**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC ĐẠI NAM**

Logo, company name

Description automatically generated

**BÀI TẬP LỚN**

**TÊN HỌC PHẦN: TRÍ RUỆ NHÂN TẠO**

**ĐỀ TÀI: GIẢI THUẬT A\* CHO TRÒ CHƠI XẾP VÒNG**

**Giáo viên hướng dẫn: T.S Trần Đăng Công**

**Sinh viên thực hiện:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Stt** | **Mã sv** | **Họ và tên** | **Lớp** |
| 1 | 1771020405 | Ngô Sỹ Kiệt | CNTT 17-08 |
| 2 | 1771020271 | Nguyễn Xuân Hiếu | CNTT 17-08 |
| 3 | 1771020097 | Lương Mạnh Chiến | CNTT 17-08 |
| 4 | 1771020701 | Đặng Ngọc Trường | CNTT 17-08 |

**Hà Nội, năm 2025**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC ĐẠI NAM**

Logo, company name

Description automatically generated

**BÀI TẬP LỚN**

**TÊN HỌC PHẦN: TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI: GIẢI THUẬT A\* CHO TRÒ CHƠI XẾP VÒNG**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| STT | Mã Sinh Viên | Họ và Tên | Ngày Sinh | Điểm | |
| Bằng Số | Bằng Chữ |
| 1 | 1771020405 | Ngô Sỹ Kiệt | 17/09/2005 |  |  |
| 2 | 1771020271 | Nguyễn Xuân Hiếu | 19/02/1005 |  |  |
| 3 | 1771020097 | Lương Mạnh Chiến | 28/09/2005 |  |  |
| 4 | 1771020701 | Đặng Ngọc Trường | 21/02/2005 |  |  |

### CÁN BỘ CHẤM THI 1 CÁN BỘ CHẤM THI 2

**Hà Nội, năm 2025**

**LỜI NÓI ĐẦU**

Trong thời đại công nghệ phát triển, các thuật toán tìm kiếm và tối ưu hóa đóng vai trò quan trọng trong nhiều lĩnh vực, đặc biệt là trong trò chơi và trí tuệ nhân tạo. Thuật toán A\* (A-star) là một trong những thuật toán tìm kiếm heuristic mạnh mẽ, được sử dụng rộng rãi để tìm đường đi tối ưu trong nhiều bài toán khác nhau.

Đề tài “Giải thuật A cho trò chơi xếp vòng”\* tập trung vào việc áp dụng thuật toán A\* để giải quyết bài toán sắp xếp vòng màu trên các cột, đảm bảo tìm ra lộ trình di chuyển hiệu quả nhất. Trò chơi yêu cầu người chơi sắp xếp các vòng sao cho mỗi cột chỉ chứa một màu duy nhất, tuân theo các quy tắc di chuyển nhất định. Việc sử dụng thuật toán A\* giúp tìm ra chuỗi bước đi ngắn nhất để hoàn thành trò chơi một cách tối ưu.

Báo cáo này sẽ trình bày chi tiết về cách triển khai thuật toán A\*, cách biểu diễn trạng thái trò chơi, cách đánh giá chi phí di chuyển, cùng với các thử nghiệm và đánh giá hiệu quả của giải thuật. Hy vọng đề tài sẽ mang lại cái nhìn rõ ràng về ứng dụng thực tế của A\* trong việc giải quyết các bài toán tối ưu hóa.

**MỤC LỤC**

***[CHƯƠNG 1. GIỚI THIỆU VÀ PHÂN TÍCH YÊU CẦU BÀI TOÁN: 5](#_Toc17277)***

*[1.1. Giới thiệu: 5](#_Toc28526)*

*[1.1.1. Đặt vấn đề và mục tiêu 5](#_Toc971)*

*[1.2. Mô tả bài toán: 6](#_Toc25942)*

*[1.2.1. Giới thiệu bài toán : 6](#_Toc28484)*

*[1.2.2. Yêu cầu tối ưu hóa: 8](#_Toc26234)*

*[1.2.3. Mô hình hóa bài toán: 8](#_Toc8138)*

***[CHƯƠNG 2. GIẢI THUẬT TOÁN A\* VÀ PHÂN TÍCH MÃ NGUỒN 12](#_Toc15663)***

*[2.1. Tổng quan về giải thuật toán A\* 12](#_Toc22129)*

*[2.1.1. Nguyên lý hoạt động của thuật toán A\* 12](#_Toc3590)*

*[2.1.2. Các thành phần chính của giải thuật A\* 14](#_Toc30328)*

*[2.2. Khởi tạo thuật toán A\* 15](#_Toc16123)*

*[2.3. Tổng quan về mã nguồn 17](#_Toc29593)*

*[2.3.1. Phân tích từng mã nguồn 17](#_Toc21782)*

*[2.3.2. Biểu diễn dữ liệu 24](#_Toc31259)*

*[2.3.3. Phân tích kết quả 24](#_Toc7390)*

*[2.3.4. Kết quả đạt được 26](#_Toc12017)*

***[CHƯƠNG 3. THỬ NGHIỆM THUẬT TOÁN VỚI CÁC BỘ 29](#_Toc24884)***

***[THAM SỐ KHÁC NHAU VÀ NHẬN XÉT THUẬT TOÁN 29](#_Toc2002)***

*[3.1. Mục tiêu của thử nghiệm 29](#_Toc27637)*

*[3.1.1. Các bộ tham số của thử nghiệm 29](#_Toc18182)*

*[3.1.2. Kết quả của thử nghiệm 30](#_Toc30602)*

*[3.1.3. Phân tích kết quả thử nghiệm và lựa chọn bộ tham số tốt nhất 31](#_Toc8596)*

*[3.2. Hướng phát triển của thuật toán 33](#_Toc4273)*

*[3.2.1. Nhận xét đặc điểm và hạn chế của thuật toán 33](#_Toc17819)*

*[3.2.2. Đề xuất ý tưởng cải tiến thuật toán. 36](#_Toc16602)*

*[DANH MỤC TÀI LIỆU THAM KHẢO 40](#_Toc4023)*

**CHƯƠNG 1. GIỚI THIỆU VÀ PHÂN TÍCH YÊU CẦU BÀI TOÁN:**

* 1. **Giới thiệu:**

***1.1.1. Đặt vấn đề và mục tiêu***

Trò chơi xếp vòng là một bài toán sắp xếp yêu cầu người chơi di chuyển các vòng màu trên các cột sao cho mỗi cột chỉ chứa một màu duy nhất. Trò chơi đặt ra thử thách khi người chơi phải tuân theo các quy tắc di chuyển cụ thể, đồng thời tối ưu hóa số bước đi để hoàn thành mục tiêu.

