ЗМІСТ

[ВСТУП 4](#_Toc390260761)

[1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ 6](#_Toc390260762)

[2 ОБҐРУНТУВАННЯ АЛГОРИТМУ І СТРУКТУРИ ПРОГРАМИ 7](#_Toc390260763)

[3 РОЗРОБКА ПРОГРАМИ 8](#_Toc390260764)

[4 ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ 15](#_Toc390260765)

[ВИСНОВОК 20](#_Toc390260766)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 21](#_Toc390260767)

ДОДАТОК А………………………………………………………………..22

# АНОТАЦІЯ

У цій курсовій роботі описано об’єктно-орієнтований підхід розробки програмного продукту мовою програмування С++. Використано уніфіковану мову моделювання (UML) для відображення акторами та прецедентами потреб до системи. Для створення графічного інтерфейсу користувача (GUI) використали Swing. В кінцевому результаті отримали гру “П’ятнашки”.

# ВСТУП

Курсовий проект є комплексним завданням по застосуванню та закріпленню вивченого матеріалу за курс з ведучої дисципліни (в даному випадку з об'єктно-орієнтованого програмування). Він включає в себе розробку повноцінної системи (згідно з варіантом) і дає можливість проявити креативність і свої здібності в програмуванні та на кодовій частині проекту.

Розробляючи проект дуже важливим аспектом є створення раціональної системи та її структури, які б підходили під усі вимоги, та найголовніше – виконувала свою задачу якомога найкраще. В даному курсовому проекті представлена програма “П’ятнашка”.

Програмування – це процес написання, тестування і підтримки комп'ютерних програм. З розвитком програмування виникла ідея поєднати в межах однієї сутності дані і код, що безпосередньо опрацьовує ці дані. Така сутність отримала назву об’єкт, а відповідний підхід до створення програм називають об’єктно-орієнтованим програмуванням.

Об’єктно-орієнтоване програмування (ООП) – це парадигма програмування, яка розглядає програму як сукупність гнучко пов’язаних між собою об’єктів.

Основні переваги концепції ООП:

* можливість створювати користувацькі типи даних (класи);
* приховування деталей реалізації (інкапсуляція);
* можливість повторного використання коду (наслідування);

Метою даної роботи є створення гри. Для цього використовувалось середовище розробки –Eclips.

Git — розподілена система керування версіями файлів та спільної роботи. Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і

комітів. (Додаток А)

Обробка винятків (також опрацьовування виняткових ситуацій, англ. exception handling) — механізм мов програмування, призначений для обробки помилок часу виконання і інших можливих проблем (винятків), які можуть виникнути при виконанні програми.

**Swing** — інструментарій для створення графічного інтерфейсу користувача (GUI) мовою програмування Java. Це частина бібліотеки базових класів Java (*JFC*, Java Foundation Classes).

Swing розробляли для забезпечення функціональнішого набору програмних компонентів для створення графічного інтерфейсу користувача, ніж у ранішого інструментарію AWT. Компоненти Swing підтримують специфічні *look-and-feel* модулі, що динамічно підключаються. Завдяки ним можлива емуляція графічного інтерфейсу платформи (тобто до компоненту можна динамічно підключити інші, специфічні для даної операційної системи вигляд і поведінку). Основним недоліком таких компонентів є відносно повільна робота, хоча останнім часом це не вдалося підтвердити через зростання потужності персональних комп'ютерів. Позитивна сторона —— універсальність інтерфейсу створених програм на всіх платформах.

# АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

Дана програма створена для вирішення проблеми 8-ми пазлів. Середовище розробки програми: Eclipse. Програма написана на мові програмування Java, вона високого рівня з підтримкою декількох парадигм програмування: об’єктно-орієнованої, узагальненої та процедурної Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель С++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм.. Передусім, Java вводить класи, які забезпечують три найважливіші властивості ООП: інкапсуляцію, успадкування і поліморфізм.

Інкапсулювання – це механізм в програмуванні, який пов’язує в одне ціле функції і дані, якими вони маніпулють, а також захищає їх від зовнішнього доступу і неправильного застосування.

Успадкування – це властивість, з допомогою якої один об’єкт може набувати властивостей іншого. При цьому підтримується концепція ієрархічної класифікації.

Поліморфізм дозволяє писати більш абстрактні програми і підвищити коефіцієнт повторного використання коду.

Програма виконує:

* реалізації задачі у вигляді масиву із розмірністю N
* Взначає функції Хемінга(Hamming) та Манхетена(Manhattan)
* Перевіряє невирішувану ситуацію, обмінюючи дві сусідні клітинки в рядку
* Робить перевірку клітинок, що є сусідами до пустої клітинки.
* Визначає послідовність дошок при найменшій кількості зроблених кроків.

# 2 ОБґРУНТУВАННЯ АЛГОРИТМУ І СТРУКТУРИ ПРОГРАМИ

Клас — це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій.

Класи та функції які знаходяться в програмі розбиті на модулі. Модуль — функціонально завершений фрагмент програми, оформлений у вигляді окремого файлу, призначений для використання в інших програмах. Модулі дозволяють розбивати складні задачі на менші відповідно до принципу модульності.

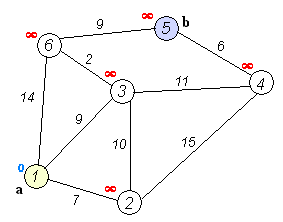
У програмі створено два класів. Клас Board та Solver.

**public class Board-** створений дляреалізації задачі у вигляді масиву із розмірністю N. Взначає функції Хемінга(Hamming) та Манхетена(Manhattan). Перевіряє невирішувану ситуацію, обмінюючи дві сусідні клітинки в рядку. Робить перевірку клітинок, що є сусідами до пустої клітинки.

