BÁO CÁO BÀI TẬP CÁ NHÂN

ĐỀ TÀI: TRÒ CHƠI 8 – PUZZLER

Môn: Trí tuệ nhân tạo

Mã lớp học phần: ARIN330585_05

Sinh viên thực hiện: Trần Thảo Tiến

MSSV: 23133076

1. Mục tiêu

Mục tiêu của bài tập cá nhân này nhằm nắm vững và mở rộng kiến thức về các kỹ thuật tìm kiếm trong lĩnh vực trí tuệ nhân tạo thông qua việc giải quyết bài toán 8-Puzzle. Trong quá trình thực hiện, em sẽ tiến hành tìm hiểu và lập trình nhiều loại thuật toán tìm kiếm. Mục tiêu chính là xây dựng và thử nghiệm các thuật toán trên bài toán 8-Puzzle, đồng thời phát triển công cụ trực quan hóa tiến trình tìm kiếm để làm rõ đặc điểm riêng biệt của từng thuật toán. Kết quả sẽ được phân tích và so sánh dựa trên thời gian thực thi và độ dài của lời giải thu được.

2. Tổng quan về trò chơi 8-puzzle

- Bài toán ghép tranh có lẽ rất quen thuộc với chúng ta, những người mới bắt đầu tiếp cận với môn trí tuệ nhân tạo. Vị trí của các hình trong game sẽ nằm ngẫu nhiên trộn lẫn trong n ô, trong đó có 1 ô trống để người chơi dịch chuyển đi từng bước. Mỗi lần di chuyển người chơi chỉ có thể đi 1 bước theo chiều qua trái, qua phải, đi lên hoặc đi xuống và không được đi chéo để ghép thành 1 hình hoàn chỉnh theo hình mẫu đã cho.
- Minh họa về bài toán 8-Puzzle với 1 bảng kích thước 3*3 và các ô số được đánh lần lượt từ 1 đến 8:

Trạng thái ban đầu

Trạng thái đích

3. Nội dung

3.1. Lý thuyết về các thuật toán

a. Thuật toán tìm kiếm không thông tin (Uninformed search algorithms)

Breadth-first Search (BFS): Là thuật toán tìm kiếm theo chiều rộng, mở rộng các nút theo thứ tự được tạo ra, sử dụng hàng đợi FIFO.

- Ưu điểm: Tìm lời giải ngắn nhất (về số bước), đảm bảo tìm lời giải nếu tồn tại.
- Nhước điểm: Tốn bộ nhớ rất lớn khi đồ thị rộng, chậm với không gian trạng thái lớn.(Độ phức tạp O(bd) với số phân nhánh b, chiều dài giải pháp tối thiểu d (nếu b ≥ 2))

Uniform Cost Search (UCS): Là thuật toán tìm kiếm theo chiều rộng tối ưu nếu tất cả chi phí hành động bằng nhau, nếu không thì không đảm bảo tối ưu

- Ưu điểm: Tối ưu vì ở mọi trạng thái, đường dẫn có chi phí thấp nhất sẽ được chọn: hiệu quả khi trọng số cạnh nhỏ vì nó khám phá các đường dẫn theo thứ tự đảm bảo tìm thấy đường dẫn ngắn nhất sớm. Thuật toán hoàn chỉnh, đảm bảo có thể tìm ra giải pháp bất cứ khi nào có giải pháp khả thi.
- Nhược điểm: Không quan tâm đến số bước liên quan đến tìm kiếm và chỉ quan tâm đến chi phí đường dẫn: có thể bị kẹt trong vòng lặp vô hạn.

Depth-first search (DFS): Là thuật toán tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO). Danh sách mở được triển khai dưới dạng ngăn xếp.

- Ưu điểm: Tốn ít bộ nhớ hơn BFS (O(b*d)), phù hợp với không gian tìm kiếm lớn.
- Hạn chế: Thuật toán tìm kiếm DFS không tối ưu vì nó có thể tạo ra số lượng lớn các bước hoặc chi phí cao để tiếp cận nút đích. Không đảm bảo tìm đường ngắn nhất, có thể bị kẹt trong vòng lặp vô hạn nếu không kiểm soát.

Iterative Deepening (IDS): Là thuật toán sự kết hợp của thuật toán DFS và BFS.

- Ý tưởng: Tìm kiếm theo chiều sâu cho đến một "giới hạn độ sâu" nhất định và tiếp tục tăng giới hạn độ sâu sau mỗi lần lặp cho đến khi tìm thấy nút mục tiêu
- Ưu điểm: Tốn ít bộ nhớ như DFS, đảm bảo tìm đường ngắn nhất như BFS trong đồ thị không trọng số.
- Nhược điểm: Lặp lại tìm kiếm nhiều lần, có thể tốn thời gian hơn.

b. Informed Search Algorithms (Thuật toán tìm kiếm có định hướng)

Greedy Search: Greedy best-first search cố gắng mở rộng nút gần nhất với đích, dựa trên giả định rằng điều này có khả năng dẫn đến lời giải nhanh chóng. Do đó, nó đánh giá các nút chỉ bằng cách sử dụng hàm heuristic; tức là, f(n) = h(n). Trong đó : h(n) là chi phí ước lượng thấp nhất của đường đi từ trạng thái tại nút n đến trạng thái đích.

- Ưu điểm: Tốc độ tìm kiếm nhanh, tiết kiệm bộ nhớ.
- Nhược điểm: Không có tính đầy đủ do có khả năng tạo thành vòng lặp vô hạn ở một số nút.

A Star: Sử dụng hàm Heuristics chấp nhận được + tìm kiếm theo chiều rộng → Loại bỏ những đường đi có chi phí cao.

- Hàm lượng giá: f(n) = g(n) + h(n) g(n) = Chi phí từ gốc đến nút hiện tại n h(n) = Lượng giá từ nút n đến đích f(n) = Ước lượng tổng chi phí từ nút gốc đến nút mục tiêu nếu đi qua n.
- Ưu điểm: Tìm lời giải tối ưu nếu heuristic hợp lệ, hiệu quả trong nhiều bài toán.
- Nhược điểm: Tốn bộ nhớ lớn, hiệu quả phụ thuộc vào chất lượng heuristic.

IDA Star: Kết hợp A* với IDS, giới hạn f(n) thay vì độ sâu, tăng dần giới hạn qua mỗi lần lặp.

- Ưu điểm: Tốn ít bộ nhớ hơn A*, phù hợp với không gian lớn.
- Hạn chế: Lặp lại node, tốn thời gian nếu heuristic không tốt.

c. Local Search (Tìm kiếm cục bộ)

Simple Hill Climbing: Chỉ kiểm tra từng trạng thái lận cận của nó và nếu nó tìm thấy trạng thái tốt hơn trạng thái hiện tại thì di chuyển

- Ưu điểm: Đơn giản, tốn ít tài nguyên.
- Hạn chế: Dễ bị kẹt ở cực đại cục bộ, không đảm bảo tối ưu.

Steepest Ascent Hill Climbing: Thuật toán này kiểm tra tất cả các nút lân cận của trạng thái hiện tại và chọn một nút lân cận gần nhất với trạng thái mục tiêu.

- Ưu điểm: Hiệu quả hơn Simple Hill Climbing.
- Hạn chế: Vẫn dễ bị kẹt ở cực đại cục bộ.

Stochastic Hill Climbing: Là một biến thể của thuật toán leo đồi đơn giản. Thay vì tìm ra hàng xóm tốt nhất, phiên bản này lựa chọn ngẫu nhiên một hàng xóm. Nếu hàng xóm đó tốt hơn trạng thái hiện thời, hàng xóm đó sẽ được chọn làm trạng thái hiện thời và thuật toán lặp lại. Ngược lại, nếu hàng xóm được chọn không tốt hơn, thuật toán sẽ chọn ngẫu nhiên một hàng xóm khác và so sánh. Thuật toán kết thúc và trả lại trạng thái hiện thời khi đã hết “kiên nhẫn”.

- Ưu điểm: Giảm khả năng kẹt, khám phá đa dạng hơn.
- Hạn chế: Không đảm bảo tối ưu, phụ thuộc vào yếu tố ngẫu nhiên.

Simulated Annealing: Mô phỏng quá trình làm nguội kim loại, chấp nhận neighbor xấu hơn với xác suất giảm dần (theo nhiệt độ).

- Ưu điểm: Có thể thoát khỏi cực đại cục bộ, tìm giải pháp gần tối ưu.
- Hạn chế: Phụ thuộc vào lịch trình làm nguội, tốn thời gian nếu điều chỉnh không tốt.

Beam Search: Giữ lại k trạng thái tốt nhất ở mỗi bước, mở rộng từ các trạng thái này.

- Ưu điểm: Giảm bộ nhớ so với BFS, phù hợp với không gian lớn.
- Hạn chế: Không đảm bảo tối ưu, phụ thuộc vào giá trị k.

Genetic Algorithm: Mô phỏng tiến hóa sinh học, sử dụng quần thể giải pháp, lai ghép (crossover) và đột biến (mutation).

- Ưu điểm: Tìm giải pháp tốt trong không gian phức tạp, thoát cực đại cục bộ.
- Hạn chế: Phụ thuộc vào tham số (tỷ lệ đột biến, kích thước quần thể), tốn thời gian.

d. Complex Environments (Môi trường phức tạp)

AND-OR Graph Search: Tìm kiếm trong không gian có hành động không xác định, sử dụng cấu trúc đồ thị AND-OR.

- Ưu điểm: Xử lý được môi trường không xác định, đảm bảo tính đầy đủ.
- Hạn chế: Phức tạp, tốn tài nguyên nếu đồ thị lớn.

Belief State Search: Tìm kiếm trên không gian các tập trạng thái khả thi khi trạng thái chính xác không biết.

- Ưu điểm: Phù hợp với môi trường quan sát một phần.
- Hạn chế: Không gian trạng thái niềm tin có thể rất lớn, tốn tài nguyên.

Sensorless Search: Giải quyết bài toán không có cảm biến (sensorless), giả định mọi trạng thái có thể.

- Ưu điểm: Xử lý bài toán thiếu thông tin quan sát.
- Hạn chế: Phức tạp, tốn tài nguyên tính toán.

e. CSPs (Constraint Satisfaction Problems - Bài toán ràng buộc)

Backtracking Search: Gán giá trị cho các biến từng bước, quay lui khi gặp mâu thuẫn với ràng buộc.

- Ưu điểm: Đơn giản, đảm bảo tìm tất cả giải pháp (nếu có).
- Hạn chế: Tốn thời gian nếu không gian tìm kiếm lớn.

Backtracking with Forward Checking (Backtracking FC): Cải tiến Backtracking, kiểm tra ràng buộc sớm (forward checking) để loại bỏ giá trị không khả thi. Khi gán giá trị cho biến, kiểm tra và thu hẹp miền giá trị của các biến liên quan.

- Ưu điểm: Giảm không gian tìm kiếm, tăng tốc quá trình giải.
- Hạn chế: Tăng chi phí tính toán mỗi bước, phức tạp hơn backtracking thuần túy.

f. Reinforcement Learning (Học tăng cường)

Q-Learning: Học chính sách tối ưu bằng cách cập nhật bảng Q (Q-table) dựa trên phần thưởng và trạng thái tương lai.

- Ưu điểm: Không cần mô hình môi trường, hội tụ về chính sách tối ưu.
- Hạn chế: Chậm trong không gian trạng thái lớn, cần nhiều lần thử.

SARSA (State-Action-Reward-State-Action): Cập nhật Q-values dựa trên hành động thực tế agent chọn, có tính chất "on-policy".

