ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

NHẬP MÔN CÔNG NGHỆ THÔNG TIN

Đề tài:

LẬP TRÌNH GAME CON RẮN BẰNG NGÔN NGỮ PYTHON

Giảng viên hướng dẫn: Lê Đức Khoan

Thành phố Hồ Chí Minh, 12/2025

THÀNH VIÊN NHÓM RẮN ĐỘC

DANH SÁCH THÀNH VIÊN

Võ Ngọc Ánh Linh - 25120202 - **25CTT3B**

Trần Ánh Như - 25120216 - **25CTT3B**

Trần Nguyễn Tấn Phát - 25120218 - **25CTT3B**

Võ Thiên Phúc - 25120222 - **25CTT3B**

Trần Nguyễn Trường Thịnh - 25120235 - **25CTT3B**

Văn Quốc Thịnh - 25120236 - **25CTT3B**

Mục lục

1 Giới thiệu tổng quan

1.1 Mục tiêu của đồ án

- Xây dựng một chương trình Snake game hoàn chỉnh với chế độ một người chơi (Solo leveling), chế độ hai người chơi (Play together và Battle Royale) và chế độ AI chơi (AI Mode) dựa vào thư viện pygame.
- Thiết kế giao diện người dùng (UI) thân thiện, điều khiển bằng chuột.
- Cài đặt các chức năng cốt lõi: di chuyển, ăn mồi, tránh các chướng ngại vật.
- Xử lý điều kiện thắng, thua, hòa.
- Phát triển các tính năng nâng cao:
  - *Lưu và tải lại game (Save/Load):* Cho phép người chơi lưu lại màn chơi và tiếp tục sau đó.
  - *Chơi lại (Play Again):* Tùy chọn chơi màn mới sau khi màn cũ kết thúc.
  - *Cá nhân hóa người chơi:* Tùy chọn đặt tên, đổi avatar.
  - *Âm thanh game:* Hiển thị âm thanh và tùy chọn tăng giảm âm lượng nhạc.
  - *Chế độ AI:* Thiết kế để AI tự chơi đơn.
  - *Chế độ 2 người chơi:* Cho phép 2 người chơi cùng lúc, hỗ trợ lẫn nhau hoặc đấu với nhau (PvP)
- Tổ chức mã nguồn một cách khoa học, dễ hiểu, dễ bảo trì thông qua việc chia thành nhiều file (module hóa).

1.2 Phạm vi và giới hạn của đồ án

- **Nền tảng:** Chương trình chạy trên môi trường Python (sử dụng thư viện Pygame).
- **Chế độ chơi:** Hỗ trợ chế độ chơi đơn, chơi đôi đối kháng, chơi đôi phối hợp và chế độ AI tự động offline.
- **Giao diện:** Giao diện đồ họa được thiết kế và thay đổi với mỗi scene.

1.3 Tính mới đồ án

Đồ án đã làm cho tựa game Rắn săn mồi cổ điển với chỉ một chế độ chơi đơn thành một trò chơi có cả tương tác giữa hai người chơi trên một màn hình và ứng dụng trí tuệ nhân tạo chơi tự động.

- **Cải tiến gameplay:** Ngoài chế độ chơi đơn truyền thống, đồ án đã thêm vào chế độ chơi đôi đối kháng và kết hợp. Chế độ chơi đôi đối kháng yêu cầu người chơi phải cạnh tranh tài nguyên để đạt đến 50 điểm trước và thắng hoặc vận dụng chiến thuật để ép đối phương vào các vật cản. Chế độ đôi kết hợp yêu cầu hai người chơi kết hợp để ăn hết các vật cản của đối phương và phải di chuyển khéo để không chặn đầu đối phương gây thua
- **Tích hợp mô hình Deep Reinforcement Learning - DQN**: AI được xây dựng dựa trên mạng thần kinh nhân tạo (Linear Q-Net) với khả năng phân tích 16 tham số môi trường (vị trí, chướng ngại vật, hướng đi). Thông qua cơ chế Replay Memory (bộ nhớ trải nghiệm) và thuật toán tối ưu hóa Adam, AI có khả năng Self-learning (tự học) từ hàng nghìn lần thử sai để hình thành chiến thuật di chuyển tối ưu mà không cần sự can thiệp của con người

1.4 Công cụ phát triển

- **Ngôn ngữ lập trình:** Ngôn ngữ Python cốt lõi xây dựng toàn bộ logic game và thuật toán AI.
- **Game Engine & Đồ họa:** Thư viện Pygame dùng để render hình ảnh (sprite, UI), xử lý sự kiện đầu vào (bàn phím, chuột) và quản lý vòng lặp trò chơi.
- **Trí tuệ nhân tạo:**
  - **PyTorch:** Framework xây dựng mạng nơ-ron, thực hiện thuật toán Deep Q-Learning (DQN) và quản lý file model.

  - **NumPy:** Xử lý các phép toán ma trận và chuyển đổi trạng thái game thành dữ liệu đầu vào cho mạng nơ-ron.

- **Công cụ hỗ trợ & Tiện ích:**
  - **Matplotlib:** Vẽ biểu đồ theo dõi hiệu suất huấn luyện của AI (Score/Mean Score).
  - **Pathlib & JSON:** Quản lý đường dẫn tập tin tương thích đa nền tảng và lưu trữ dữ liệu thống kê huấn luyện.

- **Kiến trúc phần mềm:** Áp dụng mô hình hướng đối tượng kết hợp State Machine (Máy trạng thái) trong lớp `SnakeApp` để điều phối luồng chạy giữa các màn hình (Intro, Menu, Game, End).

2 Cơ sở lý thuyết

2.1 Giới thiệu về game con Rắn

Trò chơi "Rắn săn mồi" (Snake Game) là một trong những tựa game kinh điển nhất trong lịch sử trò chơi điện tử. Ra đời từ những năm 1970 và trở nên phổ biến toàn cầu nhờ các dòng điện thoại di động cổ điển, trò chơi này có cơ chế đơn giản nhưng đầy cuốn hút.

Cốt lõi của trò chơi là điều khiển một sinh vật giống rắn di chuyển trên một bản đồ giới hạn. Mục tiêu là ăn các vật phẩm (thức ăn) xuất hiện ngẫu nhiên để ghi điểm. Mỗi khi ăn, con rắn sẽ dài ra, làm hẹp không gian di chuyển và tăng độ khó. Trò chơi kết thúc khi đầu rắn va chạm vào tường hoặc tự cắn vào thân mình.

Trong đồ án này, nhóm không chỉ tái hiện lại lối chơi cổ điển mà còn mở rộng và hiện đại hóa trò chơi với các tính năng mới:

- **Đa dạng chế độ chơi:** Ngoài chế độ chơi đơn (Solo Leveling) truyền thống, đồ án tích hợp chế độ chơi đôi phối hợp (Play Together) và đối kháng (Battle Royale), nơi người chơi phải cạnh tranh tài nguyên hoặc hỗ trợ lẫn nhau.

- **Tích hợp AI:** Ứng dụng trí tuệ nhân tạo (Deep Reinforcement Learning) để máy có thể tự học và chơi game một cách tối ưu.

- **Cải tiến đồ họa và âm thanh:** Sử dụng thư viện Pygame để thiết kế giao diện trực quan và hệ thống âm thanh sống động, thay thế cho các khối pixel đơn điệu của phiên bản cổ điển.

2.2 Kiến thức chương trình game (Game loop)

Khác với các phần mềm ứng dụng thông thường, một trò chơi điện tử cần phải liên tục cập nhật trạng thái của thế giới ảo ngay cả khi người chơi không thực hiện hành động nào. Để làm được điều này, trò chơi sử dụng một kiến trúc gọi là Vòng lặp trò chơi (Game Loop).

Game Loop là trái tim của mọi trò chơi, chịu trách nhiệm điều phối toàn bộ hoạt động từ lúc khởi động đến khi kết thúc. Trong đồ án này, thư viện Pygame được sử dụng để quản lý vòng lặp chính nhằm đảm bảo trò chơi vận hành mượt mà. Một vòng lặp game tiêu chuẩn bao gồm ba giai đoạn chính diễn ra liên tục trong mỗi khung hình (frame):

- **Xử lý đầu vào (Input Processing):** Hệ thống liên tục lắng nghe các sự kiện từ thiết bị ngoại vi (bàn phím, chuột). Trong Pygame, các sự kiện này được đưa vào hàng đợi sự kiện (Event Queue).
  - Hệ thống sẽ kiểm tra xem người chơi có nhấn các phím điều hướng (Mũi tên, WASD) hay các phím chức năng (ESC để tạm dừng) hay không.
  - Để tránh xung đột khi người chơi thao tác quá nhanh, các lệnh di chuyển thường được đưa vào một hàng đợi xử lý (Input Queue) thay vì cập nhật tức thời.

