**ĐẠI HỌC DUY TÂN**

**TRƯỜNG KHOA HỌC MÁY TÍNH**



**MÔN: ĐỒ ÁN CDIO – CS347**

**Tên đề tài: Sử dụng thuật toán dijkstra ứng dụng vào AI tạo ra trò chơi thoát khỏi mê cung**

****

**Giảng viên hướng dẫn : Ts. Lê Thanh Long**

**Nhóm thực hiện : Nhóm 4**

**Thành viên nhóm:**

1. **Nguyễn Văn Cường (28211301911)**
2. **Trần Thế Kiệt (28211353020)**
3. **Võ Thị Khánh Ly (28211347593)**
4. **Nguyễn Hồng Phúc (28211146726)**

**Đà Nẵng, 10 /2024**

**MỤC LỤC**

TRANG

[MỞ ĐẦU 1](#_Toc179237808)

[CHƯƠNG 1: Ý TƯỞNG DỰ ÁN 3](#_Toc179237809)

[I. Đề xướng ý tưởng sản phẩm (cá nhân) 3](#_Toc179237810)

* 1. [*Thành viên 1* 3](#_Toc179237811)

[1.2. Thành viên 2 4](#_Toc179237812)

[1.3. Thành viên 3 5](#_Toc179237813)

[1.4. Thành viên 4 7](#_Toc179237814)

[1.5. Thành viên 5 8](#_Toc179237815)

[**II. Đánh giá ý tưởng sản phẩm trong nhóm 10**](#_Toc179237816)

[**III. Ý tưởng đề xuất 11**](#_Toc179237817)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VÀ PHÂN TÍCH THIẾT KẾ HỆ THỐNG 13](#_Toc179237818)

[I. Cơ sở lý thuyết 13](#_Toc179237819)

[*1.1.* *Giới thiệu ngôn ngữ và công cụ lập trình* 13](#_Toc179237820)

[*1.2.* *Giới thiệu thư viện* 13](#_Toc179237821)

[*1.3.* *Giới thiệu thuật toán Dijkstra* 14](#_Toc179237822)

**[II. Thiết kế mê cung và mô hình hóa trò chơi 15](#_Toc179237823)**

[*2.1.* *Giao diện trò chơi* 15](#_Toc179237824)

[*2.1.1.* *Mục tiêu giao diện* 15](#_Toc179237825)

[*2.1.2.* *Chi tiết giao diện* 15](#_Toc179237826)

[*2.1.3.* *Ví dụ về màu sắc và đồ họa* 15](#_Toc179237827)

[*2.2.* *Thiết kế mê cung* 16](#_Toc179237828)

[*2.3.* *Mô hình hóa trò chơi* 16](#_Toc179237829)

[**III. Phân tích hoạt động trò chơi 17**](#_Toc179237830)

[*3.1.* *Cơ chế hoạt động tổng quát* 17](#_Toc179237831)

[*3.2.* *Phân tích quá trình AI giải quyết bài toán tìm đường* 18](#_Toc179237832)

[**IV. Đánh giá ưu và nhược điểm của hệ thống 19**](#_Toc179237833)

[*4.1.* *Đánh giá hiệu suất của trò chơi* 19](#_Toc179237834)

[*4.2.* *Tính tương tác và trải nghiệm người chơi* 19](#_Toc179237835)

[CHƯƠNG 3: TRIỂN KHAI VÀ CÀI ĐẶT DỰ ÁN 21](#_Toc179237836)

[**I. Mã nguồn chính 21**](#_Toc179237837)

[*1.1. Thiết kế cấu trúc thư mục dự án 21*](#_Toc179237838)

[*1.2. Nội dung thư mục CODE 23*](#_Toc179237839)

[*1.2.1. AI.py 23*](#_Toc179237840)

[*1.2.2. Algorithms.py 25*](#_Toc179237841)

[*1.2.3. ControlPanel.py 27*](#_Toc179237842)

[*1.2.4. Main.py 30*](#_Toc179237843)

[*1.2.5. Maze.py 36*](#_Toc179237844)

[*1.2.6. Welcome.py (File chính để chạy game) 41*](#_Toc179237845)

[*1.3.* *Kiểm thử, đánh giá kết quả* 43](#_Toc179237846)

[KẾT LUẬN 44](#_Toc179237847)

[TÀI LIỆU THAM KHẢO 47](#_Toc179237848)

# MỞ ĐẦU

1. **Lý do chọn đề tài**

Python là một ngôn ngữ lập trình mạnh mẽ và dễ tiếp cận, đặc biệt phù hợp với các bài toán thuật toán và trí tuệ nhân tạo (AI). Đề tài này được lựa chọn nhằm kết hợp giữa hai yếu tố quan trọng: thuật toán Dijkstra - một thuật toán nổi tiếng trong lĩnh vực tìm đường, và AI - một công nghệ tiên tiến trong phát triển trò chơi, nhằm tạo ra một ứng dụng thực tiễn và đầy sự sáng tạo và phát triển.

Việc xây dựng trò chơi thoát khỏi mê cung không chỉ giúp cho chúng em củng cố kiến thức về lập trình Python mà còn mở rộng sự hiểu biết về các thuật toán phức tạp và các ứng dụng thực tế của chúng. Bài toán mê cung với thuật toán Dijkstra thể hiện rõ cách thức Python có thể xử lý các bài toán tối ưu hóa và AI trong việc cải thiện trải nghiệm người chơi, tạo ra một trò chơi có tính tương tác và thử thách cao.

Chọn đề tài này giúp chúng em áp dụng các kiến thức đã học trong môn Python, đồng thời tăng khả năng giải quyết vấn đề và tư duy thuật toán, đặc biệt trong môi trường phát triển trò chơi.

1. **Đối tượng nghiên cứu, công cụ nghiên cứu**

*Đối tượng nghiên cứu* của đồ án này là thuật toán Dijkstra và các phương pháp ứng dụng AI vào phát triển trò chơi, cụ thể là trong việc tìm đường trong mê cung. Chúng em nghiên cứu cách thuật toán Dijkstra có thể được tích hợp vào hệ thống để giải quyết bài toán tìm kiếm đường đi ngắn nhất từ vị trí hiện tại đến lối thoát trong một mê cung. Đồng thời, việc kết hợp với trí tuệ nhân tạo (AI) giúp tăng tính tự động hóa trong việc ra quyết định, cải thiện trải nghiệm của người chơi và tạo ra một môi trường trò chơi thách thức hơn.

Ngoài ra, đối tượng nghiên cứu còn bao gồm việc áp dụng ngôn ngữ lập trình Python vào việc triển khai thuật toán và AI, cũng như xây dựng hệ thống trò chơi hoàn chỉnh, từ thiết kế đồ họa đơn giản đến việc xử lý logic trò chơi.

Trong quá trình thực hiện đồ án, chúng em sử dụng các công cụ và ngôn ngữ lập trình sau:

*Python*: Ngôn ngữ chính để triển khai thuật toán Dijkstra, xây dựng logic của trò chơi, và phát triển các yếu tố AI. Các thư viện liên quan đến đồ họa và thuật toán như Pygame, NumPy, v.v., sẽ được sử dụng để hỗ trợ phát triển hệ thống.

*Pygame*: Thư viện này sẽ được sử dụng để tạo giao diện trò chơi và quản lý các yếu tố đồ họa trong môi trường mê cung.

*Jupyter Notebook*: Được sử dụng làm môi trường phát triển chính cho việc lập trình và thử nghiệm các thuật toán cũng như cấu trúc trò chơi.

*GitHub*: Công cụ quản lý mã nguồn, giúp theo dõi quá trình phát triển và cộng tác nhóm hiệu quả.

*Các tài liệu tham khảo*: Internet, ChatGPT, các tài liệu về thuật toán Dijkstra, AI, và lập trình trò chơi bằng Python.

1. **Phạm vi nghiên cứu**

Phạm vi nghiên cứu của đồ án này tập trung vào việc ứng dụng thuật toán Dijkstra trong việc giải quyết bài toán tìm đường thoát khỏi mê cung, kết hợp với AI để nâng cao khả năng tự động hóa trong trò chơi. Cụ thể, đồ án giới hạn trong các khía cạnh sau:

*Về thuật toán*: Đồ án sẽ tập trung vào việc triển khai thuật toán Dijkstra để tìm đường đi ngắn nhất trong mê cung từ một điểm khởi đầu đến điểm kết thúc. Các yếu tố tối ưu về thời gian và không gian khi thực thi thuật toán sẽ được xem xét.

*Về AI*: Ứng dụng trí tuệ nhân tạo để cải thiện khả năng phản ứng của hệ thống, đặc biệt trong việc điều chỉnh mức độ khó của mê cung dựa trên hành vi của người chơi. Phạm vi nghiên cứu AI được giới hạn ở mức độ tự động hóa và ra quyết định của nhân vật không phải người chơi (NPC).

*Về ngôn ngữ lập trình và công cụ*: Chúng em sử dụng Python và các thư viện liên quan như Pygame để phát triển trò chơi, đồng thời kiểm thử và đánh giá hiệu suất thuật toán trên các mê cung có độ phức tạp khác nhau.

*Về hệ thống trò chơi*: Trò chơi phát triển sẽ có một mê cung 2D với đồ họa cơ bản. Mức độ phức tạp của mê cung và tính thử thách sẽ được kiểm soát thông qua các tham số đầu vào và thiết kế các mức độ chơi khác nhau.

*Phạm vi của nghiên cứu sẽ không mở rộng đến các thuật toán tìm đường khác hay AI ở mức độ cao hơn, mà chỉ tập trung vào Dijkstra và AI cơ bản cho trò chơi thoát mê cung.*

# CHƯƠNG 1: Ý TƯỞNG DỰ ÁN

# Đề xướng ý tưởng sản phẩm (cá nhân)

# *Thành viên 1*

* Họ và tên (sinh viên): Nguyễn Phi Hùng
* Chuyên ngành: Khoa học máy tính
* Tên ý tưởng (sản phẩm): Hệ thống nhận diện biển số xe tự động sử dụng OpenCV
  + - 1. *Mô tả ý tưởng sản phẩm*

Hệ thống nhận diện biển số xe tự động (ANPR - Automatic Number Plate Recognition) là một giải pháp công nghệ hiện đại mang lại nhiều lợi ích thiết thực cho giao thông và an ninh. Việc sử dụng OpenCV để xây dựng hệ thống này có thể giúp tự động hóa quá trình giám sát, thu phí tự động, và hỗ trợ trong việc kiểm soát giao thông tại các khu vực đô thị đông đúc.

Đề tài này được chọn vì việc triển khai hệ thống ANPR không chỉ giúp giảm bớt sự phụ thuộc vào nguồn nhân lực, mà còn giúp tăng cường hiệu quả quản lý giao thông bằng cách giám sát biển số xe trong thời gian thực, nhận diện các phương tiện vi phạm luật lệ giao thông như chạy quá tốc độ, hoặc theo dõi xe bị đánh cắp. Hơn nữa, việc tích hợp hệ thống ANPR vào các hệ thống hiện đại như trạm thu phí tự động, bãi đỗ xe thông minh hoặc quản lý phương tiện giao thông công cộng đều là các ứng dụng thực tiễn cao. Với sự phát triển của AI và Machine Learning, hệ thống ANPR có thể ngày càng trở nên chính xác và nhanh chóng hơn.

1. *Đánh giá cơ sở kiến thức của sinh viên*

* Lập trình cơ sở: Đã học
* Lập trình Python: Đang học
* Machine learning: Đang học
* CDIO 1, 2: Đã học

1. *Đánh giá khả năng phát triển sản phẩm*

|  |  |
| --- | --- |
| Ước tính số lượng người sẽ hưởng lợi từ sản phẩm của bạn | 10000 |
| Ước tính số lượng người sẽ bỏ tiền ra mua sản phẩm của bạn | 100 |
| Liệt kê các loại đối tượng sẽ được hưởng lợi từ ý tưởng sản phẩm của bạn (ít nhất là 1, nhiều nhất là 5): | Cơ sở quản lý giao thông, bãi đỗ xe hay khu vực đỗ xe công cộng… |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trung bình | Khá | Cao | Rất cao |
| Đánh giá khả năng các nhà đầu tư sẽ bỏ tiền ra cho ý tưởng sản phẩm của bạn: |  |  | X |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trung bình | Khá | Cao | Rất cao |
| Ước tính độ khó để phát triển thành công sản phẩm của bạn: |  |  | X |  |

|  |  |
| --- | --- |
| Ước lượng thời gian cần thiết để phát triển sản phẩm của bạn: | 1 tháng |

### Thành viên 2

* Họ và tên sinh viên: Ngô Anh Thư
* Chuyên ngành: Khoa học máy tính
* Tên ý tưởng (sản phẩm): Lập trình game Dinosaur Jumps Game
  + - 1. *Mô tả ý tưởng sản phẩm*

Game hay trò chơi là một trong những thú vui giải trí của con người từ xưa đến nay. Với sự phát triển nhanh chóng của công nghệ, trò chơi giải trí không chỉ dừng lại ở việc chơi thực tế ngoài đời mà còn được xây dựng và phát triển trên điện thoại, PC,... Một trong những thể loại game phổ biến nhất hiện nay là game online – trò chơi trực tuyến, yêu cầu người dùng kết nối với mạng để truy cập. Bên cạnh đó, game offline – trò chơi ngoại tuyến cũng thu hút lượng lớn người dùng không kém. Mọi người có thể sử dụng, chơi game mà không cần kết nối mạng, mặc dù vẫn có nhiều bất tiện vì không có mạng thì một số tính năng sẽ bị hạn chế. Tuy nhiên trò chơi ngoại tuyến vẫn là một trong những sự lựa chọn hoàn hảo để giải trí. Và một trong số những trò chơi ngoại tuyến mà chắc chắn ai trong chúng ta cũng từng chơi qua đó chính là “Dinosaur Jump” hay còn gọi là khủng long vượt chướng ngại vật.

Dinosaur Jumps Game chắc hẳn là một trong những tựa game ngoại tuyến kinh điển mà ai trong chúng ta sử dụng laptop hay PC đã từng thử qua. Với lối chơi vô cùng đơn giản và luật chơi không quá khó, Dinosaur Jumps Game vẫn thu hút một lượng lớn người chơi từ trước đến nay. Chúng em lựa chọn sử dụng ngôn ngữ Python với thư viện Pygame để lập trình nên tựa game này. Kết hợp với những gì đã học trong suốt quá trình theo học tại trường, em hi vọng tựa game này sẽ mang đến sự giải trí cao.

Mặc dù là một dự án nhỏ nhưng nó đã giúp chúng em có cái nhìn tổng quan về quá trình phát triển một trò chơi và mở ra nhiều cơ hội học hỏi thêm về lập trình game.

1. *Đánh giá cơ sở kiến thức của sinh viên*

* Lập trình cơ sở: Đã học
* Lập trình Python: Đang học
* Machine learning: Đang học
* CDIO 1, 2: Đã học

1. *Đánh giá khả năng phát triển sản phẩm*

|  |  |
| --- | --- |
| Ước tính số lượng người sẽ hưởng lợi từ sản phẩm của bạn | 1000 |
| Ước tính số lượng người sẽ bỏ tiền ra mua sản phẩm của bạn | 100 |
| Liệt kê các loại đối tượng sẽ được hưởng lợi từ ý tưởng sản phẩm của bạn (ít nhất là 1, nhiều nhất là 5): | Người dùng có nhu cầu giải trí khi ngoại tuyến; lập trình viên muốn phát triển logic trò chơi cao hơn |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trung bình | Khá | Cao | Rất cao |
| Đánh giá khả năng các nhà đầu tư sẽ bỏ tiền ra cho ý tưởng sản phẩm của bạn: |  | X |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trung bình | Khá | Cao | Rất cao |
| Ước tính độ khó để phát triển thành công sản phẩm của bạn: |  | X |  |  |

|  |  |
| --- | --- |
| Ước lượng thời gian cần thiết để phát triển sản phẩm của bạn: | 2 tuần |

### Thành viên 3

* Họ và tên sinh viên: Lê Ngọc Thịnh
* Chuyên ngành: Khoa học máy tính
* Tên ý tưởng (sản phẩm): Xây dựng hệ thống quản lí điểm sinh viên
  + - 1. *Mô tả ý tưởng sản phẩm*

Ngày nay, để quản lý một tập thể hay một cộng đồng đòi hỏi phải có một hệ thống nhất định. Cùng với thời đại công nghệ phát triển như hiện nay thì việc quản lý trở nên dễ dàng vì có các trang web hay hệ thống hỗ trợ.

Hệ thống quản lý điểm sinh viên được phát triển dựa trên nền tảng Java, là một giải pháp toàn diện, giúp tự động hóa toàn bộ quá trình quản lý điểm số từ việc nhập liệu, tính toán đến việc tạo báo cáo thống kê chi tiết. Với giao diện trực quan và thân thiện, sinh viên có thể dễ dàng tra cứu điểm số, lịch thi, thông báo của mình mọi lúc mọi nơi. Giảng viên có thể nhanh chóng nhập điểm, tạo báo cáo điểm theo lớp, theo môn học, theo học kỳ, đồng thời theo dõi tiến độ học tập của sinh viên một cách hiệu quả. Nhà trường có thể sử dụng hệ thống để quản lý toàn bộ dữ liệu sinh viên, phân tích điểm số để đánh giá chất lượng đào tạo và đưa ra các quyết định điều chỉnh phù hợp.

Hệ thống được xây dựng dựa trên cơ sở dữ liệu MySQL, sử dụng framework Spring và Hibernate để đảm bảo tính ổn định và khả năng mở rộng. Ngoài ra, hệ thống còn tích hợp tính năng gửi thông báo tự động qua email hoặc SMS, giúp sinh viên và giảng viên luôn được cập nhật thông tin mới nhất. Với các tính năng bảo mật cao, hệ thống đảm bảo an toàn cho dữ liệu của sinh viên và nhà trường.

1. *Đánh giá cơ sở kiến thức của sinh viên*

* Lập trình cơ sở: Đã học
* Lập trình Python: Đang học
* Machine learning: Đang học
* CDIO 1, 2: Đã học

1. *Đánh giá khả năng phát triển sản phẩm*

|  |  |
| --- | --- |
| Ước tính số lượng người sẽ hưởng lợi từ sản phẩm của bạn | 10000 |
| Ước tính số lượng người sẽ bỏ tiền ra mua sản phẩm của bạn | 100 |
| Liệt kê các loại đối tượng sẽ được hưởng lợi từ ý tưởng sản phẩm của bạn (ít nhất là 1, nhiều nhất là 5): | Giảng viên, giáo viên có nhu cầu quản lí điểm học sinh/sinh viên; Học sinh/sinh viên có nhu cầu quản lí điểm số của bản thân |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trung bình | Khá | Cao | Rất cao |
| Đánh giá khả năng các nhà đầu tư sẽ bỏ tiền ra cho ý tưởng sản phẩm của bạn: |  |  | X |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trung bình | Khá | Cao | Rất cao |
| Ước tính độ khó để phát triển thành công sản phẩm của bạn: |  |  | X |  |

|  |  |
| --- | --- |
| Ước lượng thời gian cần thiết để phát triển sản phẩm của bạn: | 1 tháng |

### Thành viên 4

* Họ và tên (sinh viên): Nguyễn Hồng Phúc
* Chuyên ngành: Trí Tuệ Nhân Tạo (AI)
* Tên ý tưởng (sản phẩm): Sử dụng thuật toán Dijkstra ứng dụng vào AI tạo ra trò chơi thoát khỏi mê cung

1. *Mô tả ý tưởng sản phẩm*

Trò chơi thoát khỏi mê cung sử dụng thuật toán Dijkstra và trí tuệ nhân tạo (AI) là một dự án kết hợp giữa các kỹ thuật tối ưu hóa đường đi và phát triển trò chơi. Trong trò chơi, người chơi sẽ phải điều hướng nhân vật thoát ra khỏi một mê cung phức tạp. Mỗi lần di chuyển sẽ được hỗ trợ bởi một hệ thống thông minh, ứng dụng thuật toán Dijkstra để tìm đường ngắn nhất từ vị trí hiện tại đến đích. Bên cạnh đó, AI có thể giúp điều chỉnh độ khó của trò chơi dựa trên khả năng của người chơi, tạo ra trải nghiệm phù hợp và thử thách.

Dự án này được lựa chọn vì thuật toán Dijkstra nổi tiếng trong việc tìm đường tối ưu, và có thể được tích hợp vào các trò chơi để nâng cao tính chiến lược và thử thách của người chơi. Sự kết hợp của AI sẽ làm cho trò chơi trở nên sống động và có khả năng tương tác tốt hơn với người chơi, giúp duy trì sự hứng thú lâu dài. Trò chơi có thể được phát triển cho cả thiết bị di động và máy tính cá nhân, phù hợp với nhiều đối tượng người chơi khác nhau.

1. *Đánh giá cơ sở kiến thức của sinh viên*

* Lập trình cơ sở: Đã học
* Lập trình Python: Đang học
* Machine learning: Đang học
* CDIO 1, 2: Đã học

1. *Đánh giá khả năng phát triển sản phẩm*

|  |  |
| --- | --- |
| Ước tính số lượng người sẽ hưởng lợi từ sản phẩm của bạn | 1000 |
| Ước tính số lượng người sẽ bỏ tiền ra mua sản phẩm của bạn | 100 |
| Liệt kê các loại đối tượng sẽ được hưởng lợi từ ý tưởng sản phẩm của bạn (ít nhất là 1, nhiều nhất là 5): | Người dùng có nhu cầu giải trí, lập trình viên muốn phát triển logic trò chơi |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trung bình | Khá | Cao | Rất cao |
| Đánh giá khả năng các nhà đầu tư sẽ bỏ tiền ra cho ý tưởng sản phẩm của bạn: |  | X |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trung bình | Khá | Cao | Rất cao |
| Ước tính độ khó để phát triển thành công sản phẩm của bạn: |  | X |  |  |

|  |  |
| --- | --- |
| Ước lượng thời gian cần thiết để phát triển sản phẩm của bạn: | 1 tháng |

### Thành viên 5

* Họ và tên (sinh viên): Võ Thị Khánh Ly
* Chuyên ngành: Khoa học máy tính
* Tên ý tưởng (sản phẩm): App quản lý tài chính cá nhân

1. *Mô tả ý tưởng sản phẩm*

Ngày nay, việc phát triển ứng dụng quản lý tài chính cá nhân có thể đóng vai trò quan trọng trong việc cải thiện trải nghiệm người dùng, giúp họ kiểm soát và tối ưu hóa các khía cạnh tài chính cá nhân trên nhiều nền tảng công nghệ khác nhau. Các ứng dụng này có thể cung cấp nền tảng vững chắc để người dùng theo dõi chi tiêu, lập kế hoạch tiết kiệm và đạt được các mục tiêu tài chính dài hạn. Chúng cũng có thể giúp đưa ra các phân tích, báo cáo chi tiết dựa trên dữ liệu thu thập được, từ đó cung cấp cái nhìn sâu sắc và cá nhân hóa cho người dùng.

Tuy nhiên, việc phát triển ứng dụng quản lý tài chính cá nhân yêu cầu giải quyết nhiều thách thức về phân tích dữ liệu tài chính, bảo mật thông tin, và cung cấp trải nghiệm người dùng trực quan. Việc đảm bảo tính chính xác trong việc phân loại các giao dịch, dự đoán chi tiêu và cung cấp các gợi ý tài chính phù hợp là nhiệm vụ khó khăn về cả mặt dữ liệu và trải nghiệm người dùng. Ứng dụng cần tích hợp các công nghệ như trí tuệ nhân tạo (AI) và học máy (Machine Learning) để tối ưu hóa và cải thiện hiệu quả phân tích tài chính, giúp người dùng đưa ra quyết định tài chính thông minh hơn.

1. *Đánh giá cơ sở kiến thức của sinh viên*

* Lập trình cơ sở: Đã học
* Lập trình Python: Đang học
* Machine learning: Đang học
* CDIO 1, 2: Đã học

1. *Đánh giá khả năng phát triển của sản phẩm*

|  |  |
| --- | --- |
| 1. Ước tính số lượng người sẽ hưởng lợi từ sản phẩm của bạn | 10000 |
| Ước tính số lượng người sẽ bỏ tiền ra mua sản phẩm của bạn | 200 |
| Liệt kê các loại đối tượng sẽ được hưởng lợi từ ý tưởng sản phẩm của bạn (ít nhất là 1, nhiều nhất là 5): | Cá nhân hay gia đình có nhu cầu quản lý chi tiêu, thu thập. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trung bình | Khá | Cao | Rất cao |
| Đánh giá khả năng các nhà đầu tư sẽ bỏ tiền ra cho ý tưởng sản phẩm của bạn: |  |  | X |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trung bình | Khá | Cao | Rất cao |
| Ước tính độ khó để phát triển thành công sản phẩm của bạn: |  | X |  |  |

|  |  |
| --- | --- |
| Ước lượng thời gian cần thiết để phát triển sản phẩm của bạn: | 1 tháng |

## Đánh giá ý tưởng sản phẩm trong nhóm

* Tên ý tưởng sản phẩm: “Sử dụng thuật toán Dijkstra ứng dụng vào AI tạo ra trò chơi thoát khỏi mê cung”
* Ngày bắt đầu: 9/9/2024
* Ngày kết thúc: 1/10/2024

|  |  |  |
| --- | --- | --- |
|  | [Perl & Python](https://mydtu.duytan.edu.vn/sites/index.aspx?p=home_listcoursedetailforstudent&courseid=910&academicleveltypeid=3&curriculumid=774) | [Machine Learning 1](https://mydtu.duytan.edu.vn/sites/index.aspx?p=home_listcoursedetailforstudent&courseid=1843&academicleveltypeid=3&curriculumid=774) |
| Thành viên 1 | Đã học | Đang học |
| Thành viên 2 | Đã học | Đang học |
| Thành viên 3 | Đã học | Đang học |
| Thành viên 4 | Đã học | Đang học |
| Thành viên 5 | Đã học | Đang học |

|  |  |  |
| --- | --- | --- |
| *Liệt kê các loại đối tượng hưởng lợi từ sản phẩm* | | |
|  | Đối Tượng 1 | Đối Tượng 2 |
| Thành viên 1 | Người dùng cuối (cá nhân, tổ chức, đối tác) | Tất cả mọi người, mọi lứa tuổi |
| Thành viên 2 | Người dùng cuối (cá nhân, tổ chức, đối tác) | Tất cả mọi người, mọi lứa tuổi |
| Thành viên 3 | Người dùng cuối (cá nhân, tổ chức, đối tác) | Tất cả mọi người, mọi lứa tuổi |
| Thành viên 4 | Người dùng cuối (cá nhân, tổ chức, đối tác) | Tất cả mọi người, mọi lứa tuổi |
| Thành viên 5 | Người dùng cuối (cá nhân, tổ chức, đối tác) | Tất cả mọi người, mọi lứa tuổi |

|  |  |  |
| --- | --- | --- |
| *Ước lượng số người hưởng lợi từ sản phẩm của bạn* | | |
| Thành viên  1 | Thành viên  2 | Thành viên  3 |
| 15000 | 10000 | 20000 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Ước tính số lượng người sẽ bỏ tiền ra mua sản phẩm của bạn* | | | | | | |
| Thành viên  1 | | Thành viên  2 | | | Thành viên  3 | |
| 300 | | 200 | | | 100 | |
| *Đánh giá khả năng các nhà đầu sẽ bỏ tiền ra để phát triển sản phẩm của bạn* | | | | | | |
|  | Trung bình | | Khá | Cao | | Rất Cao |
| Thành viên 1 |  | |  | x | |  |
| Thành viên 2 |  | | X |  | |  |
| Thành viên 3 |  | | X |  | |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Ước lượng độ khó trong việc phát triển thành công ý tưởng sản phẩm đã nêu:* | | | | |
|  | Trung bình | Khá | Cao | Rất Cao |
| Thành viên 1 |  |  | x |  |
| Thành viên 2 |  | x |  |  |
| Thành viên 3 |  |  | x |  |
| Thành viên 4 |  | x |  |  |
| Thành viên 5 |  | x |  |  |

## Ý tưởng đề xuất

* Tên nhóm: Nhóm 4
* Nhóm trưởng: Nguyễn Hồng Phúc
* Thành viên:
* Nguyễn Phi Hùng
* Võ Thị Khánh Ly
* Ngô Anh Thư
* Lê Ngọc Thịnh
* Cụ thể

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Họ và tên | Vai trò | Trách nhiệm |
| 1 | Nguyễn Hồng Phúc | Trưởng nhóm | Quản lý tiến độ, Phân tích, code |
| 2 | Nguyễn Phi Hùng | Thành viên | Phân tích, code, test |
| 3 | Võ Thị Khánh Ly | Thành viên | Phân tích, code, báo cáo word |
| 4 | Ngô Anh Thư | Thành viên | Phân tích, code, báo cáo slide |
| 5 | Lê Ngọc Thịnh | Thành viên | Phân tích, code, test |

* Tên sản phẩm: “Sử dụng thuật toán Dijkstra ứng dụng vào AI tạo ra trò chơi thoát khỏi mê cung”
* Phương pháp nghiên cứu:
* Nghiên cứu lý thuyết về thuật toán Dijkstra và cách thức hoạt động của nó trong việc tìm kiếm đường đi ngắn nhất trên đồ thị.
* Phân tích và lập trình AI để tích hợp thuật toán Dijkstra vào trò chơi.
* Sử dụng ngôn ngữ lập trình Python để phát triển trò chơi.

# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VÀ PHÂN TÍCH THIẾT KẾ HỆ THỐNG

# Cơ sở lý thuyết

### *Giới thiệu ngôn ngữ và công cụ lập trình*

**Python**

Python được tạo ra bởi Guido van Rossum vào cuối thập niên 1980 và chính thức ra mắt lần đầu tiên vào năm 1991. Mục tiêu của Guido là phát triển một ngôn ngữ lập trình mạnh mẽ nhưng đơn giản, dễ học, và dễ sử dụng. Ngôn ngữ này được thiết kế nhằm khuyến khích lập trình viên viết mã nguồn gọn gàng, dễ đọc, đồng thời hỗ trợ tốt cho lập trình theo hướng đối tượng.

Python là một ngôn ngữ lập trình bậc cao, đa năng và phổ biến được sử dụng trong nhiều lĩnh vực, từ phát triển web và khoa học dữ liệu đến tự động hóa và trí tuệ nhân tạo.

Python nổi tiếng với cú pháp rõ ràng, dễ đọc và dễ học, làm cho nó trở thành một lựa chọn tuyệt vời cho người mới bắt đầu và các lập trình viên có kinh nghiệm.

Sự phổ biến rộng rãi của Python là kết quả của khả năng ứng dụng đa dạng, cộng đồng hỗ trợ mạnh mẽ và các thư viện phong phú.

**Visual studio code**

Visual Studio Code (VS Code) là một trình soạn thảo mã nguồn mở, nhẹ và miễn phí do Microsoft phát triển. Nó được thiết kế để hỗ trợ nhiều ngôn ngữ lập trình và môi trường phát triển, và đang rất được ưa chuộng bởi các lập trình viên nhờ vào sự đơn giản, tốc độ và khả năng tùy biến cao.

Visual Studio Code hỗ trợ đa dạng các chức năng Debug, đi kèm với Git, có Syntax Highlighting. Đặc biệt là tự hoàn thành mã thông minh, Snippets, và khả năng cải tiến mã nguồn. Nhờ tính năng tùy chỉnh, Visual Studio Code cũng cho phép các lập trình viên thay đổi Theme, phím tắt, và đa dạng các tùy chọn khác. Mặc dù trình soạn thảo Code này tương đối nhẹ, nhưng lại bao gồm các tính năng mạnh mẽ.

Dù mới được phát hành nhưng VSCode là một trong những Code Editor mạnh mẽ và phổ biến nhất dành cho lập trình viên. Nhờ hỗ trợ nhiều ngôn ngữ lập trình phổ biến, tích hợp đầy đủ các tính năng và khả năng mở rộng, nên VSCode trở nên cực kì thân thuộc với bất kì lập trình viên nào.

### *Giới thiệu thư viện*

*Pygame:* Thư viện này sẽ được sử dụng để tạo giao diện trò chơi và quản lý các yếu tố đồ họa trong môi trường mê cung.

*Random:* là một thư viện chuẩn của Python dùng để tạo ra các giá trị ngẫu nhiên. Nó thường được sử dụng trong các trò chơi để làm cho trò chơi trở nên không đoán trước, ví dụ như tạo vị trí ngẫu nhiên cho đối tượng hoặc quyết định hành động ngẫu nhiên.

*Time:* là một thư viện tiêu chuẩn của Python cho phép bạn làm việc với thời gian. Nó cung cấp các phương thức để tạm dừng chương trình, đo thời gian thực thi, hoặc lấy thời gian hiện tại.  
*Mô đun Maze:* Bao gồm các phương thức để sinh mê cung, kiểm tra các lối đi, và áp dụng các thuật toán tìm đường như Dijkstra hoặc A\*.

### *Giới thiệu thuật toán Dijkstra*

Thuật toán Dijkstra là một thuật toán nổi tiếng để tìm đường ngắn nhất trong đồ thị có trọng số không âm. Được phát triển bởi nhà khoa học máy tính Edsger W. Dijkstra vào năm 1956, thuật toán này hoạt động trên nguyên tắc mở rộng dần dần các nút với trọng số nhỏ nhất cho đến khi tìm ra đường ngắn nhất tới tất cả các đỉnh từ điểm bắt đầu.

Thuật toán sử dụng một tập hợp các đỉnh chưa được xử lý và một danh sách ưu tiên (priority queue) để theo dõi đỉnh có khoảng cách nhỏ nhất từ điểm bắt đầu. Nó lần lượt cập nhật khoảng cách của các đỉnh lân cận thông qua các cạnh và chọn đỉnh có khoảng cách nhỏ nhất cho lần xử lý tiếp theo.

*Pseudo-code của thuật toán:*

Bước 1: Gán cho mỗi đỉnh một giá trị vô cùng (∞), trừ đỉnh bắt đầu có giá trị bằng 0.

Bước 2: Sử dụng một hàng đợi ưu tiên để chọn đỉnh có khoảng cách nhỏ nhất.

Bước 3: Với mỗi đỉnh được chọn, tính toán lại khoảng cách tới các đỉnh lân cận thông qua đỉnh này. Nếu khoảng cách ngắn hơn khoảng cách đã biết, cập nhật khoảng cách.

Bước 4: Lặp lại cho đến khi tất cả các đỉnh được xử lý.

*Ưu điểm của thuật toán*

* Đảm bảo tìm được đường ngắn nhất trong đồ thị có trọng số không âm.
* Hiệu quả trong nhiều tình huống, đặc biệt là tìm đường trong các hệ thống giao thông, mạng lưới truyền thông và trò chơi.

*Hạn chế của thuật toán*

* Không hoạt động hiệu quả với các đồ thị có cạnh có trọng số âm.
* Trong trường hợp đồ thị lớn, thuật toán có thể mất nhiều thời gian và tài nguyên để tính toán.

*Cách triển khai thuật toán Dijkstra trong AI trò chơi*

*Mô hình hóa mê cung như một đồ thị:*

Mê cung có thể được xem như một đồ thị, trong đó mỗi ô vuông là một đỉnh và các cạnh kết nối giữa các ô liền kề có trọng số bằng 1 (biểu diễn cho khoảng cách giữa các ô là đơn vị). Các bức tường trong mê cung tương ứng với các cạnh không tồn tại (hoặc có trọng số vô cùng). Thuật toán Dijkstra đóng vai trò là bộ xử lý tìm kiếm tối ưu để giúp AI xác định đường đi ngắn nhất từ điểm bắt đầu đến đích trong mê cung.

*Triển khai thuật toán:*

Đầu tiên, đặt người chơi hoặc điểm bắt đầu tại một ô cụ thể trong mê cung (góc trái trên cùng). Sau đó, thuật toán Dijkstra sẽ được sử dụng để tìm ra đường đi ngắn nhất từ điểm bắt đầu đến đích (góc phải dưới cùng).

Khi AI tìm được đường ngắn nhất, nó sẽ điều khiển nhân vật di chuyển theo từng bước một qua các ô trong mê cung, tránh các bức tường hoặc chướng ngại vật và tìm đường thoát.

## Thiết kế mê cung và mô hình hóa trò chơi

### *Giao diện trò chơi*

#### *Mục tiêu giao diện*

Giao diện của trò chơi thoát khỏi mê cung cần đảm bảo tính trực quan và dễ thao tác cho người chơi. Bố cục cần rõ ràng, dễ nhìn và các thành phần đồ họa phải hỗ trợ tốt cho việc trải nghiệm.

#### *Chi tiết giao diện*

*Bố cục tổng thể*

Giao diện bắt đầu: Có các nút chức năng chính như: bắt đầu trò chơi, thoát trò chơi, thông tin trò chơi v.v.

Khu vực mê cung: Chiếm phần lớn màn hình, nơi người chơi hoặc AI sẽ di chuyển trong mê cung để tìm đường thoát. Khu vực này sẽ được thiết kế dưới dạng lưới (grid), mỗi ô vuông trong lưới tượng trưng cho một vị trí trong mê cung. Các bức tường trong mê cung được vẽ dưới dạng các đường ngăn giữa các ô, và các khoảng trống là nơi người chơi hoặc AI có thể di chuyển.

Bảng thông tin: Góc bên phải màn hình có thể hiển thị các thông tin như thời gian, số bước đã thực hiện, và trạng thái của AI (ví dụ: đang tìm kiếm, đã hoàn thành).

Bảng điều khiển: Người chơi có thể chọn các chức năng như "Khởi động", "Tìm đường", "Chơi lại", và "Thoát khỏi trò chơi". Nếu muốn, có thể thêm tùy chọn cho người chơi tự di chuyển trong mê cung thay vì để AI điều khiển.

#### *Ví dụ về màu sắc và đồ họa*

*Màu sắc:*

Các bức tường mê cung được tô màu đen hoặc xám đậm, trong khi các đường đi có thể có màu sáng hơn (như trắng hoặc xanh nhạt) để người chơi dễ phân biệt.

Người chơi (hoặc AI) có thể được đại diện bởi một biểu tượng hoặc hình ảnh (chẳng hạn như một nhân vật nhỏ hoặc một khối màu đỏ).

Điểm xuất phát và điểm đích được đánh dấu bằng các màu đặc trưng, chẳng hạn như màu xanh cho điểm xuất phát và màu vàng hoặc cam cho điểm đích.

*Hiệu ứng đồ họa:*

Khi AI hoặc người chơi di chuyển qua mê cung, đường đi có thể được đánh dấu lại (ví dụ: bằng cách thay đổi màu sắc của các ô mà nhân vật đã đi qua).

Khi AI tìm thấy đường thoát, một hiệu ứng hoặc âm thanh có thể được kích hoạt để báo hiệu sự thành công.

### *Thiết kế mê cung*

*Cấu trúc mê cung:*

Mê cung sẽ được mô phỏng dưới dạng một ma trận lưới hai chiều (2D grid), trong đó các ô vuông tượng trưng cho các vị trí mà người chơi có thể di chuyển, và các cạnh ngăn cách các ô là các bức tường.

Kích thước mê cung có thể thay đổi tùy vào mức độ khó, ví dụ như mê cung nhỏ 5x5 hoặc mê cung lớn hơn 10x10 hay 20x20. Điều này sẽ tăng độ thử thách cho AI và người chơi.

*Thuật toán tạo mê cung*

Để thiết kế một mê cung ngẫu nhiên, sản phẩm này sử dụng một (class - Maze) để tạo mê cung bằng một trong các thuật toán như Recursive Backtracking, Kruskal's Algorithm, hoặc Prim's Algorithm v.v để tạo, cụ thể sử dụng thuật toán DFS với Backtracking để tạo ra mê cung ngẫu nhiên, đồng thời có tích hợp thêm các thành phần như phần thưởng, chướng ngại vật, và hiển thị mê cung thông qua thư viện pygame. Các thuật toán này sẽ tạo ra mê cung đảm bảo rằng có ít nhất một đường thoát từ điểm bắt đầu đến điểm đích.

Mê cung ngẫu nhiên: Trong trò chơi, mỗi lần chơi có thể tạo ra một mê cung khác nhau dựa trên thuật toán ngẫu nhiên, điều này đảm bảo rằng AI phải giải quyết các thử thách mới mỗi lần chạy.

*Kiểm soát mức độ khó:*

Để tăng mức độ thử thách, các yếu tố như số lượng bức tường, chướng ngại vật, độ phức tạp của đường đi, và kích thước mê cung có thể được tùy chỉnh. Mê cung càng lớn và càng nhiều bức tường thì AI càng cần phải xử lý nhiều hơn để tìm đường thoát.

### *Mô hình hóa trò chơi*

***Mô hình di chuyển trong mê cung:***

*Điểm bắt đầu và điểm kết thúc:*

Người chơi hoặc AI sẽ bắt đầu tại một vị trí cố định cụ thể là góc trên cùng bên trái của mê cung được tô một màu cụ thể khi bắt đầu sẽ trở thành một icon nhân vật người chơi tường minh. Mục tiêu là tìm đường đến đích, vị trí đặt tại góc dưới cùng bên phải của mê cung nơi mà có phần thưởng chính của trò chơi.

*Di chuyển:*

Mỗi bước di chuyển được giới hạn trong bốn hướng (lên, xuống, trái, phải), tương tự như cách thức di chuyển trong ma trận lưới, được kích hoạt bằng các dấu mũi tên điều hướng trên bàn phím. AI sẽ quyết định các bước di chuyển dựa trên thuật toán Dijkstra để tối ưu hóa đường đi ngắn nhất từ điểm bắt đầu đến đích.

***Hành vi của AI:***

*Áp dụng thuật toán Dijkstra:*

AI sẽ liên tục tính toán đường đi ngắn nhất từ vị trí hiện tại đến điểm đích. Nó sẽ tránh các bức tường và di chuyển theo lộ trình được xác định trước cho đến khi tìm thấy đích.

*Cập nhật và xử lý thời gian thực:*

Trong trường hợp thay đổi trò chơi mới trong mê cung (ví dụ, người chơi nhấn vào “Reset Maze” một số bức tường có thể bị thay đổi hoặc di chuyển), AI sẽ phải cập nhật lại tính toán của mình và điều chỉnh hướng đi.

*Tùy chọn điều khiển thủ công:*

Trò chơi cũng có thể cung cấp tùy chọn cho người chơi tự di chuyển trong mê cung thay vì để AI điều khiển. Điều này giúp trò chơi có tính tương tác cao hơn, và người chơi có thể tự thử thách mình để tìm đường thoát.

1. *Tùy chỉnh và mở rộng trò chơi.*

*Đây là chức năng mở năng mở rộng trò chơi yêu cầu lập trình viên phải sáng tạo thêm cho chức năng này*

*Mức độ khó:* Người chơi có thể tùy chỉnh mức độ khó của mê cung trước khi bắt đầu. Mức độ khó cao hơn có thể bao gồm mê cung lớn hơn, nhiều bức tường hơn, và các đường đi phức tạp hơn.

*Các chế độ chơi:*

Chế độ tự do (Player Play): Người chơi tự điều khiển và tìm cách thoát khỏi mê cung mà không cần sự trợ giúp của AI.

Chế độ AI (AI Play) : AI sẽ tự động tìm đường đi và người chơi có thể quan sát quá trình trò chơi.

Chế độ thử thách (phát triển trong tương lai): Người chơi và AI thi đấu xem ai tìm ra đường đi ngắn nhất nhanh hơn.

Phát triển thêm: Có thể mở rộng trò chơi bằng cách thêm các tính năng mới như chướng ngại vật di chuyển, thời gian giới hạn, hoặc thêm các yếu tố thưởng phạt (ví dụ, điểm cộng nếu đi đúng đường, điểm trừ nếu gặp phải bẫy).

## Phân tích hoạt động trò chơi

### *Cơ chế hoạt động tổng quát*

*Mục tiêu của trò chơi:*

Trò chơi được thiết kế để thử thách AI (hoặc người chơi) tìm đường thoát khỏi mê cung trong thời gian ngắn nhất. Hoạt động của trò chơi xoay quanh việc sử dụng thuật toán Dijkstra để tính toán và đưa ra lộ trình ngắn nhất từ điểm xuất phát đến điểm kết thúc trong mê cung.

*Các thành phần chính của trò chơi:*

Người chơi/AI: Người chơi có thể tự điều khiển nhân vật hoặc cho phép AI tự động điều khiển. Trong trường hợp AI điều khiển, thuật toán Dijkstra sẽ được sử dụng để xác định đường đi tối ưu.

Mê cung: Mê cung là không gian giới hạn cho nhân vật di chuyển, với các bức tường ngăn chặn và lối đi mở ra. Mê cung có cấu trúc ngẫu nhiên, tạo ra sự thay đổi và thách thức mới cho mỗi lần chơi.

Điểm xuất phát và điểm đích: Mỗi màn chơi đều có một điểm bắt đầu và điểm kết thúc cố định. Nhiệm vụ của AI là tìm đường ngắn nhất giữa hai điểm này thông qua các lối đi trong mê cung.

### *Phân tích quá trình AI giải quyết bài toán tìm đường*

*Quá trình tìm đường của AI:*

1. Khởi tạo mê cung và vị trí bắt đầu:

Khi trò chơi bắt đầu, một mê cung mới được khởi tạo với cấu trúc ngẫu nhiên, bao gồm các bức tường và đường đi. Vị trí bắt đầu của nhân vật AI được xác định (ví dụ, góc trên bên trái) và điểm đích được đặt ở một vị trí khác (thường là góc dưới bên phải hoặc một vị trí ngẫu nhiên).

1. Thuật toán Dijkstra:

AI sử dụng thuật toán Dijkstra để xác định lộ trình ngắn nhất từ vị trí hiện tại đến điểm đích. Quá trình này bao gồm:

Tạo đồ thị từ mê cung: Mê cung được mô phỏng thành một đồ thị, trong đó mỗi ô vuông là một đỉnh, và các cạnh đại diện cho các lối đi giữa các ô.

Áp dụng thuật toán: Thuật toán Dijkstra được áp dụng để tìm đường ngắn nhất bằng cách duyệt qua tất cả các lối đi có thể từ điểm xuất phát, đồng thời lưu lại lộ trình có tổng chi phí thấp nhất đến điểm đích.

Cập nhật lộ trình: Sau khi thuật toán hoàn tất, AI có thể di chuyển theo lộ trình ngắn nhất đã tính toán được.

*Quá trình di chuyển của AI:*

Sau khi xác định được đường đi, AI sẽ di chuyển từ ô này sang ô khác theo lộ trình đã được xác định. Mỗi lần di chuyển, AI sẽ cập nhật vị trí của mình và kiểm tra xem đã đến đích chưa.

Tối ưu hóa đường đi: Nếu có các cập nhật trong mê cung (ví dụ: các bức tường bị thay đổi hoặc di chuyển), AI sẽ cần phải tính toán lại lộ trình để đảm bảo luôn đi theo đường ngắn nhất.

## Đánh giá ưu và nhược điểm của hệ thống

### *Đánh giá hiệu suất của trò chơi*

***Hiệu quả của thuật toán Dijkstra:***

*Ưu điểm:*

Thuật toán Dijkstra luôn đảm bảo tìm ra lộ trình ngắn nhất, không bỏ sót bất kỳ lối đi nào khả thi. Điều này làm cho AI có thể nhanh chóng xác định được đường thoát khỏi mê cung một cách tối ưu nhất.

Thuật toán cũng hoạt động tốt với các mê cung phức tạp có nhiều ngã rẽ và đường cụt, đảm bảo AI có thể xử lý được ngay cả những trường hợp khó khăn nhất.

*Nhược điểm:*

Mặc dù thuật toán Dijkstra hoạt động rất tốt với các mê cung nhỏ và vừa, nhưng đối với các mê cung lớn (như 50x50), quá trình tính toán có thể tiêu tốn nhiều tài nguyên và thời gian hơn, làm chậm trải nghiệm chơi trò chơi.

***Phân tích thời gian xử lý:***

*Thời gian tìm đường:*

Đối với các mê cung có kích thước nhỏ (ví dụ: 10x10), thời gian tìm đường hầu như tức thì. Tuy nhiên, với các mê cung lớn hơn (ví dụ: 20x20 hoặc 50x50), thời gian cần để thuật toán Dijkstra hoàn thành có thể kéo dài.

Tùy vào cấu trúc của mê cung và số lượng bức tường, thời gian tìm đường có thể khác nhau. Các mê cung có nhiều bức tường và ngõ cụt sẽ yêu cầu nhiều bước tính toán hơn.

***Phân tích về khả năng xử lý của AI:***

*Xử lý các trường hợp ngẫu nhiên:*

Trong trường hợp mê cung thay đổi liên tục (ví dụ: các bức tường có thể di chuyển hoặc tạo mới trong quá trình chơi), AI cần phải tái tính toán lộ trình. Điều này có thể làm tăng độ phức tạp và đòi hỏi AI phải xử lý nhanh chóng để tránh làm gián đoạn quá trình chơi.

*Khả năng thích nghi:*

AI có khả năng thích nghi tốt với các thay đổi trong mê cung. Nếu lộ trình ban đầu không còn khả thi do bức tường mới xuất hiện, AI sẽ tự động điều chỉnh và tìm ra lộ trình mới dựa trên tình hình hiện tại.

### *Tính tương tác và trải nghiệm người chơi*

*Phân tích tương tác người chơi:*

Nếu trò chơi cho phép người chơi tự điều khiển nhân vật, trải nghiệm sẽ tập trung vào khả năng giải quyết vấn đề của người chơi khi tự tìm đường thoát khỏi mê cung.

Trong khi đó, nếu AI tự động điều khiển, người chơi có thể theo dõi quá trình xử lý của AI, từ việc tính toán lộ trình đến việc di chuyển trong mê cung. Điều này giúp người chơi hiểu rõ hơn về cách thuật toán Dijkstra hoạt động trong thực tế.

*Thử thách và phần thưởng:*

Trò chơi có thể thêm các yếu tố thử thách để làm tăng độ khó, chẳng hạn như giới hạn thời gian, các bẫy ngẫu nhiên xuất hiện trong mê cung, hoặc các phần thưởng đặc biệt khi AI hoặc người chơi tìm thấy các lối tắt.

# CHƯƠNG 3: TRIỂN KHAI VÀ CÀI ĐẶT DỰ ÁN

*LƯU Ý: Toàn bộ nội dung chương này được trình bày dưới dạng văn bản, nội dung bao gồm các yêu cầu hệ thống, toàn bộ mã nguồn của dự án, hình ảnh minh họa cho dự án được trình bày chi tiết trong silde PDF được đính kèm nằm trong thư mục tổng của dự án với tên ‘GAMETTNT’*

## Mã nguồn chính

### *Thiết kế cấu trúc thư mục dự án*

Toàn bộ cấu trúc thư mục dự án được tập hợp lại trong thư mục ‘GAMETTNT’. Cấu trúc này giúp quản lý rõ ràng mã nguồn, âm thanh và hình ảnh phục vụ cho việc phát triển và cài đặt trò chơi và được biểu diễn dưới dạng như sau:

GAMETTNT/

│

├── .vscode/

│ └── extensions.json

│

├── src/

│ ├── \_\_pycache\_\_/

│ ├── Code/

│ │ ├── \_\_pycache\_\_/

│ │ ├── AI.py

│ │ ├── Algorithms.py

│ │ ├── ControlPanel.py

│ │ ├── Main.py

│ │ ├── Maze.py

│ │ └── Welcome.py

│ │

│ ├── Music/

│ │ ├── button\_click.mp3

│ │ ├── dendich.mp3

│ │ ├── gameover.mp3

│ │ ├── nenStart.MP3

│ │ ├── nhac.MP3

│ │ ├── trungchuonggaivang.mp3

│ │ └── trungthuong.mp3

│ │

│ └── Picture/

│ ├── dautay.jpg

│ ├── hangrao.jpg

│ ├── icon.png

│ ├── iconbom.png

│ ├── phanthuong.jpg

│ ├── selectbackground.jpg

│ └── wellcome.jpg

*Trong đó:*

• GAMETTNT/: Thư mục chính của dự án.

• .vscode/: Thư mục chứa các tệp cấu hình cho Visual Studio Code, bao gồm extensions.json.

• src/: Thư mục chính chứa mã nguồn và tài nguyên của dự án.

• pycache/: Thư mục chứa các tệp đã biên dịch của Python.

• Code/: Thư mục chứa các tệp mã Python chính.

• AI.py: Tệp chứa mã cho phần AI.

• Algorithms.py: Tệp chứa các thuật toán.

• ControlPanel.py: Tệp quản lý bảng điều khiển.

• Main.py: Tệp mã chính khởi động trò chơi.

• Maze.py: Tệp chứa mã tạo mê cung.

• Welcome.py: Tệp xử lý màn hình chào mừng.

• Music/: Thư mục chứa các tệp âm thanh cho trò chơi.

• button\_click.mp3: Âm thanh khi nhấn nút.

• dendich.mp3: Âm thanh hoàn thành.

• gameover.mp3: Âm thanh kết thúc trò chơi.

• nenStart.MP3: Nhạc nền khi bắt đầu.

• nhac.MP3: Nhạc nền chính.

• trungchuonggaivang.mp3: Âm thanh thưởng.

• trungthuong.mp3: Âm thanh trung thưởng.

• Picture/: Thư mục chứa các hình ảnh đồ họa.

• dautay.jpg: Hình ảnh đại diện cho nhân vật.

• hangrao.jpg: Hình ảnh hàng rào.

• icon.png: Biểu tượng trò chơi.

• iconbom.png: Biểu tượng chướng ngại vật.

• phanthuong.jpg: Hình ảnh phần thưởng.

• selectbackground.jpg: Hình nền cho màn hình lựa chọn.

• wellcome.jpg: Hình ảnh chào mừng.

### *1.2. Nội dung thư mục CODE*

#### *AI.py*

|  |
| --- |
| import pygame  from Maze import Maze  from pygame.locals import \*  import threading  import os  class AI:  def \_\_init\_\_(self, maze):  # Hàm khởi tạo, nhận vào đối tượng maze và thiết lập các thuộc tính cho AI  self.base\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  self.music\_dir = os.path.join(self.base\_dir, "Music")  self.maze = maze  self.x, self.y = maze.start\_x, maze.start\_y # Lưu vị trí ban đầu của AI trong mê cung  self.start\_x = maze.start\_x # Lưu trữ vị trí ban đầu của icon theo trục X  self.start\_y = maze.start\_y # Lưu trữ vị trí ban đầu của icon theo trục Y  self.path = [] # Danh sách các bước đã đi qua, lưu lộ trình của AI  self.move\_history = [] # Lịch sử các bước đã di chuyển  def move\_towards(self, target\_x, target\_y):  # Hàm di chuyển AI về hướng mục tiêu (target\_x, target\_y)  dx = target\_x - self.x # Tính khoảng cách theo trục X  dy = target\_y - self.y # Tính khoảng cách theo trục Y  if dx != 0:  dx //= abs(dx) # Chuẩn hóa bước di chuyển theo trục X (1 hoặc -1)  if dy != 0:  dy //= abs(dy) # Chuẩn hóa bước di chuyển theo trục Y (1 hoặc -1)  next\_x = self.x + dx # Tọa độ tiếp theo theo trục X  next\_y = self.y + dy # Tọa độ tiếp theo theo trục Y  # Kiểm tra xem ô tiếp theo có phải là ô trắng (0) hoặc chứa chướng ngại vật hay không  if self.maze.grid[next\_y][next\_x] == 0 or (next\_x, next\_y) in self.maze.obstacles:  # Nếu ô tiếp theo là ô chứa chướng ngại vật  if (next\_x, next\_y) in self.maze.obstacles:  # Nếu ô tiếp theo có chướng ngại vật, lùi lại 3 bước đã đi trước đó  if len(self.move\_history) >= 4:  # Lùi lại 3 bước trong lịch sử di chuyển  for i in range(3):  x, y = self.move\_history.pop() # Lấy vị trí lùi lại  self.path.remove((x, y)) # Xóa tọa độ này khỏi lộ trình đã đi  self.maze.grid[y][x] = 0 # Đặt lại giá trị của ô thành 0 (trắng)  # Lấy tọa độ của bước cuối cùng đã đi trước đó  next\_x, next\_y = self.move\_history[-1]  else:  # Nếu không đủ 3 bước trong lịch sử, di chuyển về điểm xuất phát  next\_x, next\_y = self.maze.start\_x, self.maze.start\_y  # Cập nhật lại tọa độ sau khi lùi  self.x = next\_x  self.y = next\_y  self.path.append((self.x, self.y)) # Lưu vào lộ trình hiện tại  self.move\_history.append((self.x, self.y)) # Lưu vào lịch sử di chuyển  # Tạo một thread để phát nhạc khi lùi lại  threading.Thread(target=self.play\_sound\_after\_reverse).start()  return # Kết thúc phương thức sau khi xử lý va chạm  # Nếu ô tiếp theo là ô trắng (0), di chuyển bình thường  self.x = next\_x  self.y = next\_y  self.path.append((self.x, self.y)) # Lưu tọa độ vào lộ trình đã đi  self.move\_history.append((self.x, self.y)) # Lưu vào lịch sử di chuyển  def play\_sound\_after\_reverse(self):  # Hàm phát âm thanh khi lùi bước  pygame.mixer.init() # Khởi tạo mixer của pygame  # Phát nhạc 'trungchuongngaivat.mp3' khi gặp chướng ngại vật  pygame.mixer.music.load("trungchuongngaivat.mp3")  pygame.mixer.music.play()  # Chờ 3 giây cho nhạc phát xong  pygame.time.wait(3000)  # Phát tiếp nhạc 'nhac.mp3' sau khi chướng ngại vật được vượt qua  pygame.mixer.music.load("nhac.mp3")  pygame.mixer.music.play() |

#### *Algorithms.py*

|  |
| --- |
| # Algorithms.py  import pygame  from Maze import Maze  from pygame.locals import \*  from queue import PriorityQueue  from queue import PriorityQueue  import os  def dijkstra(maze):  # Thuật toán Dijkstra để tìm đường đi ngắn nhất trong mê cung  start\_node = (maze.start\_x, maze.start\_y) # Điểm bắt đầu  end\_node = (maze.end\_x, maze.end\_y) # Điểm kết thúc  frontier = PriorityQueue() # Hàng đợi ưu tiên để lưu các đỉnh cần kiểm tra  frontier.put(start\_node, 0) # Thêm điểm bắt đầu vào hàng đợi ưu tiên với độ ưu tiên là 0  came\_from = {} # Dùng để lưu lại đỉnh trước của mỗi đỉnh  cost\_so\_far = {} # Dùng để lưu lại chi phí tới mỗi đỉnh  came\_from[start\_node] = None # Khởi tạo điểm bắt đầu không có điểm trước đó  cost\_so\_far[start\_node] = 0 # Khởi tạo chi phí tới điểm bắt đầu là 0  # Bắt đầu vòng lặp chính của thuật toán  while not frontier.empty():  current = frontier.get() # Lấy ra đỉnh có độ ưu tiên nhỏ nhất từ hàng đợi  if current == end\_node: # Nếu đỉnh hiện tại là điểm kết thúc, thoát khỏi vòng lặp  break  # Duyệt qua các đỉnh kề của đỉnh hiện tại  # Kiểm tra nếu đỉnh kề tiếp theo chưa được xem xét trước đó hoặc chi phí mới để đến đỉnh này nhỏ hơn chi phí đã biết trước đó  for next in maze.get\_neighbors(current[0], current[1]):  # Nếu điều kiện trên đúng, ta cập nhật chi phí mới và thêm đỉnh này vào hàng đợi ưu tiên để xem xét sau này  new\_cost = cost\_so\_far[current] + 1 # Tính chi phí mới đến đỉnh kề  # Kiểm tra xem ô tiếp theo chưa được xem xét trước đó hoặc chi phí mới để đến ô này nhỏ hơn chi phí đã biết trước đó  if next not in cost\_so\_far or new\_cost < cost\_so\_far[next]:  cost\_so\_far[next] = new\_cost # Cập nhật chi phí mới cho đỉnh kề  priority = new\_cost + heuristic(end\_node, next) # Tính độ ưu tiên mới  frontier.put(next, priority) # Thêm đỉnh kề vào hàng đợi với độ ưu tiên mới  came\_from[next] = current # Lưu lại đỉnh trước của đỉnh kề  # Sau khi thuật toán kết thúc, ta lấy đường đi từ điểm kết thúc về điểm bắt đầu  current = end\_node  path = []  while current != start\_node:  path.append(current)  current = came\_from[current]  path.append(start\_node)  path.reverse() # Đảo ngược đường đi để được theo thứ tự từ điểm bắt đầu đến điểm kết thúc  return path  def heuristic(a, b):  # Hàm heuristic để ước lượng khoảng cách từ một điểm đến điểm khác  return abs(a[0] - b[0]) + abs(a[1] - b[1])  def play\_next\_music():  # Hàm để phát nhạc tiếp theo  # Lấy đường dẫn tương đối tới tệp nhạc  current\_dir = os.path.dirname(\_\_file\_\_) # Thư mục chứa tệp mã nguồn  music\_path = os.path.join(current\_dir, "Music", "nhac.mp3") # Tệp nhạc nằm trong thư mục "Music" |

#### *ControlPanel.py*

|  |
| --- |
| # ControlPanel.py  import pygame  from Maze import Maze  from pygame.locals import \*  from queue import PriorityQueue  class ControlPanel:  def \_\_init\_\_(self, width, height):  self.background\_image = pygame.image.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Picture/selectbackground.jpg")  self.background\_image = pygame.transform.scale(self.background\_image, (300, 500)) # Chỉnh kích thước hình nền  self.font = pygame.font.Font('freesansbold.ttf', 24)  self.direction\_keys = {K\_UP: (0, -1), K\_DOWN: (0, 1), K\_LEFT: (-1, 0), K\_RIGHT: (1, 0)}  self.current\_direction = None  self.move\_continuous = False  self.auto\_play = False # Thêm thuộc tính auto\_play để theo dõi trạng thái auto play  self.levels\_displayed = False # Thêm thuộc tính để theo dõi trạng thái hiển thị của các mức độ    def reset\_icon\_position(self, ai):  ai.x, ai.y = ai.start\_x, ai.start\_y # Đặt lại vị trí của icon về vị trí ban đầu  def handle\_continuous\_movement(self, maze, ai):  if self.move\_continuous and self.current\_direction:  dx, dy = self.current\_direction  # Kiểm tra xem có thể di chuyển đến ô tiếp theo không  if maze.grid[ai.y + dy][ai.x + dx] == 0:  ai.move\_towards(ai.x + dx, ai.y + dy) # Di chuyển  return maze, ai  # Phương thức để hiển thị giao diện  def display(self, screen):  # Hiển thị hình nền và các nút chức năng  bg\_width, bg\_height = self.background\_image.get\_size() # Lấy kích thước của hình nền  x = 700 # Vị trí căn lề phải  y = 0  screen.blit(self.background\_image, (x, y)) # Vẽ hình nền  text\_color = (255, 255, 255) # Màu chữ  # Tạo các văn bản cho các nút  auto\_play\_text = self.font.render("Auto Play", True, text\_color) # Tạo văn bản cho nút "Auto Play"  ai\_play\_text = self.font.render("AI Play", True, text\_color) # Tạo văn bản cho nút "AI Play"  reset\_text = self.font.render("Reset Maze", True, text\_color) # Tạo văn bản cho nút "Reset Maze"  exit\_text = self.font.render("Exit", True, text\_color) # Tạo văn bản cho nút "Exit"  # Màu và hiệu ứng nút khi di chuột qua  button\_color = (0, 153, 204) # Màu nền của nút  hover\_color = (0, 102, 153) # Màu nền của nút khi di chuột qua  # Vẽ các nút và viền  buttons = [  (750, 90, 200, 50, auto\_play\_text), # Nút "Auto Play"  (750, 190, 200, 50, ai\_play\_text), # Nút "AI Play"  (750, 290, 200, 50, reset\_text), # Nút "Reset Maze"  (750, 390, 200, 50, exit\_text), # Nút "Exit"  ]  mouse\_x, mouse\_y = pygame.mouse.get\_pos() # Lấy vị trí của chuột  # Vẽ nút và xác định vị trí của chuột  for button in buttons:  x, y, width, height, text = button  # Kiểm tra vị trí của chuột để thay đổi màu của nút  if x < mouse\_x < x + width and y < mouse\_y < y + height:  pygame.draw.rect(screen, hover\_color, (x, y, width, height)) # Màu nền khi di chuột qua  else:  pygame.draw.rect(screen, button\_color, (x, y, width, height)) # Màu nền mặc định của nút  # Vẽ viền của nút  pygame.draw.rect(screen, (255, 255, 255), (x, y, width, height), 1)  # Vị trí văn bản căng giữa trên nút  text\_rect = text.get\_rect(center=(x + width // 2, y + height // 2))  screen.blit(text, text\_rect)  # Xử lý các sự kiện từ chuột và bàn phím  def handle\_event(self, event, maze, ai):  if event.type == pygame.MOUSEBUTTONDOWN:  mouse\_x, mouse\_y = pygame.mouse.get\_pos() # Lấy vị trí chuột khi nhấn  # Kiểm tra xem chuột có nhấn vào nút "Auto Play" không  if 750 < mouse\_x < 950 and 50 < mouse\_y < 100:  self.auto\_play = True # Kích hoạt chế độ tự động chơi  self.reset\_icon\_position(ai) # Đặt lại vị trí của biểu tượng AI về vị trí ban đầu  # Kiểm tra xem chuột có nhấn vào nút "AI Play" không  elif 750 < mouse\_x < 950 and 150 < mouse\_y < 200:  self.auto\_play = False # Tắt chế độ tự động chơi  self.reset\_icon\_position(ai) # Đặt lại vị trí của biểu tượng AI về vị trí ban đầu  elif event.type == KEYDOWN:  if event.key in self.direction\_keys:  dx, dy = self.direction\_keys[event.key]  # Kiểm tra ô tiếp theo có là chướng ngại vật không  if maze.grid[ai.y + dy][ai.x + dx] == 0:  self.current\_direction = (dx, dy)  self.move\_continuous = True  if self.auto\_play:  maze, ai = self.handle\_continuous\_movement(maze, ai)  elif event.type == KEYUP:  if event.key in self.direction\_keys:  if self.current\_direction == self.direction\_keys[event.key]:  self.move\_continuous = False  self.current\_direction = None  return maze, ai |

#### *Main.py*

|  |
| --- |
| import pygame  from Maze import Maze  from pygame.locals import \*  from queue import PriorityQueue  import time  from ControlPanel import ControlPanel  from AI import AI  from Algorithms import dijkstra, heuristic  def main():  pygame.init() # Khởi tạo pygame  width, height = 1000, 500 # Đặt kích thước cửa sổ game mini sau Main chính  screen = pygame.display.set\_mode((width, height)) # Tạo cửa sổ game với kích thước đã định  pygame.mixer.init() # Khởi tạo mixer cho âm thanh  pygame.mixer.music.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Music/nhac.mp3") # Tải nhạc từ đường dẫn cụ thể  pygame.mixer.music.set\_volume(1.0) # Đặt âm lượng của nhạc  music\_start\_time = None # Khởi tạo biến để lưu thời gian bắt đầu phát nhạc  control\_panel = ControlPanel(300, 500) # Khởi tạo ControlPanel với kích thước  maze = Maze(35, 25) # Khởi tạo mê cung với số hàng và số cột  ai = AI(maze) # Khởi tạo trí tuệ nhân tạo (AI) với mê cung đã tạo  auto\_play = False # Biến để theo dõi trạng thái tự động chơi  shortest\_path = None # Biến để lưu đường đi ngắn nhất  ai.maze = maze # Thiết lập mê cung cho đối tượng AI  start\_time = None # Biến để lưu thời gian bắt đầu chơi  countdown\_time = 120 # Thời gian đếm ngược ban đầu là 120 giây  running = True # Biến để theo dõi trạng thái của trò chơi (chạy hoặc dừng)  game\_over = False # Biến để theo dõi trạng thái kết thúc của trò chơi  game\_over\_time = None # Biến để lưu thời gian khi trò chơi kết thúc  maze\_completed = False # Biến để theo dõi trạng thái hoàn thành của mê cung  congrats\_display\_time = None # Biến để lưu thời gian hiển thị thông báo chúc mừng  game\_over\_display\_time = None # Biến để lưu thời gian hiển thị thông báo game over  while running:  current\_time = pygame.time.get\_ticks()  if ai.x == maze.end\_x and ai.y == maze.end\_y:  maze\_completed = True  for event in pygame.event.get():  if event.type == QUIT:  running = False  elif event.type == MOUSEBUTTONDOWN:  maze, ai = control\_panel.handle\_event(event, maze, ai)  mouse\_x, mouse\_y = pygame.mouse.get\_pos()  if 750 <= mouse\_x <= 950 and 50 <= mouse\_y <= 100:  auto\_play = True  start\_time = time.time()  pygame.mixer.music.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Music/nhac.mp3") # Tải nhạc từ đường dẫn  pygame.mixer.music.play() # Phát nhạc khi người dùng nhấn vào Auto Play  elif 750 <= mouse\_x <= 950 and 150 <= mouse\_y <= 200:  auto\_play = True  shortest\_path = dijkstra(maze)  start\_time = time.time()  pygame.mixer.music.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Music/nhac.mp3") # Tải nhạc từ đường dẫn  pygame.mixer.music.play() # Phát nhạc khi người dùng nhấn vào AI Play  elif 750 <= mouse\_x <= 950 and 250 <= mouse\_y <= 300:  maze = Maze(35, 25)  ai = AI(maze) # Cập nhật đối tượng AI cho mê cung mới  auto\_play = False  shortest\_path = None  start\_time = None # Đặt lại thời gian về None khi reset mê cung  game\_over = False # Đặt lại trạng thái game\_over về False khi reset mê cung  game\_over\_time = None # Đặt lại thời gian kết thúc về None khi reset mê cung  pygame.mixer.music.stop()  elif 750 <= mouse\_x <= 950 and 350 <= mouse\_y <= 400:  running = False  elif event.type == KEYDOWN and auto\_play:  if not game\_over: # Thêm điều kiện này để ngăn chặn di chuyển sau khi game over  if event.key == K\_UP:  ai.move\_towards(ai.x, ai.y - 1)  elif event.key == K\_DOWN:  ai.move\_towards(ai.x, ai.y + 1)  elif event.key == K\_LEFT:  ai.move\_towards(ai.x - 1, ai.y)  elif event.key == K\_RIGHT:  ai.move\_towards(ai.x + 1, ai.y)    # Kiểm tra xem người chơi hoặc con AI đã đến điểm cuối hay chưa  if ai.x == maze.end\_x and ai.y == maze.end\_y:  if not game\_over:  game\_over\_time = time.time() # Lưu thời điểm khi đạt đến điểm cuối  game\_over = True # Đặt trạng thái game\_over thành True  total\_time = game\_over\_time - start\_time # Tính tổng thời gian chơi  start\_time = None # Dừng đếm thời gian  # Tải và phát nhạc khi đến đích  pygame.mixer.music.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Music/dendich.mp3")  pygame.mixer.music.play()  # Bắt đầu hiển thị thông báo chúc mừng  congrats\_display\_time = pygame.time.get\_ticks()    screen.fill((0, 0, 0))  maze.display\_maze(screen, ai)  control\_panel.display(screen)  # Thay vì vẽ hình tròn, bạn sẽ vẽ hình ảnh mới tại vị trí hiện tại của đối tượng di chuyển  if auto\_play:  if shortest\_path:  if len(shortest\_path) > 1:  next\_step = shortest\_path.pop(1)  ai.move\_towards(next\_step[0], next\_step[1])  # Vẽ lại icon tại vị trí mới  player\_image = pygame.image.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Picture/icon.png")  cell\_size = 20  player\_image = pygame.transform.scale(player\_image, (cell\_size, cell\_size))  screen.blit(player\_image, (ai.x \* cell\_size, ai.y \* cell\_size))  rewards\_to\_remove = [] # Danh sách để lưu các phần tử cần loại bỏ  play\_reward\_music = False # Biến đánh dấu khi cần phát nhạc khi di chuyển trúng phần thưởng  for reward in maze.rewards:  if (ai.x, ai.y) == reward:  start\_time += 3 # Cộng thêm 3 giây vào thời gian đếm ngược  rewards\_to\_remove.append(reward) # Thêm phần tử cần loại bỏ vào danh sách tạm thời  play\_reward\_music = True # Đặt biến đánh dấu để phát nhạc khi di chuyển trúng phần thưởng  # Loại bỏ các phần tử đã xử lý khỏi maze.rewards  for reward in rewards\_to\_remove:  maze.rewards.remove(reward)    # Phát nhạc khi di chuyển trúng phần thưởng  if play\_reward\_music:  try:  # Kiểm tra xem mixer đã khởi động chưa  if not pygame.mixer.get\_init():  pygame.mixer.init() # Khởi tạo mixer nếu chưa  # Phát nhạc nền (sử dụng pygame.mixer.music)  pygame.mixer.music.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Music/nhac.mp3")  pygame.mixer.music.play(-1) # -1 để lặp vô hạn  # Tạo sound object cho nhạc phần thưởng  reward\_sound = pygame.mixer.Sound("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Music/trungthuong.mp3")    # Phát nhạc phần thưởng (không làm gián đoạn nhạc nền)  reward\_sound.play()  # Bạn có thể điều chỉnh âm lượng nếu muốn  reward\_sound.set\_volume(0.8) # Giảm âm lượng nhạc phần thưởng  pygame.mixer.music.set\_volume(0.5) # Giảm âm lượng nhạc nền  except pygame.error as e:  print(f"Error playing music: {e}")    if start\_time is not None and not game\_over:  elapsed\_time = countdown\_time - int(time.time() - start\_time) # Thời gian còn lại được tính bằng cách trừ thời gian đã trôi qua từ 60  elapsed\_time = max(0, elapsed\_time) # Đảm bảo thời gian không bị âm  # Hiển thị thời gian đếm ngược  font = pygame.font.Font(None, 36)  time\_text = font.render(f"TIME: {elapsed\_time} S", True, (0, 0, 0))  text\_rect = time\_text.get\_rect()  text\_rect.topright = (width - 100, 10) # Đặt vị trí của thời gian  screen.blit(time\_text, text\_rect)  # Kiểm tra và thông báo game over nếu hết thời gian mà icon không đạt đến điểm cuối  if elapsed\_time == 0 and not maze\_completed:  game\_over = True  game\_over\_time = time.time() # Lưu thời điểm khi game over  game\_over\_display\_time = pygame.time.get\_ticks() # Lưu thời điểm hiển thị thông báo game over  # Tải và phát nhạc khi game over  pygame.mixer.music.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Music/gameover.mp3")  pygame.mixer.music.play()    # Kiểm tra và vẽ thông báo chúc mừng nếu cần  # Trong phần hiển thị thông báo Chúc mừng  if congrats\_display\_time is not None:  elapsed\_congrats\_time = current\_time - congrats\_display\_time  if elapsed\_congrats\_time < 1000: # Hiển thị trong 3 giây  # Tạo hình nền màu đẹp hơn  pygame.draw.rect(screen, (0, 0, 0), (width // 2 - 150, height // 2 - 50, 300, 100)) # Màu xanh lá cây  font = pygame.font.Font(None, 48) # Kích thước chữ lớn hơn  congrats\_text = font.render("Congratulations!", True, (255, 255, 255)) # Chữ trắng trên nền xanh lá cây  text\_rect = congrats\_text.get\_rect(center=(width // 2, height // 2))  screen.blit(congrats\_text, text\_rect)  else:  congrats\_display\_time = None # Ẩn thông báo sau khi đã hiển thị trong khoảng thời gian nhất định  # Kiểm tra và vẽ thông báo game over nếu cần  if game\_over\_display\_time is not None:  elapsed\_game\_over\_time = current\_time - game\_over\_display\_time  if elapsed\_game\_over\_time < 1000: # Hiển thị thông báo trong 3 giây  # Tạo hình nền màu đẹp hơn  pygame.draw.rect(screen, (0, 0, 0), (width // 2 - 150, height // 2 - 50, 300, 100)) # Màu đỏ  font = pygame.font.Font(None, 48) # Kích thước chữ lớn hơn  game\_over\_text = font.render("Game Over!", True, (255, 255, 255)) # Chữ trắng trên nền đỏ  text\_rect = game\_over\_text.get\_rect(center=(width // 2, height // 2))  screen.blit(game\_over\_text, text\_rect)  else:  game\_over\_display\_time = None # Ẩn thông báo sau khi đã hiển thị trong khoảng thời gian nhất định    pygame.display.flip()  pygame.time.delay(100)  pygame.quit()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

#### *Maze.py*

|  |
| --- |
| import pygame  import random  class Maze:  def \_\_init\_\_(self, width, height):  self.width = width  self.height = height  self.grid = [[1] \* width for \_ in range(height)]  self.start\_x, self.start\_y = 1, 1  self.end\_x, self.end\_y = width - 2, height - 2  self.generate\_maze()  self.grid[self.start\_y][self.start\_x] = 0  self.grid[self.end\_y][self.end\_x] = 0  self.path = []  self.obstacles = set()  self.create\_obstacles()  self.load\_images()  self.rewards = set()  self.create\_rewards()  def load\_images(self):  cell\_size = 20  self.obstacle\_image = pygame.transform.scale(pygame.image.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Picture/iconbom.png"), (cell\_size, cell\_size))  self.reward\_image = pygame.transform.scale(pygame.image.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Picture/phanthuong.jpg"), (cell\_size, cell\_size))  self.end\_image = pygame.transform.scale(pygame.image.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Picture/dautay.jpg"), (cell\_size, cell\_size))  self.wall\_image = pygame.transform.scale(pygame.image.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Picture/hangrao.jpg"), (cell\_size, cell\_size))  def create\_rewards(self):  reward\_count = 0  while reward\_count < 10:  x = random.randint(1, self.width - 2)  y = random.randint(1, self.height - 2)  if self.grid[y][x] == 0 and (x, y) not in self.rewards and not self.has\_adjacent\_reward(x, y):  self.rewards.add((x, y))  reward\_count += 1  def has\_adjacent\_reward(self, x, y):  for dy in range(-1, 2):  for dx in range(-1, 2):  if (dx != 0 or dy != 0) and (x + dx, y + dy) in self.rewards:  return True  return False    def create\_obstacles(self):  obstacle\_count = 0  while obstacle\_count < 20:  x = random.randint(1, self.width - 2)  y = random.randint(1, self.height - 2)  if self.grid[y][x] == 1 and (x, y) != (self.start\_x, self.start\_y) and (x, y) != (self.end\_x, self.end\_y) and self.is\_far\_from\_obstacles(x, y):  self.obstacles.add((x, y))  obstacle\_count += 1  def is\_far\_from\_obstacles(self, x, y):  min\_distance = 5  for obstacle in self.obstacles:  obstacle\_x, obstacle\_y = obstacle  distance = abs(x - obstacle\_x) + abs(y - obstacle\_y)  if distance < min\_distance:  return False  return True  def generate\_maze(self):  for x in range(self.width):  self.grid[0][x] = 1  self.grid[self.height - 1][x] = 1  for y in range(self.height):  self.grid[y][0] = 1  self.grid[y][self.width - 1] = 1    stack = [(1, 1)]  while stack:  x, y = stack[-1]  neighbors = [(x + dx, y + dy) for dx, dy in [(0, -2), (0, 2), (-2, 0), (2, 0)] if 0 < x + dx < self.width - 1 and 0 < y + dy < self.height - 1]  unvisited\_neighbors = [neighbor for neighbor in neighbors if self.grid[neighbor[1]][neighbor[0]] == 1]  if unvisited\_neighbors:  nx, ny = random.choice(unvisited\_neighbors)  self.grid[ny][nx] = 0  self.grid[y + (ny - y) // 2][x + (nx - x) // 2] = 0  stack.append((nx, ny))  else:  stack.pop()  def display\_maze(self, screen, ai):  cell\_size = 20  START\_COLOR = (135, 206, 250) # Màu xanh da trời nhạt (Sky Blue) - Điểm bắt đầu (không thay đổi)  END\_COLOR = (34, 139, 34) # Màu xanh lá cây rừng (Forest Green) - Điểm kết thúc  PLAYER\_PATH\_COLOR = (255, 99, 71) # Màu cà chua (Tomato) - Đường đi của người chơi  AI\_PATH\_COLOR = (100, 149, 237) # Màu xanh ngọc bích (Cornflower Blue) - Đường đi của AI  OBSTACLE\_COLOR = (80, 80, 80) # Màu xám đậm vừa (Dim Gray) - Chướng ngại vật  REWARD\_COLOR = (255, 215, 0) # Màu vàng kim (Gold) - Phần thưởng  WALL\_COLOR = (112, 128, 144) # Màu xanh xám (Slate Gray) - Tường  EMPTY\_COLOR = (245, 245, 245) # Màu trắng khói (White Smoke) - Ô trống  VISITED\_COLOR = (144, 238, 144) # Màu xanh nhạt (Light Green) - Đã ghé thăm (không thay đổi)  for y in range(self.height):  for x in range(self.width):  rect = pygame.Rect(x \* cell\_size, y \* cell\_size, cell\_size, cell\_size)  if (x, y) == (self.start\_x, self.start\_y):  pygame.draw.rect(screen, START\_COLOR, rect)  elif (x, y) == (self.end\_x, self.end\_y):  screen.blit(self.end\_image, rect.topleft)  elif (x, y) in self.path:  pygame.draw.rect(screen, PLAYER\_PATH\_COLOR, rect)  elif (x, y) in ai.path:  pygame.draw.rect(screen, AI\_PATH\_COLOR, rect)  elif (x, y) in self.obstacles:  screen.blit(self.obstacle\_image, rect.topleft)  elif (x, y) in self.rewards:  screen.blit(self.reward\_image, rect.topleft)  elif self.grid[y][x] == 1:  screen.blit(self.wall\_image, rect.topleft)  else:  pygame.draw.rect(screen, EMPTY\_COLOR, rect)  path\_color = VISITED\_COLOR  if ai.x == self.end\_x and ai.y == self.end\_y:  for step in ai.path:  x, y = step  pygame.draw.rect(screen, path\_color, pygame.Rect(x \* cell\_size, y \* cell\_size, cell\_size, cell\_size))  else:  for step in self.path:  x, y = step  pygame.draw.rect(screen, path\_color, pygame.Rect(x \* cell\_size, y \* cell\_size, cell\_size, cell\_size))  def get\_neighbors(self, x, y):  neighbors = []  for dx, dy in [(0, -1), (0, 1), (-1, 0), (1, 0)]:  nx, ny = x + dx, y + dy  if 0 <= nx < self.width and 0 <= ny < self.height and self.grid[ny][nx] == 0:  neighbors.append((nx, ny))  return neighbors |

#### *Welcome.py (File chính để chạy game)*

|  |
| --- |
| import pygame  import sys  import subprocess  # Khởi tạo Pygame  pygame.init()  # Đường dẫn đến file nhạc nền  music\_path = "C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Music/nenstart.mp3"  # Đường dẫn đến file âm thanh hiệu ứng cho nút START  start\_sound\_path = "C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Music/button\_click.mp3"  # Kích thước màn hình  SCREEN\_WIDTH = 800  SCREEN\_HEIGHT = 400  screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))  pygame.display.set\_caption("WELCOME THE MAZE GAME")  # Phát nhạc nền  pygame.mixer.music.load(music\_path)  pygame.mixer.music.play(-1) # Phát liên tục  # Tải âm thanh hiệu ứng  start\_sound = pygame.mixer.Sound(start\_sound\_path)  # Màu sắc  WHITE = (255, 255, 255)  BLACK = (0, 0, 0)  # Font  font = pygame.font.SysFont(None, 36)  # Định nghĩa biến start\_button ở đây để trở thành biến toàn cục và căn giữa màn hình  button\_width = 200  button\_height = 50  start\_button\_x = (SCREEN\_WIDTH - button\_width) // 2  start\_button\_y = (SCREEN\_HEIGHT - button\_height) // 2  start\_button = pygame.Rect(start\_button\_x, start\_button\_y, button\_width, button\_height)  def draw\_start\_screen():  # Load hình nền  background = pygame.image.load("C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Picture/wellcome.jpg").convert()  background = pygame.transform.scale(background, (SCREEN\_WIDTH, SCREEN\_HEIGHT))    # Vẽ hình nền  screen.blit(background, (0, 0))    # Vẽ văn bản và nút bắt đầu trên hình nền  text = font.render("Welcome to the maze game", True, BLACK)  text\_rect = text.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2 - 50))  screen.blit(text, text\_rect)    pygame.draw.rect(screen, BLACK, start\_button)  start\_text = font.render("START", True, WHITE)  start\_text\_rect = start\_text.get\_rect(center=start\_button.center)  screen.blit(start\_text, start\_text\_rect)  pygame.display.flip()  def main():  running = True  while running:  for event in pygame.event.get():  if event.type == pygame.QUIT:  running = False  elif event.type == pygame.MOUSEBUTTONDOWN:  if start\_button.collidepoint(event.pos):  # Phát âm thanh khi nút START được nhấp  start\_sound.play()  # Chạy tệp main.py  main\_py\_path = r"C:/Users/ASUS/Desktop/Yomost File/CDIO/GameTTNT/src/Code/Main.py"  subprocess.Popen(["python", "-u", main\_py\_path])  # Thoát khỏi tệp hiện tại  pygame.quit()  sys.exit()  draw\_start\_screen()  pygame.quit()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

### *1.3.* *Kiểm thử, đánh giá kết quả*

\*Nội dung chi tiết được trình bày tại slide PDF của dự án\*

# KẾT LUẬN

1. **Kết quả đạt được**

Trong quá trình thực hiện đồ án "Sử dụng thuật toán Dijkstra ứng dụng vào AI tạo ra trò chơi thoát khỏi mê cung", chúng em đã đạt được nhiều kết quả tích cực, phản ánh sự tiến bộ và nỗ lực của cả nhóm trong việc phát triển một hệ thống trò chơi hoàn chỉnh. Những kết quả này không chỉ đạt được về mặt kỹ thuật mà còn có ý nghĩa trong việc nâng cao kỹ năng lập trình, tư duy thuật toán, và khả năng làm việc nhóm. Cụ thể, các kết quả đã đạt được bao gồm:

1. *Triển khai thành công thuật toán Dijkstra:*

Chúng em đã áp dụng thuật toán Dijkstra một cách hiệu quả để tìm ra đường đi ngắn nhất trong mê cung từ vị trí xuất phát đến lối thoát. Quá trình triển khai này không chỉ thành công về mặt kỹ thuật mà còn giúp chúng em hiểu sâu hơn về cách thuật toán hoạt động trong thực tế, từ việc lựa chọn các đỉnh trong đồ thị mê cung, tính toán khoảng cách, đến việc tối ưu hóa lộ trình di chuyển của nhân vật. Thuật toán này đã được kiểm tra trên nhiều cấu trúc mê cung khác nhau, từ những mê cung đơn giản đến các mê cung phức tạp với nhiều chướng ngại vật. Kết quả cho thấy thuật toán luôn tìm ra con đường tối ưu nhất mà không gặp bất kỳ lỗi nào, đảm bảo tính chính xác và độ tin cậy cao.

1. *Tích hợp trí tuệ nhân tạo (AI):*

Một thành tựu lớn của dự án là việc tích hợp trí tuệ nhân tạo vào trò chơi. AI đã giúp tạo ra nhân vật không phải người chơi (NPC), có khả năng tự động di chuyển và ra quyết định dựa trên thuật toán Dijkstra. NPC này sẽ phản ứng nhanh chóng trước các tình huống trong trò chơi, như tìm đường đi đến mục tiêu hoặc tránh chướng ngại vật, tăng tính thử thách và hấp dẫn cho người chơi. AI cũng được lập trình để điều chỉnh độ khó của trò chơi, bằng cách thay đổi mức độ phức tạp của mê cung và tốc độ di chuyển của NPC. Điều này giúp trò chơi trở nên linh hoạt và phù hợp với nhiều đối tượng người chơi, từ những người mới bắt đầu đến những người chơi có kinh nghiệm.

1. *Phát triển một trò chơi hoàn chỉnh với giao diện đồ họa:*

Trò chơi thoát khỏi mê cung đã được xây dựng thành công với giao diện đồ họa 2D, sử dụng thư viện Pygame. Dù đồ họa không quá phức tạp, nhưng vẫn đáp ứng được mục tiêu ban đầu là tạo ra một trò chơi trực quan và dễ sử dụng. Các yếu tố như nhân vật, mê cung, chướng ngại vật, và phần thưởng đều được hiển thị một cách rõ ràng, dễ nhìn, giúp người chơi dễ dàng nhận biết và tương tác. Giao diện trò chơi được thiết kế đơn giản nhưng thân thiện, phù hợp với nhiều lứa tuổi. Người chơi có thể dễ dàng điều khiển nhân vật chính di chuyển trong mê cung, trải nghiệm các cấp độ khác nhau và tương tác với AI một cách tự nhiên.

1. *Cải thiện khả năng lập trình và tư duy thuật toán:*

Qua quá trình phát triển, chúng em đã cải thiện đáng kể khả năng lập trình Python, đặc biệt là trong việc áp dụng các thuật toán phức tạp vào giải quyết bài toán thực tế. Việc triển khai thuật toán Dijkstra không chỉ yêu cầu kiến thức lý thuyết mà còn đòi hỏi khả năng giải quyết các vấn đề kỹ thuật, từ việc tối ưu hóa mã nguồn đến việc xử lý tình huống phát sinh trong quá trình lập trình. Đồng thời, quá trình làm việc cũng giúp chúng em nâng cao tư duy thuật toán, đặc biệt là trong việc xử lý các bài toán tối ưu hóa như tìm đường đi ngắn nhất trong mê cung. Chúng em đã học được cách tiếp cận vấn đề một cách hệ thống, từ việc phân tích bài toán, thiết kế thuật toán, đến triển khai và kiểm thử.

1. *Tích hợp âm thanh và hình ảnh cho trò chơi:*

Ngoài các yếu tố logic và thuật toán, chúng em đã thành công trong việc tích hợp âm thanh và hình ảnh vào trò chơi, giúp tăng cường trải nghiệm người chơi. Mỗi hành động và sự kiện trong trò chơi đều đi kèm với âm thanh, tạo cảm giác sống động và chân thực hơn. Hình ảnh đại diện cho các đối tượng như phần thưởng, chướng ngại vật, cũng được thiết kế đơn giản nhưng hiệu quả, giúp trò chơi trở nên trực quan và dễ hiểu. Các yếu tố âm thanh như âm thanh khi nhân vật nhặt phần thưởng, va chạm với chướng ngại vật hay hoàn thành trò chơi đều được lập trình và đồng bộ hóa với các hành động trong game, giúp tạo cảm giác hấp dẫn và lôi cuốn hơn cho người chơi.

1. *Hoàn thiện hệ thống điều khiển và quản lý logic trò chơi:*

Hệ thống điều khiển của trò chơi đã được hoàn thiện với các chức năng cơ bản như khởi động trò chơi, điều khiển nhân vật, quản lý điểm số và kết thúc trò chơi. Người chơi có thể dễ dàng tương tác với trò chơi thông qua các phím di chuyển và các thao tác đơn giản khác, tạo nên một trải nghiệm mượt mà và thân thiện với người dùng. Chúng em cũng đã lập trình các tính năng quản lý logic của trò chơi, bao gồm việc xác định khi nào trò chơi kết thúc, khi nào người chơi thắng hoặc thua, và tính toán điểm số dựa trên thời gian và phần thưởng người chơi nhặt được trong mê cung.

1. **HẠN CHẾ CỦA ĐỀ TÀI**

Mặc dù đã đạt được nhiều thành quả tích cực, đồ án vẫn tồn tại một số hạn chế nhất định mà chúng em nhận thấy có thể cải thiện trong tương lai. Những hạn chế này chủ yếu liên quan đến khả năng mở rộng, hiệu suất và độ phức tạp của hệ thống. Cụ thể, các hạn chế bao gồm:

1. *Đồ họa chưa đủ phức tạp và hấp dẫn:*

Đồ họa của trò chơi vẫn còn khá đơn giản và chưa đạt đến mức độ chi tiết hay thẩm mỹ cao. Mặc dù việc sử dụng Pygame đã đáp ứng được các yêu cầu cơ bản về đồ họa, nhưng trò chơi thiếu đi các yếu tố đồ họa hiện đại và chân thực hơn như hiệu ứng ánh sáng, bóng đổ hay các chi tiết động khác. Hơn nữa, giao diện và thiết kế của trò chơi chưa có sự đa dạng về chủ đề hay phong cách, khiến trải nghiệm thị giác của người chơi đôi khi trở nên đơn điệu.

1. *AI còn hạn chế về khả năng học hỏi và phản ứng:*

Mặc dù AI đã giúp trò chơi trở nên tự động hóa và tương tác thông minh hơn, nhưng vẫn còn khá hạn chế ở mức độ cơ bản. AI chưa có khả năng học hỏi từ các hành động của người chơi, cũng như chưa thể thay đổi chiến lược hay đưa ra quyết định phức tạp hơn dựa trên hành vi của người chơi. Trong tương lai, AI có thể được cải tiến để trở nên "thông minh" hơn, như tự động học từ các lần chơi trước và điều chỉnh độ khó của trò chơi một cách linh hoạt hơn.

1. *Khả năng mở rộng của trò chơi còn hạn chế:*

Trò chơi hiện tại chỉ hỗ trợ một số loại mê cung với cấu trúc và kích thước cố định. Mặc dù đã có thể điều chỉnh độ khó của trò chơi thông qua AI, nhưng hệ thống chưa có khả năng tự động tạo ra các mê cung với độ phức tạp khác nhau một cách linh hoạt, điều này giới hạn sự đa dạng trong trải nghiệm của người chơi. Việc mở rộng trò chơi để bao gồm nhiều loại mê cung hơn, với các yếu tố và tính năng phức tạp khác sẽ cần phải được nghiên cứu và phát triển thêm.

1. *Hiệu suất khi xử lý mê cung lớn còn hạn chế:*

Khi kích thước mê cung hoặc số lượng chướng ngại vật trong trò chơi tăng lên, hiệu suất của hệ thống sẽ bị ảnh hưởng, đặc biệt khi thuật toán Dijkstra phải tính toán trên các mê cung lớn. Điều này có thể gây ra tình trạng trễ hoặc giảm tốc độ phản hồi của hệ thống, ảnh hưởng đến trải nghiệm của người chơi. Vấn đề này chủ yếu xảy ra trên các máy tính có cấu hình thấp hoặc khi thử nghiệm với các mê cung phức tạp. Trong tương lai, cần có các biện pháp tối ưu hóa thuật toán và mã nguồn để cải thiện hiệu suất trò chơi.

1. *Thiếu tính năng đa người chơi:*

Hiện tại, trò chơi chỉ hỗ trợ chế độ chơi đơn, trong đó người chơi phải đối mặt với AI để tìm đường thoát khỏi mê cung. Tính năng chơi cùng bạn bè hoặc thi đấu với người chơi khác vẫn chưa được triển khai, làm giảm sự hấp dẫn và tính cạnh tranh của trò chơi. Việc phát triển tính năng đa người chơi có thể làm tăng thêm tính thú vị và thử thách cho trò chơi

# TÀI LIỆU THAM KHẢO

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. (2009). Introduction to Algorithms (3rd Edition). MIT Press.

[2] Edsger W. Dijkstra. (1959). A Note on Two Problems in Connexion with Graphs. Numerische Mathematik.

[3] Slader, M. (2014). Python Programming for the Absolute Beginner (3rd Edition). Cengage Learning.

[4] Pygame Documentation. (2023). <https://www.pygame.org/docs/>

[5] Peter Norvig. (1992). Artificial Intelligence: A Modern Approach. Prentice Hall.

[6] Seth Kenlon. (2019). Build Games with Python and Pygame. Opensource.com. <https://opensource.com/article/19/10/programming-games-pygame>

[7] Richard S. Sutton, Andrew G. Barto. (2018). Reinforcement Learning: An Introduction (2nd Edition). MIT Press.

[8] GeeksforGeeks. (2023). Dijkstra's Shortest Path Algorithm. <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>

[9] Real Python. (2023). Python and Pygame: Build Your Own Game. <https://realpython.com/pygame-a-primer/>

[10] Stuart Russell, Peter Norvig. (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson.