МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра  
інформатики та програмної інженерії  
(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**з «Основи програмування. Курсова робота»  
(назва дисципліни)  
на тему: 8-puzzle

Студента 1 курсу, групи ІП-44  
Радченко Микола Сергійович  
Спеціальності 121 «Інженерія програмного  
забезпечення»

Керівник  
Куценко М.О.  
(посада, вчене звання, науковий  
ступінь, прізвище та ініціали)

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії Головченко М.М.   
(підпис) (посада, вчене звання, науковий ступінь,  
прізвище та ініціали)  
 Куценко М.О.

(підпис) (посада, вчене звання, науковий ступінь,  
прізвище та ініціали)

Київ – 2025 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування. Курсова робота

Напрям «ІПЗ»

Курс 1 Група ІП-44 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Радченко Микола Сергійович

(прізвище, ім’я, по батькові)

1. Тема роботи Гра 8-puzzle

2. Строк здачі студентом закінченої роботи 25 травня 2025 року

3. Вихідні дані до роботи додаток А Технічне завдання, додаток Б Текст програмного

коду

4. Зміст пояснювальної записки (перелік питань, які підлягають розробці)

Вступ, постановка задачі, теоретичні відомості, опис алгоритмів, опис програмного

забезпечення, результати тестування програмного забезпечення, інструкція користувача,

аналіз і узагальнення результатів, висновок

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

6. Дата видачі завдання 05.02.2025 р.

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін  виконання етапів роботи | Підписи  керівника,  студента |
| 1 | Отримання теми курсової роботи | 05.02.2025 |  |
| 2 | Підготовка ТЗ | 05.03.2025 |  |
| 3 | Пошук та вивчення літератури з питань курсової роботи | 15.03.2025 |  |
| 4 | Розробка сценарію роботи програми | 25.03.2025 |  |
| 5 | Узгодження сценарію роботи програми з керівником | 09.04.2025 |  |
| 6 | Узгодження алгоритму з керівником | 10.04.2025 |  |
| 7 | Узгодження з керівником інтерфейсу користувача | 16.04.2025 |  |
| 8 | Розробка програмного забезпечення | 30.04.2025 |  |
| 9 | Налагодження розрахункової частини програми | 01.05.2025 |  |
| 10 | Розробка та налагодження інтерфейсної частини програми | 05.05.2025 |  |
| 11 | Узгодження з керівником набору тестів для контрольного прикладу | 07.05.2025 |  |
| 12 | Тестування програми | 10.05.2025 |  |
| 13 | Підготовка пояснювальної записки | 20.05.2025 |  |
| 14 | Здача курсової роботи на перевірку | 25.05.2025 |  |
| 15 | Захист курсової роботи | 29.05.2025 |  |

Студент

(підпис)

Керівник Куценко М.О.

(підпис) (прізвище, ім’я, по батькові)

« » 20 р.

**АНОТАЦІЯ**

Пояснювальна записка до курсової роботи: 73 сторінок, 11 рисунків, 12 таблиць та 12 посилань.

Курсова робота присвячена розробці програмної реалізації класичної логічної головоломки «8-puzzle», яка використовується для вивчення алгоритмів пошуку та методів штучного інтелекту. Мета роботи - створення додатку, що дозволяє розв’язувати головоломку вручну та автоматично за допомогою інтелектуальних алгоритмів.

У рамках проєкту реалізовано графічний інтерфейс користувача з використанням технології WPF та мови програмування C#. Програма підтримує візуалізацію ігрового процесу, переміщення плиток, а також демонстрацію автоматичного пошуку рішення.

Основну увагу приділено реалізації алгоритмів A\* та RBFS. Алгоритм A\* використовує поєднання фактичної та евристичної вартості для пошуку найкоротшого шляху, тоді як RBFS застосовує рекурсивну стратегію з обмеженням пам’яті. У роботі порівнюються ці підходи, розглядаються їх переваги та недоліки у контексті задачі.

У результаті створено функціональну систему, яка дозволяє ефективно знаходити розв’язок до заданого стану головоломки, а також наочно демонструє процес пошуку для користувача.

ГРА 8-PUZZLE, А\* АЛГОРИТМ, RBFS АЛГОРИТМ, АВТОМАТИЧНИЙ РОЗВ’ЯЗОК, ПОРІВНЯННЯ СТАТИСТИКИ

Зміст

[ВСТУП 6](#_Toc199246261)

[1 ПОСТАНОВКА ЗАДАЧІ 7](#_Toc199246262)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ 8](#_Toc199246263)

[3 ОПИС АЛГОРИТМІВ 10](#_Toc199246264)

[3.1 Переміщення плиток 13](#_Toc199246265)

[3.2 Автоматичне розмішування 13](#_Toc199246266)

[3.3 Автоматичне розв’язання 13](#_Toc199246267)

[3.4 Утворення нового стану 15](#_Toc199246268)

[3.5 Перевірка на повторення 15](#_Toc199246269)

[4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 16](#_Toc199246270)

[4.1 Діаграма класів програмного забезпечення 16](#_Toc199246271)

[4.2 Опис методів частин програмного забезпечення 17](#_Toc199246272)

[5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 21](#_Toc199246273)

[5.1 ПЛАН ТЕСТУВАННЯ 21](#_Toc199246274)

[5.2 Приклади тестування 22](#_Toc199246275)

[6 ІНСТРУКЦІЯ КОРИСТУВАЧА 26](#_Toc199246276)

[6.1 Робота з програмою 26](#_Toc199246277)

[6.2 Системні вимоги 31](#_Toc199246278)

[ВИСНОВКИ 32](#_Toc199246279)

[ПЕРЕЛІК ПОСИЛАНЬ 33](#_Toc199246280)

[ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ 34](#_Toc199246281)

[ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ 37](#_Toc199246282)

# ВСТУП

Сучасний світ неможливо уявити без інформаційних технологій та інтелектуальних систем. Штучний інтелект активно проникає в усі сфери життя - від автоматизації повсякденних задач до складних аналітичних систем. Одним із важливих напрямів є розробка алгоритмів пошуку, які дозволяють ефективно знаходити рішення у великому просторі можливих станів.

Дана курсова робота присвячена розв’язанню задачі «8-puzzle» - класичної логічної головоломки, яка широко використовується для моделювання та дослідження різних підходів до пошуку рішень. Під час розробки програмного забезпечення активно використовувались принципи об’єктно-орієнтованого програмування, що дозволяє забезпечити зрозумілу структуру, гнучкість та масштабованість коду.

Метою роботи є створення програмної реалізації задачі «8-puzzle» з використанням алгоритмів пошуку, зокрема: алгоритмів RBFS та A\*, з подальшим аналізом їх ефективності щодо часу виконання, кількості відвіданих станів і глибини рішення.

Вхідними даними є початкова конфігурація головоломки у вигляді масиву випадково розмішаних значень від 1 до 8 та 0 як порожньою клітинкою без повторень. Вихідними даними є статистика вирішення двома алгоритмами та можливість їх порівняння.

# 1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмну реалізацію головоломки «8-puzzle» та виконати наступні пункти:

а) Реалізувати графічний інтерфейс користувача;

б) Забезпечити коректне переміщення плиток по полю;

в) Надати можливість автоматичного і ручного розв’язку;

г) Забезпечити перегляд статистики автоматичного рішення головоломки у зручному вигляді;

д) Забезпечити логування кожного розв’язку для повноцінної статистики.

Таким чином програмний продукт має представляти собою реалізацію як повного функціоналу фізичної версії гри так і специфічного для програмного забезпечення.

# 2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Реалізовано програму було на мові програмування C#.

Гра «8-puzzle» - це класична головоломка, що складається з поля 3×3. На ньому розміщуються 8 плиток з номерами від 1 до 8 та одна пуста клітинка. Ходи здійснюються шляхом переміщення плиток, які безпосередньо межують із порожньою клітинкою, на її місце.

Початковий стан - це довільна конфігурація плиток на полі. Проте в межах даної роботи початкові конфігурації генеруються шляхом послідовних випадкових допустимих ходів, що гарантує розв’язність головоломки. Це пов’язано з тим, що лише половина всіх можливих розміщень у «8-puzzle» мають розв’язок, інші - ні.

Метою гри є досягнення цільової конфігурації, у якій плитки розташовані в порядку зростання зліва направо і зверху вниз, а порожня клітинка знаходиться у правому нижньому куті поля.

Для ефективного автоматичного розв’язання головоломки використовується алгоритм A\*, який знаходить найкоротший (оптимальний) шлях до розв’язку та RBFS, який рекурсивно викликається на кожному стані для перерахунку найкращого наступного стану без запам’ятовування всіх інших можливих станів. Реалізація алгоритму А\* базується на наступних ключових компонентах:

а) Пріоритетна черга - використовується для зберігання черги станів з урахуванням їх вартості та евристичної оцінки.

б) Алгоритм оцінки евристики - визначає наближеність поточного стану до цільового.

в) Хеш-набір або словник - зберігає вже відвідані стани та їхню вартість для запобігання повторній обробці.

