МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

**КУРСОВА РОБОТА**

з дисципліни «Основи програмування»

на тему: «Розв’язання задач про знаходження шляху в лабіринті»

Студента I курсу, групи ІП-31

Віжуткіна І.Д.

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник

Вітковська Ірина Іванівна

Кількість балів:

Члени комісії:

Головченко Максим Миколайович

Носов Костянтин Сергійович

Київ- 2024 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям «ІПЗ»

Курс 1 Група ІП-31 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Віжуткіна І. Д.

1. Тема роботи: розв’язання задач про знаходження шляху в лабіринті.

2. Строк здачі студентом закінченої роботи: 30 травня 2024р.

3. Вихідні дані до роботи: додаток А технічне завдання, додаток Б тексти програмного забезпечення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці): постановка задачі, теоретичні відомості, опис алгоритмів, опис програмного забезпечення, тестування, інструкція користувача, аналіз та узагальнення результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень ): -

6. Дата видачі завдання: 6 березня 2024р.

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |
| --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи |
| 1. | Отримання теми курсової роботи | 06.03.2024 |
| 2. | Підготовка ТЗ | 06.03.2024 |
| 3. | Пошук та вивчення літератури з питань курсової роботи | 10.03.2024 |
| 4. | Розробка сценарію роботи програми | 17.03.2024 |
| 6. | Узгодження сценарію роботи програми з керівником | 20.03.2024 |
| 5. | Розробка (вибір) алгоритму рішення задачі | 24.03.2024 |
| 6. | Узгодження алгоритму з керівником | 27.03.2024 |
| 7. | Узгодження з керівником інтерфейсу користувача | 27.03.2024 |
| 8. | Розробка програмного забезпечення | 07.04.2024 |
| 9. | Налагодження розрахункової частини програми | 14.04.2024 |
| 10. | Розробка та налагодження інтерфейсної частини програми | 14.04.2024 |
| 11. | Узгодження з керівником набору тестів для контрольного прикладу | 17.04.2024 |
| 12. | Тестування програми | 17.04.2024 |
| 13. | Підготовка пояснювальної записки | 21.04.2024 |
| 14. | Здача курсової роботи на перевірку | 30.05.2024 |
| 15. | Захист курсової роботи | 03.06.2024 |

Студент: Віжуткін І. Д.

Керівник: Вітковська І. І.

«21» травня 2024 р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 79 сторінок, 23 рисунки, 15 таблиць, 4 посилання.

Мета роботи: метою курсової робити є забезпечення точності та надійності роботи програмного забезпечення для пошуку шляху в лабіринті за допомогою методів Дейкстри та А\*(манхеттенська та евклідова евристики).

Вивчено методи пошуку найкоротшого шляху в лабіринті, а саме, метод Дейкстри та А\*(манхеттенська та евклідова евристики).

Виконана програмна реалізація відповідних алгоритмів Дейкстри та А\*(манхеттенська та еквлідова евристики) з отриманням достовірних результатів.

ЛАБІРИНТ, НАЙКОРОТШИЙ ШЛЯХ В ЛАБІРИНТІ, МЕТОД ДЕЙКСТРИ, МЕТОД А\*.

**ЗМІСТ**

[ВСТУП 5](#_Toc166924011)

[1 ПОСТАНОВКА ЗАДАЧІ 6](#_Toc166924012)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ 7](#_Toc166924013)

[2.1. Метод Дейкстри [1] 7](#_Toc166924014)

[2.2. Метод A\*(манхеттенська та евклідова евристики) [2] 8](#_Toc166924015)

[3 ОПИС АЛГОРИТМІВ 11](#_Toc166924016)

[3.1. Загальний алгоритм 11](#_Toc166924017)

[3.2. Спільний алгоритм 13](#_Toc166924018)

[4 ОПИС ПРОГРАМНОГО ЗАБЕПЕЧЕННЯ 15](#_Toc166924019)

[4.1. Діаграма класів програмного забезпечення 15](#_Toc166924020)

[4.2. Опис методів частин програмного забезпечення 15](#_Toc166924021)

[4.2.1. Стандартні методи 15](#_Toc166924022)

[4.2.2. Користувацькі методи 18](#_Toc166924023)

[5 ТЕСТУВАННЯ 25](#_Toc166924024)

[5.1. План тестування 25](#_Toc166924025)

[5.2. Приклади тестування 25](#_Toc166924026)

[6 ІНСТРУКЦІЯ КОРИСТУВАЧА 34](#_Toc166924027)

[6.1. Робота з програмою 34](#_Toc166924028)

[6.2. Системні вимоги 43](#_Toc166924029)

[7 АНАЛІЗ ТА УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ 44](#_Toc166924030)

[ВИСНОВКИ 52](#_Toc166924031)

[ПЕРЕЛІК ПОСИЛАНЬ 54](#_Toc166924032)

[ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ 55](#_Toc166924033)

[ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 58](#_Toc166924034)

**ВСТУП**

В сучасному світі технологій, розробка нового програмного забезпечення для девайсів та його постійне оновлення є вкрай необхідним, і тема пошуку найкоротших шляхів в лабіринті не є винятком. Швидкі та ефективні алгоритми пошуку маршрутів мають велике значення в різноманітних областях, від розробки ігор до навігації роботів. Постійний розвиток технологій та зростання вимог до швидкості та точності обробки даних змушують розробників створювати нові алгоритми та підходи, для вдосконалення старих та створення нових програм пошуку шляхів.

Основною метою створення будь-яких програм, в цілому, є автоматизація праці, що дозволяє значно полегшити та прискорити процеси, які, в іншому випадку, вимагатимуть великої кількості людських ресурсів та часу. Автоматизуючи, люди мають можливість перекласти частину рутинної праці на машини, в той час зосереджуючись на більш складних процесах.

У рамках даної курсової роботи, метою є створення програмного забезпечення, яке дозволить користувачам легко і ефективно шукати найкоротші шляхи в лабіринті за допомогою двох популярних алгоритмів: Дейкстри та A\*. Це передбачає розробку інтуїтивного графічного інтерфейсу, де користувачі зможуть задавати власні лабіринти та отримувати оптимальні маршрути. Розв'язання цієї задачі сприятиме не лише поглибленню розуміння роботи вказаних алгоритмів, а й надасть можливість практичного застосування отриманих знань у сфері розробки програмного забезпечення.

**1 ПОСТАНОВКА ЗАДАЧІ**

Розробити програмне забезпечення, що буде знаходити шлях в лабіринті наступними методами:

а) метод Дейкстри;

б) метод A\*(манхеттенська та евклідова евристики).

Вхідними даними для даної роботи є метод розв’язку та графічний варіант лабіринту, в який можуть входити:

а) стіна(0..\*);

б) прохід(1..\*);

в) початок(1);

г) кінець(1).

При некоректній кількості якогось з елементів, виводиться помилка. Якщо метод не вибраний, виводиться помилка. Лабіринт має квадратну форму. Мінімальний розмір лабіринту = 2(2 рядки, 2 стовці), максималньий = 25(25 рядків, 25 стовпців). Є можливість задавати лабіринт випадковим чином.

Вихідними даними, за умови, що шлях існує, є графічний варіант лабіринту із відміченим правильним шляхом та кількість відвіданих комірок під час пошуку.

Якщо знайти шлях не вдається, виводиться помилка. Є можливість зберігати розв’язок у файл.

**2 ТЕОРЕТИЧНІ ВІДОМОСТІ**

Лабіринт можна задати, наприклад, наступним чином:

де 1 – це прохід, 0 – стіна, s – старт, e – кінець

* 1. Метод Дейкстри [1]

Сутність методу Дейкстри полягає в тому, щоб систематично розглядати всі можливі шляхи, починаючи зі стартової вершини і вибираючи на кожному кроці найкоротший шлях із доступних. Цей алгоритм гарантує знаходження найкоротшого шляху, за умови, що шлях існує, і в графі немає ребер із від’ємними вагами.

Орієнтовний приклад роботи для заданого лабіринту:

* зелений колір позначає відвідану вершину;
* помаранчевий колір позначає вершини в черзі на відвідування;
* червоний колір позначає пройдену стінку;

Для спрощення, поки що, розглянемо тільки вертикальний та горизонтальний рух.

Беремо початкову вершину і шукаємо її сусідів.

Довжина шляху до всіх сусідів від початкової вершини = 1, відвідуємо відомих нам сусідів та додаємо нових.

Тепер довжина до усіх сусідів від початкової вершини = 2, знову відвідуємо відомих нам сусідів та додаємо нових.

Довжина до усіх сусідів від початкової вершини = 3, знову проходимось по всім сусідам. Під час проходження ми знайдемо кінець.

* 1. Метод A\*(манхеттенська та евклідова евристики) [2]

Алгоритм A\* поєднує в собі ідеї алгоритму Дейкстри і евристичого «приблизного» шляху. Він робить ті ж самі дії, що й алгоритм Дейкстри, але при виборі наступного «можливо» найкращого шляху, він спирається не тільки на поточну довжину до вершини, але й на приблизну оцінку довжини від цієї вершини до кінця. Алгоритм A\* є значно ефективнішим, за умови, що підібрана правильна евристика. Цей алгоритм також гарантує знаходження найкоротшого шляху, за умови, що шлях існує, і в графі немає ребер із від’ємними вагами.

В цій курсовій роботі будуть розглянуті наступні евристики для алгоритму А\*:

а) манхеттенська ( );

б) евклідова ( ).

Орієнтовний приклад роботи для заданого лабіринту:

* зелений колір позначає відвідану вершину;
* помаранчевий колір позначає вершини в черзі на відвідування;
* червоний колір позначає пройдену стінку;

Для спрощення, поки що, розглянемо тільки вертикальний та горизонтальний рух.

Беремо початкову вершину і шукаємо її сусідів.

Довжина шляху до всіх сусідів від початкової вершини = 1, але використовуючи, наприклад, манхеттенську евристику, розуміємо, що найкраще нам підходить саме верхній прохід.

Довжина до деяких сусідів = 1, до деяких 2, але зважаючи на евристику, шлях довжиною 2 може бути вигіднішим, що і відбувається в цьому прикладі.

Аналогічно доходимо до кінця.

