# Connect4 **Functional Specifications**

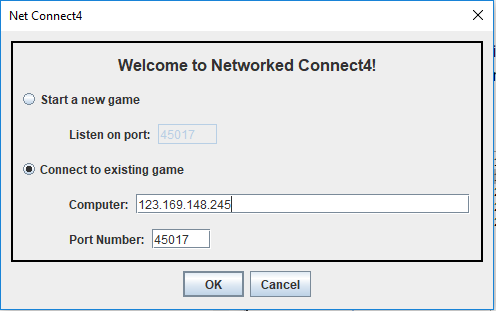
**Main  
Connect4GameHub  
Connect4GameState  
Connect4Window**

**Main Class**

Requirements

The Main class keeps track of all the main components that make up the Connect4 game and connects the netgame.connect4 package with the netgame.common package. The board is not created here, but the Main class initializes the loading dialogue boxes and facilitates the networking via TCP. The following dialogue boxes are the main portal for the user to control the application.





The Functional Specifications

To create and manage the Main class, the following functionality must be provided.

1. Store the connectivity information to transfer to the main Hub which controls the TCP procedures.
2. Provide a functional GUI for the user to interact with and input their IP address and other important information.  
     
   The Hub class and netgame.common are sourced from David Eck, a verified and reliable coder who was willing to allow us to borrow and credit his code.  
     
   The GUI dialogue box is simple, consisting of two text boxes, two radio buttons, a JFrame and a couple labels.
3. The user’s input is used to connect two separate computers/servers by using their IP addresses and listening on a specific port number. In this case, the default port number is 45017.

## Connect4GameHub Class

# Requirements

This class constructs the Connect4GameHub, a space that connects, translates, and passes code between both separate servers. You can think of it as an invisible middleman between both client-side computers.

# Functional Specifications

The use of this class allows the development team the flexibility to send any number of messages back-and-forth between both servers.

1. Connect4GameHub(int port) constructor, which, in addition to initializing a GameHub, also begins the channel for messages to flow through.
2. messageReceived(int playerID, Object message) tells the Hub what to do when a message is received. The Hub needs this method to help it interpret the message and figure out how to deal with it before it gets pushed onto GameState.
3. playerConnected(int playerID) cuts the amount of players allowed to communicate to 2 people (see the getPlayerList().length) and then communicates a command to start a game to the GameState.
4. playerDisconnected(int playerID) when one player disconnects (exits the application) for any certain reason, the game is ended and exited. This means that the state of the disconnected client must be updated and the message stream has to be updated in conjunction, which are both accomplished under this method.

## Connect4GameState Class

## Requirements

## This class provides the overall logic of the game, detailing game mechanics and creating objects and rules necessary for the game to properly function

## Functional Requirements

1. The applyMessage(int sender, Object message) method allows the GameState to be updated in conjunction with a message sent by the user.   
     
   This is considered the backend, but on the frontend side, the user (sender) places a move (message) by clicking a specific column in order to drop a token in.
2. The startGame() and startFirstGame() methods both initialize the game.  
     
   Functionally, the methods both create a backend 2D array of characters that models the GUI version of the board with which the user directly interacts.   
     
   The method also randomly assigns either RED or BLUE to both clients. RED always starts.
3. The winSequence(char player) continuously checks the board for any form of a win.   
     
   In the case of the game Connect4, a win constitutes of any 4 consecutive tokens of one color in a horizontal, vertical, or diagonal row.
4. The final method, the tie() method, checks for a tie, in essence checking if the board is completely filled and no win condition has been recognized by the winSequence() algorithm.

**Connect4Window Class**

Requirements

The Connect4Window class manages the graphical user interface, or GUI, with which the users directly interact during the course of normal gameplay.

Functional Specifications

1. Generate a Connect4Client, which extends netgame.common’s Client class and provides easier, direct communication with the Hub.
2. Draw the Connect4 board using a simple, efficient JPanel and Java Swing.
3. Make the window visible and connected to both client-side servers, enabling almost-instant transfer of information and subsequent visual representation of moves on the screen.
4. Map the mouse click location and keybind the click location to a specific turn within the game.
5. Allow the window to refresh itself, not only during gameplay, but also for new games started within the same session (aka games started after one game has been finished).

