TRƯỜNG ĐẠI HỌC BÁCH KHOA - ĐẠI HỌC ĐÀ NẴNG

KHOA ĐIỆN TỬ - VIỄN THÔNG

BÁO CÁO CUỐI KỲ

MÔN: CHUYÊN ĐỀ 2

ĐỀ TÀI: XÂY DỰNG THUẬT TOÁN CHO TRÒ CHƠI RẮN TÌM ĐƯỜNG

Giảng viên hướng dẫn: TS. Nguyễn Văn Hiếu

Lớp học phần : 21.44

Nhóm : 5

Sinh viên thực hiện : Dương Thị Thảo Vi _ 106210259

Phạm Thị Phương _ 106210050

Nguyễn Văn An _ 106210045

Đà Nẵng, 2025

LỜI NÓI ĐẦU

Trong xu thế chuyển đổi số và phát triển mạnh mẽ của trí tuệ nhân tạo, các bài toán ra quyết định trong môi trường động, có ràng buộc an toàn và yêu cầu tối ưu theo thời gian thực ngày càng trở nên quan trọng. Nhằm củng cố kiến thức và rèn luyện kỹ năng nghiên cứu – triển khai các phương pháp giải bài toán điều khiển và tối ưu, nhóm chúng em thực hiện báo cáo kết thúc học phần Chuyên đề 2 với đề tài nghiên cứu và xây dựng hệ thống giải trò chơi Snake.

Nội dung báo cáo tập trung vào việc mô hình hóa bài toán trên môi trường lưới, xây dựng mô hình hệ thống, trình bày và so sánh nhiều hướng tiếp cận khác nhau gồm các thuật toán tìm kiếm – heuristic (BFS, heuristic đường dài), phương pháp tham lam (Greedy), phương pháp dựa trên chu trình Hamilton, và hướng học tăng cường với Deep Q-Network (DQN). Trên cơ sở đó, nhóm tiến hành đánh giá kết quả bằng các chỉ số thống kê như độ dài trung bình, số bước trung bình, cùng các thông số huấn luyện (reward, loss) để phân tích mức độ ổn định và hiệu quả của từng phương pháp.

Trong quá trình thực hiện, nhóm chúng em đã nỗ lực tìm hiểu tài liệu, thử nghiệm và điều chỉnh để hoàn thiện hệ thống trong phạm vi thời gian của học phần. Tuy nhiên, do hạn chế về kinh nghiệm nghiên cứu, thời gian thực hiện cũng như phạm vi triển khai, báo cáo khó tránh khỏi những thiếu sót nhất định cả về nội dung lẫn hình thức trình bày. Nhóm chúng em kính mong nhận được những nhận xét và góp ý từ Giảng viên, Tiến sĩ Nguyễn Văn Hiếu để có thể rút kinh nghiệm và cải thiện chất lượng báo cáo cũng như phương pháp làm việc trong các học phần và dự án tiếp theo.

Nhóm chúng em xin bày tỏ lòng biết ơn chân thành tới thầy Nguyễn Văn Hiếu đã tận tình giảng dạy, định hướng và hỗ trợ nhóm trong suốt quá trình học tập và thực hiện báo cáo.

Đà Nẵng, ngày 15 tháng 12 năm 2025

Sinh viên thực hiện

Dương Thị Thảo Vi

Phạm Thị Phương

Nguyễn Văn An

PHÂN CÔNG CÔNG VIỆC

STT/NHÓM THI	TÊN	ĐÓNG GÓP (%)
7A	Nguyễn Văn An	33.3 %
10A	Phạm Thị Phương	33.3 %
30B	Dương Thị Thảo Vi	33.3 %

(Nội dung cụ thể sẽ được mô tả trong bản tóm tắt cá nhân)

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU TỔNG QUAN

1.1. Giới thiệu đề tài

Trò chơi "Rắn săn mồi" *(Snake Game)* là một môi trường mô phỏng chứa đựng nhiều đặc tính quan trọng của các bài toán điều khiển trong thực tế: không gian di chuyển hẹp, ràng buộc va chạm nghiêm ngặt, trạng thái thay đổi liên tục và yêu cầu ra quyết định tối ưu theo thời gian thực. Trong lưới 2D giới hạn, hệ thống điều khiển phải giúp con rắn di chuyển an toàn, tránh va vào tường hoặc chính cơ thể mình, đồng thời thu thập được nhiều thức ăn nhất có thể. Càng ăn nhiều, rắn càng dài và không gian trống càng thu hẹp, khiến việc lựa chọn hướng đi trở nên khó khăn hơn và sai sót ở bất kỳ thời điểm nào cũng có thể dẫn đến thất bại.

Mặc dù luật chơi rất đơn giản, việc xây dựng một bộ điều khiển hiệu quả lại là một bài toán phức tạp đòi hỏi khả năng lập kế hoạch, dự đoán và tối ưu hóa. Mỗi bước đi của rắn cần được tính toán dựa trên cấu trúc bản đồ và các ràng buộc hiện tại để duy trì sự an toàn lâu dài. Trong phạm vi đề tài này, trò chơi Snake được sử dụng làm nền tảng để nghiên cứu và so sánh nhiều phương pháp điều khiển khác nhau, bao gồm thuật toán tìm đường cổ điển, chiến lược tham lam, mô hình chu trình Hamilton và phương pháp học tăng cường DQN. Sự kết hợp này cho phép đánh giá toàn diện hiệu quả của các chiến lược từ truyền thống đến hiện đại trong cùng một môi trường thống nhất.

Bài toán trong dự án có thể diễn giải theo hướng thực tế như sau: cần thiết kế một bộ điều khiển ra quyết định theo thời gian thực cho một tác nhân di chuyển trong không gian hẹp, có ràng buộc va chạm nghiêm ngặt và chướng ngại vật "động" do chính tác nhân tạo ra. Trò chơi Snake là một mô hình hoá đơn giản nhưng giàu tính đại diện cho các bài toán điều hướng robot trong kho hẹp, AGV/xe tự hành trong nhà máy hoặc drone bay trong hành lang: mỗi bước đi đều phải đảm bảo an toàn tức thời (không va chạm) đồng thời đảm bảo an toàn dài hạn (không tự đưa hệ vào trạng thái kẹt/không còn đường thoát).

Yêu cầu bài toán của dự án:

- Đầu vào: trạng thái lưới và cấu trúc rắn trên bản đồ 8×8 (tối đa 64 ô).
- Đầu ra: hành động điều khiển tại mỗi bước (hướng di chuyển tiếp theo).
- Ràng buộc: không đâm tường/không tự cắn; thức ăn xuất hiện ngẫu nhiên; không gian trống giảm khi rắn dài dần.
- Mục tiêu: tối đa số lần ăn (tương đương tối đa độ dài), đồng thời tối ưu hiệu quả di chuyển (giảm số bước/giảm thời gian đạt độ dài lớn).

1.2. Tính cấp thiết của bài toán

Bài toán điều khiển rắn trong không gian hẹp có tính cấp thiết bởi nó mô phỏng nhiều vấn đề thực tế trong lĩnh vực robot, điều hướng và trí tuệ nhân tạo. Trước hết, bài toán này phản ánh các tình huống mà robot phải di chuyển trong môi trường có diện tích nhỏ, nhiều ràng buộc, và phải đưa ra quyết định liên tục để tránh va chạm. Những mô hình như vậy xuất hiện trong robot di chuyển trong kho hàng, xe tự hành trong nhà máy, hay cả drone tránh vật cản trong không gian hai chiều.

Ngoài ra, bài toán còn mang tính học thuật cao. Khi chiều dài rắn tăng dần, không gian trống giảm mạnh, khiến bài toán nhanh chóng trở nên phức tạp và dễ rơi vào các tình huống "tự khóa" nếu không có chiến lược hợp lý. Điều này buộc người nghiên cứu phải áp dụng những thuật toán tối ưu hóa đường đi, heuristic, hoặc mô hình học sâu để tìm giải pháp. Do đó, đề tài không chỉ giúp sinh viên hiểu rõ thuật toán BFS, heuristic mở rộng đường đi hay chu trình Hamilton mà còn tạo cơ hội tiếp cận học tăng cường *(reinforcement learning)* qua mô hình DQN.