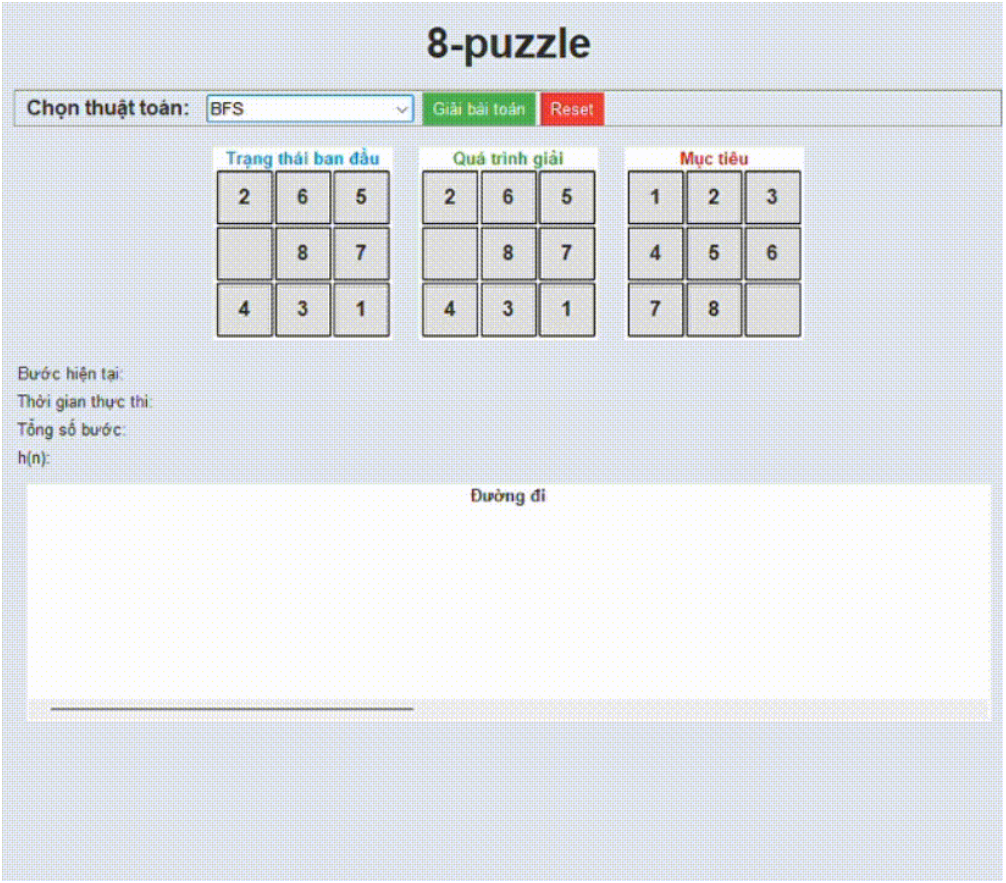
- Ưu điểm: Phù hợp với môi trường cần chính sách an toàn, học trực tiếp từ hành động thực tế.
- Hạn chế: Có thể hội tụ chậm hơn Q-Learning, phụ thuộc vào chính sách khám phá.

3.2 Các thuật toán tìm kiếm dưới góc nhìn 8 – puzzle

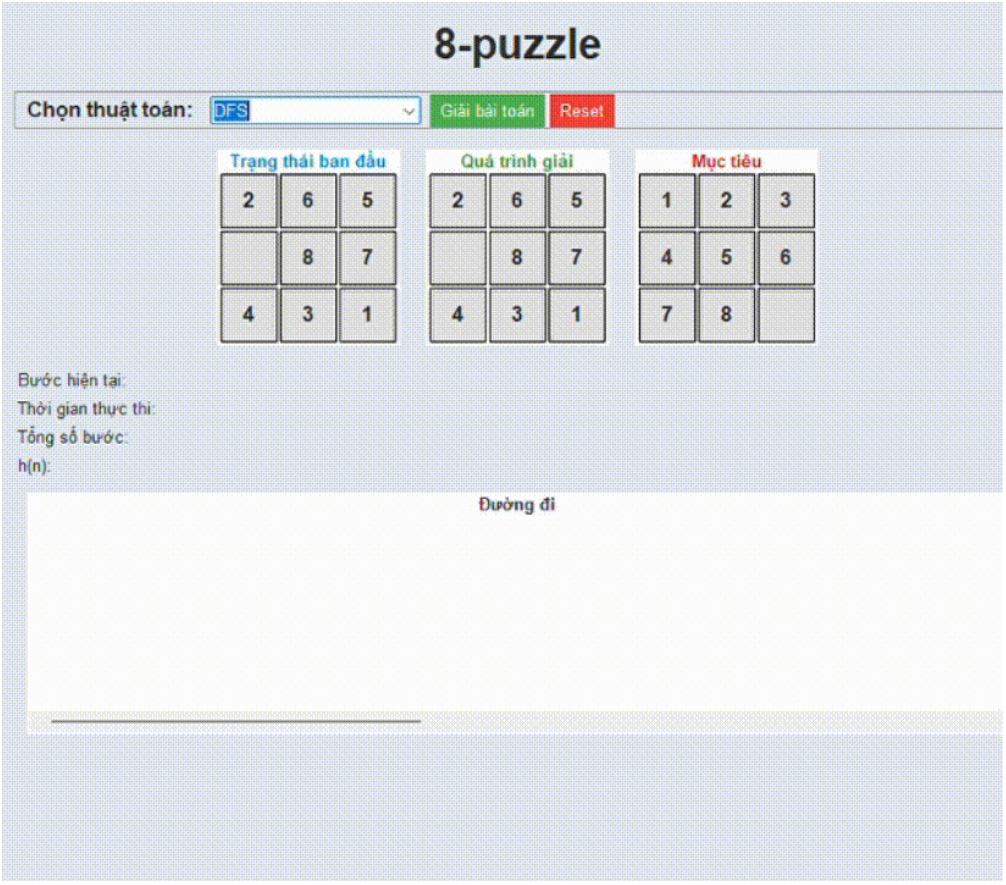
```
├── main.py
├── Uniformed_Search/
│   ├── bfs.py
│   ├── dfs.py
│   ├── ids.py
│   ├── ucs.py
├── Informed_Search/
│   ├── greedy.py
│   ├── astar.py
│   ├── idastar.py
├── Local_Search/
│   ├── simple_hill_climbing.py
│   ├── steepest_ascent_hill_climbing.py
│   ├── stochastic_hill_climbing.py
│   ├── simulated_annealing.py
│   ├── beam_search.py
│   ├── genetic.py
├── Complex_Environments/
│   ├── and_or_graph_search.py
│   ├── belief_state_search.py
│   ├── sensorless_problem.py
├── CSPs/
│   ├── backtracking_search.py
│   ├── backing_fc.py  # Mới
├── Reinforcement_Learning/
│   ├── q_learning.py
│   ├── sarsa.py
```

a. Thuật toán tìm kiếm không thông tin (Uninformed search algorithms)

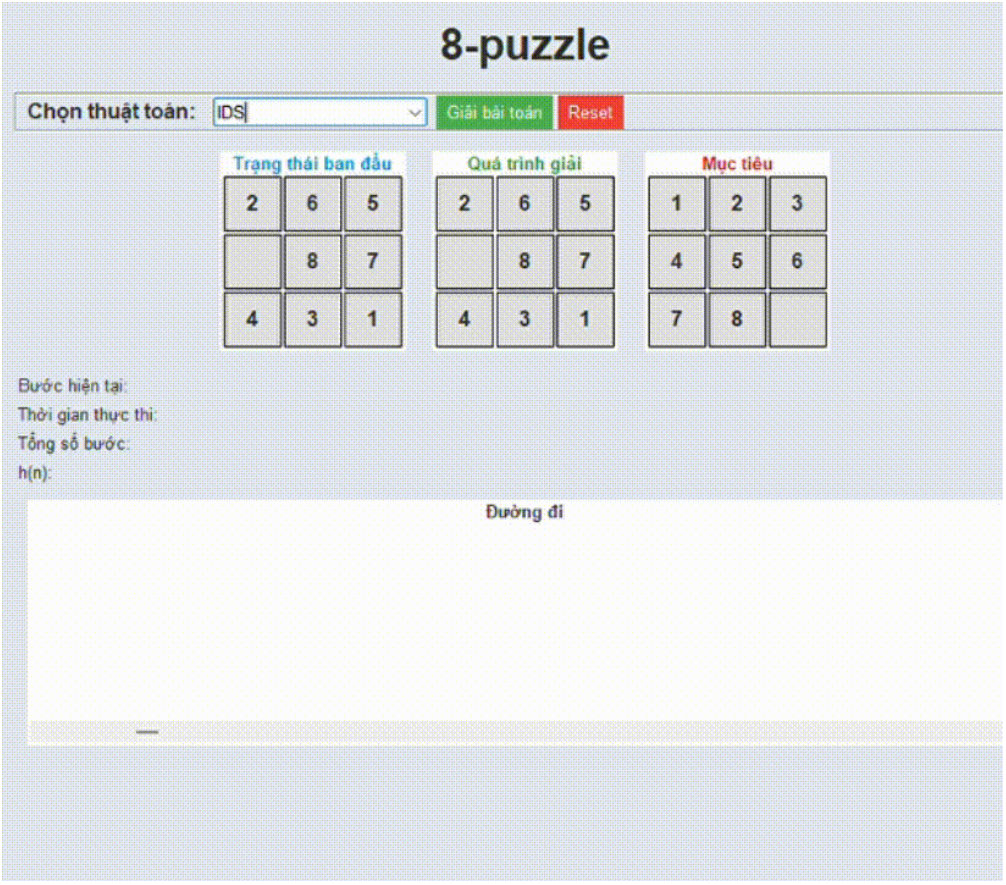
Breadth-First Search (BFS): Trong bài toán 8-puzzle, thuật toán BFS được áp dụng để khám phá không gian trạng thái bằng cách mở rộng các trạng thái theo từng mức độ, bắt đầu từ trạng thái gốc. Tại mỗi bước, thuật toán tạo ra các trạng thái con bằng cách di chuyển ô trống (giá trị 0) theo bốn hướng hợp lệ: lên, xuống, trái, phải. Tất cả trạng thái sinh ra sẽ được đưa vào một hàng đợi để xử lý theo thứ tự xuất hiện. Đồng thời, một tập các trạng thái đã duyệt được duy trì nhằm ngăn chặn việc duyệt lặp. Với cách tiếp cận này, BFS đảm bảo tìm được đường đi có số bước ngắn nhất từ trạng thái ban đầu đến trạng thái đích nếu tồn tại lời giải.



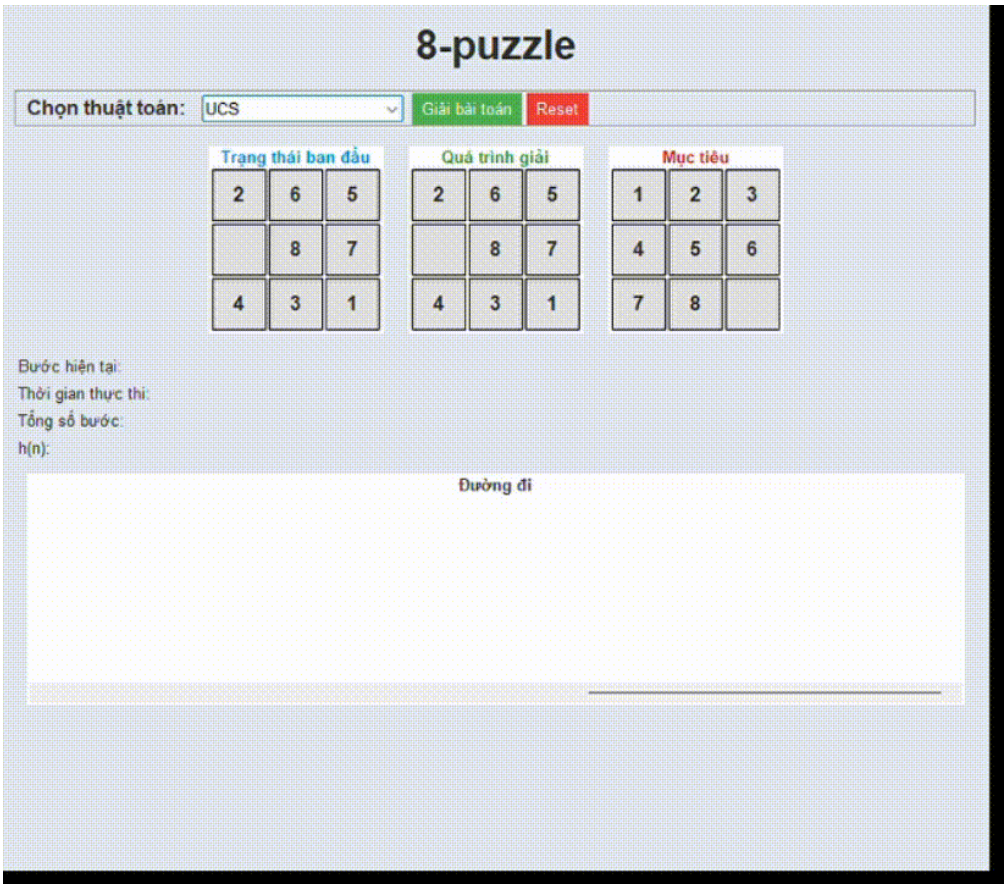
Depth-First Search (DFS): Trong bài toán 8-puzzle, thuật toán DFS hoạt động bằng cách khám phá một nhánh trạng thái đến mức sâu nhất có thể trước khi quay lại để xét các nhánh khác. Quá trình này được thực hiện bằng cách sử dụng một ngăn xếp (stack) để lưu trữ các trạng thái chờ xử lý. Tại mỗi bước, trạng thái ở đỉnh ngăn xếp được lấy ra và mở rộng bằng cách di chuyển ô trống (giá trị 0) theo những hướng hợp lệ. Nếu chưa đạt được trạng thái mục tiêu, các trạng thái con được đẩy vào ngăn xếp để tiếp tục khám phá. Để tránh vòng lặp vô hạn và giảm thiểu việc duyệt lại các trạng thái cũ, thuật toán có thể đặt giới hạn độ sâu và kiểm soát các trạng thái đã duyệt trong quá trình tìm kiếm.



