**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC ĐẠI NAM**

**Logo, company name

Description automatically generated**

**BÁO CÁO HỌC PHẦN**

**TÊN HỌC PHẦN: TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI: ÁP DỤNG THUẬT TOÁN A\* ĐỂ LẬP TRÌNH CHƯƠNG TRÌNH TRÒ CHƠI SẮP XẾP VÒNG**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Mã Sinh Viên** | **Họ và Tên** | **Ngày Sinh** | **Lớp** |
| **1** | **1771020649** | **Nguyễn Trường Thịnh** | **31/03/2005** | **CNTT – 1704** |
| **2** | **1771020624** | **Nguyễn Quang Thắng** | **24/06/2005** | **CNTT – 1704** |
| **3** | **1771020112** | **Đỗ Mạnh Cường** | **20/02/2005** | **CNTT – 1704** |

### 

**Hà Nội, năm 2025**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC ĐẠI NAM**

**Logo, company name

Description automatically generated**

**BÁO CÁO HỌC PHẦN**

**TÊN HỌC PHẦN: TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI: ÁP DỤNG THUẬT TOÁN A\* ĐỂ LẬP TRÌNH CHƯƠNG TRÌNH TRÒ CHƠI SẮP XẾP VÒNG**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Mã Sinh Viên** | **Họ và Tên** | **Ngày Sinh** | **Điểm** | |
| **Bằng Số** | **Bằng Chữ** |
| **1** | **1771020649** | **Nguyễn Trường Thịnh** | **31/03/2005** |  |  |
| **2** | **1771020624** | **Nguyễn Quang Thắng** | **24/06/2005** |  |  |
| **3** | **1771020112** | **Đỗ Mạnh Cường** | **20/02/2005** |  |  |

### 

**CÁN BỘ CHẤM THI 1 CÁN BỘ CHẤM THI 2**

**Hà Nội, năm 2025**

**MỤC LỤC**

LỜI NÓI ĐẦU 45

[CHƯƠNG 1. GIỚI THIỆU 5](#_Toc193487804)

[**1.1. Mô tả trò chơi 5**](#_Toc193487805)

[**1.2. Mục tiêu bài toán 6**](#_Toc193487806)

[CHƯƠNG 2. THUẬT TOÁN A\* 7](#_Toc193487807)

[**2.1. Nguyên lý thuật toán A\* 7**](#_Toc193487808)

[**2.1.1. Thuật toán A là gì?\* 7**](#_Toc193487809)

[**2.1.2. Nguyên lý cơ bản của A\* 7**](#_Toc193487810)

[**2.1.3. Các bước thực hiện 8**](#_Toc193487811)

[**2.1.4. Ưu điểm và nhược điểm 8**](#_Toc193487812)

[**2.1.5. Ứng dụng thực tế 8**](#_Toc193487813)

[**2.2. Hàm heuristic đề xuất 9**](#_Toc193487814)

[**2.1. Thuật toán A\* nguyên lý 9**](#_Toc193487815)

[**2.2. Hàm xuất đề heuristic 10**](#_Toc193487816)

[CHƯƠNG 3. TRIỂN KHAI TRÒ CHƠI 12](#_Toc193487817)

[**3.1. Cấu trúc dữ liệu 12**](#_Toc193487818)

[**3.2. Các bước thực hiện thuật toán 13**](#_Toc193487819)

[**3.3. Hiển thị trạng thái trò chơi 14**](#_Toc193487820)

[CHƯƠNG 4. HƯỚNG DẪN LẬP TRÌNH 16](#_Toc193487821)

[**4.1. Cấu trúc chương trình 16**](#_Toc193487822)

[**4.1.1. Class GameState: Đại diện trạng thái trò chơi. 16**](#_Toc193487823)

[**4.1.2. Class AStarSolver: Cài đặt thuật toán A\* để tìm giải pháp tối ưu. 16**](#_Toc193487824)

[**4.1.3. Hàm nhập lệnh: Cho phép người chơi nhập lệnh di chuyển vòng. 17**](#_Toc193487825)

[**4.2. Gợi ý triển khai Python 17**](#_Toc193487826)

[CHƯƠNG 5. ĐÁNH GIÁ VÀ CẢI TIẾN 30](#_Toc193487827)

[**5.1. Hiệu suất thuật toán 30**](#_Toc193487828)

[**5.1.1. Độ phức tạp thời gian 30**](#_Toc193487829)

[**5.1.2. Hạn chế 31**](#_Toc193487830)

[**5.2. Cải tiến tiềm năng 31**](#_Toc193487831)

[**5.2.1. Cải thiện hàm heuristic 31**](#_Toc193487832)

[**5.2.2. Sử dụng bộ nhớ hiệu quả hơn 32**](#_Toc193487833)

[**5.2.3. Song song hóa thuật toán 33**](#_Toc193487834)

[**5.2.4. Các cải tiến khác 34**](#_Toc193487835)

[KẾT LUẬN 36](#_Toc193487836)

# **LỜI NÓI ĐẦU**

Trong bối cảnh công nghệ thông tin và trí tuệ nhân tạo (AI) ngày càng phát triển, việc nghiên cứu và ứng dụng các thuật toán thông minh vào thực tiễn đã trở thành một xu hướng quan trọng. Học phần "Trí tuệ nhân tạo" không chỉ cung cấp nền tảng lý thuyết về các khái niệm AI mà còn tạo cơ hội để sinh viên áp dụng kiến thức vào các bài toán thực tế, từ đó nâng cao kỹ năng lập trình và tư duy giải quyết vấn đề.

Đề tài "Áp dụng thuật toán A\* để lập trình chương trình trò chơi sắp xếp vòng" được thực hiện nhằm khám phá một trong những thuật toán tìm kiếm đường đi tối ưu nổi tiếng trong lĩnh vực AI – thuật toán A\*. Thông qua việc áp dụng thuật toán này vào trò chơi sắp xếp vòng, một bài toán kinh điển mang tính logic và thách thức, đề tài không chỉ giúp làm rõ cách hoạt động của A\* mà còn thể hiện khả năng ứng dụng thực tiễn của nó trong việc giải quyết các vấn đề có không gian trạng thái lớn.

Lời nói đầu này xin gửi lời cảm ơn chân thành đến giảng viên hướng dẫn cùng các tài liệu tham khảo đã hỗ trợ trong quá trình thực hiện đề tài. Mặc dù đã có nhiều nỗ lực, bài làm không tránh khỏi những thiếu sót. Rất mong nhận được ý kiến đóng góp để đề tài được hoàn thiện hơn.

Trân trọng,

# **CHƯƠNG 1. GIỚI THIỆU**

## **1.1. Mô tả trò chơi**

Trò chơi sắp xếp vòng tròn là một logic trò chơi Yêu cầu người chơi phải sắp xếp các màu vòng trên các cột theo một trình tự nhất định. Trong trò chơi này, có tổng cộng 6 cột được sử dụng với các đặc điểm sau:

* **Trạng thái**: Một cấu hình của 6 cột, mỗi cột là một danh sách các vòng màu (có thể rỗng).
* **Trạng thái ban đầu**: 4 cột chứa 4 vòng màu mỗi cột (ví dụ: đỏ, xanh, vàng, tím) có thể lặp lại giữa các cột và không nhất thiết phải theo thứ tự đầu tiên, 2 cột trống đóng vai trò là các cột trung gian để hỗ trợ người chơi trong quá trình chuyển màu vòng lặp.

Game được quy định như sau:

* Chỉ các vòng ở vị trí trên cùng của cột mới được phép chuyển đổi.
* Khi chuyển các vòng từ cột này sang cột khác thì mỗi lần chuyển chỉ chuyển 1 vòng.
* Một vòng có thể được chuyển sang một cột trống hoặc sang một cột có cùng màu ở vị trí trên cùng.
* Mục tiêu của trò chơi là sắp xếp sao cho mỗi cột chỉ chứa các vòng cùng một màu duy nhất (hoặc để trống). Trò chơi kết thúc khi tất cả các cột đều đạt được trạng thái này.
* Người chơi sẽ thua nếu vượt quá số bước tối đa được phép mà chưa hoàn thành mục tiêu.

