**General introduction:**

This Python program named "Tensor-Chess CLI" is a command-line chess game based on TensorFlow. The main functions of this chess game are summarized as follows:

1. At the initial stage of entering the game, players can customize their own game id

2. Players can freely choose the chess pieces to move

3. The system will give the movable positions for players to choose according to the chess pieces to be moved next

4. The program supports special moves in chess (such as king and car castling, passing pawns, and pawn promotion)

5. When the player is operating, the system will record the score according to the player's killing of different chess pieces and completing special operations

6. After the game is over, the system will automatically record the number of wins and accumulated scores (winning will get extra points)

7. After the game is over, the player can choose whether to continue. If continuing, the system will automatically switch the black and white camps

**Code explanation:**

1. Initialize the chessboard:

文本

AI 生成的内容可能不正确。

1.1 :Use tensorflow to initialize and create an 8\*8 chessboard

1.2: Represent the chessboard in the form of tf.Variable so that it can be dynamically updated using TensorFlow operations

1.3: Define the encoding mapping of chess pieces (e.g. white pawn wP is 1, black queen bQ is 11)

2. Define the tool function:

文本

AI 生成的内容可能不正确。

get\_piece(): Get the chess piece code of a certain square on the board

set\_piece(): Set the chess piece code of a certain square on the board

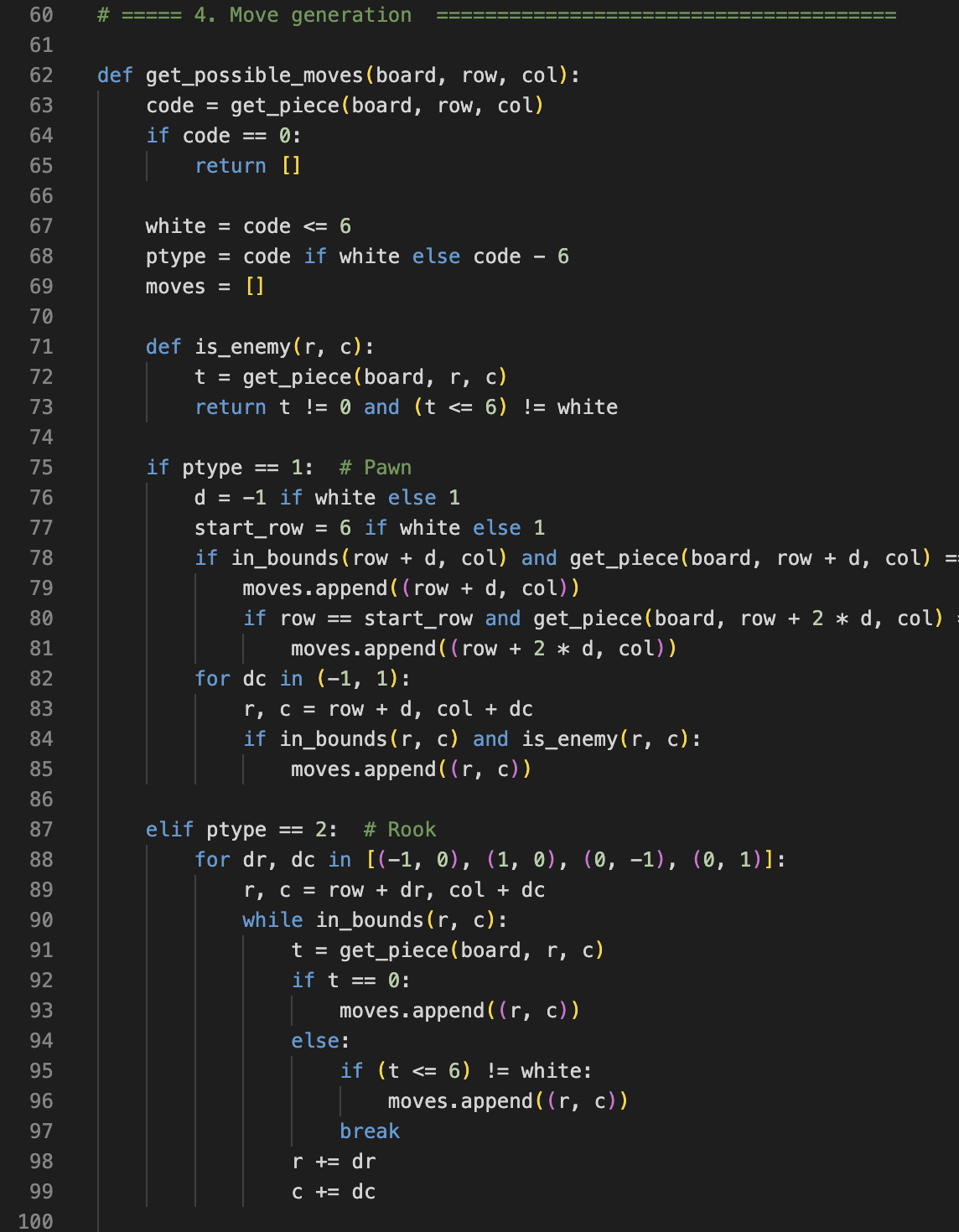
pos\_to\_coord(): Convert chessboard coordinates (such as e2) to array indexes

coord\_to\_pos(): Convert array indexes to chessboard coordinates

in\_bounds(): Check if the index is within the chessboard range

king\_exists(): Check if the king of a certain side is still on the chessboard

3. The core function of all legal moves of chess pieces

文本

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

3.1:

white = code <= 6

First determine which side the chess piece belongs to (white 1~6, black 7~12)

ptype = code if white else code – 6

Then determine the chess piece type by code-6, removing the color effect

Then use ptype to use the if-elif loop to determine the type of chess piece图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。

3.2: Example: Constructing the movement logic for a rook.：

for dr, dc in [(-1, 0), (1, 0), (0, -1), (0, 1)]:

Use a for loop to record the direction the piece can go to, which serves as the starting point for the next while loop.