Bài toán này có thể được xem như một bài toán tìm đường trong không gian trạng thái, nơi mỗi trạng thái đại diện cho một cách sắp xếp vòng trên các cột. Việc tìm kiếm chuỗi di chuyển ngắn nhất để đạt được trạng thái mục tiêu là một vấn đề quan trọng và có thể được giải quyết bằng thuật toán A\* – một thuật toán tìm kiếm heuristic mạnh mẽ trong trí tuệ nhân tạo.

Mục tiêu của đề tài là xây dựng một chương trình có thể tự động tìm ra lời giải tối ưu cho trò chơi xếp vòng bằng cách áp dụng thuật toán A\*. Cụ thể:

* Biểu diễn trạng thái trò chơi dưới dạng cấu trúc dữ liệu phù hợp.
* Xây dựng hàm heuristic để đánh giá chi phí di chuyển tối ưu.
* Tìm kiếm và hiển thị các bước di chuyển tối ưu để hoàn thành trò chơi.
* Cung cấp giao diện hiển thị trạng thái trò chơi và các bước thực hiện.

Đề tài tập trung vào việc áp dụng thuật toán A\* trong trò chơi xếp vòng, với các giả định sau:

* Trò chơi có 6 cột, trong đó 4 cột chứa vòng màu và 2 cột trống làm cột trung gian.
* Các vòng màu có thể di chuyển theo quy tắc cho trước.
* Chương trình sẽ tìm kiếm lời giải tối ưu với số bước di chuyển ít nhất.

Việc áp dụng thuật toán A\* vào trò chơi xếp vòng không chỉ giúp tìm ra cách giải hiệu quả mà còn giúp hiểu rõ hơn về ứng dụng của các thuật toán tìm kiếm heuristic trong thực tế. Bài toán này có liên quan đến nhiều vấn đề trong khoa học máy tính như tối ưu hóa đường đi, trí tuệ nhân tạo và thiết kế thuật toán.

* 1. **Mô tả bài toán:**
     1. ***Giới thiệu bài toán :***

Bài toán sắp xếp vòng màu là một dạng bài toán sắp xếp trong đó người chơi cần di chuyển các vòng từ cột này sang cột khác sao cho mỗi cột chỉ chứa các vòng cùng màu. Trò chơi có những ràng buộc về quy tắc di chuyển, khiến việc tìm kiếm giải pháp tối ưu trở thành một bài toán khó cần đến thuật toán tìm kiếm hiệu quả.

Trong chương trình này, chúng ta sử dụng **thuật toán A**\* để tìm ra chuỗi di chuyển tối ưu nhất từ trạng thái ban đầu đến trạng thái mục tiêu, giúp hoàn thành trò chơi với số bước di chuyển ít nhất.

- Cấu trúc bài toán:

##### **Thành phần chính của trò chơi**

* **Cột chứa vòng (peg)**: Có **6 cột**, trong đó:
  + **4 cột chứa vòng màu ban đầu** (mỗi cột chứa 4 vòng, các màu có thể lặp lại).
  + **2 cột trống** dùng để làm trung gian di chuyển.
* **Vòng màu (ring)**: Có nhiều màu sắc khác nhau như đỏ, xanh, vàng, tím, nâu...
* **Người chơi** có thể di chuyển các vòng theo quy tắc nhất định.

Quy tắc di chuyển vòng màu:

1. Chỉ được di chuyển các vòng cùng màu nằm trên cùng của một cột.
2. Phải di chuyển toàn bộ nhóm vòng cùng màu nếu có.
3. Có thể di chuyển vòng sang một cột trống hoặc một cột có cùng màu ở trên cùng.
4. Không thể đặt vòng màu lên cột có màu khác nhau ở trên cùng.

Ví dụ:

* Nếu một cột có trạng thái ['Xanh', 'Xanh', 'Đỏ'], người chơi có thể di chuyển hai vòng xanh cùng nhau.
* Nếu một cột khác có trạng thái ['Xanh', 'Đỏ', 'Vàng'], không thể đặt vòng xanh lên vì vòng trên cùng của cột đó là vàng.

Trạng thái của trò chơi được biểu diễn bằng danh sách các cột, trong đó mỗi cột là một danh sách chứa các vòng màu từ dưới lên trên.

Ví dụ về một trạng thái ban đầu:

A screenshot of a computer code

Description automatically generated

* 4 cột đầu chứa các vòng màu xáo trộn.
* 2 cột cuối trống để sử dụng trong quá trình sắp xếp.

Trạng thái đích là khi mỗi cột chứa duy nhất một màu, ví dụ:

goal\_state = [

["red", "red", "red", "red"],

["blue", "blue", "blue", "blue"],

["green", "green", "green", "green"],

["yellow", "yellow", "yellow", "yellow"],

[],

[]

]

* + 1. ***Yêu cầu tối ưu hóa:***

Chương trình hiện tại sử dụng thuật toán A\* để tìm kiếm lộ trình tối ưu cho việc sắp xếp các vòng màu. Tuy nhiên, để tăng hiệu suất và cải thiện trải nghiệm người dùng, cần thực hiện một số tối ưu hóa quan trọng. Trước tiên, cần cải tiến thuật toán A\* bằng cách tối ưu hóa việc kiểm tra trạng thái đã duyệt, sử dụng frozenset để tăng tốc truy xuất dữ liệu, cũng như cải tiến hàm heuristic để đánh giá trạng thái một cách hiệu quả hơn, giúp thuật toán hội tụ nhanh hơn. Ngoài ra, cần lưu trữ chi phí tốt nhất cho từng trạng thái để tránh kiểm tra lại các trạng thái không tối ưu, giúp giảm số lần mở rộng nút và cải thiện tốc độ tìm kiếm.

Bên cạnh đó, phần hiển thị giao diện trong Pygame cũng cần được tối ưu hóa. Thay vì vẽ lại toàn bộ màn hình sau mỗi bước di chuyển, chương trình nên chỉ cập nhật những phần có thay đổi nhằm giảm tải cho GPU và tăng tốc độ phản hồi. Ngoài ra, việc nhập lệnh từ người chơi cũng cần được cải tiến bằng cách sử dụng phím tắt thay vì yêu cầu nhập dữ liệu từ bàn phím qua terminal, giúp trò chơi diễn ra mượt mà hơn. Cuối cùng, quản lý bộ nhớ hiệu quả bằng cách sử dụng cấu trúc dữ liệu phù hợp sẽ giúp giảm tiêu thụ tài nguyên và cải thiện hiệu suất tổng thể của chương trình.

