Lab 2: Networked Peer-to-Peer Game

EGP405: Networking for Online Games (Fall 2017)

Assigned: **Thursday, Sept. 21**

Due date: **Thursday, Sept. 28**

Grade weight: **5%, out of 10 points**

Introduction:

In recent lessons, we have explored general networked game architecture and the traditional game loop modified to include a networking update. Here we further explore the concepts discussed by building a simple asynchronous, turn-based game using a peer-to-peer networked architecture.

Goals & Result:

This lab has two core requirements:

1. Implement a console application version of *Tic-Tac-Toe* or *Connect4*.
2. Extend the game by adding networked multiplayer functionality.

The result of this lab is a peer-to-peer networked version of *Tic-Tac-Toe* or *Connect4*.

Instructions:

Follow these general steps to complete this lab:

1. ***Start screen***: Implement a start screen that asks if the game will be local or networked. Depending on the user's selection, one of three things should occur:
    a. Begin a local game.
    b. Host a networked game.
    c. Connect to a hosted networked game.
2. ***Implement game***: Implement a *local multiplayer* version of *Tic-Tac-Toe* or *Connect4* in a C++ console application. A single player's turn consists of three core steps:
    a. *Input*: Process the player's keyboard input to either change the selected square (e.g. using WASD), commit a move (e.g. using enter), or resign (e.g. using escape).
    b. *Update*: Change the universal game state based on the player's input. This could include rejecting illegal moves, thereby allowing a player to choose again. The player inputs and effects on the game state are:
        i. *Tic-Tac-Toe*: Select one open space in a 3x3 grid. Change the game state by marking the player's shape (X or O) at the selected space. The player may not select an occupied space.
        ii. *Connect4*: Select one of seven columns in a 7x6 grid. Change the game state by marking the player's shape (X or O) at the lowest open slot in the selected column. The player may not select a full column.

   c. *Draw*: Display the game state and pertinent messages in the console using simple ASCII art (e.g. pipes for vertical lines).
3. ***Implement endgame***: Implement the win condition of your game. When the game ends, ask if another round should begin or if the game should return to the welcome state. Reverse player roles (X and O) each round. The conditions are:
   a. *Tic-Tac-Toe*: If at any time in the game a player has three of their shape lined up along any row, column or diagonal, they win the game. If the game board is full and nobody has won, a draw is declared.
   b. *Connect4*: If at any time in the game a player has four of their shape lined up along any row, column or diagonal, they win the game. If the game board is full and nobody has won, a draw is declared.
4. ***Implement networked game***: Implement a *networked multiplayer* version of the game implemented above by introducing a *peer-to-peer* (**not server-client**) networking update to the loop. This update occurs immediately before the game state update. Consider the following when implementing this step:
   a. *Setup*: The first peer, playing as X, accepts a maximum of one connection and waits for a second instance to join after starting, playing as O.
   b. *Sending game data*: The only data sent over the network after a connection is established is *the player's move*. When a player *commits a legal move* (including resignation) the packet data is simply a byte to denote the action taken. ***Do not ship the entire game board***.
   c. *Receiving game data*: If there is no packet received in a network update, the game should display a "wait" message. If there is a packet received, the data is processed, and the other player's move should affect the local game state update.
5. ***Implement networked endgame***: When a networked game ends, players have the option to keep playing or leave. If both players stay, the game loop resets with player roles reversed. If only one player opts to stay, their game should go back to '*setup*' phase and that player becomes the "first peer" awaiting a connection.

Additional Requirements:
Complete the lab in ***pairs*** and use version control often. Either set up a new project in a new repository (recommended) or use an existing repository that is unique to this team. See the briefing for lab 1 on instructions on setting up a new project in Visual Studio.

Submission Guidelines:
1) Include the following header information at the top of all source files ***modified***:
   a. "This file was modified by <*names*> with permission from author."
2) Include the following header information at the top of all source files ***created***:
   a. Team member names and student IDs.

   b. Course code, section, project name and date.
   c. Certificate of Authenticity (standard practice): "*We certify that this work is entirely our own. The assessor of this project may reproduce this project and provide copies to other academic staff, and/or communicate a copy of this project to a plagiarism-checking service, which may retain a copy of the project on its database*."
3) All sections of code modified or written by your team should be commented, explicitly stating ***who*** made the change or wrote code and ***why***, and what the code ***does*** in the context of the program.
4) ***When you are finished***, delete all garbage files. ***Do not*** delete Visual Studio project or solution files, or your source files. ***Do not*** submit developer SDKs with your project (I gave them to you; I already have them). **Delete the following files**:
   a. ***All*** build and intermediate files (Debug, Release).
   b. Intellisense database files (including but not limited to *.db, *.sdf).
5) Zip up your project's root and rename using this convention: "**EGP-405-F2017-Net-Lab2-<insert submitter's name here>**".
6) In the Canvas submission, add:
   a. The above zip file.
      Note: If the file size is larger than 1MB, you have not cleaned it correctly!
   b. A link to your public repository.
      Note: Grades will not be discussed or released through public repositories, for any reason, in compliance with FERPA.

Grading:

- **10 points**: complete and test all steps and sub-steps in the instructions.

Specific penalties:

- **–10 points**: Submission does not build and run *correctly* and *immediately* out of the box (test using both debug and release modes).
- **–10 points**: Code not documented and commented, including contributions, explanations, certificate of authenticity, etc.
- **–10 points**: Lacking or weak evidence of having used version control to maintain project. Please ask for help if needed.
- **–10 points**: Submission contains junk files. Follow instructions above to ensure you have removed the correct files, and ask for help if needed.

Good luck, learn lots and have fun! ☺