CHESS ENGINE

RESEARCH AND DEVELOPMENT OF AI INTELLIGENT CHESS SYSTEM

Present By:

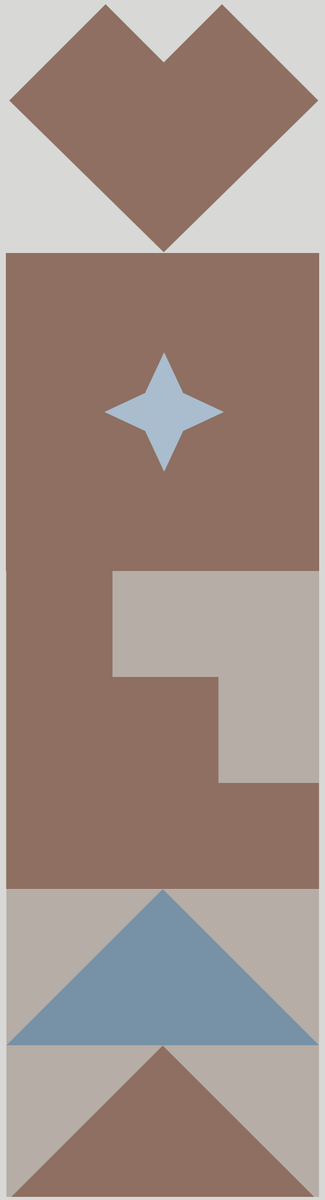
Tran Xuan Truong

Guided By:

Nguyen Hoang Diep

Criticism 1: Dao Anh Hien
Criticism 2: Do Thi Dao

Outline



Introduction

Even wondered how a simple modern computer can beat the best of grandmasters in the game of ches?

In this presentation, I will demonstrate:

- the backgroud and context of chess engines,
- the algorithms and techniques used to create them,
- the evaluation function that is crucial for their success,
- my project about implement this in real word

Background and Context

What is a Chess Engine?

Chess engine is a computer program that can play or analyse chess

History

-> A machine capable of checkmating with a king and rook vs king was created in 1912.
→ Alan wrote the first chess-playing computer program in 1951.
→ Chess engine development continued over the next 50 years, with hardware advancements leading to stronger play.
→ By 2005, chess engines surpassed the best human players in skill.

Objective And Scope

Objective:

- Developing a chess engine of mid-level intelligenece
- Developing a graphical interface to interact with the engine

Objective:

- Limited to standard game of chess with 8 * 8 cells
- 3 months for graduate protection
- Only two players at a time: human and the computer

Classical Algorithm

Phase 1

Logic game

Phase 2

Evaluation function

Phase 3

Search algorithms

Phase 4

advanced

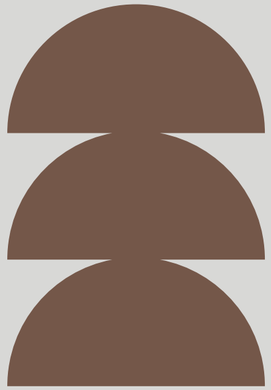
Phase 5

NNUE

Logic game

chess-python

python-chess is a chess library for Python, with move generation, move validation, and support for common formats.



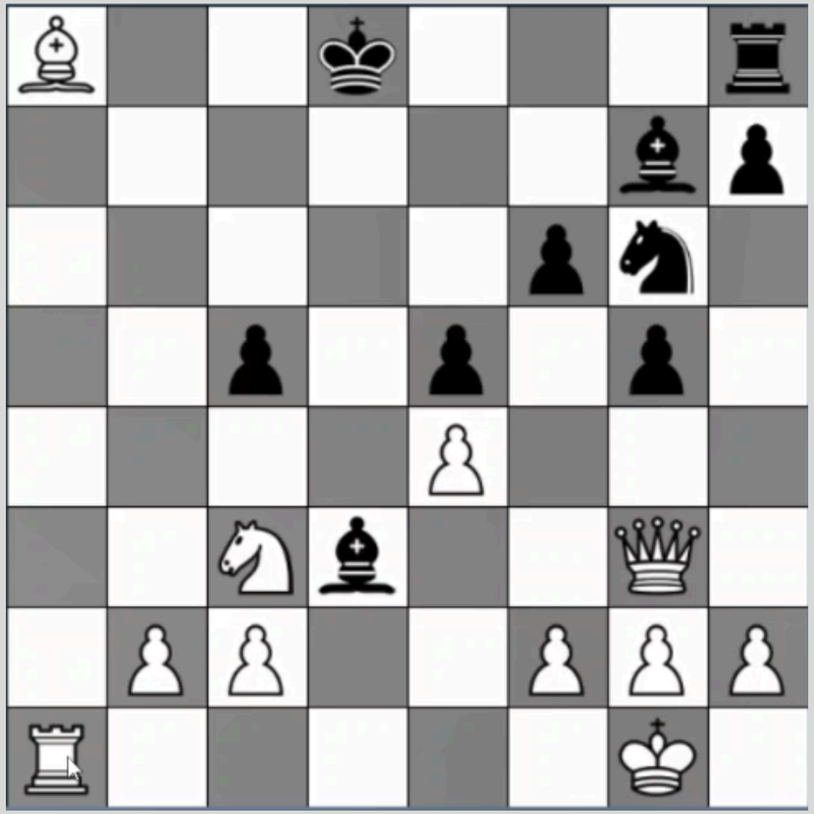
Advantage:

- Identity all pieces on the board and their positions
- Determine the legal moves for each piece based on the rules of chess(eg, pawns can move forward one or two squares on their first move, bishops move diagonally, etc,...)
- Check wether each move is legal by ensuring that it doesn't leave the player in check
- Generate a list of all legal moves for the player
- Consider special moves such as castling, en passant, and promotion

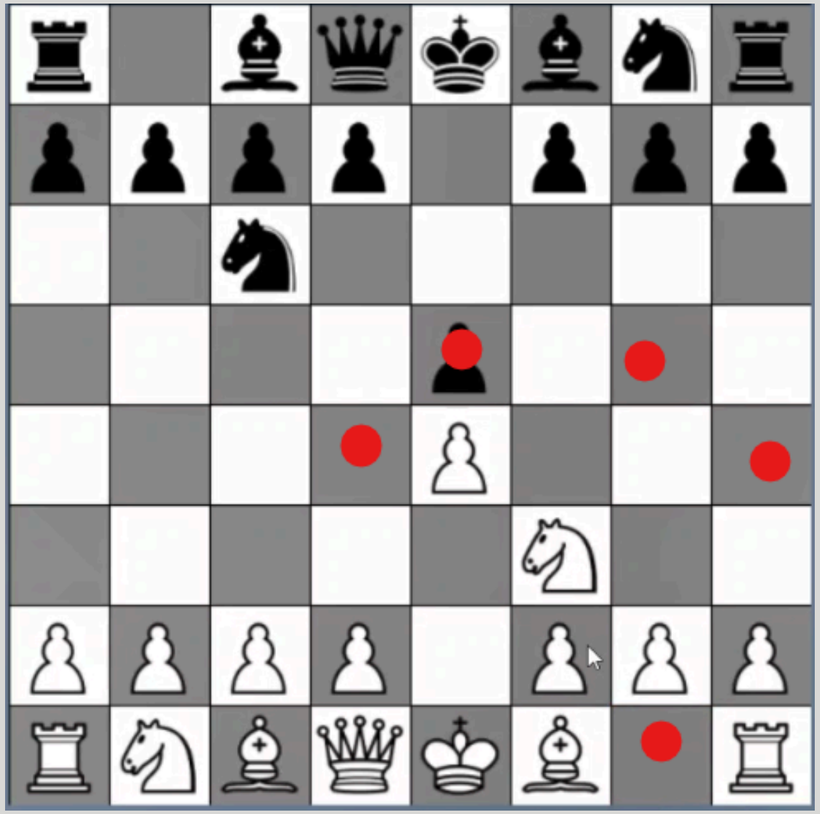
Evaluation Funtion

- Evaluation Funtion is used to evaluate the strength of each move based on various factors
- It assigns a numerical score to each move, which is then used by the engine to determine the best move to play

