University of Utah
School of Computing

CS 3500/5010 **Handout 6** **October 9, 2010**

Problem Set Five (Initial Specifications)

In this problem set, you will be creating a Visual Studio solution that solves the problem described below. Your solution must be called `ps5`, and must consist of at least projects:

- A console project called `BoggleServer`
- A forms project called `BoggleClient`
- One or more testing projects

Use C# as your implementation language.

You must work on this project with a partner. Please find one right away.

Either you or your partner must attend a lab section on Wednesday, October 20. You or your partner will need to

- Tell the TA your name and the name of your partner
- Tell the TA the login name of the partner in whose repository the solution will be committed.
- Show the TA that you have made substantial progress toward a solution. If you have not made substantial progress, it will hurt your grade.

We will retrieve your solution for grading by running the Linux command

```
svn --username cs3500 --password ######### checkout
       svn://lenny.eng.utah.edu/home/XXXXXXXX/cs3500/ps5/trunk ps5
```

(put it all on one line) where `#########` is the grading password `XXXXXXXX` is the CADE login name of the partner who in whose repository the solution resides . We will run the command, which will give us the most recently committed version of your solution, sometime on the morning of Wednesday, October 27. This is the version that we will grade.

You find find `dictionary.txt` linked into the class web page. This is a file full of English words.

More To Come

I will be following this up with a second handout that discusses implementation tips and testing requirements.

Problem

Boggle is a word game. You can learn the rules at `http://en.wikipedia.org/wiki/Boggle` and you can play the game online at `http://www.fun-with-words.com/boggle.html`.

In this problem set, you and your partner will be building a server to which clients can connect from (potentially) remote computers to play Boggle. In addition, you also will be building a client. Finally, you will be testing your client and server.

Boggle Boards

A physical Boggle board displays the faces of 16 cubes that are randomly arranged into a 4×4 grid. (The position of each cube is chosen at random, as is the exposed face of each cube.) These are the six faces of each of the 16 cubes:

```
LRYTTE  ANAEEG  AFPKFS  YLDEVR
VTHRWE  IDSYTT  XLDERI  ZNRNHL
EGHWNE  OATTOW  HCPOAS  OBBAOJ
SEOTIS  MTOICU  ENSIEU  NMIQHU
```

In a real Boggle game, the “Q” face actually shows up as “QU”. To keep things a bit simpler for you, we’ll ignore that detail and stick with just “Q”.

Over the course of a Boggle game, the two players create a list of words that they believe appear on the board. Given a player’s list of words, his or her score is computed as follows:

- All words with fewer than three characters are removed.
- All duplicate words are removed.
- All *legal* words that appear on the opponent’s list are removed.
- Each remaining illegal word is worth negative one points.
- The score of each remaining legal word depends on its length. Three- and four-letter words are worth one point, five-letter words are worth two points, six-letter words are worth three points, seven-letter words are worth five points, and longer words are worth 11 points.

Server

Your server should be a console project whose main method takes two required and one optional command line parameter:

- The number of seconds that each Boggle game should last. This should be a positive integer.

- The pathname of a file that contains all the legal words. The file should contain one word per line.
- An optional string consisting of exactly 16 letters. This will be used to initialize Boggle boards for testing purposes.

Your server should accept incoming connections from clients on port 2000 and pair them in Boggle matches. Your server will be responsible for coordinating all aspects of play: the pairings, the boards, the time limits, and the scoring. It should be able to deal with at least 100 simultaneous games.

Client

Your client should be a Windows forms project. It should behave as follows. This is only a high-level view. The details of the protocol are specified in a separate section.

1. On launch, the client should prompt the user for the address (i.e., "localhost" or an IP address) at which the server is running. It should then attempt to contact the server on port 2000. If it fails to connect, it should display an error message and reprompt.
2. The client should prompt the user for his or her name.
3. The client should ask the server to set up a game by sending the user's name. It should inform the user that it is waiting for the server to set up a game. When the game has been set up, the server will respond with the grid of letters, the name of the opponent, and the time limit.
4. The client should display the grid of letters, the names and current scores of both players, and a clock that proceeds to count down every second. It should also allow the user to begin entering words. (The client clock is unofficial. The server will determine when the game is over.)
5. The client should deal with the words that are entered (by sending them to the server) and the messages that come back from the server (by updating the score).
6. When the server notifies the client that time is up, it should tell the user that the game is over. The client should also display the word list information that is sent by the server.

Protocol

The protocol that governs the communications between the client and the server is as follows. All communications are case-insensitive.

1. First, the client connects to the server on port 2000.
2. Next, the client transmits a two-line command to the server. The two lines are:
   - The word "play".
   - The name of the player.
3. Once the server has received connections from two clients that are ready to play, it pairs them in a two-minute match. The server begins the match by transmitting a four-line command to each client. The four lines are:
   - The word "start".

- The 16 characters that appear on the Boggle board to be used in this game, beginning with row 1 and ending with row 4. (The board specified on the command line when the server was launched should be used. If one was not specified, the board should be chosen at random.)
- The opponent's name.
- The length of the game in seconds (as specified on the command line when the server was launched).

4. Over the next two minutes, the two clients and the server communicate asynchronously. At any time the client may play a word by sending a two-line command to the server. The two lines are:

   - The word "word".
   - The word being played by the client.

   Also at any time, the server may send a three-line scoring update to the client. The three lines are:

   - The word "score".
   - The client's current score.
   - The opponent's current score.

   The server sends a scoring update to each client whenever the score changes.

5. When two minutes have elapsed, the server ignores any further communication from the clients and shuts down the game. First, it transmits the final score to both clients as described above. Next, it transmits a multi-line game summary to both clients. Suppose that during the game the client played a legal words that weren't played by the opponent, the opponent played b legal words that weren't played by the client, both sides played c legal words in common, the client played d illegal words, and the opponent played e illegal words. The game summary consists of these lines:

   - The word "stop".
   - The integer a.
   - a lines containing the legal words played only by the client.
   - The integer b.
   - b lines containing the legal words played only by the opponent.
   - The integer c.
   - c lines containing the legal words played in common.
   - The integer d.
   - d lines containing the illegal words played by the client
   - The integer e
   - e lines containing the illegal words played by the opponent

6. After this final transmission, the server closes and deletes the two client sockets and the game is over.

Assumptions

Socket programming is particularly challenging because so much can go wrong. For example, clients may misbehave and send information that doesn't obey the protocol, and clients may crash in the middle of a transmission.

To make things simpler, you may make these assumptions:

- Both the client and the server will follow the communication protocol as specified. The server does not need to check for or deal with misbehaving clients, and the client does not need to check for or deal with a misbehaving server.
- Neither the client nor the server will drop a socket connection until the game is over.
- Once the first line of a multi-line command arrives, the rest of the command will arrive quickly. Neither the client nor the server needs to worry about the transmission of partial commands.

Resource Leaks

A server is designed to run continuously for a long time, so it is particularly important to implement it so as to avoid resource leaks. In addition to memory, a server can run out of such operating system resources as file descriptors, threads, and sockets. A leaky server will eventually use of all of the available resources and fail.

Be sure to close files, exit threads, and close sockets when you are through with them.