**Instruction for codes**

My code is made of Java language and there are 3 files. Each java file has each role such as PuzzleApplication for main method, Solutions for implementing A search\*, and HeuristicConstraints for offering heuristics for A\*. In this paper, an initial and goal state are defined as figure 1.

A math equation on a black background

AI-generated content may be incorrect.

Figure 1: intial state and goal state.

The process of finding the solution for 8 puzzle problem is shown with program code step by step as follows

1. **User runs program**:

Command: java PuzzleApplication

↓

PuzzleApplication.main(String[] args) // Initial point

* Begins while loop
* Calls setStrategy() method

1. **Menu to choose heuristic**:

PuzzleApplication.setStrategy() method

* Show users choics:

1: A\* search (Misplaced tile) or 2: A\* search (Manhattan)

* User choose 1 or 2

A black background with white text

AI-generated content may be incorrect.

Figure 2: The choice of heuristic types

↓

Starts:

// PuzzleApplication.java // define initial state and goal state.

goal = new Node(goalPuzzle)

initialNode = new Node (initialPuzzle)

↓

Relying on user selection:

// Both methods are in HeuristicConstraints

* 1 -> chooseHeuristics = AstarTileCalculator
* 2 -> chooseHeuristics = ManhattanCalculator

↓

Calls:

Solutions.costSearch(initialNode, chosenHeuristics) // defined in Solutions.java

1. **A\* Search Execution**:

Solution.costSearch(Node root, HeurType heurType)

* Starts:

openList = PriorityQueue<Node> // Nodes ordered by f = g+h

explored = Hashset<String> // closeList

root.heuristicCost = HeuristicConstraints.getCost()

root.totalCost = root.heuristicCost

openList.add(root)

* Iteration loop:

while (openList not empty):

current = openlist.poll() // p/u lowest f node

if current.isGoalNode(goal):

printSolution(current) // goal state

return

explored.add(current.toString())

current.expand() // defined in Node.java

* Generates successors (makeMove)
* Each successor.parentState = current
* Successor.cost = tile moved

For each successor:

successor.heuristicCost = HeuristicConstraints.getCost()

successor.totalCost = current.totalCost + successor.cost + heuristicCost

openList.add(successor)

1. **Heuristic computation**:

HeuristicConstraints (Node node, Node goal, HeurType type) // Constructor

* If user inputs =1 -> misplacedTileCounter()
* If user inputs =2 -> manhattanDistance()
* Return h(n) to Solutions.costSearch

1. **Node expansion (successors)**

Node.expand()

* Searches for blank tile index
* Calls makemove(Direction, state, blankIndex)
* Confirm bounds
* Swaps tiles
* Generates successor Node
* Adds to successors list

A screen shot of a computer

AI-generated content may be incorrect.

Figure 3: Direction,node depth & cost, total cost, and heuristic estimation(misplaced tile)

1. **Solution arrived**

Solutions.PrintResults()

* Backtrack using parentState
* Shows each Node:
* Overall statistics
* Iterations
* Time it took (in ms)
* Length
* Total cost
* Number of nodes dequeued
* Space

A black background with white text

AI-generated content may be incorrect.

Figure 4: The results of A\* search (misplaced tile)

1. **Go back to menu**

PuzzleApplication.main()

* Asks user: 0 or 1
* Loop or finish program

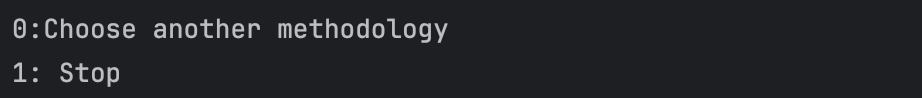


Figure 5: Choices of next steps for user.