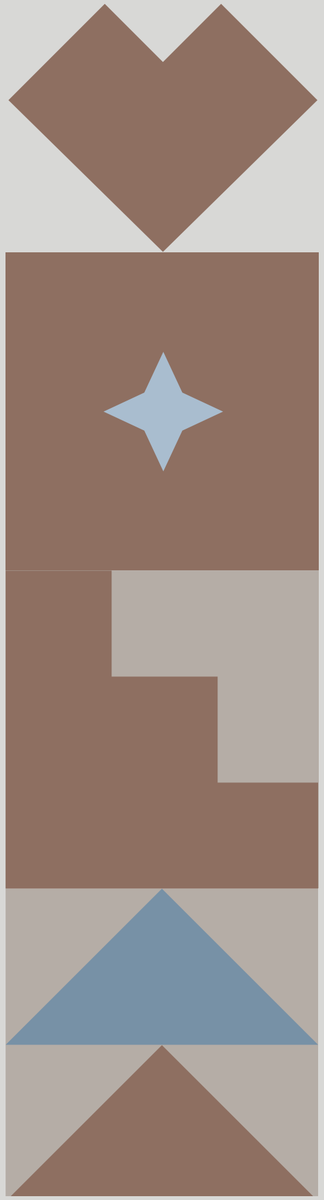
Materiall balance



Piece activity



Evaluation Funtion



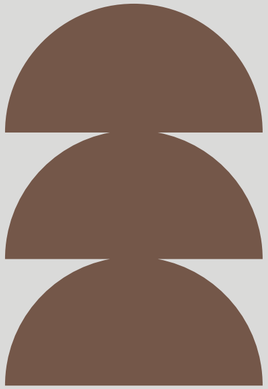
Centrel control

King safety

Pawn structure

Develop

Search algorithms



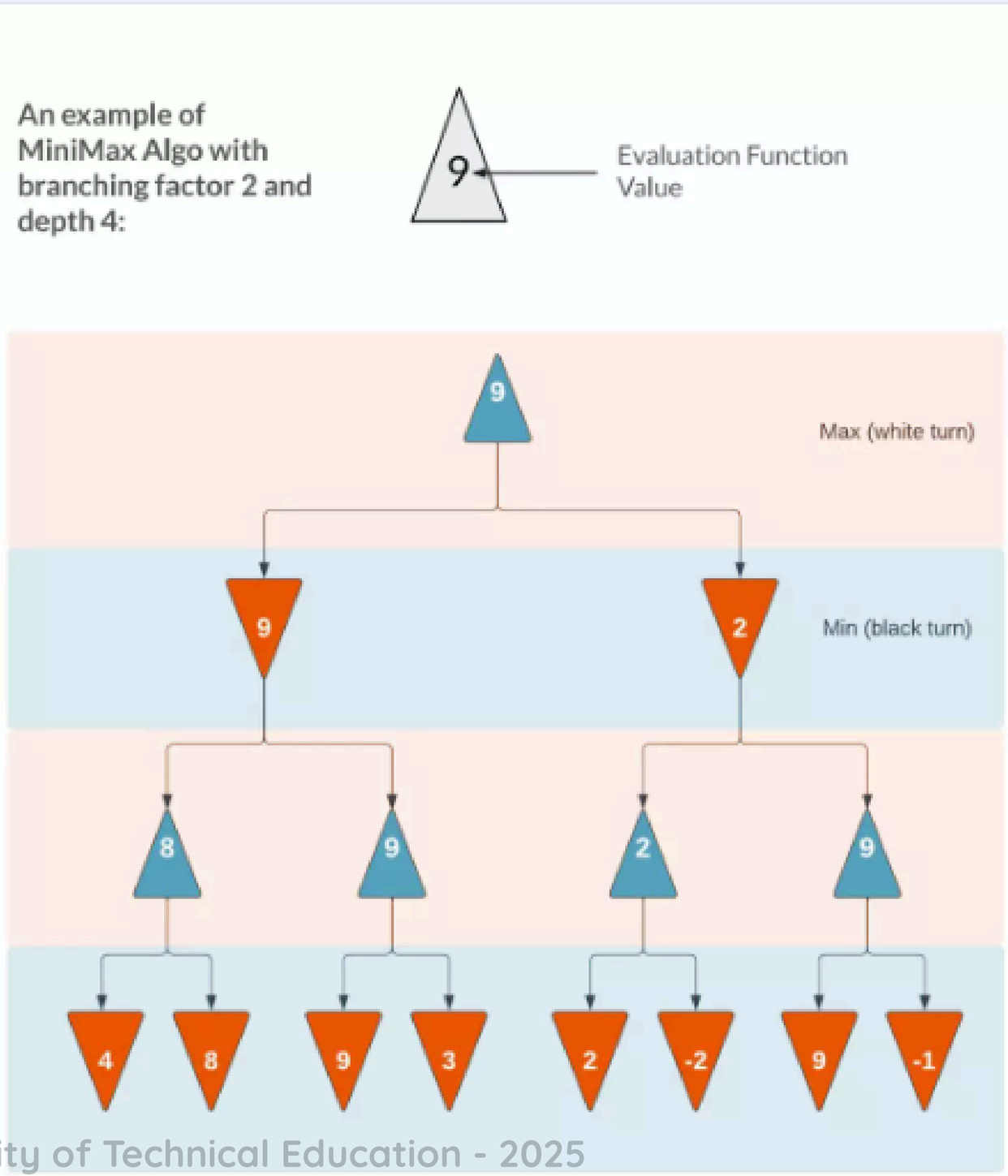
For the computer to choose the best move at a given state, it searchs all the valid moves upto a centain depth and asecrtain the best move based on the evaluation function provided

Search algos that can be used for Chess Engines are:

- Minimax Algorithm
- Alpha-Beta Pruning
- Iterative Deepening
- OpeingBook
- Endgame Tablebases
- Transposition Table
- Null Move Pruning
- Quiescence Search

