CompSci 201

Data Structures and Algorithms

Jotto — Full WriteUp

You can browse the code we provide and you can snarf it using our Ambient/Eclipse plugin. Check out the instructions on how to download the ambient system and how to snarf — check out a project

The snarf url for this CompiSci 201 is`http://www.cs.duke.edu/courses/compsci201/fall13/snarf/`.

The Code

After snarfing the code you will have:

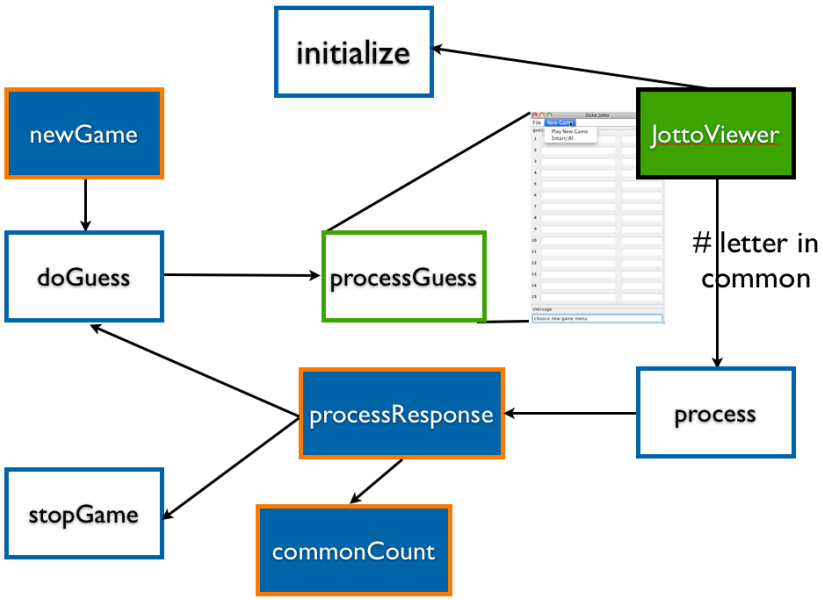
- Jotto: the main class that launches a GUI (Graphical User Interface), for playing Jotto.
- JottoViewer: a GUI class for generating the user interface for playing Jotto. You shouldn't need to edit any code in here.
- JottoModel: the model class that controls the state of all the values in the Jotto game. Search through the code and find all of the TODO/methods marked in the code. Everywhere that you see TODO is where you must add the necessary code to play Jotto.
- You're also given a file of five-letter words though you're free to use your own file.

The code has been designed so that you can implement a few, marked (TODO) methods in JottoModel, implementing the methods described in comments of that class and designing state appropriately for the model to guess the human-user's word.

You may find it useful to write helper methods in the class you write.

*******Load the file `kwords5.txt` from the GUI/File menu before starting to play.*******

Model/View Communication



A diagram of the communication between the Jotto viewer (in green) and the Jotto model (in blue) is shown to the left. The view, or the graphical-user interface, calls the model, which controls the state of the Jotto game. The methods you need to write are in solid blue.

To begin a game, first load the file`kwords5.txt` from the GUI/File menu. This will call the model's initialize method and store the words from the file into the instance variable `myWordList`. The model's `newGame` method will be called — you write this code, and the computer should then begin guessing the user's word from a word in `myWordList`.

You must start the model-view-model-view… communication by sending a guess to the view via`doGuess`. The view will then send the number of letters in common back to the model. The model must then make a guess and send the guess to the view which will send back a number of letters in common (by calling `process` which calls `processResponse`), and so on. Thus model-view communication is started by the model in this scenario and works like this:

- New game started, the model sets its state and then the model calls`myView.processGuess(guess)` to show the user the word the computer/model is guessing — this is done by calling the model's private `doGuess` method.

- View calls model's `process` method which calls `processResponse` (assuming state set appropriately above) with the number of letters in common with the last guess. If this number is six, the game is over. `processResponce` will need to update the list of possible guesses by comparing words and getting their `commonCount`, or the number of letters that the two words have in common. Once you update the wordList of possible words, call `doGuess` with a new guess unless there are no more guesses left. This step repeats.In addition to the above steps, the JottoModel can communicate with the JottoView by calling some private, helper methods:
  - `showModalMessage` which pops up a dialog box the user must take action with — e.g., when the game is over.
  - `messageViews` to show an informational message in the view, e.g., the number of guesses left. This is shown in the textbox in the lower part of the GUI/view in the screen shots above.

A Common Mistake

After finishing one Jotto game the user may want to play a second game. The user should be able to just go to New Game -> Play New Game without having to load in a new word file every time they play.

One thought on "Jotto — Full WriteUp"

Pingback: Jotto (aka Giotto) | Catalyst Art

Comments are closed.