**3 ОПИС АЛГОРИТМІВ**

|  |  |  |
| --- | --- | --- |
| Опис | Змінна | Тип |
| Всі вершини | vertices[] | Масив вершин |
| Словник для запису шляху | parentMap | Словник |
| Черга для коректної роботи алгоритму | priorityQueue | Черга з пріоритетом [3] |
| Початкова вершина | start | Вершина |
| Кінцева вершина | end | Вершина |
| Поточна вершина | current | Вершина |
| Поточний сусід | neighbour | Вершина |
| Шлях від початку до кінця | path | Список |

* 1. Загальний алгоритм

ПОЧАТОК

1. Зчитати розмірність лабіринту
2. Намалювати лабіринт, який має тільки проходи
3. Дозволити заміняти місця в лабіринті на стіну, прохід, старт, кінець
4. Дозволити вибрати метод розв’язання
5. Активувати кнопку «Сгенерувати лабіринт»
6. Активувати кнопку «Знайти»
7. Очікувати дій користувача
8. ЯКЩО натиснуто «сгенерувати лабіринт»

8.1. ЦИКЛ проходу по всіх місцях лабіринту

8.1.1. Випадково вибрати між «стінка» і «прохід»

8.1.2. ЯКЩО «стінка»

8.1.1.2.1.1. Замінити місце на стінку

ІНАКШЕ ЯКЩО «прохід»

8.1.1.2.2.1. Замінити місце на прохід

8.2. Випадково вибрати місце в лабіринті

8.3. Замінити це місце на старт

8.4. Випадково вибрати місце в лабіринті відмінне від старту

8.5. Замінити це місце на кінець

9. ЯКЩО натиснуто кнопку «Знайти»

9.1. ЯКЩО лабіринт не має рівно 1 старт чи не має рівно 1 кінець

9.1.1. Вивести «Лабіринт має мати 1 старт і 1 кінець»

9.1.2. Повернутися до кроку 7

9.2. ЯКЩО метод розв’язання не вибраний

9.2.1. Вивести «Метод розв’язання не вибрано»

9.2.2. Повернутися до кроку 7

9.3. ЯКЩО вибрано метод Дейкстри

9.3.1.1. Розв’язати лабіринт алгоритмом 3.2(із частиною 1)

ІНАКШЕ ЯКЩО вибрано метод A\*(манхеттенська евристика)

9.3.2.1. Розв’язати лабіринт алгоритмом 3.2(із частиною 2)

ІНАКШЕ ЯКЩО вибрано метод A\*(евклідова евристика)

9.3.3.1. Розв’язати лабіринт алгоритмом 3.2(із частиною 3)

9.4. ЯКЩО розв’язок знайдено

9.4.1.1. Відкрити інше вікно

9.4.1.2. Зобразити лабіринт і правильний шлях на ньому

9.4.1.3. Вивести кількість відвіданих комірок під час роботи алгоритму

9.4.1.4. Активувати кнопку «Зберегти у файл»

9.4.1.5. Активувати кнопку «Повернутися»

9.4.1.6. ЯКЩО натиснуто кнопку «Зберегти у файл»

9.4.1.6.1. Запитати шлях до папки

9.4.1.6.2. Створити текстовий файл із вихідним лабіринтом в указаній папці

9.4.1.7. ЯКЩО натиснуто кнопку «Повернутися»

9.4.1.7.1. Закрити вікно

ІНАКШЕ

9.4.2.1. Вивести «Шляху не існує»

9.4.2.2 Повернутися до кроку 7

КІНЕЦЬ

* 1. Спільний алгоритм

ПОЧАТОК

1. ЦИКЛ проходу по кожній вершині(current) з vertices

1.1. Встановити current.відвіданість = «не відвідана»

1.2. Встановити current.відстань\_від\_початку = «нескінченність»

2. Встановити вершині start.відстань\_від\_початку = 0

3. Додати вершину start до priorityQueue із пріоритетом 0

4. ПОКИ priorityQueue не пуста

4.1. Присвоїти current значення взяте із priorityQueue

4.2. Встановити current.відвіданість = «відвідана»

4.3. ЯКЩО current == end

4.3.1. ПОКИ current != start

4.3.1.1. Додати current на початок path

4.3.1.2. Присвоїти current = значення parentMap із ключом current

4.3.2. Додати start на початок path

4.3.3. ПОВЕРНУТИ path

КІНЕЦЬ

4.4. ЦИКЛ проходу по кожному сусіду(neighbour) вершини current

4.4.1. ЯКЩО neighbour.відвіданість == «не відвідана»

4.4.1.1. ЯКЩО neighbour.відстань\_від\_початку > current.відстань\_від\_початку + current.відстань\_до\_сусіда(neighbour)

4.4.1.1.1. Встановити neighbour.відстань\_від\_початку = current.відстань\_від\_початку + current.відстань\_до\_сусіда(neighbour)

4.4.1.1.2. Додати neighbor до parentMap із значенням current

**1. Алгоритм Дейкстри**

4.4.1.1.3. Додати neighbour до priorityQueue із пріоритетом = neighbour.відстань\_від\_початку

**2. Алгоритм А\*(манхеттенська евристика)**

4.4.1.1.3. Додати neighbour до priorityQueue із пріоритетом = neighbour.відстань\_від\_початку +

**3. Алгоритм А\*(евклідова евристика)**

4.4.1.1.3. Додати сусіда до priorityQueue із пріоритетом = neighbour.відстань\_від\_початку +

5. Вивести помилку, що шлях не знайдено

КІНЕЦЬ

**4 ОПИС ПРОГРАМНОГО ЗАБЕПЕЧЕННЯ**

4.1. Діаграма класів програмного забезпечення

На рисунку 4.1 зображена діаграма класів, яка описує класи, використані під час розробки ПЗ.

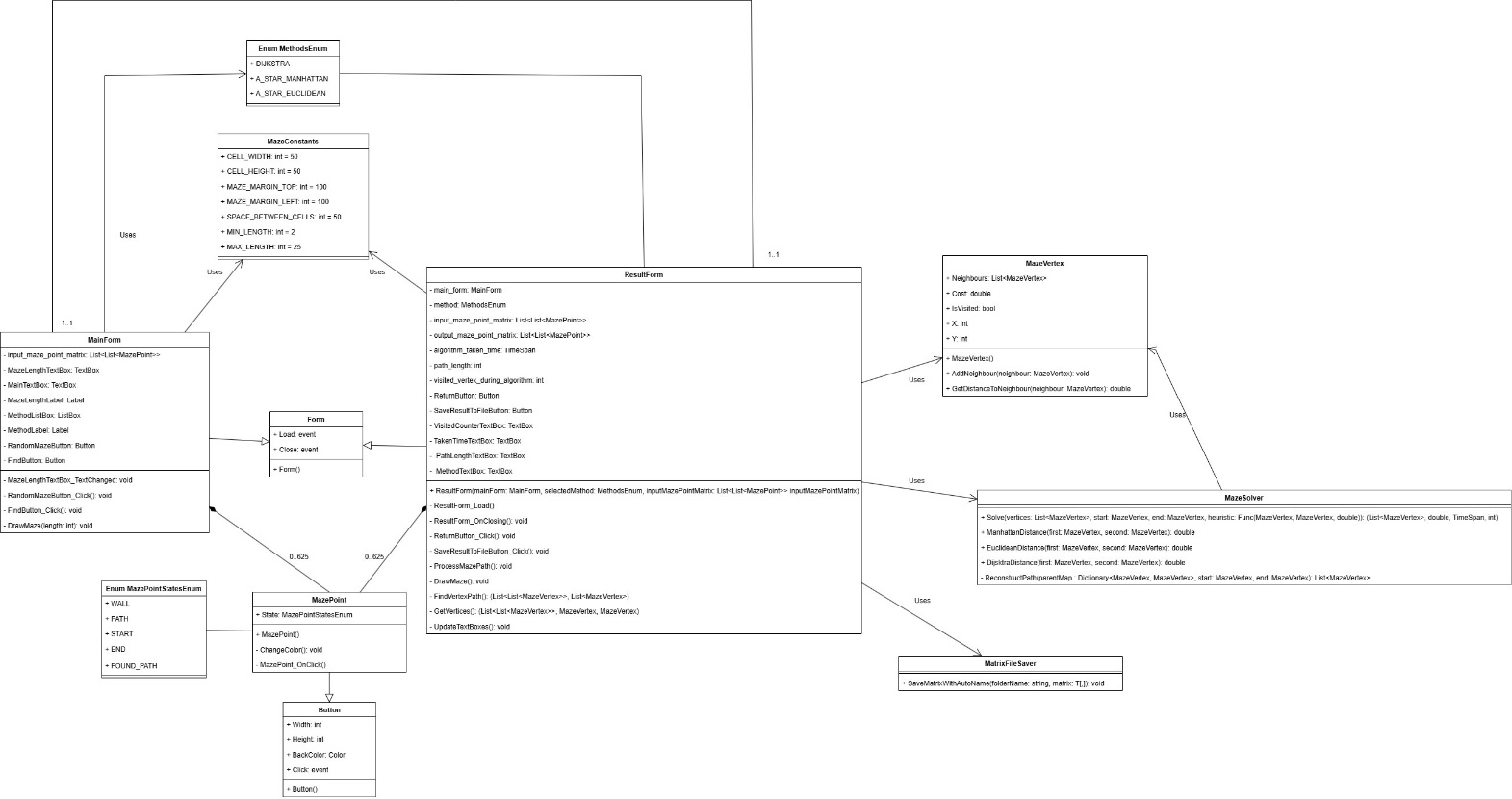


Рисунок 4.1 – Діаграма класів

4.2. Опис методів частин програмного забезпечення

4.2.1. Стандартні методи

У таблиці 4.1 наведено вбудовані методи, використані під час розробки ПЗ.

Таблиця 4.1 - Стандартні методи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 1.1 | Math | Abs | Знаходження модуля числа | Число | Модуль числа |
| 1.2 | Math | Sqrt | Знаходження квадратного кореня числа | Число | Квадратний корінь числа |
| 1.3 | Math | Pow | Піднесення числа в степінь | Число  Степінь | Число в степені |
| 2.1 | StreamWriter | Write | Написання строки в текстовий файл | Строка |  |
| 2.2 | StreamWriter | WriteLine | Написання нової строки в текстовий файл |  |  |
| 3.1 | List | Add | Додавання нового елементу в кінець списку | Елемент |  |
| 4.1 | PriorityQueue | Enqueue | Додавання елементу до черги | Елемент  Пріоритет |  |

Продовдження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 4.2 | PriorityQueue | Dequeue | Забирання елементу з черги |  | Елемент із найменшим пріоритетом |
| 5.1 | Stopwatch | Start | Початок відліку часу |  |  |
| 5.2 | Stopwatch | Stop | Закінчення відліку часу |  |  |
| 6.1 | int | TryParse | Спроба отримати ціле число із строки. Якщо успішно, то записує значення числа в out змінну | Строка  out Змінна | True/False |
| 7.1 | Random | Next | Отримання випадково цілого числа | Мінімальне число  Максимальне число | Випадкове число |

4.2.2. Користувацькі методи

У таблиці 4.2 наведено методи, що були реалізовані самостійно під час розробки ПЗ.