Iterative Deepening Search (IDS): Thuật toán IDS kết hợp giữa DFS và BFS. Thay vì đi sâu vô hạn như DFS, IDS tiến hành các vòng lặp tìm kiếm theo chiều sâu có kiểm soát, với mỗi vòng lặp áp đặt một giới hạn độ sâu cụ thể. Bắt đầu từ độ sâu 0, thuật toán dần tăng mức giới hạn và thực hiện DFS trong phạm vi đó. Mỗi lần mở rộng trạng thái, IDS chỉ tiếp tục nếu độ sâu chưa vượt ngưỡng đang xét. Quá trình lặp tiếp tục cho đến khi trạng thái mục tiêu được tìm thấy. Vì vậy thuật toán vừa tiết kiệm bộ nhớ, vừa đảm bảo tìm được đường đi ngắn nhất đến đích.

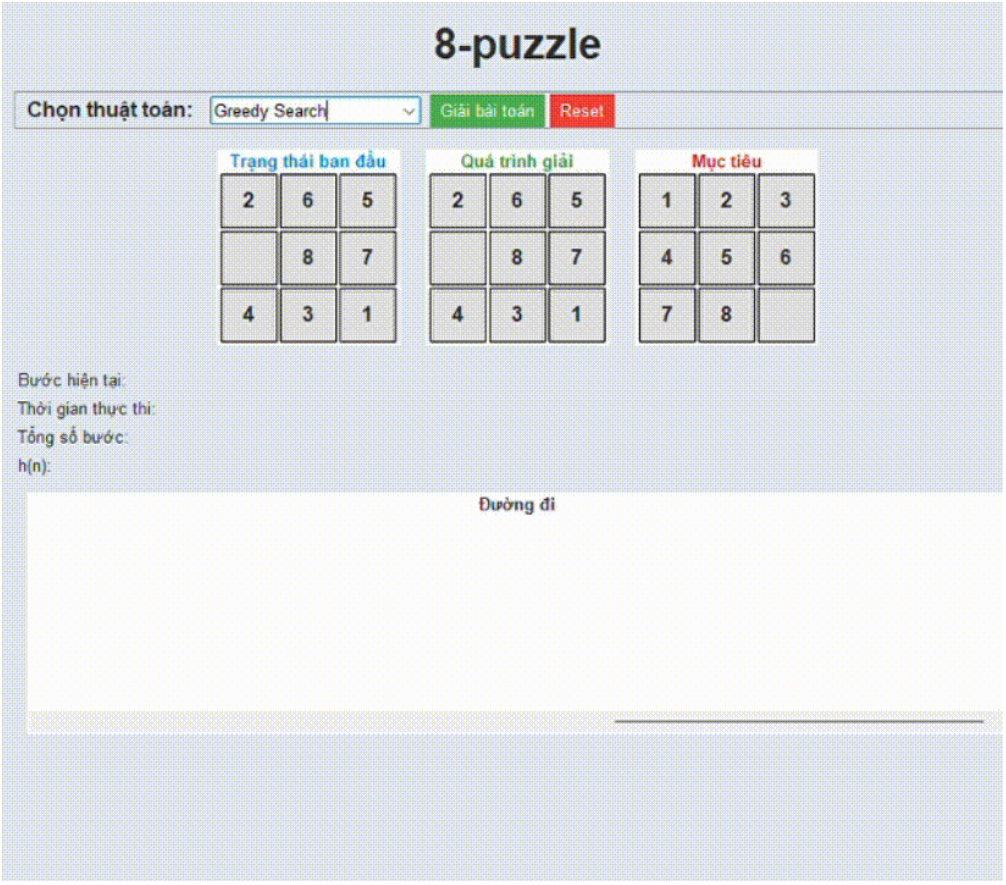


Uniform Cost Search (UCS): Trong 8-puzzle, thuật toán Uniform Cost Search (UCS) thực hiện tìm kiếm dựa trên tổng chi phí tích lũy từ trạng thái ban đầu đến từng trạng thái trung gian. UCS sử dụng một hàng đợi ưu tiên, trong đó các trạng thái có chi phí thấp hơn sẽ được xử lý trước. Tại mỗi bước, thuật toán chọn trạng thái có chi phí nhỏ nhất để mở rộng bằng cách di chuyển ô trống theo các hướng hợp lệ. Mặc dù trong bài toán này, mỗi bước di chuyển thường có cùng một chi phí (ví dụ, đều bằng 1), UCS vẫn duy trì sự chính xác bằng cách đảm bảo mở rộng theo chi phí thực tế. Nhờ đó, UCS cũng đảm bảo tìm ra lời giải tối ưu về mặt số bước di chuyển, tương tự như BFS, nhưng với cách tổ chức ưu tiên theo chi phí thay vì theo mức duyệt.

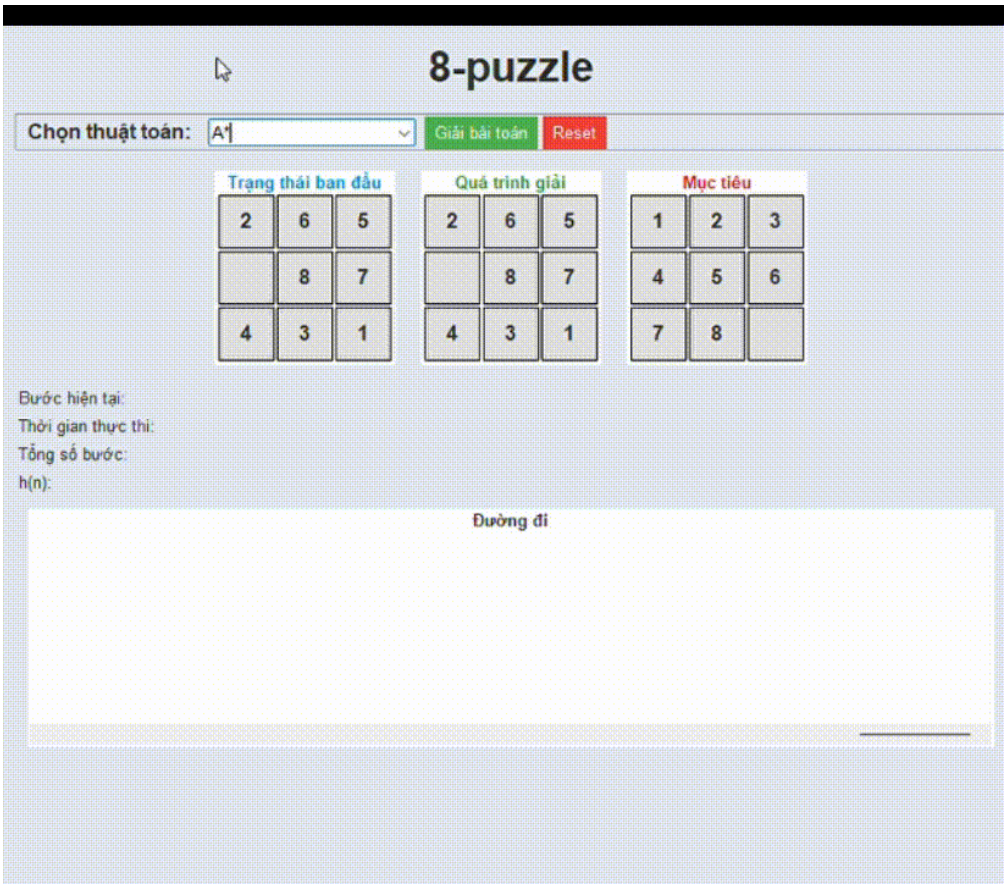


b. Thuật Toán Tìm Kiếm Có Thông Tin (Informed Search Algorithms)

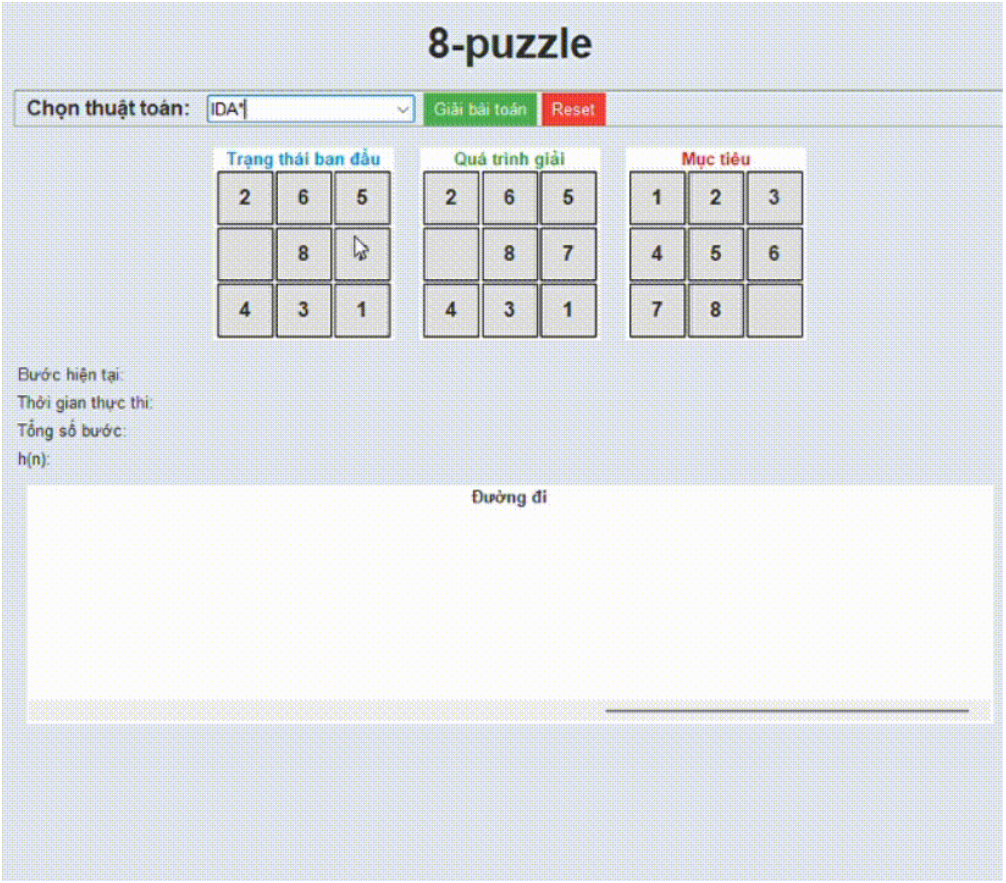
Greedy Best-First Search: Tìm kiếm tham lam chọn trạng thái 8-puzzle tiếp theo dựa trên giá trị heuristic, chẳng hạn khoảng cách Manhattan, mà không xem xét chi phí đường đi. Thuật toán sử dụng hàng đợi ưu tiên để ưu tiên các trạng thái gần mục tiêu hơn theo heuristic. Trong bài toán 8-puzzle, một hàm heuristic phổ biến là tổng khoảng cách Manhattan – tức tổng số bước hàng và cột mà mỗi ô cần di chuyển để đến đúng vị trí. Mỗi lần lặp, thuật toán chọn trạng thái có giá trị heuristic nhỏ nhất để mở rộng tiếp. Mặc dù có thể nhanh chóng tiếp cận đích trong nhiều trường hợp, Greedy không đảm bảo tìm được lời giải tối ưu vì có thể bỏ qua những đường đi ngắn hơn do chỉ nhìn vào phần còn lại mà không xét đến chi phí đã tích lũy.