- **Cập nhật trạng thái (Update Game Logic):** Đây là giai đoạn tính toán logic của trò chơi dựa trên đầu vào và thời gian trôi qua:
  - Tính toán vị trí mới của đầu rắn dựa trên hướng di chuyển hiện tại.
  - Kiểm tra các điều kiện va chạm vật lý: Rắn có đâm vào tường, vào thân mình hay vào đối thủ không?
  - Kiểm tra tương tác với vật phẩm: Nếu rắn ăn mồi, điểm số tăng lên, rắn dài ra và mồi mới được sinh ra ngẫu nhiên.

  - Cập nhật trạng thái của AI (nếu đang ở chế độ AI Mode) dựa trên dữ liệu môi trường.

- **Kết xuất đồ họa (Render):** Sau khi logic game đã được xử lý, hệ thống sẽ vẽ lại toàn bộ nội dung lên màn hình để hiển thị cho người chơi:
  - Xóa toàn bộ hình ảnh của khung hình trước đó (thường là vẽ đè màu nền).
  - Vẽ lại các đối tượng game ở vị trí mới: Thân rắn (sử dụng Sprite Mapping để vẽ đầu, thân, đuôi và các góc cua), thức ăn, và các chướng ngại vật.
  - Vẽ giao diện người dùng (UI): Điểm số, thời gian, tên người chơi.
  - Cuối cùng, lệnh `pygame.display.flip()` hoặc `update()` được gọi để đẩy toàn bộ hình ảnh đã vẽ từ bộ nhớ đệm ra màn hình hiển thị.

Quá trình này lặp đi lặp lại hàng chục lần mỗi giây (FPS - Frames Per Second), tạo ra ảo giác về sự chuyển động liên tục và mượt mà của trò chơi.

2.3 Quản lý trạng thái màn

Hệ thống quản lý giao diện của trò chơi không sử dụng cách chuyển màn hình tuần tự đơn giản, mà được xây dựng dựa trên mô hình Finite State Machine (FSM - Máy trạng thái hữu hạn). Trung tâm của mô hình này là lớp SnakeApp đóng vai trò như một bộ điều phối.

- Cơ chế chuyển trạng thái: Dựa trên biến current_scene_name, ứng dụng sẽ khởi tạo và chuyển đổi linh hoạt giữa các đối tượng màn hình như Intro, SoloLeveling, hay AIMode. Mỗi lớp scene đều có hàm run() riêng biệt, trả về tên của scene tiếp theo sau khi kết thúc.
- Quản lý vòng đời (Lifecycle): Việc tách biệt từng màn hình thành các lớp (Class) giúp giải phóng tài nguyên của màn hình cũ trước khi tải màn hình mới, từ đó tối ưu hóa bộ nhớ RAM và tránh xung đột biến toàn cục.
- Tính đóng gói (Encapsulation): Mỗi màn hình tự quản lý logic sự kiện và render riêng, giúp mã nguồn dễ bảo trì và mở rộng thêm các chế độ chơi như Battle Royale mà không làm thay đổi cấu trúc của app.py

2.4 Reinforcement Learning

2.5 Cơ sở lý thuyết Reinforcement Learning

Đồ án ứng dụng kỹ thuật **Deep Q-Learning (DQN)**, một phương pháp tiên tiến trong học tăng cường (Reinforcement Learning) kết hợp với mạng nơ-ron sâu để cho phép thực thể (Agent) tự học cách tối ưu hóa hành động thông qua quá trình tương tác với môi trường.

2.5.1 Công thức Bellman và Giá trị Q (Q-Value)

Cốt lõi của thuật toán DQN là tối ưu hóa hàm giá trị $Q(s, a)$, đại diện cho lợi ích kỳ vọng dài hạn khi thực hiện hành động a tại trạng thái s. Trong quá trình huấn luyện, giá trị này được cập nhật liên tục dựa trên phương trình Bellman:

$$Q_{new}(s, a) = R(s, a) + \gamma \cdot \max_{a'} Q(s', a') \tag{1}$$

Trong đó:

- $R(s, a)$: Phần thưởng tức thời nhận được từ môi trường.
- γ (Gamma): Hệ số chiết khấu, được thiết lập là 0.9 để cân bằng giữa lợi ích ngắn hạn và dài hạn.
- $\max_{a'} Q(s', a')$: Giá trị Q lớn nhất dự kiến có thể đạt được ở trạng thái kế tiếp s'.

2.5.2 Kiến trúc Mạng nơ-ron (Linear Q-Net)

Thay vì sử dụng bảng tra cứu truyền thống, hệ thống sử dụng mạng nơ-ron truyền thẳng (Feed-forward Neural Network) để xấp xỉ hàm Q.

- **Lớp đầu vào (Input Layer):** Gồm 16 nơ-ron tương ứng với 16 tham số trạng thái môi trường (bao gồm tầm nhìn 8 hướng, vị trí mồi 4 hướng và hướng di chuyển hiện tại 4 hướng).
- **Lớp ẩn (Hidden Layer):** Gồm 256 nơ-ron sử dụng hàm kích hoạt ReLU (*Rectified Linear Unit*) để xử lý các mối quan hệ phi tuyến tính phức tạp.
- **Lớp đầu ra (Output Layer):** Gồm 4 nơ-ron tương ứng với 4 hành động di chuyển khả thi: Lên, Xuống, Trái, Phải.

2.5.3 Cơ chế Replay Memory và Chiến lược Epsilon-Greedy

- **Replay Memory:** Hệ thống sử dụng một bộ nhớ trải nghiệm với dung lượng $100,000$ đơn vị để lưu trữ các bộ dữ liệu $(state, action, reward, next\_state, done)$. Việc lấy mẫu ngẫu nhiên (*Random Sampling*) từ bộ nhớ này giúp phá vỡ tính tương quan tuần tự, giúp mạng nơ-ron học tập ổn định hơn.
- **Epsilon-Greedy:** Để cân bằng giữa việc khám phá môi trường (*Exploration*) và khai thác tri thức cũ (*Exploitation*), tham số ϵ (Epsilon) được sử dụng để điều chỉnh xác suất chọn hành động ngẫu nhiên, giảm dần theo số lượng trận đấu đã chơi.

2.5.4 Tối ưu hóa và Hàm tổn thất

Quá trình huấn luyện sử dụng thuật toán tối ưu hóa **Adam** với tốc độ học (*Learning Rate*) là 0.001. Hàm tổn thất được sử dụng là sai số bình phương trung bình (**MSELoss - Mean Squared Error Loss**) để cực tiểu hóa sự khác biệt giữa giá trị Q dự đoán và giá trị Q mục tiêu theo công thức Bellman:

$$Loss = \text{MSE}\left(Q_{target} - Q_{predicted}\right) \tag{2}$$

3 Thiết kế hệ thống

3.1 Cách sắp xếp các file và Tổng quan về các file

Để đảm bảo tính khoa học, dễ bảo trì và mở rộng, cấu trúc thư mục của dự án được tổ chức theo mô hình phân tách giữa **Logic (Core)** và **Giao diện (Scenes)**. Mã nguồn được chia nhỏ thành các module chức năng riêng biệt thay vì dồn vào một tệp tin duy nhất.

Cấu trúc cây thư mục của dự án được tổ chức như sau:

```
|-- main.py                 # Entry point: Điểm khởi chạy chương trình
|-- snake/                  # Package mã nguồn chính
    |-- assets/             # Thư mục chứa tài nguyên (ảnh, font, âm thanh)
    |-- app.py              # Class SnakeApp: Quản lý vòng đời và chuyển cảnh
    |-- settings.py         # Các hằng số cấu hình (Màu sắc, kích thước grid)
    |-- save_manager.py     # Module xử lý lưu/tải dữ liệu JSON
    |-- rl/                 # CHỨA MODULE TRÍ TUỆ NHÂN TẠO (AI - Deep Q-Learning)
    |   |-- agent_dqn.py    # Class Agent: Quyết định hành động dựa trên chiến thuật Epsilon-Greedy
    |   |-- dqn_model.py    # Định nghĩa kiến trúc mạng Nơ-ron
    |   |-- memory.py       # Quản lý bộ nhớ Experience Replay
    |   |-- train_dqn.py    # Quy trình huấn luyện
    |   |-- train_graph.py  # Tiện ích vẽ biểu đồ Matplotlib theo dõi hiệu suất
    |-- core/               # CHỨA LOGIC GAME (Model)
    |   |-- env_snake.py    # Logic cho chế độ chơi đơn
    |   |-- env_2p.py       # Logic cho chế độ Play Together
    |   |-- env_2pvp.py     # Logic cho chế độ Battle Royale
    |-- scenes/             # CHỨA GIAO DIỆN HIỂN THỊ (View/Controller)
        |-- board.py        # Lớp cha (Base Class) cho các màn chơi
        |-- intro.py        # Màn hình Menu chính
```

```
|-- play_mode.py     # Màn hình chọn chế độ chơi
|-- solo_leveling.py # Màn chơi đơn
|-- play_together.py # Màn chơi phối hợp
|-- battle_royale.py # Màn chơi đối kháng
|-- ...
```