Таблиця 4.2 – Користувацькі методи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №№ | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 11.1 | MazeVertex | AddNeighbour | Додавання суміжної вершини | Суміжна вершина |  |
| 11.2 | MazeVertex | GetDistanceToNeighbour | Знаходження відстані до сусіда | Сусідня вершина | 1 або |
| 22.1 | MazeSolver | ManhattanDistance | Використання в методі Solve | Дві вершини | Манхеттеньска відстань між двома вершинами |
| 22.2 | MazeSolver | Euclidean Distance | Використання в методі Solve | Дві вершини | Евклідова відстань між двома вершинами |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №№ | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 22.3 | MazeSolver | Dijkstra Distance | Використання в методі Solve | Дві вершини | Завжди повертає 0 |
| 22.4 | MazeSolver | ReconstructPath | Відбудування знайденого шляху в методі Solve | Словник, де ключ це вершина №1, а значення - це інша вершина, звідки була знайдена вершина №1.  Початкова вершина  Кінцева вершина | Шлях із вершин |
| 22.5 | MazeSolver | Solve | Пошук шляху між двома вершинами методом Дейкстри чи A\*. Метод залежить від переданої евристичної функції в аргументі | Граф  Початкова вершина  Кінцева вершина  Евристична функція | Шлях із вершин |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ;№ | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 33.1 | MatrixFileSaaver | SaveMatrixWithAutoName | Збереження заданої матриці у файл, із автоматичною назвою | Шлях до папки  Матриця |  |
| 44.1 | MazePoint | ChangeColor | Зміна кольору кнопки, в залежності від її стану |  |  |
| 44.2 | MazePoint | MazePoint\_OnClick | Викликається при натисканні на кнопку. Змінює стан на наступний |  |  |
| 55.1 | MainForm | MazeLengthTextBox\_TextChanged | Викликається при зміні значення в MazeLengthTextBox.  Перевіряє, чи введена коректна довжина. Якщо так, викликає метод DrawMaze та вмикає кнопку RandomMazeButton, інакше виводить повідомлення |  |  |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №№ | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 55.2 | MainForm | RandomMazeButton\_Click | Викликається при натисканні на кнопку RandomMazebutton. Будує випадковий лабіринт |  |  |
| 55.3 | MainForm | FindButton\_Click | Викликається при натисканні на кнопку FindButton. Перевіряє введений лабіринт на коректність. Якщо коректний, створює та відкриває ResultForm, інакше виводить повідомлення |  |  |
| 55.4 | MainForm | DrawMaze | Малює пустий лабіринт із заданої розмірністю | Розмірність |  |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №№ | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 66.1 | ResultForm | ResultForm\_Load | Викликається при повному завантаженні форми. Викликає ProcessMazePath. Якщо  ProccessMazePath виконується без виключень, то викликає,  DrawMaze, UpdateTextBoxes, інакше виводить повідомлення і закриває форму |  |  |
| 66.2 | ResultForm | ResultForm\_OnClosing | Викликається при закритті форми. Відкриває минулу форму MainForm |  |  |
| 66.3 | ResultForm | ReturnButton\_Click | Викликається при натисканні на ReturnButton. Закриває форму |  |  |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №№ | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 66.4 | ResultForm | SaveResultToFileButton\_Click | Викликається при натисканні на SaveResultToFileButton. Запрошує шлях до папки і зберігає туди текстовий файл із лабіринтом |  |  |
| 66.5 | ResultForm | ProcessMazePath | Викликає FindVertexPath і за допомогою результату заповнює output\_maze\_point\_matrix |  |  |
| 66.6 | ResultForm | DrawMaze | Малювання лабіринту, використовуючи output\_maze\_point\_matrix |  |  |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №№ | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 66.7 | ResultForm | FindVertexPath | Викликає GetVertices і за допомогою результата та MazeSolver знаходить шлях в лабіринті, оновлює значення algorithm\_taken\_time, path\_length, visited\_vertex\_during\_algorithm |  | Початковий двовимірний список із вершинами, який репрезентує лабіринт  Лист із вершинами, який репрезентує правильний шлях |
| 66.8 | ResultForm | GetVertices | Використовує input\_maze\_point\_matrix, щоб знайти репрезентацію лабіринту за допомогою вершин |  | Двовимірний список із вершинами, який репрезентує лабіринт  Стартову вершину  Кінцеву вершину |
| 66.9 | ResultForm | UpdateTextBoxes | Використовуючи атрибути, оновлює текст у деяких TextBox |  |  |

**5 ТЕСТУВАННЯ**

5.1. План тестування

Вважаючи за необхідне стабільне функціювання мого програмного забезпечення, мною було заплановано проведення систематичних тестів з метою виявлення та усунення будь-яких можливих недоліків та помилок, які можуть виникнути в процесі його використання.

а) Тестування правильності введення розмірності лабіринту.

1) Тестування при введенні значення, яке менше за мінімальне або більше за максимальне.

2) Тестування при введенні не цілого значення.

б) Тестування наявності всього необхідного для знаходження шляху.

1) Тестування при не обраному методі.

2) Тестування, коли лабіринт немає 1 початку і 1 кінця.

3) Тестування, коли лабіринт має більше ніж 1 початок чи 1 кінець.

в) Тестування, коли шляху не існує.

г) Тестування знаходження шляху.

1) Тестування метода Дейкстри.

2) Тестування метода А\*(манхеттенська евристика).

3) Тестування метода А\*(евклідова евристика).

д) Тестування збереження результату в файл.

5.2. Приклади тестування

Проведемо тестування(таблиці 5.1-5.10).

Таблиця 5.1 – Приклад роботи програми при введенні розмірності, яка менше за мінімальну або більше за максимальну

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректної розмірності |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | 1 або 26 |
| Схема проведення тесту | Заповнення рядку «розмірність» вхідними даними |
| Очікуваний результат | Виділення тексту іншим кольором |
| Стан програми після проведення випробувань | Текст виділено червоним кольором |

Таблиця 5.2 – Приклад роботи програми при введенні не цілого значення у розмірність

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректної розмірності |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | 1.5 або «ba» |
| Схема проведення тесту | Заповнення рядку «розмірність» вхідними даними |
| Очікуваний результат | Виділення тексту іншим кольором та вивід повідомлення про необхідність цілого числа |
| Стан програми після проведення випробувань | Текст виділено червоним кольором та виведено «Введіть ціле число» |

Таблиця 5.3 – Приклад роботи програми при спробі знаходження шляху без вибраного методу

|  |  |
| --- | --- |
| Мета тесту | Перевірити валідування необхідних вхідних даних для пошуку шляху |
| Початковий стан програми | Лабіринт сгенеровано |
| Вхідні дані | «Метод не обрано» |
| Схема проведення тесту | Натиснення «Знайти» без вибору метода |
| Очікуваний результат | Повідомлення, що метод не обрано |
| Стан програми після проведення випробувань | Виведено «Спочатку виберіть метод» |

Таблиця 5.4 – Приклад роботи програми при спробі знаходження шляху в лабіринті, який не має 1 старту і 1 кінця

|  |  |
| --- | --- |
| Мета тесту | Перевірити валідування необхідних вхідних даних для пошуку шляху |
| Початковий стан програми | Лабіринт сгенеровано і обрано метод |
| Вхідні дані |  |
| Схема проведення тесту | Заповнення лабіринту відповідними даними і натискання «Знайти» |
| Очікуваний результат | Повідомлення, що лабіринт не має рівно 1 старту та 1 кінця |
| Стан програми після проведення випробувань | Виведено «Лабіринт не має старту чи кінця» |

Таблиця 5.5 – Приклад роботи програми при спробі знаходження шляху в лабіринті, який має більше 1 старту чи 1 кінця

|  |  |
| --- | --- |
| Мета тесту | Перевірити валідування необхідних вхідних даних для пошуку шляху |
| Початковий стан програми | Лабіринт сгенеровано і обрано метод |
| Вхідні дані |  |
| Схема проведення тесту | Заповнення лабіринту відповідними даними і натискання «Знайти» |
| Очікуваний результат | Повідомлення, що лабіринт не має рівно 1 старту та 1 кінця |
| Стан програми після проведення випробувань | Виведено «Лабіринт має більше ніж один старт чи один кінець» |

Таблиця 5.6 – Приклад роботи програми при спробі знаходження шляху в лабіринті, який не має шляху

|  |  |
| --- | --- |
| Мета тесту | Перевірити реакції програми на лабіринт без можливого шляху |
| Початковий стан програми | Лабіринт сгенеровано і обрано метод |
| Вхідні дані |  |
| Схема проведення тесту | Заповнення лабіринту відповідними даними і натискання «Знайти» |
| Очікуваний результат | Повідомлення, що шляху не існує |
| Стан програми після проведення випробувань | Виведено «Шляху не існує, повернення…» |

Таблиця 5.7 – Приклад пошуку шляху методом Дейкстри

|  |  |
| --- | --- |
| Мета тесту | Перевірити пошук шляху методом Дейкстри |
| Початковий стан програми | Лабіринт сгенеровано і обрано метод Дейкстри |
| Вхідні дані |  |
| Схема проведення тесту | Заповнення лабіринту відповідними даними і натискання «Знайти» |
| Очікуваний результат |  |
| Стан програми після проведення випробувань | Виведено коректний шлях |

Таблиця 5.8 – Приклад пошуку шляху методом А\*(манхеттенська евристика)

|  |  |
| --- | --- |
| Мета тесту | Перевірити пошук шляху методом А\*(манхеттенська евристика) |
| Початковий стан програми | Лабіринт сгенеровано і обрано метод А\*(манхеттенська евристика) |
| Вхідні дані |  |
| Схема проведення тесту | Заповнення лабіринту відповідними даними і натискання «Знайти» |
| Очікуваний результат |  |
| Стан програми після проведення випробувань | Виведено коректний шлях |

Таблиця 5.9 – Приклад пошуку шляху методом А\*(евклідова евристика)

|  |  |
| --- | --- |
| Мета тесту | Перевірити пошук шляху методом А\*(манхеттенська евристика) |
| Початковий стан програми | Лабіринт сгенеровано і обрано метод А\*(евклідова евристика) |
| Вхідні дані |  |
| Схема проведення тесту | Заповнення лабіринту відповідними даними і натискання «Знайти» |
| Очікуваний результат |  |
| Стан програми після проведення випробувань | Виведено коректний шлях |

Таблиця 5.10 – Приклад збереження вихідного лабіринту в файл

|  |  |
| --- | --- |
| Мета тесту | Перевірити збереження вихідного лабіринту у файл |
| Початковий стан програми | Вихідний лабіринт знайдено |
| Вхідні дані | «Шлях до папки» |
| Схема проведення тесту | Успішне отримання будь-якого вихідного лабіринту. Натиснення «Зберегти результат в файл» і обрання папки |
| Очікуваний результат | Створено файл із вихідним лабіринтом |
| Стан програми після проведення випробувань | Створено файл із вихідним лабіринтом |

**6 ІНСТРУКЦІЯ КОРИСТУВАЧА**

* 1. Робота з програмою

Після запуску файлу \*.exe, відкривається головне вікно програми (Рисунок 6.1).

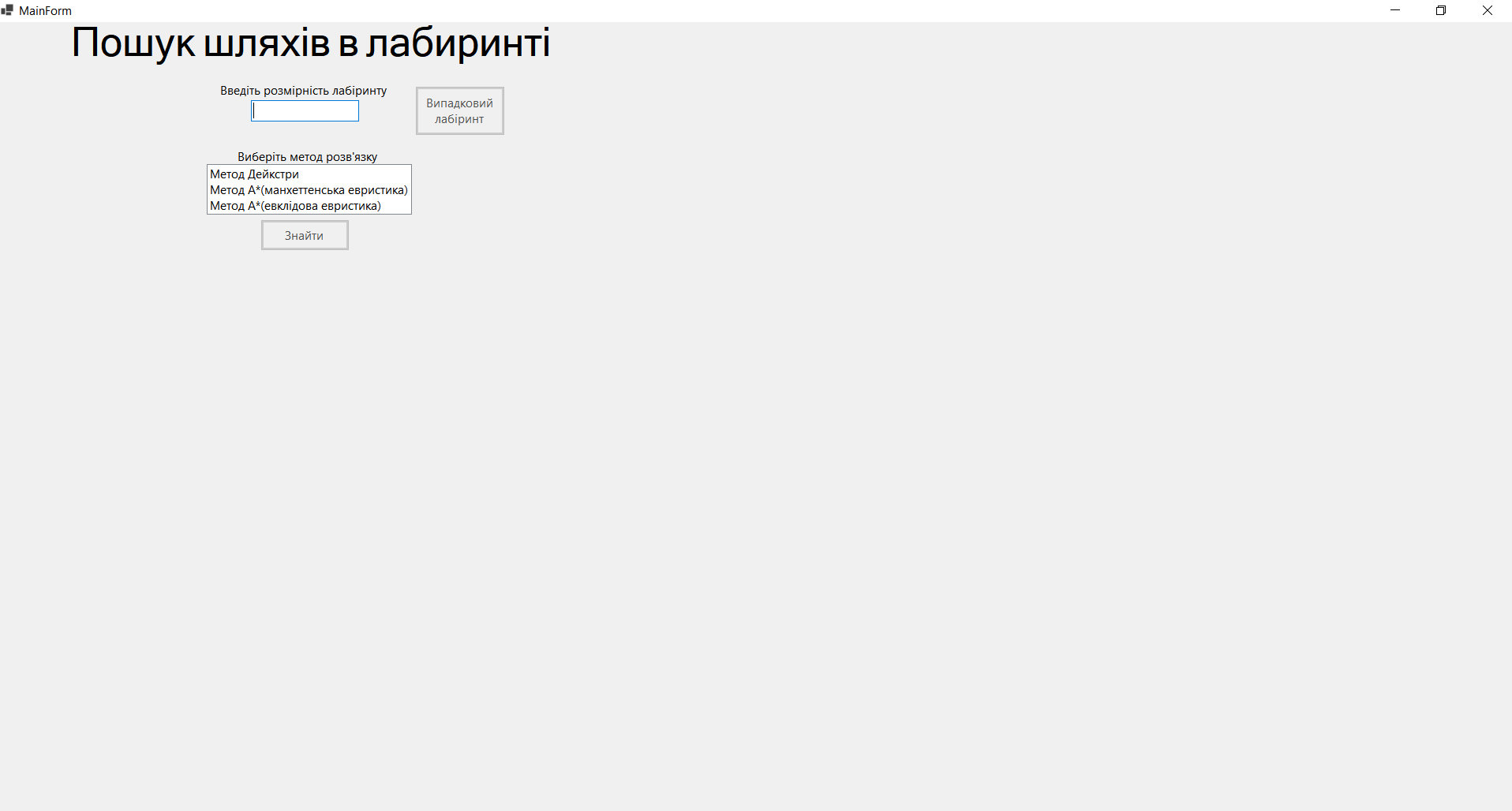


Рисунок 6.1 – Головне вікно програми

Далі за допомогою клавіатури необхідно задати розмірність лабіринту. Після коректного введення(число від 2 до 25) буде сформовано пустий лабіринт (Рисунок 6.2).

