Philip Yeeles

Nico: An Environment for Mathematical Expression in Schools

Computer Science Tripos

Selwyn College

April 15, 2012

Proforma

Name:	Philip Yeeles
College:	Selwyn College
Project Title:	Nico: An Environment for Mathematical Expression in Schools
Examination:	Computer Science Tripos, May 2012
Word Count:	TBC[1] (well less than the 12000 limit)
Project Originator:	P. M. Yeeles (`pmy22`)
Supervisors:	Dr S. J. Aaron (`sja55`), A. G. Stead (`ags46`)

Original Aims of the Project

The aim of the project was to develop an application in the Clojure programming language which would allow users to express mathematical calculations using a graphical notation. The software was to be able to generate an abstract syntax tree from the graphical notation, evaluate it and pass the results back to the application in under 300ms. An extension to the project was to conduct a user study to evaluate the utility of the software.

Work Completed

I have successfully designed and implemented the application detailed in the previous section. That is, I have developed an application in which it is possible to

---

[1]This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

express calculations using a graphical notation, that generates an abstract syntax tree from the language and that is able to parse the tree and return the results in under 300ms. I have also conducted a user study to assess whether or not the software is actually of use with regard to mathematics education.

Special Difficulties

Learning the Clojure programming language.

Declaration of Originality

I, Philip Michael Yeeles of Selwyn College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date April 15, 2012

Contents

List of Figures

Acknowledgements

Chapter 1

Introduction

The aim of this project has been to design and develop a notation and accompanying application to act as a learning aid for pre-algebra arithmetic by increasing visibility, reducing the number of hidden dependencies and making the flow of data obvious to the user. I have successfully developed such a system, intended initially for pupils in Year 5 (though extensible, through the creation of alternative question sets, to other age groups), and, as an extension, conducted a user study to assess its utility.

In this chapter, I will discuss my motivations for choosing this project, the pros and cons of the handwritten system it is attempting to augment, the technical challenges involved in developing such a system and related work that has previously been conducted with similar goals.

1.1   Motivations

My motivations behind this project lay in the limitations of the handwritten approach to solving mathematical problems that I had observed both in my own learning and in my own teaching experience. What follows is an evaluation of the pros and cons of the handwritten method of performing arithmetic calculations according to Blackwell and Green's "Cognitive Dimensions" framework [1], and a discussion of the properties a useful alternative notation should have.

1.1.1   Evaluation of the Handwritten Approach

There are a number of problems with
handwritten, pre-algebra arithmetic
that this project seeks to rectify. First
of all, the fact that it is handwritten
entails a high level of viscosity: it
is difficult to make changes to a written
calculation without sacrificing clarity.
In particular, there is a lot of repetition
viscosity involved in the modification of
an existing piece of work; if a number
is changed that is used in several
calculations, then it is time- consuming
to change it everywhere it appears in

24+35=59	24+35=59	24+35= ~~59~~ 49
12+48=60	12+48=60	12+48=60
59+72=131	59+72=131	~~59~~ 49+72= ~~131~~ 121
1+60=61	1+60=61	1+60=61
131+61=192	131+61=192	~~131~~ 121+61= ~~192~~ 182
(a)	(b)	(c)

Figure 1.1: Illustrating the hidden dependencies
and viscosity inherent in pre-algebra, handwritten
arithmetic. The original calculation is shown in
(a), has its otherwise-hidden dependencies high-
lighted in (b), and is altered slightly in (c).

the working. If several calculations are dependent upon each other, then this entails
a lot of knock-on viscosity in recalculating each stage after changing the number.
This is exacerbated by the hidden dependencies between chained calculations in
handwritten arithmetic (*Fig. 1.1*).

1.1.2   Requirements of a Replacement System

1.2   Technical Challenges

Developing such an application comprises two main challenges: developing a
backend that is capable of creating, storing, editing, deleting, evaluating, nesting
and calculations, and a graphical, user-facing frontend that is able to render
calculations into the devised notation, and allow the user to perform operations
upon the notation that affect the underlying calculation.

I chose to use the Clojure language as it provided many features that would prove to
be useful over the course of the application's development. As a dialect of LISP,
Clojure is a homoiconic programming language – that is, a programming language
in which code is represented as a data structure – which made passing around and
performing operations upon calculations themselves, rather than just their results,
considerably easier. A calculation can simply be represented as a piece of code,
which can then be utilised as needed.

As the user experience is so crucial to the success of the application, it was also important that there be well-established GUI libraries available. Clojure runs on the Java Virtual Machine (JVM), which puts Java's considerable standard library at one's disposal, whilst still being able to program in a LISP. As I am familiar with Java and the Swing GUI libraries, it was advantageous to be able to leverage this knowledge in designing the application's interface.