**public class Solver –** створений для пошуку розв’язку початкової дошки (використовуючи алгоритм А\*). Видаляє попередньо досягнутий вузол, поки не досягнута кінцева дошка. Робить перевірку на те, чи можна розв’язати початкову дошку. Визначає послідовність дошок при найменшій кількості зроблених кроків. Цей клас також зчитує вхідні дані, і розв’язує саму задачу.

**Алгоритм пошуку А\***

У своїй програмі ми використовуємо даний алгоритм для пошуку найкоротшого шляху. Даний алгоритм зручно розглядати на прикладі графів.



**Рисунок 2.1 – Алгоритм пошуку А\***

**Алгоритм пошуку А\*** належить до [алгоритмів пошуку](http://uk.wikipedia.org/w/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B8_%D0%BF%D0%BE%D1%88%D1%83%D0%BA%D1%83&action=edit&redlink=1). У своїй програмі ми використовуємо даний алгоритм для пошуку найкоротшого шляху. Даний алгоритм зручно розглядати на прикладі графів.

Алгоритм повний в тому сенсі, що завжди знаходить оптимальний розв'язок, якщо він існує.

Алгоритм ділить вершини на *невідомі* вершини( ці вершини ще не були знайдені), *відомі* вершини (вже відомий шлях до цих вершин), *повністю досліджені* вершини (до цих вершин вже відомий найкоротший шлях).

Кожна відома або повністю досліджена вершина має вказівник на попередні вершини. Завдяки цьому вказівникові, можна пройти шляхом від цієї до початкової вершини.

Коли вершину  буде повністю досліджено суміжні з нею вершини додаються до списку відомих вершин, а сама вершина додається в список повністю досліджених.

Алгоритм зупиняється коли кінцева вершина потрапляє до списку повністю досліджених вершин. Знайдений шлях відтворюється із допомогою вказівників на попередню вершину. Якщо список відомих вершин порожніє, то розв'язку задачі не існує і алгоритм припиняє пошук.

Відтворений за зворотніми вказівниками знайдений шлях починається з кінцевої вершини та прямує до початкової. Аби одразу отримати шлях в правильному напрямі, з початкової вершини до кінцевої, в умовах задачі слід переставити місцями початок та кінець. Якщо шукати шлях починаючи з кінцевої вершини, відтворений список починатиметься з початкової вершини й прямуватиме до кінцевої.

Також у нашій програмі використовувались **черги з пріоритетом**. Черга увесь час підтримується упорядкованою, тобто кожен новий елемент включається на те місце в послідовності, що визначається його пріоритетом.

*Алгоритм пріоритетного включення*передбачає так дії:

– якщо черга неповна, знайти згідно з пріоритетом таке місце в черзі, де треба поставити запит;

– звільнити місце для запита, для чого зсунути всі елементи від цього місця до кінця черги на один елемент;

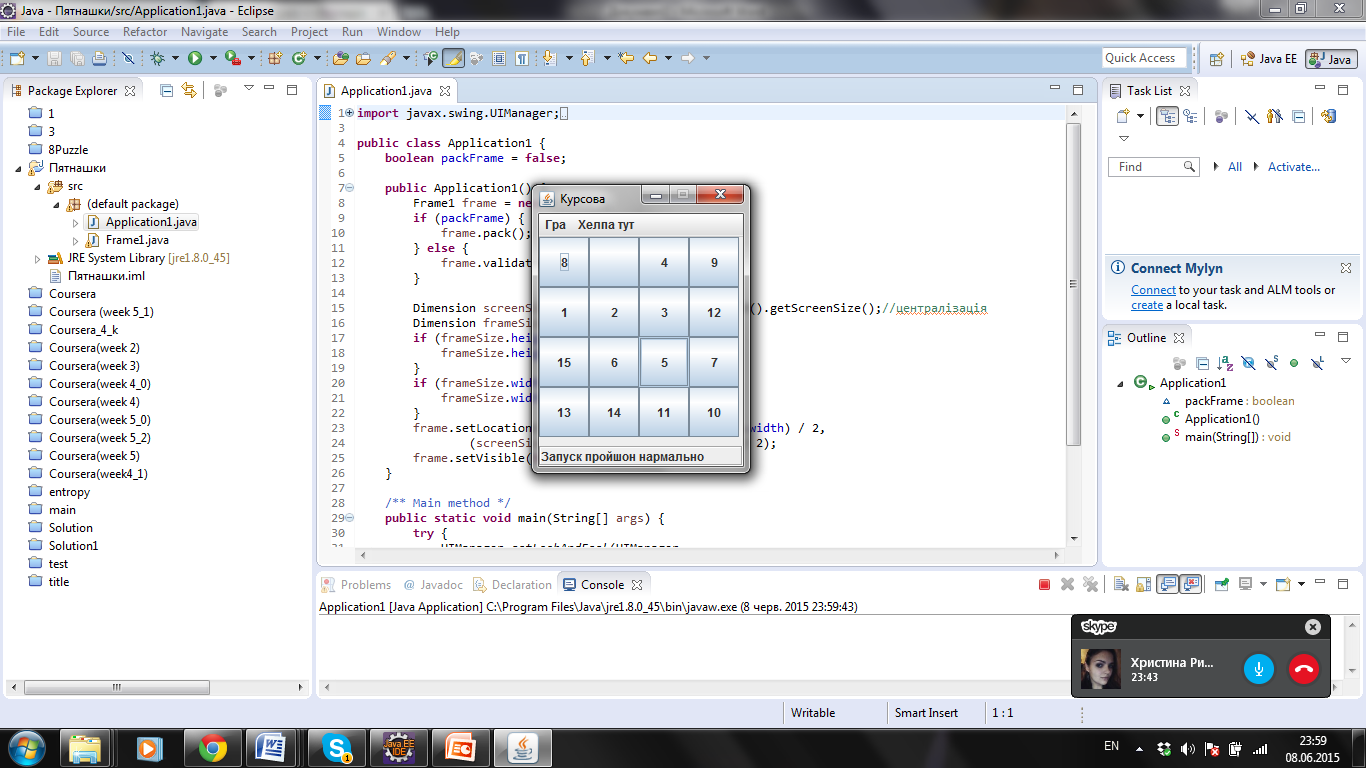
– внести запит до черги.

