1 Introduction

In this assignment, you will implement an online Tic-Tac-Toe game. The purpose of this assignment is to improve your programming skills and provide a deep understanding of the transport and application layers (layers 4-5). The assignment utilizes the *client-server* architecture as seen in class. Note that the client-server architecture involves two entities: the **client**, who consumes services, and the **server**, who provides services. In our case:

- The **client** is a user who's interested in playing the Tic-Tac-Toe game.
- The **server** is the computer that's responsible for managing the games. Its services include: access to a game, displaying moves, etc.

Remark. The Tic-Tac-Toe game is usually played on a grid that's 3 squares by 3 squares. You are X, and your opponent is O. Players take turns putting their marks in empty squares. The first player to get three of their marks in a row (up, down, across, or diagonally) is the winner. Typically, Tic-Tac-Toe is a game for two players, but adding a third player changes it as follows:

- Instead of 9 squares, it's 16 now. But you only need to get 3 in a row, not 4, to win.
- While the first two players use X's and O's, the third player will use Δ's.
- Play as usual, and the first person to get 3 in a row wins.

Now we can easily expand the game into an x-players game with board size of $(x+1)^2$.

2 Requirements

2.1 Server Side

- Implement a multi-client Tic-Tac-Toe game server using sockets.
- Clients must connect to the server to play the game.
- The server should be capable of handling multiple concurrent games between different groups of clients.
- Ensure proper validation of moves, and implement game logic to determine the winner or declare a draw.

2.2 Client Side

- Clients should connect to the server to start playing.
- Implement a user interface for the Tic-Tac-Toe game.
- Clients can request a list of available games and choose to join an existing game or create a new one.
- Implement a real-time update mechanism to display the current state of the game.
- Clients should be able to gracefully exit the game and disconnect from the server.

3 Technical Highlights

- Consider extreme cases. Use your own judgment to decide where validation is necessary. Input type validation is not required – for example, if you expect a string, you can assume it.
- You are responsible for establishing a reliable connection between the server and clients. Use the functions in the *socket* library wisely.
- Focus your efforts on the following:
  - Provide a well-documented quality code.
  - Provide a documentation in the PDF file.
  - Fully understand the theoretical material.

4 Instructions

4.1 Implementation

- Code your assignment in *Python*, as seen at class. Following is a link where you can learn python easily: https://www.w3schools.com/python
- You **must not** use any non-standard libraries for socket programming. Use only the libraries introduced at class.

4.2 Documentation

- You're required to document your code thoroughly.
- You're required to attach a PDF document including the following:
  1. Full name and ID.
  2. Operating system and Python version.
  3. An explanation about any file you created – including any classes and functions.
- Include illustration pictures (usage examples).

4.3 Submission

- You may submit in groups of 2-3 students.
- Compress all the files in a ZIP format and submit it with your ID.
- **Final submission date: 03/02/2025**