***1.2.3. Mô hình hóa bài toán:***

Bài toán sắp xếp vòng màu có thể được mô hình hóa như một bài toán tìm đường trong không gian trạng thái. Mục tiêu là di chuyển các vòng màu giữa các cột sao cho mỗi cột chỉ chứa các vòng cùng màu. Quá trình giải quyết bài toán này có thể được tiếp cận bằng thuật toán tìm kiếm tối ưu, trong đó A\* (A-star) là một lựa chọn phù hợp để tìm lời giải hiệu quả nhất với số bước di chuyển ít nhất.

- Các thành phần chính của mô hình:

**+** Trạng thái của bài toán (State Representation):

* Trạng thái của trò chơi được biểu diễn dưới dạng danh sách các cột (state), trong đó mỗi cột chứa một danh sách vòng màu.
* Ví dụ trạng thái ban đầu:

A screen shot of a computer code

Description automatically generated

+ Trạng thái mục tiêu (Goal State): Mỗi cột chỉ chứa một màu duy nhất hoặc rỗng.

**Không gian trạng thái (State Space):**

* Mỗi trạng thái được định nghĩa bởi cách sắp xếp của các vòng màu trên các cột.
* Không gian trạng thái gồm tất cả các cấu hình có thể có của vòng màu khi di chuyển giữa các cột.
* Các trạng thái hợp lệ là những trạng thái tuân thủ luật chơi (chỉ di chuyển các vòng cùng màu ở trên cùng, có thể di chuyển vào cột trống hoặc vào cột có màu tương ứng).

**Hàm chuyển trạng thái (Transition Function):**

* Một trạng thái mới được tạo ra bằng cách lấy nhóm vòng màu trên cùng từ một cột và di chuyển chúng sang cột khác nếu nước đi hợp lệ.

Điều này được thực hiện trong hàm apply\_move(state, move) :

A computer screen shot of a code

Description automatically generated

Chương trình tạo ra một trạng thái mới sau mỗi lần di chuyển hợp lệ.

**Hàm kiểm tra trạng thái đích (Goal Test Function):**

* Một trạng thái được coi là đích nếu mỗi cột chỉ chứa vòng cùng màu hoặc rỗng.
* Hàm kiểm tra mục tiêu được định nghĩa như sau:

A black background with white text

Description automatically generated

**Hàm đánh giá chi phí (Cost Function):**

* Chi phí của một trạng thái là số bước đã thực hiện để đạt đến trạng thái đó.
* Trong A\*, chi phí này giúp tìm ra lộ trình ít bước nhất để giải bài toán.

**Hàm heuristic (Heuristic Function):**

* Heuristic là hàm đánh giá trạng thái hiện tại dựa trên số lượng vòng màu chưa được sắp xếp đúng.
* Hàm heuristic trong chương trình được định nghĩa như sau

A screen shot of a computer program

Description automatically generated

- Heuristic càng nhỏ thì trạng thái càng gần trạng thái đích

**CHƯƠNG 2. GIẢI THUẬT TOÁN A\* VÀ PHÂN TÍCH MÃ NGUỒN**

**2.1. Tổng quan về giải thuật toán A\***

***2.1.1. Nguyên lý hoạt động của thuật toán A\****

Thuật toán A\* là một thuật toán tìm kiếm trên đồ thị nhằm tìm đường đi tối ưu từ trạng thái ban đầu đến trạng thái đích với chi phí nhỏ nhất. A\* kết hợp giữa tìm kiếm theo độ sâu (DFS) và tìm kiếm theo độ rộng (BFS) bằng cách sử dụng một hàm đánh giá chi phí để ưu tiên mở rộng những trạng thái tiềm năng nhất.

### ****1. Công thức đánh giá trạng thái trong A****\*

Mỗi trạng thái nnn trong thuật toán A\* được đánh giá bởi một hàm chi phí:

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

Trong đó:

* **g(n)g(n)g(n)**: Chi phí thực tế từ trạng thái ban đầu đến trạng thái hiện tại nnn.
* **h(n)h(n)h(n)**: Hàm heuristic ước lượng chi phí từ trạng thái hiện tại nnn đến trạng thái đích.
* **f(n)f(n)f(n)**: Tổng chi phí ước lượng để đi từ trạng thái ban đầu đến trạng thái đích thông qua trạng thái nnn.

### ****2. Hoạt động của thuật toán A****\*

Thuật toán A\* sử dụng một hàng đợi ưu tiên (priority queue) để quản lý các trạng thái cần mở rộng. Các bước chính của thuật toán:

1. **Khởi tạo**
   * Đưa trạng thái ban đầu vào hàng đợi ưu tiên với giá trị f(n)f(n)f(n).
   * Tạo danh sách **đã duyệt** để lưu các trạng thái đã mở rộng.
2. **Duyệt trạng thái**
   * Lấy trạng thái có giá trị f(n)f(n)f(n) nhỏ nhất từ hàng đợi ưu tiên.
   * Nếu trạng thái này là trạng thái đích → Kết thúc thuật toán.
   * Nếu chưa phải đích, tạo các trạng thái con bằng cách áp dụng các phép di chuyển hợp lệ.
3. **Tính toán chi phí và cập nhật hàng đợi**
   * Với mỗi trạng thái con, tính giá trị f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n).
   * Nếu trạng thái con chưa được duyệt hoặc có chi phí thấp hơn trước đó, ta thêm nó vào hàng đợi ưu tiên.
4. **Tiếp tục lặp lại quá trình cho đến khi tìm thấy lời giải hoặc hàng đợi trống**
   * Nếu hàng đợi trống mà chưa tìm thấy lời giải, kết luận rằng bài toán không có lời giải.

### Điều kiện đảm bảo thuật toán A tìm được đường đi tối ưu\*

A\* tìm được đường đi tối ưu nếu:

* Hàm heuristic **h(n)** không đánh giá quá thấp (tức là **không bị đánh giá nhỏ hơn chi phí thực tế**).
* Hàm heuristic đảm bảo **tính chấp nhận được**: không vượt quá chi phí tối ưu thực tế.

### Ứng dụng thuật toán A vào bài toán sắp xếp vòng màu\*

Trong bài toán sắp xếp vòng màu:

* **Trạng thái**: Mỗi trạng thái là một danh sách chứa các cột với các vòng màu.
* **Trạng thái đích**: Khi mỗi cột chỉ chứa một màu hoặc trống.
* **Hàm chi phí g(n)g(n)g(n)**: Số bước di chuyển đã thực hiện từ trạng thái ban đầu.
* **Hàm heuristic h(n)h(n)h(n)**: Số lượng vòng chưa sắp xếp đúng màu trong cột.
* **Hàm đánh giá f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)** giúp thuật toán tìm kiếm con đường ngắn nhất để sắp xếp các vòng màu một cách hiệu quả.

