

Giới thiệu

Game N - Puzzle là trò chơi cổ điển với nhiều phiên bản và tên gọi khác như : 8 - Puzzle, 15 - Puzzle, ... Bài toán N - Puzzle là một trong những bài toán điển hình mô phỏng cho các giải thuật tìm kiếm liên quan đến trí tuệ nhân tạo.

- Trạng thái: mỗi trạng thái là một sắp xếp cụ thể vị trí các ô
- Hành động: mỗi hành động tương ứng với một di chuyển ô trống trái, phải, lên, xuống
- Trạng thái xuất phát: được cho trước
- Trạng thái đích: được cho một cách tường minh
- Giá thành: bằng tổng số lần dịch chuyển ô trống. Nói cách khác, mỗi chuyển động có giá thành bằng 1. Lời giải là chuỗi các hành động cho phép di chuyển từ trạng thái xuất phát tới đích. Lời giải cần ít hành động hơn là lời giải tốt hơn.

1. Mục tiêu

- Triển khai các thuật giải tìm kiếm bao gồm: tìm kiếm không thông tin (uninformed), tìm kiếm có thông tin (informed), tìm kiếm cục bộ (local search), tìm kiếm phi quyết định (non-deterministic), bài toán thỏa mãn ràng buộc (constraint satisfaction), học tăng cường (reinforcement learning), cùng với tìm kiếm trong môi trường phức tạp, nhằm giải quyết bài toán 8-puzzle. Mục tiêu giúp người dùng hiểu sâu sắc cơ chế hoạt động cũng như hiệu suất của từng thuật giải.
- Thực hiện phân tích và so sánh chi tiết hiệu quả của các thuật giải dựa trên các tiêu chí như thời gian thực thi, mức sử dụng bộ nhớ, và độ tối ưu của đường đi tìm được qua đó làm nổi bật ưu điểm và hạn chế của từng thuật giải.
- Ngoài ra cung cấp giao diện đồ họa trực quan (GUI) hỗ trợ người dùng dễ dàng theo dõi quá trình giải bài toán 8-puzzle một cách sinh động và trực quan nhất.

2. Các thuật giải tìm kiếm

2.1. Uniformed Search

Tìm kiếm không có thông tin, còn gọi là tìm kiếm mù (blind, uninformed search) là phương pháp duyệt không gian trạng thái chỉ sử dụng các thông tin theo phát biểu của bài toán tìm kiếm tổng quát trong quá trình tìm kiếm, ngoài ra không sử dụng thêm thông tin nào khác.

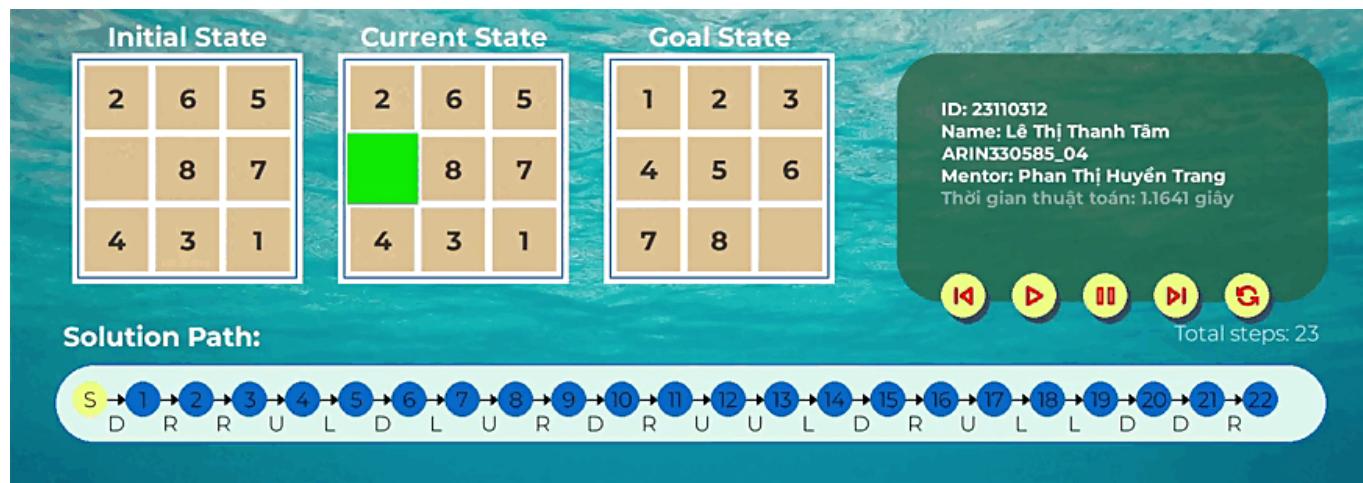
- State Space: Mỗi trạng thái là một cấu hình hợp lệ của bảng 8-puzzle, gồm 8 ô số từ 1 đến 8 và 1 ô trống (ký hiệu là 0). Tổng số trạng thái hợp lệ là **9! = 362,880**, nhưng chỉ một nửa trong số đó là khả thi (do tính chất hoán vị chẵn/lẻ).
- Initial State: Là cấu hình ban đầu của 9 ô được cung cấp.
- Operators / Actions: Các hành động di chuyển ô trống (0) theo 4 hướng: Trái (Left), Phải (Right), Lên (Up), Xuống (Down). Chỉ hợp lệ nếu không vượt khỏi biên của bảng 3x3.
- Transition Model: Sau khi thực hiện một hành động hợp lệ, ô sẽ chuyển sang trạng thái mới bằng cách hoán đổi ô trống với ô kề bên theo hướng di chuyển.
- Goal State: Là trạng thái sắp xếp đúng thứ tự mà chúng ta muốn
- Đối với tìm kiếm không thông tin như BFS, DFS, UCS, mỗi hành động thường có chi phí bằng nhau -> chi phí tổng thường là số bước từ trạng thái ban đầu đến trạng thái đích.
- Solution: Là một danh sách cách trạng thái biểu diễn đường đi từ trạng thái khởi đầu đến trạng thái mục tiêu

2.1.1. BFS (Breadth-First Search) – tìm theo bề rộng.

- BFS xem các trạng thái như là các đỉnh của một đồ thị cây với mỗi đỉnh con sẽ là trạng thái kế tiếp của đỉnh cha.

Bước thực hiện:^{*}

1. Bắt đầu từ trạng thái ban đầu, đưa nó vào hàng đợi (queue).
2. Đánh dấu trạng thái đã được duyệt (visited/closed).
3. Lặp lại đến khi hàng đợi rỗng:
 - Lấy trạng thái hiện tại từ hàng đợi.
 - Nếu như trạng thái hiện tại chính là trạng thái đích:
 - Trả về các bước đến trạng thái hiện tại
 - Duyệt từng trạng thái sinh ra từ trạng thái hiện tại:
 - Nếu trạng thái là *hợp lệ* và *chưa được duyệt*
 - Đánh dấu trạng thái đã được duyệt (đưa vào visited/closed)
 - Đưa vào hàng đợi.
4. Trả về null nếu như không có lời giải.

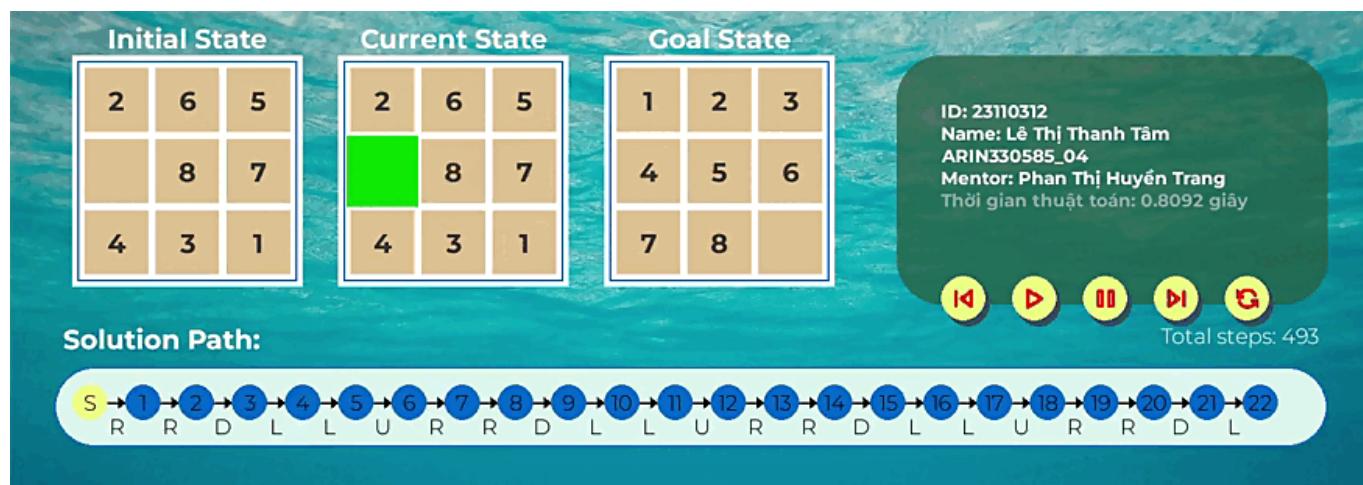


2.1.2. DFS (Depth-First Search) – tìm theo chiều sâu

- Giống như BrFS, DFS cũng coi các trạng thái như một đồ thị cây. Điểm khác biệt của DFS so với BrFS là thuật giải này sẽ duyệt hết các trạng thái có thể của một nhánh thay vì duyệt từng lớp của các nhánh.

Bước thực hiện:

- Bắt đầu từ trạng thái ban đầu, đưa nó vào ngăn xếp (stack).
- Đánh dấu trạng thái đã được duyệt.
- Lặp lại đến khi ngăn xếp rỗng:
 - Lấy trạng thái hiện tại từ ngăn xếp.
 - Nếu đường đi hiện tại vượt quá độ sâu tuyệt đối:
 - Bỏ qua trạng thái này.
 - Nếu như trạng thái hiện tại chính là trạng thái đích:
 - Trả về các bước đến trạng thái hiện tại.
 - Duyệt từng trạng thái sinh ra từ trạng thái hiện tại:
 - Nếu trạng thái là *hợp lệ* và *chưa được duyệt*:
 - Đánh dấu trạng thái đã được duyệt.
 - Đưa vào ngăn xếp.