Minimax Algorithms

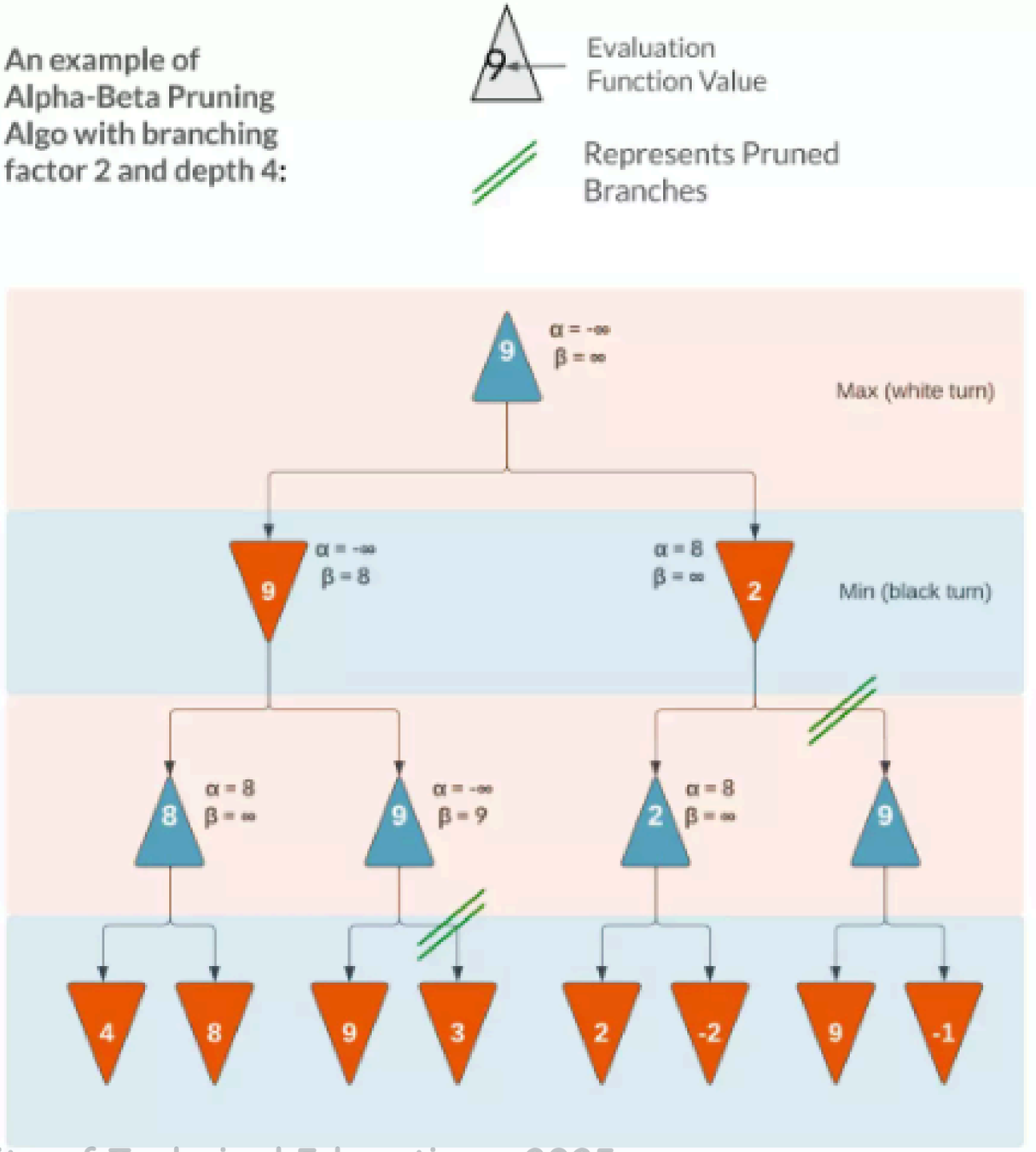
- Used to find the optimal move in a two-player zero-sum game, where one player's gain is the other player's loss.Therefore suitable for chess
- Works by recursively evaluating each possible move and its consequences
- Alternates between maximizing the score for the current player and minimizing the sore for the opponent player at each level
- The score is evaluated with Evaluation Function



Alpha-Beta Pruning

- Optimization technique used in the Minimax algorithm
- Reduce the number of nodes that need to evaluated in the Minimax algorithm, by pruning or cutting off branches of the game tree that are unlikely to lead to a better outcome
- Maintains two value α and β to store the minimum and maximum values found suring the search
- Uses these values to prune the branches which can’t give a better result

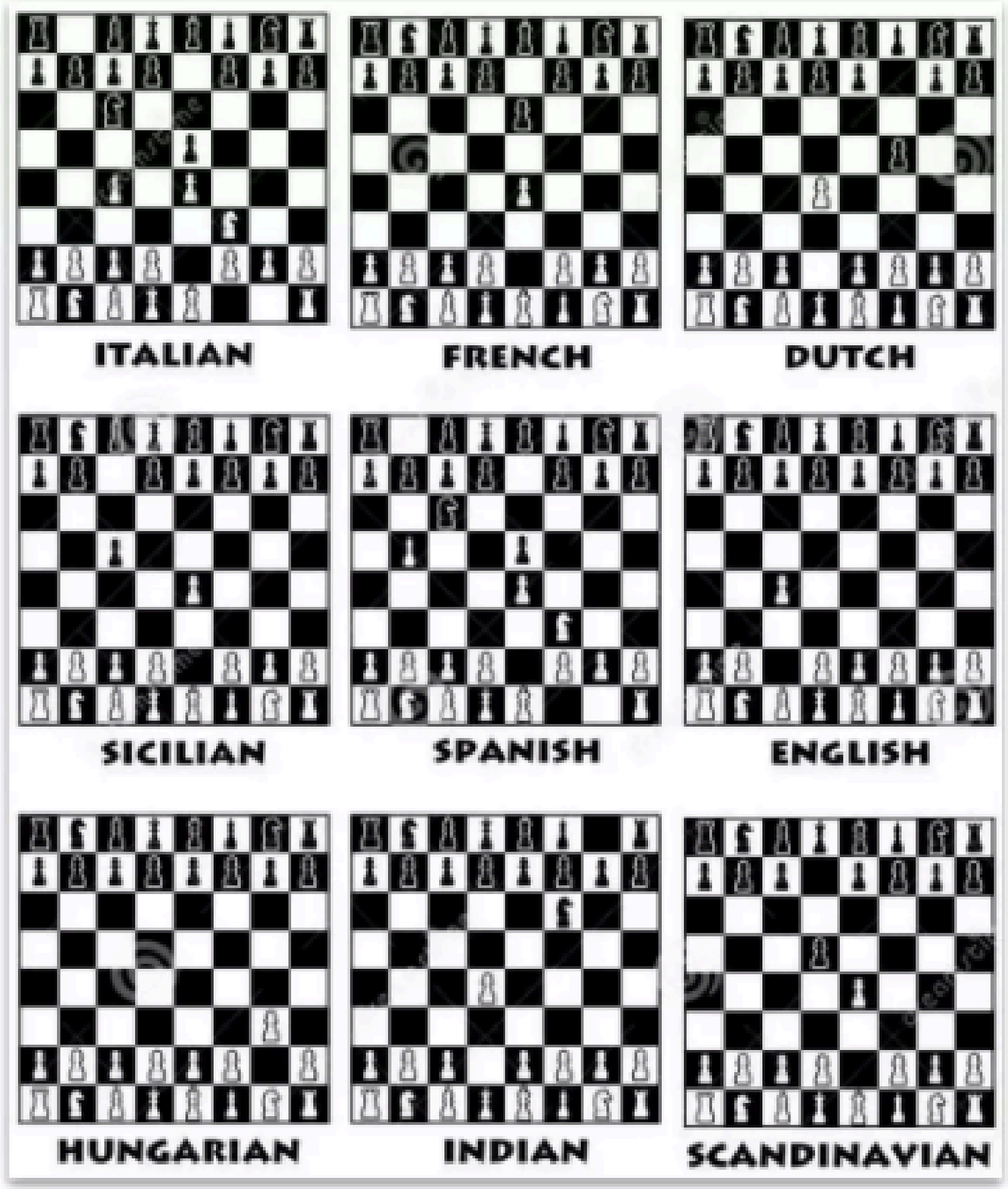
An example of Alpha-Beta Pruning Algo with branching factor 2 and depth 4:



Criteria	Minimax Algorithm	Alpha-Beta
Time complexity	O(b^d) • b is the branching factor • d is the depth	Same as Minimax Algorithm, but mostly faster due pruning
Memory usage	Requires keeping the entire game tree in memory	More memory effeciety dua pruning
Search time	Can be slow for large games	Faster than Minimax Algorithm
Solution optimality	Provides guaranteed optimal solution	May not provides guaranteed optimal solution

Opening Book and Endgame Tablebase

- A database oof known opening positions and endgame tactics and their best moves is already fed to the engine
- The engine simply take the best move as in the databse
- Database:
  Eg:https://www.chess.com/openings



Deep Learning

assets > remove > chess-data > fen > train.csv

```
1    FEN,Evaluation
2    r1b2rk1/ppp2pbp/3q1np1/n3p1B1/2B5/1Q3N2/PP1N1PPP/3R1RK1 w - - 4 14,-135
3    8/1pp2p2/6k1/4P2p/p1PR1K1P/2r2P2/6P1/8 w - - 0 33,-57
4    r2qk1nr/1b3pbp/n3p1p1/1pp1P3/p2PN3/2P2N2/PPB3PP/R1BQK2R w KQkq - 2 13,541
5    2b2rk1/5pp1/p2q1n1p/P2pn3/3N4/3BP1B1/2Q2PPP/Rr4K1 w - - 0 21,163
6    r2qkb1r/ppp2ppb/2n1p3/3n2PQ/3Pp3/2P4P/PP6/RNB1KBNR w KQkq - 0 11,-332
7    r1bq1rk1/ppp2ppp/3p4/2b1p1NP/2Bnn3/4Q3/PPPP1PP1/RNB1K2R b KQ - 1 9,-1167
8    4B3/5p1k/pp3pnp/4pN2/PP2b3/7P/5PP1/6K1 w - - 0 30,7
9    r1b1k2r/ppp2p2/2nqpn1p/3p2p1/3P4/P1P1PN1P/1P1N1PP1/R2QKB1R b KQkq - 0 11,85
10   8/R2n4/4k2p/P5p1/4P3/1K3P2/6Pb/8 w - - 1 32,379
11   r1b2b1r/pp2q1pk/2p3p1/3p2p1/4P2P/1Q4P1/PPPN1PB1/R3K1NR w KQ - 1 18,850
12   rn1qkbnr/pbpppppp/1p6/8/8/3PP3/PPP2PPP/RNBQKBNR w KQkq - 1 3,0
13   rnr3k1/4Qp2/b2p1P2/p2P2R1/1pq5/8/PP6/K3R3 b - - 0 29,339
14   8/1pk5/1p6/1PpR3p/P2b1pp1/5P1P/2P1K1P1/8 w - - 0 38,363
15   r1bqkb1r/pppp1ppp/2n5/3Pp3/8/5N2/PPPP1PPP/R1BQKB1R b KQkq - 0 5,26
16   r1b1r1k1/1pp2ppp/2nqpn2/p2p4/2PP4/PPQBPN2/3N1PPP/R4RK1 w - - 5 13,89
17   8/1pp3p1/p2k4/8/PP6/2P2K2/8/8 w - - 1 43,-510
18   r2q1rk1/1pp2pp1/p2p3p/3Pp3/4P3/3P1QPP/Pb1B1P2/1R3RK1 b - - 1 16,18
19   r5k1/5rpp/2p5/2Bpp3/1p6/P3QPN1/2q3PP/5RK1 w - - 0 27,132
20   2kr3r/p1pq1ppp/1p1pbn2/PP2p3/2P1Pn2/3PBN2/4BPPP/1R1Q1RK1 b - - 0 17,192
```