В той час як алгоритм RBFS працює на основі рекурсії (повторному виклику функції доти, поки поточний стан не буде збігатись з цільовим), оцінки евристики та запам’ятовуванні лише попереднього стану і обрахунку наступного найвигіднішого під час виклику функції.

Як евристична функція використовується манхетенська відстань - сума відстаней по горизонталі та вертикалі від кожної плитки до її цільової позиції. Ця евристика є допустимою, тобто вона ніколи не переоцінює вартість до розв’язку, що дозволяє алгоритмам A\* та RBFS знаходити оптимальний шлях.

У поєднанні з алгоритмами A\* та RBFS манхетенська евристика дозволяє ефективно та швидко розв’язувати задачу, зменшуючи кількість зайвих обчислень і скорочуючи загальний час пошуку.

# 3 ОПИС АЛГОРИТМІВ\

Таблиця 3.1 – Основні змінні та їх призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
| buttons | Масив кнопок для відображення плиток |
| tiles | Значення плиток на поточному стані дошки |
| state | Поточний стан дошки |
| goalState | Фінальна конфігурація |
| button | Натиснута кнопка |
| row, col | Позиція кнопки |
| emptyRow, emptyCol | Позиція пустої плитки |
| chosenMethod | Назва обраного алгоритму |
| initialState | Початкова конфігурація плиток |
| copy | Копія стану дошки |
| heuristic | Об’єкт евристики |
| primary, secondary | Алгоритми для відображення статистики |
| statsPrimary, statsSecondary | Статистика двох алгоритмів |
| log | Об’єкт логування |
| list | Список для генерації довільної конфігурації |
| newState | Новий стан дошки |
| inv | Кількість інверсій при генерації довільної конфігурації |
| logInfo | Дані про лог файл |
| filePath | Повний шлях до файлу логу |

### 3.1 Основний алгоритм гри «8-puzzle»:

1. ПОЧАТОК

1.1. Ініціалізувати головне вікно MainWindow()

1.2. Встановити цільовий стан goalState = {{1,2,3},{4,5,6},{7,8,0}}

1.3. Створити початковий стан tiles = goalState

1.4. Викликати InitButtons() для створення ігрового поля

1.5. Викликати UpdateButtons() для відображення початкового стану

2. Ініціалізація ігрових компонентів

2.1. Створити сітку кнопок 3x3 в GameGrid

2.2. Призначити обробник Tile\_Click для кожної кнопки

2.3. Створити елементи управління: Shuffle, Solve, Choice ComboBox

2.4. Створити кнопки для роботи з логами: ClearLogs, OpenLogs

3. Основний цикл додатку

3.1. Запуск інтерфейсу WPF Application

3.2. Обробка подій користувача:

3.2.1. ЯКЩО натиснуто Tile\_Click, ТО:

3.2.1.1. Отримати координати натиснутої кнопки (row, col)

3.2.1.2. Знайти позицію пустої плитки (emptyRow, emptyCol)

3.2.1.3. ЯКЩО плитка сусідня з пустою, ТО:

- Поміняти місцями плитки

- Викликати UpdateButtons()

3.2.2. ЯКЩО натиснуто Shuffle\_Click, ТО:

3.2.2.1. Викликати GenerateShuffledState()

3.2.2.2. Перевірити розв'язність через IsSolvable()

3.2.2.3. Оновити tiles новим станом

3.2.2.4. Викликати UpdateButtons()

3.2.3. ЯКЩО натиснуто Solve\_Click, ТО:

3.2.3.1. Показати повідомлення "Solver started"

3.2.3.2. Отримати обраний метод з Choice ComboBox

3.2.3.3. Створити initialState з поточних tiles

3.2.3.4. Створити об'єкти Statistics для обох алгоритмів

3.2.3.5. Запустити асинхронне розв'язання:

ЯКЩО chosenMethod == "A\*", ТО:

primary = AStar.Solve()

secondary = RBFS.Solve()

ЯКЩО chosenMethod == "RBFS", ТО:

primary = RBFS.Solve()

secondary = AStar.Solve()

3.2.3.6. Викликати ShowStats() для відображення результатів

3.2.3.7. Створити log запис та зберегти через Logs.LogSolution()

3.2.4. ЯКЩО натиснуто ClearLogs\_Click, ТО:

3.2.4.1. Перевірити існування файлу logs.json

3.2.4.2. ЯКЩО файл існує, ТО видалити його

3.2.4.3. Показати відповідне повідомлення

3.2.5. ЯКЩО натиснуто OpenLogs\_Click, ТО:

3.2.5.1. Перевірити існування файлу logs.json

3.2.5.2. ЯКЩО файл існує, ТО відкрити через Process.Start()

3.2.5.3. ІНАКШЕ показати повідомлення про відсутність файлу

4. Алгоритми розв'язання

4.1. A\* алгоритм (AStar.Solve()):

4.1.1. ЯКЩО initialState.IsGoalState(), ТО повернути порожній список

4.1.2. Ініціалізувати PriorityQueue та Dictionary для відвіданих станів

4.1.3. Додати початковий вузол до черги

4.1.4. ПОКИ черга не пуста:

- Витягти вузол з найменшим F

- ЯКЩО досягнуто цільового стану, ТО повернути шлях

- Згенерувати всі можливі ходи

- Додати нові стани до черги

4.1.5. Зібрати статистику та повернути результат

4.2. RBFS алгоритм (RBFS.Solve()):

4.2.1. Створити початковий вузол

4.2.2. Викликати рекурсивну функцію Solution()

4.2.3. ЯКЩО знайдено розв'язок, ТО реконструювати шлях

4.2.4. Зібрати статистику та повернути результат

5. Збереження логів

5.1. Створити об'єкт Logs з результатами

5.2. Прочитати існуючі логи з файлу (ЯКЩО існує)

5.3. Додати новий запис до списку

5.4. Серіалізувати у JSON та записати у файл

6. КІНЕЦЬ

6.1. Закрити додаток при завершенні роботи

6.2. Очистити ресурси

### 3.2 Переміщення плиток

* 1. ПОЧАТОК
  2. Отримуємо координати вибраної плитки.
  3. Перевіряємо, чи є вона суміжною з порожньою клітинкою.
  4. Якщо так - обмінюємо місцями з порожньою.
  5. Інакше - ігноруємо дію.
  6. КІНЕЦЬ

### 3.3 Автоматичне розмішування

1. ПОЧАТОК
2. Згенерувати випадковий напрямок руху.
3. Перевірити можливість переміщення порожньої клітинки у цьому напрямку.
4. Якщо можливо - виконати рух, інакше згенерувати інший напрямок.
5. Не повторювати попередній хід у зворотному напрямку.
6. КІНЕЦЬ

### 3.4 Автоматичне розв’язання

#### Алгоритм автоматичного розв’язання (A\*)

1. ПОЧАТОК
2. Ініціалізувати відкриту множину OpenList стартовим станом.
3. Ініціалізувати закриту множину ClosedList як порожню.
4. Поки OpenList не порожня, виконати:
   1. Вибрати стан з найменшим f.
   2. Якщо це цільовий стан - завершити**,** повернутишлях.
   3. Перемістити поточний стан з OpenList до ClosedList.
   4. Генерувати всі допустимі наступні стани (сусіди).
   5. Для кожного сусіда:
      1. Якщо сусід уже в ClosedList - пропустити.
      2. Обчислити g та h
      3. Якщо сусід не в OpenList - додати.
      4. Інакше, якщо нове g краще - оновити шлях.
5. КІНЕЦЬ (якщо OpenList порожній - рішення не знайдено)

#### Алгоритм автоматичного розв’язання (RBFS)

1. ПОЧАТОК
2. Викликати RBFS зі стартовим станом та початковою межою
3. У функції RBFS:
   1. Якщо поточний стан - цільовий, повернутишлях.
   2. Генерувати всі допустимі наступні стани (сусіди).
   3. Для кожного сусіда обчислити:
      1. g(n): вартість шляху до сусіда
      2. h(n): евристична оцінка
      3. f(n) = max(g(n) + h(n), f(parent))
      4. Поки існують стани для розгляду:
         1. Вибрати стан з найменшим f(n).
         2. Якщо f(n) > f\_limit - повернутиf(n) (обмеження перевищено).
         3. Визначити новий f\_limit як мінімум з f інших сусідів.
         4. Рекурсивно викликати RBFS для вибраного сусіда з f\_limit
         5. Якщо рішення знайдено - повернути рішення.
4. Якщо не знайдено рішень - повернути невдачу.
5. КІНЕЦЬ

### 3.5 Утворення нового стану

1. **ПОЧАТОК**
2. Скопіювати матрицю з батьківського стану.
3. Поміняти місцями нуль і сусідню плитку згідно з напрямком руху.
4. Обчислити евристику:
   1. Манхеттенська відстань - сума абсолютних різниць координат кожної плитки від цільового положення.
5. **КІНЕЦЬ**

### 3.6 Перевірка на повторення

1. **ПОЧАТОК**
2. Для кожного елемента стану перевірити наявність в дереві.
   1. Якщо відсутній - додати новий вузол.
   2. Перейти до наступного рівня.
3. Якщо цей шлях ще не досягався, або новий шлях коротший - оновити інформацію про кроки.
4. Стан вважається унікальним і додається до черги.
5. **КІНЕЦЬ**

