**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**



**АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ КОМП’ЮТЕРНИХ СИСТЕМ**

Лабораторна робота 4

Виконала: ст. гр. КІ-401

Оприш О.О.

Прийняв:

Федак П.Р.

Львів 2024

**Тема: Створення Doxygen документації**

**Порядок виконання лабораторної роботи:**

Task 4. Create doxygen documentation:

1. Add doxygen comments for all public functions, classes, properties, fields...

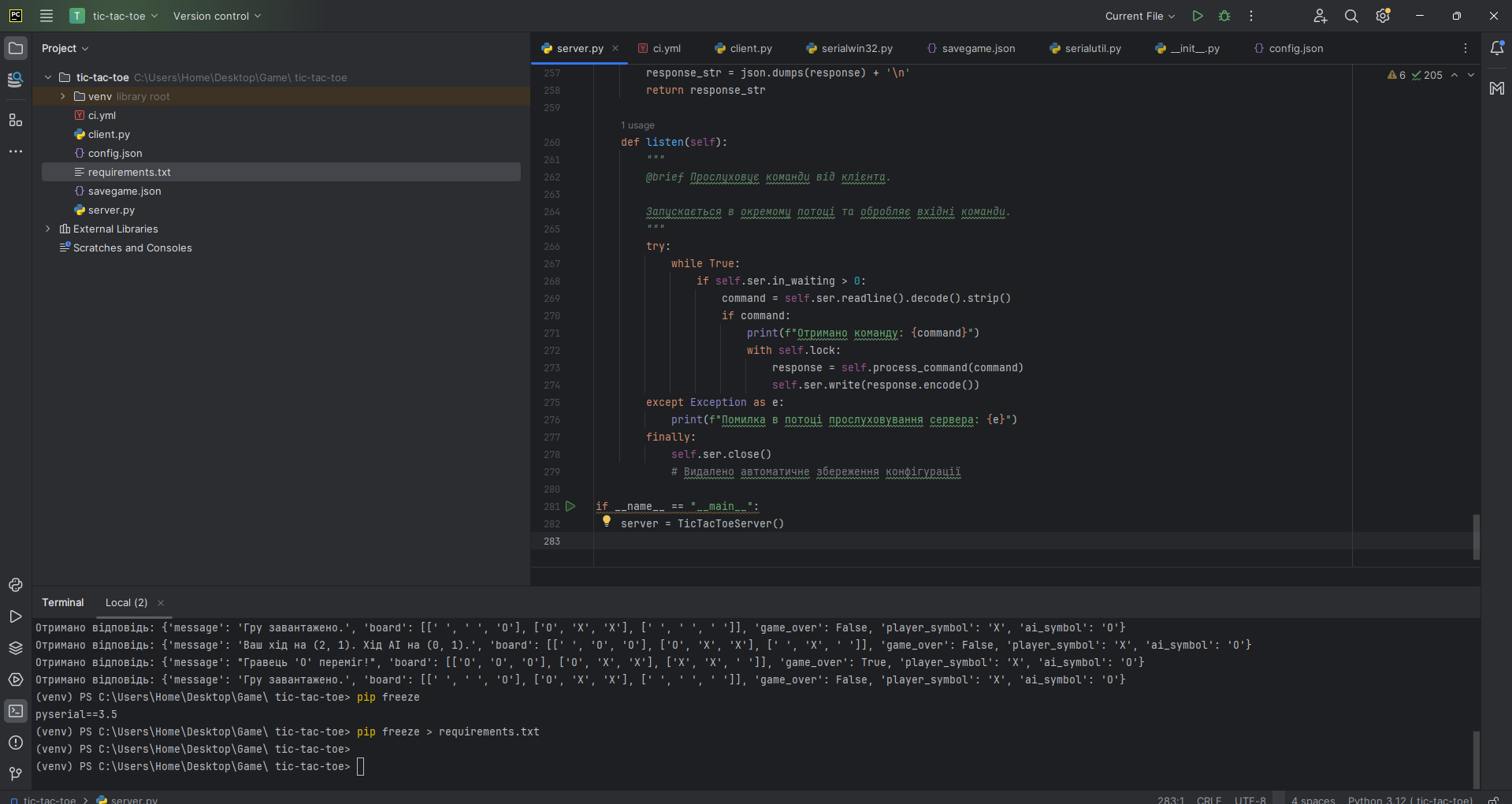
2. Generate documentation based on doxygen comments

3. Required steps

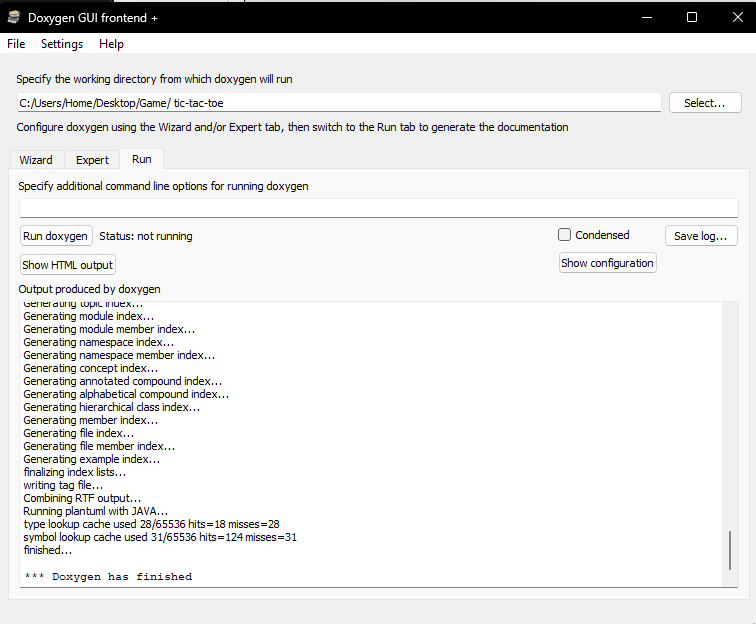
**Виконання роботи**

Було виконано наступні кроки для створення документації:

1. **Додавання Doxygen-коментарів**: Вихідний код серверної та клієнтської частини гри було доповнено коментарями у форматі Doxygen. Для кожної публічної функції, методу та параметру було додано коментарі, що описують їх призначення, параметри та значення, які вони повертають.



**Рис. 1.** Приклад коментарів у коді server.py з використанням тегів @brief, @param, @return.



**Рис. 2.** Налаштування Doxygen

**Генерація документації**: Після налаштування файлу конфігурації було запущено команду doxygen Doxyfile, яка згенерувала HTML-документацію у папці docs. У результаті, було створено структуровану документацію з описами всіх функцій та класів, що містяться у проекті.



**Рис. 3.** Згенерована HTML-документація

**Висновки**

Виконання роботи дозволило створити повноцінну технічну документацію для гри. Додавання коментарів у форматі Doxygen значно спростило процес генерації документації та забезпечило детальний опис функціональності проекту. Згенерована HTML-документація може бути використана для ознайомлення з проектом іншими розробниками та для його подальшого розвитку.

**Додатки**

**server.py**

# Server.py  
import serial  
import json  
import threading  
  
class TicTacToeServer:  
 *"""  
 @class TicTacToeServer  
 @brief Серверна частина гри "Хрестики-Нулики".  
  
 Клас відповідає за логіку гри, обробку команд від клієнта та відправку відповідей.  
 """* def \_\_init\_\_(self, port='COM11', baudrate=9600):  
 *"""  
 @brief Ініціалізує серверну програму.  
  
 @param port Порт для з'єднання з клієнтом.  
 @param baudrate Швидкість передачі даних.  
  
 Встановлює з'єднання з клієнтом та запускає потік для прослуховування команд.  
 """* 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):  
 *"""  
 @brief Ініціалізує порожню дошку гри.  
  