assets > remove > chess-data > fen > train.csv

```
1979351  r2qkb1r/ppp2ppp/3p1n2/1N2p1Q1/8/1P2P3/P1nP1PPP/R1B1K1NR w KQkq - 0 9,-497
1979352  6k1/2p2p1p/1p1p2p1/pR1P4/r1P5/2P3P1/5P1P/6K1 w - - 0 24,-417
1979353  1k1r2n1/pb1nq2p/1p1pp3/6P1/7P/1P3P2/P1P3B1/BN1QK1NR b K - 2 16,453
1979354  r2q1rk1/p4ppp/5b2/1Rp5/3pQ3/2P2N2/P4PPP/2R3K1 b - - 0 20,-200
1979355  rn2kb1r/p1pqn1p1/1p1p1p1p/3Pp3/4P3/2N2N2/PPP1QPPP/R1B1K2R w KQkq - 2 10,110
1979356  rnbqkbnr/3ppppp/p7/1pp5/2B1P3/5N2/PPPP1PPP/RNBQK2R w KQkq - 0 5,23
1979357  r1b4r/1p2kppp/2p5/p2pPn2/3N4/2N5/PP3PPP/R1B1KB1R w KQ - 0 13,687
1979358  k2r1bnr/p3q1pp/Pp1p1pn1/1Pp5/5P2/3PPQP1/1B5P/RN2K1NR b KQ - 2 18,143
1979359  r1bq1rk1/2p2pp1/p1np1n1p/1p6/1b1PP3/1B3N1P/PP3PP1/RNBQR1K1 w - - 1 12,58
1979360  Rr4k1/6pp/2K2p2/1r3p2/8/8/2P3PP/8 w - - 9 40,-453
1979361  8/6k1/1p2p1pp/4P3/1r4PP/p3K3/8/8 w - - 0 41,-725
1979362  3k4/3n3p/4R3/1p6/8/2P2P2/P1K3P1/RN6 b - - 0 33,893
1979363  r4rk1/5ppp/1pp3n1/1p1pP2q/3P4/P1P1QRP1/1PB4P/5RK1 w - - 1 24,145
1979364  rnbqk2r/pp2bppp/2p1pn2/3p4/2PP4/2N1PN2/PP3PPP/R1BQKB1R w KQkq - 1 6,0
1979365  rn1qkbnr/ppp1p1pp/5p2/3p4/3P1Bb1/2P4P/PP2PPP1/RN1QKBNR b KQkq - 0 4,91
1979366  rn3rk1/2pbbppp/3p4/4p3/1pP1Pn2/pP1P1N1P/P1B2P2/3R1KNR b - - 1 18,-714
1979367  rnbqk2r/pppp2pp/1b3n2/4NP2/3P4/8/PPP1BPPP/RNB1K2R w KQkq - 2 8,-533
1979368  8/p7/1pn1Nkp1/8/P7/2P5/1KP3r1/7R w - - 2 28,-21
1979369  2r1k2r/2q1bppp/p1bppn2/1pp5/2P1P3/1P1PNN2/P1Q2PPP/R1B2RK1 b k - 0 13,-19
1979370  r2q1rk1/1bp1n1pp/p2p4/2bP1p2/2P1p1N1/P7/BP3PPP/R1BQ1RK1 w - - 0 15,-77
1979371  r1bqk2r/p1p2ppp/2p5/8/2Qb4/1P4PP/P1PN4/R5KR w kq - 0 16,-466
1979372  r1b1k2r/1p3pp1/p1pp2qp/4p3/1PBbP3/P2P1Q1P/2P2PP1/1RB2RK1 b kq - 3 14,3
1979373  1r4k1/4Rpp1/p3p2p/B2pP3/2rP4/5NP1/P3QP1P/6K1 b - - 0 24,446
1979374  2kr4/p3bp2/3p3p/1P2p3/2R5/B1P2r2/P6P/3R3K b - - 0 25,-180
1979375  r2q1rk1/5pp1/p2p3p/1pbNnP2/1P6/8/P1P1Q1PP/R4R1K b - - 0 20,-272
1979376  r1bq4/ppp1bkpp/2n2n2/3pp3/4P3/3P1Q2/PPP2PPP/RNB1K2R w KQ - 0 9,-280
1979377  4r1k1/3R2p1/5p1p/4p3/p6n/PrP1P1BP/2K2R2/8 b - - 2 31,-255
1979378  r1bqkb1r/pp1p1p1p/2n1pnp1/2p5/3P4/NP3P1N/PBP1P1PP/R2QKB1R b KQkq - 3 6,-153
1979379  r1b1k2r/2p1qpb1/p1p2n1p/4p1p1/2P1P3/2N1BN2/PP3PPP/R2Q1RK1 w kq - 2 12,97
1979380  8/5r2/1k3P1P/5K2/8/6q1/8/8 w - - 3 66,-442
1979381  rnbqkbnr/ppp3pp/4p3/4Np2/3P4/8/PPP2PPP/RNBQKB1R b KQkq - 0 5,-422
1979382  rnbqkbnr/p1p1pppp/1p6/3P4/2B5/5Q2/PPPP1PPP/RNB1K1NR b KQkq - 1 4,188
1979383  4r1k1/p4ppn/3q3p/1pp5/3p1P2/1B1P1Q2/PPPN2PP/R3R1K1 b - - 0 22,656
1979384  8/4k2B/1p6/pP1p2P1/P2P1p2/5P2/6K1/8 b - - 2 53,355
1979385
```

STEP 1: INSTALL PACKAGE

```
%pip install chess
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import pandas as pd
from sklearn.model_selection import train_test_split
import chess
from torch.utils.tensorboard import SummaryWriter
import time
import os
import numpy as np
torch.manual_seed(42)
```

STEP 2: CONSTANT VARIABLES

```
%pip install chess
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import pandas as pd
from sklearn.model_selection import train_test_split
import chess
from torch.utils.tensorboard import SummaryWriter
import time
import os
import numpy as np
torch.manual_seed(42)
```

STEP3 : DEFINE ARCHITECTURE

```python
class NNUE(nn.Module):
    def __init__(self):
        super(NNUE, self).__init__()
        self.fc1 = nn.Linear(768, 8)
        self.fc2 = nn.Linear(8, 8)
        self.fc3 = nn.Linear(8, 1)

    def clipped_relu(self, x):
        return torch.clamp(x, 0, 1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.clipped_relu(x)
        x = self.fc2(x)
        x = self.clipped_relu(x)
        x = self.fc3(x)
        return x

# model = NNUE()
# print(model)
# dummy_input = torch.randn(1, 768) # Batch size 1, 768 features
# output = model(dummy_input)
# print(output)
```

STEP 4: LOAD DATASET

```python
def preprocess_fen(fen):
    board = chess.Board(fen)
    feature = torch.zeros(64, 6, 2)
    for square in chess.SQUARES:
        piece = board.piece_at(square)
        if piece is not None:
            color = int(piece.color)
            piece_type = piece.piece_type - 1
            feature[square, piece_type, color] = 1

    # [64,6,2] -> [768] shape (batch size, 768 features)
    feature = feature.reshape(-1)
    return feature

class ChessDataset(Dataset):
    def __init__(self, features, targets):
        self.features = features
        self.targets = targets

    def __len__(self):
        return len(self.targets)

    def __getitem__(self, idx):
        feature = self.features[idx]
        target = self.targets[idx]

        feature = preprocess_fen(feature)
        target = torch.tensor(target, dtype=torch.float32)

        return feature, target

# fen = "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 0"
# print(preprocess_fen("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 0"))
```

STEP 5: SPLIT DATA

```
df = pd.read_csv(PATH_DATA)
print('Original shape:', df.shape)
df = df[:LEN_DATA]
print('Used shape:', df.shape)

X = df['FEN'].to_list()
y = df['Evaluation'].to_list()
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
train_dataset = ChessDataset(X_train, y_train)
test_dataset = ChessDataset(X_test, y_test)

# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=True)
print(f"train_loader: {len(train_loader)}\t test_loader: {len(test_loader)}")
```

```
# train_dataset[0]
```

STEP 6: SETUP TRAINING

```python
class EarlyStopping:
    def __init__(self, tolerance=10, min_delta=10000):
        self.tolerance = tolerance
        self.counter = 0
        self.prev = False
        self.early_stop = False

        self.min_delta = min_delta
        self.min_train_loss = float('inf')

    def condition(self, train_loss, validation_loss):
        return abs(train_loss - validation_loss) <= self.min_delta and self.min_train_loss <= train_loss

    def __call__(self, train_loss, validation_loss):
        self.min_train_loss = min(self.min_train_loss, train_loss)
        if self.condition(train_loss, validation_loss):
            if self.prev == True:
                self.counter +=1
            else:
                self.counter = 1
            self.prev = True

            if self.counter >= self.tolerance:
                self.early_stop = True
        else:
            self.prev = False
```

```python
model = NNUE()
checkpoint = torch.load(f'./checkpoint/nnue_3_1978880d_512bs_300es_61e.pth', weights_only=True, map_location=torch.device(device))
model.load_state_dict(checkpoint['model_state_dict'])
model = model.to(device)

optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.MSELoss()

writer = SummaryWriter(PATH_RUNS)
os.makedirs('checkpoint', exist_ok=True)
model
```

