Vietnam National University, Ho Chi Minh City

University of Technology

Faculty of Computer Science and Engineering

Course: Nhập môn trí tuệ nhân tạo (CO3061)

Assignment Report

Bài tập lớn số 2 - GAME PLAYING

Instructor: Vương Bá Thịnh

STT	Họ tên SV	MSSV
1	Lê Nhân Trung	2213689
2	Nguyễn Mạnh Thi	2313205
3	Nguyễn Minh Hiếu	2310959
4	Nguyễn Hoàng Quân	2312834

12th December 2025

Contents

1 Giới thiệu vấn đề

Trò chơi **Cờ Vua** là một trò chơi chiến lược cổ điển có tuổi đời hàng nghìn năm, xuất hiện lần đầu vào khoảng thế kỷ VI tại Ấn Độ. "Cờ Vua" trong tiếng Anh là "Chess", và nhiệm vụ của người chơi là sử dụng các quân cờ để kiểm soát bảng cờ, tấn công vua đối phương trong khi bảo vệ vua của mình. Người chơi có thể di chuyển các quân cờ theo quy tắc cụ thể: vua, hậu, xe, tượng, mã, và tốt. Tuy nhiên, mỗi nước đi đều cần được tính toán kỹ lưỡng, vì chỉ một sai lầm nhỏ cũng có thể dẫn đến thất bại và phải bắt đầu lại từ đầu.

Figure 1: Trò chơi Cờ Vua

Cờ Vua không chỉ là một trò chơi giải trí đơn thuần mà còn là một công cụ tuyệt vời để rèn luyện trí não. Mỗi ván cờ đều đòi hỏi người chơi phải suy nghĩ logic, lên kế hoạch và dự đoán các nước đi tiếp theo của đối phương. Điều này giúp cải thiện khả năng tư duy chiến lược và sự tập trung.
Ngoài ra, Cờ Vua còn mang đến những bài học cuộc sống ý nghĩa. Trò chơi dạy chúng ta rằng đôi khi chỉ cần một chút kiên nhẫn và sự cẩn thận, chúng ta có thể vượt qua mọi thử thách. Đây cũng là lý do tại sao Cờ Vua vẫn giữ được sức hút sau hàng thiên niên kỷ, trở thành một trong những trò chơi chiến lược được yêu thích nhất mọi thời đại.

1.1 Ý nghĩa của Trí tuệ nhân tạo trong Cờ Vua

Trí Tuệ Nhân Tạo (AI) đã mở ra một bước tiến lớn trong việc chơi tự động trò chơi Cờ Vua. Thay vì dựa hoàn toàn vào suy nghĩ của con người, các thuật toán AI có thể được áp dụng để giúp máy tính tự tìm ra nước đi tối ưu. Với các thuật toán như Minimax và học máy, chúng ta có thể tạo ra những "người máy" có khả năng tư duy logic và chơi Cờ Vua một cách hiệu quả.
Các ứng dụng của Trí Tuệ Nhân Tạo trong Cờ Vua bao gồm:

- Tự động hóa việc tìm nước đi tối ưu cho trò chơi Cờ Vua.
- Phát hiện và loại bỏ trạng thái chết (deadlock) giúp tăng hiệu quả chơi.
- Áp dụng hàm heuristic để đánh giá và rút ngắn thời gian tìm kiếm.
- So sánh và đánh giá hiệu suất giữa các thuật toán AI khác nhau, như Minimax và học máy.
- Mở rộng ứng dụng trong lĩnh vực phân tích chiến lược và huấn luyện người chơi.

1.2 Input của bài toán

Trò chơi Cờ Vua được thiết lập đơn giản nhưng để chơi được nó đòi hỏi người chơi phải tính toán kỹ lưỡng từng nước đi hợp lý.
Bản đồ: Trò chơi diễn ra trên một bảng cờ 8x8 ô vuông, trong đó mỗi ô có vai trò cụ thể:

- **Ô trống:** Các ô mà quân cờ có thể di chuyển vào theo quy tắc.
- **Quân cờ đối phương:** Các ô chứa quân cờ của đối thủ, có thể bị ăn.
- **Mục tiêu:** Kiểm soát bảng cờ để chiếu hết vua đối phương.
- **Quân cờ:** Các quân cờ như vua, hậu, xe, tượng, mã, tốt cần được di chuyển chiến lược.

Người chơi: Là bên điều khiển các quân cờ để tấn công và phòng thủ.
Quy tắc di chuyển: Người chơi luân phiên đi, di chuyển quân cờ theo quy tắc riêng (ví dụ: mã đi chữ L, xe đi ngang/dọc); có thể ăn quân đối phương, nhưng không thể di chuyển vào ô bị chiếu hoặc vi phạm luật.
Mục tiêu cuối cùng: Chiếu hết vua đối phương mà không để vua mình bị chiếu hết.

1.3 Các biến thể và mức độ khó trong Cờ Vua

Các biến thể trong Cờ Vua bao gồm các mức độ khó khác nhau, thể hiện sự tiến hóa về độ phức tạp và quy mô của trò chơi:

- **Mức cơ bản:** Bảng cờ tiêu chuẩn với vị trí ban đầu, phù hợp cho người mới bắt đầu.
- **Mức trung cấp:** Tăng độ phức tạp bằng cách thêm các tình huống đặc biệt như nhập thành, bắt tốt qua đường.
- **Mức nâng cao:** Ván cờ với nhiều ràng buộc không gian, đòi hỏi tư duy sâu để giải quyết thế cờ phức tạp.
- **Mức chuyên gia:** Thử thách cao nhất, với độ sâu cây trạng thái từ 30 trở lên và hệ số nhánh cao, đòi hỏi phân tích chiến lược kỹ lưỡng.

Figure 2: Vị trí ban đầu trong Cờ Vua

1.4 Mục tiêu và Yêu cầu Triển khai

Mục tiêu của bài tập lớn này là triển khai agent chơi game cho trò chơi Cờ Vua (dạng đối kháng), nhằm nâng cao kỹ năng lập trình và giải quyết vấn đề. Chúng em chọn Cờ Vua vì cây trạng thái có hệ số nhánh cao và độ sâu từ 30 trở lên.
Yêu cầu cụ thể:

1. Agent phải chơi đúng luật Cờ Vua.
2. Agent sử dụng Minimax thắng được agent chơi ngẫu nhiên với tỉ lệ 90%.
3. Agent sử dụng học máy (ví dụ: học tăng cường) thắng được agent chơi ngẫu nhiên với tỉ lệ 60%.

2 Nội dung cơ sở và Giao diện trực quan game Chess

2.1 Trình soạn thảo LATEX

LaTeX là một hệ thống soạn thảo rất phù hợp cho việc tạo các tài liệu như bài báo, báo cáo, luận văn, sách hoặc bài thuyết trình. Latex cho phép chèn hình ảnh, bảng biểu và công thức toán học vào trong văn bản mà vẫn đảm bảo định dạng trang thống nhất. Các tài liệu được soạn thảo bằng Latex có chất lượng định dạng cao, bố cục đẹp và chất lượng in ấn tốt, đồng nhất, rất thích hợp cho tác phong báo cáo chuyên nghiệp cho bài tập lớn này.

2.2 Ngôn ngữ Python và Python phiên bản 3.13

Python là một ngôn ngữ lập trình mạnh mẽ và linh hoạt, được sử dụng rộng rãi trong xử lý và phân tích dữ liệu. Với nhiều thư viện và khung làm việc mạnh mẽ như *Pandas*, *NumPy* và *Matplotlib*,.. Python cung cấp một môi trường lập trình thuận tiện cho việc thao tác dữ liệu nhanh chóng và hiệu quả.
Phiên bản Python 3.13 mang lại nhiều cải tiến về hiệu năng, khả năng quản lý bộ nhớ, cùng các tính năng mới giúp tối ưu hóa việc viết mã và chạy chương trình. Ngoài ra, Python 3.13 vẫn duy trì khả năng tương thích cao với các thư viện phổ biến, hỗ trợ tốt cho các lĩnh vực như khoa học dữ liệu, trí tuệ nhân tạo và phát triển phần mềm.(chẳng hạn như Pygame chỉ cần Python ở phiên bản ≥ 3.8 nhưng dù vậy bản 3.13 vẫn hoạt động bình thường.)

2.3 Các thư viện sử dụng

```
import chess
import time
import os
import sys
import requests
import threading
import pygame
import numpy
import tqdm
import random
import pandas
import math
```

Sơ lược về các thư viện

Trong bài tập lớn này sẽ có kha khá phục vụ tốt để giải bài toán Sokoban này. Trong đó nhóm em sẽ dùng nhiều tập trung sử dụng các thư viện như đã kể ở trên và dưới đây là phần giới thiệu sơ lược về chúng:

- **chess**: Thư viện *chess* trong Python cung cấp các công cụ để làm việc với trò chơi cờ vua, bao gồm biểu diễn bàn cờ, kiểm tra nước đi hợp lệ, và phân tích vị trí. Trong bài toán Sokoban, nó có thể được sử dụng để mô phỏng các vấn đề tìm kiếm trạng thái tương tự như trong trò chơi chiến lược.

- **time**: Thư viện *time* trong Python cung cấp các hàm liên quan đến việc đo thời gian, tạm dừng chương trình, hoặc lấy thời gian hệ thống hiện tại. Ở đây nhóm dùng để đo thời gian giải thuật chạy cũng như thời gian chơi game

- **os**: Thư viện *os* cung cấp các hàm giúp tương tác với hệ điều hành, như thao tác với tệp tin, thư mục, biến môi trường hoặc lệnh hệ thống. Đây là một thư viện cơ bản nhưng vô cùng quan trọng, thường được sử dụng để quản lý cấu trúc thư mục, kiểm tra đường dẫn, hoặc tự động hóa các tác vụ trong môi trường hệ điều hành. Ở đây *os* giúp cho việc đọc đường dẫn liên kết file thuận tiện, phục vụ việc chia nhỏ code thành các file cho dễ quản lý chức năng.

- **sys**: Thư viện *sys* cung cấp truy cập đến các biến và hàm liên quan đến trình thông dịch Python, như xử lý dòng lệnh, thoát chương trình, hoặc quản lý đường dẫn module. Nó hữu ích trong việc tùy chỉnh hành vi chương trình và xử lý lỗi hệ thống.

- **requests**: Thư viện *requests* cho phép gửi các yêu cầu HTTP một cách đơn giản và dễ đọc trong Python. Nó hỗ trợ các phương thức như GET, POST, và xử lý phản hồi từ máy chủ, thường được dùng để tương tác với API hoặc tải dữ liệu từ web.

- **threading**: Thư viện *threading* hỗ trợ tạo và quản lý các luồng (threads) trong Python, cho phép chạy nhiều tác vụ song song để cải thiện hiệu suất, đặc biệt trong các ứng dụng I/O-bound hoặc mô phỏng đa nhiệm.

- **pygame**: *Pygame* là một thư viện mạnh mẽ hỗ trợ phát triển trò chơi 2D và các ứng dụng tương tác. Thư viện này cung cấp các công cụ để xử lý âm thanh, hình ảnh, sự kiện bàn phím, chuột và hiển thị đồ họa trên màn hình. Nhờ đó, nhóm dùng Pygame để để visualize bài toán thành một game cổ điển để có thể giải trí.

- **numpy**: *NumPy* là thư viện nền tảng cho tính toán khoa học trong Python. Nó hỗ trợ thao tác trên mảng đa chiều (ndarray) và cung cấp hàng trăm hàm toán học tối ưu hóa cho xử lý số liệu. Nhờ khả năng tính toán nhanh chóng và hiệu quả, *NumPy* thường được sử dụng trong các bài toán ma trận, đại số tuyến tính và xử lý dữ liệu lớn.

- **tqdm**: Thư viện *tqdm* cung cấp thanh tiến trình (progress bar) đơn giản và linh hoạt cho các vòng lặp trong Python, giúp theo dõi tiến độ thực thi của các tác vụ dài, như huấn luyện mô hình hoặc xử lý dữ liệu lớn.

- **random**: Thư viện *random* cung cấp các hàm để tạo số ngẫu nhiên, chọn ngẫu nhiên từ danh sách, hoặc xáo trộn dữ liệu. Nó thường được sử dụng trong mô phỏng, trò chơi, hoặc các thuật toán ngẫu nhiên hóa.

- **pandas**: Thư viện *pandas* là công cụ mạnh mẽ cho phân tích và thao tác dữ liệu trong Python, hỗ trợ cấu trúc dữ liệu như DataFrame và Series. Nó giúp dễ dàng đọc, viết, lọc, và tổng hợp dữ liệu từ nhiều nguồn như CSV, Excel.

- **math**: Thư viện *math* cung cấp các hàm toán học cơ bản như hàm lượng giác, logarit, căn bậc hai, và hằng số toán học. Nó được sử dụng để thực hiện các phép tính số học chính xác và hiệu quả trong chương trình.

