

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"



**АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ КОМП'ЮТЕРНИХ
СИСТЕМ**

Лабораторна робота №3

Виконав: ст. гр. КІ-405
Легкобит Н.В

Прийняв:
Шпіцер А.С

Львів 2024

Тема: Імплементація Ігри до клієнт серверу

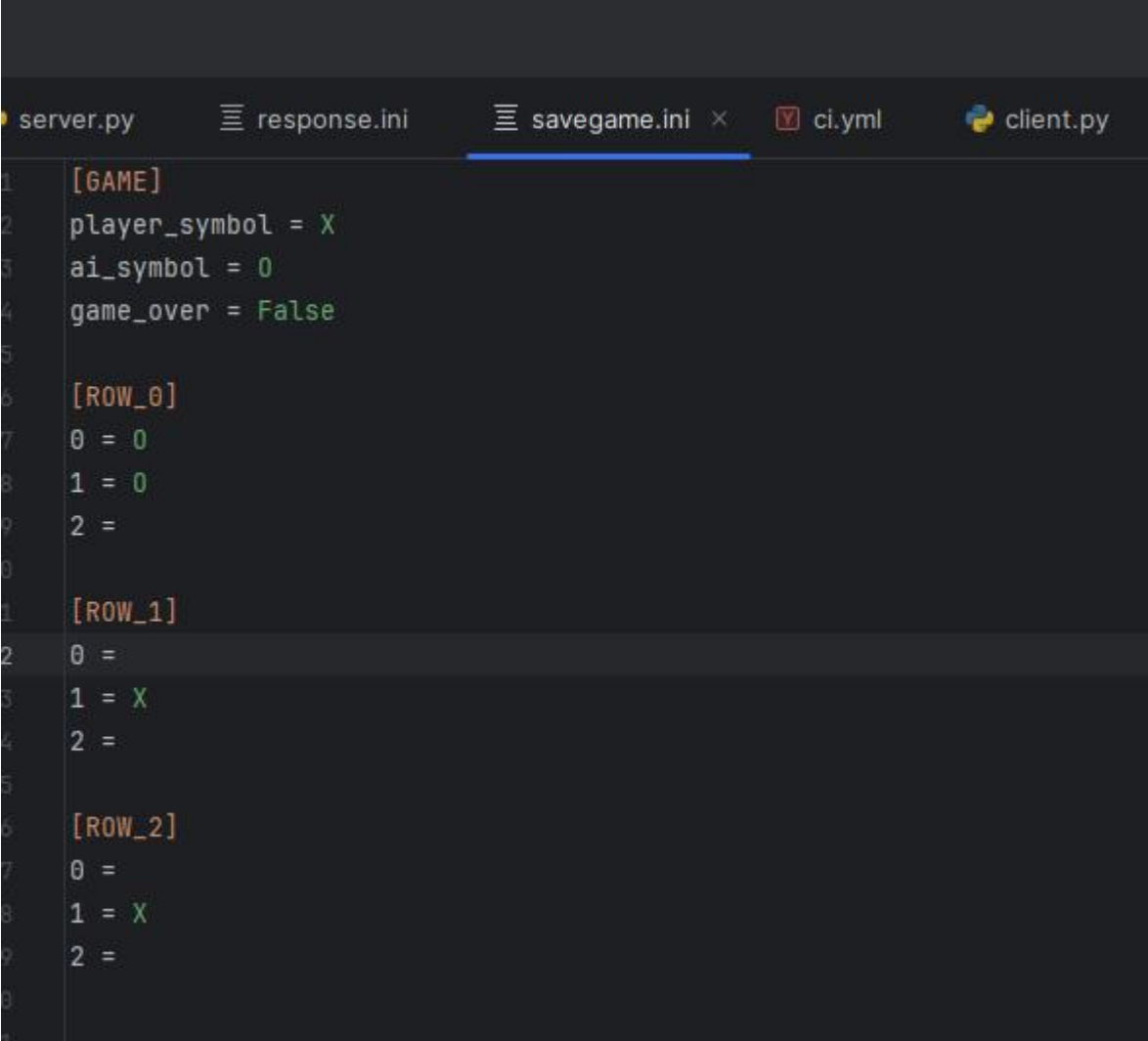
Порядок виконання лабораторної роботи:

1. Develop Server and Client.
2. Required steps.

Виконання роботи

1. Збереження конфігурації в форматі INI

Для збереження стану гри "Хрестики-Нулики" використано формат INI. Це дозволило ефективно зберігати поточний стан поля, символи гравців та інформацію про завершення гри. Бібліотека Python ConfigParser забезпечила зручну роботу з цими даними, що зберігаються у файлі `savegame.ini`.



```
server.py  response.ini  savegame.ini x  ci.yml  client.py
1  [GAME]
2  player_symbol = X
3  ai_symbol = 0
4  game_over = False
5
6  [ROW_0]
7  0 = 0
8  1 = 0
9  2 =
10
11 [ROW_1]
12 0 =
13 1 = X
14 2 =
15
16 [ROW_2]
17 0 =
18 1 = X
19 2 =
```

ої частини

Розробка серверної частини: Серверна частина, що виконується на порту

COM11, відповідає за обробку запитів від клієнта, таких як "новий хід", "збереження" та "завантаження". Сервер отримує запити від клієнта, обробляє їх та генерує відповідь, яка включає поточний стан гри, інформацію про черговий хід, результат останнього ходу та оновлені рахунки. Залежно від отриманих команд, сервер:

1. Створює нову гру та обнуляє рахунки.
2. Виконує гру в режимі "гравець проти AI", де AI вибирає випадковий хід у межах доступних клітинок. Реалізовано логіку перевірки переможця на основі розташування хрестиків і нуликів на ігровому полі.

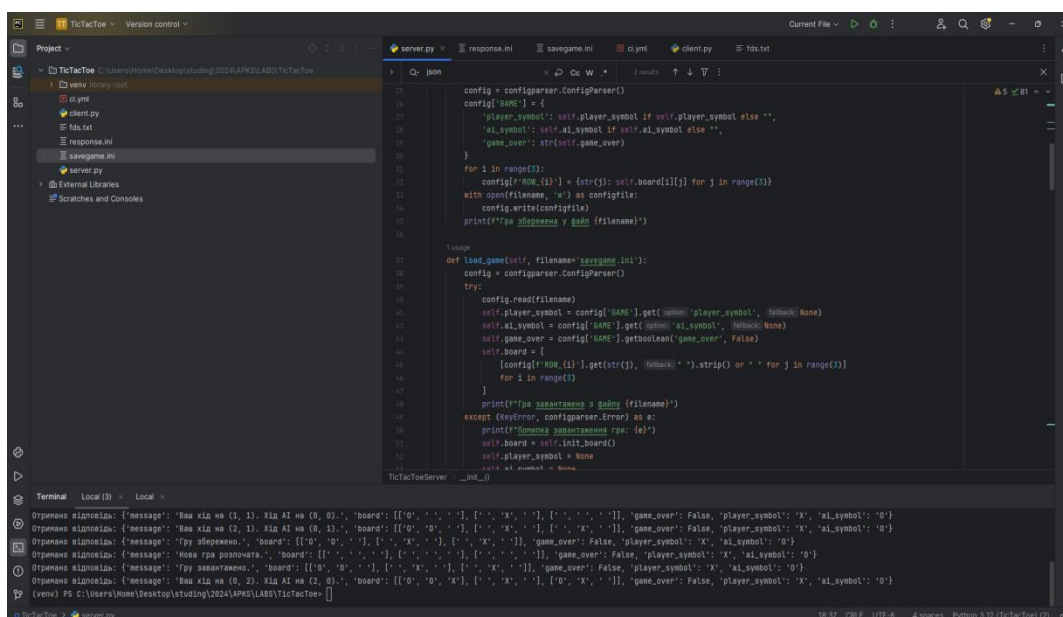


Рис. 2. Серверна частина

3. Розробка клієнтської частини

Розробка клієнтської частини: Клієнтська частина, що виконується на порту COM12, забезпечує текстовий інтерфейс для взаємодії з користувачем. Клієнт надає меню, в якому можна обрати наступні дії:

- Розпочати нову гру.
- Зберегти поточний стан гри.
- Завантажити збережений стан гри.
- Зробити хід у грі (вибір вільної клітинки на полі). Після вибору клітинки користувачем клієнт надсилає команду на сервер і отримує відповідь з оновленим станом гри, який відображається на екрані.