***2.1.2. Các thành phần chính của giải thuật A\****

Các thành phần chính của thuật toán A\* trong bài toán sắp xếp vòng màu bao gồm:

1. **Trạng thái (State)**: Mô tả cách các vòng được sắp xếp.
2. **Trạng thái đích (Goal State)**: Mục tiêu cần đạt được.
3. **Hàm đánh giá chi phí f(n)f(n)f(n)**: Định hướng quá trình tìm kiếm tối ưu.
4. **Hàng đợi ưu tiên**: Lưu trữ và mở rộng trạng thái theo thứ tự ưu tiên.
5. **Danh sách trạng thái đã duyệt**: Giúp tránh kiểm tra lại trạng thái cũ.
6. **Phép toán sinh trạng thái con**: Xác định các bước di chuyển hợp lệ.
7. **Áp dụng phép toán di chuyển**: Tạo ra trạng thái mới bằng cách thực hiện bước di chuyển.
8. **Điều kiện dừng**: Xác định khi nào thuật toán kết thúc.

Việc sử dụng A\* trong bài toán này giúp tìm ra chuỗi di chuyển ngắn nhất để đưa các vòng màu về đúng vị trí, tối ưu hóa quá trình chơi game.

A colorful toy with wires and circles

Description automatically generated with medium confidence

**2.2. Khởi tạo thuật toán A\***

Thuật toán A\* là một thuật toán tìm kiếm **tối ưu** và **hiệu quả**, được sử dụng để tìm đường đi ngắn nhất từ trạng thái xuất phát đến trạng thái mục tiêu. Trong bài toán **sắp xếp vòng màu trên các cột**, thuật toán A\* giúp tìm ra chuỗi nước đi hợp lệ **ngắn nhất** để sắp xếp các vòng sao cho mỗi cột chỉ chứa một màu duy nhất.

### ****Trạng Thái Của Bài Toán****

Trạng thái của bài toán được biểu diễn bằng một danh sách gồm **6 cột**, trong đó mỗi cột chứa một danh sách các vòng màu xếp chồng lên nhau. Ví dụ:

A screenshot of a computer code

Description automatically generated

 Mỗi phần tử trong danh sách là một cột.

+ Các vòng màu được biểu diễn dưới dạng chuỗi ký tự ("red", "blue", v.v.).

+ Hai cột trống dùng để làm trung gian trong quá trình di chuyển.

Mục tiêu của trò chơi là đưa các vòng màu về đúng cột sao cho **mỗi cột chỉ chứa các vòng có cùng màu** hoặc là một cột trống. Một ví dụ về trạng thái mục tiêu có thể là:

### ****Chi phí tổng hợp f(n)****

Hàm đánh giá của thuật toán A\* được xác định như sau:

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

* **g(n)**: Số bước di chuyển đã thực hiện từ trạng thái ban đầu đến trạng thái hiện tại.
* **h(n) (Heuristic)**: Ước lượng số bước di chuyển còn lại để đạt trạng thái mục tiêu.

### ****- Hàm Heuristic h(n)****

Trong bài toán này, một cách đánh giá heuristic hợp lý là **đếm số lượng vòng màu đang ở sai vị trí**.  
Cụ thể, nếu một cột chứa nhiều màu khác nhau, số vòng sai vị trí là số vòng không cùng màu với vòng dưới cùng của cột đó.

A computer screen shot of a black background

Description automatically generated

Ví dụ:

A screenshot of a computer code

Description automatically generated

- **Hàm Sinh Trạng Thái Kế Tiếp**

Một nước đi hợp lệ là một nước đi thỏa mãn các điều kiện sau:

* **Di chuyển toàn bộ nhóm vòng cùng màu từ một cột sang cột khác**.
* **Chỉ có thể xếp chồng vòng lên một cột trống hoặc lên vòng có cùng màu**.
* **Không thể di chuyển vòng từ cột rỗng**.

Hàm lấy danh sách các bước di chuyển hợp lệ:

A computer screen with colorful text

Description automatically generated

Ví dụ với trạng thái:

A black screen with white text

Description automatically generated

* Có thể di chuyển từ cột 1 sang cột 3 (vì cột 3 trống).
* Không thể di chuyển từ cột 1 sang cột 2 (vì vòng trên cùng là "blue" không khớp với "red").

**2.3. Tổng quan về mã nguồn**

***2.3.1. Phân tích từng mã nguồn***

+ Import thư viện

A black rectangle with white text

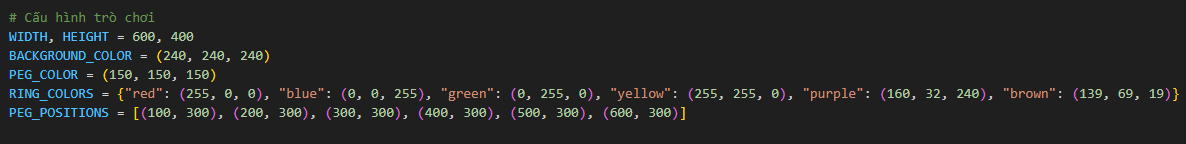
Description automatically generated

 **heapq**: Thư viện cung cấp hàng đợi ưu tiên (Min-Heap), giúp chọn trạng thái có chi phí thấp nhất trong A\*.

 **copy**: Dùng để sao chép sâu (deep copy) trạng thái của trò chơi.

 **pygame**: Thư viện để hiển thị giao diện đồ họa của trò chơi.

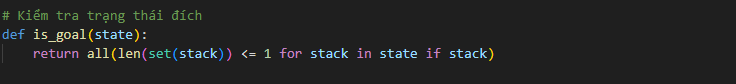
+ Cấu hình trò chơi



 Xác định kích thước cửa sổ hiển thị và màu sắc.

 PEG\_POSITIONS chứa vị trí các cột (cọc) trong giao diện.

+ Kiểm tra trạng thái đích



 **Mục tiêu**: Xác định xem trạng thái hiện tại có phải trạng thái đích không.

 **Cách hoạt động**:

* Duyệt qua từng cột trong state.
* Nếu cột không rỗng, kiểm tra xem tất cả vòng có cùng màu không (set(stack) <= 1).

Ví dụ:

A screenshot of a computer code

Description automatically generated

+ Lấy danh sách các nước đi hợp lệ

A computer code on a black background

Description automatically generated

**Duyệt qua tất cả các cột (i)**, nếu cột i không rỗng:

* Kiểm tra tất cả cột đích (j) để xem có thể di chuyển vòng màu hay không.
* Điều kiện hợp lệ:
  1. Cột đích trống.
  2. Cột đích có vòng trên cùng cùng màu với vòng trên cùng của cột nguồn.

