TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐỒ ÁN TỐT NGHIỆP

NGÀNH: KHOA HỌC MÁY TÍNH

ĐỀ TÀI: NGHIÊN CỨU ÁP DỤNG MÔ HÌNH HỌC TĂNG CƯỜNG VÀO BÀI TOÁN GIẢI MÊ CUNG

Giảng viên hướng dẫn : TS.Nguyễn Mạnh Cường

Lớp : KHMT02 – K15

Sinh viên thực hiện : Nguyễn Ngọc Duy

Mã sinh viên : 2020605168

Hà Nội, 2025

MỤC LỤC

MỤC LỤC HÌNH ẢNH

MỤC LỤC BẢNG BIỂU

DANH MỤC TỪ VIẾT TẮT

AI	Artificial Intelligence
RL	Reinforcement Learning
DQN	Deep Q-Network
MDP	Markov Decision Process
BFS	Breadth-First Search

LỜI CẢM ƠN

Trước tiên, tôi xin gửi lời cảm ơn chân thành và sâu sắc đến Tiến Sĩ Nguyễn Mạnh Cường, người đã tận tình hướng dẫn, hỗ trợ và đóng góp những ý kiến quý báu giúp tôi hoàn thành tốt đồ án tốt nghiệp này.

Tôi cũng xin gửi lời cảm ơn đến tất cả các thầy cô đã từng giảng dạy và hỗ trợ tôi trong suốt thời gian học đại học. Chính những bài học, sự tận tâm và truyền cảm hứng từ quý thầy cô là nền tảng giúp tôi hoàn thành đồ án này.

Bên cạnh đó, tôi cũng xin cảm ơn các bạn bè, anh chị và đồng nghiệp đã chia sẻ kiến thức, tài liệu và luôn đồng hành cùng tôi trong suốt quá trình học tập và thực hiện đồ án.

Mặc dù đã cố gắng hoàn thiện, nhưng do thời gian và năng lực còn hạn chế, đồ án chắc chắn không tránh khỏi những thiếu sót. Tôi rất mong nhận được sự góp ý của thầy cô và các bạn để hoàn thiện hơn trong tương lai.

Tôi xin chân trọng cảm ơn!

Sinh viên thực hiện

Nguyễn Ngọc Duy

LỜI NÓI ĐẦU

Trong những năm gần đây, trí tuệ nhân tạo đã và đang trở thành một trong những lĩnh vực công nghệ phát triển nhanh chóng và có tầm ảnh hưởng sâu rộng trên nhiều lĩnh vực của đời sống như y tế, tài chính, giáo dục, giao thông và cả trong lĩnh vực trò chơi, mô phỏng. Một trong những nhánh quan trọng và đầy tiềm năng của AI là học tăng cường (Reinforcement Learning - RL).

Khác với các phương pháp học máy truyền thống, học tăng cường hướng đến việc xây dựng các tác nhân thông minh có khả năng học cách đưa ra quyết định thông qua tương tác với môi trường, từ đó tối ưu hóa hành vi để đạt được mục tiêu cụ thể.

Bài toán mê cung là một ví dụ kinh điển thường được sử dụng để nghiên cứu và đánh giá các thuật toán học tăng cường. Việc tìm đường ra khỏi mê cung một cách tối ưu là một bài toán không chỉ thú vị mà còn mang tính thực tiễn cao, ứng dụng trong robot tự hành, game AI, và các hệ thống điều khiển tự động.

Tuy nhiên, trong thực tế, việc xây dựng một hệ thống có khả năng tự học cách giải quyết mê cung một cách thông minh và hiệu quả vẫn còn gặp nhiều thách thức: từ việc mô hình hóa môi trường, thiết kế phần thưởng phù hợp, đến lựa chọn thuật toán RL tối ưu cho từng loại mê cung cụ thể. Do đó, việc nghiên cứu và triển khai các mô hình học tăng cường trong bài toán giải mê cung là một hướng đi tiềm năng, đóng vai trò như một bước đệm để mở rộng ra các ứng dụng thực tế khác.

Xuất phát từ thực tế đó, tôi đã lựa chọn thực hiện đề tài **"Nghiên cứu áp dụng mô hình học tăng cường vào bài toán giải quyết mê cung"**, với mục tiêu áp dụng các thuật toán học tăng cường hiện đại như Q-Learning, Deep Q-Network (DQN) để xây dựng một hệ thống có khả năng tự động học và giải các bài toán mê cung với độ phức tạp khác nhau.

Việc thực hiện đề tài này không chỉ giúp tôi củng cố và nâng cao kiến thức về AI và học máy, mà còn đóng góp một góc nhìn thực tiễn trong việc ứng dụng

học tăng cường vào các bài toán mô phỏng, định hướng cho các nghiên cứu và phát triển ứng dụng trong tương lai.

Nội dung quyển báo cáo đồ án tốt nghiệp sẽ bao gồm các chương sau

Chương 1: Phát biểu bài toán

Chương này trình bày tổng quan về AI và phát biểu bài toán giải mê cung một dạng bài toán tối ưu đường đi trong môi trường ràng buộc. Mục tiêu là xây dựng một tác nhân có khả năng tự học cách di chuyển từ điểm xuất phát đến đích trong mê cung với số bước đi ít nhất. Chương cũng xác định rõ đầu vào, đầu ra, không gian trạng thái, tập hành động

Chương 2: Các Phương pháp tiếp cận bài toán

Chương này trình bày các nền tảng lý thuyết dùng để giải quyết bài toán, bắt đầu từ những phương pháp cơ bản dựa trên thuật toán. Tiếp theo là phần giới thiệu về lý thuyết học tăng cường, với các khái niệm cốt lõi như: tác nhân, môi trường, trạng thái, hành động, phần thưởng, chính sách và hàm giá trị. Các thuật toán học tăng cường phổ biến như Q-Learning, SARSA và Deep Q-Network (DQN) được phân tích chi tiết, làm cơ sở cho việc áp dụng vào bài toán mê cung. Cuối chương, các ưu điểm và hạn chế của từng thuật toán được so sánh nhằm lựa chọn phương pháp phù hợp nhất cho bối cảnh cụ thể của bài toán.

Chương 3: Thực nghiệm

Chương này trình bày chi tiết quá trình xây dựng môi trường mê cung – nơi tác nhân học hỏi và tương tác trong bối cảnh học tăng cường. Mê cung được thiết kế dưới dạng lưới 2D với các ô đại diện cho đường đi, tường chắn, vị trí bắt đầu và đích đến. Các cấu trúc mê cung được tạo ra từ các thuật toán sinh mê cung phổ biến như DFS (Depth-First Search), Prim và Wilson, cho phép tạo ra cả mê cung hoàn hảo.

Chương cũng mô tả cách biểu diễn trạng thái, hành động, ma trận phần thưởng, và các quy tắc chuyển trạng thái trong môi trường. Môi trường được

triển khai theo phong cách tương tự OpenAI Gym, hỗ trợ việc mô phỏng và huấn luyện các thuật toán học tăng cường.

Bên cạnh đó, chương này trình bày quá trình xây dựng tác nhân và chiến lược hành động, bao gồm lựa chọn chính sách, thiết kế kiến trúc agent và tích hợp các thuật toán như Q-Learning, SARSA và Deep Q-Network (DQN). Các tác nhân được huấn luyện trên nhiều môi trường mê cung. Việc triển khai sử dụng ngôn ngữ lập trình Python cùng các thư viện hỗ trợ như NumPy, Matplotlib, và các công cụ trực quan hóa khác.

Phần cuối chương tập trung vào quá trình huấn luyện, lựa chọn và tinh chỉnh các siêu tham số (hyperparameters), đánh giá hiệu suất thông qua số bước đến đích, tổng phần thưởng nhận được, và biểu diễn trực quan kết quả học thông qua các biểu đồ như heatmap chính sách. Các tác nhân sử dụng thuật toán khác nhau được so sánh để rút ra nhận xét về hiệu quả và tính phù hợp của từng phương pháp trong môi trường mê cung đã xây dựng.

Chương 4: Thiết kế giao diện và đánh giá

Chương mô tả quá trình thiết kế và xây dựng giao diện người dùng, bao gồm yêu cầu chức năng, phi chức năng, cấu trúc thành phần và cách hiển thị thông tin. Giao diện cho phép người dùng trực quan hóa mê cung, theo dõi quá trình tác nhân di chuyển, và lựa chọn thuật toán hoặc cấu hình huấn luyện. Ảnh chụp màn hình được cung cấp để minh họa chi tiết các thành phần của giao diện.

Mô hình đã được huấn luyện sẽ được tích hợp vào giao diện, cho phép chạy thử nghiệm trực tiếp và kiểm tra hiệu suất thực tế. Hệ thống được đánh giá dựa trên nhiều tiêu chí như: số bước trung bình để đến đích, tỷ lệ thành công, tốc độ hội tụ và khả năng tổng quát hóa khi áp dụng lên các mê cung chưa từng thấy.

Cuối cùng, chương đưa ra đánh giá tổng thể về hiệu quả của hệ thống, rút ra kết luận từ các kết quả thực nghiệm, đồng thời đề xuất các hướng phát triển

mở rộng trong tương lai như cải tiến thuật toán, mở rộng môi trường 3D hoặc ứng dụng trong các bài toán thực tiễn hơn.

Phần kết luận:

Đề tài đã hoàn thành việc xây dựng và triển khai một hệ thống học tăng cường giúp tác nhân giải quyết bài toán mê cung, với các thuật toán như Q-Learning, SARSA và DQN. Hệ thống được đánh giá qua nhiều chỉ số như số bước trung bình, tổng phần thưởng, và tỷ lệ thành công trong các môi trường huấn luyện, cho thấy mô hình có khả năng học hiệu quả trong phạm vi các mê cung đã thấy.

Tuy nhiên, kết quả thực nghiệm cũng chỉ ra rằng các mô hình huấn luyện chưa đạt được mức độ tổng quát hóa tốt. Khi áp dụng lên các mê cung chưa từng gặp trong quá trình huấn luyện, hiệu suất suy luận giảm rõ rệt, cho thấy mô hình còn bị phụ thuộc vào đặc trưng của môi trường huấn luyện. Đây là một hạn chế quan trọng và mở ra nhiều hướng nghiên cứu tiếp theo, chẳng hạn như sử dụng kỹ thuật học chuyển tiếp, học mô hình môi trường, hoặc áp dụng các kiến trúc sâu hơn như Actor-Critic, PPO để cải thiện khả năng tổng quát.

Dù còn tồn tại hạn chế, đề tài vẫn mang ý nghĩa học thuật trong việc triển khai và phân tích các thuật toán học tăng cường, đồng thời thể hiện tiềm năng ứng dụng thực tiễn trong các bài toán ra quyết định. Những kết quả ban đầu này có thể làm nền tảng cho các nghiên cứu mở rộng và ứng dụng sâu hơn trong các lĩnh vực như robot tự hành, trò chơi, tối ưu hóa và các hệ thống thông minh trong tương lai.

CHƯƠNG 1: PHÁT BIỂU BÀI TOÁN

Nội dung của chương này sẽ trình bày tổng quan về trí tuệ nhân tạo và giới thiệu về đề tài **"***Nghiên cứu áp dụng mô hình học tăng cường vào bài toán giải mê cung***".** Lý do là vì bài toán mê cung là một trường hợp ứng dụng tiêu biểu của học tăng cường, một nhánh trong AI. Do đó, việc có cái nhìn tổng quan về AI sẽ giúp làm rõ hơn bối cảnh, mục tiêu cũng như phương pháp giải quyết bài toán được trình bày trong đồ án.

1.1. Tổng quan về trí tuệ nhân tạo

1.1.1 Định nghĩa và mục tiêu của AI

Trí tuệ nhân tạo có nhiều cách định nghĩa khác nhau nhưng đều nhấn mạnh khả năng mô phỏng các năng lực nhận thức của con người. John McCarthy – một trong những người sáng lập lĩnh vực AI đã định nghĩa : "AI là khoa học và kỹ thuật chế tạo những máy tính thông minh" [1]. IBM mô tả AI là công nghệ giúp máy tính và máy móc mô phỏng khả năng học hỏi, hiểu biết, giải quyết vấn đề, ra quyết định, sáng tạo và tự chủ của con người [2]. Nói cách khác, AI là năng lực của một máy tính hoặc robot điều khiển bằng máy tính để thực hiện những nhiệm vụ thường gắn với quá trình trí tuệ của con người, như khả năng suy luận [3].

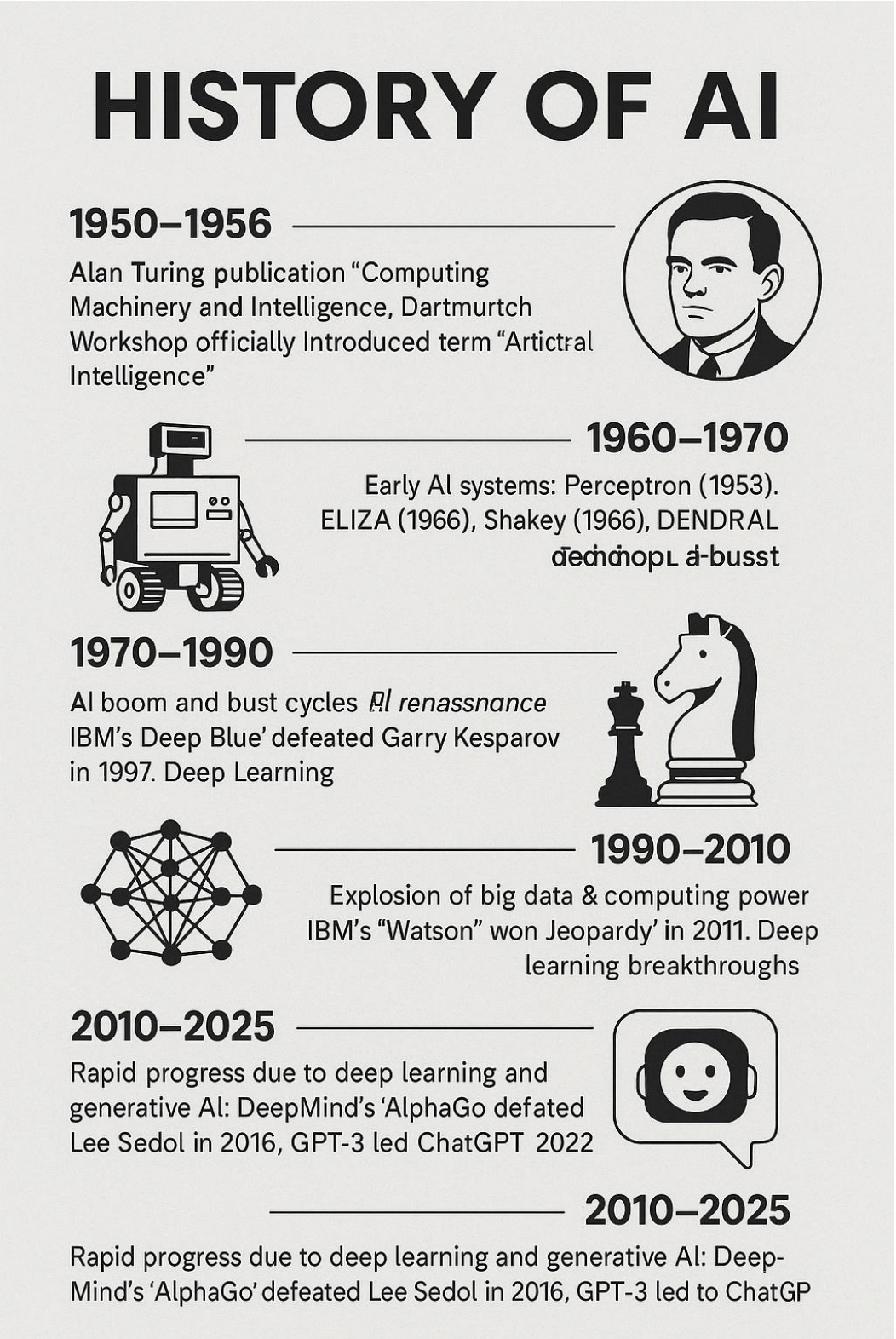
Mục tiêu chung của nghiên cứu AI là thiết kế được các hệ thống có khả năng: học hỏi từ dữ liệu, suy luận và ra quyết định, nhận thức được ngôn ngữ, và thực hiện các hành động tự động tương tự hoặc vượt qua khả năng của con người. Các hệ thống AI hướng tới việc tăng cường hoặc thay thế sức người trong việc giải các bài toán phức tạp, từ nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên đến điều khiển hệ thông. Chẳng hạn, một số mục tiêu cụ thể thường được nhắc đến bao gồm phân tích và lý giải dữ liệu, lập kế hoạch, và ra quyết định thông minh dựa trên thông tin đầu vào [1] [2].

1.1.2 Lịch sử phát triển của AI

Lĩnh vực AI bắt đầu vào giữa thế kỷ 20 với những ý tưởng và cộc mốc quan trọng như sau:

- 1950 – 1956: Năm 1950, Alan Turing công bố bài báo "Computing Machinery and Intelligence", giới thiệu bài kiểm tra Turing đánh giả khả năng trí tuệ của máy móc [4]. Năm 1956, John McCarthy cùng các cộng sự tại hội thảo Dartmouth đã chính thức đề xuất thuật ngữ "Artificial Intelligence", đánh dấu sự khởi đầu của lĩnh vực nghiên cứu AI [4].
- 1960 – 1970: Giai đoạn này chứng kiến nhiều hệ thống AI ban đầu. Năm 1958, Rosenblatt phát triển mạng nơ-ron nhân tạo Perceptron đầu tiên có khả năng học hỏi từ dữ liệu [4]. Đến giữa thập niên 60, các chương trình như ELIZA (1966) – một chatbot đầu tiên – robot di động Shakey (1966) kết hợp thị giác và điều hướng đã ra đời, thể hiện khả năng mô phỏng giao tiếp và hoạt động tự động của máy [4]. Đồng thời, các hệ thống chuyên gia như DENDRAL (1965) ra đời để hỗ trợ giải quyết vấn đề chuyên ngành (ví dụ: hóa hữu cơ) [4]
- 1970 – 1990: Thập niên 70 – 80 trải qua những giai đoạn *đông lạnh AI* do kỳ vọng bị thổi phồng không được đáp ứng. Tuy nhiên, cuối thập niên 80, một song "Phục hung AI" xuất hiện cùng với việc phát triển các máy xử lý chuyên dụng và mạng nơ-ron nhân tạo mới. Năm 1997, cột mốc quan trọng là máy tính Deep Blue của IBM đánh bại nhà vô địch cờ vua Garry Kasparov trong điều kiện thi đấu chính thức [4]. Tháng 6/1997, đó là lần đầu tiên một máy tính đánh bại kỳ thủ cờ vua chuyên nghiệp trong cuộc thi đẳng cấp.
- 1990 – 2010: Giai đoạn này chứng kiến sự bùng nổ của dữ liệu lớn và khả năng tính toán. Năm 2011, hệ thống Waston của IBM thắng cuộc thi đố trí tuệ nhân tạo *Jeopardy!* Với con người. Công nghệ

- thị giác máy tính và nhận dạng giọng nói được áp dụng rộng rãi. Đến năm 2012, sự kiện nổi bật là cuộc thi ImageNet mà đội AlexNet (Hinton và cộng sự) chiến thắng bằng cách áp dụng mạng nơ-ron sâu [4]. Kể từ đó, deep learning trở thành công nghệ chủ lực của AI.
- 2010 – 2025: AI phát triển với tốc độ nhanh chóng nhờ phát triển mạng nơ-ron học sâu và lượng dữ liệu khổng lồ. Ví dụ năm 2016, hệ thống AlphaGo của DeepMind đánh bại cờ thủ cờ vây hàng đầu thế giới Lee Sedol [4], đánh dấu một bước tiến vượt bậc trong khả năng chiến thắng các trò chơi chiến lược. Gần đây, sự bùng nổ của AI tạo sinh (Generative AI) được khởi xướng bởi các mô hình ngôn ngữ lớn. Năm 2020, OpenAI ra mắt GPT-3-mô hình ngôn ngữ 175 tỷ tham số có khả năng sinh văn bản tự nhiên. Cuối năm 2022, công cụ ChatGPT (dựa trên GPT-3.5) được công bố và nhanh chóng phổ biến [4]. Đến năm 2023, GPT-4 ra đời [4], cùng với các sản phẩm như Google Bard, cho thấy AI đang bước vào giai đoạn ứng dụng rộng rãi và thách thức đạo đức, chính sách.



Hình 1.1: Lịch sử phát triển trí tuệ nhân tạo từ giữa thế kỷ 20 đến nay

Các mốc lịch sử trên cho thấy AI trải qua nhiều giai đoạn từ khái niệm lý thuyết đến thực tế ứng dụng rộng rãi khắp trong đời sống hiện đại.

1.1.3 Các lĩnh vực và ứng dụng tiêu biểu của AI

AI bao gồm nhiều lĩnh vực chuyên sâu, trong đó có thể kể đến:

- Machine Learning (Học máy)**:** Là ngành cốt lõi của AI, sử dụng các thuật toán cho phép máy tính học từ dữ liệu và đưa ra dự đoán hoặc quyết định mà không cần lập trình cụ thể cho từng tác vụ . Học máy bao gồm nhiều kỹ thuật như học có giám sát, học không giám sát, và học tăng cường. Các thuật toán phổ biến như hồi quy, cây quyết định, SVM, KNN, và mạng nơ-ron nhân tạo đều thuộc Machine Learning [4].
- Deep Learning (Học sâu)**:** Là một phân nhánh của Machine Learning dựa trên mạng nơ-ron nhân tạo có nhiều tầng. Học sâu giúp mô phỏng cấu trúc phân tầng của não người để xử lý dữ liệu lớn và phức tạp [4]. Các mạng nơ-ron sâu gồm nhiều tầng ẩn, có khả năng tự trích xuất đặc trưng từ dữ liệu thô, giúp tự động hóa việc xử lý văn bản, hình ảnh, âm thanh. Học sâu đặc biệt hiệu quả trong các bài toán nhận dạng giọng nói, nhận diện khuôn mặt, dịch máy, và đã trở thành nền tảng cho hầu hết ứng dụng AI hiện đại [4].
- Xử lý ngôn ngữ tự nhiên (Natural Language Processing – NLP)**:** Nghiên cứu cách máy tính hiểu và sinh ngôn ngữ con người. NLP bao gồm các nhiệm vụ như phân tích cú pháp, nhận dạng ngôn ngữ tự nhiên, dịch máy, tóm tắt văn bản, phân tích tâm lý từ ngữ, trả lời câu hỏi, hội thoại. Ví dụ, khi bạn hỏi Siri hay Google Assistant, chính NLP giúp hệ thống "hiểu" câu hỏi và phản hồi bằng ngôn ngữ tự nhiên [5]. NLP đã ứng dụng rộng trong các trợ lý ảo, công cụ dịch tự động, tìm kiếm web, lọc thư rác, và chatbots.
- Robotics (Robot học)**:** Kết hợp AI với kỹ thuật cơ khí và điện tử để thiết kế và điều khiển robot có khả năng tự động thực hiện nhiệm vụ trong môi trường vật lý [6]. Robotics tập trung vào xây dựng các máy móc có "sự tự chủ" nhất định, tức là có thể cảm biến môi

trường, xử lý thông tin và hành động phù hợp. Ví dụ, robot công nghiệp tự động hàn, robot y tế phẫu thuật, hay robot giao hàng, đều ứng dụng AI để di chuyển và ra quyết định trong thời gian thực. Robotics và AI thường đi đôi với nhau: AI cung cấp "bộ não" còn robotics cung cấp "thân" (phần cứng và cơ cấu để thực hiện hành động).

- Các lĩnh vực khác**:** Ngoài ra, có các chuyên ngành như thị giác máy tính cho phép máy xử lý và hiểu hình ảnh, video [4], Machine Reasoning, Agent-based AI, Expert Systtôis dùng để ra quyết định trên cơ sở kiến thức chuyên sâu… Các lĩnh vực này đều đóng góp tạo nên toàn bộ hệ sinh thái của trí tuệ nhân tạo.

Bởi tính đã dạng ở nhiều lĩnh vực nên hiện nay, nên hiện nay AI hiện diện trong hầu hết khía cạnh của đời sống nhờ khả năng tự động hóa, phân tích dữ liệu lớn và đưa ra giải pháp thông minh. Một số ứng dụng tiêu biểu:

- Y tế**:** AI hỗ trợ phân tích dữ liệu y tế (hồ sơ bệnh án, hình ảnh y khoa) để phát hiện sớm bệnh và đề xuất phương pháp điều trị. Ví dụ, hệ thống IBM Watson Health hoặc các công cụ Deep6 AI sử dụng AI để phân tích hồ sơ bệnh nhân và hình ảnh, giúp phát hiện mẫu bệnh và nguy cơ sớm hơn [7]. Công nghệ thị giác máy tính (AI) cũng giúp đọc phim X-quang, MRI và phát hiện tổn thương.
- Giao thông**:** AI là "bộ não" của các phương tiện tự lái và hệ thống điều khiển giao thông thông minh. Xe tự lái của Tesla, Waymo dùng cảm biến, camera và LiDAR, kết hợp AI để nhận biết môi trường xung quanh và tự lái an toàn [8]. Hệ thống điều khiển tín hiệu giao thông thông minh tận dụng AI để điều tiết đèn giao thông theo lưu lượng thực tế, giảm ùn tắc và tai nạn [8]. AI cũng ứng dụng trong quản lý phương tiện, tối ưu lộ trình logistics, và dịch vụ chia sẻ xe.
- Giáo dục**:** AI góp phần cá nhân hóa quá trình học. Các nền tảng học trực tuyến như Coursera sử dụng thuật toán AI để đề xuất khoá học

phù hợp với từng học viên dựa trên sở thích và hiệu suất học tập [7]. AI có thể chấm điểm tự động, xây dựng lộ trình học tập cá, hoặc trợ giúp gia sư ảo, giúp giáo viên tiết kiệm thời gian và nâng cao chất lượng giảng dạy.

- Tài chính**:** Trong ngành ngân hàng và chứng khoán, AI được dùng để phòng chống gian lận, đánh giá tín dụng và quản lý đầu tư. Chẳng hạn, các chatbot AI hỗ trợ khách hàng tra cứu thông tin tài khoản hay thực hiện giao dịch cơ bản [7]. Các thuật toán máy học phân tích giao dịch để phát hiện hành vi gian lận và cảnh báo kịp thời. AI cũng được dùng để tối ưu danh mục đầu tư, dự báo thị trường, và quản lý rủi ro.
- Các lĩnh vực khác**:** AI còn ứng dụng trong nông, sản xuất, thương mại điện tử, dịch vụ (chatbots, nhận dạng khuôn mặt), và giải trí (phim, game có nhân vật AI thông minh). Tóm lại, AI đang thâm nhập sâu vào mọi ngành nghề, góp phần nâng cao năng suất và sáng tạo ra các sản phẩm mới.

Trí tuệ nhân tạo đã và đang góp phần mạnh mẽ vào tự động hóa các công việc thường nhật và hỗ trợ ra quyết định. Các hệ thống AI có thể tự động hóa các tác vụ lặp lại vốn tốn nhiều thời gian, như xử lý dữ liệu, đăng nhập hệ thống, tính toán báo cáo, qua đó giúp tăng hiệu quả và giảm sai sót do con người [2]. Ngoài ra, nhờ khả năng phân tích dữ liệu lớn và nhận diện mẫu , AI cung cấp những thông tin chi tiết có giá trị giúp cải thiện chất lượng ra quyết định. Ví dụ, trong kinh doanh, AI có thể dự báo xu hướng thị trường, tối ưu hóa quy trình sản xuất, và đề xuất chiến lược tăng trưởng. IBM nhận định: **"***AI đang mở ra một kỷ nguyên mới về hiệu quả bằng cách tự động hóa các tác vụ lặp lại, phân tích dữ liệu lớn để phát hiện xu hướng, tối ưu hóa quy trình phức tạp và cung cấp các thông tin hỗ trợ quyết định tốt hơn***"** [2]. Nhờ đó, tổ chức sử dụng AI có thể cải thiện dịch vụ khách hàng, tiết kiệm chi phí và tập trung nguồn lực vào các công việc chiến lược hơn.

Trong tương lai, tầm quan trọng của AI còn tăng lên khi mà các hệ thống tự động hóa và *intelligent agents* sẽ ngày càng thay thế con người ở các công việc có tính chất lặp lại. AI cũng giúp giảm thiên kiến trong ra quyết định bằng cách dựa trên dữ liệu khoa học. Tuy nhiên, cũng cần chú ý đến những thách thức như tính minh bạch của AI, vấn đề đạo đức và an ninh, nhưng không thể phủ nhận AI đang định hình lại cách thức tự động hóa và ra quyết định trong mọi lĩnh vực.

1.2. Giới thiệu về bài toán giải mê cung

1.2.1 Tổng quan bài toán mê cung

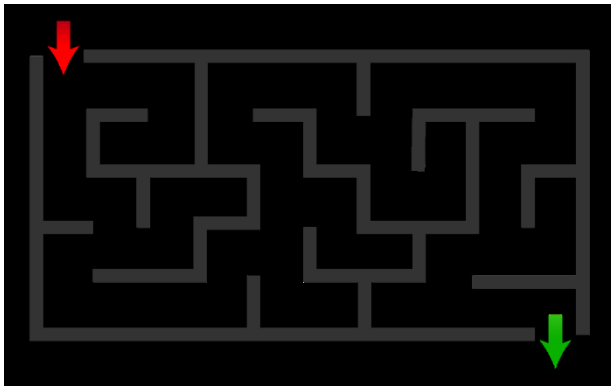
Trong giai đoạn hiện nay, trí tuệ nhân tạo không chỉ còn là một khái niệm học thuật mà đã dần trở thành một công nghệ lõi trong các sản phẩm, dịch vụ và hệ thống thông minh trên toàn cầu. AI đã chứng minh tiềm năng to lớn trong nhiều lĩnh vực như y tế, tài chính, giáo dục, giao thông, cũng như trong lĩnh vực mô phỏng và trò chơi điện tử. Cùng với sự phát triển đó, nhu cầu xây dựng các hệ thống có khả năng tự ra quyết định, thích nghi với môi trường, và học hỏi thông qua tương tác đang trở nên cấp thiết hơn bao giờ hết.

Một trong những hướng tiếp cận nổi bật và được nghiên cứu rộng rãi trong những năm gần đây chính là học tăng cường (ReinforcementLearning - RL). Khác với các phương pháp học máy truyền thống, học tăng cường không yêu cầu tập dữ liệu huấn luyện có nhãn rõ ràng. Thay vào đó, một tác nhân sẽ học thông qua quá trình tương tác liên tục với môi trường – thực hiện hành động, nhận phản hồi, và từ đó điều chỉnh hành vi để tối đa hóa tổng phần thưởng nhận được trong dài hạn. Tư duy này phản ánh khá sát với cách con người và động vật học hỏi từ kinh nghiệm thực tế, cho phép các hệ thống RL phát triển chiến lược hành động không chỉ cho một bước đi cụ thể, mà còn cho cả chuỗi hành động trong tương lai.

Trong bối cảnh đó*, bài toán mê cung* là một trong những bài toán kinh điển thường xuyên được sử dụng để thử nghiệm và đánh giá khả năng của các thuật toán học tăng cường. Bài toán giải *mê cung* được hiểu là việc tìm đường

đi từ một ô xuất phát đến một ô đích trong một mê cung hai chiều, thường biểu diễn dưới dạng lưới các ô vuông. Mỗi ô trong mê cung có thể là *ô tự do* (có thể di chuyển qua) hoặc *ô tường* (không thể đi vào).

Ví dụ, Hình 1.2 dưới đây minh họa một mê cung đơn giản: mũi tên đỏ đánh dấu vị trí bắt đầu của tác nhân, mũi tên xanh đánh dấu vị trí đích. Các ô xám đậm là ô tường (không thể đi vào), còn phần nền màu đen hoặc trắng là ô tự do mà tác nhân có thể di chuyển.



Hình 1.2: Ví dụ về một mê cung mẫu.

Chú thích: Mũi tên đỏ chỉ ô xuất phát, mũi tên xanh chỉ ô đích. Ô màu tối là chướng ngại (không thể đi), còn phần nền là ô tự do. Trong mê cung này, tác nhân di chuyển trên lưới ô vuông để đến đích.

Bài toán mê cung là ví dụ kinh điển về bài toán lập kế hoạch đường đi. Theo khái niệm của học tăng cường, một tác nhân di chuyển từng bước trong mê cung để đạt được mục tiêu tối đa phần thưởng tích lũy. Một ví dụ minh họa phổ biến từ ngành lý thuyết RL là chú chuột trong mê cung tìm đường đến ô chứa miếng phô mai (đích) [9]. Mỗi bước di chuyển tác nhân có thể nhận một phần thưởng âm nhỏ để khuyến khích tìm đường ngắn nhất. Khi đến đích, tác nhân nhận phần thưởng dương lớn. Mục tiêu của tác nhân là tìm dãy di chuyển sao cho tổng phần thưởng tích lũy là lớn nhất [9]. Việc thiết kế phần thưởng như trên gọi là khen thưởng trì hoãn vì tác nhân phải chịu nhiều hình phạt trước khi nhận được phần thưởng lớn khi đạt đích.

1.2.2 Phân tích bài toán

Bài toán giải mê cung có thể được mô hình hóa như một bài toán học tăng cường theo quy ước MDP (Markov Decision Process) [10] [11]. Trong đó, môi trường là không gian mê cung gồm các ô đi được và chướng ngại, và tác nhân là đối tượng có nhiệm vụ di chuyển qua mê cung. Ở mỗi bước thời gian, tác nhân quan sát trạng thái hiện tại của môi trường và chọn một hành động để thực hiện [10].

Các thành phần chính của mô hình bao gồm:

- Tác nhân: Là đối tượng học tập hoặc ra quyết định có khả năng thực hiện các hành động trong mê cung để đạt mục tiêu. Tác nhân học thông qua cơ chế thử sai, nhận phần thưởng hoặc bị phạt từ môi trường và điều chỉnh hành vi sao cho tổng phần thưởng tích luỹ được tối đa [10] [11].
- Trạng thái: Mỗi trạng thái ứng với vị trí hiện tại của tác nhân trong mê cung (ví dụ tọa độ hàng và cột trên lưới) và có thể bao gồm cả thông tin môi trường xung quanh nếu cần [11]. Tập không gian trạng thái bao gồm tất cả các vị trí hợp lệ trong mê cung.
- Hành động: Tập hợp các hành động khả thi mà tác nhân có thể thực hiện ở mỗi trạng thái, ở đây là di chuyển lên, xuống, trái, phải [11]. Các hành động này bị giới hạn bởi ranh giới và chướng ngại của mê cung.
- Phần thưởng: Hàm phần thưởng quy định giá trị cấp cho tác nhân sau mỗi hành động hoặc khi đạt một trạng thái nhất định. Ví dụ, có thể gán phần thưởng dương khi tác nhân chạm tới ô đích và phần thưởng âm khi va chạm tường; đồng thời có thể phạt nhỏ cho mỗi bước di chuyển để khuyến khích tìm đường ngắn nhất [11].
- Mục tiêu: Mục tiêu của tác nhân là tìm một chính sách hành động tối ưu sao cho nó di chuyển từ trạng thái bắt đầu đến trạng thái đích với tổng phần thưởng tích luỹ lớn nhất, tức là tìm đường đi ngắn nhất và hiệu quả nhất [11].

1.2.3 Mô tả dữ liệu đầu vào, đầu ra

Trong mô hình học tăng cường giải mê cung, đầu vào của hệ thống bao gồm các thành phần sau:

- Ma trận mê cung: Ma trận hai chiều biểu diễn kết cấu của mê cung, trong đó mỗi phần tử xác định một ô là đường đi hay tường. Ma trận này xác định không gian trạng thái của môi trường [11].
- Vị trí bắt đầu và kết thúc: Tọa độ của ô bắt đầu nơi tác nhân xuất phát và ô đích cần đạt. Xác định rõ điểm bắt đầu và điểm kết thúc là cần thiết để huấn luyện tác nhân tìm đường đi từ đầu đến đích [11].
- Không gian hành động và chuyển trạng thái: Tập hợp các hành động hợp lệ và quy tắc xác định trạng thái mới khi tác nhân thực hiện hành động. Các chuyển trạng thái tuân theo ranh giới của mê cung, không cho phép di chuyển xuyên tường [11].
- Hàm phần thưởng: Định nghĩa giá trị phần thưởng hoặc hình phạt khi tác nhân thực hiện hành động hoặc đạt trạng thái nhất định. Thông thường, gán phần thưởng dương khi tới ô đích, phần thưởng âm khi va chạm tường hoặc bước đi không hiệu quả, và có thể phạt nhỏ cho mỗi bước đi để khuyến khích tìm đường ngắn nhất [11].

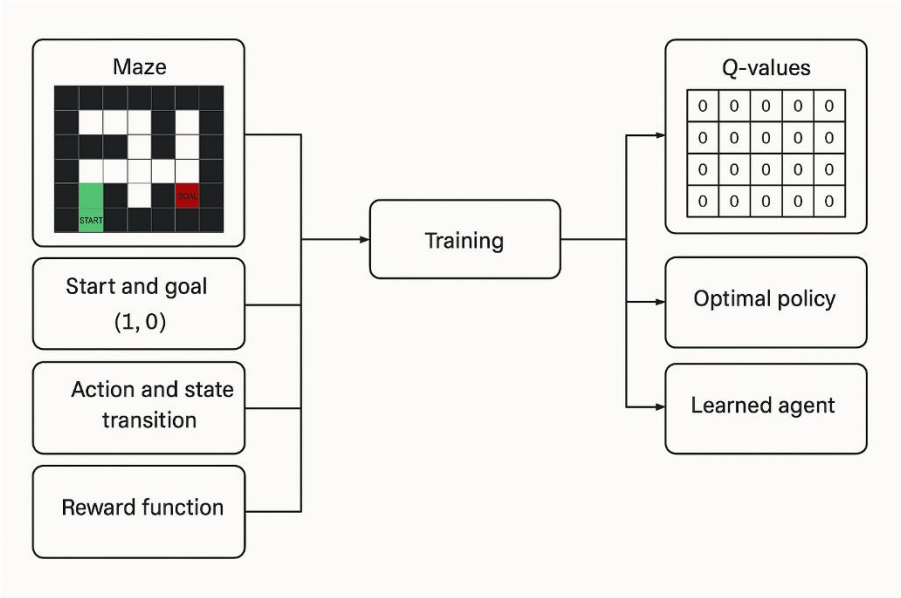
Quá trình huấn luyện RL (ví dụ Q-learning) sẽ sử dụng các thông tin đầu vào trên để cập nhật bảng giá trị Q hoặc chính sách. Ban đầu, bảng Q được khởi tạo với giá trị 0 cho mọi cặp trạng thái–hành động [11]. Tác nhân sẽ trải qua nhiều tập bắt đầu từ ô xuất phát và di chuyển cho đến khi về đích hoặc kết thúc. Ở mỗi bước, bảng Q được cập nhật dựa trên phương trình Bellman và chính sách ε-greedy được sử dụng để cân bằng giữa khám phá và khai thác [11].

Sau quá trình huấn luyện, đầu ra của hệ thống bao gồm:

- Bảng giá trị Q: Ma trận $\mathrm{Q}(s, a)$ đã hội tụ, trong đó mỗi mục ghi nhận giá trị kỳ vọng tối đa thu được khi thực hiện hành động a từ trạng thái s. Bảng Q thể hiện kiến thức đã học được và cho phép so sánh các hành động tại mỗi trạng thái [11].

- Chính sách tối ưu: Luật hoặc bảng quyết định ánh xạ mỗi trạng thái sang hành động tốt nhất (có giá trị Q cao nhất). Chính sách này cho tác nhân biết cách di chuyển hiệu quả nhất để đạt đích [11].
- Tác nhân đã học: Một đối tượng phần mềm mang bảng Q hoặc chính sách tối ưu, có thể đưa ra quyết định dựa trên chiến lược đã học khi đối mặt với mê cung. Tác nhân này đại diện cho giải pháp của bài toán và có khả năng tái áp dụng cho các mê cung tương tự.

Mối quan hệ giữa đầu vào, quá trình huấn luyện và đầu ra được thể hiện như sau: từ các thông tin đầu vào (cấu trúc mê cung, vị trí bắt đầu/đích, luật chuyển trạng thái, hàm phần thưởng), thuật toán Q-learning sẽ lặp đi lặp lại nhiều tập huấn luyện để cập nhật bảng Q cho đến khi hội tụ. Kết quả cuối cùng là bảng giá trị Q và chính sách tối ưu làm đầu ra, cho phép tác nhân tìm được đường đi ngắn nhất đến đích trong mê cung [11].



Hình 1.3: Mô hình hóa dữ liệu của bài toán giải mê cung

1.2.4 Ứng dụng của bài toán giải mê cung trong thực tế

Bài toán giải mê cung là mô hình trừu tượng cho rất nhiều ứng dụng thực tế, đặc biệt trong các lĩnh vực robot di động, trò chơi điện tử và hệ thống tự động hóa:

- Robot di động và điều hướng: Mê cung mô phỏng môi trường có chướng ngại vật phức tạp, tương tự như bản đồ hoặc sơ đồ nhà xưởng. Các nghiên cứu đã sử dụng học tăng cường để huấn luyện robot định hướng và tránh vật cản trong mê cung 2D, sau đó áp dụng cho robot thực ngoài đời. Ví dụ, Luisa Chiu (2024) xây dựng mô hình DQN để điều hướng robot di động trong mê cung động [12]. Gleason và Jenkin (2022) cũng chỉ ra rằng học tăng cường có thể cung cấp cơ chế điều khiển hiệu quả cho xe tự hành không holonomic trong môi trường dạng mê cung [13]. Qua đó, giải mê cung giúp kiểm thử khả năng lập kế hoạch đường đi và thích nghi với chướng ngại trong thuật toán điều hướng tự động.
- Trí tuệ nhân tạo trong trò chơi: Mê cung vốn là cấu trúc quen thuộc trong nhiều game (ví dụ trò Pac-Man, Zelda, các trò giải đố labyrint), nơi nhân vật phải tìm đường đến đích tránh kẻ địch. Học tăng cường được áp dụng để huấn luyện AI điều khiển nhân vật chơi game qua mê cung. Thực tế, các thành tựu RL nổi tiếng như chơi cờ Go, Dota hay game Atari cũng có nhiều tựa game xoay quanh tối ưu đường đi và chiến lược [14]. Sử dụng mê cung làm môi trường huấn luyện giúp đánh giá và phát triển các thuật toán RL trong bối cảnh tương tự trò chơi.
- Hệ thống tự động hóa và logistics: Trong các hệ thống tự động (ví dụ robot kho tự hành, xe AGV), bài toán tìm đường đi ngắn nhất qua mạng lưới lối đi cũng rất phổ biến. Mê cung cũng được dùng để mô phỏng sơ đồ nhà máy hoặc đường đi robot thu gom, cho phép kiểm chứng chiến lược điều khiển và lập lịch tự động. Học tăng cường

huấn luyện trên mô hình mê cung giúp đánh giá hiệu quả điều hướng trong môi trường ràng buộc mà không cần lập trình cứng kịch bản di chuyển

CHƯƠNG 2: CÁC PHƯƠNG PHÁP TIẾP CẬN BÀI TOÁN

2.1. Các phương pháp tìm kiếm cơ bản

2.1.1 Thuật toán A*

A* là một thuật toán tìm đường thông minh kết hợp giữa tìm kiếm dijkstra và hàm đánh giá heuristic, được sử dụng rộng rãi trong trí tuệ nhân tạo và các ứng dụng như định tuyến, trò chơi…. [15] .Thuật toán này hoạt động trên đồ thị có trọng số và được thiết kế để tìm đường đi ngắn nhất từ nút bắt đầu đến nút đích bằng cách đánh giá mỗi đỉnh theo hàm f(n) = g(n) + h(n), trong đó g(n) là chi phí đường đi từ đầu tới n, còn h(n) là ước lượng chi phí từ n đến đích [16]. Nếu hàm heuristic hợp lệ, A* đảm bảo tìm đường ngắn nhất [15]. So với thuật toán Dijkstra, A* có "thông minh" hơn vì dùng hướng dẫn (heuristic) để tập trung tìm kiếm vào gần đích, nhờ đó có thể hiệu quả hơn về thời gian.

Mô tả thuật toán

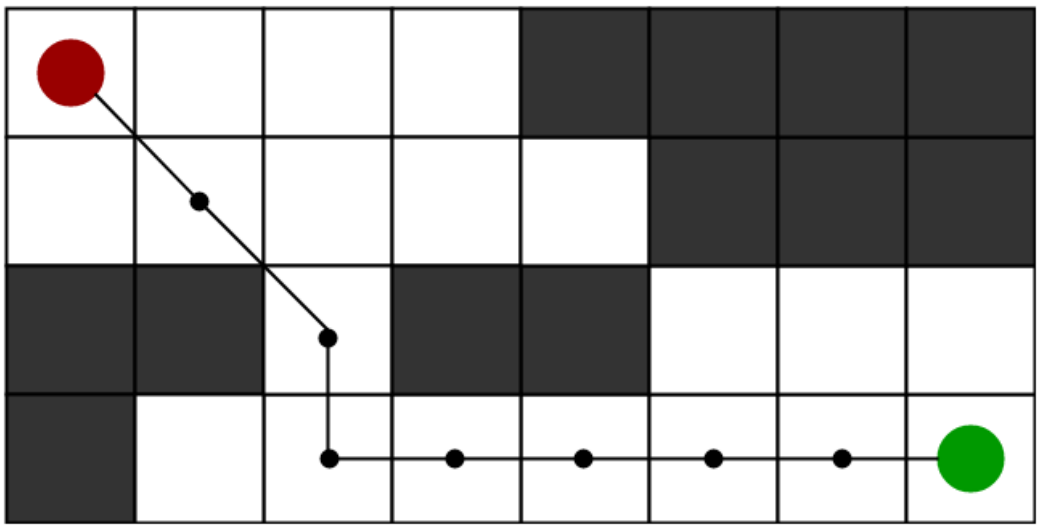
A* sử dụng hai tập hợp chính: Open List (đỉnh chờ khám phá) và Closed List (đỉnh đã khám phá). Thuật toán được mô tả đơn giản qua các bước chính:

1. Khởi tạo**:** Đặt trạng thái đầu vào Open List với g = 0 và f = g + h (h là giá trị heuristic từ đầu đến đích) [16]. Closed List rỗng.
2. Lặp**:** Khi Open List không rỗng:
    - Chọn đỉnh q có giá trị f (q) nhỏ nhất trong Open List [16]. Chuyển q ra Closed List (đánh dấu đã mở rộng).
    - Sinh các đỉnh kề (successors) của q, tính giá trị g, h và f cho mỗi đỉnh kế tiếp [16]:
        - g (successor) = g(q) + chi phí di chuyển từ q đến successor.
        - h(successor) = ước lượng chi phí từ successor đến đích (ví dụ dùng khoảng cách Manhattan hoặc Euclid tuỳ bài toán) [16].
        - f(successor) = g(successor) + h(successor) [16].

- Với mỗi successor, nếu nó đã có trong Closed List với f nhỏ hơn, bỏ qua; nếu nó chưa có trong Open List hoặc có f cao hơn, thêm nó vào Open List (hoặc cập nhật giá trị f mới thấp hơn) [16].

3. Kết thúc**:** Nếu chọn đỉnh q là đích, dừng và thu hồi đường đi bằng cách theo ngược cha. Ngược lại, lặp lại bước 2. Quá trình này đảm bảo A* luôn mở rộng các đường dẫn có giá trị f thấp nhất trước, nhờ đó tìm ra đường ngắn nhất một cách tối ưu [15] [16].

Các bước trên rất giống với Dijkstra nhưng có bổ sung bước tính heuristic. Việc dùng f=g+h giúp A* "dự đoán" đường ngắn nhất còn lại, do đó thường mở rộng ít đỉnh hơn so với tìm toàn bộ cây đường đi ngắn nhất. Tuy nhiên, A* yêu cầu bộ nhớ lớn (lưu tất cả đỉnh đã mở rộng) và phụ thuộc vào chất lượng heuristic để đạt hiệu năng tốt nhất.



Hình 2.1: Ví dụ đường đi tìm được bằng thuật toán A trên lưới

Chú thích: ô (ô xám là cản, ô trắng là ô có thể đi). Đỉnh bắt đầu (màu đỏ) và đích (màu xanh) được nối bằng đường tối ưu (đi qua các nút màu đen đậm). Xét một lưới 2D có các ô cản (màu xám) và ô trống (màu trắng). Núi gốc (đỏ) ở góc trên trái, núi đích (xanh) ở góc dưới phải. A* sẽ tính f=g+h tại mỗi bước và luôn chọn ô có f nhỏ nhất để mở rộng. Kết quả thu được là đường di chuyển ngắn nhất (theo ví dụ dùng chi phí 1 cho mỗi bước ngang/dọc) như vạch đen*

trong hình [16]. *Quá trình minh họa: bắt đầu tại ô đỏ, A* lan rộng dần sang các ô kề, ưu tiên các ô gần mục tiêu (theo heuristic) và không đi vào ô cản, rồi nối cuối cùng đến ô mục tiêu xanh. Đường tìm được là đường ngắn nhất trên lưới này (chi phí tối ưu), điều mà thuật toán đảm bảo nhờ f = g + h hướng tìm kiếm đúng cách* [15] [16].

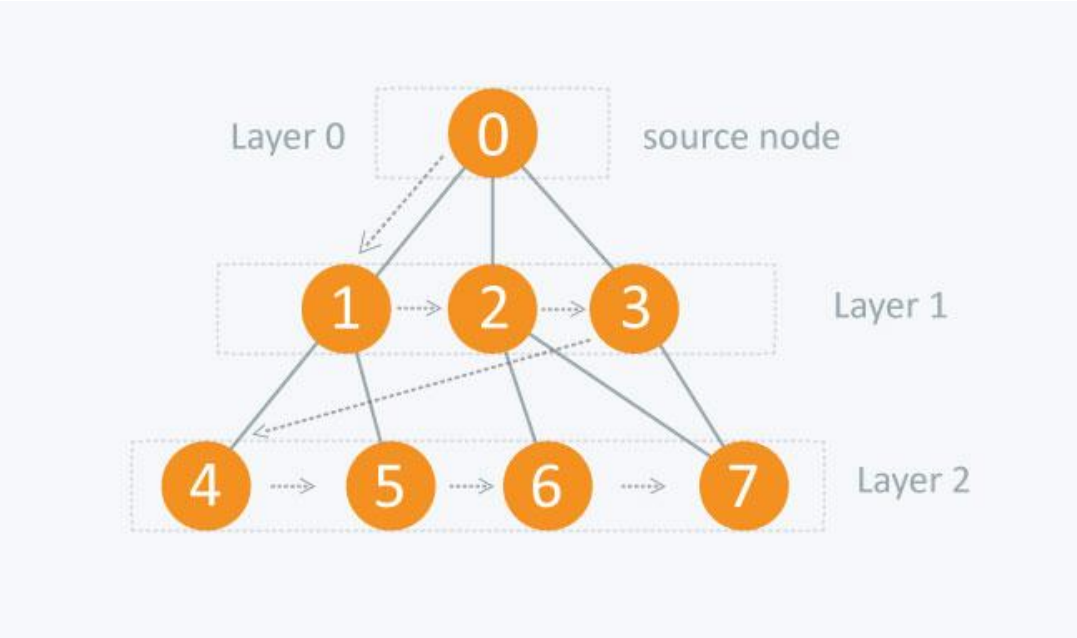
2.1.2 Tìm kiếm theo chiều rộng

Breadth-First Search hay tìm kiếm theo chiều rộng là thuật toán duyệt đồ thị cổ điển, đi theo chiều rộng, tức là khám phá lần lượt các đỉnh tại từng mức độ sâu trước khi sang mức kế tiếp [17]. BFS bắt đầu từ một nút nguồn rồi lần lượt duyệt tất cả các đỉnh kề của nó (mức 1), sau đó duyệt các đỉnh kề của những đỉnh vừa duyệt (mức 2),… cho đến khi tìm được mục tiêu hoặc hết các đỉnh cần duyệt [18]. Thuật toán này sử dụng một hàng đợi (queue) để lưu các nút cần thăm tiếp theo và một mảng visited để đánh dấu các nút đã duyệt tránh lặp lại [19]. BFS có các đặc điểm chính sau: nó đảm bảo tìm ra nút đích nếu tồn tại [17], và trong đồ thị vô hướng hoặc có trọng số bằng 1 thì tìm đường đi ngắn nhất về số cạnh. Tuy nhiên, BFS có độ phức tạp thời gian O(|V| + |E|) và yêu cầu bộ nhớ O(|V|) (cần lưu queue với nhiều đỉnh cùng lúc) [17].

Mô tả thuật toán

Thuật toán BFS được diễn đạt đơn giản bằng các bước sau (ví dụ trên đồ thị G với đỉnh xuất phát s):

- Bước 1 (Khởi tạo)**:** Tạo một hàng đợi rỗng Q và cho đỉnh s vào Q; đánh dấu s đã thăm [19].
- Bước 2 (Lặp)**:** Trong khi Q không rỗng: lấy đỉnh v ở đầu Q (dequeue). Với mỗi đỉnh kề w của v mà chưa thăm, đánh dấu thăm và đưa w vào Q [19].
- Bước 3 (Kết thúc): Lặp lại bước 2 cho đến khi Q rỗng hoặc đã tìm được đích. Khi kết thúc, BFS đã duyệt theo mức, đảm bảo lần lượt khám phá tất cả các đỉnh ở độ sâu 0, 1, 2, … từ đỉnh gốc.



Hình 2.2: Ví dụ đồ thị và thứ tự duyệt BFS (nguồn: Hackerearth).

Chú thích: Xét đồ thị đơn giản có nút nguồn là 0 (màu cam) và các nút kề tổ chức thành nhiều tầng. BFS sẽ khám phá lần lượt các nút theo từng lớp. Cụ thể, bắt đầu từ nút 0 (Layer 0), ta duyệt tất cả nút kề của nó là 1, 2, 3 (Layer 1) rồi mới đến các đỉnh của chúng (Layer 2) [17] [19]. Thứ tự duyệt có thể là 0 → 1 → 2 → 3 → 4 → 5 → 6 → 7. Quá trình duyệt này đảm bảo BFS tìm đường ngắn nhất từ nút 0 đến bất kỳ nút nào trong đồ thị. Ví dụ, để đi từ 0 đến 7, BFS qua 0→2→7 chỉ qua 2 cạnh, và 2 là số cạnh tối thiểu trong đồ thị này

2.1.3 Thuật toán leo đồi

Thuật toán leo đồi hay *Hill Climbing* là một thuật toán tìm kiếm địa phương dùng cho bài toán tối ưu hóa, lấy cảm hứng từ ý tưởng "leo lên đỉnh đồi" [20] [21]. Thuật toán này bắt đầu từ một nghiệm ban đầu tùy ý, sau đó lặp đi lặp lại: tại mỗi bước, nó chọn một nghiệm láng giềng sao cho giá trị hàm mục tiêu được cải thiện hơn nghiệm hiện tại. Nếu nghiệm láng giềng tốt hơn, thuật toán di chuyển đến đó và tiếp tục; nếu không còn nghiệm tốt hơn trong vùng lân cận, thuật toán kết thúc ở giá trị hiện tại [20]. Hill Climbing mang tính tham lam và đơn giản, hiệu quả cao trong thời gian ngắn [20]. Tuy nhiên, do chỉ tìm kiếm "cục bộ" từng bước, nó có thể kẹt ở cực trị cục bộ thay vì cực trị toàn cục nếu bài toán không đơn điệu [20]. Các biến thể phổ biến của hill

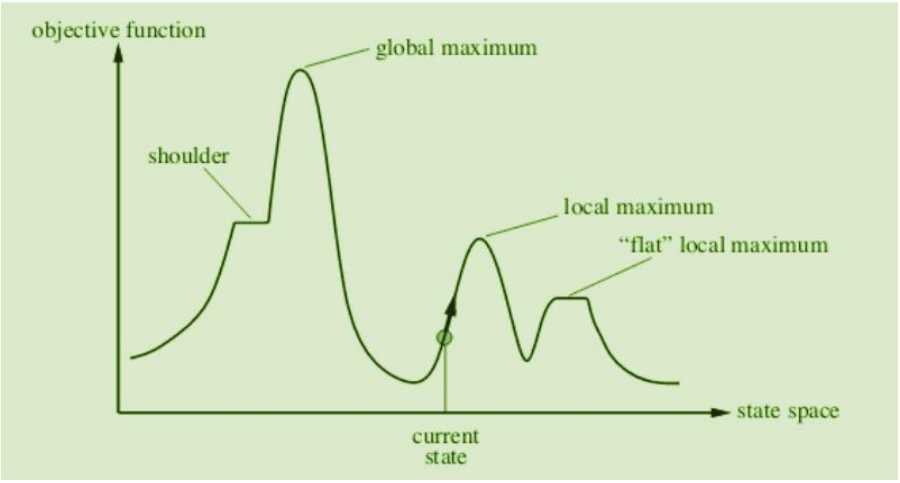
climbing bao gồm *simple* (chọn láng giềng đầu tiên tốt lên), *steepest-ascent* (chọn láng giềng tốt nhất trong tất cả láng giềng), và *stochastic* (chọn ngẫu nhiên) [21].

Mô tả thuật toán

Thuật toán leo đồi đơn giản có thể trình bày theo các bước sau:

- Bước 1**:** Đánh giá giá trị mục tiêu tại trạng thái ban đầu **s**; nếu s đã là nghiệm tối ưu hoặc đích thì dừng. Đặt s làm trạng thái hiện tại.
- Bước 2**:** Tại trạng thái hiện tại x, sinh các trạng thái láng giềng (những nghiệm có thể đạt được bằng một thay đổi nhỏ so với x). Tính giá trị mục tiêu cho từng láng giềng.
- Bước 3**:** Nếu tồn tại láng giềng có giá trị mục tiêu lớn hơn giá trị hiện tại, chọn láng giềng có cải thiện tốt và đặt nó làm trạng thái hiện tại mới. Quay về bước 2. Ngược lại, nếu không còn cải thiện nào thì dừng.
- Bước 4**:** Kết thúc khi không thể tìm được trạng thái láng giềng tốt hơn .

Các bước trên đảm bảo hill climbing liên tục "leo lên" các giá trị mục tiêu cao hơn cho đến khi không thể cải thiện nữa [20] [21]. Tuy nhiên, cần lưu ý rằng hill climbing không đảm bảo tìm cực trị toàn cục nếu không gian trạng thái có nhiều "đồi" [20].



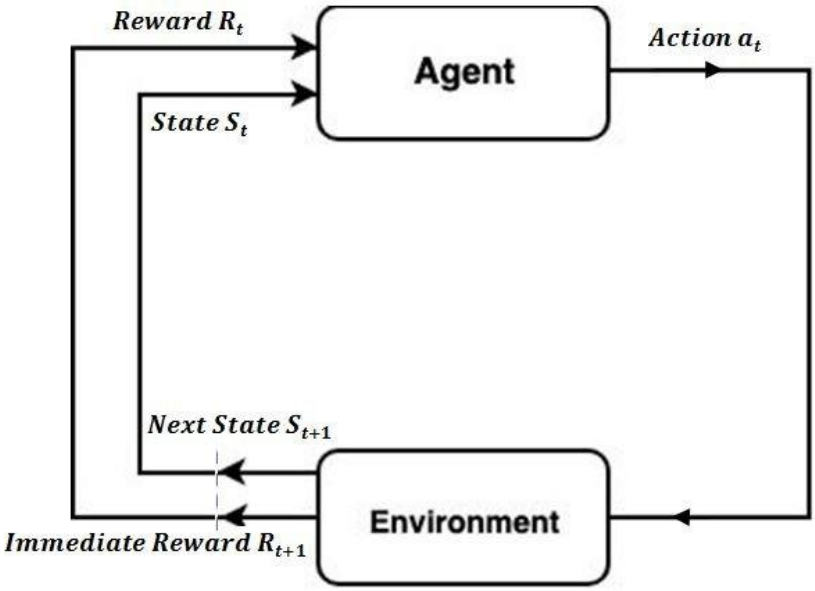
Hình 2.3: Minh họa thuật toán leo đồi

Chú thích: Đồ thị minh họa hàm mục tiêu (trục tung) theo không gian trạng thái (trục hoành); có cực trị toàn cục và cục bộ. Giả sử bài toán tối ưu hóa có hàm mục tiêu như đồ thị trên, với các cực trị (đỉnh) ở các trạng thái khác nhau. Giả sử thuật toán bắt đầu tại trạng thái "current" (điểm màu đen). Ở mỗi bước, hill climbing sẽ so sánh các trạng thái lân cận và di chuyển tới trạng thái có giá trị hàm mục tiêu cao hơn. Ví dụ, từ vị trí "current state" thuật toán tìm trạng thái tốt hơn (hướng lên đỉnh nhỏ màu xanh nhạt). Sau vài bước, nó đạt đỉnh cục bộ ("local maximum") và dừng lại do không có láng giềng nào cao hơn [20]. *Lưu ý rằng đỉnh hiện tại (round dot) không phải đỉnh cao nhất của toàn hàm (global maximum); nếu bắt đầu ở vùng khác hoặc dùng kỹ thuật khởi tạo ngẫu nhiên nhiều lần (random restarts), có thể tìm tới cực đỉnh lớn hơn. Đồ thị này minh hoạ trực quan ưu nhược điểm của hill climbing: nó dễ dàng leo lên các vùng cao nhưng có thể bị kẹt ở cực trị cục bộ thay vì cực trị toàn cục* [20] [21].

2.2. Giới thiệu học tăng cường

2.2.1 Khái niệm và vai trò

Học tăng cường (Reinforctôient Learning – RL) là một nhánh của học máy trong trí tuệ nhân tạo, nơi một tác nhân học cách ra quyết định trong một môi trường động thông qua các trải nghiệm và phản hồi dạng phần thưởng. Mục tiêu của tác nhân là tìm một chính sách tối ưu để tối đa hóa phần thưởng tích lũy lâu dài. Khác với học có giám sát đòi hỏi dữ liệu đã được gán nhãn đầy đủ, và học không giám sát chỉ khám phá cấu trúc dữ liệu mà không có mục tiêu rõ ràng, học tăng cường dựa trên tương tác trực tiếp với môi trường và không cần dữ liệu đầu ra được gán sẵn [22] [23]. Ở đây, tác nhân tự học hỏi qua các lần thử và sai, nhận được phần thưởng hay hình phạt từ môi trường tùy vào hành động đã thực hiện, và dần hình thành chiến lược tối ưu.



Hình 2.4: Quá trình huấn luyện agent

Quan trọng, trong RL không có người hướng dẫn trực tiếp mà chỉ có tín hiệu phản hồi tổng hợp từ môi trường. Sự khác biệt giữa ba phương pháp chính của học máy được tóm tắt trong bảng phân biệt: trong khi học có giám sát dựa vào dữ liệu đầu vào-đầu ra đã biết và học không giám sát khai thác cấu trúc trong dữ liệu không có nhãn, thì học tăng cường học thông qua tương tác liên tục với môi trường, khai thác phần thưởng để điều chỉnh chính sách hành động [22] [23]. Ví dụ, trong chơi cờ vây, thuật toán học củng cố (như AlphaGo) kết hợp học có giám sát và tự chơi để cải thiện nước đi, phần tự chơi thuộc về học tăng cường khi tác nhân đánh giá nước đi dựa trên điểm số đạt được mà không cần dữ liệu nhãn sẵn [24] [22].

Bảng 1.1: *So sánh Supervised, Unsupervised và RL*

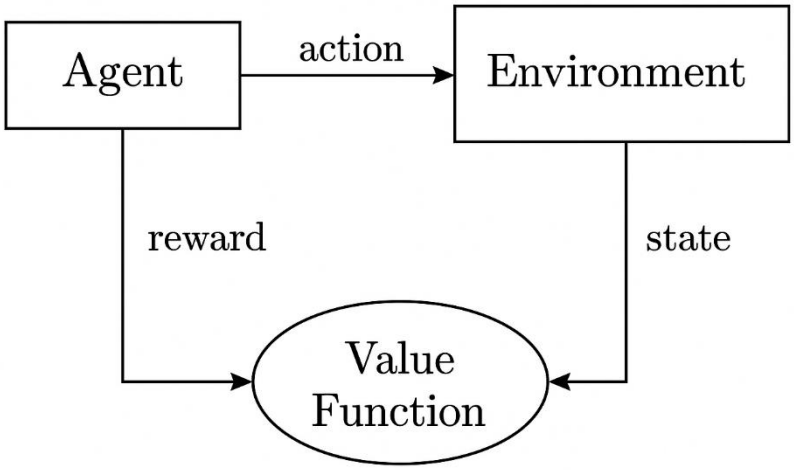
Tiêu chí	**Học có giám sát**	**Học không giám sát**	**Học tăng cường (RL)**
Dữ liệu	Có nhãn	Không nhãn	Không nhãn, có phản hồi từ môi trường
Phản hồi	Ngay lập tức, rõ ràng	Không có	Trễ, tích lũy theo hành động
Người hướng dẫn	Có dữ liệu gán nhãn	Không có	Không có trực tiếp
Mục tiêu	Dự đoán đầu ra	Tìm mẫu, nhóm	Tối ưu hành vi qua tương tác
Ví dụ	Phân loại ảnh	Phân cụm khách hàng	Chơi game, robot tự hành

Học tăng cường đặc biệt phù hợp cho các bài toán quyết định liên tục trong môi trường động như điều khiển robot, lái xe tự động, hoặc chơi game, nơi kết quả của hành động phụ thuộc vào chuỗi hành động trước đó và các phần thưởng nhận được liên tục. Khái niệm cộng dồn phần thưởng là trung tâm: tác nhân không chỉ quan tâm phần thưởng tức thì mà còn đánh giá lâu dài, với mục tiêu tối đa hóa tổng phần thưởng sau nhiều bước thời gian [22] [23]. Trong bối cảnh này, học tăng cường học được vai trò quan trọng vì nó định nghĩa một khung lý thuyết toàn diện cho học tập từ kết quả và cân bằng giữa khám phávà khai thác [23].

2.2.2 Thành phần chính trong học tăng cường

Hệ thống học tăng cường mô hình hóa gồm các thành phần chính như tác nhân môi trường, trạng thái, hành động, phần thưởng, chính sách và hàm giá trị Mỗi bước thời gian t, tác nhân quan sát trạng thái S_t của môi trường và chọn

một hành động A_t dựa trên chính sách hiện tại [14] [23]. Môi trường sau đó chuyển sang trạng thái mới S_{t+1} và đưa ra phần thưởng R_{t+1} phản ánh chất lượng của hành động vừa thực hiện. Vòng tương tác này lặp đi lặp lại, qua đó tác nhân học điều chỉnh chính sách để tăng tổng phần thưởng tích lũy. Cụ thể, chính sách π là một ánh xạ từ trạng thái sang hành động có thể xác định hoặc xác suất: $\pi(s, a) = \Pr(A_t = a \mid S_t = s)$ [23] [25]. Mục tiêu là tìm một chính sách tối ưu π^* sao cho giá trị kỳ vọng của phần thưởng tích lũy đạt cực đại.



Hình 2.5: Khung tương tác điển hình trong học tăng cường:

Chú thích: tác nhân quan sát môi trường (state), chọn hành động (action) và nhận phần thưởng cùng trạng thái mới (reward, next state) [22] [23]

Các thành phần chính bao gồm :

- Tác nhân (Agent): Thực thể quyết định hành động.
- Môi trường (Environment): Hệ thống bên ngoài mà tác nhân tương tác.
- Trạng thái (State)**:** Mô tả quan sát hiện tại của môi trường tác nhân nhìn thấy. Trong một Hệ thống Quyết định Markov, trạng thái bao hàm tất cả thông tin cần thiết cho quyết định trong tương lai.
- Hành động (Action)**:** Lựa chọn của tác nhân trong mỗi trạng thái, được sinh ra từ chính sách π.

- Phần thưởng (Reward): Tín hiệu số cung cấp phản hồi về kết quả của hành động vừa thực hiện; thường là số thực (có thể dương hoặc âm). Tác nhân có mục tiêu tối đa hóa phần thưởng dài hạn.
- Chính sách (Policy)**:** Hàm ánh xạ từ trạng thái đến phân phối xác suất trên hành động. Chính sách xác định chiến lược hành động của tác nhân.
- Hàm giá trị (Value function)**:** Đánh giá "độ tốt" của một trạng thái dưới một chính sách nhất định. Ví dụ, hàm giá trị trạng thái $V^{\pi}(s)$ cho biết giá trị kỳ vọng của tổng phần thưởng tích lũy bắt đầu từ trạng thái s và theo chính sách π [23]. Tương tự, hàm giá trị hành động hàm $Q^{\pi}(s,a)$ đánh giá giá trị của việc thực hiện hành động a ở trạng thái s rồi theo π tiếp theo.

Mối quan hệ giữa các thành phần cơ bản theo luồng sau: mỗi bước, tác nhân lấy đầu vào là trạng thái S_t, chọn hành động A_t (theo π), và nhận phản hồi từ môi trường gồm phần thưởng R_{t+1} và trạng thái mới S_{t+1} [23] [14]. Chu trình này lặp đi lặp lại cho đến khi đạt điều kiện dừng. Hàm giá trị và phần thưởng đóng vai trò làm thước đo để đánh giá chính sách: thông qua cập nhật giá trị, tác nhân biết được hành động nào dẫn đến phần thưởng dài hạn cao [22] [23]

2.2.3 Mô hình Quyết định Markov

Học tăng cường thường được định nghĩa thông qua Mô hình Quyết định Markov (Markov Decision Process – MDP). MDP cung cấp cách mô hình hóa môi trường cùng với tính chất Markov (tính phụ thuộc duy nhất vào trạng thái hiện tại), bao gồm tập các phần tử chính:

- Tập trạng thái S: Tập hợp tất cả các trạng thái có thể có của môi trường.
- Tập hành động A: Tập hợp các hành động mà tác nhân có thể thực hiện.
- Hàm chuyển trạng thái P($s' \mid s, a$) : Xác suất chuyển từ trạng thái s sang s' khi tác nhân chọn hành động a.

- Hàm phần thưởng $\mathrm{R}(s, a, s')$ hoặc $\mathrm{R}(s, a)$: Phần thưởng kỳ vọng nhận được khi thực hiện hành động a ở trạng thái s và chuyển đến trạng thái s'.
- Hệ số giảm giá $\gamma \in [0,1)$: Trọng số để tính phần thưởng tương lai (xác định tầm quan trọng của phần thưởng ở các bước sau so với phần thưởng ngay lúc đó).

Một MDP có thể được viết dưới dạng (S, A, P, R, γ) [25] .Quy tắc Markov ở đây nghĩa là xác suất và phần thưởng ở bước $\mathrm{t} + 1$ phụ thuộc duy nhất vào trạng thái S_t và hành động A_t, không phụ thuộc trực tiếp vào các hành động/trạng thái trước đó. Điều này cho phép định nghĩa phương trình *Bellman* liên kết giá trị của một trạng thái với giá trị của các trạng thái kế tiếp. Cụ thể, hàm giá trị trạng thái tối ưu $\mathrm{V} * (s)$ thoả mãn phương trình Bellman tối ưu:

$$V^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s, a)\, V^*(s') \right]$$

Và hàm giá trị hành động tối ưu $\mathrm{Q} * (s, a)$ thỏa mãm:

$$Q^*(s,a) = \mathrm{R}(s,a)\, \gamma \sum_{s' \in S} P(s' \mid s, a) + \max_{a' \in A} Q^*(s', a')$$

Những phương trình này nhấn mạnh rằng giá trị của hành động a ở trạng thái s dựa trên phần thưởng ngay tức thì $\mathrm{R}(s, a)$ cộng với giá trị dài hạn (được định nghĩa qua trạng thái mới s'). Nhiều thuật toán học tăng cường về cơ bản là các phép xấp xỉ để giải hệ phương trình Bellman này mà không cần biết trước hoàn toàn P và R thật.Vai trò của MDP trong học tăng cường là định khung chính xác vấn đề học dưới dạng bài toán tối ưu hoá trong MDP: tác nhân cố gắng tìm chính sách tối ưu π^* sao cho giá trị kì vọng nhận được cực đại. Điều này có thể được thực hiện bằng các thuật toán lặp lại hoặc các kỹ thuật học dựa trên mẫu mà không cần biết trước mô hình chính xác. Như đã đề cập, hầu hết các thuật toán RL hiện đại giả định môi trường có tính Markov và thường dựa trên ý tưởng của các phương pháp giải MDP thông qua lập trình động hay cập nhật Bellman [23]. Ví dụ, bài toán tìm chính sách tối ưu trong RL có thể coi là

bài toán tìm chính sách tối ưu trong MDP với phương pháp tối ưu hoá phần thưởng dài hạn [23].

2.2.4 Các thuật toán học tăng cường

Trong học tăng cường, có nhiều thuật toán nổi bật sử dụng các ý tưởng về cập nhật hàm giá trị và chính sách. Dưới đây trình bày chi tiết ba thuật toán điển hình: Q-Learning, SARSA và Deep Q-Network (DQN), đồng thời so sánh ưu nhược điểm của chúng.

a. Q-Learning

Q-Learning là thuật toán học tăng cường không theo chính sách (off-policy) tiêu biểu, được phát triển để ước lượng hàm giá trị-hành động tối ưu $Q*(s,a)$ trong MDP mà không cần biết mô hình. Thuật toán duy trì một bảng Q hoặc hàm xấp xỉ $Q(s,a)$ và cập nhật dần dần bằng quy tắc Bellman ngắn hạn. Cụ thể, khi tác nhân quan sát bước chuyển từ trạng thái s sang s' sau khi thực hiện a và nhận phần thưởng r, giá trị Q được cập nhật theo công thức:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Trong đó α là tốc độ học, γ là hệ số giảm giá , và $\max_{a'} Q(s',a') - Q(s,a)$ là giá trị tốt nhất mà tác nhân có thể đạt được từ trạng thái kế tiếp s′ theo bảng Q hiện tại [22]. Thuật toán lặp đi lặp lại cho đến khi bảng Q hội tụ, lúc đó $Q(s,a) \approx Q^*(s,a)$. Với học Q, tác nhân luôn cập nhật dựa trên giá trị tốt nhất trong tương lai, vì vậy Q-Learning có khả năng tìm chính sách tối ưu trong mọi trường hợp, miễn là đủ khám phá các trạng thái và hành động [22] [23].

Ví dụ minh hoạ: Trong mê cung, nếu phần thưởng là đặt tại đích, Q-Learning sẽ dần tăng giá trị Q cho những hành động dẫn đến đích. Qua nhiều tập trải nghiệm, bảng Q sẽ chứa giá trị kỳ vọng phần thưởng dài hạn cho mỗi bước đi, và tác nhân chọn hành động có Q lớn nhất để tối đa hóa phần thưởng. Như GeeksforGeeks mô tả, Q-Learning "dạy tác nhân gán giá trị cho các hành động dựa trên trạng thái hiện tại mà không cần mô hình môi trường" và có thể xử lý các bài toán chuyển đổi ngẫu nhiên [23].

Ưu điểm của Q-Learning: Do là thuật toán off-policy, nó học giá trị tối ưu bất kể chính sách đang dùng để thu thập dữ liệu, nên có khả năng hội tụ đến chính sách tối ưu với lượng thăm dò đủ lớn [22]. Cập nhật dựa trên max giúp Q-Learning hội tụ nhanh hơn đến cực trị toàn cục khi môi trường cho phép khám phá đầy đủ [22].

Nhược điểm của Q-Learning: Việc luôn dùng giá trị tối ưu giả định (max) làm cho Q-Learning có thể thổi phồng giá trị Q trong một số trường hợp, đặc biệt là khi ước lượng hàm giá trị có nhiễu; vì thuật toán "cho rằng hành động tối ưu nhất sẽ luôn được chọn" nên có thể đánh giá quá cao phần thưởng tương lai [22]. Ngoài ra, Q-Learning truyền thống chỉ áp dụng trực tiếp cho không gian trạng thái-hành động rời rạc nhỏ; khi không gian lớn hoặc liên tục cần dùng phép xấp xỉ Q hoặc học sâu, có thể gặp vấn đề bất ổn nếu không có biện pháp phòng ngừa [26].

b. SARSA

SARSA (State-Action-Reward-State-Action) là thuật toán học tăng cường theo chính sách (on-policy) điển hình. Khác với Q-Learning, SARSA cập nhật giá trị Q dựa trên hành động thực tế mà tác nhân đã thực hiện theo chính sách hiện thời. Cụ thể, mỗi lần tác nhân thực hiện hành động A_t ở trạng thái S_t, chuyển sang trạng thái S_{t+1} và nhận phần thưởng R_{t+1}, đồng thời lựa chọn tiếp hành động A_{t+1}(theo chính sách, ví dụ $\varepsilon -$ greedy), thì bảng Q cập nhật theo công thức:

$$\mathrm{Q}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Công thức này đòi hỏi biết được chính xác hành động a_{t+1} mà chính sách chọn ở trạng thái mới, nên phản ánh việc SARSA "học từ chính chính sách đang sử dụng" [22]. Ngắn gọn, SARSA cập nhật giá trị Q bằng phần thưởng tức thì và giá trị Q của cặp trạng thái-hành động kế tiếp.

Ví dụ: Trong mê cung, nếu chính sách của tác nhân có thêm khám phá (ví dụ $\varepsilon -$ greedy), thì SARSA tính toán kỳ vọng phần thưởng dài hạn dựa trên chính xác đường đi đã chọn (kể cả trường hợp chọn ngẫu nhiên). Điều này

khiến SARSA phản ánh đúng "phương sai chính sách" của tác nhân. Theo GeeksforGeeks, SARSA "giúp tác nhân học chính sách tối ưu dựa trên kinh nghiệm thực tế, nơi tác nhân liên tục cải thiện chính sách trong quá trình tương tác" [22]. Thuật toán cập nhật Q tương tự Bellman, nhưng sử dụng giá trị Q của hành động kế tiếp thực tế a_{t+1} thay vì giá trị max ở trạng thái mới.

Ưu điểm của SARSA: Vì là on-policy, SARSA ổn định hơn khi tác nhân tuân theo chính sách thăm dò (ví dụ $\varepsilon -$ greedy), do nó tính đến hình phạt từ các bước đi ngẫu nhiên. Nhờ vậy, SARSA thường ít "thổi phồng" Q-values hơn Q-Learning [22]. Thuật toán này phù hợp trong các tình huống cần an toàn và ổn định, nơi tác nhân nên nhận diện rủi ro gắn với khám phá [22].

Nhược điểm của SARSA: Cập nhật dựa trên hành động thực tế có thể làm cho tốc độ hội tụ chậm hơn so với Q-Learning, bởi vì SARSA liên tục thích nghi theo chính sách hiện tại chứ không nhất thiết hướng tới tối ưu toàn cục ngay lập tức [22] Nếu chính sách thăm dò mạnh (nhiều ngẫu nhiên), SARSA có thể mất nhiều thời gian để đạt chính sách tối ưu. Ngoài ra, như Q-Learning, SARSA cơ bản cũng chỉ áp dụng trực tiếp cho không gian nhỏ rời rạc; với không gian lớn cần thêm xấp xỉ hoặc học sâu.

c. Deep Q-Network

Deep Q-Network (DQN) là một bước đột phá nhằm kết hợp mạng nơ-ron sâu (deep neural network) với Q-Learning để giải quyết các bài toán RL với không gian trạng thái lớn, liên tục, hoặc dữ liệu đầu vào dạng tín hiệu cao chiều (như ảnh). Bản chất của DQN là dùng một mạng nơ-ron convolutional để xấp xỉ hàm Q. Trong công trình kinh điển của Mnih và cộng sự, họ thiết kế DQN cho tác vụ chơi game Atari, với đầu vào là ảnh RGB từ màn hình [26]. Mạng gồm nhiều lớp tích chập (convolutional layers) và lớp kết nối đầy đủ (fully connected), xuất ra giá trị Q cho mỗi hành động có thể [26].

Điểm mấu chốt của DQN là cách huấn luyện mạng Q ổn định hơn, khắc phục các bất ổn khi dùng hàm xấp xỉ phi tuyến trong RL [26]. Để làm được điều này, DQN sử dụng hai ý tưởng chính:

- Kinh nghiệm hồi quy (Experience Replay)**:** Mỗi lần tác nhân tương tác, chuyển (state, action, reward, next state) được lưu vào bộ nhớ replay. Quá trình học sẽ rút ngẫu nhiên các mẫu từ bộ nhớ này để cập nhật trọng số mạng, giúp làm giảm tương quan trong dãy quan sát và tránh trùng lập dữ liệu [26].
- Mạng mục tiêu **:** Giữ hai bản sao mạng Q: một bản cập nhật thường xuyên, một bản mục tiêu được cập nhật gián đoạn sau mỗi CCC bước. Bằng cách này, mục tiêu trong hàm mất mát được tính bằng bản sao giữ cố định một thời gian, giảm độ dao động trong quá trình học [26].

Thuật toán DQN tận dụng Q-Learning, nghĩa là vẫn cập nhật giá trị Q theo quy tắc tương tự, nhưng dùng mạng nơ-ron để ước lượng $\mathrm{Q}(s, a; \theta)$ với tham số θ. Trong nghiên cứu của Mnih et al., DQN đã cho kết quả vượt trội, "nhận đầu vào chỉ từ điểm ảnh và điểm số trò chơi, vượt qua mọi thuật toán trước đó và đạt mức trình độ tương đương hoặc vượt người chơi chuyên nghiệp trên 49 trò chơi Atari" [26]. Đây là minh chứng cho ưu thế của DQN trong xử lý đầu vào hình ảnh độ phân giải cao và bài toán có không gian trạng thái rất lớn.

Ưu điểm của DQN**:** Khả năng học trực tiếp từ dữ liệu thô phức tạp (như hình ảnh, âm thanh) và tìm chính sách tốt mà không cần đặc trưng thủ công. Với kiến trúc học sâu, DQN có thể tổng hợp thông tin ở nhiều cấp độ trừu tượng, phù hợp với các bài toán phức tạp. DQN cũng tận dụng kinh nghiệm hồi quy nên sử dụng dữ liệu hiệu quả hơn (tăng sample efficiency) [27].

Nhược điểm của DQN**:** Đào tạo DQN thường rất tốn kém tài nguyên tính toán và dữ liệu. Mạng phải có lượng lớn trải nghiệm để khớp hàm Q, dẫn đến thời gian huấn luyện lâu. Nếu không gian hành động liên tục, DQN phải được mở rộng . Hơn nữa, thiết lập các siêu tham số (như kích thước replay, tần suất cập nhật mục tiêu) rất nhạy cảm, cần tinh chỉnh kỹ để đảm bảo hội tụ. DQN cũng dễ gặp vấn đề quá khớp hoặc bất ổn nếu không sử dụng replay và target network như đề xuất [26].

d. So sánh ưu và nhược điểm của các thuật toán

Bảng 1.2: *Bảng so sánh các thuật toán học tăng cường*

Tiêu chí	Q-Learning	SARSA	DQN(Deep Q-Network)
Loại thuật toán	Off-policy	On-policy	Off-policy(dựa trên Q-learning)
Tốc độ hội tụ	Nhanh hơn (nếu thăm dò tốt)	Chậm hơn (Phục thuộc chính sách hiện tại)	Chậm nhất (do huấn luyện mạng phức tạp)
Độ ổn định	Dễ bị overestimation	ổn định hơn, ít bị đánh giá quá cao	Bất ổn ban đầu, cần dung replay và target network
Không gian ứng dụng	Tốt cho không gian rời rạc, nhỏ	Tương tự Q-learning	Tốt cho không gian lớn, liên tục, ảnh

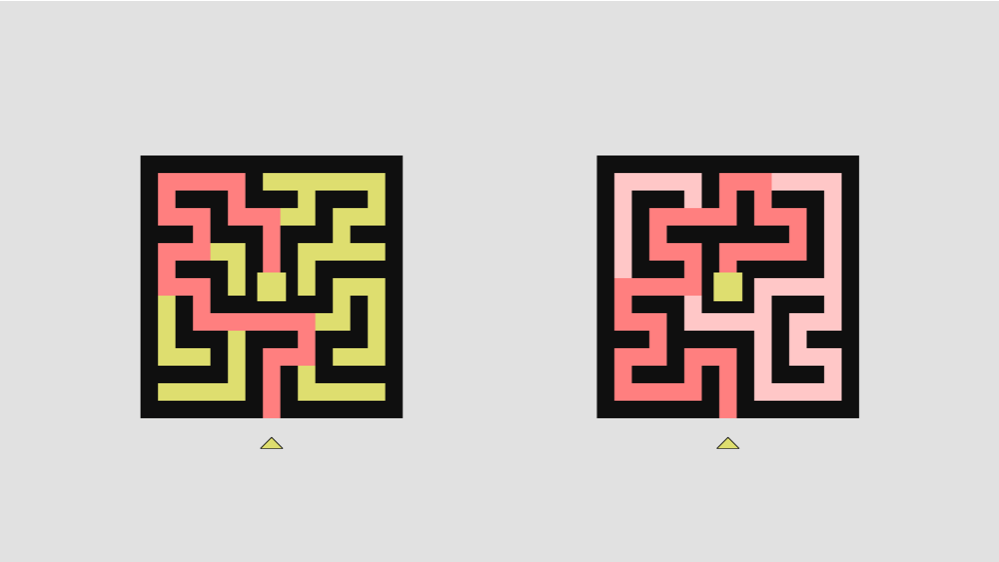
CHƯƠNG 3: THỰC NGHIỆM

3.1. Thiết kế mê cung

3.1.1 Phân loại mê cung: Perfect Maze và Imperfect Maze

Mê cung là cấu trúc gồm các ô và tường tạo thành một lưới đường đi. Trong đó, *mê cung hoàn hảo* (Perfect Maze) là mê cung không có vòng lặp và mọi ô đều liên thông, tức là bất kỳ hai ô nào đều có duy nhất một đường đi nối giữa chúng [28]. Ngược lại, *mê cung không hoàn hảo (Imperfect Maze)* cho phép tồn tại nhiều đường đi giữa hai ô và có thể xuất hiện các vòng lặp do có đường đi bị trùng khúc [29].

Hình minh họa dưới đây so sánh một mê cung hoàn hảo và một mê cung không hoàn hảo:



Hình 3.1: Hình minh họa hai ví dụ về mê cung:

Chú thích: bên trái là mê cung hoàn hảo (không có vòng lặp), bên phải không hoàn hảo mê(có vòng lặp).

Trong mê cung hoàn hảo, tác nhân chỉ có một lối duy nhất đến đích, giúp giải thuật tìm đường trở nên đơn giản hơn. Mặt khác, mê cung không hoàn hảo tăng độ khó vì cho phép có nhiều đường đi, tạo ra nhiều lựa chọn khác nhau cho tác nhân [29].

Các thuật toán sinh mê cung thường tạo ra một cây khung ngẫu nhiên trên đồ thị của các ô. Ba thuật toán phổ biến để sinh mê cung là Depth-First Search (DFS) (hay còn gọi là *đệ quy lui*), Prim (phiên bản ngẫu nhiên của thuật toán Prim) và Wilson (dùng ngẫu nhiên giải duyệt đường đi có xóa vòng lặp).

3.1.2 Các thuật toán tạo mê cung: DFS, Prim, Wilson

a. Thuật toán DFS (Depth-First Search)

Thuật toán DFS sinh ra mê cung theo cách đi sâu tới tận cùng rồi mới quay lui để mở đường mới [30]. Cụ thể, bắt đầu từ một ô ngẫu nhiên, thuật toán thực hiện:

1. Khởi tạo: Đánh dấu tất cả các ô là chưa thăm. Chọn một ô ban đầu, đánh dấu ô đó đã thăm, đưa vào ngăn xếp để tiện backtracking [30].
2. Chọn ô kế: Từ ô hiện tại, chọn ngẫu nhiên một ô láng giềng chưa thăm. Xóa bức tường ngăn giữa hai ô và đánh dấu ô láng giềng là đã thăm [30]. Đẩy ô hiện tại vào ngăn xếp, sau đó chuyển đến ô láng giềng đó.
3. Tiếp tục: Lặp lại bước 2 cho đến khi ô hiện tại không còn láng giềng chưa thăm . Khi gặp ngõ cụt, thuật toán quay lui: lấy một ô từ ngăn xếp, đặt làm ô hiện tại và tiếp tục bước 2 [30].
4. Kết thúc: Thuật toán kết thúc khi mọi ô đã được thăm (ngăn xếp rỗng).

Mã giả tương ứng:

Thuật toán sinh mê cung bằng DFS:

1. Khởi tạo tất cả ô là chưa thăm.
2. Chọn ngẫu nhiên ô ban đầu, đánh dấu đã thăm, đẩy vào stack.
3. Trong khi stack không rỗng:
   i Lấy ô hiện tại = pop(stack).
   ii Nếu ô hiện tại có láng giềng chưa thăm:
   iii Đẩy ô hiện tại lại vào stack.
   IV Chọn ngẫu nhiên một ô láng giềng chưa thăm.
   V Xóa bức tường ngăn giữa hai ô.
   VI Đánh dấu láng giềng là đã thăm, đẩy nó vào stack.

Ví dụ minh họa: bắt đầu từ ô góc trên bên trái, DFS sẽ lần lượt chọn một hướng ngẫu nhiên, xóa tường và đi sâu. Khi gặp dead-end, nó sẽ quay lui, đảm bảo tất cả ô cuối cùng đều được đi qua [30].

Đặc điểm

- Mê cung tạo ra từ DFS thường có các hành lang dài và ít ngã rẽ.
- Độ phân nhánh thấp, phù hợp để kiểm tra khả năng học của tác nhân trong các đường đi phức tạp.
- Do tính chất của DFS, mê cung sẽ luôn đảm bảo có đường đi từ mọi ô đến mọi ô khác (nếu triển khai đúng), tạo ra một perfect maze.

b. Thuật toán Prim

Thuật toán Prim cho việc sinh mê cung dựa trên việc mở rộng dần mê cung hiện tại bằng cách thêm những ô từ một danh sách tường [30]. Cụ thể, thuật toán có thể thực hiện như sau:

1. Khởi tạo: Xây dựng lưới chỉ toàn tường. Chọn một ô ban đầu, đánh dấu ô đó là đã thăm và thêm tất cả các bức tường quanh ô vào danh sách tường (wall list) [30].
2. Mở rộng: Lặp cho đến khi danh sách tường trống:
    - Chọn ngẫu nhiên một bức tường từ danh sách. Bức tường này phân cách hai ô: nếu chỉ có một trong hai ô đã được thăm, thì:
        - Xóa bức tường đó (tạo hành lang).
        - Đánh dấu ô kia là đã thăm, đồng thời thêm các bức tường xung quanh ô mới vào danh sách [30].
    - Loại bỏ bức tường đã chọn ra khỏi danh sách.
3. Kết thúc: Khi danh sách tường rỗng, mọi ô đã được kết nối thành một mê cung hoàn chỉnh.

Mã giả [30] có thể diễn giải

Thuật toán sinh mê cung bằng Prim:

1. Bắt đầu với lưới đầy tường.
2. Chọn ô đầu tiên, đánh dấu đã thăm, thêm tường xung quanh vào danh sách.
3. Trong khi danh sách tường không rỗng:

    I Chọn ngẫu nhiên một tường từ danh sách.

    II Giả sử nó ngăn giữa ô A và ô B.

    III Nếu chỉ có một trong hai ô (A hoặc B) đã thăm:

    IV Xóa bức tường (làm hành lang).

    V Đánh dấu ô chưa thăm là đã thăm.

    VI Thêm tường xung quanh ô mới vào danh sách.

    VII Xóa bức tường này khỏi danh sách.

Thuật toán Prim sinh ra mê cung có nhiều ngõ cụt ngắn hơn, ít hành lang dài so với DFS [30]. Hình sinh mê cung bằng Prim cho thấy cách tường được lựa chọn ngẫu nhiên và dần được mở rộng từ ô ban đầu [30].

Đặc điểm

- Mê cung sinh bởi thuật toán Prim thường có nhiều ngã rẽ và các đoạn đường ngắn, tạo thành mạng lưới rẽ nhánh đều đặn hơn so với DFS.
- Do tính ngẫu nhiên trong việc chọn tường biên, mỗi mê cung tạo ra sẽ có hình dạng khác nhau, làm tăng tính đa dạng cho môi trường học tập
- Thuật toán này cũng đảm bảo tạo ra mê cung hoàn hảo – không chu trình, chỉ có một đường duy nhất nối giữa hai điểm bất kỳ trong mê cung.

c. Thuật toán Wilson

Wilson là thuật toán sinh mê cung theo cây khung ngẫu nhiên mà không tạo độ nghiêng, dựa trên phép random walk với cơ chế xóa vòng lặp. Nguyên lý của Wilson như sau [30]:

1. Khởi tạo: Bắt đầu với một mê cung chứa một ô tùy ý đã được đánh dấu (visited).
2. Lập đường đi ngẫu nhiên: Chọn một ô chưa thăm bất kỳ. Thực hiện một phép đi bộ ngẫu nhiên từ ô đó cho đến khi chạm tới một ô đã thăm trong mê cung hiện tại.
3. Xóa vòng lặp: Trong quá trình đi bộ, nếu đường đi ngẫu nhiên tạo thành một vòng (nghĩa là bước lại đến một ô đã từng xuất hiện trên đường đi hiện tại), thì xóa phần vòng lặp đó và tiếp tục đi tiếp [30].
4. Thêm vào mê cung: Khi đường đi ngẫu nhiên chạm ô đã thăm, các bức tường trên đường đi được xóa và đánh dấu toàn bộ các ô trên đường đó là đã thăm. Như vậy, đường đi này được thêm hoàn toàn vào mê cung.
5. Lặp lại: Lặp lại các bước 2–4 cho đến khi tất cả ô đều được thăm (không còn ô chưa thăm).

Thuật toán này sẽ bắt đầu từ một ô gốc, sau đó với mỗi ô mới, di chuyển ngẫu nhiên, xóa vòng lặp tự động và gắn kết với mê cung hiện tại. Kết quả là một mê cung hoàn hảo mà không bị lệch chi tiết nào [30]. Đặc điểm của Wilson:

là phân bố đồng đều trên tập các cây khung, tức mọi cây khung đều có xác suất tạo ra như nhau [30]. Mã giả cho Wilson có thể viết

Thuật toán sinh mê cung bằng wilson

1. Khởi tạo mê cung với một ô đã thăm.
2. Trong khi còn ô chưa thăm:

I, Chọn ngẫu nhiên một ô chưa thăm làm start.

Đường đi = [start].

II, Trong khi ô cuối trong đường đi chưa chạm mê cung:

i Chọn một ô láng giềng ngẫu nhiên.

ii Nếu ô đó đã xuất hiện trong đường đi:

Xóa phần chu trình (loại bỏ các ô sau ô đó trong đường đi).

iii Thêm ô đó vào cuối đường đi.

III, Thêm toàn bộ đường đi vào mê cung (xóa tường giữa các ô).

IV, Đánh dấu các ô trên đường đi là đã thăm.

3.1.3 Triển khai môi trường RL

Môi trường giải mê cung được thiết kế tương tự với các môi trường trong thư viện OpenAI Gym, đảm bảo có thể tích hợp với các thuật toán học tăng cường một cách thuận tiện. Cấu trúc môi trường được tổ chức thành các mô-đun để tách biệt rõ ràng giữa phần sinh mê cung và phần tương tác học tăng cường. Cụ thể:

- *Base_generator.py*: Định nghĩa lớp trừu tượng BaseGenerator, cung cấp giao diện chung cho các thuật toán sinh mê cung. Lớp này quy định các phương thức cơ bản như khởi tạo mê cung, thiết lập vị trí bắt đầu/kết thúc, và truy xuất ma trận mê cung.
- *dfs_generator.py*: Kế thừa từ BaseGenerator và thực hiện thuật toán sinh mê cung dựa trên Depth-First Search **(**tương tự cho 2 thuật toán wilson–*wilson_generator.py* và prim–*prim_generator.py***)**. Mỗi mê cung sinh ra là một "perfect maze" — chỉ tồn tại duy nhất một đường

đi từ điểm bắt đầu đến đích. Trong trường hợp không tồn tại đường đi nào, lớp có thể tạo trực tiếp 1 đường đi từ điểm bắt đầu đến điểm kết thúc

- maze_env.py: Triển khai lớp môi trường MazeEnv cho tác nhân học tăng cường. Lớp này đóng vai trò trung gian giữa tác nhân và mê cung, cung cấp các chức năng tương tác như *step()*, *reset()*, *get_state_size(), get_action_size()*. Điểm đặc biệt trong lớp này là việc tìm đường đi ngắn nhất từ điểm bắt đầu đến điểm kết thúc bằng thuật toán BFS, dựa trên kết quả của thuật toán chúng ta có thể biết so sánh với tác nhân của chúng ta đã huấn luyện tốt chưa

Thực hiện thiết kế môi trường:

Mỗi phiên bản môi trường MazeEnv bao gồm các yếu tố sau:

- Trạng thái: được biểu diễn bằng tọa độ vị trí hiện tại của tác nhân trong mê cung, ví dụ (x, y)
- Tập hành động: Bao gồm 4 hành động rời rạc tương ứng với 4 hướng di chuyển: lên, xuống, trái phải
- Điều kiện kết thúc (done): Môi trường kết thúc khi agent chạm đến ô đích hoặc vượt quá số bước tối đa được phép.

Khung triển khai cơ bản của lớp MazeEnv:

```python
class MazeEnvironment:
    """
    Môi trường mê cung cho bài toán học tăng cường.

    Môi trường này tuân theo giao diện tương tự như OpenAI Gym, với các phương thức
    reset(), step() và render() để agent có thể tương tác.
    """
    PATH = BaseMazeGenerator.PATH
    WALL = BaseMazeGenerator.WALL
    START = BaseMazeGenerator.START
    GOAL = BaseMazeGenerator.GOAL

    # Các hành động: 0: lên, 1: xuống, 2: trái, 3: phải,
    ACTIONS = [(0, -1), (0, 1), (-1, 0), (1, 0)]
    ACTION_NAMES = ["UP", "DOWN", "LEFT", "RIGHT"]

    def __init__(self, maze: Optional[np.ndarray] = None,
                 maze_generator: Optional[BaseMazeGenerator] = None,
                 start_pos: Optional[Tuple[int, int]] = None,
                 goal_pos: Optional[Tuple[int, int]] = None,
                 max_steps: int = 1000,
                 move_reward: float = -1.0,
                 wall_penalty: float = -5.0,
                 goal_reward: float = 100.0,
                 time_penalty: float = -0.1):
        """
        Khởi tạo môi trường mê cung.

        Args:
            maze (np.ndarray, optional): Ma trận mê cung đã tạo sẵn
            maze_generator (BaseMazeGenerator, optional): Bộ sinh mê cung
            start_pos (Tuple[int, int], optional): Vị trí bắt đầu, mặc định từ maze
            goal_pos (Tuple[int, int], optional): Vị trí đích, mặc định từ maze
            max_steps (int): Số bước tối đa trong một episode
            move_reward (float): Phần thưởng cho mỗi bước di chuyển
            wall_penalty (float): Phạt khi đi vào tường
            goal_reward (float): Phần thưởng khi đến đích
            time_penalty (float): Phạt theo thời gian (tăng theo số bước)
        """
        self.maze_generator = maze_generator
        self.max_steps = max_steps
        self.move_reward = move_reward
        self.wall_penalty = wall_penalty
        self.goal_reward = goal_reward
        self.time_penalty = time_penalty
```

Hình 3.2: Code khởi tạo môi trường mê cung

3.2. Xây dựng agent và chiến lược hành động

Trong học tăng cường, agent là thành phần trung tâm có nhiệm vụ tương tác với môi trường để học cách đưa ra hành động tối ưu thông qua quá trình

thử–sai. Trong đồ án này, agent được thiết kế để di chuyển trong mê cung hai chiều, nơi mỗi trạng thái tương ứng với một ô trong lưới và mỗi hành động là một bước di chuyển sang một trong bốn hướng: lên, xuống, trái hoặc phải.

Mỗi agent đều được gắn với một chính sách hành động — chiến lược mà theo đó agent chọn hành động từ một trạng thái cụ thể. Quá trình cập nhật chính sách được thực hiện thông qua các thuật toán học tăng cường như Q-learning và Sarsa, nơi giá trị Q (Q-value) đại diện cho kỳ vọng phần thưởng của một hành động tại một trạng thái. Agent liên tục cập nhật bảng Q và điều chỉnh hành vi để tối đa hóa tổng phần thưởng nhận được về lâu dài.

3.2.1 Lựa chọn policy

a. ε -greedy

Chiến lược epsilon-greedy là một trong những phương pháp đơn giản và phổ biến nhất để lựa chọn hành động trong học tăng cường. Ý tưởng chính là: với xác suất 1 - ε, agent sẽ chọn hành động có giá trị Q lớn nhất tại trạng thái hiện tại, còn với xác suất ε, agent sẽ chọn một hành động bất kỳ theo phân phối đồng đều.

Cụ thể:

- Nếu r là một số ngẫu nhiên thuộc đoạn [0, 1):
  - Nếu r < ε, chọn một hành động ngẫu nhiên.
  - Ngược lại, chọn hành động a sao cho Q(s, a) là lớn nhất.

$$a = \begin{cases} argmax_a Q(s,a), & với\ xác\ suất\ 1-\varepsilon \\ random(A), & với\ \ xác\ suất\ \ \varepsilon \end{cases}$$

Trong đó:

- $\varepsilon \in [0, 1]$ là tham số điều chỉnh tỷ lệ khám phá.
- Q(s, a) là giá trị Q của hành động a tại trạng thái s.

Cách tiếp cận này giúp agent không bị mắc kẹt trong các hành động hiện tại mà có cơ hội thử các hành động mới, đồng thời vẫn ưu tiên các hành động có giá trị cao. Tuy nhiên, nếu ε không được điều chỉnh phù hợp theo từng giai

đoạn học, agent có thể khám phá quá nhiều hoặc quá ít, dẫn đến hiệu quả học tập không tối ưu.

b. Softmax

Softmax là một chiến lược lựa chọn hành động dựa trên phân phối xác suất được tính từ các giá trị Q, giúp phân biệt rõ ràng mức độ ưu tiên của từng hành động. Khác với ε-greedy, softmax không chọn hành động tốt nhất hoặc ngẫu nhiên hoàn toàn, mà chọn hành động theo xác suất tương ứng với giá trị Q của nó.

Xác suất chọn hành động a tại trạng thái s được tính theo công thức:

$$P(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}}$$

Trong đó:

- Q(s, a) là giá trị Q tại trạng thái s với hành động a.
- τ là tham số nhiệt độ (ttôiperature), $\tau > 0$.
- A là tập hợp các hành động có thể thực hiện tại trạng thái s.

Khi τ lớn, sự khác biệt giữa các xác suất là nhỏ, dẫn đến hành vi gần giống ngẫu nhiên (nhiều khám phá). Khi τ nhỏ, xác suất sẽ tập trung mạnh vào hành động có giá trị Q cao nhất (nhiều khai thác). Vì vậy, lựa chọn τ phù hợp giúp softmax trở thành một công cụ hữu hiệu để kiểm soát hành vi agent trong quá trình học.

c. Decaying ε

Decaying epsilon-greedy là phiên bản cải tiến của chiến lược epsilon-greedy, trong đó giá trị epsilon được giảm dần theo thời gian huấn luyện. Mục tiêu là đảm bảo rằng agent có khả năng khám phá mạnh trong giai đoạn đầu (khi chưa biết nhiều về môi trường), và dần chuyển sang khai thác khi đã tích lũy đủ kinh nghiệm.

Giá trị epsilon ở thời điểm t được tính theo công thức:

$$\varepsilon_t = \max(\varepsilon_{\min}, \varepsilon_0 \cdot \exp^{-kt})$$

Trong đó:

- ε_0 là giá trị khởi tạo epsilon ban đầu (ví dụ 1.0).
- ε_{min} là giá trị epsilon nhỏ nhất cần duy trì (ví dụ 0.01).
- k là hệ số điều chỉnh tốc độ giảm.
- t là số bước hoặc số episode đã diễn ra.

Chiến lược này giúp agent tiếp cận được nhiều hành động khác nhau trong giai đoạn đầu, từ đó học được bức tranh tổng thể của môi trường. Sau khi đã có nền tảng vững, agent sẽ khai thác triệt để các hành động tốt nhất, nhờ đó cải thiện hiệu quả và tốc độ hội tụ trong quá trình huấn luyện.

3.2.2 Kiến trúc Agent

Trong hệ thống học tăng cường giải mê cung, tác nhân là thành phần trung tâm có nhiệm vụ tương tác với môi trường để học chính sách hành động tối ưu. Agent được cài đặt dưới dạng một lớp với các thành phần chính sau:

- Q-table hoặc Neural Network: tùy thuộc vào thuật toán (Q-learning/Sarsa sử dụng bảng Q; DQN sử dụng mạng neural).
- Hàm chọn hành động (policy): như ε-greedy.
- Hàm cập nhật chính sách học được.
- Buffer (nếu có): trong DQN có sử dụng replay buffer để lưu kinh nghiệm

3.3. Quá trình huấn luyện và tối ưu hóa

3.3.1 Tham số huấn luyện

Trong quá trình huấn luyện agent để giải quyết bài toán mê cung bằng học tăng cường, việc thiết lập các tham số huấn luyện đóng vai trò quyết định đến hiệu quả học và khả năng tổng quát hóa chính sách hành động. Các tham số này được tổ chức rõ ràng và lưu trữ trong một file cấu hình trung tâm (utils/config.py), cho phép tái sử dụng và dễ dàng điều chỉnh giữa các lần huấn luyện.

Các tham số chính bao gồm:

1. Tham số môi trường

- Kích thước mê cung (MAZE_SIZES): Hỗ trợ nhiều cấp độ mê cung với độ phức tạp tăng dần:
  - small: 11 × 11
  - medium: 15 × 15
  - large: 21 × 21
  - xlarge: 31 × 31
- Phần thưởng (Reward):
  - MOVE_REWARD = - 0.1: Hình phạt nhẹ cho mỗi bước đi, nhằm khuyến khích agent tìm đường ngắn nhất.
  - WALL_PENALTY = -2.0: Hình phạt mạnh khi agent cố đi vào tường, giúp tránh hành vi kém hiệu quả.
  - GOAL_REWARD = 100.0: Phần thưởng lớn cho việc đến đích, giúp agent xác định rõ mục tiêu.
  - TIME_PENALTY = -0.01: Giảm nhẹ reward theo thời gian, giúp agent hành động nhanh hơn.
  - MAX_STEPS = 2000: Giới hạn số bước tối đa để tránh các episode kéo dài không cần thiết.

2. Tham số học tăng cường

- Tốc độ học (LEARNING_RATE = 0.2**)**: Điều chỉnh mức độ cập nhật Q-value sau mỗi bước. Giá trị cao hơn giúp học nhanh nhưng có thể gây dao động.
- Hệ số chiết khấu (DISCOUNT_FACTOR = 0.99): Xác định mức độ ưu tiên phần thưởng tương lai, với giá trị cao ưu tiên các chuỗi hành động dài hạn.
- Chiến lược khám phá–khai thác (Exploration–Exploitation):
  - EXPLORATION_RATE = 1.0: Agent bắt đầu với trạng thái khám phá hoàn toàn (random).

- EXPLORATION_DECAY = 0.998: Tốc độ giảm của epsilon sau mỗi episode.
- MIN_EXPLORATION = 0.01: Giới hạn dưới để đảm bảo agent vẫn còn khả năng khám phá.

3. Tham số huấn luyện

- TRAINING_EPISODES = 2000: Tổng số episode huấn luyện để học chính sách hành động.
- EVAL_INTERVAL = 100: Mỗi 100 episode sẽ thực hiện đánh giá agent để theo dõi quá trình hội tụ và điều chỉnh chiến lược nếu cần.

4. Kỹ thuật mở rộng

- Double Q-Learning:
  - USE_DOUBLE_Q = True: Kích hoạt Double Q-learning để giảm bias trong cập nhật Q-value.
  - TARGET_UPDATE_STEPS = 500: Cập nhật Q-target định kỳ nhằm tăng ổn định.
- Replay Buffer (chỉ áp dụng cho DQN):
  - REPLAY_BUFFER_SIZE = 10000: Kích thước bộ nhớ lưu trữ kinh nghiệm để huấn luyện lại.
  - REPLAY_BATCH_SIZE = 64: Số lượng mẫu chọn ra mỗi lần huấn luyện mini-batch.

5. Cấu hình khác

- RANDOM_SEED = 42: Đặt hạt giống ngẫu nhiên để đảm bảo tính tái lập của các thí nghiệm.
- Q_LEARNING_MODEL_DIR, SARSA_MODEL_DIR: Đường dẫn lưu mô hình học được để sử dụng lại hoặc đánh giá.

Việc kiểm soát và phân tích các tham số trên là cần thiết để đảm bảo tính khách quan và nhất quán trong đánh giá kết quả huấn luyện. Các thiết lập này có thể được thay đổi linh hoạt để thực hiện các thí nghiệm khác nhau, đánh giá

tác động của từng tham số đến hiệu suất và tốc độ hội tụ của agent trong môi trường mê cung.

```python
# config.py
PROJECT_NAME = "Maze Reinforcement Learning"
VERSION = "1.0.0"

MAZE_SIZES = {
    "small": (11, 11),  # Mê cung 11x11
    "medium": (15, 15), # Mê cung 15x15
    "large": (21, 21),  # Mê cung 21x21
    "xlarge": (31, 31)  # Mê cung 31x31
}

# Tham số học tăng cường
LEARNING_RATE = 0.1         # Tốc độ học cao hơn (Alpha)
DISCOUNT_FACTOR = 0.99      # Hệ số giảm cao hơn (Gamma)
EXPLORATION_RATE = 1.0      # Tỷ lệ khám phá ban đầu (Epsilon)
EXPLORATION_DECAY = 0.995   # Giảm chậm hơn
MIN_EXPLORATION = 0.01      # Mức khám phá tối thiểu cao hơn

# Tham số môi trường
MOVE_REWARD = -0.05          # Giảm phạt cho mỗi bước di chuyển
WALL_PENALTY = -2.0         # Giảm phạt khi đi vào tường
GOAL_REWARD = 100.0         # Phần thưởng khi đến đích
TIME_PENALTY = -0.001        # Giảm phạt theo thời gian
MAX_STEPS = 2000            # Tăng số bước tối đa

# Tham số huấn luyện
TRAINING_EPISODES = 2000    # Tăng số episode
EVAL_INTERVAL = 100         # Đánh giá sau mỗi 100 episode

# Đường dẫn mô hình
Q_LEARNING_MODEL_DIR = "./models/q_learning"
SARSA_MODEL_DIR = "./models/sarsa"
DQN_MODEL_DIR = "./models/dqn"

# Hạt giống ngẫu nhiên
RANDOM_SEED = 42

# Tham số mới cho Experience Replay
REPLAY_BUFFER_SIZE = 1000  # Kích thước buffer
REPLAY_BATCH_SIZE = 64
REPLAY_START_SIZE = 1000    # Kích thước batch
ADAPTIVE_ALPHA = True       # Sử dụng tốc độ học thích ứng
PRIORITIZED_REPLAY = True   # Sử dụng prioritized experience replay
# Thêm các hằng số cấu hình cho DQN
DQN_HIDDEN_SIZE = 128
DQN_BATCH_SIZE = 64
DQN_BUFFER_SIZE = 100000
DQN_TARGET_UPDATE_FREQ = 1000
DQN_LEARNING_RATE = 0.001  # Mặc định cho DQN

# Tham số cho Double Q-Learning
USE_DOUBLE_Q = True
TARGET_UPDATE_STEPS = 500



def get_model_filename(agent_type, maze_size, episodes):
    height, width = maze_size
    return f"{agent_type}_{height}x{width}_{episodes}ep.pkl"
```

Hình 3.3: Code các tham số huấn luyện của agent

3.3.2 Hiệu ứng hyperparameter

Hiệu quả và tốc độ hội tụ của agent trong môi trường giải mê cung phụ thuộc mạnh mẽ vào các hyperparameter – tức là những tham số không được học từ dữ liệu, mà được cấu hình thủ công trước khi huấn luyện. Các hyperparameter ảnh hưởng đến quá trình học, chiến lược khám phá, và độ ổn định của mô hình. Việc tinh chỉnh và phân tích các tham số này là một phần thiết yếu trong việc tối ưu hóa mô hình học tăng cường.

Dưới đây là các hyperparameter chính và phân tích ảnh hưởng của từng nhóm:

1. Tốc độ học (LEARNING_RATE)

- Đại diện cho hệ số cập nhật giá trị trong hàm Q.
- Giá trị cao (ví dụ $\alpha = 0.5$ hoặc lớn hơn) giúp mô hình học nhanh hơn nhưng dễ dao động nếu môi trường có tính ngẫu nhiên.
- Giá trị thấp (ví dụ $\alpha < 0.1$) dẫn đến cập nhật ổn định hơn nhưng yêu cầu nhiều episode hơn để hội tụ.

Ví dụ: Khi tăng LEARNING_RATE từ 0.1 lên 0.3, tốc độ hội tụ nhanh hơn ở các mê cung nhỏ, nhưng ở mê cung lớn, Q-value không ổn định dẫn đến reward dao động.

2. Hệ số chiết khấu (DISCOUNT_FACTOR)

- Ký hiệu γ, dùng để xác định tầm quan trọng của phần thưởng tương lai.
- γ gần 1.0 khuyến khích agent cân nhắc phần thưởng dài hạn — thích hợp trong các mê cung lớn.
- γ thấp sẽ dẫn đến hành vi ngắn hạn, chỉ quan tâm phần thưởng gần.

Ví dụ: Với $\gamma = 0.99$, agent học được đường đi ngắn hơn trong các mê cung lớn. Khi giảm γ xuống 0.8, agent thường dừng sớm hoặc không tối ưu hóa đường đi.

3. Chiến lược khám phá (ε, ε_decay, min_ε)

- Điều khiển sự cân bằng giữa khai thác chính sách hiện tại và khám phá hành động mới.
- ε ban đầu thường đặt là 1.0, sau đó giảm dần qua mỗi episode.
- Tốc độ giảm (ε_decay) và giá trị tối thiểu (min_ε) cần điều chỉnh phù hợp để không dừng khám phá quá sớm.

Ví dụ: Khi ε_decay = 0.998, agent duy trì khám phá trong ~1000 episode đầu, từ đó học được chính sách tốt hơn. Nếu decay quá nhanh (ε_decay = 0.95), agent có xu hướng khai thác sớm và dễ rơi vào tối ưu cục bộ.

4. Replay Buffer và Batch Size (áp dụng cho DQN)

- Replay buffer giúp lưu trữ và học lại từ kinh nghiệm cũ, làm giảm tương quan giữa các mẫu học liên tiếp.
- Batch size điều chỉnh số lượng kinh nghiệm được lấy ra mỗi lần cập nhật trọng số.

Ví dụ: Với REPLAY_BUFFER_SIZE = 10000 và BATCH_SIZE = 64, mô hình DQN có reward ổn định hơn, tránh overfit vào các tình huống gần đây.

5. Double Q-Learning (USE_DOUBLE_Q, TARGET_UPDATE_STEPS)

- Giúp giảm sai lệch trong cập nhật Q-value do overestimation.
- TARGET_UPDATE_STEPS quyết định tần suất cập nhật mô hình mục tiêu.

Ví dụ: Khi bật USE_DOUBLE_Q = True, reward tăng chậm hơn lúc đầu nhưng về lâu dài ổn định và hội tụ chính xác hơn, nhất là trong mê cung có nhiều lựa chọn gần như ngang nhau.

3.3.3 Theo dõi Reward theo episode

Trong quá trình huấn luyện agent bằng các thuật toán học tăng cường như Q-Learning, SARSA hay DQN, một chỉ số quan trọng để đánh giá hiệu quả học là tổng reward thu được trong mỗi episode. Việc theo dõi chỉ số này theo thời gian (qua từng episode) cho phép xác định liệu agent đang học tốt hơn hay không, liệu chính sách hành động có đang tiến dần đến tối ưu hay không, và liệu mô hình có hội tụ hay không.

Cách thu thập phần thưởng

Sau mỗi episode, hệ thống sẽ tính tổng reward mà agent nhận được trong toàn bộ hành trình từ trạng thái bắt đầu cho đến khi đến đích (hoặc hết bước). Quy trình này có thể được cài đặt như sau:

```
for episode in range(num_episodes):
            # Reset môi trường và biến theo dõi
            state = env.reset()
            self.recent_states = []  # Reset lịch sử trạng thái
            episode_reward = 0
            step = 0

            # Đếm số trạng thái đã thăm trong episode này
            visited_states = set()
            stuck_counter = 0  # Đếm số bước bị kẹt

            while step < max_steps:
                # Chọn hành động
                action = self.choose_action(state)

                # Thực hiện hành động
                next_state, reward, done, info = env.step(action)

                # Theo dõi trạng thái đã thăm để phát hiện bị kẹt
                state_key = (next_state[0], next_state[1])
                if state_key in visited_states:
                    stuck_counter += 1
                else:
                    visited_states.add(state_key)
                    stuck_counter = 0

                # Nếu agent bị kẹt quá lâu, tăng epsilon tạm thời để thúc đẩy khám phá
                if stuck_counter > 30:  # Ngưỡng thấp hơn để phản ứng nhanh hơn
                    self.epsilon = min(1.0, self.epsilon * 2)
                    # Xóa một phần experience buffer để quên đi các trải nghiệm không tốt
                    if len(self.experience_buffer) > 100:
                        self.experience_buffer = self.experience_buffer[-100:]
                    stuck_counter = 0

                # Học từ trải nghiệm
                self.learn(state, action, reward, next_state, done)
```

Hình 3.4: Code training agent

Trong đó:

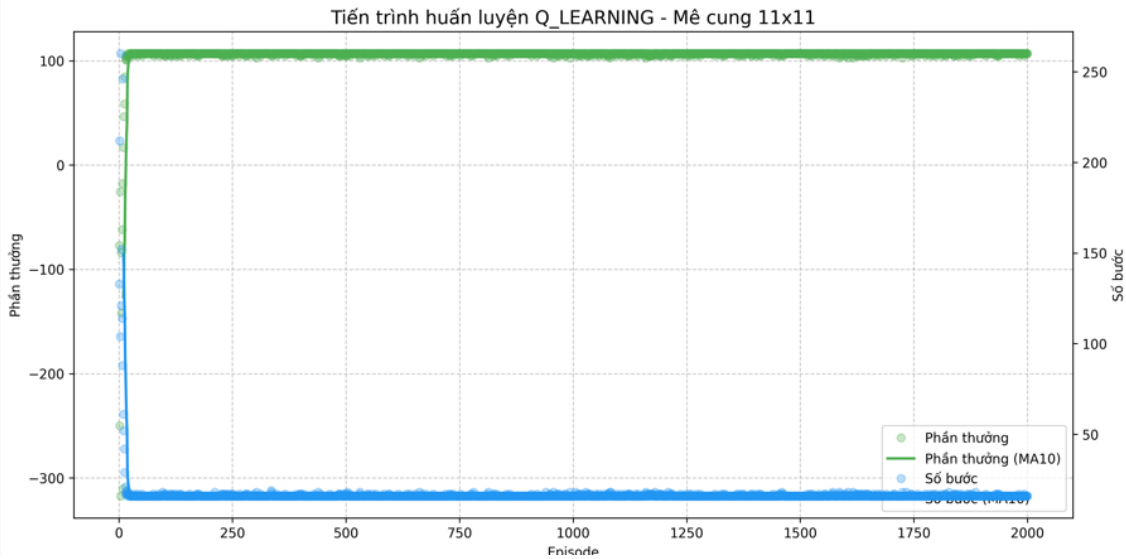
- Episode_rewards : là một list lưu trữ các phần thưởng trong mỗi episode
- Steps_per_episode: là một list lưu trữ số bước mà tác nhân thực hiện trong mỗi episode

3.4. Huấn luyện và biểu diễn trực quan

Trong khi quá trình huấn luyện hoàn tất, việc đánh giá chất lượng tác nhân là bước không thể thiếu để kiểm tra khả năng học được chính sách tối ưu và khả năng tổng quát hóa sang các mê cung khác nhau.

3.4.1 Số bước đến đích, tổng reward

Quá trình huấn luyện với Q_learning với mê cung kích cỡ (small – 11x11)



Hình 3.5: Sự thay đổi phần thưởng theo số bước của Q_Learning

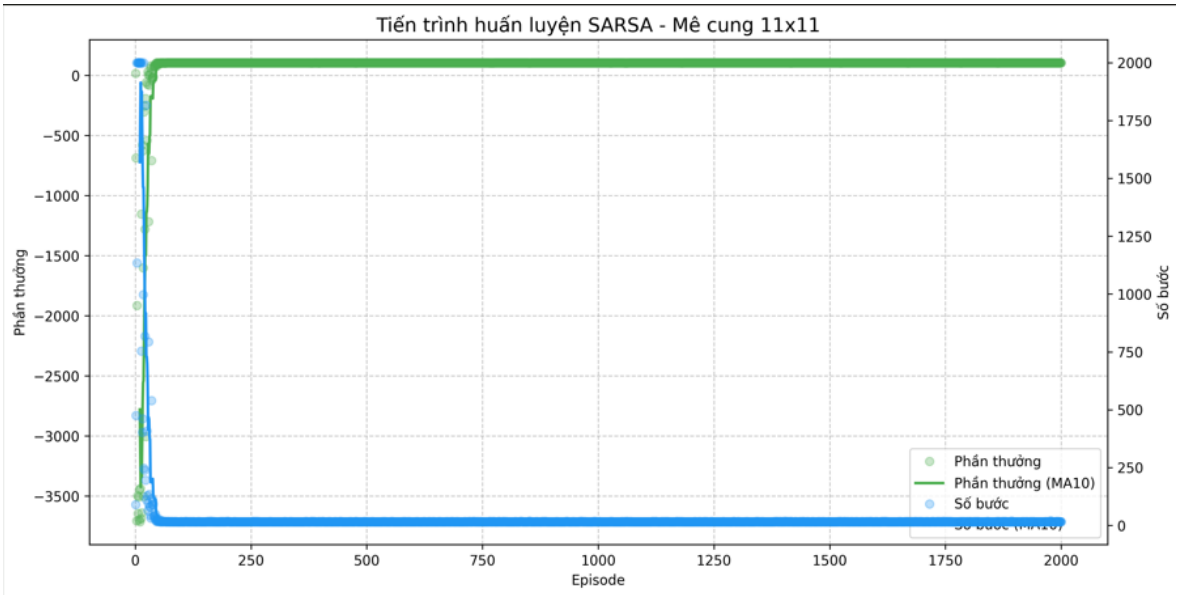
Phân tích:

Biểu đồ này minh hoạt sự thay đổi của hàm phần thưởng và số bước mà tác nhân Q-learning thực hiện để dạt được mục tiêu tìm đường ra khỏi mê cung 11x11 theo từng eposode huẩn luyện.

- Hàm phần thưởng: Ban đầu, tổng phần thưởng có thể thấp do tác nhân khám phá ngẫu nhiên và mắc lỗi. Tuy nhiên, theo thời gian và số lượn episode tăng lên, tác nhân học được cách hành vi tối ưu hơn, dẫn đến sự gia tăng dần của tổng phần thưởng. Điều này cho thấy tác nhân đang học cách tìm đường đi hiệu quả hơn để đến đích và tránh các trạng thái tiêu cực (ví dụ: đi vào tưởng). Sự ổn định của đường công phần thưởng ở các episode cuối cho thấy tác nhân đã hội tụ đến một chính sách ổn định.
- Số bước đến đích**:** Tương tự, số bước mà tác nhân cần để đạt được mục tiêu ban đầu có thể rất lớn. Khi tác nhân học hỏi, số bước này sẽ giảm

dần. Một tác nhân được huấn luyện tốt sẽ tìm được đường đi ngắn nhất hoặc gần nhất đến đích, thể hiện bằng sự giảm và ổn định của đường cong số bước ở các episode sau. Nếu có sự tăng đột biến ở một số episode, có thể là do tác nhân đang khám phá các con đường mới hoặc bị kẹt trong một vòng lặp tạm thời trước khi tìm được cách thoát ra.

Quá trình huấn luyện với SARSA với mê cung kích cỡ (small – 11x11)



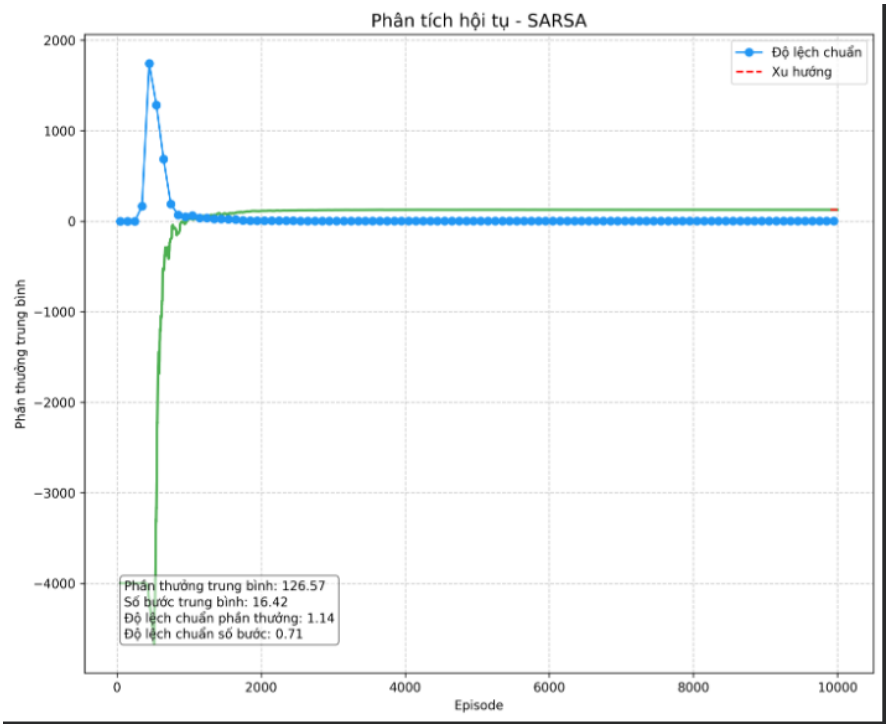
Hình 3.6: Sự thay đổi phần thưởng theo số bước của SARSA

Phân tích:

Biểu đò này thể hiện sự biến đổi của hàm phân thưởng tích lũy theo từng episode trong quá trình huấn luyện tác nhân SARSA. Tương tự như Q-Learning, SARSA cũng là một thuật toán học tăng cường dựa trên giá trị

- Đường cong phần thưởng của SARSA cho thấy sự tăng lên đáng kể qua các episode ban đầu, chứng tỏ tác nhân đang học được các hành vi mang lại phần thưởng.
- Sau một số lượng episode nhất định, đường cong phần thưởng có xu hướng ổn định, cho thấy tác nhân đã hội tụ đến một chính sách gần tối ưu. Mức độ phần thưởng ổn định này phản ánh hiệu quả của thuật toán SARSA trong việc tìm ra đường đi có lợi trong môi trường mê cung.

- Nếu có sự dao động, nó có thể chỉ ra rằng tác nhân vẫn đang tiếp tục khám phá hoặc môi trường có tính ngẫu nhiên.



Hình 3.7: Quá trình hội tụ của Sarsa

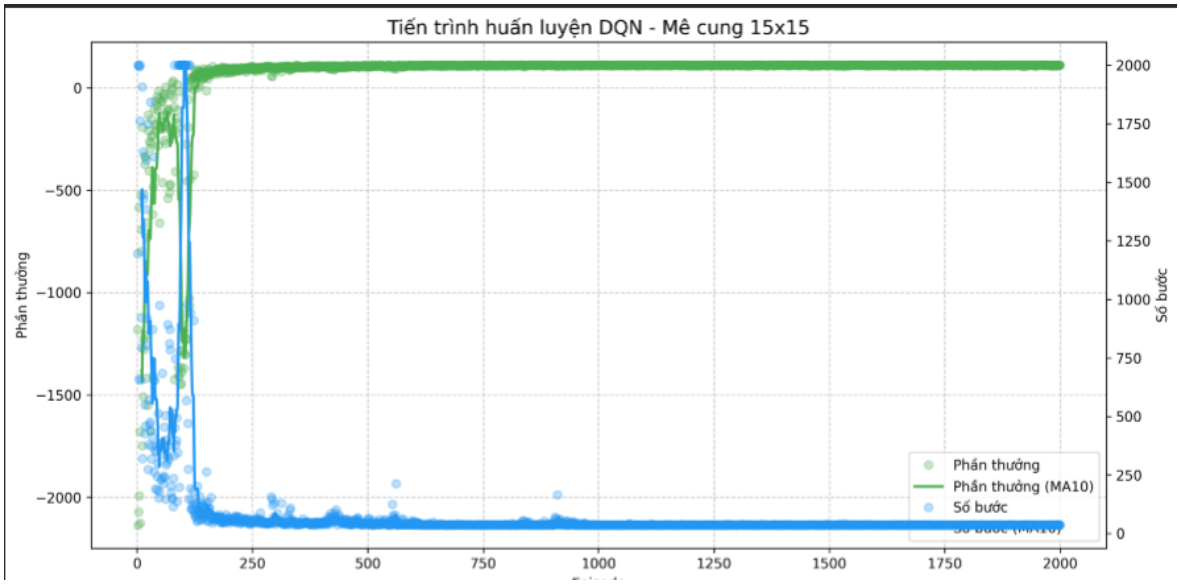
Phân tích:

Biểu đồ này thường minh hoạt sự hội tụ của thuật toán SARSA thông qua sự thay đổi của hàm lỗi (ví dụ lỗi Bellman) hoặc sự ổn định của các giá trị Q.

- Ban đầu, khi tác nhân chưa được huấn luyện, các giá trị Q ước tính có thể rất khác so với giá trị thực tế, dẫn đến lỗi lớn.
- Khi quá trình huấn luyện diễn ra, thuật toán SARSA cập nhật các giá trị Q dựa trên kinh nghiệm. Lỗi sẽ giảm dần qua các episode, cho thấy các giá trị Q đang hội tụ về giá trị thực của hàm Q tối ưu.
- Một đường cong lỗi giảm đều đặn và cuối cùng tiệm cận về 0 hoặc một giá trị nhỏ cho thấy quá trình học đã thành công và tác nhân đã tìm được một chính sách ổn định.

Sự dao động nhỏ ở giai đoạn cuối có thể là do yếu tố ngẫu nhiên trong việc khám phá (epsilon-greedy) hoặc đặc điểm của môi trường.

Quá trình huấn luyện với DQN với mê cung kích cỡ (medium – 15x15)

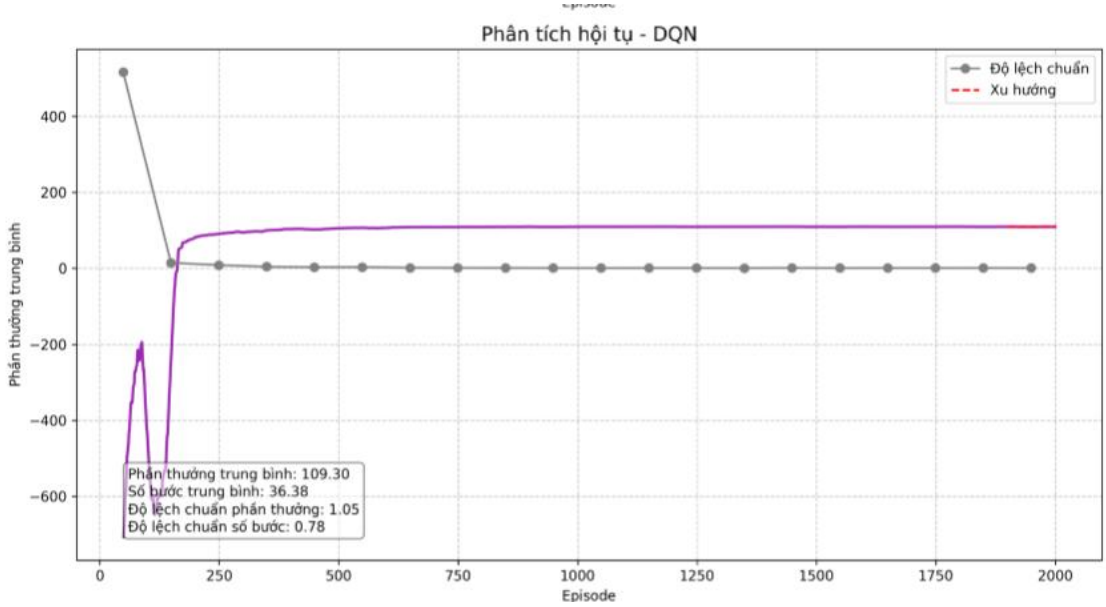


Hình 3.8: Sự thay đổi phần thưởng theo số bước của DQN

Phân tích:

Biểu đồ này mô tả trực quan phần thưởng và số bước di chuyển của tác nhân trong mỗi episode.

- Trong những episode đầu tiên, phần thưởng rất thấp (thậm chí âm sâu), số bước đi lớn, điều này phản ánh việc tác nhân còn chưa tối ưu hóa chính sách di chuyển.
- Sau khoảng 200 episodes, phần thưởng tăng mạnh và ổn định, đồng thời số bước giảm rõ rệt. Điều này thể hiện tác nhân đã học được cách đi đến đích một cách hiệu quả, tránh các trạng thái không mong muốn.
- Đường trung bình động (MA10) cho phần thưởng và số bước đều cho thấy xu hướng hội tụ rõ ràng.

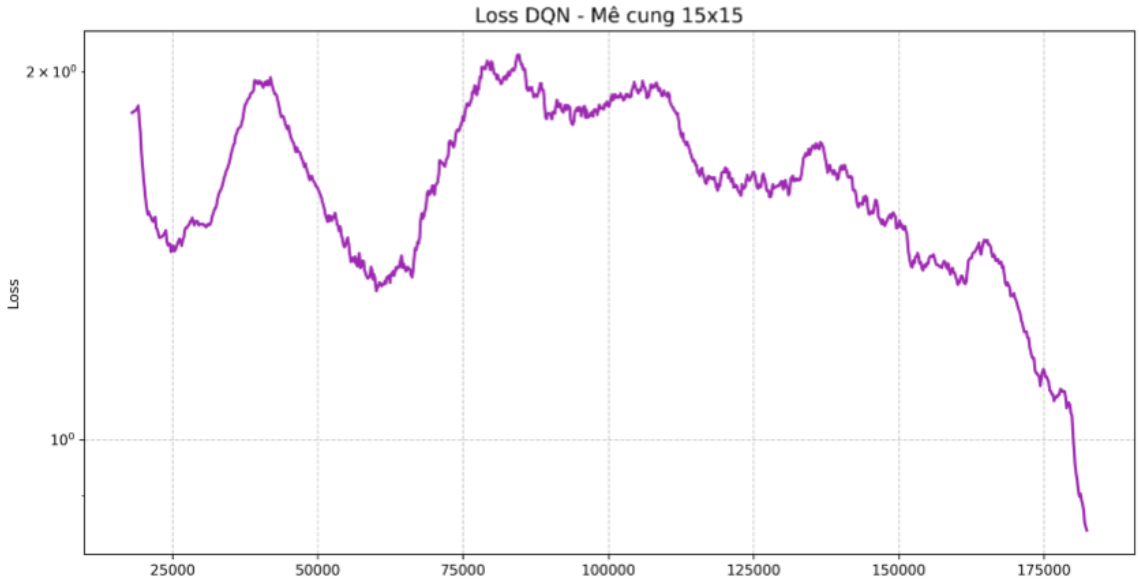


Hình 3.9: Quá trình hội của DQN

Phân tích:

Biểu đồ này mô tả quá trình hội tụ của DQN dựa trên phần thưởng trung bình và độ lệch chuẩn theo từng episode.

- Ban đầu, tác nhân chưa học được chiến lược tốt nên phần thưởng trung bình rất thấp và có thể âm.
- Tuy nhiên, sau khoảng 100-250 episodes, phần thưởng trung bình tăng nhanh và dần ổn định, trong khi độ lệch chuẩn giảm mạnh. Đây là giai đoạn hội tụ, cho thấy tác nhân đã tìm được chính sách hiệu quả.
- Phần thưởng trung bình ổn định quanh giá trị ~109 và độ lệch chuẩn nhỏ chứng minh chính sách không chỉ hiệu quả mà còn ổn định qua nhiều lần thử.
- Đây là minh chứng rõ ràng cho sự thành công trong việc huấn luyện DQN trên môi trường mê cung.



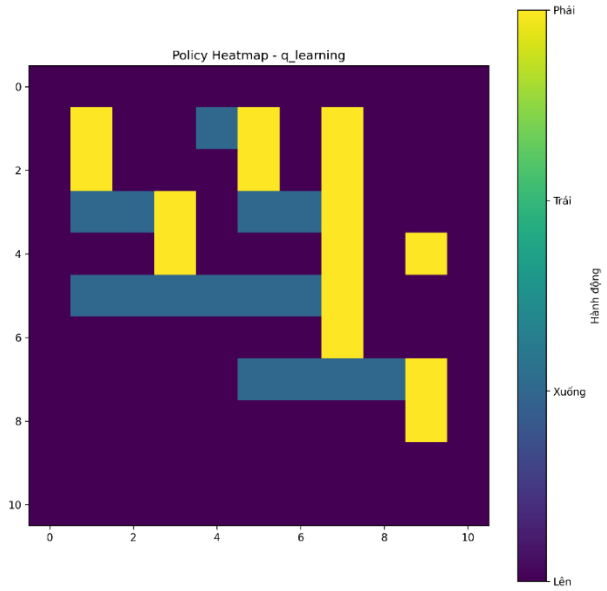
Hình 3.10: Quá trình thay đổi hàm mất mát trong huấn luyện DQN

Phân tích:

Biểu đồ này thể hiện sự thay đổi của hàm mất mát trong quá trình huấn luyện DQN.

- Giai đoạn đầu, khi tác nhân chưa được huấn luyện tốt, các giá trị Q ước tính sai lệch đáng kể so với giá trị tối ưu, dẫn đến hàm mất mát cao và dao động mạnh.
- Khi huấn luyện tiếp tục, loss có xu hướng giảm dần, chứng tỏ DQN đang học được các giá trị Q chính xác hơn thông qua kinh nghiệm.
- Đặc biệt, giai đoạn cuối của biểu đồ cho thấy loss giảm mạnh và tiệm cận về giá trị nhỏ, là dấu hiệu rõ rệt của sự hội tụ. Điều này cho thấy tác nhân đã tìm được một chính sách ổn định và đáng tin cậy.
- Một số dao động nhỏ trước khi hội tụ hoàn toàn là điều thường gặp, có thể xuất phát từ chiến lược khám phá (ví dụ: epsilon-greedy) hoặc tính ngẫu nhiên trong môi trường.

3.4.2 Heatmap chính sách và hành vi



Hình 3.11: Policy Heatmap – Q_learning

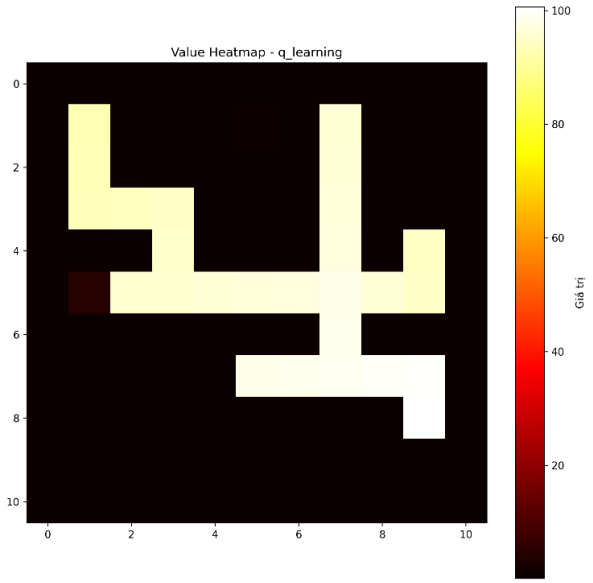
Mô tả: Hình ảnh này là một heatmap chính sách (Policy Heatmap) được tạo ra sau khi huấn luyện tác nhân Q-learning trên mê cung. Mỗi ô trong lưới đại diện cho một trạng thái trong mê cung, và màu sắc của ô biểu thị hành động tối ưu mà tác nhân nên thực hiện tại trạng thái đó theo chính sách đã học.

- Thang màu bên phải cho thấy các hành động: "Phải" (Vàng), "Trái"(xanh là cây), "Xuống" (xanh lam), "lên"(tím)
- Các ô màu vàng thường chỉ ra hành độn "Phải", xanh lam là 'xuống', vv.

Phân tích:

- Các ô màu trên heatmap cho thấy chiến lược hành động của tác nhân tại từng vị trí trong mê cung. Ví dụ, nếu một ô có màu vàng, điều đó có nghĩa là tác nhân đã học được rằng hành động tốt nhất tại vị trí đó là di chuển sang phải để đạt được mục tiêu hoặc tránh chướng ngại vật một cách hiệu quả.
- Các vùng có màu sắc đồng nhất hoặc thay đổi theo một quy luật nhất định thường cho thấy một đường đi đã được học.

- Sự phân bố của các màu sắc trên bản đồ cho thấy tác nhân Q-learning đã học được một chính sách rõ ràng để di chuyển trong mê cung, tránh các vùng không thể đi được và hướng tới mục tiêu.



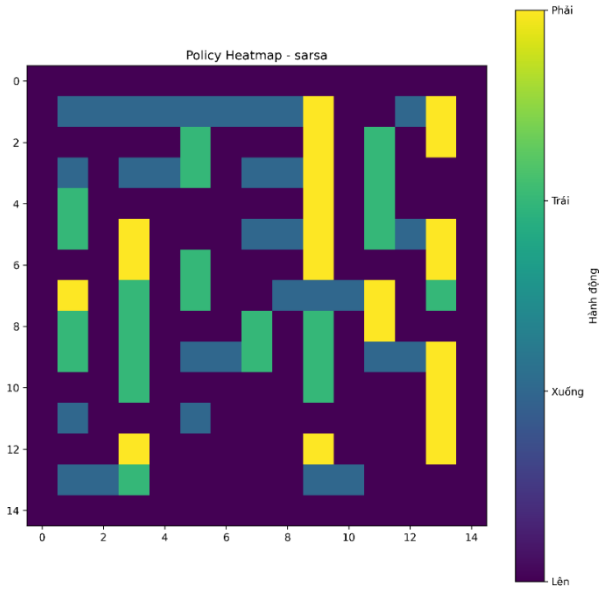
Hình 3.12: Value Heatmap – Q_learning

Mô tả: Hình ảnh này là một heatmap giá trị (Value Heatmap) được tạo ra sau khi huấn luyện tác nhân Q-learning trên mê cung. Mỗi ô trong lưới đại diện cho một trạng thái, và màu sắc của ô biểu thị giá trị của trạng thái đó. Giá trị này phản ánh tổng phần thưởng dự kiến mà tác nhân sẽ nhận được nếu bắt đầu từ trạng thái đó và tuân theo chính sách đã học.

- Thang màu bên phải cho thấy giá trị từ thấp (đen/đỏ sẫm) đến cao (trắng/vàng sáng)
- Các ô có màu trắng hoặc vàng sáng thể hiện các trạng thái có giá trị cao, thường là các trạng thái gần đích hoặc nằm trên đường đi tối ưu.
- Các ô màu đen hoặc đỏ sẫm thể hiện các trạng thái có giá trị thấp, có thể là các trạng thái xa đích, gần chướng ngại vật hoặc không thể đến được.

Phân tích:

- Heatmap này trực quan hóa hàm giá trị V(s) hoặc Q(s,a) đã được học. Các ô có giá trị cao nhất (sáng nhất) thường là vị trí đích hoặc các ô ngay sát đích, nơi phần thưởng cao nhất có thể đạt được.
- Khi di chuyển ra xa đích, giá trị của các ô giảm dần (màu sẫm hơn), thể hiện chi phí hoặc số bước cần thiết để đến đích tăng lên.
- Các vùng màu đen hoặc rất sẫm thường là các chướng ngại vật hoặc các vị trí mà tác nhân không thể đạt được hoặc rất khó để đạt được phần thưởng cao từ đó.
- Sự chuyển tiếp màu sắc từ tối đến sáng tạo thành một "con đường" hoặc "dòng chảy" của giá trị, cho thấy đường đi tối ưu mà tác nhân đã học được để từ bất kỳ trạng thái nào đi đến trạng thái đích.

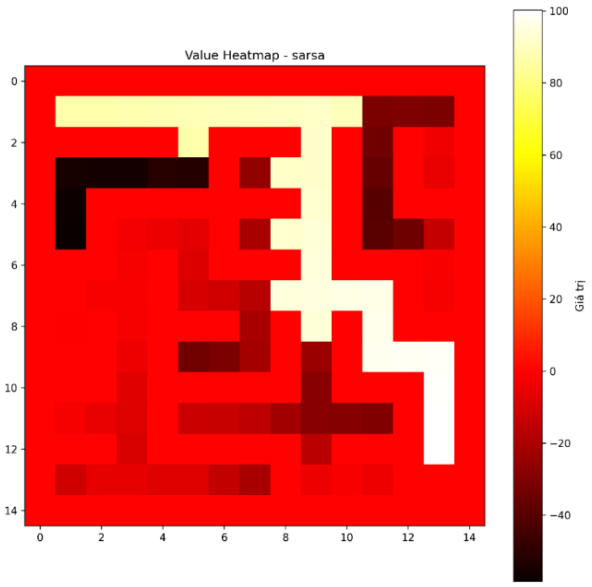


Hình 3.13: Policy Heatmap Sarsa

Mô tả: Tương tự như Policy Heatmap của Q-learning, hình ảnh này là một heatmap chính sách cho tác nhân SARSA. Nó thể hiện hành động tối ưu tại mỗi trạng thái trong mê cung sau quá trình huấn luyện SARSA. Thang màu và ý nghĩa của từng màu sắc cho hành động là giống nhau.

Phân tích:

- So với Policy Heatmap của Q-learning, heatmap này của SARSA có thể cho thấy sự khác biệt về chính sách do sự khác biệt trong cơ chế cập nhật (on-policy của SARSA so với off-policy của Q-learning).
- Mặc dù cả hai đều hướng đến việc tìm ra chính sách tối ưu, SARSA có thể học một chính sách "an toàn" hơn trong môi trường có chướng ngại vật hoặc phần thưởng tiêu cực, vì nó học dựa trên các hành động thực sự được thực hiện.
- Các vùng màu sắc cụ thể cho thấy hành động ưu tiên tại từng vị trí. Ví dụ, có thể thấy các vùng màu vàng (phải) hoặc xanh lam (xuống) chiếm ưu thế ở những khu vực nhất định, chỉ ra hướng di chuyển chung của tác nhân.
- Sự phân bố của các hành động cho thấy khả năng của SARSA trong việc tìm ra một đường đi hiệu quả.



Hình 3.14: Value heatmap Sarsa

Mô tả: Đây là heatmap giá trị cho tác nhân SARSA, minh họa giá trị của mỗi trạng thái trong mê cung sau khi huấn luyện. thang màu đỏ/vàng sáng thể hiện giá trị cao và màu đen/đỏ sẫm thể hiện giá trị thấp.

Phân tích:

- Giống như Value Heatmap của Q-learning, bản đồ này cho thấy các trạng thái có giá trị cao (màu sáng) là các trạng thái gần đích hoặc nằm trên đường đi tối ưu.
- Điểm khác biệt chính giữa Value Heatmap của Q-learning và SARSA có thể nằm ở mức độ "khám phá" và "tối ưu". Do SARSA là thuật toán on-policy, nó có thể có xu hướng đánh giá cao các trạng thái trên con đường mà tác nhân thực sự đã đi, ngay cả khi nó không phải là con đường tối ưu tuyệt đối.
- Các vùng màu tối (ví dụ: các ô màu đen hoặc đỏ sẫm) đại diện cho các chướng ngại vật hoặc các vị trí có giá trị thấp, mà tác nhân đã học được để tránh hoặc không nên đến từ đó.
- Con đường" màu sáng xuyên qua mê cung một lần nữa cho thấy đường đi mà tác nhân SARSA đã học được để tối đa hóa phần thưởng.

3.5. Kết quả huấn luyện và phân tích

Qua quá trình huấn luyện với môi trường mê cung, cả hai thuật toán học tăng cường Q-learning SARSA, DQN đều đã chứng minh khả năng tìm ra đường đi hiệu quả từ điểm xuất phát đến đích.

- *Q-learning* đã hội tụ để tìm ra một chính sách tối ưu, được thể hiện rõ qua sự gia tăng của phần thưởng tích lũy và sự giảm của số bước theo thời gian. Heatmap giá trị và chính sách của Q-learning cho thấy một sự hiểu biết sâu sắc về cấu trúc mê cung và các hành động tối ưu để đạt được mục tiêu.
- *SARSA* cũng thể hiện quá trình học tập hiệu quả, với đường cong phần thưởng tăng dần và sự hội tụ của hàm giá trị. Heatmap của

SARSA cũng minh họa một chính sách rõ ràng, tuy nhiên, do đặc tính on-policy, chính sách này có thể hơi khác so với Q-learning, có thể ưu tiên các con đường "an toàn" hơn trong quá trình khám phá.

- DQN đã thể hiện khả năng học tập vượt trội thông qua mạng nơ-ron sâu, giúp tác nhân không chỉ hội tụ về chính sách tối ưu mà còn thích nghi tốt với môi trường mê cung có không gian trạng thái lớn. Biểu đồ hàm mất mát (loss) cho thấy quá trình huấn luyện ổn định, với lỗi giảm dần và hội tụ về giá trị nhỏ — minh chứng cho việc tối ưu hóa thành công hàm Q xấp xỉ. Song song đó, biểu đồ phần thưởng trung bình và số bước theo từng tập (episode) phản ánh rõ sự cải thiện về hiệu suất: phần thưởng tăng đều và số bước giảm mạnh, cho thấy tác nhân đã học được chiến lược tối ưu hóa đường đi từ điểm xuất phát đến đích.

Như vậy, cả ba thuật toán Q-learning, SARSA và DQN đều thể hiện khả năng giải quyết bài toán mê cung hiệu quả, mỗi thuật toán mang đặc trưng riêng trong cách cập nhật và hội tụ chính sách. DQN nổi bật với khả năng tổng quát hóa mạnh mẽ nhờ mạng học sâu, đặc biệt phù hợp với môi trường phức tạp có không gian trạng thái lớn như mê cung 15x15.

CHƯƠNG 4: TRIỂN KHAI GIAO DIỆN VÀ ĐÁNH GIÁ HỆ THỐNG

4.1. Thiết kế giao diện người dùng

Giao diện người dùng (User Interface – UI) của hệ thống giải mê cung bằng học tăng cường được thiết kế nhằm mục đích cung cấp một công cụ trực quan và dễ sử dụng cho người dùng, cho phép họ tương tác với các thuật toán, cấu hình mê cung và quan sát quá trình giải mê cung một cách rõ ràng. Thiết kế tập trung vào sự đơn giản, rõ ràng và hiệu quả, đảm bảo người dùng có thể dễ dàng tiếp cận và kiểm soát các chức năng chính của hệ thống.

Cấu trúc giao diện được phân chia thành các khu vực chức năng riêng biệt: khu vực hiển thị mê cung, khu vực cấu hình mê cung, khu vực lựa chọn thuật toán giải, và khu vực tùy chọn hiển thị. Sự phân chia này giúp tối ưu hóa trải nghiệm người dùng bằng cách nhóm các chức năng liên quan lại với nhau, giảm thiểu sự phức tạp và tăng cường khả năng điều hướng. Màu sắc và font chữ được lựa chọn để đảm bảo tính thẩm mỹ và khả năng đọc, đồng thời tạo điểm nhấn cho các yếu tố tương tác quan trọng.

4.2. Yêu cầu và chức năng của giao diện

Giao diện người dùng được phát triển dựa trên các yêu cầu chức năng và phi chức năng sau:

Yêu cầu chức năng:

- Tạo mê cung**:** Cho phép người dùng tạo ra các mê cung mới với các kích thước và thuật toán sinh mê cung khác nhau (ví dụ: Prim's Algorithm).
- Lựa chọn thuật toán giải**:** Cung cấp tùy chọn cho người dùng để chọn một trong các thuật toán học tăng cường đã được huấn luyện (ví dụ: Q-Learning, SARSA) để giải mê cung.
- Hiển thị quá trình giải**:** Trực quan hóa động quá trình tác nhân di chuyển và giải mê cung, bao gồm đường đi đã tìm thấy.

- Điều khiển tốc độ hoạt ảnh**:** Cho phép người dùng điều chỉnh tốc độ hiển thị của quá trình giải mê cung để dễ dàng quan sát hoặc tăng tốc độ.
- Hiển thị đường đi**:** Tùy chọn bật/tắt hiển thị đường đi mà tác nhân đã tìm được sau khi giải mê cung.
- Đặt điểm bắt đầu/kết thúc (nếu có)**:** Khả năng tùy chỉnh vị trí bắt đầu và kết thúc của tác nhân trong mê cung. *(Nếu hệ thống của bạn có chức năng này, bạn có thể thêm vào).*

Yêu cầu phi chức năng:

- Tính trực quan**:** Giao diện phải dễ hiểu và dễ sử dụng, ngay cả với người dùng không có chuyên môn sâu về học tăng cường.
- Tính ổn định**:** Hệ thống phải hoạt động ổn định, không gặp lỗi hay treo trong quá trình sử dụng.
- Tính đáp ứng**:** Giao diện cần có khả năng phản hồi nhanh chóng đối với các tương tác của người dùng.
- Tính thẩm mỹ**:** Bố cục và màu sắc cần hài hòa, dễ nhìn.

4.3. Mô tả chi tiết giao diện và các thành phần

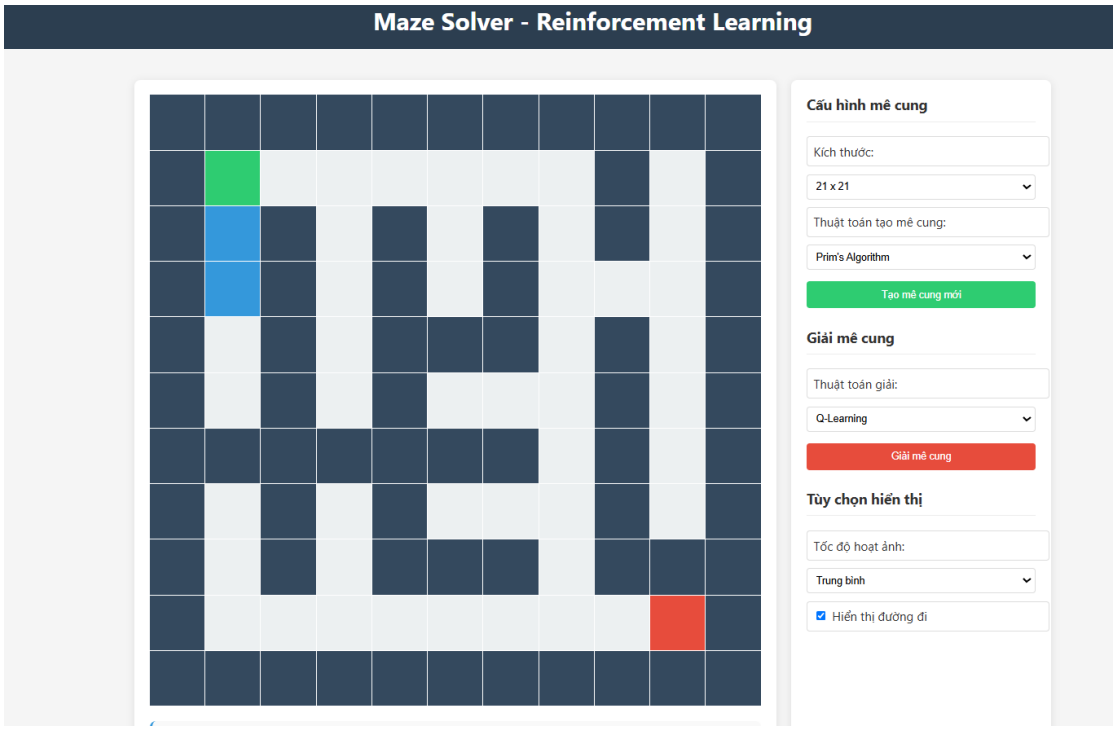
Giao diện người dùng được thiết kế với bố cục chính bao gồm ba phần: Khu vực hiển thị mê cung, và hai khu vực cấu hình và điều khiển ở phía bên phải.

1. Khu vực hiển thị mê cung:
    - Nằm ở trung tâm màn hình, đây là nơi hiển thị ma trận mê cung.
    - Mê cung được biểu diễn dưới dạng lưới các ô vuông.
    - Các ô màu xanh đậm đại diện cho tường (chướng ngại vật).
    - Các ô màu trắng đại diện cho đường đi có thể di chuyển được.
    - Ô màu xanh lá cây đậm thường là điểm bắt đầu (Start Point) của tác nhân.
    - Ô màu đỏ đậm thường là điểm đích (Goal Point) của tác nhân.

- Trong quá trình giải, đường đi của tác nhân có thể được hiển thị bằng một màu sắc khác hoặc hiển thị động quá trình di chuyển của tác nhân.

2. Khu vực Cấu hình mê cung:
    - Kích thước**:** Cho phép người dùng chọn kích thước của mê cung (ví dụ: 21x21, 11x11, v.v.) thông qua một dropdown menu.
    - Thuật toán tạo mê cung**:** Cung cấp các lựa chọn thuật toán để sinh mê cung (ví dụ: Prim's Algorithm, Kruskal's Algorithm). Đây là bước tiền xử lý để tạo ra môi trường cho thuật toán học tăng cường.
    - Tạo mê cung mới**:** Một nút bấm kích hoạt việc sinh mê cung mới dựa trên các thông số đã chọn.
3. Khu vực "Giải mê cung":
    - Thuật toán giải**:** Một dropdown menu cho phép người dùng lựa chọn thuật toán học tăng cường để giải mê cung (ví dụ: Q-Learning, SARSA, DQN).
    - Giải mê cung**:** Nút bấm kích hoạt quá trình giải mê cung bằng thuật toán đã chọn.
4. Khu vực "Tùy chọn hiển thị":
    - Tốc độ hoạt ảnh**:** Một dropdown menu điều chỉnh tốc độ hiển thị của quá trình giải mê cung (ví dụ: Nhanh, Trung bình, Chậm).
    - Hiển thị đường đi**:** Một hộp kiểm (checkbox) cho phép người dùng bật/tắt hiển thị đường đi mà tác nhân đã tìm thấy sau khi quá trình giải kết thúc.

4.4. Giao diện thiết kế



Hình 4.1: Giao diện người dùng của hệ thống

Chú thích: Hìn 4.1 trình bày tổng quan về giao diện người dùng của hệ thống. Bên trái là khu vực hiển thị mê cung, thể hiện rõ các bức tường (màu xanh đậm), đường đi (màu trắng), điểm bắt đầu (ô xanh lá cây) và điểm kết thúc (ô đỏ). Bên phải là các bảng điều khiển cho phép người dùng cấu hình mê cung, chọn thuật toán giải và điều chỉnh các tùy chọn hiển thị. Cụ thể, trong ảnh chụp này, mê cung có kích thước 21x21 được tạo bằng Prim's Algorithm, và thuật toán giải Q-Learning đã được chọn. Tốc độ hoạt ảnh đang được đặt là "Trung bình" và tùy chọn "Hiển thị đường đi" đang được bật.

4.5. Tích hợp mô hình đã huấn luyện vào giao diện

Quá trình tích hợp mô hình học tăng cường đã được huấn luyện vào giao diện người dùng được thực hiện theo các bước sau:

1. *Lưu trữ mô hình*: Sau khi hoàn tất quá trình huấn luyện (đạt được sự hội tụ của hàm giá trị và chính sách), các trọng số hoặc bảng Q-value của mô hình (Q-table) được lưu trữ lại. Trong trường hợp của Q-learning và

SARSA, đây là bảng Q(s, a) chứa các giá trị dự kiến của việc thực hiện hành động 'a' từ trạng thái 's'.

2. *Tải mô hình*: Khi người dùng lựa chọn một thuật toán giải (ví dụ: Q-Learning), giao diện sẽ tải bảng Q-value tương ứng đã được lưu trữ trước đó.
3. *Áp dụng chính sách*: Khi nút "Giải mê cung" được nhấn, hệ thống sẽ sử dụng bảng Q-value đã tải để xác định hành động tối ưu tại mỗi trạng thái. Từ điểm bắt đầu, tác nhân sẽ tra cứu giá trị Q cho các hành động khả thi và chọn hành động mang lại Q-value cao nhất (hoặc theo chính sách ϵ-greedy nếu muốn mô phỏng quá trình khám phá).
4. *Trực quan hóa*: Các bước di chuyển của tác nhân được cập nhật trên giao diện, và đường đi được vẽ lại theo thời gian thực hoặc sau khi tìm thấy đích. Điều này tạo ra một hoạt ảnh trực quan về quá trình giải mê cung. Tốc độ hoạt ảnh được điều khiển bởi tham số "Tốc độ hoạt ảnh" mà người dùng đã chọn.
5. *Cập nhật trạng thái*: Khi tác nhân di chuyển từ trạng thái hiện tại sang trạng thái mới, giao diện cập nhật vị trí của tác nhân và tiếp tục xác định hành động tối ưu cho trạng thái mới cho đến khi đạt được đích.

4.6. Đánh giá hiệu quả tổng thể

Việc đánh giá hiệu quả tổng thể của hệ thống được thực hiện trên hai khía cạnh chính: hiệu suất giải bài toán của các thuật toán học tăng cường và trải nghiệm người dùng thông qua giao diện trực quan.

Về mặt hiệu suất thuật toán, các mô hình Q-learning, SARSA và DQN đã được triển khai và huấn luyện trong môi trường mê cung được thiết kế tùy biến. Kết quả cho thấy các agent có khả năng học và tìm ra đường đi tối ưu trong các môi trường quen thuộc, với đường đi hội tụ sau một số lượng episode nhất định. Tuy nhiên, một hạn chế rõ rệt được ghi nhận là độ tổng quát hóa (generalization) của các mô hình còn thấp. Khi thay đổi cấu trúc mê cung , agent thường không thể áp dụng chính sách đã học trước đó và phải học lại gần như từ đầu. Điều

này phản ánh điểm yếu phổ biến của các thuật toán RL truyền thống – khả năng tái sử dụng kiến thức trong môi trường mới còn hạn chế.

Đối với thuật toán DQN, mặc dù có tiềm năng xử lý các không gian trạng thái lớn và trừu tượng hơn nhờ vào khả năng biểu diễn của mạng nơ-ron, nhưng việc tối ưu kiến trúc mạng, lựa chọn siêu tham số và tránh overfitting đòi hỏi tài nguyên tính toán lớn và quy trình thử nghiệm phức tạp. Trong khuôn khổ đồ án, do hạn chế về thời gian và tài nguyên phần cứng, việc huấn luyện DQN mới chỉ đạt mức hội tụ ổn định trong các môi trường nhỏ và vừa, chưa thể mở rộng sang các mê cung phức tạp hơn.

Về mặt trải nghiệm người dùng, giao diện trực quan được xây dựng đã hỗ trợ hiệu quả trong việc theo dõi quá trình học của agent. Việc mô phỏng đường đi, biểu diễn chính sách học được và hiển thị phần thưởng theo thời gian đã giúp người dùng dễ dàng quan sát, đánh giá và hiểu được cơ chế hoạt động của từng thuật toán. Giao diện cũng đóng vai trò quan trọng trong việc kiểm thử và so sánh hiệu quả giữa các thuật toán RL.

Tổng thể, hệ thống đã hoàn thành được mục tiêu ban đầu là triển khai và đánh giá các thuật toán học tăng cường trong môi trường mê cung, đồng thời cung cấp một nền tảng thực nghiệm dễ tiếp cận và trực quan. Dù còn một số hạn chế, kết quả đạt được là cơ sở vững chắc để tiếp tục mở rộng và cải tiến trong tương lai.

4.7. Kết luận và hướng phát triển

Đồ án này đã mang đến cho tôi một trải nghiệm toàn diện về việc ứng dụng học tăng cường vào một bài toán cụ thể – giải mê cung. Qua việc thiết kế môi trường, lựa chọn và triển khai các thuật toán RL như Q-learning, SARSA và DQN, tôi đã học được cách xây dựng một hệ thống học từ tương tác, đánh giá hiệu quả học thông qua phần thưởng, và đặc biệt là đối mặt với thách thức thực tế về tổng quát hóa mô hình.

Dù mô hình hoạt động tốt trong các môi trường đã huấn luyện, nhưng khả năng áp dụng cho các môi trường mới chưa thực sự hiệu quả, từ đó đặt ra yêu

cầu nghiên cứu các kỹ thuật RL hiện đại hơn như Transfer Learning trong RL, Meta-RL, hoặc Model-based RL – nơi agent có thể tận dụng kinh nghiệm đã học để thích nghi nhanh chóng với môi trường chưa từng gặp.

Về phần giao diện, hệ thống hiện tại mới chỉ dừng lại ở mức hiển thị trực quan cơ bản. Trong tương lai, có thể tích hợp thêm các tính năng như cho phép người dùng tự thiết kế mê cung, thiết lập các bài toán RL khác nhau (ví dụ: nhiều đích đến, mê cung thay đổi theo thời gian), và theo dõi biểu đồ động trong quá trình huấn luyện.

Ngoài ra, mở rộng nghiên cứu sang môi trường ba chiều, học tăng cường có giám sát, hoặc thậm chí là sử dụng RL kết hợp với các mô hình ngôn ngữ để ra quyết định sẽ là những hướng đi tiềm năng. Đây không chỉ là bước phát triển của đồ án, mà còn là cầu nối để tiến xa hơn trong các ứng dụng thực tế của trí tuệ nhân tạo hiện đại.

Thông qua đồ án, tôi đã củng cố được nền tảng lý thuyết, tích lũy kinh nghiệm thực tiễn, và đặc biệt là rèn luyện được tư duy giải quyết vấn đề một cách hệ thống. Đây sẽ là hành trang quý giá để tôi tiếp tục học hỏi và phát triển trong lĩnh vực học tăng cường và trí tuệ nhân tạo trong tương lai.

KẾT LUẬN CHUNG

Trong suốt quá trình thực hiện đồ án, việc nghiên cứu và ứng dụng các phương pháp học tăng cường (ReinforcementLearning - RL) vào bài toán giải mê cung không chỉ là một thử thách kỹ thuật mà còn là một trải nghiệm học thuật đầy thú vị và bổ ích. Qua từng giai đoạn từ tìm hiểu lý thuyết nền tảng đến triển khai thực nghiệm, tôi đã dần hình dung rõ hơn về cách mà các agent có thể học hỏi và ra quyết định tối ưu thông qua việc tương tác với môi trường và điều chỉnh hành vi dựa trên phần thưởng nhận được.

Việc áp dụng ba thuật toán tiêu biểu là Q-learning, SARSA và Deep Q-Network (DQN) giúp tôi thấy rõ sự khác biệt về cách thức cập nhật giá trị, mức độ hội tụ cũng như hiệu quả tìm đường trong các tình huống khác nhau. Đặc biệt, việc so sánh giữa Q-learning và SARSA trên cùng một cấu trúc mê cung đã cho thấy vai trò của chính sách học (on-policy và off-policy) ảnh hưởng như thế nào đến hiệu quả huấn luyện. Với DQN, dù yêu cầu tài nguyên tính toán cao hơn, nhưng nó mở ra khả năng xử lý môi trường có không gian trạng thái lớn hơn thông qua mạng nơ-ron sâu – một điểm nổi bật cho hướng nghiên cứu tương lai.

Bên cạnh phần lý thuyết, quá trình thiết kế môi trường mê cung – bao gồm cả Perfect Maze (không có vòng lặp) và Imperfect Maze (có nhiều đường đi sai) – là một bước đi quan trọng giúp tôi hiểu rõ hơn về vai trò của môi trường trong việc học của agent. Tôi cũng đã thử nghiệm với nhiều cấu hình khác nhau về kích thước, độ phức tạp và vị trí đích đến nhằm đánh giá tính tổng quát của từng thuật toán.

Một trong những phần mà tôi tâm đắc nhất chính là quá trình trực quan hóa kết quả học: từ việc vẽ biểu đồ reward qua từng episode để theo dõi sự tiến bộ của agent, cho đến việc xây dựng heatmap thể hiện chính sách hành động học được. Các công cụ trực quan không chỉ giúp tôi phát hiện và điều chỉnh kịp thời các vấn đề trong quá trình huấn luyện, mà còn giúp việc trình bày kết quả trở nên thuyết phục hơn với người xtôi. Đặc biệt, giao diện trực quan mô phỏng quá trình di chuyển của agent trong mê cung là một yếu tố giúp chuyển hóa lý thuyết khô khan thành trải nghiệm sinh động, dễ tiếp cận.

Tuy vẫn còn nhiều giới hạn về mặt thời gian và tài nguyên, đặc biệt khi triển khai các mô hình học sâu như DQN, nhưng đồ án này đã mang đến cho tôi một cái nhìn toàn diện và sâu sắc về học tăng cường. Tôi nhận thấy tiềm năng rất lớn của RL không chỉ trong các bài toán điều hướng như mê cung, mà

còn trong nhiều ứng dụng thực tế như robot tự hành, trò chơi, hệ thống gợi ý, hay huấn luyện chiến lược.

Qua đồ án, tôi không chỉ tích lũy thêm kiến thức chuyên môn mà còn học được cách tư duy theo hướng hệ thống: từ phân tích bài toán, thiết kế môi trường, chọn thuật toán phù hợp, đến đánh giá kết quả một cách khách quan. Đây sẽ là nền tảng quan trọng cho những nghiên cứu và ứng dụng chuyên sâu hơn của tôi trong lĩnh vực trí tuệ nhân tạo nói chung và học tăng cường nói riêng.

Tôi xin chân thành cảm ơn thầy Nguyễn Mạnh Cường đã luôn đồng hành, định hướng và hỗ trợ tôi trong suốt quá trình thực hiện đồ án. Tôi cũng xin gửi lời cảm ơn đến các thầy cô trong khoa đã truyền cảm hứng và tạo điều kiện để tôi có cơ hội thực hiện một đề tài mang tính thực tiễn cao và đầy tiềm năng như thế này.

Tôi xin trân trọng cảm ơn !

Nguyễn Ngọc Duy

TÀI LIỆU THAM KHẢO

[1] WHAT is AI? - AI Technology [Truy cập: 07/05/2025]
[2] S. Cole & E. Kavlakoglu, IBM - Artificial Intelligence [Truy cập: 07/05/2025]
[3] B. J. Copeland, Britannica - Artificial Intelligence [Truy cập: 07/05/2025]
[4] R. Karjian, SearchEnterpriseAI - AI Timeline [Truy cập: 07/05/2025]
[5] C. Eppright, Oracle - NLP Overview [Truy cập: 07/05/2025]
[6] Telefonica - Robotics vs AI [Truy cập: 07/05/2025]
[7] A. Zavgorodniy, Helpware - AI in Business [Truy cập: 07/05/2025]
[8] M. Kunwar, HashStudioz - AI in Transportation [Truy cập: 07/05/2025]
[9] Deep RL for Maze Solving [Truy cập: 07/05/2025]
[10] GeeksforGeeks - Agent-Environment Interface [Truy cập: 10/05/2025]
[11] P. V. R. Reddy et al., Pathfinding Intelligence, IJRP, 2024
[12] CalPoly - Deep RL for Maze Navigation [Truy cập: 07/05/2025]
[13] D. Gleason & M. Jenkin, Maze Navigation using RL (PDF) [Truy cập: 10/05/2025]
[14] OpenAI Spinning Up - RL Intro [Truy cập: 07/05/2025]
[15] Wikipedia - A* Algorithm
[16] GeeksforGeeks - A* Search
[17] Wikipedia - BFS
[18] GeeksforGeeks - BFS
[19] Edureka - BFS Explained
[20] Wikipedia - Hill Climbing
[21] GeeksforGeeks - Hill Climbing
[22] GeeksforGeeks - ML Types [Truy cập: 07/05/2025]
[23] Wikipedia - Reinforcement Learning [Truy cập: 07/05/2025]
[24] Machine Learning Cơ bản [Truy cập: 07/05/2025]
[25] Neptune.ai - MDP in RL [Truy cập: 07/05/2025]
[26] V. Mnih et al., Human-level Control through Deep Reinforcement, Nature, 2015
[27] StackExchange - DQN Discussion [Truy cập: 07/05/2025]
[28] Stanford - CS106B Maze Assignment [Truy cập: 07/05/2025]
[29] V. Suryani et al., Room Cleaning Robot using A Algorithm*, Telkom University, 2023
[30] Wikipedia - Maze Generation Algorithm [Truy cập: 07/05/2025]
[31] Gymnasium Documentation [Truy cập: 07/05/2025]