import pygame

import sys

import time

import random

import copy

# Constants

BOARD\_SIZE = 8

SQUARE\_SIZE = 80

BACKGROUND\_COLOR = (121, 144, 104) # Color #799068

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

RED = (255, 0, 0)

GRAY = (128, 128, 128)

FONT\_SIZE = 36

# Initialize Pygame

pygame.init()

screen = pygame.display.set\_mode((0, 0), pygame.FULLSCREEN) # Fullscreen mode

screen\_width, screen\_height = screen.get\_size()

font = pygame.font.Font(None, FONT\_SIZE)

# Calculate board position for centering

board\_width = BOARD\_SIZE \* SQUARE\_SIZE

board\_height = BOARD\_SIZE \* SQUARE\_SIZE

board\_x\_offset = (screen\_width - board\_width) // 2

board\_y\_offset = (screen\_height - board\_height) // 2

# Game states

STATE\_MAIN\_MENU = "main\_menu"

STATE\_PLAYER\_SETUP = "player\_setup"

STATE\_BOT\_SETUP = "bot\_setup"

STATE\_DIFFICULTY\_SELECTION = "difficulty\_selection"

STATE\_GAME = "game"

STATE\_WINNER = "winner"

game\_state = STATE\_MAIN\_MENU

winner\_message = ""

# Input fields for player names

player1\_name = ""

player2\_name = ""

active\_input = None # Track which input is active

# Player colors and turn

player1\_color = BLACK

player2\_color = RED

current\_turn = BLACK # Start with Player 1

# Piece types for clarity

BLACK\_PIECE = "B"

RED\_PIECE = "R"

BLACK\_KING = "BK"

RED\_KING = "RK"

# Pieces

black\_piece = pygame.Surface((SQUARE\_SIZE, SQUARE\_SIZE), pygame.SRCALPHA)

pygame.draw.circle(black\_piece, BLACK, (SQUARE\_SIZE // 2, SQUARE\_SIZE // 2), SQUARE\_SIZE // 2 - 10)

red\_piece = pygame.Surface((SQUARE\_SIZE, SQUARE\_SIZE), pygame.SRCALPHA)

pygame.draw.circle(red\_piece, RED, (SQUARE\_SIZE // 2, SQUARE\_SIZE // 2), SQUARE\_SIZE // 2 - 10)

# Board setup

board = [[None for \_ in range(BOARD\_SIZE)] for \_ in range(BOARD\_SIZE)]

selected\_piece = None

invalid\_move\_timer = None # Timer for displaying "X" for invalid moves

#bot settings

bot\_game = False

bot\_difficulty = None

bot\_color = BLACK

# Functions

def initialize\_board():

for row in range(BOARD\_SIZE):

for col in range(BOARD\_SIZE):

if row < 3 and (row + col) % 2 != 0:

board[row][col] = BLACK\_PIECE # Black piece

elif row > 4 and (row + col) % 2 != 0:

board[row][col] = RED\_PIECE # Red piece

def switch\_turn():

global current\_turn

current\_turn = RED if current\_turn == BLACK else BLACK

def get\_square\_under\_mouse():

mouse\_x, mouse\_y = pygame.mouse.get\_pos()

row = (mouse\_y - board\_y\_offset) // SQUARE\_SIZE

col = (mouse\_x - board\_x\_offset) // SQUARE\_SIZE

if 0 <= row < BOARD\_SIZE and 0 <= col < BOARD\_SIZE:

return row, col

return None, None

def is\_valid\_move(start, end):

"""Validate the move according to checkers rules, including simple moves and captures."""

start\_row, start\_col = start

end\_row, end\_col = end

piece = board[start\_row][start\_col]

direction = 1 if piece in [BLACK\_PIECE, BLACK\_KING] else -1 # Black moves down, red moves up

king\_piece = (piece == BLACK\_KING or piece == RED\_KING)

# Move must be diagonal

if abs(start\_col - end\_col) != abs(start\_row - end\_row):

return False

# Regular move

if abs(start\_row - end\_row) == 1 and board[end\_row][end\_col] is None:

# Single piece can only move forward, while king can move in any direction

if piece in [BLACK\_PIECE, RED\_PIECE] and end\_row - start\_row != direction:

return False

return True

# Capture move

elif abs(start\_row - end\_row) == 2:

mid\_row = (start\_row + end\_row) // 2

mid\_col = (start\_col + end\_col) // 2

mid\_piece = board[mid\_row][mid\_col]

if mid\_piece and (piece in [BLACK\_PIECE, BLACK\_KING] and mid\_piece in [RED\_PIECE, RED\_KING]) or \

