# BÁO CÁO ĐỒ ÁN CÁ NHÂN MÔN TRÍ TUỆ NHÂN TẠO

# 8-PUZZLE SOLVER WITH AI ALGORITHMS

**Giảng viên hướng dẫn:** TS. Phan Thị Huyền Trang
**Sinh viên thực hiện:** Nguyễn Tấn Yên
**Mã số sinh viên:** 23110273
**Mã lớp học:** ARIN330585_04

## MỤC LỤC

---

# 1. PHÂN TÍCH VÀ ĐỀ RA MỤC TIÊU DỰ ÁN

Dự án 8-Puzzle Solver with AI Algorithms hướng đến việc triển khai và so sánh đa dạng các thuật toán trí tuệ nhân tạo nhằm giải quyết bài toán 8-Puzzle - một bài toán kinh điển trong lĩnh vực AI và khoa học máy tính.

Cụ thể:

- Triển khai các thuật toán AI đa dạng: Tích hợp các nhóm thuật toán tìm kiếm khác nhau bao gồm tìm kiếm không thông tin, tìm kiếm có thông tin, tìm kiếm cục bộ, tìm kiếm trong môi trường phức tạp, tìm kiếm có ràng buộc, và học tăng cường. Việc này giúp người dùng hiểu sâu sắc cơ chế hoạt động và sự khác biệt giữa các phương pháp trong cùng một bài toán cụ thể.
- Phân tích và so sánh hiệu suất: Đánh giá chi tiết các thuật toán dựa trên các tiêu chí quan trọng như thời gian chạy thực tế, bộ nhớ sử dụng, số bước giải thuật thực hiện, và tính tối ưu của lời giải (chuỗi di chuyển ít bước nhất hoặc chi phí thấp nhất). Kết quả so sánh sẽ làm rõ ưu, nhược điểm cũng như hiệu quả ứng dụng thực tiễn của từng thuật toán.
- Trực quan hóa kết quả: Xây dựng giao diện đồ họa (GUI) tương tác, thể hiện trực quan quá trình giải bài toán 8-puzzle qua từng bước, giúp người dùng dễ dàng quan sát, so sánh và đánh giá cách thức vận hành cũng như hiệu quả của từng thuật toán thông qua các hoạt ảnh (GIF), biểu đồ hiệu suất và các chỉ số đo lường.

---

# 2. NỘI DUNG THỰC HIỆN DỰ ÁN

Dự án bao gồm việc triển khai và trình bày toàn diện bài toán 8-Puzzle dưới dạng một hệ thống giải thuật AI đa thuật toán với các thành phần và đặc điểm chính như sau:

## 2.1. Bài toán 8-Puzzle

- **Mô tả**: Bàn cờ gồm 9 ô, trong đó có 8 ô chứa số từ 1 đến 8 và 1 ô trống. Mục tiêu là sắp xếp các ô số theo thứ tự từ 1 đến 8, sao cho ô trống nằm ở vị trí cuối cùng, tạo thành trạng thái mục tiêu.
- **Thành phần bài toán**:
    - **Trạng thái**: Cấu hình các ô số và ô trống trên bàn cờ tại thời điểm hiện tại.
    - **Hành động**: Di chuyển ô trống sang trái, phải, lên hoặc xuống, với điều kiện di chuyển hợp lệ (không vượt ra ngoài bàn cờ).
    - **Kiểm tra mục tiêu**: So sánh trạng thái hiện tại với trạng thái mục tiêu để xác định liệu bài toán đã được giải quyết hay chưa.
    - **Hàm heuristic**: Ước lượng khoảng cách từ trạng thái hiện tại đến trạng thái mục tiêu, thường dùng các hàm như số ô sai vị trí hoặc khoảng cách Manhattan.

## 2.2. Nhóm thuật toán triển khai

- **Tìm kiếm không thông tin (Uninformed Search)**: Bao gồm các thuật toán như BFS (Breadth-First Search), DFS (Depth-First Search), UCS (Uniform Cost Search), và IDS (Iterative Deepening Search). Đây là những thuật toán duyệt trạng thái mà không sử dụng thông tin heuristic, thích hợp cho bài toán có kích thước nhỏ, đảm bảo tính đầy đủ và đôi khi đảm bảo tối ưu.
- **Tìm kiếm có thông tin (Informed Search)**: Bao gồm Greedy Best-First Search, A* và IDA* — các thuật toán tận dụng hàm heuristic để tăng tốc độ tìm kiếm và nâng cao chất lượng lời giải, giảm đáng kể không gian trạng thái cần duyệt.
- **Tìm kiếm cục bộ (Local Search)**: Bao gồm các phương pháp Hill Climbing (đơn giản, steepest-ascent, stochastic), Simulated Annealing, Genetic Search và Beam Search. Các thuật toán này ưu tiên cải thiện trạng thái hiện tại theo hướng tối ưu cục bộ, phù hợp với không gian trạng thái lớn và yêu cầu lời giải gần đúng.
- **Tìm kiếm trong môi trường phức tạp (Complex Environment Search)**: Triển khai các thuật toán xử lý môi trường không chắc chắn như AND-OR Search, Partially Observable Search và No Observation Search, giúp giải quyết bài toán trong điều kiện quan sát hạn chế hoặc hành động không chắc chắn.
- **Tìm kiếm có điều kiện ràng buộc (Constraint Satisfaction Problem - CSP)**: Bao gồm Constraint Testing, Backtracking CSP và Backtracking AC-3. Các thuật toán này tập trung xử lý các ràng buộc phức tạp trong bài toán, giúp thu hẹp không gian tìm kiếm hiệu quả, đồng thời kiểm tra tính hợp lệ của các trạng thái.
- **Học tăng cường (Reinforcement Learning)**: Ứng dụng thuật toán Q-Learning để học từ kinh nghiệm và tối ưu chính sách giải quyết bài toán mà không cần mô hình môi trường chính xác.

## 2.3. Bảng tổng quan nhóm thuật toán

| Nhóm Thuật Toán | Ưu điểm chính | Nhược điểm chính | Ứng dụng trong 8-Puzzle |
|---|---|---|---|
| Tìm kiếm môi trường không chắc chắn | Xử lý không chắc chắn, quan sát không đầy đủ | Phức tạp, tốn tài nguyên | Ít dùng trong 8-Puzzle chuẩn |
| Tìm kiếm cục bộ | Tiết kiệm bộ nhớ, thực thi nhanh | Mắc kẹt cực trị cục bộ, không đảm bảo tối ưu | Tìm lời giải gần đúng khi không gian lớn |
| Học củng cố | Học được chính sách từ dữ liệu thực tế | Tốn thời gian học, khó mở rộng | Ứng dụng thử nghiệm, cần tối ưu thuật toán |
| Tìm kiếm không thông tin | Đơn giản, đảm bảo tìm lời giải nếu có | Tốn bộ nhớ, thời gian khi không gian lớn | Thích hợp với bài toán nhỏ |
| Tìm kiếm có thông tin | Hiệu quả cao, tối ưu với heuristic tốt | Phụ thuộc chất lượng heuristic | Thuật toán tiêu chuẩn của 8-Puzzle |
| Tìm kiếm có ràng buộc | Loại bỏ lựa chọn vô lý, cắt giảm không gian | Phức tạp mô hình hóa | Hỗ trợ kiểm tra, CSP ứng dụng hạn chế |

## 2.4. Tích hợp trực quan hóa (Visualization)

- **Giao diện GUI**:
    - Hiển thị rõ ràng trạng thái ban đầu của bàn cờ, quá trình chuyển đổi qua từng bước hành động, và trạng thái mục tiêu cuối cùng.
    - Đồ họa bàn cờ với các ô số và ô trống được thiết kế động, cập nhật theo từng bước di chuyển giúp người dùng dễ dàng quan sát.
    - Các chỉ số hiệu suất như thời gian chạy, số bước đã thực hiện, số trạng thái đã duyệt cũng được hiển thị trực quan kèm theo.
- **GIF minh họa**:
    - Tạo các hình ảnh động thể hiện quá trình giải bài toán 8-Puzzle theo từng thuật toán khác nhau.
    - Giúp người dùng dễ dàng so sánh và đánh giá trực quan cách thức vận hành cũng như hiệu quả của từng thuật toán.

---

# 3. CÁC THUẬT TOÁN SỬ DỤNG

## 3.1. Tìm kiếm không thông tin (Uninformed Search)

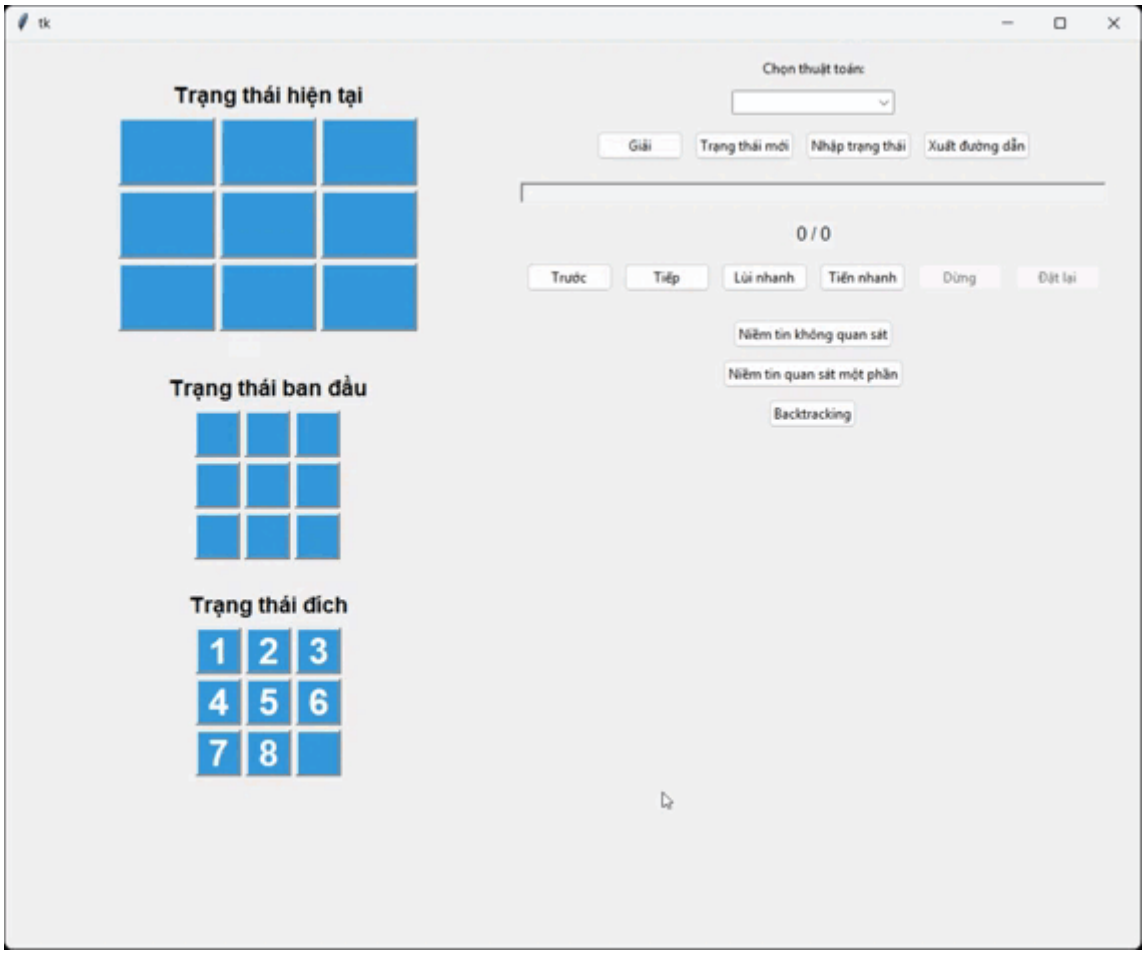
**Định nghĩa lý thuyết:** Tìm kiếm không thông tin (Uninformed Search) là các chiến lược tìm kiếm mà không có kiến thức bổ sung nào về không gian trạng thái ngoài thông tin được cung cấp trong định nghĩa bài toán. Các thuật toán này không sử dụng bất kỳ hàm heuristic nào để định hướng quá trình tìm kiếm mà chỉ dựa vào cấu trúc cây trạng thái và các quy tắc sinh trạng thái kế tiếp.

**Áp dụng vào bài 8-Puzzle:** Trong bài toán 8-Puzzle, trạng thái là cách sắp xếp của các ô số, hành động là việc di chuyển ô trống lên/xuống/trái/phải. Các thuật toán tìm kiếm không thông tin như BFS, DFS, UCS, IDS có thể được sử dụng để tìm lời giải mà không cần biết trước trạng thái nào gần đích hơn, chỉ dựa vào việc duyệt toàn bộ không gian trạng thái một cách có hệ thống.