Với sự thay đổi này, trò chơi trở nên đơn giản hơn so với việc chuyển nhóm vòng cùng màu, nhưng vẫn giữ được các tính chất trong việc thiết lập kế hoạch để đạt được trạng thái mục tiêu. Trò chơi có thể được xem như một biến thể của Tháp toán Hà Nội, với lực lượng rõ ràng về màu sắc thay vì vòng kích thước.

## **1.2. Mục tiêu bài toán**

Mục tiêu chính của bài toán là thiết kế và phát triển một chương trình sử dụng thuật toán A\* để giải quyết vấn đề sắp xếp trò chơi một cách hiệu quả. Cụ thể, chương trình cần đáp ứng các yêu cầu sau:

* Tìm đường đi tối ưu : Sử dụng thuật toán A\* để xác định các bước chuỗi chuyển ngắn nhất từ ​​trạng thái ban đầu đến tiêu điểm trạng thái, trong đó mỗi cột chỉ chứa các màu tương tự.
* Hiển thị trạng thái : Cung cấp giao diện được phép cho người chơi theo dõi trạng thái hiện tại của các cột và màu vòng lặp sau mỗi bước chuyển đổi.
* Ghi nhận các bước : Lưu trữ và hiển thị danh sách các bước chuyển đã được thực hiện để người chơi có thể hiểu được quy trình giải quyết.
* Người dùng tương tác : Cho phép người chơi nhập lệnh thủ công (ví dụ: “chuyển từ cột A sang cột B”) để thực hiện các bước chuyển đổi, đồng thời tích hợp khả năng tự động giải toán bằng thuật toán A\*.

Bài toán không chỉ nhắm tới việc quyết định một logic trò chơi mà còn có thể triển khai ứng dụng thực tế của thuật toán tìm kiếm heuristic trong các công việc ưu tiên hóa các vấn đề không có trạng thái lớn.

# **CHƯƠNG 2. THUẬT TOÁN A\***

## **2.1. Nguyên lý thuật toán A\***

### **2.1.1. Thuật toán A là gì?\***

Thuật toán A\* là một thuật toán tìm đường (pathfinding) và tìm kiếm đồ thị (graph search) được sử dụng rộng rãi trong trí tuệ nhân tạo và lập trình game. Nó giúp tìm đường đi ngắn nhất từ một điểm xuất phát (start) đến một điểm đích (goal) trong một không gian có các nút (nodes) và các cạnh (edges) kết nối chúng.

A\* nổi bật vì nó kết hợp ưu điểm của:

* **Tìm kiếm theo chi phí thấp nhất (Uniform Cost Search)**: Đảm bảo đường đi ngắn nhất.
* **Tìm kiếm heuristic (Greedy Best-First Search)**:Tăng tốc độ bằng cách ưu tiên hướng về đích.

### **2.1.2. Nguyên lý cơ bản của A**\*

Thuật toán A\* hoạt động dựa trên việc kết hợp hai yếu tố chính:

1. Chi phí thực tế g(n) : Đây là chi phí từ điểm xuất phát đến một nút công cụ (n) trong không gian tìm kiếm. Chi phí này thường được tính toán dựa trên khoảng cách thực tế hoặc các yếu tố yếu tố khác tùy thuộc vào tiền toán.
2. Chi phí giá trị h(n) : Đây là hàm heuristic, giá trị giá trị từ nút đến đích. Hàm heuristic phải thỏa mãn tính chất tam giác (bất đẳng thức tam giác), tức là h(n) ≤ g(n) + g(m) + h(m) với mọi nút n và m, để đảm bảo tính tối ưu của thuật toán.

A\* sử dụng một hàm đánh giá được xác định rõ ràng như sau:

**f(n) = g(n) + h(n)**

Trong đó:

* f(n) : Chi phí tổng giá trị từ điểm phát hiện nút đến đích.
* g(n) : Chi phí từ điểm xuất phát đến nút n.
* h(n) : Chi phí heuristic từ nút n đến đích.

Mục tiêu của A:\* Tìm đường đi sao cho f(n) nhỏ nhất, nghĩa là chi phí tổng cộng từ xuất phát đến đích là tối ưu.

### **2.1.3. Các bước thực hiện**

* Khởi tạo : Bắt đầu từ phát nút xuất, đặt g(n) = 0 và tính f(n) = h(n).
* Duy trì hàng ưu tiên : Sử dụng một hàng ưu tiên (hàng đợi ưu tiên) để lưu các nút cần khám phá, sắp xếp theo giá trị f(n) tăng dần.
* Mở nút mở rộng : Lấy nút có f(n) nhỏ nhất từ ​​hàng được mong đợi, mở rộng các nút lân cận của nó.
* Cập nhật chi phí : Với mỗi nút lân cận, tính g(n) và f(n) mới. Nếu chi phí mới nhỏ hơn chi phí cũ, cập nhật lại đường đi.
* Kiểm tra đích : Lặp lại quá trình cho đến khi đạt được nút đích hoặc không còn nút nào để khám phá.
* Truy tìm đường đi : Khi đến đích, truy ngược lại từ đích về phát hiện để xây dựng đường đi tối ưu.

### **2.1.4. Ưu điểm và nhược điểm**

#### **Ưu điểm**:

* Tính tối ưu : A\* đảm bảo tìm được đường đi ngắn nhất nếu hàm heuristic h(n) không bao giờ đánh giá quá cao chi phí thực tế (tính chất được chấp nhận ) và mãn tính tính chất tam giác.
* Hiệu quả : A\* hiệu quả hơn các thuật toán như Dijkstra vì nó sử dụng heuristic để hướng dẫn tìm kiếm, giảm số lượng nút cần khám phá.

#### **Nhược điểm**:

* Bộ nhớ : Lưu trữ nhiều nút trong hàng ưu tiên, không phù hợp với không gian lớn.
* Phụ thuộc heuristic : Hiệu quả giảm nếu heuristic gần gũi, Đòi hỏi thiết kế kỹ cẩn thận.
* Dòng trong trường hợp lý xấu : Độ phức tạp của thời gian cao nếu không tìm kiếm lớn hoặc heuristic không tốt.
* Không hoạt động : Không phù hợp với môi trường thay đổi

### **2.1.5. Ứng dụng thực tế**

* Tìm đường trong game (như nhân vật di chuyển).
* Định tuyến GPS.
* Robot điều hướng trong không gian.

## **2.2. Hàm heuristic đề xuất**

Thuật toán A\* là một thuật toán tìm kiếm đường đi tối ưu, được phát triển dựa trên sự hợp lý giữa tìm kiếm theo chi phí (như thuật toán Dijkstra) và tìm kiếm heuristic (như Greedy Best-First Search). A\* sử dụng một hàm đánh giá để ưu tiên các trạng thái tiềm năng, giúp giảm thiểu số lượng trạng thái cần xem xét trong quá trình tìm kiếm.

Nguyên lý cốt lõi của A\* dựa trên hàm đánh giá tổng hợp: f(n) = g(n) + h(n) trong đó:

* g(n ): Chi phí thực tế từ trạng thái ban đầu đến trạng thái hiện tại N, thường được tính bằng cách thực hiện chuyển đổi số bước.
* h(n): Hàm heuristic, ước lượng chi phí tối thiểu từ trạng thái N to focus status. Hàm này phải đồng mãn tính chất tam giác, tức h(n) ≤ h(n′) + c(n ,N′), với c(n,N′) là chi phí từ N đến N′, để đảm bảo tính ưu việt của thuật toán.
* nếu f(n ): Tổng chi phí dự kiến ​​từ trạng thái ban đầu qua N đến tiêu điểm trạng thái, là giá trị được sử dụng để sắp xếp các trạng thái trong mức độ ưu tiên hàng đợi.

Quy trình hoạt động của thuật toán A\* bao gồm các bước sau:

1. **Khởi tạo một hàng ưu tiên:** (Hàng đợi ưu tiên) chứa trạng thái ban đầu, với f(n) = h(n) vì g(n) = 0.
2. **Trong mỗi vòng lặp:** 
   * Lấy trạng thái có f(n) hàng mong đợi nhỏ nhất.
   * Nếu trạng thái này là tiêu điểm trạng thái, hãy dừng và trả về đường đi.
   * Tạo tất cả các trạng thái con (các trạng thái có thể đạt được bằng một chuyển hợp lệ bước).
   * Tính g(n), h(n), và f(n) cho từng trạng thái, sau đó thêm chúng vào hàng dự kiến ​​nếu chưa được xem xét.
3. Nếu hàng trống hoặc vượt quá giới hạn tài nguyên (ví dụ: số bước tối đa), kết thúc thuật toán mà không thể tìm thấy giải pháp.

