6.005 Elements of Software Construction | Fall 2011 Problem Set 7: Jotto Client GUI

The purpose of this problem set is to give you experience with GUI programming. In particular you will be using Java Swing. Finally, you will get some more practice with multithreading.

You have substantial design freedom on this problem set. However, in order for your solution to be graded, your solution must include the following objects in your GUI as specified in the problems. They are defined for you in the JottoGUI class (and instantiated with default constructors), but you can instantiate them however you would like. These objects must be added directly to the layout of the JottoGUI JFrame.

- a JButton named "newPuzzleButton"
- a JLabel named "puzzleNumber"
- a JTextField named "guess"
- a JTextField named "newPuzzleNumber"
- a JTable named "guessTable"

NOTE: You can rename the variables themselves, but their "name" property must be set appropriately using the setName() method.

Overview

In this problem set, you will implement a simple Jotto playing client that communicates with a server that we have provided for you. If you've never played Jotto, you can find a short description of it here.

In our particular version of the game, the client selects a puzzle ID at random which corresponds to a secret 5-letter word on the server. The user can then submit 5-letter dictionary word guesses to the server, which will respond with the number of letters in common between the two words and the number of letters in the correct position.

You will communicate with the server through scripts.mit.edu. All communication from the client to the server will be through a request of the following form:

http://6.005.scripts.mit.edu/jotto.py?puzzle=[puzzle #]&guess=[5-letter dictionary word]

where [puzzle #] will be replaced by an integer generated by the client, and [5-letter dictionary word] is a (you guessed it) 5-letter dictionary word.

The response from the server should be:

guess [in common] [correct position]

where [in common] is the number of letters that the secret word, which the server determines by the puzzle #, and the guess have in common. [correct position] is the number of letters in the guess that are in the correct position in the secret word. Thus, [in common] will always be greater than or equal to [correct position].

If the request sent to the server was invalid, the response will be one of the following errors (in **bold**). The unbold text is a description of the error.

error 0 Ill-formatted request.

error 1 Non-number puzzle ID.

error 2 Invalid guess. Length of guess != 5 or guess is not a dictionary word.

For example, if you were to submit

http://6.005.scripts.mit.edu/jotto.py?puzzle=16952&guess=crazy

The secret word is "cargo," so the server should respond

guess 3 1

But if you submitted

http://6.005.scripts.mit.edu/jotto.py?puzzle=16952&guess=elephant

The server would respond with

error 2

This problem set requires the use of multithreading. To help in debugging and testing multithreading behavior, the server will provide delayed responses for any guess containing an asterisk ("*"). Ensure that your GUI is still responsive during this delay.

For example, if you were to submit

http://6.005.scripts.mit.edu/jotto.py?puzzle=16952&guess=*bean

The server will provide a delayed response, waiting for 5 seconds before responding back with:

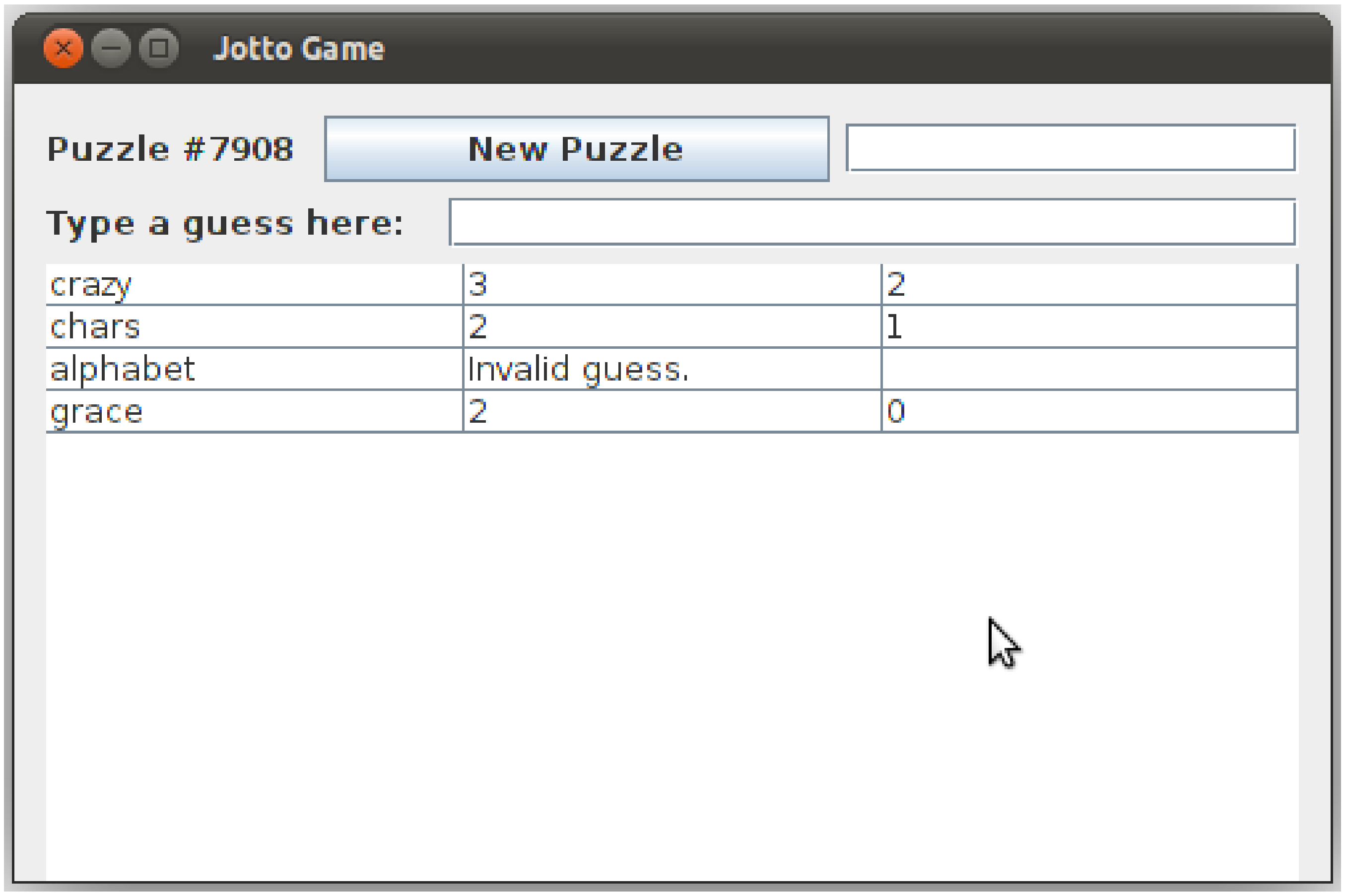
guess 1 0

The '*' will be considered a character in the server, so any guesses of incorrect length including the '*' returns an error:

http://6.005.scripts.mit.edu/jotto.py?puzzle=16952&guess=*bea

error 2

By the end of this problem set you will have built a GUI for your client to interact seamlessly with the server. The interface should eventually look something like this:



crazy	3	2
chars	2	1
alphabet	Invalid guess.	
grace	2	0

Your GUI does not have to look exactly like the window above. It should pass our tests as long as it functions, and the necessary components are there.

Problem 1: Communicate with the Server

[20 points] Write a Java method that uses java.net.URL (See the Java Tutorial for more details on how to use java.net.URL) to send a guess to the server, read back the reply, and return it. You must handle exceptions and errors from the server appropriately. We've provided a method signature for you in JottoModel called makeGuess, but you are free to change anything about it that you like. In fact, its return type is currently void, so you *must* change that. Make sure to include a spec and test your method thoroughly.

Problem 2: Set the Puzzle Number

a. [10 points] Create a GUI that includes a "New Puzzle" button (newPuzzleButton) next to a text field (newPuzzleNumber) and a label (puzzleNumber) to display the puzzle number. We recommend that you use the GroupLayout layout manager. It should basically look like the top row of the window above.

b. [10 points] Next, register an ActionListener to set the input from the text field as the new puzzle number, displaying it in the label next to the button. If no number is provided or the input is not a positive integer, pick a random positive integer. Make sure you can generate at least 10,000 different numbers. We've provided a JottoGUI class for you to use, but as stated earlier, you are free to change whatever about this that you like as long as you have the necessary components named correctly. Your program should start with a puzzle number already selected, be it random or always the same.

Problem 3: Make a Guess

[20 points] Add a JTextField (guess) to the GUI for the user to input a guess. Then use an ActionListener to send the guess to the server when the user presses ENTER, and print the result to standard output. Also clear the textbox after every guess, so that it's easy for the user to type the next guess. You should print "You win! The secret word was [secret word]!" when the user guesses the word correctly, ie. the server responds with "guess 5 5." Feel free to be creative with the message, but make sure it contains the phrase "you win."

Problem 4: Record Your Guesses

[20 points] Add a JTable (guessTable) to the GUI, to record both the guess and the server's response for that guess. The table should contain exactly 3 columns, one displaying the guesses the user mades, one displaying the number of characters "in common", and one displaying the number of characters in the "correct position". The table should be cleared each time the puzzle number is reset. If the server returns an error for a guess, the GUI should display a human-readable error message for that guess in the table instead of its score. The "You win!" message should also be displayed in the table.

Problem 5: Make it Multithreaded

[20 points] Move the server communication to a background thread. Make sure that you are still able to interact with the GUI even if the server is taking a long time to respond. The user should be able to submit multiple guesses even if the result from the original query has not yet appeared in the table. In particular, the guess should appear in the table as soon as the user submits it, and the results for all the guesses should eventually appear in the appropriate rows.

Remember, to test and debug the multithreaded nature of your Client, guesses that include a

'*' are delayed. You should make sure that your GUI remains responsive during this delay.