### 3.1.1. Breadth-First Search (BFS)

- **Lý thuyết:** BFS là chiến lược tìm kiếm theo chiều rộng, mở rộng tất cả các nút ở độ sâu hiện tại trước khi đi sâu hơn.
- **Áp dụng vào 8-Puzzle:** BFS bảo đảm tìm ra lời giải tối ưu nếu chi phí các bước di chuyển là như nhau. Tuy nhiên, dễ gặp tình trạng sử dụng nhiều bộ nhớ.
- **Các bước thực hiện:**
    1. Đưa trạng thái ban đầu vào hàng đợi.
    2. Lặp lại: lấy trạng thái đầu ra khỏi hàng đợi.
    3. Kiểm tra nếu là trạng thái đích thì trả về lời giải.
    4. Tạo các trạng thái con và đưa vào hàng đợi nếu chưa được duyệt.
- **Độ phức tạp:** O(b^d) với b là nhánh trung bình, d là độ sâu.
- **Ưu điểm:** Tìm được lời giải ngắn nhất.
- **Nhược điểm:** Tốn bộ nhớ, chậm với không gian trạng thái lớn.

- **Link tham khảo:** BFS - GeeksforGeeks
- **Nhận xét:** Hữu ích cho bài toán có chi phí đều, nhưng không hiệu quả với độ sâu lớn.
- **Hình ảnh minh họa:**



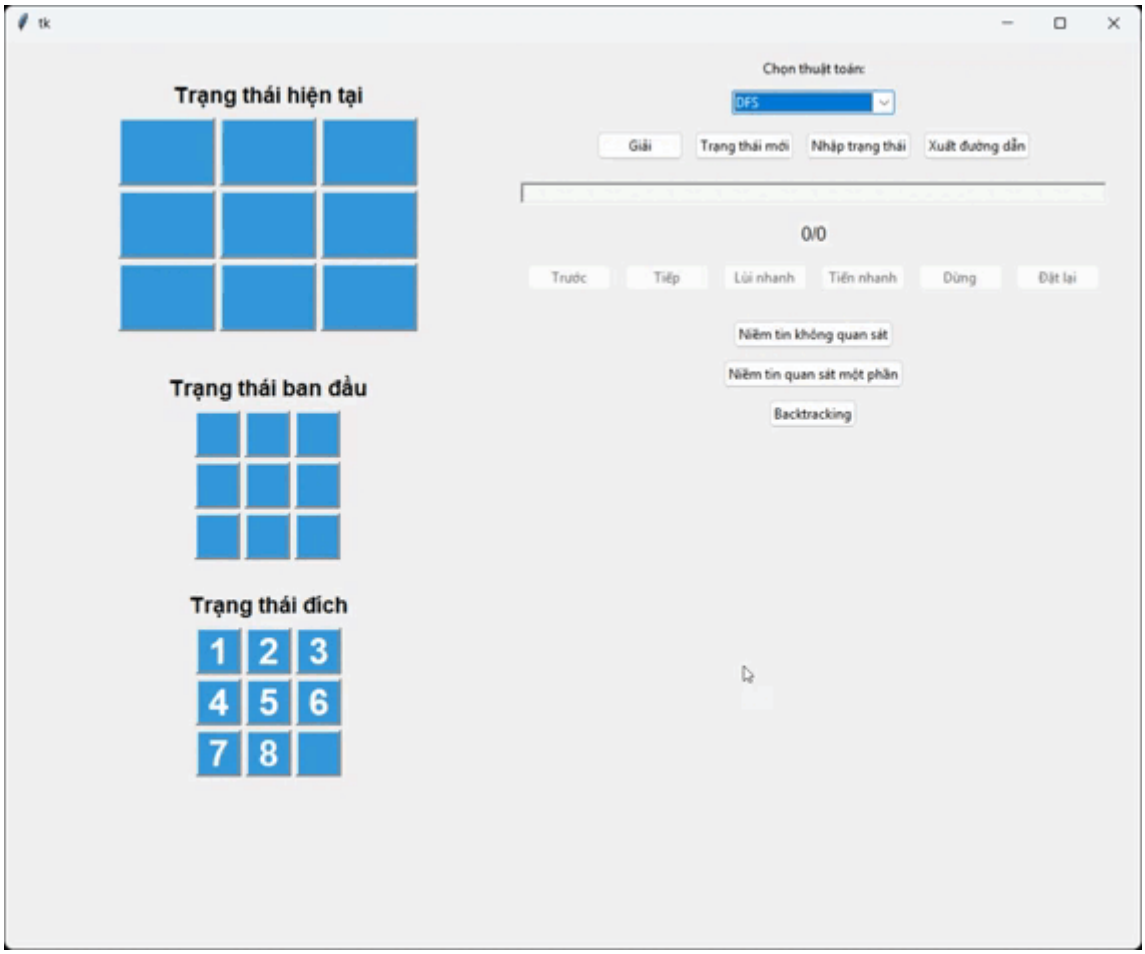

### 3.1.2. Depth-First Search (DFS)

- **Lý thuyết:** DFS là chiến lược mở rộng sâu nhất có thể trước khi quay lại.
- **Áp dụng:** DFS ít tốn bộ nhớ hơn BFS nhưng không đảm bảo lời giải ngắn nhất.
- **Các bước:**

    1. Đưa trạng thái đầu vào stack.
    2. Lặp lại: lấy trạng thái ở đỉnh stack ra.
    3. Nếu là trạng thái đích, trả về lời giải.
    4. Đưa các trạng thái con vào stack.

- **Độ phức tạp:** O(b^m) với m là độ sâu tối đa.
- **Ưu điểm:** Ít tốn bộ nhớ.
- **Nhược điểm:** Dễ đi vào vòng lặp, không đảm bảo tối ưu.
- **Link:** DFS - GeeksforGeeks
- **Nhận xét:** Không phù hợp với bài toán có lời giải sâu hoặc vòng lặp nhiều.

- **Hình ảnh minh họa:**

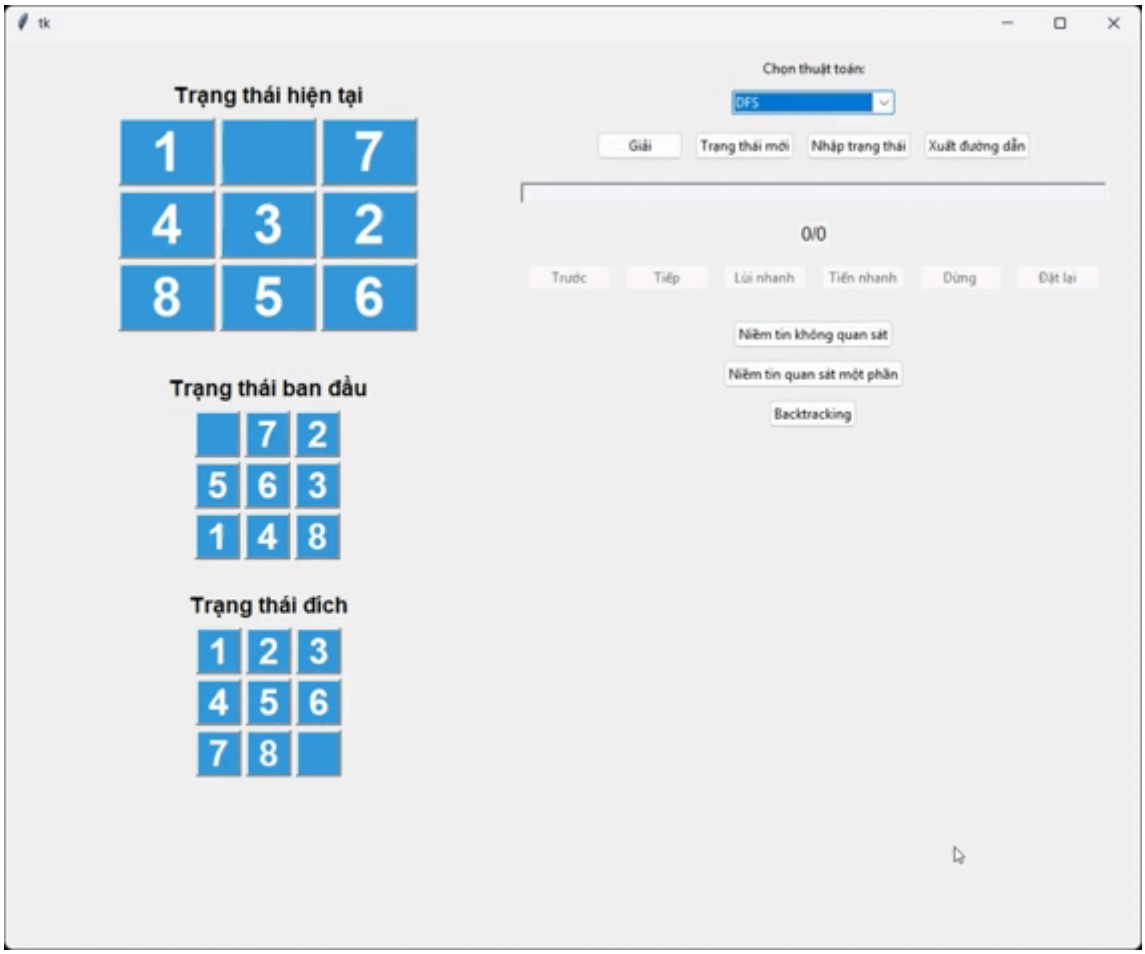



### 3.1.3. Uniform Cost Search (UCS)

- **Lý thuyết:** Mở rộng nút theo chi phí nhỏ nhất từ gốc đến hiện tại.
- **Áp dụng:** Tìm lời giải tối ưu, phù hợp với chi phí di chuyển khác nhau.
- **Các bước:**

    1. Đưa trạng thái đầu vào hàng đợi ưu tiên.
    2. Mỗi bước, lấy trạng thái có chi phí nhỏ nhất.
    3. Kiểm tra nếu là đích, trả lời giải.
    4. Thêm các trạng thái con vào hàng đợi với chi phí tương ứng.

- **Độ phức tạp:** O(b^d)
- **Ưu điểm:** Tối ưu.
- **Nhược điểm:** Tốn bộ nhớ.
- **Link:** UCS - Educative
- **Nhận xét:** Lựa chọn tốt nếu chi phí không đều.
- **Hình ảnh minh họa:**

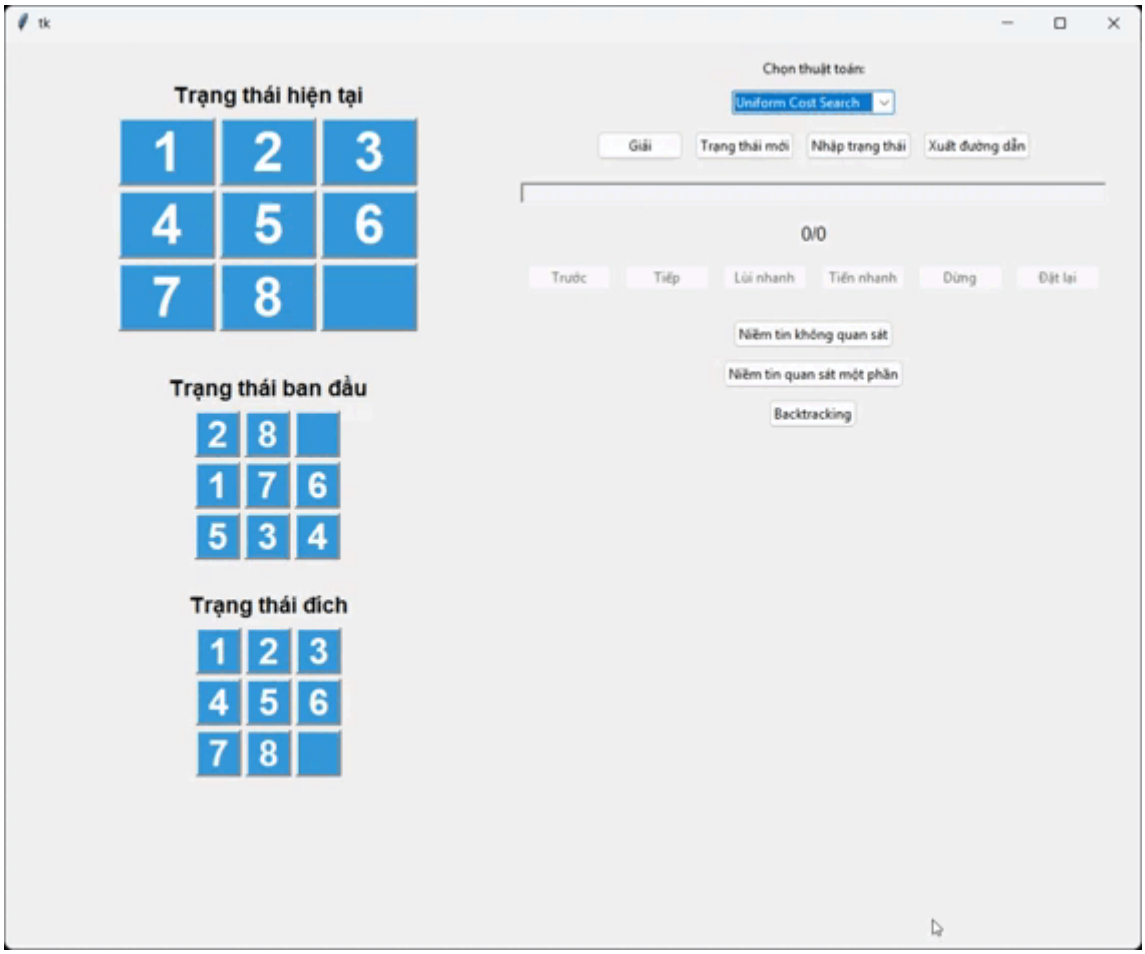



### 3.1.4. Iterative Deepening Search (IDS)

- **Lý thuyết:** Kết hợp BFS và DFS bằng cách tìm kiếm sâu dần từng mức.
- **Áp dụng:** Tối ưu bộ nhớ và đảm bảo tìm lời giải tối ưu.
- **Các bước:**