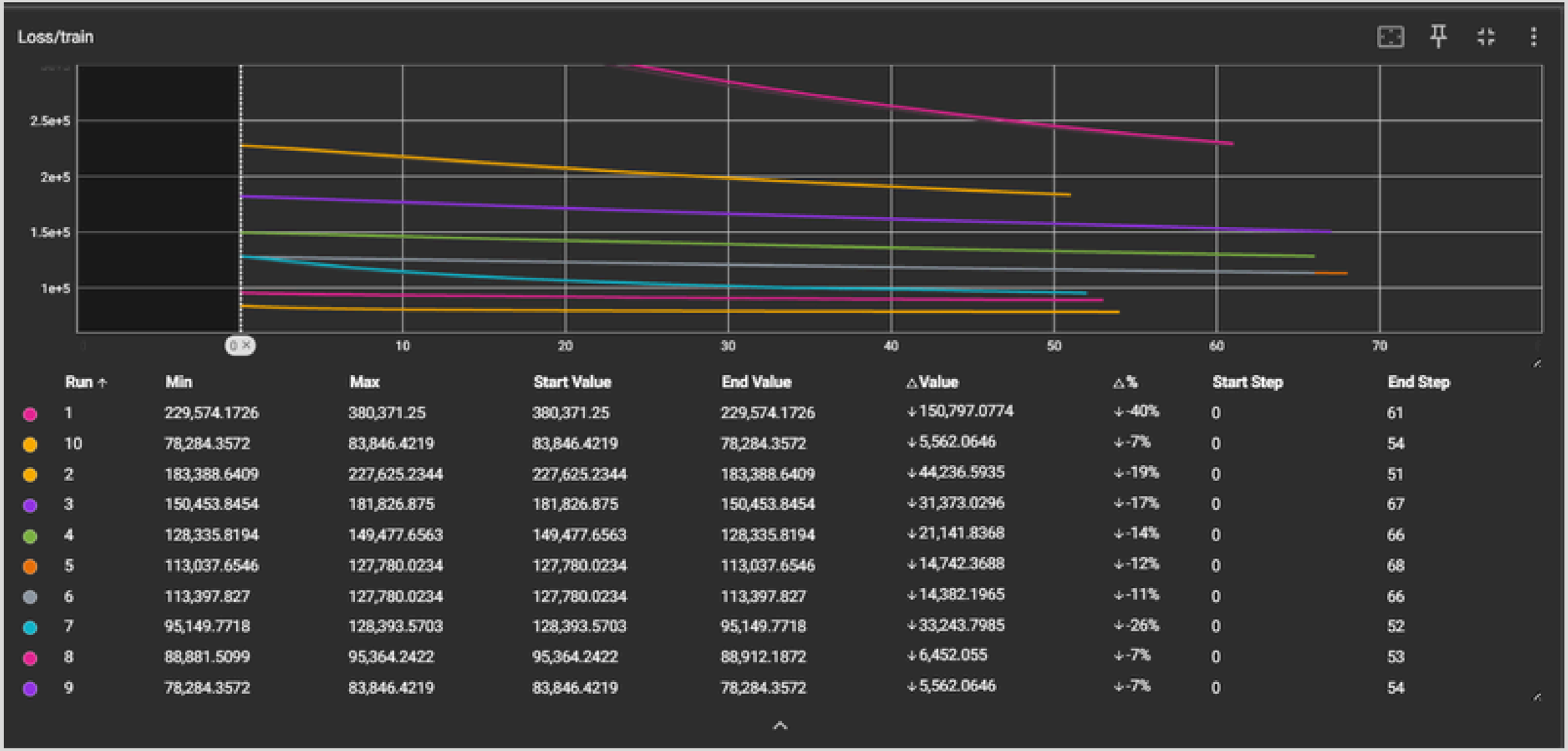
STEP 7: TRAINING AND VALUATION PHARE

```python
early_stopping = EarlyStopping(tolerance=TOLERANCE, min_delta=MIN_DELTA)
min_avg_val_loss = checkpoint['loss']

for epoch in range(NUM_EPOCHS):
    start_time = time.time()
    # training phase
    model.train()
    train_loss = 0.0
    for batch in train_loader:
        features, targets = batch
        features = features.to(device)
        targets = targets.to(device)

        outputs = model(features)
        # outputs: (batchsize, 1) -> (batchsize), target; (batchsize)
        loss = criterion(outputs.reshape(-1), targets)
        train_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    avg_train_loss = train_loss / len(train_loader)
    writer.add_scalar('Loss/train', avg_train_loss, epoch)

    # evaluation phase
    model.eval()  # Set model to evaluation mode
    val_loss = 0.0
    with torch.no_grad():
        for batch in test_loader:
            features, targets = batch
            features = features.to(device)
            targets = targets.to(device)

            outputs = model(features)

            loss = criterion(outputs.reshape(-1), targets)
            val_loss += loss.item()
    avg_val_loss = val_loss / len(test_loader)
    writer.add_scalar('Loss/val', avg_val_loss, epoch)
```