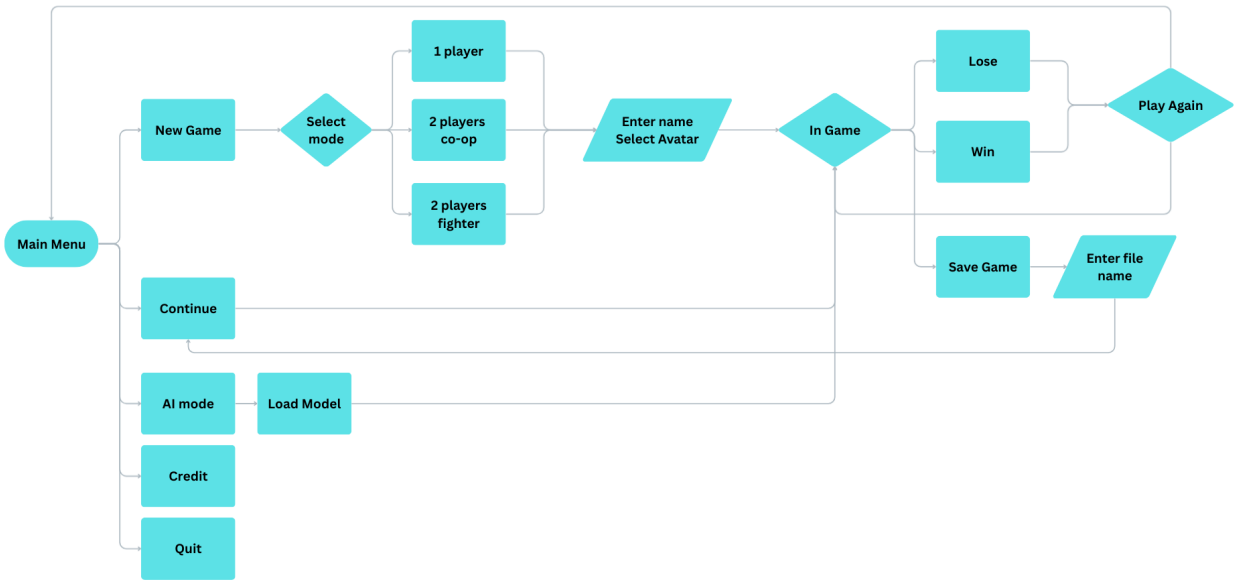
3.1.1 Chi tiết chức năng các tệp tin quan trọng

Bảng 1: Mô tả chức năng các module chính trong dự án

Tên file	**Chức năng nhiệm vụ**
`main.py`	Tệp tin khởi động, chịu trách nhiệm khởi tạo đối tượng `SnakeApp` và kích hoạt vòng lặp chính `run()`.
`snake/app.py`	Đóng vai trò là Director. Class `SnakeApp` khởi tạo cửa sổ Pygame, quản lý biến trạng thái `current_scene_name` để điều phối việc chuyển đổi giữa Intro, Menu và các màn chơi Game.
`core/env_*.py`	Chứa toàn bộ logic toán học của trò chơi. Các file này tính toán vị trí, va chạm, điểm số mà không phụ thuộc vào thư viện đồ họa. Ví dụ: `env_2p.py` xử lý logic cho 2 người chơi phối hợp, `env_2pvp.py` xử lý logic đối kháng.
`rl/*.py`	Chứa các module cốt lõi của hệ thống AI (Deep Q-Learning). Bao gồm: `agent_dqn.py` (Tác tử ra quyết định dựa trên Epsilon-Greedy), `dqn_model.py` (Kiến trúc mạng nơ-ron Linear_QNet), `memory.py` (Lưu trữ Experience Replay) và `train_dqn.py` (Vòng lặp huấn luyện mô hình).
`scenes/board.py`	Lớp cha (Parent Class) chứa các thuộc tính và phương thức chung cho các màn chơi như: load hình ảnh rắn, xử lý hàng đợi nhập liệu (`input_queue`), và vẽ các UI cơ bản (Game Over, Pause).
`scenes/*.py`	Các lớp con kế thừa từ `Board`, chịu trách nhiệm hiển thị hình ảnh cụ thể cho từng chế độ chơi và xử lý input riêng biệt cho từng người chơi (Ví dụ: `PlayTogether` nhận input từ cả cụm phím mũi tên và WASD).

Cách tổ chức này tuân thủ nguyên lý *Separation of Concerns* (Phân tách mối quan tâm): Thư mục `core` chỉ quan tâm đến dữ liệu và logic đúng/sai, trong khi thư mục `scenes` chỉ quan tâm đến việc vẽ hình ảnh và bắt sự kiện người dùng. Điều này giúp nhóm dễ dàng sửa lỗi logic mà không làm hỏng giao diện và ngược lại.

3.2 Sơ đồ hoạt động (Flowchart)



Hình 1: Sơ đồ khối hoạt động của game.

4 Cài đặt chương trình

4.1 Cách con rắn hoạt động

4.1.1 Kiến trúc tổng quan

Hệ thống được xây dựng dựa trên mô hình MVC (Model-View-Controller) tách biệt logic xử lý và giao diện hiển thị, giúp mã nguồn dễ dàng mở rộng sang các chế độ chơi phức tạp như chế độ 2 người kết hợp và đối kháng.

- **Model (Logic):** Các tệp `env_snake.py`, `env_2p.py`, `env_2pvp.py` chịu trách nhiệm tính toán vị trí, va chạm và điểm số.
- **View (Hiển thị):** Các tệp `solo_leveling.py`, `battle_royale.py` chịu trách nhiệm vẽ hình ảnh con rắn lên màn hình dựa trên dữ liệu từ Model.
- **Controller (Điều khiển):** Xử lý sự kiện bàn phím để thay đổi hướng đi trong các vòng lặp game `hàm run`.

4.1.2 Cấu trúc dữ liệu và Trạng thái

Con rắn không được lưu trữ dưới dạng một hình ảnh liền mạch mà là một danh sách liên kết các tọa độ trên lưới ma trận.

- **Biểu diễn:** `snake_pos = [(x_head, y_head), (x_1, y_1), ..., (x_tail, y_tail)]`
- **Nguyên lý:**
  - Phần tử đầu tiên `(index 0)` luôn là Đầu rắn.
  - Phần tử cuối cùng `(index -1)` là Đuôi rắn.
  - Các phần tử ở giữa là Thân rắn.

4.1.3 Thuật toán Di chuyển

Cơ chế di chuyển của rắn được thực hiện trong hàm step() thông qua kỹ thuật "Thêm đầu - Bỏ đuôi", giúp tối ưu hóa hiệu năng tính toán thay vì phải cập nhật lại toàn bộ vị trí các đốt thân. Quy trình xử lý trong mỗi khung hình:

- **Xác định hướng** Hệ thống nhận vector hướng di chuyển hiện tại `(dx, dy)`.
- **Tính toán đầu mới:** `new_head = (old_head_x + dx, old_head_y + dy)`.
- **Cập nhật danh sách:**
  - Chèn `new_head` vào đầu danh sách `snake_pos`.
  - Nếu không ăn mồi: Xóa phần tử cuối cùng (đuôi) để giữ nguyên độ dài.
  - Nếu ăn mồi: Không xóa đuôi, khiến độ dài danh sách tăng lên 1 (Rắn dài ra).

Algorithm 1 Snake Movement & State Update Logic

Require: SnakeList (Coordinate List), Direction (Vector), FoodPos (Coordinate)
Ensure: Updated Snake State, Score, Game Status

procedure UpdateSnake
  Set $Head \leftarrow SnakeList[0]$
  Set $NewHead \leftarrow (Head.x + Direction.x,\ Head.y + Direction.y)$
  ▷ 1. Check Collision Conditions
  if $NewHead$ is Wall **or** $NewHead \in SnakeList$ **then**
    Set $GameOver \leftarrow$ **true**
    return
  end if
  ▷ 2. Move Snake Forward
  Call $SnakeList$.insert(0, $NewHead$)
  ▷ 3. Handle Food Interaction
  if $NewHead == FoodPos$ **then**
    Set $Score \leftarrow Score + 1$
    Call SpawnNewFood()
  else
    Call $SnakeList$.pop() ▷ Remove tail to maintain length
  end if
end procedure

4.1.4 Cơ chế Va chạm và Tương tác

Hệ thống vật lý trong game xử lý ba loại va chạm chính:

- **Va chạm Tường:** Kiểm tra xem tọa độ đầu rắn có nằm ngoài giới hạn lưới `(GRID_WIDTH, GRID_HEIGHT)` hoặc vùng chơi `(PLAY_AREA)` đã định nghĩa trong `settings.py` hay không.
- **Va chạm Thân:** Kiểm tra xem tọa độ `new_head` có trùng với bất kỳ tọa độ nào đang tồn tại trong danh sách `snake_pos` (trừ đuôi) hay không.
- **Tương tác Vật phẩm:**
  - Thức ăn (Food): Tăng điểm số `(score)`, tăng độ dài và kích hoạt sinh thức ăn mới ngẫu nhiên.
  - Chướng ngại vật (Poop): Trong các chế độ đối đối kháng `(env_2pvp.py)`, va chạm với "Poop" sẽ trừ điểm và có thể làm rắn ngắn đi hoặc thua cuộc, còn trong chế độ đối kết hợp `(env2p.py)` thì ngược lại.