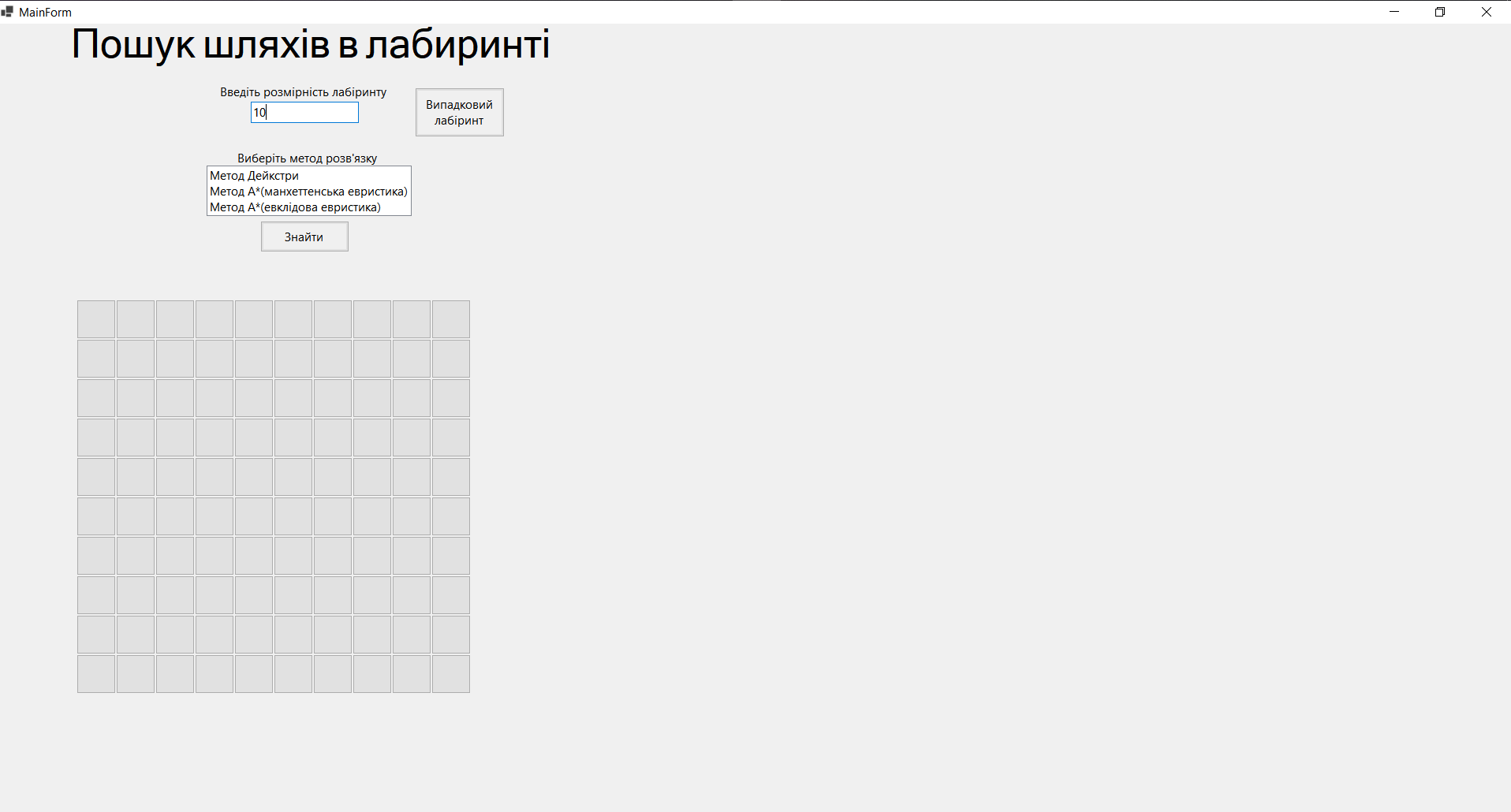


Рисунок 6.2 – Формування лабіринту розмірністю 10

Наступним кроком буде введення лабіринту вручну або автоматично. Для введення вручну необхідно натискати на клітинки:

* перше натискання змінить прохід на стінку (Рисунок 6.3);
* друге натискання змінить стінку на початок (Рисунок 6.4);
* третя натискання змінить початок на кінець (Рисунок 6.5);
* четверте натискання поверне клітинку до початкового стану – прохід (Рисунок 6.6);

Для введення випадковим чином необхідно натиснути кнопку «Випадковий лабіринт» (Рисунок 6.7).