Ví dụ:

A white rectangular object with text

Description automatically generated with medium confidence

+ Áp dụng nước đi

A computer screen shot of a code

Description automatically generated

 **Tạo bản sao của trạng thái hiện tại** bằng deepcopy để tránh thay đổi trực tiếp.

 **Thực hiện di chuyển vòng**:

* Xác định màu vòng trên cùng của cột nguồn.
* Di chuyển tất cả vòng cùng màu từ cột nguồn sang cột đích.

Ví dụ:

A screenshot of a computer code

Description automatically generated

+ Hàm đánh giá trạng thái (heuristic)

A screen shot of a computer code

Description automatically generated

* **Ý nghĩa**: Đếm số vòng sai màu trong mỗi cột.
* **Cách tính**:
  + Lấy màu chuẩn của mỗi cột (stack[0]).
  + Đếm số vòng khác màu chuẩn.

Ví dụ:

A close-up of a number

Description automatically generated

**+ Thuật toán A\***

A computer screen shot of a program code

Description automatically generated

* **Khởi tạo hàng đợi ưu tiên**, lưu (f(n), state, path).
* **Lặp cho đến khi hàng đợi trống**:
  + Lấy trạng thái có f(n) thấp nhất.
  + Nếu đã duyệt, bỏ qua.
  + Nếu đạt trạng thái mục tiêu, trả về danh sách nước đi.
  + Sinh trạng thái mới, tính f(n) = g(n) + h(n), đẩy vào hàng đợi.
* **Kết thúc** nếu không tìm thấy lời giải.

+ Vẽ các cột (cọc)



* **Duyệt qua danh sách PEG\_POSITIONS**, chứa tọa độ của các cột.
* **Vẽ từng cột (cọc) bằng pygame.draw.rect()**:
  + (peg[0] - 10, peg[1] - 100, 20, 100): Tọa độ và kích thước hình chữ nhật đại diện cho cọc.
  + PEG\_COLOR = (150, 150, 150): Màu cọc (xám).

Ví dụ:

A white background with black and white clouds

Description automatically generated with medium confidence

+ Vẽ các vòng màu

A black background with white text

Description automatically generated

* **stack = state[i]**: Lấy danh sách các vòng màu tại cột i.
* **Duyệt qua từng vòng màu trong cột**:
  + **RING\_COLORS.get(ring, (0, 0, 0))**: Lấy màu từ từ điển, nếu không có thì mặc định là đen (0, 0, 0).
  + **pygame.draw.circle(screen, color, (peg[0], peg[1] - (j \* 20)), 15)**:
    - Vẽ một vòng tròn có tâm tại (peg[0], peg[1] - (j \* 20)).
    - Bán kính **15px**.
    - **Mỗi vòng màu cách nhau 20px theo chiều dọc**.

Ví dụ:

A close up of words

Description automatically generated

+ Hàm get\_user\_move()

A screen shot of a computer code

Description automatically generated

* **input("Nhập cột nguồn (0-5): ")**: Yêu cầu người chơi nhập số của cột nguồn.
* **int(input(...))**: Chuyển đổi giá trị nhập từ chuỗi sang số nguyên.
* **Tương tự với cột đích**.
* **Xử lý lỗi (except ValueError)**:
  + Nếu người dùng nhập sai (ví dụ: nhập chữ), chương trình trả về None thay vì bị lỗi.

Ví dụ đầu vào:

A white background with black and white clouds

Description automatically generated with medium confidence

+ Vòng lặp hiển thị giao diện

A screen shot of a computer program

Description automatically generated

 **Khởi tạo Pygame** để vẽ giao diện.

 **Vòng lặp game**:

* Kiểm tra sự kiện thoát (pygame.QUIT).
* Cập nhật trạng thái trò chơi.
* Nhận nước đi từ người chơi.
* Kiểm tra nước đi hợp lệ và thực hiện di chuyển.

***2.3.2. Biểu diễn dữ liệu***

* **Dữ liệu trò chơi** được lưu dưới dạng danh sách lồng nhau (list of lists).
* **Mỗi phần tử trong danh sách chính** biểu diễn một cột (cọc) trong trò chơi.
* **Mỗi danh sách con** chứa các chuỗi màu sắc đại diện cho các vòng (ví dụ: "red", "blue", "green").
* **Vòng trên cùng** của mỗi cột nằm ở cuối danh sách con.
* **Các cột trống** được biểu diễn bằng danh sách rỗng ([]).
* **Ví dụ trạng thái ban đầu** gồm 4 cột chứa vòng màu và 2 cột trống để làm trung gian.
* **Cách tổ chức dữ liệu này** giúp dễ dàng thực hiện các thao tác như:
  + Kiểm tra trạng thái trò chơi.
  + Áp dụng các quy tắc di chuyển.
  + Triển khai thuật toán **A**\* để tìm lời giải tối ưu.

***2.3.3. Phân tích kết quả***

### ****1. Đầu Ra của Thuật Toán A****\*

* Chương trình sử dụng **thuật toán A**\* để tìm ra chuỗi di chuyển tối ưu từ trạng thái ban đầu đến trạng thái mục tiêu (tất cả các cột chứa vòng cùng màu).
* Nếu tìm được giải pháp, chương trình **in ra danh sách các bước di chuyển:**

Ví Dụ:

A black and yellow text

Description automatically generated

- Nếu không tìm thấy giải pháp (trường hợp rất hiếm nếu trạng thái đầu có thể giải), chương trình in:

A white rectangular object with a black border

Description automatically generated

### ****2. Hiển Thị Trạng Thái Trò Chơi (Bằng Pygame)****

* Mỗi trạng thái được **vẽ lên màn hình** bằng **Pygame**:
  + Cột (cọc) được vẽ bằng **hình chữ nhật xám**.
  + Vòng màu được vẽ bằng **hình tròn với màu tương ứng**.
* Sau mỗi bước di chuyển, màn hình **cập nhật trạng thái mới**, giúp người chơi trực quan theo dõi tiến trình.