# 4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 4.1 Діаграма класів програмного забезпечення

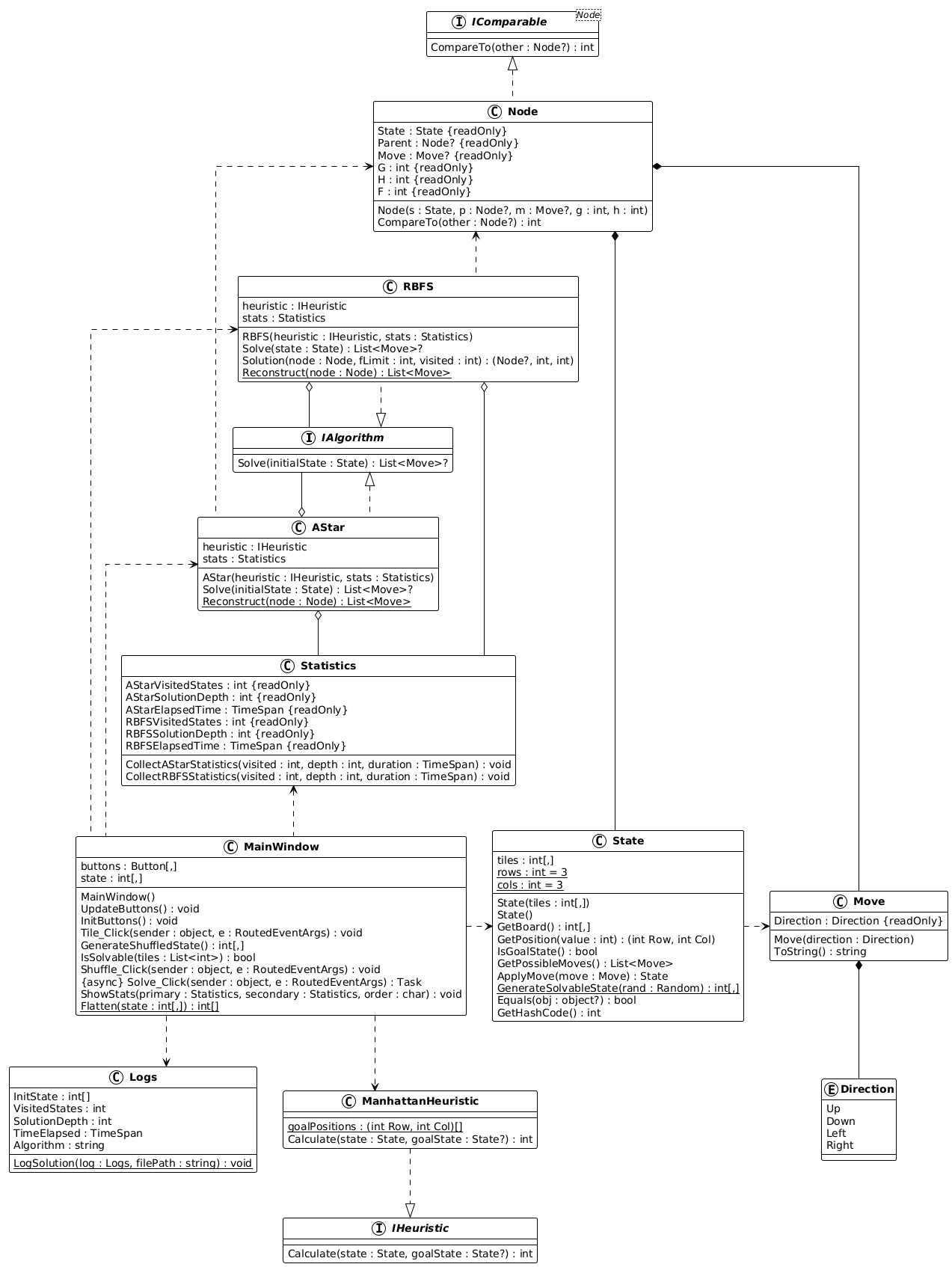


Рисунок 1.1 – Діаграма класів

### 4.2 Опис методів частин програмного забезпечення

#### 4.2.1 Стандартні методи

У таблиці 4.1 наведено стандартні методи, реалізація яких наявна в стандартних та зовнішніх бібліотеках .NET та не є власною реалізацією.

**Таблиця 4.1 – Стандартні методи**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва класу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** | **Заголовний файл** |
| 1 | PriorityQueue | Enqueue() | Додавання елементу в чергу з пріоритетом | елемент, пріоритет | - | System.Collections.Generic |
| 2 | PriorityQueue | Dequeue() | Вилучення елементу з найвищим пріоритетом | - | елемент | System.Collections.Generic |
| 3 | Dictionary | TryGetValue() | Спроба отримання значення за ключем | ключ, out значення | bool | System.Collections.Generic |
| 4 | List | Add() | Додавання елементу до списку | елемент | - | System.Collections.Generic |
| 5 | List | Reverse() | Реверс порядку елементів у списку | - | - | System.Collections.Generic |
| 6 | DateTime | Now | Отримання поточного часу | - | DateTime | System |
| 7 | Math | Abs() | Обчислення абсолютного значення | число | абсолютне значення | System |
| 8 | Math | Min() | Знаходження мінімального значення | значення1, значення2 | мінімальне значення | System |
| 9 | Math | Max() | Знаходження максимального значення | значення1, значення2 | максимальне значення | System |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | Enumerable | Range() | Створення послідовності чисел | початок, кількість | IEnumerable<int> | System.Linq |
| 11 | Enumerable | OrderBy() | Сортування за критерієм | selector | відсортована послідовність | System.Linq |
| 12 | Enumerable | ToArray() | Конвертація в масив | - | масив | System.Linq |
| 13 | Enumerable | ToList() | Конвертація в список | - | список | System.Linq |
| 14 | Enumerable | Select() | Проекція елементів | selector | трансформована послідовність | System.Linq |
| 15 | JsonSerializer | Serialize() | Серіалізація об'єкта в JSON | об'єкт | JSON рядок | System.Text.Json |
| 16 | JsonSerializer | Deserialize() | Десеріалізація JSON в об'єкт | JSON рядок | об'єкт | System.Text.Json |
| 17 | File | Exists() | Перевірка існування файлу | шлях до файлу | bool | System.IO |
| 18 | File | ReadAllText() | Читання всього тексту з файлу | шлях до файлу | текст | System.IO |
| 19 | File | WriteAllText() | Запис тексту у файл | шлях, текст | - | System.IO |
| 20 | Random | Next() | Генерація випадкового числа | - або межі | випадкове число | System |

Продовження таблиці 4.1

#### 4.2.2 Користувацькі методи

У таблиці 4.2 наведено користувацькі методи створені та використані під час написання програмного забезпечення.

**Таблиця 4.2 – Користувацькі методи**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва класу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** | **Заголовний файл** |
| 1 | AStar | Solve() | Розв'язання головоломки алгоритмом A\* | початковий стан | список ходів або null | AStar.cs |
| 2 | AStar | Reconstruct() | Відновлення шляху розв'язання | кінцевий вузол | список ходів | AStar.cs |
| 3 | RBFS | Solve() | Розв'язання головоломки алгоритмом RBFS | початковий стан | список ходів або null | RBFS.cs |
| 4 | RBFS | Solution() | Рекурсивний пошук з обмеженою пам'яттю | вузол, обмеження, відвідано | вузол, вартість, кількість відвіданих | RBFS.cs |
| 5 | RBFS | Reconstruct() | Відновлення шляху розв'язання | кінцевий вузол | список ходів | RBFS.cs |
| 6 | ManhattanHeuristic | Calculate() | Обчислення манхеттенської відстані | стан, цільовий стан | евристична оцінка | ManhattanHeuristic.cs |
| 7 | State | IsGoalState() | Перевірка чи є стан цільовим | - | bool | State.cs |
| 8 | State | GetPossibleMoves() | Отримання можливих ходів | - | список можливих ходів | State.cs |
| 9 | State | ApplyMove() | Застосування ходу до стану | хід | новий стан | State.cs |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | State | GetPosition() | Знаходження позиції значення | значення | координати (рядок, стовпець) | State.cs |
| 11 | State | GetBoard() | Отримання копії дошки | - | двовимірний масив | State.cs |
| 12 | State | GenerateSolvableState() | Генерація розв'язуваного стану | генератор випадкових чисел | двовимірний масив | State.cs |
| 13 | State | IsSolvable() | Перевірка розв'язуваності стану | одновимірний масив | bool | State.cs |
| 14 | Node | CompareTo() | Порівняння вузлів для сортування | інший вузол | результат порівняння | Node.cs |
| 15 | Statistics | CollectAStarStatistics() | Збирання статистики A\* | відвідано, глибина, час | - | Statistics.cs |
| 16 | Statistics | CollectRBFSStatistics() | Збирання статистики RBFS | відвідано, глибина, час | - | Statistics.cs |
| 17 | Logs | LogSolution() | Логування результату розв'язання | лог, шлях до файлу | - | Logs.cs |
| 18 | MainWindow | UpdateButtons() | Оновлення візуального стану кнопок | - | - | MainWindow.xaml.cs |
| 19 | MainWindow | InitButtons() | Ініціалізація кнопок інтерфейсу | - | - | MainWindow.xaml.cs |
| 20 | MainWindow | GenerateShuffledState() | Генерація перемішаного стану | - | двовимірний масив | MainWindow.xaml.cs |
| 21 | MainWindow | IsSolvable() | Перевірка розв'язуваності стану | список плиток | bool | MainWindow.xaml.cs |
| 22 | MainWindow | ShowStats() | Відображення статистики алгоритмів | первинна статистика, вторинна статистика, порядок | - | MainWindow.xaml.cs |
| 23 | MainWindow | Flatten() | Перетворення 2D масиву в 1D | двовимірний масив | одновимірний масив | MainWindow.xaml.cs |

Продовження таблиці 4.2

# 5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 5.1 План тестування

Метою даного розділу є перевірка головних функцій програми та обробки виключних ситуацій. Тестування охоплює ключові спроби взаємодії користувача з інтерфейсом. Результати тестів оформлено у вигляді таблиць. Результати тестів оформлено у вигляді таблиць.

а) Перевірка можливості переміщення плитки, яка межує з пустою;

б) Перевірка заборони руху плитки, яка не межує з пустою;

в) Перевірка запуску розв’язування при вже вирішеній конфігурації за умови ручного розмішування;

г) Перевірка відображення вікна статистики;

д) Перевірка згенерованого стану на можливість розв’язання;

е) Перевірка збереження логів;

ж) Перевірка коректності видалення та створення файлу логів;

з) Перевірка правильності запуску файлу логу для перегляду повної статистики.

### 5.2 Приклади тестування

Таблиця 5.1 – Перевірка можливості переміщення плитки, яка межує з пустою клітинкою.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість переміщення плитки, яка межує з пустою |
| Початковий стан програми | Відкрите поле гри |
| Вхідні дані | Клік по суміжній до пустої плитці |
| Схема проведення тесту | Натискання на суміжну плитку |
| Очікуваний результат | Плитка переміщується |
| Стан програми після проведення випробувань | Плитка переміщена |

Таблиця 5.2 – Перевірка заборони руху фішки, яка не межує з пустою клітинкою.

|  |  |
| --- | --- |
| Мета тесту | Перевірити заборону руху плитки, яка не межує з пустою |
| Початковий стан програми | Відкрите поле гри |
| Вхідні дані | Клік по плитці, не суміжній з пустою |
| Схема проведення тесту | Натискання на не суміжну плитку |
| Очікуваний результат | Нічого не відбувається |
| Стан програми після проведення випробувань | Плитка не змінила положення на полі гри |

Таблиця 5.3 – Перевірка запуску розв’язування при вже вирішеній конфігурації за умови ручного розмішування.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість запуску розв’язування при вже вирішеній конфігурації за умови ручного розмішування |
| Початковий стан програми | Відкрите поле гри, вручну вирішена конфігурація плиток |
| Вхідні дані | Конфігурація на полі ідентична до цільової конфігурації |
| Схема проведення тесту | Ручне виставлення вирішеної конфігурації поля |
| Очікуваний результат | Виведення повідомлення про вже вирішену конфігурацію на полі |
| Стан програми після проведення випробувань | Далі продовжується ручне розмішування |

Таблиця 5.4 – Перевірка відображення вікна статистики.

|  |  |
| --- | --- |
| Мета тесту | Перевірити відображення вікна статистики |
| Початковий стан програми | Відкрите поле гри; довільна конфігурація , яка може бути розв’язана |
| Вхідні дані | Клік по кнопці «Solve» |
| Схема проведення тесту | Завершити гру шляхом натискання «Solve» |
| Очікуваний результат | Відображення вікна статистики |
| Стан програми після проведення випробувань | Відображено вікно з повідомленням про кількість відвіданих станів, глибину рішення та час роботи двох алгоритмів задля порівняння |

Таблиця 5.5 – Перевірка згенерованого стану на можливість розв’язання.

|  |  |
| --- | --- |
| Мета тесту | Перевірити згенерований стан на можливість розв’язання |
| Початковий стан програми | Відкрите поле гри, довільно згенерована конфігурація |
| Вхідні дані | Клік по кнопці «Shuffle», після чого клік по кнопці «Solve» |
| Схема проведення тесту | Згенерувати новий стан та перевірити чи може він бути вирішений |
| Очікуваний результат | Конфігурацію розв’язано, виведено вікно статистики |
| Стан програми після проведення випробувань | Можна знову згенерувати новий стан, після чого розв’язати його вручну або автоматично |

Таблиця 5.6 – Перевірка збереження логів.

|  |  |
| --- | --- |
| Мета тесту | Перевірити збереження логів |
| Початковий стан програми | Відкрите поле гри, довільно згенерована конфігурація |
| Вхідні дані | Клік по кнопці «Solve» |
| Схема проведення тесту | Розв’язати довільну конфігурація автоматично, після чого лог про цей стан буде автоматично збережено після закриття вікна статистики |
| Очікуваний результат | У файл «logs.json» успішно збережено інформацію про початковий стан, алгоритм розв’язання, кількість відвіданих станів, глибину рішення та час роботи цього алгоритму. |
| Стан програми після проведення випробувань | Файл «logs.json» можна відкрити для перегляду статистики |

Таблиця 5.7 – Перевірка коректності видалення та створення файлу логів.

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректність видалення та створення файлу логів |
| Початковий стан програми | Відкрите поле гри, файл «logs.json» існує в теці |
| Вхідні дані | Файл існує та не є пустим (є хочаб один лог розв’язання довільної конфігурації) |
| Схема проведення тесту | 1. Очистити файл (видалити)  2. Створити файл |
| Очікуваний результат | При натисканні кнопки «Clear logs» має з’явитись повідомлення про успішне видалення файл.  Для того щоб знову його створити, необхідно хоча б 1 раз автоматично розв’язати довільну конфігурацію. |
| Стан програми після проведення випробувань | Файл «logs.json» було успішно видалено та створено знову. Фактично відбулась очистка логу. |

Таблиця 5.8 – Перевірка правильності запуску файлу логу для перегляду повної статистики.

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильні запуску файлу логу для перегляду повної статистики |
| Початковий стан | Відкрите поле гри, файл «logs.json» існує та не є пустим. |
| Вхідні дані | Клік по кнопці «Open logs.json» |
| Схема проведення тесту | Якщо файл існує і не є пустим, то при натисканні на кнопку «Open logs.json» відкриється «Блокнот» з логами |
| Очікуваний результат | Відкриття файлу для читання логів |
| Стан програми після проведення випробувань | Файл «logs.json» успішно відкрився. |

# 6 ІНСТРУКЦІЯ КОРИСТУВАЧА

### 6.1 Робота з програмою

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

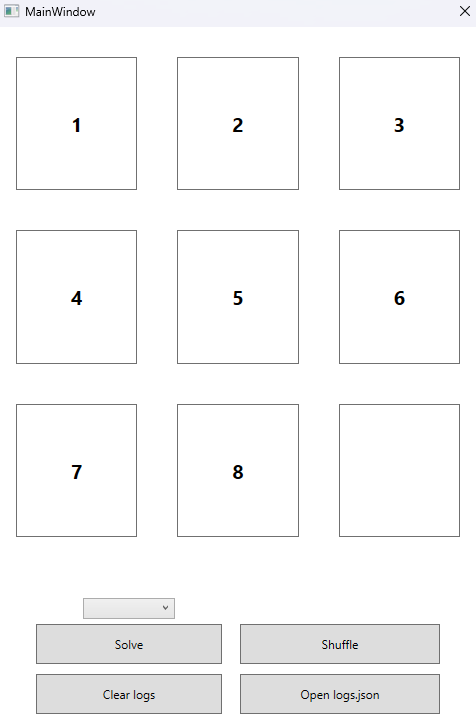


Рисунок 6.1. – Головне вікно програми.

При натисненні кнопки «Shuffle» буде згенерована довільна конфігурація, яку можливо розв’язати (Рисунок 6.2.).

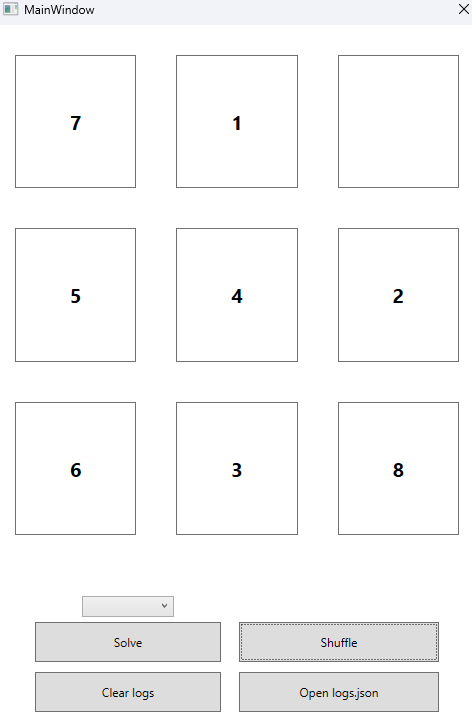


Рисунок 6.2. – Генерація довільної конфігурації, яку можливо розв’язати.

