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

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:

Мета роботи: Головною метою створення програмного забезпечення для задачі «8-puzzle» є забезпечення користувачів інструментами для інтерактивного розв’язання головоломки з використанням алгоритмів пошуку. Розроблена система призначена як для вивчення та демонстрації роботи алгоритмів штучного інтелекту (А\* та RBFS алгоритми з Манхетенською евристикою), так і для практичного застосування у процесі навчання та дослідження методів автоматизованого пошуку рішень.

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

# 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.

\*Стан – характеристика поля, що представляє собою послідовність плиток у вигляді логічного списку, або ж матриці елементів поля.

# 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.

\*Логування - це процес автоматичного запису інформації про події, що відбуваються під час роботи програми або системи. Логи (журнали подій) дозволяють відстежувати хід виконання програми, фіксувати помилки, попередження, дії користувача, а також системні повідомлення.

# 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

Основний алгоритм гри «8-puzzle» складається з кількох основних етапів:

1. Переміщення плиток вручну користувачем;
2. Автоматичне розмішування;
3. Автоматичне вирішення задачі за допомогою алгоритмів пошуку.

Перші два етапи реалізуються як реакція на події (сигнали), не мають чітко визначеної точки входу та є частиною базової логіки гри. Тому вони розглядаються без суворої алгоритмічної послідовності.

#### 3.1 Переміщення плиток

Виконується або у відповідь на натиснення користувачем плитки, або в процесі автоматичного розв’язання:

1. Отримуємо координати вибраної плитки.
2. Перевіряємо, чи є вона суміжною з порожньою клітинкою.
3. Якщо так — обмінюємо місцями з порожньою.
4. Інакше — ігноруємо дію.

#### 3.2 Автоматичне розмішування

Для уникнення невирішуваних комбінацій використовується розмішування шляхом послідовного виконання валідних ходів із правильної конфігурації.

1. Повторити фіксовану кількість раз:  
   a. Згенерувати випадковий напрямок руху.  
   b. Перевірити можливість переміщення порожньої клітинки у цьому напрямку.  
   c. Якщо можливо — виконати рух, інакше згенерувати інший напрямок.  
   d. Не повторювати попередній хід у зворотному напрямку.

#### 3.3 Автоматичне розв’язання

Автоматичне розв’язання реалізоване з використанням алгоритмів **A\*** або **RBFS**. Нижче наведено опис етапів для **A\***, як основного алгоритму.

#### 3.4 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:
   * Манхеттенська відстань — сума абсолютних різниць координат кожної плитки від цільового положення.
5. **КІНЕЦЬ**

#### 3.5 Перевірка на повторення

1. **ПОЧАТОК**
2. Для кожного елемента стану перевірити наявність в дереві.
   * Якщо відсутній — додати новий вузол.
   * 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**

### 3.6 Реалізація RBFS

Алгоритм **RBFS** виконується рекурсивно та має схожі кроки з A\*, однак не використовує велику чергу:

1. **ПОЧАТОК**
2. Запустити рекурсивну функцію пошуку з початкового стану.
3. На кожному рівні вибирається найкращий стан (з найменшим f).
4. Якщо найкращий стан гірший за поточну межу — відкат.
5. Інакше — спроба заглиблення з новою пороговою оцінкою.
6. Повторення до знаходження цілі або повернення без результату.
7. **КІНЕЦЬ**