2.4 Giao diện trò chơi của nhóm và hướng dẫn cơ bản thao tác

Giao diện trò chơi

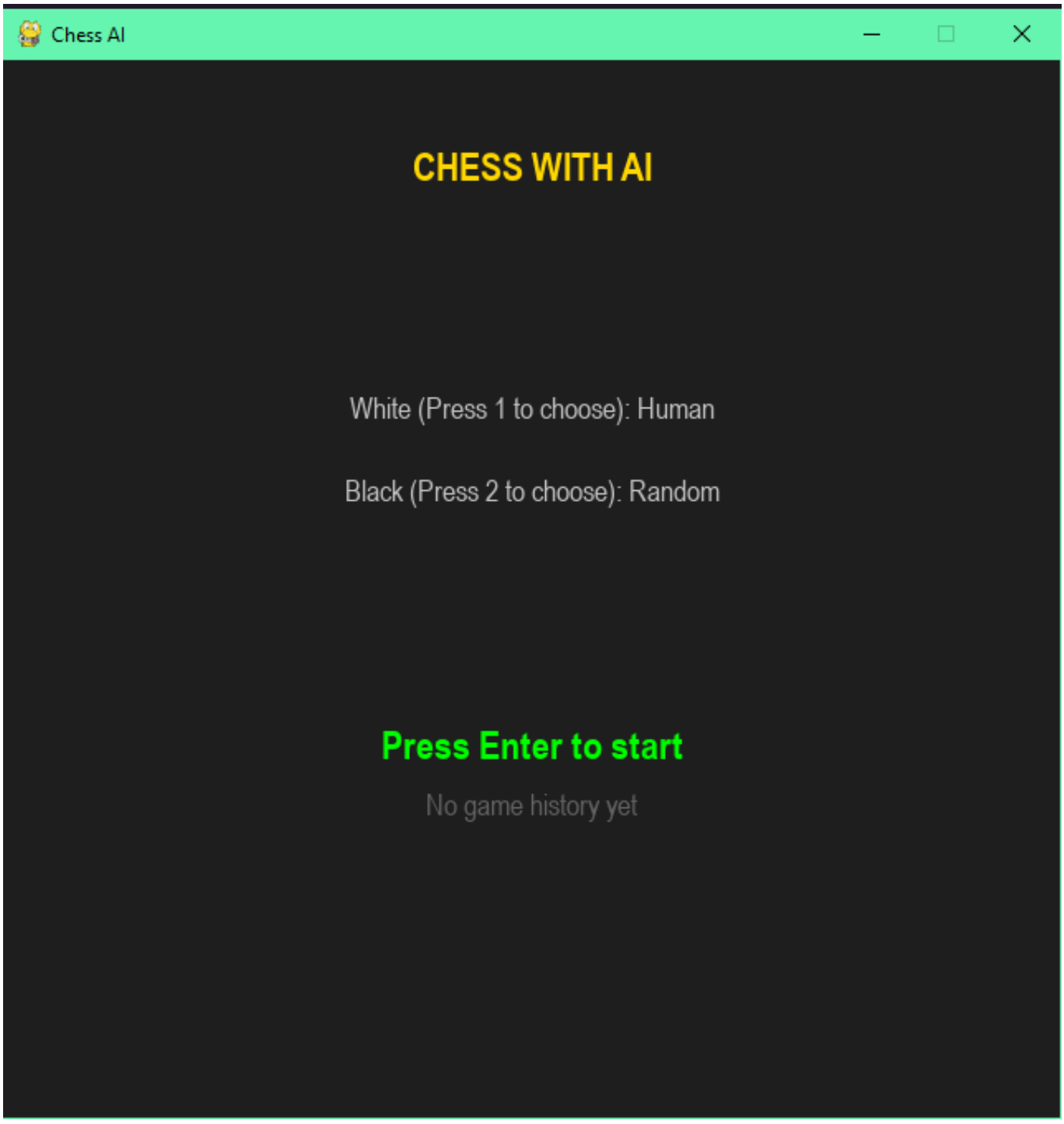

Figure 3: Giao diện vào game

Figure 4: Giao diện Mô cho giải thuật chạy

Figure 5: Giao diện khi player muốn tự chơi

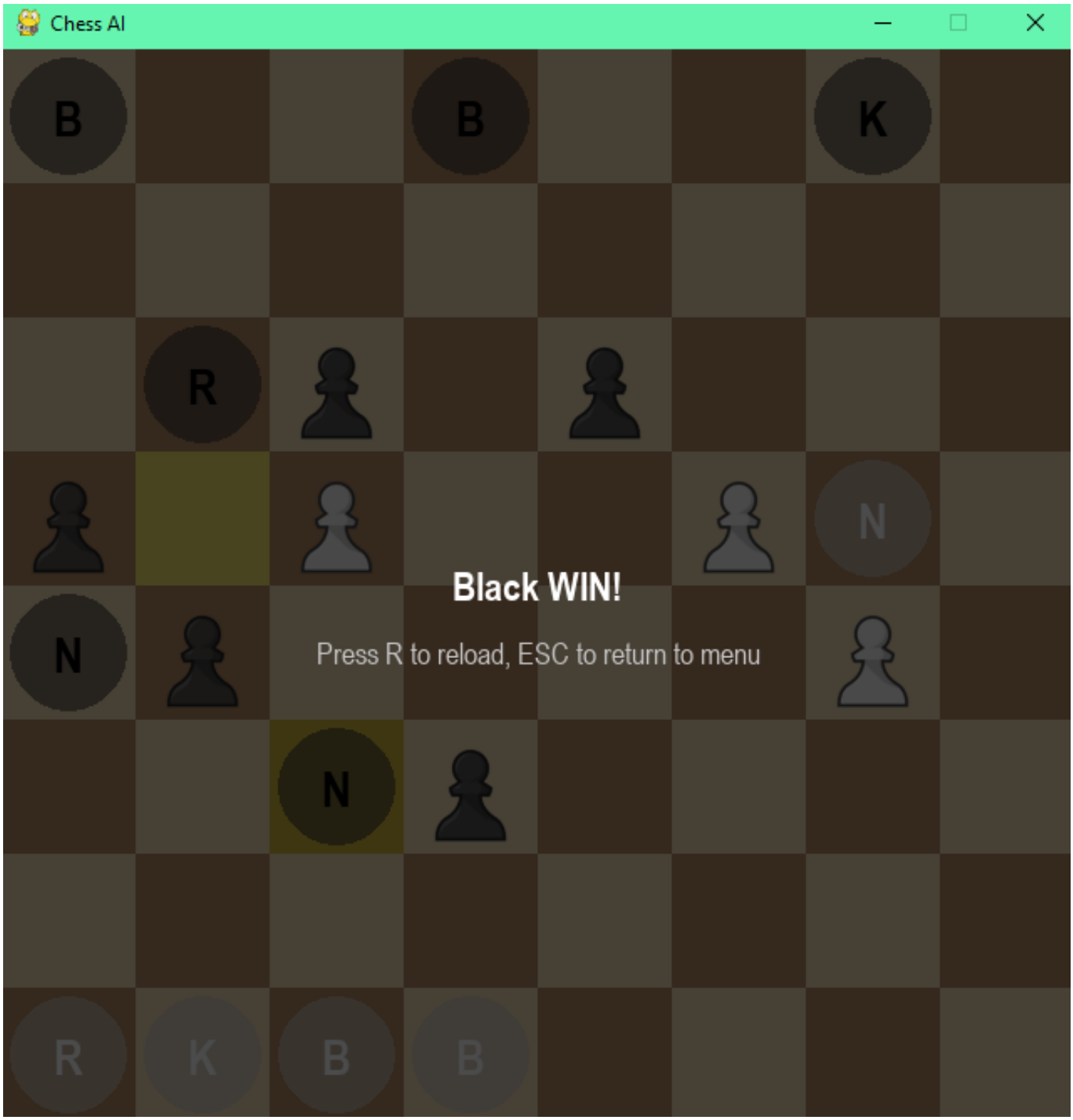

Figure 6: Giao diện khi agent tự chơi và hoàn thành

Các thao tác để chơi

- **Click chuột**: Khi vào chế độ Human, người chơi buộc phải click vào các quân bên mình để di chuyển.
- **Phím R**:dùng để reload màn chơi.
- **Các phím số 1,** : Dùng để chọn chế độ chơi giữa 2 bên, 1 là bên trắng và 2 là bên đen. Ở

đây, nhóm 4 tác nhân chơi là Human, Random, Minimax và MLP.

- **Phím Esc**: Dùng để thoát màn chơi, trở lại giao diện sảnh chờ nhưng không thoát game.

3 Giải thuật và Phương pháp Hiện thực

Để giải quyết bài toán Game Playing cờ vua, nhóm tiếp cận vấn đề theo hai hướng bổ trợ: phương pháp tìm kiếm cổ điển dựa trên lý thuyết trò chơi (*Game Theoretic Search*) và phương pháp xấp xỉ hàm đánh giá dựa trên dữ liệu (*Data-driven Function Approximation*).

3.1 Giải thuật Minimax và Hàm Đánh giá Tapered Evaluation

3.1.1 Cơ sở lý thuyết Minimax

Cờ vua là một trò chơi đối kháng có tổng bằng không (*Zero-sum game*) với thông tin hoàn hảo. Không gian trạng thái của cờ vua ước tính khoảng 10^{40} trạng thái hợp lệ (Shannon number), do đó việc duyệt toàn bộ cây trò chơi là bất khả thi.
Chúng tôi sử dụng giải thuật **Minimax** với độ sâu cố định d để tìm kiếm nước đi tối ưu cục bộ. Tại một trạng thái s, giá trị hữu dụng $V(s)$ được định nghĩa đệ quy:

$$V(s,d) = \begin{cases} \mathcal{H}(s) & \text{nếu } d = 0 \text{ hoặc } s \in \mathcal{T} \text{ (tập trạng thái kết thúc)} \\ \max_{a \in \mathcal{A}(s)} V(\text{Successor}(s,a), d-1) & \text{nếu Player}(s) = \text{MAX} \\ \min_{a \in \mathcal{A}(s)} V(\text{Successor}(s,a), d-1) & \text{nếu Player}(s) = \text{MIN} \end{cases} \tag{1}$$

Trong đó $\mathcal{H}(s)$ là hàm đánh giá heuristic tại nút lá.

3.1.2 Tối ưu hoá Alpha-Beta Pruning

Để tăng hiệu suất tìm kiếm, kỹ thuật cắt tỉa Alpha-Beta được áp dụng. Giải thuật duy trì hai tham số trong quá trình duyệt sâu (Depth-First Search):