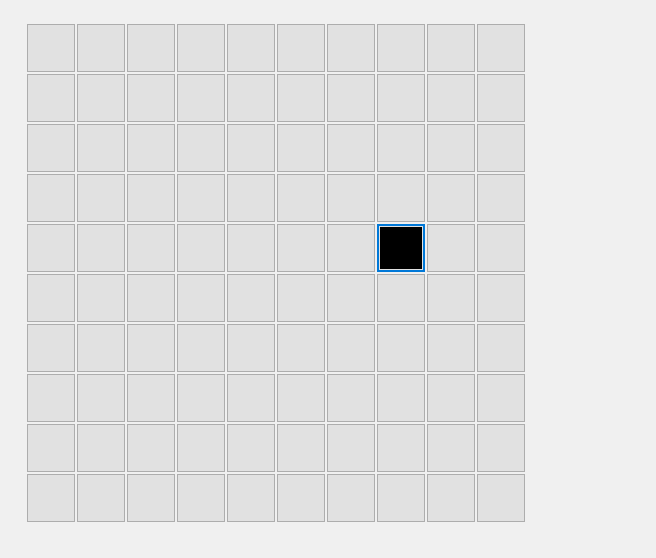


Рисунок 6.3 – Перше натискання

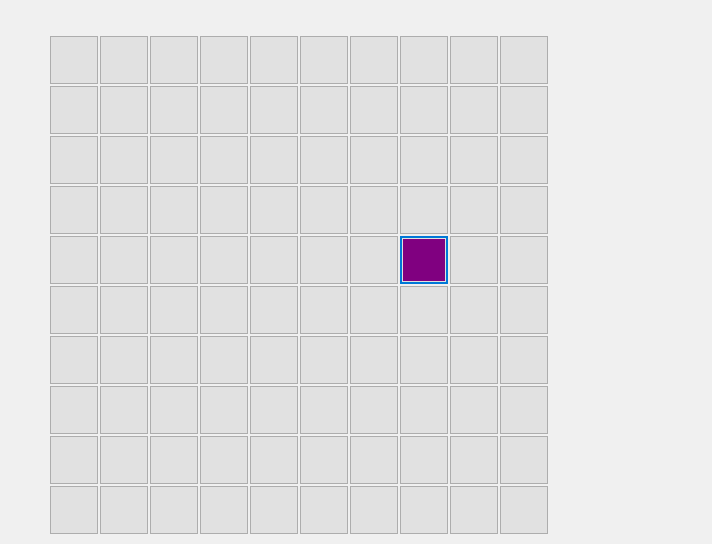


Рисунок 6.4 – Друге натискання

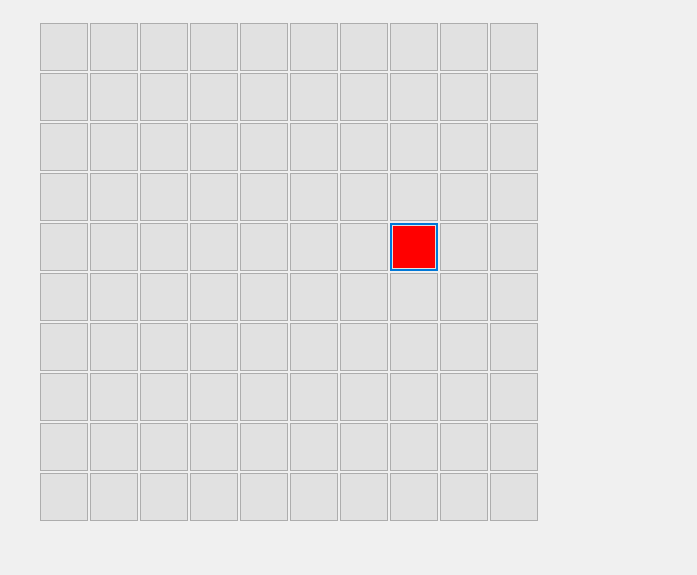


Рисунок 6.5 – Третє натискання

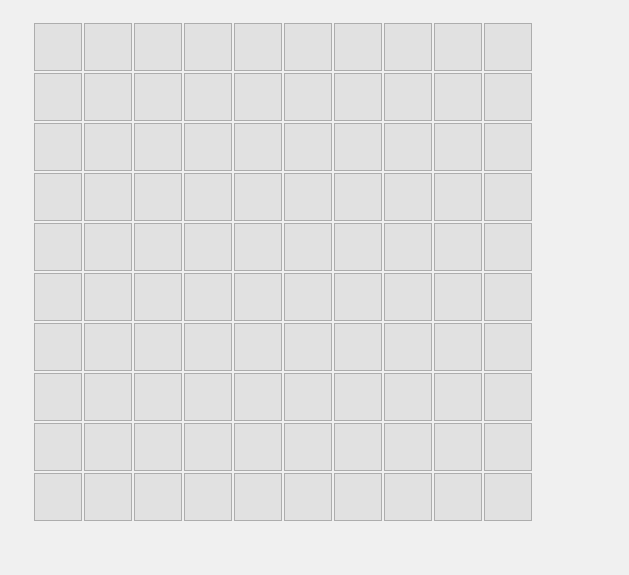


Рисунок 6.6 – Четверте натискання

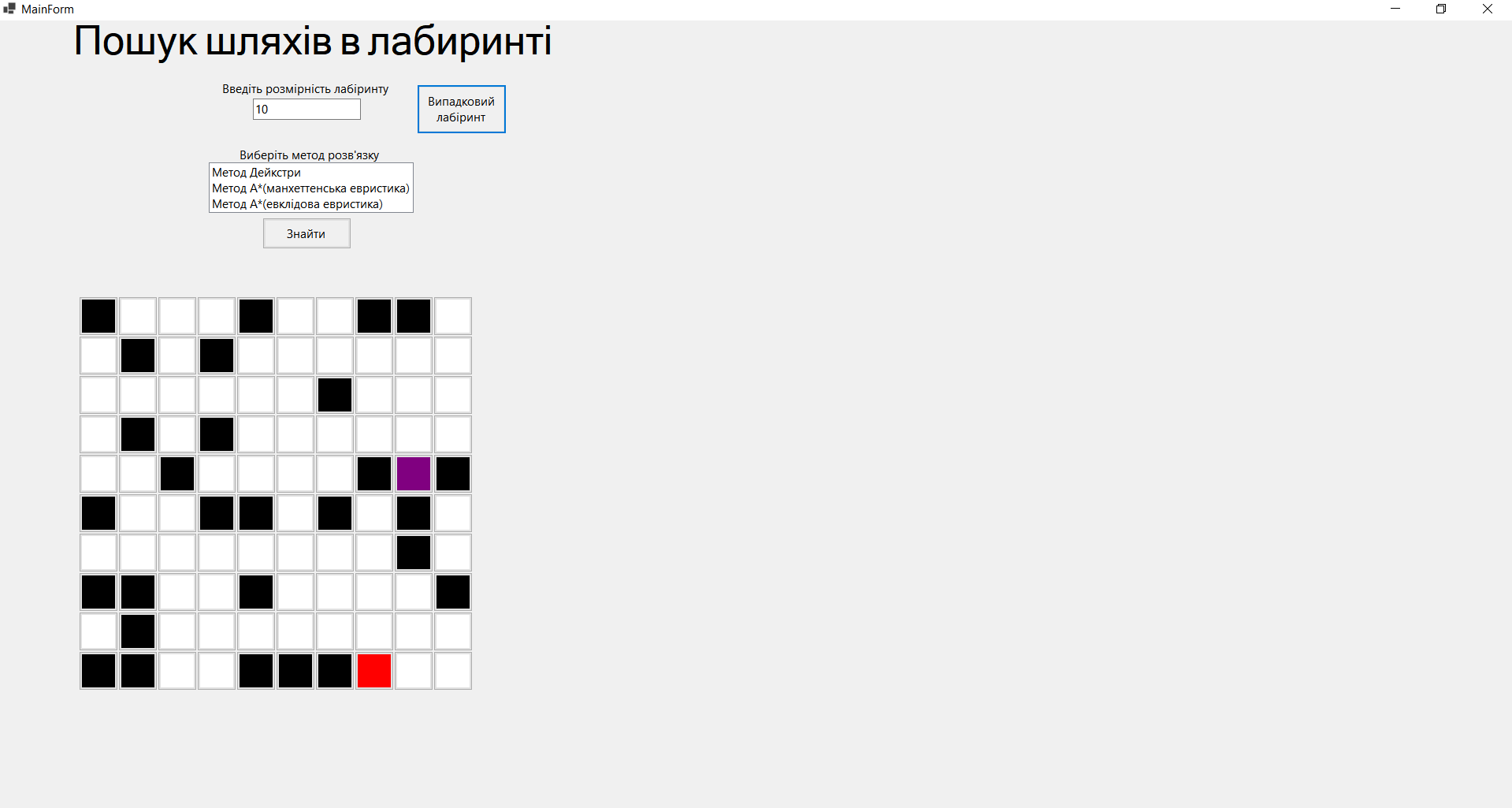


Рисунок 6.7 – Генерування випадкового лабіринту

Для знаходження шляху, спочатку треба вибрати метод. Вибираємо один із трьох доступних (Рисунок 6.8).



Рисунок 6.8 – Вибір метода Дейкстри

Далі натискаємо кнопку «Знайти». На цьому этапі можливі деякі повідомлення:

а) коли не обрано метод (Рисунок 6.9);

б) коли лабіринт має не рівно один кінець і старт (Рисунок 6.10);

в) коли лабіринт не має розв’язку (Рисунок 6.11).

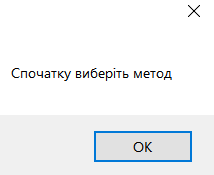


Рисунок 6.9 – Метод не обрано

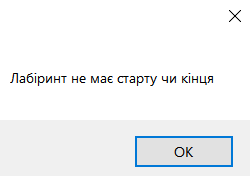


Рисунок 6.10 – Лабіринт не має рівно одного початку і одного кінця

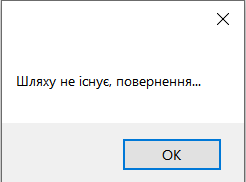


Рисунок 6.11 – Лабіринт не має розв’язку

Якщо помилок не виникло, буде відкрито друге вікно з розв’язком (Рисунок 6.12).

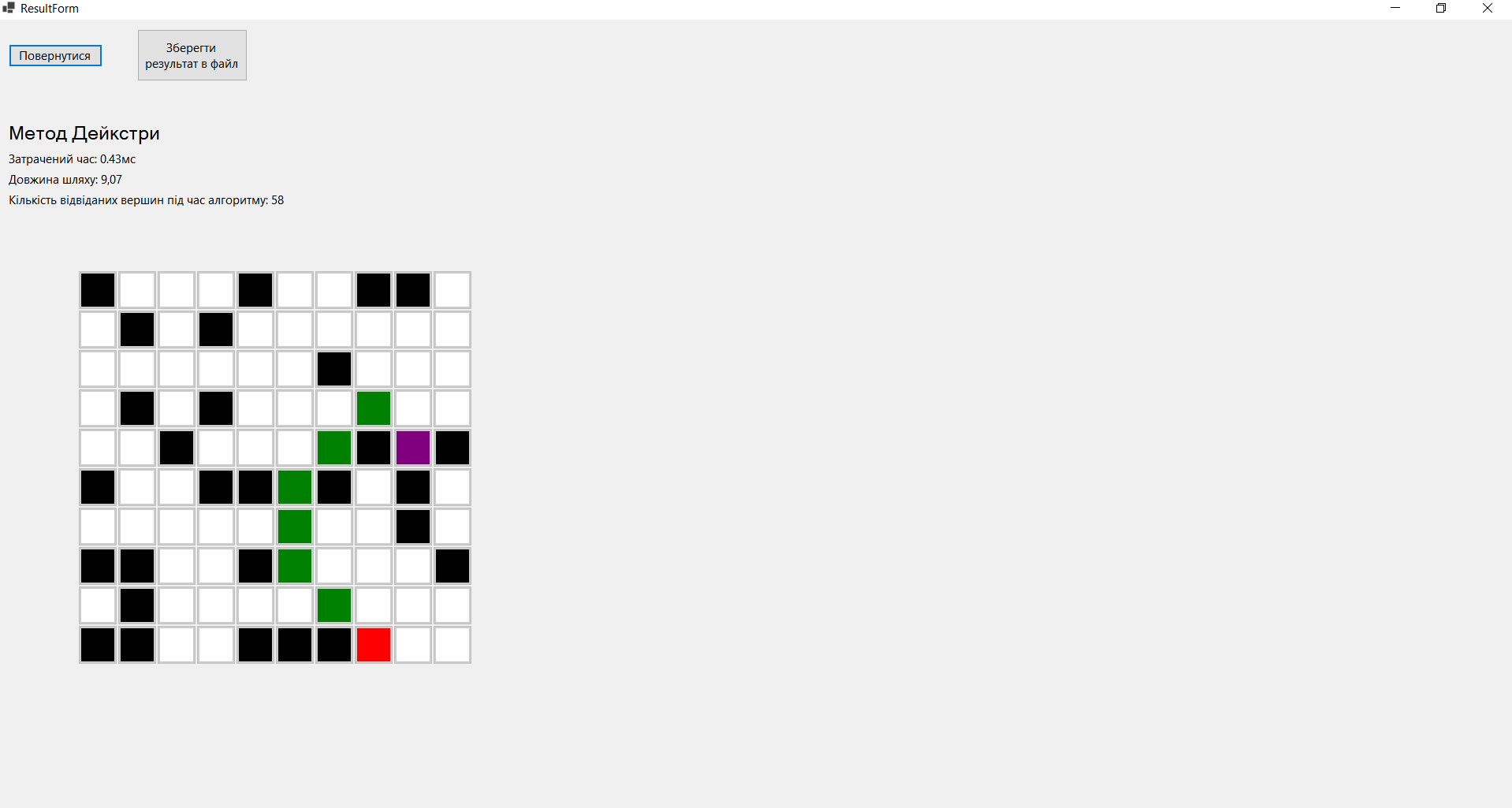


Рисунок 6.12 - Відкриття вікна з розв’язком

Тут можна побачити сам шлях в лабіринті, а також затрачений час алгоритму, довжину шляху і кількість відвіданих вершин під час алгоритму.

Кнопка «Повернутися» поверне нас на минулий этап – головне вікно. Для збереження вихідного лабіринту в файл, необхідно натиснути кнопку «Зберегти результат в файл», після чого, вибрати папку (Рисунок 6.13). У папці з’явиться файл із репрезентацією лабіринту (Рисунок 6.14).

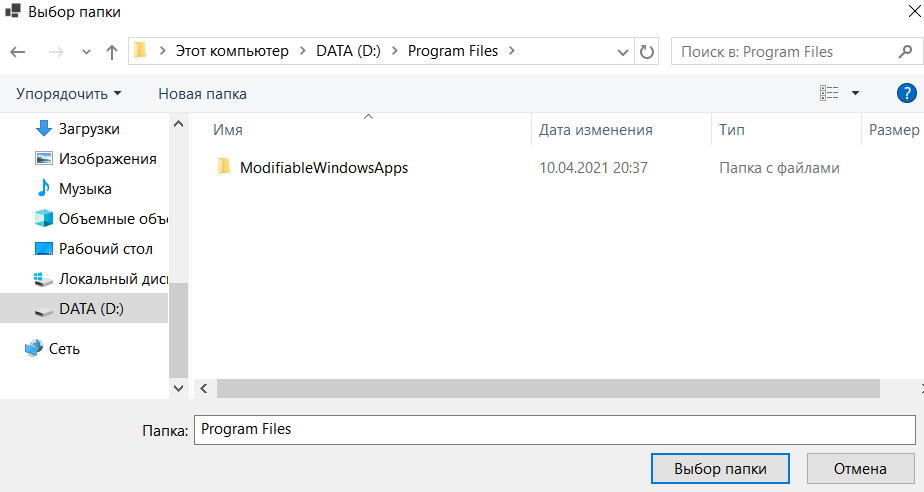


Рисунок 6.13 – Вибір папки

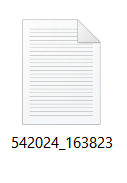


Рисунок 6.14 – Файл створено

* 1. Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги до програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows 10/11 (з останніми обновленнями) | |
| Процесор | Intel Pentium ІІІ  1.0 GHz або  AMD Athlon 1.0 GHz або аналог | Intel i5-10400F або AMD Ryzen 5 3600 або аналог |
| Оперативна пам'ять | 4 GB RAM | 16 GB RAM |
| Відеоадаптер | Nvidia GT9600 1Gb або аналог | |
| Дисплей | 800х600 | 1920x1080 або краще |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | Microsoft .Net Framework 7 або вище | |

**7 АНАЛІЗ ТА УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ**

Головною задачею курсової роботи була реалізація програмного забезпечення для пошуку найкоротшого шляху в лабіринті наступними алгоритмами: Дейкстри, А\*(Манхеттенська евристика), А\*(Евклідова евристика).

