає консольний вивід лабіринту, що матиме вигляд (рис.4.4):

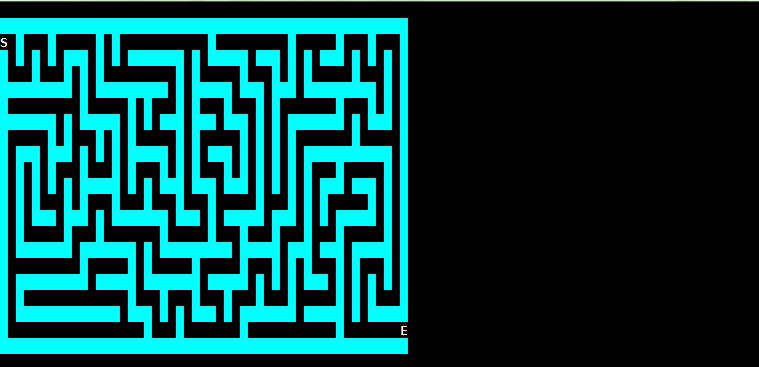
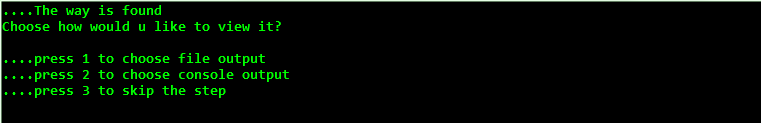


Рисунок 4.4 — Вивід згенерованого лабіринту в консоль

В будь - якому випадку, після вибору способу показу або останнього пункту меню (для пропуску даного етапа), користувач отримає повідомлення про успішне знаходження виходу із лабіринту та матиме змогу знову обрати спосіб показу результатів роботи програми.

Вікно матиме наступний вигляд (рис.4.5):

Рисунок 4.5 — Меню вибору способу демонстрації знайденого шляху

Користувачу пропонується вибір способу показу знайденого шляху. При виборі першого пункта меню, користувач повинен ввести шлях та ім’я для створюваного файлу для подальшого запису в нього. Для наглядності оберемо ту ж саму директорію (рис.4.6):



Рисунок 4.6 — Вибір директорії запису файлу

Отже новостворений файл під назвою «solved\_maze.txt» буде мати наступний вигляд (рис.4.7):

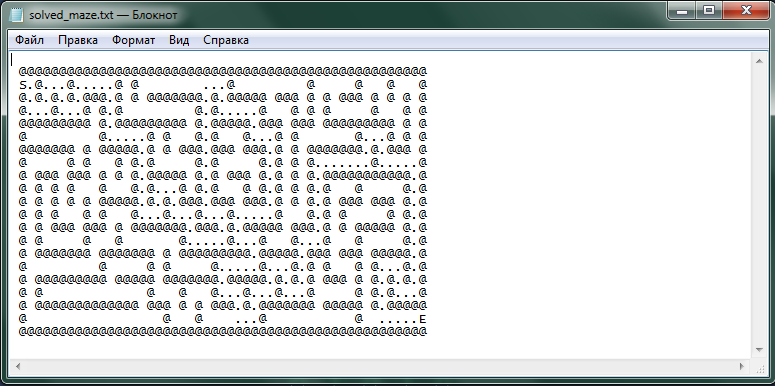


Рисунок 4.7 — Вивід вирішеного лабіринту в файл

При виборі другого пункта меню, вікно програми матиме вигляд (рис.4.8):

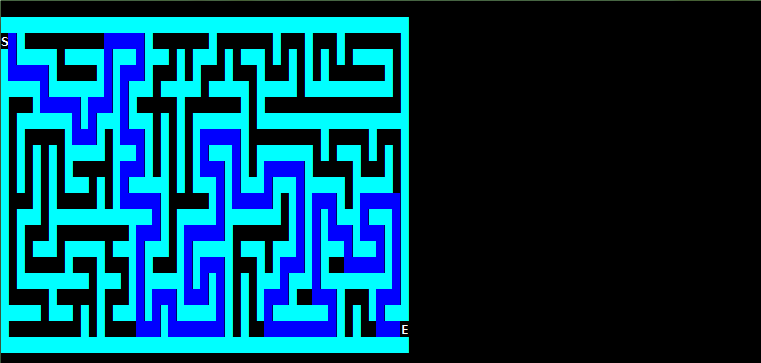


Рисунок 4.8 — Вивід вирішеного лабіринту в консоль

## 4.3 Завершення виконання программи

В будь - якому випадку, після вибору способу показу або останнього пункту меню (для пропуску даного етапа), користувач отримає запит (рис 4.9) на повторну генерацію, що матиме наступний вигляд:

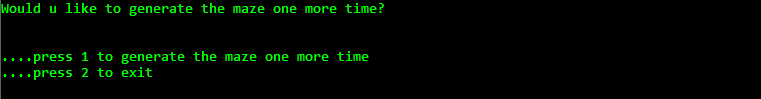


Рисунок 4.9 — Запит розпочати виконання програми спочатку

При виборі першого пункта — програма почне свою роботу спочатку, при виборі другого — завершить виконання супроводжуючись наступним повідомленням (рис.4.10):



Рисунок 4.10 — Завершення виконання програми

# ВИСНОВКИ

Під час виконання курсової роботи було розглянуто декілька відомих алгоритмів генерації та знаходження виходу із лабіринту і обрано алгоритми під назвою «recurcive backtracking» та «recurcive solving», що були детально описані в теоретичній частині курсової роботи.

Із переваг «recurcive backtracking» можна назвати простоту в реалізації, коректність роботи при будь-яких розмірах лабіринту та можливість роботи лише з двома станами елемента массиву. Із недоліків, на мою думку, можна виділити лише генерацію простих за структорою лабіринтів — так званих, «ідеальних» лабіринтів.

Також мною були розроблені користувацький інтерфейс разом із обробкою виключень та два можливих способи демонстрації згенерованого та вирішеного лабіринтів: вивід в консоль та вивід у файл.

Поставлену задачу реалізовано повністю. Програму можна вдосконалити шляхом додавання інших алгоритмів генерації лабіринту та знаходження виходу і порівняння швидкості роботи цих алгоритмів. Результати тестувань данного додатку помилок не виявили. Із недоліків можна виділити недопрацьований користувацький інтерфейс та доволі вузьку сферу використання розробленої програми.

Програма реалізована на мові програмування С++ в середовищі Microsoft Visual Studio. В ході виконання курсової роботи, я значно вдосконалив свої вміння програмування на мові С++ та закріпив практичні навички, пов’язані із оформленням офіційної документації на кожному етапі розробки.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Классические алгоритмы генерации лабиринтов. Часть 1 // Хабр: веб-сайт в формате коллективного блога. 2018. URL: <https://habr.com/ru/post/320140/>
2. Классические алгоритмы генерации лабиринтов. Часть 2 // Хабр: веб-сайт в формате коллективного блога. 2018. URL: <https://habr.com/ru/post/321210/>
3. Think Labyrinth: Maze Algorithms. URL: <http://www.astrolog.org/labyrnth/algrithm.htm>
4. Шилдт Г., C++: базовый курс, 3-е издание. –М.: Вильямс,2005. -624 ст.
5. Страуструп Б. Язык программирования C++, 3-е изд./ Пер. с англ.- СПб.; М.: ”Невский Диалект” - “Издательство БИНОМ”,1999 г. - 991 с.

# Додаток А Текст програми

#include "pch.h"

#include <Windows.h>

#include <iostream>

#include <ctime>

#include <fstream>

#include <string>

using namespace std;

//----CONSTANTS----------------------------

int width, height;

//----DEFINES------------------------------

#define UP 0

#define RIGHT 1

#define DOWN 2

#define LEFT 3

#define START 'S'

#define END 'E'

#define FREE ' '

#define VISITEDPOINT '.'