# 4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 4.1 Діаграма класів програмного забезпечення

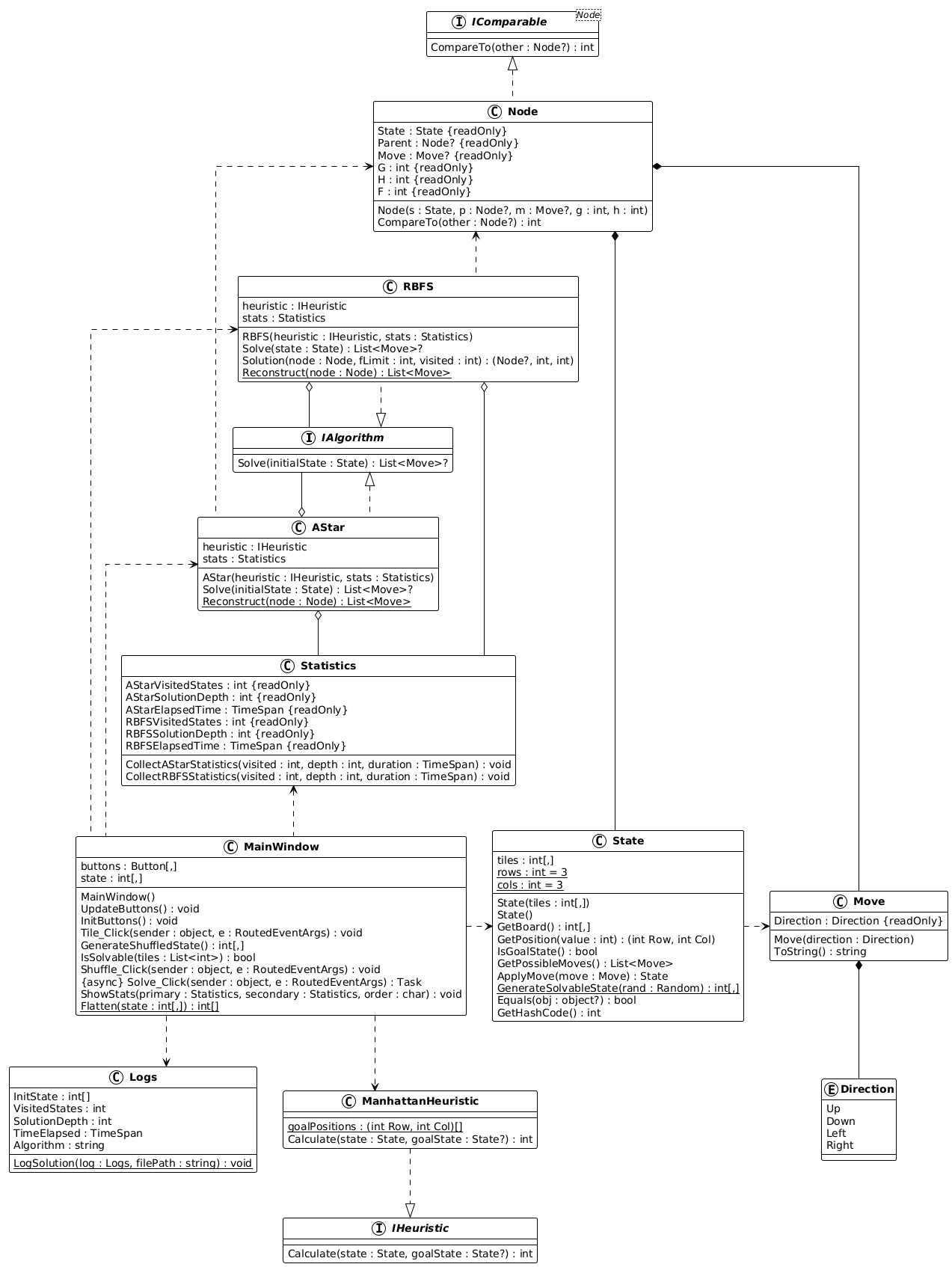


Рисунок 1.1 – Діаграма класів

### 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 |

#### 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 |

# 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 програми | 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 застосунок «Блокнот» з логами роботи програми |
| 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).

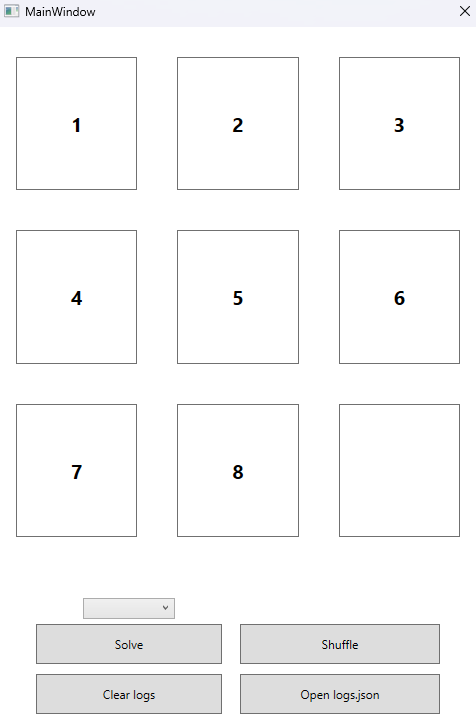


Рисунок 6.1. – Головне вікно програми.

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

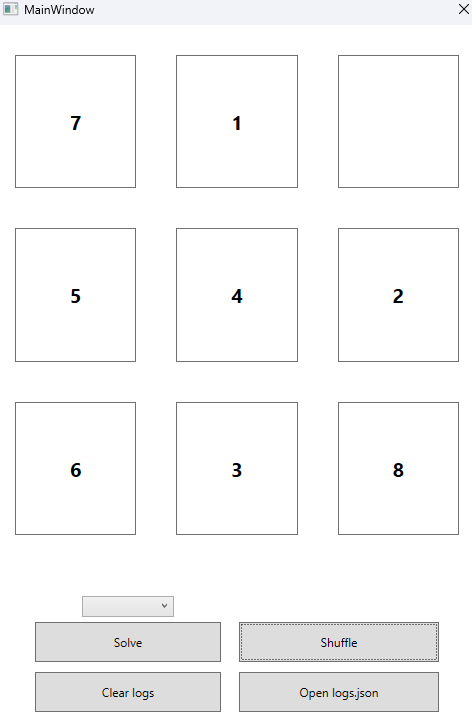


Рисунок 6.2. – Генерація довільної конфігурації, яку можливо розв’язати.

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

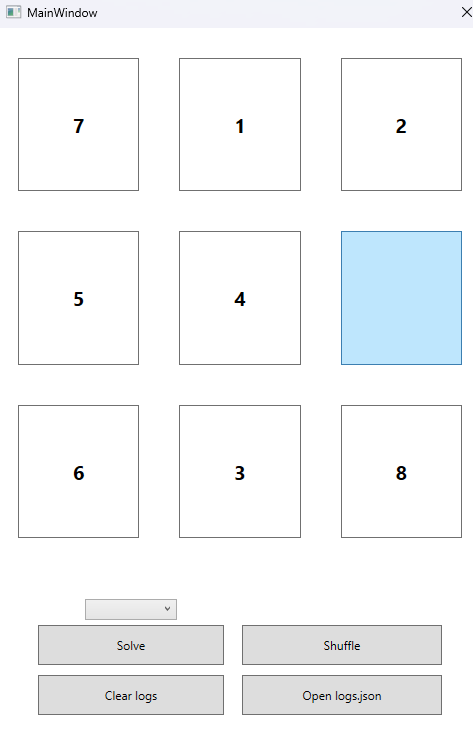


Рисунок 6.3 – Ручне розмішування шляхом натискання на плитки суміжні з пустою.

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).

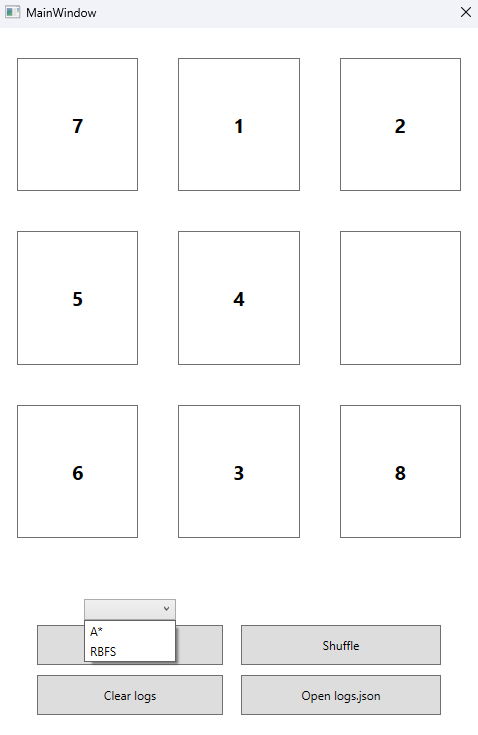


Рисунок 6.4. – Відображення списку вибору основного алгоритму розв’язання.

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

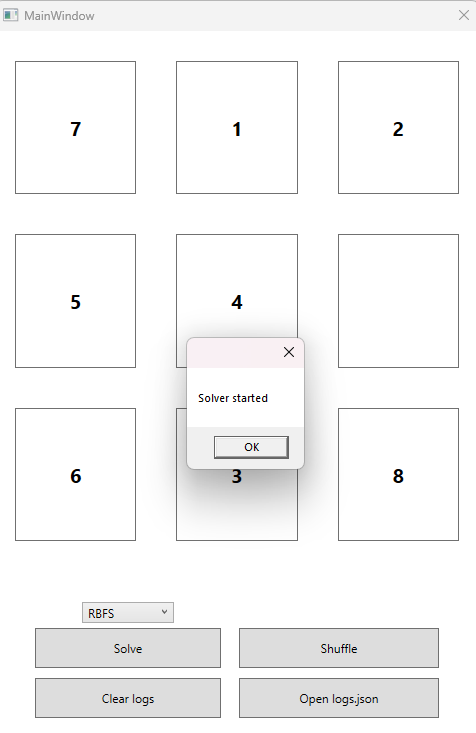


Рисунок 6.5. – Повідомлення про успішний початок розв’язання конфігурації

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

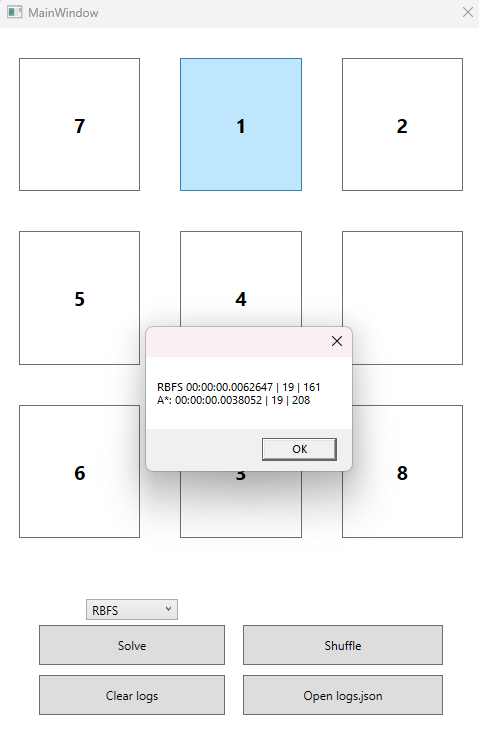


Рисунок 6.6. – Вікно статистики роботи двох алгоритмів.

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)

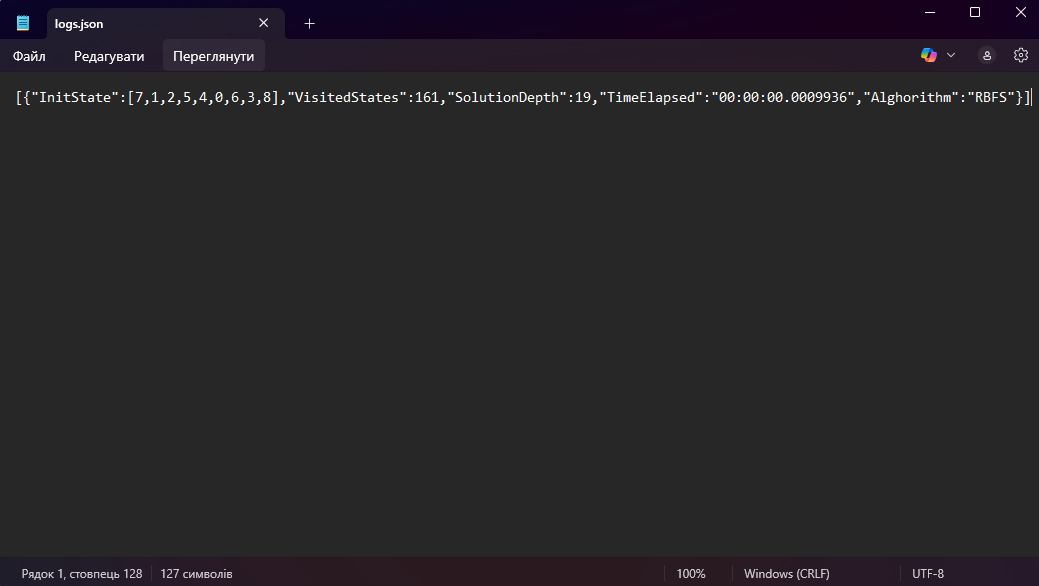


Рисунок 6.7 – Успішне збереження логу та перегляд усіх логів роботи програми.

