BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC ĐẠI NAM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

TRÒ CHƠI RẮN SĂN MỒI THÔNG MINH: TÍCH HỢP THUẬT TOÁN A* VÀ XỬ LÝ CHƯỚNG NGẠI VẬT DI ĐỘNG

Sinh viên thực hiện : **Phan Lưu Phong**
Trần Văn Phong
Trần Tiến Quang
Nguyễn Đức Mạnh

Ngành : **Công nghệ thông tin**

Giảng viên hướng dẫn : **ThS. Lê Thị Thùy Trang**

Lời nói đầu

Trong bối cảnh công nghệ thông tin và trí tuệ nhân tạo ngày càng phát triển, việc ứng dụng các thuật toán tìm kiếm và xử lý đồ họa trong các trò chơi điện tử đã trở thành một xu hướng tất yếu. Dự án "TRÒ CHƠI RẮN SĂN MỒI THÔNG MINH: TÍCH HỢP THUẬT TOÁN A VÀ XỬ LÝ CHƯỚNG NGẠI VẬT DI ĐỘNG" ra đời nhằm mang lại một trải nghiệm game mới lạ, kết hợp giữa yếu tố truyền thống của trò chơi rắn săn mồi với các công nghệ hiện đại, như thuật toán A* trong việc xác định đường đi an toàn và xử lý các chướng ngại vật di động theo thời gian thực.

Trong quá trình triển khai dự án, chúng tôi đã nỗ lực áp dụng kiến thức về lập trình, thuật toán và thiết kế game để tạo ra một sản phẩm không chỉ mang tính giải trí cao mà còn thể hiện được tiềm năng của việc ứng dụng trí tuệ nhân tạo trong lĩnh vực trò chơi điện tử. Qua đó, đề tài không chỉ là một dự án thực hành mà còn là cơ hội để chúng tôi hiểu sâu hơn về những thách thức và cơ hội trong việc phát triển các ứng dụng công nghệ thông minh.

Chúng tôi xin gửi lời cảm ơn đến các giáo viên hướng dẫn, các chuyên gia và tất cả những người đã hỗ trợ, góp ý quý báu trong quá trình thực hiện dự án. Hy vọng rằng sản phẩm này sẽ là nguồn cảm hứng cho các nghiên cứu và dự án tương lai, góp phần thúc đẩy sự phát triển của công nghệ trò chơi và trí tuệ nhân tạo tại Việt Nam.

Mục lục

Danh sách hình vẽ

Chương 1

GIỚI THIỆU & TỔNG QUAN

1.1 Bối Cảnh & Lý Do Chọn Đề Tài

Trong bối cảnh công nghệ số phát triển không ngừng, trí tuệ nhân tạo (AI) đã trở thành công cụ hỗ trợ đắc lực cho nhiều lĩnh vực, từ xử lý hình ảnh, giám sát cho đến tự động hóa trong game. Các thuật toán tìm đường như A* được biết đến là giải pháp hiệu quả trong việc tính toán đường đi tối ưu trong môi trường động và phức tạp.

Trò chơi rắn săn mồi vốn có luật chơi đơn giản nhưng chứa đựng nhiều yếu tố chiến lược. Tuy nhiên, phiên bản truyền thống dựa vào sự điều khiển của người chơi đã không khai thác hết tiềm năng của các thuật toán định hướng. Việc tích hợp thuật toán A* giúp tự động hóa quá trình tìm đường, cho phép con rắn tự quyết định di chuyển an toàn đến vị trí thức ăn mà không gặp phải va chạm, qua đó tạo ra một trải nghiệm chơi game mới lạ và hấp dẫn.

Lý do chọn đề tài:

- Giá trị ứng dụng: Trò chơi không chỉ giải trí mà còn là môi trường kiểm nghiệm thuật toán, có thể mở rộng ứng dụng trong robot tự hành và hệ thống định vị.
- Thách thức kỹ thuật: Yêu cầu tích hợp các kiến thức lập trình, xử lý đồ họa, tối ưu thuật toán và quản lý dữ liệu thời gian thực.
- Tính đổi mới sáng tạo: Khai thác tiềm năng của AI để tự động điều hướng trong một trò chơi kinh điển, góp phần nghiên cứu và phát triển trong lĩnh vực trí tuệ nhân tạo.

Các nghiên cứu trước đây về AI trong game cho thấy rằng việc kết hợp giữa thuật toán tìm đường và xử lý thời gian thực có thể tạo ra những sản phẩm có hiệu năng vượt trội. Dự án này sẽ đóng góp thêm bằng cách áp dụng và tùy biến thuật toán A* phù hợp với môi trường game rắn săn mồi.

