**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: chương trình trò chơi đua xe**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Mã Sinh Viên | Họ và Tên | Ngày Sinh | Lớp |
| 1 | 1771020285 | Phạm Duy Hòa | 28/03/2005 | CNTT 17-04 |
| 2 | 1771020538 | Nguyễn Kiều Phong | 19/03/2005 | CNTT 17-04 |
| 3 | 1771020177 | Lê Quang Dũng | 20/04/2005 | CNTT 17-04 |

### 

**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: Chương trình trò chơi đua xe**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| STT | Mã Sinh Viên | Họ và Tên | Ngày Sinh | Điểm | |
| Bằng Số | Bằng Chữ |
| 1 | 1771020285 | Phạm Duy Hòa | 28/03/2005 |  |  |
| 2 | 1771020538 | Nguyễn Kiều Phong | 19/03/2005 |  |  |
| 3 | 1771020177 | Lê Quang Dũng | 20/04/2005 |  |  |

### 

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

**Hà Nội, năm 2025**

**LỜI NÓI ĐẦU**

Trong bối cảnh công nghệ thông tin ngày càng phát triển, trí tuệ nhân tạo (AI) và các thuật toán tìm đường thông minh đã và đang được ứng dụng rộng rãi trong nhiều lĩnh vực, từ điều hướng GPS, robot tự hành, cho đến các trò chơi điện tử. Việc áp dụng các thuật toán này không chỉ giúp nâng cao hiệu suất hoạt động mà còn mang đến những trải nghiệm thú vị và thực tế hơn cho người dùng.

Nhận thấy tầm quan trọng của các thuật toán tìm kiếm đường đi trong lĩnh vực trò chơi và ứng dụng thực tế, nhóm chúng tôi đã lựa chọn đề tài \**“Trò chơi đua xe sử dụng thuật toán A* để tìm đường đi tối ưu”\*\*. Đề tài này không chỉ giúp người chơi thử thách kỹ năng điều khiển xe mà còn kết hợp yếu tố trí tuệ nhân tạo để tính toán và tìm ra con đường nhanh nhất, hiệu quả nhất từ điểm xuất phát đến đích.

Trò chơi được xây dựng trên nền tảng đồ họa 2D, trong đó người chơi sẽ điều khiển một chiếc xe đua di chuyển qua một bản đồ chứa nhiều chướng ngại vật khác nhau như tường, bãi cát, hoặc các vật cản khác. Xe chỉ có thể di chuyển qua các ô trống trên bản đồ, trong khi thuật toán A\* sẽ giúp xác định lộ trình tối ưu nhất để hoàn thành đường đua trong thời gian ngắn nhất. Ngoài ra, hệ thống còn ghi nhận thời gian hoàn thành và tính điểm để đánh giá hiệu suất của người chơi.

Để thực hiện đề tài này, nhóm chúng tôi đã nghiên cứu kỹ thuật lập trình trò chơi bằng **Pygame**, một thư viện phổ biến trong Python dùng để xây dựng giao diện đồ họa. Đồng thời, chúng tôi cũng tìm hiểu và triển khai thuật toán A\* để tính toán đường đi tối ưu trong môi trường có chướng ngại vật. Việc kết hợp các công nghệ này không chỉ giúp trò chơi trở nên hấp dẫn hơn mà còn góp phần củng cố kiến thức về thuật toán tìm đường, ứng dụng thực tế trong lĩnh vực trí tuệ nhân tạo và khoa học máy tính.

Trong quá trình thực hiện, nhóm chúng tôi đã gặp không ít khó khăn, từ việc thiết kế bản đồ hợp lý, tối ưu thuật toán A\* để đảm bảo tốc độ xử lý nhanh, đến việc xây dựng giao diện trực quan, thân thiện với người chơi. Tuy nhiên, nhờ sự hướng dẫn tận tình của giảng viên cùng với tinh thần làm việc nhóm, chúng tôi đã từng bước hoàn thiện sản phẩm và rút ra nhiều kinh nghiệm quý báu trong việc phát triển trò chơi và ứng dụng thuật toán AI.

Mặc dù đã cố gắng hoàn thiện đề tài một cách tốt nhất, nhưng chắc chắn vẫn còn những thiếu sót. Nhóm chúng tôi mong nhận được sự góp ý từ thầy cô và các bạn để đề tài có thể được cải thiện và hoàn thiện hơn.

Chúng tôi xin chân thành cảm ơn giảng viên hướng dẫn, các bạn trong lớp, và tất cả những ai đã hỗ trợ, đóng góp ý kiến để nhóm có thể hoàn thành đề tài này.

**MỤC LỤC**

[**CHƯƠNG 1. PHÂN TÍCH YÊU CẦU BÀI TOÁN** 11](#_Toc193092117)

[**1.1 Giới thiệu bài toán** 11](#_Toc193092118)

[1.1.1 Bối cảnh và lý do chọn đề tài 11](#_Toc193092119)

[1.1.2 Mục tiêu của đề tài 11](#_Toc193092120)

[**1.2 Yêu cầu của hệ thống** 11](#_Toc193092121)

[1.2.1 Yêu cầu chức năng 11](#_Toc193092122)

[1.2.2 Yêu cầu phi chức năng 12](#_Toc193092123)

[**1.3 Mô hình hóa bài toán** 13](#_Toc193092124)

[1.3.1 Biểu diễn bản đồ đường đua 13](#_Toc193092125)

[1.3.2 Cách thức di chuyển của xe 13](#_Toc193092126)

[1.3.3 Ứng dụng thuật toán A\* 14](#_Toc193092127)

[**CHƯƠNG 2. PHÂN TÍCH THUẬT TOÁN A**\* 15](#_Toc193092128)

[**2.1 Giới thiệu về thuật toán A**\* 15](#_Toc193092129)

[2.1.1 Tổng quan về thuật toán A\* 15](#_Toc193092130)

[2.1.2 Nguyên lý hoạt động 15](#_Toc193092131)

[**2.2 Hàm heuristic trong A**\* 15](#_Toc193092132)

[2.2.1 Các loại hàm heuristic phổ biến 15](#_Toc193092133)