#define WALL (char)219

//----FUNCTIONS PROTOTYPES-----------------

void fillMaze(char\*\* maze);

void createMaze(char\*\* maze, int x, int y);

bool findWay(char\*\* maze, int x, int y);

void output(char\*\* maze);

void printMaze(char\*\* maze);

void printMaze(char\*\* maze, string path);

void enterSizes();

bool checkSizes(int width, int height);

bool inLimits(int x, int y);

void badInput();

void setColour(int text, int back);

void oneMore();

//----MAIN FUNCTION------------------------

int main() {

srand(time(0));

setColour(10, 0);

cout << "\n\t\t--- THE MAZE GENERATION AND SOLVING APPLICATION ---\n\n";

enterSizes();

// Створення масиву для зберігання лабіринта

char\*\* maze = new char\*[::width];

for (int i = 0; i < ::width; i++)

maze[i] = new char[::height];

// Створюю повноцінний лабіринт

fillMaze(maze);

createMaze(maze, 1, 1);

system("cls");

// Виводжу в консоль згенерований лабіринт

cout << "....The maze is created" << endl;

output(maze);

// Виводжу в консоль пройдений лабіринт

findWay(maze, 1, 1);

cout << "....The way is found" << endl;

output(maze);

// Видаляю двумірний динамічний масив

for (int i = 0; i < ::width; i++)

delete maze[i];

delete[]maze;

// Пропоную користувачу почати заново

oneMore();

return 0;

}

//----OTHER FUNCTIONS----------------------

void fillMaze(char\*\* maze) {

// Заповнення масива "стінами"

for (int y = 0; y < ::height; y++)

for (int x = 0; x < ::width; x++)

maze[x][y] = WALL;

}

void createMaze(char\*\* maze, int x, int y) {

// Створення "проходів" між стінами

maze[x][y] = FREE;

// Створюю массив для зберігання чотирьох можливих напрямків

int way[4];

for (int i = 0; i < 4; i++) way[i] = i;

// Змінюю порядок массива

for (int i = 0; i < 4; i++) {

int j = rand() & 3;

int temp = way[j];

way[j] = way[i];

way[i] = temp;

}

// Перебираю кожен напрямок і намагаюсь його пройти.

for (int i = 0; i < 4; i++) {

// remX,remY це зміщення відносно [x][y]

int remX = 0, remY = 0;

switch (way[i]) {

case UP: remY = -1; break;

case RIGHT: remX = 1; break;

case DOWN: remY = 1; break;

case LEFT: remX = -1; break;

}

// Знаходжу наступні координати [x][y] через два елементи

int nextX = x + (remX << 1);

int nextY = y + (remY << 1);

if (inLimits(nextX, nextY)) {

if (maze[nextX][nextY] == WALL) {

//"рyйуную" стіну між [x][y] та [nextX][nextY]

maze[nextX - remX][nextY - remY] = FREE;

createMaze(maze, nextX, nextY);

}

}

}

// Координати початку та кінця

maze[0][1] = START;

maze[::width - 1][::height - 2] = END;

}

bool findWay(char\*\* maze, int x, int y) {

// Рекурсивна функція для знаходження маршруту

if (maze[x][y] == END)

return true;

if (maze[x][y] == WALL || maze[x][y] == VISITEDPOINT)

return false;

maze[x][y] = VISITEDPOINT;

if (findWay(maze, x, y + 1)) //вниз

return true;

if (findWay(maze, x + 1, y)) //вправо

return true;

if (findWay(maze, x - 1, y)) //вліво

return true;

if (findWay(maze, x, y - 1)) //вверх

return true;

return false;

}

void output(char\*\* maze) {

// Меню вибора способа показу лабіринта

string path;

cout << "Choose how would u like to view it?";

cout << "\n\n....press 1 to choose file output";

cout << "\n....press 2 to choose console output";

cout << "\n....press 3 to skip the step\n\n";

int choice; cin >> choice;

switch (choice) {

case 1:

system("cls");

cout << "Enter the path -> "; cin >> path;

system("cls");

printMaze(maze, path);

break;

case 2:

system("cls");

printMaze(maze);

break;

case 3:

system("cls");

return;

default:

badInput();

output(maze);

break;

}

}

void printMaze(char\*\* maze) {

// Функція виводу лабіринта в консоль

for (int y = 0; y < ::height; y++) {

cout << "\n";

for (int x = 0; x < ::width; x++) {

setColour(11, 0);

if (maze[x][y] == START || maze[x][y] == END)setColour(15, 0);

if (maze[x][y] == VISITEDPOINT) setColour(9, 9);

cout << maze[x][y];