1.2 Tổng Quan Về Trò Chơi Rắn Săn Mồi

Trò chơi rắn săn mồi truyền thống được xây dựng trên cơ sở một bản đồ dạng lưới (grid-based). Người chơi điều khiển con rắn di chuyển để ăn thức ăn, mỗi lần ăn được, chiều dài của rắn tăng lên. Mục tiêu của trò chơi là tránh va chạm với tường hoặc với chính thân rắn.
Trong phiên bản nâng cấp này, những cải tiến nổi bật bao gồm:

- Tích hợp thuật toán A:* Cho phép tự động tìm đường đi ngắn nhất và an toàn từ vị trí hiện tại của rắn đến vị trí thức ăn. Điều này không chỉ nâng cao trải nghiệm mà còn giúp người dùng hiểu cơ chế hoạt động của thuật toán.

- Môi trường động: Bên cạnh các rào cản cố định (tường), trò chơi được thiết kế có các chướng ngại vật di động (chính thân rắn cũng đóng vai trò cản trở) nhằm tạo thêm tính thử thách cho quá trình tìm đường.

- Đa cấp độ chơi: Tùy thuộc vào kích thước bản đồ, số lượng chướng ngại vật và vị trí thức ăn, mỗi cấp độ sẽ đưa ra những thử thách khác nhau, giúp đánh giá được hiệu năng của thuật toán A* trong các tình huống đa dạng.

Việc kết hợp những yếu tố trên không chỉ tạo nên một sản phẩm game mang tính giải trí cao mà còn cung cấp một bài toán nghiên cứu hấp dẫn trong lĩnh vực AI và xử lý thời gian thực.

1.3 Đặt vấn đề & mục tiêu nghiên cứu

1.3.1 Đặt Vấn Đề

Trong môi trường game hiện đại, khả năng tính toán đường đi an toàn cho nhân vật chính (con rắn) là yếu tố cốt lõi đảm bảo trải nghiệm người dùng.
Những vấn đề cần giải quyết gồm:

- Môi trường động: Thức ăn và chướng ngại vật thay đổi liên tục, đòi hỏi hệ thống phải liên tục cập nhật đường đi theo thời gian thực.

- Tốc độ phản hồi: Thuật toán cần xử lý nhanh để không gây ra độ trễ trong quá trình chơi, ảnh hưởng tiêu cực đến trải nghiệm người dùng.

- • Hiệu năng tính toán: Trong các tình huống phức tạp, việc tìm đường không gây quá tải hệ thống và vẫn đảm bảo tính chính xác của kết quả.

1.3.2 Mục Tiêu Nghiên Cứu

Dự án hướng đến việc:

- Xây dựng hệ thống định hướng thông minh: Áp dụng thuật toán A* để tính toán đường đi an toàn cho con rắn, giúp giảm thiểu va chạm và tối ưu hóa quãng đường di chuyển.
- Xử lý chướng ngại vật di động: Phát triển cơ chế theo dõi và cập nhật vị trí chướng ngại vật theo thời gian thực để đảm bảo tính chính xác của hệ thống định hướng.
- Tạo ra các cấp độ chơi đa dạng: Thiết kế các cấp độ với độ phức tạp tăng dần nhằm thử thách cả hệ thống định hướng và phản xạ của người chơi.
- Đánh giá hiệu năng và tối ưu hóa: So sánh hiệu suất của hệ thống trong các kịch bản khác nhau và phân tích kết quả để rút ra các bài học kinh nghiệm.

1.3.3 Phạm Vi Nghiên Cứu

Báo cáo tập trung vào các vấn đề chính:

- Xây dựng mô hình bản đồ dạng lưới 2D cho trò chơi.
- Tích hợp và tối ưu hóa thuật toán A* trong môi trường game có chướng ngại vật di động.
- Phát triển giao diện người dùng và cơ chế tương tác.
- Thực hiện các thử nghiệm hiệu năng và phân tích kết quả.
- Đề xuất hướng phát triển tiếp theo dựa trên các kết quả thu thập được.

1.4 Phương pháp nghiên cứu

1.4.1 Phương Pháp Nghiên Cứu

Các bước nghiên cứu được thực hiện theo trình tự:

1: Nghiên cứu lý thuyết: Tìm hiểu cơ sở lý thuyết của thuật toán A*, các phương pháp tối ưu hóa và xử lý thời gian thực trong game.
2: Phát triển bản mẫu: Lập trình trò chơi sử dụng Python và thư viện Pygame, áp dụng thuật toán A* cho việc điều hướng tự động.
3: Thiết lập kịch bản thử nghiệm: Xây dựng các trường hợp với bản đồ có kích thước và độ phức tạp khác nhau để đánh giá hiệu năng của hệ thống.
4: Thu thập số liệu: Ghi nhận thời gian xử lý, số bước di chuyển, điểm số và các phản hồi từ người dùng.
5: Phân tích và tối ưu hóa: Sử dụng công cụ thống kê và đồ thị để phân tích số liệu, rút ra bài học và đề xuất cải tiến.