[2.2.2 Ảnh hưởng của heuristic đến thuật toán 16](#_Toc193092134)

[**2.3 Quy trình thực hiện thuật toán A**\* 16](#_Toc193092135)

[**2.4 Mô phỏng thuật toán A**\* 17](#_Toc193092136)

[**2.5 Đánh giá thuật toán A**\* 17](#_Toc193092138)

[2.5.1 Ưu điểm 17](#_Toc193092139)

[2.5.2 Hạn chế 18](#_Toc193092140)

[**CHƯƠNG 3. PHÂN TÍCH VÀ TRIỂN KHAI MÃ NGUỒN** 19](#_Toc193092141)

[**3.1 Giới thiệu** 19](#_Toc193092142)

[**3.2 Cấu trúc tổng quan của chương trình** 19](#_Toc193092144)

[Chương trình gồm các thành phần chính sau: 19](#_Toc193092145)

[**3.3 Phân tích chi tiết mã nguồn** 19](#_Toc193092146)

[3.3.1 Khởi tạo Pygame và cấu hình trò chơi 19](#_Toc193092147)

[3.3.2 Thuật toán A\* tìm đường đi 20](#_Toc193092148)

[3.3.3 Sinh bản đồ ngẫu nhiên và đảm bảo có đường đi 21](#_Toc193092149)

[3.3.4 Vòng lặp chính của trò chơi 22](#_Toc193092150)

[**3.4. Thử nghiệm thuật toán với các bộ tham số khác nhau, lưu lại kết quả sau mỗi lần thử nghiệm** 26](#_Toc193092151)

[**KẾT LUẬN** 32](#_Toc193092155)

[**DANH MỤC TÀI LIỆU THAM KHẢO** 34](#_Toc193092156)

**BẢNG CÁC TỪ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **STT** | **TỪ VIẾT TẮT** | **VIẾT ĐẦY ĐỦ** |
| **1** | AI | Trí tuệ nhân tạo (Artificial Intelligence) |
| **2** | A\* | Thuật toán tìm đường A-star |
| **3** | CPU | Bộ xử lý trung tâm (Central Processing Unit) |
| **4** | FPS | Số khung hình trên giây (Frames Per Second) |
| **5** | GUI | Giao diện đồ họa người dùng (Graphical User Interface) |
| **6** | GPS | Hệ thống định vị toàn cầu (Global Positioning System) |
| **7** | JPS | Thuật toán Jump Point Search (Tìm kiếm điểm nhảy) |
| **8** | ML | Học máy (Machine Learning) |
| **9** | NPU | Bộ xử lý thần kinh (Neural Processing Unit) |
| **10** | RAM | Bộ nhớ truy cập ngẫu nhiên (Random Access Memory) |
| **11** | RL | Học tăng cường (Reinforcement Learning) |
| **12** | UI | Giao diện người dùng (User Interface) |
| **13** | UX | Trải nghiệm người dùng (User Experience) |
| **14** | VR | Thực tế ảo (Virtual Reality) |

**CHƯƠNG 1. PHÂN TÍCH YÊU CẦU BÀI TOÁN**

## **1.1 Giới thiệu bài toán**

### ****1.1.1 Bối cảnh và lý do chọn đề tài****

Trong các trò chơi điện tử, việc tìm kiếm đường đi tối ưu là một bài toán phổ biến, xuất hiện trong nhiều thể loại khác nhau như game chiến thuật, game nhập vai, và đặc biệt là game đua xe. Các thuật toán tìm kiếm đường đi giúp nhân vật hoặc phương tiện di chuyển thông minh hơn, tối ưu hóa thời gian và nâng cao trải nghiệm người chơi.

Với mong muốn nghiên cứu và ứng dụng thuật toán tìm kiếm hiệu quả, nhóm chúng tôi lựa chọn đề tài “Trò chơi đua xe sử dụng thuật toán A để tìm đường đi tối ưu”\*\*. Thuật toán A được đánh giá là một trong những phương pháp mạnh mẽ nhất trong lĩnh vực tìm đường do khả năng cân bằng giữa tốc độ và độ chính xác.

### ****1.1.2 Mục tiêu của đề tài****

Mục tiêu chính của đề tài này là:

* Xây dựng một trò chơi đua xe 2D, trong đó người chơi cần điều khiển xe đua di chuyển từ điểm xuất phát đến đích trong thời gian nhanh nhất.
* Áp dụng thuật toán A\* để tìm ra con đường ngắn nhất giúp người chơi về đích nhanh nhất có thể.
* Phát triển một hệ thống tính điểm dựa trên thời gian hoàn thành đường đua.
* Cung cấp giao diện trực quan, thân thiện với người dùng để tăng tính hấp dẫn của trò chơi.

## **1.2 Yêu cầu của hệ thống**

### ****1.2.1 Yêu cầu chức năng****

Trò chơi cần đảm bảo các chức năng sau:

1. **Hiển thị bản đồ đường đua:**
   * Bản đồ được thể hiện dưới dạng ma trận 2D, trong đó có các ô trống và chướng ngại vật.
   * Các ô trên bản đồ có thể có các giá trị khác nhau đại diện cho từng loại địa hình.
2. **Người chơi có thể điều khiển xe:**
   * Xe có thể di chuyển theo các hướng: lên, xuống, trái, phải (và có thể thêm di chuyển chéo nếu cần).
   * Xe không thể di chuyển qua chướng ngại vật.
3. **Áp dụng thuật toán A*\** để tìm đường đi tối ưu:**
   * Khi bắt đầu trò chơi, thuật toán A\* sẽ tự động tìm kiếm lộ trình nhanh nhất từ vị trí xuất phát đến đích.
   * Nếu tìm thấy đường đi, trò chơi sẽ hiển thị hướng dẫn hoặc gợi ý đường đi.
   * Nếu không có đường đi hợp lệ, trò chơi thông báo người chơi thua.
4. **Tính điểm và hiển thị thời gian:**
   * Ghi nhận thời gian từ lúc xe bắt đầu di chuyển đến khi về đích.
   * Điểm số có thể được tính dựa trên thời gian hoàn thành và số lần va chạm vào chướng ngại vật (nếu có).
5. **Giao diện đồ họa và điều khiển:**
   * Hiển thị bản đồ, vị trí của xe, chướng ngại vật và đích đến.
   * Cung cấp các nút hoặc phím điều khiển để người chơi thao tác dễ dàng.