Критичні ситуації у роботі програми виявлені не були. Під час тестування було виявлено, що більшість помилок виникало, при введенні користувачем некоректного лабіринту. Тому всі варіанти вводу лабіринту, ретельно перевіряються на валідність і лише потім передаються на обробку алгоритмам.

Для перевірки та доведення достовірності результатів виконання програмного забезпечення скористаюся PathFinding від qiao [4]:

а) Алгоритм Дейкстри.

Результат виконання алгоритму Дейкстри наведено на рисунку 7.1.



Рисунок 7.1 – Результат виконання алгоритму Дейкстри

Оскільки результат виконання збігається з результатом в PathFinding(рисунок 7.2), то даний алгоритм працює правильно.

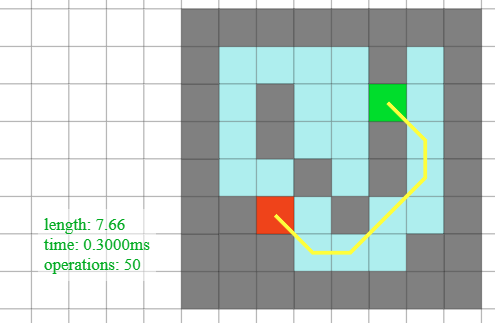


Рисунок 7.2 – Перевірка алгоритму Дейкстри в PathFinding

б) Алгоритм А\*(Манхеттенська евристика).

Результат виконання алгоритму А\*(Манхеттенська евристика) наведено на рисунку 7.3.

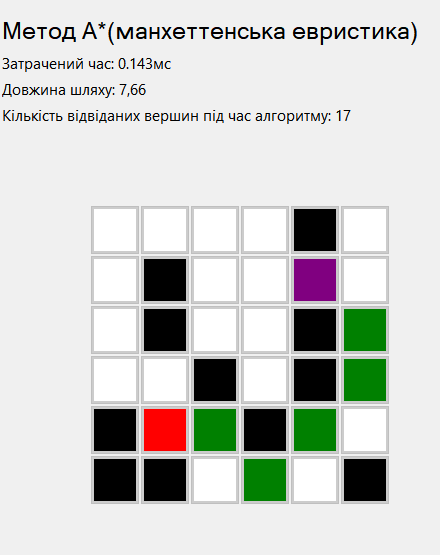


Рисунок 7.3 – Результат виконання алгоритму А\*(Манхеттенська евристика)

Оскільки результат виконання збігається з результатом в PathFinding(рисунок 7.4), то даний алгоритм також працює правильно.

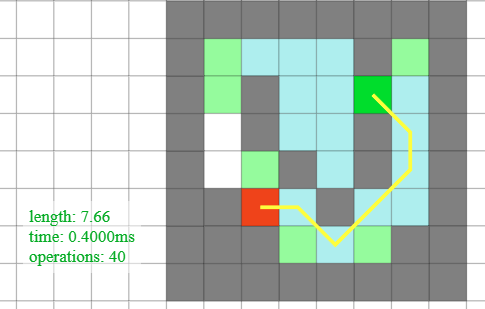


Рисунок 7.4 – Перевірка алгоритму А\*(Манхеттенська евристика) в PathFinding

в) Алгоритм А\*(Евклідова евристика).

Результат виконання алгоритму А\*(Евклідова евристика) наведено на рисунку 7.5.

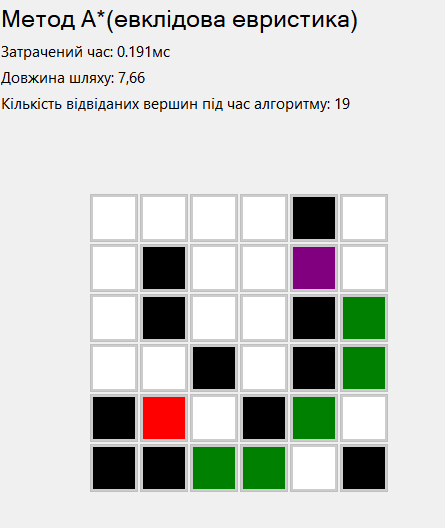


Рисунок 7.5 – Результат виконання алгоритму А\*(Евклідова евристика)

Оскільки результат виконання збігається з результатом в PathFinding(рисунок 7.6), то даний алгоритм також працює правильно.

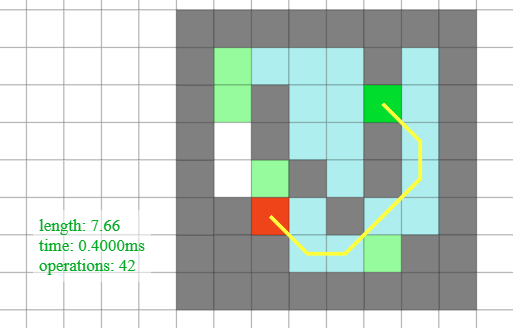


Рисунок 7.6 – Перевірка алгоритму А\*(Евклідова евристика) в PathFinding

Для проведення тестування ефективності програми було створено лабіринти наступних виглядів:

а) без стін(найкращий випадок), наприклад:

б) зі стінами перед кінцем(поганий випадок), наприклад:

Де 1 – це прохід, 0 – стіна, s – старт, e – кінець

Результати тестування ефективності алгоритмів розв’язання лабіринтів наведено в таблицях 7.1-7.2.

Таблиця 7.1 – Тестування ефективності алгоритмів для найкращого випадку

|  |  |  |  |
| --- | --- | --- | --- |
| Розмірність лабіринту | Кількість відвіданих вершин | | |
| Дейкстри | А\*(Манхеттенська евристика) | А\*(Евклідова евристика) |
| 5 | 25 | 5 | 5 |
| 10 | 100 | 10 | 10 |
| 15 | 225 | 15 | 15 |
| 20 | 400 | 20 | 20 |
| 25 | 625 | 25 | 25 |

Таблиця 7.1 – Тестування ефективності алгоритмів для поганого випадку

|  |  |  |  |
| --- | --- | --- | --- |
| Розмірність  лабіринту | Кількість відвіданих вершин | | |
| Дейкстри | А\*(Манхеттенська евристика) | А\*(Евклідова евристика) |
| 5 | 20 | 14 | 18 |
| 10 | 85 | 74 | 83 |
| 15 | 200 | 184 | 198 |
| 20 | 365 | 344 | 363 |
| 25 | 580 | 554 | 578 |

Візуалізація результатів таблиць 7.1-7.2 наведено на рисунках 7.7-7.8.

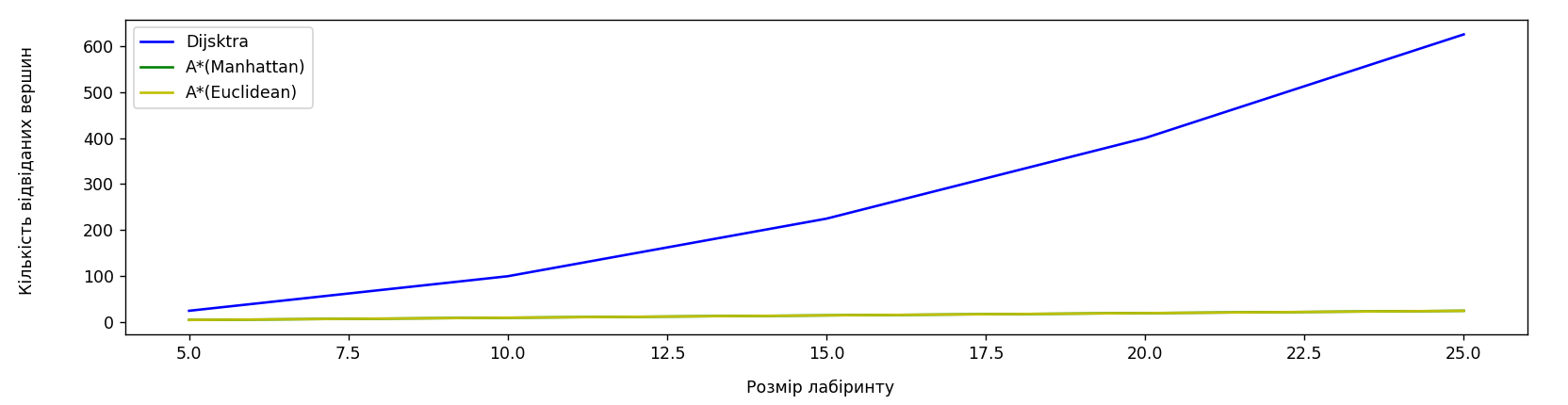


Рисунок 7.7 – Графік для найкращого випадку

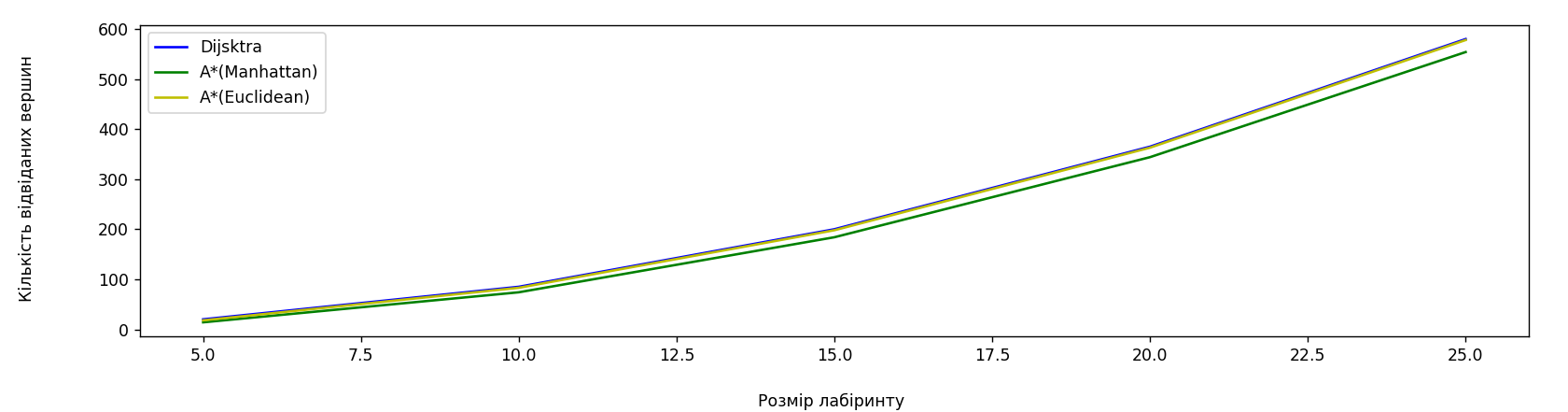


Рисунок 7.8 – Графік для поганого випадку

За результатами тестування можна зробити такі висновки:

а) всі розглянуті алгоритми дозволяють знаходити найкоротші шляхи в будь-яких лабіринтах;

б) практична складність алгоритму А\* досить сильно залежить від обраної евристики;

в) практична складність алгоритму А\* з будь-якою евристикою буде завжди не більшою за складність алгоритму Дейкстри, а в загальному випадку буде на порядок меншою.

**ВИСНОВКИ**

Виконання даної курсової роботи, вимагало від мене праці із багатьма різними етапами, щоб в результаті досягти бажаного результату й отримати необхідне програмне забезпечення.

Початковим етапом була **постановка задачі**. Його суть полягала у ретельному аналізі необхідних функціональних та нефункціональних вимог до програми та їх документування.

Після цього було проведено активний пошук **теоретичних відомостей** і сформовано відповідний розділ, в якому було описано основні принципи роботи алгоритмів, які будуть використані пізніше в програмі. Також було наведено їх орієнтовні приклади роботи.

Потім, проаналізувавши всю необхідну теорію, я зробив **опис алгоритмів.** Для детального і зрозумілого опису, було використано псевдокод.