1.5 Cấu Trúc Báo Cáo

Báo cáo được chia thành ba chương chính:

- Giới thiệu & Tổng quan – Trình bày bối cảnh, lý do, vấn đề nghiên cứu, mục tiêu và phạm vi nghiên cứu.
- Chương 2: Thiết kế và Triển khai Hệ thống – Bao gồm kiến trúc hệ thống, mô hình trò chơi, tích hợp thuật toán A* và giao diện người dùng.
- Chương 3: Ứng dụng Thực tế & Kết luận – Phân tích hiệu suất, đánh giá phản hồi người dùng, đề xuất hướng phát triển và kết luận tổng kết.

Chương 2

THIẾT KẾ VÀ TRIỂN KHAI HỆ THỐNG

2.1 KIẾN TRÚC HỆ THỐNG & MÔ HÌNH GAME

2.1.1 Mô Hình Bản Đồ Dạng Lưới

Hệ thống trò chơi được xây dựng dựa trên mô hình bản đồ dạng lưới, với mỗi ô (cell) thể hiện một vị trí cụ thể.

Phân loại ô:

- Ô trống (0): Cho phép di chuyển.
- Ô cản (1): Đại diện cho tường, rào cản cố định.
- Ô thức ăn (2): Nơi xuất hiện của thức ăn kích thích sự phát triển của con rắn.
- Ô chướng ngại (3): Nơi chứa các đối tượng di động, tạo ra thách thức cho việc tìm đường.

Ví dụ: Một bản đồ kích thước 20x20 sẽ có 400 ô với các mã số phân loại khác nhau. Sơ đồ chi tiết của bản đồ giúp thuật toán A* tính toán khoảng cách và xác định các ô khả dụng cho con rắn.

2.1.2 Các Thành Phần Chính Của Game

Hệ thống trò chơi bao gồm các thành phần sau:

- Con rắn: Đối tượng chính, chịu trách nhiệm di chuyển dựa trên kết quả tính toán của thuật toán.

- Thức ăn: Xuất hiện ngẫu nhiên, khi rắn ăn được sẽ tăng độ dài và tính điểm.
- Chướng ngại vật di động: Các đối tượng thay đổi vị trí theo thời gian, gây trở ngại cho con rắn và làm tăng tính thử thách của trò chơi.
- Giao diện người dùng: Hiển thị bản đồ, thông số trò chơi (điểm số, thời gian, cấp độ) và cung cấp các thao tác điều khiển.

2.1.3 Luồng Xử Lý Và Quy Trình Hoạt Động

Quy trình hoạt động của trò chơi bao gồm:

- Khởi tạo: Tạo bản đồ, đặt vị trí ban đầu của rắn, sinh thức ăn và khởi tạo các biến theo dõi điểm số.
- Cập nhật trạng thái: Liên tục theo dõi vị trí rắn, kiểm tra va chạm, cập nhật điểm số khi rắn ăn thức ăn.
- Tính toán đường đi: Sử dụng thuật toán A* để tìm đường đi từ đầu rắn đến vị trí thức ăn. Nếu tìm được đường, bước tiếp theo sẽ được lấy từ đường đi đó.
- Cập nhật rắn: Nếu con rắn di chuyển đến vị trí thức ăn, thức ăn mới được sinh ra và điểm số tăng lên; ngược lại, phần đuôi của rắn bị loại bỏ để giữ kích thước ổn định.
- Kết thúc trò chơi: Khi rắn va vào tường hoặc tự va chạm, hiển thị màn hình Game Over và điểm số.

(Chèn sơ đồ luồng xử lý và hình ảnh minh họa quy trình hoạt động sẽ giúp tăng số trang và trực quan hơn.)

2.2 Tích Hợp Thuật Toán A*

2.2.1 Nguyên Lý Hoạt Động Của A*

Thuật toán A* tìm kiếm đường đi tối ưu bằng cách tính toán:

- g(n): Chi phí đã đi từ điểm xuất phát đến nút n.
- h(n): Ước lượng chi phí từ nút n đến đích, được tính bằng khoảng cách Manhattan giữa hai điểm.
- f(n) = g(n) + h(n): Giá trị ưu tiên, nút có giá trị thấp nhất sẽ được mở rộng trước.