Для організації пріоритетної черги потрібен пошук і переміщення елементів. Однак важливо, при виборі елемента з пріоритетної черги кожного разу вибирається елемент із найбільшим (найменшим) пріоритетом.

Також до програми підключається текстовий файл даних puzzle1.txt для відповідної подальшої обробки цих результатів програмою.

Програма реалізує вікно, в якому можна побачити результати.

Також користувач може самостійно грати гру.



**Рисунок 2.2 – інтерфейс програми**

Сучасні програми потребують графічному інтерфейсі користувача (GUI) . Користувачі відвикли працювати через консоль : вони керують програмою і вводять вхідні дані за допомогою так званих елементів управління ( в програмуванні їх також називають візуальними компонентами ) , до яких відносяться кнопки, текстові поля, списки, що випадають і т.д.

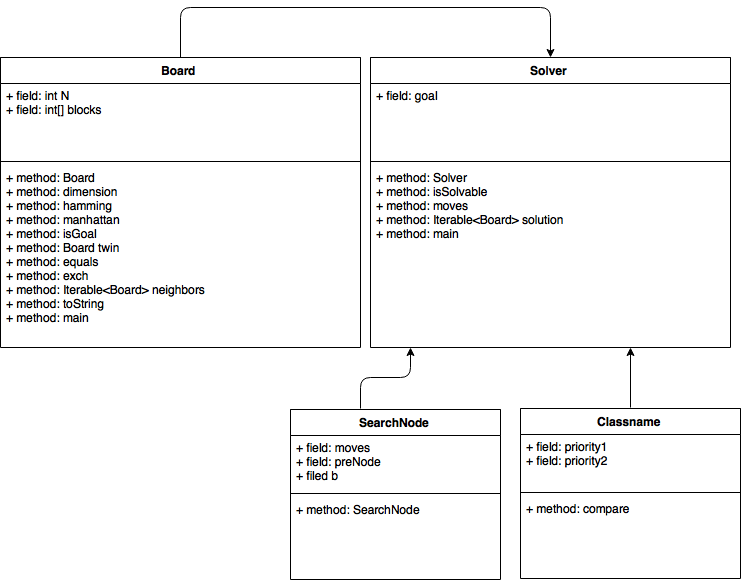
Кожен із сучасних мов програмування надає безліч бібліотек для роботи зі стандартним набором елементів управління. Нагадаємо, що під бібліотекою в програмуванні набір готових класів та інтерфейсів, призначених для вирішення певного кола завдань.

У Java є три бібліотеки візуальних компонентів для створення графічного інтерфейсу користувача. Найбільш рання з них називається AWT. Вважається, що при її проектуванні було допущено ряд недоліків, внаслідок яких з нею досить складно працювати. Бібліотека Swing розроблена на базі AWT і замінює більшість її компонентів своїми, спроектованими більш ретельно і зручно. Третя, найбільша нова бібліотека, називається SWT.

Кожна бібліотека надає набір класів для роботи з кнопками , списками , вікнами , меню і т.д., але ці класи спроектовані по- різному: вони мають різний набір методів з різними параметрами , тому « перевести » програму з однієї бібліотеки в іншу ( наприклад , з метою збільшення швидкодії ) не так -то просто.

Отже, наша програма складається із двох класів **Board та**  **Solver,** також до програми підключається текстовий файл даних puzzle1.txt для відповідної подальшої обробки цих результатів програмою. У програмі реалізований **Алгоритм пошуку А\*.** Також у нашій програмі використовувались **черги з пріоритетом**

# 3 РОЗРОБКА ПРОГРАМИ



**Рисунок 3.1 – Діаграма класів**

**public** **class** **Board** (Рис. 3.2) в створений дляреалізації задачі у вигляді масиву із розмірністю N. Взначає функції Хемінга(Hamming) та Манхетена(Manhattan). Перевіряє невирішувану ситуацію, обмінюючи дві сусідні клітинки в рядку. Робить перевірку клітинок, що є сусідами до пустої клітинки.

**Представлені такі методи:**

* Функція Хемінга(Hamming) – визначає кількість блоків, які не стоять на своїх місцях:

**Public** **int** hamming**()** **{**

**int** cnt **=** 0**;**

**for** **(int** i **=** 0**;** i **<** blocks**.**length**;** i**++)**

**if** **(**blocks**[**i**]** **!=** 0 **&&** blocks**[**i**]** **!=** i**+**1**)**

cnt**++;**

**return** cnt**;**

**}**

* Функція Манхетена(Manhattan) – сумує відстані між поточним блоком і місцем, де він повинен розміщуватися:

**public** **int** manhattan**()** **{**

**int** sum**;**

**int** row**,** col**;**

sum **=** 0**;**

**for** **(int** i **=** 0**;** i **<** blocks**.**length**;** i**++)** **{**

**if** **(**blocks**[**i**]** **!=** 0 **&&** blocks**[**i**]** **!=** i**+**1**)** **{**

col **=** Math**.**abs**(**i **%** N **-** **(**blocks**[**i**]-**1**)** **%** N**);**

row **=** Math**.**abs**(**i **/** N **-** **(**blocks**[**i**]-**1**)** **/** N**);**

sum **+=** **(**row **+** col**);**

**}**

**}**

**return** sum**;**

**}**

* Метод **public boolean isGoal()** робить перевірку поточної дошки, на те, чи є вона кінцевою дошкою:

