Computer Science Tripos – Part II – Project Proposal

A System for Giving Programming Exercises Adaptive Difficulty

R. J. McFarland, Homerton College

Originator: M. B. Gale

20 October 2016

Project Supervisor: M. B. Gale

Director of Studies: Dr J. Fawcett

Project Overseers: Dr D. J. Greaves & Prof J. G. Daugman

Introduction

Different people learn at different speeds, and learn some things more quickly than others. When a group of students are given a set of programming exercises, this typically isn't taken account of. The aim of this project is to design a system that varies the difficulty of a programming exercise depending on how the individual has performed completing previous exercises.

The difficulty of an exercise can be measured using two metrics: how complex the problem being solved is and how much code the student is expected to write. The system will vary both of these to adapt the difficulty of proposed exercises automatically (i.e. the system will not look for preprogrammed mistakes, but rather interpret the mistakes made and react accordingly). When an exercise is completed to a sufficient standard, a similar exercise will be given, but expecting more code to be written. When the system has determined that the problem has been mastered, a more difficult problem will be presented, requiring less code to be written again.

Resources required

I shall use my own Macintosh laptop for the majority of this project. Backup will be to GitHub and to a 5TB hard drive I keep in my room. I shall make use of the Java standard library, as well as the JavaFX library for the graphical user interface element. I require no special resources.

Starting point

I am able to program in Java, and have used JavaFX before for the Part IB group project. During my A-Levels I wrote a program to test the maths abilities of Year 4 students, which involved the programmatic generation of various kinds of maths questions.

Work to be done

The work for this project can be split up into the following sub-projects:

1. A representation of a programming exercise must be coded up to allow the final product to manipulate them.
2. A heuristic that measures the difficulty of a given exercise must be chosen and implemented.
3. A heuristic that assesses how well a "student" has completed an exercise must be chosen and implemented.
4. I must devise a system, using the previous three items, that determines which exercise should be given to the student next, as well as how much of the required code is already filled in, based on the student's performance completing previous exercises.
5. A Graphical User Interface should be designed to enable a user to manually adjust the content of a given programming exercise.
6. In order to test the program, a student will be simulated by creating a system that tells the program what errors were made where and how long the exercise took to be solved, so that the program's response to various stimuli can be measured.

Success Criteria

The project will be considered completed when:

1. I have a system that presents the same initial problem to all students.
2. The system can determine that code written by the student solves the given problem.
3. On registering completion of the exercise, the system will then present a new problem to the student.
4. If the student solved this problem with sufficient ease as measured by my performance heuristic, then this new problem will be more difficult, as measured by my difficulty heuristic.
5. If the student failed to solve the problem, or took too much time or made too many errors, then the next problem will be easier.

6. The system will adjust the difficulty of a given problem in line with the difficulty heuristic by changing the amount of code required to solve the problem, and also the underlying problem being solved.

Possible extensions

- This system could be improved such that it identifies which concepts a student is struggling with and adjusts the content of future exercises to encourage learning of these concepts.
- The program could produce a graph of progress against time to motivate learners.
- A simple game that utilises this system could be made, to showcase how the system might be used to teach programming.

Timetable

24th Oct - 6th Nov:

- Research adaptive difficulty and see how it has been done before.
- Research how errors in code have been quantified in the past.
- Research where and how simulated users have been designed to test systems.

7th Nov - 20th Nov:

- Describe how code that solves an exercise could be split up into discrete sections, and a way to store how well each of those sections was completed.

Milestones:

- Have a representation of a programming puzzle in Java code.

21st Nov - 4th Dec:

- Prototype and compare heuristics to measure the "difficulty" of a given programming exercise, based on the difficulty of the goal to be achieved, and how much of the code is presented initially.

Milestones:

- Select and implement a difficulty heuristic.

5th Dec - 18th Dec:

- Prototype and compare heuristics to measure performance, based perhaps on how much time was taken, the number of errors made and the time the solution takes to run.

Milestones:

- Select and implement a performance heuristic.

19th Dec - 1st Jan:

- Slack time over Christmas to catch up if I need to.

2nd Jan - 15th Jan:

- Implement a system to adapt the difficulty of the programming exercises by presenting a new, more difficult problem each time an exercise is completed satisfactorily, and presenting an easier one if the student is struggling too much.

Milestones:

- Have a system that will present a problem to a student, determine that the student has solved the problem with the code they have written, determine if the student found it too easy or too hard, and present the student with a new problem that is either easier or harder respectively.

16th Jan - 29th Jan:

- Write Progress Report.
- Improve the system by implementing the concept of changing the amount of code that needs to be written to solve a given problem as another way of affecting its difficulty.

Milestones:

- Have the system extended such that it may require the student to write more or less code to make more fine grained adjustments to the difficulty of a programming problem.

30th Jan - 19th Feb:

- **3rd Feb:** Submit the Progress Report
- Design a user interface to change the content of the programming exercise to be solved.

Milestones:

- Have a user interface with controls to determine the exact difficulty of the problem about to be presented.

20th Feb - 5th Mar:

- Write unit tests for all units.

Milestones:

- Have unit tests written for every unit of the code.

6th Mar - 19th Mar:

- Test the implementation by simulating a student "using" the system by "solving" problems with varying success.

Milestones:

- Have a simulated student that will tell the system what mistakes were made in the code and where, as well as how long it took to solve the problem (along with any

other parameters deemed important to measuring the performance of the student), such that the response of my system can be measured against the performance of a student using it.

20th Mar - 2nd Apr:

- Slack time for catching up and doing extensions.

3rd Apr - 16th Apr:

- Write Introduction and Conclusion (about 2,500 words).
- Write up Proforma (excluding word count), Declaration of Originality, Project Proposal and Cover Sheet.

Milestones:

- Have the above sections of the dissertation written in draft form.

17th Apr - 30th Apr

- Write Preparation and Implementation (about 5000 words).

Milestones:

- Have the above sections of the dissertation written in draft form.

1st May - 14th May

- Write Evaluation (about 2500 words).
- Write Contents Page, finish Bibliography, Appendices and Index as required.
- Fill in word count of Proforma.
- Submit draft to DoS and Supervisor.

Milestones:

- Have the above sections of the dissertation written in draft form.

15th May - 19th May

- Incorporate feedback into dissertation.
- **19th May:** Submit dissertation.