    1. Chọn giới hạn độ sâu d.
    2. Thực hiện DFS đến độ sâu d.
    3. Tăng d và lặp lại nếu chưa tìm thấy lời giải.

- **Độ phức tạp:** O(b^d), bộ nhớ O(bd).
- **Ưu điểm:** Giải quyết nhược điểm bộ nhớ của BFS.
- **Nhược điểm:** Tốn thời gian do lặp lại nhiều DFS.
- **Link:** IDS - GeeksforGeeks
- **Nhận xét:** Phù hợp cho bài toán có không gian lớn và lời giải tối ưu.
- **Hình ảnh minh họa:**




### 3.1.5. Bảng So sánh các thuật toán Uninformed Search

| Thuật toán | Tối ưu | Bộ nhớ | Độ phức tạp | Ghi chú |
|---|---|---|---|---|
| BFS | Có | Cao | O(b^d) | Tốt khi chi phí đều |
| DFS | Không | Thấp | O(b^m) | Dễ vào vòng lặp |
| UCS | Có | Cao | O(b^d) | Dùng khi chi phí khác nhau |
| IDS | Có | Trung bình | O(b^d) | Cân bằng giữa DFS và BFS |

**Kết luận:** IDS là lựa chọn cân bằng giữa tối ưu và hiệu quả bộ nhớ cho bài toán 8-Puzzle nếu chi phí đều. Nếu chi phí khác nhau, UCS là thuật toán phù hợp nhất.

## 3.2. Tìm kiếm có thông tin (Informed Search)

**Định nghĩa lý thuyết:** Tìm kiếm có thông tin (Informed Search) là các chiến lược sử dụng kiến thức bổ sung (thường là hàm heuristic) để đánh giá mức độ tiệm cận của trạng thái hiện tại đến trạng thái mục tiêu. Các thuật toán này có khả năng thu hẹp không gian tìm kiếm và tăng tốc độ tìm ra lời giải.

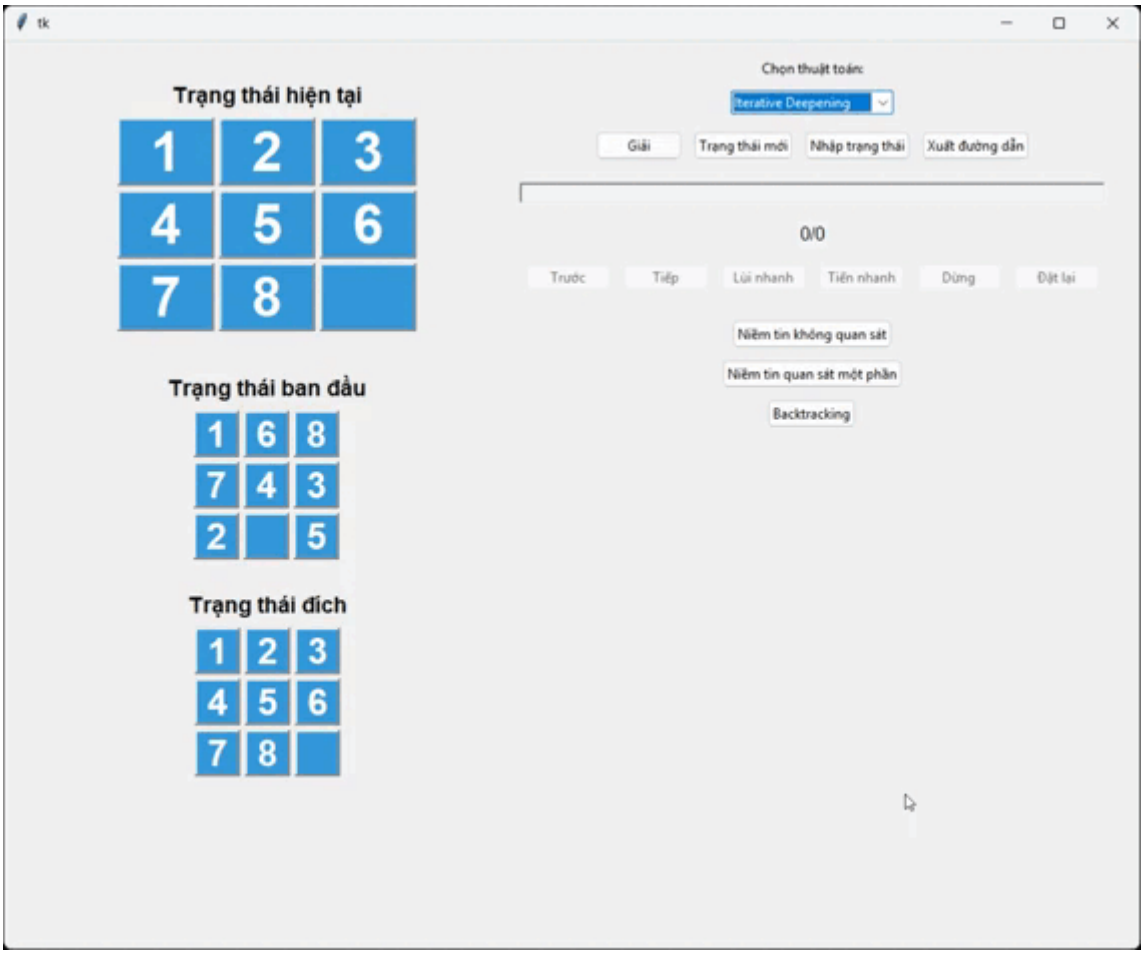
**Áp dụng vào bài 8-Puzzle:**
Trong bài toán 8-Puzzle, các heuristic thường được sử dụng bao gồm:

- Số lượng ô sai vị trí (Misplaced Tiles)
- Tổng khoảng cách Manhattan giữa mỗi ô và vị trí mục tiêu của nó

Các thuật toán như Greedy Best-First Search, A*, và IDA* tận dụng các hàm heuristic này để dẫn dắt việc lựa chọn trạng thái tiếp theo cần mở rộng.

#### 3.2.1. Greedy Best-First Search

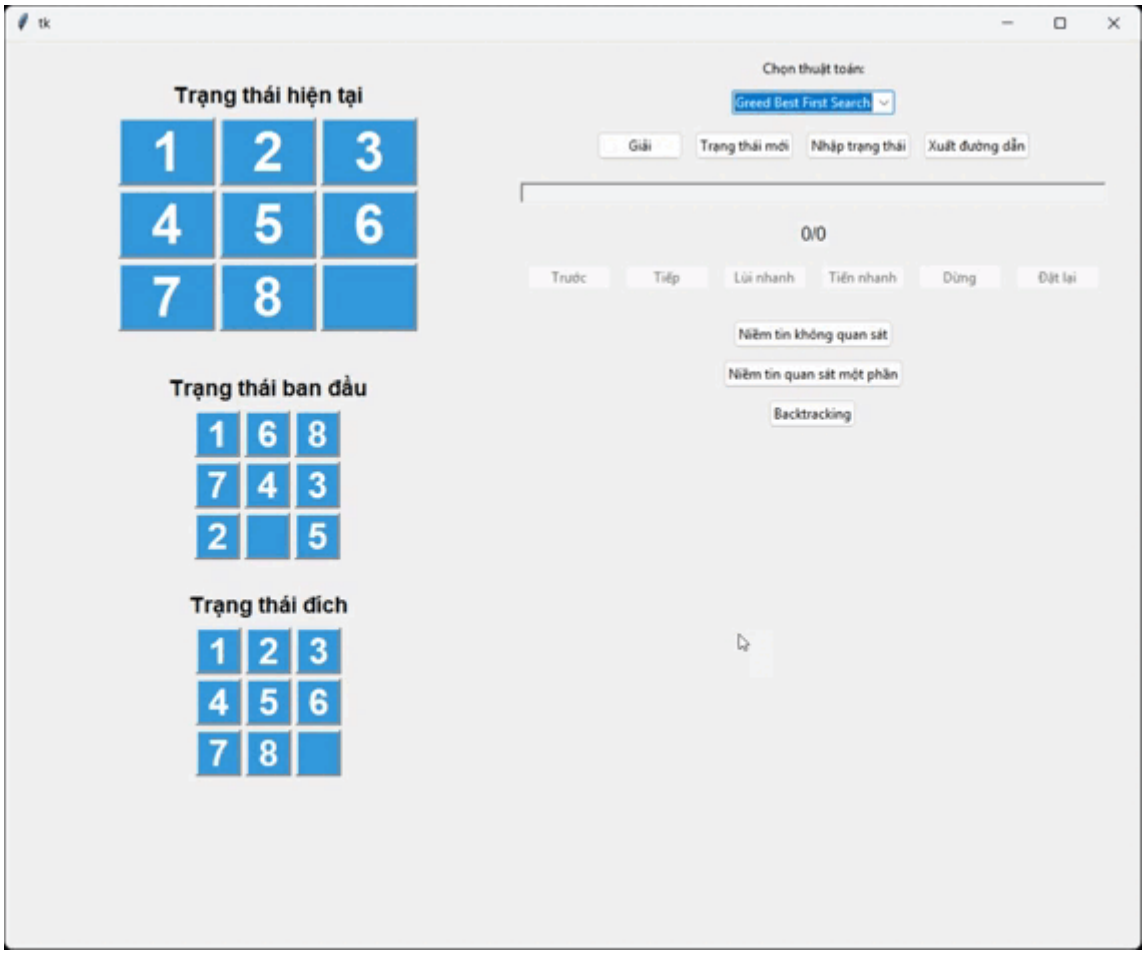
- **Lý thuyết:** Luôn chọn mở rộng trạng thái có giá trị heuristic h(n) nhỏ nhất, tức là trạng thái được cho là gần mục tiêu nhất.
- **Áp dụng:** Tập trung mạnh vào mục tiêu nhưng không đảm bảo lời giải tối ưu.
- **Các bước:**
    1. Đưa trạng thái đầu vào hàng đợi ưu tiên theo h(n).
    2. Lặp lại: chọn trạng thái có h(n) thấp nhất.
    3. Kiểm tra trạng thái đích.
    4. Sinh các trạng thái con và tính h(n), đưa vào hàng đợi.
- **Độ phức tạp:** O(b^m)
- **Ưu điểm:** Nhanh, đơn giản.
- **Nhược điểm:** Dễ rơi vào ngõ cụt, không đảm bảo tối ưu.
- **Link tham khảo:** Greedy Best-First Search
- **Nhận xét:** Phù hợp khi thời gian quan trọng hơn độ chính xác lời giải.
- **Hình ảnh minh họa:**




### 3.2.2. A Search*

- **Lý thuyết:**
  Dựa vào hàm f(n) = g(n) + h(n), trong đó:
  - g(n): chi phí từ trạng thái đầu đến n
  - h(n): chi phí ước lượng từ n đến mục tiêu
- **Áp dụng:** Tối ưu và hoàn chỉnh nếu h(n) không đánh giá quá cao (admissible).
- **Các bước:**
  1. Đưa trạng thái đầu vào hàng đợi ưu tiên theo f(n).
  2. Mỗi bước chọn trạng thái có f(n) nhỏ nhất.
  3. Kiểm tra trạng thái đích.
  4. Sinh trạng thái con, tính f(n) và đưa vào hàng đợi.
- **Độ phức tạp:** O(b^d)
- **Ưu điểm:** Tìm được lời giải ngắn nhất.
- **Nhược điểm:** Có thể tốn nhiều bộ nhớ.
- **Link tham khảo:** A* Search Algorithm
- **Nhận xét:** Là một trong những thuật toán hiệu quả nhất cho 8-Puzzle.