STEP 8: SAVE CHECKPOINT AND EARLY STOPPING

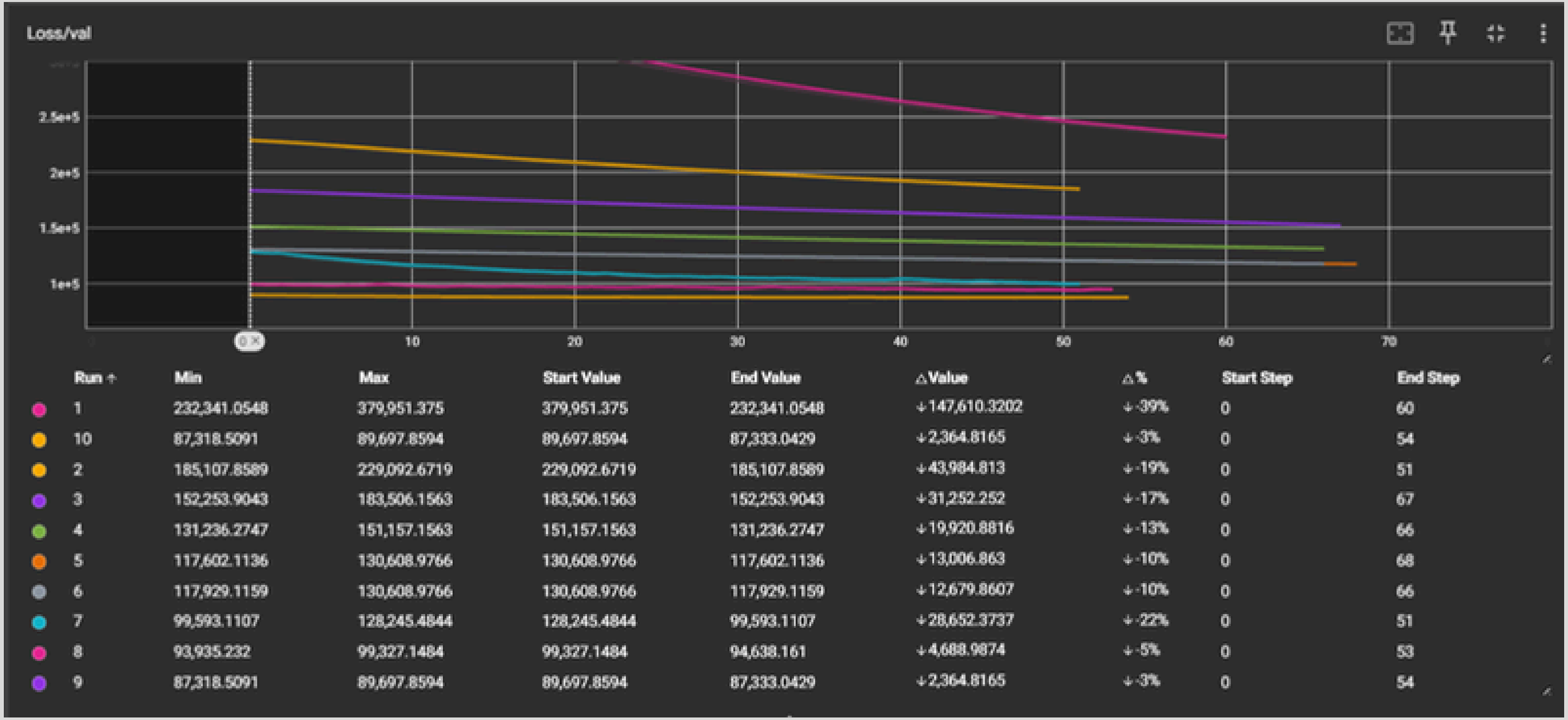
```
        early_stopping(avg_train_loss, avg_val_loss)
        if early_stopping.early_stop:
            print(f"Early stopping triggered after {epoch+1} epochs.")
            break

        # Save model checkpoint if validation loss improves
        if avg_val_loss < min_avg_val_loss:
            min_avg_val_loss = avg_val_loss
            path = f'./checkpoint/nnue_3_{LEN_DATA}d_{BATCH_SIZE}bs_{NUM_EPOCHS}es_{epoch+1}e.pth'
            checkpoint = {
                'epoch': epoch + 1,
                'model_state_dict': model.state_dict(),
                'optimizer_state_dict': optimizer.state_dict(),
                'loss': min_avg_val_loss
            }
            torch.save(checkpoint, path)

        end_time = time.time()
        print(f"Epoch [{epoch+1}/{NUM_EPOCHS}], Train Loss: {avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, Time: {round(end_time - start_time)}s")

    writer.close()
```

LOSS/TRAIN - STEP



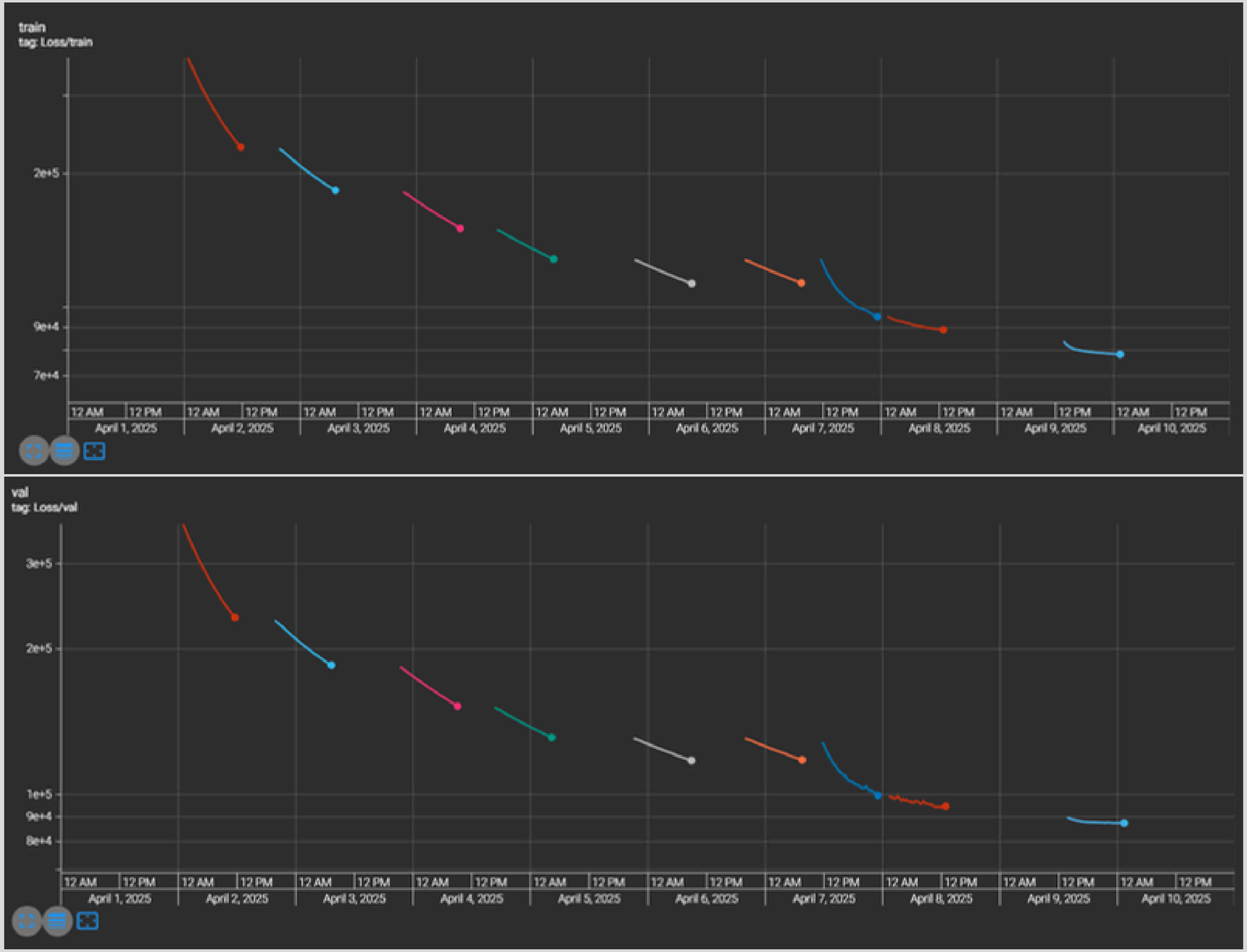
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
1	229,574.1726	380,371.25	380,371.25	229,574.1726	↓150,797.0774	↓-40%	0	61
10	78,284.3572	83,846.4219	83,846.4219	78,284.3572	↓5,562.0646	↓-7%	0	54
2	183,388.6409	227,625.2344	227,625.2344	183,388.6409	↓44,236.5935	↓-19%	0	51
3	150,453.8454	181,826.875	181,826.875	150,453.8454	↓31,373.0296	↓-17%	0	67
4	128,335.8194	149,477.6563	149,477.6563	128,335.8194	↓21,141.8368	↓-14%	0	66
5	113,037.6546	127,780.0234	127,780.0234	113,037.6546	↓14,742.3688	↓-12%	0	68
6	113,397.827	127,780.0234	127,780.0234	113,397.827	↓14,382.1965	↓-11%	0	66
7	95,149.7718	128,393.5703	128,393.5703	95,149.7718	↓33,243.7985	↓-26%	0	52
8	88,881.5099	95,364.2422	95,364.2422	88,912.1872	↓6,452.055	↓-7%	0	53
9	78,284.3572	83,846.4219	83,846.4219	78,284.3572	↓5,562.0646	↓-7%	0	54