Також можливе ручне розмішування шляхом натискання на плитки, суміжні з пустою (Рисунок 6.3).

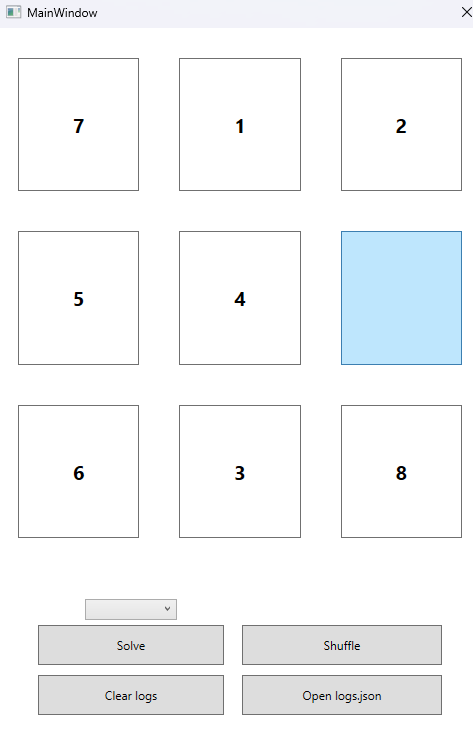


Рисунок 6.3 – Ручне розмішування шляхом натискання на плитки суміжні з пустою.

При натисканні елементу ComboBox вилазить список з вибором основного алгоритму розв’язання, статистику якого буде записано в лог. Якщо алгоритм не вибирати то автоматично основним стане алгоритм А\* (Рисунок 6.4).

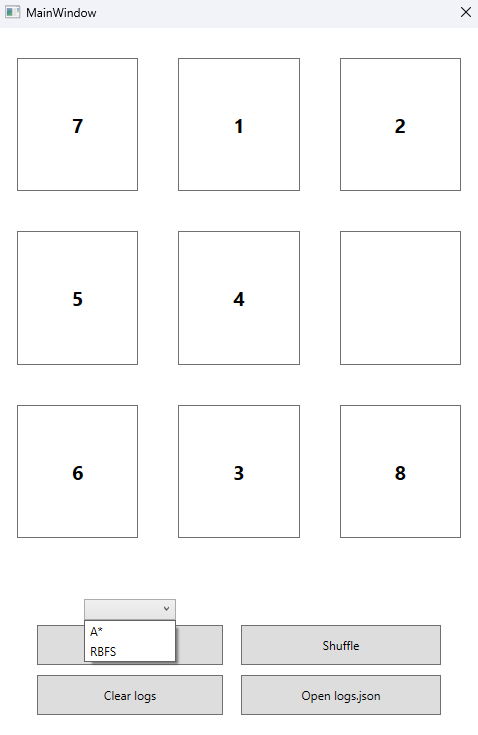


Рисунок 6.4. – Відображення списку вибору основного алгоритму розв’язання.

Якщо конфігурація не вирішена вручну, то після натискання на клавішу «Solve» з’явиться повідомлення про успішний початок розв’язання. (Рисунок 6.5.)

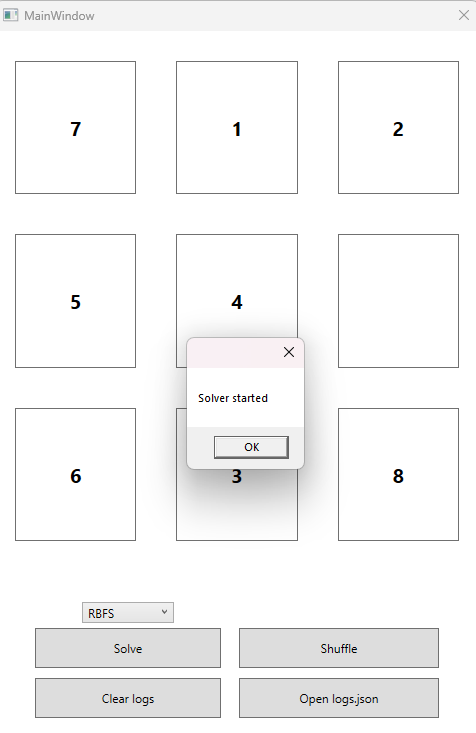


Рисунок 6.5. – Повідомлення про успішний початок розв’язання конфігурації

Після закінчення вирішення алгоритмами даної конфігурації буде відображено вікно зі статистикою обох алгоритмів, де на першому місці завжди стоїть обраний алгоритм. (Рисунок 6.6)

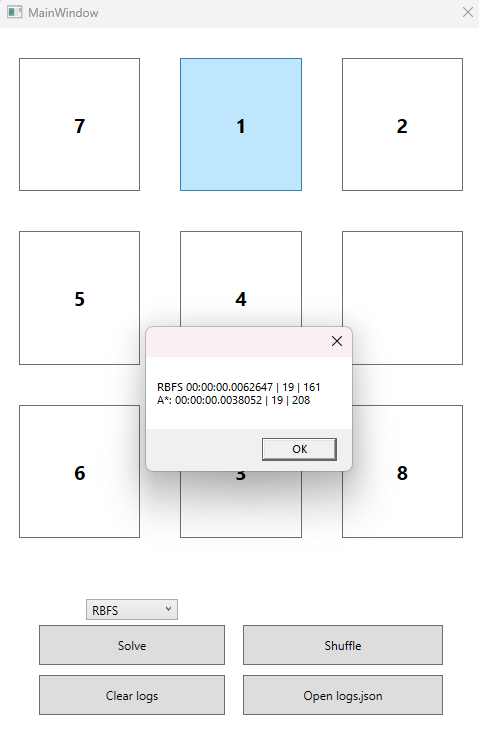


Рисунок 6.6. – Вікно статистики роботи двох алгоритмів.

Після закриття вікна статистики (рис. 6.6) інформація про розв’язок буде автоматично збережена у файл логів. Щоб це перевірити, необхідно натиснути кнопку «Open logs.json» та перевірити чи було успішно збережено лог. (Рисунок 6.7)

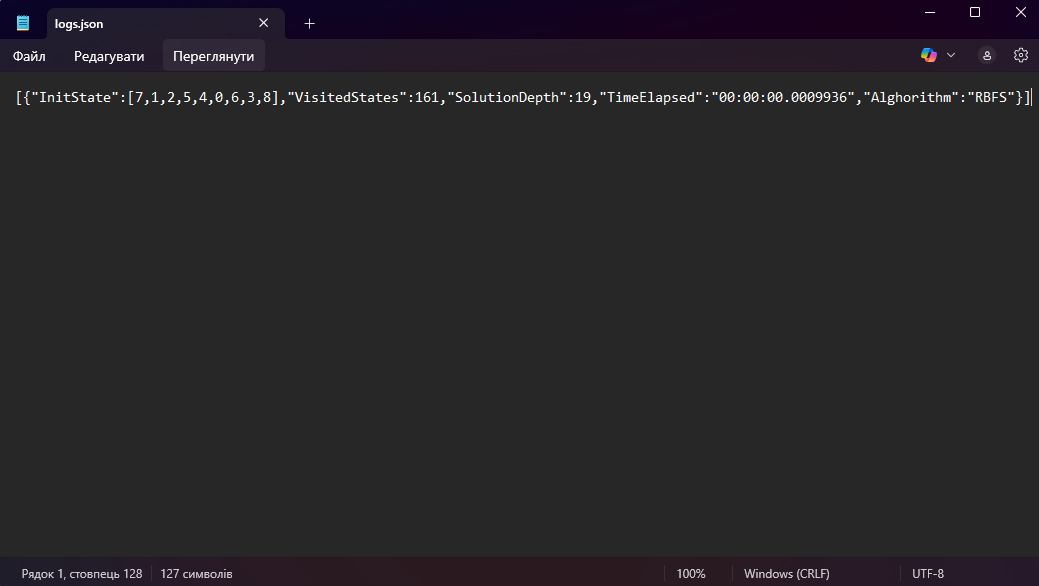


Рисунок 6.7 – Успішне збереження логу та перегляд усіх логів роботи програми.

Щоб очистити логи (фактично їх видалити) необхідно натиснути на кнопку «Clear logs», після чого, якщо файл існував і був успішно видалений, виведеться повідомлення про успішне видалення файлу логів (Рисунок 6.8).