A\* đảm bảo tìm được đường đi tối ưu nếu hàm heuristic h(n) được chấp nhận (không bao giờ đánh giá quá cao chi phí thực tế) và nhất quán (thỏa mãn tính chất tam giác).

## **2.2. Hàm xuất đề heuristic**

Để áp dụng thuật toán A\* vào trò chơi sắp xếp vòng luật chơi mới (mỗi lần chuyển chỉ chuyển 1 vòng), hàm heuristic h(n) cần thiết kế sao cho mức độ chính xác cao nhất ở mức tối thiểu từ trạng thái hiện tại đến mục tiêu trạng thái. Trong trò chơi này, mục tiêu trạng thái là mỗi cột chỉ chứa các màu cùng vòng hoặc để trống.

Dưới đây là một hàm heuristic được xuất ra:

* Định nghĩa: h(n) được tính toán bằng cách sử dụng tổng số vòng chưa được định vị, tức là số vòng không chính xác cùng màu với màu chủ đạo của cột đích mà chúng cần đến.
* Cách tính :
  1. Với mỗi cột không trống, xác định màu chủ đạo (màu sử dụng số lượng lớn nhất trong cột hoặc màu của vòng lặp nếu cột được xây dựng).
  2. Đếm số vòng trong cột không khớp với màu chủ đạo này.
  3. Tổng hợp các vị trí vòng lặp không đúng trên tất cả các cột.
* Ví dụ :
  1. Giả sử hiện tại trạng thái:
     + Cột 1: [Đỏ, Xanh, Vàng, Đỏ] (màu chủ đạo là Đỏ vì có 2 Đỏ).
     + Vải 2: [Xanh, Đỏ, Xanh, Vàng] (màu chủ đạo là Xanh vì có 2 Xanh).
     + Cột 3: [Vàng, Vàng, Đỏ, Vàng] (màu chủ đạo là Vàng vì có 3 Vàng).
     + Cột 4: [Đỏ, Vàng, Xanh, Đỏ] (màu chủ đạo là Đỏ vì có 2 Đỏ).
     + Cột 5: [] (trống).
     + Cột 6: [] (trống).
  2. Tính h(n):
     + Cột 1: 2 vòng (Xanh, Vàng) không phải Đỏ 🡪 2.
     + Cột 2: 2 vòng (Đỏ, Vàng) không phải Xanh 🡪 2.
     + Cột 3: 1 vòng (Đỏ) không phải Vàng 🡪 1.
     + Cột 4: 2 vòng (Vàng, Xanh) không phải Đỏ 🡪 2.
     + Tổng: h(n) = 2 + 2 + 1 + 2 = 7
* Tính chất :
  1. Có thể chấp nhận được : Hàm này không bao giờ đánh giá quá cao chi phí thực tế, bởi vì mỗi vòng chưa đúng vị trí cần ít nhất một bước chuyển để đến cột đúng và theo luật mới (chỉ chuyển 1 vòng mỗi lần), số bước thực tế thường lớn hơn hoặc số vòng chưa đúng vị trí.
  2. Nhất quán : Với mỗi bước chuyển đổi, số vòng chưa đúng vị trí chỉ giảm tối đa 1 (khi một vòng được đặt đúng cột) hoặc không đổi (khi chuyển sang cột trung gian), mãn tính tính chất tam giác h(n) ≤ h( n′) + c( n ,N′), trong đó c(n,N′)=1(một bước di chuyển).
* Ưu điểm : Hàm heuristic này đơn giản, dễ tính toán và phù hợp với luật chơi mới, phản ánh mức độ xa của trạng thái hiện tại so với mục tiêu.
* Chế độ : Hàm có thể không có mức độ ưu tiên cao nhất trong việc phân tích các trạng thái có cùng một số vị trí vòng nhưng khác nhau về độ khó chuyển đổi (cần sử dụng cột trung gian). Để cải thiện, có thể bổ sung yếu tố như số cột chưa hoàn thiện hoặc khoảng cách trung bình của các vòng đến đích cột.

Hàm heuristic này sẽ được tích hợp vào thuật toán A\* để định hướng quá trình tìm kiếm, đảm bảo chương trình vừa tối ưu trong việc giải quyết trò chơi với luật sửa đổi trò chơi.

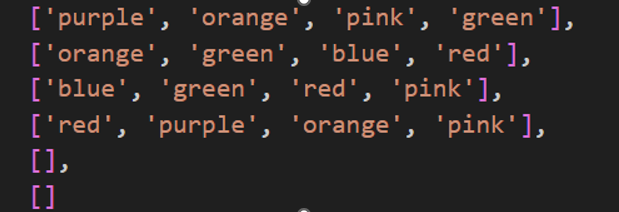
# **CHƯƠNG 3. TRIỂN KHAI TRÒ CHƠI**

## **3.1. Cấu trúc dữ liệu**

Phần này mô tả dữ liệu cấu hình được sử dụng trong chương trình, bao gồm cả cấu hình loại và cách thiết lập dữ liệu để biểu thị sắp xếp vòng trò chơi trong diễn đàn.

Danh sách (List) :

* Cấu trúc loại : Một danh sách chính chứa 6 danh sách con, mỗi danh sách hoạt động như một ngăn xếp (ngăn xếp) để lưu trữ các màu vòng.
* Cấu hình :
  + Trạng thái trò chơi bao gồm 6 cột, trong đó 4 cột cấm đầu chứa 4 vòng màu (ví dụ: 'tím', 'cam', 'hồng', 'xanh') và 2 cột trống.
  + Các bước được chuyển được lưu trong một danh sách riêng biệt dưới dạng các cặp (cột nguồn, cột đích).
* Ví dụ:



Hàng đợi ưu tiên (Hàng đợi ưu tiên) :

* Cấu hình loại : Một hàng dựa trên heap, sắp xếp các trạng thái chi phí f(n) = g(n) + h.
* Cấu hình : Mỗi phần tử là một bộ ba (mức độ ưu tiên, số bước, trạng thái), trong đó mức độ ưu tiên là giá trị ưu tiên, số bước là g(n) và trạng thái là cấu hình của 6 cột tại một thời điểm.
* Ví dụ:



Tập hợp (Set) :

* Cấu trúc loại : Một tập hợp lưu trữ các trạng thái đã xem xét để tránh lặp lại.
* Cấu hình : Trạng thái được chuyển thành chuỗi (chuỗi) từ cấu hình của 6 cột để đảm bảo tính duy nhất và có thể lưu trữ trong tập hợp.

## **3.2. Các bước thực hiện thuật toán**

Thuật toán A\* được phát triển để tìm đường đi tối ưu từ trạng thái ban đầu đến trạng thái mục tiêu (mỗi cột chứa một màu duy nhất). Dưới đây là các bước thực hiện:

1. **Khởi tạo** :
   * Tạo hàng đợi ưu tiên với đầu trạng thái, phân bổ g(n) = 0(số bước) và tính h(n) bằng hàm heuristic.
   * Tạo rỗng tập hợp.
2. **Vòng lặp chính** :
   * Lấy trạng thái có chi phí thấp nhất từ ​​​​hàng ưu tiên được mong đợi.
   * Kiểm tra chiến thắng trong điều kiện: Có đúng 4 cột đầy đủ (mỗi cột 4 vòng), mỗi cột chỉ chứa một màu duy nhất và tổng số vòng là 16. Nếu thắng, hãy trả về danh sách các bước chuyển tiếp.
3. **Kiểm tra lặp lại** :

Chuyển trạng thái thành chuỗi và kiểm tra bản xem đã có trong trình duyệt chưa hợp nhất. Nếu có, hãy bỏ qua.

1. **Tạo trạng thái con** :

Duyệt qua tất cả các nguồn và đích cột của các cặp (0 đến 5):

* Kiểm tra tính hợp lệ của việc chuyển đổi:
* Cột nguồn không rỗng.
* Điểm đích không vượt quá 4 vòng.
* Điểm trống hoặc vòng trên cùng của nguồn khớp với vòng trên cùng của điểm đến.
* Nếu hợp lệ:
* Tạo bản trạng thái mới.
* Di chuyển vòng qua cùng một nguồn cột sang đích cột.
* Thêm bước chuyển vào danh sách.
* Tínhg g(n′) = g(n) + 1 và h(n′) hàm heuristic.
* Thêm trạng thái vào hàng được chờ đợi với chi phí f(n′) = g(n′) + h(n′)

1. **Kết thúc** :

Nếu hàng đợi trống, thông báo không thể tìm được giải pháp.

Hàm heuristic: h được tính bằng cách:

* Tăng điểm cho mỗi cột có nhiều màu khác nhau (số màu - 1).
* Giảm điểm (-10) cho mỗi cột đầy đủ (4 vòng) và chỉ chứa một màu.

## **3.3. Hiển thị trạng thái trò chơi**

Chương trình sử dụng Pygame giao diện đồ họa để hiển thị trạng thái trò chơi một cách trực quan và hỗ trợ người dùng tương tác. Các thành phần hiển thị bao gồm:

* Vẽ cột và vòng :
  + Cột :
    - Hình chữ cập nhật cao 200px,rộng 20px, cách nhau 120px, màu xám (xám) nếu không chọn, màu vàng (vàng) nếu đang chọn.
  + Vòng :
    - Hình chữ nhật rộng 80px, cao 40px, xếp chồng từ dưới lên trên cột, màu sắc tương ứng (tím, cam, hồng, xanh lá cây, vv).
* Hiển thị số cột : Số thứ tự (0-5) được hiển thị bên dưới mỗi cột bằng chữ màu đen.
* Hiển thị số bước chuyển đổi : Văn bản "Moves: <số bước>/50" ở góc trên bên trái.
* Hiển thị hướng dẫn : Văn bản "Điều khiển: 0-5 để chọn, H để gợi ý, R để đặt lại, Q để thoát" ở góc dưới bên trái.
* Tương tác người dùng :
  + Chọn cột : Nhấn phím 0-5 để chọn nguồn cột, sau đó chọn mục tiêu cột.
    - Nếu nguồn trống hoặc chuyển hướng không hợp lệ, hãy hiển thị thông báo lỗi.
    - Nếu hợp lệ, vòng cuối cùng sẽ được chuyển và cập nhật trạng thái.
  + Gợi ý (Gợi ý) : Phím 'H' hiển thị bước chuyển tối ưu theo dựa trên heuristic.
  + Reset : Key 'R' khởi động lại trò chơi.
  + Thoát : Key 'Q' kết thúc chương trình.
  + Thông báo (gợi ý, lỗi) được hiển thị ở vị trí cố định trên màn hình.
* Kết quả trò chơi :
  + Nếu vượt qua 50 bước: Hiển thị "Bạn thua!" giữa màn hình trong 2 giây, sau đó đặt lại.
  + Nếu thắng: Hiển thị "You Win!" màu xanh trong 3 giây, sau đó bắt đầu màn hình mới.

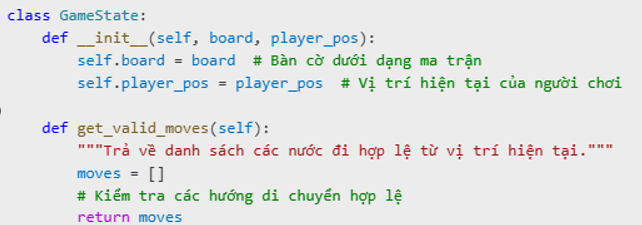
# **CHƯƠNG 4. HƯỚNG DẪN LẬP TRÌNH**

## **4.1. Cấu trúc chương trình**

### **4.1.1. Class GameState: Đại diện trạng thái trò chơi.**

Lớp này lưu trữ trạng thái của trò chơi, bao gồm thông tin về bàn cờ và vị trí hiện tại của người chơi. Nó cũng cung cấp các phương thức để xác định các nước đi hợp lệ.

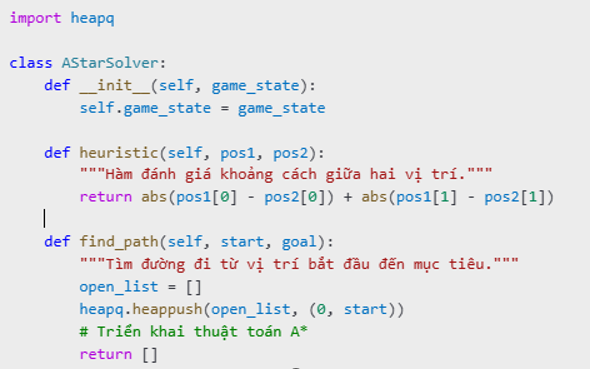
Ví dụ:



### **4.1.2. Class AStarSolver: Cài đặt thuật toán A\* để tìm giải pháp tối ưu.**

Lớp này thực hiện thuật toán A\* để tìm đường đi tối ưu từ vị trí hiện tại đến mục tiêu.

Ví dụ:



### **4.1.3. Hàm nhập lệnh: Cho phép người chơi nhập lệnh di chuyển vòng.**

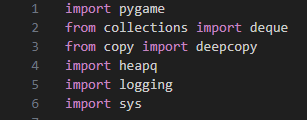
Hàm này cho phép người chơi nhập lệnh di chuyển (ví dụ: lên, xuống, trái, phải) và cập nhật trạng thái trò chơi tương ứng.

Ví dụ:



## **4.2. Gợi ý triển khai Python**

1. **Import các thư viện cần thiết**



**Giải thích:**

* + pygame: Thư viện để tạo giao diện đồ họa và xử lý sự kiện trò chơi.
  + deque từ collections: Dùng để quản lý các cấu trúc dữ liệu dạng hàng đợi (không sử dụng trực tiếp trong code này, có thể là dư thừa).
  + deepcopy từ copy: Tạo bản sao sâu của trạng thái trò chơi để không làm thay đổi bản gốc khi thử nghiệm các nước đi.
  + heapq: Thư viện để triển khai hàng đợi ưu tiên (priority queue), dùng trong thuật toán tìm đường đi tối ưu.
  + logging: Ghi lại thông tin debug và lỗi để theo dõi hoạt động của chương trình.
  + sys: Dùng để thoát chương trình khi có lỗi.

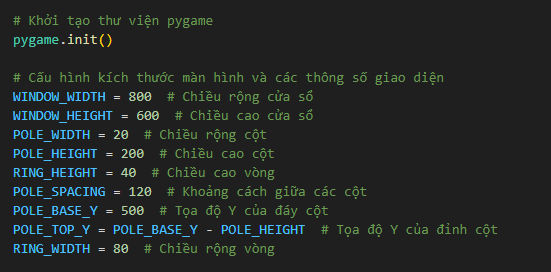
1. **Thiết lập logging**



**Giải thích:**

* Cấu hình logging ở mức DEBUG, hiển thị thời gian, mức độ (DEBUG, ERROR, v.v.) và thông điệp trong mỗi log.
* Hữu ích để theo dõi chi tiết hoạt động của chương trình trong quá trình phát triển.

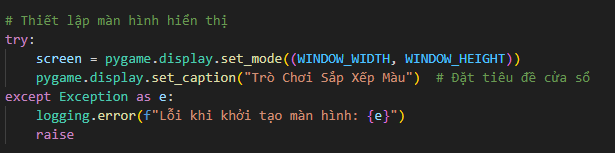
1. **Khởi tạo Pygame và các hằng số**

****

**Giải thích:**

* pygame là một thư viện Python phổ biến dùng để phát triển trò chơi 2D hoặc các ứng dụng đồ họa.
* pygame.init() là hàm khởi tạo tất cả các module cần thiết của Pygame (như hiển thị, âm thanh, sự kiện, v.v.) để bạn có thể sử dụng chúng trong chương trình.
* WINDOW\_WIDTH và WINDOW\_HEIGHT là hai hằng số được định nghĩa để xác định kích thước của cửa sổ giao diện trò chơi.
* WINDOW\_WIDTH = 800: Chiều rộng của cửa sổ là 800 pixel.
* WINDOW\_HEIGHT = 600: Chiều cao của cửa sổ là 600 pixel.
* POLE\_WIDTH = 20: Chiều rộng của một "cột" (pole) là 20 pixel.
* POLE\_HEIGHT = 260: Chiều cao của cột là 260 pixel.
* POLE\_TOP\_Y = 500: Đây là tọa độ Y (theo trục dọc) của đỉnh cột.
* POLE\_BASE\_Y là tọa độ Y của đế (phần dưới cùng) của cột.
* RING\_WIDTH = 80: Chiều rộng của một "vòng" (ring) là 80 pixel.

1. **Thiết lập màn hình và font**

****

**Giải thích:**

**** pygame.display.set\_mode() là hàm trong Pygame dùng để tạo một cửa sổ hiển thị (window) cho trò chơi hoặc ứng dụng đồ họa.

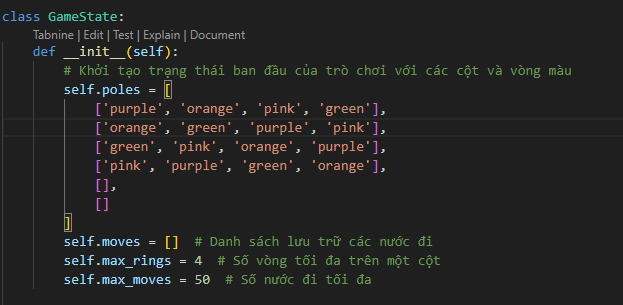
 WINDOW\_WIDTH và WINDOW\_HEIGHT là hai hằng số đã được định nghĩa trước đó (ở đoạn code trước bạn cung cấp):

* WINDOW\_WIDTH = 800: Chiều rộng cửa sổ là 800 pixel.
* WINDOW\_HEIGHT = 600: Chiều cao cửa sổ là 600 pixel.

 (WINDOW\_WIDTH, WINDOW\_HEIGHT) là một tuple (800, 600) đại diện cho kích thước của cửa sổ.

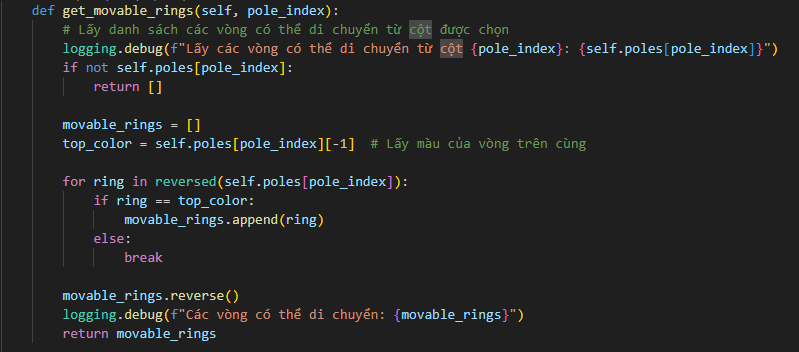
 screen là biến lưu trữ bề mặt của cửa sổ, được sử dụng để vẽ các đối tượng đồ họa lên màn hình.

1. **Lớp GameState**

****

**Giải thích:**

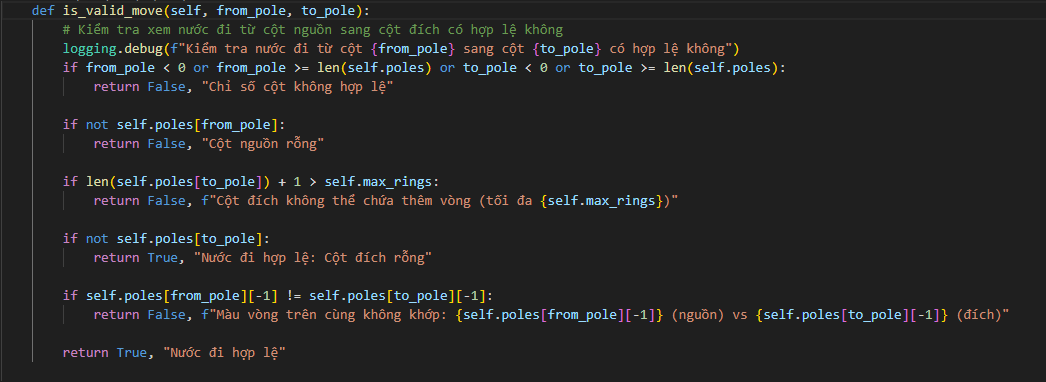
* Lớp GameState đại diện cho trạng thái trò chơi.
* self.poles: Danh sách 6 cột, mỗi cột chứa các vòng màu ban đầu (4 cột đầy, 2 cột rỗng).
* self.moves: Lưu trữ lịch sử các nước đi.
* self.max\_rings: Số vòng tối đa trên mỗi cột (4).
* self.max\_moves: Số nước đi tối đa cho phép (50).



1. **Kiểm tra nước đi từ cột nguồn sang cột địch**

**Giải thích:**

* Xác định các vòng có thể di chuyển từ một cột (chỉ những vòng cùng màu ở trên cùng).
* Trả về danh sách các vòng có thể di chuyển.



**Giải thích:**

**** self: Tham chiếu đến đối tượng hiện tại của lớp (instance of the class).

 from\_pole: Cột nguồn (source pole) – cột mà bạn muốn lấy vòng ra.

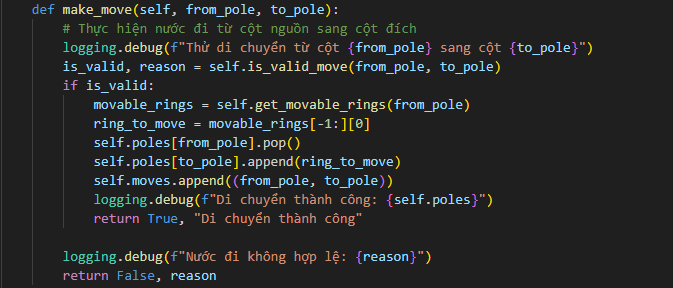
 to\_pole: Cột đích (destination pole) – cột mà bạn muốn đặt vòng vào.

 logging.debug(): Ghi lại thông tin debug vào log (có thể là file log hoặc console, tùy thuộc vào cách bạn cấu hình logging).

 self.poles: Đây là một danh sách chứa các cột trong trò chơi. Mỗi cột có thể là một danh sách con chứa các được xếp trên cột đó.

 len(self.poles): Số lượng cột trong trò chơi

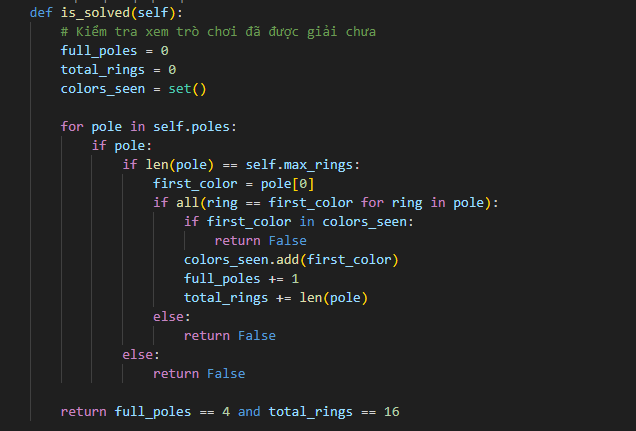
1. **Thực hiện nước đi từ cột nguồn sang cột đích**



**Giải thích:**

* Thực hiện nước đi nếu hợp lệ, di chuyển vòng từ cột nguồn sang cột đích và ghi lại nước đi.

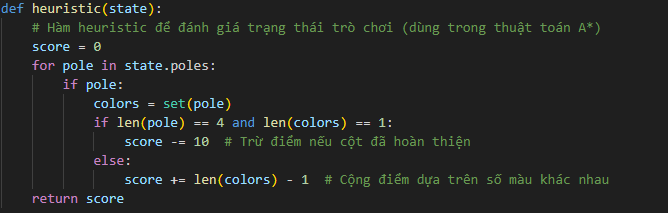
1. **Kiểm tra**



**Giải thích:**

* Kiểm tra xem trò chơi đã hoàn thành chưa: 4 cột đầy (mỗi cột 4 vòng cùng màu) và tổng cộng 16 vòng.

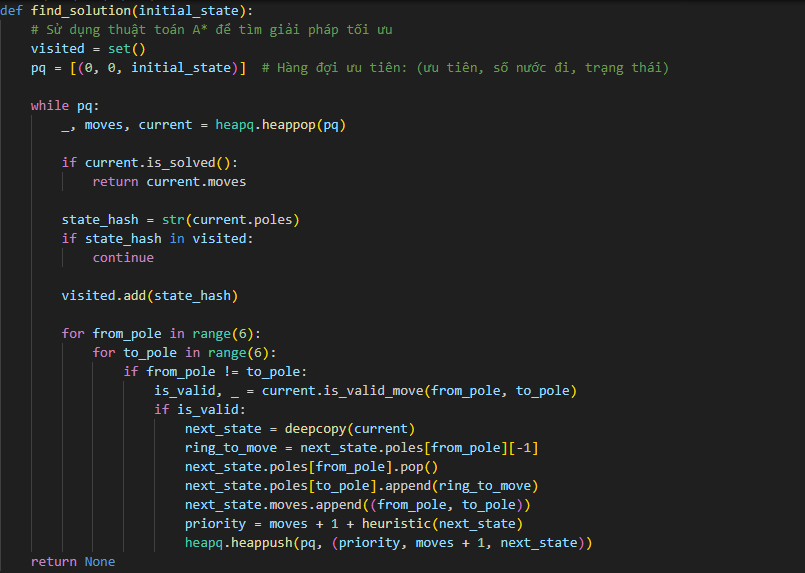
1. **Hàm heuristic**

****

**Giải thích:**

* Hàm heuristic đánh giá trạng thái trò chơi để hỗ trợ thuật toán tìm đường đi tối ưu.
* Giảm điểm nếu cột đầy và cùng màu, tăng điểm nếu cột có nhiều màu khác nhau.

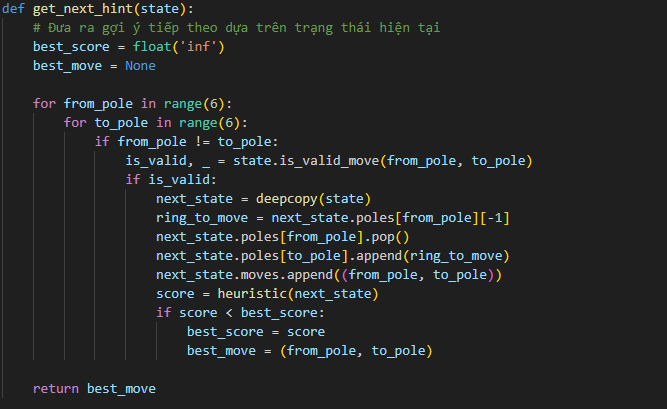
1. **Hàm find\_solution**

****

**Giải thích:**

* Sử dụng thuật toán A\* với hàng đợi ưu tiên để tìm giải pháp tối ưu.
* Trả về danh sách các nước đi nếu tìm thấy giải pháp, hoặc None nếu không.

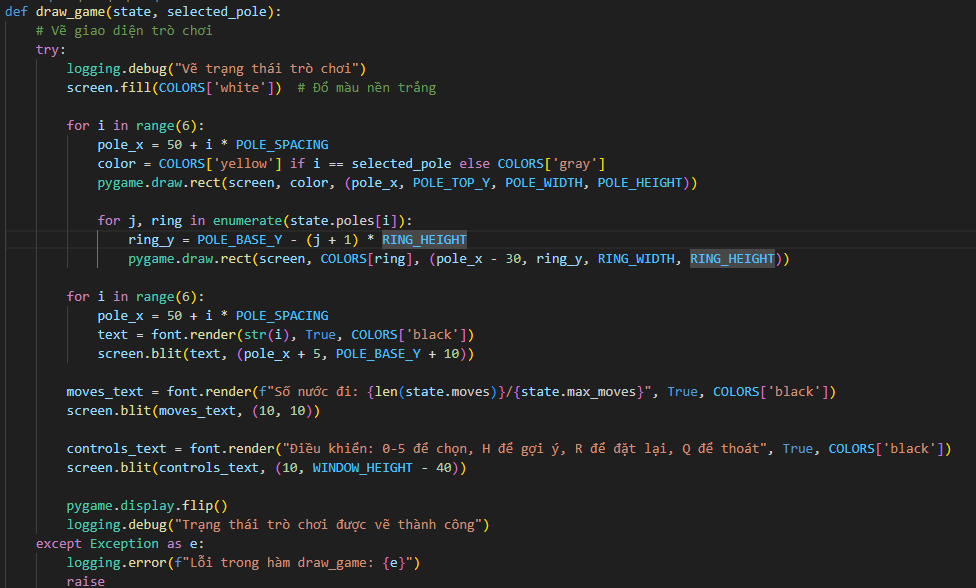
1. **Hàm get\_next\_hint**

****

**Giải thích:**

* Tìm nước đi tiếp theo tốt nhất dựa trên heuristic, trả về một tuple (from\_pole, to\_pole).

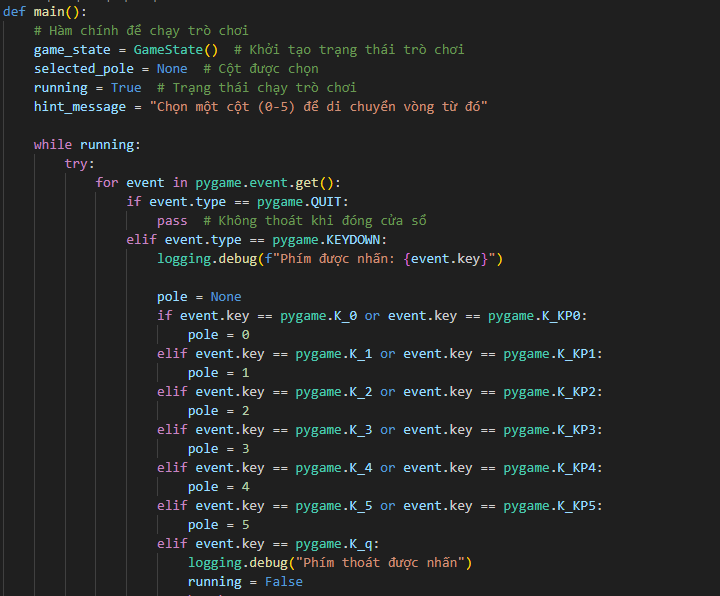
1. **Hàm draw\_game**

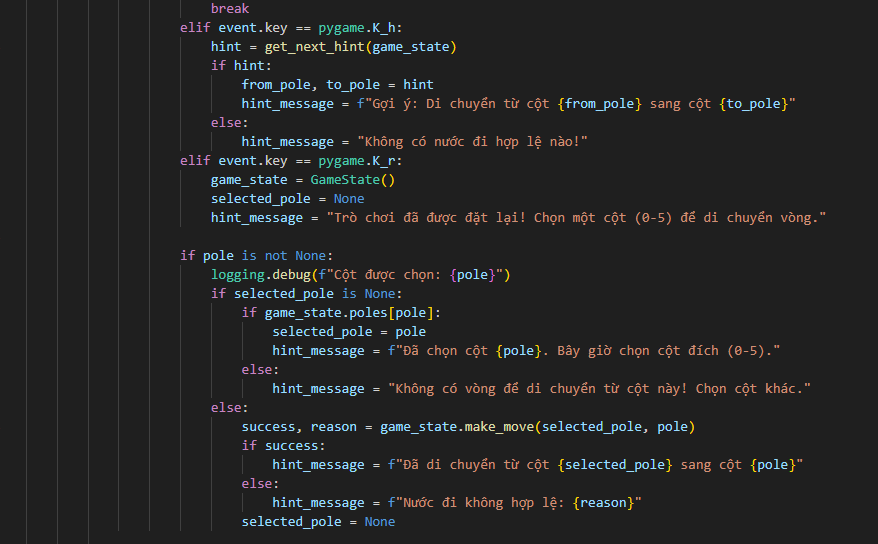
****

**Giải thích:**

* Vẽ trạng thái trò chơi lên màn hình: cột (màu vàng nếu được chọn), vòng, số cột, số nước đi, và hướng dẫn điều khiển.

1. **Hàm main**

****

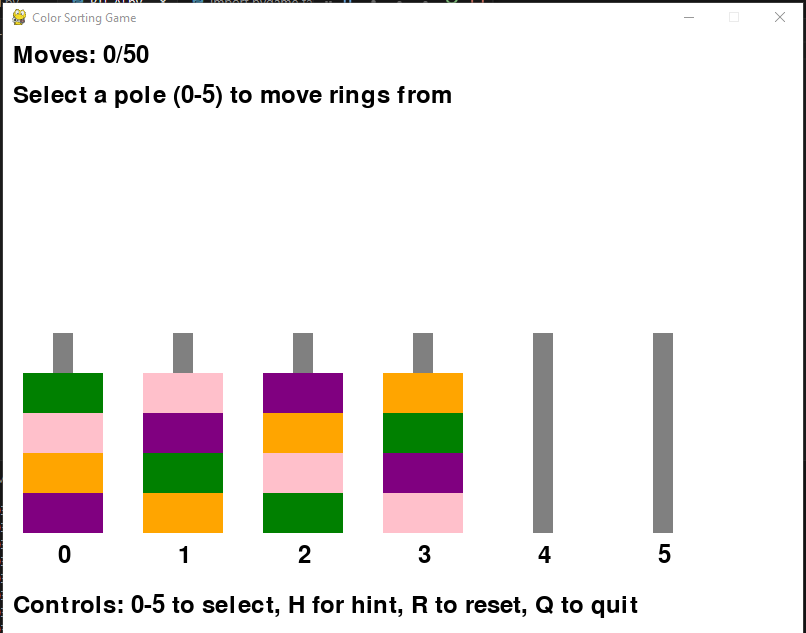
****

****

**Giải thích:**

* Vòng lặp chính của trò chơi:
  + Xử lý sự kiện phím (chọn cột, gợi ý, reset, thoát).
  + Vẽ giao diện.
  + Kiểm tra thua (vượt quá số nước đi) hoặc thắng (giải quyết trò chơi).

1. **Chạy chương trình**

****

Chương trình sau khi hoàn thiện

# **CHƯƠNG 5. ĐÁNH GIÁ VÀ CẢI TIẾN**

## **5.1. Hiệu suất thuật toán**

### **5.1.1. Độ phức tạp thời gian**

Thuật toán tìm kiếm giải pháp trong trò chơi "Color Sorting Game" được triển khai trong hàm find\_solution sử dụng thuật toán A\* (A-star) với hàng đợi ưu tiên (priority queue) dựa trên thư viện heapq. Độ phức tạp thời gian của thuật toán này phụ thuộc vào số trạng thái có thể có của trò chơi và cách hàm heuristic (heuristic) được thiết kế.

* **Phân tích độ phức tạp**:
  + Trong trường hợp xấu nhất, thuật toán A\* phải duyệt qua toàn bộ không gian trạng thái để tìm giải pháp. Với trò chơi này, không gian trạng thái được xác định bởi số cột (6 cột) và số vòng (16 vòng, mỗi cột tối đa 4 vòng).
  + Số trạng thái có thể có là rất lớn, vì mỗi vòng có thể nằm trên bất kỳ cột nào, và thứ tự của các vòng trên mỗi cột cũng quan trọng. Gần đúng, số trạng thái có thể lên đến 616 6^{16} 616 (mỗi vòng có 6 lựa chọn cột), nhưng do các ràng buộc của trò chơi (mỗi cột tối đa 4 vòng, các vòng trên cùng cột phải cùng màu để hợp lệ), con số thực tế nhỏ hơn.
  + Thuật toán A\* sử dụng hàng đợi ưu tiên, với mỗi thao tác thêm hoặc lấy phần tử từ hàng đợi có độ phức tạp O(logn), trong đó n là số trạng thái trong hàng đợi.
  + Tổng độ phức tạp thời gian của thuật toán A\* trong trường hợp xấu nhất là O(bdlogbd), trong đó b là số nhánh trung bình (branching factor – số nước đi hợp lệ từ một trạng thái) và d là độ sâu của giải pháp tối ưu. Trong trò chơi này, b có thể lên đến 6×5=30 (mỗi cột có thể di chuyển đến 5 cột khác), và d phụ thuộc vào số bước tối ưu để giải quyết trò chơi.
  + Tuy nhiên, nhờ hàm heuristic, thuật toán không phải duyệt toàn bộ không gian trạng thái, nên độ phức tạp thực tế thường nhỏ hơn. Trong trường hợp trung bình, độ phức tạp thời gian có thể được xem là O(nlogn), với n là số trạng thái được duyệt.
* **Kết luận**: Độ phức tạp thời gian của thuật toán hiện tại là O(nlogn), trong đó n là số trạng thái được thuật toán duyệt qua. Tuy nhiên, khi số vòng hoặc số cột tăng, không gian trạng thái sẽ tăng theo cấp số nhân, dẫn đến hiệu suất giảm đáng kể.

### **5.1.2. Hạn chế**

Mặc dù thuật toán A\* được sử dụng trong trò chơi này có hiệu quả trong việc tìm giải pháp tối ưu, nhưng vẫn tồn tại một số hạn chế:

* **Không gian trạng thái lớn**: Với 16 vòng và 6 cột, số trạng thái có thể có là rất lớn, ngay cả khi có các ràng buộc (mỗi cột tối đa 4 vòng, các vòng trên cùng cột phải cùng màu). Khi số vòng hoặc số cột tăng, không gian trạng thái tăng theo cấp số nhân, khiến thuật toán chạy chậm hơn đáng kể.
* **Hàm heuristic chưa tối ưu:** Hàm heuristic hiện tại chỉ tính điểm dựa trên số màu khác nhau trên mỗi cột và ưu tiên các cột đã hoàn thành (có 4 vòng cùng màu). Tuy nhiên, hàm này không phản ánh đầy đủ độ khó của việc di chuyển các vòng để đạt được trạng thái mục tiêu, dẫn đến việc thuật toán có thể duyệt qua nhiều trạng thái không cần thiết.
* **Tiêu tốn bộ nhớ:** Thuật toán A\* lưu trữ tất cả các trạng thái đã duyệt trong tập visited và hàng đợi ưu tiên pq. Với không gian trạng thái lớn, lượng bộ nhớ cần thiết có thể vượt quá khả năng của hệ thống, đặc biệt trên các thiết bị có tài nguyên hạn chế.
* **Hiệu suất của hàm** deepcopy: Trong hàm find\_solution và get\_next\_hint, việc sử dụng deepcopy để tạo bản sao của trạng thái (GameState) gây ra chi phí tính toán đáng kể, đặc biệt khi số trạng thái được duyệt tăng lên.

## **5.2. Cải tiến tiềm năng**

### **5.2.1. Cải thiện hàm heuristic**

Hàm heuristic hiện tại chỉ tập trung vào số màu khác nhau trên mỗi cột và ưu tiên các cột đã hoàn thành. Tuy nhiên, để thuật toán A\* hoạt động hiệu quả hơn, hàm heuristic cần được cải thiện để phản ánh chính xác hơn chi phí thực tế để đạt được trạng thái mục tiêu.

* **Đề xuất cải tiến**:
  + **Tính đến vị trí của các vòng:** Thay vì chỉ đếm số màu khác nhau, hàm heuristic có thể tính điểm dựa trên vị trí của các vòng so với trạng thái mục tiêu. Ví dụ, nếu một cột cần chứa 4 vòng màu "purple", nhưng hiện tại có 2 vòng "purple" ở cột khác, hàm heuristic có thể cộng thêm điểm phạt dựa trên khoảng cách (số bước cần thiết) để di chuyển các vòng "purple" đó đến cột mục tiêu.
  + **Ưu tiên các nước đi chiến lược:** Hàm heuristic có thể ưu tiên các nước đi giúp giải phóng các vòng bị chặn (ví dụ: một vòng màu "purple" bị chặn bởi các vòng màu khác). Điều này có thể được thực hiện bằng cách cộng điểm phạt cho các vòng bị chặn và giảm điểm phạt khi các vòng được giải phóng.
  + **Thử nghiệm nhiều hàm heuristic:** Có thể thử nghiệm các hàm heuristic khác, chẳng hạn như:
    - Hàm dựa trên số vòng cần di chuyển để đạt trạng thái mục tiêu.
    - Hàm dựa trên số cột đã gần hoàn thành (ví dụ: cột có 3/4 vòng cùng màu được ưu tiên hơn cột có 2/4 vòng cùng màu).
  + **Kết hợp nhiều yếu tố:** Kết hợp các yếu tố như số màu khác nhau, số vòng bị chặn, và số cột gần hoàn thành để tạo ra một hàm heuristic chính xác hơn.
* **Lợi ích:** Một hàm heuristic tốt hơn sẽ giúp thuật toán A\* tập trung vào các trạng thái có khả năng dẫn đến giải pháp nhanh hơn, giảm số trạng thái cần duyệt và cải thiện hiệu suất tổng thể.