A Star Search (AStar): Thuật toán A* là một phương pháp tìm kiếm hiệu quả có sử dụng thông tin ước lượng để định hướng hành động. Nó kết hợp giữa hai yếu tố: chi phí đã đi (g), là tổng chi phí từ trạng thái xuất phát đến trạng thái hiện tại, và chi phí dự đoán (h), là ước lượng khoảng cách từ trạng thái hiện tại đến mục tiêu. Tổng chi phí f = g + h được dùng làm tiêu chí để lựa chọn trạng thái tiếp theo cần mở rộng, thông qua hàng đợi ưu tiên. Trong bài toán 8-puzzle, hàm heuristic thường dùng là tổng khoảng cách Manhattan của các ô. Khi heuristic được thiết kế tốt (chấp nhận được và không vượt quá chi phí thực), A* có thể tìm ra lời giải tối ưu nhanh hơn so với BFS hoặc UCS, đồng thời vẫn đảm bảo tính đúng đắn và tối ưu của kết quả.

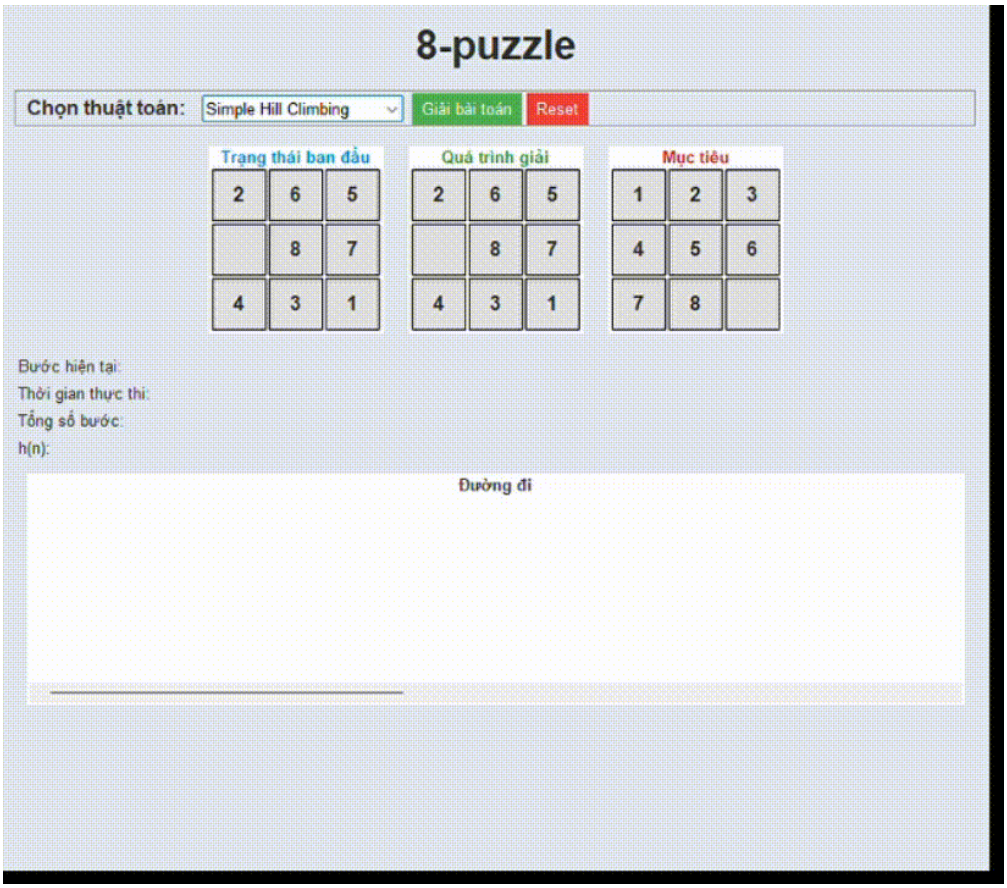


Iterative Deepening A Star (IDAStar): Thuật toán IDA* kết hợp của A* và IDS để tiết kiệm bộ nhớ. Thay vì sử dụng hàng đợi ưu tiên như A*, IDA* thực hiện tìm kiếm theo chiều sâu giới hạn dựa trên ngưỡng giá trị f = g + h, trong đó g là chi phí thực đã đi và h là ước lượng heuristic đến đích. Thuật toán bắt đầu với một ngưỡng f ban đầu và chỉ mở rộng các trạng thái có f ≤ ngưỡng đó. Nếu không tìm được lời giải trong ngưỡng này, ngưỡng được nâng lên thành giá trị f nhỏ nhất vượt qua ngưỡng cũ được gặp trong quá trình tìm kiếm, rồi lặp lại quá trình.

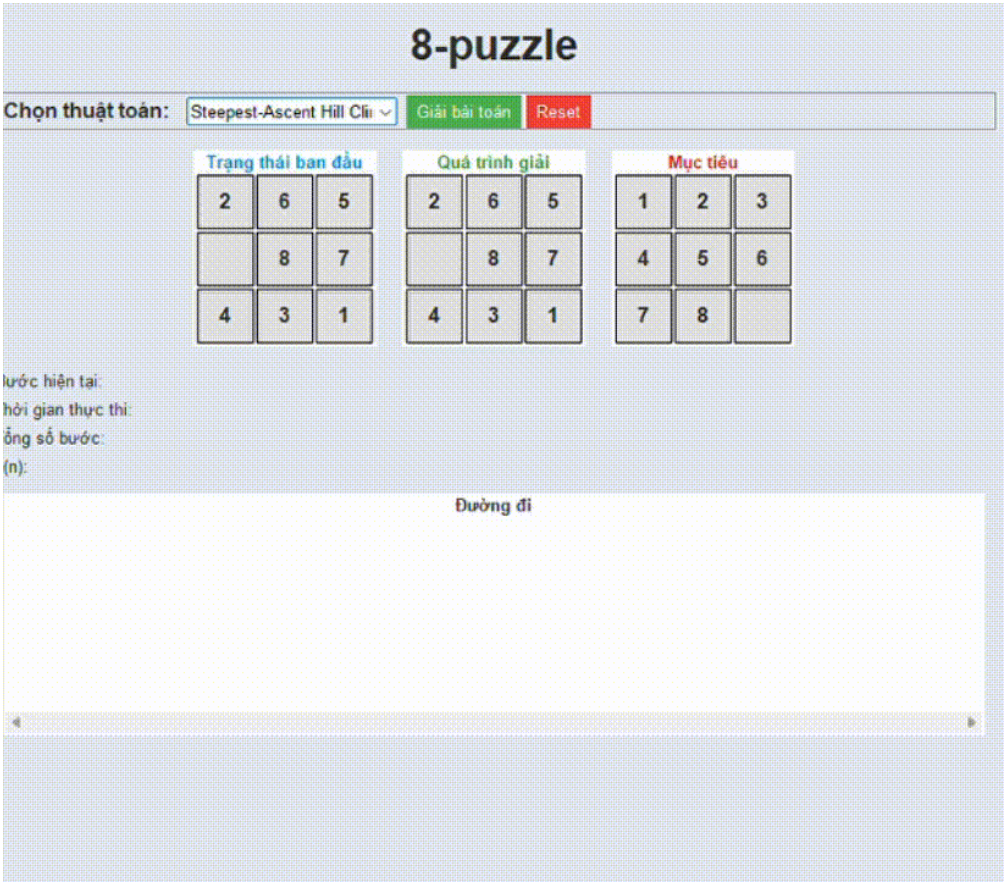


c. Thuật Toán Tìm Kiếm Cục Bộ (Local Search)

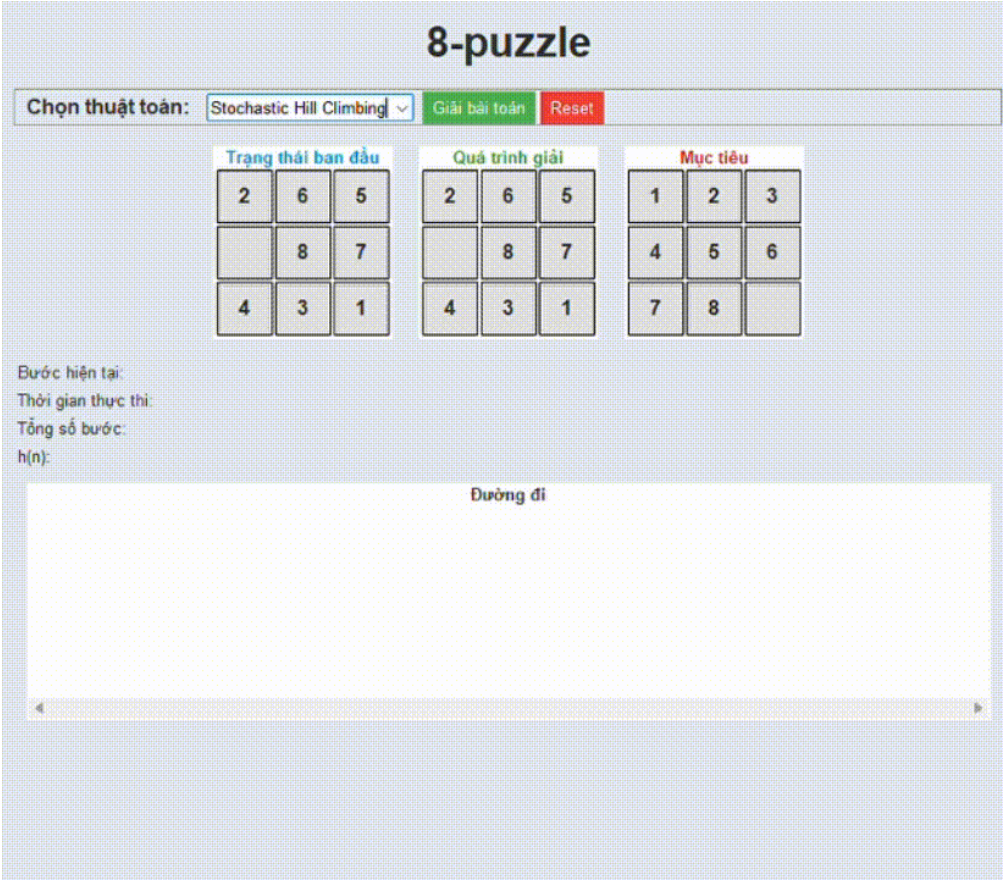
Simple Hill Climbing: Leo đồi đơn giản bắt đầu từ trạng thái ban đầu của 8-puzzle và chọn trạng thái con có giá trị heuristic (như khoảng cách Manhattan) tốt hơn trạng thái hiện tại, lặp lại cho đến khi đạt mục tiêu hoặc không còn cải thiện. Thuật toán nhanh và sử dụng ít bộ nhớ, nhưng dễ bị kẹt ở cực đại cục bộ, dẫn đến giải pháp không tối ưu hoặc thất bại trong 8-puzzle. Nó phù hợp cho các trường hợp cần giải pháp nhanh mà không yêu cầu chất lượng cao.