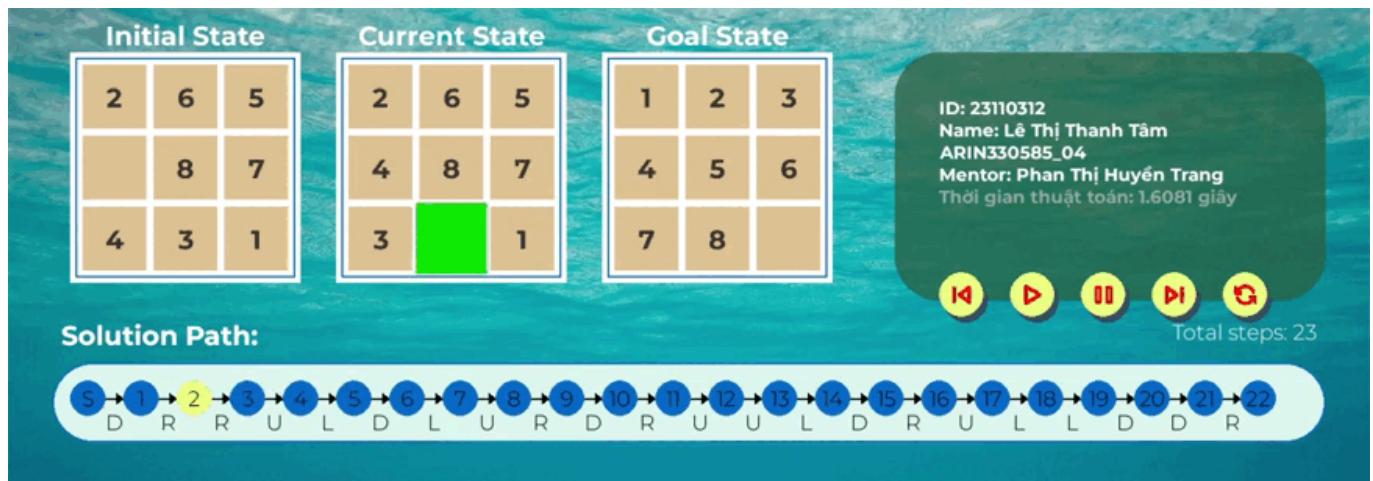


2.1.3. UCS (Uniform Cost Search) – ưu tiên trạng thái có chi phí thấp

- UCS giống như BFS nhưng có thêm yếu tố chi phí: nó luôn chọn trạng thái có tổng chi phí thấp nhất để mở rộng trước.
- Trong 8-puzzle, chi phí mỗi bước bằng nhau nên UCS sẽ cho kết quả giống BFS, nhưng UCS vẫn quan trọng khi mỗi bước có chi phí khác nhau.
- Hàm đàm bảo tìm được lời giải có chi phí thấp nhất bằng cách ưu tiên các đường đi rẻ hơn.

Bước thực hiện:

1. Bắt đầu từ trạng thái ban đầu, đưa nó vào hàng đợi ưu tiên với cost bằng 0.
2. Đánh dấu trạng thái này đã được duyệt.
3. Lặp lại đến khi hàng đợi rỗng:
 - Nếu trạng thái hiện tại là trạng thái đích:
 - Trả về các bước đến trạng thái hiện tại.
 - Duyệt qua từng trạng thái con được sinh ra:
 - Cost cho trạng thái đang duyệt tăng 1
 - Nếu như trạng thái chưa từng được duyệt lúc trước hoặc đã được duyệt nhưng có cost thấp hơn:
 - Đưa (cập nhật) trạng thái vào tập hợp các trạng thái đã được duyệt.
 - Thêm trạng thái vào trong hàng đợi.
4. Trả về rỗng nếu không có câu trả lời.



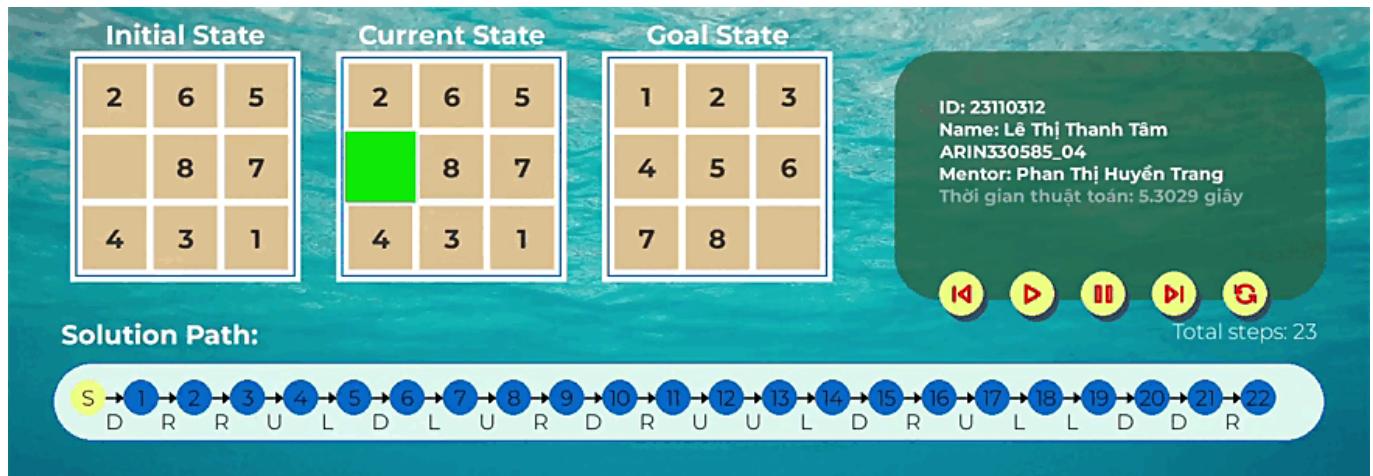
2.1.4. IDS (Iterative Deepening Search) – kết hợp DFS và BFS

Phương pháp: Tìm theo DFS nhưng không bao giờ mở rộng các nút có độ sâu quá một giới hạn nào đó. Giới hạn độ sâu được bắt đầu từ 0, sau đó tăng lên 1, 2, 3 v.v. cho đến khi tìm được lời giải.

Bước thực hiện:

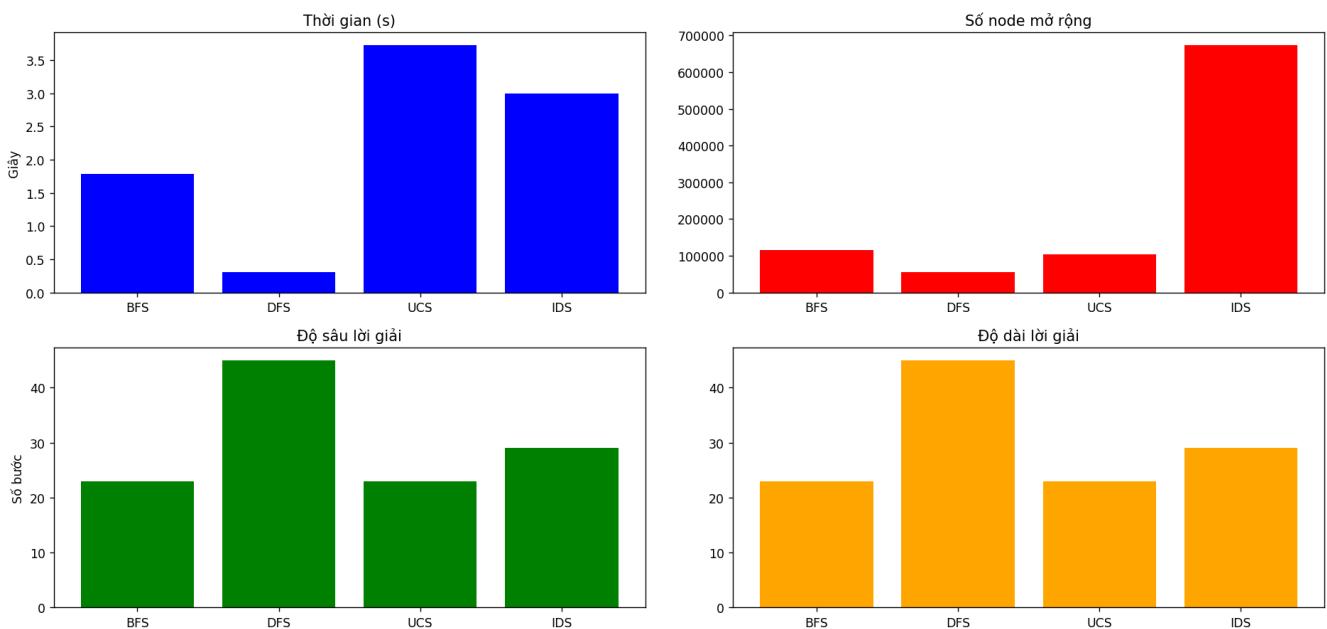
- * Hàm IDS:
 1. Duyệt từng mức độ sâu từ 1 đến max_depth:
 - Gọi hàm DLS với độ sâu depth và trạng thái hiện tại để trả về kết quả.
 - Trả về kết quả
- * Hàm DLS:
 1. Khởi tạo path, visited nếu chúng là null.
 2. Nếu trạng thái hiện tại chưa được duyệt:

- Đánh dấu là đã được duyệt.
- 3. Nếu trạng thái hiện tại là trạng thái đích:
 - Trả về các bước đến trạng thái đích.
- 4. Nếu như độ sâu là 0:
 - Trả về null
- 5. Duyệt qua các trạng thái tiếp theo được sinh ra:
 - Gọi DLS với các tham số path, visited, và độ sâu giảm đi 1
 - Nếu kết quả khác null:
 - Trả về các bước đến trạng thái hiện tại.
- 6. Trả về null nếu không có kết quả.



2.1.5. So sánh hiệu suất

So sánh hiệu suất các thuật toán Uninformed Search



- BFS và UCS đảm bảo tìm lời giải ngắn nhất nhưng tốn nhiều thời gian và mở rộng nhiều node.
- DFS nhanh nhất và mở ít node nhất nhưng lời giải dài hơn (không tối ưu).
- IDS vừa có ưu điểm tìm lời giải tương đối ngắn, nhưng tốn rất nhiều thời gian do tính chất lặp lại.

2.2. Informed Search

Chiến lược tìm kiếm có thông tin (Informed search) hay còn được gọi là tìm kiếm heuristic sử dụng thêm thông tin từ bài toán để định hướng tìm kiếm, cụ thể là lựa chọn thứ tự mở rộng nút theo hướng mau dẫn tới đích hơn. Thêm yếu tố Heuristic vào đánh giá trạng thái.

- **State Space:** Mỗi trạng thái là một cấu hình hợp lệ của bảng 8-puzzle, gồm 8 ô số từ 1 đến 8 và 1 ô trống (ký hiệu là 0).
- **Initial State:** Là cấu hình ban đầu của 9 ô được cung cấp.
- **Operators / Actions:** Các hành động di chuyển ô trống (0) theo 4 hướng: Trái (Left), Phải (Right), Lên (Up), Xuống (Down). Chỉ hợp lệ nếu không vượt khỏi biên của bảng 3x3.
- **Transition Model:** Sau khi thực hiện một hành động hợp lệ, ô sẽ chuyển sang trạng thái mới bằng cách hoán đổi ô trống với ô kề bên theo hướng di chuyển.
- **Goal State:** Là trạng thái sắp xếp đúng thứ tự mà chúng ta muốn
- **Evaluation Function:** Hàm $f(n) = g(n) + h(n)$ với:

* $g(n)$: chi phí từ trạng thái đầu đến trạng thái đang xét*

* $h(n)$: ước lượng chi phí từ trạng thái đang xét đến trạng thái mục tiêu*

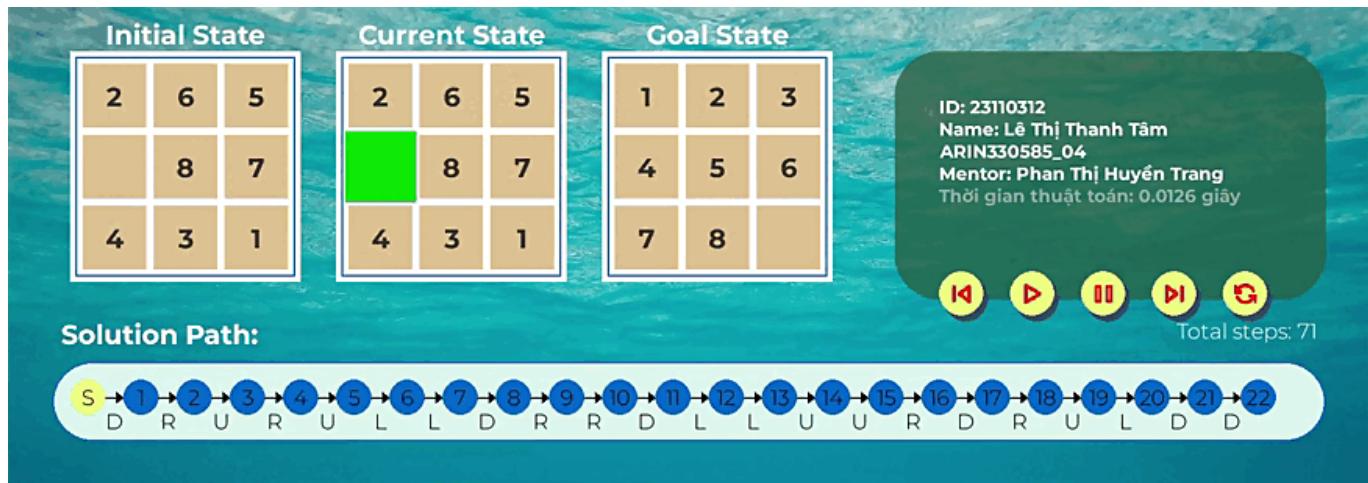
Trong bài toán 8-puzzle $h(n)$ sẽ là tổng khoảng cách Manhattan của các ô đến vị trí đúng của chúng

2.2.1. Greedy Best-First Search

- BestFS là thuật giải dựa trên khả năng lựa chọn những đường đi có vẻ "gần đích" nhất, dựa trên hàm đánh giá heuristic
- Thuật giải không quan tâm đến chi phí đã đi qua, mà chỉ tập trung ước lượng khoảng cách đến mục tiêu $h(n)$

Bước thực hiện:

1. Bắt đầu từ trạng thái ban đầu và đưa vào hàng đợi ưu tiên.
2. Thêm trạng thái ban đầu vào visited.
3. Lặp lại đến khi hàng đợi rỗng
 - Lấy giá trị có $h(n)$ thấp nhất có trong hàng đợi.
 - Nếu $current = Goal$, trả về giá trị đường đi tới kết quả.
 - Thêm $current$ vào trong visited
 - Duyệt qua các trạng thái liền kề được sinh ra:
 - Nếu trạng thái chưa được duyệt:
 - Tính $h(neighbor)$ và đưa vào hàng đợi.



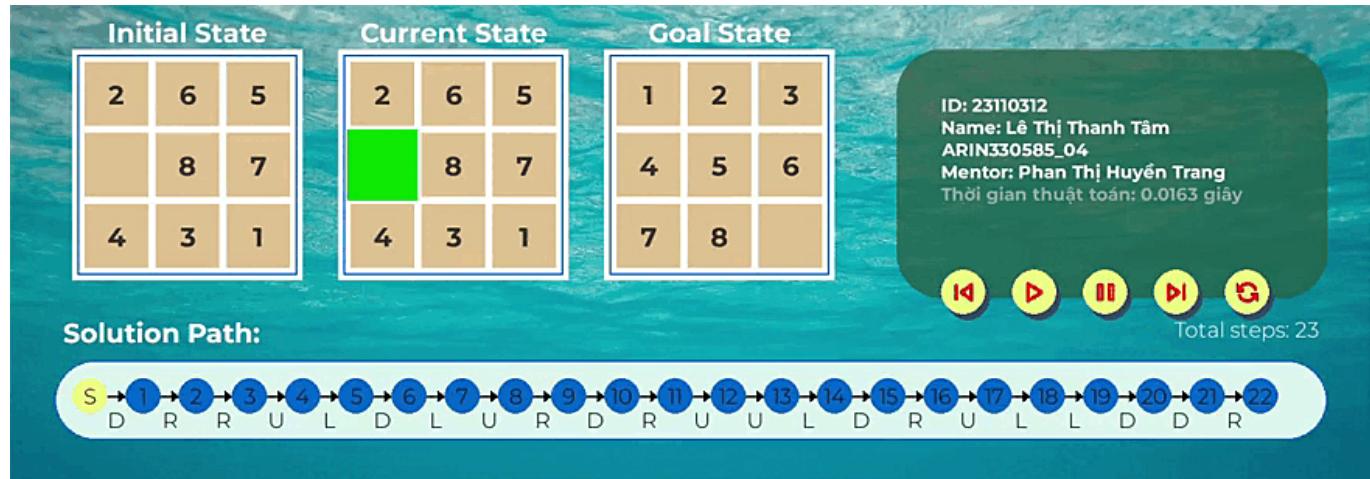
4. Trả về rỗng nếu không có kết quả.

2.2.2. A Search*

- Thuật giải A* là sự kết hợp của BestFS và Uniform Cost Search:
 - Chi phí thực tế tại nút hiện tại $g(n)$
 - Ước lượng chi phí từ nút hiện tại đến đích $h(n)$
- A* quan tâm giá trị ước lượng $f(n) = h(n) + g(n)$

Bước thực hiện

1. Bắt đầu từ trạng thái ban đầu và đưa vào hàng đợi ưu tiên với $f(\text{start}) = h(\text{start})$.
2. Thêm trạng thái ban đầu vào visited.
3. Lặp lại đến khi hàng đợi rỗng
 - Lấy giá trị có $f(n)$ thấp nhất có trong hàng đợi.
 - Nếu $\text{current} = \text{Goal}$, trả về giá trị đường đi tới kết quả.
 - Thêm current vào trong visited
 - Duyệt qua các trạng thái liền kề được sinh ra:
 - $f(n) = g[\text{current}] + \text{cost}(\text{current} \rightarrow \text{neighbor})$
 - Nếu trạng thái chưa được duyệt hoặc $f(\text{neighbor}) < g[\text{neighbor}]$:
 - Cập nhật $g[\text{neighbor}] = f(\text{neighbor})$
 - Thêm vào hàng đợi
4. Trả về rỗng nếu không có kết quả.

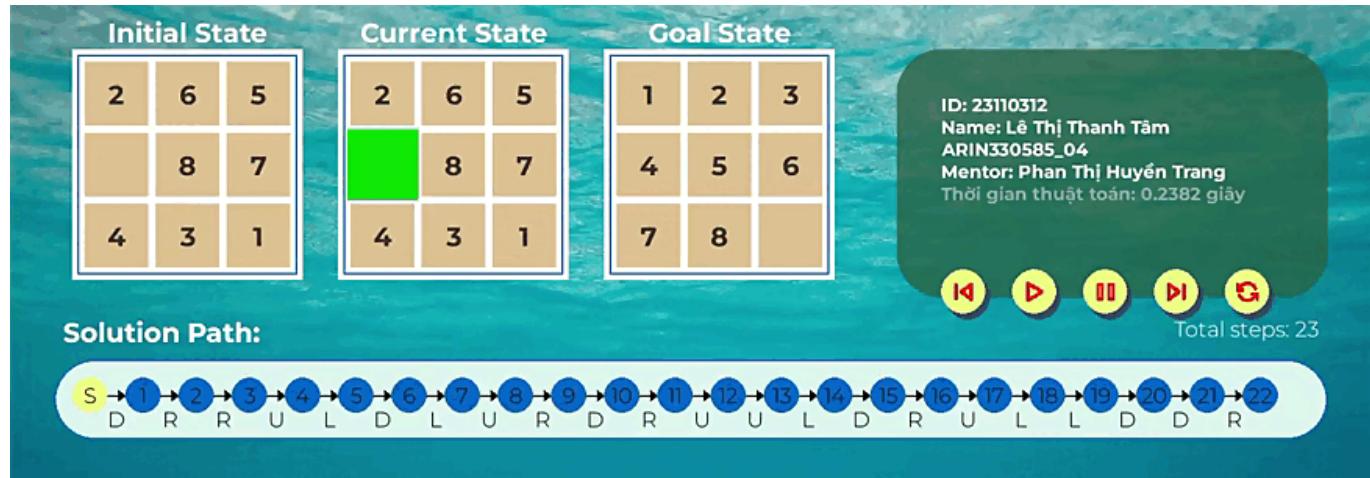


2.2.3. IDA (Iterative Deepening A)**

- IDA là sự kết hợp của A* và DLS nhằm tối ưu khả năng của cả hai thuật giải

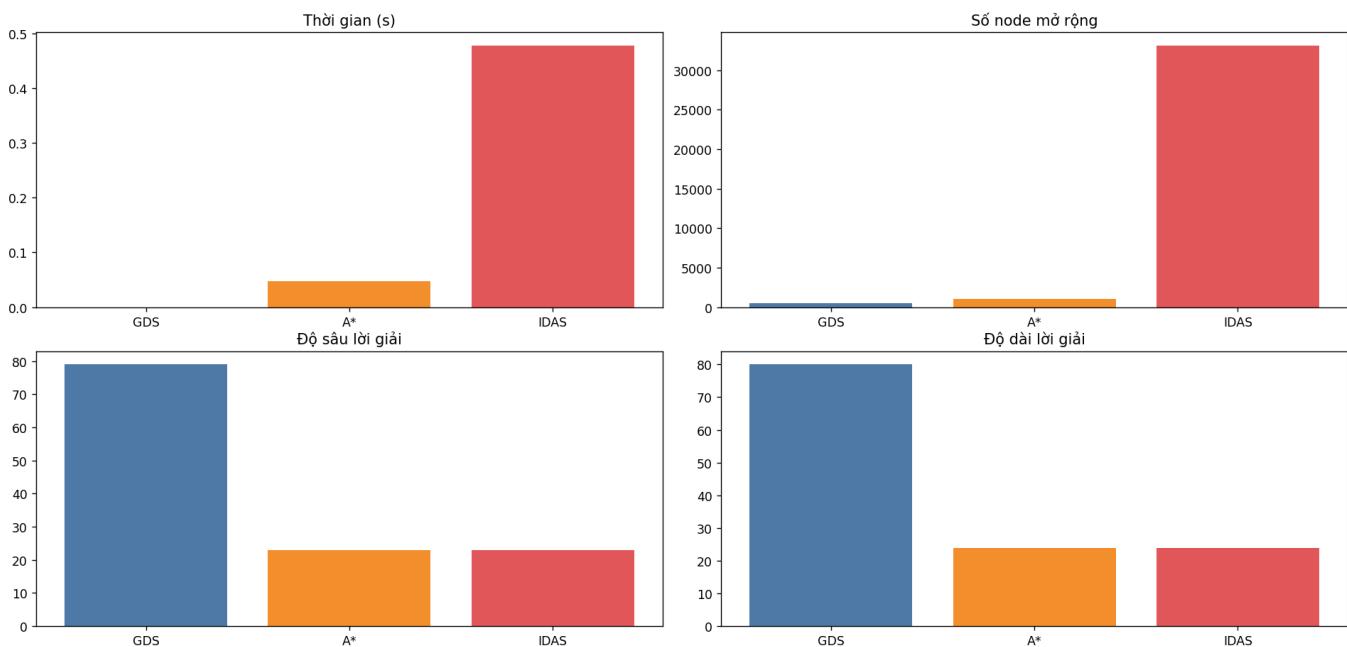
Bước thực hiện:

- Khởi tạo Threshold = $g(\text{start}) + h(\text{start})$
- Lặp lại đến khi có lời giải:
 - Gọi đệ quy DFS giới hạn theo threshold
 - Nếu tìm thấy Goal, trả về kết quả
 - Nếu không, tăng threshold lên giá trị nhỏ nhất mà vượt qua ngưỡng trong lần chạy trước
 - Nếu không có nút nào vượt ngưỡng --> không có lời giải



2.2.4. So sánh hiệu suất

So sánh hiệu suất các thuật toán Informed Search



- A* là thuật giải hiệu quả nhất, đảm bảo tìm được lời giải tối ưu nếu heuristic phù hợp.
- Greedy nhanh nhưng dễ bỏ qua lời giải tối ưu.
- IDA* tiết kiệm bộ nhớ nhưng có thể lặp lại nhiều trạng thái

2.3. Local Search

Local Search (Tìm kiếm cục bộ) là phương pháp tìm kiếm dựa trên việc bắt đầu từ một trạng thái hoàn chỉnh và tìm cách cải thiện nó bằng cách di chuyển trong lân cận (neighbor states), thay vì mở rộng toàn bộ không gian trạng thái

- **State Space:** Mỗi trạng thái là một cấu hình hợp lệ của bảng 8-puzzle, gồm 8 ô số từ 1 đến 8 và 1 ô trống (ký hiệu là 0).
- **Initial State:** Là cấu hình ban đầu của 9 ô được cung cấp.
- **Operators / Actions:** Các hành động di chuyển ô trống (0) theo 4 hướng: Trái (Left), Phải (Right), Lên (Up), Xuống (Down). Chỉ hợp lệ nếu không vượt khỏi biên của bảng 3x3.
- **Transition Model:** Sau khi thực hiện một hành động hợp lệ, ô sẽ chuyển sang trạng thái mới bằng cách hoán đổi ô trống với ô kề bên theo hướng di chuyển.
- **Goal State:** Là trạng thái sắp xếp đúng thứ tự mà chúng ta muốn
- **Solution:** Là một danh sách cách trạng thái biểu diễn đường đi từ trạng thái khởi đầu đến trạng thái mục tiêu có thể là tối ưu cục bộ hoặc tối ưu toàn cục tùy thuật toán (cần lưu ý vấn đề Local Maximum)

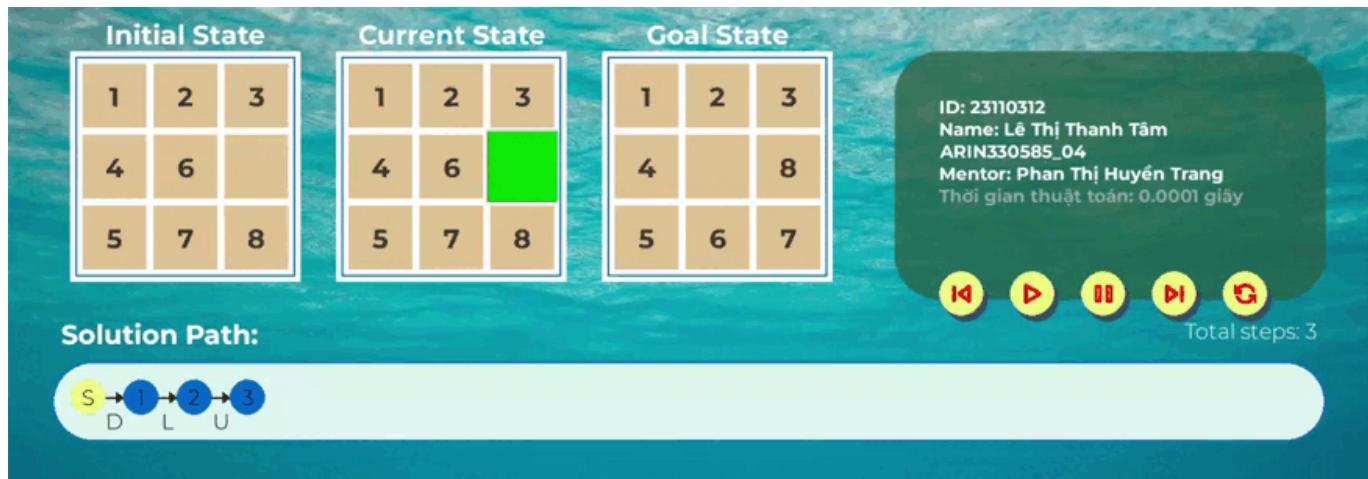
Các Thuật Toán Tìm Kiếm Dựa Trên Hill Climbing

2.3.1. Simple Hill Climbing

- Là một mở rộng của DFS có hướng dẫn bằng heuristic. Tại mỗi bước, thuật toán chỉ mở rộng trạng thái kế tiếp nếu nó **cải thiện giá trị heuristic** so với trạng thái hiện tại.
- Do luôn chọn các bước "cải thiện", thuật toán thường chạy nhanh hơn DFS thuần. Tuy nhiên, nó **không đảm bảo tìm được lời giải tối ưu** vì có thể rơi vào **cực đại địa phương (local maximum), cao nguyên (plateau) hoặc gờ núi (ridge)**.

Bước thực hiện:

1. Khởi tạo stack chứa trạng thái ban đầu.
2. Đưa trạng thái ban đầu vào tập visited.
3. Lặp cho đến khi stack rỗng:
 - Lấy trạng thái trên cùng của stack ra.
 - Tính heuristic của trạng thái hiện tại.
 - Nếu trạng thái hiện tại là đích → trả về đường đi.
 - Sinh các trạng thái con:
 - Chọn **trạng thái đầu tiên có giá trị heuristic tốt hơn** trạng thái hiện tại → thêm vào stack và visited.
 - Nếu không có trạng thái nào tốt hơn → dừng và trả về null (local maximum).
4. Nếu không tìm thấy lời giải, trả về null.

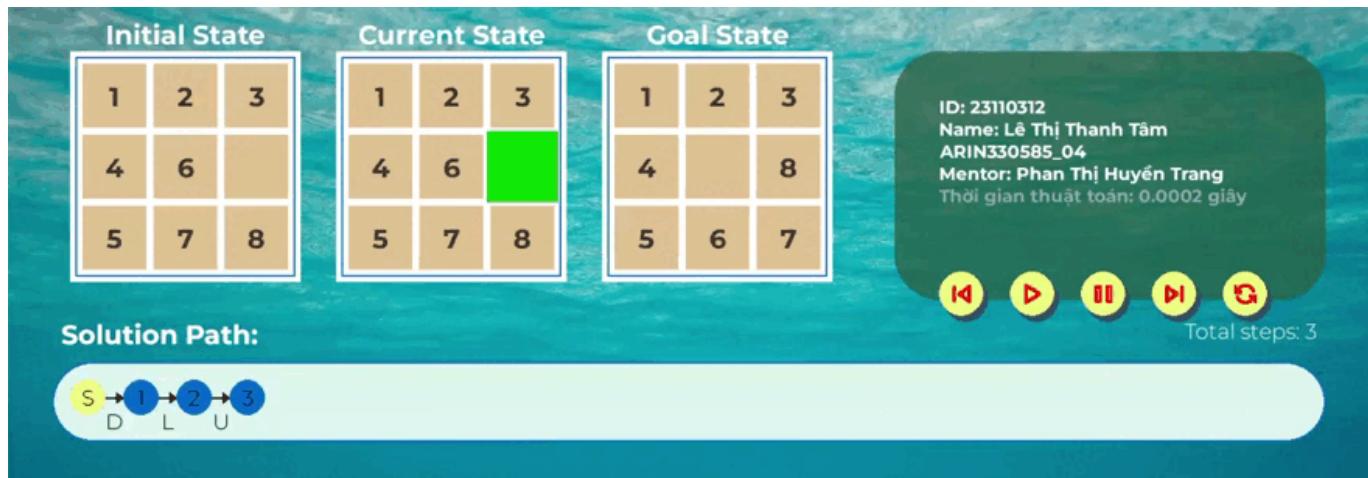


2.3.2. Steepest-Ascent Hill Climbing

- Là biến thể cải tiến của Simple Hill Climbing. Thay vì chọn **bất kỳ trạng thái con nào tốt hơn**, thuật toán sẽ **duyệt tất cả trạng thái con** và chọn **trạng thái có cải thiện lớn nhất** (steepest).
- Điều này giúp tránh các bước đi ngẫu nhiên và đưa ra lựa chọn tốt nhất tại mỗi bước.