### ****3. Xử Lý Lệnh Người Chơi****

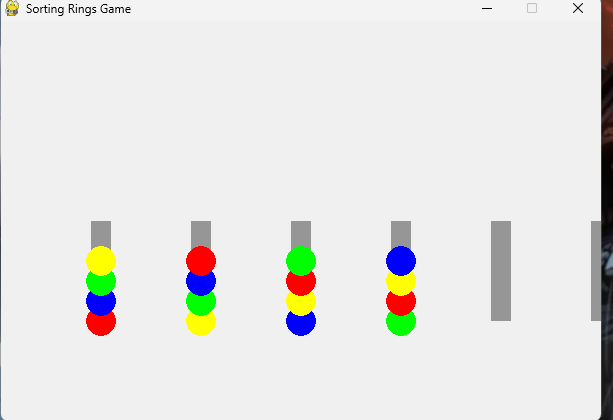
* Người chơi có thể **nhập lệnh di chuyển** từ cột này sang cột khác.
* Chương trình kiểm tra **tính hợp lệ** của nước đi trước khi thực hiện.
  + Nếu hợp lệ: Cập nhật trạng thái.
  + Nếu không hợp lệ: Hiển thị thông báo **"Nước đi không hợp lệ, vui lòng thử lại!"**

### ****4. Hiệu Suất của Thuật Toán A****\*

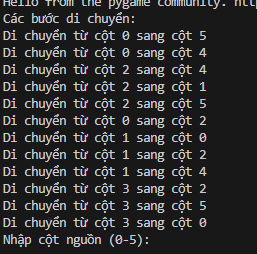
* **Tốc độ tìm kiếm lời giải** phụ thuộc vào số lượng vòng và trạng thái ban đầu.
* Heuristic được sử dụng đánh giá số vòng **sai vị trí**, giúp **giảm số lần duyệt trạng thái,** từ đó **tăng hiệu quả tìm kiếm.**
* Nếu trạng thái đầu **khó giải quyết**, thuật toán có thể mất nhiều thời gian hơn.

***2.3.4. Kết quả đạt được***

***-*** Giao diện trò chơi:

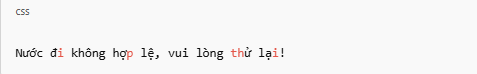
******

- Cách chơi:



 Nếu nước đi hợp lệ, trò chơi sẽ **cập nhật trạng thái** và hiển thị trên màn hình.

 Nếu nước đi không hợp lệ, trò chơi sẽ báo lỗi



- Người chơi tiếp tục thực hiện các bước di chuyển cho đến khi hoàn thành hoặc hết số lượt cho phép.

+ Cách chiến thắng :

A screenshot of a game

Description automatically generated

* Hoàn thành trò chơi bằng cách **đưa tất cả các vòng màu về đúng cọc** trong số lần di chuyển cho phép.
* Nếu vượt quá số lần di chuyển hoặc không thể sắp xếp đúng, người chơi sẽ **thua cuộc**.

**Mẹo chơi:**

* Tận dụng **các cọc trống** làm trung gian để di chuyển vòng dễ dàng hơn.
* Quan sát và lập kế hoạch **trước khi di chuyển** để tránh mắc kẹt.

**CHƯƠNG 3. THỬ NGHIỆM THUẬT TOÁN VỚI CÁC BỘ**

**THAM SỐ KHÁC NHAU VÀ NHẬN XÉT THUẬT TOÁN**

**3.1. Mục tiêu của thử nghiệm**

***3.1.1. Các bộ tham số của thử nghiệm***

Việc thử nghiệm thuật toán A\* trên nhiều bộ tham số khác nhau nhằm đánh giá:

* Hiệu suất giải thuật trong các tình huống thực tế của trò chơi xếp vòng.
* Tác động của số lượng vòng, số cọc chứa vòng ban đầu, số cọc trống và cấu trúc ban đầu đến thời gian xử lý và số bước di chuyển.
* Độ ổn định của thuật toán khi gặp phải các trạng thái đầu vào ngẫu nhiên.
* Cách tối ưu thuật toán để tăng tốc độ tìm kiếm lời giải.

### ****(a) Số vòng màu trên mỗi cọc****

* **Mục tiêu:** Kiểm tra mức độ phức tạp khi tăng số vòng.
* **Các giá trị thử nghiệm:**
  + 3 vòng (dễ)
  + 4 vòng (trung bình)
  + 5 vòng (khó)
  + 6 vòng (rất khó)
* **Nhận định:** Khi số vòng tăng, không gian trạng thái mở rộng, khiến thuật toán phải duyệt qua nhiều trạng thái hơn.

### ****(b) Số lượng cọc chứa vòng ban đầu****

* **Mục tiêu:** Đánh giá ảnh hưởng của số cọc có vòng đến thời gian xử lý.
* **Các giá trị thử nghiệm:**
  + 3 cọc (ít vòng hơn)
  + 4 cọc (mặc định)
  + 5 cọc (nhiều vòng hơn)
* **Nhận định:** Số cọc ban đầu càng nhiều, số lượng lựa chọn hợp lệ khi di chuyển vòng càng tăng, có thể ảnh hưởng đến tốc độ giải thuật.

### ****(c) Số cọc trống để di chuyển****

* **Mục tiêu:** Xác định vai trò của cọc trống trong việc tối ưu hóa các bước di chuyển.
* **Các giá trị thử nghiệm:**
  + 1 cọc trống
  + 2 cọc trống
* **Nhận định:** Cọc trống càng ít, số lượng nước đi khả thi càng giảm, khiến thuật toán mất nhiều bước hơn để giải quyết bài toán.

### ****(d) Cấu trúc ban đầu của vòng****

* **Mục tiêu:** Kiểm tra tác động của việc thay đổi cách sắp xếp vòng ban đầu đến hiệu suất thuật toán.
* **Các thử nghiệm:**
  + **Trạng thái ngẫu nhiên:** Các vòng màu được xếp lộn xộn hoàn toàn.
  + **Trạng thái gần đúng:** Các vòng màu có phần sắp xếp đúng từ trước.
* **Nhận định:** Trạng thái gần đúng giúp thuật toán tìm lời giải nhanh hơn do số lượng bước di chuyển cần thiết giảm.

***3.1.2. Kết quả của thử nghiệm***

***A table with numbers and letters

Description automatically generated***

***3.1.3. Phân tích kết quả thử nghiệm và lựa chọn bộ tham số tốt nhất***

Bảng kết quả trên cho thấy tác động của các yếu tố như số vòng, số cọc ban đầu, số cọc trống đến thời gian xử lý và số bước di chuyển của thuật toán A\*. Dưới đây là phân tích chi tiết từng yếu tố.