**public** **boolean** isGoal**()** **{**

**for** **(int** i **=** 0**;** i **<** blocks**.**length**;** i**++)**

**if** **(**blocks**[**i**]** **!=** 0 **&&** blocks**[**i**]** **!=** i**+**1**)**

**return** **false;**

**return** **true;**

**}**

* Метод **Public Board twin()** визначає дошку, отриману шляхом обміну двох сусідніх блоків у тому ж рядку:

**public** Board twin**()** **{**

Board tBoard**;**

**if** **(**N **<=** 1**)** **return** **null;**

tBoard **=** **new** Board**(**blocks**);**

**if** **(**blocks**[**0**]** **!=** 0 **&&** blocks**[**1**]** **!=** 0**)**

exch**(**tBoard**.**blocks**,** 0**,** 1**);**

**else**

exch**(**tBoard**.**blocks**,** N**,** N**+**1**);**

**return** tBoard**;**

**}**

* Метод **public boolean equals(Object y)** визначає чи рівна поточна дошка до об’єкта ***у***:

**public** **boolean** equals**(**Object y**)** **{**

**if** **(**y **==** **null)** **return** **false;**

**if** **(**y **==** **this)** **return** **true;**

**if** **(**y**.**getClass**()** **!=** **this.**getClass**())**

**return** **false;**

Board yBoard **=** **(**Board**)** y**;**

**return** Arrays**.**equals**(**blocks**,** yBoard**.**blocks**);**

**}**

**private** **void** exch**(int[]** a**,** **int** i**,** **int** j**)** **{**

**int** temp **=** a**[**j**];**

a**[**j**]** **=** a**[**i**];**

a**[**i**]** **=** temp**;**

**}**

* Метод **boolean equals(Object y)** визначає усі дошки, які є сусідами даної.

**public** Iterable**<**Board**>** neighbors**()** **{**

**int** i**;**

Board neiBoard**;**

Queue**<**Board**>** bq **=** **new** Queue**<**Board**>();**

**for** **(**i **=** 0**;** i **<** blocks**.**length**;** i**++)**

**if** **(**blocks**[**i**]** **==** 0**)** **break;**

**if** **(**i **>=** blocks**.**length**)** **return** **null;**

**if** **(**i **>=** N**)** **{**

neiBoard **=** **new** Board**(**blocks**);**

exch**(**neiBoard**.**blocks**,** i**,** i**-**N**);**

bq**.**enqueue**(**neiBoard**);**

**}**

**if** **(**i **<** blocks**.**length **-** N**)** **{**

neiBoard **=** **new** Board**(**blocks**);**

exch**(**neiBoard**.**blocks**,** i**,** i**+**N**);**

bq**.**enqueue**(**neiBoard**);**

**}**

**if** **(**i **%** N **!=** 0**)** **{**

neiBoard **=** **new** Board**(**blocks**);**

exch**(**neiBoard**.**blocks**,** i**,** i**-**1**);**

bq**.**enqueue**(**neiBoard**);**

**}**

**if** **((**i**+**1**)** **%** N **!=** 0**)** **{**

neiBoard **=** **new** Board**(**blocks**);**

exch**(**neiBoard**.**blocks**,** i**,** i**+**1**);**

bq**.**enqueue**(**neiBoard**);**

**}**

**return** bq**;**

**}**

Метод **String toString()** відповідає запредставлення дошки у вигляді стрічок.

**public** String toString**()** **{**

StringBuilder s **=** **new** StringBuilder**();**

**int** digit **=** 0**;**

String format**;**

s**.**append**(**N**);**

s**.**append**(**"\n"**);**

**for** **(int** n **=** blocks**.**length**;** n **!=** 0**;** n **/=** 10**)**

digit**++;**

format **=** "%" **+** digit **+** "d "**;**

**for** **(int** i **=** 0**;** i **<** blocks**.**length**;** i**++)** **{**

s**.**append**(**String**.**format**(**format**,** blocks**[**i**]));**

**if** **((**i**+**1**)** **%** N **==** 0**)**

s**.**append**(**"\n"**);**

**}**

**return** s**.**toString**();**

**}**

**public class Solver –** створений для пошуку розв’язку початкової дошки (використовуючи алгоритм А\*). Видаляє попередньо досягнутий вузол, поки не досягнута кінцева дошка. Робить перевірку на те, чи можна розв’язати початкову дошку. Визначає послідовність дошок при найменшій кількості зроблених кроків. Цей клас також зчитує вхідні дані, і розв’язує саму задачу.

За допомогою Solver ми знаходимо рішення вихідної дошки , використовуючи алгоритм A\*.

**public** Solver(Board initial) {

PriorityOrder po = **new** PriorityOrder();

MinPQ<SearchNode> pq = **new** MinPQ<SearchNode>(po);

SearchNode sn = **new** SearchNode(initial);

PriorityOrder twinPo = **new** PriorityOrder();

MinPQ<SearchNode> twinPq = **new** MinPQ<SearchNode>(twinPo);

SearchNode twinSn = **new** SearchNode(initial.twin());

pq.insert(sn);

twinPq.insert(twinSn);

SearchNode minNode = pq.delMin();

SearchNode twinMinNode = twinPq.delMin();

**while** (!minNode.board.isGoal() && !twinMinNode.board.isGoal()) {

**for** (Board b : minNode.board.neighbors()) {

**if** ((minNode.preNode == **null**)

|| !b.equals(minNode.preNode.board)) {

SearchNode node = **new** SearchNode(b);

node.moves = minNode.moves + 1;

node.preNode = minNode;

pq.insert(node);

}

}

**for** (Board b : twinMinNode.board.neighbors()) {

**if** ((minNode.preNode == **null**)

|| !b.equals(twinMinNode.preNode.board)) {

SearchNode node = **new** SearchNode(b);

node.moves = twinMinNode.moves + 1;

node.preNode = twinMinNode;

twinPq.insert(node);

}

}

minNode = pq.delMin();

twinMinNode = twinPq.delMin();

}

**if** (minNode.board.isGoal())

goal = minNode;

**else**

goal = **null**;

}

Для алгоритму пошуку A\* ми використовуємо пріоритетну чергу.