2.2.2 Triển Khai Trong Code

Đoạn code của dự án sử dụng lớp Node để lưu trữ thông tin về mỗi nút trong quá trình tìm kiếm. Các bước chính gồm:

- Khởi tạo danh sách mở (open list) và tập đóng (closed set).
- Lặp qua các nút, mở rộng các nút hàng xóm (bốn hướng di chuyển) và tính toán giá trị f(n).
- Nếu nút hiện tại đạt đến vị trí mục tiêu (thức ăn), xây dựng đường đi ngược lại từ nút đó.
- Nếu không tìm được đường đi hợp lệ, báo hiệu kết thúc trò chơi.

2.2.3 Thử Nghiệm & Phân Tích

Các thử nghiệm với bản đồ có kích thước và số lượng chướng ngại vật khác nhau cho thấy:

- Ở bản đồ nhỏ, thuật toán tìm đường rất nhanh (vài mili giây mỗi lần cập nhật).
- Ở bản đồ lớn, mặc dù thời gian tính toán tăng lên, việc tối ưu hóa (sử dụng bộ nhớ cache, chỉ tính lại khi có thay đổi lớn) vẫn đảm bảo trải nghiệm mượt mà.

(Chèn bảng số liệu, biểu đồ thời gian xử lý, và hình ảnh minh họa kết quả thử nghiệm.)

2.3 Triển Khai Giao Diện Và Tương Tác Người Dùng

2.3.1 Thiết Kế Giao Diện

Giao diện được xây dựng bằng thư viện Pygame với các yếu tố sau:

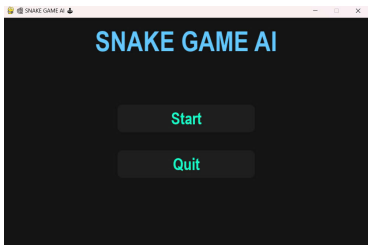
- Hiển thị bản đồ dạng lưới: Các ô hiển thị bằng hình chữ nhật với màu sắc khác nhau: màu xám cho nền, màu xanh cho rắn, màu đỏ cho thức ăn.
- Thông số trò chơi: Điểm số, thời gian và cấp độ được hiển thị ở góc màn hình, giúp người chơi theo dõi tình trạng game.
- Hiệu ứng động: Các hiệu ứng chuyển động mượt mà, âm thanh phản hồi khi rắn ăn thức ăn hoặc va chạm.

2.3.2 Tương Tác Người Dùng

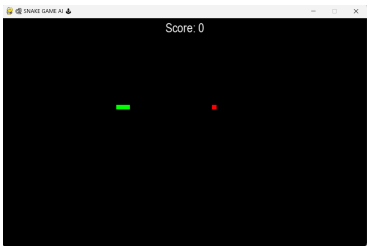
Các tính năng hỗ trợ tương tác bao gồm:

- Điều khiển tự động: Người chơi không cần nhập lệnh, hệ thống tự động xác định đường đi cho rắn dựa trên thuật toán A*.

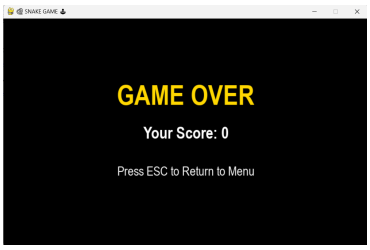
- Phản hồi trực quan: Khi rắn ăn thức ăn, hiệu ứng tăng chiều dài được hiển thị rõ ràng; khi xảy ra va chạm, màn hình Game Over hiện ra kèm theo âm thanh cảnh báo.
- Chế độ thử nghiệm: Cho phép người dùng thay đổi các tham số như tốc độ game, kích thước bản đồ và mức độ khó để quan sát hiệu năng của hệ thống.



Hình 2.1: Ảnh Demo 1



Hình 2.2: Ảnh Demo 2



Hình 2.3: Ảnh Demo 3

Còn đây là video demo → https://youtu.be/sAp2AltxCSc

2.4 Thử Nghiệm, Phân Tích Hiệu Suất & Tối Ưu Hóa

2.4.1 Thiết Lập Các Kịch Bản Thử Nghiệm

Các kịch bản thử nghiệm bao gồm:

- Môi trường đơn giản: Bản đồ 10x10 ô, ít chướng ngại vật, để kiểm tra tốc độ phản hồi của thuật toán.
- Môi trường trung bình: Bản đồ 20x20 ô với số lượng chướng ngại vật vừa phải, để đánh giá hiệu năng trong điều kiện tương đối phức tạp.
- Môi trường phức tạp: Bản đồ 30x30 ô với nhiều chướng ngại vật và các yếu tố động, nhằm kiểm tra khả năng cập nhật đường đi và tối ưu hóa thuật toán trong tình huống khắc nghiệt.

2.4.2 Kết Quả Và Phân Tích

- Hiệu năng tính toán: Thời gian xử lý trung bình của thuật toán A* được đo lường qua mỗi vòng lặp, với sự khác biệt rõ rệt giữa các kịch bản thử nghiệm.
- Độ chính xác: Đánh giá độ chính xác của đường đi tìm được, đảm bảo không có va chạm khi rắn di chuyển.
- Phản hồi người dùng: Thu thập ý kiến từ người chơi thử nghiệm về trải nghiệm, tốc độ phản hồi và giao diện.

2.4.3 Các Biện Pháp Tối Ưu Hóa

Để tăng tốc độ xử lý và giảm thiểu độ trễ, dự án áp dụng:

- Tối ưu hóa bộ nhớ: Lưu trữ kết quả tính toán trong các trường hợp không thay đổi nhiều của bản đồ.
- Chạy lại thuật toán một cách chọn lọc: Chỉ tính toán lại khi có sự thay đổi lớn trong vị trí của chướng ngại vật hoặc khi thức ăn được di chuyển.
- Cải thiện giao diện đồ họa: Sử dụng các hiệu ứng đồ họa mượt mà, giảm tải cho CPU và GPU.

Chương 3

ỨNG DỤNG THỰC TIẾT VÀ KẾT LUẬN

3.1 ỨNG DỤNG THỰC TIẾT VÀ GIÁ TRỊ CỦA SẢN PHẨM

3.1.1 Giá Trị Giải Trí Và Giáo Dục

Trò chơi rắn săn mồi thông minh không chỉ mang tính giải trí cao mà còn là công cụ hỗ trợ giáo dục:

- Rèn luyện tư duy chiến lược: Người chơi cần tính toán và lập kế hoạch di chuyển để tránh va chạm, từ đó phát triển khả năng logic và tư duy chiến lược.
- Học lập trình và AI: Sản phẩm là minh chứng cho việc áp dụng thuật toán A* và xử lý thời gian thực, có thể dùng làm tài liệu tham khảo trong các khóa học lập trình và trí tuệ nhân tạo.
- Ứng dụng trong hệ thống tự hành: Các giải pháp định hướng có thể được chuyển giao sang các ứng dụng như robot tự hành, xe tự lái và các hệ thống giám sát.

3.2 Ưu Điểm, Hạn Chế & Phản Hồi Người Dùng

Ưu Điểm

- Tính tự động cao: Thuật toán A* giúp con rắn tự động tìm đường đi tối ưu, giảm tải cho người chơi.
- Hiệu năng ổn định: Trong hầu hết các kịch bản thử nghiệm, hệ thống phản hồi nhanh và chính xác.
- Giao diện trực quan: Thiết kế đồ họa thân thiện, dễ hiểu giúp người chơi dễ dàng theo dõi

quá trình di chuyển của rắn và các thông số trò chơi.

Hạn Chế

- Thời gian xử lý: Trong các bản đồ phức tạp với nhiều chướng ngại vật, thời gian tính toán có thể tăng lên, ảnh hưởng đến trải nghiệm chơi.
- Thiếu tính năng mở rộng: Phiên bản hiện tại chưa tích hợp đầy đủ các yếu tố như chướng ngại vật di động nâng cao hay chế độ chơi đa người.
- Khả năng xử lý ngoại lệ: Một số trường hợp đường đi bị chặn hoàn toàn chưa được xử lý tối ưu, dẫn đến kết thúc game đột ngột.

Phản Hồi Người Dùng

- Người chơi đánh giá cao tính tự động của hệ thống, cảm thấy thú vị khi quan sát con rắn tự tìm đường.
- Một số ý kiến góp ý về việc cải thiện tốc độ phản hồi và mở rộng chế độ chơi để tăng tính cạnh tranh.

3.2.1 Hướng Phát Triển Tương Lai

Các hướng mở rộng và cải tiến dự kiến:

- Tích hợp chướng ngại vật di động nâng cao: Thêm các đối tượng có chuyển động tự động, tạo ra môi trường chơi đa dạng hơn.
- Phát triển chế độ chơi đa người (multiplayer): Cho phép nhiều người chơi cùng tham gia, tạo ra trải nghiệm cạnh tranh và hợp tác.
- Ứng dụng trên nền tảng di động và VR/AR: Phát triển phiên bản trò chơi trên smartphone, console hoặc tích hợp với công nghệ thực tế ảo để tăng tính tương tác.
- Tối ưu hóa thuật toán: Áp dụng các mô hình học máy để cải thiện dự đoán chuyển động của chướng ngại vật và tối ưu hóa thời gian xử lý của A*.
- Mở rộng nội dung trò chơi: Thiết kế thêm các cấp độ chơi với cốt truyện, thử thách mới và phần thưởng hấp dẫn, nhằm thu hút người chơi lâu dài.