Use while and if nesting to build the core function of moving pieces in a straight line（eg:rock,queen,bishop）：

while in\_bounds(r, c):

Check if the current position (r, c) is still within the 8x8 range of the board. If it is outside the boundaries, stop searching in that direction.

if t == 0:

moves.append((r, c))

If the grid is empty, you can go and continue to the next grid in the same direction (i.e. r += dr below)

else:

if (t <= 6) != white:

moves.append((r, c))

break

(t <= 6) != white: Determine if the piece is an enemy piece (different colors)

If it is an enemy piece → can capture → add moves and break

If it is your own piece → cannot move → break directly

r += dr

r += dr

c += dc

Continue moving in the current direction (up, down, left, right, or diagonal) until you hit a wall, encounter your own piece, or capture it.

4. The construction of special moves in chess:：

文本

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

4.1:

is\_castling\_move(): Checks if it is a castling move

handle\_castling(): Executes castling (moves the king and rook)

is\_en\_passant(): Checks if a passant can be captured

handle\_en\_passant(): Executes a passant capture

handle\_promotion(): Checks if the player has reached the end point and promoted to queen

4.2:

Here, sr, sc, er, and ec represent the horizontal and vertical coordinates of the initial and target respectively.

rook = get\_piece(board, sr, 7)

Purpose: Determine what kind of chess piece it is, or whether it is empty.

set\_piece(board, sr, 3, rook)

Castling

set\_piece(board, sr, 0, 0)

Clear the original location

5. Main loop of the code

5.1: Determine whether the currently selected chess piece is your own chess piece:

start = input(prompt).strip()

try:

sr, sc = pos\_to\_coord(start)

scode = get\_piece(board, sr, sc)

if scode == 0 or (scode <= 6) != white\_to\_move:

print("That square doesn't contain your piece.");

continue

Determine whether the square is the current chess piece, otherwise re-enter

5.2: Get all the paths that the current chess piece can take, and determine whether there is a path that can be taken：

moves = get\_possible\_moves(board, sr, sc)

if not moves:

print("No legal moves for this piece.");

continue

5.3: Provide legal paths：

print("Legal moves:")

for i,(r,c) in enumerate(moves):

print(f"{i+1}. {coord\_to\_pos(r,c)}")

idx = int(input("Choose move # ")) - 1

er, ec = moves[idx]

5.4: Extra points for capturing pieces:

表格

AI 生成的内容可能不正确。

if target\_piece != 0:

piece\_score = {1:1, 2:3, 3:3, 4:5, 5:9, 6:0,

7:1, 8:3, 9:3,10:5,11:9,12:0}

scores[actor]["points"] += piece\_score.get(target\_piece, 0)

if is\_castling\_move(board, sr, sc, er, ec, white\_to\_move):

handle\_castling(board, sr, sc, er, ec)

scores[actor]["points"] += 3

elif is\_en\_passant(board, sr, sc, er, ec, white\_to\_move):

handle\_en\_passant(board, sr, sc, er, ec, white\_to\_move)

scores[actor]["points"] += 2

if scode in [1, 7] and abs(er - sr) == 2:

en\_passant\_target = ((sr + er) // 2, sc)

else:

en\_passant\_target = None

if not king\_exists(board, not white\_to\_move):

winner = white\_id if white\_to\_move else black\_id

scores[winner]["wins"] += 1

scores[winner]["points"] += 10

5.5: Determine whether the game is over and output the final score and win rate

Determine whether the king is still there, and if the king dies, which side wins：

if not king\_exists(board, not white\_to\_move):

winner = white\_id if white\_to\_move else black\_id

Output final win and score

print(f"\nCheckmate! {winner} wins.")

print("Current scoreboard:")

for pid, sc in scores.items():

print(f" {pid}: {sc['wins']} win(s), {sc['points']} point(s)")

return

5.6: Define the main function, program welcome and player registration, initialize player scores, determine whether the game is over, and ask the player whether to continue

def main():

print("=== Tensor-Chess with score tracking ===")

p1 = input("Player 1, enter your ID (will start as White): ").strip()

p2 = input("Player 2, enter your ID (will start as Black): ").strip()

scores = {p1: {"wins": 0, "points": 0}, p2: {"wins": 0, "points": 0}}

white\_id, black\_id = p1, p2

while True:

play\_game(white\_id, black\_id, scores)

again = input("\nPlay again? (y/n): ").strip().lower()

if again != 'y':

print("Thanks for playing!")

break

white\_id, black\_id = black\_id, white\_id

print("\n=== Colors swapped! New game starts. ===\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()