- **Hình ảnh minh họa:**

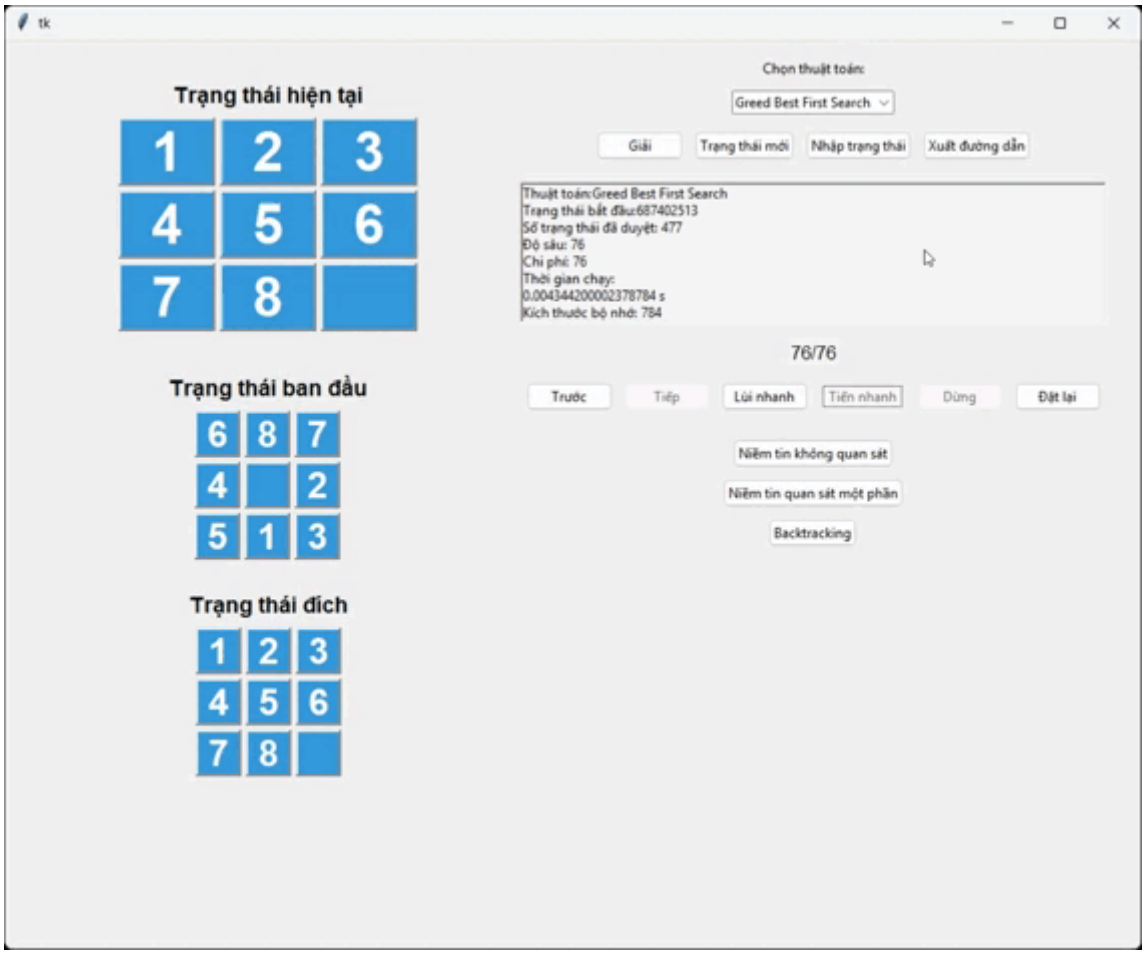



### 3.2.3. Iterative Deepening A (IDA)*

- **Lý thuyết:** Là phiên bản của A* dùng tìm kiếm sâu dần theo giới hạn f(n).
- **Áp dụng:** Tiết kiệm bộ nhớ so với A*, nhưng vẫn tối ưu.
- **Các bước:**

    1. Thiết lập ngưỡng f(n) ban đầu.
    2. Thực hiện tìm kiếm theo DFS có ràng buộc f(n).
    3. Nếu không tìm thấy, tăng ngưỡng f(n) và lặp lại.

- **Độ phức tạp:** O(b^d), bộ nhớ O(d)
- **Ưu điểm:** Tối ưu và tiết kiệm bộ nhớ.
- **Nhược điểm:** Chạy lại nhiều lần.
- **Link tham khảo:** Iterative Deepening A* - Wikipedia
- **Nhận xét:** Tốt cho bài toán lớn, khi không đủ bộ nhớ cho A*.
- **Hình ảnh minh họa:**




#### 3.2.4. Bảng so sánh các thuật toán Informed Search

| Thuật toán | Tối ưu | Bộ nhớ | Độ phức tạp | Ghi chú |
|---|---|---|---|---|
| Greedy Best-First | Không | Trung bình | O(b^m) | Tập trung mạnh vào mục tiêu |
| A* Search | Có | Cao | O(b^d) | Phổ biến và mạnh mẽ nhất |
| IDA* | Có | Thấp | O(b^d) | Hiệu quả với bài toán lớn |

**Kết luận:** A* là thuật toán tốt nhất về độ chính xác và hiệu quả tổng thể trong bài toán 8-Puzzle. Nếu cần tiết kiệm bộ nhớ, nên dùng IDA*.

### 3.3. Tìm kiếm cục bộ (Local Search)

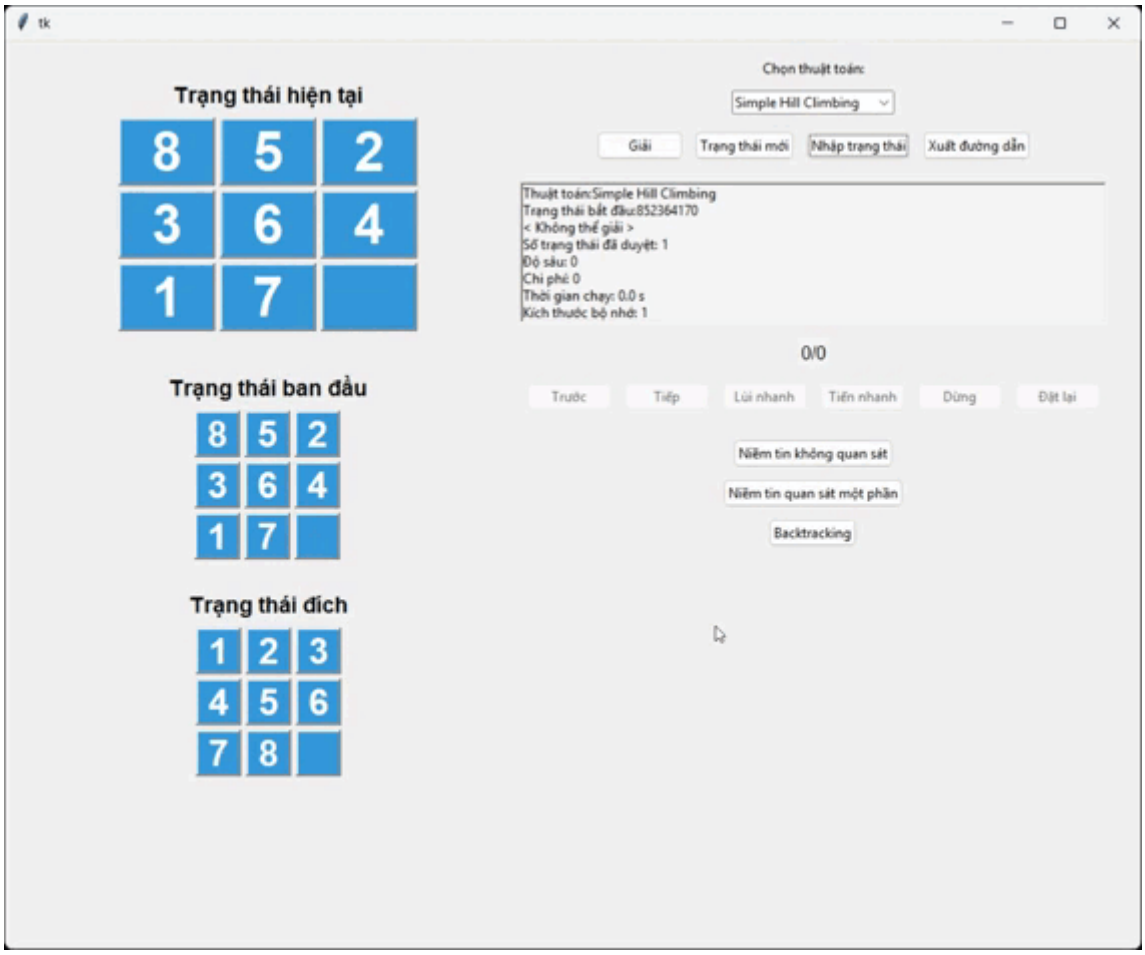
**Định nghĩa lý thuyết:** Tìm kiếm cục bộ là chiến lược tìm kiếm cải tiến liên tục dựa trên trạng thái hiện tại, không mở rộng theo cây mà chỉ tập trung vào trạng thái lân cận. Nó đặc biệt phù hợp với các bài toán không cần lời giải đầy đủ mà chỉ cần tối ưu một trạng thái (như 8-Puzzle).

**Áp dụng vào bài 8-Puzzle:** Trong 8-Puzzle, mỗi bước ta xét các trạng thái lân cận bằng cách di chuyển ô trống, sau đó chọn trạng thái tốt nhất theo một tiêu chí đánh giá như heuristic. Các thuật toán như Hill Climbing, Simulated Annealing, Genetic Algorithm,... là đại diện cho hướng tiếp cận này.

#### 3.3.1. Simple Hill Climbing

- **Lý thuyết:** Luôn chọn trạng thái kế tiếp có giá trị đánh giá tốt hơn hiện tại.

- **Áp dụng:** Mỗi bước chọn trạng thái con có h(n) nhỏ hơn.
- **Các bước:**
    1. Khởi tạo trạng thái ban đầu.
    2. Sinh các trạng thái kế cận.
    3. Chọn trạng thái tốt hơn hiện tại (nếu có).
    4. Nếu không có cải thiện, dừng lại.
- **Độ phức tạp:** O(bm)
- **Ưu điểm:** Đơn giản, nhanh.
- **Nhược điểm:** Mắc kẹt tại cực trị địa phương.
- **Link tham khảo:** Simple Hill Climbing
- **Nhận xét:** Không thích hợp cho không gian tìm kiếm có nhiều cực trị.
- **Hình minh họa:**



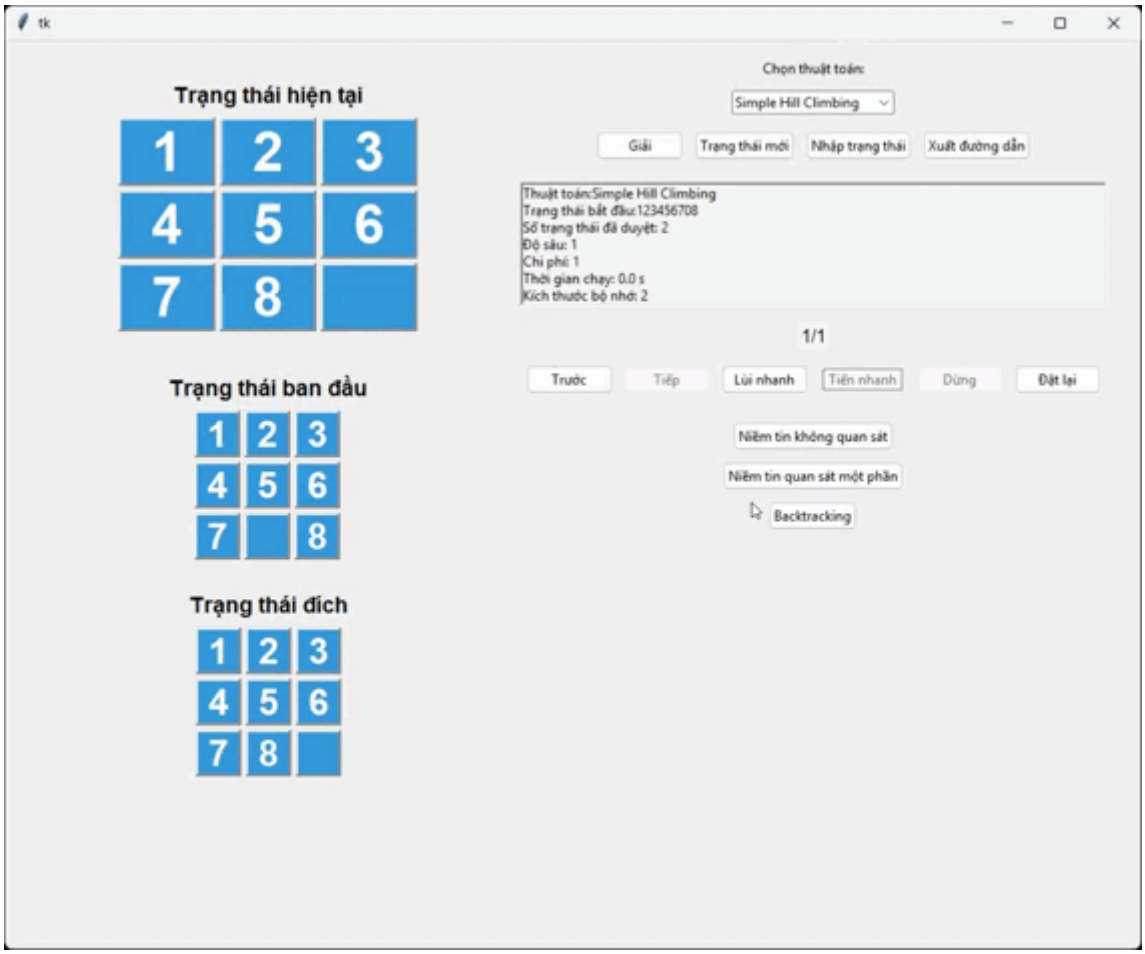

### 3.3.2. Steepest-Ascent Hill Climbing

- **Lý thuyết:** Là biến thể cải tiến của Simple Hill Climbing, chọn trạng thái tốt nhất trong tất cả trạng thái con.
- **Áp dụng:** Đánh giá toàn bộ trạng thái lân cận và chọn tốt nhất.