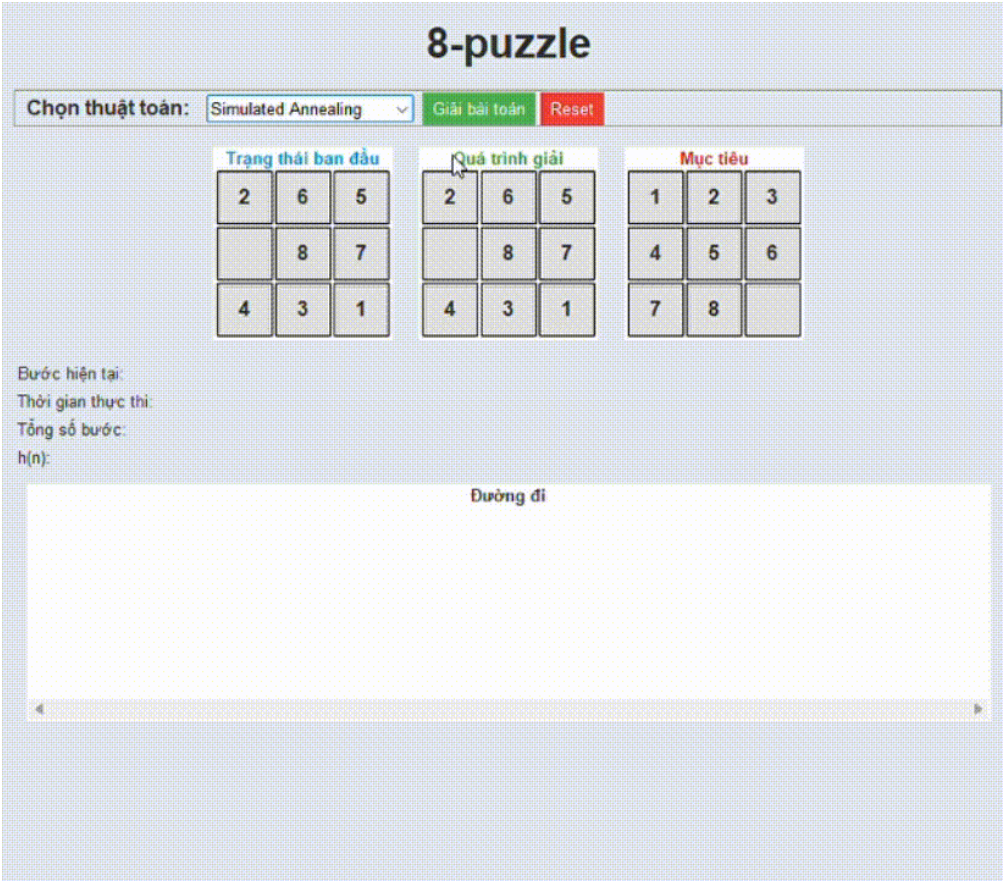
Steepest-Ascent Hill Climbing: Leo đồi dốc nhất cải tiến leo đồi đơn giản bằng cách xem xét tất cả trạng thái con của trạng thái 8-puzzle hiện tại và chọn trạng thái có giá trị heuristic tốt nhất (thấp nhất). Điều này tăng cơ hội tìm giải pháp tốt hơn, nhưng vẫn có nguy cơ kẹt ở cực đại cục bộ. Trong 8-puzzle, thuật toán này hiệu quả hơn leo đồi đơn giản về chất lượng giải pháp, nhưng vẫn không đảm bảo tối ưu và có thể dừng lại trước khi đạt mục tiêu nếu không có cải thiện.



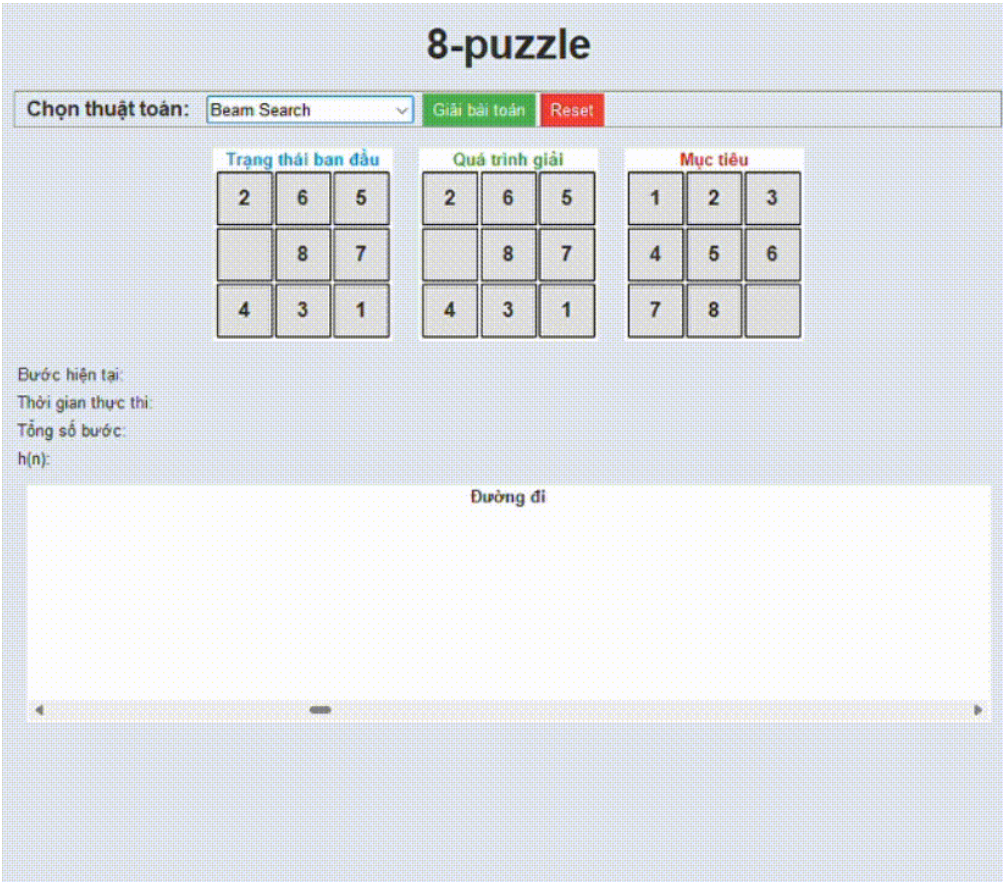
Stochastic Hill Climbing: Leo đồi ngẫu nhiên chọn ngẫu nhiên một trạng thái con tốt hơn trạng thái 8-puzzle hiện tại dựa trên giá trị heuristic, thay vì luôn chọn trạng thái tốt nhất. Tính ngẫu nhiên giúp thuật toán tránh một số cực đại cục bộ, tăng khả năng tìm giải pháp trong 8-puzzle so với leo đồi đơn giản hoặc dốc nhất. Tuy nhiên, nó vẫn không đảm bảo giải pháp tối ưu và phụ thuộc vào yếu tố ngẫu nhiên, dẫn đến kết quả không ổn định.



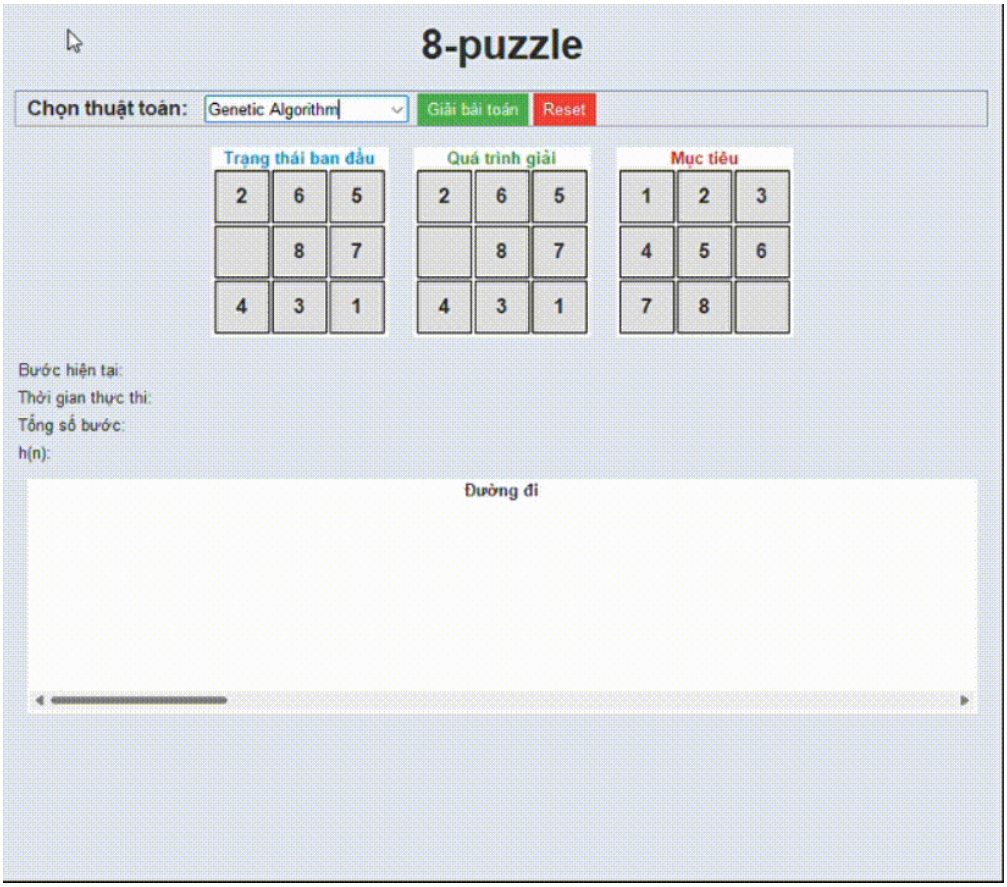
Simulated Annealing: Ủ nhiệt mô phỏng bắt đầu từ trạng thái 8-puzzle ban đầu và cho phép chọn cả trạng thái con tệ hơn với xác suất giảm dần theo thời gian, dựa trên tham số nhiệt độ. Điều này giúp thuật toán thoát khỏi cực đại cục bộ, tăng cơ hội tìm giải pháp toàn cục. Trong 8-puzzle, ủ nhiệt mô phỏng hiệu quả khi được điều chỉnh tốt, nhưng không đảm bảo tối ưu và phụ thuộc vào lịch trình làm nguội. Nó phù hợp khi cần cân bằng giữa tốc độ và chất lượng giải pháp.



Beam Search: Thuật toán Beam giới hạn số lượng trạng thái 8-puzzle được khám phá ở mỗi bước bằng một tham số độ rộng chùm, chỉ giữ lại các trạng thái tốt nhất dựa trên giá trị heuristic (như khoảng cách Manhattan). Thuật toán này nhanh hơn A* vì giảm không gian tìm kiếm, nhưng không đảm bảo giải pháp tối ưu và có thể bỏ qua các trạng thái dẫn đến mục tiêu. Trong 8-puzzle, tìm kiếm chùm hữu ích khi cần giải pháp nhanh với tài nguyên hạn chế, nhưng kết quả phụ thuộc vào độ rộng chùm được chọn.

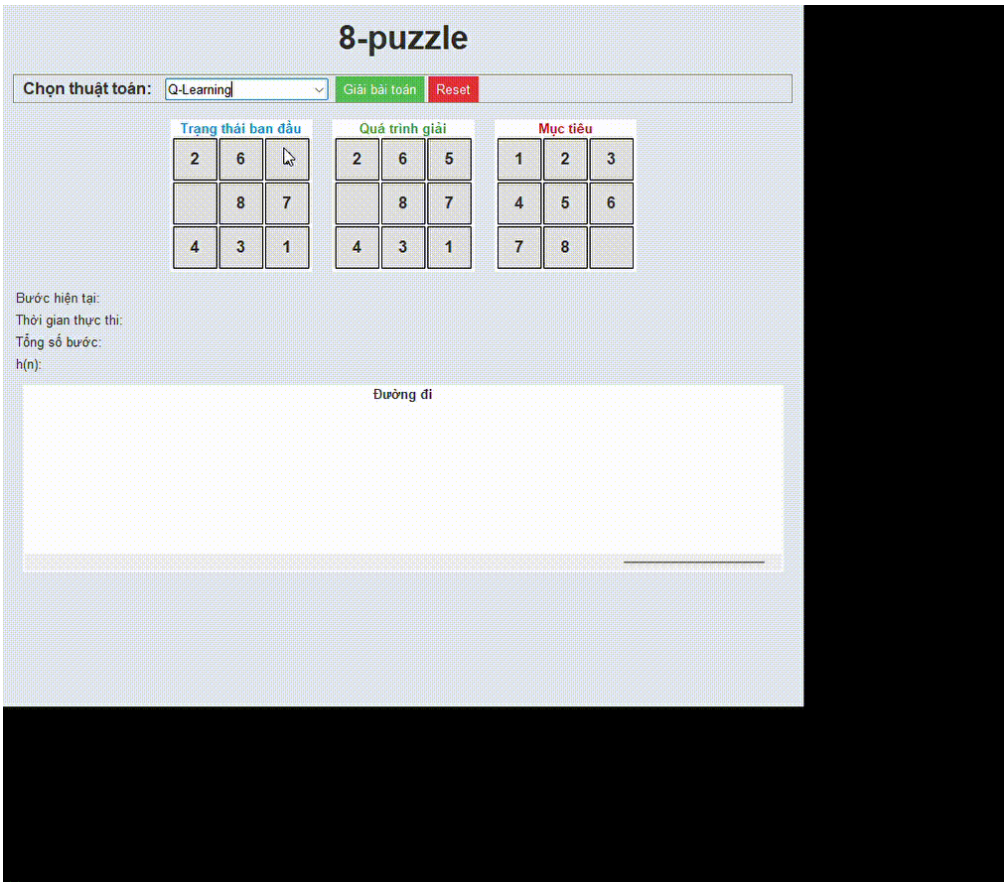


Genetic Algorithm: Thuật toán di truyền mô phỏng quá trình tiến hóa để tìm chuỗi di chuyển cho 8-puzzle, sử dụng quần thể các chuỗi di chuyển ngẫu nhiên, tiến hành lai ghép và đột biến để cải thiện. Trong 8-puzzle, thuật toán này có thể tìm giải pháp khả thi, nhưng không đảm bảo tối ưu và phụ thuộc vào kích thước quần thể, số thế hệ. Nó phù hợp khi cần giải pháp sáng tạo cho không gian tìm kiếm phức tạp.