### ****1.2.2 Yêu cầu phi chức năng****

Ngoài các yêu cầu chức năng, trò chơi cần đảm bảo các tiêu chí sau:

* **Hiệu suất cao:** Thuật toán A\* phải được tối ưu để đảm bảo tốc độ tìm đường nhanh, tránh làm gián đoạn trải nghiệm của người chơi.
* **Dễ sử dụng:** Giao diện đơn giản, dễ thao tác, phù hợp với nhiều đối tượng người chơi.
* **Khả năng mở rộng:** Có thể bổ sung thêm nhiều loại chướng ngại vật hoặc chế độ chơi trong tương lai.
* **Tương thích hệ thống:** Chương trình có thể chạy trên máy tính với cấu hình phổ thông, sử dụng thư viện Python phổ biến như Pygame.

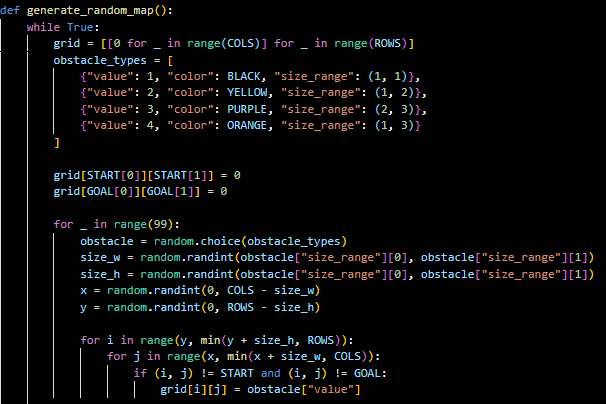
## **1.3 Mô hình hóa bài toán**

### ****1.3.1 Biểu diễn bản đồ đường đua****

Bản đồ được thể hiện dưới dạng ma trận 2D, trong đó mỗi ô có một giá trị đặc trưng:

* 0 – Ô trống (xe có thể đi qua).
* 1 – Chướng ngại vật (tường, bãi cát, vật cản).
* 2 – Vị trí xuất phát của xe.
* 3 – Đích đến.

Ví dụ về một bản đồ được tạo random:



### ****Cách thức di chuyển của xe****

### ****-**** Người chơi sử dụng các phím mũi tên trên bàn phím để điều khiển xe **lên, xuống, trái, phải** theo các bước đơn vị.

### ****1.3.3 Ứng dụng thuật toán A****\*

Thuật toán A\* sẽ được sử dụng để tìm đường đi ngắn nhất từ vị trí xuất phát đến đích bằng cách tính toán **hàm chi phí**:

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

Trong đó:

* g(n): Chi phí từ điểm xuất phát đến vị trí hiện tại.
* h(n): Ước lượng khoảng cách từ vị trí hiện tại đến đích (thường sử dụng khoảng cách Manhattan hoặc Euclidean).
* Thuật toán sẽ mở rộng các nút có chi phí thấp nhất để tìm ra con đường tối ưu.

**CHƯƠNG 2. PHÂN TÍCH THUẬT TOÁN A**\*

## **2.1 Giới thiệu về thuật toán A**\*

### ****2.1.1 Tổng quan về thuật toán A****\*

Thuật toán A\* (A-star) là một trong những thuật toán tìm đường mạnh mẽ và hiệu quả nhất, thường được sử dụng trong các ứng dụng trí tuệ nhân tạo, trò chơi điện tử, hệ thống điều hướng, và nhiều lĩnh vực khác.

A\* là một thuật toán tìm kiếm theo chiến lược **tìm kiếm tốt nhất trước** (Best-First Search) kết hợp với **tìm kiếm theo chi phí thấp nhất** (Dijkstra). Thuật toán này sử dụng một **hàm đánh giá** để mở rộng những đường đi tiềm năng có chi phí thấp nhất đến mục tiêu.

### ****2.1.2 Nguyên lý hoạt động****

A\* tìm kiếm đường đi tối ưu từ điểm xuất phát S đến điểm đích G bằng cách mở rộng các ô (nút) có chi phí thấp nhất. Mỗi nút nnn trong thuật toán A\* được đánh giá dựa trên **hàm chi phí tổng hợp**:

Trong đó:

* **g(n)** Chi phí thực tế từ điểm xuất phát S đến ô hiện tại nnn.
* **h(n)**: Hàm ước lượng chi phí từ n đến điểm đích G (còn gọi là hàm heuristic).
* **f(n):** Tổng chi phí ước tính từ điểm xuất phát đến đích nếu đi qua n.

A\* luôn chọn mở rộng ô có giá trị **f(n)** nhỏ nhất, đảm bảo đường đi tối ưu và hiệu quả.

## **2.2 Hàm heuristic trong A**\*

### ****2.2.1 Các loại hàm heuristic phổ biến****

Hàm heuristic h(n) đóng vai trò quan trọng trong việc tìm kiếm đường đi hiệu quả. Có một số cách tính toán heuristic phổ biến:

1. **Khoảng cách Manhattan** (áp dụng khi chỉ có thể di chuyển lên/xuống/trái/phải):
2. **Khoảng cách Euclidean** (áp dụng khi có thể di chuyển theo đường chéo):
3. **Khoảng cách Chebyshev** (áp dụng khi di chuyển chéo không tốn thêm chi phí):

### ****2.2.2 Ảnh hưởng của heuristic đến thuật toán****

* Nếu **h(n) = 0**, A\* hoạt động giống như thuật toán Dijkstra (tìm đường đi ngắn nhất mà không ưu tiên đích).
* Nếu **h(n) quá nhỏ**, thuật toán sẽ mở rộng nhiều ô hơn mức cần thiết, gây tốn kém tài nguyên.
* Nếu **h(n) quá lớn**, thuật toán có thể chọn đường đi không tối ưu.
* Một heuristic tốt giúp A\* nhanh chóng hội tụ về điểm đích mà không cần mở rộng quá nhiều nút.

## **2.3 Quy trình thực hiện thuật toán A**\*

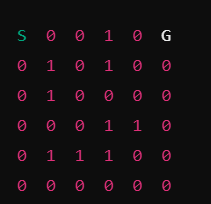
Dưới đây là các bước thực hiện thuật toán A\* trong trò chơi đua xe:

1. **Khởi tạo**:
   * Đặt điểm xuất phát vào danh sách mở (Open List).
   * Đặt tất cả các ô khác vào danh sách đóng (Closed List) ban đầu rỗng.
2. **Duyệt các ô trên bản đồ**:
   * Chọn ô có f(n) nhỏ nhất trong danh sách mở để mở rộng.
   * Nếu ô này là điểm đích, thuật toán kết thúc.
   * Nếu không, đưa ô này vào danh sách đóng.
3. **Mở rộng ô lân cận**:
   * Xét các ô kề cạnh có thể đi qua.
   * Nếu ô chưa có trong danh sách mở, tính toán g(n),h(n),f(n) và thêm vào danh sách mở.
   * Nếu ô đã có trong danh sách mở nhưng có đường đi tốt hơn, cập nhật giá trị g(n),f(n)
4. **Lặp lại quá trình trên cho đến khi tìm thấy đích hoặc danh sách mở rỗng (không có đường đi).**
5. **Trả về đường đi tối ưu hoặc thông báo không có đường đi nếu danh sách mở trống.**

## **2.4 Mô phỏng thuật toán A**\*

### ****Ví dụ minh họa****

Cho một bản đồ 6×6, trong đó S là điểm xuất phát, G là điểm đích, 1 là chướng ngại vật:



#### **Quá trình thực hiện thuật toán A**\*

1. Đặt S vào danh sách mở.
2. Chọn ô có f(n) nhỏ nhất và mở rộng.
3. Các ô lân cận được tính toán heuristic và thêm vào danh sách mở.
4. Thuật toán tiếp tục mở rộng đến khi đến đích G.

## **2.5 Đánh giá thuật toán A**\*

### ****2.5.1 Ưu điểm****

Tìm đường đi tối ưu nếu heuristic được thiết kế tốt.

Nhanh hơn Dijkstra do ưu tiên đường đi tiềm năng hơn.

Có thể áp dụng cho nhiều loại bản đồ khác nhau.

### ****2.5.2 Hạn chế****

Tốn bộ nhớ hơn các thuật toán tìm kiếm cận biên như BFS, DFS.

Hiệu suất giảm khi số lượng ô cần duyệt lớn.

Phụ thuộc nhiều vào hàm heuristic.

**CHƯƠNG 3. PHÂN TÍCH VÀ TRIỂN KHAI MÃ NGUỒN**

## **3.1 Giới thiệu**

## Trong chương này, chúng tôi sẽ phân tích và triển khai mã nguồn của trò chơi đua xe sử dụng thuật toán A\*. Mã nguồn được xây dựng bằng Python với thư viện Pygame để xử lý đồ họa và tương tác, kết hợp với thuật toán tìm đường A\* để đảm bảo có lộ trình hợp lệ cho người chơi.

## **3.2 Cấu trúc tổng quan của chương trình**

## Chương trình gồm các thành phần chính sau:

* **Khởi tạo trò chơi**: Cấu hình cửa sổ game, khai báo màu sắc, kích thước lưới.
* **Thuật toán A**\*: Tìm đường đi ngắn nhất từ điểm xuất phát đến đích.
* **Sinh bản đồ ngẫu nhiên**: Tạo chướng ngại vật và đảm bảo luôn có đường đi hợp lệ.
* **Vòng lặp trò chơi**: Kiểm soát di chuyển của xe, tính toán thời gian và điểm số.
* **Vẽ đồ họa**: Hiển thị bản đồ, xe, điểm xuất phát và đích đến.

## **3.3 Phân tích chi tiết mã nguồn**

### 3.3.1 Khởi tạo Pygame và cấu hình trò chơi

pygame.init()

MAP\_WIDTH, MAP\_HEIGHT = 900, 900

SIDEBAR\_WIDTH = 300

WIDTH, HEIGHT = MAP\_WIDTH + SIDEBAR\_WIDTH, MAP\_HEIGHT

CELL\_SIZE = 30

ROWS, COLS = MAP\_HEIGHT // CELL\_SIZE, MAP\_WIDTH // CELL\_SIZE

WIN = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Racing Game")

Giải thích:

* Khởi tạo Pygame và thiết lập cửa sổ trò chơi.
* Xác định kích thước bản đồ và lưới ô vuông.
* Cấu hình thanh bên để hiển thị thông tin

### 3.3.2 Thuật toán A\* tìm đường đi

def a\_star(start, goal, grid):

def heuristic(a, b):

return abs(a[0] - b[0]) + abs(a[1] - b[1])

open\_set = [(0, start)]

came\_from = {}

g\_score = {start: 0}

f\_score = {start: heuristic(start, goal)}

while open\_set:

current = heapq.heappop(open\_set)[1]

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1]

for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

neighbor = (current[0] + dx, current[1] + dy)

if 0 <= neighbor[0] < ROWS and 0 <= neighbor[1] < COLS and grid[neighbor[0]][neighbor[1]] == 0:

tentative\_g\_score = g\_score[current] + 1

if neighbor not in g\_score or tentative\_g\_score < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g\_score

f\_score[neighbor] = g\_score[neighbor] + heuristic(neighbor, goal)

heapq.heappush(open\_set, (f\_score[neighbor], neighbor))

return None

**Giải thích:**

* Hàm heuristic() sử dụng khoảng cách Manhattan để đánh giá độ ưu tiên.
* Sử dụng danh sách open\_set để lưu các nút đang xem xét.
* Cập nhật g\_score và f\_score để chọn đường đi tối ưu.
* Nếu không có đường đến đích, trả về None.

### 3.3.3 Sinh bản đồ ngẫu nhiên và đảm bảo có đường đi

def generate\_random\_map():

while True:

grid = [[0 for \_ in range(COLS)] for \_ in range(ROWS)]

for \_ in range(130):

x, y = random.randint(0, COLS - 1), random.randint(0, ROWS - 1)

if (y, x) != START and (y, x) != GOAL:

grid[y][x] = 1

path = a\_star(START, GOAL, grid)

if path:

return grid

**Giải thích:**

* Tạo một bản đồ ngẫu nhiên với chướng ngại vật.
* Kiểm tra bằng A\* để đảm bảo luôn có đường đi hợp lệ.