Algorithm 2 Battle Royale Collision Handling Logic

Require: $P1\_Pos, P2\_Pos$ (Lists of Snake Body Coordinates)
Ensure: Alive/Dead Status for each Player & Game Winner

procedure CHECKCOLLISION
  for each $Player \in \{P1, P2\}$ **do**
    Set $Head \leftarrow Player.Pos[0]$
    Set $Body \leftarrow Player.Pos[1 \dots N]$ ▷ Remaining body parts
    Set $Opponent \leftarrow$ (The other player)
    ▷ 1. Check Wall Collision
    if $Head$ is outside $GridBoundaries$ **then**
      Set $Player.Alive \leftarrow$ **false**
    end if
    ▷ 2. Check Body Collision (Self & Enemy)
    if $Head \in Body$ **or** $Head \in Opponent.Pos$ **then**
      Set $Player.Alive \leftarrow$ **false**
    end if
  end for
  ▷ 3. Determine Winner based on Survival
  if $P1.Alive ==$ **false and** $P2.Alive ==$ **false then**
    Set $Winner \leftarrow$ "DRAW"
  else if $P1.Alive ==$ **false then**
    Set $Winner \leftarrow$ "Player 2 Wins"
  else if $P2.Alive ==$ **false then**
    Set $Winner \leftarrow$ "Player 1 Wins"
  end if
end procedure

4.1.5 Kỹ thuật Render Đồ họa

Đồ án sử dụng kỹ thuật Sprite Mapping để tạo hình ảnh chân thực. Trong quá trình vẽ (`_draw_elements`), hệ thống duyệt qua từng đốt của con rắn và xác định hình ảnh cần vẽ dựa vào mối quan hệ với đốt trước và đốt sau:

- **Đầu/Đuôi:** Tự động xoay hình ảnh theo hướng di chuyển (Lên, Xuống, Trái, Phải).
- **Thân:**
  - Nếu 3 đốt thẳng hàng: Vẽ thân thẳng (`body_vertical hoặc body_horizontal`).
  - Nếu tạo thành góc vuông: Vẽ các khớp cua (`turn_UL, turn_DR, v.v.`) để tạo cảm giác con rắn uốn lượn mượt mà thay vì gấp khúc vuông vức.

Algorithm 3 Detailed Turn Sprite Selection Logic

Require: Vec_{In} (Vector vào), Vec_{Out} (Vector ra)
Ensure: $SpriteID$ (Tên ảnh góc cua)

procedure SELECTTURNSPRITE
▷ 1. Define Constant Direction Vectors
  Set $UP \leftarrow (0, -1)$
  Set $DOWN \leftarrow (0, 1)$
  Set $LEFT \leftarrow (-1, 0)$
  Set $RIGHT \leftarrow (1, 0)$
▷ 2. Check Vector Pairs
  if ($Vec_{In} == DOWN$ **and** $Vec_{Out} == LEFT$) **or** ($Vec_{In} == RIGHT$ **and** $Vec_{Out} == UP$) **then**
    Set $SpriteID \leftarrow$ "TURN_DL" ▷ Góc cua Dưới-Trái
  else if ($Vec_{In} == DOWN$ **and** $Vec_{Out} == RIGHT$) **or** ($Vec_{In} == LEFT$ **and** $Vec_{Out} == UP$) **then**
    Set $SpriteID \leftarrow$ "TURN_DR" ▷ Góc cua Dưới-Phải
  else if ($Vec_{In} == UP$ **and** $Vec_{Out} == LEFT$) **or** ($Vec_{In} == RIGHT$ **and** $Vec_{Out} == DOWN$) **then**
    Set $SpriteID \leftarrow$ "TURN_UL" ▷ Góc cua Trên-Trái
  else if ($Vec_{In} == UP$ **and** $Vec_{Out} == RIGHT$) **or** ($Vec_{In} == LEFT$ **and** $Vec_{Out} == DOWN$) **then**
    Set $SpriteID \leftarrow$ "TURN_UR" ▷ Góc cua Trên-Phải
  else
    Set $SpriteID \leftarrow$ "ERROR_SPRITE"
  end if
  return $SpriteID$
end procedure

4.2 Môi trường con rắn hoạt động

Trong kiến trúc của đồ án, Môi trường là thành phần thuộc lớp **Model**. Nó chịu trách nhiệm lưu trữ toàn bộ trạng thái của bàn cờ và thực thi các quy tắc vật lý của trò chơi mà không cần biết đến sự tồn tại của giao diện đồ họa.

Các class môi trường như `SnakeEnv2P` (Chơi cùng nhau) hay `SnakeEnv2Pvp` (Đối kháng) hoạt động dựa trên hệ tọa độ lưới (Grid System) với trục x (cột) và trục y (hàng).

4.2.1 Cấu trúc dữ liệu trạng thái

Mỗi môi trường quản lý một tập hợp các biến trạng thái quan trọng tại mỗi thời điểm t:

- **Vị trí rắn (`snake_pos`):** Danh sách các tuple tọa độ $[(x_{head}, y_{head}), (x_1, y_1), ..., (x_{tail}, y_{tail})]$.
- **Hướng di chuyển (`direction`):** Vector chỉ hướng hiện tại, ví dụ $(0, -1)$ là đi lên, $(1, 0)$ là sang phải.
- **Vật phẩm (`food_pos`, `poop_pos`):** Tọa độ của thức ăn và chướng ngại vật trên lưới.
- **Trạng thái game:** Các cờ boolean như `game_over`, `p1_alive`, `winner`.

4.2.2 Thuật toán cập nhật môi trường (Step Function)

Trái tim của môi trường là hàm `step()`. Hàm này nhận vào hành động (hướng đi) của người chơi và trả về trạng thái mới của thế giới. Dưới đây là mã giả mô tả thuật toán xử lý logic trong chế độ chơi 2 người :

Algorithm 4 Logic cập nhật môi trường 2 người chơi (Environment Step)

Require: $Actions\ (D_{P1}, D_{P2})$, $CurrentState$
Ensure: $NewState$, $Rewards$, $GameOver$
procedure STEP(D_{P1}, D_{P2})
▷ 1. Cập nhật Hướng & Vị trí
  for $P, D \in \{(P1, D_{P1}), (P2, D_{P2})\}$ **do**
    if $P.Alive \wedge D \neq null$ **then** $P.Dir \leftarrow D$
    end if
    $P.Pos.insert(0, P.Pos[0] + P.Dir)$ ▷ Thêm đầu mới
  end for
▷ 2. Xử lý Va chạm & Ăn uống
  for $P \in \{P1, P2\}$ **do**
    if not $P.Alive$ **then continue**
    end if
    $H \leftarrow P.Pos[0]$
    $Hit \leftarrow$ CHECKWALL(H) $\vee$ ($H \in P.Body$) $\vee$ ($H \in Opponent.Pos$)
    if Hit **then** $P.Alive \leftarrow$ False ▷ Chết do va chạm
    else if $H == FoodPos$ **then** ▷ Ăn Táo
      $P.Score$ += 1; SPAWNFOOD
    else if $H == PoopPos$ **then** ▷ Ăn Độc
      $P.Score \leftarrow \max(0, P.Score - 2)$; SPAWNPOOP
      $P.Pos.pop()$; $P.Pos.pop()$ ▷ Rắn ngắn đi
      if $len(P.Pos) < 1$ **then** $P.Alive \leftarrow$ False
      end if
    else
      $P.Pos.pop()$ ▷ Di chuyển thường
    end if
  end for
▷ 3. Kiểm tra điều kiện kết thúc
  if not $P1.Alive \wedge$ **not** $P2.Alive$ **then**
    $GO \leftarrow$ True; $Win \leftarrow$ "Draw"
  else if not $P1.Alive$ **then** $GO \leftarrow$ True; $Win \leftarrow$ "P2 Wins"
  else if not $P2.Alive$ **then** $GO \leftarrow$ True; $Win \leftarrow$ "P1 Wins"
  end if
  return GETSTATE
end procedure

Thuật toán trên cho thấy sự phức tạp trong việc xử lý đồng bộ hai thực thể rắn cùng lúc. Đặc biệt, logic ăn "Poop" được thiết kế để trừng phạt người chơi bằng cách trừ điểm và thu nhỏ kích thước rắn, có thể dẫn đến thua cuộc nếu rắn bị thu nhỏ quá mức. Việc tách biệt logic này ra khỏi lớp giao diện giúp AI có thể dễ dàng gọi hàm `step()` để huấn luyện mà không cần hiển thị cửa sổ game.

4.3 Âm thanh

Chức năng âm thanh được xây dựng nhằm nâng cao trải nghiệm của người chơi thông qua việc:

- Cung cấp nhạc nền cho các trạng thái khác nhau của trò chơi.
- Thêm hiệu ứng âm thanh phản hồi cho các thao tác: click chuột, nhập liệu, ăn thức ăn.
- Cho phép điều chỉnh âm lượng nhạc và hiệu ứng một cách độc lập thông qua giao diện Setting.

4.3.1 Thư viện sử dụng

Hệ thống âm thanh được xây dựng dựa trên module pygame.mixer của thư viện Pygame.

Bảng 2: Chi tiết các phương thức Pygame.mixer được sử dụng

Nhóm chức năng	Các phương thức cụ thể
Khởi tạo hệ thống	`pre_init()`, `init()`: Cấu hình và kích hoạt driver âm thanh. `get_init()`: Kiểm tra trạng thái khởi tạo để tránh xung đột.
Nhạc nền	`music.load()`, `music.play()`, `music.stop()`: Điều khiển luồng nhạc. `music.set_volume()`: Chỉnh âm lượng nhạc nền. `music.get_busy()`: Kiểm tra trạng thái phát để xử lý lặp lại.
Hiệu ứng	`Sound()`: Tạo đối tượng âm thanh từ file. `.play()`, `.stop()`: Phát hoặc ngắt hiệu ứng tức thì. `.set_volume()`: Điều chỉnh cường độ âm thanh cho từng hiệu ứng riêng biệt.

4.3.2 Cài đặt lớp quản lý tài nguyên

Hệ thống âm thanh được xây dựng dựa trên mô hình **Centralized Manager** (Quản lý tập trung). Thay vì tải và phát âm thanh rải rác ở từng file, ta sử dụng class **SoundManager** để quản lý tài nguyên, điều chỉnh âm lượng toàn cục và tránh việc tải lại file nhiều lần gây lãng phí bộ nhớ. Để tối ưu hiệu năng, class `SoundManager` được thiết kế theo mô hình Singleton, đảm bảo tài nguyên chỉ được nạp một lần duy nhất. Quy trình khởi tạo hệ thống và nạp tài nguyên được mô tả trong thuật toán 5:

Algorithm 5 SoundManager Initialization & Volume Setup

Require: Asset Files (mp3, wav, ogg)
Ensure: Singleton Instance with Configured Volume
procedure __INIT__
    Call `pygame.mixer.pre_init`(frequency=48000, buffer=1024)
    Call `pygame.mixer.init`()
    Initialize `self.sounds` dictionary
    Call `_load_sound`("eat", "eat.mp3")
    Call `_load_sound`("die", "die.wav")
    Call `_load_sound`("input", "input.ogg")
    Call `_load_sound`("poop", "poop.wav")
    Set `self.vol_sfx` ← 50
    Set `self.vol_music` ← 50
    Call `self.apply_volume`() ▷ Apply initial volume levels
end procedure

Phương thức `apply_volume()` được gọi ngay khi khởi tạo để đảm bảo âm lượng trong game khớp với giá trị cài đặt mặc định (50%). Hàm này sẽ chuẩn hóa giá trị về thang đo 0.0 - 1.0 và áp dụng đồng thời cho nhạc nền lẫn toàn bộ danh sách hiệu ứng âm thanh.

4.3.3 Kĩ thuật xử lý sự kiện âm thanh

Logic phản hồi âm thanh được tích hợp chặt chẽ vào vòng lặp chính (Game Loop) của trò chơi. Thuật toán dưới đây mô tả cách hệ thống bắt sự kiện và phát âm thanh tương ứng:

Algorithm 6 Audio Feedback Logic (Main Game Loop)

Require: Game state (S), User Input (I), SoundManager Instance (SM)
Ensure: Audio Output Signal
Call `SM.play_music("game")` with loop=-1
repeat ▷ Inside `Board.run()`
  if Event I is KeyDown (Arrow/WASD) **then**
    Call `SM.play_sfx`("input")
  end if
  $S, reward, done \leftarrow env.step(direction)$
  if $reward > 0$ **then** ▷ Snake eats food
    Call `SM.play_sfx`("eat")
  else if $reward < 0$ **and not** *done* **then**
    Call `SM.play_sfx`("poop") ▷ Snake eats poison
  else if *done* **is** True **then** ▷ Game Over Condition
    Call `SM.stop_music()`
    Call `SM.play_sfx`("die")
    Break Loop
  end if
until Application Closed

4.4 Lưu và tải file

Tính năng lưu và tải game (Save/Load) được xây dựng nhằm đảm bảo tính liên tục của trò chơi, cho phép người chơi lưu lại thành tích và trạng thái màn chơi để tiếp tục sau đó. Hệ thống hỗ trợ lưu trữ đa dạng dữ liệu bao gồm: tên người chơi, điểm số, vị trí rắn, vị trí mồi và chế độ chơi hiện tại.

4.4.1 Cơ chế lưu trữ và Thư viện sử dụng

Hệ thống sử dụng định dạng **JSON (JavaScript Object Notation)** để tuần tự hóa (serialize) dữ liệu.

- **Thư viện:** Sử dụng `json` để mã hóa/giải mã dữ liệu và `os` để kiểm tra tính toàn vẹn của file.
- **File lưu trữ:** Tất cả dữ liệu được lưu trong một file duy nhất là `save_games.json` dưới dạng một Dictionary, với *key* là tên file save do người dùng đặt và *value* là toàn bộ trạng thái game.

4.4.2 Cài đặt lớp quản lý

Module `save_manager.py` đóng vai trò là lớp trung gian xử lý các tác vụ nhập/xuất (I/O). Các hàm chính bao gồm:

- `save_game(name, state)`: Ghi trạng thái game vào file.
- `load_game(name)`: Đọc và trả về dữ liệu game từ file.
- `get_save_list()`: Lấy danh sách các màn chơi đã lưu để hiển thị lên giao diện người dùng (UI).

4.4.3 Thuật toán Lưu trạng thái game

Quá trình lưu game diễn ra khi người chơi kích hoạt menu Pause và chọn "Save & Quit". Dữ liệu từ lớp `Board` (hoặc các lớp con như `SoloLeveling`, `PlayTogether`, `Battle royale`) sẽ được đóng gói và gửi tới `save_manager`. Quy trình này được mô tả như sau:

Algorithm 7 Game state Serialization & Saving

Require: Game Instance (G), Save Name (N), File Path (F)
Ensure: Data written to JSON storage
procedure SAVE_GAME_PROCESS
  $data \leftarrow$ Dictionary() ▷ Khởi tạo dictionary mới
  $data$["mode"] $\leftarrow G.mode\_id$ ▷ Identify game mode
  $data$["score"] $\leftarrow G.env.score$
  $data$["snake_pos"] $\leftarrow G.env.snake\_pos$
  $data$["direction"] $\leftarrow G.env.direction$
  $data$["food_pos"] $\leftarrow G.env.food\_pos$
  if $G.mode\_id \in$ { "PLAY_TOGETHER", "BATTLE_ROYALE" } **then**
    Append Player 2 data ($p2\_pos, p2\_score, \dots$) **to** $data$
  end if
  SAVE_MANAGER.SAVE_GAME($N, data$)
  try
    Read existing data from F
    Update/Insert $data$ with key N
    Write updated dictionary back to F
  catch IOError
    Display error message
  end try
end procedure

4.4.4 Thuật toán Tải trạng thái game

Quá trình tải game được thực hiện tại `ContinueScene`. Khi người dùng chọn một slot save, hệ thống sẽ đọc dữ liệu, xác định chế độ chơi (Router) và khởi tạo màn chơi tương ứng.

Algorithm 8 Game state Restoration & Routing

Require: Save Name (N), App Instance (A)
Ensure: Game resumed at exact state
procedure LOAD_GAME_PROCESS
  $data \leftarrow$ Call `save_manager.load_game`(N)
  if $rawData$ is Empty **then**
    return
  end if
  $mode \leftarrow data$["mode"]
  if $mode ==$ "SOLO_LEVELING" **then**
    $A.current\_scene \leftarrow$ Init `SoloLeveling`
  else if $mode ==$ "PLAY_TOGETHER" **then**
    $A.current\_scene \leftarrow$ Init `PlayTogether`
  else if $mode ==$ "BATTLE_ROYALE" **then**
    $A.current\_scene \leftarrow$ Init `Battle`
  end if
  Call $A.current\_scene.restore\_game\_state(rawData)$ ▷ Inject data
  $G.env.snake\_pos \leftarrow rawData$["snake_pos"]
  $G.env.score \leftarrow rawData$["score"]
  Set game status to *Playing*
end procedure

4.5 Chọn thông tin người chơi

Để đảm bảo tính linh hoạt cho các chế độ chơi khác nhau (Solo Leveling, Play Together, Battle Royale), hệ thống khởi tạo một màn hình trung gian để thu thập dữ liệu cấu hình. Module này xử lý việc nhập tên, chọn ảnh đại diện và thiết lập độ khó trước khi chuyển giao dữ liệu cho logic trò chơi chính.