}

}

cout << "\n\n";

setColour(10, 0);

}

void printMaze(char\*\* maze, string path) {

// Функція виводу в файл

ofstream fout(path);

for (int y = 0; y < ::height; y++) {

fout << "\n ";

for (int x = 0; x < ::width; x++) {

if (maze[x][y] == WALL) fout << '@';

else fout << maze[x][y];

}

}

fout.close();

}

void enterSizes() {

// Ввод ширини та висоти і перевірка їх на валідність

cout << "Enter the maze WIDTH and HEIGHT of the maze -> ";

cin >> ::width >> ::height;

if (cin.fail() || !checkSizes(::width, ::height)) {

badInput();

cout << "....You can enter unpair numbers only in [5, 163] for a WIDTH \n\t\t\t\t and in [5, 41] for a HEIGHT\n";

enterSizes();

}

}

bool checkSizes(int width, int height) {

// Допоміжна функція для перевірки ширини в висоти

if (width < 5 || height < 5)

return false;

else if (width > 163 || height > 41)

return false;

else if (width % 2 == 0 || height % 2 == 0)

return false;

return true;

}

bool inLimits(int x, int y) {

// Перевірка індексів на коректність

if (x < 0 || x >= ::width)

return false;

if (y < 0 || y >= ::height)

return false;

return true;

}

void badInput() {

// функція для виправлення некоректного вводу

cin.clear();

cin.ignore(132, '\n');

system("cls");

}

void setColour(int text, int back) {

// Встановлення кольору консолі

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD)((back << 4) | text));

}

void oneMore() {

// Запит користувача почати спочатку

cout << "Would u like to generate the maze one more time?" << endl;

cout << "\n\n....press 1 to generate the maze one more time";

cout << "\n....press 2 to exit\n\n";

int choice; cin >> choice;

switch (choice) {

case 1:

system("cls");

main();

break;

case 2:

system("cls");

cout << "\n\t\t--- THANK U FOR USING --- \n\n ";

break;

default:

badInput();

oneMore();

break;

}

}

# Додаток Б Можливі результати роботи програми

Зауваження: при будь якому некоректному вводі, консоль очищується і користувач отримує можливість ввести дані повторно. Програма не рушить з місця, поки не отримає валідні дані, необхідні для коректного функціонування.

Далі наведено приклади реакції програми на будь які зразки некоректного вводу на різних етапах роботи програми.



Рисунок Б.1 — Користувач ввів некоректне значення розмірностей лабіринту.



Рисунок Б.2 — Користувач ввів недопустиме значення при виборі способу демонстрації лабіринту.

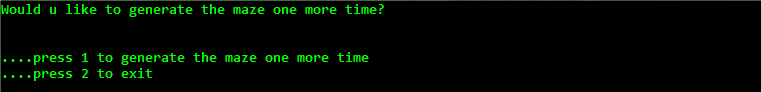


Рисунок Б.3 — Користувач ввів недопустиме значення при запиті розпочати програму спочатку.

# Додаток В Блок-схема алгоритму генерації лабіринту

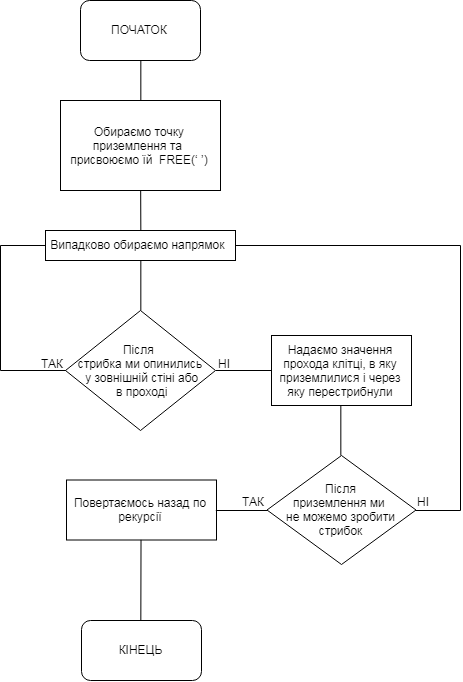


Рисунок В.1 — Блок-схема алгоритму генерації лабіринту