- α: Giá trị lớn nhất mà phe MAX đã đảm bảo được (Lower bound).
- β: Giá trị nhỏ nhất mà phe MIN đã đảm bảo được (Upper bound).

Tại một nút MIN, nếu giá trị tạm thời $v \leq \alpha$, phe MAX ở nút cha sẽ không bao giờ chọn nhánh này, do đó quá trình tìm kiếm tại nhánh con bị ngắt (pruned). Tương tự cho nút MAX với β. Điều này giúp giảm hệ số nhánh hiệu dụng từ b xuống xấp xỉ $\sqrt{b}$ trong trường hợp tốt nhất.

3.1.3 Hàm Đánh giá Nội suy (Tapered Evaluation)

Một thách thức lớn trong lập trình cờ vua là giá trị quân cờ thay đổi theo thời gian (ví dụ: Vua cần an toàn ở đầu trận nhưng cần linh hoạt ở cuối trận). Sử dụng một hàm đánh giá tĩnh duy nhất sẽ gây ra sai số lớn. Nhóm đề xuất sử dụng kỹ thuật **Tapered Evaluation** để nội suy tuyến tính giữa hai hàm đánh giá chuyên biệt: Trung cuộc (E_{MG}) và Tàn cuộc (E_{EG}).

a. Định lượng Giai đoạn (Game Phase Quantification) Giai đoạn trận đấu ϕ được tính toán dựa trên tổng "lực lượng" của các quân mạnh (*Major & Minor pieces*) còn lại trên bàn cờ. Trọng số w_p cho từng loại quân được xác định thực nghiệm:

$$w_{Knight} = 1, \quad w_{Bishop} = 1, \quad w_{Rook} = 2, \quad w_{Queen} = 4$$

Tổng pha hiện tại ϕ_{curr} và giới hạn pha $\Phi_{total} = 24$ (tương ứng đội hình xuất phát đầy đủ):

$$\phi_{curr} = \min\left(\sum_{p \in Board} w_p, \Phi_{total}\right) \tag{2}$$

b. Bảng điểm Vị trí (Piece-Square Tables - PST) Mỗi loại quân k được gán hai ma trận giá trị vị trí PST^k_{MG} và PST^k_{EG} kích thước 8×8. Giá trị của quân cờ tại ô sq được tính:

$$Val(k, sq, \text{phase}) = \text{Material}(k, \text{phase}) + PST^k_{\text{phase}}[sq]$$

Đối với quân Đen, chỉ số ô được ánh xạ qua trục ngang (Mirroring) để tận dụng lại bảng điểm của Trắng: $sq' = \text{sq} \oplus 56$.

c. Công thức Hợp nhất Điểm số cuối cùng $\mathcal{H}(s)$ là tổ hợp lồi của hai giai đoạn:

$$\mathcal{H}(s) = \frac{E_{MG}(s) \cdot \phi_{curr} + E_{EG}(s) \cdot (\Phi_{total} - \phi_{curr})}{\Phi_{total}} \tag{3}$$

Phương pháp này loại bỏ hoàn toàn hiện tượng "Horizon Effect" khi chuyển giai đoạn, giúp Agent chơi mượt mà từ khai cuộc đến cờ tàn.

3.2 Agent Mạng Nơ-ron (MLP Agent)

Khác với Minimax dựa trên tìm kiếm vét cạn, MLP Agent ra quyết định dựa trên xấp xỉ hàm giá trị $V(s) \approx f_\theta(s)$ thông qua một mạng nơ-ron lan truyền thẳng (*Feed-Forward Neural Network*).

3.2.1 Kiến trúc Mạng (Network Architecture)

Mô hình được xây dựng "from scratch" sử dụng tính toán ma trận, bao gồm các tầng sau:

- **Input Layer:** Tensor đầu vào $X \in \mathbb{R}^{13\times8\times8}$.
  - 12 kênh đầu (6×2): mã hóa one-hot vị trí các quân (P, N, B, R, Q, K) cho hai phe Trắng/Đen, mỗi ô nhận giá trị 0 hoặc 1.
  - 1 kênh cuối: mã hóa lượt đi, toàn bộ mặt phẳng nhận giá trị $+1$ nếu đến lượt Trắng, -1 nếu đến lượt Đen.
- **Flattening:** Tensor được duỗi phẳng thành vector $x \in \mathbb{R}^{832}$ (13×64).
- **Hidden Layers:** Mạng gồm 2 lớp ẩn với hàm kích hoạt ReLU ($g(z) = \max(0, z)$):

$$h_1 = \text{ReLU}(W_1 x + b_1), \qquad W_1 \in \mathbb{R}^{1024\times832},$$
$$h_2 = \text{ReLU}(W_2 h_1 + b_2), \qquad W_2 \in \mathbb{R}^{512\times1024}.$$

- **Output Layer:** Lớp đầu ra sử dụng hàm kích hoạt Tanh để chuẩn hóa giá trị về $[-1, 1]$:

$$\hat{y} = \tanh(W_3 h_2 + b_3), \quad W_3 \in \mathbb{R}^{1\times512}. \tag{4}$$

Ý nghĩa giá trị đầu ra và cách diễn giải. Trong quá trình huấn luyện, nhãn mục tiêu được gán theo kết quả ván cờ:

$$y = \begin{cases} +1 & \text{nếu ván cờ kết thúc với chiến thắng cho Trắng,} \\ -1 & \text{nếu ván cờ kết thúc với chiến thắng cho Đen.} \end{cases}$$

Nhờ sử dụng hàm Tanh, đầu ra của mạng sau huấn luyện là một giá trị liên tục $\hat{y} \in [-1, 1]$. Nhóm diễn giải:

- $\hat{y} \approx 1$: Trắng có lợi thế rất lớn.
- $\hat{y} \approx -1$: Đen có lợi thế rất lớn.
- $\hat{y} \approx 0$: thế cờ cân bằng.

Khi cần biểu diễn dưới dạng "mức độ lợi thế" trong khoảng $[0, 1]$, có thể ánh xạ tuyến tính:

$$p = \frac{\hat{y} + 1}{2} \in [0, 1],$$

và coi p là mức độ ưu thế của Trắng: p càng gần 1 thì Trắng càng ưu thế, càng gần 0 thì Đen càng ưu thế. Ví dụ, $\hat{y} \approx 0.8$ tương ứng với $p \approx 0.9$, có thể hiểu là Trắng đang có lợi thế rất lớn.

3.2.2 Cơ chế Ra quyết định (Decision Policy)

Hàm `select_move` thực hiện quy trình 4 bước kết hợp giữa suy luận nơ-ron và luật cứng (Rule-based heuristics):

Bước 1: Kiểm tra Chiếu hết (Mate-in-1). Trước khi tính toán, Agent duyệt qua tất cả nước đi hợp lệ $\mathcal{A}(s)$. Nếu tồn tại $a \in \mathcal{A}(s)$ sao cho trạng thái kế tiếp s' là chiếu hết (`is_checkmate`):

$$\text{return } a \quad \text{(ưu tiên tuyệt đối).}$$

Bước 2: Batch Inference. Thay vì đưa từng trạng thái vào mạng (gây nghẽn cổ chai), Agent sinh toàn bộ N trạng thái kế tiếp và xếp chồng thành một batch tensor $B \in \mathbb{R}^{N \times 13 \times 8 \times 8}$. Quá trình suy luận diễn ra song song trên GPU (sử dụng thư viện CuPy) hoặc CPU (NumPy):

$$\mathbf{v} = \text{Forward}(B) \in \mathbb{R}^N.$$

Bước 3: Heuristic Chống Hòa (Anti-Draw Adjustment). Các mô hình học máy thường có xu hướng chọn những thế cờ an toàn (dẫn tới hoà) thay vì mạo hiểm để thắng. Để khắc phục, Agent áp dụng phạt điểm $\delta = 0{,}5$ vào điểm số thô v_i nếu nước đi a_i dẫn đến thế cờ hoà (*stalemate*, *repetition*):

$$v_i' = \begin{cases} v_i - 0.5 & \text{nếu lượt đi thuộc về Trắng (cần Max) và is\_draw}(s_i), \\ v_i + 0.5 & \text{nếu lượt đi thuộc về Đen (cần Min) và is\_draw}(s_i). \end{cases} \tag{5}$$

Bước 4: Lựa chọn tham lam (Greedy Selection). Nước đi cuối cùng được chọn dựa trên giá trị đã hiệu chỉnh:

$$a^* = \arg\max_{a_i} v_i' \quad \text{(cho Trắng)}, \qquad a^* = \arg\min_{a_i} v_i' \quad \text{(cho Đen)}.$$

