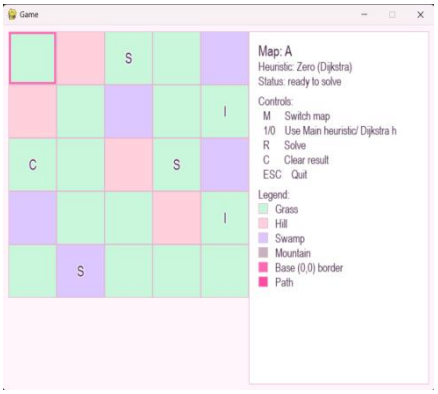
Report Project 1 Part 2

Student: Maria Alejandra Zapata Montano



1)Solution flow

1. Start at the base (0,0) with an empty backpack (capacity = 2).
2. Move on the 5×5 grid in 4 directions (up/down/left/right).
   Entering a cell costs according to its terrain: Grass=1, Hill=2, Swamp=3, Mountain=4.
3. Pick up a resource automatically if you step on it and still need that type (and you have space).
4. When the backpack is full (2 items) or you already collected everything you still need, head back to base.
5. Deposit happens automatically on base: delivered counters go up, backpack empties, and the resource tile becomes "empty" for the rest of the search.
6. Repeat until the delivery goal is reached: Stone×3, Iron×2, Crystal×1.
7. The final plan starts and ends at base, delivers exactly the required amounts, and minimizes total cost (sum of terrain entry costs).

We developed this in a UI that shows:

- the terrain colors and "S/I/C" labels,
- the optimal path (pink polyline),
- and metrics on the right panel (status, total cost, path length, expanded nodes, runtime, delivered S/I/C).

2)Heuristics used

1. Main heuristic

Intuition:

- If the backpack is full, you must go to base before making more progress:

estimate = Manhattan distance to base * min terrain cost.

- If you still need to collect something (after discounting what you already carry):

  estimate = Manhattan distance to the nearest needed resource × min terrain cost.

- If you don't need to collect more but still carry items, you must return to base. Same formula as above.
- Otherwise, .

Why admissible?
We multiply Manhattan distance by the **cheapest possible per step cost** on the map. That's an optimistic lower bound, so it **never overestimates**, therefore is admissible.

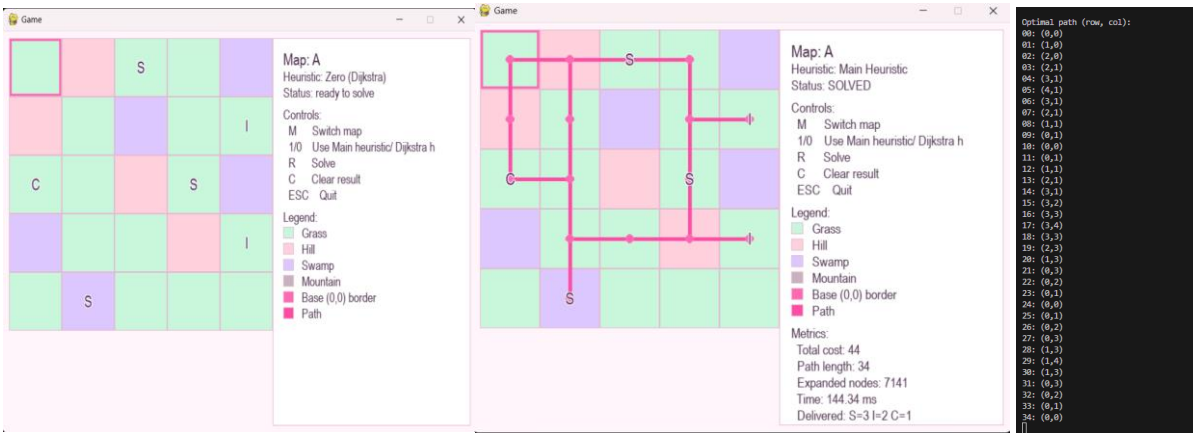
2. **Zero heuristic (baseline a.k.a. Dijkstra)**

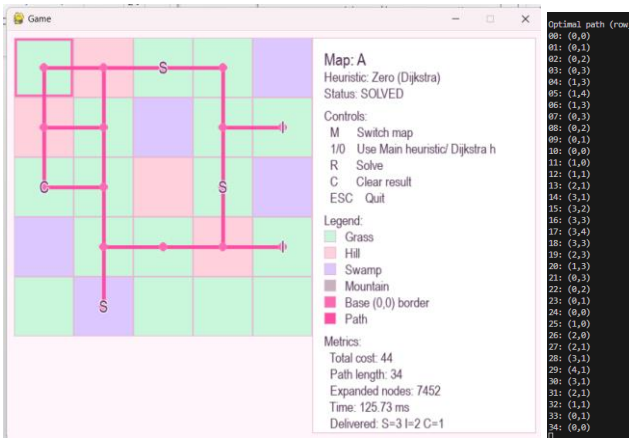
h(s) = 0 for all states.
Why include it? It's a **gold standard** to validate correctness:

- A* with h=0 is Dijkstra's algorithm ia always optimal, just **slower**.
- If **total cost** (and typically path length) matches between **Main** and **Zero**, our solver is behaving correctly.
- In the UI you can press **1** (Main) or **0** (Zero), solve, and compare.