 @return Порожня дошка 3x3.  
 """* return [[" " for \_ in range(3)] for \_ in range(3)]  
  
 def save\_game(self, filename='savegame.json'):  
 *"""  
 @brief Зберігає поточний стан гри у файл.  
  
 @param filename Ім'я файлу для збереження.  
  
 Зберігає дошку, символи гравців та стан гри у JSON-файл.  
 """* save\_data = {  
 "board": self.board,  
 "player\_symbol": self.player\_symbol,  
 "ai\_symbol": self.ai\_symbol,  
 "game\_over": self.game\_over  
 }  
 with open(filename, 'w') as savefile:  
 json.dump(save\_data, savefile)  
 print(f"Гра збережена: {save\_data}")  
  
 def load\_game(self, filename='savegame.json'):  
 *"""  
 @brief Завантажує стан гри з файлу.  
  
 @param filename Ім'я файлу для завантаження.  
  
 Відновлює дошку, символи гравців та стан гри з JSON-файлу.  
 """* try:  
 with open(filename, 'r') as savefile:  
 data = json.load(savefile)  
 print(f"Гра завантажена: {data}")  
 self.board = data["board"]  
 self.player\_symbol = data["player\_symbol"]  
 self.ai\_symbol = data["ai\_symbol"]  
 self.game\_over = data["game\_over"]  
 except (FileNotFoundError, KeyError) 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):  
 *"""  
 @brief Перевіряє, чи є переможець.  
  
 @param board Поточна дошка.  
 @param symbol Символ гравця ('X' або 'O').  
  
 @return True, якщо гравець з заданим символом переміг, інакше False.  
 """* 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):  
 *"""  
 @brief Перевіряє, чи є нічия.  
  
 @param board Поточна дошка.  
  
 @return True, якщо всі клітинки заповнені і немає переможця, інакше False.  
 """* return all(cell != " " for row in board for cell in row)  
  
 def minimax(self, board, depth, is\_maximizing):  
 *"""  
 @brief Алгоритм Minimax для вибору найкращого ходу AI.  
  
 @param board Поточна дошка.  
 @param depth Глибина рекурсії.  
 @param is\_maximizing Логічна змінна, що визначає, чи максимізуємо ми оцінку.  
  
 @return Оцінка найкращого ходу.  
 """* 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):  
 *"""  
 @brief Здійснює хід AI.  
  
 Використовує алгоритм Minimax для вибору найкращого ходу та оновлює дошку.  
  
 @return Кортеж з координатами ходу AI.  
 """* 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):  
 *"""  
 @brief Обробляє отриману команду від клієнта.  
  
 @param command Строка з командою.  
  
 @return Строка з відповіддю у форматі JSON.  
 """* response = {}  
 if command.startswith("start"):  
 parts = command.split()  
 if len(parts) == 2 and parts[1] in ['X', 'O']:  
 self.player\_symbol = parts[1]  
 self.ai\_symbol = 'O' if self.player\_symbol == 'X' else 'X'  
 self.board = self.init\_board()  
 self.game\_over = False  
 response['message'] = f"Гра розпочата. Ви граєте за '{self.player\_symbol}'."  
 else:  
 response['message'] = "Некоректна команда старту. Використовуйте 'start X' або 'start O'."  
 elif command == "new":  
 if self.player\_symbol is None:  
 response['message'] = "Спочатку виберіть символ гравця за допомогою команди 'start X' або 'start O'."  
 else:  
 self.board = self.init\_board()  
 self.game\_over = False  
 response['message'] = "Нова гра розпочата."  
 elif command == "save":  
 self.save\_game()  
 response['message'] = "Гру збережено."  
 elif command == "load":  
 self.load\_game()  
 response['message'] = "Гру завантажено."  
 elif command.startswith("move"):  
 if self.game\_over:  
 response['message'] = "Гра вже закінчена. Розпочніть нову гру."  
 elif self.player\_symbol is None:  
 response['message'] = "Спочатку виберіть символ гравця за допомогою команди 'start X' або 'start O'."  
 else:  
 parts = command.split()  
 if len(parts) == 3:  
 \_, row, col = parts  
 try:  
 row, col = int(row), int(col)  
 if 0 <= row <= 2 and 0 <= col <= 2:  
 if self.board[row][col] == " ":  
 self.board[row][col] = self.player\_symbol  
 if self.check\_winner(self.board, self.player\_symbol):  
 response['message'] = f"Гравець '{self.player\_symbol}' переміг!"  
 self.game\_over = True  
 elif self.check\_tie(self.board):  
 response['message'] = "Нічия!"  
 self.game\_over = True  
 else:  
 ai\_move = self.make\_ai\_move()  
 if self.check\_winner(self.board, self.ai\_symbol):  
 response['message'] = f"Гравець '{self.ai\_symbol}' переміг!"  
 self.game\_over = True  
 elif self.check\_tie(self.board):  
 response['message'] = "Нічия!"  
 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):  
 *"""  
 @brief Прослуховує команди від клієнта.  
  
 Запускається в окремому потоці та обробляє вхідні команди.  
 """* 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

# Client.py  
import tkinter as tk  
from tkinter import messagebox, simpledialog  
import serial  
import json  
import threading  
  
class TicTacToeClient:  
 *"""  
 @class TicTacToeClient  
 @brief Клієнтська частина гри "Хрестики-Нулики".  
  
 Клас відповідає за взаємодію з користувачем через графічний інтерфейс,  
 відправку команд на сервер та обробку відповідей від сервера.  
 """* def \_\_init\_\_(self, port='COM12', baudrate=9600):  
 *"""  
 @brief Ініціалізує клієнтську програму.  
  
 @param port Порт для з'єднання з сервером.  
 @param baudrate Швидкість передачі даних.  
  
 Створює вікно гри, встановлює з'єднання з сервером та запускає потік для прослуховування відповідей.  
 """* 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):  
 *"""  
 @brief Запитує у гравця вибір символу ('X' або 'O').  
  
 Викликає діалогове вікно для введення символу та відправляє команду старту на сервер.  
 """* 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):  
 *"""  
 @brief Створює графічні елементи інтерфейсу.  
  
 Створює кнопки для поля гри та кнопки керування грою.  
 """* # 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):  
 *"""  
 @brief Розпочинає нову гру.  
  
 Відправляє команду "new" на сервер та оновлює інтерфейс.  
 """* 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):  
 *"""  
 @brief Зберігає поточний стан гри.  
  
 Відправляє команду "save" на сервер.  
 """* self.ser.write("save\n".encode())  
 messagebox.showinfo("Збереження", "Гру збережено.")  
  
 def load\_game(self):  
 *"""  
 @brief Завантажує збережену гру.  
  
 Відправляє команду "load" на сервер.  
 """* self.ser.write("load\n".encode())  
 self.game\_over = False  
  
 def make\_move(self, row, col):  
 *"""  
 @brief Здійснює хід гравця.  
  
 @param row Рядок на дошці.  
 @param col Стовпець на дошці.  
  
 Перевіряє коректність ходу та відправляє команду "move" на сервер.  
 """* 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):  
 *"""  
 @brief Прослуховує відповіді від сервера.  
  
 Запускається в окремому потоці та обробляє вхідні повідомлення.  
 """* 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):  
 *"""  
 @brief Обробляє відповіді від сервера.  
  
 @param response Словник з даними від сервера.  
  
 Оновлює стан гри та інтерфейс на основі отриманої відповіді.  
 """* 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):  
 *"""  
 @brief Оновлює відображення дошки на інтерфейсі.  
  
 Проставляє символи на кнопках відповідно до поточного стану дошки.  
 """* 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):  
 *"""  
 @brief Закриває клієнтський додаток.  
  
 Зупиняє потік прослуховування та закриває з'єднання з сервером.  
 """* self.running = False  
 if self.ser.is\_open:  
 self.ser.close()  
 self.listen\_thread.join()  
 self.root.quit()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 client = TicTacToeClient()