

Philip Yeeles

# **Nico: An Environment for Mathematical Expression in Schools**

Computer Science Tripos

Selwyn College

May 11, 2012



# Proforma

<b>Name:</b>	Philip Yeeles
<b>College:</b>	Selwyn College
<b>Project Title:</b>	Nico: An Environment for Mathematical Expression in Schools
<b>Examination:</b>	Computer Science Tripos, May 2012
<b>Word Count:</b>	TBC <sup>1</sup> (well less than the 12000 limit)
<b>Project Originator:</b>	P. M. Yeeles (pmy22)
<b>Supervisors:</b>	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 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

---

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

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 May 11, 2012



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivations . . . . .	1
1.2	Technical Challenges . . . . .	2
1.3	Previous Work . . . . .	2
1.4	Summary . . . . .	3
<b>2</b>	<b>Preparation</b>	<b>5</b>
2.1	Requirements Analysis . . . . .	5
2.1.1	Current System . . . . .	5
2.1.2	Proposed System . . . . .	6
2.2	User Interface . . . . .	8
2.2.1	Prototyping . . . . .	8
2.3	Third-Party Tools . . . . .	21
2.4	Summary . . . . .	22
<b>3</b>	<b>Implementation</b>	<b>23</b>
3.1	System Architecture . . . . .	23
3.1.1	Back-End . . . . .	24
3.1.2	Interaction Handler . . . . .	28
3.2	User Interface . . . . .	32
3.2.1	Notation . . . . .	38
3.3	Summary . . . . .	39
<b>4</b>	<b>Evaluation</b>	<b>41</b>
4.1	Testing . . . . .	41
4.2	UI Evaluation . . . . .	42
4.2.1	Notation Evaluation . . . . .	42
4.3	User Study . . . . .	42
4.3.1	Pilot Study . . . . .	43
4.3.2	Results . . . . .	44

4.4	Goals . . . . .	53
4.5	Summary . . . . .	53
<b>5</b>	<b>Conclusions</b>	<b>55</b>
	<b>Bibliography</b>	<b>57</b>
<b>A</b>	<b>Questionnaire</b>	<b>59</b>
<b>B</b>	<b>Project Proposal</b>	<b>71</b>

# List of Figures

2.1	Illustrating the hidden dependencies and viscosity inherent in pre-algebra, handwritten arithmetic. The original calculation is shown in (a), has its otherwise-hidden dependencies highlighted in (b), and is altered slightly in (c). . . . .	5
2.2	Early designs for flowgraph-based calculation metaphors. . . . .	8
2.3	A more refined sketch of a flowgraph-based language and application.	9
2.4	Early designs for triangle-based calculation metaphors. . . . .	9
2.5	A more refined sketch of a triangle-based language. . . . .	10
2.6	Early designs for circle-based calculation metaphors. . . . .	10
2.7	A more refined sketch of a circle-based language. . . . .	11
2.7	Assorted early designs for other calculation metaphors. . . . .	12
2.8	A sample progression of a calculation in a language based around infinitely halving a rectangle. . . . .	12
2.9	The prototype flowgraph-style language. . . . .	14
2.10	The prototype Sierpiński-triangle-based language. . . . .	17
2.11	The prototype circle-based language. . . . .	19
3.1	A model of <i>Nico</i> 's management of mutable state; the sixth agent is omitted as it is not utilised by the back-end. . . . .	25
3.2	Circles are initialised on the current location of the mouse cursor, generating a name " <i>cn</i> ", where <i>n</i> is the number of circles that have previously been created, and with an initial expression of $(\lambda? \ \lambda? \ \lambda?)$ . . . . .	25
3.3	Three instances of the question text being highlighted as a corresponding circle is moused-over. . . . .	30
3.4	Decision tree to determine the appropriate response to a mouse click event. . . . .	31
3.5	A screenshot of a <i>Nico</i> session. . . . .	33
3.6	The dialogue boxes used to modify existing circles in the application. . . . .	34

3.7	An example of where nesting-based colour-coding would be confusing. The circle containing (2+2) can be considered to be both one and two circles removed from the root circle. . . . .	35
3.8	Examples of colour-coding in <i>Nico</i> . . . . .	36
3.9	<i>Nico</i> 's user interface went through a number of revisions before reaching its current state. . . . .	38
3.10	The notation used also went through a number of revisions before reaching its current state. . . . .	38
4.1	An example of a block of inline tests, taken from <code>core.clj</code> , showing some tests to be run to test the implementation of <code>detect-subs</code> . . . . .	42
4.2	Time taken in milliseconds per subject to answer Question 1. . . . .	46
4.3	Time taken in milliseconds per subject to answer Question 2. . . . .	46
4.4	Time taken in milliseconds per subject to answer Question 3. . . . .	47
4.5	Time taken in milliseconds per subject to answer Question 4. . . . .	47
4.6	Time taken in milliseconds per subject to answer Question 5. . . . .	48
4.7	Time taken in milliseconds per subject to answer Question 6. . . . .	48
4.8	Time taken in milliseconds per subject to answer Question 7. . . . .	49
4.9	Time taken in milliseconds per subject to answer Question 8. . . . .	49
4.10	Time taken in milliseconds per subject to answer Question 9. . . . .	50
4.11	Time taken in milliseconds per subject to answer Question 10. . . . .	50

## **Acknowledgements**

My thanks to Luke Church for his advice regarding user studies, and to Alistair Stead and Sam Aaron for their encouragement and patience.

x

# **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 [2], and a discussion of the properties a useful alternative notation should have.

## 1.2 Technical Challenges

Developing such an application comprises two main challenges: developing a back-end 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. In addition to this, Java’s portability was an advantage, due to the wide range of platforms available both in educational and home settings. A Java-based solution standardises the *Nico* experience across many environments.

## 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 [8]. *Soulver*, conversely, tries to achieve this by simulating “back-of-the-envelope” calculations, whereby notes in English augment the calculation [9]. Another approach is that of the *Scrubbing Calculator* [14], 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

This chapter concerns the work that was completed prior to beginning the project proper. It comprises a requirements analysis, followed by a discussion of the prototyping process of the graphical notation to be implemented in the final application. Finally, there will be a brief examination of the tools used in the development of the project.

### 2.1 Requirements Analysis

#### 2.1.1 Current System

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

$$\begin{array}{lll} 24+35=59 & 24+35=\textcolor{blue}{59} & 24+35=\textcolor{red}{59} \textcolor{green}{49} \\ 12+48=60 & 12+48=\textcolor{blue}{60} & 12+48=\textcolor{red}{60} \\ 59+72=131 & \textcolor{blue}{59}+72=\textcolor{red}{131} & \textcolor{red}{59} \textcolor{green}{49}+72=\textcolor{red}{131} \textcolor{green}{121} \\ 1+60=61 & 1+\textcolor{blue}{60}=\textcolor{red}{61} & 1+\textcolor{red}{60}=61 \\ 131+61=192 & \textcolor{blue}{131}+\textcolor{red}{61}=192 & \textcolor{red}{131} \textcolor{green}{121}+\textcolor{blue}{61}=\textcolor{red}{192} \textcolor{green}{182} \end{array}$$

(a)                    (b)                    (c)

Figure 2.1: Illustrating the hidden dependencies and viscosity inherent in pre-algebra, handwritten arithmetic. The original calculation is shown in (a), has its otherwise-hidden dependencies highlighted in (b), and is altered slightly in (c).

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 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. 2.1). The problem of hidden dependencies is made worse by the low juxtaposability of the system; although the notation is quite visible, in that every calculation can be seen easily on the page, juxtaposing two sets of calculations entails considerable premature commitment on the part of the user, as components cannot easily be edited or relocated due to the system's high viscosity. Whilst viscosity can be acceptable in some situations, it is harmful with regard to modification and exploration within a notational system, once again requiring a non-trivial amount of premature commitment on the part of the user. In many cases, it can actually quicker for the user to start all over again, as opposed to making the changes required to rectify their calculations.

Handwritten arithmetic does have one key advantage: it has a very low initial abstraction barrier. Other than learning the appropriate symbols for each operation and digit, and how to combine them, the traditional system of arithmetic allows a learner to begin using it almost immediately. Algebra, on the other hand, has a much higher abstraction barrier, requiring the much more abstract concept of a variable, rather than a set quantity, to be used effectively. Although it is possible to add abstractions to arithmetic by the use of secondary notation, there is no provision for abstraction included in the primary notation. It can, therefore, be said that algebra is an *abstraction-hungry* system, whereas arithmetic is an *abstraction-hating* system. Without abstractions, arithmetic is easy to get started with, but can be a very verbose and inefficient notation with low visibility, as outlined above.

