#Pathfinding & Maze Lab (Vanilla HTML/CSS/JS)

An interactive, single-page playground for graph search and maze generation. You can draw walls/weights, generate mazes, and watch classic algorithms—A*, Dijkstra, BFS, DFS—race to the goal with live stats.

Why this exists

Real pathfinding sits at the heart of many systems: ride-hailing dispatch, logistics routing, robotics navigation, warehouse AGVs, RTS games, evacuation planning, even network packet routing analogies. Most demos are either framework-heavy or toy-simple. This app stays framework-free yet feature-rich, so you can study the algorithms without the noise.

What it does • Algorithms: A*, Dijkstra, BFS, DFS (step-through or animated run) • Heuristics (A*): Manhattan, Euclidean, Chebyshev • Topology: 4-neighborhood or 8-neighborhood (diagonals) • Terrain: paint walls and weights (costs 1–9) • Mazes: Recursive Division (rooms/corridors) & DFS (perfect mazes) • Grid editing: draw freehand, straight lines (Cmd/Ctrl-drag), move start/end • Telemetry: visited count, frontier peak, path length, runtime • I/O: export/import board to JSON • UX: light/dark theme, keyboard shortcuts, accessible ARIA live log

New • User context panel: shows your local time, time zone, and (on permission) geolocation coordinates. • Footer signature: Developed by ioazlan01.

How it works (brief) • The grid is a 2D array of Cell objects. Each cell holds flags (wall/visited/frontier/path), costs (g/h/f), and links to prev to reconstruct the path. • Pathfinders are implemented as ES6 generator functions (function*), yielding at each expansion. The UI runner advances the generator on a timer for smooth animation. • Min-heap implements the priority queues for Dijkstra/A*, keyed on g or f. • The renderer keeps a DOM div per cell and updates classes as state evolves; path reconstruction marks cells as path from goal back to start.

Where it's used in the real world (analogues) • Robotics/AV: local motion planning on occupancy grids • Warehousing: AGV routing with dynamic costs (traffic/charging) • Maps & logistics: quickest path under road weights/speeds • Games: enemy AI navigation on tile maps (A*) • Evacuation planning: safe routes under dynamic hazards (weights)

How to run 1. Place the three files in one folder: • index.html (structure) • styles.css (styling) • app.js (logic) 2. Open index.html in any modern browser.

No build, no dependencies. Everything is vanilla.

How to use • Pick an algorithm from the top bar (A*, Dijkstra, BFS, DFS). • Edit the grid: • Wall tool (W): click/drag to create obstacles. • Weight tool (E): cycles cell cost 1→9 (higher = slower). • Erase (R): remove walls/weights. • Drag (D): move start/end nodes. • Cmd/Ctrl + drag: paint straight lines. • Generate a maze: Recursive Division (rooms) or DFS (perfect corridors). • Run: press Space or click Run. Use Step for manual stepping. Stop halts the run. • Stats update live; Export/Import snapshot the board.

Shortcuts • Space Run/Stop • S Step • 1..4 Algorithms (A*/Dijkstra/BFS/DFS) • W/E/R/D Tools

Uniqueness • Generator-driven algorithms → true stepwise introspection. • Weighted terrain + 8-neighborhood → more realistic routing. • Zero dependencies → copy, tweak, ship anywhere. • Teaching mode: ARIA live log and clear visual states (frontier/visited/path).

Architecture map

index.html → layout & semantic regions (header/main/aside/footer) styles.css → CSS variables, grid layout, themes, animations app.js → models (Cell, Grid), algorithms, renderer, runner, IO

Extending • Add algorithms (IDA*, Jump Point Search, bidirectional BFS) • Add dynamic costs (time-varying weights), portals/teleporters • Record/play search traces; compare heuristics side-by-side • Export GIF of run; integrate with a service worker for offline lab

Credits

Developed by ioazlan01. Built with and vanilla web tech for clarity and speed. project_cloudflare