Маючи тепер необхідну розписану інформацію, я перейшов до **опису програмного забезпечення.** Спочаткуза допомогою діаграми класів, мною було зображено необхідні сутності й відношення між ними. Після чого ж було описано стандартні та користувацькі методи, використані в програмному забезпеченні.

Важливим етапом, після виконання програмної реалізації, було **тестування**. Під час цього програмне забезпечення перевірялося різноманітними способами та при різних умовах для виявлення потенційних проблем та їх подальшого вирішення.

Після усунення всіх недосконалостей, було описано **інструкцію користувача**. В ній було детально описано працю користувача із програмою, а також зазначено системні вимоги.

Завершальним же етапом був **аналіз та узагальнення результатів**, що дозволило оцінити правильність та ефективність розробленої програми, а також порівняти її з аналогами.

Підбиваючи підсумки, хочу підкреслити, що реалізоване програмне забезпечення відзначається високою швидкістю і коректністю у вирішенні поставленої задачі. Надзвичайно стабільна і надійна робота програми була підтверджена численними тестуваннями. Її ефективність та надійність роблять її незамінним інструментом для розв’язання конкретних завдань у відповідній сфері.

**ПЕРЕЛІК ПОСИЛАНЬ**

1. Dijsktra’s algorithm. URL: [https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm%20) (дата звернення: 23.04.2024)
2. A\* algorithm. URL: [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm) (дата звернення: 23.04.2024)
3. Priority queue. URL: <https://en.wikipedia.org/wiki/Priority_queue> (дата звернення: 23.04.2024)
4. Qiao PathFinding. URL: <https://qiao.github.io/PathFinding.js/visual/> (дата звернення: 23.04.2024)

**ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ**

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

Затвердив

Керівник Вітковська І. І.

«6» березня 2024р.

Виконавець:

Студент Віжуткін Ілля Дмитрович

«6» березня 2024р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Розв’язання задач про знаходження шляху в лабіринті»

з дисципліни:

«Основи програмування»

Київ 2024

*Мета*: Метою курсової робити є забезпечення точності та надійності роботи програмного забезпечення для пошуку шляху в лабіринті за допомогою методів Дейкстри та А\*(манхеттенська та евклідова евристики)

*Дата початку роботи*: «6» березня 2024р.

*Дата закінчення роботи*: «21» травня 2024р.

*Вимоги до програмного забезпечення*.

Функціональні вимоги:

* Можливість вибирати один з методів розв’язання
* Можливість задавати розмірність лабіринту
* Можливість задавати лабіринт
* Генерація лабіринту заданої розмірності
* Можливість графічно відображати лабіринт та знайдений у ньому шлях
* Можливість встановити початкову та кінцеву точку
* Можливість отримати коректний розв’язок
* Можливість зберігати розв’язання задачі у текстовий файл
* Можливість відображати значення складності алгоритмів пошуку шляху в лабіринті
* Обробка помилок та некоректних дій користувача

Нефункціональні вимоги:

* Можливість коректної роботи програми на платформі «Windows 10»

*Стадії та етапи розробки*.

1. Об'єктно-орієнтований аналіз предметної області задачі (до 27.03.2024 р.)
2. Об'єктно-орієнтоване проектування архітектури програмної системи (до 01.04.2024 р.)
3. Розробка програмного забезпечення (до 14.04.2024 р.)
4. Тестування розробленої програми (до 17.04.2024 р.)
5. Розробка пояснювальної записки (до 21.04.2024 р.).
6. Захист курсової роботи (до 03.06.2024 р.).

*Порядок контролю та приймання*: поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

**ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

*Тексти програмного коду програмного забепечення розв’язання задач про знаходження шляху в лабіринті*

(Найменування програми (документа))

GitHub

(Вид носія даних)

*21 арк., 17.5 Кб*

(Обсяг програми (документа), арк., Кб)

Студента групи ІП-31 I курсу

Віжуткіна І. Д.

<https://github.com/IlyaViz/coursework-1>

**Файл MatrixFileSaver.cs**

namespace ClassLibrary

{

public static class MatrixFileSaver

{

public static void SaveMatrixWithAutoName<T>(string folderPath, T[,] matrix)

{

int rows = matrix.GetLength(0);

int cols = matrix.GetLength(1);

string name;

string fullPath;

DateTime date = DateTime.Now;

name = $"{date.Day}{date.Month}{date.Year}\_{date.Hour}{date.Minute}{date.Second}.txt";

fullPath = Path.Combine(folderPath, name);

using (StreamWriter writer = new StreamWriter(fullPath))

{

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

{

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

{

writer.Write(matrix[i, j]);

if (j != cols - 1)

{

writer.Write(" ");

}

}

if (i != rows - 1)

{

writer.WriteLine();

}

}

}

}

}

}

**Файл MazeSolver.cs**

using System.Diagnostics;

namespace ClassLibrary

{

public static class MazeSolver

{

public delegate double HeuristicDistance(MazeVertex first, MazeVertex second);

public static (List<MazeVertex>, double, TimeSpan, int) Solve(List<MazeVertex> vertices, MazeVertex start, MazeVertex end, HeuristicDistance heuristic)

{

Stopwatch stopwatch = new Stopwatch();

int visitedCounter = 0;

stopwatch.Start();

Dictionary<MazeVertex, MazeVertex> parentMap = new Dictionary<MazeVertex, MazeVertex>();

PriorityQueue<MazeVertex, double> priorityQueue = new PriorityQueue<MazeVertex, double>();

foreach (MazeVertex vertex in vertices)

{

vertex.Cost = double.MaxValue;

vertex.IsVisited = false;

}

start.Cost = 0;

priorityQueue.Enqueue(start, start.Cost);

MazeVertex current;

while (priorityQueue.Count > 0)

{

current = priorityQueue.Dequeue();

if (current.IsVisited)

{

continue;

}

current.IsVisited = true;

visitedCounter++;

if (current.Equals(end))

{

(List<MazeVertex>, double) pathInfo = ReconstructPath(parentMap, start, end);

stopwatch.Stop();

return (pathInfo.Item1, pathInfo.Item2, stopwatch.Elapsed, visitedCounter);

}

foreach (MazeVertex neighbour in current.Neighbours)

{

if (!neighbour.IsVisited)

{

double neighbourCost = neighbour.Cost;

double newCost = current.Cost + current.GetDistanceToNeighbour(neighbour);

if (newCost < neighbourCost)

{

neighbour.Cost = newCost;

parentMap[neighbour] = current;

priorityQueue.Enqueue(neighbour, neighbour.Cost + heuristic(neighbour, end));

}

}

}

}

throw new PathNotFoundException();

}

private static (List<MazeVertex>, double) ReconstructPath(Dictionary<MazeVertex, MazeVertex> parentMap, MazeVertex start, MazeVertex end)

{

double pathLength = 0;

List<MazeVertex> path = new List<MazeVertex>();

MazeVertex current = end;

MazeVertex next;

while (current != start)

{

path.Add(current);

next = parentMap[current];

pathLength += current.GetDistanceToNeighbour(next);

current = next;

}

path.Add(start);

path.Reverse();

return (path, pathLength);

}

public static double ManhattanDistance(MazeVertex first, MazeVertex second)

{

return Math.Abs(first.X - second.X) + Math.Abs(first.Y - second.Y);

}

public static double EuclideanDistance(MazeVertex first, MazeVertex second)

{

return Math.Sqrt(Math.Pow(first.X - second.X, 2) + Math.Pow(first.Y - second.Y, 2));

}

public static double DijsktraDistance(MazeVertex first, MazeVertex second)

{

return 0;

}

}

}

**Файл MazeVertex.cs**

using System.Collections.Generic;

namespace ClassLibrary

{

public class MazeVertex

{

public List<MazeVertex> Neighbours { get; set; } = new List<MazeVertex>();

public double Cost { get; set; }

public bool IsVisited { get; set; }

public int X { get; set; }

public int Y { get; set; }

public void AddNeighbour(MazeVertex neighbour)

{

Neighbours.Add(neighbour);

}

public double GetDistanceToNeighbour(MazeVertex neighbour)

{

if (Math.Abs(X - neighbour.X) + Math.Abs(Y - neighbour.Y) == 1)

{

return 1;

}

return Math.Sqrt(2);

}

}

}

**Файл PathNotFoundException.cs**

namespace ClassLibrary

{

public class PathNotFoundException : Exception

{

}

}

**Файл MainForm.cs**

namespace WinForms

{

public partial class MainForm : Form

{

private List<List<MazePoint>> input\_maze\_point\_matrix = new List<List<MazePoint>>();

public MainForm()

{

InitializeComponent();

}

private void MazeLengthTextBox\_TextChanged(object sender, EventArgs e)

{

int length;

string input = MazeLengthTextBox.Text;

if (int.TryParse(input, out length))

{

if (length >= MazeConstants.MIN\_LENGTH && length <= MazeConstants.MAX\_LENGTH)

{

MazeLengthTextBox.ForeColor = Color.Black;

DrawMaze(length);

RandomMazeButton.Enabled = true;

FindButton.Enabled = true;

}

else

{

ToolTip tp = new ToolTip();

tp.Show($"Мінімальне число = {MazeConstants.MIN\_LENGTH}, максимальне число = {MazeConstants.MAX\_LENGTH}", MazeLengthTextBox, 2500);

MazeLengthTextBox.ForeColor = Color.Red;

}

}

else if (input != "")

{

ToolTip tp = new ToolTip();

tp.Show("Введіть ціле число", MazeLengthTextBox, 2500);

MazeLengthTextBox.ForeColor = Color.Red;

}

}

private void DrawMaze(int length)

{

int prevLength = input\_maze\_point\_matrix.Count;

Point topElementLocation = FindButton.Location;

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

{

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

{

Controls.Remove(input\_maze\_point\_matrix[i][j]);

}

}

input\_maze\_point\_matrix.Clear();

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

{

input\_maze\_point\_matrix.Add(new List<MazePoint>());

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

{

MazePoint mazePoint = new MazePoint();

mazePoint.Location = new Point(MazeConstants.MAZE\_MARGIN\_LEFT + j \* MazeConstants.SPACE\_BETWEEN\_CELLS, topElementLocation.Y + MazeConstants.MAZE\_MARGIN\_TOP + i \* MazeConstants.SPACE\_BETWEEN\_CELLS);

input\_maze\_point\_matrix[i].Add(mazePoint);

Controls.Add(mazePoint);

}

}

}

private void RandomMazeButton\_Click(object sender, EventArgs e)

{

int length = input\_maze\_point\_matrix.Count;

Random random = new Random();

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

{

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

{

input\_maze\_point\_matrix[i][j].State = MazePointStatesEnum.PATH;

bool isWall = random.Next(0, 3) == 1;

if (isWall)

{

input\_maze\_point\_matrix[i][j].State = MazePointStatesEnum.WALL;

}

}

}

int startI, startJ, endI, endJ;

do

{

startI = random.Next(0, length);

startJ = random.Next(0, length);

endI = random.Next(0, length);

endJ = random.Next(0, length);

} while (startI == endI && startJ == endJ);

input\_maze\_point\_matrix[startI][startJ].State = MazePointStatesEnum.START;

input\_maze\_point\_matrix[endI][endJ].State = MazePointStatesEnum.END;

}

private void FindButton\_Click(object sender, EventArgs e)

{

int length = input\_maze\_point\_matrix.Count;

bool hasStart = false, hasEnd = false, hasOnlyOneStartAndEnd = true;

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

{

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

{

if (input\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.START)

{

if (hasStart) {

hasOnlyOneStartAndEnd = false;

}

hasStart = true;

}

else if (input\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.END)

