COMS W4115 Project Proposal

GAQ

Generator of Adaptive Questionnaires

Esther Kundin, Ayla Brayer

June 7, 2011

I. Motivation:

As everyone who has visited a doctor recently knows, appointments to see doctors can take a long time. Filling out the questionnaires can take a while too. As doctors' offices are increasingly computerized, it is conceivable that diagnostic tools can be generated to hone in on diagnoses based on answers to simple questions about a patients' symptoms. Filling out secure adaptive diagnostic questionnaires, that ask questions based on your previous answers, would help doctors save time during visits and hopefully get you out of the office faster. A language that would require no prior programming experience and would be intuitive to use would be used to generate adaptive questionnaires that specialists can use to pre-diagnose patients before they even get into the office. The language should be simple enough that doctors or even their assistants should be able to use it to tailor questions to their clientele easily.

Similar questionnaires can be used by car mechanics to diagnose car trouble. Yet another application for an adaptive questionnaire generator would be to customer-satisfaction surveys or even phone menu systems. All of them would benefit from a more adaptive approach. Rather than forcing people to answer reams of unrelated questions, adaptive questionnaires can adapt to user inputs and have users answer

fewer but more targeted questions. This would make the taking of surveys less tedious, which would in turn get more people to take them.

II. The Language:

GAQ has a simple and intuitive design to insure that users with little or no programming experience will find it easy to use. Each command will be written in a single line and tokens will be separated by space.

- **Simple Data Types:** GAQ will support floats, ints, booleans. True value for a Boolean is "yes", false value for a Boolean is "no".
- **Complex Data Types/Built-in Functions:**
    - text – similar to C++ strings
    - Question (Question-text, Response-data-type) - Complex type that will host a question text and the anticipated response data type (simple data type)
        - Data Member:
            - Response – holds the response value
        - Member Function: Member functions are called on the object using a "->" syntax.
            - next_question (Boolean-condition , Question) - Will attach the next question to be asked, if the boolean condition evaluates to 'true'. An empty Boolean condition will always evaluate to true. Boolean conditions will be evaluated in the order that they were attached via next_question. Thus, if two possible conditions are met, the first one attached via next question will be the first one triggered. To reference the value of the variable of the parent, the keyword **response** is used. References to other variables are not allowed.
    - Result – Complex type that holds the results of the questionnaire – the list of questions asked and their answers
    - results_to_file (file-name, Result) - Prints the questionnaire results into the given file
    - results_to_stdout(Result) – prints the questionnaire results to stdout

- run_que(Question) – Runs the questionnaire, starting from the given question, returns a Result object
- print (text) – Prints a text to the output stream

- **Relational operators:** < = , < , >= , > , = , !=
- **Logical operators:** & (and), | (or)
- **Mathematical operators** (apply only to float or int)**:** + , –, *, /
- **Comments :** All characters between /* and */ will be treated as comments
- **Line endings**: Line endings will be marked by a period ('.') character

III. Sample Program:

```
Question root("Are you ill?", boolean).

Question ill("Do you have a fever?", boolean).

Question well ("Are you here for a checkup?", boolean).

Question howHigh("How high is your fever in Farenheit?", float).

Question er("Have you been to the ER?", boolean).

Question.other("What are your other symptoms?", text).

root->next_question(response = yes, ill).

root->next_question(response = no, well).

ill->next_question(response = yes, howHigh).

howHigh->next_question(response > 104.9, er).

howHigh->next_question(other).

/* now questionnaire is set up, so run it, and print out the results
to stdout and file. */

print("Questionnaire started!").

Result result = run_que(root).

results_to_file("result.txt", result).

results_to_stdout(result).

print("Finished!").
```