Порядок вибірки елементів з таких черг визначається пріоритетами елементів. Пріоритет у загальному випадку може бути представлений значенням, що обчислюється, або на основі значень поля будь-якого елемента, або на основі зовнішніх факторів. Так, FІFO, і LІFO - черги трактуються як пріоритетні черги, у яких пріоритетом є час включення елемента в чергу.

Над пріоритетною чергою визначені наступні операції:

– включення елемента в кінець черги;

– включення елемента в чергу по пріоритету;

– виключення елемента з черги по пріоритету;

– виключення елемента з початку черги;

– визначення розміру черги;

– очищення черги.

При роботі з пріоритетними чергами частіше говорять не про елементи, а про запити, що записують у чергу. При цьому можлива організація черг із пріоритетним включенням елемента в чергу і з пріоритетним виключенням запиту з черги.

Для черги з пріоритетним включенням послідовність запитів у черзі увесь час підтримується упорядкованою, тобто кожен новий елемент включається на те місце в послідовності, що визначається його пріоритетом.

*Алгоритм пріоритетного включення*передбачає так дії:

– якщо черга неповна, знайти згідно з пріоритетом таке місце в черзі, де треба поставити запит;

– звільнити місце для запита, для чого зсунути всі елементи від цього місця до кінця черги на один елемент;

– внести запит до черги.

При виключенні завжди вибирається запит із початку черги. Зрозуміло, що при такому алгоритмі читання черга буде “переміщатися” в напрямку до кінця пам’яті, що виділена для черги, лишаючи вільне місце на її початку. Тому треба або переміщувати елементи черги на початок пам’яті, або використовувати кільцевий масив.

Для черги з пріоритетним виключенням новий елемент включається завжди в кінець черги, а при виключенні з черги відшукується потрібний запит.

*Алгоритм пріоритетного виключення* запиту такий:

– якщо черга непорожня, знайти за пріоритетом потрібний запит (цей пошук може бути тільки лінійним);

– прочитати запит;

–видалити запит з черги, для чого зрушити всі елементи черги від наступного до кінцевого на одну позицію ліворуч.

Як видно, і в одному, і в другому варіанті організації пріоритетної черги потрібен пошук і переміщення елементів. Як наслідок, у першому випадку час на постановку запиту в чергу більший ніж час читання і навпаки у другому. Тому, який алгоритм обрати, вирішується в кожному випадку окремо. Однак зауважимо, при виборі елемента з пріоритетної черги кожного разу вибирається елемент із найбільшим (найменшим) пріоритетом.

Також ми маємо метод moves(). Даний метод визначеє кількість ходів для вирішення проблеми(тобто для розміщення всіх елементів на своєму місці).

**public** **int** moves() {

**if** (goal == **null**) **return** -1;

**return** goal.moves;

}

Наступний метод у нашій програмі Iterable<Board> solution() .Він визначає послідовність пазлів у найкоротшому вирішенні.

**public** Iterable<Board> solution() {

**if** (!isSolvable()) **return** **null**;

Stack<Board> bStack = **new** Stack<Board>();

**for** (SearchNode s = goal; s != **null**; s = s.preNode)

bStack.push(s.board);

**return** bStack;

}

Метод **main()** виводить дані на екран.

**public** **static** **void** main(String[] args) {

In in = **new** In(args[0]);

**int** N = in.readInt();

**int**[][] blocks = **new** **int**[N][N];

**for** (**int** i = 0; i < N; i++)

**for** (**int** j = 0; j < N; j++)

blocks[i][j] = in.readInt();

Board initial = **new** Board(blocks);

Solver solver = **new** Solver(initial);

**if** (!solver.isSolvable())

StdOut.*println*("No solution possible");

**else** {

StdOut.*println*("Minimum number of moves = " + solver.moves());

**for** (Board board : solver.solution())

StdOut.*println*(board);

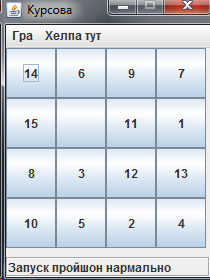
}

}

Також у даному методі ми використовуємо MinPQ.

# ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ

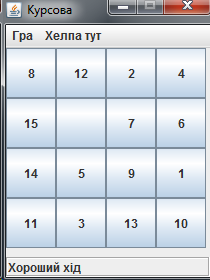
Наша програма має привітний користувацький графічний інтерфейс, зрозумілий користувачу. При початковому запуску програми ми бачимо у нижньому вікні програми повідомлення, що запуск програми пройшов успішно, і програма готова до роботи.



**Рисунок 4.1 Початкове вікно запуску програми**

Після запуску програма випадковим чином, завдяки підключеній бібліотеці “Math.random()”,)генерує цифри та вставляє їх у масив “matrix”, створений для ініціалізації цифр у кнопки. Для відображення клавіш на екрані створено спеціальний масив “buttons”.

Після того як програма створила ігрову дошку можна починати грати у гру. Пересування проводиться за допомогою лівої клавіші мишки. Даний процес відбувається завдяки методу “mousePressed”, який обробляє натиснення клавіші мишки на кнопку у програмі. Після пересування викликається метод “mouseReleased”, який реагує на відпускання миші гравцем. Після даної операції програма проводить пересування кнопки залежно від координат на задану гравцем позицію завдяки тому ж методу.

**

**Рисунок 4.2 Стан програми після пересування кнопки**

Для застереження нечесної гри гравця створена спеціальна функція, яка не дозволяє пересувати комірки у хаотичному порядку, а робить це лише у випадку коли поруч з коміркою, яку ми хочемо пересунути є пуста комірка. У випадку «мухлювання» програма виведе нам у статусі повідомлення, що ми граємо не чесно.