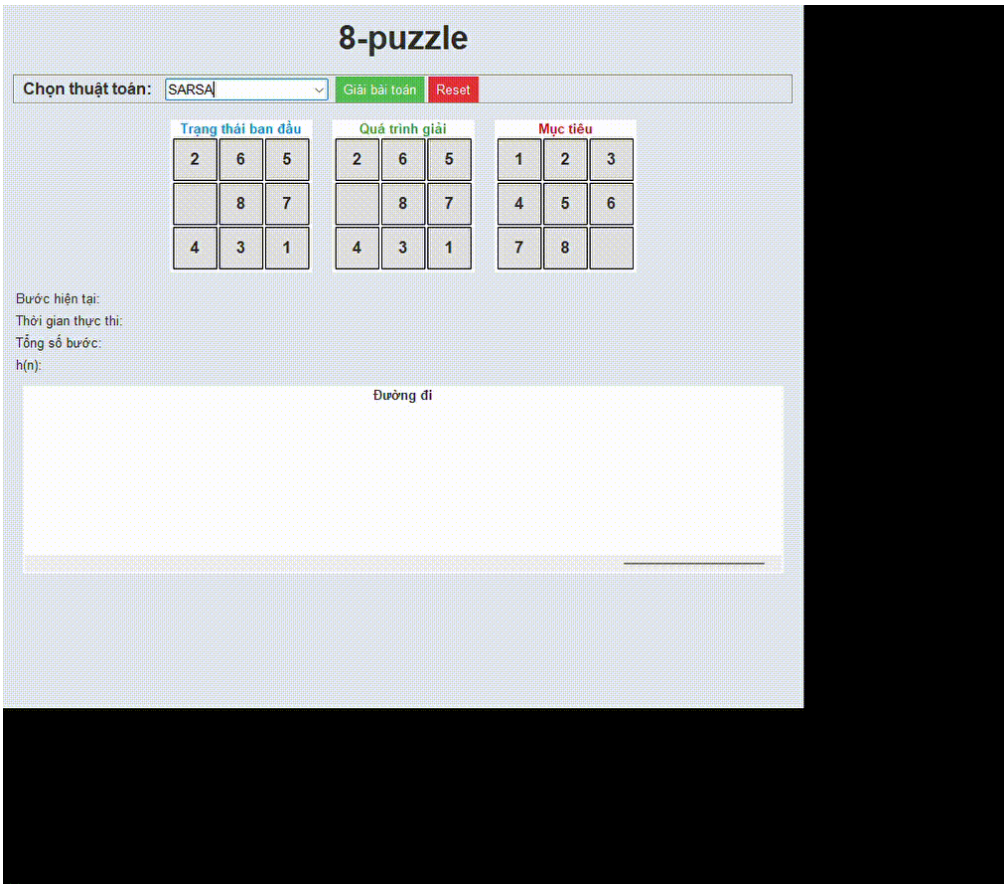


d. Thuật Toán Học Tăng Cường (Reinforcement Learning)

Q-Learning: Q-Learning là thuật toán học tăng cường, học cách chọn các di chuyển trong 8-puzzle thông qua thử và sai, cập nhật bảng Q dựa trên phần thưởng (ví dụ: -1 cho mỗi bước, +100 khi đạt mục tiêu). Thuật toán sử dụng chính sách epsilon-greedy để cân bằng giữa khám phá và khai thác. Trong 8-puzzle, Q-Learning có thể tìm giải pháp sau nhiều lần lặp, nhưng không đảm bảo tối ưu và yêu cầu thời gian huấn luyện dài. Nó phù hợp cho các bài toán cần học từ kinh nghiệm mà không có mô hình rõ ràng.

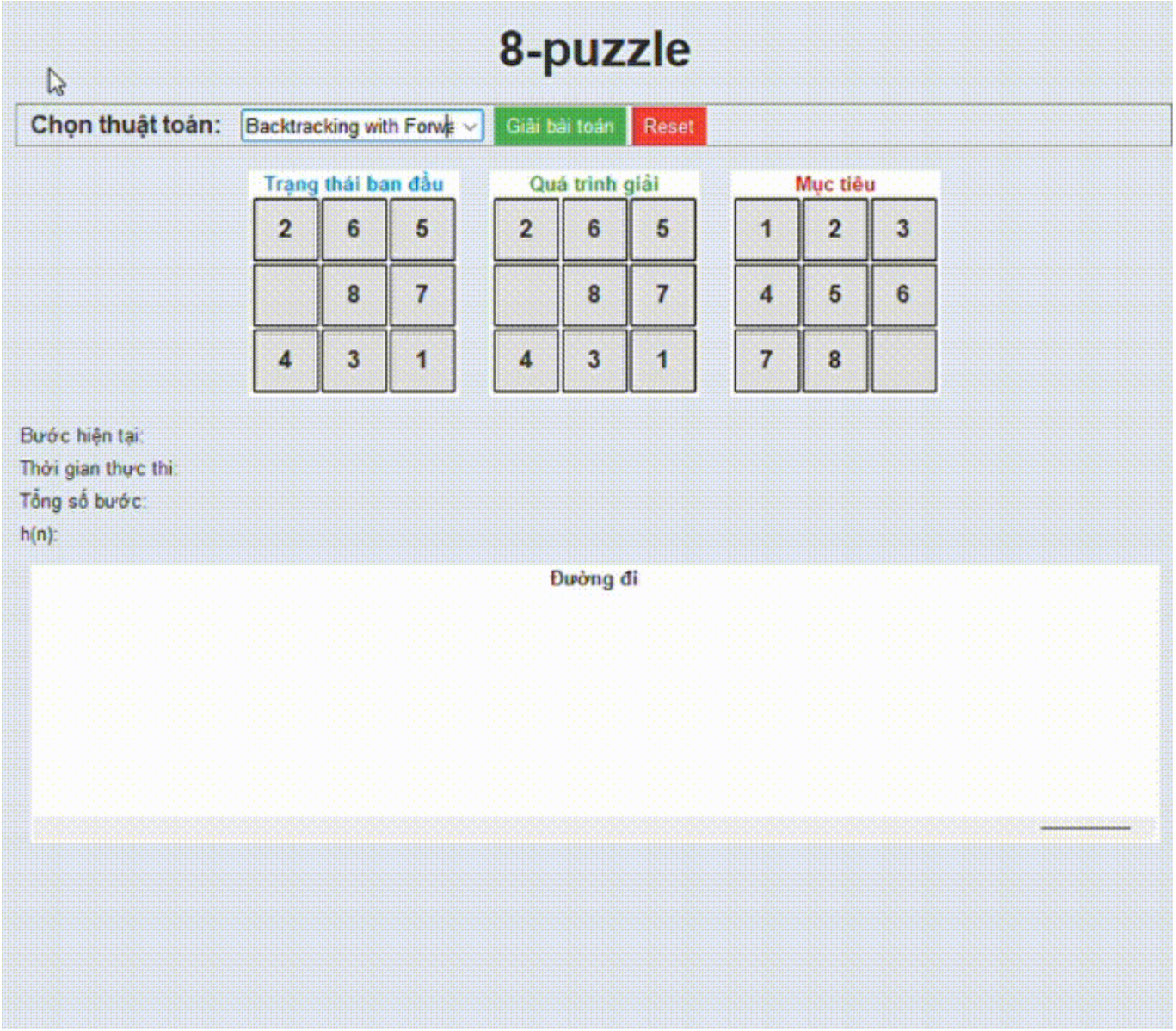


SARSA (State-Action-Reward-State-Action): SARSA, thuật toán học tăng cường on-policy, cập nhật bảng Q dựa trên hành động thực tế được chọn ở trạng thái tiếp theo, sử dụng chính sách epsilon-greedy. Trong 8-puzzle, SARSA học chính sách qua 1000 tập, với phần thưởng tương tự Q-Learning, và tạo đường đi bằng cách chọn hành động có Q cao nhất. SARSA chậm hơn A*/IDA* do huấn luyện, nhưng linh hoạt trong môi trường động. Nó hiệu quả khi cần chính sách thích nghi, nhưng không đảm bảo đường đi tối ưu.

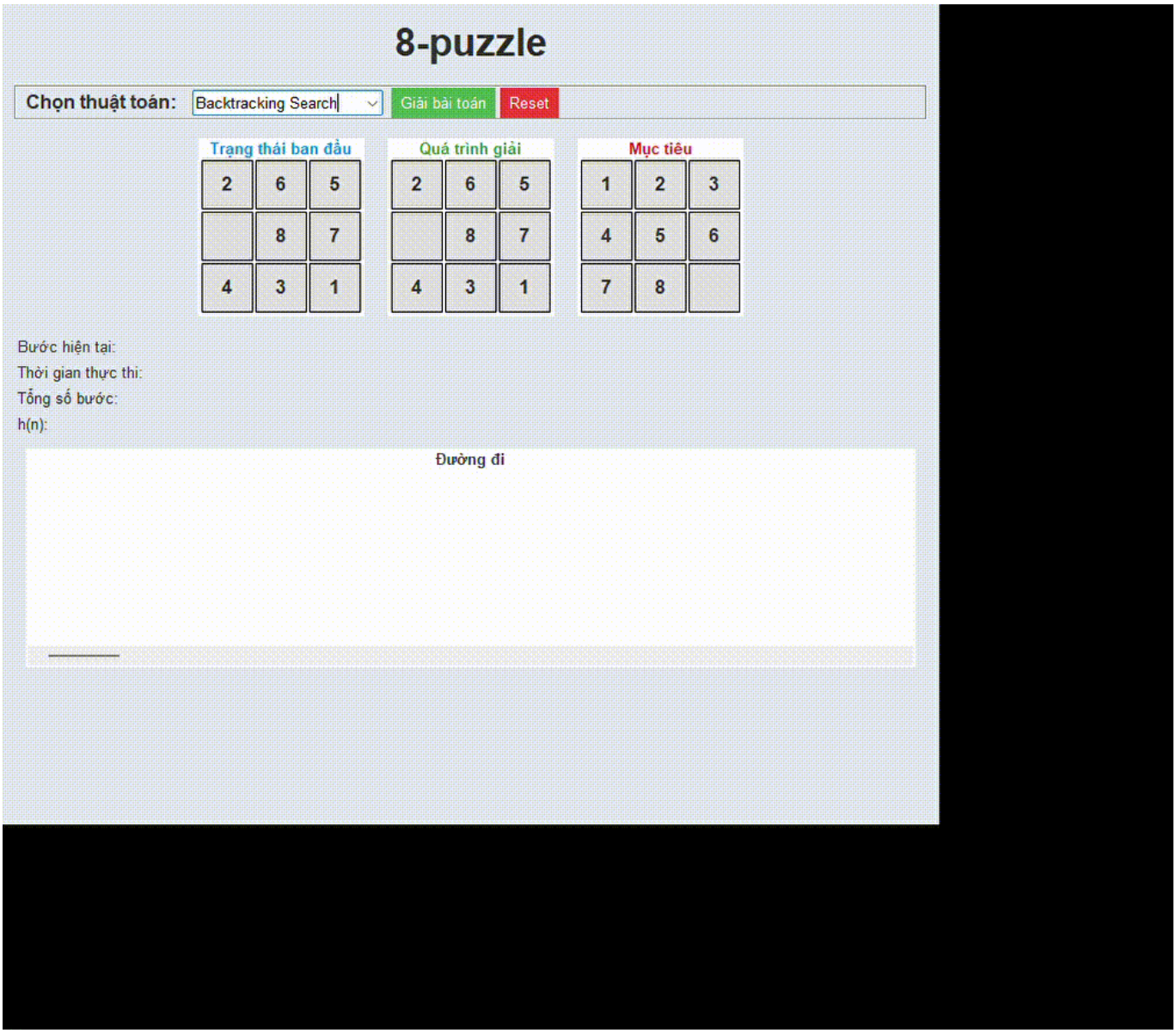


e. Thuật Toán Ràng Buộc (Constraint Satisfaction Problems - CSPs)

Backtracking with Forward Checking: Tìm kiếm quay lui với kiểm tra trước mô hình bài toán 8-puzzle như một bài toán thỏa mãn ràng buộc, sử dụng quay lui để thử các trạng thái và kiểm tra trước để giảm không gian tìm kiếm bằng cách loại bỏ các giá trị không khả thi. Thuật toán đảm bảo giải pháp nếu bài toán có lời giải, nhưng có thể chậm trong trường hợp xấu. Trong 8-puzzle, nó hiệu quả hơn quay lui đơn thuần nhờ kiểm tra trước, nhưng vẫn yêu cầu tính toán đáng kể.