Algorithm 9 Player Configuration & Information Setup

Require: $screen, mode$ ▷ Dữ liệu từ Scene trước
Ensure: $mode, nick1, nick2, ava1, ava2, diff$ ▷ Cấu hình hoàn chỉnh

procedure SELECT_INFO_PROCESS
  $nick1, nick2 \leftarrow$ `" "`
  $idx1, idx2 \leftarrow 0, diff \leftarrow$ `NORMAL`
  $active1, active2, running \leftarrow$ `False, False, True`
  while $running$ **do**
    for $event$ **in** PYGAME.EVENT.GET **do**
      if $event.type ==$ `QUIT` **then return** `"QUIT"`
      end if
      ▷ **1. Xử lý nhập liệu Bàn phím**
      if $event.type ==$ `KEYDOWN` **then**
        $target \leftarrow (active1 ? nick1 : (active2 ? nick2 :$ `None`$))$
        if $target \neq$ `None` **then**
          if $event.key ==$ `BACKSPACE` **then**
            $target \leftarrow target[:-1]$
          else if $\text{len}(target) < 15$ **then**
            $target \leftarrow target + event.unicode$
          end if
        end if
      end if
      ▷ **2. Xử lý tương tác Chuột**
      if $event.type ==$ `MOUSEBUTTONDOWN` **then**
        $active1 \leftarrow$ COLLIDE$(pos, Nickname_Box1)$
        $active2 \leftarrow$ COLLIDE$(pos, Nickname_Box2)$
        if $mode ==$ `SOLO` **then**
          $diff \leftarrow$ UPDATEDIFFICULTY(pos) ▷ EASY/NORMAL/HARD
        end if
        ▷ Duyệt Avatar bằng toán tử Modulo
        $idx \leftarrow (idx \pm 1) \pmod{\text{len}(AVATAR\_LIST)}$
        if COLLIDE$(pos, Next_Button)$ **then** $running \leftarrow$ `False`
        end if
      end if
    end for
    RENDERER.DRAW$(states)$ ▷ Cập nhật UI, Placeholder và Hover
  end while
  return $mode, nick1, nick2, \text{List}[idx1], \text{List}[idx2], diff$
end procedure

4.5.1 Chi tiết cài đặt các thành phần

- **Cơ chế nhập liệu và Placeholder:** Hệ thống sử dụng hai biến cờ `nickname_player1_active` và `nickname_player2_active` để xác định tiêu điểm nhập liệu. Nếu người dùng chưa nhập tên và ô không được chọn, hệ thống sẽ hiển thị dòng chữ "Enter Nickname..." với màu sắc mờ (grey) để hướng dẫn. Độ dài tên được giới hạn ở 15 ký tự để đảm bảo tính thẩm mỹ cho UI.
- **Quản lý Avatar và Độ khó:** Việc chuyển đổi ảnh đại diện được thực hiện thông qua chỉ số (*index*) trong danh sách `AVATAR_LIST`, cho phép xoay vòng liên tục nhờ phép toán chia lấy dư (*modulo*). Đối với chế độ Solo Leveling, người chơi có thể chọn một trong ba mức độ khó, mỗi mức độ sẽ thay đổi hình nền tương ứng để phản hồi thị giác ngay lập tức.
- **Hiệu ứng tương tác (Hover):** Hệ thống tích hợp hàm `Hover()` để kiểm tra vị trí chuột thời gian thực. Khi chuột nằm trong vùng `Rect` của các nút "Next" hoặc "Back", hình ảnh nút sẽ được thay thế bằng phiên bản "hover" (sáng hơn) để tăng trải nghiệm người dùng.

4.6 AI mode

Chế độ AI cho phép người dùng quan sát khả năng tự học và xử lý tình huống của "Rắn Độc" thông qua mô hình Deep Q-Learning (DQN). Hệ thống sử dụng một mạng nơ-ron sâu để đưa ra các quyết định tối ưu dựa trên trạng thái môi trường hiện tại.

Algorithm 10 AI Inference & Decision Making Loop

Require: *model_path* ▷ Đường dẫn tệp trọng số .pth
Ensure: *move* ▷ Vector hướng di chuyển tối ưu

procedure AI_Mode_Process
  LoadModel(*model_path*)
  agent.epsilon ← −1 ▷ Tắt tính ngẫu nhiên (Greedy Policy)
  running ← True
  while *running* **do**
    state ← Env.get_state_rl ▷ Lấy trạng thái 16 chiều
    ▷ **1. Dự đoán hành động (Inference)**
    prediction ← Model.forward(*state*)
    action_idx ← Argmax(*prediction*) ▷ Chọn hành động có Q-value cao nhất
    ▷ **2. Chuyển đổi và Kiểm soát hướng**
    directions ← $[(0,-1),(0,1),(-1,0),(1,0)]$
    move ← *directions*[*action_idx*]
    if *move* + Env.current_dir == $\vec{0}$ **then**
      move ← Env.current_dir ▷ Chặn quay đầu 180 độ
    end if
    ▷ **3. Cập nhật môi trường**
    done ← Env.step(*move*)
    if *done* **then**
      Env.reset ▷ Tự động chơi lại sau khi chết
    end if
    Renderer.draw(*env*) ▷ Vẽ giao diện AI_board
  end while
end procedure

4.6.1 Chi tiết kỹ thuật và Kiến trúc mạng

- **Biểu diễn trạng thái (State Representation):** Thay vì sử dụng hình ảnh toàn màn hình, hệ thống sử dụng một vector trạng thái 16 chiều:
  - **Tầm nhìn (8 chiều):** Sử dụng kỹ thuật *Ray-casting* để quét vật cản (thân mình, tường) theo 8 hướng. Giá trị được chuẩn hóa theo công thức $1/(dist+1)$.
  - **Vị trí mồi (4 chiều):** Xác định hướng của mồi tương đối so với đầu rắn (Trái, Phải, Trên, Dưới).

  - **Hướng hiện tại (4 chiều):** Vector One-hot biểu diễn hướng di chuyển hiện tại của thực thể.

- **Mạng nơ-ron Deep Q-Network (DQN):** Sử dụng kiến trúc mạng tuyến tính (`Linear_QNet`) bao gồm 1 lớp đầu vào (16 nodes), 1 lớp ẩn (256 nodes) với hàm kích hoạt ReLU, và 1 lớp đầu ra (4 nodes) tương ứng với 4 hướng di chuyển.
- **Cơ chế nạp tri thức:** Module `AIMode` thực hiện nạp tệp trọng số đã được huấn luyện (`.pth`) thông qua phương thức `load_state_dict`. Chế độ này thiết lập `epsilon = -1` để đảm bảo AI luôn chọn hành động mà nó tin là tốt nhất thay vì thực hiện các bước đi ngẫu nhiên để khám phá như trong quá trình huấn luyện (*Training phase*).
- **Hệ thống phần thưởng (Reward System):** AI được tối ưu hóa dựa trên các chỉ số: Ăn mồi (+20), Va chạm (-150), Dẫm phải phân (-10) và phần thưởng nhỏ khi tiến lại gần mồi (+0.3) giúp định hướng di chuyển thẳng.

4.6.2 Môi trường trò chơi (Snake Environment)

Môi trường đóng vai trò là không gian tương tác của tác tử, chịu trách nhiệm quản lý trạng thái, tính toán phần thưởng và xác định điều kiện kết thúc.

Algorithm 11 Environment Step & Reward Logic

procedure STEP(*action_direction*)
  new_head ← CALCULATEHEAD(*snake_pos*[0], *action_direction*)
  ▷ **Kiểm tra va chạm (Tường, Thân mình)**
  if *new_head* in *snake_pos* **or** ISOUTOFBOUNDS(*new_head*) **then**
    return *state*, −150, True ▷ Phạt nặng khi chết
  end if
  snake_pos.INSERT(0, *new_head*)
  ▷ **Xử lý các sự kiện vật phẩm**
  if *new_head* == *food_pos* **then**
    score ← *score* + 1, *reward* ← 20
    SPAWNFOOD
  else if *new_head* in *poops* **then**
    reward ← −10, *score* ← max(0, *score* − 1)
    SHRINKSNAKE ▷ Rút ngắn thân rắn
  else
    snake_pos.POP ▷ Di chuyển bình thường
    reward ← CALCULATEDISTANCEREWARD(*new_head*, *food_pos*)
  end if
  return GETSTATERL, *reward*, *game_over*
end procedure

4.6.3 Mô hình mạng nơ-ron (DQN Model)

Mô hình `Linear_QNet` sử dụng kiến trúc Deep Learning cơ bản để xấp xỉ hàm giá trị Q (Q-value) cho từng hành động khả thi từ trạng thái đầu vào.

Algorithm 12 Model Forward Pass

procedure FORWARD(x)
  $x \leftarrow \text{LINEAR}(16 \to 256)(x)$ ▷ Lớp ẩn đầu tiên
  $x \leftarrow \text{RELU}(x)$ ▷ Hàm kích hoạt phi tuyến
  $x \leftarrow \text{LINEAR}(256 \to 4)(x)$ ▷ Lớp đầu ra cho 4 hướng
  return x
end procedure

4.6.4 Tác tử học tăng cường (DQNAgent)

Tác tử quản lý tri thức thông qua bộ nhớ đệm (*Replay Memory*) và đưa ra quyết định dựa trên chiến thuật *Epsilon-Greedy*.

