Sudoku Game in Python with Pygame - Code and Explanation

import pygame  
import requests  
import time  
import sys  
from threading import Thread

These are the necessary imports. `pygame` is used for the game interface, `requests` fetches the puzzle, and `time`, `sys`, and `Thread` are standard Python modules used for timing and system operations.

pygame.init()

Initializes all imported Pygame modules.

# Constants  
WIDTH, HEIGHT = 600, 700  
GRID\_POS = (50, 100)  
CELL\_SIZE = 60  
GRID\_SIZE = CELL\_SIZE \* 9  
BUTTON\_AREA\_Y = 600

Defines constants for screen size, grid position and cell size. These help in laying out the interface properly.

screen = pygame.display.set\_mode((WIDTH, HEIGHT))  
pygame.display.set\_caption("Sudoku Game")  
font = pygame.font.SysFont("comicsans", 40)  
small\_font = pygame.font.SysFont("comicsans", 24)  
clock = pygame.time.Clock()

Sets up the Pygame window, fonts for rendering text, and a clock to manage the game frame rate.

# Colors  
WHITE = (255, 255, 255)  
LIGHT\_GRAY = (240, 240, 240)  
BLACK = (0, 0, 0)  
BLUE = (50, 100, 250)  
GREEN = (20, 180, 50)  
RED = (220, 30, 30)

Defines RGB color tuples used in the UI for tiles, highlights, and text.

def fetch\_puzzle():  
 try:  
 response = requests.get("https://sugoku.onrender.com/board?difficulty=easy")  
 return response.json()['board']  
 except:  
 return [[0]\*9 for \_ in range(9)]

Fetches a Sudoku puzzle from an online API. If the request fails, returns a blank board.

original\_grid = fetch\_puzzle()  
grid = [row[:] for row in original\_grid]  
selected = None  
start\_time = time.time()

Initializes the game board and copies the puzzle into a working grid. Also starts a game timer.

def draw\_grid():  
 for i in range(10):  
 thick = 4 if i % 3 == 0 else 1  
 pygame.draw.line(screen, BLACK, (GRID\_POS[0] + i\*CELL\_SIZE, GRID\_POS[1]),  
 (GRID\_POS[0] + i\*CELL\_SIZE, GRID\_POS[1]+GRID\_SIZE), thick)  
 pygame.draw.line(screen, BLACK, (GRID\_POS[0], GRID\_POS[1] + i\*CELL\_SIZE),  
 (GRID\_POS[0]+GRID\_SIZE, GRID\_POS[1]+i\*CELL\_SIZE), thick)

Draws the Sudoku board grid. Thicker lines are used to separate the 3x3 boxes.

def draw\_numbers():  
 for i in range(9):  
 for j in range(9):  
 if grid[i][j] != 0:  
 color = BLACK if original\_grid[i][j] != 0 else BLUE  
 text = font.render(str(grid[i][j]), True, color)  
 screen.blit(text, (GRID\_POS[0] + j\*CELL\_SIZE + 20, GRID\_POS[1] + i\*CELL\_SIZE + 10))

Draws the numbers onto the board. Original puzzle numbers are black; user inputs are blue.

def draw\_selection():  
 if selected:  
 pygame.draw.rect(screen, RED, (GRID\_POS[0] + selected[1]\*CELL\_SIZE, GRID\_POS[1] + selected[0]\*CELL\_SIZE, CELL\_SIZE, CELL\_SIZE), 3)

Draws a red rectangle around the currently selected cell.

def draw\_buttons():  
 buttons = ["Check", "Solve", "Hint", "Reset", "Exit"]  
 for i, label in enumerate(buttons):  
 pygame.draw.rect(screen, LIGHT\_GRAY, (50 + i\*105, BUTTON\_AREA\_Y, 100, 40), border\_radius=5)  
 text = small\_font.render(label, True, BLACK)  
 screen.blit(text, (50 + i\*105 + 20, BUTTON\_AREA\_Y + 10))

Draws functional buttons below the Sudoku board.

def get\_cell(pos):  
 if GRID\_POS[0] <= pos[0] < GRID\_POS[0]+GRID\_SIZE and GRID\_POS[1] <= pos[1] < GRID\_POS[1]+GRID\_SIZE:  
 return ((pos[1] - GRID\_POS[1]) // CELL\_SIZE, (pos[0] - GRID\_POS[0]) // CELL\_SIZE)  
 return None

Converts a mouse click position into grid cell indices.

def is\_valid(num, row, col):  
 for i in range(9):  
 if grid[row][i] == num or grid[i][col] == num:  
 return False  
 start\_row, start\_col = 3\*(row//3), 3\*(col//3)  
 for i in range(3):  
 for j in range(3):  
 if grid[start\_row+i][start\_col+j] == num:  
 return False  
 return True

Checks if placing a number in the grid is valid according to Sudoku rules.

def solve\_sudoku():  
 for i in range(9):  
 for j in range(9):  
 if grid[i][j] == 0:  
 for num in range(1, 10):  
 if is\_valid(num, i, j):  
 grid[i][j] = num  
 if solve\_sudoku():  
 return True  
 grid[i][j] = 0  
 return False  
 return True

Solves the Sudoku board using backtracking.

def give\_hint():  
 for i in range(9):  
 for j in range(9):  
 if grid[i][j] == 0:  
 for num in range(1, 10):  
 if is\_valid(num, i, j):  
 grid[i][j] = num  
 return

Provides a hint by filling the first valid empty cell.

def check\_solution():  
 for i in range(9):  
 for j in range(9):  
 if grid[i][j] == 0 or not is\_valid(grid[i][j], i, j):  
 return False  
 return True

Checks if the entire grid is filled correctly.

def draw\_timer():  
 elapsed = int(time.time() - start\_time)  
 minutes, seconds = divmod(elapsed, 60)  
 time\_text = small\_font.render(f"Time: {minutes:02d}:{seconds:02d}", True, BLACK)  
 screen.blit(time\_text, (WIDTH - 150, 20))

Draws a live timer showing how long the game has been played.

# Main Loop  
running = True  
while running:  
 screen.fill(WHITE)  
 draw\_grid()  
 draw\_numbers()  
 draw\_selection()  
 draw\_buttons()  
 draw\_timer()  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 running = False  
  
 elif event.type == pygame.MOUSEBUTTONDOWN:  
 pos = pygame.mouse.get\_pos()  
 selected = get\_cell(pos)  
  
 # Button clicks  
 if BUTTON\_AREA\_Y <= pos[1] <= BUTTON\_AREA\_Y + 40:  
 if 50 <= pos[0] < 150:  
 result = check\_solution()  
 print("✅ Correct!" if result else "❌ Incorrect")  
 elif 155 <= pos[0] < 255:  
 solve\_sudoku()  
 elif 260 <= pos[0] < 360:  
 give\_hint()  
 elif 365 <= pos[0] < 465:  
 grid = [row[:] for row in original\_grid]  
 elif 470 <= pos[0] < 570:  
 pygame.quit()  
 sys.exit()  
  
 elif event.type == pygame.KEYDOWN and selected:  
 row, col = selected  
 if original\_grid[row][col] == 0:  
 if event.unicode.isdigit() and event.unicode != '0':  
 grid[row][col] = int(event.unicode)  
 elif event.key == pygame.K\_BACKSPACE:  
 grid[row][col] = 0  
  
 pygame.display.flip()  
 clock.tick(30)  
  
pygame.quit()

Main game loop: handles rendering, input events, and calls the appropriate game functions based on user actions.