3.3 ĐÁNH GIÁ HIỆU SUẤT VÀ PHẢN HỒI NGƯỜI DÙNG

3.3.1 Phương Pháp Đánh Giá

Các tiêu chí đánh giá gồm:

- Tốc độ xử lý: Thời gian tính toán đường đi của thuật toán A*.
- Độ chính xác của đường đi: Mức độ phù hợp giữa đường đi dự kiến và đường đi thực tế khi tránh chướng ngại vật.
- Phản hồi người dùng: Thu thập ý kiến từ người chơi qua khảo sát và phân tích trải nghiệm.

3.4 Đánh giá chung về đề tài

Những Kết Quả Đạt Được

- Tích hợp thành công thuật toán A*:
  Dự án đã áp dụng thuật toán A* vào trò chơi rắn săn mồi một cách hiệu quả, cho phép con rắn tự động tìm đường đi tối ưu đến vị trí thức ăn mà không cần sự can thiệp của người chơi. Việc tính toán dựa trên hàm f(n)=g(n)+h(n) giúp hệ thống đưa ra lựa chọn đường đi an toàn, tránh được va chạm với tường và thân của chính nó.
- Cải thiện trải nghiệm chơi game:
  Nhờ vào việc tự động điều hướng, trò chơi trở nên mượt mà và hấp dẫn hơn. Người chơi có thể quan sát cách con rắn tự động tìm đường, từ đó hiểu thêm về cách hoạt động của thuật toán tìm kiếm. Điều này không chỉ tạo sự thú vị mà còn có giá trị giáo dục trong việc tìm hiểu các khái niệm về AI và lập trình game.
- Hiệu suất xử lý tốt trong môi trường mô phỏng:
  Các thử nghiệm cho thấy hệ thống có thể xử lý nhanh chóng trên các bản đồ kích thước nhỏ đến trung bình. Trong các kịch bản với bản đồ phức tạp, mặc dù thời gian tính toán có xu hướng tăng lên, các biện pháp tối ưu (như sử dụng bộ nhớ cache và chỉ tính toán lại khi cần thiết) vẫn đảm bảo trải nghiệm chơi game không bị gián đoạn quá nhiều.
- Giao diện trực quan và dễ sử dụng:
  Sử dụng thư viện Pygame, giao diện trò chơi được xây dựng với màu sắc rõ ràng, hiển thị bản đồ, điểm số và các thông số cần thiết, tạo nên trải nghiệm trực quan cho người chơi. Các hiệu ứng âm thanh và chuyển động được xử lý mượt mà, góp phần nâng cao chất lượng sản phẩm.

3.4.1 Một số hạn chế của đề tài

- Thời gian xử lý trong môi trường phức tạp:
  Mặc dù hệ thống hoạt động tốt trên các bản đồ nhỏ và trung bình, trong trường hợp bản đồ lớn với nhiều chướng ngại vật, thời gian tính toán của thuật toán A* có thể tăng lên, gây ra độ trễ nhất định trong quá trình cập nhật đường đi.

- Giới hạn về tính năng chướng ngại vật di động:
  Phiên bản hiện tại chủ yếu xử lý các chướng ngại vật cố định (như tường và thân rắn). Việc tích hợp và xử lý các chướng ngại vật di động phức tạp chưa được phát triển toàn diện, điều này có thể ảnh hưởng đến khả năng định hướng tự động của con rắn trong một số tình huống.

- Hạn chế trong khả năng xử lý ngoại lệ:
  Trong một số trường hợp, khi đường đi bị chặn hoàn toàn (do sự sắp xếp của thân rắn hoặc các yếu tố môi trường khác), hệ thống chưa có cơ chế xử lý đặc biệt để khắc phục, dẫn đến kết thúc trò chơi đột ngột.

- Độ phức tạp của giao diện người dùng:
  Giao diện mặc dù trực quan nhưng vẫn còn có thể được cải tiến thêm về mặt tương tác và hiển thị thông tin, đặc biệt khi mở rộng sang các chế độ chơi nâng cao như đa người chơi hay tích hợp với các công nghệ mới như VR/AR.

Chương 4

Phụ Lục

4.1 Công Nghệ Đã Sử Dụng

- Ngôn ngữ lập trình:
  Dự án được xây dựng bằng Python (phiên bản 3.x), nhờ tính dễ đọc, khả năng phát triển nhanh và thư viện phong phú hỗ trợ phát triển game.

- Công cụ phát triển:
  Để phát triển và gỡ lỗi ứng dụng, nhóm sử dụng các IDE như Visual Studio Code hoặc PyCharm. Các công cụ này cung cấp môi trường lập trình thân thiện và hỗ trợ tích hợp các thư viện cần thiết.

- Thư viện chính:

  – Pygame: Dùng để xây dựng giao diện đồ họa, xử lý sự kiện và vẽ các đối tượng (rắn, thức ăn) trên cửa sổ game.

  – Heapq: Hỗ trợ xây dựng hàng đợi ưu tiên trong thuật toán A* để tìm đường đi tối ưu.

  – Random: Được sử dụng để sinh tọa độ ngẫu nhiên cho thức ăn, đảm bảo trò chơi luôn có yếu tố bất ngờ.

4.2 Tiến Độ Thực Hiện

Dự án được triển khai theo các bước sau. Toàn bộ mã nguồn đã được lưu trữ trên GitHub tại: https://github.com/PhanPhong13112005/ran-san-moi-thong-minh.git
Bước 1: Thiết Lập Môi Trường Phát Triển

- Cài đặt Python (phiên bản 3.x) và đảm bảo cài đặt các thư viện cần thiết, đặc biệt là Pygame

(cài qua pip: pip install pygame).

- Chọn IDE phù hợp (Visual Studio Code hoặc PyCharm) và tạo một dự án mới.

Bước 2: Tạo File Code

- Tạo file chính, ví dụ: $game_r an.py$ trong thư mục dự án.
- Xây dựng cấu trúc thư mục hợp lý nếu cần (ví dụ: tách riêng các module liên quan đến logic game, giao diện,...).

Bước 3: Viết Code Và Triển Khai Chức Năng

- Lớp Node & Thuật Toán A*:
  Xây dựng lớp Node để lưu thông tin của các nút trong quá trình tìm kiếm và triển khai hàm a_star sử dụng thuật toán A* nhằm xác định đường đi tối ưu từ vị trí hiện tại của rắn đến vị trí thức ăn.
- Các Hàm Hỗ Trợ:
  Triển khai hàm heuristic tính khoảng cách Manhattan, hàm $generate_food$ để sinh thức ăn ngẫu nhiên, hàm $draw_score$ để hiển thị điểm số, và hàm $game_over_screen$ để thông báo khi trò chơi kết thúc.
- Vòng Lặp Chính (Game Loop):
  Thiết lập vòng lặp chính của trò chơi để liên tục cập nhật vị trí của rắn, tính toán đường đi, cập nhật giao diện và xử lý sự kiện từ người dùng.

Bước 4: Kiểm Tra Và Gỡ Lỗi

- Chạy thử dự án để kiểm tra các chức năng chính như di chuyển của rắn, việc tìm đường bằng A* và hiển thị điểm số.
- Sử dụng các công cụ debug của IDE để xác định và khắc phục lỗi phát sinh trong quá trình phát triển.

4.3 Kết quả thực hiện

- Hoạt Động Ổn Định:
  Trò chơi được triển khai hoạt động như mong đợi. Con rắn tự động di chuyển theo đường tính toán từ thuật toán A* đến vị trí của thức ăn, tăng chiều dài sau mỗi lần ăn, và hệ thống cập nhật điểm số kịp thời.
- Giao Diện Đơn Giản Nhưng Hiệu Quả:
  Sử dụng Pygame, giao diện hiển thị bản đồ dạng lưới, con rắn (màu xanh), thức ăn (màu

đỏ) và các thông tin như điểm số được cập nhật liên tục. Hiệu ứng chuyển động của rắn và thông báo Game Over được thực hiện mượt mà.

- Đánh Giá Hiệu Năng:
  Các thử nghiệm cho thấy thuật toán A* hoạt động nhanh trên các bản đồ kích thước nhỏ đến trung bình. Tuy nhiên, khi bản đồ có kích thước lớn hoặc số lượng đối tượng tăng, thời gian tính toán có thể kéo dài hơn, mặc dù các biện pháp tối ưu đã được áp dụng để giảm thiểu độ trễ.

Code minh họa đầy đủ đã thực hiện:

```
import pygame
import sys
import random
import heapq

# Khởi tạo pygame
pygame.init()

# Cấu hình màn hình
WIDTH, HEIGHT = 800, 500
CELL_SIZE = 10
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("🐍 SNAKE GAME AI 🕹️  ")

# Màu sắc nền tối và neon
BACKGROUND_COLOR = (20, 20, 20)
BUTTON_COLOR = (30, 30, 30)
BUTTON_HOVER_COLOR = (50, 50, 50)
TEXT_COLOR = (0, 255, 200)
TITLE_COLOR = (100, 200, 255)
SNAKE_COLOR = (0, 200, 0)
FOOD_COLOR = (255, 50, 50)

# Load font
font_title = pygame.font.SysFont("Arial", 60, bold=True)
font_button = pygame.font.SysFont("Arial", 36, bold=True)
font_score = pygame.font.SysFont("Arial", 28)

clock = pygame.time.Clock()

def draw_text(text, x, y, font, color, center=False):
    text_surface = font.render(text, True, color)
    rect = text_surface.get_rect(center=(x, y) if center else (x, y))
    screen.blit(text_surface, rect)

# Thuật toán A* tìm đường đi tối ưu
class Node:
    def __init__(self, position, parent=None):
        self.position = position
        self.parent = parent
```

```
        self.g = self.h = self.f = 0
    def __lt__(self, other):
        return self.f < other.f

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def a_star(start, goal, obstacles):
    open_list = []
    closed_set = set()
    start_node = Node(start)
    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)
        closed_set.add(current_node.position)
        if current_node.position == goal:
            path = []
            while current_node:
                path.append(current_node.position)
                current_node = current_node.parent
            return path[::-1]

        for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
            neighbor_pos = (current_node.position[0] + dx * CELL_SIZE, current_node.position[1] + dy * CELL_
            if neighbor_pos in closed_set or neighbor_pos in obstacles or not (0 <= neighbor_pos[0] < WIDTH
                continue
            neighbor_node = Node(neighbor_pos, current_node)
            neighbor_node.g = current_node.g + 1
            neighbor_node.h = heuristic(neighbor_pos, goal)
            neighbor_node.f = neighbor_node.g + neighbor_node.h
            heapq.heappush(open_list, neighbor_node)
    return None

def game_loop():
    snake = [(100, 50), (90, 50), (80, 50)]
    food = (random.randint(0, WIDTH // CELL_SIZE - 1) * CELL_SIZE, random.randint(0, HEIGHT // CELL_SIZE - 1
    score = 0
    running = True
```

```
    while running:
        screen.fill(BACKGROUND_COLOR)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        path = a_star(snake[0], food, set(snake))
        if path and len(path) > 1:
            direction = (path[1][0] - snake[0][0], path[1][1] - snake[0][1])
        else:
            running = False

        new_head = (snake[0][0] + direction[0], snake[0][1] + direction[1])
        if new_head == food:
            food = (random.randint(0, WIDTH // CELL_SIZE - 1) * CELL_SIZE, random.randint(0, HEIGHT
            score += 10
        else:
            snake.pop()

        if new_head in snake or not (0 <= new_head[0] < WIDTH and 0 <= new_head[1] < HEIGHT):
            running = False

        snake.insert(0, new_head)
        for part in snake:
            pygame.draw.rect(screen, SNAKE_COLOR, (*part, CELL_SIZE, CELL_SIZE))
        pygame.draw.rect(screen, FOOD_COLOR, (*food, CELL_SIZE, CELL_SIZE))

        draw_text(f"Score: {score}", WIDTH // 2, 20, font_score, TEXT_COLOR, center=True)
        pygame.display.flip()
        clock.tick(10)

def main_menu():
    while True:
        screen.fill(BACKGROUND_COLOR)
        draw_text("SNAKE GAME AI", WIDTH // 2, HEIGHT // 10, font_title, TITLE_COLOR, center=True)
        play_rect = pygame.Rect(WIDTH // 2 - 150, HEIGHT // 2 - 60, 300, 60)
        quit_rect = pygame.Rect(WIDTH // 2 - 150, HEIGHT // 2 + 40, 300, 60)
        pygame.draw.rect(screen, BUTTON_COLOR, play_rect, border_radius=10)
        pygame.draw.rect(screen, BUTTON_COLOR, quit_rect, border_radius=10)




        draw_text("Start", WIDTH // 2, HEIGHT // 2 - 30, font_button, TEXT_COLOR, center=True)
        draw_text("Quit", WIDTH // 2, HEIGHT // 2 + 70, font_button, TEXT_COLOR, center=True)
        pygame.display.flip()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if play_rect.collidepoint(event.pos):
                    game_loop()
                if quit_rect.collidepoint(event.pos):
                    pygame.quit()
                    sys.exit()

if __name__ == "__main__":
    main_menu()
```

Tài liệu tham khảo

1: Russell, S., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach.

2: Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths.