Project Overview

The Computer Science Department wants an automatic quiz delivery system. Instructors put questions into the system, the system presents the questions to students, automatically grading them. The instructor can see the students' performance to copy to a gradebook or checkpoint.

Introductory course labs would benefit from quizzes at the start of the lab that review lecture topics (especially those to be used in the lab). Students would benefit from a system that can present multiple versions of the *same* quiz (or question) so that the student retakes the quiz until they get a passing score. The different versions would be to avoid students memorizing the answers (enough different questions, different types of questions on the same material, different correct answers, and different distractors should all be possible). Here the passing grade should be shown to the lab instructor so that the student can go on with the lab checkpoint recorded.

Machine grading implies question types that admit machine grading. Efficient use of instructor time implies that there must be some sort of bulk import/export of questions is necessary.

Introduction

The Computer Science Department has a problem: lab sessions in introductory courses should begin with a set of review questions that involve tracing code, rearranging code to make it work, and defining terms important for the lab. This can be done with paper and pencil with the lab instructor checking each answer, done on-line with the lab instructor checking the answers (the computer is solely used as a word processor), or could be presented **and** graded by the computer, providing the lab instructor with a summary at the end of the process.

If the computer presents/grades the work, it is possible to imagine having a collection of questions (or a collection of answers/distractors for a given question) to cover various topics. Different students would get varied presentations of the questions and, more importantly, a student finishing the "quiz" with an unsatisfactory score could be given *another* version of the quiz, letting them practice what they learned rather than play-back what they have memorized.

No satisfactory free or open-source solution to this problem has been found. Your team is being asked to design and implement the **Quizzinator 9000**, a Web-based solution.

Gradiance (see Ullman's *Gradiance*) seems close. The ideas explored in the paper should inform your design.

Operation

The faculty has a view of two different types of users: faculty and student. Each has a different view of the system when running it.

Faculty

Faculty will log in to the system in order to have authority to define questions and quizzes.

Types of questions is discussed below but it is expected that questions will be given short names (to make adding them to quizzes easy) and different wrong answers can lead to different partial credit and (optional) help information for the student. There *must* be a straight-forward syntax for bulk upload of questions (and, reciprocally, a way to export the question data for ease of reuse). **Preference**: for an existing question format to permit interoperability if Q9000 is not used.

A quiz is a collection of questions. Multiple questions might be marked as covering the same material so the quiz can select from the collection when presenting the quiz.

There could be (discuss with product owner) a *class*, a collection of student accounts associated with zero or more quizzes. Makes it easier for students to find assigned quizzes.

Student

Students will log in and the system will keep track of progress against various assigned quizzes. This makes it easier to make sure that repeated questions/distractors happen less frequently.

Student logs in and finds a quiz they are assigned. Questions are presented in a randomized order. As each is answered, if the student is incorrect, any provided guidance for the wrong answer is shown. Students will see all the questions in the quiz before their score is checked against the required score to pass. If they pass, a message for the lab instructor is shown (with the quiz identifier and student name prominently displayed).

If the student does not pass, the program offers to run the quiz again (with different ordering and distractors).

Which questions the student answered (and which answers they gave) should be logged so the student can look at their past history.

Questions

The types of questions are important: multiple choice, permutational multiple choice, and Parson's problems.

Multiple Choice A standard multiple choice question: a *root* (or, perhaps, multiple roots), some number of *correct* answers and some number of plausible *distractors*.
Simplest form: Show root, one answer, and some number of distractors (order of results randomized). Student indicates which result they pick.
Each distractor could be associated with a *hint*, shown when the distractor is (incorrectly) selected as the answer.
When **retaking** a quiz, Q9000 should randomize the selection of root, answer, distractors, and order as much as possible. Chance of repeating a root on two successive runs of the same quiz should be fairly low.
Alternative forms: Show root, some number of answers and distractors (random order). Student indicates subset of results that are correct. Score based on match of correct subset. Hints might be included for missed answers in this case.

Permutational Multiple Choice In an effort to overcome some of the shortcomings of multiple choice questions, Farthing, *et.al.* developed *permutational multiple choice questions*. Here **two** closely related roots are shown with a larger selection of results. The results include two answers, one best for *each* of the two questions and a collection of plausible distractors. Scoring is based on getting **both** answers correct (no partial credit).
Again: order of results to be randomized. Distractors might have different hints for each root. Answers might have hints if they are skipped and different hints if they are chosen for the wrong root.
There might be multiple roots. If there are, there will be two *sets* of roots, one for each part of the question.
This question type needs on-screen help.

Parson's Problems An ordering exercise: a question is followed by a snippet of computer code is presented in mixed-up order, possibly with distractors. The student must put the lines in order to answer the question.
Order should be randomized (**and** it should never show lines in correct order). There might be more distractors than are specified to be shown. It might be possible to show fewer (or more) distractors on a subsequent taking of the quiz.
See Du's *Review of Research on Parson's Problems* for more information.