1.3 Previous Work

There already exists a wide variety of educational software for mathematics, but much of this is in the form of "games", in which a series of mathematical problems to be solved is poorly disguised as a game – indeed, such problems would be more accurately said to be embedded into a game, rather than becoming the game themselves. Thus, the object becomes not to solve the problems, but to play the game that happens to surround the problems. Such software also does not often offer any means of solving the problems, other than the traditional pen-and-paper method (with a piece of paper next to the computer screen), or the mental approach. Hence, what the user is then presented with is essentially a game and a worksheet, awkwardly interleaved. In some cases, it is even possible for the user to simply press arbitrary buttons until they pass the questions, effectively removing the maths element of the game and replacing it with a series of short breaks in gameplay.

There also exist a few applications intended to represent calculations on a computer in novel ways. A relatively common approach to this has been to try to make on-screen calculations more like on-paper calculations. *Pi Cubed* takes this approach by trying to make complex calculations appear as they would be written in an exam or exercise book [2]. *Soulver*, conversely, tries to achieve this by simulating "back-of-the-envelope" calculations, whereby notes in English augment the calculation [3]. Another approach is that of the *Scrubbing Calculator* [4], which extends the *Soulver*-style environment by helping the user to solve equations by dragging values to increase and decrease them, showing how changing a value affects the overall result. Values can be linked by dragging a line between them, which means that they are two instances of the same value – hence dragging one changes the value at every location in which it appears. This is a neat means of visualising equations, but it, too, is not intended for use in education, and still requires the user to be able to formulate some kind of equation. The *Scrubbing Calculator* is more a tool for facilitating algebraic understanding, as opposed to arithmetic

understanding; indeed, it is inherently a **calculator**, and so does not encourage thinking about how to work out the arithmetic parts of a calculation manually.

1.4   Summary

Existing educational "games" for mathematics either have too much focus on being a game, rather than helping to learn mathematics, or are such that the mathematical element is circumventable. There exists software to aid in calculation and arithmetic by representing it clearly, but it is not intended for educational use, and often its purpose is to make on-screen calculations appear as one would handwrite them.

There is a niche for a tool for use in education that represents calculations in a visual manner, with a particular focus on making the method by which arithmetic problems are solved clear. My project aims to provide an environment in which the user can explore the many ways in which a problem can be solved using a novel graphical notation.

Chapter 2

Preparation

2.1 Overview

lol

2.2 User Interface

lol

Chapter 3

Implementation

lol

Chapter 4

Evaluation

lol

Chapter 5

Conclusion

lol

Bibliography

[1] A. F. Blackwell and T. R. G. Green. Cognitive dimensions of information artefacts: a tutorial, 1998.

[2] Sunset Lake Software LLC. Pi Cubed | Sunset Lake Software. `http://www.sunsetlakesoftware.com/picubed`, 2012.

[3] Acqualia Software Pty. Ltd. Soulver | Acqualia. `http://www.acqualia.com/soulver/`, 2011.

[4] B. Victor. Scrubbing Calculator. `http://worrydream.com/ScrubbingCalculator/`, 2011.

Appendix A

Project Proposal

The original project proposal follows.

Nico: An Environment for Mathematical Expression in Schools

P. M. Yeeles, Selwyn College
Originator: P. M. Yeeles
18 October 2011

Special Resources Required
- PWF account
- SRCF account
- GitHub account
- Toshiba Satellite L500-19X (Intel Pentium T4300 2.0GHz, 4GB RAM, 500GB disk)
- Samsung NC10 Plus (Intel Atom 1.66GHz, 1GB RAM, 250GB disk)

Project Supervisors: Dr S. J. Aaron & A. G. Stead
Director of Studies: Dr R. R. Watts
Project Overseers: Dr J. A. Crowcroft & Dr S. Clark

Introduction

Discussions with local teachers have led me to hypothesise that educational software for mathematics could be used to reinforce learning by focussing on method, rather than on a numerical answer. My aim is to develop a problem-solving system aimed at pupils in year 5 in which the solution to a problem can be represented as a tree of operations – a block-based graphical language to describe mathematical method. The correctness of the solution is then assessed with respect to the structure of the tree. The application will be written in Clojure, using JavaFX 2 for the graphical elements, though if this becomes infeasible I will use either the Eclipse SWT or Swing with GUIFTW. This dissertation will determine whether Nico offers an improvement regarding pupils' ability to recall the correct method for answering mathematical problems. The success of the project will be gauged by whether or not the software is able to generate an abstract syntax tree in Clojure from the graphical language and evaluate such a tree, passing the results back to the graphical application and displaying this to user in less than 300ms[1]. As an extension, I will distribute Nico with anonymous feedback forms to local schools, to determine if the software is actually of use in the classroom.

Work that has to be done

The project breaks down into the following sections:–

1. Core system
    a. A syntax for questions and a means of loading them
    b. A set of basic functions available to the student
    c. A means of inputting an answer that can be evaluated on-the-fly
    d. A means of re-expressing the question to reflect how the student works (e.g. *12 × 34 ⇒ (10 × 34) + (2 × 34)*)
    e. A method of validating the answer
    f. A means of tracking the current result of evaluating the method input so far
    g. A system of hints for students who may not know where to start
2. GUI
    a. A collection of drag-and-drop elements that can be used to construct a diagram representing how to solve the question

---

[1] *Interactive multimedia and next generation networks: Second International Workshop on Multimedia Interactive Protocols and Systems, MIPS 2004 Grenoble, France, November 2004, Proceedings (LNCS 3311)* by Roca and Rousseau has this to say on interactivity: "An abundance of studies into user tolerance of round-trip latency [...] has been conducted and generally agrees upon the following levels of tolerance: excellent, 0-300ms; good, 300-600ms; poor, 600-700ms; and quality becomes unacceptable [...] in excess of 700ms."

    b. A means of validating combinations of the drag-and-drop elements

    c. A means of defining functions

    d. A means of viewing documentation

3. Evaluation

    a. Test software on non-technical but mathematically-able subjects

    b. Evaluate the correctness of Nico's translations between diagram and code

4. Extensions

    a. Create and distribute questionnaires to test classes

    b. Collect and interpret data

    c. Create a tutorial mode for new users

Difficulties to Overcome

The following main learning tasks will have to be undertaken before the project can be started:

- Learn Clojure
- Become familiar with JavaFX 2
- Spend time designing the GUI and language

Starting Point

I have spent some months learning Clojure, and continue to do so. I have a good working knowledge of Java and experience teaching Mathematics and IT in Years 4 to 6. The first two years of the undergraduate course have familiarised me with Java and its libraries, and thanks to Clojure's interoperability I will be able to leverage these skills for this dissertation. My experience in schools has allowed me to develop the idea for this dissertation, and has given me an insight into what resources are useful in the classroom.

Below is a mockup of what I aim for Nico to look like. Notice how the method is expressed in the form of a flowchart, with outputs (in this case two) for the answer. Arrows show the direction of input and output, and the QUOT and REM boxes show the result of evaluating the functions being passed to them. Ideally, the question "232 ÷ 7" would also change to reflect how the student breaks down the question.
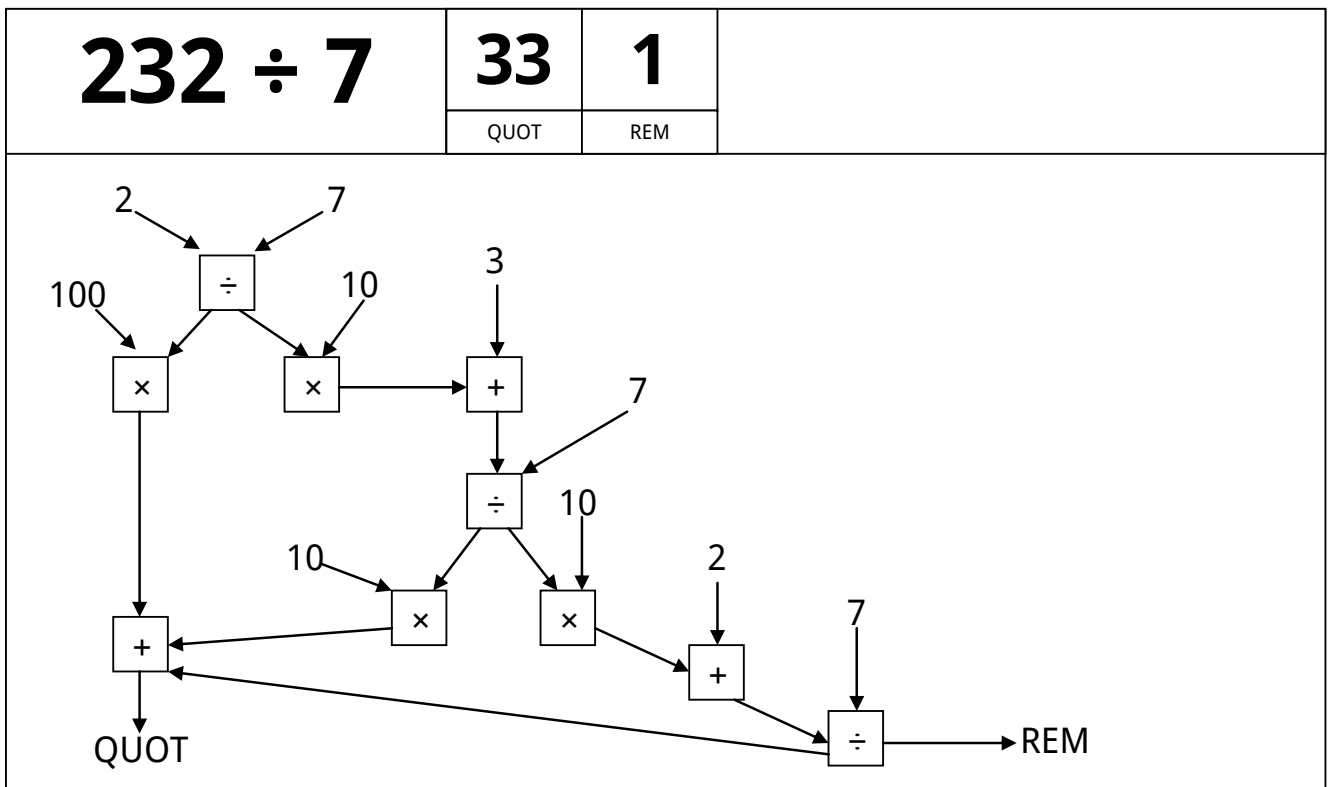
The above solution would auto-generate the following abstract syntax tree represented in Clojure code:–

QUOT is the output of:–

```
(+
  (*
    100
    (:quot
      (div
        2
        7)))
  (*
    10
    (:quot
      (div
        (+
          (*
            (:rem
              (div
                2
                7))
            10)
          3)
        7)))
  (:quot
    (div
      (+
        (*
          (:rem
            (div
              (+
                (*
                  (:quot
                    (div
                      2
                      7))
                  10)
                3)
              7))
          10)
        2)
      7)))
```

REM is the output of:–

```
(:rem
  (div
    (+
      (*
        (:rem
          (div
            (+
              (*
                (:rem
                  (div
                    2
                    7))
                10)
              3)
            7))
        10)
      2)
    7))
```

This assumes that we have a function `div` that takes two arguments `x` and `y` and returns an associative map `{:quot q :rem r}` such that `q` is the quotient of $x \div y$ and `r` is the remainder. Such a function will be included in the basic functions available to the user. Other functions of use would be addition, multiplication, subtraction, exponentiation, function definition and commenting (i.e. labels that are not evaluated), with options available in the question syntax (e.g. `:inhibit+ true`) to restrict arguments to a value of less than or equal to 10 (useful, for example, in questions on long multiplication, to prevent the student from simply giving `(* a b)` as the answer to $a \times b$). Hence a possible means of representing the question above could be:–

```
{:title "232 ÷ 7"
 :topic "arithmetic"
 :answer {:quot 33
          :rem 1}
 :inhibit+ false
 :inhibit- false
 :inhibit* false
 :inhibitdiv true}
```

Resources

This project requires little file space so my Toshiba PC's disk should be sufficient. I plan to use the same PC as well as my Samsung PC to work on the project, and to back my files up to the PWF, the SRCF and GitHub. I will be using Git for version control.

Work Plan

Planned starting date is 27/10/2011.

October 2011

27/10/2011 - 10/11/2011
Work begins. Start covering the problems outlined in *Difficulties to Overcome*. Design the look and feel of the language and application.

November 2011

10/11/2011 - 24/11/2011
Design the question syntax. Implement the question interpreter. Implement the tree evaluator.

24/11/2011 - 08/12/2011
Implement the hints system. Begin work on the GUI.

December 2011

08/12/2011 - 22/12/2011
Finish the non-language section of the GUI. Begin implementing the graphical language.

29/12/2011 - 12/01/2012
Finish implementing the graphical language and its interpreter.

January 2012

12/01/2012 - 26/01/2012
Finish coding the core project. Begin extension work and evaluation.

26/01/2012 - 09/02/2012
Progress report written to be handed in by 03/02/2012. Preparation for presentation on 09/02/2012.

February 2012

09/02/2012 - 23/02/2012
Finish evaluation. Begin drafting the dissertation.

23/02/2012 - 08/03/2012
Finish extension work. Continue drafting the dissertation and evaluate extension work.

March 2012

08/03/2012 - 22/03/2012
Submit first draft of dissertation to supervisors by 16/03/2012. Begin redrafting on receipt of feedback.

22/03/2012 - 05/04/2012
Continuing redraft and resubmission of dissertation.

April 2012

05/04/2012 - 19/04/2012
Continuing redraft and resubmission of dissertation.

19/04/2012 - 03/05/2012
Dissertation complete 01/05/2012.

May 2012
03/05/2012 - 18/05/2012
Dissertation complete. Final edits, corrections. Binding and submission.