# Опис алгоритмів

Перелік всіх основних змінних та їхнє призначення наведено в таблиці 3.1.

Таблиця 3.1 – Основні змінні та їхні призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
|  |  |
| positions | Масив координат стовпців |
| searchTree | Дерево станів системи |
| solutionStatus | Ознака вирішеності задачі (false –задачу не вирішено,  true – задачу вирішено) |

## Загальний алгоритм

1. ПОЧАТОК
2. Отримати інформацію про вибір методу вирішення, заданий користувачем (**solvingMethod**)
3. Зчитати поточне положення фігур на шаховій дошці (**matrixPositions**)
4. Перевірити правильність розстановки фігур **solutionStatus** відповідно до критерію вирішення задачі
   1. ЯКЩО фігури розставлені правильно (**solutionStatus** == true), ТО вивести відповідне повідомлення та завершити роботу програми
   2. ІНАКШЕ вирішити задачу програмно
5. Перевірити чи кожна фігура займає лише один рядок **checkRows**

5.1. ЯКЩО декілька фігур займають один рядок, ТО змінити їх положення так, щоб кожна королева займала окремий рядок (**rowsInsert(matrixPositions)**)

5.2. Записати координати фігур у вектор **positions** (**matrixToVector(matrixPositions)**)

6. Побудувати searchTree

6.1. Ініціалізувати rootNode: rootNode = currentState

6.2 Створити нащадків rootNode: rootNode.children – усі можливі перестановки 1-ї фігури

6.2.1. queenIndex = 0

6.2.2. freePositions = [0,1,2,3,4,5,6,7] /// should I coppy current state?

6.2.3.generateChildren(rootNode, queenIndex, freePositions)

1. Почати пошук рішення searchSolution обраним алгоритмом **solvingMethod**
2. Зупинити роботу програми при знайденому рішенні
   1. ІНАКШЕ вивести повідомлення про помилку
3. КІНЕЦЬ

* rowsInsert(matrixPositions)
* findFreePositions (parentNode, index of queen to be placed)

1. freePositions = [0, 1, 2, 3, 4, 5, 6, 7]
2. Отримати інформацію про parentNode
3. Отримати індекс фігури, яку буде переставлено - **currentQueenIndex**
4. Для заданої фігури знайти PositionsUnderAttack = [qN.pos + (**currentQueenIndex** – qN.index)], де **N** – усі цілі числа від 0 до **currentQueenIndex** (qN.index – порядковий номер у векторі розстановки, qN.pos – координата, що знаходиться за даним порядковим номером)
5. Видалити positionsUnderAttack із freePositions
6. Повернути freePositions

* generateChildren (сurrentStateNode, queenIndex, freePositions)

1. Для усіх positionNumber у freePositions
   1. newStateInfo = currentStateNode.stateInfo;
   2. newStateInfo[queenIndex] = positionNumber;
   3. Node newState = newNode (parent –> currentStateNode, stateInfo -> newStateInfo);
   4. currentStateNode.children.add(newState);

* checkState(currentState)

перевірити чи фігури знаходяться у одному стовбці solutionStatus = checkColumns(currentState)

ЯКЩО (solutionStatus == true)

перевірити чи фігури знаходяться на одній діагоналі solutionStatus = checkDiagonals(currentState)

ВСЕ ЯКЩО

Повернути solutionStatus

* checkColumns(currentState)
* ПОЧАТОК

ДЛЯ УСІХ і від 0 до 8 ПОВТОРИТИ

ДЛЯ УСІХ j від і+1 до 8 ПОВТОРИТИ

ЯКЩО (currentState[i] == currentState[j]

Повернути false

ВСЕ ЯКЩО

ВСЕ ПОВТОРИТИ

ВСЕ ПОВТОРИТИ

Повернути true

КІНЕЦЬ

* checkDiagonals(currentState)

ПОЧАТОК

ДЛЯ УСІХ і від 0 до 8 ПОВТОРИТИ

checkedItem = [i, currentState[i]]

ДЛЯ УСІХ j від і+1 до 8 ПОВТОРИТИ

comparison = [j, currentState[j]]

ЯКЩО (| checkedItem[0] - comparison[0] | ==

| checkedItem[1] – comparison[1] |)

Повернути false

ВСЕ ЯКЩO

ВСЕ ПОВТОРИТИ

ВСЕ ПОВТОРИТИ

Повернути true

КІНЕЦЬ

* BFS

**BFS (root)**

Нехай Q – черга

Позначити root як пройдений

Додати root до Q

**Повторити** поки Q не порожня

v: = останній елемент черги

**Якщо** checkState(v.stateInfo) == true, то

Повернути шлях від root до v

**Все якщо**

**generateChildren(v)**

**Повторити** child y v.children

**Якщо** child не є дослідженою, то

Позначити child як досліджену

Додати child до Q

**Все якщо**

**Все повторити**

**Все повторити**

* IDS

**Функція IDS(node, goal)**

**Повторити для** l = 0 до

result = **LDFS(node, goal, l)**

**Якщо** (result != cutoff) **то**

**Повернути** result

**Все якщо**

**Все повторити**

* LDFS

**Функція** LDFS (node, max\_depth) **повертає** result (шлях від *node* до *goal*) **або** failure/ cutoff (індикатор невдачі)

**Якщо** (!cutoff\_occured)

**Якщо** ( max\_depth >= 0) **то**

**Додати node до solutionPath**

**Якщо** (checkState(node) == true) **то**

**Повернути** solutionPath

**Інакше якщо (**Depth[node] == l) **то**

**Повернути** cutoff

**Інакше**

**generateChildren(node)**

**Повторити** для усіх children вузла node

result = **LDFS (v, l - 1)**

**Якщо** (result = cutoff)

cutoff\_occured = true

**Інакше якщо** (result != failure)

**Повернути** result

**Все якщо**

**Все повторити**

**Все якщо**

**Все якщо**

**Все якщо**

**Якщо** (cutoff\_occured) **то**

**Повернути** cutoff

**Інакше**

**Повернути** failure

**Все якщо**