ĐẠI HỌC DUY TÂN

TRƯỜNG ĐẠI HỌC KHOA HỌC MÁY TÍNH

KHOA KHOA HỌC MÁY TÍNH

dtu-logo-072005_r1_c1--------------🙖🙐✰🙖🙐-------------

**BÁO CÁO ĐỒ ÁN CDIO**

**BÁO CÁO MÔN ĐỒ ÁN CDIO**

**ĐỀ TÀI: HUẤN LUYỆN CON RẮN SĂN MỒI BẰNG HỌC TĂNG CƯỜNG**



**Giáo viên hướng dẫn: Th. S Nguyễn Dũng**

**Sinh viên thực hiện: Hồ Nhật An - 25211210609  
 Phạm Đức Bình - 25211905472**

**Nguyễn Tiến Pháp - 25211200820**

**Nhóm: 1  
Lớp: DS 397 D**

**Tt (19)***Đà Nẵng, tháng 6 năm 2022*

MỤC LỤC

[DANH SÁCH CÁC HÌNH MINH HỌA 3](#_Toc106607680)

[DANH MỤC CÁC KÝ HIỆU 4](#_Toc106607681)

[CHƯƠNG 1. TỔNG QUAN VỀ ĐỀ TÀI VÀ CƠ SỞ LÝ THUYẾT 5](#_Toc106607682)

[1.1. Lý do chọn đề tài 5](#_Toc106607683)

[1.2. Mục tiêu và phương pháp nghiên cứu 5](#_Toc106607684)

[1.2.1. Mục tiêu của đề tài 5](#_Toc106607685)

[1.2.2. Phương pháp nghiên cứu 5](#_Toc106607686)

[1.3. Bố cục báo cáo 5](#_Toc106607687)

[1.4. Giới thiệu lý thuyết về Machine Learning 6](#_Toc106607688)

[1.5. Giới thiệu ngôn ngữ và công cụ lập trình 7](#_Toc106607689)

[1.6. Deep Learning 8](#_Toc106607690)

[1.7. Giới thiệu thư viện 9](#_Toc106607691)

[CHƯƠNG 2: GIỚI THIỆU THUẬT TOÁN 11](#_Toc106607692)

[2.1. Giới thiệu học tăng cường 11](#_Toc106607693)

[2.2. Các thuật ngữ trong học tăng cường 12](#_Toc106607694)

[2.3. Markov Decision Process (MDP) 15](#_Toc106607695)

[2.4. Q\_learning 16](#_Toc106607696)

[2.5. Deep Q Learning 17](#_Toc106607697)

[2.6. Experience replay 18](#_Toc106607698)

[2.7. Ví dụ về học tăng cường 19](#_Toc106607699)

[CHƯƠNG 3: XÂY DỰNG CHƯƠNG TRÌNH 20](#_Toc106607700)

[3.1. Xây dựng mô hình 20](#_Toc106607701)

[3.1.1. Xây dựng lớp SnakeGameAI: 20](#_Toc106607702)

[3.1.2. Xây dựng model: 24](#_Toc106607703)

[3.1.3. Xây dựng lớp Helper: 26](#_Toc106607704)

[3.1.4. Xây dựng lớp Agent 27](#_Toc106607705)

[3.2. Demo 30](#_Toc106607706)

[KẾT LUẬN 34](#_Toc106607707)

[TÀI LIỆU THAM KHẢO 36](#_Toc106607708)

# DANH SÁCH CÁC HÌNH MINH HỌA

[Hình 1. Các loại học máy 12](#_Toc106575905)

[Hình 2. Agent 12](#_Toc106575906)

[Hình 3. Enviroment 13](#_Toc106575907)

[Hình 4. Observations 14](#_Toc106575908)

[Hình 5. State 15](#_Toc106575909)

[Hình 6. Reward 16](#_Toc106575910)

[Hình 7. Minh họa cấu trúc của Q learning và Deep learning 18](#_Toc106575911)

[Hình 8. Mô hình deep learning 19](#_Toc106575912)

[Hình 9. Giao diện khi mới bắt đầu train 31](#_Toc106575913)

[Hình 10. Kết quả sau khi train 5 phút 32](#_Toc106575914)

[Hình 11. Hình ảnh sau khi train 10 phút 32](#_Toc106575915)

[Hình 12. Số liệu thống kê sau khi train 30 phút 33](#_Toc106575916)

[Hình 13. Số liệu thống kê sau khi train 1tiếng 33](#_Toc106575917)

[Hình 14. Hình ảnh thống kê sau khi train 6 tiếng 34](#_Toc106575918)

# DANH MỤC CÁC KÝ HIỆU

|  |  |
| --- | --- |
| NN | Neural Network |
| Q (s,a) | Giá trị Q value tại trạng thái s |
| MDP | Markov Decision Process |
|  | Hệ số chiết khấu |
| Lr | Learning rate |
|  | Phần thưởng tại trạng thái s |
| TD | Temporal Difference |
| Max | Giá trị lớn nhất của một tập nào đó |

# CHƯƠNG 1. TỔNG QUAN VỀ ĐỀ TÀI VÀ CƠ SỞ LÝ THUYẾT

## 1.1. Lý do chọn đề tài

Các trò chơi điện tử không còn là xa lạ đối với cuộc sống chúng ta. Khi tham gia một trò chơi, ai cũng muốn dành về cho mình một điểm số cao nhất. Và đến ngày nay, với công nghệ AI có những bước tiến thần tốc tại thị trường Việt Nam, len lỏi ở mọi mặt của đời sống, các máy móc (AI) đã có thể tham gia các trò chơi và học hỏi thông qua các học máy học tăng cường (Reinforcement Learning). Khi chúng ta lập trình cho máy xong, máy có thể tự học hỏi, tự rút kinh nghiệm, tự đưa ra quyết định tiếp theo đến khi nào đạt kết quả tốt nhất.   
 Gần đây, reinforcement learning đã đạt được những thành tựu đáng kể khi các thuật toán của DeepMind (AlphaGo, AlphaZero, AlphaStar, ...) đã chiến thắng áp đảo các tuyển thủ thế giới trong những trò chơi mà con người đã từng nghĩ rằng máy móc sẽ không bao giờ có thể vượt mặt như cờ vây hay StarCraft.Ngoài lĩnh vực trò chơi điện tử ra, chúng ta cũng có thể áp dụng phương pháp này cho lĩnh vực Kinh doanh, Tiếp thị, Quảng cáo và Khoa Học, ...   
 Để hiểu hơn về cách thức hoạt động của Reinforcement Learning, chúng ta sẽ tìm hiểu thông qua trò chơi “Rắn săn mồi”. “Rắn săn mồi” là một game cổ điển, xuất hiện vào năm 1997 trên Nokia 6610 là những ô vuông liên tiếp nhau di chuyển trên một màn hình màu đen đơn giản, nhưng trò chơi này đã xây dựng rất thành công tên tuổi của mình. Là một trong những trò chơi hấp dẫn từng nổi bật với lứa tuổi 9x, 10x. Và bây giờ, chúng ta sẽ áp dụng học tăng cường vào trò chơi để có thể chinh phục được trò chơi tuy đơn nhìn giản mà khó khăn này.