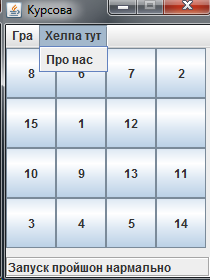


**Рисунок 4.3 – Спроба нечесної гри**

Для більш комфортної гри додано 2 меню. У Головному меню «Гра» ми можемо завершити гру, або ж за бажанням розпочати нову гру, якщо вам не сподобалася як програма розклала цифри на дошці, або ви зайли у момент коли не можете завершит игру. Дана функція реалізована завдяки методу “JMenubar”, який дозволяє додавати у програму стрічку з різними меню на якій ми вже створюємо самі випадаючі меню. Для виклику меню слід натиснути лівою клавішою мишки на назву меню.

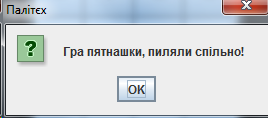


**Рисунок 4.4 Головне випадаюче меню програми**



**Рисунок 4.5 - Меню «Хелпа тут»**

Задля поширення нашої програми було вирішено додати інформацію про тих, хто робив програму.



**Рисунок 2.6 – Вікно «Про нас»**

# 

# ВИСНОВОК

В результаті виконання роботи було розроблено програму – гра “П’ятнашки” у середовищі розробки Eclipse, на мові програмування Java.

Дана програма надає користувачеві можливість грати інтелектуальну гру. Завдяки інтуїтивно зрозумілому інтерфейсу будь-який користувач не зіткнеться з жодними проблемами при її використанні. Користувач може самостійно пересувати клітинки,щоб розташувати їх від 1 до 15. Також можна задати дані системі у текстовому документі і вона самостійно розташує клітинки у правильному порядку.

Організовано роботу з файлами: створення файлу, зчитування файлу, зміни файлу та вивід даних на екран.

# 

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Орлов С. А. Технологии разработки программного обеспечения: Учебник. - СПб.: Питер, 2002. – 464 с. ISBN: 5-94723-145-X(рус.).
2. "Алгоритмы. Руководство по разработке" Стивен Скиена
3. Роберт Седжвик, Кевин Уэйн. Алгоритмы на Java, 4-е издание = Algorithms, 4th Edition. — М.: «Вильямс», 2012. — 848 с. — ISBN 978-5-8459-1781-2.

# 

# ДОДАТОК А

Лістинг class Board

**import** java.util.Arrays;

**public** **class** Board {

**private** **int** N;

**private** **int**[] blocks;

**public** Board(**int**[][] blocks) {

N = blocks[0].length;

**this**.blocks = **new** **int**[N\*N];

**for** (**int** i = 0; i < N; i++)

**for** (**int** j = 0; j < N; j++)

**this**.blocks[i\*N+j] = blocks[i][j];

}

**private** Board(**int**[] blocks) {

N = (**int**) Math.*sqrt*(blocks.length);

**this**.blocks = **new** **int**[blocks.length];

**for** (**int** i = 0; i < blocks.length; i++)

**this**.blocks[i] = blocks[i];

}

**public** **int** dimension() {

**return** N;

}

**public** **int** hamming() {

**int** cnt = 0;

**for** (**int** i = 0; i < blocks.length; i++)

**if** (blocks[i] != 0 && blocks[i] != i+1)

cnt++;

**return** cnt;

}

**public** **int** manhattan() {

**int** sum;

**int** row, col;

sum = 0;

**for** (**int** i = 0; i < blocks.length; i++) {

**if** (blocks[i] != 0 && blocks[i] != i+1) {

col = Math.*abs*(i % N - (blocks[i]-1) % N);

row = Math.*abs*(i / N - (blocks[i]-1) / N);

sum += (row + col);

}

}

**return** sum;

}

**public** **boolean** isGoal() {

**for** (**int** i = 0; i < blocks.length; i++)

**if** (blocks[i] != 0 && blocks[i] != i+1)

**return** **false**;

**return** **true**;

}

**public** Board twin() {

Board tBoard;

**if** (N <= 1) **return** **null**;

tBoard = **new** Board(blocks);

**if** (blocks[0] != 0 && blocks[1] != 0)

exch(tBoard.blocks, 0, 1);

**else**

exch(tBoard.blocks, N, N+1);

**return** tBoard;

}

**public** **boolean** equals(Object y) {

**if** (y == **null**) **return** **false**;

**if** (y == **this**) **return** **true**;

**if** (y.getClass() != **this**.getClass())

**return** **false**;

Board yBoard = (Board) y;

**return** Arrays.*equals*(blocks, yBoard.blocks);

}

**private** **void** exch(**int**[] a, **int** i, **int** j) {

**int** temp = a[j];

a[j] = a[i];

a[i] = temp;

}

**public** Iterable<Board> neighbors() {

**int** i;

Board neiBoard;

Queue<Board> bq = **new** Queue<Board>();

**for** (i = 0; i < blocks.length; i++)

**if** (blocks[i] == 0) **break**;

**if** (i >= blocks.length) **return** **null**;

**if** (i >= N) {

/\* вгору \*/

neiBoard = **new** Board(blocks);

exch(neiBoard.blocks, i, i-N);

bq.enqueue(neiBoard);

}

**if** (i < blocks.length - N) {

/\* вниз \*/

neiBoard = **new** Board(blocks);

exch(neiBoard.blocks, i, i+N);

bq.enqueue(neiBoard);

}

**if** (i % N != 0) {

/\* вліво \*/

neiBoard = **new** Board(blocks);

exch(neiBoard.blocks, i, i-1);

bq.enqueue(neiBoard);

}

**if** ((i+1) % N != 0) {

/\* вправо \*/

neiBoard = **new** Board(blocks);

exch(neiBoard.blocks, i, i+1);

bq.enqueue(neiBoard);

}

**return** bq;

}

**public** String toString() {

StringBuilder s = **new** StringBuilder();

**int** digit = 0;

String format;

s.append(N);

s.append("\n");

**for** (**int** n = blocks.length; n != 0; n /= 10)

digit++;

format = "%" + digit + "d ";

**for** (**int** i = 0; i < blocks.length; i++) {

s.append(String.*format*(format, blocks[i]));

**if** ((i+1) % N == 0)

s.append("\n");

}

**return** s.toString();

}

**public** **static** **void** main(String[] args) {

In in = **new** In(args[0]);

**int** N = in.readInt();

**int**[][] blocks = **new** **int**[N][N];

**for** (**int** i = 0; i < N; i++)

**for** (**int** j = 0; j < N; j++)

blocks[i][j] = in.readInt();

Board initial = **new** Board(blocks);

StdOut.*print*(initial.toString());

StdOut.*print*(initial.twin().toString());

StdOut.*println*(initial.hamming());

StdOut.*println*(initial.manhattan());

StdOut.*println*(initial.dimension());

StdOut.*println*(initial.isGoal());

**for** (Board b : initial.neighbors()) {

StdOut.*println*(b.toString());

**for** (Board d : b.neighbors()) {

StdOut.*println*("===========");

StdOut.*println*(d.toString());

StdOut.*println*("===========");

}

}

}

}

Лістинг class Solver

**import** java.util.Comparator;

**public** **class** Solver {

**private** SearchNode goal;

**private** **class** SearchNode {

**private** **int** moves;

**private** Board board;

**private** SearchNode preNode;

**public** SearchNode(Board b) {

moves = 0;

preNode = **null**;

board = b;

}

}

**public** Solver(Board initial) {

PriorityOrder po = **new** PriorityOrder();

MinPQ<SearchNode> pq = **new** MinPQ<SearchNode>(po);

SearchNode sn = **new** SearchNode(initial);

PriorityOrder twinPo = **new** PriorityOrder();

MinPQ<SearchNode> twinPq = **new** MinPQ<SearchNode>(twinPo);

SearchNode twinSn = **new** SearchNode(initial.twin());

pq.insert(sn);

twinPq.insert(twinSn);

SearchNode minNode = pq.delMin();

SearchNode twinMinNode = twinPq.delMin();

**while** (!minNode.board.isGoal() && !twinMinNode.board.isGoal()) {

**for** (Board b : minNode.board.neighbors()) {

**if** ((minNode.preNode == **null**)

|| !b.equals(minNode.preNode.board)) {

SearchNode node = **new** SearchNode(b);

node.moves = minNode.moves + 1;

node.preNode = minNode;

pq.insert(node);

}

}

**for** (Board b : twinMinNode.board.neighbors()) {

**if** ((minNode.preNode == **null**)

|| !b.equals(twinMinNode.preNode.board)) {

SearchNode node = **new** SearchNode(b);

node.moves = twinMinNode.moves + 1;

node.preNode = twinMinNode;

twinPq.insert(node);

}

}

minNode = pq.delMin();

twinMinNode = twinPq.delMin();

}

**if** (minNode.board.isGoal())

goal = minNode;

**else**

goal = **null**;

}

**private** **class** PriorityOrder **implements** Comparator<SearchNode> {

**public** **int** compare(SearchNode s1, SearchNode s2) {

**int** priority1 = s1.board.manhattan() + s1.moves;

**int** priority2 = s2.board.manhattan() + s2.moves;

**if** (priority1 > priority2) **return** 1;

**else** **if** (priority1 < priority2) **return** -1;

**else** **return** 0;

}

}

**public** **boolean** isSolvable() {

**return** goal != **null**;

}

**public** **int** moves() {

**if** (goal == **null**) **return** -1;

**return** goal.moves;

}

**public** Iterable<Board> solution() {

**if** (!isSolvable()) **return** **null**;

Stack<Board> bStack = **new** Stack<Board>();

**for** (SearchNode s = goal; s != **null**; s = s.preNode)

bStack.push(s.board);

**return** bStack;

}

**public** **static** **void** main(String[] args) {

In in = **new** In(args[0]);

**int** N = in.readInt();

**int**[][] blocks = **new** **int**[N][N];

**for** (**int** i = 0; i < N; i++)

**for** (**int** j = 0; j < N; j++)

blocks[i][j] = in.readInt();

Board initial = **new** Board(blocks);

Solver solver = **new** Solver(initial);

**if** (!solver.isSolvable())

StdOut.*println*("No solution possible");

**else** {

StdOut.*println*("Minimum number of moves = " + solver.moves());

**for** (Board board : solver.solution())

StdOut.*println*(board);

}

}

}

Лістинг class Application1

import javax.swing.UIManager;

import java.awt.\*;

public class Application1 {

boolean packFrame = false;

public Application1() {

Frame1 frame = new Frame1();

if (packFrame) {

frame.pack();

} else {

frame.validate();

}

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();//централізація

Dimension frameSize = frame.getSize();

if (frameSize.height > screenSize.height) {

frameSize.height = screenSize.height;

}

if (frameSize.width > screenSize.width) {

frameSize.width = screenSize.width;

}

frame.setLocation((screenSize.width - frameSize.width) / 2,

(screenSize.height - frameSize.height) / 2);

frame.setVisible(true);

}

/\*\* Main method \*/

public static void main(String[] args) {

try {

UIManager.setLookAndFeel(UIManager

.getCrossPlatformLookAndFeelClassName());

} catch (Exception e) {

e.printStackTrace();

}

new Application1();

}

}

Лістинг class Frame1

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class Frame1 extends JFrame {

JPanel contentPane;

JPanel gamePane = new JPanel();

BorderLayout borderLayout1 = new BorderLayout();

JLabel status = new JLabel();

JButton[][] buttons = new JButton[4][4];

int[][] matrix = new int[4][4];

public Frame1() {

enableEvents(AWTEvent.WINDOW\_EVENT\_MASK); //проба запуску події

try {

jbInit();

status.setText("Запуск пройшон нармально");

