

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"



**АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ КОМП'ЮТЕРНИХ
СИСТЕМ**

Лабораторна робота №4

Виконав: ст. гр. КІ-405
Легкобит Н.В

Прийняв:
Шпіцер А.С

Львів 2024

Тема: Створення Doxygen документації

Порядок виконання лабораторної роботи:

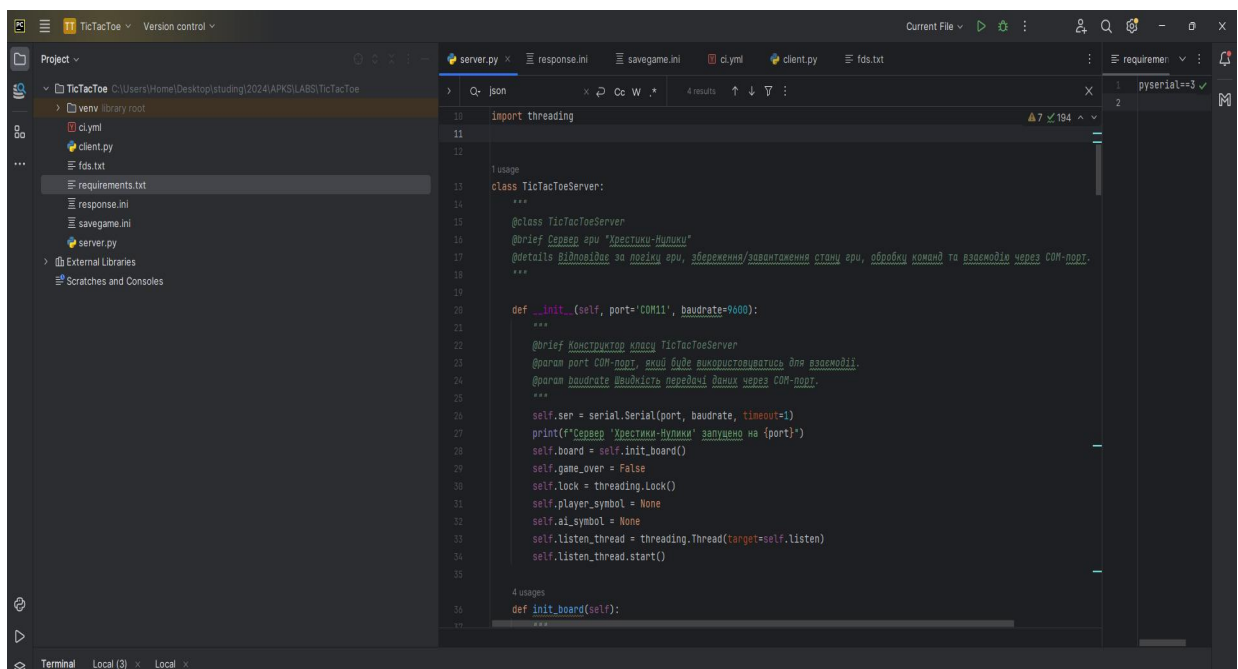
Task 4. Create doxygen documentation:

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.



```
server.py
10 import threading
11
12 1 usage
13 class TicTacToeServer:
14     """
15     @brief Сервер гри "Хрестини-Нольки"
16     @details Відповідає за логіку гри, збереження/завантаження стану гри, обробку команд та взаємодію через COM-порт.
17     """
18
19
20 def __init__(self, port="COM11", baudrate=9600):
21     """
22     @brief Конструктор класу TicTacToeServer
23     @param port COM-порт, який буде використовуватись для взаємодії.
24     @param baudrate Швидкість передачі даних через COM-порт.
25     """
26     self.ser = serial.Serial(port, baudrate, timeout=1)
27     print(f"Сервер 'Хрестини-Нольки' запущено на {port}")
28     self.board = self.init_board()
29     self.game_over = False
30     self.lock = threading.Lock()
31     self.player_symbol = None
32     self.ai_symbol = None
33     self.listen_thread = threading.Thread(target=self.listen)
34     self.listen_thread.start()
35
36 4 usages
37 def init_board(self):
38     """
39     """
```

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

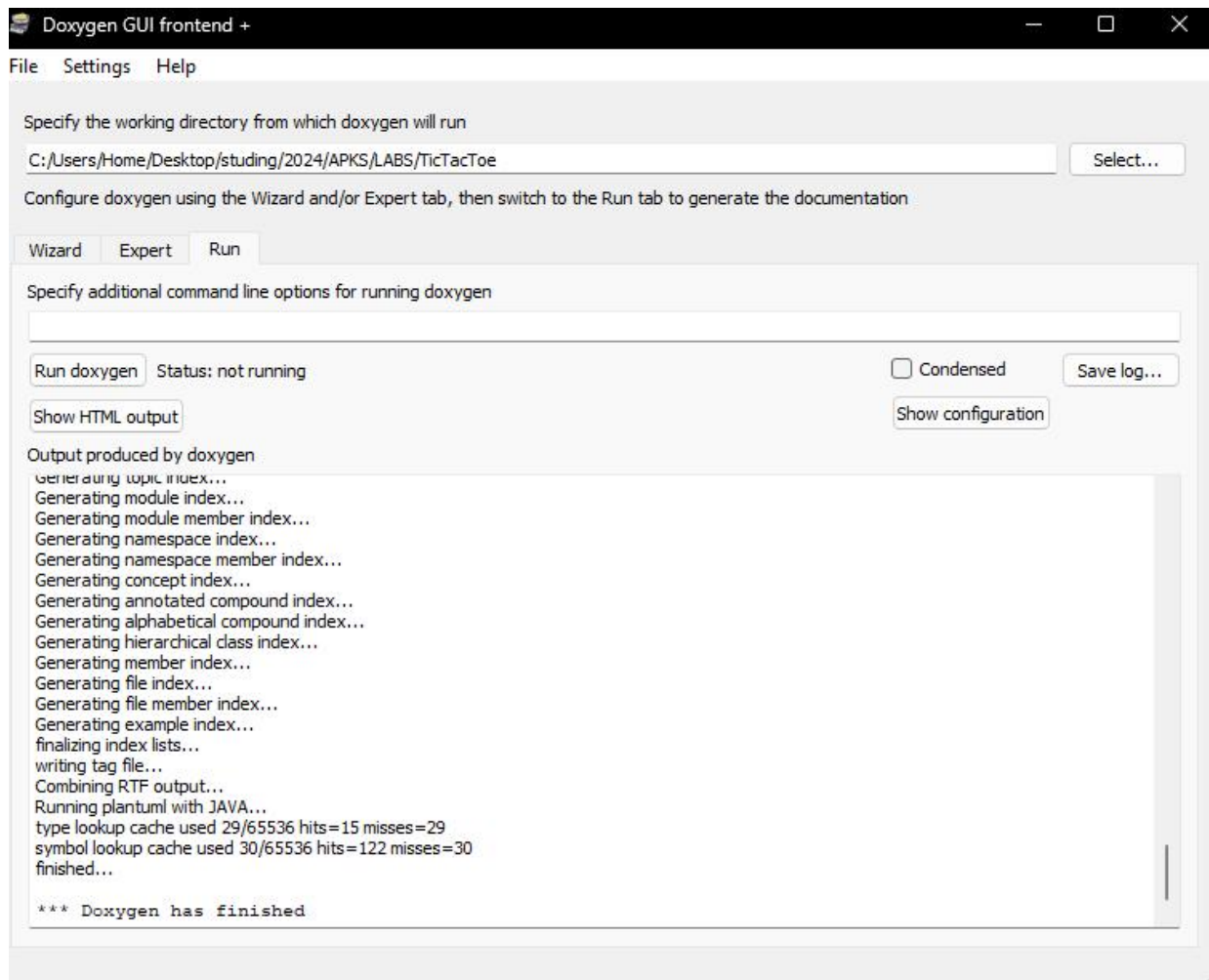


Рис. 2. Налаштування Doxygen

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.

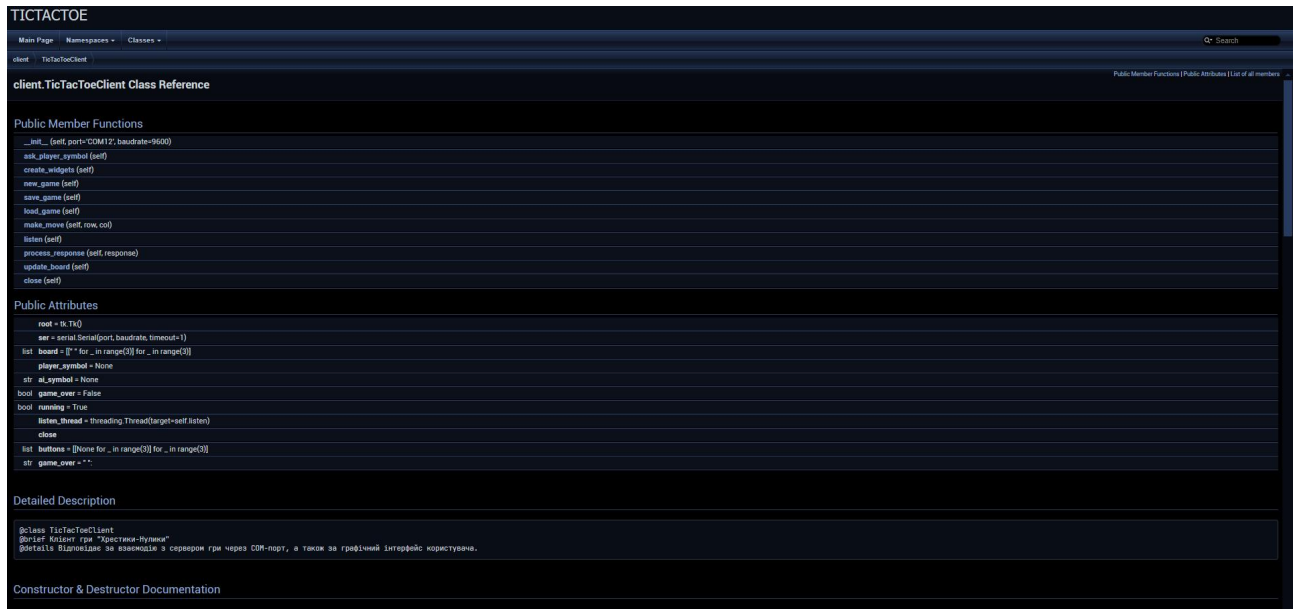


Рис. 3. Згенерована HTML-документація

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

```
"""
@file server.py
@brief Сервер гри "Хрестика-Нулика" на основі COM-порту
@details Цей модуль реалізує серверну логіку гри "Хрестика-Нулика", використовуючи
бібліотеки `serial`, `json`, та `threading`.
"""

import configparser
import serial
import json
```

```

import threading

class TicTacToeServer:
    """
    @class TicTacToeServer
    @brief Сервер гри "Хрестики-Нулики"
    @details Відповідає за логіку гри, збереження/завантаження стану гри, обробку команд та взаємодію через COM-порт.
    """

    def __init__(self, port='COM11', baudrate=9600):
        """
        @brief Конструктор класу TicTacToeServer
        @param port COM-порт, який буде використовуватись для взаємодії.
        @param baudrate Швидкість передачі даних через COM-порт.
        """

        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
        self.listen_thread = threading.Thread(target=self.listen)
        self.listen_thread.start()

    def init_board(self):
        """
        @brief Ініціалізує порожню ігрову дошку 3x3.
        @return Список списків, що представляє ігрову дошку.
        """

        return [[" " for _ in range(3)] for _ in range(3)]

    def save_game(self, filename='savegame.ini'):
        """
        @brief Зберігає стан гри у файл.
        @param filename Ім'я файлу для збереження.
        """

        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'):
    """
    @brief Завантажує стан гри з файлу.
    @param filename Ім'я файлу для завантаження.
    """
    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):
    """
    @brief Перевіряє, чи є перемога на дошці для заданого символу.
    @param board Ігрова дошка.
    @param symbol Символ для перевірки ('X' або 'O').
    @return True, якщо символ виграв; інакше False.
    """
    lines = (
        board[0], board[1], board[2],
        [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]],
        [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 на дошці.
    @param board Ігрова дошка.
    @return True, якщо нічия; інакше False.
    """
    return all(cell != " " for row in board for cell in row)

```

```

def minimax(self, board, depth, is_maximizing):
    """
    @brief Алгоритм мінімакс для вибору найкращого ходу AI.
    @param board Ігрова дошка.
    @param depth Глибина рекурсії.
    @param is_maximizing Прапорець, що вказує, чи AI максимізує оцінку.
    @return Оцінка 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 Робить хід AI, використовуючи алгоритм мінімакс.
    @return Координати зробленого ходу або None, якщо хід неможливий.
    """

    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

```



```

        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 Основний цикл прослуховування комунікації через COM-порт.
    """
    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

```

"""
@file client.py
@brief Клієнт гри "Хрестики-Нулики" на основі COM-порту
@details Цей модуль реалізує клієнтську частину гри "Хрестики-Нулики" з графічним
інтерфейсом на основі Tkinter.
"""

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

class TicTacToeClient:
    """
    @class TicTacToeClient
    @brief Клієнт гри "Хрестики-Нулики"
    @details Відповідає за взаємодію з сервером гри через COM-порт, а також за графічний
    інтерфейс користувача.
    """

    def __init__(self, port='COM12', baudrate=9600):
        """
        @brief Конструктор класу TicTacToeClient
        @param port COM-порт, який буде використовуватись для взаємодії.
        @param baudrate Швидкість передачі даних через COM-порт.
        """

        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 символ гравця у користувача.
    @details Відображає діалог для вибору символу ('X' або 'O') і передає вибір 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 графічний інтерфейс користувача.
    @details Додає кнопки для ігрової дошки, а також елементи керування для нової гри,
збереження та завантаження.
    """
    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

    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 Починає нову гру.
        @details Очищає ігрову дошку та повідомляє сервер про початок нової гри.
        """

        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.
        @details Надсилає серверу команду збереження гри.
        """

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

    def load_game(self):
        """
        @brief Завантажує стан гри.
        @details Надсилає серверу команду завантаження гри.
        """

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

    def make_move(self, row, col):
        """
        @brief Робить хід гравця.
        @param row Рядок, у якому гравець хоче зробити хід.
        @param col Стовпець, у якому гравець хоче зробити хід.
        """

        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.
        @details У окремому потоці отримує відповіді від сервера та обробляє їх.
        """

        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 відповідь від сервера.
    @param response JSON-об'єкт з відповіддю від сервера.
    """
    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)

    self.player_symbol = player_symbol
    self.ai_symbol = ai_symbol

    if board:
        self.board = board
        self.update_board()
    if message:
        if "перемір" in message or "Нічия" in message:
            messagebox.showinfo("Результат гри", message)
            self.game_over = True
        else:
            messagebox.showinfo("Інформація", message)
            self.game_over = game_over
    else:
        self.game_over = game_over

def update_board(self):
    """
    @brief Оновлює графічне представлення ігрової 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 клієнтську програму.

```

```
@details Закриває з'єднання з сервером і завершує потік.  
"""  
  
self.running = False  
if self.ser.is_open:  
    self.ser.close()  
self.listen_thread.join()  
self.root.quit()  
  
if __name__ == "__main__":  
    client = TicTacToeClient()
```