## 1.2. Mục tiêu và phương pháp nghiên cứu

### **1.2.1. Mục tiêu của đề tài**

- Lập trình giúp máy học và chiến thắng trò chơi “Rắn săn mồi”.

- Tìm hiểu, học tập và vận dụng kiến thức trong học tăng cường

- Vận dụng được các thư viện PyTorch và Pygame

### 1.2.2. Phương pháp nghiên cứu

- Nghiên cứu lý thuyết: nghiên cứu các tài liệu về lý thuyết về Machine Learning, Mạng Neural và Reinforcement learning.   
- Nghiên cứu cách xây dựng chương trình và ứng dụng thư viện để máy học.

## 1.3. Bố cục báo cáo

Nội dụng báo cáo bao gồm các chương sau:

* Chương 1. Tổng quan đề tài và Cơ sở lý thuyết: Giới thiệu tổng quan, mục tiêu và phương pháp nghiên cứu đề tài. Giới thiệu về machine learning, deep learning, thư viện và ngôn ngữ lập trình cùng với các công cụ hỗ trợ.
* Chương 2. Giới thiệu thuật toán: Giới thiệu tổng về phương pháp học tăng cường, thuật toán Q learning từ đó phát triển deep Q learning. Các công thức toán học đằng sau Q learning và deep Q learning.
* Chương 3. Xây dựng chương trình: Xây dựng model và các lớp cần thiết. Demo kết quả sau khi train 200 lần và nhiều hơn sau đó.
* Kết luận. Đưa ra đánh giá về phương pháp và mô hình. Nêu các kết quả đạt được và hạn chế của đề tài. Ưu nhược điểm của phương pháp học tăng cường.

1.4. Giới thiệu lý thuyết về Machine Learning

Cuộc cách mạng công nghiệp 4.0 với bằng chứng là sự nổi lên của AI - Artificial Intelligence (Trí tuệ nhân tạo), cụ thể hơn là Machine learning (Học máy) đã làm thay đổi thế giới (1.0 – Động cơ hơi nước, 2.0 – Điện năng, 3.0 – Công nghệ thông tin). Ta có thể thấy rằng trí tuệ nhân tạo đang dần len lõi vào mọi mặt của đời sống của chúng ta. Một số ví dụ về AI trong cuộc sống mà mọi người có thể dễ dàng nhận thấy như là: Hệ thống đề xuất video trên theo sở thích người dùng trên Youtube, xe tự hành của Tesla hay Camera AI trên các mẫu điện thoại thông minh đời mới, đặc biệt là trị tuệ nhận tạo AlphaGo được tạo ra bởi Google DeepMind tại London, trong một trận thi đấu cờ vây, đã có thể chiến thắng trước Lee Sedol – Người từng 18 lần vô địch thế giới trong bộ môn này. Hay gần đây hơn là OpenAI có khả năng chơi game Dota2 chiến thắng trước những tuyển thủ chuyên nghiệp hàng đầu thế giới. Từ những vị dụ trên đã cho thấy được sự phát triển không ngừng của AI trên mọi lĩnh vực trong đời sống.

Machine learning hay học máy (máy học) là một lĩnh vực con thuộc Trị tuệ nhận tạo (AI) nhằm nghiên cứu và sử dụng các thuật toán giúp máy tính hoặc các hệ thống có thể “học” từ dữ liệu nhằm giải quyết những vấn đề cụ thể. Các thuật toán máy học cần được đưa vào một lượng dữ liệu cần thiệt để cho ra một mô hình chính xác. Các tập dữ liệu này sẽ được thu gom, làm sạch và sử dụng một thuật toán máy học để đưa ra một mô hình mô tả bộ dữ liệu này. Do tập dữ liệu đầu vào là khác nhau nên các thuật toán máy học sẽ có những phương pháp và kỹ thuật “học” khác nhau. Các thuật toán máy học có thể được chia làm thành 3 nhóm:

* Học có giám sát (Supervised Learning): là một phương pháp học mà ở đó các tập dữ liệu đầu vào đã được gán nhãn và nhiệm vụ là tạo ra một mô hình để mô tả hoặc một dự đoán dựa trên tập dự liệu tương tự khác. Học có giám sát có thế được phân loại thành hai bài toán nhỏ hơn đó là Phân loại(Classification) và Hồi quy(Regession). Bài toán Phân loài là bài toán mà các nhãn của bộ dữ liệu đầu vào thuộc vào một số nhóm hữu hạn. Bài toán Hồi quy là bài toán mà các nhãn của bộ giữ liệu đầu vào là các giá trị cụ thể nào đó.
* Học không có giám sát (Unsupervised Learning): Là một phương pháp học mà ở đó các tập dữ liệu đầu vào khi đó chưa hoặc không được gán nhãn. Phương pháp này sẽ dựa vào tập dữ liệu đó để có thể thực hiện một công việc nào đó như phân cụm dữ liệu hoặc tìm ra các quy luật hay nguyên tắc cho bộ dữ liệu đó. Do đó sẽ có hai bài toán con trong Học không có giám sát là Phân cụm dữ liệu (Clustering) và tìm Quy tắc kết hợp (association rules). Thuật toán Phân cụm dữ liệu giúp phân nhóm các điễm dữ liệu dữa trên các tính chất đặc điểm tương tự nhau. Quy tắc kết hợp là dựa vào tập dữ liệu đã cho để khai phá, tìm ra những quy luật hoặc quy tắc nào đó có ích trong một công việc cụ thể.
* Học tăng cường (Reinforcement Learning): là một phương pháp học tiên tiến nhằm tự xác định các hành động tiếp theo sao cho đạt được hiệu quả hay lợi ích cao nhất. Bài toán này có thể dễ dàng thấy được trong hệ thống xe tự hành.

## 1.5. Giới thiệu ngôn ngữ và công cụ lập trình

**Python** Python là một ngôn ngữ lập trình bậc cao được sử dụng cho nhiều mục đích khác nhau do Guido Van Rossum tạo ra và ra mắt lần đầu tiên vào năm 1991. Đặc điểm của Python là một ngôn ngữ mạnh mẽ, dễ học, dễ sử dụng. Nó chứa các cấu trúc câu lệnh với cú pháp tối giản, đơn giản và rõ ràng giúp việc học tập và trao đổi trong lập trình trở nên dễ dàng.

Python là cộng cụ tuyệt vời cho việc lập trình các thuật toán máy học. Ngôn ngữ này cho phép việc lập trình trở nên ngắn ngọn và dễ hiểu. Đồng thời, chứa đựng nhiều các hàm và thư viện được xây dựng và tích hợp sẵn trong nó. Trong khi đó, máy học và trí tuệ nhân tạo có nhiều các thuật toán và các công thức toán học phức tạp cùng với lượng dữ liệu lớn để có thể hoạt động. Sự đơn giản của Python giúp lập trình viên tập trung vào các vấn đề của máy học thay vì là các kĩ thuật sử dụng ngôn ngữ lập trình.

Python chứa nhiều các thư viện hỗ trợ việc tính toán và nghiên cứu máy học. Việc thực thi các thuật toán máy học và sử dụng dữ liệu lớn có thể sẽ phức tạp và tốn nhiều thời gian. Trong khi đó, Python cung cấp môi trường làm việc, thư viện hỗ trợ đã được lập trình và có hiệu năng mạnh mẽ. Việc này sẽ giúp tiết kiệm được nhiều vấn đề khác nhau và giúp quá trình lập trình diễn ra được tốt nhất.

**Pycharm** Phần mềm PyCharm cung cấp một bộ công cụ hoàn chỉnh cho các nhà phát triển Python chuyên nghiệp. PyCharm được xây dựng xung quanh một trình soạn thảo hiểu mã sâu sắc, và một trình sửa lỗi cho cái nhìn rõ ràng về hoạt động của mã. PyCharm cung cấp khả năng tích hợp với các công cụ cộng tác như hệ thống kiểm soát phiên bản và các tracker. Trình biên tập chuyên nghiệp mở rộng các yếu tố cần thiết bằng cách tích hợp liền mạch với các khuôn khổ web, các công cụ JavaScript, ảo hóa và hỗ trợ containerization.

Một khía cạnh quan trọng của chương trình là hiểu được nền tảng mã mà bạn đang đưa vào. PyCharm đảm bảo bạn có thể khám phá dự án của bạn chỉ với một vài thao tác trên phím, nó cung cấp cho bạn một cái nhìn tổng quan về cấu trúc dự án và cho phép bạn truy cập vào các tài liệu có liên quan ngay từ trình soạn thảo. Hiểu được một nền tảng code nhanh hơn có nghĩa là thúc đẩy nhanh hơn quá trình phát triển của bạn.

## 1.6. Deep Learning

Deep Learning được bắt nguồn từ thuật toán **Neural network** vốn xuất phát chỉ là một ngành nhỏ của machine learning. Deep Learning là một chi của ngành máy học dựa trên một tập hợp các thuật toán để cố gắng mô hình dữ liệu trừu tượng hóa ở mức cao bằng cách sử dụng nhiều lớp xử lý với cấu trúc phức tạp, hoặc bằng cách khác bao gồm nhiều biến đổi phi tuyến .

Deep Learning đã giúp máy tính thực thi những việc tưởng chừng như không thể vào 15 năm trước: phân loại cả ngàn vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn, phim, ảnh, âm nhạc .

Khi dữ liệu của bạn phần lớn không có cấu trúc và bạn có rất nhiều dữ liệu.Các thuật toán Deep Learning có thể lấy dữ liệu lộn xộn và không có nhãn rộng rãi – chẳng hạn như video, hình ảnh, bản ghi âm thanh và văn bản – và áp đặt đủ thứ tự cho dữ liệu đó để đưa ra dự đoán hữu ích, xây dựng hệ thống phân cấp các tính năng tạo nên con chó hoặc con mèo một hình ảnh hoặc âm thanh tạo thành một từ trong lời nói.

Deep Learning ngày càng được nhiều người biết đến và nó những bước đột phá vô cùng to lớn. Những đột phá to lớn này là việc thiết kế ra những trợ lý ảo bằng giọng nói, các hệ thống xe tự lái hay sử dụng vào thiết kế đồ họa, phân tích tình hình giao thông của thành phố, cho đến phát triển các nguyên liệu mới giúp robot thấu hiểu thế giới xung quanh hơn. Xu hướng về Robot và Deep Learning đang được nhiều công ty công nghệ lớn chú trọng đầu tư và phát triển.

Việc Deep Learning phát triển tạo nên sự chủ động trong mọi việc, con người dần có thể điều khiển cuộc sống của mình. Cùng điểm qua các hình thức mà việc học sâu mang lại.

## 1.7. Giới thiệu thư viện

**Pygame** Thư viện pygame là một module open-souce dành cho ngôn ngữ lập trình Python nhằm mục đích đặc biệt giúp bạn tạo trò chơi và các ứng dụng đa phương tiện khác. Được xây dựng dựa trên thư viện phát triển SDL (Simple DirectMedia Layer) có tính di động cao, pygame có thể chạy trên nhiều nền tảng và hệ điều hành.   
 Bằng cách sử dụng module pygame, bạn có thể kiểm soát logic và đồ họa của trò chơi của bạn mà không cần lo lắng về những phức tạp backend cần thiết để làm việc với video và âm thanh.

**Ramdom** Random và “secrets” module, bộ tạo số giả ngẫu nhiên mặc định của random module được thiết kế với trọng tâm vào mô phỏng chứ không phải về bảo mật. Vì vậy, bạn không nên tạo ra các thông tin nhạy cảm như mật khẩu, secure tokens, khóa phiên và những thứ tương tự bằng cách sử dụng các hàm ngẫu nhiên đó.

**Enum** Là một công cụ mạnh mẽ để xử lý các tập dữ liệu không thay đổi. Điều này không chỉ là yếu tố không thể thiếu để phát triển các phần mềm tổng quát mà còn là một công cụ hữu ích để xây dựng các trò chơi – khiến nó trở thành một kỹ năng và nền tảng quan trọng trong python.   
 Enum viết tắt của từ enumeration có nghĩa là phép liệt kê, tính năng này xác định một tập hợp các tên được liên kết với các hằng số như số, chuỗi, v.v. Enum rất hữu ích để biểu diễn dữ liệu đại diện cho một tập hợp hữu hạn các trạng thái ví dụ như ngày trong tuần, tháng trong năm, …

**Collection** Module collection trong Python được định nghĩa là một bộ chứa được sử dụng để lưu trữ các bộ sưu tập dữ liệu, ví dụ: list, dict, set và tuple, ... Nó được giới thiệu để cải thiện các chức năng của bộ chứa bộ sưu tập tích hợp.   
 Python OrderedDict() tương tự như một đối tượng Dictionary trong đó các khóa duy trì thứ tự chèn. Nếu chúng ta cố gắng chèn khóa một lần nữa, giá trị trước đó sẽ bị ghi đè cho khóa đó.

**Torch** Torch là một framework được xây dựng dựa trên python cung cấp nền tảng tính toán khoa học phục vụ lĩnh vực Deep learning. Torch tập trung vào 2 khả năng chính:   
 - Một sự thay thế cho bộ thư viện numpy để tận dụng sức mạnh tính toán của GPU.  
 - Một platform Deep learning phục vụ trong nghiên cứu, mang lại sự linh hoạt và tốc độ.

**Ipython**IPython là một interactive shell của python khá hữu ích (so với shell chuẩn của python thì nó hữu ích hơn vì nó hổ trợ nhắc lệnh, một số magic function, …).   
 IPython không hẳn là interactive shell duy nhất, bạn có thể tham khảo thêm bpython, đây cũng là một interactive shell của python khá hữu ích.

**Numpy** Numpy (Numeric Python): là một thư viện toán học phổ biến và mạnh mẽ của Python. Cho phép làm việc hiệu quả với ma trận và mảng, đặc biệt là dữ liệu ma trận và mảng lớn với tốc độ xử lý nhanh hơn nhiều lần khi chỉ sử dụng “core Python” đơn thuần. Cụ thể, Python là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính.

**Matplotlib** Để thực hiện các suy luận thống kê cần thiết, cần phải trực quan hóa dữ liệu của bạn và Matplotlib là một trong những giải pháp như vậy cho người dùng Python. Nó là một thư viện vẽ đồ thị rất mạnh mẽ hữu ích cho những người làm việc với Python và NumPy. Module được sử dụng nhiều nhất của Matplotib là Pyplot cung cấp giao diện như MATLAB nhưng thay vào đó, nó sử dụng Python và nó là nguồn mở.

# CHƯƠNG 2: GIỚI THIỆU THUẬT TOÁN

## 2.1. Giới thiệu học tăng cường

Các thuật toán học máy thường được phân thành 3 loại lớn: supervised learning (học có giám sát), unsupervised learning (học không giám sát) và reinforcement learning (học tăng cường). Nếu như supervised learning là học tập từ một tệp các dữ liệu được gắn nhãn để suy luận ra quan hệ giữa đầu vào và đầu ra, thì unsupervised learning không được cung cấp các dữ liệu được gắn nhãn ấy, thay vào đó chỉ được cung cấp dữ liệu mà thuật toán tìm cách mô tả dữ liệu và cấu trúc của chúng. Loại thứ 3 là reinforcement learning - phương pháp tập trung vào việc làm thế nào để cho một tác tử trong môi trường có thể hành động sao cho lấy được phần thưởng nhiều nhất có thể. Khác với học có giám sát, học tăng cường không có cặp dữ liệu gán nhãn trước làm đầu vào và cũng không có đánh giá các hành động là đúng hay sai.

Reinforcement learning là đào tạo các mô hình học máy để đưa ra một chuỗi các quyết định. Tác tử học cách đạt được mục tiêu trong một môi trường không chắc chắn, có thể là phức tạp.

Đến đây ta có thể thấy, reinforcement learning là một nhánh của machine learning.

Diagram, venn diagram

Description automatically generated

Hình 1. Các loại học máy

## **2.2. Các thuật ngữ trong học tăng cường**

**Agent** Trong reinforcement learning có một thuật ngữ gọi là agent - được định nghĩa là “anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators” (máy quan sát môi trường và sinh ra hành động tương ứng).

Text

Description automatically generated with medium confidence

Hình 2. Agent

**Enviroment**  
 Môi trường là không gian xung quanh của agent, nơi mà agent tồn tại và tương tác

Text

Description automatically generated

Hình 3. Enviroment

**Action**  
 Hành động là phương thức của agent cho phép nó tương tác với môi trường và thay đổi môi trường. Dựa trên **State S(t)** của environment hiện tại mà agent sẽ đưa ra **action** **a(t)**

Timeline

Description automatically generated

Hình 3. Action

**Observation**  
 Sau khi nhận được sự tương tác từ agent thì environment có sự chuyển đổi trạng thái đối với agent

Diagram, timeline

Description automatically generated

Hình 4. Observations

**State**  
 Là trạng thái của môi trường mà agent nhận được

Diagram, timeline

Description automatically generated

Hình 5. State

**Policy**  
 Chính sách là yếu tố xác định cách thức hoạt động của agent tại một thời điểm nhất định. Nói cách khác, chính sách là một ánh xạ từ các trạng thái (state) của môi trường đến các hành động sẽ được thực hiện khi ở trong các trạng thái đó. Chính sách là cốt lõi của agent trong việc xác định hành vi. Trong một số trường hợp, chính sách có thể là một hàm hoặc bảng tra cứu đơn giản. Trong một số trường hợp khác, chính sách có thể liên quan đến tính toán mở rộng, ví dụ như quá trình tìm kiếm.

**Reward**  
 Ở mỗi hành động, môi trường gửi đến cho agent một phần thưởng xác định. Mục tiêu của agent là tối đa hóa tổng phần thưởng mà nó nhận được trong một thời gian dài. Tín hiệu phần thưởng (reward signal) giúp xác định đâu là sự kiện tốt và xấu đối với agent, đồng thời nó cũng là cơ sở chính để thay đổi chính sách. Nếu một hành động được lựa chọn bởi chính sách mang đến phần thưởng thấp, thì chính sách đó có thể bị thay đổi. Agent sẽ lựa chọn các hành động khác trong các tình huống tương tự ở tương lai.

Diagram, timeline

Description automatically generated

Hình 6. Reward

**Khai thác và khám phá** Một trong những thách thức nảy sinh trong reinforcement learning, đó là sự đánh đổi giữa khai thác và khám phá (exploit or explore). Để nhận được nhiều phần thưởng, agentphải ưu tiên lựa chọn các hành động mà nó đã từng thử trong quá khứ và giúp nó đạt được phần thưởng. Agent sẽ xem tất cả các hành động có thể xảy ra cho một trạng thái nhất định, sau đó lựa chọn hành động dựa trên giá trị tối đa của những hành động đó. Đây gọi là khai thác (exploit) vì chúng ta sử dụng thông tin có sẵn để đưa ra một quyết định.

Ngoài ra, agent thay vì chọn các hành động dựa trên phần thưởng tối đa trong tương lai, nó có thể chọn hành động một cách ngẫu nhiên. Hành động ngẫu nhiên rất quan trọng vì nó cho phép agentthăm dò và khám phá các trạng thái mới mà không được lựa chọn trong quá trình khai thác. Tóm lại, agent phải khai thác những gì mà nó đã trải qua để nhận được phần thưởng, nhưng cũng phải khám phá để đưa ra lựa chọn hành động tốt hơn trong tương lai.

## 2.3. Markov Decision Process (MDP)

MDP là một framework giúp agent đưa ra quyết định tại một state nào đó. Để áp dụng được framework này, ta giả sử các states có thuộc tính Markov (**Markov Property**): mỗi state chỉ phụ thuộc vào state trước nó vào xác suất chuyển đổi giữa 2 states này. Nghe hơi khó hiểu nhưng thực ra rất đơn giản. Đầu tiên, sao lại có **"*xác suất chuyển đổi giữa 2 states*"**? Trong một "thế giới hoàn hảo", nếu ta lặp lại một hành động thì sẽ cho ra 2 kết quả giống hệt nhau. Nhưng mọi việc không đơn giản như vậy vì hầu hết mọi sự việc đều là các tiến trình ngẫu nhiên (**stochastic process**)

Với Markov property, chúng ta có thể áp dụng MDP để biểu diễn bài toán dưới dạng:   
 Trong đó: S là tập các states, A là tập các actions, R là reward nhận được khi chuyển state, P là phân bố xác suất chuyển đổi.

Việc biểu diễn này rất quan trọng, khi mà ta không phải lưu một chuỗi các states trước đó để biểu diễn state hiện tại khiến cho việc tính toán trở nên phức tạp và tiêu tốn bộ nhớ.

## 2.4. Q\_learning

Q-learning là một thuật toán học tăng cường không có mô hình (Model-free). Nó học tập dựa trên các giá trị (values-based). Các thuật toán dựa trên giá trị cập nhật hàm giá trị dựa trên một phương trình (đặc biệt là phương trình Bellman). Trong khi loại còn lại, dựa trên chính sách (policy-based) ước tính hàm giá trị với một chính sách tham lam có được từ lần cải tiến chính sách cuối cùng.

**Q-learning is an off-policy learner**

Có nghĩa là nó học được giá trị của chính sách (policy) tối ưu một cách độc lập với các hành động của chủ thể (agent). Mặt khác, on-policy learner tìm hiểu giá trị của chính sách đang được thực hiện bởi chủ thể, bao gồm các bước thăm dò và họ sẽ tìm ra một chính sách tối ưu, có tính đến việc khám phá (exploration) vốn có trong chính sách.

Vậy là chúng ta đã biết được MDP, nhưng áp dụng vào RL learning như thế nào? Làm thế nào mà agent biết phải chọn action nào để đạt được reward lớn nhất? Câu trả lời là sử dụng một giá trị gọi là Q-value được tính bằng công thức:

Trong đó Q (s, a) là Q-value khi thực hiện action a tại state s; r(s, a) là reward nhận được; s' là state kế tiếp. γ là hệ số discount, đảm bảo càng "xa" đích Q-value càng nhỏ

Công thức này cho thấy Q-value của action a tại state s bằng reward r(s,a) cộng với Q-value lớn nhất của các states s' tiếp theo khi thực hiện các action a. Vậy đó, chỉ với công thức đơn giản kia chúng ta có thể tạo ra một ma trận state-action như một lookup table. Từ đó với mỗi state agent chỉ cần tìm action nào có Q-value lớn nhất là xong. Tuy nhiên, như mình đã nói ở trên thì RL là một stochastic process nên Q-value ở thời điểm trước và sau khi thực hiện action sẽ khác nhau. Khác biệt này gọi là Temporal Difference:

Như vậy, ma trận Q (s, a) cần phải cập nhật trọng số dựa trên TD:

Trong đó α là learning rate. Qua các lần agent thực hiện actions, Q (s, a) sẽ dần hội tụ. Quá trình này chính là Q-Learning

## 2.5. Deep Q Learning

Mục đích của chúng ta là chọn ra action thích hợp cho một state nào đó; hay nói cách khác, chúng ta state làm input, output là một action. Đúng vậy đấy, không có gì thích hợp để giải quyết việc này hơn một Neural Network (NN). Những gì cần làm chỉ là bỏ đi lookup table Q (s, a) và thay thế bằng một NN đơn giản.

Diagram

Description automatically generated

Hình 7. Minh họa cấu trúc của Q learning và Deep learning

Tuy nhiên, vẫn còn thiếu phần quan trọng nhất của NN. Đó chính là **hàm Loss.** Mục đích của ta là bắt NN học được cách ước lượng Q-Value cho các actions một cách chính xác nên đương nhiên hàm Loss phải tính được sai số giữa Q-value thực tế và dự đoán. Hóa ra hàm Loss này ta đã định nghĩa từ trước rồi các bạn ạ. Nó chính là TD2(a, s). Viết dưới dạng đầy đủ:

Timeline

Description automatically generated

Hình 8. Mô hình deep learning

Deep Q-Learning thực hiện các bước sau:

1. Enviroment đưa vào mạng một state s; đầu ra là các Q-value của các actions tương ứng.
2. Agent chọn action bằng một Policy và thực hiện action đó.
3. Environment trả lại state s' và reward r là kết quả của action a và lưu experience tuple [s, a, r, s'] vào memory
4. Thực hiện sample các experience thành một vài batches và tiến hành train NN
5. Lặp lại đến khi kết thúc M episodes

## 2.6. Experience replay

Một NN lấy input là state hiện tại và output các Q-value. Thế nhưng nếu NN cứ liên tục bị đẩy vào từng state một sẽ rất dễ bị overfitting vì các states liên tục thường giống nhau hoặc có tính tuyến tính (ví dụ: liên tục đi thẳng/sang trái/phải). Kỹ thuật Experience Replay được sử dụng để loại bỏ vấn đề này.

Thay vì mỗi state mạng update một lần, ta lưu lại các states vào bộ nhớ (memory). Sau đó thực hiện sampling thành các batch nhỏ đưa vào NN học. Việc này giúp đa dạng hóa input và tránh NN bị overfitting.

## 2.7. Ví dụ về học tăng cường

Để hiểu rõ hơn về reinforcement learning, ta sẽ xem xét một số ví dụ và các ứng dụng khả thi đã định hướng cho sự phát triển của nó.

* Một người chơi cờ vua thực hiện một nước đi. Nước đi được lựa chọn dựa trên việc lên chiến lược (dự đoán các nước đi của đối thủ có thể xảy ra), và bằng các phán đoán trực quan về các vị trí và nước đi cụ thể mà người chơi đó mong muốn thực hiện.
* Một con bê linh dương vật lộn để đứng dậy vài phút sau khi được sinh ra. Nửa giờ sau nó đang chạy ở mức 20 dặm một giờ.
* Robot di động quyết định xem nó có nên vào một căn phòng mới để tìm thêm rác để thu dọn hay cố gắng tìm đường trở lại trạm sạc pin của nó. Nó đưa ra quyết định dựa trên mức sạc hiện tại của pin và mức độ nhanh chóng và dễ dàng mà nó có thể tìm thấy bộ sạc trong quá khứ.

Tất cả những ví dụ trên đều liên quan đến sự tương tác giữa agent và môi trường của nó, trong đó agent tìm cách đạt được mục tiêu bất chấp sự không chắc chắn về môi trường nơi mà nó tồn tại. Các hành động của agent ảnh hưởng đến trạng thái tương lai của môi trường (thế cờ tiếp theo, tốc độ của bê linh dương sau 2 giờ, vị trí tiếp theo của robot và mức sạc trong tương lai của pin…), do đó nó cũng ảnh hướng đến các hành động và cơ hội của agent trong những lần tiếp theo.

Đồng thời, trong tất cả các ví dụ này, không thể dự đoán đầy đủ các ảnh hưởng của hành động; do đó agent phải theo dõi môi trường của nó thường xuyên và phản hồi thích hợp. Các ví dụ trên đều liên quan đến các mục tiêu rõ ràng, nghĩa là agent có thể đánh giá tiến trình đạt được mục tiêu của mình dựa trên những gì nó cảm nhận trực tiếp (người chơi cờ vua biết mình có thắng hay không, con linh dương biết khi nào nó ngã, robot di động biết khi nào pin cạn). Agent có thể sử dụng kinh nghiệm để cải thiện hiệu suất của nó theo thời gian (người chơi cờ tinh chỉnh trực giác mà anh ta sử dụng để đánh giá các vị trí, từ đó cải thiện cách chơi của mình; bê linh dương cải thiện khả năng chạy của nó…).

# CHƯƠNG 3: XÂY DỰNG CHƯƠNG TRÌNH

## **3.1. Xây dựng mô hình**

### **3.1.1. Xây dựng lớp SnakeGameAI:**

Để tạo ra môi trường để rắn thực hiện những hành động cũng như mục đích của mình muốn rắn thực hiện, ta có hàm sau:

|  |
| --- |
| def \_\_init\_\_(self, w=800, h=600):  self.w = w  self.h = h  self.display = pygame.display.set\_mode((self.w, self.h))  pygame.display.set\_caption('Snake')  self.clock = pygame.time.Clock()  self.reset() |

Sau khi có được môi trường thì việc chúng ta quan tâm là tác tử sẽ ở đâu thì tác tử sẽ được biểu diễn là như ô vuông màu xanh dương có kích thước 20x20 như sau:

|  |
| --- |
| def \_update\_ui(self):  self.display.fill(BLACK)  for pt in self.snake:  pygame.draw.rect(self.display, BLUE1, pygame.Rect(pt.x, pt.y, BLOCK\_SIZE, BLOCK\_SIZE))  pygame.draw.rect(self.display, BLUE2, pygame.Rect(pt.x + 4, pt.y + 4, 12, 12))  pygame.draw.rect(self.display, RED, pygame.Rect(self.food.x, self.food.y, BLOCK\_SIZE, BLOCK\_SIZE))  text = font.render("Score: " + str(self.score), True, WHITE)  self.display.blit(text, [0, 0])  pygame.display.flip() |