} catch (Exception exeption) {

exeption.printStackTrace();

}

}

private void jbInit() throws Exception {

contentPane = (JPanel) this.getContentPane();

contentPane.setLayout(borderLayout1);

this.setSize(new Dimension(210, 280));

this.setTitle("Курсова");

this.setResizable(false);

JMenuBar menuBar = new JMenuBar();

JMenu menuGame = new JMenu("Гра");

JMenu menuHelp = new JMenu("Хелпа тут");

menuBar.add(menuGame);

menuBar.add(menuHelp);

this.setJMenuBar(menuBar);

JMenuItem newGameButton = new JMenuItem("Нова гра");

JMenuItem exitButton = new JMenuItem("Вихід");

JMenuItem aboutButton = new JMenuItem("Про нас");

newGameButton.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(ActionEvent e) {

newGame();

}

});

exitButton.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(ActionEvent e) {

System.exit(0);//вихід

}

});

aboutButton.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(ActionEvent e) {JOptionPane.showMessageDialog(null,"Гра пятнашки, пиляли спільно!","Палітєх", JOptionPane.QUESTION\_MESSAGE);

}

});

menuGame.add(newGameButton);

menuGame.add(exitButton);

menuHelp.add(aboutButton);

gamePane.setLayout(null);

int count = 0;

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

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

buttons[i][j] = new JButton("" + count);

buttons[i][j].addMouseListener(new mAdapter(i, j));

buttons[i][j].setSize(50, 50);

buttons[i][j].setLocation(50 \* j, 50 \* i);

buttons[i][j].setCursor(new Cursor(Cursor.HAND\_CURSOR));

gamePane.add(buttons[i][j]);

matrix[i][j] = count;

count++;

}

buttons[0][0].setText(" ");

contentPane.add(gamePane, BorderLayout.CENTER); //панель з кнопками додат

status.setBorder(BorderFactory.createEtchedBorder()); // статус стрічка рамка, обрамлення

contentPane.add(status, BorderLayout.SOUTH); //стрічка статусу

newGame();

}

protected void processWindowEvent(WindowEvent e) {

super.processWindowEvent(e);

if (e.getID() == WindowEvent.WINDOW\_CLOSING) {

System.exit(0);

}

}

public void randomizeMatrix() { //випадкове заповнення

for (int i = 0; i < 100; i++) {

int a = (int) (Math.random() \* 4);

int b = (int) (Math.random() \* 4);

int a2 = (int) (Math.random() \* 4);

int b2 = (int) (Math.random() \* 4);

int c = matrix[a][b];

matrix[a][b] = matrix[a2][b2];

matrix[a2][b2] = c;

}

}

public void newGame() {

randomizeMatrix();

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

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

if (matrix[i][j] != 0)

buttons[i][j].setText("" + matrix[i][j]);

else

buttons[i][j].setText("");

}

status.setText("Нова гра почата");

gamePane.setVisible(true);

}

class mAdapter extends java.awt.event.MouseAdapter {

int posi, posj;

int startx = 0;

int starty = 0;

mAdapter(int posI, int posJ) { //конструктор кнопок

this.posi = posI;

this.posj = posJ;

}

public void mousePressed(MouseEvent e) { //натиснені клавіші

buttons[posi][posj].setCursor(new Cursor(Cursor.MOVE\_CURSOR));

startx = e.getX();

starty = e.getY();

}

public void mouseReleased(MouseEvent event) { // це при відпусканні

buttons[posi][posj].setCursor(new Cursor(Cursor.HAND\_CURSOR));

int endx = event.getX();

int endy = event.getY();

int shiftx = endx - startx;

int shifty = endy - starty;

if (Math.abs(shiftx) > Math.abs(shifty)) {

if (shiftx > 0) {// вправо

if ((posj != 3) && (matrix[posi][posj + 1] == 0)) {

matrix[posi][posj + 1] = matrix[posi][posj];

matrix[posi][posj] = 0;

buttons[posi][posj].setText("");

buttons[posi][posj + 1].setText(""

+ matrix[posi][posj + 1]);

status.setText("Хороший хід");

} else

status.setText("не мухлюй");

} else {// вліво

if ((posj != 0) && (matrix[posi][posj - 1] == 0)) {

matrix[posi][posj - 1] = matrix[posi][posj];

matrix[posi][posj] = 0;

buttons[posi][posj].setText("");

buttons[posi][posj - 1].setText(""

+ matrix[posi][posj - 1]);

status.setText("Хороший хід");

} else

status.setText("не мухлюй");

}

} else {

//по осі y

if (shifty > 0) {// вниз

if ((posi != 3) && (matrix[posi + 1][posj] == 0)) {

matrix[posi + 1][posj] = matrix[posi][posj];

matrix[posi][posj] = 0;

buttons[posi][posj].setText("");

buttons[posi + 1][posj].setText(""

+ matrix[posi + 1][posj]);

status.setText("Хороший хід");

} else

status.setText("не мухлюй");

} else {// вверх

if ((posi != 0) && (matrix[posi - 1][posj] == 0)) {

matrix[posi - 1][posj] = matrix[posi][posj];

matrix[posi][posj] = 0;

buttons[posi][posj].setText("");

buttons[posi - 1][posj].setText(""

+ matrix[posi - 1][posj]);

status.setText("Хороший хід");

} else

status.setText("не мухлюй");

}

}

int count = 1;

int error = 0;

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

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

if (matrix[i][j] != count)

error++;

count++;

}

if (error == 1) {

status.setText("Ви перемогли");

int result = JOptionPane.showConfirmDialog(null, "Ви перемогли",

"Ще партійку?", JOptionPane.YES\_NO\_OPTION);

if (result == JOptionPane.YES\_OPTION)

newGame(); //тут тіпа початок якшошо

else

gamePane.setVisible(false); // тут тіпа хепі енд

}

} }}