(piece in [RED\_PIECE, RED\_KING] and mid\_piece in [BLACK\_PIECE, BLACK\_KING]):

# Ensure regular pieces can only capture forward

if not king\_piece and (end\_row - start\_row != 2 \* direction):

return False

# Ensure the destination is empty

if board[end\_row][end\_col] is None:

return True

return False

def get\_valid\_moves(color):

"""Get all valid moves for a given color"""

valid\_moves = []

for row in range(BOARD\_SIZE):

for col in range(BOARD\_SIZE):

piece = board[row][col]

if piece and ((piece in [BLACK\_PIECE, BLACK\_KING] and color == BLACK) or (piece in [RED\_PIECE, RED\_KING] and color == RED)):

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

new\_row, new\_col = row + dr, col + dc

if 0 <= new\_row < BOARD\_SIZE and 0 <= new\_col < BOARD\_SIZE:

if board[new\_row][new\_col] is None and is\_valid\_move((row, col), (new\_row, new\_col)):

valid\_moves.append(((row, col), (new\_row, new\_col)))

new\_row, new\_col = row + 2\*dr, col + 2\*dc

if 0 <= new\_row < BOARD\_SIZE and 0 <= new\_col < BOARD\_SIZE:

if board[new\_row][new\_col] is None and is\_valid\_move((row, col), (new\_row, new\_col)):

valid\_moves.append(((row, col), (new\_row, new\_col)))

return valid\_moves

def check\_for\_king(row, col):

"""Promote to king if reaching the opposite side"""

piece = board[row][col]

if piece == BLACK\_PIECE and row == BOARD\_SIZE - 1:

board[row][col] = BLACK\_KING

elif piece == RED\_PIECE and row == 0:

board[row][col] = RED\_KING

def check\_for\_winner():

"""Count pieces and set the winner if one player has no pieces left"""

global game\_state, winner\_message

black\_pieces = sum(row.count(BLACK\_PIECE) + row.count(BLACK\_KING) for row in board)

red\_pieces = sum(row.count(RED\_PIECE) + row.count(RED\_KING) for row in board)

if black\_pieces == 0:

winner\_message = f"{player2\_name if player2\_color == RED else player1\_name} is the Winner!"

game\_state = STATE\_WINNER

elif red\_pieces == 0:

winner\_message = f"{player1\_name if player1\_color == BLACK else player2\_name} is the Winner!"

game\_state = STATE\_WINNER

def draw\_invalid\_move\_marker(row, col):

x\_pos = col \* SQUARE\_SIZE + board\_x\_offset + SQUARE\_SIZE // 2

y\_pos = row \* SQUARE\_SIZE + board\_y\_offset + SQUARE\_SIZE // 2

pygame.draw.line(screen, RED, (x\_pos - 15, y\_pos - 15), (x\_pos + 15, y\_pos + 15), 3)

pygame.draw.line(screen, RED, (x\_pos - 15, y\_pos + 15), (x\_pos + 15, y\_pos - 15), 3)

def draw\_main\_menu():

screen.fill(BACKGROUND\_COLOR)

local\_play\_text = font.render("Local Play", True, BLACK)

bot\_play\_text = font.render("Bot Play", True, BLACK)

quit\_text = font.render("Quit", True, BLACK)

SELECTION\_BACKGROUND\_COLOR = (217, 217, 217) # Color D9D9D9

# Calculate background rectangles for each menu option

local\_play\_rect = pygame.Rect(

screen\_width // 2 - local\_play\_text.get\_width() // 2 - 20,

screen\_height // 2 - 100 - 10,

local\_play\_text.get\_width() + 40,

local\_play\_text.get\_height() + 20

)

bot\_play\_rect = pygame.Rect(

screen\_width // 2 - bot\_play\_text.get\_width() // 2 - 20,

screen\_height // 2 - 10,

bot\_play\_text.get\_width() + 40,

bot\_play\_text.get\_height() + 20

)

quit\_rect = pygame.Rect(

screen\_width // 2 - quit\_text.get\_width() // 2 - 20,

screen\_height // 2 + 100 - 10,

quit\_text.get\_width() + 40,

quit\_text.get\_height() + 20

)

