# 15 puzzle

The **15-puzzle** (also called **Gem Puzzle**, **Boss Puzzle**, **Game of Fifteen**, **Mystic Square** and many others) is a sliding puzzle that consists of a frame of numbered square tiles in random order with one tile missing. The puzzle also exists in other sizes, particularly the smaller **8-puzzle**. If the size is 3×3 tiles, the puzzle is called the 8-puzzle or 9-puzzle, and if 4×4 tiles, the puzzle is called the 15-puzzle or 16-puzzle named, respectively, for the number of tiles and the number of spaces. The object of the puzzle is to place the tiles in order (see diagram) by making sliding moves that use the empty space.


The *n*-puzzle is a classical problem for modelling algorithms involving heuristics. Commonly used heuristics for this problem include counting the number of misplaced tiles and finding the sum of the taxicab distances between each block and its position in the goal configuration. Note that both are *admissible*, i.e. they never overestimate the number of moves left, which ensures optimality for certain search algorithms such as A*.

**Best-first search.** Now, we describe a solution to the problem that illustrates a general artificial intelligence methodology known as the A* search algorithm. We define a *search node* of the game to be a board, the number of moves made to reach the board, and the previous search node. First, insert the initial search node (the initial board, 0 moves, and a null previous search node) into a priority queue. Then, delete from the priority queue the search node with the minimum priority, and insert onto the priority queue all neighboring search nodes (those that can be reached in one move from the dequeued search node). Repeat this procedure until the search node dequeued corresponds to a goal board. The success of this approach hinges on the choice of *priority function* for a search node. We consider two priority functions:

- *Hamming priority function.* The number of blocks in the wrong position, plus the number of moves made so far to get to the search node. Intuitively, a search node with a small number of blocks in the wrong position is close to the goal, and we prefer a search node that have been reached using a small number of moves.
- *Manhattan priority function.* The sum of the Manhattan distances (sum of the vertical and horizontal distance) from the blocks to their goal positions, plus the number of moves made so far to get to the search node.

**A critical optimization.** Best-first search has one annoying feature: search nodes corresponding to the same board are enqueued on the priority queue many times. To reduce unnecessary exploration of useless search nodes, when considering the neighbors of a search node, don't enqueue a neighbor if its board is the same as the board of the previous search node.

**Detecting unsolvable puzzles.**

To detect such situations, use the fact that boards are divided into two equivalence classes with respect to reachability: (i) those that lead to the goal board and (ii) those that lead to the goal board if we modify the initial board by swapping any pair of blocks (the blank square is not a block). (Difficult challenge for the mathematically inclined: prove this fact.) To apply the fact, run the A* algorithm on *two* puzzle instances—one with the initial board and one with the initial board modified by swapping a pair of blocks—in lockstep (alternating back and forth between exploring search nodes in each of the two game trees). Exactly one of the two will lead to the goal board.