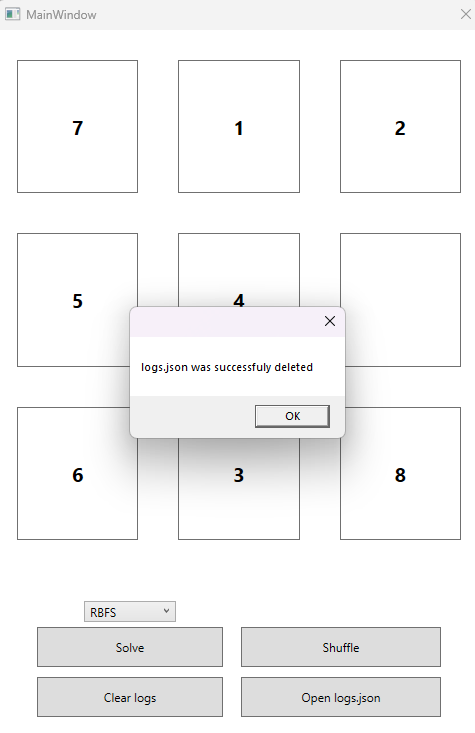


Рисунок 6.8 – Успішне видалення файлу логів.

Якщо файлу не існує на момент натискання на кнопку, виведеться повідомлення про те, що файлу не існує з підказкою про те, як його створити. (Рисунок 6.9)

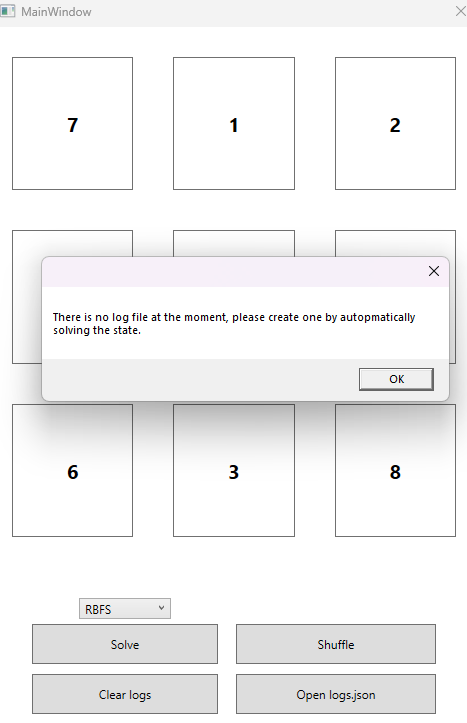


Рисунок 6.9 – Повідомлення про те, що файлу логів не існує.

Також при спробі відкрити файл для читання натискання кнопки «Open logs.json» коли його не існує, буде виведено повідомлення. (Рисунок 6.10)

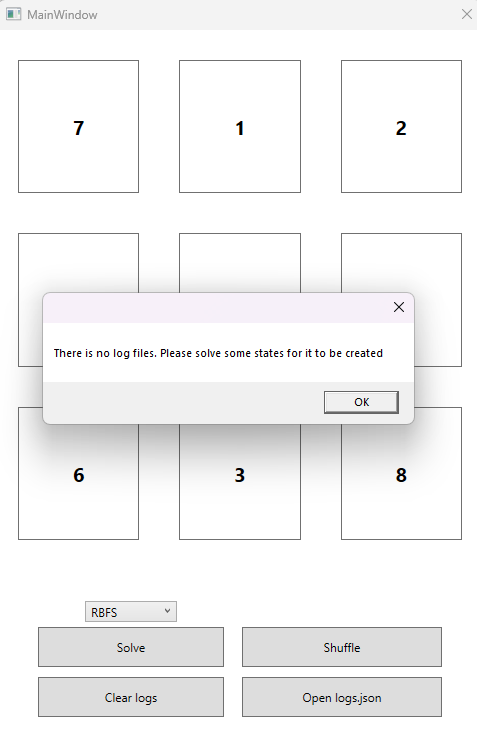


Рисунок 6.10 – Повідомлення про помилку відкриття файлу логів.

### 6.2 Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| ОС | Windows® 10  (з останніми оновленнями) | Windows® 10  (з останніми оновленнями) |
| Процесор | Intel® Pentium® III 1.0 GHz або аналогічні | 2.0 GHz або вище |
| ОЗП | 2 GB RAM | 4 GB RAM |
| Відеоадаптер | Intel® UHD Graphics | AMD® Radeon® Vega |
| Дисплей | 800x600 | 1024x768 або краще |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | .NET Runtime 6.0, .NET SDK 6.0 | |

# ВИСНОВКИ

У результаті виконання курсової роботи було розроблено та реалізовано програмне забезпечення для розв'язання головоломки «8-пазл» з графічним інтерфейсом користувача, створене з використанням технології WPF на платформі .NET. Програма забезпечує можливість взаємодії користувача з ігровим полем, включаючи ручне пересування плиток, генерацію випадкових розв'язуваних початкових станів та автоматичне розв'язання головоломки за допомогою двох різних алгоритмів: A\* (A-star) та RBFS (Recursive Best-First Search) з використанням Манхетенської евристичної функції.

У процесі роботи було проведено аналіз предметної області та математичних основ головоломки 8-пазл, визначено основні вимоги до функціональності та інтерфейсу, спроектовано та реалізовано алгоритми штучного інтелекту для пошуку оптимального рішення. Програмна реалізація виконана з дотриманням принципів об'єктно-орієнтованого програмування. Також було проведено комплексне тестування розробленого продукту та перевірено його працездатність на різних конфігураціях головоломки.

Особливо важливим результатом роботи є порівняльний аналіз ефективності алгоритмів A\* та RBFS, який показав їх різні переваги: A\* демонструє вищу швидкість роботи завдяки збереженню всіх відвіданих станів у пам'яті, тоді як RBFS є більш економним щодо використання пам'яті, що робить його придатним для розв'язання складніших задач з обмеженими ресурсами. Реалізована система логування дозволяє зберігати та аналізувати статистику роботи алгоритмів для подальших досліджень.

Розроблена програма підтвердила практичну доцільність застосування сучасних технологій програмування та алгоритмів штучного інтелекту для розв'язання прикладних задач. Робота сприяла поглибленню знань у сфері алгоритмізації, евристичних методів пошуку, об'єктно-орієнтованого підходу, побудови графічних інтерфейсів користувача на базі WPF та розробки програмного забезпечення загалом. Отримані практичні навички роботи з алгоритмами пошуку в просторі станів можуть бути застосовані для розв'язання більш складних задач оптимізації та планування

# ПЕРЕЛІК ПОСИЛАНЬ

1. Рассел С., Норвиг П. Искусственный интеллект: современный подход. 2-е изд. / С. Рассел, П. Норвиг. – М.: Вильямс, 2006. – 1408 с.
2. Корнейчук В.І., Тарасенко В.П. Алгоритми та структури даних: навчальний посібник / В.І. Корнейчук, В.П. Тарасенко. – К.: НАУ, 2018. – 392 с.
3. Richter J. CLR via C#. 4th Edition / J. Richter. – Microsoft Press, 2012. – 896 с.
4. Korf R.E. Linear-Space Best-First Search / R.E. Korf // Artificial Intelligence. – 1993. – Vol. 62, No. 1. – С. 41-78.
5. Johnson W.W., Story W.E. Notes on the "15" puzzle / W.W. Johnson, W.E. Story // American Journal of Mathematics. – 1879. – Vol. 2, No. 4. – С. 397-404.
6. Ratner D., Warmuth M. The (n²-1)-puzzle and related relocation problems / D. Ratner, M. Warmuth // Journal of Symbolic Computation. – 1990. – Vol. 10, No. 2. – С. 111-137.
7. Microsoft Documentation. Windows Presentation Foundation [Електронний ресурс] // Microsoft Docs. – Режим доступу:   
   <https://docs.microsoft.com/en-us/dotnet/desktop/wpf/>
8. Microsoft Documentation. C# Programming Guide [Електронний ресурс] // Microsoft Docs. – Режим доступу:   
   <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/>
9. A\* Search Algorithm - GeeksforGeeks [Електронний ресурс]. – Режим доступу: <https://www.geeksforgeeks.org/a-search-algorithm/>
10. 8 Puzzle Problem using A\* Algorithm [Електронний ресурс] // GeeksforGeeks. – Режим доступу:  
    <https://www.geeksforgeeks.org/8-puzzle-problem-using-search-algorithm/>
11. Manhattan Distance Heuristic [Електронний ресурс] // Wikipedia. – Режим доступу: <https://en.wikipedia.org/wiki/Taxicab_geometry>
12. Best-First Search [Електронний ресурс] // Wikipedia. – Режим доступу: <https://en.wikipedia.org/wiki/Best-first_search>

# ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

Інформатики та програмної інженерії

Затвердив

Керівник Головченко М. М.

« » 202\_ р.

Виконавець

Студент Радченко М. С.

« » 202\_ р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «8-puzzle»

з дисципліни:

«Основи програмування. Курсова робота»

Київ 2025

1. *Мета:* Метою курсової роботи є розробка програмного забезпечення для вирішення головоломки 8-puzzle за допомогою двох алгоритмів: A\* (із манхеттенською евристикою) та RBFS (із манхеттенською евристикою).
2. *Дата початку роботи*: «19» лютого 2025 р.
3. *Дата закінчення роботи*: «25» травня 2025 р.
4. *Вимоги до програмного забезпечення:*
   1. Функціональні вимоги:
5. Можливість приймати випадкову або задану початкову конфігурацію головоломки 8-puzzle.
6. Можливість вибору алгоритму розв’язання (A\* або RBFS).
7. Можливість візуалізація процесу розв’язання головоломки.
8. Можливість виведення статистики виконання: кількість відвіданих станів, час виконання, глибина рішення.
9. Можливість порівняння результатів роботи двох алгоритмів.
10. Можливість логування статистики.