Algorithm 13 Action Selection Strategy

```
procedure GET_ACTION(state)
    epsilon ← max(10, 700 − n_games)                ▷ Giảm dần tính khám phá
    if RANDOM(0, 1000) < epsilon then
        move ← RANDOMACTION(0, 3)                   ▷ Khám phá ngẫu nhiên
    else
        state_tensor ← TOTENSOR(state)
        prediction ← MODEL(state_tensor)
        move ← ARGMAX(prediction)                   ▷ Khai thác tri thức
    end if
    return move
end procedure
```

4.6.5 Quy trình huấn luyện (Training Process)

Quy trình huấn luyện lặp đi lặp lại việc thu thập trải nghiệm và tối ưu hóa mạng nơ-ron thông qua phương trình Bellman.

Algorithm 14 Deep Q-Learning Training Loop

```
AGENT.MODEL.LOAD('model.pth')                       ▷ Tải tri thức cũ
while Huấn luyện do
    s ← ENV.GETSTATERL
    a ← AGENT.GETACTION(s)
    s', r, done ← ENV.STEP(a)
    ▷ Học ngắn hạn (Short Memory)
    AGENT.TRAIN_STEP(s, a, r, s', done)
    ▷ Ghi nhớ trải nghiệm
    AGENT.MEMORY.PUSH(s, a, r, s', done)
    if done then
        ENV.RESET
        ▷ Học dài hạn (Experience Replay)
        AGENT.TRAIN_LONG_MEMORY
        if score > record then
            AGENT.MODEL.SAVE('best_model.pth')
        end if
        PLOTPROGRESS(scores, mean_scores)           ▷ Vẽ đồ thị hiệu suất
    end if
end while
```

4.6.6 Đánh giá kỹ thuật các thành phần

- **Cơ chế Reward Shaping:** Hệ thống không chỉ thưởng khi ăn mồi (+20) mà còn cung cấp các phần thưởng trung gian (+0.3 khi lại gần mồi và +0.3 khi đi xa mồi). Điều này giúp giải quyết bài toán "phần thưởng thưa thớt" (*sparse rewards*), giúp AI học được hướng đi đúng đắn nhanh hơn.
- **Kỹ thuật Experience Replay:** Việc sử dụng `ReplayMemory` với kích thước $100,000$ mẫu giúp phá vỡ tính tương quan giữa các mẫu dữ liệu liên tiếp, giúp quá trình huấn luyện mạng nơ-ron ổn định hơn.
- **Tối ưu hóa Bellman:** Trong hàm `train_step`, giá trị Q mục tiêu được cập nhật theo công thức: $Q_{new} = r + \gamma \cdot \max(Q(s'))$, cho phép tác tử dự đoán được lợi ích dài hạn của các hành động hiện tại.

5 Kết quả và thảo luận

5.1 Giao diện người dùng

Giao diện người dùng (UI) trong trò chơi được thiết kế theo hướng tối giản (minimalist) nhưng hiện đại, tập trung vào tính trực quan và khả năng điều hướng mượt mà. Hệ thống UI được chia thành các phân hệ chính sau:

5.1.1 Hệ thống Menu và Điều hướng

Màn hình chính (Intro Scene) và các màn hình chọn chế độ được xây dựng dựa trên thư viện `pygame.Surface` để xử lý hình ảnh và sự kiện chuột khi có tương tác từ người chơi.

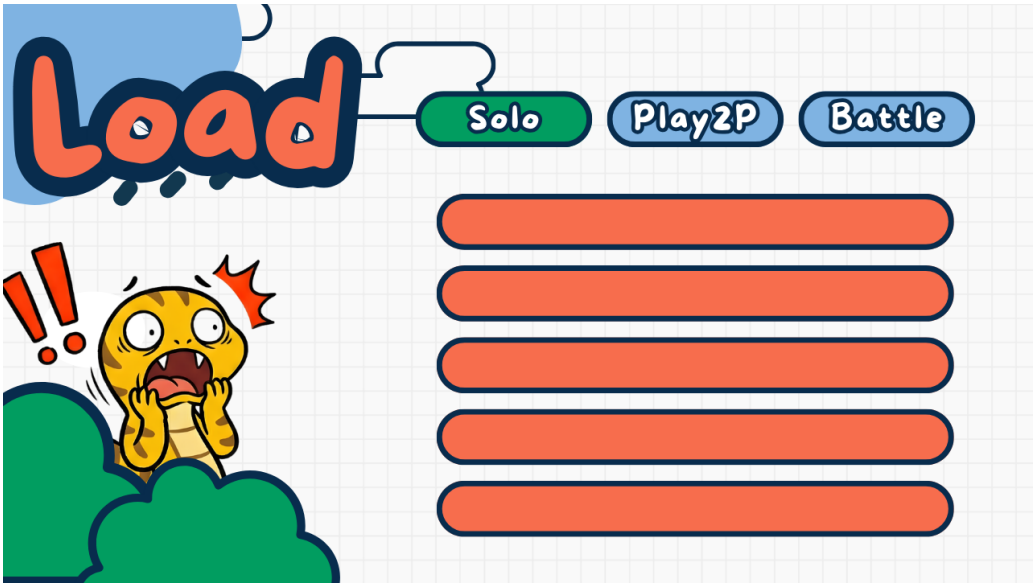
- **Màn hình Chào mừng (Intro):** Tích hợp các nút chức năng lớn: *Play, Continue, AI Mode* với hiệu ứng thay đổi kích thước khi di chuột (Hover Effect) và đổi màu (từ màu xanh dương sang màu cam) giúp người chơi dễ dàng nhận biết tương tác.

Hình 2: Màn hình chính.

- **Màn hình Chọn chế độ (Play Mode):** Sử dụng bố cục phân cấp rõ ràng cho ba chế độ chơi: *Solo Leveling, Play Together* và *Battle Royale*. Mỗi nút được gắn kết với một vùng va chạm (hitbox) riêng biệt và có tính năng đổi màu tương tự như Intro để đảm bảo độ chính xác khi thao tác.

Hình 3: Chọn chế độ

- **Màn hình Tiếp tục (Continue Scene):** Được thiết kế dạng thẻ (Tabs layout), cho phép người chơi lọc các file save theo từng chế độ chơi. Các slot lưu trữ hiển thị thông tin chi tiết (Tên, Điểm số) giúp người chơi dễ dàng lựa chọn.



Hình 4: Giao diện load

  - *Đối với các file trống, game có một hệ thống phản hồi trực quan như sau:* Đầu tiên, sử dụng mã màu để phân biệt slot trống (Màu cam) và slot đã có dữ liệu (Lớp phủ màu vàng). Kế đến là tích hợp hiệu ứng Hover (phóng to vùng chọn) khi người chơi di chuột, giúp xác nhận thao tác chính xác.
  - *Hệ thống quản lý danh sách theo nguyên tắc LifO (Last In, First Out) cục bộ:* File save mới nhất luôn được chèn vào vị trí đầu tiên, tự động đẩy các file cũ xuống dưới và loại bỏ file cũ nhất khi danh sách đã đầy.

5.1.2 Giao diện Tùy chỉnh

Trước khi vào game, người chơi được cung cấp giao diện nhập liệu trực quan để cá nhân hóa trải nghiệm.

- **Nhập liệu:** Hỗ trợ nhập tên người chơi (Nickname) từ bàn phím với giới hạn ký tự để đảm bảo hiển thị đẹp mắt trên bảng điểm.
- **Độ khó:** Hệ thống nút chọn độ khó (*Easy, Normal, Hard*) thay đổi màu sắc trạng thái (Active/Inactive) để phản hồi lựa chọn của người dùng.

Hình 5: Sololeveling

Hình 6: Playtogether và Battleroyale

- **Luật chơi:** Người chơi sẽ được cung cấp một màn hình hướng dẫn nút bấm và cách ghi điểm để tối ưu hóa trải nghiệm tốt nhất.

Hình 7: Rule Sololeveling

Hình 8: Rule Playtogether

Hình 9: Rule Battleroyale

5.1.3 Giao diện Màn chơi

Giao diện trong màn chơi được tối ưu hóa để không che khuất tầm nhìn nhưng vẫn cung cấp đủ thông tin thời gian thực.

- **Thông số HUD:** Điểm số (Score) và Tên người chơi được hiển thị cố định ở góc màn hình.

Hình 10: Hub Sololeveling

Hình 11: Hub Playtogether và Battleroyale

- **Overlay thông báo:** Một lớp phủ bán trong suốt (Overlay) màu đen (‘alpha=180‘) được vẽ đè lên màn hình chơi cho cả 3 chế độ tương ứng với các hiển thị riêng biệt (lấy ví dụ ở chế độ Sololeveling):
  - **Khi nhấn nút ESC:** Tính năng trên có tác dụng làm nổi bật các nút điều hướng như *Resume, Save & Quit* hay *Play Again*, giúp người chơi tập trung vào các lựa chọn tiếp theo.
  - **Khi có người chơi chết:** Tính năng trên có tác dụng làm nổi bật hai nút điều hướng là *Play Again* và *Main Menu* với chức năng tương ứng tên gọi.

5.2 Thao tác bổ trợ

Bên cạnh các cơ chế cốt lõi, các thao tác bổ trợ đóng vai trò quan trọng trong việc duy trì tính ổn định của trò chơi (Game Stability) và nâng cao trải nghiệm người dùng (UX).

5.2.1 Cơ chế Quản lý Luồng

Trò chơi không hoạt động trên một vòng lặp đơn lẻ mà sử dụng mô hình *state Machine* (Máy trạng thái) để quản lý luồng màn hình.

- Lớp `SnakeApp` đóng vai trò là trình quản lý trung tâm (Router). Dựa trên biến trạng thái `current_scene_name`, ứng dụng sẽ khởi tạo và chuyển đổi linh hoạt giữa các đối tượng màn hình khác nhau (`Intro`, `PlayMode`, `Battle`, v.v.) mà không cần khởi động lại ứng dụng.
- Cơ chế này cho phép giải phóng tài nguyên của màn hình cũ trước khi tải màn hình mới, tối ưu hóa bộ nhớ RAM.

5.2.2 Xử lý Tạm dừng và Lưu trữ

Khả năng ngắt quãng và lưu trữ trạng thái là một tính năng nâng cao được tích hợp sâu vào vòng lặp game.

- **Trạng thái Pause:** Khi người chơi nhấn phím `ESC`, biến cờ `is_paused` được kích hoạt. Vòng lặp cập nhật logic game (`_update_game`) bị chặn lại, nhưng vòng lặp vẽ (`_draw_elements`) vẫn hoạt động để hiển thị menu Pause.
- **Hộp thoại Save:** Trong trạng thái Pause, một hộp thoại nhập tên file được kích hoạt. Hệ thống xử lý sự kiện bàn phím riêng biệt để người dùng nhập tên file lưu. Dữ liệu sau đó được đóng gói thành Dictionary và ghi xuống đĩa cứng thông qua `save_manager`.

5.2.3 Hàng đợi Sự kiện Đầu vào

Để khắc phục vấn đề độ trễ hoặc mất thao tác khi người chơi bấm phím quá nhanh, trò chơi sử dụng kỹ thuật *Input Queue*.

- Thay vì cập nhật hướng di chuyển ngay lập tức, các phím bấm được đưa vào danh sách `input_queue`.
- Ở mỗi frame cập nhật game, hệ thống chỉ lấy một thao tác từ hàng đợi để xử lý. Điều này ngăn chặn lỗi logic khi người chơi bấm hai phím hướng ngược nhau trong cùng một frame (ví dụ: đang đi lên, bấm nhanh Trái-Xuống sẽ khiến rắn tự cắn mình nếu không có hàng đợi).

6 Kết luận và hướng phát triển

6.1 Tóm tắt quá trình thực hiện đồ án

Đồ án trò chơi con rắn (Snake Game) của nhóm đã được xây dựng và phát triển hoàn thiện, được lập trình dựa trên ngôn ngữ Python , thư viện Pygame và môi trường WSL . Quá trình thực hiện được chia thành các giai đoạn rõ ràng:

- **Phân tích và thiết kế:** Xác định các yêu cầu chức năng (chế độ chơi đơn, chơi cùng nhau, giao đấu, AI, lưu/tải game) và thiết kế kiến trúc hệ thống theo hướng đối tượng. Nhóm đã xây dựng lớp quản lý chính `SnakeApp` để điều phối các màn hình `Intro`, `SelectInfo`, `Rules` và 3 chế độ chơi chính tách biệt với logic game `(SnakeEnv)`.
- **Xây dựng lõi hệ thống:** Nhóm đã tập trung phát triển logic riêng biệt cho 3 chế độ chơi khác nhau:
  - **Chế độ chơi đơn (Solo Leveling)**: Xây dựng môi trường SnakeEnv với cơ chế rắn phải ăn mồi để tăng điểm nhưng phải tránh các vật thể độc hại (nếu ăn sẽ bị trừ điểm và cắt ngắn thân). Đặc biệt, nhóm đã xử lý hàng đợi đầu vào `(input_queue)` để đảm bảo rắn di chuyển mượt mà, không bị lỗi tự cắn mình khi người chơi thao tác nhanh.
  - **Chế độ chơi cùng nhau(Play Together)**: Xây dựng môi trường `SnakeEnv2P` cho phép hai người chơi cùng lúc trên một bàn phím (Cụm phím Mũi tên và WASD). Chế độ này áp dụng cơ chế mục tiêu bất đối xứng: Player 1 ăn Táo để ghi điểm, trong khi Player 2 ăn "chất thải" để ghi điểm, tạo ra trải nghiệm phối hợp thú vị.
  - **Chế độ giao đấu (Battle Royale)**: Xây dựng môi trường `SnakeEnv2Pvp` tập trung vào tính đối kháng. Logic game được bổ sung các điều kiện thắng/thua cụ thể, đồng thời xử lý va chạm giữa hai con rắn để xác định người chiến thắng.
- **Xây dựng giao diện đồ họa (UI/UX):**
  - **Về đồ họa**: Sử dụng bộ Sprite hình ảnh chi tiết cho từng phần của rắn (đầu, thân, đuôi, các góc cua) giúp hình ảnh hiển thị sinh động hơn so với các khối màu cơ bản.
  - **Về âm thanh**: Xây dựng module `SoundManager` theo mẫu thiết kế Singleton để quản lý tập trung, với nhạc nền thay đổi riêng biệt cho từng chế độ (Menu, In-game, Battle) và các hiệu ứng âm thanh chi tiết cho các sự kiện (ăn mồi, va chạm, thông báo thắng thua). Đồng thời tích hợp giao diện `SettingPopup` để người chơi tùy chỉnh âm lượng trực quan.
- **Hoàn thiện tính năng:** Tích hợp các chức năng phụ trợ quan trọng như hệ thống "Lưu/Tải game" sử dụng định dạng JSON. Xây dựng màn hình `SelectInfo` cho phép chọn Avatar, nhập tên người chơi, và đặc biệt là tùy chọn độ khó (Easy, Normal, Hard) ảnh hưởng trực tiếp đến tốc độ game trong chế độ chơi đơn.
- **Tích hợp Trí tuệ nhân tạo (AI)**: Triển khai thành công mô hình Học tăng cường (Reinforcement Learning) sử dụng thuật toán Deep Q-Network (DQN) trên nền tảng thư viện `PyTorch` và `NumPy`. AI có khả năng tự nhận thức môi trường thông qua kỹ thuật Raycasting để tránh vật cản và tìm đường ăn mồi.

6.2 Hướng phát triển trong tương lai

Mặc dù đồ án đã hoàn thành các mục tiêu đề ra, sản phẩm vẫn có rất nhiều tiềm năng để mở rộng và nâng cấp vào các đợt tiếp theo:

- **Phát triển chế độ chơi qua mạng LAN**: Hiện tại trò chơi hỗ trợ 2 người chơi cục bộ trên cùng một bàn phím. Hướng phát triển tiếp theo là sử dụng kỹ thuật lập trình mạng (Socket) để cho phép hai người chơi kết nối và thi đấu qua mạng nội bộ (LAN).
- **Mở rộng Gameplay**: Tạo thêm cơ chế đặc biệt giúp "hợp thể" hai con rắn thành một thực thể duy nhất và chọn ngẫu nhiên một người chơi giành quyền điều khiển hoặc bổ sung các vật phẩm hỗ trợ (Power-ups) đa dạng như: tăng tốc, đóng băng đối thủ để tăng thêm độ kịch tích cho game.
- **Hệ thống bảng xếp hạng**: Nâng cấp tính năng lưu điểm từ cục bộ (file JSON) để người chơi có thể so sánh với các lần chơi trước đó.

7 Trích dẫn

Đồ án đã sử dụng bộ hình ảnh rắn săn mồi mẫu từ nguồn mở [1] và thay đổi màu. Các hiệu ứng âm thanh trong trò chơi được thu thập từ thư viện trực tuyến MyInstants [2]. Nhạc nền trong Battle được tham khảo từ thư viện âm thanh trên YouTube [3]. Nhạc nền chính của trò chơi và một số hiệu ứng khác được tham khảo từ thư viện âm thanh trên kho lưu trữ cộng đồng OpenGameArt [4].

Tài liệu

[1] Clear_code, “Snake Game Assets.” https://opengameart.org/content/snake-game-assets, 2020. Truy cập ngày: 21/12/2025.

[2] MyInstants, “Sound Effects Category.” https://www.myinstants.com/en/categories/sound Truy cập ngày: 21/12/2025.

[3] Gaming Sound FX, “Cinema Sins Background Song (Clowning Around).” https://www.youtube.com/watch?v=0bI3bd40q6w, 2015. Truy cập ngày: 21/12/2025.

[4] OpenGameArt, “OpenGameArt.org - Free Game Assets.” https://opengameart.org/, 2025. Truy cập ngày: 21/12/2025.