LOSS/TEST- WALL



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
1	232,341.0548	379,951.375	379,951.375	232,341.0548	↓147,610.3202	↓-39%	0	60
10	87,318.5091	89,697.8594	89,697.8594	87,333.0429	↓2,364.8165	↓-3%	0	54
2	185,107.8589	229,092.6719	229,092.6719	185,107.8589	↓43,984.813	↓-19%	0	51
3	152,253.9043	183,506.1563	183,506.1563	152,253.9043	↓31,252.252	↓-17%	0	67
4	131,236.2747	151,157.1563	151,157.1563	131,236.2747	↓19,920.8816	↓-13%	0	66
5	117,602.1136	130,608.9766	130,608.9766	117,602.1136	↓13,006.863	↓-10%	0	68
6	117,929.1159	130,608.9766	130,608.9766	117,929.1159	↓12,679.8607	↓-10%	0	66
7	99,593.1107	128,245.4844	128,245.4844	99,593.1107	↓28,652.3737	↓-22%	0	51
8	93,935.232	99,327.1484	99,327.1484	94,638.161	↓4,688.9874	↓-5%	0	53
9	87,318.5091	89,697.8594	89,697.8594	87,333.0429	↓2,364.8165	↓-3%	0	54

LOSS/TRAIN+TEST WALL



Technologies Used:

Python 3.11

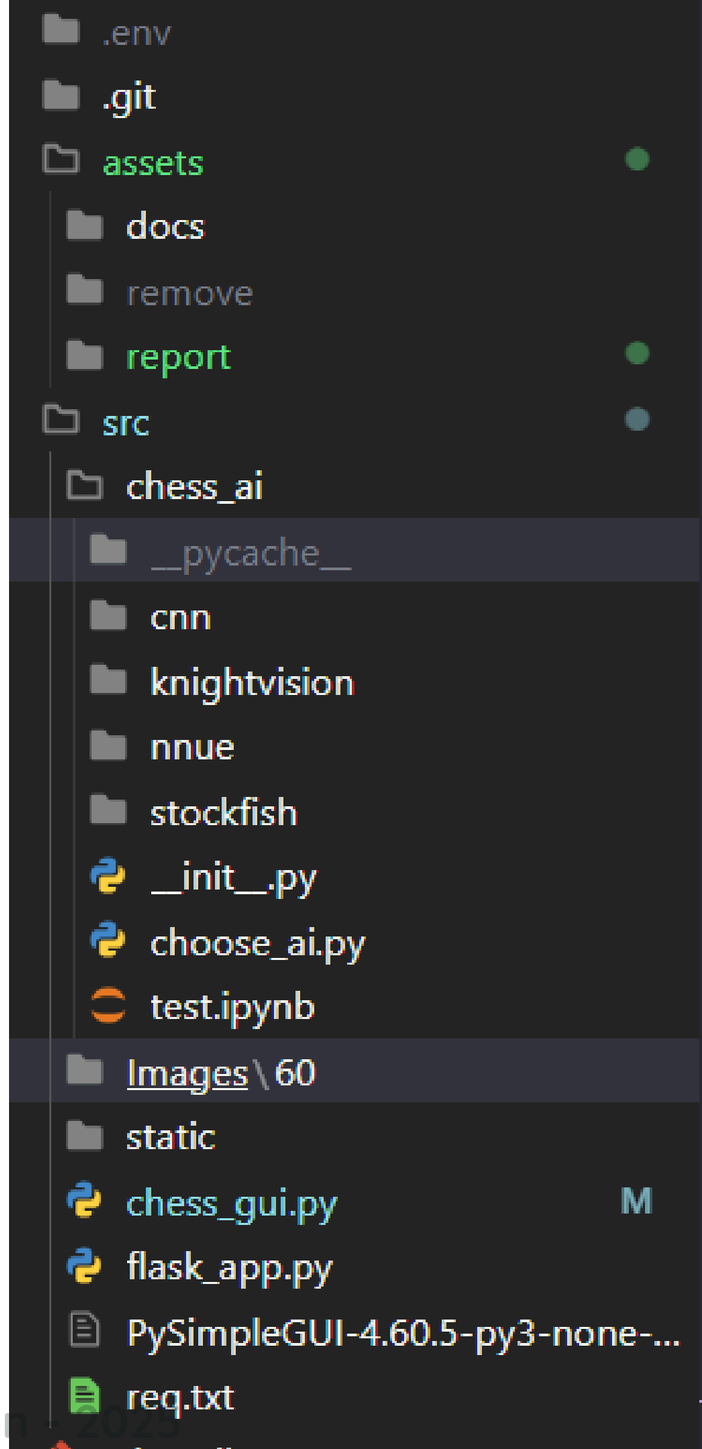
PySimpleGUI 4.60.5

Project structure

assets: Document, report, anything

src: Course code chess game, chess ai

images: Includes photo of each piece

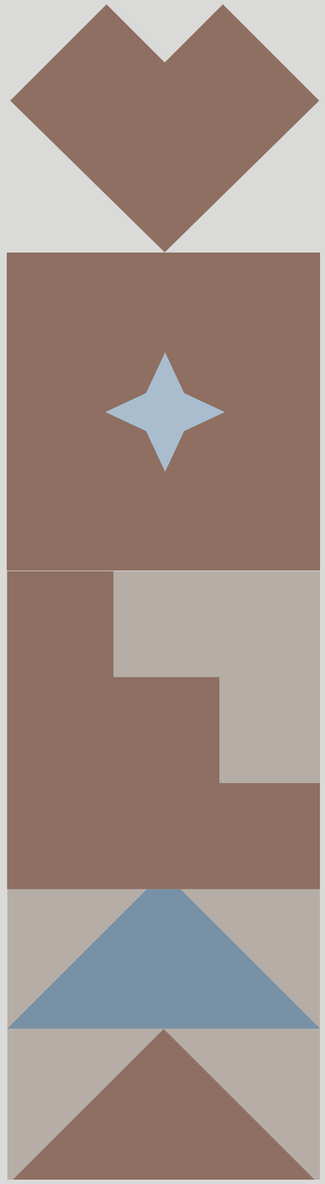


ScreenShots

Initial Screen:

During the game:

References and Tools



Stockfish Evaluation:

- Guidehttps://hxim.github.io/Stockfish-Evaluation-Guide

Opening Book:

- https://github.com/gmcheems-org/free-opening-books

Chess program wiki:

- https://www.chessprogramming.org/

Tools: notebooklm, grok, github copilot, gemmi, deepwiki

Thank you everybody.