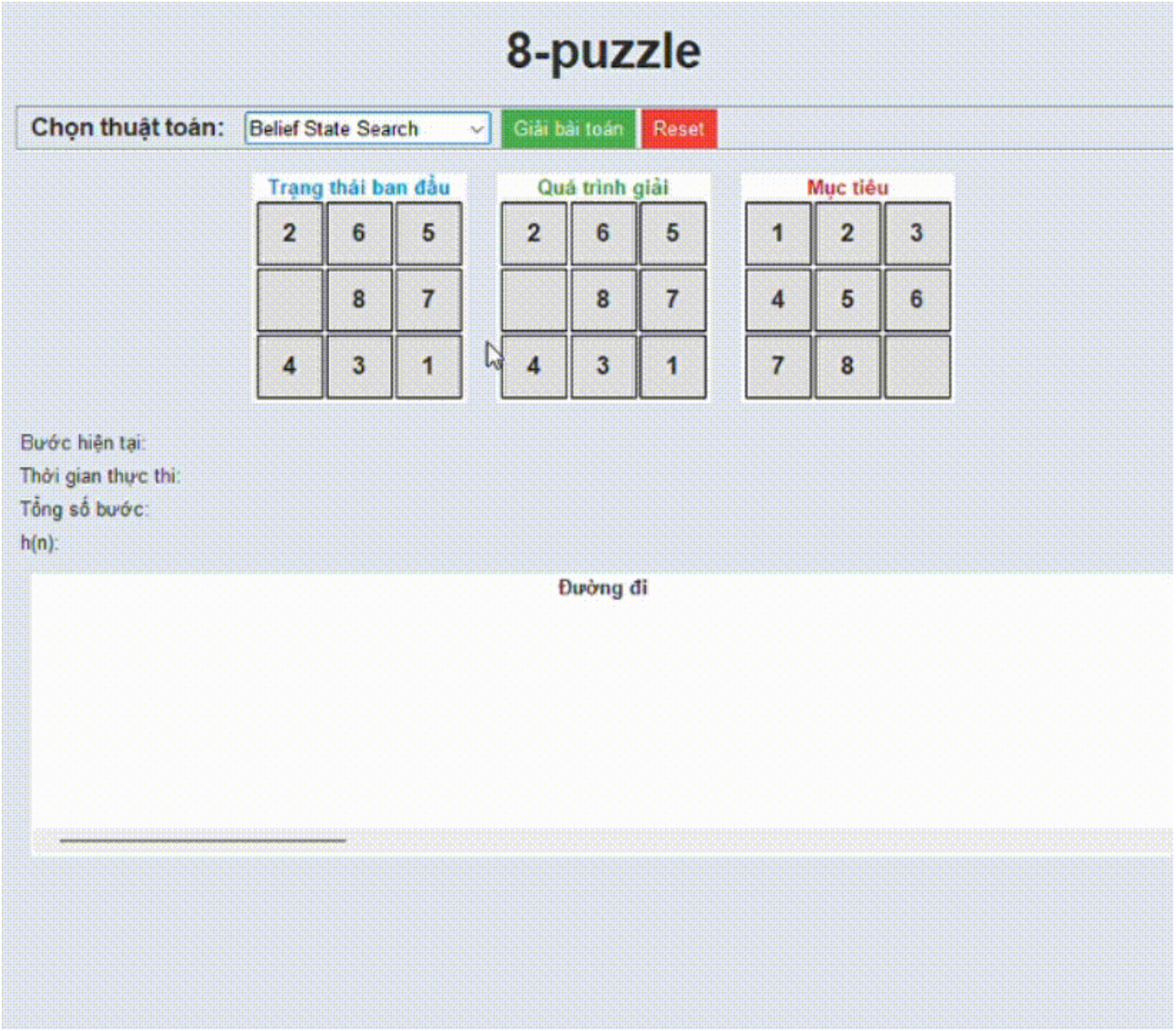


Backtracking Search: Tìm kiếm quay lui khám phá các trạng thái 8-puzzle bằng cách thử từng di chuyển và quay lại khi gặp ngõ cụt, sử dụng heuristic như khoảng cách Manhattan để ưu tiên trạng thái. Thuật toán đơn giản nhưng có thể chậm do không gian trạng thái lớn của 8-puzzle, và không đảm bảo tối ưu trừ khi kết hợp với các cải tiến. Dù có thể kết hợp với heuristic như khoảng cách Manhattan để cải thiện thứ tự duyệt, backtracking vẫn không phù hợp để giải bài toán 8-puzzle trong trạng thái này.

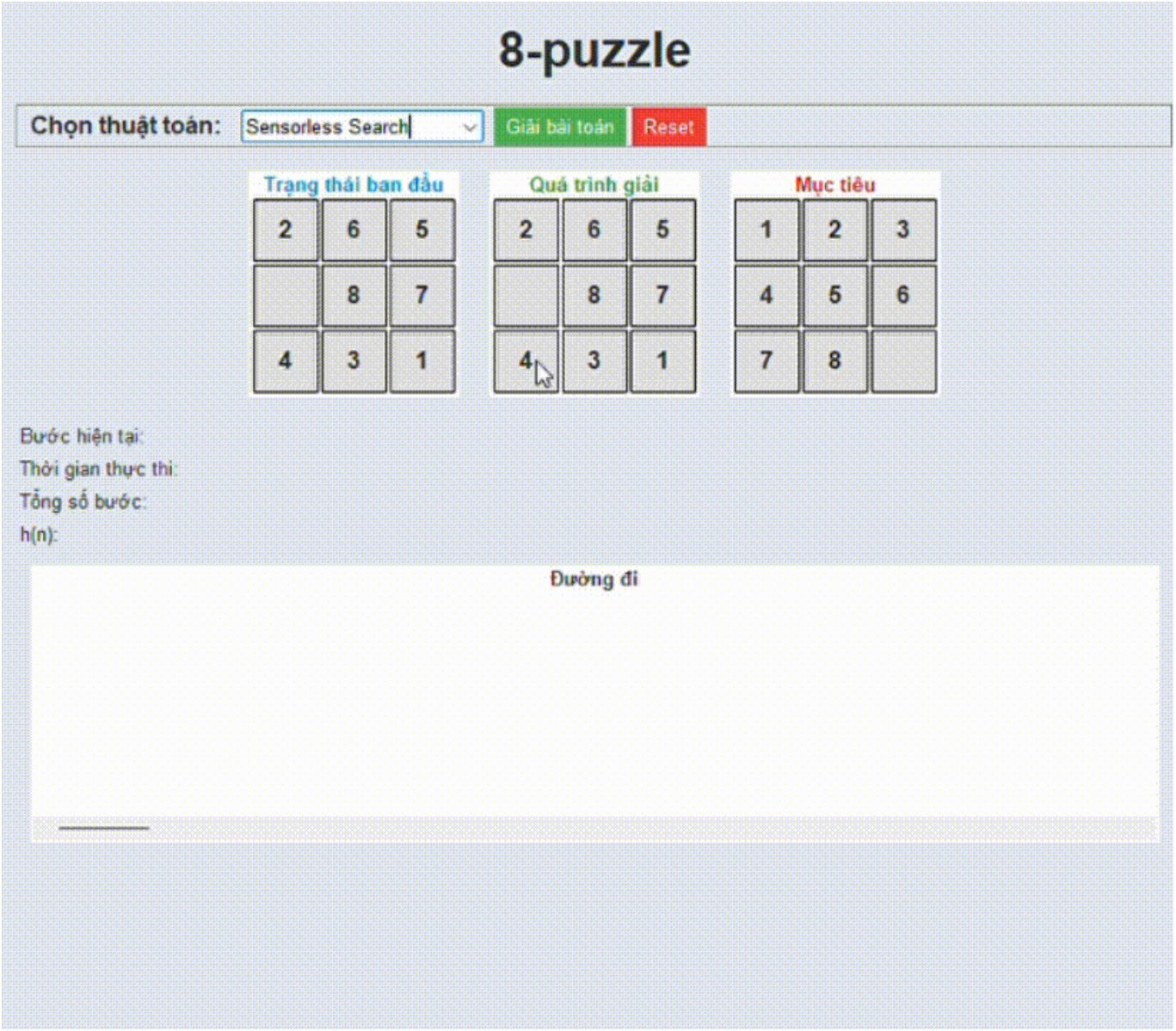


f. Thuật Toán Môi Trường Phức Tạp (Complex Environments)

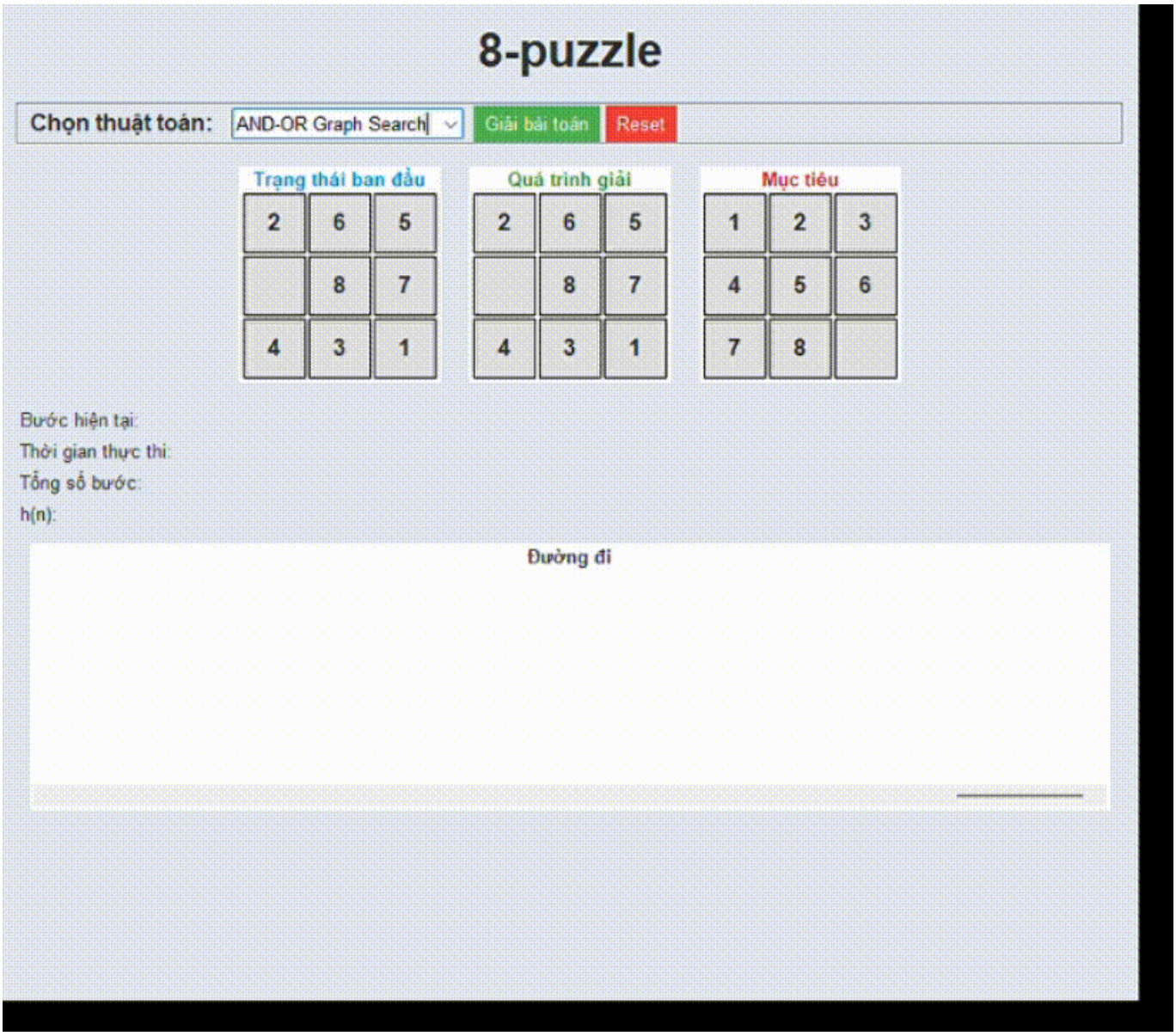
Belief State Search: Tìm kiếm trạng thái niềm tin giả định rằng trạng thái 8-puzzle hiện tại không được biết chính xác, mà thuộc một tập hợp các trạng thái có thể (niềm tin). Thuật toán mở rộng tất cả trạng thái trong tập niềm tin, sử dụng heuristic như khoảng cách Manhattan tối thiểu để hướng dẫn. Trong 8-puzzle, nó phù hợp cho các kịch bản thiếu thông tin đầy đủ, nhưng yêu cầu tính toán cao do phải xử lý nhiều trạng thái, và không luôn đảm bảo tối ưu.