3.2.3 Nguồn dữ liệu và quy trình tiền xử lý cho MLP Agent

Để huấn luyện MLP Agent, nhóm cần một tập dữ liệu đủ lớn gồm các vị trí cờ vua (states) và nhãn đánh giá tương ứng. Trong đồ án này, nhóm sử dụng bộ dữ liệu *Chess games* được công bố công khai trên GitHub/Hugging Face [5], sau đó chuyển đổi sang định dạng `Parquet` và lưu tại `training/dataset.parquet`. Toàn bộ quá trình trích xuất và tiền xử lý dữ liệu được tự động hoá trong script `prepare_data_hf.py`, với các bước chính như sau:

- **Nạp dữ liệu ở chế độ streaming:** Do kích thước bộ dữ liệu rất lớn, nhóm không nạp toàn bộ vào RAM một lần mà sử dụng cơ chế *streaming* của thư viện `datasets` (Hugging Face):

  ```
  dataset = load_dataset(
      "parquet",
      data_files={"train": "training/dataset.parquet"},
      split="train",
      streaming=True
  )
  ```

  Cách làm này cho phép duyệt lần lượt từng ván cờ (record) mà vẫn giữ mức sử dụng bộ nhớ ổn định, phù hợp với cấu hình phần cứng thực nghiệm.

- **Lọc các ván cờ có kết quả rõ ràng:** Mỗi phần tử trong `dataset` tương ứng với một ván cờ, chứa thông tin `winner` và danh sách nước đi ở dạng UCI `moves_uci`. Script bỏ qua các ván không có người thắng hoặc kết thúc hoà:

  ```
  winner = game.get("winner")
  if winner not in ["white", "black"]:
      continue
  ```

  Sau khi lọc, nhãn mục tiêu được gán theo kết quả ván cờ:

  $$\text{eval} = \begin{cases} 1.0 & \text{nếu Trắng thắng (winner = white)}, \\ -1.0 & \text{nếu Đen thắng (winner = black)}. \end{cases}$$

  Như vậy, mục tiêu của bài toán học là xấp xỉ mối quan hệ giữa một trạng thái trung cuộc và kết quả cuối cùng của toàn ván đấu, với nhãn $y \in \{-1, +1\}$.

- **Giới hạn số ván và loại bỏ ván quá ngắn:** Để kiểm soát thời gian xử lý, script chỉ duyệt tối đa `MAX_GAMES_TO_PROCESS = 100000` ván cờ. Đồng thời, các ván có ít hơn 10 nước đi bị loại bỏ:

```
n_moves = len(moves_uci)
if n_moves < 10:
    continue
```

Điều này đảm bảo rằng các thế cờ trích xuất không nằm trong những ván "mở" quá ngắn hoặc bất thường.

- **Trích xuất các thế cờ trung cuộc:** Với mỗi ván hợp lệ, script khởi tạo một `chess.Board()` ở vị trí xuất phát, sau đó lần lượt đẩy từng nước đi trong danh sách `moves_uci`:

```
board = chess.Board()
for i, move_str in enumerate(moves_uci):
    move = chess.Move.from_uci(move_str)
    board.push(move)
```

Để tập trung vào giai đoạn trung cuộc, chỉ những chỉ số nước đi thoả:

$$8 < i < n_{\text{moves}} - 5$$

mới được xét đến. Các nước đầu tiên (khai cuộc) và các nước cuối cùng (cờ tàn, giai đoạn dọn dẹp) bị loại bỏ để tránh tạo ra quá nhiều mẫu quá "giống nhau" hoặc mang tính chất kỹ thuật thuần túy.

- **Lấy mẫu ngẫu nhiên để giảm tương quan:** Ngay cả trong khoảng trung cuộc, các trạng thái ở các nước liên tiếp vẫn có mức độ tương quan rất cao. Để giảm bớt tương quan và kiểm soát kích thước dữ liệu, script chỉ chọn mỗi trạng thái với xác suất 0,1:

```
if 8 < i < (n_moves - 5):
    if random.random() < 0.1:
        fen = board.fen()
        data_entries.append({"fen": fen, "eval": label})
```

Cách lấy mẫu này giúp:

  - Giảm hiện tượng "trùng lặp thông tin" giữa các thế cờ quá gần nhau.
  - Giảm kích thước dataset trong khi vẫn giữ được tính đa dạng về cấu trúc thế trận.

- **Tổng hợp và lưu thành file CSV:** Sau khi hoàn thành việc duyệt (tối đa) 100 000 ván cờ, script tổng hợp danh sách các bản ghi `{"fen": ..., "eval": ...}` vào một `DataFrame` của `pandas`:

```
df = pd.DataFrame(data_entries)
df.to_csv("training/dataset_large.csv", index=False)
```

Kết quả là một file `training/dataset_large.csv` gồm hai cột:

  - **fen**: chuỗi FEN biểu diễn trạng thái bàn cờ.
  - **eval**: nhãn giá trị $\pm 1{,}0$ tương ứng với kết quả thắng/thua của ván cờ chứa trạng thái đó.

File CSV này chính là đầu vào trực tiếp cho script huấn luyện `train_mlp.py` được trình bày ở mục tiếp theo.

Quy trình tiền xử lý như trên giúp thu được một tập dữ liệu lớn các thế cờ trung cuộc, đã được gán nhãn nhất quán và đa dạng về cấu trúc. Điều này tạo nền tảng tốt cho MLP Agent học được mối quan hệ giữa trạng thái bàn cờ và kết quả ván đấu, thay vì chỉ ghi nhớ các mẫu rời rạc.

3.2.4 Quy trình huấn luyện MLP Agent

Trong bài toán này, nhóm huấn luyện MLP Agent theo hướng **supervised learning** với bài toán **hồi quy giá trị thế cờ**. Mạng nơ-ron được huấn luyện để xấp xỉ hàm giá trị

$$V(s) \approx f_\theta(s) \in [-1, 1],$$

trong đó s là một trạng thái cờ vua (biểu diễn dưới dạng FEN), còn $f_\theta(s)$ là giá trị đánh giá thế cờ do mạng MLP dự đoán.