б. Нефункціональні вимоги:

* Процесор: Intel Core i3-1115G4
* Об’єм ОЗП: 8 Гб
* Реалізація мовою програмування C#
* Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:
* ДСТУ 3008 - 2015 - Розробка технічної документації.

1. *Стадії та етапи розробки програмного забезпечення:*
2. Розробка алгоритмічної складової програмного забезпечення (до 02.03.2025 р.)
3. Об’єктно-орієнтований аналіз предметної області завдання (до 16.03.2025 р.)
4. Об’єктно-орієнтоване проєктування програмного забезпечення (до 30.03.2025 р.)
5. Розробка програмного забезпечення (до 04.05.2025 р.)
6. Тестування розробленого програмного забезпечення (до 18.05.2025 р.)
7. Демонстрація та захист програмного забезпечення (до \_\_.\_\_.202\_ р.)
8. *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 КОДУ

*Тексти програмного коду «8-puzzle»*

(назва програми (документа))

*GitHub репозиторій*

(вид носія даних)

*36 арк. 35 Кб*

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

*студента групи ІП-44 I курсу*

*Радченка М.С.*

Посилання на GitHub репозиторій: <https://github.com/flistrr/8Puzzle>

**MainWindow.xaml.cs**

using System.Collections.ObjectModel;

using System.Net.Mail;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Text.Json.Serialization;

using System.Text.Json;

using \_8Puzzle.classes;

using System.IO;

using System.Diagnostics;

namespace \_8Puzzle

{

public partial class MainWindow : Window

{

public Button[,] buttons = new Button[3, 3];

public int[,] tiles = new int[3,3];

public State state = new State();

public int[,] goalState = new int[,] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 0 } };

public MainWindow()

{

this.InitializeComponent();

InitButtons();

tiles = new int[,] { { 1, 2, 3}, { 4, 5, 6 }, {7, 8, 0 } };

state = new State(tiles);

UpdateButtons();

}

private void UpdateButtons()

{

for (int row = 0; row < 3; row++)

{

for (int col = 0; col < 3; col++)

{

int value = tiles[row, col];

buttons[row, col].Content = value == 0 ? "" : value.ToString();

}

}

}

private void InitButtons()

{

for (int row = 0; row < 3; row++)

{

for (int col = 0; col < 3; col++)

{

var button = new Button

{

FontSize = 20,

Margin = new Thickness(20),

Background = Brushes.White,

Foreground = Brushes.Black,

FontWeight = FontWeights.Bold,

};

button.Click += Tile\_Click;

Grid.SetRow(button, row);

Grid.SetColumn(button, col);

buttons[row, col] = button;

GameGrid.Children.Add(button);

}

}

}

private void Tile\_Click(object sender, RoutedEventArgs e)

{

var button = sender as Button;

int row = Grid.GetRow(button);

int col = Grid.GetColumn(button);

int emptyRow = -1, emptyCol = -1;

for (int r = 0; r < 3; r++)

for (int c = 0; c < 3; c++)

if (tiles[r, c] == 0)

{

emptyRow = r;

emptyCol = c;

}

if ((Math.Abs(row - emptyRow) == 1 && col == emptyCol) || (Math.Abs(col - emptyCol) == 1 && row == emptyRow))

{

tiles[emptyRow, emptyCol] = tiles[row, col];

tiles[row, col] = 0;

UpdateButtons();

}

}

private int[,] GenerateShuffledState()

{

var list = Enumerable.Range(0, 9).ToList();

var rnd = new Random();

int[,] newState = new int[3, 3];

do

{

list = list.OrderBy(x => rnd.Next()).ToList();

}

while (!IsSolvable(list));

for (int i = 0; i < 9; i++)

newState[i / 3, i % 3] = list[i];

return newState;

}

private bool IsSolvable(List<int> tiles)

{

int inv = 0;

for (int i = 0; i < 9; i++)

{

if (tiles[i] == 0) continue;

for (int j = i + 1; j < 9; j++)

{

if (tiles[j] == 0) continue;

if (tiles[i] > tiles[j]) inv++;

}

}

return inv % 2 == 0;

}

private void Shuffle\_Click(object sender, RoutedEventArgs e)

{

tiles = GenerateShuffledState();

UpdateButtons();

}

private async void Solve\_Click(object sender, RoutedEventArgs e)

{

MessageBox.Show("Solver started");

string chosenMethod = (Choice.SelectedItem as ComboBoxItem)?.Content?.ToString() ?? "A\*";

var initialState = new State(tiles);

var copy = new State(initialState.GetBoard());

var heuristic = new ManhattanHeuristic();

List<Move>? primary, secondary;

Statistics statsPrimary = new Statistics();

Statistics statsSecondary = new Statistics();

await Task.Run(() =>

{

switch (chosenMethod)

{

case "A\*":

primary = new AStar(heuristic, statsPrimary).Solve(initialState);

secondary = new RBFS(heuristic, statsSecondary).Solve(initialState);

ShowStats(statsPrimary, statsSecondary, '1');

break;

case "RBFS":

primary = new RBFS(heuristic, statsPrimary).Solve(initialState);

secondary = new AStar(heuristic, statsSecondary).Solve(initialState);

ShowStats(statsPrimary, statsSecondary, '2');

break;

default:

throw new ArgumentException();

}

});

var log = new Logs()

{

InitState = Flatten(tiles),

Alghorithm = chosenMethod,

VisitedStates = (chosenMethod == "A\*") ? statsPrimary.AStarVisitedStates : statsPrimary.RBFSVisitedStates,

SolutionDepth = (chosenMethod == "A\*") ? statsPrimary.AStarSolutionDepth : statsPrimary.RBFSSolutionDepth,

TimeElapsed = (chosenMethod == "A\*") ? statsPrimary.AStarElapsedTime : statsPrimary.RBFSElapsedTime

};

Logs.LogSolution(log, "logs.json");

}

private void ShowStats(Statistics primary, Statistics secondary, char order)

{

switch (order)

{

case '1':

MessageBox.Show($"A\*: {primary.AStarElapsedTime} | {primary.AStarSolutionDepth} | {primary.AStarVisitedStates}"

+ $"\nRBFS {secondary.RBFSElapsedTime} | {secondary.RBFSSolutionDepth} | {secondary.RBFSVisitedStates}");

break;

case '2':

MessageBox.Show($"RBFS {primary.RBFSElapsedTime} | {primary.RBFSSolutionDepth} | {primary.RBFSVisitedStates}"

+ $"\nA\*: {secondary.AStarElapsedTime} | {secondary.AStarSolutionDepth} | {secondary.AStarVisitedStates}");

break;

default:

break;

}

}

public static int[] Flatten(int[,] state)

{

int rows = state.GetLength(0);

int cols = state.GetLength(1);

int[] flat = new int[rows \* cols];

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

{

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

{

flat[i \* cols + j] = state[i, j];

}

}

return flat;

}

private void ClearLogs\_Click(object sender, RoutedEventArgs e)

{

FileInfo logInfo = new FileInfo("logs.json");

if (File.Exists(logInfo.FullName) && logInfo.Length > 0)

{

File.Delete(logInfo.FullName);

MessageBox.Show("logs.json was successfuly deleted");

}

else

{

MessageBox.Show("There is no log file at the moment, please create one by autopmatically solving the state.");

}

}

private void OpenLogs\_Click(object sender, RoutedEventArgs e)

{

if (File.Exists("logs.json"))

{

string filePath = "C:\\Users\\flistrrr\\Documents\\kr\\8Puzzle\\bin\\Debug\\net6.0-windows\\logs.json";

Process.Start("notepad.exe", filePath);

}

else

{

MessageBox.Show("There is no log files. Please solve some states for it to be created");

}

}

}

}

**AStar.cs**

using \_8Puzzle.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Xml.Linq;

namespace \_8Puzzle

{

public class AStar : IAlghorithm

{

private readonly IHeuristic heuristic;

private readonly Statistics stats;

public AStar(IHeuristic heuristic, Statistics stats)

{

this.heuristic = heuristic;

this.stats = stats;

}

public List<Move>? Solve(State initialState)

{

var start = DateTime.Now;

if (initialState.IsGoalState())

{

MessageBox.Show("The state is already solved.");

return new List<Move>();

}

int visited = 0;

var open = new PriorityQueue<Node, int>();

var all = new Dictionary<State, Node>();

var startNode = new Node(initialState, null, null, 0, heuristic.Calculate(initialState));

open.Enqueue(startNode, startNode.F);

all[initialState] = startNode;

while (open.Count > 0)

{

var cur = open.Dequeue();

visited++;

if (cur.State.IsGoalState())

{

var sol = Reconstruct(cur);

stats.CollectAStarStatistics(visited, sol.Count, DateTime.Now - start);

return sol;