Bước thực hiện:

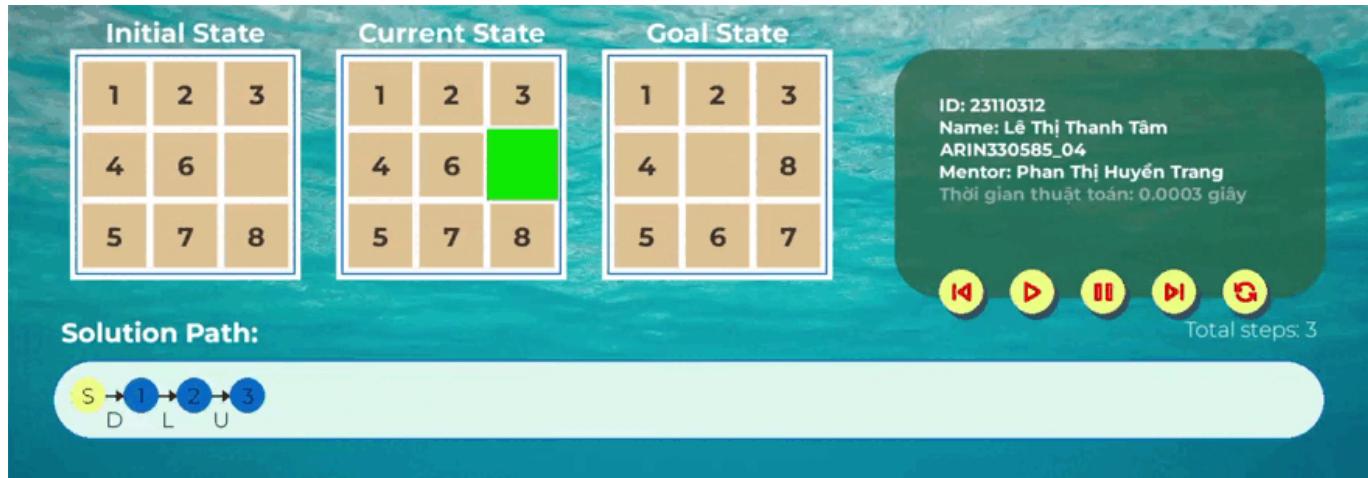
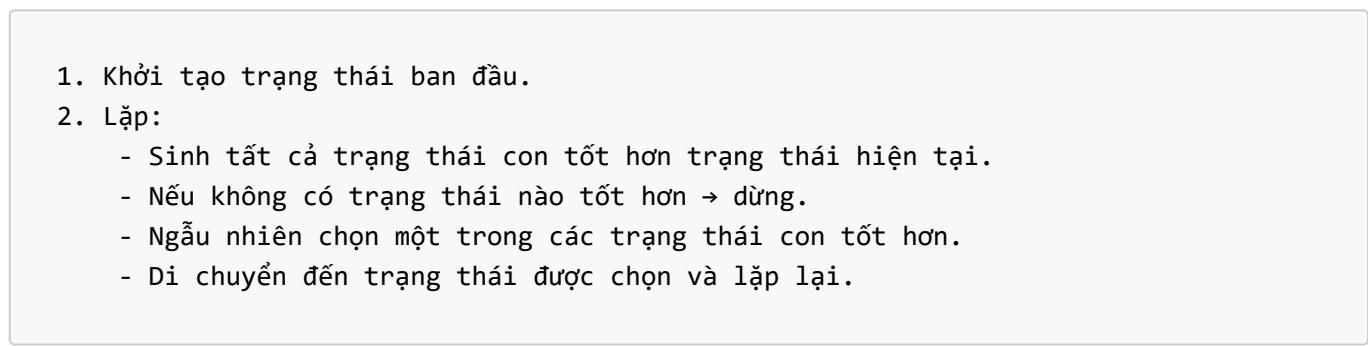
1. Khởi tạo trạng thái ban đầu.
2. Lặp:
 - Sinh tất cả trạng thái con của trạng thái hiện tại.
 - Tính heuristic cho từng trạng thái con.
 - Chọn trạng thái con **có giá trị heuristic tốt nhất**.
 - Nếu không có trạng thái nào tốt hơn → dừng và trả về trạng thái hiện tại.
 - Di chuyển tới trạng thái tốt nhất và lặp lại.



2.3.3. Stochastic Hill Climbing

- Thay vì chọn trạng thái con tốt nhất, thuật toán chọn **ngẫu nhiên một trạng thái tốt hơn** trong số các trạng thái con có cải thiện.
- Cách làm này giảm khả năng rơi vào local maximum so với Steepest-Ascent, nhưng **kết quả không ổn định**.

Bước thực hiện:

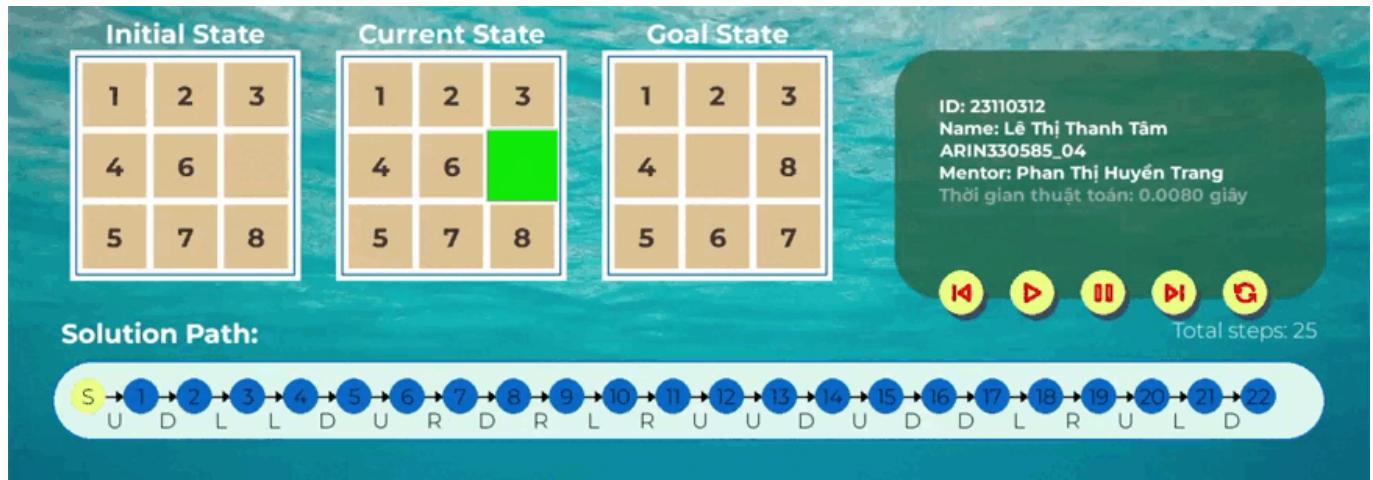


2.3.4. Simulated Annealing

- Là một biến thể của Hill Climbing sử dụng ý tưởng từ quá trình **tôi luyện kim loại** (annealing).
- Thuật toán **cho phép di chuyển tới trạng thái xấu hơn** với xác suất giảm dần theo thời gian, giúp nó **thoát khỏi local maximum**.
- Xác suất chọn trạng thái xấu hơn phụ thuộc vào một **nhiệt độ T**, và **T giảm dần theo thời gian**.

Bước thực hiện:

1. Khởi tạo trạng thái ban đầu và thiết lập nhiệt độ `T`.
2. Lặp đến khi `T` gần 0:
 - Sinh một trạng thái con ngẫu nhiên.
 - Tính $\Delta = h(\text{con}) - h(\text{hiện tại})$.
 - Nếu $\Delta < 0 \rightarrow$ chấp nhận chuyển sang con (cải thiện).
 - Nếu $\Delta > 0 \rightarrow$ chấp nhận chuyển với xác suất $\exp(-\Delta / T)$.
 - Giảm `T` theo chu kỳ.
3. Trả về trạng thái hiện tại là lời giải gần đúng.



2.3.5. Genetic Algorithm

- Không dựa trên một trạng thái mà dựa trên **tập hợp các trạng thái** (quần thể).
- Mỗi trạng thái (cá thể) được đánh giá bằng một **hàm fitness**.
- Tại mỗi thế hệ, các cá thể tốt nhất được **chọn lọc, lai ghép, và đột biến** để tạo ra thế hệ mới.
- Lặp lại quá trình này cho đến khi tìm thấy lời giải tốt hoặc đạt giới hạn số thế hệ.

Bước thực hiện:

1. Tạo quần thể ban đầu (tập hợp các lời giải ngẫu nhiên).
2. Lặp qua các thế hệ:
 - Đánh giá fitness cho từng cá thể.
 - Chọn một nhóm cá thể tốt để lai ghép.
 - Tạo cá thể mới thông qua lai và đột biến.
 - Thay thế quần thể cũ bằng quần thể mới.
3. Trả về cá thể tốt nhất tìm được.