- **Dữ liệu huấn luyện:**
  - Dữ liệu được tổ chức dưới dạng một file `CSV` với các cột chính:
    - `fen`: chuỗi FEN biểu diễn trạng thái bàn cờ tại một thời điểm.
    - `eval`: nhãn rời rạc trong tập $\{-1, +1\}$ như mô tả ở trên (Trắng thắng/Đen thắng), thuộc khoảng $[-1, 1]$.
  - Script huấn luyện `train_mlp.py` cho phép chỉ định:
    - `--data`: đường dẫn tới tập dữ liệu gốc, mặc định `training/dataset_large.csv`.
    - `--finetune-data` (tuỳ chọn): nếu được cung cấp, mô hình sẽ ưu tiên sử dụng tập dữ liệu này để **fine-tune** trên một bộ dữ liệu "chất lượng cao" hơn (ví dụ các ván phân tích sâu, ván GM), thay vì `--data`.
  - Trước khi huấn luyện, toàn bộ DataFrame được **xáo trộn (shuffle)** bằng `df.sample(frac=1)` để phá vỡ mọi cấu trúc theo thứ tự có sẵn trong file (tránh mô hình học lệch theo vị trí dòng).
  - Mỗi trạng thái s được chuyển từ FEN sang tensor đầu vào bằng pipeline:

    $$\text{FEN} \to \texttt{python-chess.Board} \to \texttt{board\_to\_tensor} \to \mathbb{R}^{13\times8\times8},$$

    trong đó:
    - 12 kênh đầu mã hóa sự hiện diện của từng loại quân Trắng/Đen,
    - kênh cuối cùng mã hóa lượt đi (bên nào đang đi).

    Tensor nhận được có kiểu `PyTorch` và được chuyển sang `NumPy` bằng `.numpy()` trước khi đưa vào backend tính toán `xp` (NumPy/CuPy).
- **Hàm mất mát và tối ưu:**

– Bài toán được mô hình hóa là **hồi quy** giá trị đánh giá thế cờ, do đó nhóm sử dụng **Mean Squared Error (MSE)** làm hàm mất mát:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\big(f_\theta(s_i) - y_i\big)^2,$$

trong đó y_i là giá trị `eval` trong file dữ liệu.

– Lan truyền ngược (backpropagation) được cài đặt **thủ công** trong lớp `ChessMLP_Scratch`: hàm `backward` nhận đầu vào là dự đoán và nhãn mục tiêu, tính gradient

$$\frac{\partial \mathcal{L}}{\partial y_{\text{pred}}} = y_{\text{pred}} - y_{\text{true}},$$

sau đó lan truyền gradient qua các lớp `Linear`, `ReLU` và `Tanh`.

– Việc cập nhật trọng số sử dụng **Gradient Descent thuần tuý** trên từng lớp tuyến tính:

$$W \leftarrow W - \eta \cdot \frac{\partial \mathcal{L}}{\partial W}, \qquad b \leftarrow b - \eta \cdot \frac{\partial \mathcal{L}}{\partial b},$$

với η là **learning rate**. Mặc định, script sử dụng `lr = 0.01`.

– Để đảm bảo ổn định khi huấn luyện với ReLU, trọng số của các lớp `LinearLayer` được khởi tạo theo **Kaiming He Initialization**:

$$W \sim \mathcal{N}\Big(0, \frac{2}{n_{\text{in}}}\Big),$$

nơi n_{in} là số chiều đầu vào của lớp.

- **Thiết lập huấn luyện:**

– Huấn luyện được tổ chức theo kiểu **mini-batch**:

* **Batch size**: mặc định `64`.
* Với N mẫu, số batch mỗi epoch được tính bằng $\lceil N/\text{batch\_size} \rceil$.

– Số epoch mặc định là `30` (`--epochs 30`). Ở mỗi epoch:

1. Dữ liệu được shuffle lại một lần nữa để tránh overfitting theo thứ tự.
2. Vòng lặp duyệt từng batch:
   (a) Lấy lát dữ liệu tương ứng từ `DataFrame`.
   (b) Chuyển cột `fen` thành tensor $(13, 8, 8)$ bằng `board_to_tensor`, ghép thành mảng 3D có shape `(Batch, 13, 8, 8)`.
   (c) Chuyển sang backend `xp` bằng `xp.asarray` (NumPy trên CPU hoặc CuPy trên GPU).
   (d) Thực hiện `forward` qua mạng `ChessMLP_Scratch`, thu được dự đoán $f_\theta(s)$.
   (e) Tính loss MSE, gọi `backward` để lan truyền gradient.
   (f) Gọi `update(lr)` để cập nhật trọng số sau mỗi mini-batch.

3. Loss trung bình trên toàn bộ các batch trong epoch được log ra màn hình (sử dụng `tqdm` để hiển thị tiến trình).

– Toàn bộ mạng được cài đặt trên một alias `xp`, cho phép:

* Tự động sử dụng **NumPy** nếu không có GPU.
* Tận dụng **CuPy** để tăng tốc trên GPU nếu môi trường cho phép.

- **Checkpoint và Fine-tune:**

– Sau mỗi epoch, trọng số của tất cả các lớp tuyến tính được lưu ra file `.npz` (mặc định `training/best_model_mlp.npz`) thông qua hàm `save_weights`. Nếu đang dùng CuPy, các tensor được chuyển ngược về NumPy trước khi lưu.

– Tham số `--resume-from` cho phép nạp lại một checkpoint đã train trước đó (hàm `load_weights`):

* Hỗ trợ **resume training** (tiếp tục train trên cùng dataset).
* Hỗ trợ **transfer learning / fine-tuning** khi kết hợp với tham số `--finetune-data`.

– Trong phiên bản hiện tại, mô hình dừng huấn luyện sau khi hoàn thành số epoch được chỉ định; chưa sử dụng cơ chế *early stopping* dựa trên tập validation, mà tập trung vào theo dõi loss huấn luyện để đánh giá xu hướng hội tụ.

Sau khi hoàn tất quá trình huấn luyện, trọng số tốt nhất được lưu lại và được nạp vào trong `MLPAgent` để sử dụng ở chế độ suy luận (inference). Trong game thực, MLP Agent chỉ thực hiện `forward` trên các trạng thái kế tiếp để chấm điểm và **không** cập nhật trọng số nữa.

