***Overview:***

The code represents a simple chess game where the player (user) controls a king and a boat against an opponent king. The game has a graphical interface created using the Pygame library.

# Import necessary libraries

import pygame

from pygame.locals import \*

import sys

import random

import time

# Constants for the game window and colors

WIDTH, HEIGHT = 700, 700

SQUARE\_SIZE = WIDTH // 8

BROWN = (139, 69, 19)

PALE\_WHITE = (255, 222, 173)

LIGHT\_GREEN = (144, 238, 144)

FONT = pygame.font.Font(None, 36)

# Define the positions and initializations

user\_king\_col, user\_king\_row = 4, 7

user\_boat\_col, user\_boat\_row = 0, 7

opponent\_king\_col, opponent\_king\_row = 4, 0

user\_points = 1000

opponent\_king\_kills = 0

# Create the chessboard surface

chessboard = pygame.Surface((WIDTH, HEIGHT))

# ... (chessboard initialization)

* ***Load and Resize Images:***

Images for user king, user boat, and opponent king are loaded and resized.

# Load and resize images for pieces

user\_king\_img = pygame.image.load("king.png")

user\_king\_img = pygame.transform.scale(user\_king\_img, (SQUARE\_SIZE, SQUARE\_SIZE))

user\_boat\_img = pygame.image.load("boat.png")

user\_boat\_img = pygame.transform.scale(user\_boat\_img, (SQUARE\_SIZE, SQUARE\_SIZE))

opponent\_king\_img = pygame.image.load("opponent\_king.png")

opponent\_king\_img = pygame.transform.scale(opponent\_king\_img, (SQUARE\_SIZE, SQUARE\_SIZE))

* ***Create Pygame Window:***

Pygame window is created with a specified caption and size.

# Create a Pygame window

pygame.display.set\_caption("Chess Game")

window = pygame.display.set\_mode((WIDTH + 300, HEIGHT))

* ***Game Variables and Functions:***

Various game variables, functions, and constants are defined.

selected\_piece = None

user\_turn = True

highlighted\_squares = []

KING\_MOVE\_COST = 10

BOAT\_MOVE\_COST = 20

# ... (other functions and constants)

* ***Move Validation Functions:***

Functions to calculate valid moves for the king and boat are defined.

def calculate\_valid\_moves(piece\_col, piece\_row, piece\_type, user\_king\_col, user\_king\_row, user\_boat\_col, user\_boat\_row):

# ... (valid move calculations)

def evaluate\_game\_state(user\_king\_col, user\_king\_row, user\_boat\_col, user\_boat\_row):

# ... (heuristic evaluation)

* ***Minimax Algorithm:***

Minimax algorithm is implemented for the opponent's decision-making.

def minimax(opponent\_king\_col, opponent\_king\_row, user\_king\_col, user\_king\_row, user\_boat\_col, user\_boat\_row, depth):

# ... (minimax algorithm implementation)

* Opponent's Turn Function:

Function to handle the opponent's turn.

def opponent\_turn():

# ... (opponent's turn logic)

* ***Game Over Functions:***

Functions to display game over message and check game over conditions.

def display\_game\_over\_message(result):

# ... (display game over message)

def game\_over(result):

# ... (game over conditions)

* ***Highlight and Move Functions:***

Functions to highlight valid moves, move the selected piece, and handle piece removal.

def highlight\_valid\_moves(valid\_moves):

# ... (highlight valid moves)

def move\_piece(selected\_piece, col, row):

# ... (move the selected piece)

def remove\_piece\_with\_animation(col, row):

# ... (remove a piece with animation)

* ***Main Game Loop:***

The main game loop handles user and opponent turns, checks for collisions, updates the display, and checks for game over conditions.

while True:

for event in pygame.event.get():

# ... (event handling in the main loop)

# Check if the user's king or boat has been killed by the opponent's king

# ... (check collisions)

# Draw user's points within the larger box

# ... (draw user's points)

pygame.display.update()

***Conclusion:***

The main game loop runs indefinitely, handling user and opponent turns, updating the display, and checking for game over conditions.

**The provided code demonstrates the implementation of a simple chess game using Pygame. It includes functionality for moving pieces, calculating valid moves, using the minimax algorithm for opponent decision-making, and handling game over conditions. The code structure is organized, making it easy to understand and extend for future development.**