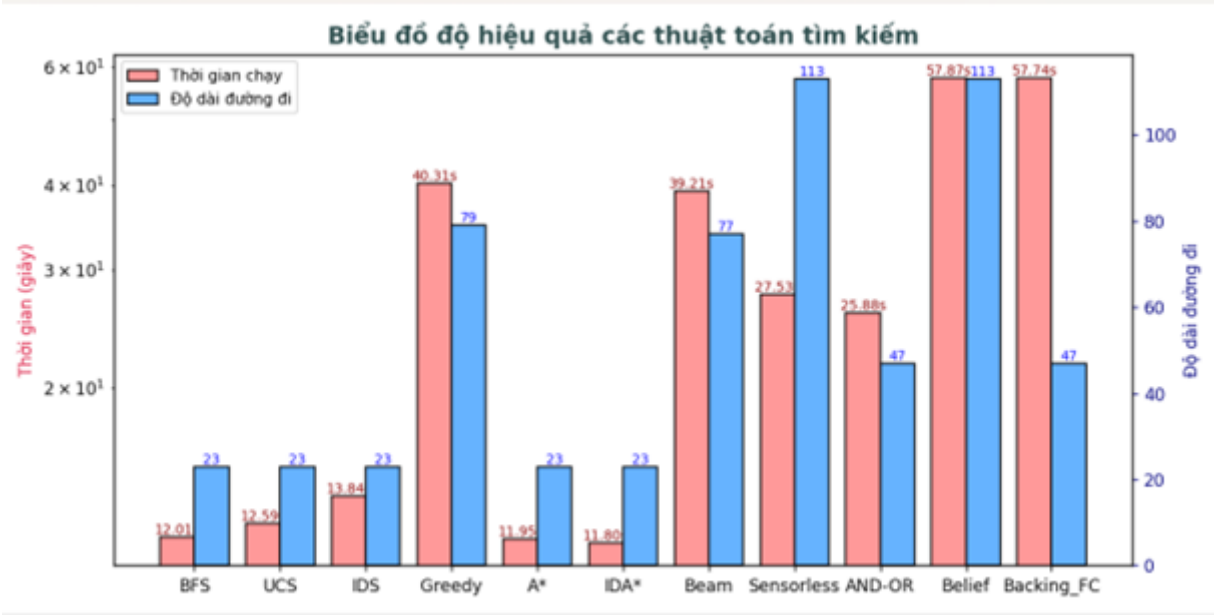
Sensorless Search: Tìm kiếm không cảm biến, tương tự tìm kiếm trạng thái niềm tin, giả định không có thông tin về trạng thái 8-puzzle hiện tại, xem tất cả trạng thái có thể là tập niềm tin ban đầu. Thuật toán cố gắng đạt mục tiêu bằng cách thu hẹp tập niềm tin qua các di chuyển. Trong 8-puzzle, nó rất tốn kém về tính toán do không gian trạng thái lớn, và thường không thực tế trừ khi bài toán được đơn giản hóa. Giải pháp không đảm bảo tối ưu.



AND-OR Graph Search: Tìm kiếm đồ thị AND-OR mô hình bài toán 8-puzzle như một đồ thị với các nút OR (trạng thái) và AND (tất cả trạng thái con phải được xem xét). Thuật toán sử dụng heuristic để ưu tiên trạng thái và quay lui khi cần. Trong 8-puzzle, nó có thể tìm giải pháp, nhưng phức tạp hơn các phương pháp khác do cấu trúc đồ thị, và không đảm bảo tối ưu trừ khi được tối ưu hóa.



4. So sánh hiệu suất của các thuật toán tìm kiếm



***Nhận xét**:

Biểu đồ thể hiện hiệu quả của các thuật toán tìm kiếm thông qua hai tiêu chí: thời gian chạy và độ dài đường đi. Nhóm thuật toán có hiệu suất cao nhất bao gồm A*, IDA* và BFS. Cụ thể, A* và IDA* đều cho thời gian chạy lần lượt là 11.95s và 11.80s, với độ dài đường đi ngắn nhất là 23 bước. Đây là hai thuật toán có định hướng tốt nhờ tận dụng heuristic để dẫn dắt tìm kiếm, đồng thời đảm bảo lời giải tối ưu. BFS cũng đạt kết quả tốt với thời gian 12.01s và độ dài đường đi 23 bước, tuy nhiên nhược điểm là tốn bộ nhớ do phải mở rộng toàn bộ các trạng thái cùng mức.

Nhóm hiệu suất trung bình gồm UCS, IDS và Beam Search. UCS và IDS đều đạt độ dài đường đi 23 bước, với thời gian lần lượt là 12.59s và 13.84s. UCS đảm bảo tìm kiếm tối ưu nhưng không định hướng, trong khi IDS khắc phục được vấn đề bộ nhớ của BFS nhờ lặp sâu tăng dần, tuy nhiên phải lặp lại nhiều lần nên tốn thời gian. Beam Search có thời gian 39.21s và độ dài đường đi 77, mặc dù chỉ mở rộng những nhánh "tốt nhất", nhưng dễ bỏ lỡ lời giải tối ưu do giới hạn số lượng trạng thái mở rộng.

Greedy Search và AND-OR thể hiện hiệu suất thấp hơn. Greedy có thời gian 40.31s và độ dài đường đi 79, trong khi AND-OR là 25.88s với độ dài 47. Cả hai thuật toán này đều dễ bị lạc hướng khi phụ thuộc hoàn toàn vào heuristic hoặc phải xử lý nhiều nhánh trong môi trường không xác định.

Cuối cùng, Sensorless là thuật toán có hiệu quả thấp nhất trong biểu đồ, với thời gian thực hiện 57.74 giây và độ dài đường đi lên tới 113 bước – dài nhất trong tất cả các thuật toán. Điều này phản ánh đúng bản chất của thuật toán khi phải hoạt động trong môi trường không chắc chắn, thiếu thông tin cảm biến rõ ràng, buộc phải xử lý tất cả các khả năng trạng thái có thể xảy ra, dẫn đến việc mở rộng nhiều nhánh và khó kiểm soát hướng đi, gây tốn thời gian và sinh lời giải không tối ưu. Tương tự, Belief cũng hoạt động trong môi trường không xác định nhưng dựa trên hệ thống trạng thái niềm tin (belief states), biểu diễn tập các trạng thái có thể xảy ra thay vì một trạng thái cụ thể. Thuật toán này có thời gian thực hiện rất cao (57.87 giây) và độ dài đường đi cũng lớn (113 bước), cho thấy việc duy trì và cập nhật tập trạng thái khiến việc tính toán phức tạp và ít hiệu quả trong tìm đường đi ngắn nhất. Trong khi đó, Backing_FC (Backtracking kết hợp Forward Checking) cho kết quả tốt hơn với thời gian 27.53 giây và độ dài đường đi 47 bước. Mặc dù thời gian giảm hơn 2 thuật toán sensorless và belief, kỹ thuật kiểm tra trước giúp loại bỏ sớm các nhánh không hợp lệ, thu hẹp không gian tìm kiếm và cải thiện đáng kể lời giải so với Belief và Sensorless, dù chi phí tính toán vẫn còn lớn.

Tổng kết lại, A*, IDA* và BFS là những lựa chọn tối ưu cho bài toán 8-Puzzle, nhờ vào khả năng định hướng tìm kiếm và đảm bảo lời giải ngắn nhất với thời gian hợp lý. Các thuật toán sử dụng heuristic không hiệu quả hoặc không kiểm soát tốt không gian tìm kiếm thường cho kết quả kém hơn cả về thời gian lẫn độ dài lời giải.

3. Kết quả đạt được

Trong quá trình xây dựng game mô phỏng bài toán 8-puzzle, em đã triển khai 20 thuật toán tìm kiếm trong môn học trí tuệ nhân tạo khác nhau trên cùng một bài toán cụ thể. Mỗi thuật toán đều được trực quan hóa quá trình giải theo thời gian thực, giúp người chơi dễ dàng quan sát cách thuật toán vận hành và ra quyết định. Dự án không chỉ dừng lại ở việc áp dụng, mà còn tập trung so sánh khách quan hiệu suất giữa các thuật toán dựa trên các tiêu chí như độ dài lời giải, thời gian thực thi. Thông qua project này, nhóm đã có cơ hội hiểu sâu hơn về cách thức hoạt động của từng nhóm thuật toán AI, bao gồm:

- Tìm kiếm không có thông tin
- Tìm kiếm có thông tin
- Tìm kiếm cục bộ
- Tìm kiếm trong môi trường phức tạp
- Tìm kiếm trong điều kiện ràng buộc

- Học tăng cường

Kết quả thực nghiệm cho thấy: các thuật toán như BFS, A* và IDA* có khả năng tìm ra lời giải ngắn nhất và ổn định nhất. Trong khi đó, các thuật toán như DFS, Simple Hill Climbing, Steepest-Ascent Hill Climbing, Stochastic Hill Climbing, Simulated Annealing, Genetic Algorithm, Backtracking Search, Q-Learning, SARSA tuy có tốc độ thực thi nhanh hơn, nhưng dễ mắc kẹt tại các trạng thái không tối ưu. Nhìn chung, dự án đã cung cấp một góc nhìn toàn diện và trực quan về hiệu quả của các thuật toán AI trong giải quyết bài toán kinh điển này.

4. Hướng phát triển

Trong tương lai, sẽ mở rộng khả năng áp dụng các thuật toán giải bài toán xếp hình sang các phiên bản có không gian trạng thái lớn hơn như 15-puzzle, cần triển khai nhiều hướng cải tiến. Trước hết, việc nâng cao hiệu suất các thuật toán hiện tại là cần thiết, chẳng hạn tối ưu hoá việc lưu trữ và xử lý trạng thái để giảm thiểu tiêu thụ bộ nhớ cũng như tăng tốc độ tìm kiếm. Bên cạnh đó, có thể tích hợp các phương pháp học máy tiên tiến như Deep Q-Network (DQN), cho phép hệ thống học chiến lược giải từ dữ liệu với khả năng tổng quát hóa cao hơn, phù hợp với các bài toán có độ phức tạp cao. Ngoài ra, xây dựng một môi trường mô phỏng đa dạng và khắt khe hơn – chẳng hạn có yếu tố nhiễu, giới hạn thời gian, hoặc ràng buộc về tài nguyên – sẽ giúp kiểm tra độ bền và khả năng thích nghi của các thuật toán. Cuối cùng, việc bổ sung công cụ trực quan hóa chi tiết tiến trình giải và chức năng so sánh giữa các thuật toán sẽ giúp đánh giá hiệu quả và hành vi của từng phương pháp một cách trực quan, hỗ trợ quá trình nghiên cứu và tối ưu hoá tốt hơn.