4 Kết quả và Đánh giá Thực nghiệm

4.1 Kịch bản và Môi trường kiểm thử

Để đánh giá toàn diện hiệu năng của các tác nhân (Agents), nhóm xây dựng một bộ kịch bản benchmark bao gồm các cặp đấu Minimax–Random, Random–Minimax, Minimax–Minimax, MLP–Random và MLP–Minimax. Mỗi kịch bản được lặp lại nhiều ván độc lập để thu thập số liệu thống kê đủ lớn về tỉ lệ thắng, hoà, thua và thời gian tính toán trung bình.
Môi trường thực nghiệm cụ thể như sau:

- **Phần cứng:** Hệ thống sử dụng vi xử lý **AMD Ryzen 5 7500F** (6 nhân, 12 luồng) kết hợp với GPU **NVIDIA GeForce RTX 5070** và bộ nhớ **RAM 32 GB**. Cấu hình này đủ mạnh để vừa chạy giao diện đồ hoạ Pygame, vừa thực thi các thuật toán tìm kiếm Minimax và suy luận MLP trên nhiều ván đấu liên tiếp với độ trễ chấp nhận được.
- **Phần mềm:** Môi trường lập trình được thiết lập trên **Python 3.13**, cùng các thư viện chính: `python-chess` (mô phỏng luật cờ vua), `numpy/cupy` (tính toán ma trận trên CPU/GPU), `pandas` (xử lý dữ liệu huấn luyện), `tqdm` (theo dõi tiến trình huấn luyện và benchmark) và `pygame` (hiển thị giao diện trò chơi).
- **Cấu hình Minimax:** Agent Minimax được cấu hình với độ sâu tìm kiếm cố định $d = 3$, áp dụng cắt tỉa *Alpha–Beta* và hàm đánh giá **Tapered Evaluation** dựa trên giá trị quân cờ (material) và các bảng điểm vị trí (*Piece-Square Tables*) cho hai giai đoạn trung cuộc và tàn cuộc. Trong phiên bản hiện tại, nhóm không sử dụng bảng băm (*transposition table*) để giữ cho hiện thực gọn, tập trung vào chất lượng hàm đánh giá và cấu trúc cây tìm kiếm cơ bản.
- **Cấu hình MLP:** MLP Agent sử dụng mô hình `ChessMLP_Scratch` với kiến trúc ba lớp tuyến tính: đầu vào 832 chiều (tương ứng tensor $13 \times 8 \times 8$ duỗi phẳng), hai lớp ẩn 1024 và 512 nơ-ron với hàm kích hoạt ReLU, và lớp đầu ra 1 nơ-ron với hàm kích hoạt Tanh (chuẩn hoá giá trị đánh giá về khoảng $[-1, 1]$). Trọng số các lớp tuyến tính được khởi tạo theo He Initialization, huấn luyện bằng hàm mất mát MSE và tối ưu hoá bằng Gradient Descent mini-batch với **batch size** mặc định 64, **learning rate** 0,01 và **30 epoch** qua toàn bộ tập dữ liệu.
- **Thiết lập benchmark:** Trong các thí nghiệm báo cáo, mỗi cặp Agent được đấu **100 ván** cho mỗi kịch bản. Mỗi ván khởi đầu từ thế cờ chuẩn, các nước đi của Agent được đo thời gian tính toán, và kết quả cuối cùng được phân loại thành thắng, hoà hoặc thua. Từ đó, nhóm tính toán các chỉ số:
  - Tỉ lệ thắng (Win Rate), hoà (Draw Rate), thua (Loss Rate) của Agent đang xét.
  - Thời gian trung bình trên mỗi ván, làm chỉ báo cho chi phí tính toán của từng phương pháp.

4.2 Kết quả Thống kê Chi tiết

Bảng 1 trình bày số liệu chi tiết về tỉ lệ Thắng (Win), Hòa (Draw), và Thua (Loss) của phe Trắng trong từng kịch bản.

Table 1: Kết quả thực nghiệm trên 4 kịch bản (Benchmark Tool)

ID	Agent	Đối thủ	Mục tiêu kiểm thử	Thắng	Hòa	Thua	Win Rate
1	Minimax ($d = 3$)	Random	Tấn công	99	1	0	99%
2	Minimax ($d = 3$)	Random	Phòng thủ (đi sau)	98	2	0	98%
3	MLP Agent	Random	Khả năng học	99	1	0	99%
4	MLP Agent	Minimax ($d = 3$)	AI vs Classic	0	0	100	0%

4.3 Phân tích và Đánh giá

4.3.1 Đánh giá Giải thuật Minimax (Kịch bản 1, 2, 3)

- **Khả năng Tấn công (ID 1):** Minimax thể hiện sự áp đảo tuyệt đối trước đối thủ ngẫu nhiên với tỉ lệ thắng 99%. Agent biết cách dồn quân và thực hiện chiếu hết hiệu quả ở tàn cuộc.
- **Khả năng Phòng thủ (ID 2):** Khi đi sau, Minimax vẫn duy trì tỉ lệ thắng/hòa rất cao (98%), chứng tỏ khả năng trừng phạt sai lầm của đối phương ngay cả khi gặp bất lợi về tiên thủ.
- **Độ ổn định (ID 3):** Trong kịch bản Minimax tự đấu (Self-play), kết quả thường thiên về Hòa hoặc Trắng thắng nhẹ, phản ánh tính chất cân bằng của thuật toán khi không có yếu tố ngẫu nhiên.

⇒ **Kết luận:** Minimax đạt yêu cầu đồ án (Thắng Random $\geq 90\%$).

4.3.2 Đánh giá Mô hình Học máy MLP (Kịch bản 4, 5)

- **Khả năng Tự học (ID 4):** MLP Agent đạt tỉ lệ thắng 99% trước Random. Điều này chứng minh mạng nơ-ron đã học được các khái niệm cơ bản như giá trị quân cờ và kiểm soát không gian thông qua quá trình huấn luyện, thay vì đi quân mù quáng.
- **So sánh với Thuật toán Tìm kiếm (ID 5):** Khi đối đầu với Minimax, MLP Agent gặp khó khăn (tỉ lệ thắng 0%).
  - *Nguyên nhân:* MLP ra quyết định dựa trên "trực giác" tĩnh (Static Intuition) tại một thời điểm, trong khi Minimax có khả năng tính trước (Look-ahead) 3 nước đi. Do đó, MLP dễ bị rơi vào các bẫy chiến thuật (Tactical traps) mà nó không nhìn thấy trước được.

⇒ **Kết luận:** MLP Agent đạt yêu cầu đồ án (Thắng Random $\geq 60\%$).

4.4 Tổng kết chung

Qua thực nghiệm, nhóm nhận thấy sự đánh đổi rõ rệt:

- **Minimax:** Chính xác, chặt chẽ nhưng tốn kém tài nguyên tính toán (chậm dần khi depth tăng).
- **MLP:** Tốc độ ra quyết định cực nhanh (phản xạ tức thì) nhưng thiếu chiều sâu tính toán nếu không kết hợp với Search (như MCTS).

References

[1] Shannon, C. E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, 41(314), 256-275.

[2] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

[3] Silver, D., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419), 1140-1144.

[4] Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). *Diving into Deep Learning*. Cambridge University Press. https://d2l.ai

[5] A. Uriot. Chess games dataset. *GitHub repository & Hugging Face dataset*, https://github.com/angeluriot/Chess_games, 2024.