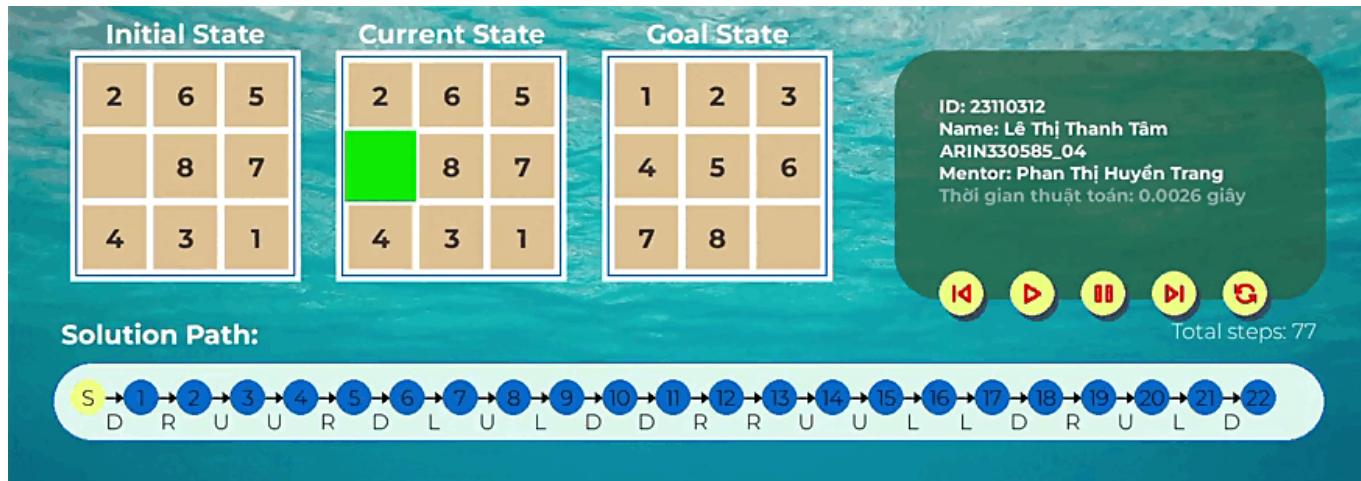
2.3.6. Beam Search

Beam Search là một thuật toán tìm kiếm có hướng dẫn (heuristic-based), tương tự như Best-First Search nhưng có cơ chế **giới hạn số lượng trạng thái mở rộng tại mỗi bước** bằng một giá trị gọi là **beam width**.

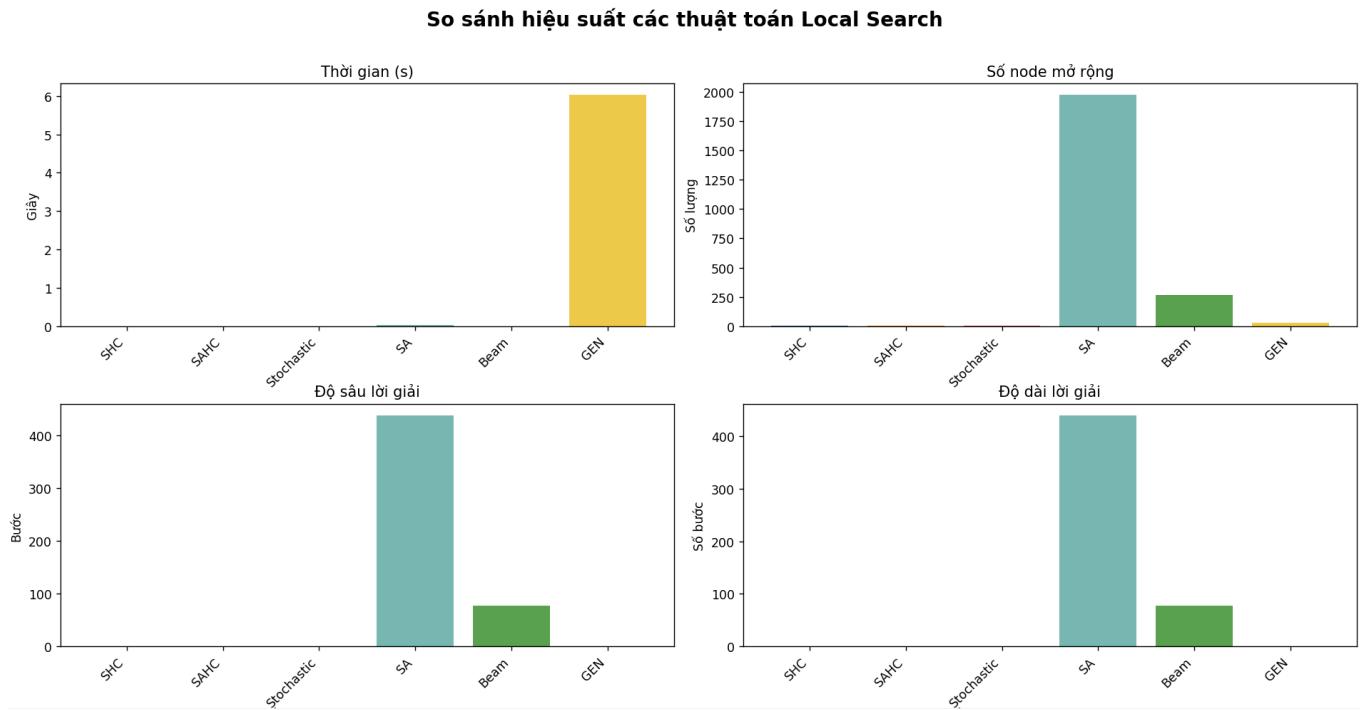
- Tại mỗi bước mở rộng, thuật toán chỉ giữ lại **k trạng thái tốt nhất** dựa trên giá trị heuristic.
- Bỏ qua các trạng thái còn lại để tiết kiệm tài nguyên.
- Beam width càng lớn → kết quả càng chính xác nhưng tốn thời gian và bộ nhớ hơn.

Bước thực hiện:

1. Khởi tạo danh sách trạng thái hiện tại với trạng thái ban đầu.
2. Lặp lại cho đến khi tìm thấy trạng thái goal hoặc danh sách trạng thái rỗng:
 - Sinh tất cả trạng thái con từ các trạng thái hiện tại.
 - Tính heuristic cho từng trạng thái con.
 - **Chọn `k` trạng thái con có heuristic tốt nhất** → làm danh sách trạng thái cho bước tiếp theo.
3. Nếu tìm thấy trạng thái goal → trả về đường đi đến đó.
4. Nếu không còn trạng thái nào để mở rộng → trả về null.



2.3.7. So sánh hiệu suất



- SHC : Thường gặp Local Maximum do trạng thái ban đầu quá phức tạp
- SAHC : Có cải thiện hàm chọn **Hàng Xóm** nhưng vẫn gặp phải Local Maximum
- Stochastic: Chọn ra **Hàng Xóm** ngẫu nhiên trong các hàng xóm tốt nhất nhưng số lượng node mở rộng không đủ để thoát Local Maximum
- SA : Có thể thoát LM nhờ chấp nhận trạng thái xấu để đi tiếp
- Gen : Số node mở rộng thấp cho thấy hàm **fitness**, **crossover** hay **mutation** chưa đạt hiệu quả, không đủ số lượng thế hệ để có thể tìm thấy lời giải
- BS : Tìm lời giải nhanh dựa vào mở rộng các node với **beam width** nhất định nhưng trong một số trường hợp có thể bỏ qua lời giải tối ưu

2.4. Search in Complex Environment

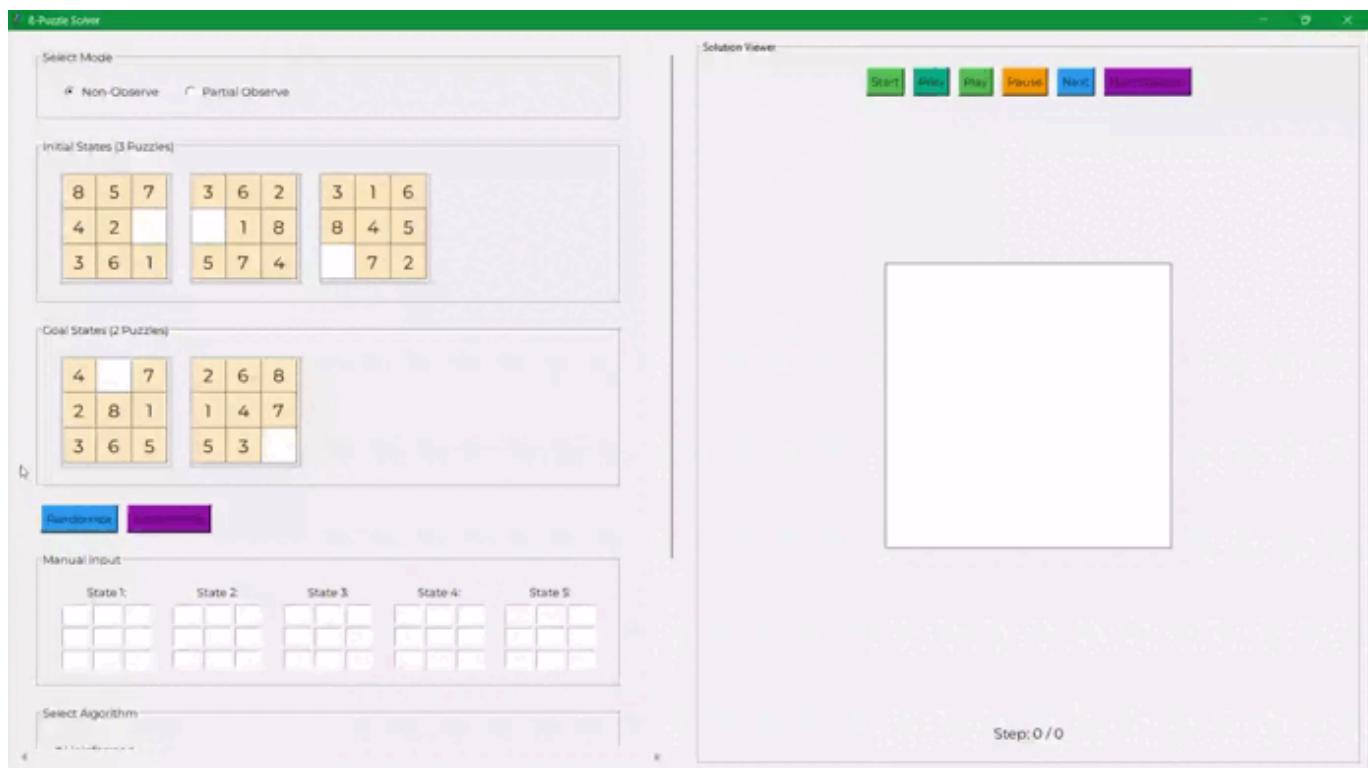
Tìm kiếm trong môi trường phức tạp là một lĩnh vực quan trọng trong trí tuệ nhân tạo, nơi các thuật giải phải đối mặt với nhiều yếu tố không chắc chắn và biến động. Môi trường phức tạp có thể bao gồm nhiều trạng thái, các yếu tố tương tác, và các ràng buộc khó khăn, đòi hỏi các phương pháp tìm kiếm phải linh hoạt và hiệu quả. Đặc điểm của Môi trường Phức tạp

- Nhiều trạng thái: Môi trường có thể có hàng ngàn trạng thái khác nhau làm cho việc tìm kiếm trở nên khó khăn hơn.
- Tương tác giữa các yếu tố: Các yếu tố trong môi trường có thể tương tác với nhau ảnh hưởng đến quyết định và kết quả.
- Tính chắc chắn: Thông tin có thể không đầy đủ hoặc không chính xác, yêu cầu các thuật giải phải xử lý sự không chắc chắn này.
- Thay đổi theo thời gian: Môi trường có thể thay đổi theo thời gian, yêu cầu các thuật giải phải thích ứng nhanh chóng.
- Đối với 2 môi trường quan sát trên thì ta áp dụng các thuật toán của 3 nhóm Uninformed, Informed và Local để giải quyết

2.4.1. Non-Observable Environment (Môi trường không quan sát được)

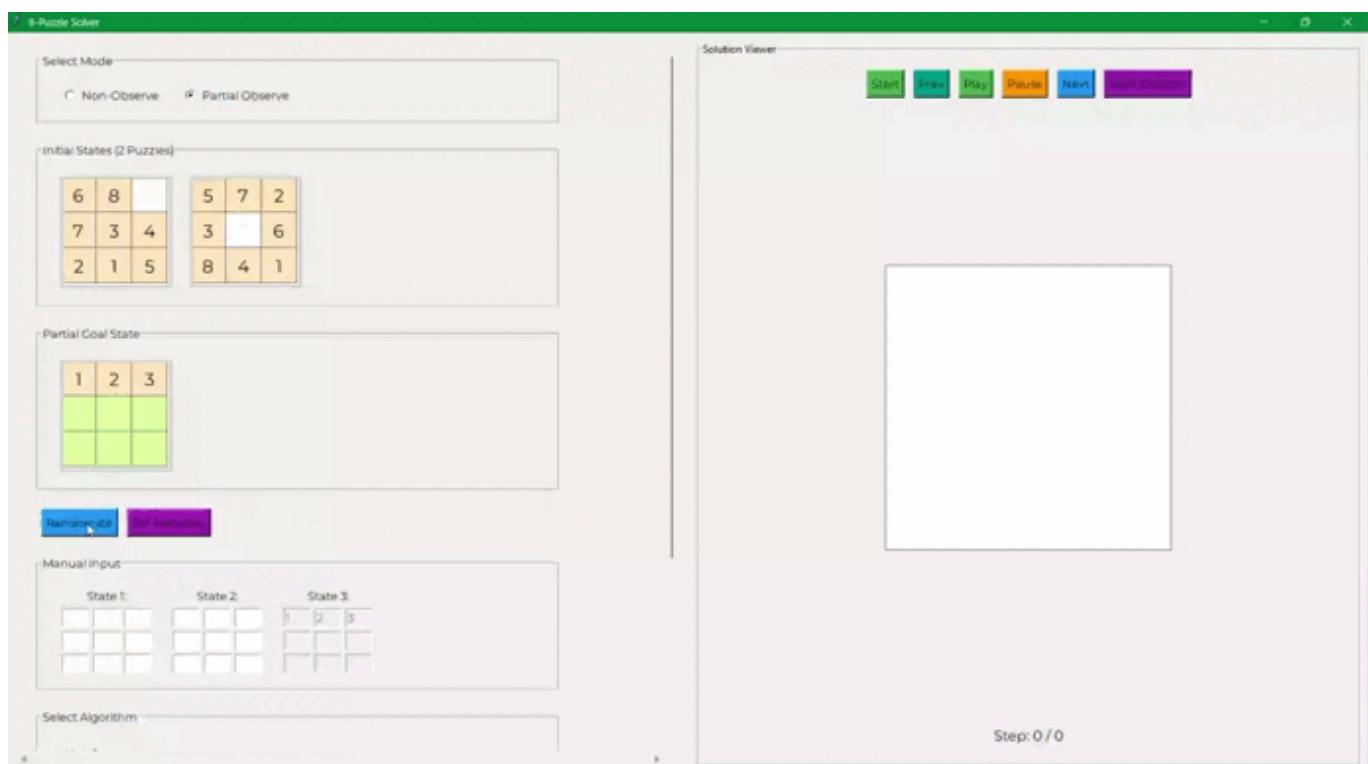
- Agent không có khả năng quan sát trạng thái hiện tại.

- Hành vi thường là "tìm kiếm mù quáng" dựa vào các giả định hoặc mô hình xác suất.
- Initial State: Tình huống agent không biết vị trí ban đầu, nhưng có một tập các trạng thái khởi đầu.
- Goal State: Tập trạng thái mục tiêu mà agent tin rằng mình sẽ đi được đến đó



2.4.2. Partially Observable Environment (Môi trường quan sát một phần)

- Agent chỉ có thể quan sát một phần trạng thái thực tế.
- Initial State: Tình huống agent không biết vị trí ban đầu, nhưng có một tập các trạng thái khởi đầu.
- Goal State: Tập trạng thái mục tiêu hay trạng thái mục tiêu mà agent nhìn thấy được một phần



2.4.3. AND-OR Search

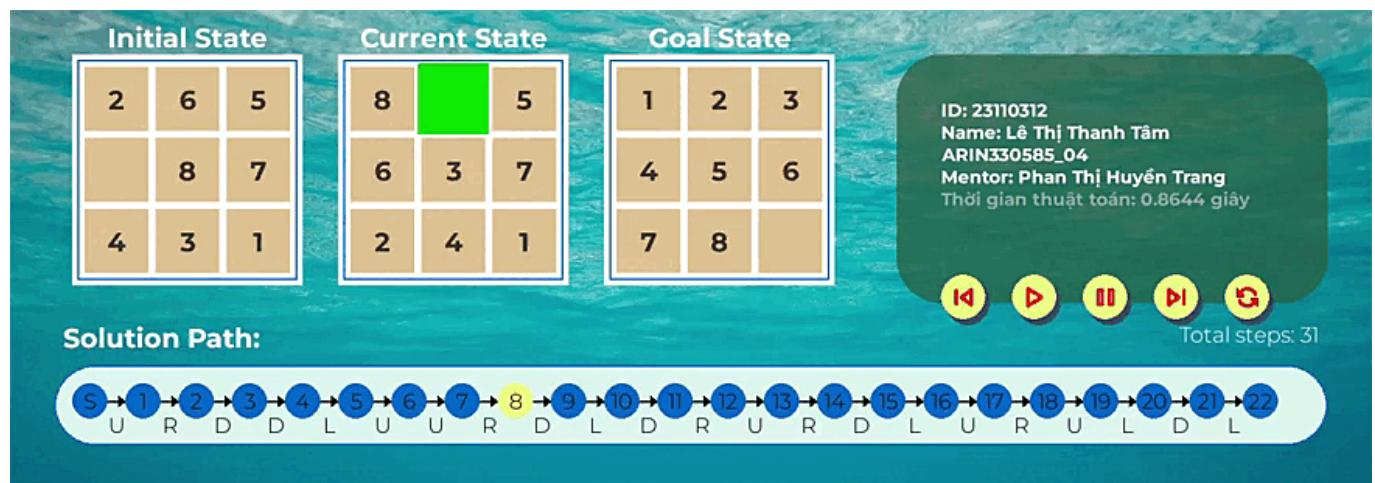
AND-OR Graph Search là một kỹ thuật tìm kiếm đặc biệt được sử dụng trong các môi trường không chắc chắn hoặc có nhiều kết quả có khả quan cho mỗi hành động. Khác với các thuật toán tìm kiếm khác chỉ quan tâm đến một chuỗi hành động đơn lẻ, thuật toán xử lý nhiều trường hợp có thể xảy ra ở mỗi bước đi.

Node **OR**: Đại diện cho điểm ra quyết định – chọn một trong các hành động có thể dẫn đến thành công.

- Agent có thể chọn di chuyển lên, xuống, trái, phải → chọn một hướng đi phù hợp nhất.

Node **AND**: Đại diện cho các kết quả không thể tránh khỏi – cần xử lý tất cả các kết quả để đảm bảo thành công.

- Nếu di chuyển sang trái có thể dẫn đến 2 trạng thái khác nhau do môi trường không ổn định, thì cả hai kết quả đều phải xử lý được. Trong các môi trường phức tạp, di chuyển 1 bước có thể dẫn đến nhiều trạng thái khác nhau



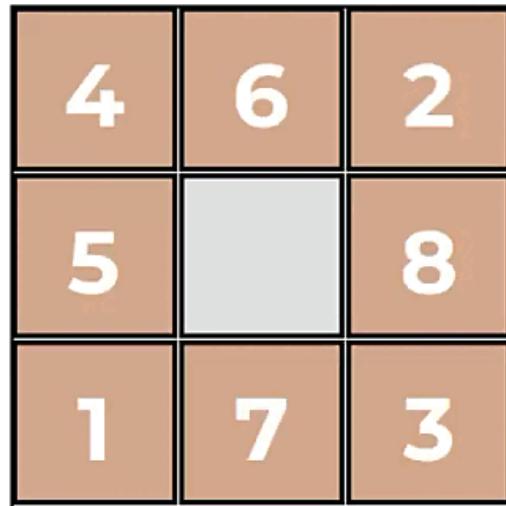
2.5. Constraint Satisfaction Problem (CSP)

Bài toán tìm trạng thái thỏa mãn các ràng buộc (khác với tối ưu đường đi).

- Biến: Vị trí các con số trong ma trận 3x3.
- Giá trị: Giá trị số từ 1 đến 8 (và 0).
- Các ràng buộc: không được trùng giá trị, không được bỏ trống sắp xếp đúng thứ tự: hàng ngang số bên phải lớn hơn số bên trái 1 đơn vị, hàng dọc số ở dưới lớn hơn số ở trên 3 đơn vị
- State Space:** Tập các gán giá trị cho biến thỏa mãn tất cả ràng buộc.
- Initial State:** Biến chưa được gán giá trị.
- Operators / Actions:** Gán giá trị hợp lệ cho biến theo thứ tự. Transition Model: Cập nhật các ràng buộc và loại bỏ các giá trị không phù hợp.
- Goal State:** Gán giá trị đầy đủ cho các biến thỏa mãn mọi ràng buộc.
- Solution:** Một tập các giá trị biến thỏa mãn toàn bộ ràng buộc.

2.5.1. Kiểm thử

- Sau mỗi lần gán giá trị cho biến, kiểm tra ngay ràng buộc giữa các biến để tránh vi phạm. Nếu một biến không còn giá trị hợp lệ nào → backtrack sớm.
- Vì gán giá trị cho 9 biến cùng 1 lúc nên khả năng tìm được lời giải khá thấp

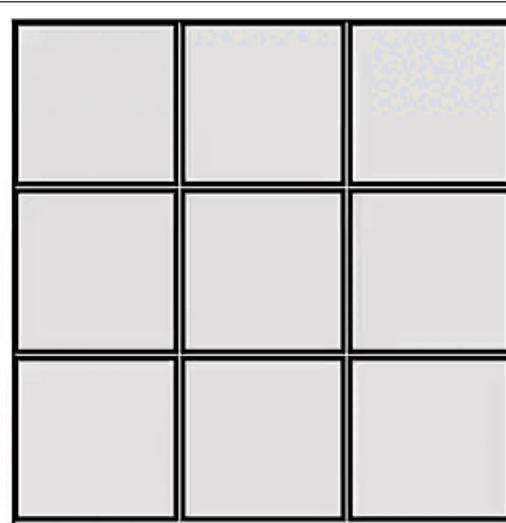


Đang tạo trạng thái...

Lượt thử: 29

2.5.2. Backtracking

- Gán giá trị từng biến theo thứ tự.
- Nếu gán hợp lệ, tiếp tục; nếu vi phạm ràng buộc, quay lui (backtrack) để thử giá trị khác.
- Có thể dùng để tìm cấu hình trung gian hợp lệ giữa trạng thái ban đầu và đích.
- Hiệu quả với AC3 hoặc Forward Checking kết hợp.



Đã reset puzzle.

Lượt thử: 0

2.5.3. AC-3

- Duyệt qua tất cả cặp biến có ràng buộc → loại bỏ các giá trị không thỏa mãn.
- Khi một biến bị rút gọn domain, các biến có liên quan cũng phải cập nhật lại.
- Giảm không gian tìm kiếm trước khi chạy Backtracking.



2.6. Reinforcement Learning

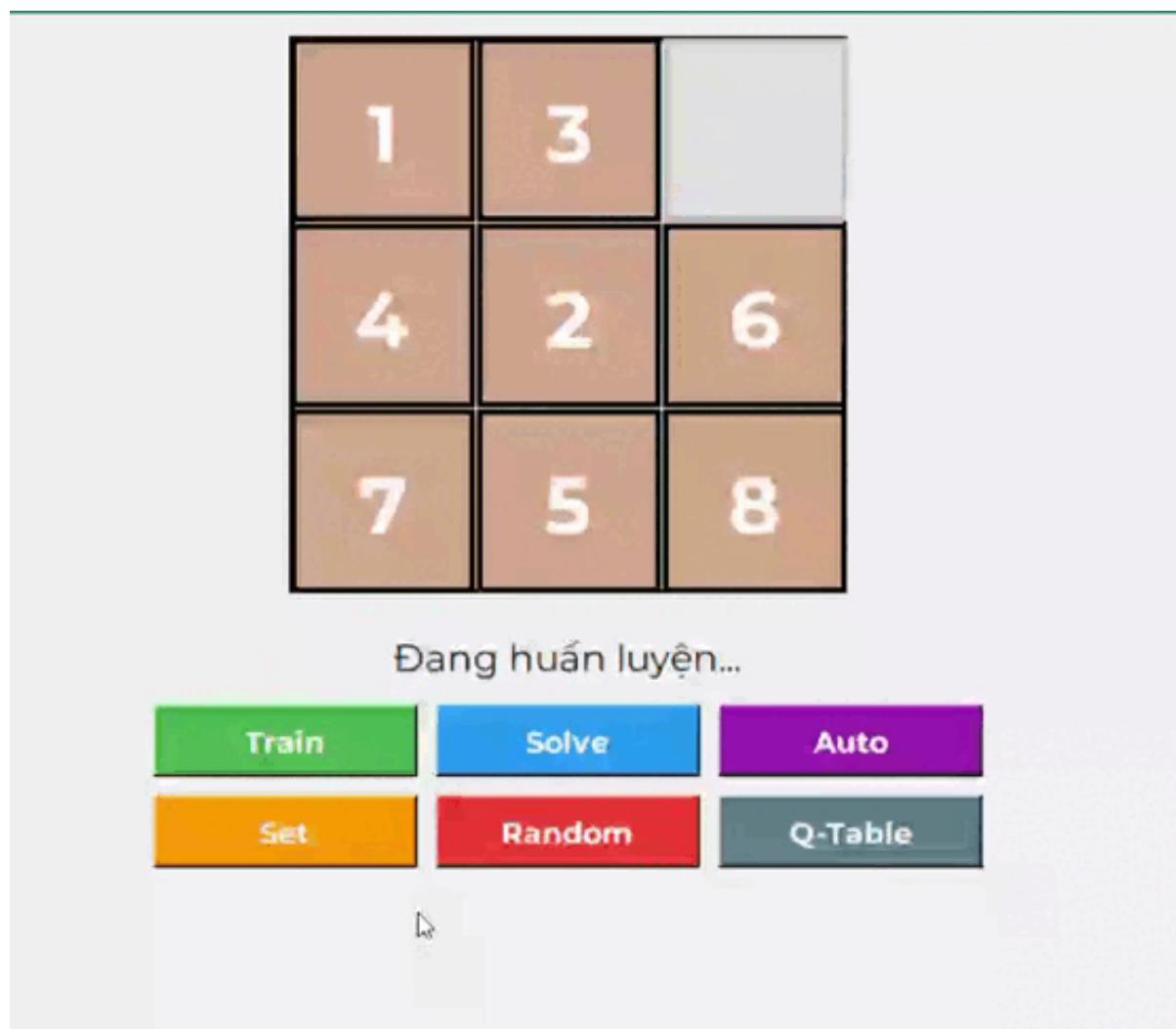
- **Agent:** Thực thể học đưa ra các hành động
- **Environment:** Không gian trạng thái là toàn bộ puzzle
- **State:** Một trạng thái puzzle cụ thể
- **Action:** Các hành động di chuyển Lên, Xuống, Trái, Phải
- **Reward:** Phần thưởng khi đạt được hoặc tiến gần hơn đến trạng thái mục tiêu

* $Q(s, a)$: giá trị hành động a tại trạng thái s .

- Học qua thử-sai (trial and error): chọn hành động → quan sát kết quả → cập nhật Q .

Bước thực hiện:

1. Khởi tạo Q-table trống.
2. Trong mỗi episode:
 - Bắt đầu từ trạng thái ngẫu nhiên.
 - Chọn hành động bằng ϵ -greedy.
 - Cập nhật Q-value:
$$Q(s, a) = Q(s, a) + \alpha [r + \gamma * \max_{a'} Q(s', a') - Q(s, a)]$$
 - Dừng khi đến trạng thái đích hoặc vượt quá số bước giới hạn.
3. Sau khi huấn luyện: chọn hành động theo Q-table để giải.



3. Kết luận

Thông qua quá trình triển khai và so sánh 20 thuật toán tìm kiếm áp dụng vào bài toán 8-puzzle, em nhận thấy:

- Thuật toán tìm kiếm không thông tin (BFS, DFS, UCS, IDS) đơn giản, dễ cài đặt nhưng không hiệu quả với không gian trạng thái lớn do mở rộng các node dư thừa.
- Thuật toán tìm kiếm có thông tin (Greedy, A*, IDA*) sử dụng heuristic để hướng dẫn tìm kiếm, từ đó giúp giảm thời gian và số lượng trạng thái cần mở rộng.
- Tìm kiếm cục bộ như Hill Climbing, Simulated Annealing, Beam Search khai thác sự tối ưu cục bộ nhưng có nguy cơ mắc kẹt ở điểm kém tối ưu (Local Maximum).
- Thuật toán tiến hóa như Genetic Algorithm hoạt động trên quần thể và có khả năng tìm lời giải tốt qua các thế hệ
- Thuật toán học tăng cường (Q-learning) cho phép agent học từ tương tác với môi trường, thể hiện tiềm năng lớn trong các môi trường phức tạp và không chắc chắn.
- Các kỹ thuật như Backtracking kết hợp AC-3 giúp rút gọn không gian tìm kiếm hiệu quả khi giải quyết bài toán dạng CSP như Sudoku, CPU Scheduling,...

- Ngoài ra, việc mở rộng giải thuật cho các môi trường phức tạp (Non-observable, Partially Observable, AND-OR Search) thể hiện khả năng thích ứng và mở rộng của các phương pháp này trong thực tế, nơi thông tin có thể không đầy đủ hoặc không chắc chắn.
- Nhìn chung, mỗi thuật toán đều có ưu điểm và hạn chế riêng. Việc lựa chọn thuật toán phù hợp phụ thuộc vào tính chất bài toán, giới hạn tài nguyên và yêu cầu về hiệu quả. Việc thực nghiệm với nhiều chiến lược khác nhau giúp em hiểu rõ cách mà trí tuệ nhân tạo đưa ra quyết định trong các môi trường đa dạng.

Tài liệu tham khảo

- [1]. Phan Thị Huyền Trang, Slide Bài giảng Nhập môn Trí tuệ nhân tạo, Trường Đại Học Sư Phạm Kỹ Thuật TPHCM, 2025.
- [2]. Từ Minh Phương, Giáo trình Nhập môn Trí tuệ nhân tạo, Học viện Bưu chính viễn thông, 2014.
- [3]. Russell, S. J., & Norvig, P. (2021), Artificial intelligence: A modern approach (4th ed.), Pearson.