Tiếp theo là phần thưởng. Phần sẽ được biểu diễn bằng màu đỏ và một đơn vị phần thưởng có kích thước bằng 1 đơn vị chiều dài của rắn là một ô vuông có kích thước 20x20 và mỗi khi rắn đạt được phần thưởng thì phần thưởng mới được sinh ra ngẫu nhiên nhưng nằm ngoài cơ thể của rắn.

|  |
| --- |
| def \_place\_food(self):  x = random.randint(0, (self.w - BLOCK\_SIZE) // BLOCK\_SIZE) \* BLOCK\_SIZE  y = random.randint(0, (self.h - BLOCK\_SIZE) // BLOCK\_SIZE) \* BLOCK\_SIZE  self.food = Point(x, y)  if self.food in self.snake:  self.\_place\_food() |

Khi bắt đầu trò chơi chúng ta khởi tạo một số chỉ số như sau:

|  |
| --- |
| def reset(self):  self.direction = Direction.RIGHT  self.head = Point(self.w / 2, self.h / 2)  self.snake = [self.head,  Point(self.head.x - BLOCK\_SIZE, self.head.y),  Point(self.head.x - (2 \* BLOCK\_SIZE), self.head.y)]  self.score = 0  self.food = None  self.\_place\_food()  self.frame\_iteration = 0 |

Khi bắt đầu rắn ở trạng thái hướng về bên phải, đầu rắn sẽ nằm ở chính giữa khung hình (môi trường), chiều dài của rắn sẽ là 3 với một cái đầu và 2 ô vuông phần thân. Điểm của trò chơi sẽ khởi tạo là 0. Phần thưởng sẽ được tạo ra ngẫu nhiễn.

Tiếp theo chúng ta sẽ đến với vấn đề rắn sẽ xác định hành động tiếp theo như nào?

|  |
| --- |
| def \_move(self, action):  # [straight, right, left]  clock\_wise = [Direction.RIGHT, Direction.DOWN, Direction.LEFT, Direction.UP]  idx = clock\_wise.index(self.direction)  if np.array\_equal(action, [1, 0, 0]):  new\_dir = clock\_wise[idx] # no change  elif np.array\_equal(action, [0, 1, 0]):  next\_idx = (idx + 1) % 4  new\_dir = clock\_wise[next\_idx] # right turn r -> d -> l -> u  else: # [0, 0, 1]  next\_idx = (idx - 1) % 4  new\_dir = clock\_wise[next\_idx] # left turn r -> u -> l -> d  self.direction = new\_dir  x = self.head.x  y = self.head.y  if self.direction == Direction.RIGHT:  x += BLOCK\_SIZE  elif self.direction == Direction.LEFT:  x -= BLOCK\_SIZE  elif self.direction == Direction.DOWN:  y += BLOCK\_SIZE  elif self.direction == Direction.UP:  y -= BLOCK\_SIZE  self.head = Point(x, y) |

Ở hàm này giúp chúng ta xác định vị trí của đầu rắn khi nó thực hiện hành động tiếp theo. Theo cơ chế, đi lên tọa độ trục y sẽ tăng thêm một đơn vị chiều dài của rắn, đi xuống sẽ ngược lại, đi sang phải tọa độ trục x sẽ tăng thêm một đơn vị chiều dài của rắn và ngược lại với đi sang trái.

Vậy những điều kiện nào trò chơi nào sẽ kết thúc, chúng ta sẽ có 2 điều kiện kết thúc trò chơi. Thứ nhất là chạm tường, thứ 2 là đầu rắn chạm vào phần thân của chính nó. Và nó được thể hiện qua hàm dưới đây:

|  |
| --- |
| def is\_collision(self, pt=None):  if pt is None:  pt = self.head  # hits boundary  if pt.x > self.w - BLOCK\_SIZE or pt.x < 0 or pt.y > self.h - BLOCK\_SIZE or pt.y < 0:  return True  # hits itself  if pt in self.snake[1:]:  return True  return False |

Tiếp theo, chúng ta sẽ tìm hiểu rắn sẽ di chuyển, thoát trò chơi, game thất bại như thế nào qua hàm dưới đây:

|  |
| --- |
| def play\_step(self, action):  self.frame\_iteration += 1  # 1. collect user input  for event in pygame.event.get():  if event.type == pygame.QUIT:  pygame.quit()  quit()  # 2. move  self.\_move(action) # update the head  self.snake.insert(0, self.head)  # 3. check if game over  reward = 0  game\_over = False  if self.is\_collision() or self.frame\_iteration > 100 \* len(self.snake):  game\_over = True  reward = -10  return reward, game\_over, self.score  # 4. place new food or just move  if self.head == self.food:  self.score += 1  reward = 10  self.\_place\_food()  else:  self.snake.pop()  # 5. update ui and clock  self.\_update\_ui()  self.clock.tick(40)  # 6. return game over and score  return reward, game\_over, self.score |

Chúng ta có thể thoát game bất cứ lúc bằng bấm vào nút X trên khung giao diện mà chúng ta tạo ra.

Rắn di chuyển theo cơ chế luôn cập nhật đầu rắn sau mỗi hành động, nếu không ăn được phần thưởng sẽ xóa đi phần ô vuông cuối cùng của rắn, nếu ăn được phần thưởng chúng ta chỉ cần cập nhật đầu rắn và tạo ra phần thưởng mới. Sau đó thì cập nhật lại hay là vẽ lại cơ thể rắn. Tốc độ di chuyển của rắn tùy thuộc vào chúng ta muốn nó là bao nhiêu.

### 3.1.2. Xây dựng model:

Đầu tiên chúng ta xây dựng lớp Linear\_Qnet(nn.Module)

|  |
| --- |
| class Linear\_QNet(nn.Module):  def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):  super().\_\_init\_\_()  self.linear1 = nn.Linear(input\_size, hidden\_size)  self.linear2 = nn.Linear(hidden\_size, output\_size) |

Lớp này kế thừa lớp Module trong thư viện nn trong Torch. Ở đây chúng ta xây dựng model mạng NN với 1 lớp ẩn tương ứng với 2 bộ (weight,bias).

Khi đã có mạng NN thì chúng ta thực hiện lan truyền tiến với hàm kích hoạt là relu như sau:

|  |
| --- |
| def forward(self, x):  x = F.relu(self.linear1(x))  x = self.linear2(x)  return x |

Vậy làm sao để khi train chúng ta lưu lại được model của mình, để không phải thực hiện train lại mỗi khi bắt đầu lại hàm save() sẽ giúp chúng ta điều này.

|  |
| --- |
| def save(self, file\_name='phap2.pth'):  model\_folder\_path = 'D:/'  if not os.path.exists(model\_folder\_path):  os.makedirs(model\_folder\_path)  file\_name = os.path.join(model\_folder\_path, file\_name)  torch.save(self.state\_dict(), file\_name) |

Ở đây chúng ta đang mặc định file chúng ta lưu lại được lưu lại ở ổ D.

Tiếp theo chúng ta sẽ thực hiện quá trình deep Q learning.

|  |
| --- |
| class QTrainer:  def \_\_init\_\_(self, model, lr, gamma):  self.lr = lr  self.gamma = gamma  self.model = model  self.optimizer = optim.Adam(model.parameters(), lr=self.lr)  self.criterion = nn.MSELoss() |

Chúng ta khởi tạo một số biến như là hệ số chiết suất, model chính là model mình đã tạo ở lớp Linear\_Qnet(nn.Module). Hàm tối ưu Adam từ thư viện Torch, với learning rate do chúng ta đưa vào, các tham số được tối tối ưu chính là 2 bộ (weight, bias). Hàm mất mát được sử dụng ở đây là nn.MSELoss(), nó có dạng giống như hàm mất mát chúng ta dẫn ra ở trên.

|  |
| --- |
| def train\_step(self, state, action, reward, next\_state, done):  state = torch.tensor(state, dtype=torch.float)  next\_state = torch.tensor(next\_state, dtype=torch.float)  action = torch.tensor(action, dtype=torch.long)  reward = torch.tensor(reward, dtype=torch.float)  # (n, x)  if len(state.shape) == 1:  # (1, x)  state = torch.unsqueeze(state, 0)  next\_state = torch.unsqueeze(next\_state, 0)  action = torch.unsqueeze(action, 0)  reward = torch.unsqueeze(reward, 0)  done = (done,)  # 1: predicted Q values with current state  pred = self.model(state) #dự đoán  target = pred.clone() #  for idx in range(len(done)):  Q\_new = reward[idx] # thực tế  if not done[idx]:  Q\_new = reward[idx] + self.gamma\* torch.max(self.model(next\_state[idx]))  target[idx][torch.argmax(action[idx]).item()] = Q\_new  self.optimizer.zero\_grad()  loss = self.criterion(target, pred)  loss.backward()  print(loss)  self.optimizer.step() |

Tham số đầu vào của hàm này là trạng thái, hành động, phần thưởng hiện tại và trạng thái kế tiếp. Chúng ta sẽ chuyển các tham số này thành các tensor để phù hợp cho việc tính toán của chúng ta. Chúng ta sẽ dự đoán được một tensor có kích thước 3 là một tensor hành động mà model chúng ta dự đoán được. Sau đó chúng ta dùng Q learning để tính hành động tiếp theo mà chúng ta mong muốn. Khi đó chúng ta thực việc tính hàm mất mát rồi lan truyền ngược và cuối cùng là cập nhật các tham số (weight và bias)

### 3.1.3. Xây dựng lớp Helper:

|  |
| --- |
| import matplotlib.pyplot as plt  from IPython import display  plt.ion()  def plot(scores, mean\_scores):  display.clear\_output(wait=True)  display.display(plt.gcf())  plt.clf()  plt.title('Training...')  plt.xlabel('Number of Games')  plt.ylabel('Score')  plt.plot(scores)  plt.plot(mean\_scores)  plt.ylim(ymin=0)  plt.text(len(scores)-1, scores[-1], str(scores[-1]))  plt.text(len(mean\_scores)-1, mean\_scores[-1], str(mean\_scores[-1]))  plt.show(block=False)  plt.pause(.1) |

Với hàm này chúng ta sẽ lưu lại được lịch sử training của mình bằng đồ thị và điểm cao nhất khi đang train. Ngoài ra, chúng ta sẽ biết được điểm trung bình khi chúng ta train với số game nào đó.

### 3.1.4. Xây dựng lớp Agent

|  |
| --- |
| MAX\_MEMORY = 100000  BATCH\_SIZE = 1000  LR = 0.001  class Agent:  def \_\_init\_\_(self):  self.n\_games = 0  self.epsilon = 0 # randomness  self.gamma = 0.9 # discount rate  self.memory = deque(maxlen=MAX\_MEMORY) # popleft()  self.model = Linear\_QNet(11, 256, 3)  #self.model.load\_state\_dict(torch.load('D:/phap2.pth'))  self.trainer = QTrainer(self.model, lr=LR, gamma=self.gamma) |

Bắt đầu thì chúng ta có số game đã chơi hay đã train bao nhiêu game. Tiếp là biến epsilon để chúng ta cố thể phân biệt khi nào khai phá, khi nào thì khai thác. Tiếp theo là biến gamma, là một chỉ số không thể thiết trong việc thực hiện cập nhật giá trị Q value và đây chúng ta khởi tạo là 0.9. Chúng ta sẽ lưu lại những kinh nghiệm của rắn bằng một danh sách memory và độ dài của lịch sử này là 100000. Chúng ta khởi tạo model, với số node đầu vào là 11, một lớp ẩn với 256 node và đầu ra là 3 tương ứng với ba hành động, đi thẳng, sang trái, sang phải. Nếu chúng ta đã train model trước đó rồi và muốn train tiếp model đó thì chúng ta có thể khởi tạo model bằng model đã được lưu với câu lệnh như sau:

|  |
| --- |
| self.model.load\_state\_dict(torch.load('D:/phap2.pth')) |

Cuối cùng để train deep Q learning chúng ta khởi tạo biến trainer là một đối tượng của lớp QTrainer với cách tham số như: model là model phía trên, learning rate LR=0.001 và gamma bằng với chỉ số gamma ta đã tạo ở trên.

Tiếp theo chúng ta sẽ xem trạng thái đầu vào gồm những điều kiện gì, khi chúng ta đã model của chúng ta gồm 11 node.

|  |
| --- |
| def get\_state(self, game):  head = game.snake[0]  point\_l = Point(head.x - 20, head.y)  point\_r = Point(head.x + 20, head.y)  point\_u = Point(head.x, head.y - 20)  point\_d = Point(head.x, head.y + 20)  dir\_l = game.direction == Direction.LEFT  dir\_r = game.direction == Direction.RIGHT  dir\_u = game.direction == Direction.UP  dir\_d = game.direction == Direction.DOWN  state = [  # Danger straight  (dir\_r and game.is\_collision(point\_r)) or  (dir\_l and game.is\_collision(point\_l)) or  (dir\_u and game.is\_collision(point\_u)) or  (dir\_d and game.is\_collision(point\_d)),  # Danger right  (dir\_u and game.is\_collision(point\_r)) or  (dir\_d and game.is\_collision(point\_l)) or  (dir\_l and game.is\_collision(point\_u)) or  (dir\_r and game.is\_collision(point\_d)),  # Danger left  (dir\_d and game.is\_collision(point\_r)) or  (dir\_u and game.is\_collision(point\_l)) or  (dir\_r and game.is\_collision(point\_u)) or  (dir\_l and game.is\_collision(point\_d)),  # Move direction  dir\_l,  dir\_r,  dir\_u,  dir\_d,  # Food location  game.food.x < game.head.x, # food left  game.food.x > game.head.x, # food right  game.food.y < game.head.y, # food up  game.food.y > game.head.y # food down  ]  return np.array(state, dtype=int) |

Chúng ta có thể thấy 3 số đầu trong danh sách trạng thái đó là 3 các nguy hiểm mà rắn có thể gặp phải tiếp theo 4 số tiếp theo là 4 hướng di chuyển lên, xuống, trái, phải. Cuối là 4 vị trị của phần thưởng. Ở trên, dưới, bên phải, bên trái của rắn.

Chúng ta sẽ lưu lại những gì mà rắn học được bằng hàm remember(self, state, action, reward, next\_state, done):

|  |
| --- |
| def remember(self, state, action, reward, next\_state, done):  self.memory.append((state, action, reward, next\_state, done)) |

Để cho rắn biết khi nào khám phá khi nào khái thác, chúng ta sẽ thu được một hành động tiếp theo của rắn.

|  |
| --- |
| def get\_action(self, state):  # random moves: tradeoff exploration / exploitation  self.epsilon = 80 - self.n\_games  final\_move = [0, 0, 0]  if random.randint(0,80) < self.epsilon:  move = random.randint(0, 2)  final\_move[move] = 1  else:  state0 = torch.tensor(state, dtype=torch.float)  prediction = self.model(state0)  move = torch.argmax(prediction).item()  final\_move[move] = 1  return final\_move |

Để train từng state chúng xây dựng hàm train\_short\_memory():

|  |
| --- |
| def train\_short\_memory(self, state, action, reward, next\_state, done):  self.trainer.train\_step(state, action, reward, next\_state, done) |

Nhưng việc cứ train từng sate sẽ khiến chúng gặp vấn đề overfitting. Vấn đề này chúng ta đã nói ở mục Experience replay rồi. Để giải quyết vấn đề đó ta có hàm train\_long\_memory().

|  |
| --- |
| def train\_long\_memory(self):  if len(self.memory) > BATCH\_SIZE:  mini\_sample = random.sample(self.memory, BATCH\_SIZE) # list of tuples  else:  mini\_sample = self.memory  states, actions, rewards, next\_states, dones = zip(\*mini\_sample)  self.trainer.train\_step(states, actions, rewards, next\_states, dones) |

## 3.2. Demo

Bắt đầu thì chúng ta có giao diện như dưới đây:

Graphical user interface, application

Description automatically generated

Hình 9. Giao diện khi mới bắt đầu train

Graphical user interface

Description automatically generated

Hình 10. Kết quả sau khi train 5 phút

Graphical user interface, application

Description automatically generated

Hình 11. Hình ảnh sau khi train 10 phút

Chart, histogram

Description automatically generated

Hình 12. Số liệu thống kê sau khi train 30 phút

Chart, histogram

Description automatically generated

Hình 13. Số liệu thống kê sau khi train 1tiếng

A screenshot of a computer

Description automatically generated with medium confidence

Hình 14. Hình ảnh thống kê sau khi train 6 tiếng

Chúng ta dễ thấy sau 80 games thì rắn đã bắt đầu thông minh hơn vì đã kết thúc quá trình khám phá. Bắt đầu đi theo chính sách nên đạt kết quả tốt hơn. Số điểm cao nhất là 79. Tại sao học nhiều như vậy rắn vẫn mắc lỗi đó là vì khi cơ thể rắn dài lên tỉ lệ nó cắn trúng thân mình ngày càng cao lên. Còn khi cơ thể còn ngắn rắn có xu hướng va vào tường để rút kinh nghiệm cho lần sau.

# KẾT LUẬN

**1. Kết quả đạt được:**

- Hiểu được cách hoạt động của một bài toán về học tăng cường.

- Biết cách sử dụng một số thư viện như Pytorch, Enum, ….

- Hiểu thêm về thuật toán Deep Q Learning.

**2. Ưu điểm:**

- Nó có thể giải quyết các vấn đề phức tạp và bậc cao hơn. Ngoài ra, các giải pháp thu được sẽ rất chính xác.

- Lý do cho sự hoàn hảo của nó là nó rất giống với kỹ thuật học tập của con người.

- Mô hình này sẽ trải qua một quá trình đào tạo nghiêm ngặt có thể mất thời gian. Điều này có thể giúp sửa chữa bất kỳ lỗi nào.

- Là mô hình học hỏi liên tục nên một sai lầm được thực hiện trước đó sẽ khó có thể xảy ra trong tương lai.

- Có thể xây dựng nhiều mô hình giải quyết vấn đề khác nhau bằng cách sử dụng phương pháp Reinforcement Learning.

- Khi nói đến việc tạo mô phỏng, phát hiện đối tượng trong ô tô tự động, rô bốt, v.v., Reinforcement Learning đóng một vai trò lớn trong các mô hình.

- Phần tốt nhất là ngay cả khi không có dữ liệu đào tạo, nó sẽ học được thông qua kinh nghiệm mà nó có được từ việc xử lý dữ liệu đào tạo.

**3. Nhược điểm:**

Việc sử dụng các mô hình Reinforcement Learning để giải quyết các vấn đề đơn giản hơn sẽ không đúng. Lý do là, các mô hình thường giải quyết các vấn đề phức tạp.

Chúng ta sẽ lãng phí sức mạnh xử lý và không gian không cần thiết bằng cách sử dụng nó cho các vấn đề đơn giản hơn.

Reinforcement Learning cần nhiều dữ liệu để cung cấp cho mô hình tính toán. Các mô hình này yêu cầu nhiều dữ liệu đào tạo để phát triển các kết quả chính xác. Điều này tiêu tốn thời gian và nhiều sức mạnh tính toán.

Khi nói đến việc xây dựng mô hình trên các ví dụ thực tế, chi phí bảo trì rất cao. Giống như để chế tạo các phương tiện không người lái, rô bốt, chúng tôi sẽ yêu cầu bảo trì rất nhiều cho cả phần cứng và phần mềm.   
 Việc đào tạo quá nhiều có thể dẫn đến quá tải các trạng thái của mô hình. Điều này có thể xảy ra nếu quá nhiều dung lượng bộ nhớ hết trong quá trình xử lý dữ liệu huấn luyện.

**4. Hướng phát triển:**

- Tiếp tục ứng dụng thuật toán vào các lĩnh vực khác để nâng cao khả năng xử lý.

- Tối ưu thuật toán, cải thiện tốc độ tải.

- Hỗ trợ đa ngôn ngữ.

- Chỉnh sửa giao diện đẹp mắt hơn.

- Áp dụng học tăng cường vào trong cuộc sống hiện nay, nó giải quyết cho chúng ta những điều mà chúng ta khó có thể thực hiện được.

# **TÀI LIỆU THAM KHẢO**

[1]. <https://pythonmana.com/2021/07/20210731001919826f.html?fbclid=IwAR3eEnC2z8B7jt5j-i8vBIsc3Le0J9nO8DtEvJcm6CBmu3ZBY0e0pQ5mCbY>

[2]. <https://codelearn.io/sharing/reinforcement-learning-la-gi?fbclid=IwAR1_t5PJnQEFTy-HISCmZClw96dn6b-UOIZ_HQ9ibYSd7dS8WVVGntKRrdo>

[3]. <https://viblo.asia/p/gioi-thieu-ve-hoc-tang-cuong-va-ung-dung-deep-q-learning-choi-game-cartpole-Az45bYy6lxY>

[4]. <https://codelearn.io/sharing/lap-trinh-game-co-ban-voi-pygame?fbclid=IwAR1i3IwJYNnSQ9yA_NMMjtZOi_gO_WN4lT5e2iFVVSosMsk3wHPm-4xxFrE>