- **Các bước:** Tương tự Simple Hill Climbing nhưng xét toàn bộ lân cận.
- **Độ phức tạp:** Cao hơn Simple Hill Climbing.
- **Ưu điểm:** Ít bị kẹt hơn Simple.
- **Nhược điểm:** Vẫn có thể kẹt tại cực trị địa phương.
- **Link tham khảo:** Steepest-Ascent Hill Climbing
- **Nhận xét:** Hiệu quả hơn Simple nhưng chưa khắc phục được nhược điểm lớn.

#### 3.3.3. Stochastic Hill Climbing

- **Lý thuyết:** Thay vì chọn trạng thái tốt nhất, chọn ngẫu nhiên một trạng thái tốt hơn.
- **Áp dụng:** Giảm khả năng kẹt cực trị địa phương.
- **Các bước:**
    1. Sinh trạng thái con.
    2. Chọn ngẫu nhiên một trạng thái con tốt hơn.
    3. Tiếp tục đến khi không còn cải thiện.
- **Độ phức tạp:** O(bm)
- **Ưu điểm:** Tăng khả năng khám phá.
- **Nhược điểm:** Không đảm bảo tối ưu.
- **Link tham khảo:** Stochastic Hill Climbing
- **Nhận xét:** Hiệu quả trong không gian tìm kiếm phức tạp.
- **Hình ảnh minh họa:**

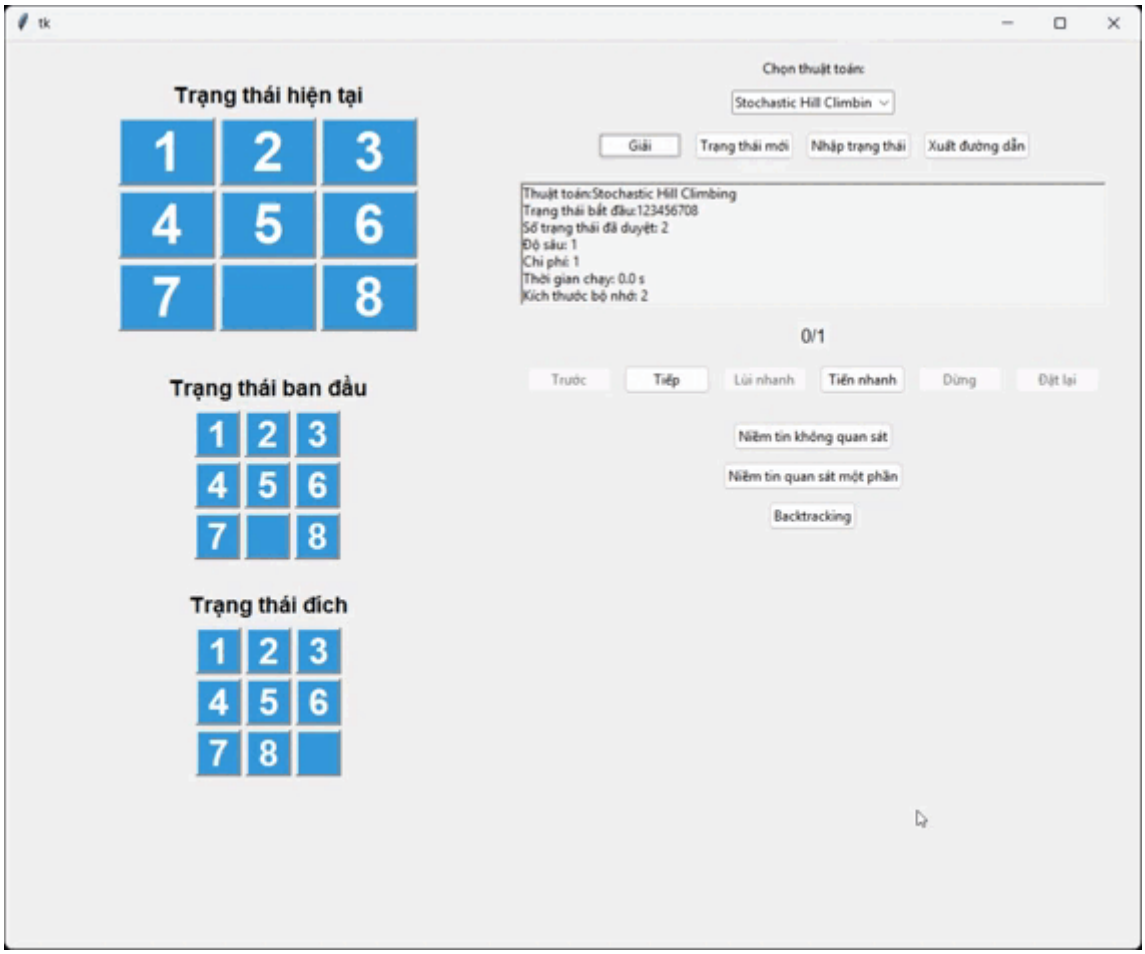



### 3.3.4. Simulated Annealing

- **Lý thuyết:** Cho phép chọn trạng thái xấu hơn với xác suất giảm dần theo thời gian (temperature).
- **Áp dụng:** Khắc phục tình trạng kẹt cứng tại cực trị.
- **Các bước:**

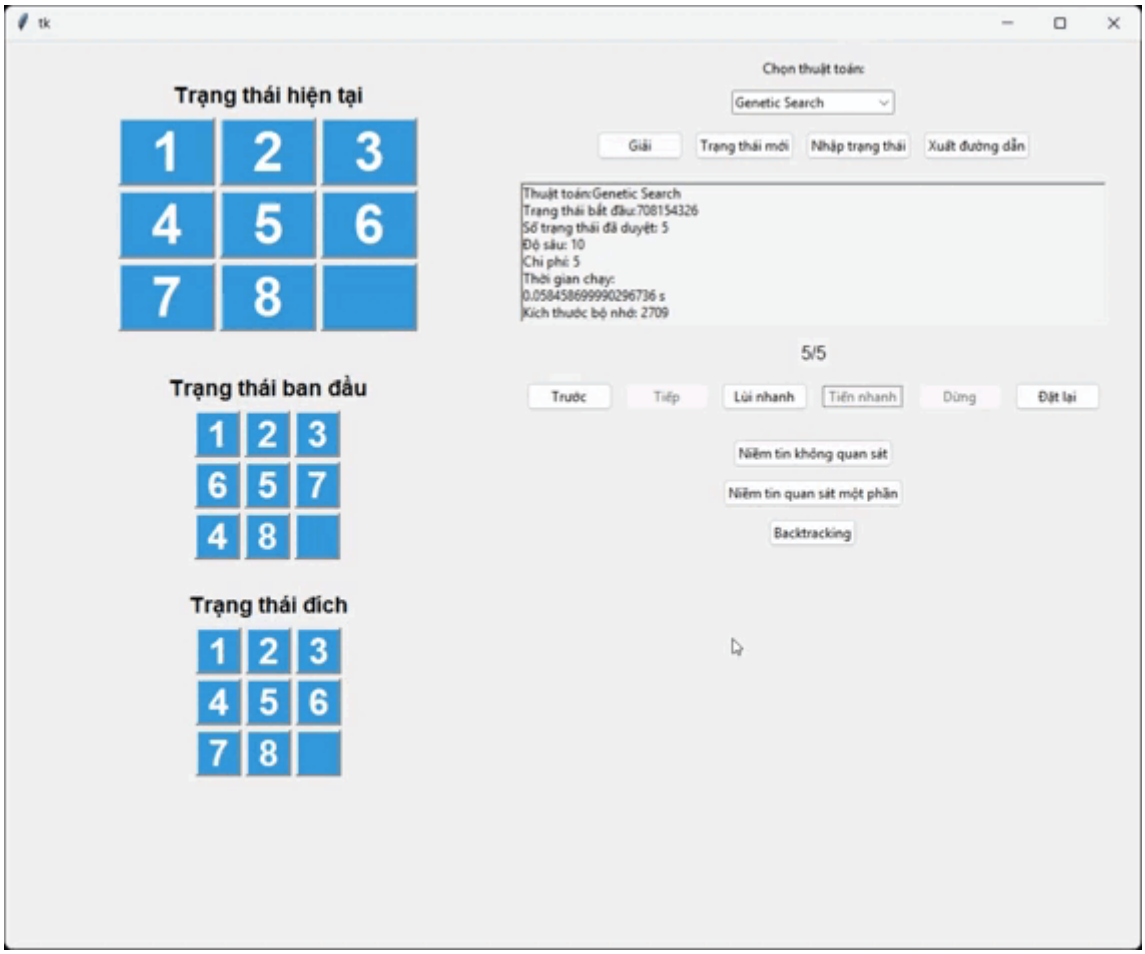
    1. Khởi tạo trạng thái và nhiệt độ.
    2. Lặp: chọn ngẫu nhiên trạng thái lân cận.
    3. Nếu tốt hơn, chấp nhận; nếu xấu hơn, chấp nhận theo xác suất.
    4. Giảm nhiệt độ dần.

- **Độ phức tạp:** O(bm)
- **Ưu điểm:** Khám phá rộng hơn.
- **Nhược điểm:** Thiết lập nhiệt độ không phù hợp dễ gây sai lệch.
- **Link tham khảo:** Simulated Annealing
- **Nhận xét:** Hiệu quả cho bài toán nhiều cực trị.
- **Hình ảnh minh họa:**



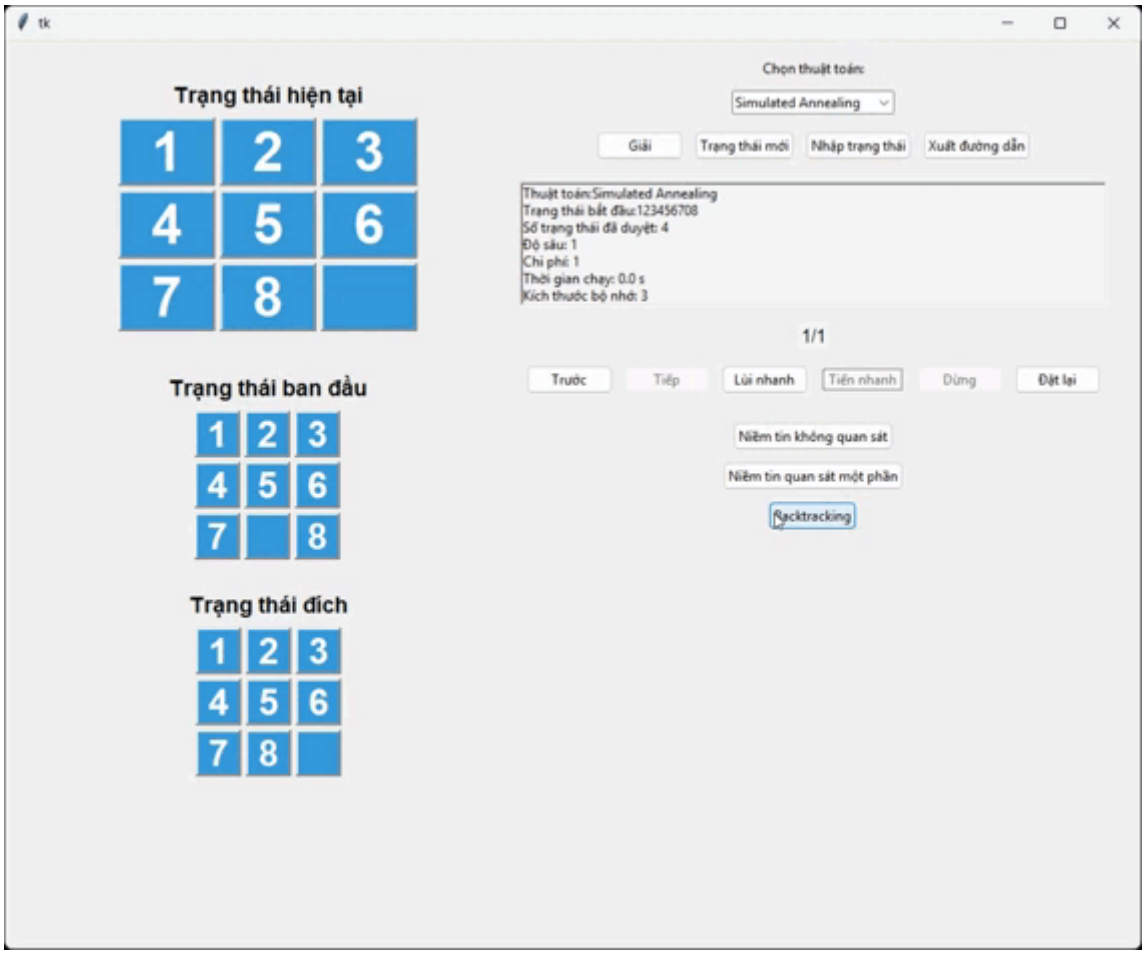

### 3.3.5. Genetic Algorithm

- **Lý thuyết:** Mô phỏng tiến hóa tự nhiên với các bước chọn lọc, lai ghép, đột biến.
- **Áp dụng:** Mỗi cá thể là một trạng thái 8-Puzzle.
- **Các bước:**
    1. Khởi tạo quần thể ngẫu nhiên.
    2. Đánh giá mức độ phù hợp (fitness).
    3. Chọn cá thể tốt.
    4. Lai ghép và đột biến.
    5. Lặp lại cho đến khi đạt tiêu chí.
- **Độ phức tạp:** Phụ thuộc số cá thể và số thế hệ.
- **Ưu điểm:** Khả năng khám phá toàn diện.
- **Nhược điểm:** Không đảm bảo hội tụ, cần hiệu chỉnh tham số.
- **Link tham khảo:** Genetic Algorithm
- **Nhận xét:** Thích hợp cho tối ưu hóa trạng thái 8-Puzzle phức tạp.
- **Hình ảnh minh họa:**




### 3.3.6. Beam Search

- **Lý thuyết:** Giới hạn số lượng trạng thái được mở rộng tại mỗi bước.
- **Áp dụng:** Duy trì k trạng thái tốt nhất ở mỗi mức độ sâu.
- **Các bước:**
    1. Khởi tạo k trạng thái.
    2. Mỗi bước sinh trạng thái con từ k trạng thái.
    3. Giữ lại k trạng thái con tốt nhất.
    4. Lặp lại đến khi đạt mục tiêu.
- **Độ phức tạp:** Giảm so với tìm kiếm toàn bộ.
- **Ưu điểm:** Tiết kiệm bộ nhớ.
- **Nhược điểm:** Dễ bỏ lỡ lời giải tối ưu.
- **Link tham khảo:** Beam Search
- **Nhận xét:** Cân bằng giữa hiệu suất và độ chính xác.
- **Hình ảnh minh họa:**




### 3.3.7. Bảng so sánh các thuật toán Local Search

| Thuật toán | Tối ưu | Khả năng tránh cực trị | Đặc điểm |
|---|---|---|---|
| Simple Hill Climbing | Không | Thấp | Nhanh, đơn giản |
| Steepest-Ascent | Không | Trung bình | Tốt hơn Simple |
| Stochastic Hill Climbing | Không | Trung bình | Ngẫu nhiên lựa chọn |
| Simulated Annealing | Có | Cao | Cân bằng khai phá |
| Genetic Algorithm | Có | Cao | Dựa trên tiến hóa |
| Beam Search | Có | Trung bình | Giới hạn trạng thái |

**Kết luận:** Simulated Annealing và Genetic Algorithm là hai lựa chọn nổi bật cho bài toán 8-Puzzle khi cần tối ưu hóa và tránh kẹt cực trị địa phương.

## 3.4. Tìm kiếm trong môi trường phức tạp (Complex environment search)

**Định nghĩa lý thuyết:** Tìm kiếm trong môi trường phức tạp đề cập đến các phương pháp tìm kiếm áp dụng cho bài toán có tính không chắc chắn, quan sát không đầy đủ hoặc tồn tại nhiều khả năng dẫn đến các nhánh hành động khác nhau (ví dụ: AND-OR search).

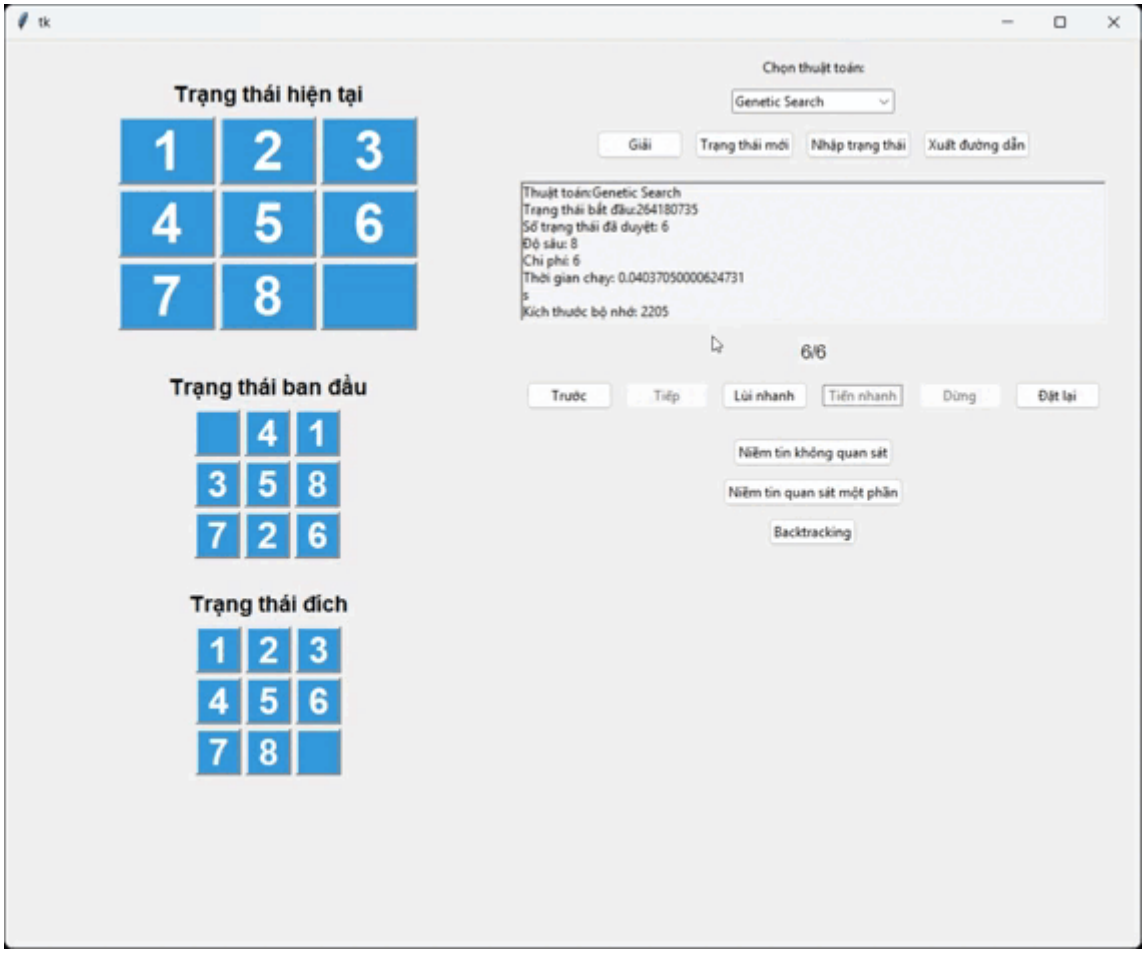
**Áp dụng vào bài 8-Puzzle:** Dù 8-Puzzle là một môi trường xác định và đầy đủ thông tin, ta vẫn có thể mô phỏng các điều kiện không chắc chắn (ví dụ: bàn cờ không rõ vị trí ban đầu, hành động bị giới hạn) để đánh giá các thuật toán này.

### 3.4.1. AND-OR Search Algorithm

- **Lý thuyết:** Dành cho môi trường không xác định, xây dựng cây AND-OR thể hiện các lựa chọn và hậu quả.
- **Áp dụng:** Mô phỏng môi trường không chắc chắn như di chuyển có thể thất bại.
- **Các bước:**

  1. Xây cây trạng thái AND-OR.
  2. Duyệt theo các nhánh có thể xảy ra (OR) và các nhánh tất yếu (AND).
  3. Lặp lại cho đến khi toàn bộ cây đạt mục tiêu hoặc không thể giải được.

- **Độ phức tạp:** Rất cao do phân nhánh mạnh.
- **Ưu điểm:** Mô hình hóa tốt tình huống không chắc chắn.
- **Nhược điểm:** Không phù hợp với 8-Puzzle chuẩn.
- **Link tham khảo:** AND-OR Graph in AI
- **Nhận xét:** Chủ yếu mang tính lý thuyết trong ngữ cảnh 8-Puzzle.
- **Hình ảnh minh họa:**



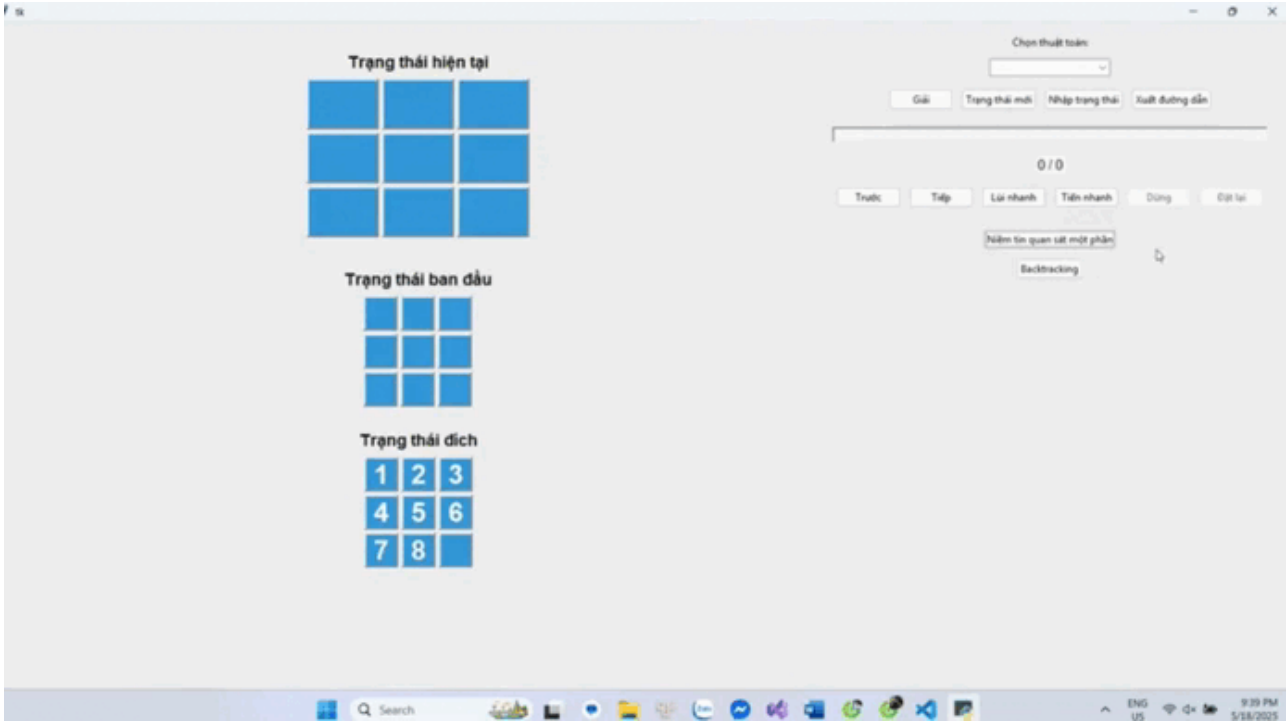

### 3.4.2. Belief State Search

- **Lý thuyết:** Tìm kiếm với trạng thái niềm tin (tập hợp các trạng thái khả dĩ).

- **Áp dụng:** Khi không biết rõ trạng thái ban đầu hoặc bị giới hạn trong quan sát.
- **Các bước:**
    1. Biểu diễn trạng thái bằng tập hợp trạng thái khả dĩ.
    2. Cập nhật niềm tin sau mỗi hành động hoặc quan sát.
    3. Tìm đường đi đến mục tiêu trong không gian trạng thái niềm tin.
- **Độ phức tạp:** Cao do kích thước trạng thái niềm tin.
- **Ưu điểm:** Giải quyết tốt bài toán thiếu thông tin.
- **Nhược điểm:** Khó áp dụng trực tiếp với 8-Puzzle chuẩn.
- **Link tham khảo:** Belief State Search Notes
- **Nhận xét:** Hữu ích trong mô phỏng bài toán ẩn trạng thái.

#### 3.4.3. Partially Observable Search

- **Lý thuyết:** Tìm kiếm trong không gian với thông tin quan sát bị hạn chế.
- **Áp dụng:** Khi chỉ biết một phần trạng thái 8-Puzzle (ví dụ: một phần lưới bị che).
- **Các bước:**
    1. Xây dựng mô hình xác suất của trạng thái thật.
    2. Cập nhật trạng thái dự đoán từ dữ liệu quan sát.
    3. Quyết định hành động dựa trên trạng thái dự đoán.
- **Độ phức tạp:** Cao, yêu cầu mô hình hóa xác suất.
- **Ưu điểm:** Giải quyết bài toán thực tế với thông tin không đầy đủ.
- **Nhược điểm:** Phức tạp về mặt triển khai và tính toán.
- **Link tham khảo:** Partially Observable Search
- **Nhận xét:** Mô phỏng phù hợp cho các biến thể nâng cao của 8-Puzzle.
- **Hình minh họa:**




### 3.4.4. No Observation Search

- **Lý thuyết:** Tìm kiếm mà không có bất kỳ thông tin nào về trạng thái hiện tại sau mỗi hành động.
- **Áp dụng:** Mô phỏng trường hợp robot mù không biết vị trí sau mỗi bước di chuyển.
- **Các bước:**
    1. Ghi nhớ toàn bộ chuỗi hành động từ đầu.
    2. Lập kế hoạch hành động không phụ thuộc vào quan sát.
    3. Sử dụng xác suất để ước lượng trạng thái.
- **Độ phức tạp:** Cực cao.
- **Ưu điểm:** Áp dụng cho hệ thống không có cảm biến.
- **Nhược điểm:** Khó triển khai và không thực tế cho 8-Puzzle chuẩn.
- **Link tham khảo:** General Search Algorithms
- **Nhận xét:** Mang tính học thuật nhiều hơn là ứng dụng thực tế.

### 3.4.5. Bảng So sánh các thuật toán Complex Environment

| Thuật toán | Thông tin quan sát | Mô phỏng 8-Puzzle chuẩn | Ưu điểm chính | Nhược điểm lớn |
|---|---|---|---|---|
| AND-OR Search | Không xác định | Không phù hợp | Mô hình hóa quyết định phức tạp | Phân nhánh lớn, khó mở rộng |
| Belief State Search | Một phần | Hạn chế | Mô phỏng thiếu thông tin | Trạng thái niềm tin lớn |

| Thuật toán | Thông tin quan sát | Mô phỏng 8-Puzzle chuẩn | Ưu điểm chính | Nhược điểm lớn |
|---|---|---|---|---|
| Partially Observable | Một phần | Có thể mô phỏng | Thực tế, sát các bài toán robot | Tốn tài nguyên tính toán |
| No Observation | Không có | Không phù hợp | Phù hợp hệ thống mù cảm biến | Gần như không áp dụng được |

**Kết luận:**
Các thuật toán này không phù hợp trực tiếp với 8-Puzzle cổ điển nhưng có thể dùng để mô phỏng bài toán ẩn thông tin hoặc kiểm thử trong môi trường nâng cao.

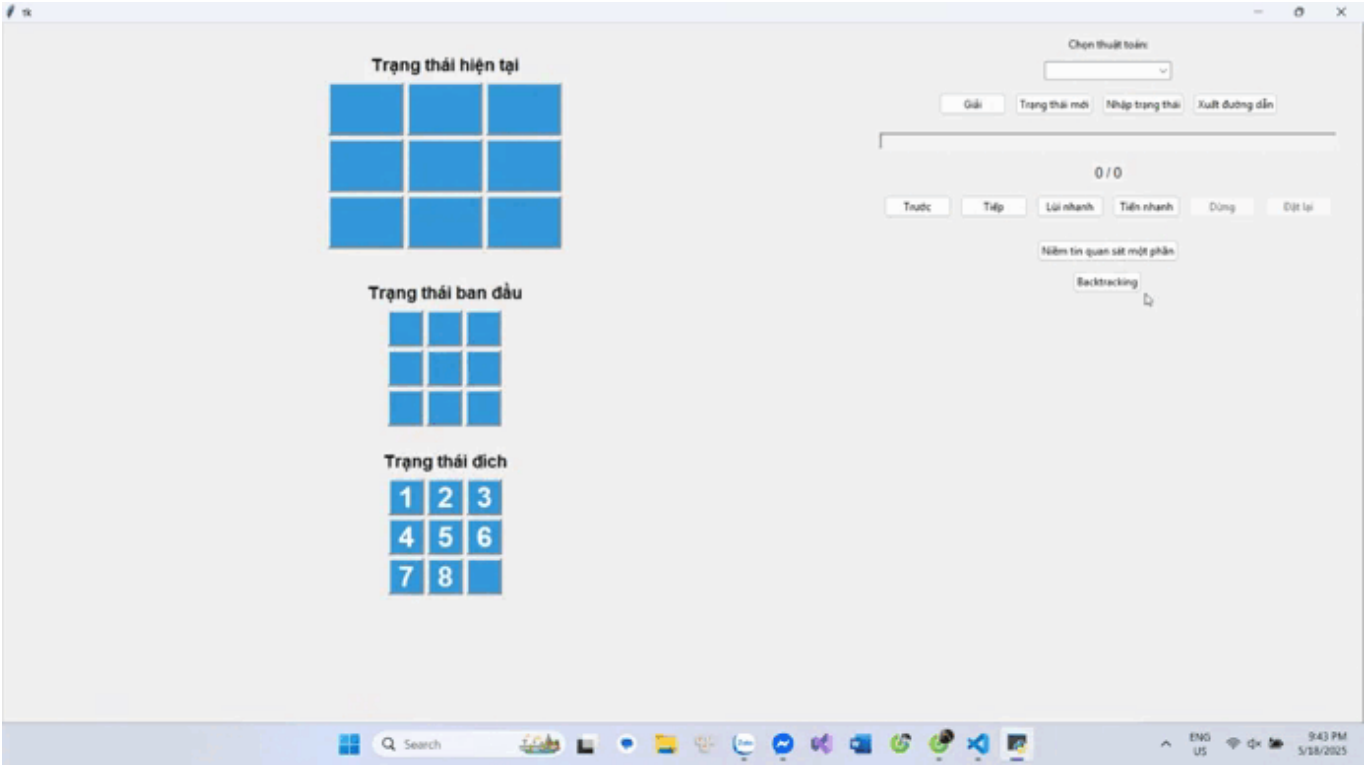
## 3.5. Tìm kiếm có điều kiện ràng buộc (Constraint satisfaction problem - CSP)

**Định nghĩa lý thuyết:** CSP là lớp bài toán trong đó lời giải được xác định bằng cách tìm tập giá trị thỏa mãn tập các ràng buộc (constraints) đã cho. Các bài toán CSP điển hình gồm: giải sudoku, tô màu bản đồ, phân công thời khóa biểu...

**Áp dụng vào bài 8-Puzzle:** 8-Puzzle vốn là bài toán theo chuỗi hành động, tuy nhiên có thể mô phỏng thành bài toán CSP bằng cách coi mỗi ô là một biến, mỗi vị trí là một giá trị, và ràng buộc là sự hợp lệ của từng bước di chuyển (chỉ một ô trống, không trùng lặp,...).

### 3.5.1. Tìm kiếm kiểm thử (Constraint Testing)

- **Lý thuyết:** Thử các tổ hợp giá trị cho biến và kiểm tra xem có thỏa mãn tất cả ràng buộc hay không.
- **Áp dụng:** Mỗi bước kiểm tra cấu hình bàn cờ hiện tại có thỏa mãn điều kiện của một trạng thái hợp lệ hay không.
- **Các bước:**
    1. Sinh một cấu hình trạng thái.
    2. Kiểm tra cấu hình đó có trùng giá trị hay sai định dạng không.
    3. Nếu hợp lệ, giữ lại; nếu không thì loại.
- **Độ phức tạp:** O(n!) với n là số biến.
- **Ưu điểm:** Đơn giản.
- **Nhược điểm:** Không hiệu quả, không mở rộng được cho bài toán lớn.
- **Link tham khảo:** Constraint Satisfaction Problem in AI
- **Nhận xét:** Mang tính nền tảng, chưa tối ưu.
- **Hình minh họa:**

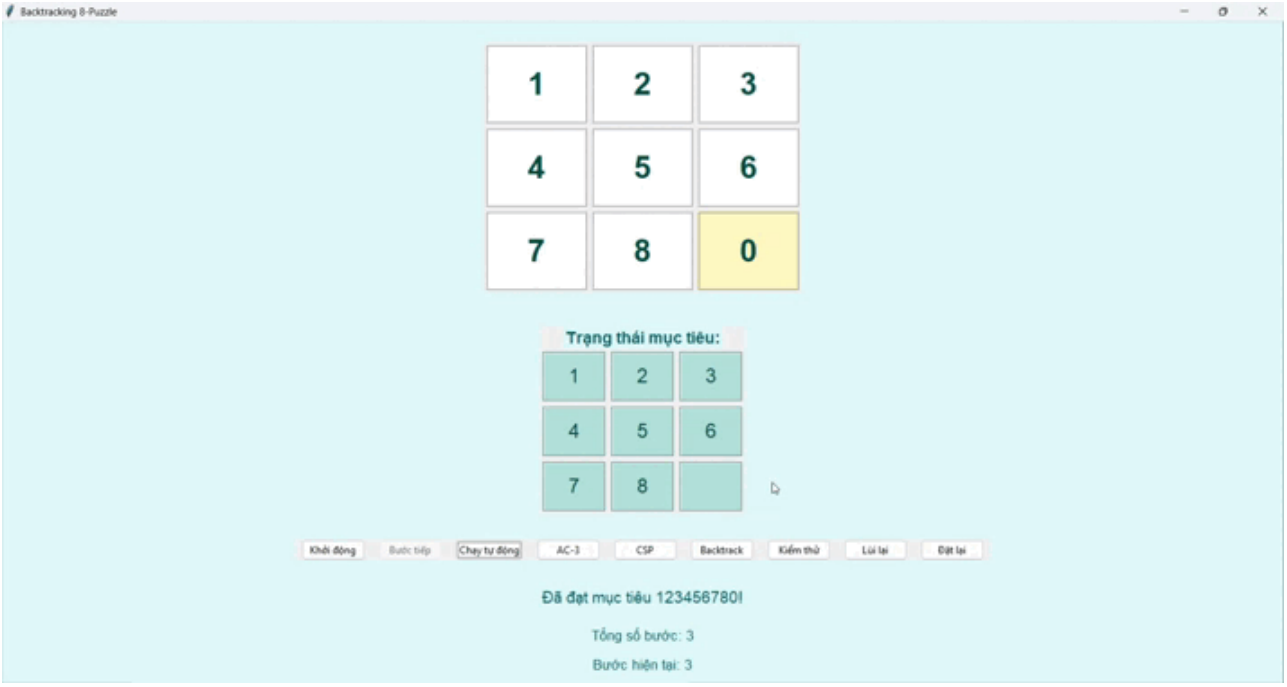



### 3.5.2. Backtracking CSP

- **Lý thuyết:** Gán giá trị cho từng biến theo thứ tự, lùi lại khi gặp xung đột.
- **Áp dụng:** Gán từng vị trí ô và quay lui khi bàn cờ không hợp lệ.
- **Các bước:**

    1. Chọn một biến chưa gán giá trị.
    2. Gán một giá trị khả dĩ.
    3. Nếu có xung đột thì quay lui, thử giá trị khác.
    4. Nếu không có xung đột, tiếp tục đến biến kế tiếp.

- **Độ phức tạp:** O(d^n)
- **Ưu điểm:** Giảm số lượng trạng thái cần kiểm tra.
- **Nhược điểm:** Vẫn chậm nếu không có heuristic hỗ trợ.
- **Link tham khảo:** Backtracking Algorithm
- **Nhận xét:** Có thể kết hợp với kiểm thử ràng buộc để nâng cao hiệu quả.
- **Hình minh họa:**

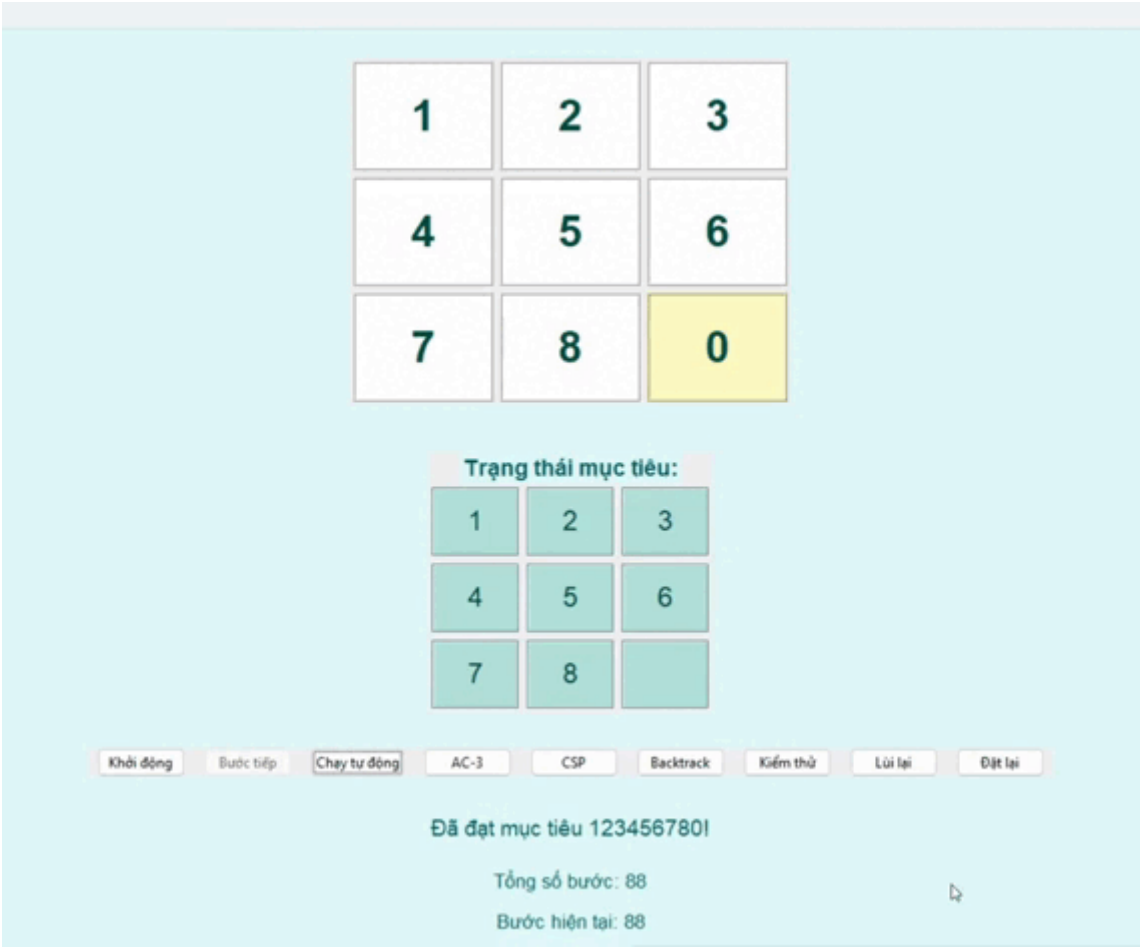



### 3.5.3. Backtracking kết hợp AC-3 (Arc Consistency 3)

- **Lý thuyết:** Thuật toán AC-3 được dùng để loại bỏ giá trị không hợp lệ sớm bằng cách duy trì tính nhất quán của ràng buộc nhị phân.
- **Áp dụng:** Kết hợp với Backtracking để rút gọn miền giá trị trước khi gán.
- **Các bước:**

    1. Áp dụng AC-3 để giảm miền giá trị.
    2. Tiến hành Backtracking trên miền đã thu gọn.
    3. Nếu xung đột xảy ra, quay lui và thử miền khác.

- **Độ phức tạp:** AC-3: O(ed^3), tổng hợp: cải thiện đáng kể so với backtracking đơn thuần.
- **Ưu điểm:** Hiệu quả cao hơn, giảm đáng kể không gian tìm kiếm.
- **Nhược điểm:** Cần cài đặt phức tạp hơn.
- **Link tham khảo:** Arc Consistency Algorithm
- **Nhận xét:** Giải pháp mạnh cho mô hình CSP phức tạp.
- **Hình minh họa:**




#### 3.5.4. Bảng So sánh các thuật toán CSP

| Thuật toán | Có hoàn chỉnh | Tối ưu | Hiệu quả tìm kiếm | Đặc điểm |
|---|---|---|---|---|
| Constraint Testing | Có | Không | Thấp | Duyệt toàn bộ trạng thái |
| Backtracking | Có | Có | Trung bình | Duyệt có quay lui |
| Backtracking + AC-3 | Có | Có | Cao | Rút gọn miền giá trị trước |

**Kết luận:** Mặc dù không phải là mô hình chuẩn của 8-Puzzle, việc áp dụng CSP có thể hữu ích trong việc kiểm tra hợp lệ cấu hình đầu vào, sinh trạng thái khởi tạo hoặc thiết kế bài toán biến thể.

## 3.6. Học tăng cường (Reinforcement learning)

**Định nghĩa lý thuyết:** Học tăng cường là một nhánh của học máy, trong đó một tác nhân (agent) học cách hành động trong môi trường để tối đa hóa phần thưởng (reward) dài hạn thông qua quá trình thử - sai. Thuật toán không cần biết trước hành động nào là tốt nhất mà học từ kinh nghiệm tương tác với môi trường.

**Áp dụng vào bài 8-Puzzle:** Mỗi trạng thái của bàn cờ là một trạng thái của môi trường, hành động là di chuyển ô trống, phần thưởng là điểm số (thường gán -1 mỗi bước để khuyến khích tối ưu số bước). Tác nhân học chính sách hành động tốt nhất từ nhiều lần tương tác.

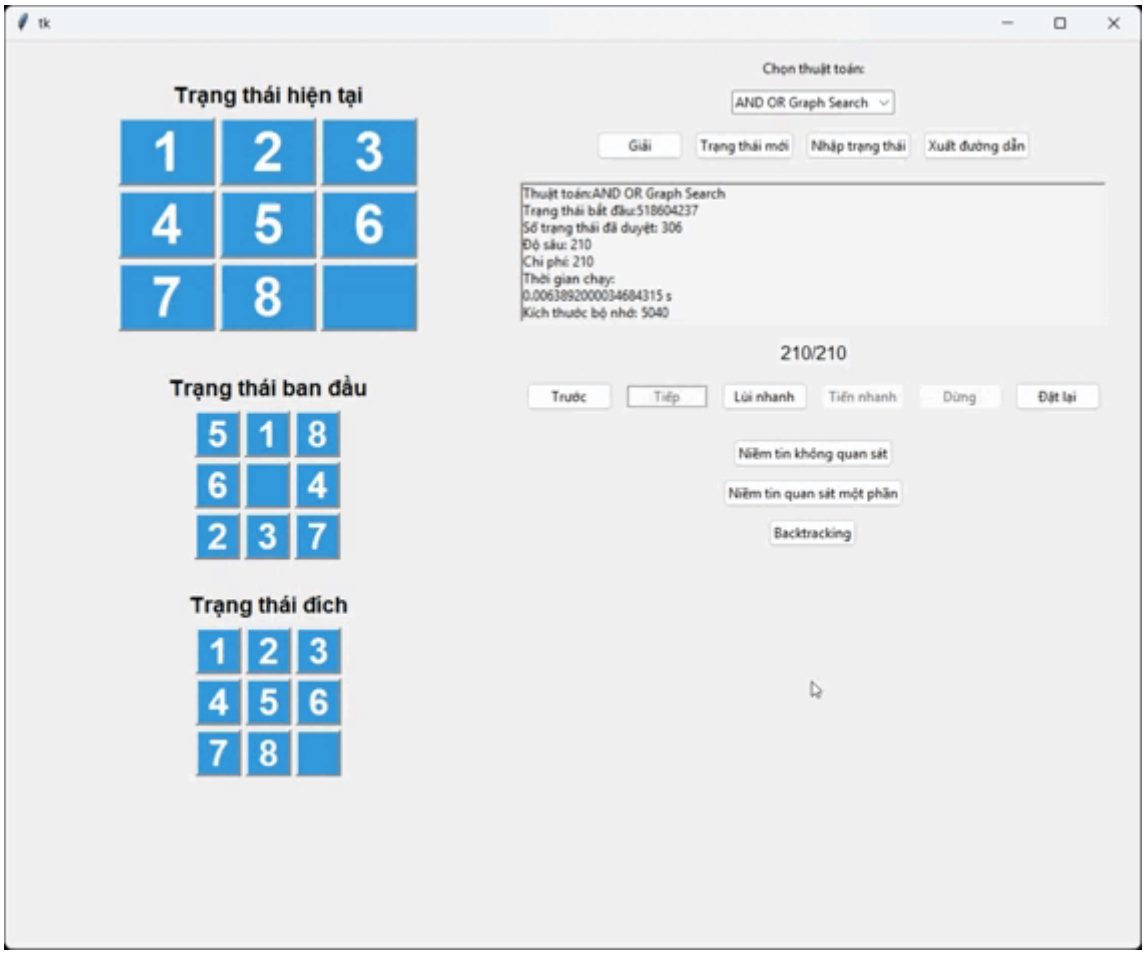
#### 3.6.1. Q-Learning

- **Lý thuyết:** Q-learning là thuật toán học tăng cường không mô hình (model-free), học giá trị hành động Q(s, a) thông qua hàm cập nhật:

  ```
  Q(s, a) <- Q(s, a) + α * [ r + γ * max_a' Q(s', a') - Q(s, a) ]
  ```

Trong đó: - Q(s, a): giá trị hiện tại của hành động a ở trạng thái s - α (alpha): tốc độ học - r: phần thưởng nhận được sau khi thực hiện hành động a - γ (gamma): hệ số chiết khấu - max_a' Q(s', a'): giá trị tối đa của các hành động khả thi ở trạng thái kế tiếp s'

- **Áp dụng:**
  - Trạng thái: cấu hình bàn cờ (mã hóa dạng tuple hoặc index).
  - Hành động: di chuyển ô trống (trái, phải, lên, xuống).
  - Phần thưởng: -1 cho mỗi bước, 0 khi đạt trạng thái đích.
- **Các bước thực hiện:**
  1. Khởi tạo bảng Q cho tất cả cặp trạng thái - hành động.
  2. Tại mỗi vòng lặp:
     - Chọn hành động theo chính sách (\varepsilon)-greedy.
     - Thực hiện hành động, quan sát trạng thái mới và phần thưởng.
     - Cập nhật giá trị Q theo công thức trên.
  3. Lặp lại cho đến khi chính sách hội tụ.
- **Độ phức tạp:** Phụ thuộc vào số trạng thái và số hành động (|S| × |A|).
- **Ưu điểm:** Không cần biết trước mô hình môi trường.
- **Nhược điểm:** Với không gian trạng thái lớn như 8-Puzzle, cần tối ưu hóa (ví dụ: sử dụng Deep Q-Network).
- **Link tham khảo:** Q-learning in Python
- **Nhận xét:** Là cơ sở quan trọng cho các thuật toán học tăng cường hiện đại. Tuy cần nhiều tập dữ liệu và thời gian huấn luyện, Q-learning có thể học được chiến lược giải 8-Puzzle hiệu quả.
- **Hình ảnh minh họa:**




## Nhận xét tổng quan

- Trong nhóm tìm kiếm có thông tin, thuật toán **A*** là lựa chọn tối ưu nhất cho bài toán 8-Puzzle tiêu chuẩn nhờ cân bằng hiệu quả và tối ưu lời giải.
- **IDA*** thích hợp khi bộ nhớ bị giới hạn nhưng chấp nhận thời gian dài hơn.
- Các thuật toán trong nhóm tìm kiếm không thông tin như **BFS** và **IDS** đảm bảo tối ưu nhưng hạn chế về bộ nhớ.
- Tìm kiếm cục bộ thích hợp khi không gian trạng thái quá lớn hoặc cần lời giải nhanh gần đúng, tuy nhiên không đảm bảo tối ưu và dễ mắc kẹt cực trị cục bộ.
- Học củng cố là hướng tiềm năng cho các bài toán phức tạp hơn nhưng đòi hỏi kỹ thuật biểu diễn tốt và thời gian học dài.
- Nhóm tìm kiếm trong môi trường không xác định và tìm kiếm có ràng buộc thường ứng dụng trong các biến thể mở rộng hoặc mô hình hóa đặc biệt của bài toán, với chi phí tính toán cao hơn.

---

# 4. KẾT LUẬN

Dự án **"8-Puzzle Solver with AI Algorithms"** đã hoàn thành thành công các mục tiêu đề ra: xây dựng một hệ thống giải bài toán 8-Puzzle bằng nhiều thuật toán trí tuệ nhân tạo, đồng thời phân tích, đánh giá và trực quan hóa quá trình hoạt động của từng phương pháp.

Qua việc triển khai và kiểm thử 6 nhóm thuật toán – từ các chiến lược tìm kiếm truyền thống (BFS, DFS, A*) đến các thuật toán hiện đại như Q-Learning và mô hình hóa ràng buộc – dự án đã chứng minh rằng không tồn tại một thuật toán duy nhất tối ưu cho mọi hoàn cảnh. Mỗi thuật toán có thế mạnh riêng tùy vào mục tiêu cụ thể: tốc độ, bộ nhớ, độ chính xác hay khả năng học hỏi từ môi trường.

### Những điểm nổi bật của dự án:

- **Tích hợp đa dạng thuật toán AI:** Cho phép người dùng lựa chọn nhiều chiến lược giải khác nhau trên cùng một bài toán.
- **Giao diện trực quan sinh động:** Hiển thị rõ ràng quá trình giải bài toán, hỗ trợ người học và người dùng dễ tiếp cận và quan sát kết quả.
- **Bảng so sánh hiệu suất và đánh giá chi tiết:** Giúp làm rõ điểm mạnh, điểm yếu và khả năng ứng dụng thực tiễn của từng phương pháp.

### Giá trị thực tiễn:

Dự án không chỉ phục vụ mục tiêu học tập và nghiên cứu mà còn là nền tảng hữu ích cho việc:

- Giảng dạy và minh họa thuật toán AI trong các khóa học tin học và trí tuệ nhân tạo.
- Phát triển các hệ thống giải bài toán logic, tối ưu hóa trạng thái hoặc ra quyết định tự động.
- Là bước đệm để mở rộng sang các dự án AI phức tạp hơn như Rubik Solver, Robot Navigation, hoặc các game chiến thuật AI.

### Định hướng phát triển tiếp theo:

- Mở rộng bài toán lên 15-Puzzle hoặc các biến thể động.
- Tích hợp thêm các thuật toán hiện đại như Deep Q-Learning hoặc Monte Carlo Tree Search.
- Đưa ứng dụng lên nền tảng web, thiết bị di động, hoặc làm module tương tác cho giáo dục AI.

---

## 5. VIDEO DEMO

Linh video youtube

---