# Draw rounded rectangles as background for each menu option

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, local\_play\_rect, border\_radius=15)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, bot\_play\_rect, border\_radius=15)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, quit\_rect, border\_radius=15)

# Draw text on top of each background rectangle

screen.blit(local\_play\_text, (local\_play\_rect.x + 20, local\_play\_rect.y + 10))

screen.blit(bot\_play\_text, (bot\_play\_rect.x + 20, bot\_play\_rect.y + 10))

screen.blit(quit\_text, (quit\_rect.x + 20, quit\_rect.y + 10))

def draw\_player\_setup():

global active\_input, player1\_name, player2\_name

screen.fill(BACKGROUND\_COLOR)

SELECTION\_BACKGROUND\_COLOR = (217, 217, 217) # Color D9D9D9

# Text elements for labels

player1\_text = font.render("Player 1 Name:", True, BLACK)

player2\_text = font.render("Player 2 Name:", True, BLACK)

select\_color\_text = font.render("Select Color:", True, BLACK)

start\_text = font.render("Start Game", True, BLACK)

back\_text = font.render("Back", True, BLACK)

# Background rectangles for labels

player1\_label\_rect = pygame.Rect(screen\_width // 2 - 200, screen\_height // 2 - 200, player1\_text.get\_width() + 220, player1\_text.get\_height() + 20)

player2\_label\_rect = pygame.Rect(screen\_width // 2 - 200, screen\_height // 2 - 100, player2\_text.get\_width() + 220, player2\_text.get\_height() + 20)

start\_rect = pygame.Rect(screen\_width // 2 - start\_text.get\_width() // 2 - 20, screen\_height // 2 + 200, start\_text.get\_width() + 40, start\_text.get\_height() + 20)

back\_rect = pygame.Rect(screen\_width // 2 - back\_text.get\_width() // 2 - 20, screen\_height // 2 + 250, back\_text.get\_width() + 40, back\_text.get\_height() + 20)

# Draw rounded rectangles for labels and buttons

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, player1\_label\_rect, border\_radius=15)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, player2\_label\_rect, border\_radius=15)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, start\_rect, border\_radius=15)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, back\_rect, border\_radius=15)

# Render and position label texts

screen.blit(player1\_text, (player1\_label\_rect.x + 10, player1\_label\_rect.y + 10))

screen.blit(player2\_text, (player2\_label\_rect.x + 10, player2\_label\_rect.y + 10))

screen.blit(select\_color\_text, (screen\_width // 2 - select\_color\_text.get\_width() // 2, screen\_height // 2 + 10))

screen.blit(start\_text, (start\_rect.x + 20, start\_rect.y + 10))

screen.blit(back\_text, (back\_rect.x + 20, back\_rect.y + 10))

# Draw input boxes with player name texts

pygame.draw.rect(screen, WHITE, (screen\_width // 2, player1\_label\_rect.y, 200, player1\_text.get\_height() + 20))

pygame.draw.rect(screen, WHITE, (screen\_width // 2, player2\_label\_rect.y, 200, player2\_text.get\_height() + 20))

player1\_name\_text = font.render(player1\_name, True, BLACK)

player2\_name\_text = font.render(player2\_name, True, BLACK)

screen.blit(player1\_name\_text, (screen\_width // 2 + 10, player1\_label\_rect.y + 10))

screen.blit(player2\_name\_text, (screen\_width // 2 + 10, player2\_label\_rect.y + 10))

# Draw color selection circles

pygame.draw.circle(screen, BLACK, (screen\_width // 2 - 50, screen\_height // 2 + 60), 20)

pygame.draw.circle(screen, RED, (screen\_width // 2 + 50, screen\_height // 2 + 60), 20)

# Outline selected color

if player1\_color == BLACK:

pygame.draw.circle(screen, WHITE, (screen\_width // 2 - 50, screen\_height // 2 + 60), 25, 2)

else:

pygame.draw.circle(screen, WHITE, (screen\_width // 2 + 50, screen\_height // 2 + 60), 25, 2)

# Display selected color texts

player1\_color\_text = font.render(f"Player 1 Color: {'Black' if player1\_color == BLACK else 'Red'}", True, BLACK)

player2\_color\_text = font.render(f"Player 2 Color: {'Black' if player2\_color == BLACK else 'Red'}", True, BLACK)

screen.blit(player1\_color\_text, (screen\_width // 2 - player1\_color\_text.get\_width() // 2, screen\_height // 2 + 100))

screen.blit(player2\_color\_text, (screen\_width // 2 - player2\_color\_text.get\_width() // 2, screen\_height // 2 + 150))

def draw\_bot\_setup():

global active\_input, player1\_name, player2\_name

screen.fill(BACKGROUND\_COLOR)

SELECTION\_BACKGROUND\_COLOR = (217, 217, 217) # Color D9D9D9

# Text elements for labels

player1\_text = font.render("Player Name:", True, BLACK)

select\_color\_text = font.render("Select Color:", True, BLACK)

start\_text = font.render("Start Game", True, BLACK)

back\_text = font.render("Back", True, BLACK)

# Background rectangles for labels

player1\_label\_rect = pygame.Rect(screen\_width // 2 - 200, screen\_height // 2 - 200, player1\_text.get\_width() + 220, player1\_text.get\_height() + 20)

start\_rect = pygame.Rect(screen\_width // 2 - start\_text.get\_width() // 2 - 20, screen\_height // 2 + 200, start\_text.get\_width() + 40, start\_text.get\_height() + 20)

back\_rect = pygame.Rect(screen\_width // 2 - back\_text.get\_width() // 2 - 20, screen\_height // 2 + 250, back\_text.get\_width() + 40, back\_text.get\_height() + 20)

# Draw rounded rectangles for labels and buttons

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, player1\_label\_rect, border\_radius=15)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, start\_rect, border\_radius=15)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, back\_rect, border\_radius=15)

# Render and position label texts

screen.blit(player1\_text, (player1\_label\_rect.x + 10, player1\_label\_rect.y + 10))

screen.blit(select\_color\_text, (screen\_width // 2 - select\_color\_text.get\_width() // 2, screen\_height // 2 + 10))

screen.blit(start\_text, (start\_rect.x + 20, start\_rect.y + 10))

screen.blit(back\_text, (back\_rect.x + 20, back\_rect.y + 10))

# Draw input box with player name text

pygame.draw.rect(screen, WHITE, (screen\_width // 2, player1\_label\_rect.y, 200, player1\_text.get\_height() + 20))

player1\_name\_text = font.render(player1\_name, True, BLACK)

screen.blit(player1\_name\_text, (screen\_width // 2 + 10, player1\_label\_rect.y + 10))

# Draw color selection circles

pygame.draw.circle(screen, BLACK, (screen\_width // 2 - 50, screen\_height // 2 + 60), 20)

pygame.draw.circle(screen, RED, (screen\_width // 2 + 50, screen\_height // 2 + 60), 20)

# Outline selected color

if player1\_color == BLACK:

pygame.draw.circle(screen, WHITE, (screen\_width // 2 - 50, screen\_height // 2 + 60), 25, 2)

else:

pygame.draw.circle(screen, WHITE, (screen\_width // 2 + 50, screen\_height // 2 + 60), 25, 2)

# Display selected color texts

bot\_color\_text = font.render(f"Bot Color: {'Red' if player1\_color == BLACK else 'Black'}", True, BLACK)

screen.blit(bot\_color\_text, (screen\_width // 2 - bot\_color\_text.get\_width() // 2, screen\_height // 2 + 100))

def bot\_moves():

global winner\_message, game\_state

moves = get\_valid\_moves(bot\_color)

if not moves:

# Bot has no valid moves, player wins

winner\_message = f"{player1\_name if player1\_color == BLACK else player2\_name} wins!"

game\_state = STATE\_WINNER

return

if bot\_difficulty == 'easy':

# Random move

move = random.choice(moves)

elif bot\_difficulty == 'medium':

# Use minimax with moderate depth

\_, move = minimax(board, depth=3, alpha=float('-inf'), beta=float('inf'), maximizing\_player=True, bot\_color=bot\_color)

elif bot\_difficulty == 'hard':

# Use minimax with a deeper search

\_, move = minimax(board, depth=5, alpha=float('-inf'), beta=float('inf'), maximizing\_player=True, bot\_color=bot\_color)

# Execute move

(start\_row, start\_col), (end\_row, end\_col) = move

if abs(end\_row - start\_row) == 2:

mid\_row = (start\_row + end\_row) // 2

mid\_col = (start\_col + end\_col) // 2

board[mid\_row][mid\_col] = None

board[end\_row][end\_col] = board[start\_row][start\_col]

board[start\_row][start\_col] = None

check\_for\_king(end\_row, end\_col)

switch\_turn()

check\_for\_winner()

def evaluate\_board(board, bot\_color):

black\_pieces = 0

red\_pieces = 0

for row in board:

for piece in row:

if piece == BLACK\_PIECE:

black\_pieces += 1

elif piece == RED\_PIECE:

red\_pieces += 1

elif piece == BLACK\_KING:

black\_pieces += 1.5

elif piece == RED\_KING:

red\_pieces += 1.5

if bot\_color == BLACK:

return black\_pieces - red\_pieces

else:

return red\_pieces - black\_pieces

def make\_move(board, move):

new\_board = copy.deepcopy(board)

(start\_row, start\_col), (end\_row, end\_col) = move

piece = new\_board[start\_row][start\_col]

new\_board[start\_row][start\_col] = None

new\_board[end\_row][end\_col] = piece

if abs(end\_row - start\_row) == 2:

mid\_row = (start\_row + end\_row) // 2

mid\_col = (start\_col + end\_col) // 2

new\_board[mid\_row][mid\_col] = None

if piece == BLACK\_PIECE and end\_row == BOARD\_SIZE - 1:

new\_board[end\_row][end\_col] = BLACK\_KING

elif piece == RED\_PIECE and end\_row == 0:

new\_board[end\_row][end\_col] = RED\_KING

return new\_board

def minimax(board, depth, alpha, beta, maximizing\_player, bot\_color):

if depth == 0 or game\_over(board):

return evaluate\_board(board, bot\_color), None

current\_color = bot\_color if maximizing\_player else (RED if bot\_color == BLACK else BLACK)

moves = get\_valid\_moves(current\_color)

if not moves:

return evaluate\_board(board, bot\_color), None

best\_move = None

if maximizing\_player:

max\_eval = float('-inf')

for move in moves:

new\_board = make\_move(board, move)

eval\_score, \_ = minimax(new\_board, depth - 1, alpha, beta, False, bot\_color)

if eval\_score > max\_eval:

max\_eval = eval\_score

best\_move = move

alpha = max(alpha, eval\_score)

if beta <= alpha:

break

return max\_eval, best\_move

else:

min\_eval = float('inf')

for move in moves:

new\_board = make\_move(board, move)

eval\_score, \_ = minimax(new\_board, depth - 1, alpha, beta, True, bot\_color)

if eval\_score < min\_eval:

min\_eval = eval\_score

best\_move = move

beta = min(beta, eval\_score)

if beta <= alpha:

break

return min\_eval, best\_move

def draw\_difficulty\_selection():

screen.fill(BACKGROUND\_COLOR)

SELECTION\_BACKGROUND\_COLOR = (217, 217, 217) # Color D9D9D9

# Text elements

easy\_text = font.render("Easy", True, BLACK)

medium\_text = font.render("Medium", True, BLACK)

hard\_text = font.render("Hard", True, BLACK)

back\_text = font.render("Back", True, BLACK)

# Background rectangles for each option

easy\_rect = pygame.Rect(screen\_width // 2 - easy\_text.get\_width() // 2 - 20, screen\_height // 2 - 100, easy\_text.get\_width() + 40, easy\_text.get\_height() + 20)

medium\_rect = pygame.Rect(screen\_width // 2 - medium\_text.get\_width() // 2 - 20, screen\_height // 2, medium\_text.get\_width() + 40, medium\_text.get\_height() + 20)

hard\_rect = pygame.Rect(screen\_width // 2 - hard\_text.get\_width() // 2 - 20, screen\_height // 2 + 100, hard\_text.get\_width() + 40, hard\_text.get\_height() + 20)

back\_rect = pygame.Rect(screen\_width // 2 - back\_text.get\_width() // 2 - 20, screen\_height // 2 + 200, back\_text.get\_width() + 40, back\_text.get\_height() + 20)

# Draw rounded rectangles as backgrounds

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, easy\_rect, border\_radius=15)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, medium\_rect, border\_radius=15)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, hard\_rect, border\_radius=15)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, back\_rect, border\_radius=15)

# Render and position text elements

screen.blit(easy\_text, (easy\_rect.x + 20, easy\_rect.y + 10))

screen.blit(medium\_text, (medium\_rect.x + 20, medium\_rect.y + 10))

screen.blit(hard\_text, (hard\_rect.x + 20, hard\_rect.y + 10))

screen.blit(back\_text, (back\_rect.x + 20, back\_rect.y + 10))

def draw\_board():

screen.fill(BACKGROUND\_COLOR)

SELECTION\_BACKGROUND\_COLOR = (217, 217, 217) # Color D9D9D9

# Draw board squares and pieces

for row in range(BOARD\_SIZE):

for col in range(BOARD\_SIZE):

color = WHITE if (row + col) % 2 == 0 else GRAY

pygame.draw.rect(screen, color,

(col \* SQUARE\_SIZE + board\_x\_offset, row \* SQUARE\_SIZE + board\_y\_offset,

SQUARE\_SIZE, SQUARE\_SIZE))

piece = board[row][col]

if piece in [BLACK\_PIECE, BLACK\_KING]:

screen.blit(black\_piece, (col \* SQUARE\_SIZE + board\_x\_offset, row \* SQUARE\_SIZE + board\_y\_offset))

if piece == BLACK\_KING:

pygame.draw.circle(screen, WHITE, (col \* SQUARE\_SIZE + board\_x\_offset + SQUARE\_SIZE // 2,

row \* SQUARE\_SIZE + board\_y\_offset + SQUARE\_SIZE // 2), 10)

elif piece in [RED\_PIECE, RED\_KING]:

screen.blit(red\_piece, (col \* SQUARE\_SIZE + board\_x\_offset, row \* SQUARE\_SIZE + board\_y\_offset))

if piece == RED\_KING:

pygame.draw.circle(screen, WHITE, (col \* SQUARE\_SIZE + board\_x\_offset + SQUARE\_SIZE // 2,

row \* SQUARE\_SIZE + board\_y\_offset + SQUARE\_SIZE // 2), 10)

# Define player name text and turn indicator triangle color

player1\_text = font.render(player1\_name, True, player1\_color)

player2\_text = font.render(player2\_name, True, player2\_color)

triangle\_color = BLACK if current\_turn == BLACK else RED

# Position the names and turn indicator triangle based on color selection

if player1\_color == RED:

# Draw rounded background for Player 1 name at the bottom

player1\_name\_rect = pygame.Rect(screen\_width // 2 - player1\_text.get\_width() // 2 - 20, board\_y\_offset + board\_height + 10, player1\_text.get\_width() + 40, player1\_text.get\_height() + 20)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, player1\_name\_rect, border\_radius=15)

screen.blit(player1\_text, (player1\_name\_rect.x + 20, player1\_name\_rect.y + 10))

# Draw rounded background for Player 2 name at the top

player2\_name\_rect = pygame.Rect(screen\_width // 2 - player2\_text.get\_width() // 2 - 20, board\_y\_offset - 60, player2\_text.get\_width() + 40, player2\_text.get\_height() + 20)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, player2\_name\_rect, border\_radius=15)

screen.blit(player2\_text, (player2\_name\_rect.x + 20, player2\_name\_rect.y + 10))

else:

# Draw rounded background for Player 1 name at the top

player1\_name\_rect = pygame.Rect(screen\_width // 2 - player1\_text.get\_width() // 2 - 20, board\_y\_offset - 60, player1\_text.get\_width() + 40, player1\_text.get\_height() + 20)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, player1\_name\_rect, border\_radius=15)

screen.blit(player1\_text, (player1\_name\_rect.x + 20, player1\_name\_rect.y + 10))

# Draw rounded background for Player 2 name at the bottom

player2\_name\_rect = pygame.Rect(screen\_width // 2 - player2\_text.get\_width() // 2 - 20, board\_y\_offset + board\_height + 10, player2\_text.get\_width() + 40, player2\_text.get\_height() + 20)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, player2\_name\_rect, border\_radius=15)

screen.blit(player2\_text, (player2\_name\_rect.x + 20, player2\_name\_rect.y + 10))

# Draw turn indicator triangle next to active player name

triangle\_points = []

if current\_turn == BLACK:

triangle\_points = [

(player1\_name\_rect.x - 30, player1\_name\_rect.y + 20), # top

(player1\_name\_rect.x - 10, player1\_name\_rect.y + 10), # bottom left

(player1\_name\_rect.x - 10, player1\_name\_rect.y + 30) # bottom right

]

else:

triangle\_points = [

(player2\_name\_rect.x - 30, player2\_name\_rect.y + 20), # top

(player2\_name\_rect.x - 10, player2\_name\_rect.y + 10), # bottom left

(player2\_name\_rect.x - 10, player2\_name\_rect.y + 30) # bottom right

]

pygame.draw.polygon(screen, triangle\_color, triangle\_points)

# Draw "Back" button with rounded background

back\_text = font.render("Back", True, BLACK)

back\_rect = pygame.Rect(screen\_width - 150, screen\_height - 50, back\_text.get\_width() + 40, back\_text.get\_height() + 20)

pygame.draw.rect(screen, SELECTION\_BACKGROUND\_COLOR, back\_rect, border\_radius=15)

screen.blit(back\_text, (back\_rect.x + 20, back\_rect.y + 10))

def draw\_winner\_screen():

"""Displays the winner message and option to return to the main menu."""

screen.fill(BACKGROUND\_COLOR)

win\_text = font.render(winner\_message, True, BLACK)

screen.blit(win\_text, (screen\_width // 2 - win\_text.get\_width() // 2, screen\_height // 2 - FONT\_SIZE // 2))

# Option to return to main menu

main\_menu\_text = font.render("Return to Main Menu", True, BLACK)

screen.blit(main\_menu\_text, (screen\_width // 2 - main\_menu\_text.get\_width() // 2, screen\_height // 2 + 50))

def game\_over(board):

black\_moves = get\_valid\_moves(BLACK)

red\_moves = get\_valid\_moves(RED)

if not black\_moves or not red\_moves:

return True

return False

# Main game loop

running = True

initialize\_board()

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.MOUSEBUTTONDOWN:

mouse\_x, mouse\_y = pygame.mouse.get\_pos()

y\_start = screen\_height // 2 - 200 # Define y\_start within the main loop

if game\_state == STATE\_MAIN\_MENU:

player1\_name = ""

player2\_name = ""

bot\_game = False

active\_input = None

if screen\_height // 2 - 100 <= mouse\_y <= screen\_height // 2 - 60:

game\_state = STATE\_PLAYER\_SETUP

elif screen\_height // 2 <= mouse\_y <= screen\_height // 2 + 40:

game\_state = STATE\_DIFFICULTY\_SELECTION

elif screen\_height // 2 + 100 <= mouse\_y <= screen\_height // 2 + 140:

pygame.quit()

sys.exit()

elif game\_state == STATE\_PLAYER\_SETUP:

if y\_start <= mouse\_y <= y\_start + 40 and screen\_width // 2 <= mouse\_x <= screen\_width // 2 + 200:

active\_input = "player1"

elif y\_start + 100 <= mouse\_y <= y\_start + 140 and screen\_width // 2 <= mouse\_x <= screen\_width // 2 + 200:

active\_input = "player2"

elif y\_start + 250 - 20 <= mouse\_y <= y\_start + 250 + 20:

if screen\_width // 2 - 50 - 20 <= mouse\_x <= screen\_width // 2 - 50 + 20:

player1\_color, player2\_color = BLACK, RED

elif screen\_width // 2 + 50 - 20 <= mouse\_x <= screen\_width // 2 + 50 + 20:

player1\_color, player2\_color = RED, BLACK

elif y\_start + 400 <= mouse\_y <= y\_start + 440:

board = [[None for \_ in range(BOARD\_SIZE)] for \_ in range(BOARD\_SIZE)]

initialize\_board()

current\_turn = BLACK

selected\_piece = None

bot\_game = False

game\_state = STATE\_GAME

elif y\_start + 450 <= mouse\_y <= y\_start + 490:

game\_state = STATE\_MAIN\_MENU

elif game\_state == STATE\_BOT\_SETUP:

if y\_start <= mouse\_y <= y\_start + 40 and screen\_width // 2 <= mouse\_x <= screen\_width // 2 + 200:

active\_input = "player1"

elif y\_start + 250 - 20 <= mouse\_y <= y\_start + 250 +20:

if screen\_width // 2 - 50 - 20 <= mouse\_x <= screen\_width // 2 - 50 + 20:

player1\_color, bot\_color = BLACK, RED

elif screen\_width // 2 + 50 - 20 <= mouse\_x <= screen\_width // 2 + 50 + 20:

player1\_color, bot\_color = RED, BLACK

elif y\_start + 400 <= mouse\_y <= y\_start + 440:

player2\_name = f"{bot\_difficulty.capitalize()} Bot"

player2\_color = bot\_color

board = [[None for \_ in range(BOARD\_SIZE)] for \_ in range(BOARD\_SIZE)]

initialize\_board()

current\_turn = player1\_color

selected\_piece = None

game\_state = STATE\_GAME

elif y\_start + 450 <= mouse\_y <= y\_start + 490:

game\_state = STATE\_MAIN\_MENU

elif game\_state == STATE\_DIFFICULTY\_SELECTION:

y\_start = screen\_height // 2 - 100

if y\_start <= mouse\_y <= screen\_height // 2 + 40:

bot\_game = True

bot\_difficulty = 'easy' # easy bot game

game\_state = STATE\_BOT\_SETUP

elif y\_start +100 <= mouse\_y <= screen\_height // 2 + 140:

bot\_game = True

bot\_difficulty = 'medium' # medium bot game

game\_state = STATE\_BOT\_SETUP

elif y\_start + 200 <= mouse\_y <= screen\_height // 2 + 240:

bot\_game = True

bot\_difficulty = 'hard' # hard bot game

game\_state = STATE\_BOT\_SETUP

elif y\_start + 300 <= mouse\_y <= screen\_height // 2 + 340:

game\_state = STATE\_MAIN\_MENU

elif game\_state == STATE\_GAME:

row, col = get\_square\_under\_mouse()

if row is not None and col is not None:

if selected\_piece:

if is\_valid\_move(selected\_piece, (row, col)):

old\_row, old\_col = selected\_piece

if abs(row - old\_row) == 2:

mid\_row = (old\_row +row) // 2

mid\_col = (old\_col + col) // 2

board[mid\_row][mid\_col] = None

board[row][col] = board[old\_row][old\_col]

board[old\_row][old\_col] = None

check\_for\_king(row, col)

selected\_piece = None

switch\_turn()

check\_for\_winner() # Check for a winner after each valid move

else:

invalid\_move\_timer = {"position": (row, col), "start\_time": time.time()}

selected\_piece = None

elif board[row][col] and ((board[row][col] in [BLACK\_PIECE, BLACK\_KING] and current\_turn == BLACK) or (board[row][col] in [RED\_PIECE, RED\_KING] and current\_turn == RED)):

selected\_piece = (row, col)

elif screen\_width - 150 <= mouse\_x <= screen\_width - 50 and screen\_height - 50 <= mouse\_y <= screen\_height - 20:

game\_state = STATE\_MAIN\_MENU

elif game\_state == STATE\_WINNER:

# If on the winner screen, check if the player clicks to return to the main menu

main\_menu\_text = font.render("Return to Main Menu", True, BLACK)

if screen\_width // 2 - main\_menu\_text.get\_width() // 2 <= mouse\_x <= screen\_width // 2 + main\_menu\_text.get\_width() // 2 and \

screen\_height // 2 + 50 <= mouse\_y <= screen\_height // 2 + 50 + FONT\_SIZE:

game\_state = STATE\_MAIN\_MENU

elif event.type == pygame.KEYDOWN and active\_input:

if event.key == pygame.K\_BACKSPACE:

if active\_input == "player1":

player1\_name = player1\_name[:-1]

elif active\_input == "player2":

player2\_name = player2\_name[:-1]

elif event.unicode.isprintable():

if active\_input == "player1" and len(player1\_name) < 10:

player1\_name += event.unicode

elif active\_input == "player2" and len(player2\_name) < 10:

player2\_name += event.unicode

if game\_state == STATE\_GAME and bot\_game and current\_turn == bot\_color:

bot\_moves()

if invalid\_move\_timer and time.time() - invalid\_move\_timer["start\_time"] > 2:

invalid\_move\_timer = None

if game\_state == STATE\_MAIN\_MENU:

draw\_main\_menu()

elif game\_state == STATE\_PLAYER\_SETUP:

draw\_player\_setup()

elif game\_state == STATE\_BOT\_SETUP:

draw\_bot\_setup()

elif game\_state == STATE\_DIFFICULTY\_SELECTION:

draw\_difficulty\_selection()

elif game\_state == STATE\_GAME:

draw\_board()

elif game\_state == STATE\_WINNER:

draw\_winner\_screen()

pygame.display.flip()