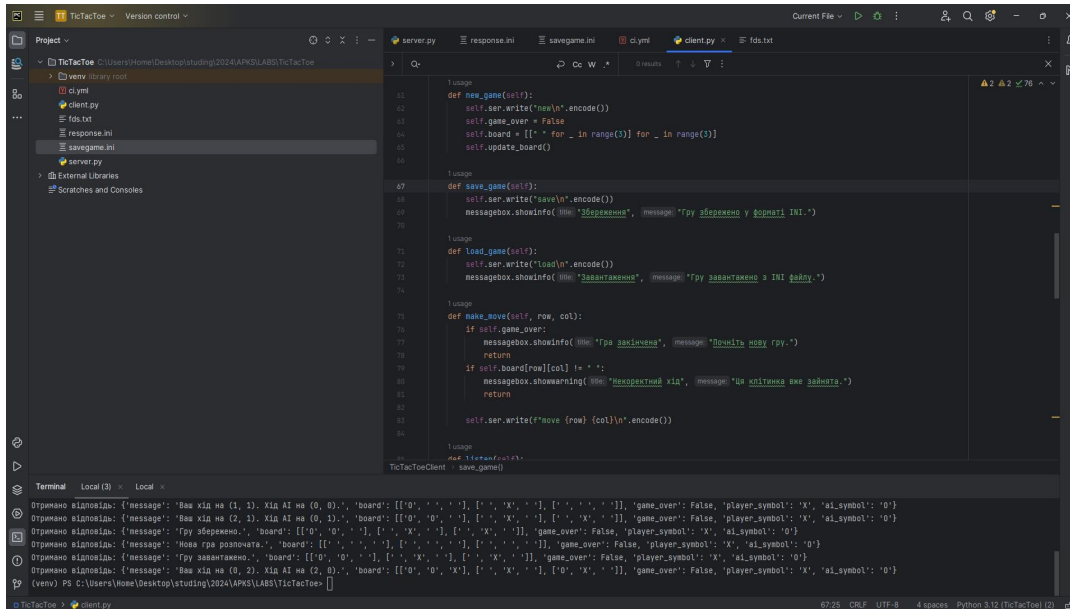


Рис. 3. Клієнтська частина

4. Збереження та завантаження стану гри

Збереження та завантаження стану гри: Використовуючи ini-файл savegame.ini, сервер зберігає та відновлює конфігурацію гри. Це дозволяє користувачеві зберігати прогрес ігрового поля та продовжувати гру з того місця, де він зупинився. Наприклад, якщо гравець вийде з гри, він може завантажити попередній стан і продовжити гру з попереднім розташуванням знаків на полі та черговістю ходу.

5. Тестування гри

Тестування гри: Проведено тестування роботи клієнт-серверної взаємодії, включаючи перевірку команд "новий хід", "збереження" та "завантаження". Клієнт успішно надсилає запити на сервер, який обробляє їх та повертає правильні відповіді. Переконався, що збереження та завантаження стану гри відбувається коректно, а сервер правильно обробляє результати ходів та оновлює рахунки.

6. Автоматизація тестування за допомогою GitHub Actions

Оновлено файл ci.yml для автоматизації процесу тестування гри у середовищі GitHub Actions. Це забезпечило автоматичне виконання тестів при

кожному оновленні коду, що сприяє підвищенню якості програмного забезпечення.

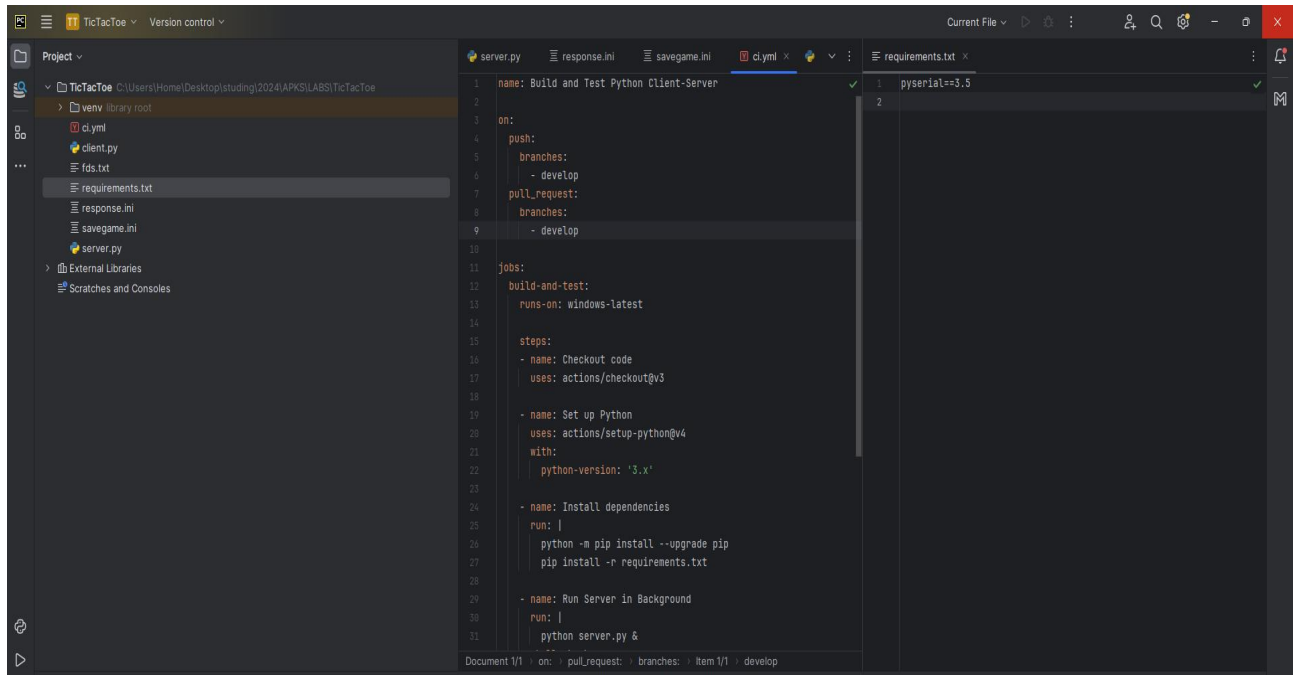


Рис. 4. YML та requirments.txt

Додатки

server.py

```
import configparser

import serial
import json
import threading

class TicTacToeServer:
    def __init__(self, port='COM11', baudrate=9600):
        self.ser = serial.Serial(port, baudrate, timeout=1)
        print(f"Сервер 'Хрестики-Нулики' запущено на {port}")
        self.board = self.init_board()
        self.game_over = False
        self.lock = threading.Lock()
        self.player_symbol = None
        self.ai_symbol = None
        # Видалено автоматичне завантаження конфігурації
        self.listen_thread = threading.Thread(target=self.listen)
        self.listen_thread.start()

    def init_board(self):
        return [["_ " for _ in range(3)] for _ in range(3)]
```

```

def save_game(self, filename='savegame.ini'):
    config = configparser.ConfigParser()
    config['GAME'] = {
        'player_symbol': self.player_symbol if self.player_symbol else "",
        'ai_symbol': self.ai_symbol if self.ai_symbol else "",
        'game_over': str(self.game_over)
    }
    for i in range(3):
        config[f'ROW_{i}'] = {str(j): self.board[i][j] for j in range(3)}
    with open(filename, 'w') as configfile:
        config.write(configfile)
    print(f"Гра збережена у файл {filename}")

def load_game(self, filename='savegame.ini'):
    config = configparser.ConfigParser()
    try:
        config.read(filename)
        self.player_symbol = config['GAME'].get('player_symbol', None)
        self.ai_symbol = config['GAME'].get('ai_symbol', None)
        self.game_over = config['GAME'].getboolean('game_over', False)
        self.board = [
            [config[f'ROW_{i}'].get(str(j), " ").strip() or " " for j in range(3)]
            for i in range(3)
        ]
        print(f"Гра завантажена з файлу {filename}")
    except (KeyError, configparser.Error) as e:
        print(f"Помилка завантаження гри: {e}")
        self.board = self.init_board()
        self.player_symbol = None
        self.ai_symbol = None
        self.game_over = False

def check_winner(self, board, symbol):
    lines = (
        # Рядки
        board[0], board[1], board[2],
        # Стовпці
        [board[0][0], board[1][0], board[2][0]],
        [board[0][1], board[1][1], board[2][1]],
        [board[0][2], board[1][2], board[2][2]],
        # Діагоналі
        [board[0][0], board[1][1], board[2][2]],
        [board[0][2], board[1][1], board[2][0]]
    )
    for line in lines:
        if all(cell == symbol for cell in line):
            return True
    return False

def check_tie(self, board):
    return all(cell != " " for row in board for cell in row)

```

```

def minimax(self, board, depth, is_maximizing):
    if self.check_winner(board, self.ai_symbol):
        return 1
    if self.check_winner(board, self.player_symbol):
        return -1
    if self.check_tie(board):
        return 0

    if is_maximizing:
        best_score = -float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = self.ai_symbol
                    score = self.minimax(board, depth + 1, False)
                    board[i][j] = " "
                    best_score = max(score, best_score)
            return best_score
    else:
        best_score = float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = self.player_symbol
                    score = self.minimax(board, depth + 1, True)
                    board[i][j] = " "
                    best_score = min(score, best_score)
            return best_score

def make_ai_move(self):
    best_score = -float('inf')
    move = None
    for i in range(3):
        for j in range(3):
            if self.board[i][j] == " ":
                self.board[i][j] = self.ai_symbol
                score = self.minimax(self.board, 0, False)
                self.board[i][j] = " "
                if score > best_score:
                    best_score = score
                    move = (i, j)
    if move:
        self.board[move[0]][move[1]] = self.ai_symbol
        return move
    return None

def process_command(self, command):
    response = {}
    if command.startswith("start"):
        parts = command.split()

```



```

        self.game_over = True
    else:
        response['message'] = f"Ваш хід на ({row}, {col}). Хід AI на ({ai_move[0]}, {ai_move[1]})."
    else:
        response['message'] = "Клітинка зайнята. Спробуйте ще раз."
    else:
        response['message'] = "Некоректні координати. Використовуйте числа від 0 до 2."

    except ValueError:
        response['message'] = "Некоректний формат команди."
    else:
        response['message'] = "Некоректний формат команди."
    else:
        response['message'] = "Невідома команда."
    # Завжди додаємо дошку та стан гри до відповіді
    response['board'] = self.board
    response['game_over'] = self.game_over
    response['player_symbol'] = self.player_symbol
    response['ai_symbol'] = self.ai_symbol
    response_str = json.dumps(response) + '\n'
    return response_str

def listen(self):
    try:
        while True:
            if self.ser.in_waiting > 0:
                command = self.ser.readline().decode().strip()
                if command:
                    print(f"Отримано команду: {command}")
                    with self.lock:
                        response = self.process_command(command)
                        self.ser.write(response.encode())
            except Exception as e:
                print(f"Помилка в потоці прослуховування сервера: {e}")
            finally:
                self.ser.close()
                # Видалено автоматичне збереження конфігурації

if __name__ == "__main__":
    server = TicTacToeServer()

```

client.py

```

import tkinter as tk
from tkinter import messagebox, simpledialog
import serial
import json
import threading

```

```

class TicTacToeClient:
    def __init__(self, port='COM12', baudrate=9600):
        self.root = tk.Tk()
        self.root.title("Хрестики-Нулики Клієнт")
        self.ser = serial.Serial(port, baudrate, timeout=1)
        print(f"Клієнт 'Хрестики-Нулики' запущено на {port}")
        self.board = [[" " for _ in range(3)] for _ in range(3)]
        self.player_symbol = None
        self.ai_symbol = None
        self.game_over = False
        self.running = True # Прапорець для контролю потоку
        self.create_widgets()
        self.listen_thread = threading.Thread(target=self.listen)
        self.listen_thread.daemon = True
        self.listen_thread.start()
        self.root.protocol("WM_DELETE_WINDOW", self.close)
        self.ask_player_symbol()
        self.root.mainloop()

    def ask_player_symbol(self):
        symbol = None
        while symbol not in ['X', 'O']:
            symbol = simpledialog.askstring("Вибір символу", "Виберіть ваш символ (X або O):",
parent=self.root)
            if symbol is None:
                self.close()
                return
            symbol = symbol.upper()
            if symbol not in ['X', 'O']:
                messagebox.showwarning("Невірний вибір", "Будь ласка, введіть 'X' або 'O'.")
        self.player_symbol = symbol
        self.ai_symbol = 'O' if self.player_symbol == 'X' else 'X'
        self.ser.write(f"start {self.player_symbol}\n".encode())

    def create_widgets(self):
        # Створення кнопок для поля
        self.buttons = [[None for _ in range(3)] for _ in range(3)]
        for i in range(3):
            for j in range(3):
                button = tk.Button(self.root, text=" ", font=("Arial", 24), width=5, height=2,
                                command=lambda row=i, col=j: self.make_move(row, col))
                button.grid(row=i, column=j)
                self.buttons[i][j] = button

        # Кнопки керування
        new_game_button = tk.Button(self.root, text="Нова гра", command=self.new_game)
        new_game_button.grid(row=3, column=0, columnspan=3, sticky="we")

        save_game_button = tk.Button(self.root, text="Зберегти гру", command=self.save_game)

```

```

save_game_button.grid(row=4, column=0, columnspan=3, sticky="we")

load_game_button = tk.Button(self.root, text="Завантажити гру",
command=self.load_game)
load_game_button.grid(row=5, column=0, columnspan=3, sticky="we")

def new_game(self):
    self.ser.write("new\n".encode())
    self.game_over = False
    self.board = [[" " for _ in range(3)] for _ in range(3)]
    self.update_board()

def save_game(self):
    self.ser.write("save\n".encode())
    messagebox.showinfo("Збереження", "Гру збережено у форматі INI.")

def load_game(self):
    self.ser.write("load\n".encode())
    messagebox.showinfo("Завантаження", "Гру завантажено з INI файлу.")

def make_move(self, row, col):
    if self.game_over:
        messagebox.showinfo("Гра закінчена", "Почніть нову гру.")
        return
    if self.board[row][col] != " ":
        messagebox.showwarning("Некоректний хід", "Ця клітинка вже зайнята.")
        return

    self.ser.write(f"move {row} {col}\n".encode())

def listen(self):
    try:
        while self.running:
            if self.ser.in_waiting > 0:
                try:
                    response_str = self.ser.readline().decode().strip()
                    if response_str:
                        try:
                            response = json.loads(response_str)
                            print(f"Отримано відповідь: {response}")
                            self.process_response(response)
                        except json.JSONDecodeError as e:
                            messagebox.showerror("Помилка", f"Невірний формат відповіді від сервера: {e}")

                    except serial.SerialException as e:
                        print(f"Помилка в потоці прослуховування клієнта: {e}")
                        break
                except Exception as e:
                    print(f"Помилка в потоці прослуховування клієнта: {e}")

def process_response(self, response):

```

```

message = response.get('message', '')
board = response.get('board', None)
game_over = response.get('game_over', False)
player_symbol = response.get('player_symbol', self.player_symbol)
ai_symbol = response.get('ai_symbol', self.ai_symbol)

# Оновлюємо символи гравця та AI, якщо вони змінилися
self.player_symbol = player_symbol
self.ai_symbol = ai_symbol

if board:
    self.board = board
    self.update_board()
if message:
    if "переміг" in message or "Нічия" in message:
        messagebox.showinfo("Результат гри", message)
        self.game_over = True
    elif "Клітинка зайнята" in message or "Некоректні координати" in message:
        messagebox.showwarning("Помилка", message)
    elif "Гра розпочата" in message or "Нова гра розпочата" in message or "Гру
завантажено" in message:
        messagebox.showinfo("Хрестики-Нулики", message)
        self.game_over = game_over
    else:
        messagebox.showinfo("Інформація", message)
        self.game_over = game_over
else:
    self.game_over = game_over

def update_board(self):
    for i in range(3):
        for j in range(3):
            self.buttons[i][j].config(text=self.board[i][j])
    self.root.update_idletasks()

def close(self):
    self.running = False
    if self.ser.is_open:
        self.ser.close()
    self.listen_thread.join()
    self.root.quit()

if __name__ == "__main__":
    client = TicTacToeClient()

```

ci.yml

name: Build and Test Python Client-Server

```
on:
  push:
    branches:
      - develop
  pull_request:
    branches:
      - develop

jobs:
  build-and-test:
    runs-on: windows-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.x'

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt

      - name: Run Server in Background
        run: |
          python server.py &
        shell: bash

      - name: Run Client Test
        run: python client.py

      - name: Cleanup
        run: |
          pkill -f server.py || true
          pkill -f client.py || true
        shell: bash
```