

Temple Trap Puzzle Solver Report

Author: Aaron Jason Baptist

Date: November 2025

Overview

This project implements an **AI-based solver** for the **Temple Trap puzzle** using the **A* search algorithm**. The puzzle is represented as a 3×3 grid of tiles, where a pawn must navigate through openings, holes, and stairs to reach the exit. The solver automatically finds a sequence of moves (pawn + tile slides) that leads to the solution.

Objective

The goal is to find the minimum-cost path for the pawn to reach the exit (cell 0 with an open left side) by:

- Moving through connected openings on the same layer.
- Using stairs to switch layers.
- Standing on the tiles having holes.
- Sliding tiles into the blank space to open new routes.

Puzzle Representation

Each tile is represented as an object with:

- **Top openings** and **Ground openings** – indicate directions open for movement.
- **Holes** – tiles on which the pawn can stand.
- **Stairs** – allow switching between top and ground layers.
- **Symbol** – used for move visualization.

The board configuration is stored as a list of 9 tiles (3×3 grid), with a space “ ” denoting the blank cell.

Approach

1. State Representation

Each state stores:

- Current board layout (`board`)
- Pawn position (`pawn_pos`)
- Pawn layer (`pawn_layer`)

2. Reachability

`reachable_cell()` uses **Breadth-First Search (BFS)** to find all positions the pawn can move to from the current state, considering openings, stairs, and tiles with holes where it can land.

3. Child Generation

`get_children()` generates all possible next states:

- Moving the pawn to reachable tiles which have a hole.
- Sliding a tile into the blank position.

4. Heuristic Function

`heuristic()` estimates the minimum remaining cost to reach the goal state. It computes the Manhattan distance between the pawn's current position and the goal cell (0,0) and adds an additional penalty of 1 if the tile at (0,0) does not have a left-side opening.

This heuristic is admissible because:

- The pawn must make at least as many moves as the Manhattan distance to reach the exit.
- At least one tile slide is required if the exit cell lacks a left opening.

Thus, it never overestimates the true cost, maintaining optimality while reducing explored states.

Results Summary

The table below compares the number of nodes explored using the A* search (with heuristic) versus the uniform-cost search (without heuristic) for all tested Temple Trap configurations.

Test Case	Nodes Explored (A*)	Nodes Explored (No Heuristic)	Total Cost
1	96	156	11
2	37	43	9
3	167	182	10
4	146	230	12
5	310	477	14
6	406	476	15
7	334	432	15
8	732	982	17
9	186	214	12
10	134	203	14
11	389	471	15
12	254	365	14

Observations

- In every case, the heuristic-based A* algorithm explored fewer nodes while maintaining the same optimal path cost.
- The reduction in explored states ranges from **10% to 40%**, depending on puzzle complexity.
- This confirms that the implemented heuristic is **admissible** and effectively improves search efficiency.

5. A* Search Algorithm

The main solver (`solve()`) implements the **A* Search Algorithm**: A priority queue (min-heap) is used to always expand the most promising node — the one with the lowest estimated total cost:

$$f(n) = g(n) + h(n)$$

Where:

- $g(n)$ = Number of moves so far.
- $h(n)$ = Heuristic = Manhattan distance + exit penalty.

This ensures efficient exploration while guaranteeing an optimal solution.

6. Goal Check

`goal_reachable()` verifies if the pawn can reach the exit at cell 0 with an open left side.

Example Input

```
initial_board = ['C', 'D', 'G', 'B', ' ', 'H', 'A', 'E', 'F']
rotations     = [0, 0, 2, 3, 0, 1, 0, 0, 2]
pawn_pos      = 8
pawn_floor    = 'ground'
```

Instructions to Run

1. **Input the initial state of the board:** Provide an array representing the tiles placed on the 3×3 board. The index i of the array corresponds to the tile type placed at that location (e.g., ‘A’, ‘B’, etc.).
2. **Specify the rotations of the tiles:** Input another array that specifies the rotation of each tile in the initial state. The rotation is defined as the number of **anticlockwise turns** made from the original tile orientation (where the symbol is initially at the bottom-left corner).
 - The array must be of length 9, where each index i represents the rotation of the respective tile at position i .
 - If the tile is not rotated, then `rotation[i] = 0`.

- Valid rotation values range from 0 to 3.
 - The blank position (representing the empty cell) is always assigned a rotation value of 0.
3. **Specify the pawn's initial position and layer:** Provide the pawn's starting cell index and the layer it is currently on ('ground' or 'top'). For tiles containing stairs, the pawn can be placed on either layer, since moving up or down a stair has a cost of 0.
4. The console will display:
- Whether a solution is found.
 - The total number of nodes explored.
 - The optimal move sequence (tile slides + pawn movements).