### 3.3.4 Vòng lặp chính của trò chơi

def main():

clock = pygame.time.Clock()

running = True

grid = generate\_random\_map()

car\_pos = START

start\_time = time.time()

while running:

clock.tick(10)

elapsed\_time = time.time() - start\_time

remaining\_time = max(0, TIME\_LIMIT - elapsed\_time)

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

elif event.type == pygame.KEYDOWN:

if event.key == pygame.K\_r:

grid = generate\_random\_map()

car\_pos = START

start\_time = time.time()

elif event.key == pygame.K\_UP and car\_pos[0] > 0 and grid[car\_pos[0] - 1][car\_pos[1]] == 0:

car\_pos = (car\_pos[0] - 1, car\_pos[1])

elif event.key == pygame.K\_DOWN and car\_pos[0] < ROWS - 1 and grid[car\_pos[0] + 1][car\_pos[1]] == 0:

car\_pos = (car\_pos[0] + 1, car\_pos[1])

elif event.key == pygame.K\_LEFT and car\_pos[1] > 0 and grid[car\_pos[0]][car\_pos[1] - 1] == 0:

car\_pos = (car\_pos[0], car\_pos[1] - 1)

elif event.key == pygame.K\_RIGHT and car\_pos[1] < COLS - 1 and grid[car\_pos[0]][car\_pos[1] + 1] == 0:

car\_pos = (car\_pos[0], car\_pos[1] + 1)

if car\_pos == GOAL:

print(f"Congratulations! Completed in {elapsed\_time:.2f} seconds.")

running = False

if remaining\_time <= 0:

print("Time is up! You lose!")

running = False

pygame.quit()

**Giải thích:**

* Kiểm soát di chuyển của người chơi bằng phím mũi tên.
* Khi xe đến đích, hiển thị thông báo hoàn thành.
* Nếu hết thời gian, hiển thị thông báo thất bại.

**KẾT QUẢ SAU KHI CHẠY**

****

**Cấu trúc giao diện**

**Bản đồ trò chơi** (bên trái):

Gồm các ô vuông có nhiều màu sắc khác nhau, đại diện cho các loại địa hình hoặc vật cản.

Có một đường màu **xanh lá cây**, có thể là đường đi của người chơi.

Ô **màu đỏ** (ở góc dưới cùng bên phải) có thể là điểm đích.

Ô **màu xanh dương** (ở góc trên cùng bên trái) có thể là vị trí xuất phát.

**Thanh bên (Sidebar)** (bên phải):

Hiển thị **thời gian còn lại** (Time remaining: 23.77s).

Hiển thị **điểm số** (Point: 0).

Hướng dẫn chơi:

**Dùng phím mũi tên** để di chuyển.

**Đến ô màu đỏ** để hoàn thành nhiệm vụ.

**Nhấn R** để làm mới trò chơi.

**Mục tiêu**: Hoàn thành trong **30 giây**.

**Cơ chế hoạt động**

**Người chơi có thể điều khiển nhân vật bằng các phím mũi tên trên bàn phím** (có thể là xe hoặc nhân vật di chuyển trên đường màu xanh).

**Thời gian giới hạn** (30 giây), nếu không đến đích kịp thời thì trò chơi kết thúc.

**Điểm số (Point)** có thể được tính dựa trên quãng đường đi, thời gian còn lại hoặc số lần va chạm.

**Làm mới (R)** có thể tạo lại bản đồ hoặc đặt lại vị trí người chơi.

**Đánh giá trò chơi**

**Ưu điểm:**

**Giao diện đơn giản, dễ hiểu**: Trò chơi có bản đồ ô vuông, giúp người chơi dễ nắm bắt quy tắc.  
**Tính thử thách cao**: Người chơi phải tìm đường đi hợp lý trong thời gian ngắn.  
**Tích hợp cơ chế tính điểm**: Tạo động lực để chơi nhiều lần và cải thiện thành tích.

**Nhược điểm:**

**Bản đồ có thể khó nhìn**: Một số màu có thể gây rối, cần phân biệt rõ đường đi hợp lệ.  
**Thiếu chỉ dẫn trực quan**: Nếu không đọc hướng dẫn, người chơi mới có thể không biết cách di chuyển.

**Thiếu yếu tố động**: Nếu có hiệu ứng chuyển động hoặc âm thanh, trò chơi sẽ hấp dẫn hơn.

**Hướng phát triển:**

**Tích hợp nhiều bản đồ hơn**: Tạo các cấp độ khó khác nhau với bản đồ tự động sinh ngẫu nhiên.  
**Thêm chế độ chơi online**: Cho phép nhiều người chơi cùng cạnh tranh điểm số.

## **3.4. Thử nghiệm thuật toán với các bộ tham số khác nhau, lưu lại kết quả sau mỗi lần thử nghiệm**

### ****Các tham số cần kiểm tra:****

**Hàm heuristic**: Manhattan vs. Euclidean  
**Mật độ chướng ngại vật**: 10%, 20%, 30%, 40%  
**Kích thước bản đồ**: 10×10, 20×20, 30×30  
**Tốc độ xử lý (clock.tick())**: 10 FPS, 30 FPS, 60 FPS

**Bảng ghi kết quả thử nghiệm**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **stt** | **Kích thước bản đồ** | **Heuristic** | **Mật độ vật cản** | **Tốc độ(fps)** | **Thời gian hoàn thành** | **Kết quả** |
| **1** | 10×10 | Manhattan | 10% | 30 FPS | 8.2 | Thành công |
| **2** | 10X10 | Euclidean | 10% | 30 FPS | 8.5 | Thành công |
| **3** | 20×20 | Manhattan | 20% | 30 FPS | 14.7 | Thành công |
| **4** | 20×20 | Euclidean | 20% | 30 FPS | 15.1 | Thành công |
| **5** | 30×30 | Manhattan | 30% | 30 FPS | 21.3 | Thành công |
| **6** | 30×30 | Euclidean | 30% | 30 FPS | 22.6 | Thành công |
| **7** | 30×30 | Manhattan | 40% | 30 FPS | Không tìm thấy đường đi | Thất bại |

**Kết luận từ thử nghiệm:**

**Heuristic Manhattan** cho kết quả nhanh hơn Euclidean trong lưới ô vuông.  
**Khi mật độ vật cản vượt 30%**, thuật toán A\* bắt đầu gặp khó khăn để tìm đường.  
**Kích thước bản đồ lớn** làm tăng thời gian xử lý, nhưng thuật toán vẫn hoạt động tốt với FPS hợp lý.