## 2.1.2 Proposed System

### 2.1.2.1 Overview

To improve upon the standard approach of listing the steps comprising a calculation, a system must acknowledge and try to overcome the drawbacks listed above. To this end, I have designed and developed a notational system and accompanying application that aims to overcome many of the disadvantages inherent in traditional, handwritten arithmetic. The intention of the system is to

reduce viscosity, increase visibility and to remove many of the hidden dependencies that beleaguer the traditional method.

### 2.1.2.2 Functional Requirements

To be an improvement upon the current system detailed above, the new system must satisfy the following properties:-

- Allows the user to create graphical structures representative of complex calculations
- Provides the user with a suitable environment in which to do so
- Allows the user to reposition elements of the structure at will
- Displays the user's current progress
- Is able to evaluate the correctness of the user's answer
- Accepts a file containing a set of questions to be answered
- Displays the current question being answered
- Progresses through the current question set as the user answers each question correctly

### 2.1.2.3 Non-Functional Requirements

The system must also satisfy a number of requirements outside of its basic functionality. These are listed below.

- Offers a significant improvement in visibility over handwritten arithmetic
- Offers a significant improvement in juxtaposability over handwritten arithmetic
- Reduces premature commitment relative to handwritten arithmetic
- Reduces hidden dependencies relative to handwritten arithmetic
- Is interactive: is able to pass results back to the user in less than 300ms<sup>1</sup>

---

<sup>1</sup>Roca and Rousseau [11] have this to say on the subject of 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."

- Is appealing to the target audience of 9- to 10-year-olds without being childish
  - must be applicable to a wider audience if needed (e.g. could it be extended for use in adult education?)

## 2.2 User Interface

As this project is primarily concerned with human-computer interaction, the user interface and experience, and the design thereof, constitutes a significant part of this project. As such, this section outlines the initial development stages of the user interface, first introducing several designs for the graphical notation, and then discussing in more detail three that were developed further.

### 2.2.1 Prototyping

#### 2.2.1.1 Low-Fidelity Prototyping

To try to get some initial ideas for what could become the calculation metaphor of choice for the project, a target was set of devising at least twenty, significantly different, potential designs. These were recorded as very rough sketches, a few of which were refined in larger examples, and three of which were deemed good enough to warrant an application mockup, as detailed in Sec. 2.2.1.2.

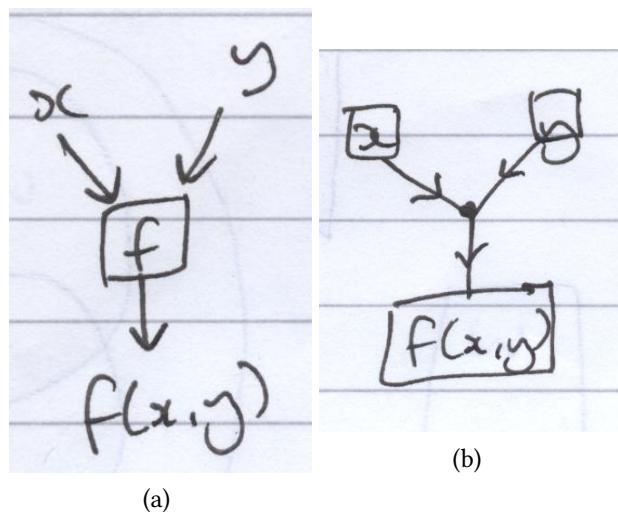


Figure 2.2: Early designs for flowgraph-based calculation metaphors.

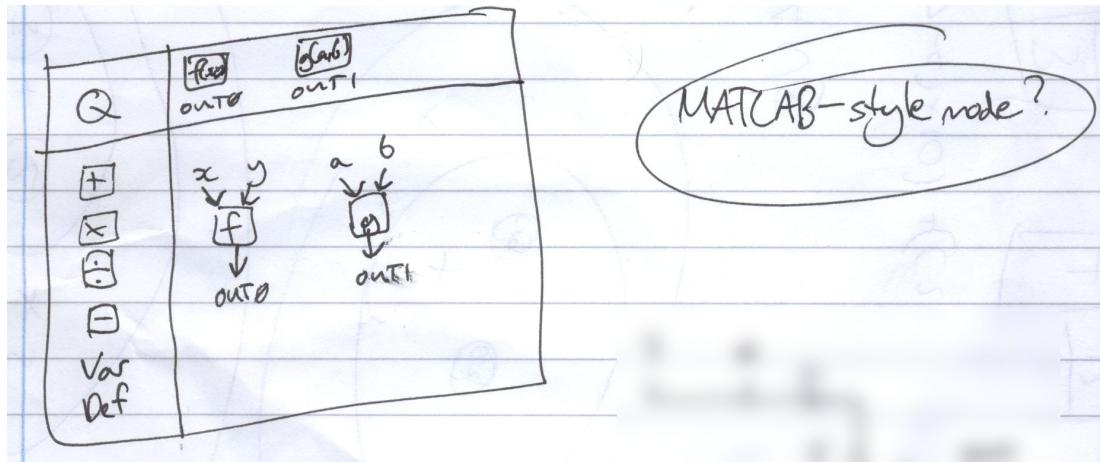


Figure 2.3: A more refined sketch of a flowgraph-based language and application.

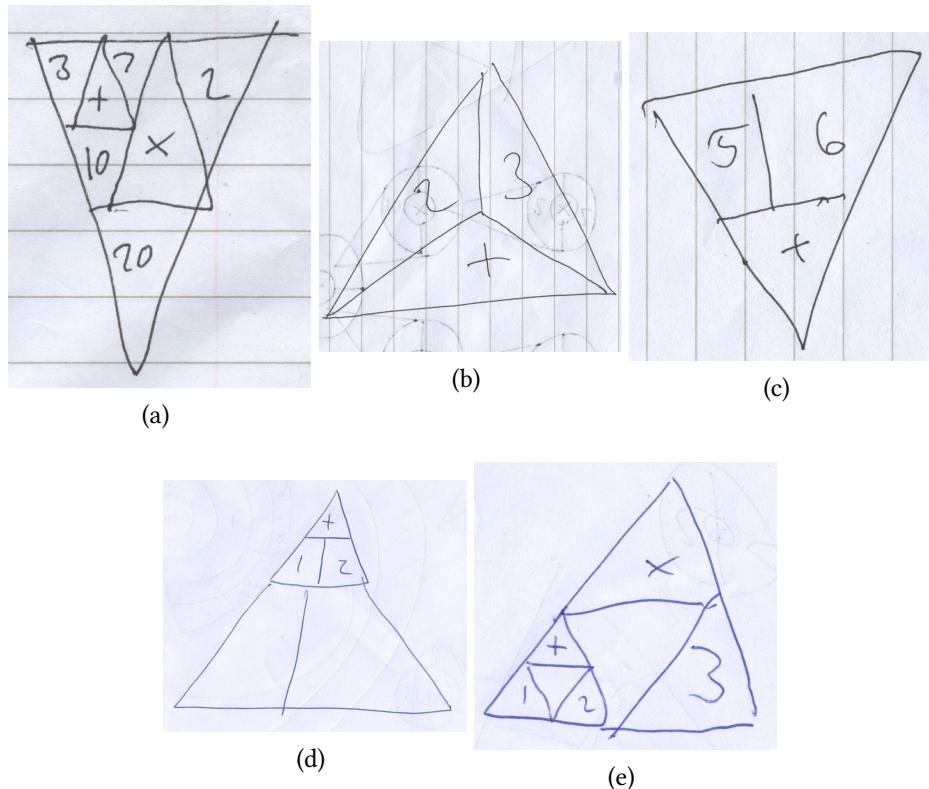


Figure 2.4: Early designs for triangle-based calculation metaphors.

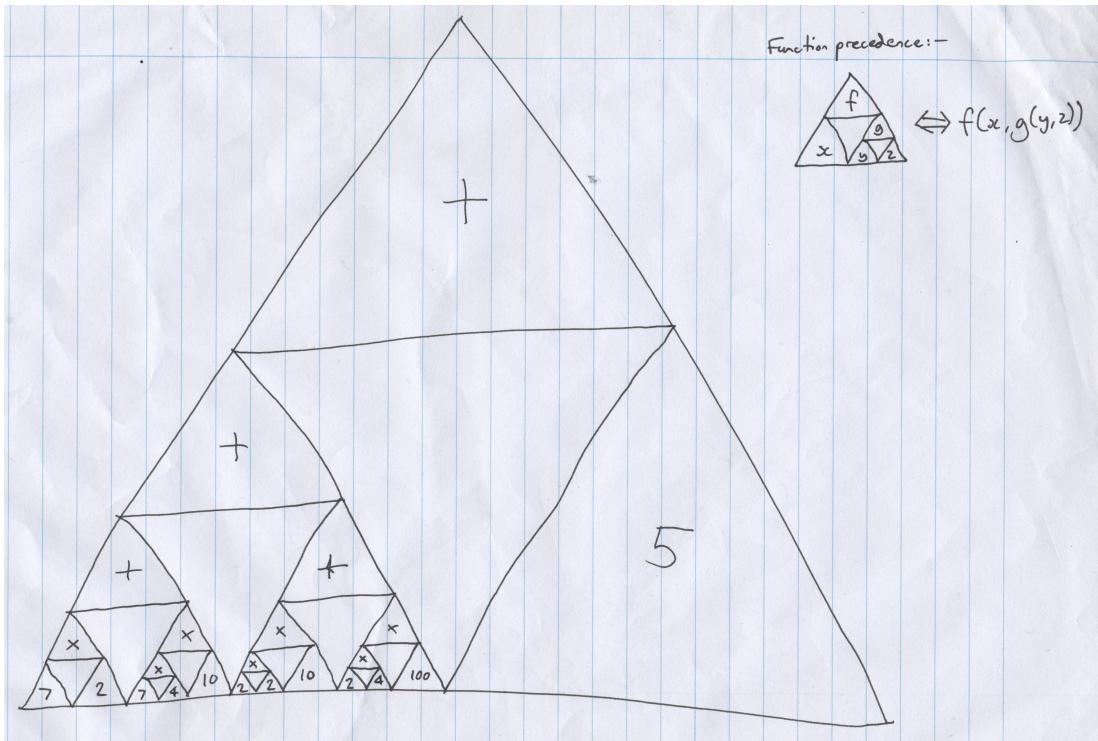


Figure 2.5: A more refined sketch of a triangle-based language.

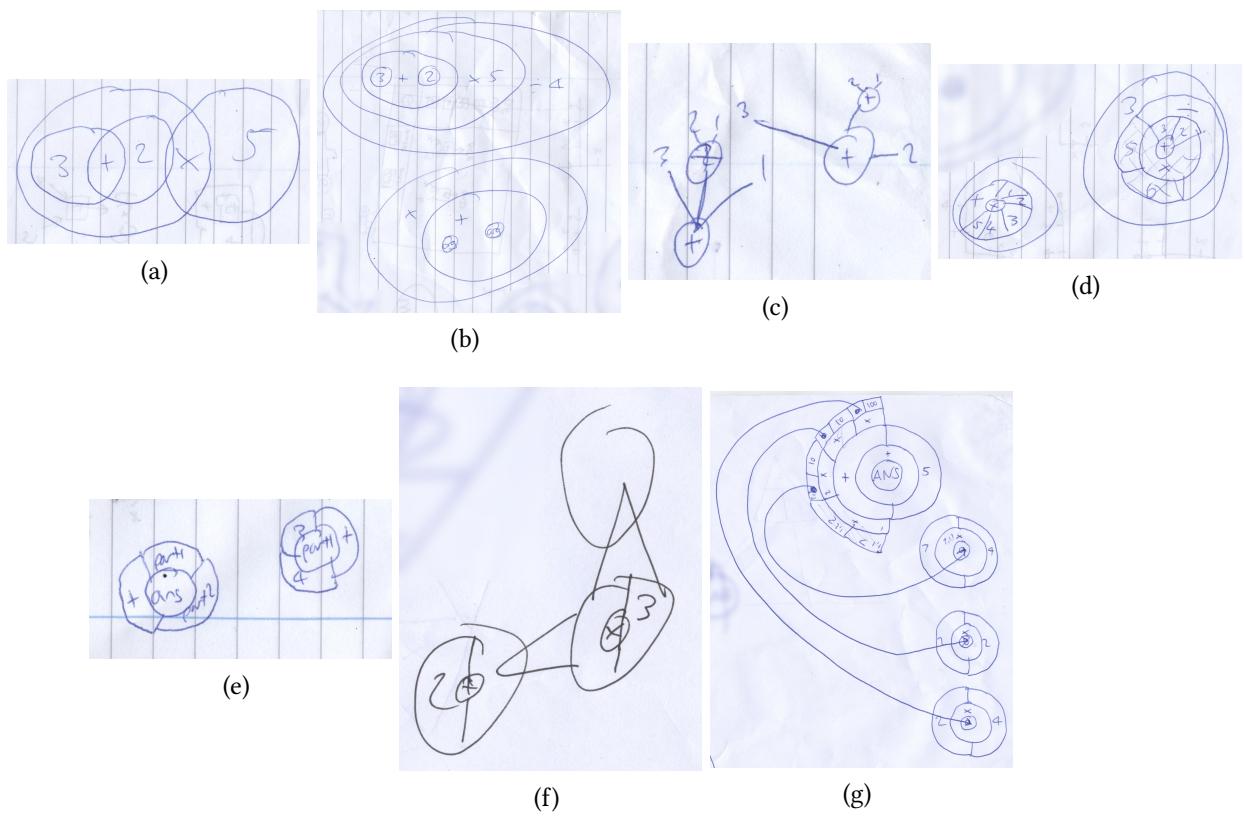


Figure 2.6: Early designs for circle-based calculation metaphors.

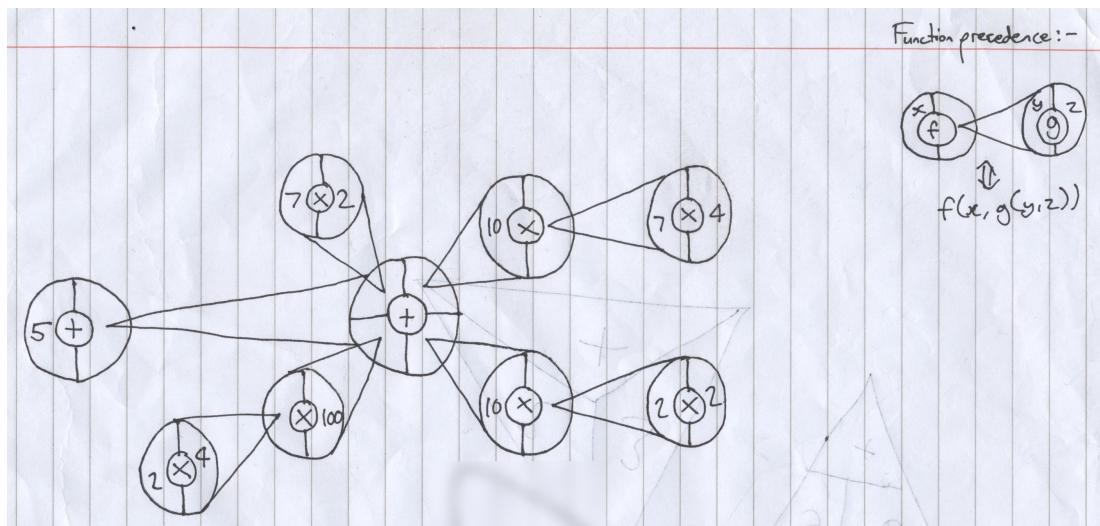
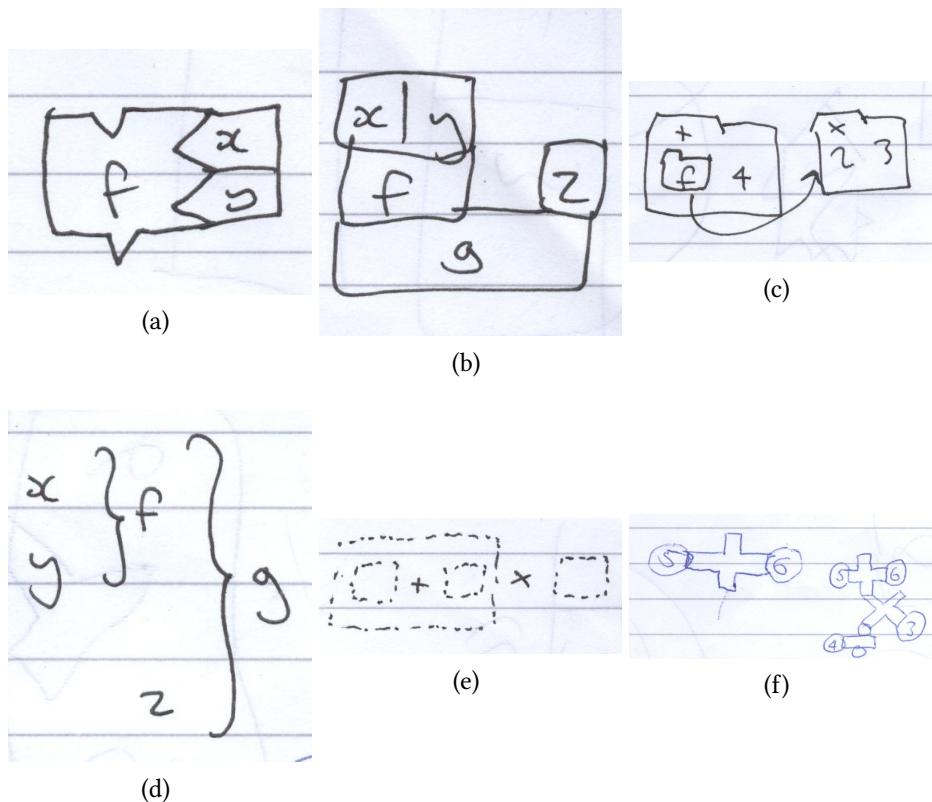


Figure 2.7: A more refined sketch of a circle-based language.



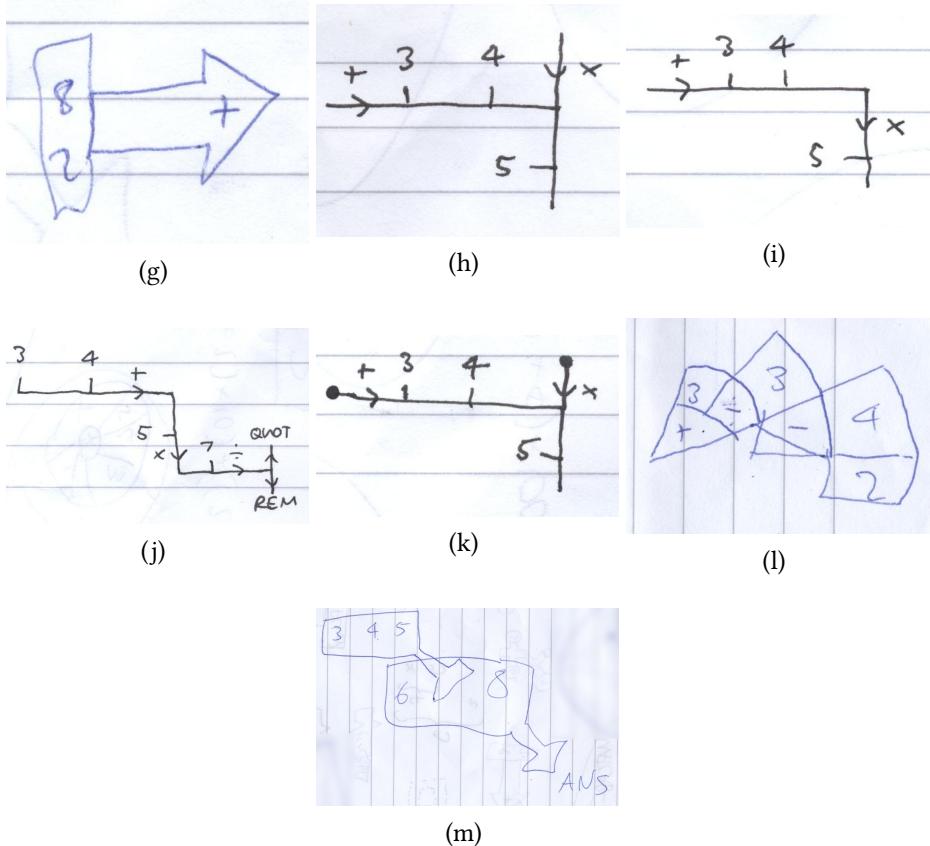


Figure 2.7: Assorted early designs for other calculation metaphors.

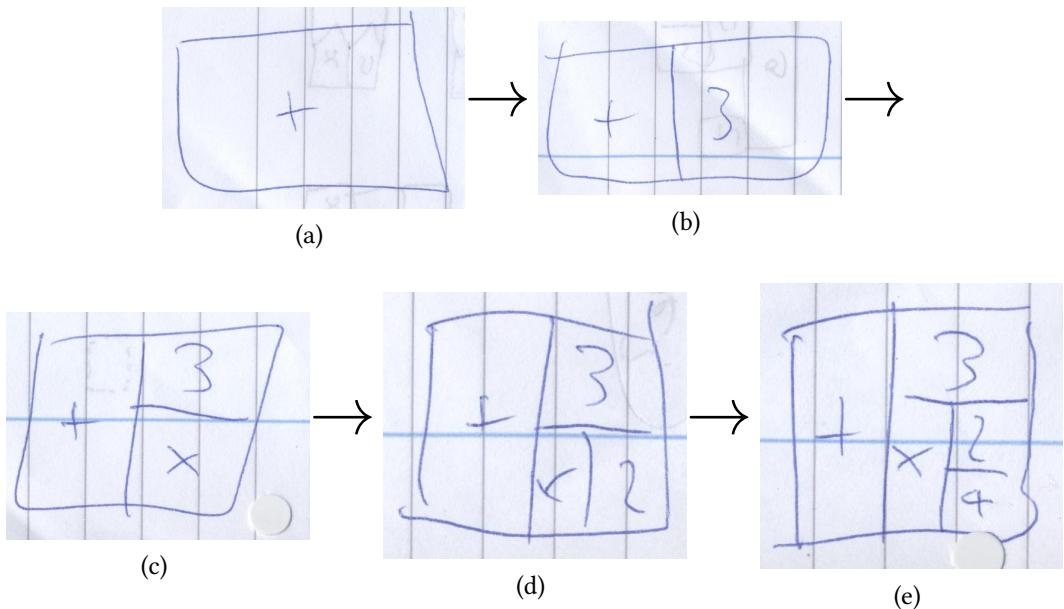


Figure 2.8: A sample progression of a calculation in a language based around infinitely halving a rectangle.

### **2.2.1.2 Detailed Mockups**

Some preliminary designs for the application and graphical language follow, being more mature versions of three of the low-fidelity prototypes presented above. First is a combined flowgraph representation and Read-Evaluate-Print-Loop (REPL) design. The second design uses the Sierpiński triangle as a basis for the visual metaphor. The final design is the circle-based language that eventually became the foundation for the rest of the project.

#### **2.2.1.2.1 Flowgraph Metaphor**

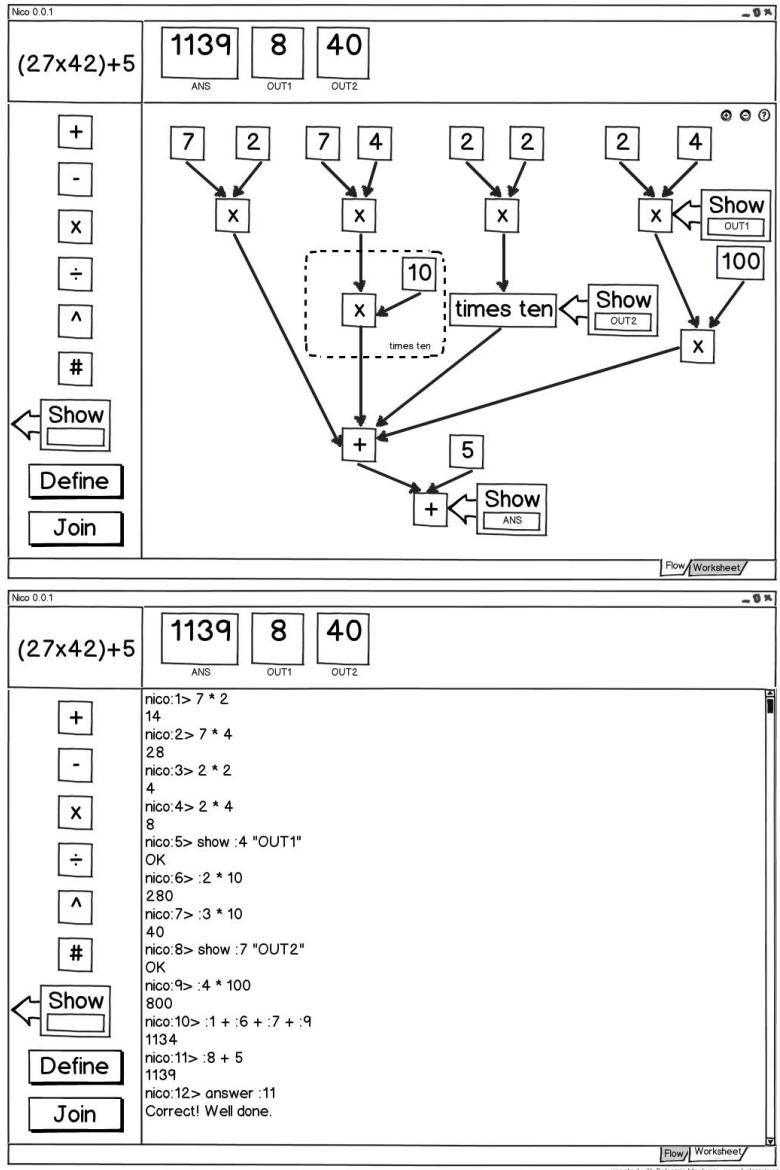


Figure 2.9: The prototype flowgraph-style language.

The flowgraph idea initially outlined in the project proposal and included in the low-fidelity prototypes above appears here as a more mature, revised prototype, showing two screens of a sample application. The top screen shows the proposed application manipulating the graphical language, whilst the bottom screen shows a REPL utilising a simple text-based language for users more comfortable with writing than the visual metaphor.

The application is comprised of one window containing several panels. The current

question being answered is displayed in the top left-hand corner. Below this, a scrollable panel running down the left side of the window contains the fundamental components of the notation: boxes representing numbers and operations, both provided and user-defined, with two buttons below to change the action of clicking the mouse (explained in greater detail below). A panel that runs along the top of the window contains output boxes, areas that display the result of a calculation at a user-determined point in the application. Further boxes appear here as the user demands them, by using the Show function. The ANS box is a specialised case of this, which displays what the user currently wishes to submit as the answer to the question. The canvas panel occupies the majority of the window, and this is the area in which the answer is constructed using the various components of the graphical language. In the top right-hand corner of the canvas are three buttons, to zoom in and out of the current calculation and to bring up a help window. For especially large or zoomed-in calculations, the canvas is scrollable. At the bottom of the window are two tabs, Flow and Worksheet. When Flow is selected, the graphical language is available for use in answering the question. When Worksheet is selected, the user is able to use a simple textual language in a REPL to answer the question.

Of all the prototypes shown here, this version of the graphical language is the most like a traditional programming language, in that it allows the user to define reusable functions, and includes functionality similar to print statements. The icons at the side of the application are used to select the desired function, from a set that includes addition, subtraction, multiplication, division, exponentiation, number input, Show, Define and Join. Addition, subtraction, multiplication, division and exponentiation are all dragged-and-dropped from the icons at the side to a point on the canvas, causing a box containing their respective functions to appear at that point. Number input (#) is similar, but the user inputs a number using the keyboard after the box is created. The box then contains that number. Show displays the output of the junction to which its arrow points, in the output box at the top of the screen with the label input by the user after placing it. If the box with the label does not yet exist, it is created next to the existing output boxes. Define and Join are not boxes to be dragged-and-dropped; they are modes of operation. When Define is activated, dragging the mouse on the canvas draws a box around sections of the diagram – the user is able to define functions by doing so, and by providing a label for the section of diagram that has been highlighted, a corresponding box can be dragged-and-dropped from the list of icons (as show in Fig. 2.2 with the times ten function). When Join is active, dragging between two points on the canvas creates an arrow between them that means that the source of the arrow is used as an argument to the destination of the arrow. In defining functions, if arrows cross the

boundary of the definition box and they are incoming, input is required to future uses of the function. One outgoing arrow is allowed to indicate the output of the function. Using this notation, answers are submitted by showing the output at a point in the calculation to the output box `ANS`.

This particular graphical language tries to have a very high visibility, minimising the number of hidden dependencies between calculations by making every connection explicit: as a dataflow representation of a calculation [2], the flow of information through the diagram is made very clear. This language is considerably less viscous than the handwritten method, as it is trivial (albeit not demonstrated in Fig. 2.2) to remove and replace a box or arrow in the diagram, as opposed to writing out a new set of calculations or replacing several instances of a component by crossing them out. Each box is able to be relocated on the canvas, increasing the juxtaposability of the system, and the user is able to define their own abstractions.

The textual language improves upon the traditional means of handwriting arithmetic by revealing otherwise-hidden dependencies using references to line numbers. As the target audience is not yet required to have formally learnt (or, indeed, encountered) algebra, this notation includes a function `:`, which takes a single number  $n$  as an argument and returns the result of the calculation performed on line  $n$ . Answers are submitted using a function `answer` that takes a single number (shown here using a line reference that is resolved to a number) as an argument. Other functions include addition, subtraction, multiplication, division and exponentiation, all of which take two or more numerical arguments and are used in the familiar infix form. A function `define` is also included, although not shown here. Finally, there is the `show` function, which behaves similarly to a print statement. It takes a numerical argument  $n$  followed by a string argument `str`, and displays  $n$  in the output box named `str` at the top of the screen. If there exists no such output box, it is created next to the existing boxes. Note that `answer n` is logically equivalent to `show n "ANS"`; the `answer` function was included here to increase clarity.

This design was abandoned as it was deemed to be too complex for the target audience (Year 5). The intention of this project is not to implement yet another programming language for learners, and by providing features like the REPL and function definition, the system unnecessarily overprovides. In addition to this, the flowgraph representation had a number of shortcomings. For example, it is not immediately clear in what order the arguments feed into an operation, which is confusing for non-commutative functions such as subtraction. Furthermore, the language is quite verbose. Although this is not inherently a disadvantage, a more

compact representation would be preferable, in order to make expressions written in the notations more readable to people other than the author.

### 2.2.1.2.2 Sierpiński Triangle Metaphor

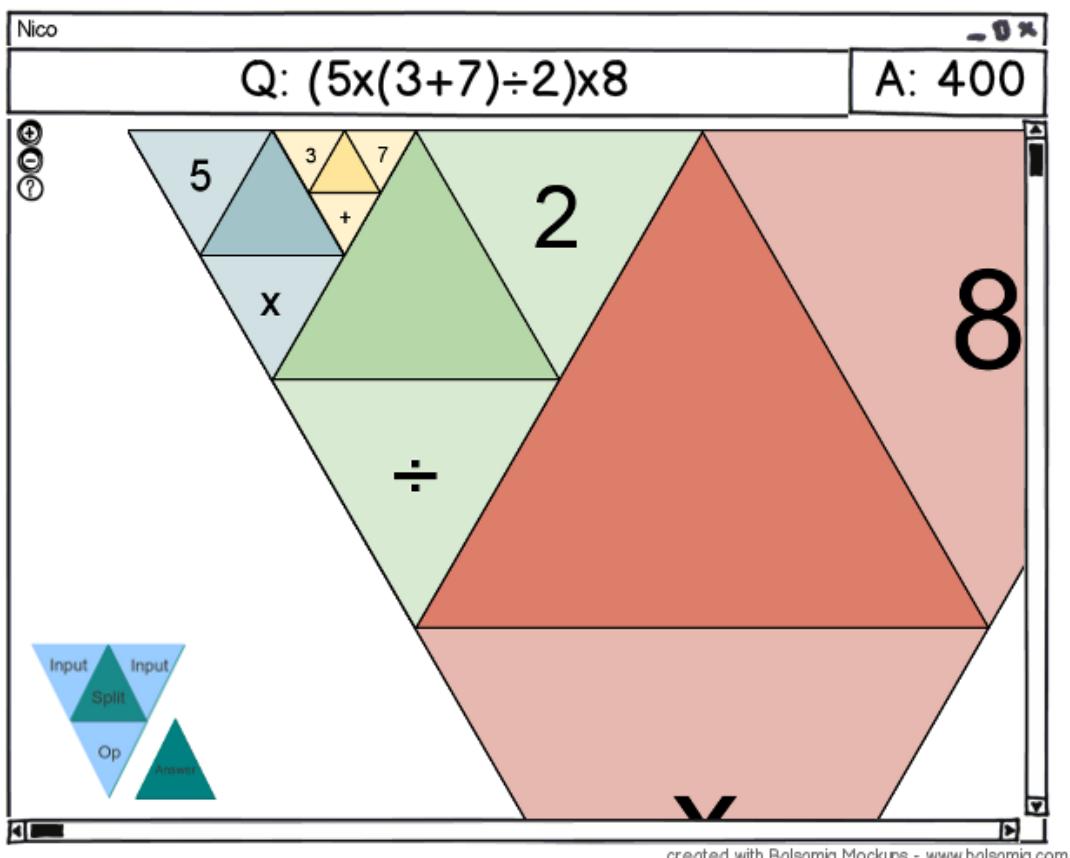


Figure 2.10: The prototype Sierpiński-triangle-based language.

This prototype uses a zooming instance of the Sierpiński triangle fractal as the basis of its notation. It is shown here manipulating a simple calculation.

The application consists of a single window containing two panels and a canvas area. The right-hand panel displays the current total of the calculation, and the other panel displays the question currently being answered. In the canvas area, there are buttons for zooming in and out, a help button and buttons controlling the composition of structures within the language. When a triangle has been selected by clicking on it, clicking the Op button presents the user with a choice of operations (namely +, -, × and ÷) to use in the bottom segment of the triangle.

Clicking either of the Input buttons allows the user to input a value into their respective sections, and clicking on the Split button creates a new triangle expression within the left-hand triangle.

This notational system is based around binary operations represented as the component triangles of a Sierpiński fractal, the logic behind this being that the fractal structure emphasises that each calculation is part of a larger whole. The notation is read by using the left and right sub-triangles as arguments (left first, right second) to the operation in the bottom triangle. The structure is colour-coded, such that each subexpression is a different colour to its precedent superexpression. This allows the user to abstract away subexpressions as units to be manipulated as any other input to an operation would be. The hidden dependencies of the handwritten method are greatly diminished in this manner, as viewing the internal structure of a calculation's subexpressions is simply a matter of zooming-in to the appropriate triangle. Viscosity is also considerably reduced, as making a change in one expression does not require the change to made anywhere else, as the results of that expression are passed to the rest of the calculation. This does not completely remove viscosity, as a misrepresented value still has to be replaced several times if it appears in several different expressions, but this is still an improvement over handwriting. Indeed, the fact that calculations are editable more than once, by virtue of using a computer rather than paper, makes this less viscous than pen and paper. The system is quite visible, but has a very low juxtaposability, as all calculations are restricted to their respective places in the triangle template.

This design was ultimately abandoned as it was deemed to be awkward for a number of reasons, not least because it was based solely around binary operations, meaning that to solve the simple question  $1+2+3+4$ , one has to either calculate  $((1+2)+3)+4$ , which is inefficient, or mentally calculate that  $1+2+3=6$ , and then calculate  $6+4$ , which is obviously unacceptable for a system that intends to help the user express themselves mathematically. The severe lack of juxtaposability is also a problem: as each expression must fit into the larger structure, it is not possible to reorder them at will, without fundamentally altering the calculation being performed. It also enforces a top-down approach to calculation: one is required by the system to begin with the outermost calculation and work inwards. There is no provision for putting a larger triangle around an existing calculation. In addition to this, a somewhat-informal survey of potential notations was conducted, asking approximately ten subjects to answer some sample questions in this notation and the circle-based notation (below). Subjects found this notation less clear, and were much better able to complete questions correctly using the circle notation.

### 2.2.1.2.3 Circle Metaphor

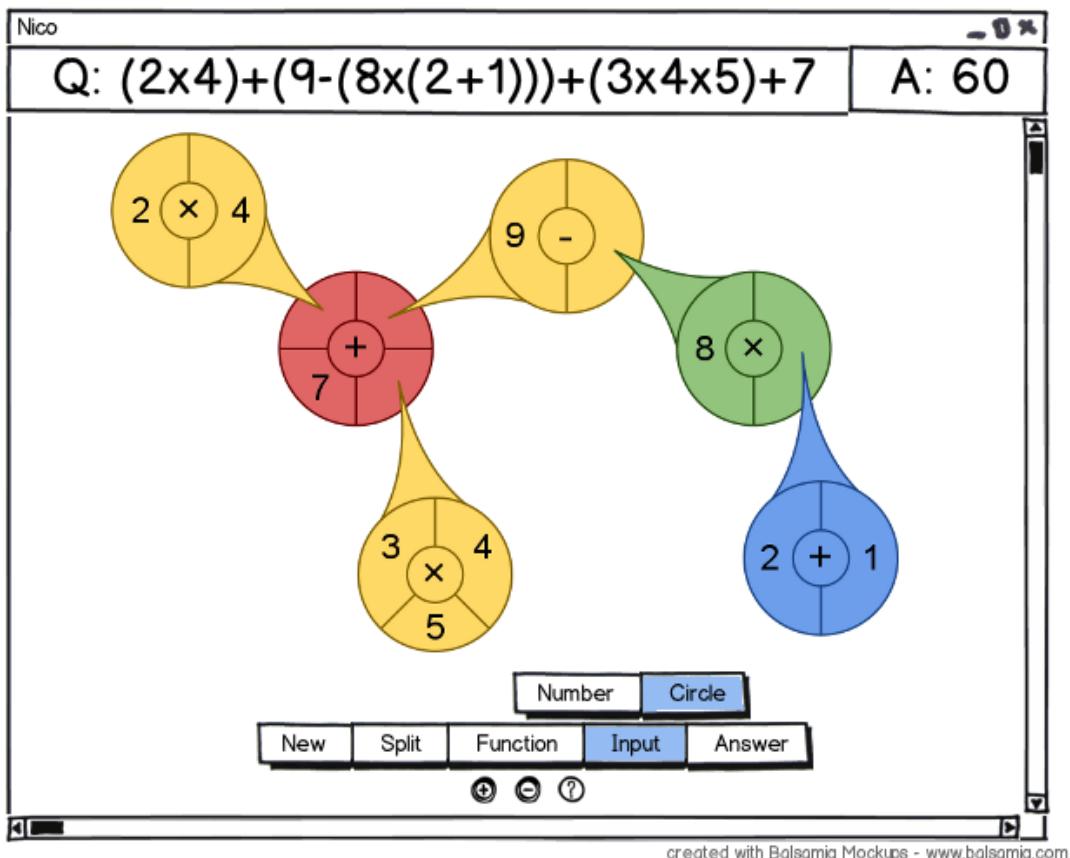


Figure 2.11: The prototype circle-based language.

This prototype uses a notation based around circles as individual units of calculation, linked in such a way as to indicate the flow of information. Although it is related to the flowgraph metaphor (above), it has a number of key differences.

The application comprises, similarly to the Sierpiński model above, a single window with two panels and a scrollable canvas area, with some buttons to control the input of data. The right-hand panel shows a running total of the user's calculation, and the other panel shows the question currently being answered. The canvas area has three buttons at the bottom: zoom in, zoom out and help. The larger buttons above these control the action that clicking the mouse on the canvas performs. When New is active, clicking on the canvas creates a new circle at that location. When Split is active, clicking on a circle increases the number of arguments it has (initially two, up to a maximum of eight). When Function is selected, clicking on a circle presents the user with a choice of +, -, × and ÷ to use as that circle's operation. When Input is

selected, as shown in *Fig. 2.4*, the user is presented with two further options, Number and Circle. When Number is active, clicking on a circle's argument allows the user to change the value of that argument. When Circle is active, the user is able to click and drag from one circle to an argument of another, indicating that they wish to use the results of one circle's calculation as the argument to another circle's calculation. Finally, the Answer button submits the current total as the answer to the current question.

This notational system uses linked circles representing individual expressions as its primary metaphor for constructing calculations. It is similar to the flowgraph metaphor presented above, except that instead of boxes for each argument and operation all linked together, single, linkable objects in this system are complete expressions, rather than components. Also, there is only one visible output, namely the overall result of the calculation as a whole. As for the circles themselves, they consist of an operation at the centre, orbited by arguments, which are evaluated clockwise from  $0^\circ$ . Each argument can either be a number or the tail of another circle, representing the result of the calculation that that circle stands for. The circles are colour-coded, with each 'layer' having its own colour. In this case, the root circle is red, the circles feeding directly into it are yellow, the circles feeding into those circles are green and the circles feeding into those are blue. It is possible to move circles around the canvas by clicking and dragging them to the desired location. Compared to the handwritten approach, this notational system is much more visible, and very much more juxtaposable, allowing the user to construct the diagram in a way that is comfortable for them, and allowing them to compare elements side-by-side if they need to. Unlike with the Sierpiński metaphor, the user is not restricted to one order or pattern of constructing a calculation: calculations can be performed bottom-up, top-down or a mixture of both, as is comfortable for the individual user. The system of tails feeding into other circles also makes the dataflow clear, eliminating many of the hidden dependencies of handwritten calculation. For similar reasons to the flowgraph metaphor, the circle notation also reduces viscosity, as changing a subexpression automatically alters all of its superexpressions. Finally, the system requires little premature commitment, as piece of the diagram can be created, deleted and swapped in and out at will, so it is ideal for exploration, allowing the user to try many ways of solving a problem within the application, rather than having to use a secondary notation or device to help work through a problem in a more familiar manner and then transcribing that solution into the new notation.

This design was the one that was ultimately chosen as the basis for the project as it neatly solved, or at least ameliorated, many of the problems listed with handwritten

arithmetic, without being too complex for the target audience (as with the flowgraph system), and without being too restrictive with regard to mathematical expression. In addition to this, the results of the survey mentioned above also indicated that people got on well with this notation; the test subjects found it considerably easier to use than the Sierpiński notation, and made fewer mistakes in handwriting this notation than the alternative.

## 2.3 Third-Party Tools

What follows is a list of the third-party tools that were used in the development of the project.

- Ubuntu Linux 10.04, Arch Linux 2010.05, Microsoft Windows 7
- Clojure 1.2.0
- Leiningen 1.6.1.1
- OpenJDK 6
- Seesaw 1.3.1-SNAPSHOT
- swank-clojure 1.3.4-SNAPSHOT
- GNU Emacs 23.1.1
- A modified version of Overtone's Emacs configuration [12], including:
  - SLIME/SWANK (revision as of 15/10/2009)
  - clojure-mode 1.11.5
  - undo-tree 0.3.3
- Git 1.7.0.4
- GitHub
- Balsamiq Mockups
- Google Docs
- R 2.15.0

## 2.4 Summary

In this chapter, the work done in preparation for beginning the main implementation of the project was detailed. A thorough requirements analysis has been conducted to outline the expected behaviour of the product, contrasting it with the existing system of handwritten calculations. In Sec. 2.2, prototype designs for a suitable graphical notation were introduced. Three in particular were explored in detail, before deciding on the final notation upon which to base the project. The next chapter contains a detailed analysis of the work completed to implement the project.

# **Chapter 3**

## **Implementation**

This chapter details the implementation of the project. The code itself totals approximately 1,500 lines of Clojure, and the resultant application, *Nico*, satisfies the requirements laid out in the project proposal, namely:-

“[to be] 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.”

In addition to this, an extension to the project was completed, in the form of a study that tested the software on real people.

### **3.1 System Architecture**

*Nico* can be considered as having three distinct layers of operation: the user interface, the interaction handler and the back-end. The user interface is what the user directly interacts with, with events in this layer being processed by the interaction handler, with the resultant data being passed to the back-end, which handles the mathematical and memory-management operations, to interpret. Data can also flow in the other direction, with output from the back-end being passed up to the interaction handler to cause effects in the user interface. An analysis of each layer in greater detail follows.

### 3.1.1 Back-End

The application back-end handles the mathematical aspect of the application; that is, it is able to interpret data gathered from the user's input and process it as a calculation. Each circles is represented by an associative map, containing information regarding its position on the canvas, the calculation it represents and its label. A Clojure agent stores a list of all of the circles currently in use. It was decided to use an agent after the original approach of defining circle maps as variables using a dedicated macro at runtime proved to be unviable; such variables were not accessible by the interaction handler. An agent solved this problem neatly, providing a useful means of storing data whilst making it available to be accessed and changed at will during the program's execution.

#### 3.1.1.1 Management of Mutable State

*Nico*'s back-end stores and manages circles and questions using just six Clojure agents. They are detailed below:

- An agent which contains the list of circles, as detailed above
- An agent which also contains the current question set, as a list of maps containing the question number and the S-expression that constitutes the question to be answered
- An agent which contains an integer counting how many circles have previously been created whilst answering this question
- An agent which tracks the current co-ordinates of the mouse cursor
- An agent which contains an integer indicating the number of the question currently being answered
- An agent which is used by the interaction handler to temporarily store information about circles that are currently being repositioned by the user

Again, agents were chosen for these structures as they needed to be continually changed whilst the program was running. Agents specifically, rather than the vars, refs and atoms that Clojure also provides for mutable state, were chosen to allow asynchronous updates, and are able to share information amongst several threads.  
[?]

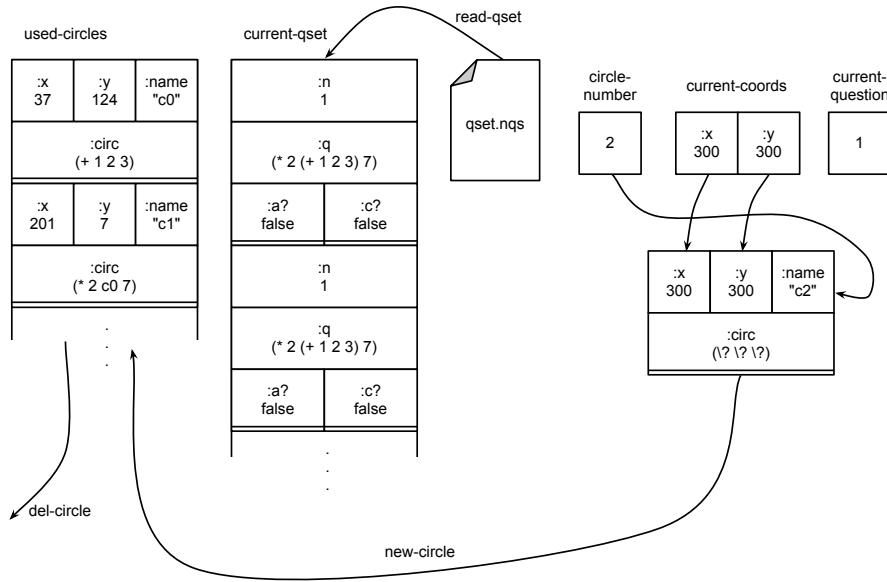


Figure 3.1: A model of *Nico*'s management of mutable state; the sixth agent is omitted as it is not utilised by the back-end.

### 3.1.1.2 Circle Management

*Nico* includes two main functions for the management of circles: creation and deletion. New circles are initialised to the following values:

```

1  {:x (- (@x @current-coords) 50)
2  :y (- (@y @current-coords) 50)
3  :name (str "c" @circle-number)
4  :circ (\? \? \?)}

```

Figure 3.2: Circles are initialised on the current location of the mouse cursor, generating a name "*cn*", where *n* is the number of circles that have previously been created, and with an initial expression of (\? \? \?).

This map is then stored in the list of circles. The counter for the number of circles is incremented by one, and the user-facing canvas area is cleared and re-rendered. This particular initial state for each circle was chosen so that the circle is centred upon the mouse cursor (rather than having its top-left corner appear at the location of the cursor if 50 had not been subtracted from the current co-ordinates), and so that its name is predictable, for use in other functions that refer to it; that is, the order in which the circles in the list were created can be inferred from their names if need be. The circle is also initialised to a placeholder expression, rather than one of

value 0, so that it does not interfere with the root-circle inference, which will be discussed later.

Deletion is a more complex function that removes a circle from the list. It first compares its argument to the head of the circle list. If it does not match its name or co-ordinate range (each circle is contained within an area of  $100\text{px}^2$ ), it recursively calls itself on the tail of the list, and appends the rejected circle to a new list. If there is a match, it again calls itself on the tail of the main circle list, but does not add the head to the new list. Thus, a list identical to the original list of circles is constructed, excluding only the matched circle.

This naïve implementation of deletion led to many problems: the function described above does not deal well with circles that are used as arguments to other circles. When such a circle is removed from the main list, there still remain references to it in the circles that were using it as part of a larger calculation, causing the application to throw `java.lang.NullPointerException`. As such, a ‘safe’ version of circle deletion was implemented. The safe deletion function works by first checking if any of the circles currently in the main list contain references to the circle being deleted. If it is found that there are circles that contain references to it, the function determines if that circle is a root circle (see below), and, if so, uses the previous method of deletion to remove the circle and replace it with a similar circle that uses placeholder ? values where there were references that would have been invalidated. If the circle is non-root, the deletion function recursively calls itself upon the circle, thus removing all circles that formed the original chain dependent upon the circle to be deleted. The circle can then safely be deleted using the previous method, and the intermediate circles are replaced.

This solved the problem of the leftover references to non-existent circles, but it has also given rise to a new problem: using this method, and in particular due to the stage in which the safe deletion function must call itself upon the other circles in the chain, deleting a circle that forms part of a chain deletes all of the links between the rest of the circles in the chain, from the circle being deleted down to the root circle. On reflection, a better way to implement safe deletion would have been to include either as part of the rendering loop or as part of the safe deletion function a function to inspect the main circle list, checking for any invalid references and replacing them with \?s, rather than doing it at delete-time.

### 3.1.1.3 Evaluation

There are two main stages required to evaluate the user's work: finding the root circle, into which all other circles feed, and evaluating a circle or collection of circles to give a valid, computable S-expression.

The root circle is the one circle in a finished diagram that is not used as an argument to any of the other circles. Therefore, it can be found by searching through the list of circles, comparing each one to the rest of the list and discarding it if there are references to it in other circles. As stated, in a **finished** diagram, there can only be one root circle, so this method is sufficient for use in marking answers. However, whilst the user is working, it is possible that they may construct two or more calculation structures in parallel, with intent to join them later. In this case, there can be multiple root circles. The method chosen to identify the root returns the first instance of a root circle that it comes across. It would have been possible to, for example, add an extra **Answer** panel for each root and maintain a total for each, but it was decided that this would be confusing for the user, who may not expect the structure of the application window to change. This would also require a labelling system for circles and structures, which was abandoned in the user interface design as it is an unfamiliar concept to non-programmers and those without algebra: most of the target audience.

Once the root has been found, the user's diagram can then be evaluated to give an S-expression. As each circle contains an S-expression for the calculation unit that it represents, all that needs to happen is for the references to other circles to be resolved, creating a nested S-expression that can simply have Clojure's `eval` called on it to return an answer. This was, in fact, a significant reason for choose Clojure, as a dialect of LISP, as the language in which to program *Nico*; as a homoiconic language, code to be evaluated later can easily be built up as a data structure, which can then be passed around, evaluated and otherwise utilised as is convenient. By creating one unified S-expression representing the entire diagram, not only can the value of the structure be calculated, but other operations, such as the selective highlighting of the question (discussed below), can be performed.