{

if (hasEnd)

{

hasOnlyOneStartAndEnd = false;

}

hasEnd = true;

}

}

}

if (!hasStart || !hasEnd)

{

MessageBox.Show("Лабіринт не має старту чи кінця");

}

else if (!hasOnlyOneStartAndEnd)

{

MessageBox.Show("Лабіринт має більше ніж один старт чи один кінець");

}

else

{

MethodsEnum method;

int index = MethodListBox.SelectedIndex;

if (index != -1)

{

if (index == 0)

{

method = MethodsEnum.DIJSKTRA;

}

else if (index == 1)

{

method = MethodsEnum.A\_STAR\_MANHATTAN;

} else

{

method = MethodsEnum.A\_STAR\_EUCLIDEAN;

}

Hide();

new ResultForm(this, method, input\_maze\_point\_matrix).Show();

}

else

{

MessageBox.Show("Спочатку виберіть метод");

}

}

}

}

}

**Файл ResultForm.cs**

using System.ComponentModel;

using ClassLibrary;

namespace WinForms

{

public partial class ResultForm : Form

{

private MainForm main\_form;

private MethodsEnum method;

private List<List<MazePoint>> input\_maze\_point\_matrix;

private List<List<MazePoint>> output\_maze\_point\_matrix = new List<List<MazePoint>>();

private TimeSpan algorithm\_taken\_time;

private double path\_length;

private int visited\_vertex\_during\_algorithm;

public ResultForm(MainForm mainForm, MethodsEnum selectedMethod, List<List<MazePoint>> inputMazePointMatrix)

{

InitializeComponent();

main\_form = mainForm;

method = selectedMethod;

input\_maze\_point\_matrix = inputMazePointMatrix;

}

private void ResultForm\_Load(object sender, EventArgs e)

{

try

{

ProcessMazePath();

}

catch (PathNotFoundException)

{

MessageBox.Show("Шляху не існує, повернення...");

Close();

return;

}

DrawMaze();

UpdateTextBoxes();

}

private void UpdateTextBoxes()

{

if (method == MethodsEnum.DIJSKTRA)

{

MethodTextBox.Text = "Метод Дейкстри";

}

else if (method == MethodsEnum.A\_STAR\_MANHATTAN)

{

MethodTextBox.Text = "Метод A\*(манхеттенська евристика)";

}

else if (method == MethodsEnum.A\_STAR\_EUCLIDEAN)

{

MethodTextBox.Text = "Метод А\*(евклідова евристика)";

}

SaveResultToFileButton.Enabled = true;

TakenTimeTextBox.Text += $"{algorithm\_taken\_time.Milliseconds}.{algorithm\_taken\_time.Microseconds}мс";

PathLengthTextBox.Text += Math.Round(path\_length, 2);

VisitedCounterTextBox.Text += visited\_vertex\_during\_algorithm;

}

protected override void OnClosing(CancelEventArgs e)

{

base.OnClosing(e);

main\_form.Show();

}

private void ReturnButton\_Click(object sender, EventArgs e)

{

Close();

}

private void ProcessMazePath()

{

(List<List<MazeVertex>>, List<MazeVertex>) verticesTuple = FindVertexPath();

List<List<MazeVertex>> initVertexMatrix = verticesTuple.Item1;

List<MazeVertex> resultVertices = verticesTuple.Item2;

int length = input\_maze\_point\_matrix.Count;

Point topElementLocation = VisitedCounterTextBox.Location;

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

{

output\_maze\_point\_matrix.Add(new List<MazePoint>());

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

{

MazePoint mazePoint = new MazePoint();

mazePoint.Location = new Point(MazeConstants.MAZE\_MARGIN\_LEFT + j \* MazeConstants.SPACE\_BETWEEN\_CELLS, topElementLocation.Y + MazeConstants.MAZE\_MARGIN\_TOP + i \* MazeConstants.SPACE\_BETWEEN\_CELLS);

mazePoint.Enabled = false;

if (resultVertices.Contains(initVertexMatrix[i][j]))

{

if (input\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.START || input\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.END)

{

mazePoint.State = input\_maze\_point\_matrix[i][j].State;

}

else

{

mazePoint.State = MazePointStatesEnum.FOUND\_PATH;

}

}

else

{

mazePoint.State = input\_maze\_point\_matrix[i][j].State;

}

output\_maze\_point\_matrix[i].Add(mazePoint);

}

}

}

private void DrawMaze()

{

int length = output\_maze\_point\_matrix.Count;

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

{

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

{

Controls.Add(output\_maze\_point\_matrix[i][j]);

}

}

}

private (List<List<MazeVertex>>, List<MazeVertex>) FindVertexPath()

{

(List<List<MazeVertex>>, MazeVertex, MazeVertex) verticesTuple = GetVertices();

List<List<MazeVertex>> initVertexMatrix = verticesTuple.Item1;

MazeVertex start = verticesTuple.Item2;

MazeVertex end = verticesTuple.Item3;

List<MazeVertex> initVertices = new List<MazeVertex>();

foreach(List<MazeVertex> row in initVertexMatrix)

{

foreach(MazeVertex vertex in row)

{

initVertices.Add(vertex);

}

}

(List<MazeVertex>, double, TimeSpan, int) algorithmResult;

if (method == MethodsEnum.DIJSKTRA)

{

algorithmResult = MazeSolver.Solve(initVertices, start, end, MazeSolver.DijsktraDistance);

}

else if (method == MethodsEnum.A\_STAR\_MANHATTAN)

{

algorithmResult = MazeSolver.Solve(initVertices, start, end, MazeSolver.ManhattanDistance);

}

else

{

algorithmResult = MazeSolver.Solve(initVertices, start, end, MazeSolver.EuclideanDistance);

}

path\_length = algorithmResult.Item2;

algorithm\_taken\_time = algorithmResult.Item3;

visited\_vertex\_during\_algorithm = algorithmResult.Item4;

return (initVertexMatrix, algorithmResult.Item1);

}

private (List<List<MazeVertex>>, MazeVertex, MazeVertex) GetVertices()

{

int length = input\_maze\_point\_matrix.Count;

List<List<MazeVertex>> vertexMatrix = new List<List<MazeVertex>>();

MazeVertex start = null;

MazeVertex end = null;

MazeVertex temp;

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

{

vertexMatrix.Add(new List<MazeVertex>());

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

{

temp = new MazeVertex();

temp.X = j;

temp.Y = i;

vertexMatrix[i].Add(temp);

}

}

(int, int)[] verticalAndHorizontalCoordinates =

{

(0, 1),

(0, -1),

(1, 0),

(-1, 0),

};

(int, int)[] diagonalCoordinates =

{

(1, 1),

(1, -1),

(-1, -1),

(-1, 1)

};

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

{

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

{

if (input\_maze\_point\_matrix[i][j].State != MazePointStatesEnum.WALL)

{

if (input\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.START)

{

start = vertexMatrix[i][j];

}

else if (input\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.END)

{

end = vertexMatrix[i][j];

}

foreach ((int, int) path in verticalAndHorizontalCoordinates)

{

int neighbourI = i + path.Item1;

int neighbourJ = j + path.Item2;

if (neighbourI >= 0 && neighbourI < length && neighbourJ >= 0 && neighbourJ < length)

{

if (input\_maze\_point\_matrix[neighbourI][neighbourJ].State != MazePointStatesEnum.WALL)

{

vertexMatrix[i][j].AddNeighbour(vertexMatrix[neighbourI][neighbourJ]);

}

}

}

foreach ((int, int) path in diagonalCoordinates)

{

int neighbourI = i + path.Item1;

int neighbourJ = j + path.Item2;

if (neighbourI >= 0 && neighbourI < length && neighbourJ >= 0 && neighbourJ < length)

{

if (input\_maze\_point\_matrix[neighbourI][neighbourJ].State != MazePointStatesEnum.WALL)

{

if (input\_maze\_point\_matrix[neighbourI][j].State != MazePointStatesEnum.WALL || input\_maze\_point\_matrix[i][neighbourJ].State != MazePointStatesEnum.WALL)

{

vertexMatrix[i][j].AddNeighbour(vertexMatrix[neighbourI][neighbourJ]);

}

}

}

}

}

}

}

return (vertexMatrix, start, end);

}

private void SaveResultToFileButton\_Click(object sender, EventArgs e)

{

FolderBrowserDialog dialog = new FolderBrowserDialog();

DialogResult result = dialog.ShowDialog();

if (result == DialogResult.OK)

{

int length = output\_maze\_point\_matrix.Count;

char[,] charMatrix = new char[length, length];

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

{

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

{

if (output\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.START)

{

charMatrix[i, j] = 's';

}

else if (output\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.END)

{

charMatrix[i, j] = 'e';

}

else if (output\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.FOUND\_PATH)

{

charMatrix[i, j] = '\*';

}

else if (output\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.PATH)

{

charMatrix[i, j] = '-';

}

else if (output\_maze\_point\_matrix[i][j].State == MazePointStatesEnum.WALL)

{

charMatrix[i, j] = '#';

}

}

}

try

{

MatrixFileSaver.SaveMatrixWithAutoName(dialog.SelectedPath, charMatrix);

}

catch (UnauthorizedAccessException)

{

MessageBox.Show("Помилка... Спробуйте інший диск");

}

}

else

{

MessageBox.Show("Помилка... Спробуйте іншу папку");

}

}

}

}

**Файл MazeConstants.cs**

namespace WinForms

{

public static class MazeConstants

{

public const int CELL\_WIDTH = 50;

public const int CELL\_HEIGHT = 50;

public const int MAZE\_MARGIN\_TOP = 100;

public const int MAZE\_MARGIN\_LEFT = 100;

public const int SPACE\_BETWEEN\_CELLS = 50;

public const int MIN\_LENGTH = 2;

public const int MAX\_LENGTH = 25;

}

}

**Файл MazePoint.cs**

namespace WinForms

{

public partial class MazePoint : Button

{

private MazePointStatesEnum state;

private Dictionary<MazePointStatesEnum, Color> color\_dict = new Dictionary<MazePointStatesEnum, Color>()

{

{MazePointStatesEnum.PATH, Color.White},

{MazePointStatesEnum.WALL, Color.Black},

{MazePointStatesEnum.START, Color.Purple},

{MazePointStatesEnum.END, Color.Red},

{MazePointStatesEnum.FOUND\_PATH, Color.Green},

};

public MazePointStatesEnum State {

get { return state; }

set

{

state = value;

ChangeColor();

}

}

public MazePoint()

{

InitializeComponent();

Width = MazeConstants.CELL\_WIDTH;

Height = MazeConstants.CELL\_HEIGHT;

State = MazePointStatesEnum.PATH;

Click += MazePoint\_OnClick;

}

private void MazePoint\_OnClick(object sender, EventArgs e)

{

switch (State)

{

case MazePointStatesEnum.PATH:

State = MazePointStatesEnum.WALL;

break;

case MazePointStatesEnum.WALL:

State = MazePointStatesEnum.START;

break;

case MazePointStatesEnum.START:

State = MazePointStatesEnum.END;

break;

case MazePointStatesEnum.END:

State = MazePointStatesEnum.PATH;

break;

}

}

private void ChangeColor()

{

BackColor = color\_dict[State];

}

}

}

**Файл MazePointStatesEnum.cs**

namespace WinForms

{

public enum MazePointStatesEnum

{

WALL = 0,

PATH = 1,

START = 2,

END = 3,

FOUND\_PATH = 4,

}

}

**Файл MethodsEnum.cs**

namespace WinForms

{

public enum MethodsEnum

{

DIJSKTRA,

A\_STAR\_MANHATTAN,

A\_STAR\_EUCLIDEAN

}

}