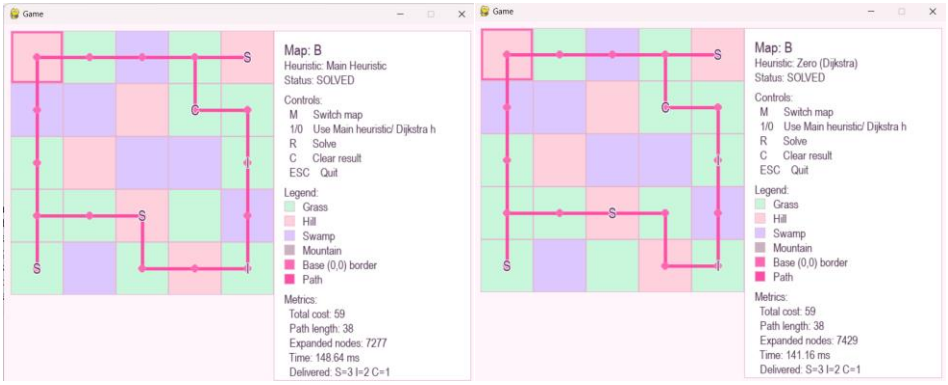
3) Results

For **Map A**, both heuristics found the **same optimal plan**: **total cost = 44** and **path length = 34**, with the required deliveries **S=3, I=2, C=1**. That match tells us the solution is correct (Zero/Dijkstra is the gold-standard baseline). The difference is efficiency: the **Main heuristic** expanded **7141** nodes versus **7452** with **Zero**, showing it gives useful guidance to A*; so the stronger signal is the **lower expansions** under Main. Bottom line: both yield the same optimal path, Zero validates correctness, and Main is measurably more search-efficient. Also, you can check in the terminal the specific path. For the map B we will not include the path in the report, just the analysis.

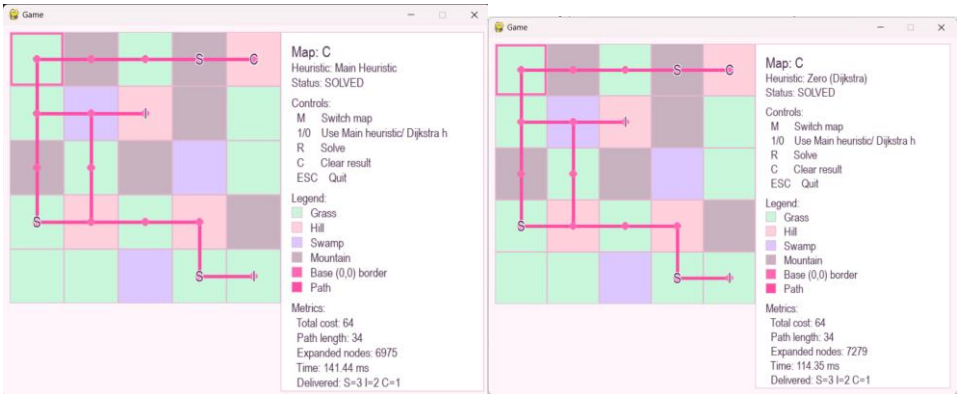




For **Map B**, both heuristics return the **same optimal plan**, **total cost = 59**, **path length = 38**, and the deliveries **S=3, I=2, C=1**, so correctness is validated by **Zero Dijkstra**. The difference is efficiency: **Main** expands **7,277** nodes vs **7,429** with **Zero**, showing the heuristic gives useful guidance.



For **Map C**, both heuristics agree on the **same optimal plan**, **total cost = 64** and **path length = 34** with deliveries **S=3, I=2, C=1**, so correctness is validated by **Zero Dijkstra**. **Main** expanded **6,975** nodes vs **7,279** with **Zero**, showing the heuristic provides useful guidance.



4)Discussion

At first, lets explain how A* worked:

A* = best-first search guided by a score f = g + h:

- g = cost so far (sum of terrain entry costs along the path).
- h = heuristic (our guess of the cost from the current state to the goal).

The algorithm always pulls from a priority queue the state with smallest f. If h is *admissible* (never overestimates the remaining cost), A* is guaranteed to find an optimal plan.

Our A* details

- State = (pos, backpack, delivered, consumed_mask)
  - pos: current cell (r,c)
  - backpack: sorted tuple of items you carry (size ≤ 2)
  - delivered: (Stone, Iron, Crystal) already deposited
  - consumed_mask: bitmask so each physical resource can be picked once
- Transitions:
  - Move to a 4-neighbor cell -> pay its terrain entry cost.
  - If stepping onto a needed resource and there is room -> pick it.
  - If stepping onto base -> deposit everything (counts clamp to targets).
  - Resource tiles turn empty after pickup (consistent with the assignment Q&A).
- Output:
  - We return path (list of cell coordinates), total_cost, expanded (nodes expanded).

A* is excellent in this case because, using an admissible heuristic, it typically discovers the cheapest plan much faster than blind search and provides you with transparent metrics (cost, expansions, and time). The trade-offs are as follows: A* may become memory-hungry as maps/resources increase, the state space explodes, and our heuristic won't anticipate smart combos, so it may still explore more than you'd want, particularly when several equal-cost paths tie.