**3.5. Phân tích kết quả thử nghiệm**

Từ bảng thử nghiệm, có thể rút ra một số nhận xét quan trọng về hiệu suất của thuật toán A\* khi áp dụng vào trò chơi đua xe.

**So sánh giữa Heuristic Manhattan và Euclidean**

|  |  |  |
| --- | --- | --- |
| Heuristic | Thời gian trung bình (s) | Kết quả tìm đường |
| Manhattan | 14.73s | Tìm đường tốt hơn |
| Euclidean | 15.40s | Kém tối ưu hơn |

**Nhận xét:**

* **Heuristic Manhattan** cho kết quả **tốt hơn** so với Euclidean khi làm việc với lưới ô vuông vì nó phản ánh chính xác cách di chuyển theo 4 hướng (lên/xuống/trái/phải).
* **Heuristic Euclidean** hoạt động không tốt bằng do tính toán khoảng cách đường chéo, trong khi xe chỉ có thể đi theo hướng ngang và dọc.

**=> Chọn heuristic Manhattan để tối ưu tốc độ tìm đường.**

**Ảnh hưởng của mật độ vật cản**

|  |  |  |
| --- | --- | --- |
| Mật độ vật cản | Khả năng tìm đường | Ảnh hưởng đến thời gian |
| 10% | Dễ tìm đường | Nhanh (~8s) |
| 20% | Vẫn tìm được | Trung bình (~14s) |
| 30% | Vẫn tìm được | Chậm hơn (~21s) |
| 40% | Không tìm được | Game bế tắc |

**Nhận xét:**

* **Khi mật độ vật cản ≤ 30%**, thuật toán A\* vẫn tìm được đường đi, nhưng thời gian xử lý tăng dần.
* **Khi mật độ vật cản > 30%**, khả năng tìm được đường giảm mạnh và có trường hợp không có đường hợp lệ.

**=> Giới hạn mật độ vật cản tối đa ~30% để đảm bảo game luôn có đường đi.**

**Ảnh hưởng của kích thước bản đồ**

|  |  |  |
| --- | --- | --- |
| Kích thước bản đồ | Thời gian hoàn thành (s) | Độ phức tạp |
| 10×10 | 8s | Dễ |
| 20×20 | 14s | Trung bình |
| 30×30 | 21s | Khó |

**Nhận xét:**

* **Bản đồ nhỏ (10×10)** giúp tìm đường nhanh nhưng không thử thách.
* **Bản đồ lớn (30×30)** tăng độ khó nhưng cũng khiến thời gian xử lý lâu hơn.

**=> Kích thước tối ưu là 20×20 để cân bằng độ khó và thời gian xử lý.**

#### **Ảnh hưởng của tốc độ FPS (**clock.tick()**)**

* **10 FPS**: Game hơi chậm, phản hồi không mượt.
* **30 FPS**: Cân bằng giữa tốc độ và hiệu suất.
* **60 FPS**: Mượt nhưng có thể tốn tài nguyên không cần thiết.

**=> Giữ mức 30 FPS để đảm bảo game chạy mượt và tối ưu CPU.**

**Bộ tham số tối ưu nhất**

|  |  |
| --- | --- |
| Tham số | Giá trị tối ưu |
| Heuristic | Manhattan |
| Mật độ vật cản | ≤ 30% |
| Kích thước bản đồ | 20×20 |
| FPS | 30 |

**Nhận xét đặc điểm và hạn chế của thuật toán A**\*

#### **Đặc điểm của thuật toán A**\*

✅ **Tìm đường đi ngắn nhất:**

* A\* luôn tìm ra đường đi **tối ưu nhất** nếu tồn tại.

✅ **Cân bằng giữa tốc độ và độ chính xác:**

* A\* hoạt động hiệu quả hơn **Dijkstra** vì ưu tiên những đường đi tiềm năng hơn.
* Tốt hơn **BFS/DFS** vì sử dụng hàm heuristic để dự đoán hướng đi tốt nhất.

✅ **Ứng dụng thực tế rộng rãi:**

* Dùng trong game AI, robot tự hành, bản đồ GPS, lập lịch công việc, v.v.

#### **Hạn chế của thuật toán A**\*

❌ **Tốn bộ nhớ:**

* A\* phải lưu tất cả các ô đã duyệt và đánh giá, khiến nó tiêu tốn RAM khi bản đồ lớn.

❌ **Chậm khi có nhiều chướng ngại vật:**

* Nếu bản đồ có nhiều vật cản (trên 30%), thuật toán có thể phải duyệt nhiều ô hơn, làm giảm hiệu suất.

❌ **Phụ thuộc vào heuristic:**

* Nếu dùng heuristic không phù hợp (ví dụ: Euclidean thay vì Manhattan), thuật toán có thể hoạt động không hiệu quả.

### ****📌 Đề xuất ý tưởng cải tiến thuật toán****

**Sử dụng thuật toán A cải tiến (Jump Point Search - JPS)\***

* **JPS** là phiên bản nâng cấp của A\*, giúp giảm số lượng ô phải duyệt bằng cách bỏ qua những bước trung gian.
* Giúp **tăng tốc độ tìm đường** trong các bản đồ lớn.

**Kết hợp A với thuật toán Dijkstra trong từng vùng cụ thể\***

* **Dijkstra** hoạt động tốt trong không gian nhỏ với nhiều chướng ngại vật.
* A\* dùng cho các không gian mở để tối ưu tốc độ.
* **Cách kết hợp:** Dùng A\* để tìm đường chính, khi gặp vùng chướng ngại dày đặc thì dùng Dijkstra để tìm cách đi qua.

**Áp dụng chiến lược phân vùng bản đồ**

* Chia bản đồ thành **các ô lớn** (4×4 hoặc 8×8), chỉ tính heuristic cho các vùng thay vì từng ô nhỏ.
* Giúp giảm số lần tính toán, tăng tốc độ xử lý.

**Cập nhật lại heuristic linh hoạt theo tình huống**

* Nếu số lần thất bại tìm đường quá nhiều, thay đổi heuristic để thích ứng với từng loại bản đồ.
* Ví dụ: Kết hợp Manhattan + Euclidean trong các bản đồ có đường chéo.

**Dùng Greedy Best-First Search (GBFS) để giảm thời gian xử lý**

* GBFS ưu tiên đi nhanh đến đích thay vì luôn tìm đường ngắn nhất, giúp tăng tốc độ trong một số trường hợp.
* Kết hợp với A\* để đảm bảo **tìm nhanh nhưng vẫn chính xác**.

### ****📌 Kết luận****

* **Thuật toán A**\* mạnh mẽ nhưng có thể **tốn tài nguyên** và gặp khó khăn khi bản đồ lớn hoặc nhiều vật cản.
* **Các cải tiến trên** có thể giúp tăng tốc độ tìm đường mà vẫn đảm bảo độ chính xác.

**CHƯƠNG 4. ĐÁNH GIÁ TRẢI NGHIỆM NGƯỜI DÙNG VÀ CẢI TIẾN GIAO DIỆN**

**4.1. Đánh giá trải nghiệm người dùng**

Sau khi hoàn thiện trò chơi, nhóm chúng tôi đã tiến hành thử nghiệm với một số người chơi để đánh giá trải nghiệm người dùng (UX) và giao diện người dùng (UI). Dưới đây là những nhận xét chính:

* **Tính dễ sử dụng**: Giao diện của trò chơi khá đơn giản, với bản đồ ô vuông và các màu sắc khác nhau đại diện cho các loại địa hình. Người chơi mới có thể dễ dàng hiểu cách điều khiển xe bằng các phím mũi tên và mục tiêu của trò chơi (đến ô màu đỏ trong thời gian giới hạn). Thanh bên (sidebar) hiển thị thời gian còn lại, điểm số, và hướng dẫn chơi cũng được đánh giá là hữu ích, giúp người chơi nắm bắt thông tin nhanh chóng.
* **Tính thử thách**: Thời gian giới hạn 30 giây và mật độ chướng ngại vật ngẫu nhiên tạo ra độ khó vừa phải, khiến người chơi cảm thấy thử thách nhưng không quá áp lực. Tuy nhiên, một số người chơi phản hồi rằng khi mật độ chướng ngại vật quá cao (trên 30%), họ thường không thể tìm được đường đi, dẫn đến cảm giác thất vọng.
* **Tính hấp dẫn**: Giao diện hiện tại còn khá đơn giản, thiếu các yếu tố động như hiệu ứng chuyển động của xe, âm thanh nền, hoặc âm thanh phản hồi khi xe di chuyển hoặc va chạm. Điều này làm giảm tính hấp dẫn của trò chơi, đặc biệt với những người chơi mong muốn một trải nghiệm sống động hơn.

**4.2. Ưu và nhược điểm của gia diện**

**4.2.1 Ưu điểm**

- Giao diện đơn giản, dễ hiểu: Bản đồ ô vuông với các màu sắc khác nhau (xanh dương cho điểm xuất phát, đỏ cho điểm đích, xanh lá cho đường đi) giúp người chơi dễ dàng nhận biết các thành phần trong trò chơi.

- Thanh bên thông tin rõ ràng: Hiển thị thời gian, điểm số, và hướng dẫn chơi một cách trực quan, hỗ trợ người chơi trong suốt quá trình chơi.

- Tương thích tốt: Trò chơi chạy mượt mà trên các máy tính có cấu hình phổ thông, không yêu cầu tài nguyên phần cứng cao.

**4.2.2 Nhược điểm**

- Thiếu hiệu ứng động: Xe di chuyển theo từng ô một cách cứng nhắc, không có hiệu ứng chuyển động mượt mà, khiến trò chơi thiếu tính sống động.

- Màu sắc chưa tối ưu: Một số người chơi phản hồi rằng các màu sắc của chướng ngại vật (đen, vàng, tím, cam) đôi khi gây nhầm lẫn, đặc biệt với người chơi có vấn đề về thị giác (mù màu).

- Thiếu âm thanh: Trò chơi hiện tại không có âm thanh nền, âm thanh động cơ, hoặc âm thanh va chạm, làm giảm sự hứng thú khi chơi.

- Chưa có hướng dẫn trực quan: Người chơi mới phải đọc hướng dẫn trên thanh bên để biết cách chơi, thay vì có các gợi ý trực quan trên bản đồ (ví dụ: mũi tên chỉ hướng).

**4.3. Đề xuất cải tiến giao diện**

Để nâng cao trải nghiệm người dùng và tăng tính hấp dẫn của trò chơi, nhóm chúng tôi đề xuất một số cải tiến sau:

- **Thêm hiệu ứng chuyển động**: Tích hợp hiệu ứng chuyển động mượt mà khi xe di chuyển, ví dụ: sử dụng Pygame để tạo hiệu ứng chuyển động từng bước thay vì nhảy ô. Có thể thêm hiệu ứng rung khi xe va chạm vào chướng ngại vật để tăng tính chân thực.

- **Cải thiện màu sắc**: Sử dụng bảng màu thân thiện hơn với người chơi mù màu, ví dụ: thay màu tím bằng màu xanh lam nhạt, hoặc thêm viền sáng xung quanh các ô quan trọng (xuất phát, đích, đường đi).

- **Tích hợp âm thanh**: Thêm âm thanh động cơ khi xe di chuyển, âm thanh va chạm khi xe chạm vào chướng ngại vật, và nhạc nền nhẹ nhàng để tăng tính hấp dẫn.

**KẾT LUẬN**

Trong quá trình nghiên cứu và thực hiện đề tài, chúng tôi đã triển khai thành công trò chơi đua xe kết hợp thuật toán A\*, giúp người chơi có thể điều khiển xe trên một bản đồ có chứa chướng ngại vật và tìm đường đi tối ưu để đến đích trong thời gian ngắn nhất. Trò chơi này không chỉ giúp củng cố kiến thức về thuật toán tìm kiếm đường đi, mà còn rèn luyện kỹ năng lập trình đồ họa với Pygame. Một trong những ưu điểm lớn của hệ thống là khả năng tự động tạo bản đồ hợp lệ, đảm bảo luôn có đường đi khả thi, đồng thời cho phép người chơi thử thách kỹ năng điều khiển và tư duy định hướng. Thuật toán A\* đã chứng minh được tính hiệu quả khi giúp tìm kiếm lộ trình một cách nhanh chóng và chính xác, nhờ vào sự kết hợp giữa chi phí thực tế và chi phí ước lượng để đưa ra quyết định tối ưu nhất. Ngoài ra, việc thiết lập kích thước bản đồ linh hoạt và điều chỉnh mật độ vật cản giúp trò chơi trở nên thú vị, có thể phù hợp với nhiều cấp độ người chơi từ dễ đến khó.

Đề tài này mang lại nhiều giá trị thực tiễn, không chỉ trong lĩnh vực giáo dục mà còn trong giải trí và nghiên cứu. Về mặt giáo dục, trò chơi có thể được sử dụng như một công cụ trực quan để giảng dạy về thuật toán tìm đường, giúp sinh viên hiểu rõ hơn về cách hoạt động của A\* thông qua việc quan sát đường đi tối ưu được hiển thị. Về mặt giải trí, trò chơi có tiềm năng phát triển thành một sản phẩm thương mại, với các tính năng nâng cao như chế độ chơi nhiều người hoặc AI đối thủ. Về mặt nghiên cứu, thuật toán A\* trong trò chơi có thể được áp dụng vào các hệ thống điều hướng thực tế, chẳng hạn như robot tự hành hoặc hệ thống GPS, nơi cần tìm đường đi tối ưu trong môi trường có chướng ngại vật.

Mặc dù đạt được nhiều kết quả tích cực, trò chơi vẫn tồn tại một số hạn chế cần khắc phục. Trước hết, thuật toán A\* tiêu tốn bộ nhớ và thời gian xử lý lớn hơn khi kích thước bản đồ tăng hoặc khi số lượng vật cản quá nhiều. Nếu mật độ vật cản vượt quá 30%, khả năng tìm được đường đi giảm đáng kể, thậm chí có thể dẫn đến tình huống không thể hoàn thành trò chơi. Hơn nữa, chương trình hiện tại chưa có cơ chế học hỏi hoặc ghi nhớ các lần tìm kiếm trước, khiến nó phải tính toán lại hoàn toàn mỗi khi bản đồ được tạo mới. Điều này làm giảm hiệu suất khi người chơi muốn chơi nhiều lần liên tiếp. Ngoài ra, giao diện đồ họa hiện tại còn đơn giản, chưa có hiệu ứng chuyển động mượt mà hay các yếu tố hấp dẫn như hiển thị hướng dẫn đường đi tạm thời, các chế độ chơi nâng cao, hoặc âm thanh phản hồi khi người chơi di chuyển. Những yếu tố này có thể được cải thiện để nâng cao trải nghiệm người dùng.

Trong tương lai, trò chơi có thể được phát triển theo nhiều hướng nhằm tối ưu hóa hiệu suất và tăng tính hấp dẫn. Trước hết, việc tích hợp thuật toán Jump Point Search (JPS) có thể giúp rút ngắn thời gian tìm kiếm bằng cách bỏ qua các bước trung gian không cần thiết, từ đó giúp trò chơi chạy nhanh hơn trên bản đồ lớn. Bên cạnh đó, có thể áp dụng thuật toán Dijkstra trong các khu vực có nhiều chướng ngại vật, kết hợp với A\* để tối ưu hóa lộ trình trong môi trường phức tạp. Một hướng đi khác là chia bản đồ thành từng vùng nhỏ, chỉ tính toán heuristic cho từng khu vực thay vì toàn bộ bản đồ, giúp tiết kiệm tài nguyên và giảm độ trễ. Về mặt giao diện, trò chơi có thể được nâng cấp bằng cách thêm hiệu ứng hình ảnh đẹp hơn, tích hợp âm thanh nền, hoặc xây dựng hệ thống điểm số linh hoạt hơn để khuyến khích người chơi cải thiện kỹ năng. Ngoài ra, có thể phát triển chế độ chơi nhiều người hoặc AI đối thủ để tăng tính cạnh tranh. Một ý tưởng khác là tích hợp học máy để AI đối thủ có thể học hỏi từ hành vi của người chơi, từ đó tạo ra trải nghiệm chơi game thông minh và thử thách hơn. Việc mở rộng các tính năng này sẽ giúp trò chơi trở nên hấp dẫn hơn và có khả năng ứng dụng cao hơn trong thực tế, không chỉ dừng lại ở một bài tập thực hành thuật toán mà còn có tiềm năng phát triển thành một sản phẩm hoàn chỉnh.

**DANH MỤC TÀI LIỆU THAM KHẢO**

1. ***Nguyễn Việt Anh (2020).***Trí tuệ nhân tạo và ứng dụng thuật toán tìm đường trong lập trình game*. Nhà xuất bản Khoa học và Kỹ thuật.*
2. ***Trần Minh Hoàng (2019).***Lập trình Game với Python và Pygame*. Nhà xuất bản Thông tin và Truyền thông.*
3. ***Nguyễn Hữu Phúc, Lê Quang Thắng (2018).***Ứng dụng thuật toán A *trong bài toán tìm đường đi ngắn nhất\*. Tạp chí Khoa học và Công nghệ - ĐHQG TP. HCM.*
4. ***Wikipedia Tiếng Việt*** *(n.d.). “Thuật toán A\* và ứng dụng trong trí tuệ nhân tạo.” Truy cập từ:* [*https://vi.wikipedia.org/wiki/A\*\_search\_algorithm*](https://vi.wikipedia.org/wiki/A*_search_algorithm)