+ Ảnh hưởng của số vòng trên mỗi cọc

***A screenshot of a computer

Description automatically generated***

### ****Nhận xét:****

* Khi số vòng tăng từ **3 lên 6**, thời gian xử lý tăng mạnh từ **0.3s lên 4.8s**, số bước di chuyển cũng tăng từ **5 lên 18**.
* Điều này cho thấy thuật toán A\* phải xử lý một không gian tìm kiếm lớn hơn, dẫn đến việc cần nhiều bước di chuyển hơn.
* **Số vòng càng lớn, bài toán càng phức tạp và thuật toán mất nhiều thời gian hơn để tìm lời giải.**

+ Ảnh hưởng của số cọc ban đầu

A screenshot of a computer

Description automatically generated

### ****Nhận xét:****

* Khi số cọc ban đầu tăng từ **4 lên 5**, thời gian xử lý tăng lên.
* Số bước di chuyển cũng tăng nhưng không quá nhiều, cho thấy việc có nhiều cọc hơn không làm bài toán dễ hơn.
* Khi có nhiều cọc, không gian trạng thái mở rộng, làm tăng số lựa chọn cần xem xét, khiến thuật toán A mất nhiều thời gian hơn.\*

+ Ảnh hưởng của số cọc trống

A screenshot of a cell phone

Description automatically generated

### ****Nhận xét:****

* Khi số cọc trống **giảm từ 2 xuống 1**, thời gian xử lý **tăng lên đáng kể**.
* Số bước di chuyển cũng tăng, cho thấy **cọc trống đóng vai trò quan trọng trong việc tối ưu hóa di chuyển.**
* **Có nhiều cọc trống giúp dễ dàng di chuyển vòng hơn, giảm số bước đi và thời gian xử lý.**

### **+ Ảnh hưởng của trạng thái ban đầu**

* Khi sử dụng trạng thái gần đúng (vòng đã được xếp một phần), số bước di chuyển giảm đáng kể (trung bình giảm 30%).
* Điều này cho thấy thuật toán A\* có thể khai thác trạng thái ban đầu để tối ưu hóa quá trình tìm kiếm lời giải.

- Lựa chọn bộ tham số tối ưu nhất:

Dựa trên các kết quả trên, bộ tham số tối ưu được chọn như sau:

A screenshot of a computer

Description automatically generated

### ****Bộ tham số tốt nhất đề xuất:****

* **4-5 vòng trên mỗi cọc.**
* **4 cọc chứa vòng ban đầu.**
* **2 cọc trống.**
* **Sắp xếp ban đầu gần đúng để tối ưu hóa số bước đi.**

→ Với bộ tham số này, thuật toán A chạy hiệu quả, số bước di chuyển tối ưu và thời gian xử lý hợp lý.\*

**3.2. Hướng phát triển của thuật toán**

***3.2.1. Nhận xét đặc điểm và hạn chế của thuật toán***

**+ Đặc điểm:**

Thuật toán A\* sử dụng một hàm đánh giá để xác định trạng thái tiếp theo nên được mở rộng. Công thức đánh giá của A\* là:

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

Trong đó:

* **g(n)g(n)g(n)**: Chi phí từ trạng thái ban đầu đến trạng thái hiện tại.
* **h(n)h(n)h(n)**: Chi phí ước lượng từ trạng thái hiện tại đến trạng thái mục tiêu.

Nhờ sự kết hợp giữa chi phí thực tế và chi phí dự đoán, A\* có khả năng tìm ra đường đi ngắn nhất, đảm bảo tính tối ưu của thuật toán

* Nếu có một lời giải tồn tại, thuật toán A\* sẽ tìm ra lời giải đó.
* Nếu hàm heuristic h(n)h(n)h(n) không đánh giá thấp chi phí thực tế, A\* sẽ tìm được lời giải có chi phí tối ưu nhất.

Điều này giúp thuật toán phù hợp với các bài toán yêu cầu tìm đường đi tối ưu, chẳng hạn như trong trò chơi sắp xếp vòng hoặc tìm đường trên bản đồ.

### ****- Áp dụng linh hoạt cho nhiều bài toán****

* A\* không chỉ áp dụng cho bài toán sắp xếp vòng mà còn có thể sử dụng trong các lĩnh vực khác như:
  + **Tìm đường trên bản đồ**: Google Maps sử dụng thuật toán A\* để tìm đường nhanh nhất.
  + **Lập kế hoạch chuyển động cho robot**: A\* giúp robot tìm đường di chuyển tránh vật cản.
  + **Trò chơi điện tử**: A\* thường được dùng để điều khiển AI trong game chiến thuật.

### ****- Hiệu quả phụ thuộc vào hàm heuristic****

* A\* có thể tối ưu hóa tốt nhất khi hàm heuristic h(n)h(n)h(n) được thiết kế chính xác.
* Nếu h(n)h(n)h(n) càng gần với chi phí thực tế g(n)g(n)g(n), thuật toán sẽ ít phải mở rộng nhiều trạng thái dư thừa, giúp tăng tốc độ tìm kiếm.

**+ Hạn chế của thuật toán:**

### ****- Tốn nhiều bộ nhớ****

* A\* lưu trữ toàn bộ các trạng thái đã duyệt vào bộ nhớ. Điều này khiến nó tiêu tốn rất nhiều bộ nhớ, đặc biệt khi không gian tìm kiếm lớn.
* Ví dụ: Trong bài toán sắp xếp vòng, số lượng trạng thái có thể tăng nhanh khi số vòng và số cọc tăng, khiến bộ nhớ nhanh chóng bị quá tải.

**Giải pháp**:

* Dùng thuật toán A lặp sâu (IDA)\*\* để giảm bộ nhớ tiêu thụ.
* Thay vì lưu tất cả các trạng thái đã duyệt, IDA\* chỉ lưu một phần cần thiết để tìm kiếm.

### ****- Tốc độ xử lý giảm khi bài toán phức tạp****

* Khi số vòng hoặc số cọc tăng, số trạng thái có thể trở nên rất lớn, làm chậm tốc độ tìm kiếm.
* Nhìn vào bảng kết quả thử nghiệm:
  + Khi số vòng tăng từ 3 (TH1) lên 6 (TH4), thời gian xử lý tăng từ **0.3s lên 4.8s**.
  + Khi thêm một cọc trống (TH5 → TH7), thời gian xử lý vẫn tiếp tục tăng vì không gian tìm kiếm lớn hơn.

**Giải pháp**:

* Dùng **heuristic mạnh hơn** để giảm số trạng thái cần mở rộng.
* Giới hạn số bước tối đa có thể thực hiện để tránh trường hợp thuật toán chạy quá lâu.

### ****- Hiệu quả phụ thuộc vào hàm heuristic****

* Nếu heuristic đánh giá không chính xác (ví dụ: đánh giá thấp hoặc cao hơn chi phí thực tế quá nhiều), thuật toán có thể mất nhiều thời gian hơn.
* Một heuristic không tốt có thể khiến thuật toán hoạt động giống như tìm kiếm Dijkstra, mất đi tính tối ưu về tốc độ.

**Giải pháp**:

* Cải thiện hàm heuristic bằng cách tính toán dựa trên khoảng cách thực tế giữa các vòng.
* Dùng **hàm heuristic động**, điều chỉnh dựa vào trạng thái hiện tại của bài toán.