### **5.2.2. Sử dụng bộ nhớ hiệu quả hơn**

Hiện tại, thuật toán A\* lưu trữ tất cả các trạng thái đã duyệt trong tập visited và hàng đợi ưu tiên pq, dẫn đến tiêu tốn bộ nhớ lớn khi không gian trạng thái tăng.

* **Đề xuất cải tiến**:
  + **Sử dụng bảng băm (hash table) để lưu trạng thái:** 
    - Thay vì lưu toàn bộ trạng thái GameState trong tập visited, có thể lưu trữ một giá trị băm (hash) của trạng thái. Điều này giảm lượng bộ nhớ cần thiết để lưu trữ các trạng thái đã duyệt.
    - Ví dụ: Chuyển đổi self.poles thành một chuỗi (string) hoặc tuple bất biến (immutable) và sử dụng hàm hash() của Python để tạo giá trị băm. Điều này đã được thực hiện trong code hiện tại state\_hash = str(current.poles), nhưng có thể tối ưu hơn bằng cách sử dụng các cấu trúc dữ liệu nhẹ hơn.
  + **Giảm số lượng bản sao trạng thái:** 
    - Hiện tại, hàm deepcopy được sử dụng để tạo bản sao của trạng thái trong find\_solution và get\_next\_hint. Thay vì sao chép toàn bộ trạng thái, có thể chỉ sao chép các phần cần thiết (ví dụ: chỉ sao chép self.poles thay vì toàn bộ đối tượng GameState).
    - Một cách khác là triển khai một cơ chế undo (hoàn tác) để tránh việc tạo bản sao. Ví dụ: Thay vì tạo bản sao, thực hiện nước đi trên trạng thái hiện tại, sau đó hoàn tác nước đi nếu không cần thiết.
  + **Giới hạn kích thước hàng đợi ưu tiên:** 
    - Có thể giới hạn số lượng trạng thái trong hàng đợi ưu tiên pq bằng cách loại bỏ các trạng thái có điểm số heuristic quá cao (ít có khả năng dẫn đến giải pháp). Điều này giúp giảm bộ nhớ sử dụng, nhưng cần cẩn thận để không bỏ sót giải pháp tối ưu.
* **Lợi ích**: Các cải tiến này sẽ giảm lượng bộ nhớ cần thiết, cho phép thuật toán xử lý các trò chơi với không gian trạng thái lớn hơn mà không gây tràn bộ nhớ.

### **5.2.3. Song song hóa thuật toán**

Thuật toán A\* hiện tại chạy tuần tự (sequential), nhưng có thể được song song hóa để tận dụng sức mạnh của các hệ thống đa lõi (multi-core).

* **Đề xuất cải tiến**:
  + **Chia nhỏ không gian trạng thái:** 
    - Chia không gian trạng thái thành các phần nhỏ hơn và xử lý đồng thời trên nhiều luồng (threads) hoặc tiến trình (processes). Ví dụ: Mỗi luồng có thể khám phá một tập hợp các nước đi ban đầu khác nhau (ví dụ: luồng 1 khám phá các nước đi từ cột 0, luồng 2 khám phá các nước đi từ cột 1, v.v.).
    - Sử dụng thư viện multiprocessing hoặc threading của Python để triển khai song song hóa.
  + **Song song hóa hàm heuristic:** 
    - Nếu hàm heuristic phức tạp và tốn thời gian tính toán, có thể song song hóa việc tính điểm heuristic cho các trạng thái khác nhau.
  + **Sử dụng thuật toán song song như Parallel A\*:** 
    - Thay vì sử dụng A\* tuần tự, có thể triển khai một phiên bản song song của A\* (Parallel A\*), trong đó nhiều luồng cùng khám phá không gian trạng thái và chia sẻ thông tin về các trạng thái đã duyệt.
* **Lợi ích**: Song song hóa sẽ tận dụng được sức mạnh của các CPU đa lõi, giảm thời gian chạy của thuật toán, đặc biệt khi không gian trạng thái lớn.

### **5.2.4. Các cải tiến khác**

Ngoài các cải tiến chính trên, có thể xem xét một số hướng khác để nâng cao hiệu suất và trải nghiệm người dùng:

* **Tối ưu hóa giao diện đồ họa**:
  + Hiện tại, hàm draw\_game vẽ lại toàn bộ màn hình trong mỗi khung hình, ngay cả khi không có thay đổi. Có thể tối ưu bằng cách chỉ vẽ lại các phần thay đổi (ví dụ: chỉ vẽ lại cột có vòng được di chuyển).
  + Sử dụng các kỹ thuật như double buffering hoặc sprite để cải thiện hiệu suất vẽ.
* **Cải thiện trải nghiệm người dùng**:
  + Thêm tính năng "undo" (hoàn tác) để người chơi có thể quay lại nước đi trước đó, thay vì phải reset toàn bộ trò chơi.
  + Cung cấp gợi ý chi tiết hơn trong get\_next\_hint, ví dụ: hiển thị một chuỗi các nước đi thay vì chỉ một nước đi tiếp theo.
* **Tối ưu hóa logic trò chơi**:
  + Trong phương thức make\_move, có thể cải thiện cách di chuyển nhiều vòng cùng màu cùng lúc (hiện tại chỉ di chuyển một vòng mỗi lần). Điều này sẽ giảm số nước đi cần thiết để giải quyết trò chơi.

# **KẾT LUẬN**

Đề tài "Áp dụng thuật toán A\* để lập trình chương trình trò chơi sắp xếp vòng" đã thành công trong việc thiết kế và triển khai một trò chơi logic sử dụng thuật toán tìm kiếm tối ưu A\*. Qua quá trình nghiên cứu và phát triển, các mục tiêu chính đã được hoàn thành:

* Ứng dụng thuật toán A\*: Thuật toán A\* được tích hợp hiệu quả để tìm đường đi tối ưu từ trạng thái ban đầu đến trạng thái mục tiêu, trong đó mỗi cột chỉ chứa các vòng cùng màu. Hàm heuristic được đề xuất, dựa trên số vòng chưa đúng vị trí hoặc số màu khác nhau trong cột, đã chứng minh tính khả thi trong việc định hướng tìm kiếm.
* Triển khai chương trình: Chương trình sử dụng Pygame cung cấp giao diện đồ họa trực quan, cho phép người chơi tương tác qua các lệnh chọn cột, và theo dõi trạng thái trò chơi. Các cấu trúc dữ liệu như danh sách, hàng đợi ưu tiên và tập hợp được kết hợp với cấu hình cụ thể của trò chơi (6 cột, 4 vòng mỗi cột) để đảm bảo hiệu suất và tính chính xác.
* Kết quả đạt được: Chương trình không chỉ tự động giải bài toán mà còn hỗ trợ người chơi thông qua các tính năng như hiển thị trạng thái, ghi nhận bước di chuyển và cung cấp gợi ý, đáp ứng đầy đủ các yêu cầu đề ra.

Tuy nhiên, chương trình vẫn tồn tại một số hạn chế, chẳng hạn như hàm heuristic có thể chưa tối ưu trong các trường hợp phức tạp, dẫn đến thời gian tìm kiếm tăng khi không gian trạng thái lớn. Ngoài ra, chức năng tự động giải toàn bộ bài toán (hàm find\_solution) chưa được tích hợp hoàn toàn vào giao diện người dùng, mà chỉ dừng ở mức cung cấp gợi ý từng bước.

Trong tương lai, đề tài có thể được mở rộng bằng cách:

* Tối ưu hóa hàm heuristic để giảm chi phí tính toán.
* Tích hợp tính năng tự động hiển thị toàn bộ đường đi tối ưu trên giao diện đồ họa.
* Thêm các mức độ khó khác nhau (ví dụ: thay đổi số cột hoặc số vòng) để tăng tính thử thách và đa dạng.

Đề tài không chỉ chứng minh được tính ứng dụng của thuật toán A\* trong giải quyết bài toán logic mà còn mở ra tiềm năng phát triển các trò chơi giáo dục hoặc công cụ mô phỏng dựa trên trí tuệ nhân tạo.

# **THƯ MỤC THAM KHẢO**