}

foreach (var mv in cur.State.GetPossibleMoves())

{

var ns = cur.State.ApplyMove(mv);

int g = cur.G + 1;

int h = heuristic.Calculate(ns);

if (!all.TryGetValue(ns, out var ex) || g < ex.G)

{

var nn = new Node(ns, cur, mv, g, h);

open.Enqueue(nn, nn.F);

all[ns] = nn;

}

}

}

stats.CollectAStarStatistics(visited, -1, DateTime.Now - start);

return null;

}

public static List<Move> Reconstruct(Node node)

{

var path = new List<Move>();

while (node.Parent != null)

{

path.Add(node.Move!);

node = node.Parent;

}

path.Reverse();

return path;

}

}

}

**Logs.cs**

using \_8Puzzle.Interfaces;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Text.Json;

using System.Text.Json.Serialization;

using System.Threading.Tasks;

namespace \_8Puzzle.classes

{

class Logs

{

public int[] InitState { get; set; }

public int VisitedStates { get; set; }

public int SolutionDepth { get; set; }

public TimeSpan TimeElapsed { get; set; }

public string Alghorithm { get; set; }

public static void LogSolution(Logs log, string filePath)

{

List<Logs> logs = new List<Logs>();

if (File.Exists(filePath))

{

string existingJson = File.ReadAllText(filePath);

if (!string.IsNullOrEmpty(existingJson))

{

logs = JsonSerializer.Deserialize<List<Logs>>(existingJson);

}

}

logs.Add(log);

var options = new JsonSerializerOptions { WriteIndented = false };

string json = JsonSerializer.Serialize(logs, options);

File.WriteAllText(filePath, json);

if(logs == null)

{

Console.WriteLine("No log was written.");

}

}

}

}

**ManhattenHeuristic.cs**

using \_8Puzzle.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_8Puzzle

{

public class ManhattanHeuristic : IHeuristic

{

private static readonly (int Row, int Col)[] goalPositions = new (int, int)[9]

{

(2,2), (0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1)

};

public int Calculate(State state, State? goalState = null)

{

int[,] board = state.GetBoard();

int dist = 0;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

{

int v = board[i, j];

if (v != 0)

{

var (gr, gc) = goalPositions[v];

dist += Math.Abs(i - gr) + Math.Abs(j - gc);

}

}

return dist;

}

}

}

**Move.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_8Puzzle

{

public class Move

{

public Direction Direction { get; }

public Move(Direction direction) => Direction = direction;

public override string ToString() => Direction.ToString();

}

}

**Node.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_8Puzzle

{

public class Node : IComparable<Node>

{

public State State { get; }

public Node? Parent { get; }

public Move? Move { get; }

public int G { get; }

public int H { get; }

public int F => G + H;

public Node(State s, Node? p, Move? m, int g, int h) { State = s; Parent = p; Move = m; G = g; H = h; }

public int CompareTo(Node? other) => other == null ? 1 : F.CompareTo(other.F);

}

}

**RBFS.cs**

using \_8Puzzle;

using \_8Puzzle.Interfaces;

using System.Windows;

public class RBFS : IAlghorithm

{

private IHeuristic heuristic;

private Statistics stats;

public RBFS(IHeuristic heuristic, Statistics stats)

{

this.heuristic = heuristic;

this.stats = stats;

}

public List<Move>? Solve(State state)

{

var start = DateTime.Now;

var startNode = new Node(state, null, null, 0, heuristic.Calculate(state));

int visited = 0;

var visitedStates = new HashSet<State>();

var (goalNode, \_, count) = Solution(startNode, int.MaxValue, visited);

if (goalNode == null)

return null;

var path = Reconstruct(goalNode);

int depth = path.Count;

stats.CollectRBFSStatistics(count, depth, DateTime.Now - start);

return path;

}

public (Node? node, int fCost, int visited) Solution(Node node, int fLimit, int visited)

{

visited++;

if (node.State.IsGoalState())

return (node, node.F, visited);

var successors = node.State.GetPossibleMoves()

.Select(move =>

{

var nextState = node.State.ApplyMove(move);

if (node.Parent != null && node.Parent.State.Equals(nextState))

return null;

int g = node.G + 1;

int h = heuristic.Calculate(nextState);

return new Node(nextState, node, move, g, h);

})

.Where(n => n != null)

.Cast<Node>()

.ToList();

if (successors.Count == 0)

return (null, int.MaxValue, visited);

var fValues = successors.ToDictionary(s => s, s => Math.Max(s.F, node.F));

while (true)

{

successors.Sort((a, b) => fValues[a].CompareTo(fValues[b]));

var best = successors[0];

if (fValues[best] > fLimit)

return (null, fValues[best], visited);

int alternative = successors.Count > 1 ? fValues[successors[1]] : int.MaxValue;

var (result, newF, newVisited) = Solution(best, Math.Min(fLimit, alternative), visited);

visited = newVisited;

fValues[best] = newF;

if (result != null)

return (result, newF, visited);

}

}

public static List<Move> Reconstruct(Node node)

{

var path = new List<Move>();

while (node.Parent != null)

{

path.Add(node.Move!);

node = node.Parent;

}

path.Reverse();

return path;

}

}

**State.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_8Puzzle

{

public enum Direction

{

Up, Down, Left, Right

}

public class State

{

private readonly int[,] tiles;

private const int rows = 3;

private const int cols = 3;

private int[,] goalState = new int[,] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 0 } };

public State(int[,] tiles)

{

this.tiles = new int[rows, cols];

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

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

this.tiles[i, j] = tiles[i, j];

}

public void UpdateBoard(State state)

{

int[,] ints = state.GetBoard();

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

{

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

{

}

}

}

public State()

{

Random rng = new Random();

tiles = GenerateSolvableState(rng);

}

public static int[,] GenerateSolvableState(Random rand)

{

int[] flatState;

do

{

flatState = Enumerable.Range(0, 9)

.OrderBy(\_ => rand.Next())

.ToArray();

} while (!IsSolvable(flatState));

int[,] board = new int[3, 3];

for (int i = 0; i < 9; i++)

{

board[i / 3, i % 3] = flatState[i];

}

return board;

}

private static bool IsSolvable(int[] puzzle)

{

int inversions = 0;

for (int i = 0; i < puzzle.Length; i++)

{

for (int j = i + 1; j < puzzle.Length; j++)

{

if (puzzle[i] != 0 && puzzle[j] != 0 && puzzle[i] > puzzle[j])

inversions++;

}

}

return inversions % 2 == 0;

}

public int[,] GetBoard()

{

int[,] copy = new int[rows, cols];

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

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

copy[i, j] = tiles[i, j];

return copy;

}

public (int Row, int Col) GetPosition(int value)

{

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

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

if (tiles[i, j] == value)

return (i, j);

throw new ArgumentException("Value not found");

}

public bool IsGoalState()

{

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (this.tiles[i, j] != goalState[i, j])

return false;

return true;

}

public List<Move> GetPossibleMoves()

{

var moves = new List<Move>();

var (r, c) = GetPosition(0);

if (r > 0) moves.Add(new Move(Direction.Up));

if (r < rows - 1) moves.Add(new Move(Direction.Down));

if (c > 0) moves.Add(new Move(Direction.Left));

if (c < cols - 1) moves.Add(new Move(Direction.Right));

return moves;

}

public State ApplyMove(Move move)

{

int[,] newTiles = GetBoard();

var (r, c) = GetPosition(0);

int nr = r, nc = c;

switch (move.Direction)

{

case Direction.Up: nr = r - 1; break;

case Direction.Down: nr = r + 1; break;

case Direction.Left: nc = c - 1; break;

case Direction.Right: nc = c + 1; break;

}

newTiles[r, c] = newTiles[nr, nc];

newTiles[nr, nc] = 0;

return new State(newTiles);

}

public override bool Equals(object? obj)

{

if (obj is not State other) return false;

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

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

if (tiles[i, j] != other.tiles[i, j])

return false;

return true;

}

public override int GetHashCode()

{

int hash = 17;

foreach (int t in tiles) hash = hash \* 31 + t;

return hash;

}

}

}

**Statistics.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_8Puzzle

{

public class Statistics

{

public int AStarVisitedStates { get; private set; }

public int AStarSolutionDepth { get; private set; }

public TimeSpan AStarElapsedTime { get; private set; }

public int RBFSVisitedStates { get; private set; }

public int RBFSSolutionDepth { get; private set; }

public TimeSpan RBFSElapsedTime { get; private set; }

public void CollectAStarStatistics(int visited, int depth, TimeSpan duration)

{

AStarVisitedStates = visited;

AStarSolutionDepth = depth;

AStarElapsedTime = duration;

}

public void CollectRBFSStatistics(int visited, int depth, TimeSpan duration)

{

RBFSVisitedStates = visited;

RBFSSolutionDepth = depth;

RBFSElapsedTime = duration;

}

}

}

**Interfaces.cs**

namespace \_8Puzzle.Interfaces

{

public interface IAlghorithm

{

List<Move>? Solve(State initialState);

}

public interface IHeuristic

{

int Calculate(State state, State? goalState = null);

}

}