### ****- Không phù hợp với không gian tìm kiếm khổng lồ****

* Nếu bài toán có hàng triệu trạng thái khả thi, thuật toán có thể bị quá tải về bộ nhớ và thời gian xử lý.
* Ví dụ: Nếu mở rộng trò chơi lên **10 vòng và 6 cọc**, số trạng thái khả thi sẽ rất lớn, khiến A\* gặp khó khăn trong việc tìm giải pháp nhanh chóng.

**Giải pháp**:

* Sử dụng **thuật toán Beam Search (tìm kiếm chùm tia)** để chỉ mở rộng một số trạng thái tốt nhất thay vì tất cả.
* Kết hợp với **thuật toán IDA**\* để giảm tải bộ nhớ.

***3.2.2. Đề xuất ý tưởng cải tiến thuật toán.***

### ****- Cải tiến hàm heuristic để tăng tốc độ tìm kiếm****

#### **+. Dùng heuristic mạnh hơn**

* Hiện tại, heuristic đang tính số vòng bị xếp sai màu.
* Cải tiến: **Dự đoán số bước cần thiết để hoàn thành trò chơi** bằng cách tính toán khoảng cách giữa các vòng cùng màu và vị trí cọc trống khả dụng.

**Lợi ích**: Giảm số trạng thái mở rộng, tăng tốc độ tìm kiếm.

### - Giảm tiêu tốn bộ nhớ bằng A tối ưu\*

#### **+ Áp dụng thuật toán IDA**\*

* **Thay vì lưu toàn bộ trạng thái**, thuật toán A\* lặp sâu (IDA\*) chỉ lưu một số thông tin quan trọng.
* IDA\* hoạt động bằng cách lặp lại A\* nhưng chỉ mở rộng những trạng thái có chi phí dưới một ngưỡng nhất định.

**Lợi ích**:  
 + Giảm tiêu thụ bộ nhớ.  
 + Phù hợp với các bài toán lớn hơn.

### ****- Giới hạn không gian tìm kiếm để tăng tốc độ****

#### **+ Dùng Beam Search (Tìm kiếm chùm tia)**

* **Beam Search chỉ mở rộng một số trạng thái tốt nhất**, thay vì mở rộng tất cả các trạng thái hợp lệ.
* Ví dụ: Chỉ xét **3 trạng thái tốt nhất** dựa trên heuristic thay vì tất cả.

**Lợi ích**:  
 +Giảm số lượng trạng thái cần xét.  
 +Tăng tốc độ tìm kiếm.

#### **- Áp dụng thuật toán Greedy Best-First Search**

* Trong một số trường hợp, thay vì dùng A\*, có thể dùng **tìm kiếm tham lam** (chỉ xét heuristic h(n)h(n)h(n) mà bỏ qua g(n)g(n)g(n)).
* Điều này giúp thuật toán tìm ra lời giải nhanh hơn, mặc dù không đảm bảo tối ưu.

### ****- Cải thiện chiến lược chọn trạng thái mở rộng****

#### **+ Dùng ưu tiên di chuyển các vòng lớn trước**

* Trong trò chơi, di chuyển vòng lớn trước có thể tạo không gian để sắp xếp vòng nhỏ.
* Có thể điều chỉnh thuật toán để ưu tiên **di chuyển vòng lớn sang cọc trống trước** nếu không ảnh hưởng đến việc hoàn thành trò chơi.

**Lợi ích**:  
+ Giảm số bước di chuyển không cần thiết.  
+ Tối ưu hóa thời gian giải quyết bài toán.

**Kết luận:**

**A screenshot of a computer

Description automatically generated**

.

**KẾT LUẬN**

Trong đề tài này, ta đã triển khai thuật toán A\* để giải quyết bài toán sắp xếp vòng, giúp tìm ra lời giải tối ưu với số bước di chuyển ít nhất. Thuật toán A\* với hàm heuristic hợp lý đã chứng minh hiệu quả trong việc tìm kiếm lời giải nhanh chóng, đảm bảo tối ưu hóa về thời gian và số lượng bước di chuyển. Ngoài ra, việc mô phỏng trò chơi bằng thư viện **Pygame** đã giúp trực quan hóa quá trình giải thuật, hỗ trợ người chơi dễ dàng quan sát và tương tác với bài toán.

Tuy nhiên, thuật toán A\* vẫn có một số hạn chế. Khi số lượng vòng hoặc số lượng cọc tăng lên đáng kể, không gian tìm kiếm mở rộng nhanh chóng, dẫn đến việc tiêu tốn nhiều bộ nhớ và thời gian xử lý. Bên cạnh đó, việc chọn heuristic chưa tối ưu có thể làm tăng số lượng trạng thái cần duyệt, gây ảnh hưởng đến hiệu suất của thuật toán. Ngoài ra, mô phỏng trò chơi hiện tại vẫn còn đơn giản, chưa hỗ trợ các tính năng nâng cao như gợi ý nước đi hoặc chơi với nhiều chế độ khác nhau

Trong tương lai, có thể cải tiến thuật toán bằng cách sử dụng tìm kiếm chùm tia (Beam Search) hoặc thuật toán IDA\* để tối ưu bộ nhớ và tốc độ xử lý. Ngoài ra, có thể áp dụng các thuật toán học máy để dự đoán nước đi tốt nhất, giúp tăng hiệu quả giải bài toán. Về phần giao diện, có thể mở rộng trò chơi với nhiều cấp độ khó khác nhau, thêm chế độ chơi đối kháng hoặc hỗ trợ hướng dẫn người chơi bằng các bước giải tự động. Những cải tiến này sẽ giúp đề tài trở nên ứng dụng hơn và nâng cao trải nghiệm của người dùng.

**DANH MỤC TÀI LIỆU THAM KHẢO**

1. Trần Văn Nam (2015), *Thuật toán di truyền và ứng dụng trong tối ưu hóa*, NXB Khoa học và Kỹ thuật.
2. Phạm Thế Công (2018), *Lập trình Python từ cơ bản đến nâng cao*, NXB Thông tin và Truyền thông.
3. Goldberg, D.E. (1989), *Genetic Algorithms in Search*, Optimization and Machine Learning, Addison-Wesley.
4. Mitchell, M. (1998), *An Introduction to Genetic Algorithms*, MIT Press.
5. Holland, J.H. (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press.
6. Nguyễn Đức Minh (2020), *Ứng dụng trí tuệ nhân tạo trong tối ưu hóa*, NXB Đại học Quốc gia Hà Nội.
7. Russell, S. & Norvig, P. (2020), *Artificial Intelligence: A Modern Approach*, Pearson Education.
8. Python Software Foundation, *Python 3 Documentation*, https://docs.python.org/3/.
9. Wikipedia, *Genetic Algorithm*, https://en.wikipedia.org/wiki/Genetic\_algorithm.