# BÁO CÁO PHÂN TÍCH CHƯƠNG TRÌNH GIẢI 8-PUZZLE

**Mã sinh viên:** 23133030

**Họ và tên:** Đỗ Kiến Hưng

**GitHub**: https://github.com/darktheDE/AI_PersonalProject

## MỤC LỤC

# NỘI DUNG CHI TIẾT

## 1. Tổng quan chương trình

### 1.1. Mục tiêu và Phạm vi

Chương trình được thiết kế với mục tiêu chính là cung cấp một môi trường để giải quyết bài toán 8-Puzzle cổ điển và các biến thể phức tạp hơn của nó. Vượt xa một công cụ giải đố đơn thuần, chương trình này đóng vai trò như một nền tảng học tập, trình diễn và nghiên cứu mạnh mẽ, cho phép người dùng:

- Triển khai và thử nghiệm một loạt các thuật toán Trí tuệ Nhân tạo (AI).
- Trực quan hóa quá trình tìm kiếm và các bước của lời giải.
- Phân tích và so sánh hiệu suất của các thuật toán khác nhau thông qua các số liệu thống kê và biểu đồ.
- Khám phá cách giải quyết vấn đề trong các môi trường không chắc chắn và quan sát được một phần.

### 1.2. Bài toán 8-Puzzle: Từ kinh điển đến các môi trường phức tạp

Bài toán 8-Puzzle là một bài toán tìm đường đi tiêu chuẩn trong AI. Nó bao gồm một lưới 3x3 với 8 ô được đánh số và một ô trống. Mục tiêu là di chuyển các ô để đạt được một trạng thái đích xác định. Chương trình này mở rộng bài toán này sang:

- **Bài toán 8-Puzzle Tiêu chuẩn:** Trạng thái hoàn toàn quan sát được, hành động có kết quả xác định.
- **8-Puzzle Không Xác Định (Nondeterministic):** Được mô hình hóa trong thuật toán and_or_graph_search, nơi một hành động có thể dẫn đến nhiều kết quả khác nhau với một xác suất nhất định (ví dụ: di chuyển "trượt").
- **8-Puzzle Quan Sát Được Một Phần (Partially Observable):** Được mô hình hóa trong thuật toán belief_state_search, nơi một số ô trên bàn cờ bị ẩn, và tác nhân phải duy trì một "trạng thái niềm tin" (tập hợp các trạng thái có thể) để hành động.

## 2. Kiến trúc và Thiết kế

Chương trình được cấu trúc theo mô hình ba lớp chính, đảm bảo tính module, dễ bảo trì và mở rộng:

### 2.1. Lớp PuzzleState

- **Chức năng:** Đóng gói tất cả thông tin liên quan đến một cấu hình cụ thể của bàn cờ 8-Puzzle.
- **Thuộc tính chính:**
  - board: Ma trận 3x3 (list of lists) biểu diễn cấu hình.
  - parent: Tham chiếu đến PuzzleState cha, dùng để truy vết đường đi.
  - move: Hành động ("UP", "DOWN", "LEFT", "RIGHT") dẫn đến trạng thái này.
  - depth: Độ sâu của trạng thái trong cây tìm kiếm (chi phí g(n) cho nhiều thuật toán).
  - cost: Chi phí tích lũy để đến trạng thái này (dùng cho UCS, A*, IDA*).
  - key: Biểu diễn chuỗi của board, tối ưu cho việc kiểm tra trùng lặp trong set và dict.
  - blank_row, blank_col: Tọa độ của ô trống, giúp tăng tốc việc tạo trạng thái con.
- **Phương thức quan trọng:**
  - __init__: Khởi tạo trạng thái, tìm ô trống.
  - __lt__, __eq__, __hash__: Cho phép sử dụng đối tượng PuzzleState trong hàng đợi ưu tiên (heapq) và tập hợp (set).
  - get_children(): Sinh ra tất cả các trạng thái con hợp lệ có thể đạt được bằng một bước di chuyển ô trống.

### 2.2. Lớp PuzzleSolver

- **Chức năng:** Là "bộ não" của chương trình, chứa logic triển khai của tất cả các thuật toán giải đố.
- **Thuộc tính chính:**
  - goal_state: Lưu trữ cấu hình đích của puzzle.
- **Hàm Heuristic:**
  - get_manhattan_distance(): Tính tổng khoảng cách Manhattan của các ô đến vị trí đích.
  - get_misplaced_tiles(): Đếm số ô không ở đúng vị trí đích.
- **Hàm Tiện Ích:**
  - is_goal(): Kiểm tra xem một trạng thái có phải là đích không.
  - get_path(): Tái tạo đường đi từ trạng thái gốc đến trạng thái hiện tại.
  - get_random_state(): Tạo một trạng thái puzzle ngẫu nhiên hợp lệ (dùng cho Random Restart Hill Climbing).
  - generate_path_from_moves(): Xây dựng một đường đi PuzzleState từ một chuỗi các hành động (dùng cho Genetic Algorithm).
  - csp_callback(): Hàm gọi lại đặc biệt được truyền cho các thuật toán CSP để cập nhật GUI trong quá trình xây dựng lời giải.
- **Triển khai Thuật Toán:** (Chi tiết ở Mục 3)

### 2.3. Lớp PuzzleGUI

- **Chức năng:** Xây dựng và quản lý toàn bộ giao diện người dùng đồ họa (GUI), xử lý tương tác của người dùng, và hiển thị kết quả cũng như quá trình giải.
- **Thư viện sử dụng:** tkinter và tkinter.ttk cho các widget giao diện, matplotlib và matplotlib.backends.backend_tkagg cho việc nhúng biểu đồ.
- **Thuộc tính chính:**
  - root: Cửa sổ chính tk.Tk().
  - initial_state, goal_state: Cấu hình puzzle.
  - solver: Một instance của PuzzleSolver.
  - solution_path, current_step, animation_speed: Biến quản lý animation.

- Các biến tk.StringVar, tk.IntVar, tk.BooleanVar để liên kết với các widget điều khiển.
- cell_frames: Danh sách các frame và label của lưới puzzle.
- viz_container: Frame chứa các biểu đồ Matplotlib.

- **Phương thức quan trọng:**
  - create_widgets(), create_ribbon(), create_visualization_controls(): Thiết lập các thành phần giao diện.
  - solve_puzzle(): Gọi thuật toán đã chọn từ PuzzleSolver, xử lý kết quả và cập nhật GUI.
  - update_puzzle_display(): Cập nhật lưới puzzle.
  - Các hàm điều khiển animation: step_forward(), step_back(), play_solution(), update_speed().
  - Các hàm chức năng: save_configuration(), load_configuration(), generate_random_puzzle(), toggle_goal_display(), run_benchmark(), show_about(), show_help().
  - **2.3.1. Tích hợp Matplotlib cho Trực quan hóa Dữ liệu:**
    - Các phương thức plot_benchmark_results(), plot_solution_stats(), plot_search_tree(), plot_heuristic_analysis(), plot_search_structure(), plot_performance_metrics() sử dụng matplotlib.pyplot để tạo các biểu đồ.
    - FigureCanvasTkAgg được dùng để nhúng các Figure của Matplotlib vào một widget Tkinter (cụ thể là self.viz_container hoặc các tab con của nó).
    - Điều này cho phép hiển thị các phân tích đồ họa về hiệu suất thuật toán, đặc điểm của lời giải, và cấu trúc không gian tìm kiếm ngay trong ứng dụng.

## 3. Các Thuật Toán Giải Đố Được Triển Khai

PuzzleSolver triển khai một bộ sưu tập thuật toán đa dạng và mạnh mẽ, bao gồm:

### 3.1. Tìm kiếm Không Thông Tin (Uninformed Search)

Các thuật toán này không sử dụng thông tin về "khoảng cách" đến đích.

- bfs(): Tìm kiếm theo Chiều Rộng (Đảm bảo lời giải ngắn nhất về số bước).
- dfs(): Tìm kiếm theo Chiều Sâu (Có thể nhanh nếu may mắn, nhưng có thể lạc lối; hỗ trợ giới hạn độ sâu).
- ucs(): Tìm kiếm Chi Phí Đồng Nhất (Đảm bảo lời giải có tổng chi phí thấp nhất).
- iterative_deepening(): Tìm kiếm Sâu Dần (Kết hợp ưu điểm bộ nhớ của DFS và tính đầy đủ, tối ưu của BFS).

### 3.2. Tìm kiếm Có Thông Tin (Informed Search / Heuristic Search)

Sử dụng hàm đánh giá heuristic (h(n)) để ước lượng khoảng cách đến đích.

- greedy_search(): Tìm kiếm Tham lam Best-First (Chỉ dựa vào h(n), nhanh nhưng không đảm bảo tối ưu).
- a_star_search(): Tìm kiếm A* (Sử dụng f(n) = g(n) + h(n), đảm bảo tối ưu nếu heuristic là admissible).
- ida_star_search(): Tìm kiếm Sâu Dần A* (IDA*) (Phiên bản A* tiết kiệm bộ nhớ).

### 3.3. Tìm kiếm Cục Bộ (Local Search)

Hoạt động trên một trạng thái hiện tại và cố gắng cải thiện nó bằng cách di chuyển đến các trạng thái lân cận.

- simple_hill_climbing(): Leo Đồi Đơn Giản (Chọn láng giềng tốt hơn đầu tiên).
- steepest_ascent_hill_climbing(): Leo Đồi Dốc Nhất (Chọn láng giềng tốt nhất).
- random_restart_hill_climbing(): Leo Đồi với Khởi Tạo Ngẫu Nhiên (Giúp thoát khỏi cực đại địa phương).
- simulated_annealing(): Luyện Kim Mô Phỏng (Cho phép các bước di chuyển xấu hơn với xác suất giảm dần để thoát cực đại địa phương).
- beam_search(): Tìm kiếm Chùm Tia (Giữ lại một số lượng beam_width trạng thái tốt nhất ở mỗi mức).

- genetic_algorithm(): Thuật toán Di Truyền (Mô phỏng tiến hóa tự nhiên với các toán tử lai ghép, đột biến, lựa chọn).

### 3.4. Bài toán Thỏa mãn Ràng buộc (Constraint Satisfaction Problems - CSP)

Các thuật toán này xây dựng lời giải từ đầu bằng cách gán giá trị cho các biến sao cho thỏa mãn các ràng buộc.

- backtracking_search(): Tìm kiếm Quay Lui Cơ Bản.
- intelligent_backtracking(): Tìm kiếm Quay Lui Thông Minh (sử dụng heuristic MRV và LCV).
- min_conflicts(): Thuật toán Xung Đột Tối Thiểu (sửa chữa một cấu hình ban đầu bằng cách giảm thiểu xung đột).

### 3.5. Học Tăng Cường (Reinforcement Learning)

Tác nhân học cách hành động trong một môi trường thông qua thử và sai để tối đa hóa phần thưởng.

- q_learning(): Thuật toán Q-Learning (học giá trị của các cặp trạng thái-hành động).

### 3.6. Thuật toán cho Môi trường Không Chắc Chắn và Quan Sát Một Phần

Giải quyết các vấn đề phức tạp hơn khi kết quả hành động không chắc chắn hoặc thông tin về trạng thái bị hạn chế.

- **3.6.1. Tìm kiếm Đồ thị AND-OR (and_or_graph_search)**
    - **Mục tiêu:** Xử lý các vấn đề mà hành động có thể có nhiều kết quả không xác định (ví dụ: di chuyển có thể "trượt" sang hướng khác với một xác suất nhất định).
    - **Cách tiếp cận:** Xây dựng một *kế hoạch có điều kiện* (conditional plan) thay vì một chuỗi hành động tuyến tính. Kế hoạch này bao gồm các nhánh để xử lý các kết quả khác nhau của hành động.

    - **Cấu trúc:** Sử dụng các nút OR (đại diện cho lựa chọn hành động của tác nhân) và nút AND (đại diện cho các kết quả có thể xảy ra của một hành động).
    - **Kết quả:** Trả về một kế hoạch (nếu thành công) chỉ ra hành động cần thực hiện ở mỗi trạng thái có thể gặp phải.
- **3.6.2. Tìm kiếm Không Gian Niềm Tin (belief_state_search)**
    - **Mục tiêu:** Giải quyết vấn đề khi tác nhân không thể quan sát đầy đủ trạng thái hiện tại của môi trường (ví dụ: một số ô puzzle bị ẩn).
    - **Cách tiếp cận:** Tác nhân duy trì một *trạng thái niềm tin (belief state)*, là một tập hợp tất cả các trạng thái vật lý mà môi trường có thể đang ở, dựa trên lịch sử hành động và quan sát.
    - **Hoạt động:** Tìm kiếm trong không gian các trạng thái niềm tin. Mỗi hành động sẽ chuyển đổi trạng thái niềm tin hiện tại thành một trạng thái niềm tin mới.
    - **Đặc điểm:** Số lượng trạng thái trong một belief state có thể rất lớn, làm tăng độ phức tạp. Thuật toán cần cơ chế cập nhật belief state khi có thông tin mới (ví dụ, một ô ẩn được di chuyển và lộ diện).

## 4. Giao Diện Người Dùng (GUI) Chi Tiết

Giao diện được thiết kế thân thiện và cung cấp nhiều chức năng, tổ chức như sau:

### 4.1. Thanh Ribbon Điều Hướng

Nằm ở phía trên cùng, được tổ chức thành các tab (ttk.Notebook):

- **File:** "Reset", "Save Config", "Load Config", "Exit".
- **Puzzle:** "Random" (tạo puzzle ngẫu nhiên), "Difficulty" (chọn độ khó), "Show Goal State" (checkbox).
- **Tools:** "Benchmark" (so sánh thuật toán), "Animation Speed" (thanh trượt).
- **Help:** "About" (thông tin chương trình), "Help" (tài liệu hướng dẫn).

### 4.2. Panel Hiển Thị Puzzle (Trái, trên)

- Lưới 3x3 (ttk.Frame chứa ttk.Label) hiển thị trực quan các ô số và ô trống.
- Cập nhật động trong quá trình giải và khi xem animation.

### 4.3. Panel Thông Tin, Thống Kê và Chi Tiết Thuật Toán (Trái, dưới, dạng Tab ttk.Notebook)

- **Information Tab:** Hiển thị thông tin tổng quan về thuật toán đang chạy/đã chạy: tên thuật toán, số bước giải, thời gian thực thi, số nút đã duyệt, kích thước hàng đợi/ngăn xếp tối đa.
- **Statistics Tab:** Cung cấp các số liệu thống kê chi tiết hơn: số nút duyệt trên giây. Các mục "Memory usage" và "Solution optimality" có thể được mở rộng trong tương lai.
- **Algorithm Details Tab:** (tk.Text có thanh cuộn) Hiển thị mô tả về thuật toán được chọn, đặc biệt là mô hình CSP và các khái niệm liên quan đến các thuật toán nâng cao.

### 4.4. Panel Điều Khiển Thuật Toán và Animation (Phải, có thanh cuộn)

- **Algorithm Selection:** Danh sách ttk.Radiobutton cho phép chọn một trong số rất nhiều thuật toán, bao gồm cả AND-OR Graph Search và Belief State Search.
- **Solve Button:** Khởi chạy thuật toán đã chọn.
- **Animation Controls:** Các nút "<<", "Play", ">>" để điều khiển xem lại các bước giải.
- **Speed Control:** ttk.Scale điều chỉnh tốc độ animation.
- **Reset Button:** Đặt lại puzzle.

### 4.5. Tab Trực Quan Hóa Dữ Liệu (Visualization - Trái, dưới, một tab trong info_tabs)

- **4.5.1. Điều khiển tạo biểu đồ:**
    - Nút "Generate Visualizations": Kích hoạt việc vẽ tất cả các biểu đồ phân tích dựa trên lời giải hiện tại (nếu có).
- **4.5.2. Các loại biểu đồ (hiển thị trong các tab con của viz_notebook):**
    - **Heuristic Analysis Tab:**
        - Biểu đồ đường thể hiện sự thay đổi của các giá trị heuristic (Manhattan Distance, Misplaced Tiles) qua từng bước của đường đi lời giải. Giúp đánh giá tính "dẫn đường" của heuristic.
    - **Search Structure Tab:**
        - Minh họa đơn giản về cấu trúc cây tìm kiếm, làm nổi bật đường đi của lời giải và các nút đã được mở rộng (mô phỏng). Giúp hình dung không gian tìm kiếm đã được khám phá.
    - **Performance Tab:**
        - Biểu đồ cột thể hiện số lượng nút được mở rộng ở mỗi độ sâu của cây tìm kiếm. Giúp phân tích sự phân bố của nỗ lực tìm kiếm.
    - **Trong cửa sổ Benchmark:** Sau khi hoàn thành, nút "Visualize Results" cho phép vẽ biểu đồ cột so sánh thời gian thực thi và số nút đã mở rộng của các thuật toán đã chạy trong benchmark.

## 5. Hướng Dẫn Sử Dụng và Chức Năng Chính

1. **Khởi chạy và Thiết lập Ban đầu:** Chạy file Python. Giao diện hiển thị với puzzle ban đầu.
2. **Chọn và Chạy Thuật Toán:**
    - Từ panel bên phải, chọn thuật toán mong muốn từ danh sách mở rộng.
    - Nhấn "Solve". Chương trình sẽ thực thi thuật toán.
    - Đối với các thuật toán phức tạp như and_or_graph_search hoặc belief_state_search, quá trình có thể mất nhiều thời gian hơn và kết quả có thể là một kế hoạch hoặc một trạng thái niềm tin.

3. **Tương tác với Lời Giải (Animation):** (Áp dụng cho các thuật toán trả về một đường đi PuzzleState tuần tự)
    - Sử dụng "Play", "<<", ">>" và thanh trượt "Speed".
4. **Tạo Puzzle Ngẫu Nhiên:** Sử dụng nút "Random" trên Ribbon.
5. **Lưu và Tải Cấu Hình:** Sử dụng các nút "Save Config", "Load Config" trên Ribbon.
6. **Chạy Benchmark và Trực Quan Hóa Kết Quả So Sánh:**
    - Trong tab "Tools" trên Ribbon, nhấn "Benchmark".
    - Một cửa sổ mới sẽ hiển thị quá trình chạy các thuật toán.
    - Sau khi hoàn tất, nhấn nút **"Visualize Results"** trong cửa sổ benchmark để xem biểu đồ so sánh hiệu suất (thời gian, số nút) của các thuật toán.
7. **Sử dụng Tab Trực Quan Hóa để Phân Tích Lời Giải:**
    - Sau khi một thuật toán (thường là các thuật toán tìm đường đi tuần tự) tìm ra lời giải, chuyển đến tab "Visualization" (bên trái, dưới).
    - Nhấn nút **"Generate Visualizations"**.
    - Xem các biểu đồ trong các tab con "Heuristic Analysis", "Search Structure", "Performance" để hiểu rõ hơn về quá trình tìm kiếm và đặc điểm của lời giải.

## 6. Phân Tích và Đánh Giá

### 6.1. Ưu điểm Nổi Bật

- **Phạm vi thuật toán Vô cùng Rộng:** Từ các thuật toán tìm kiếm cơ bản, thuật toán cục bộ, CSP, học tăng cường, đến các thuật toán tiên tiến cho môi trường không chắc chắn và quan sát một phần. Đây là một bộ công cụ AI thực sự toàn diện.
- **Khả năng Trực quan hóa Dữ liệu Xuất sắc:** Việc tích hợp matplotlib để vẽ nhiều loại biểu đồ phân tích (heuristic, cấu trúc tìm kiếm, hiệu suất, so sánh benchmark) là một điểm mạnh vượt trội, nâng cao giá trị học thuật và phân tích của chương trình.

- **Giao diện Người dùng Trực quan và Đa năng:** Thiết kế GUI rõ ràng, dễ sử dụng với thanh Ribbon, các panel chức năng, và tab trực quan hóa giúp người dùng dễ dàng tương tác và khám phá.
- **Kiến trúc Phần mềm Tốt:** Phân chia rõ ràng thành các lớp PuzzleState, PuzzleSolver, PuzzleGUI giúp mã nguồn dễ đọc, bảo trì và mở rộng.
- **Công cụ Benchmark Hiệu quả:** Chức năng benchmark cùng với khả năng trực quan hóa kết quả so sánh cung cấp một công cụ mạnh mẽ để đánh giá khách quan các thuật toán.
- **Tính Giáo dục Cao:** Chương trình là một công cụ tuyệt vời để học và dạy về các khái niệm tìm kiếm và giải quyết vấn đề trong AI.

### 6.2. Hạn chế và Các Vấn đề Cần Lưu Ý

- **Độ phức tạp và Hiệu suất của các Thuật toán Nâng cao:**
    - belief_state_search: Không gian các trạng thái niềm tin có thể bùng nổ, khiến thuật toán chỉ khả thi với số lượng ô ẩn (num_hidden) nhỏ và độ sâu tìm kiếm (max_depth) hạn chế.
    - and_or_graph_search: Độ phức tạp có thể cao do tính chất đệ quy và các nhánh AND. Việc tìm kiếm một kế hoạch có điều kiện đầy đủ thường khó khăn hơn tìm một chuỗi hành động.
- **Minh họa Cây Tìm kiếm/Cấu trúc Tìm kiếm:** Các biểu đồ plot_search_tree và plot_search_structure hiện tại là các minh họa đơn giản. Trực quan hóa một cây tìm kiếm thực tế với hàng ngàn hoặc hàng triệu nút là một thách thức lớn về cả tính toán và hiển thị.
- **Callback Trực quan hóa:** Hàm csp_callback hoạt động tốt cho CSP. Mở rộng callback để hiển thị trực tiếp quá trình duyệt nút của các thuật toán tìm kiếm truyền thống (BFS, A*,...) có thể làm chậm đáng kể các thuật toán này nếu không được tối ưu cẩn thận.

- **Thông tin Thống kê Chi tiết:** Các mục như "Memory usage" và "Solution optimality" (ví dụ: so sánh độ dài đường đi tìm được với đường đi tối ưu thực sự, nếu biết) có thể được làm chi tiết hơn.

### 6.3. Đánh giá Hiệu Suất Thời Gian của Các Nhóm Thuật Toán

- **Tìm kiếm Không Thông tin (BFS, DFS, UCS, ID):** Thời gian tăng theo hàm mũ với độ sâu của lời giải (BFS, UCS, ID) hoặc có thể rất biến động (DFS).
- **Tìm kiếm Có Thông Tin (Greedy, A*, IDA*):** Hiệu quả hơn nhiều nếu heuristic tốt. A* và IDA* thường là lựa chọn tốt nhất cho 8-Puzzle tiêu chuẩn về cân bằng giữa thời gian và tính tối ưu.
- **Tìm kiếm Cục Bộ (Hill Climbing, SA, Beam, GA):** Thường nhanh về thời gian thực thi mỗi bước nhưng có thể không tìm thấy lời giải hoặc lời giải không tối ưu. Thời gian tổng phụ thuộc vào số lần lặp, số lần khởi động lại, kích thước quần thể, v.v.
- **CSP (Backtracking, Intelligent Backtracking, Min-Conflicts):** Cực kỳ nhanh khi áp dụng để *xây dựng* cấu hình đích cố định từ đầu do không gian tìm kiếm bị ràng buộc chặt chẽ.
- **Học Tăng Cường (Q-Learning):** Giai đoạn huấn luyện qua nhiều episodes sẽ tốn thời gian đáng kể. Sau khi học xong, việc trích xuất chính sách có thể nhanh.
- **Thuật toán cho Môi trường Phức tạp:**
    - and_or_graph_search: Thời gian phụ thuộc vào max_depth, slip_probability và cấu trúc của không gian vấn đề. Có thể rất chậm.
    - belief_state_search: Thời gian tăng nhanh theo num_hidden và max_depth do kích thước của không gian niềm tin.

### 6.4. Đề xuất Hướng Phát Triển và Cải Tiến

- **Tham số hóa Thuật toán Nâng cao từ GUI:** Cho phép người dùng tùy chỉnh các tham số đặc thù của từng thuật toán (ví dụ: num_hidden cho Belief State Search,

slip_probability cho AND-OR, beam_width, mutation_rate, các tham số của Q-Learning) trực tiếp trên giao diện.

- **Cải thiện Trực quan hóa Cây Tìm kiếm:**
    - Xem xét sử dụng các thư viện chuyên dụng hơn cho việc vẽ đồ thị/cây (ví dụ: graphviz thông qua Python binding, hoặc các thư viện JavaScript nếu chuyển sang web).
    - Cho phép tương tác với cây tìm kiếm (zoom, pan, click vào nút để xem thông tin).
- **Lưu/Tải Trạng thái Benchmark/Trực quan hóa:** Cho phép người dùng lưu kết quả benchmark hoặc các biểu đồ đã tạo ra (ví dụ: xuất ảnh).
- **Tối ưu hóa Hiệu suất:** Đối với các thuật toán duyệt nhiều nút, xem xét các kỹ thuật tối ưu hóa (ví dụ: sử dụng Cython cho các phần tính toán nặng).
- **So sánh Heuristic Động:** Thêm chức năng cho phép người dùng chọn các heuristic khác nhau và xem trực tiếp ảnh hưởng của chúng lên quá trình tìm kiếm và các biểu đồ phân tích.
- **Giao diện Chỉnh sửa Puzzle:** Cho phép người dùng tự thiết kế trạng thái ban đầu của puzzle bằng cách kéo thả hoặc nhập số.

## 7. Kết Luận

Chương trình là một thành quả trong việc xây dựng một công cụ giải 8-Puzzle và khám phá các khái niệm AI. Sự phong phú về thuật toán, từ các phương pháp tìm kiếm cơ bản đến các kỹ thuật xử lý môi trường không chắc chắn và quan sát một phần, kết hợp với khả năng trực quan hóa dữ liệu mạnh mẽ bằng Matplotlib, đã tạo nên một ứng dụng có giá trị học thuật và thực tiễn cao. Mặc dù có một số thách thức về hiệu suất và độ phức tạp đối với các thuật toán nâng cao, chương trình này cung cấp một nền tảng vững chắc và nhiều tiềm năng để tiếp tục phát triển và nghiên cứu sâu hơn về Trí tuệ Nhân tạo.