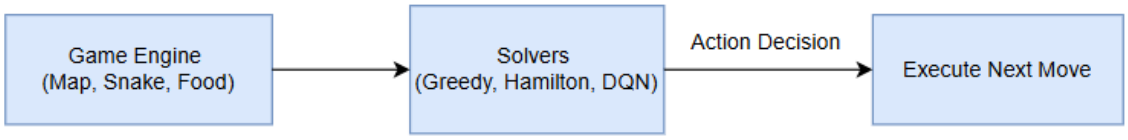
Cuối cùng, việc tổng hợp nhiều hướng tiếp cận trong cùng một hệ thống giúp quá trình đánh giá trở nên toàn diện hơn. Các thuật toán cổ điển có ưu điểm về độ ổn định và tốc độ tính toán, trong khi các phương pháp học sâu có tiềm năng tự tối ưu hóa qua thời gian. Việc phân tích và so sánh những phương pháp này là cần thiết để lựa chọn giải pháp phù hợp cho các ứng dụng thực tế.

1.3. Mô hình hệ thống và cách thức hoạt động

Hệ thống trong đề tài được thiết kế xoay quanh bốn bộ giải thuật (solver), mỗi bộ đảm nhiệm một cách tiếp cận khác nhau. Tất cả cùng hoạt động trên bản đồ 8×8 và

chia sẻ chung cơ chế vào – ra: đầu vào là trạng thái bản đồ, đầu ra là hành động tiếp theo của rắn. *Path Solver* đóng vai trò nền tảng, sử dụng *BFS* để tìm đường ngắn nhất và một thuật toán heuristic để xấp xỉ đường dài nhất từ đầu rắn đến đuôi rắn. *Greedy Solver* dựa trên các đường đi này để đưa ra quyết định: nếu có đường an toàn đến thức ăn thì tiến tới, còn không thì tạm thời di chuyển theo hướng an toàn nhất. *Hamilton Solver* xây dựng một chu trình Hamilton bao phủ toàn bộ bản đồ, giúp rắn có một lộ trình an toàn tuyệt đối, sau đó bổ sung thêm quy tắc shortcut nhằm tăng tốc độ ăn thức ăn. Cuối cùng, *DQN Solver* sử dụng mạng neural để học trực tiếp chính sách di chuyển tối ưu từ kinh nghiệm của chính nó, dựa trên trạng thái bản đồ và các tín hiệu phần thưởng.

Toàn bộ hệ thống hoạt động theo vòng lặp: game engine cập nhật trạng thái bản đồ → solver phân tích trạng thái và đưa ra hành động → con rắn di chuyển → game chuyển sang trạng thái mới → solver tiếp tục xử lý. Vòng lặp này diễn ra liên tục cho đến khi rắn chết hoặc bản đồ hoàn toàn bị chiếm đầy bởi thân rắn.



Hình 1.1 Sơ đồ hoạt động của hệ thống

Cụ thể, hệ thống gồm các phần chính sau:

- Game Engine: Mô phỏng bàn cờ 2D với kích thước cố định (ví dụ 8x8, chứa tối đa 64 ô – tương ứng chiều dài tối đa của rắn là 64). Môi trường bao gồm rắn (danh sách các tọa độ thân rắn, có đầu và đuôi), thức ăn (một ô thức ăn ngẫu nhiên xuất hiện tại một thời điểm), và các quy tắc trò chơi (rắn di chuyển mỗi bước một ô, nếu ăn thức ăn thì dài ra, nếu đâm vào tường hoặc thân mình thì thua). Môi trường có thể được cài đặt theo chuẩn OpenAI Gym (cung cấp hàm reset(), step() trả về trạng thái, phần thưởng) để tiện huấn luyện AI. Trong dự án này, Python + Tkinter được sử dụng để tạo giao diện hiển thị trò chơi.

- Solvers: Đây là thành phần trí tuệ nhân tạo quyết định hướng di chuyển của rắn tại mỗi bước. Dự án triển khai nhiều chiến lược AI khác nhau (gọi là các solver), bao gồm:
    + Hamilton Solver: thuật toán "đi theo chu trình Hamilton".
    + Greedy Solver: thuật toán "tham lam an toàn" tìm đường ngắn nhất tới mồi nhưng tránh tự sát.
    + DQN Solver: thuật toán Deep Q-Network (DQN) dùng học tăng cường sâu.
    + Ngoài ra, có thể có những solver khác (ví dụ thuật toán đường đi *BFS/A*** cơ bản để thử nghiệm), nhưng ba thuật toán trên là nổi bật trong dự án.
- Game Loop: Đây là quy trình liên kết môi trường và agent. Cụ thể:
    + Khởi tạo: Môi trường game được thiết lập (đặt rắn ở vị trí ban đầu, tạo thức ăn), chọn solver AI.
    + Quan sát trạng thái: Agent nhận trạng thái hiện tại của môi trường (vị trí đầu rắn, hướng di chuyển, vị trí thức ăn, độ dài rắn, v.v.).
    + Ra quyết định: Dựa trên trạng thái, solver AI tính toán và chọn một hành động (hướng di chuyển tiếp theo: lên, xuống, trái, phải).
    + Cập nhật môi trường: Môi trường nhận hành động và cập nhật: di chuyển rắn một ô theo hướng đó, kiểm tra va chạm (nếu đầu rắn đâm vào tường hoặc thân thì kết thúc game), kiểm tra ăn mồi (nếu rắn ăn thức ăn thì tăng chiều dài, tạo thức ăn mới ở vị trí ngẫu nhiên).
    + Lặp lại: Quay lại bước 2 cho đến khi game kết thúc (rắn chết hoặc đạt chiến thắng khi lấp đầy bản đồ).
- Giao diện người dùng (GUI): Hiển thị trực quan trạng thái trò chơi – thường dưới dạng ô vuông lưới, rắn được vẽ bằng các ô màu. GUI giúp người phát triển quan sát quá trình AI chơi game.
- Mô hình hệ thống (I/O): Hệ thống là một vòng lặp đóng, trong đó môi trường cập nhật trạng thái và solver trả về hành động điều khiển: *game engine → state → solver → action → game engine*.

Điểm mới của hệ thống:

- Tích hợp đa phương pháp trong một nền tảng thống nhất để so sánh trực tiếp (cùng bản đồ, cùng cơ chế trạng thái–hành động, cùng thước đo).
- Có chế độ Benchmark chạy nhiều episode không GUI để thu thống kê khách quan (Average Length, Average Steps).
- Với DQN, hệ thống có cơ chế lưu checkpoint + log huấn luyện (solver-net-* và solver-var-*.json) giúp phân tích hội tụ và tái lập kết quả.
- Greedy có ý tưởng "an toàn hóa" bằng dự đoán/đánh giá rủi ro thay vì chỉ tối ưu đường đi ngắn.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Mô tả và mô hình hóa bài toán

Bài toán điều khiển rắn trong trò chơi Snake có thể được xem như một bài toán điều hướng trong một môi trường lưới hai chiều có ràng buộc. Bản đồ của trò chơi là một lưới kích thước 8×8, gồm 64 ô. Tại mỗi thời điểm, rắn phải lựa chọn một trong ba hành động: rẽ trái, đi thẳng hoặc rẽ phải. Khi rắn ăn được thức ăn, chiều dài tăng lên và không gian di chuyển còn lại giảm xuống, khiến bài toán ngày càng khó hơn.

Để mô hình hóa trò chơi một cách chặt chẽ, toàn bộ bản đồ được xem như một đồ thị vô hướng, trong đó:

$$G=(V,E)$$

với:

- $V = \{(x,y) \mid 0 \leq x,y < 8\}$: tập các vị trí trên bản đồ
- $E = \{(u,v) \mid u \text{ kề } v\}$: các cạnh nối các ô kề nhau

Khoảng cách giữa hai điểm kề nhau được xem là:

$$d(u,v) = 1$$

Điều này có nghĩa rằng, nếu rắn di chuyển từ ô u sang v liền kề, nó sẽ được coi là đi một bước đơn vị. Việc xác định khoảng cách theo đơn vị này giúp đơn giản hóa việc mô hình hóa các bước di chuyển của rắn trên lưới.

Trạng thái của bản đồ tại thời điểm t được biểu diễn bởi ma trận:

$$M_t \in \{0,1,2,3\}^{8x8}$$

Trong đó, mỗi phần tử của ma trận đại diện cho một loại thông tin cụ thể:

- Giá trị 0: ô trống.
- Giá trị 1: đầu rắn.
- Giá trị 2: thân rắn, các ô chiếm bởi phần còn lại của cơ thể rắn.

- Giá trị 3: thức ăn, vị trí mà rắn có thể ăn để tăng độ dài.

Trong mô hình học sâu, ma trận này có thể được mã hóa thành véc-tơ nhị phân dạng one-hot:

$$S_t \in \{0,1\}^{8x8x4}$$

Mục tiêu của bài toán là lựa chọn chuỗi hành động:

$$A = (a_1, a_2, ...)$$

sao cho rắn có thể di chuyển trên bản đồ mà không va chạm vào tường hay vào chính cơ thể của nó, đồng thời tối đa hóa số lượng thức ăn thu thập được. Mỗi hành động a_t tương ứng với một hướng di chuyển (trên, dưới, trái, phải) tại thời điểm t. Bài toán do đó là một bài toán tối ưu tuần tự, trong đó rắn cần cân nhắc không chỉ hành động hiện tại mà còn tác động của nó đến trạng thái trong tương lai, nhằm tránh các tình huống chết sớm và đạt được điểm số cao nhất.

2.2 Các thuật toán giải quyết bài toán

Dự án sử dụng nhiều phương pháp để điều khiển rắn, từ các thuật toán đồ thị cơ bản đến học tăng cường sâu. Mỗi phương pháp có đặc điểm riêng về tốc độ, độ an toàn và khả năng tối ưu số thức ăn. Dưới đây là mô tả chi tiết từng thuật toán.

2.2.1. Thuật toán BFS - tìm đường ngắn nhất

Breadth-First Search (BFS) là thuật toán cơ bản được sử dụng để giải bài toán đường đi ngắn trên đồ thị lưới để tiến tới mồi trong chi phí bước nhỏ nhất (tối ưu cục bộ về số bước)

Xét môi trường dạng lưới như một đồ thị không trọng số $G = (V, E)$:

- Mỗi ô hợp lệ và một đỉnh $v \in V$
- Hai ô kề nhau theo 4 hướng tạo một cạnh $(u, v) \in E$
- Các ô bị vật cản (tường, phần thân rắn) được loại khỏi tập đỉnh hợp lệ tại thời điểm xét.

Trong hệ thống, bài toán được biểu diễn bằng công thức tối ưu hóa:

$$ShortestPath(s,f) = min\_path \sum_{(u,v)\in path} d(u,v)$$

Trong đó $d(u,v) = 1$ là khoảng cách giữa hai ô liền kề. Do tất cả các cạnh có trọng số bằng nhau, BFS đảm bảo rằng đường đi tìm được luôn là đường ngắn nhất về số bước.

Độ phức tạp tính toán của BFS là $O(|V| + |E|)$, trong lưới 8x8 tương ứng $O(n^2)$ với n=8. Tuy BFS rất nhanh và ổn định, thuật toán này không đảm bảo đường đi an toàn lâu dài. Ví dụ, rắn có thể đi theo đường ngắn nhất nhưng lại vào một khu vực hẹp, dẫn đến khả năng tự khóa sau khi ăn thức ăn. Do đó, mặc dù BFS tối ưu số bước di chuyển, nhưng không tối ưu chiến lược sống còn lâu dài.

2.2.2. Heuristic tìm đường dài nhất

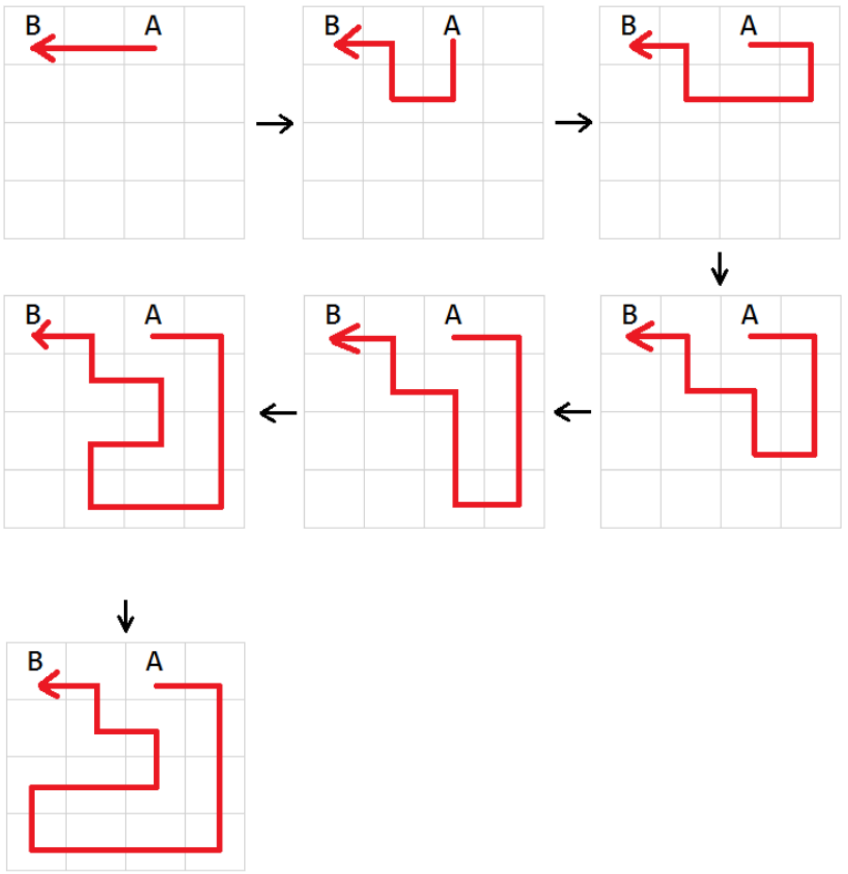
Khác với BFS, bài toán tìm đường dài nhất trong đồ thị tổng quát là NP-hard, nghĩa là không thể tính toán chính xác trong thời gian thực với chi phí thấp. Do đó, hệ thống sử dụng một heuristic để ước lượng đường dài nhất gần tối ưu. Quy trình hoạt động của heuristic bao gồm:

- Sử dụng BFS để tìm đường ngắn nhất P_S từ đầu rắn đến thức ăn.
- Đánh dấu các ô trên đường P_S làm cơ sở.
- "Bẻ" từng đoạn trên đường P_S theo các hướng vuông góc để kéo dài đường đi và tạo ra một đường dài nhất khả thi.

Mục tiêu tối ưu hóa được biểu diễn như sau:

$$LongestPath(s,f) = max\_simplepath \sum_{(u,v)\in path} d(u,v)$$

Heuristic này giúp rắn tránh bị tự nhốt trong các khu vực hẹp sau khi ăn thức ăn, đồng thời vẫn giữ chi phí tính toán thấp, với độ phức tạp thực nghiệm khoảng $O(n^3)$ do số lần thử mở rộng tỉ lệ với độ dài đường và mỗi lần kiểm tra quét lưới. Cụ thể như minh họa sau:



Hình 2.1 minh họa quá trình mở rộng đường ngắn nhất trên bản đồ 4×4

Đường đi ngắn nhất *(Shortest Path)* giữa A và B, là cơ sở để xây dựng. Sau đó, thuật toán sẽ liên tục lặp qua đường đi ngắn nhất này và xét từng cặp ô liên tiếp để chèn thêm các cặp bước đi "lượn" *(zig-zag)* nếu hợp lệ, tạo ra các vòng lặp hẹp giữa các đoạn đường đã có (như hình ảnh thể hiện đường màu đỏ đang uốn lượn để lấp đầy không gian). Mỗi bước mở rộng đều nhằm mục đích kéo dài tổng độ dài đường đi và chỉ được thực hiện nếu con đường mới vẫn an toàn và không đi vào các ô đã được sử dụng. Quá trình này được lặp lại cho đến khi không thể tìm thấy bất kỳ phần mở rộng nào nữa, dẫn đến kết quả là một đường đi dài hơn đáng kể so với ban đầu, giúp bao phủ tối đa không gian trống trong bản đồ hẹp.

2.2.3. Greedy Solver

Greedy Solver là một phương pháp kết hợp giữa BFS và heuristic đường dài để đưa ra quyết định nhanh và hiệu quả. Nguyên tắc hoạt động như sau:

- Nếu tồn tại đường ngắn nhất → kiểm tra xem rắn có còn an toàn hay không.
- Nếu an toàn → rắn đi theo đường ngắn nhất để tiết kiệm bước di chuyển.
- Nếu không an toàn → rắn chọn đường dài nhất nhằm tránh nguy cơ bị khóa trong khu vực hẹp.

Một điểm quan trọng và mới của *Greedy Solver* là mô phỏng trước trạng thái tương lai của rắn *(virtual snake)* trước khi đi. Điều này giúp hệ thống dự đoán khả năng va chạm hoặc tự nhốt sau vài bước di chuyển, từ đó đưa ra hành động an toàn.

Thay vì chỉ xét khoảng cách đến mồi, greedy được mô hình hóa bằng một hàm đánh giá:

$$a^{\backslash *} = \arg\max_{a \in \mathcal{A}(s)} F(s, a)$$

Trong đó F có thể kết hợp nhiều thành phần lý thuyết: khoảng cách tới mồi, khả năng còn đường đi, mức tránh rủi ro, …

Độ phức tạp của Greedy Solver phụ thuộc vào ba bước chính: tìm đường ngắn nhất bằng BFS $O(n^2)$, mô phỏng trạng thái tương lai của rắn với các hướng di chuyển tiềm năng $O(4^k)$ nếu mô phỏng k bước, và áp dụng heuristic đường dài khi đường ngắn nhất không an toàn $O(n^3)$ với lưới nhỏ. Trong thực tế, nhờ giới hạn số bước mô phỏng và loại bỏ sớm các trạng thái nguy hiểm, *Greedy Solver* chạy nhanh, ổn định và cân bằng tốt giữa tốc độ xử lý, độ an toàn và khả năng thu thập thức ăn, phù hợp với đa số tình huống trong trò chơi Snake.

2.2.4. Hamilton Solver

Hamilton Solver sử dụng ý tưởng xây dựng một chu trình Hamilton trên bản đồ - tức là một đường đi đi qua tất cả các ô đúng một lần. Khi rắn đi theo chu trình này, khả năng tự mắc kẹt gần như bằng không. Chu trình được xây dựng theo các bước:

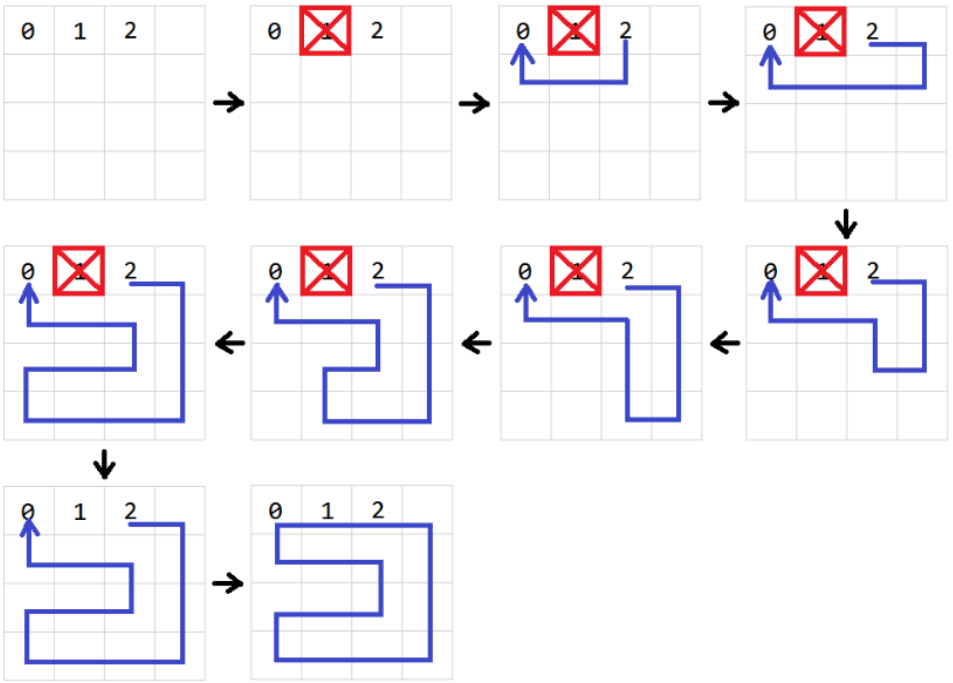
- Cố định một số điểm ban đầu trên lưới.
- Loại bỏ một điểm để buộc thuật toán tạo ra đường dài.

- Ghép đầu và cuối của đường dài lại để tạo thành *chu trình Hamilton* hoàn chỉnh.

Chu trình Hamilton đảm bảo:

$$safe\_path(t)=true \ \forall t$$

Trong đó, *safe_path(t)* đại diện cho đường đi an toàn của rắn tại thời điểm t, true nghĩa là rắn đi theo đường đó mà không nguy cơ va chạm hoặc bị kẹt.



Hình 2.2 Minh họa chu trình Hamilton

Đầu tiên, quy trình xây dựng *Chu trình Hamilton* trên bản đồ 4x4 được thực hiện bằng cách cố định ba điểm ban đầu và vô hiệu hóa một điểm, sau đó tìm đường đi dài nhất từ điểm đầu (2) đến điểm đuôi (0) để lấp đầy không gian còn lại. Cuối cùng, nối hai điểm này để tạo thành một vòng lặp kín đi qua tất cả các ô. Thứ hai, thuật toán Heuristic tìm Đường đi Dài nhất giữa hai điểm A và B được minh họa bằng cách bắt đầu với đường đi ngắn nhất, sau đó lặp lại việc mở rộng đường đi bằng cách chèn các đoạn uốn lượn *(zig-zag)* qua các ô trống để tối đa hóa chiều dài đường đi giữa hai điểm.

2.2.5. Thuật toán DQN (Deep Q-Network)

Deep Q-Network *(DQN)* là một phương pháp học tăng cường sâu kết hợp giữa Q-learning và mạng nơ-ron sâu. Mục tiêu của thuật toán là học được chính sách tối ưu, giúp tác nhân *(agent)* chọn hành động tốt nhất tại mỗi trạng thái để tối đa hóa phần thưởng dài hạn. Đối với các chiến lược phức tạp, hệ thống sử dụng học tăng cường sâu với mạng nơ-ron để ước lượng giá trị hành động Q. Hàm Q được định nghĩa như sau:

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

Hàm mất mát được sử dụng trong quá trình huấn luyện là:

$$L(\theta) = (y - Q(s,a;\theta))^2$$

Trong đó:

- r là phần thưởng nhận được sau hành động.
- γ là hệ số chiết khấu.
- θ là tham số của mạng nơ-ron.
- s và s′ lần lượt là trạng thái hiện tại và trạng thái tiếp theo.
- max_a′ Q(s′,a′) là giá trị Q tối đa có thể đạt được từ trạng thái mới s′, tức hành động tốt nhất rắn có thể thực hiện tiếp theo.

Về lý thuyết, DQN (hàm xấp xỉ phi tuyến + off-policy) không có bảo đảm hội tụ chặt như Q-learning do đó đánh giá hội tụ chủ yếu dựa vào thực nghiệm qua reward/loss/độ dài trung bình. Loss giảm là dấu hiệu tối ưu hoá TD-error tốt hơn, nhưng chỉ được xem là hội tụ khi điểm đánh giá trên episode độc lập ổn định theo thời gian.

Tóm lại, *DQN* giúp *agent* học chính sách tối ưu trong môi trường phức tạp và là một trong những thuật toán chủ lực trong các bài toán học tăng cường hiện đại như Snake, hay robot điều khiển.

CHƯƠNG 3: THIẾT KẾ VÀ TRIỂN KHAI

3.1 Thiết kế thuật toán

Trước khi áp dụng học tăng cường sâu, hệ thống được xây dựng dựa trên các thuật toán cổ điển nhằm đảm bảo khả năng điều khiển rắn ổn định và có tính tham chiếu. Các solver này bao gồm *BFS* để tìm đường ngắn nhất, *Greedy Solver* kết hợp heuristic đường dài để tránh tự khóa, và *Hamilton Solver* sử dụng chu trình Hamilton để đạt độ an toàn tuyệt đối. Việc xây dựng các thuật toán này giúp hệ thống có nền tảng logic vững, dễ kiểm chứng và cho phép đánh giá độ khó của môi trường trước khi huấn luyện mô hình học sâu.

Dựa trên các thuật toán cổ điển đó, hệ thống tiếp tục mở rộng bằng cách tích hợp Deep Q-Network *(DQN)* để rắn có khả năng tự học chiến lược mà không cần quy tắc cố định. Khác với các phương pháp heuristic, *DQN* dùng mạng nơ-ron để dự đoán giá trị Q và liên tục cập nhật mô hình dựa trên kinh nghiệm.

Trong quá trình huấn luyện, *DQN* sử dụng hai dạng thông tin đầu vào: *global state* (ma trận 8×8×4 mô tả toàn bộ bản đồ) và *local state* (ba tín hiệu nguy hiểm phía trước - trái - phải). Mạng học sâu được thiết kế theo kiến trúc *Dueling DQN* kết hợp *CNN*, bao gồm các lớp tích chập trích xuất đặc trưng không gian và hai nhánh *Value* - *Advantage* giúp mô hình ổn định hơn khi dự đoán Q-value.

Bên cạnh kiến trúc mạng, hệ thống sử dụng nhiều siêu tham số quan trọng để điều chỉnh tốc độ học.

Các tham số chính gồm:

- Tần suất cập nhật: cứ 4 bước môi trường mới cập nhật một lần → học chậm nhưng ổn định.
- Replay Buffer: chứa 100.000 mẫu kinh nghiệm và dùng Prioritized Replay để ưu tiên các trạng thái khó.
- Chính sách epsilon-greedy: epsilon giảm từ 1.0 → 0.01, giúp rắn chuyển từ hành động ngẫu nhiên sang ưu tiên những chiến lược mà mô hình đã học được.

- Optimizer: RMSProp (lr = 1e-6, momentum 0.95) để tránh dao động mạnh trong giai đoạn đầu học.

DQN kiểm soát tốc độ học bằng siêu tham số và cơ chế cập nhật chậm để giữ ổn định. Replay Buffer lưu và chọn mẫu ưu tiên giúp mô hình tập trung vào các tình huống quan trọng. Epsilon-greedy giảm dần để chuyển từ thử ngẫu nhiên sang khai thác chiến lược. Mô hình được lưu checkpoint định kỳ để dễ theo dõi và tiếp tục huấn luyện. Nhờ đó, DQN cải thiện hành vi ổn định theo thời gian.

3.2 Cài đặt môi trường phát triển

Trước khi chạy dự án, cần đảm bảo môi trường lập trình đáp ứng các yêu cầu cơ bản và cài đặt đầy đủ các thư viện phụ thuộc.

3.2.1 Yêu cầu hệ thống

Hệ thống yêu cầu các thành phần sau:

- Ngôn ngữ lập trình: Python 3.6 trở lên (Trong trường hợp sử dụng TensorFlow 1.x để train DQN, cần Python 3.7 để đảm bảo tương thích.)
- Giao diện đồ họa (Tkinter): Tkinter thường được tích hợp sẵn trong Python trên Windows và Linux. Thư viện này được dùng để hiển thị trò chơi và quan sát trực tiếp quá trình rắn di chuyển trong chế độ GUI.
- Học sâu: TensorFlow 1.x Nếu sử dụng DQN Solver để huấn luyện mô hình học tăng cường, cần cài đặt TensorFlow 1.x (phiên bản cuối cùng tương thích là TensorFlow 1.15).

3.2.2. Cài đặt thư viện phụ thuộc:

Dự án sử dụng file *requirements.txt* để liệt kê đầy đủ các thư viện cần thiết. Sau khi tải mã nguồn, người dùng chỉ cần mở terminal hoặc PowerShell tại thư mục gốc của dự án và chạy:

```
pip install -r requirements.txt
```

Các gói quan trọng bao gồm:

- matplotlib: dùng để vẽ biểu đồ kết quả huấn luyện và đánh giá hiệu suất.
- pygame hoặc tkinter: hiển thị giao diện người dùng và môi trường mô phỏng.
- pytest: hỗ trợ chạy kiểm thử đơn vị nếu cần.
- numpy: xử lý mảng và phép toán ma trận.
- tensorflow 1.x: dành cho DQN Solver.

Việc cài đặt thông qua *requirements.txt* giúp đảm bảo tính đồng nhất thư viện giữa các máy và tránh xung đột phiên bản.

3.3. Thực hiện

Các thuật toán cổ điển *Hamilton Solver* và *Greedy Solver* là thuật toán xác định và có thể được chạy ngay lập tức ở hai chế độ chính thông qua file run.py.

3.3.1 Chế độ Chơi (Normal Mode)

Chế độ này hiển thị giao diện trò chơi (GUI) và cho phép người dùng quan sát trực tiếp hoạt động.

Đối với thuật toán *Hamilton Solver*, chạy lệnh: `python run.py -s hamilton` để quan sát rắn di chuyển theo Chu trình Hamilton và sử dụng Shortcuts.

Đối với thuật toán *Greedy Solver*, chạy lệnh `python run.py -s greedy` để quan sát rắn thực hiện chiến lược tham lam (tìm đường ngắn nhất/dài nhất).

3.3.2 Chế độ Kiểm định (Benchmark Mode)

Chế độ này chạy thuật toán tự động qua nhiều episode mà không cần GUI để thu thập dữ liệu thống kê khách quan về hiệu suất. `python run.py -s [hamilton/greedy] -m bcmk`

Hệ thống sẽ yêu cầu người dùng nhập số lượng episodes muốn chạy và sau đó in ra kết quả *Average Length* và *Average Steps* cuối cùng.

3.3.3 Quy trình Huấn luyện và Kiểm thử DQN Solver

3.3.3.1 Huấn luyện mô hình

Quá trình huấn luyện DQN được thực hiện bằng lệnh với tham số -m train_dqn.

Lệnh Huấn luyện:

- Không GUI (Tốc độ cao): python run.py -s dqn -m train_dqn
- Có GUI (Theo dõi trực quan): python run.py -s dqn -m train_dqn_gui

Lưu trữ: Trong quá trình huấn luyện, mô hình mạng *(TensorFlow Checkpoint)* và các biến số học tập được tự động lưu trữ định kỳ trong thư mục logs dưới các tên file *solver-net-** và *solver-var-*.json*. Việc này xảy ra mỗi khi mô hình đạt hiệu suất cao nhất hoặc sau mỗi 20,000 bước học.

Khôi phục: Để chạy hoặc tiếp tục huấn luyện từ một mô hình đã lưu, người dùng cần mở file cấu hình DQN *(snake/solver/dqn/__init__.py)* và thiết lập biến *self._restore_step* bằng số bước học của checkpoint muốn tải.

3.3.3.2 Kiểm thử

Sau khi mô hình đã được huấn luyện hoặc khôi phục, có thể tiến hành kiểm thử và đánh giá. Chạy mô hình đã học sau khi thiết lập *_restore_step*, chạy chế độ chơi bình thường để quan sát quá trình đã học: python run.py -s dqn -m normal

Vẽ biểu đồ hiệu suất: Đây là bước quan trọng nhất để đánh giá sự hội tụ và tốc độ học tập. Sử dụng công cụ *plot_dqn_history.py* với phạm vi bước học *(beg_step, end_step)* tương ứng với dữ liệu đã lưu: python tools/plot_dqn_history.py [beg_step] [end_step]

CHƯƠNG 4: KẾT QUẢ VÀ ĐÁNH GIÁ

4.1 Giới thiệu chung

Thông số đánh giá hệ thống:

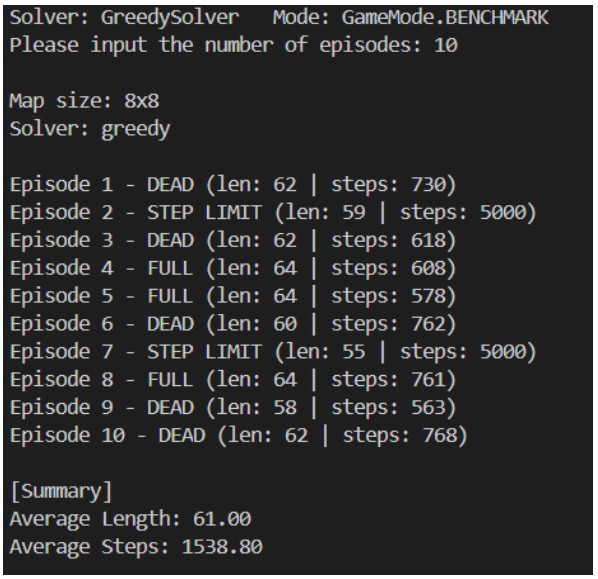
- Snake Length tại episode i: L_i (độ dài cuối episode).
- Snake Steps tại episode i: S_i (tổng số bước sống sót)
- Average Length: $\overline{L} = \frac{1}{K}\sum_{i=1}^{K} L_i$
- Average Steps: $\overline{S} = \frac{1}{K}\sum_{i=1}^{K} S_i$
- Với DQN:
    + Reward trung bình theo bước học (đường Average và dải Min/Max).
    + Loss (TD-error) theo bước học để theo dõi tối ưu hoá.

Mô tả các file kết quả (cho DQN):

- TensorFlow checkpoint: logs/solver-net-* (trọng số mạng tại các mốc).
- Biến/nhật ký huấn luyện: logs/solver-var-*.json (các chuỗi số liệu để vẽ biểu đồ Reward/Loss/Snake Length/Snake Step theo bước học).
- Ngoài ra, kết quả benchmark ở chế độ -m bcmk in ra Average Length và Average Steps sau khi chạy KKK episodes.

4.2 Kết quả Greedy Solver

Hình 4.1 Giao diện Greedy Solver trong quá trình chạy



Hình 4.2 Kết quả Benchmark của Greedy Solver (10 episodes)

Nhận xét:

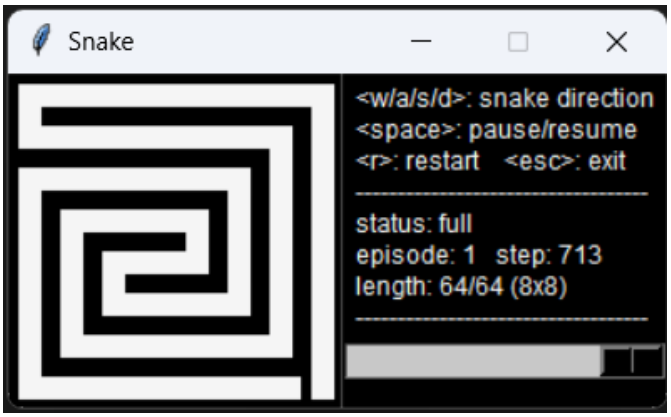
Greedy Solver sử dụng chiến lược "ăn nhanh nhất có thể khi an toàn" bằng cách kết hợp Shortest Path (BFS) và Longest Path heuristic.

Kết quả thực nghiệm (10 episode) cho thấy:

- Greedy đạt độ dài trung bình: 61.0/64, tức chỉ còn 3 ô chưa ăn hết bản đồ.
- Một số episode đạt FULL LENGTH (62–64), chứng tỏ thuật toán hoạt động tốt trong nhiều trường hợp.
- Tuy nhiên, vẫn có episode rơi vào trạng thái DEAD (Episode 1,3,6,9,10), chủ yếu do rắn tự khóa khi khoảng trống hẹp.
- Số bước thực hiện mỗi episode dao động từ ~560 đến ~760, cho thấy rắn thường đi gần optimal nhưng không hoàn toàn tránh được các bẫy cấu trúc.

Đánh giá: Greedy Solver có hiệu suất khá tốt, ăn được trung bình 61/64 thức ăn và phù hợp với bản đồ nhỏ. Tuy nhiên, chiến lược tham lam khiến thuật toán vẫn có nguy cơ rơi vào tình huống tự khóa, đặc biệt khi trạng thái bản đồ trở nên phức tạp.

4.3 Kết quả Hamilton



Hình 4.3 Giao diện Hamilton Solver với chu trình Hamilton

```
Solver: HamiltonSolver   Mode: GameMode.BENCHMARK
Please input the number of episodes: 10

Map size: 8x8
Solver: hamilton

Episode 1 - FULL (len: 64 | steps: 659)
Episode 2 - FULL (len: 64 | steps: 693)
Episode 3 - FULL (len: 64 | steps: 695)
Episode 4 - FULL (len: 64 | steps: 708)
Episode 5 - FULL (len: 64 | steps: 709)
Episode 6 - FULL (len: 64 | steps: 711)
Episode 7 - FULL (len: 64 | steps: 712)
Episode 8 - FULL (len: 64 | steps: 624)
Episode 9 - FULL (len: 64 | steps: 839)
Episode 10 - FULL (len: 64 | steps: 681)

[Summary]
Average Length: 64.00
Average Steps: 703.10
```

Hình 4.4 Kết quả Benchmark của Hamilton Solver (10 episodes)

Nhận xét:

Hamilton Solver sử dụng chu trình Hamilton để bao phủ toàn bộ bản đồ mà không bị bế tắc.

Kết quả 10 episode cho thấy:

- Tất cả episode đều đạt FULL LENGTH 64/64, không có bất kỳ trường hợp DEAD nào.

- Đây là kết quả tối đa có thể đạt được, chứng minh độ an toàn tuyệt đối của thuật toán.
- Số bước trung bình ~703 bước, thấp hơn Greedy một chút do rắn phải đi theo một chu trình cố định trước khi đến vị trí thức ăn.
- Hamilton Solver phù hợp với mục tiêu sống sót tuyệt đối nhưng không tối ưu tốc độ ăn, vì thứ tự đi theo vòng không luôn dẫn tới thức ăn một cách nhanh nhất.

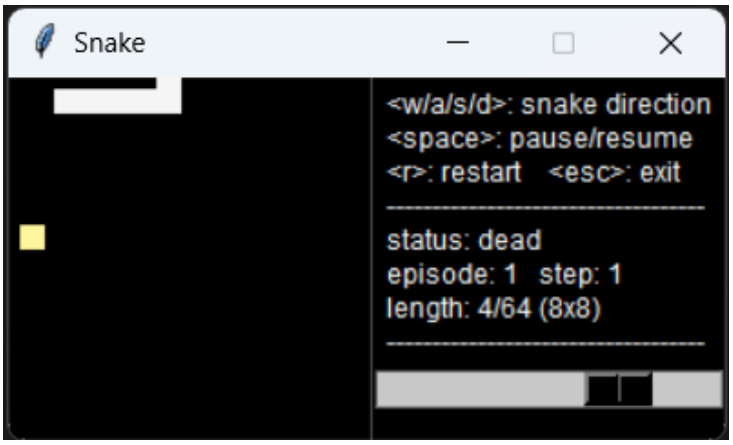
Đánh giá:

Hamilton Solver thể hiện tính ổn định và an toàn cao nhất trong tất cả các solver cổ điển. Thuật toán luôn ăn toàn bộ bản đồ mà không bao giờ tự chết.

Nhược điểm là đường đi dài và không linh hoạt → không tối ưu nếu mục tiêu là thời gian ngắn nhất.

4.4 Kết quả DNQ Solver

Sau khoảng 12 giờ huấn luyện, mô hình DQN mới đạt mức gần 1 triệu bước học (~904.573 bước) với những kết quả thu được sau đây:

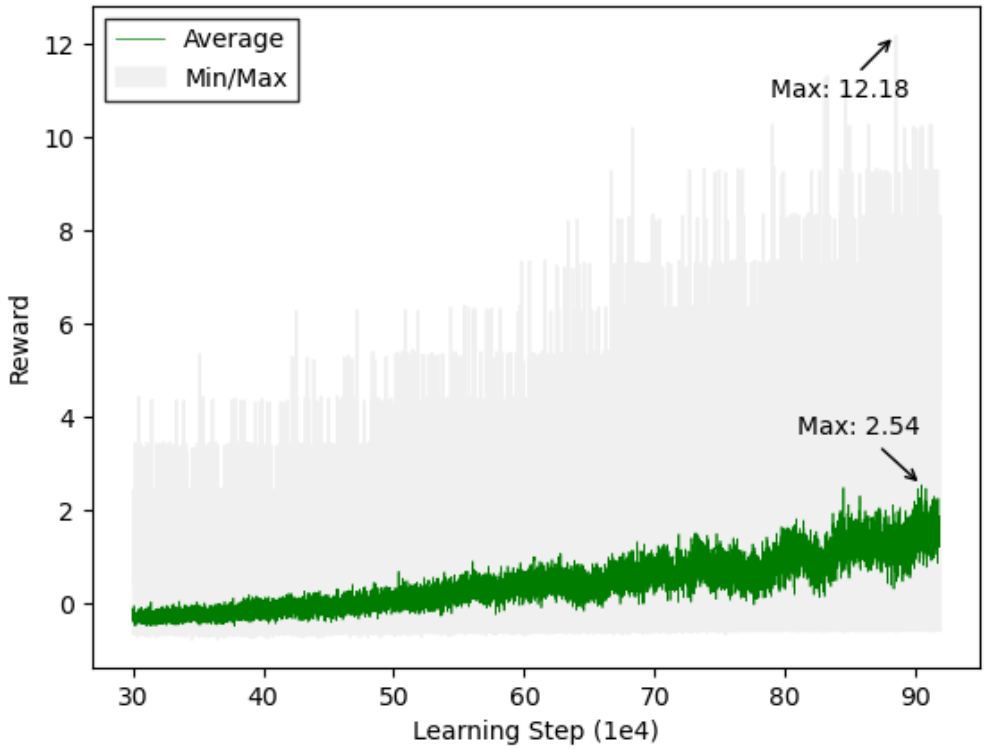


Hình 4.5 Giao diện game với mô hình DQN ở step 904.573

Nhận xét:

Hình 4.5 minh họa trạng thái trò chơi khi mô hình DQN điều khiển rắn. Tại giai đoạn đầu huấn luyện, rắn thường di chuyển thiếu ổn định và dễ va chạm vào tường hoặc

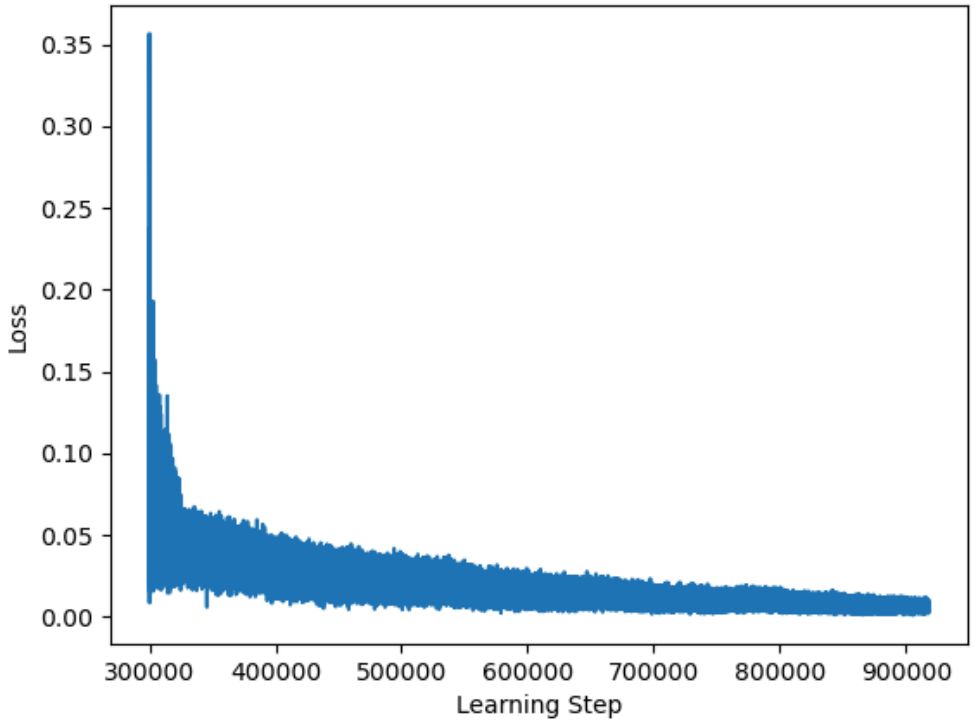
thân, dẫn đến kết thúc sớm như hình minh họa. Điều này phù hợp với đặc tính của DQN: cần trải qua nhiều bước trải nghiệm và cập nhật Q-value trước khi hình thành chiến lược di chuyển hợp lý.



Hình 4.6 Biểu đồ Reward theo bước học (Learning Step)

Nhận xét:

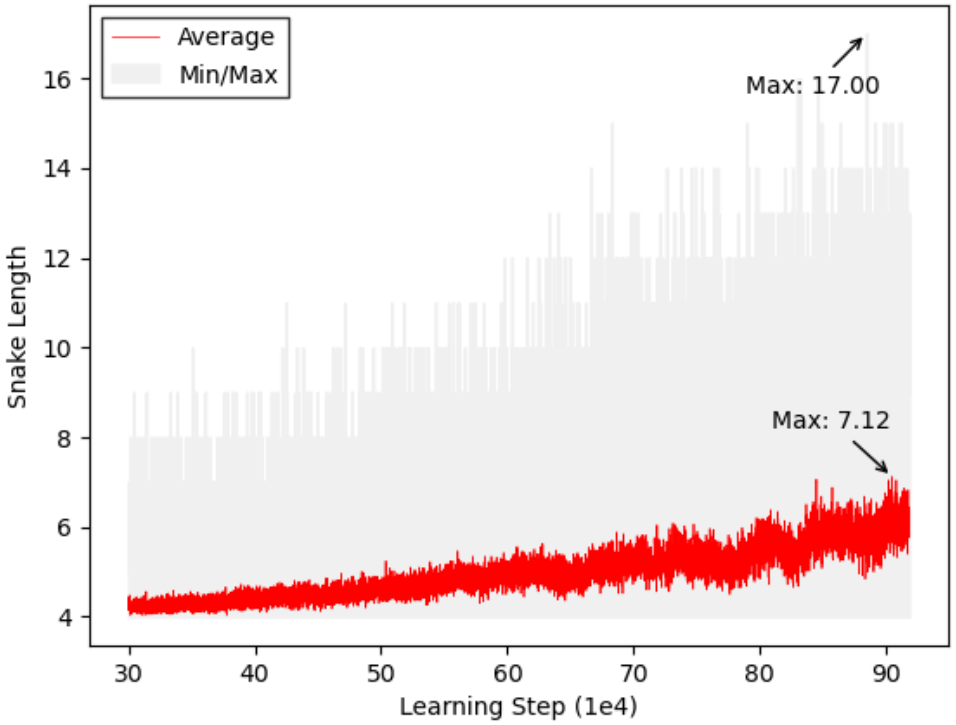
Biểu đồ Reward cho thấy phần thưởng trung bình tăng dần theo thời gian, chứng tỏ mô hình dần học được các hành vi có lợi. Mặc dù độ dao động ban đầu lớn, nhưng càng về sau reward trở nên ổn định hơn, thể hiện quá trình giảm epsilon và tăng mức độ khai thác chiến lược đã học. Khoảng min–max rộng cho thấy trong quá trình học, mô hình thử nghiệm nhiều hành vi khác nhau, đúng với bản chất của chính sách epsilon–greedy.



Hình 4.7 Biểu đồ Loss giảm dần theo quá trình huấn luyện

Nhận xét:

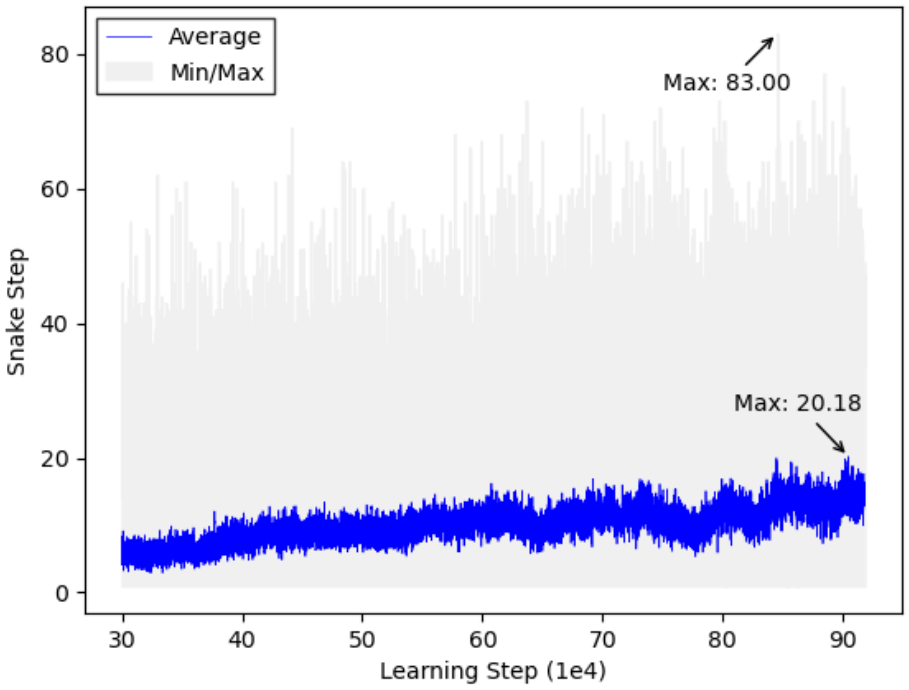
Loss giảm dần đều theo thời gian, thể hiện mạng Q đang hội tụ ổn định. Giai đoạn đầu xuất hiện các giá trị loss lớn do mô hình còn điều chỉnh mạnh theo các sai số TD ban đầu. Khi học tiến triển, loss thu nhỏ dần và dao động quanh mức thấp, chứng tỏ rằng quá trình cập nhật Q-value trở nên chính xác hơn. Đây là dấu hiệu quan trọng cho thấy DQN hoạt động đúng hướng.



Hình 4.8 Biểu đồ độ dài trung bình của rắn (Snake Length) theo bước học

Nhận xét:

Độ dài trung bình của rắn tăng đều theo thời gian học. Ban đầu, rắn chỉ ăn được rất ít thức ăn trước khi va chạm, nhưng càng về sau mô hình dần cải thiện khả năng tìm đường và tránh chướng ngại vật. Mặc dù mức tăng không quá lớn, xu hướng đi lên cho thấy mô hình đang học được kỹ năng cơ bản để kéo dài thời gian sống và tối ưu phần thưởng thông qua việc thu thập thức ăn.



Hình 4.9 Biểu đồ số bước trung bình mà rắn sống sót (Snake Step) theo bước học

Nhận xét:

Số bước sống sót trung bình tăng dần theo quá trình huấn luyện, phản ánh khả năng né tránh và duy trì sự sống được cải thiện. Ở giai đoạn đầu, rắn thường kết thúc sớm do va chạm. Khi mô hình ổn định hơn, số bước trung bình tăng thể hiện rằng rắn đã học được cách điều chỉnh hướng đi an toàn hơn. Đây là dấu hiệu rõ ràng cho việc DQN đang phát triển chiến lược duy trì sự sống trước khi tối ưu việc ăn thức ăn.

Hình 4.10 Giao diện game với mô hình DQN ở step 3.000.0000

Đánh giá chung:

Sau khoảng 12 giờ huấn luyện, mô hình DQN đạt mức gần 1 triệu bước học (~904.573 bước) và ở giai đoạn đó mô hình chỉ học được các kỹ năng cơ bản như tránh tường, né thân và tiếp cận thức ăn theo hướng tương đối an toàn. Mặc dù reward, độ dài rắn và số bước sống sót đều có xu hướng tăng, nhưng chiến lược tổng thể vẫn còn đơn giản và chưa đủ ổn định để xem như hội tụ.

Tuy nhiên, khi tiếp tục huấn luyện đến 3.000.000 bước, mô hình bắt đầu thể hiện mức cải thiện rõ rệt và ổn định hơn. Theo kết quả tổng hợp, DQN đạt:

- Average Length: 24
- Average Steps: 132

Điều này cho thấy mô hình đã học được các chiến lược phức tạp hơn, không chỉ tránh va chạm mà còn chủ động tạo không gian di chuyển, duy trì sự sống lâu hơn và thu thập thức ăn hiệu quả hơn. Độ dài trung bình tăng mạnh từ mức 4–8 (ở giai đoạn dưới 1M bước) lên hơn 24 chứng minh rằng mô hình đã dần học được cấu trúc của môi trường 8×8.

Số bước sống sót cũng tăng đáng kể, phản ánh rằng mô hình không còn di chuyển ngẫu nhiên mà đã hình thành nhận thức môi trường tương đối tốt. Tại 3.000.000 bước, mô hình gần như đã tiến sát giai đoạn hội tụ, vì thành tích trung bình đã ổn định và không còn tăng trưởng mạnh qua các checkpoint sau đó. Mặc dù chưa đạt mức tối ưu của các thuật toán có cấu trúc như Hamilton, mức tăng này cho thấy DQN đang tiến rất gần đến vùng hội tụ ổn định.

4.5 So sánh giữa các thuật toán

Tiêu chí	Greedy Solver	Hamilton Solver	DQN Solver
Hiệu suất trung bình	~61/64 (cao nhưng không ổn định)	64/64 và số bước tối đa (tối ưu tuyệt đối)	24 / 132 (học dần qua thời gian)

Chiến lược di chuyển	Ưu tiên ăn nhanh, dễ kẹt trong vùng hẹp	Di chuyển theo chu trình Hamilton cố định	Học được tránh tường, tìm thức ăn, mở rộng không gian
Mức độ ổn định	Dao động – phụ thuộc hình dạng bản đồ	Rất ổn định – hành vi cố định	Ổn định dần theo quá trình huấn luyện
Yêu cầu tài nguyên	Thấp	Rất thấp	Cao (train nhiều giờ, dùng mạng nơ-ron sâu)

Bảng 4.1 So sánh các thuật toán

Nhận xét:

Hamilton Solver là thuật toán ổn định và an toàn nhất, luôn đạt kết quả tối đa.

Greedy Solver cho hiệu suất tốt nhưng không ổn định và dễ chết vì tự khóa.

DQN Solver học được chiến lược tương đối tốt và có khả năng cải thiện theo thời gian, nhưng đòi hỏi tài nguyên lớn và chưa đạt mức tối ưu tuyệt đối.

KẾT LUẬN

Qua thực nghiệm trên các solver cổ điển, có thể thấy rõ những ưu nhược điểm khác nhau của từng phương pháp. Greedy Solver sử dụng chiến lược "ăn nhanh nhất có thể khi an toàn", kết hợp giữa Shortest Path và Longest Path heuristic. Kết quả thực nghiệm cho thấy rắn đạt trung bình 61/64 thức ăn, với một số episode đạt FULL LENGTH 62–64, chứng tỏ thuật toán hoạt động khá hiệu quả trên bản đồ nhỏ. Tuy nhiên, một số episode vẫn rơi vào trạng thái DEAD, chủ yếu do rắn tự khóa trong những khu vực hẹp, cho thấy chiến lược tham lam vẫn tồn tại rủi ro khi môi trường trở nên phức tạp.

Trong khi đó, Hamilton Solver thể hiện ưu thế về độ an toàn tuyệt đối nhờ sử dụng chu trình Hamilton để bao phủ toàn bộ bản đồ. Tất cả các episode đều đạt FULL LENGTH 64/64 mà không xảy ra trường hợp rắn chết, chứng minh thuật toán cực kỳ ổn định. Mặc dù số bước trung bình của Hamilton Solver thấp hơn Greedy một chút, nhưng đường đi cố định khiến việc thu thập thức ăn không phải lúc nào cũng nhanh nhất. Nhìn chung, Hamilton Solver tối ưu cho việc sống sót và đảm bảo ăn toàn bộ bản đồ, nhưng kém linh hoạt và không tối ưu về tốc độ.

Đối với DQN Solver, kết quả huấn luyện cho thấy mô hình học tăng cường sâu đã cải thiện hành vi rõ rệt theo thời gian. Ở giai đoạn đầu (~900.000 bước), DQN mới chỉ hình thành các kỹ năng cơ bản như tránh tường và tiếp cận thức ăn một cách an toàn. Khi hoàn thành 3.000.000 bước, mô hình đạt Average Length = 24 và Average Steps = 132, phản ánh khả năng di chuyển ổn định hơn và sống sót lâu hơn. Tuy vậy, hiệu suất của DQN vẫn còn thấp hơn đáng kể so với Greedy và Hamilton do yêu cầu dữ liệu lớn, thời gian huấn luyện dài và hạn chế trong việc tối ưu chiến lược trên bản đồ nhỏ dễ bị bế tắc.

Tổng thể, ba phương pháp thể hiện ba hướng tiếp cận khác nhau: Greedy nhanh và hiệu quả nhưng thiếu ổn định; Hamilton an toàn tuyệt đối nhưng cứng nhắc; còn DQN linh hoạt, có khả năng học và thích ứng, nhưng cần được tối ưu thêm để đạt hiệu suất cao hơn.

Về hướng phát triển, có thể nâng cao DQN bằng cách cải thiện kiến trúc mạng nơ-ron, áp dụng các kỹ thuật RL hiện đại như Double DQN, Rainbow DQN hoặc n-step returns; kết hợp chiến lược heuristic (Greedy hoặc Hamilton) để tăng tính an toàn; mở rộng huấn luyện sang các bản đồ lớn hơn; và thiết kế hàm thưởng phù hợp để khuyến khích mô hình thu thập thức ăn hiệu quả và tránh tự khóa. Những hướng này hứa hẹn giúp mô hình đạt hiệu suất ổn định và thông minh hơn trong các môi trường phức tạp.

TÀI LIỆU THAM KHẢO

[1]J. Tapsell, “Nokia 6110 Part 3 – Algorithms,” John Tapsell, Aug. 02, 2020. https://johnflux.com/2015/05/02/nokia-6110-part-3-algorithms/

[2]V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[3]Z. Wang, T. Schaul, M. Hessel, V. H. Hado, M. Lanctot, and D. F. Nando, “Dueling network architectures for deep reinforcement learning,” arXiv.org, Nov. 20, 2015. https://arxiv.org/abs/1511.06581

[4]GeeksforGeeks, “Hamiltonian Cycle,” GeeksforGeeks, Jul. 26, 2025. https://www.geeksforgeeks.org/dsa/hamiltonian-cycle/

[5]C. Quang, “BÀI LUẬN VỀ TRÍ TUỆ NHÂN TẠO: Lịch Sử và Ứng Dụng AI Trong Snake Game,” *Studocu*, Aug. 21, 2025. https://www.studocu.vn/vn/document/dai-hoc-thuy-loi/tri-tue-nhan-tao/bai-luan-ve-tri-tue-nhan-tao-lich-su-va-ung-dung-ai-trong-snake-game/136221045 (accessed Dec. 04, 2025).

[6]trituenhantao.io, “Tạo AI đơn giản cho game rắn với BFS,” *Trí tuệ nhân tạo*, Dec. 22, 2019. https://trituenhantao.io/tao-ai-don-gian-cho-game-ran-voi-bfs/ (accessed Dec. 04, 2025).

PHỤ LỤC

Source code thực hiện: *https://github.com/elsielimmm/snake_game.git*