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).

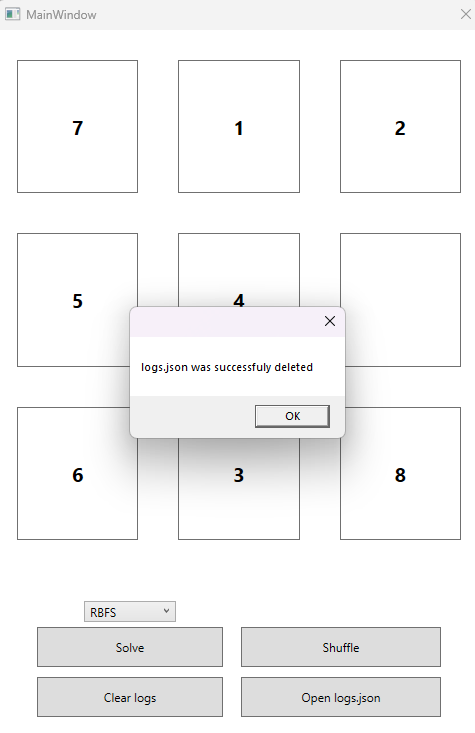


Рисунок 6.8 – Успішне видалення файлу логів.

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)

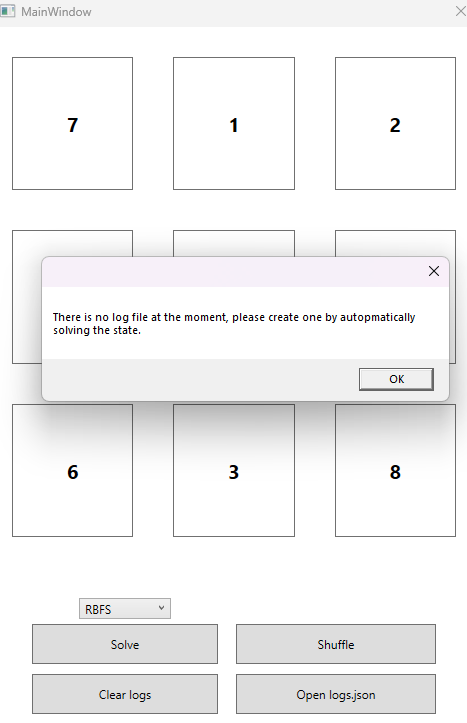


Рисунок 6.9 – Повідомлення про те, що файлу логів не існує.

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

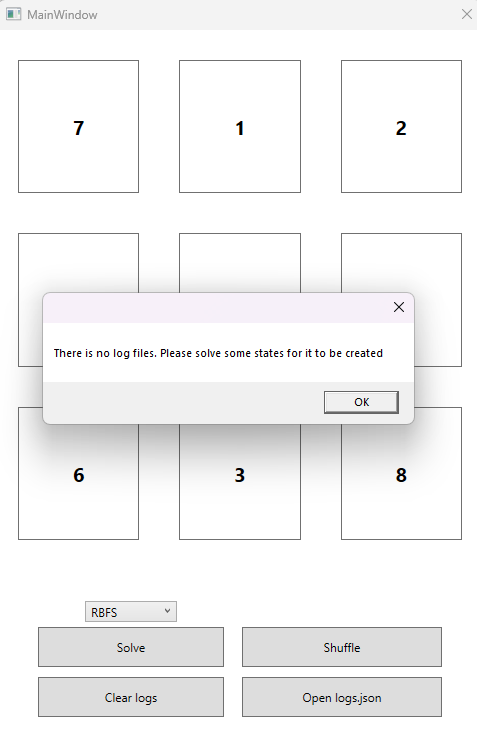


Рисунок 6.10 – Повідомлення про помилку відкриття файлу логів.

### 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 | |

# 7 ВИСНОВКИ