### 3.1.1.4 Question Management

*Nico* uses external files containing sets of questions to provide tasks for the user to complete. An NQS (*Nico Question Set*) file is simply a plain text file containing S-expressions corresponding to questions, separated by newlines. This particular

approach was chosen so that questions can easily be read in using functionality provided by Clojure’s standard library. Nico is then able to navigate between questions by setting the agent containing the current question to the value of the desired question in the list.

A future extension to this project could be to develop a partner application to *Nico* to aid tutors unfamiliar with LISP to develop their own question sets in a more ‘user-friendly’ manner, rather than by transcribing lists of problems into S-expressions.

### 3.1.2 Interaction Handler

The interaction handler is a collection of functions that form the ‘middle layer’ of the application: that is, the functions that arbitrate between the back-end and the user interface.

The rendering function is called every time the mouse is moved, or when what is displayed on the canvas needs to be updated. It paints a white rectangle over the canvas area, draws a bin icon in the corner (for circle, deletion, see information on the control scheme below) and iterates across the circle list, drawing a circle to match each element. References to other circles are visualised as lines extending from the source circle to the appropriate argument slot on the recipient circle. By re-rendering every time the mouse is moved, rather than continuously, the application does not have to render when the canvas is not being changed, as every action in the control scheme requires some use of the mouse.

#### 3.1.2.1 GUI Libraries

The original intention of the project was to develop *Nico* using the JavaFX library. However, a number of difficulties were encountered with this: firstly, JavaFX version 2.0 was not available for Linux, the primary development environment, at the time of beginning the project. Installation was attempted on a Windows machine, but caused a number of problems. In the interest of making progress, it was decided to proceed using the Eclipse Foundation’s SWT toolkit and Szymon Witamborski’s corresponding Clojure bindings, *GUI FTW!* [?]. During the development of the user interface, many problems were encountered with SWT and *GUI FTW!*, particularly regarding drawing arbitrary shapes on the canvas area and extracting user input from dialogue boxes. To keep the project on-schedule, the application was migrated to Java Swing, using David Ray’s *Seesaw* library to provide bindings for Clojure [?].

### 3.1.2.2 Main Window

At the heart of the interaction handler is the structure defining the main window, which comprises the majority of the user interface. It controls what is to be displayed and the actions to be performed under certain conditions. It listens for certain events (i.e. user input), and prompts the user for input when required.

Of particular note are the various listeners embedded into the main window structure that monitor the canvas for mouse-based events. While the application is running, it listens for five mouse-based events: motion, dragging, a button press, a button release and a button click.<sup>1</sup> When motion is detected, the canvas is cleared and re-rendered, as mentioned above. Motion also triggers a check to determine whether or not the mouse is over a circle; if it is, a highlighting function is called. This draws a ring around the circle currently being moused over, and also performs a string comparison: using the same function written to convert the questions into traditional mathematics for display in the information panel, the circle's S-expression is converted and compared to the question string to see if it appears as a substring. If so, the appropriate section of the question string is also highlighted, to help reinforce the notion of the circle notation as a metaphor for calculation. It was decided to implement this a string search, rather than by using the actual S-expressions for the question and for the circle's calculation as although this would be a more efficient method to determine whether or not a circle corresponds to a part of the question, it is less efficient to do so and then calculate where the matched S-expression appears, rather than just matching a potential substring against a longer string.

---

<sup>1</sup>It should be noted that a mouse click is defined by Java as a button press followed by a button release, and that, upon a click, it issues a press event, followed by a release event, followed by a click event. [?]

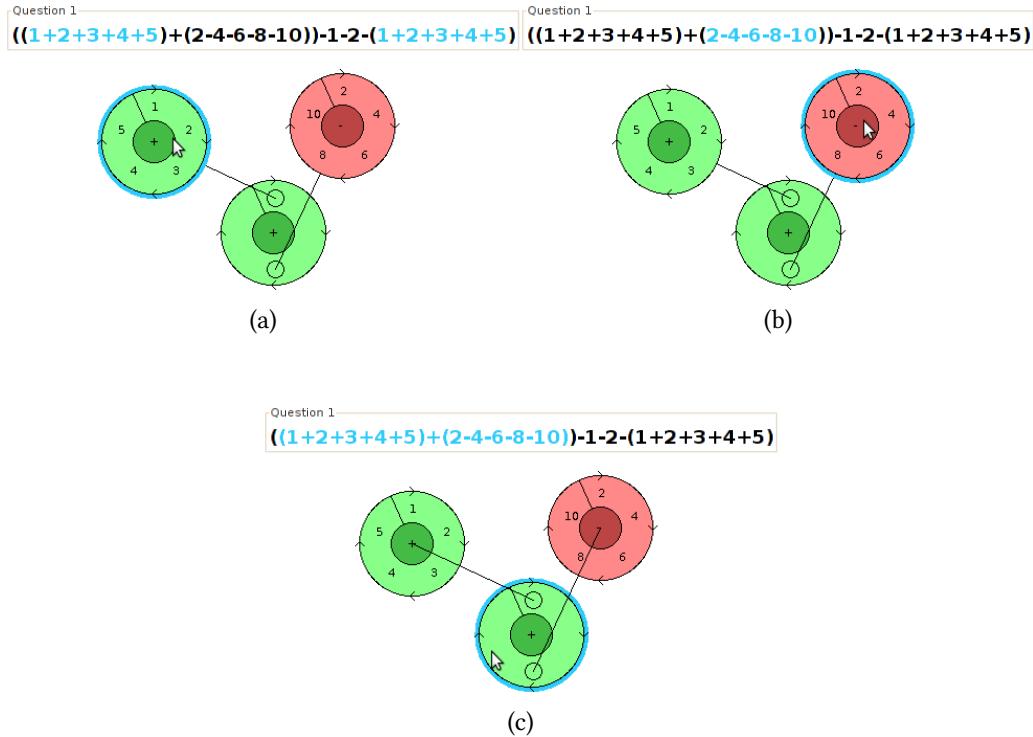


Figure 3.3: Three instances of the question text being highlighted as a corresponding circle is moused-over.

### 3.1.2.3 Control Scheme

Upon the detection of a mouse click, the software must determine the exact context in which the mouse was clicked, as this constitutes much of the application's control scheme. The mouse's current co-ordinates on the canvas, as well as information regarding which mouse button was pressed and any modifiers that were active, are used to determine the correct action to take, as illustrated in *Fig. x.y*.

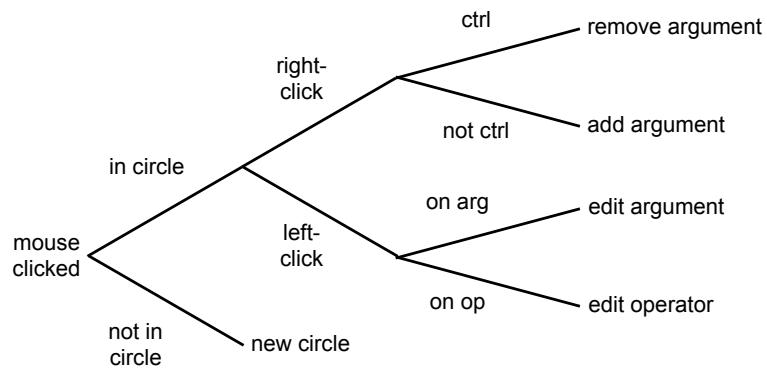


Figure 3.4: Decision tree to determine the appropriate response to a mouse click event.

A more in-depth analysis of the control scheme follows.

- Mouse not in circle
  - A blank circle is created, centred on the current location of the mouse cursor. It has two arguments, ? and ?, and its operator is ?.
- Mouse in circle
  - Left-click
    - \* Mouse on argument
      - A small blue circle is drawn around the argument and a dialogue box is launched, containing a spinner with which to set the new value of the selected argument.
    - \* Mouse on operator
      - A small blue circle is drawn around the argument and a dialogue box is launched, containing four radio buttons corresponding to the four arithmetic operations, with which to set the new value of the operator.
  - Right-click
    - \* Ctrl key pressed
      - The final argument is removed from the circle, down to a minimum of two arguments. An alert box warns the user if they try to remove arguments when there are only two remaining.

\* Ctrl key not pressed

- A placeholder argument is added to the circle, up to a maximum of eight arguments. An alert box warns the user if they try to add more than the maximum number of arguments.

*Nico* also includes some drag-and-drop functionality in its control scheme. When the application detects that a mouse press has occurred, if the event was triggered whilst the mouse cursor was over a circle, that circle's map is stored, along with the current system time in milliseconds and the mouse button that was pressed, in one of the agents. A drag event then uses the information stored in the agent to perform one of two operations upon the circle. If the left mouse button is held whilst dragging, the circle's positioning information is updated to the current co-ordinates of the mouse cursor. If the right mouse button is held whilst dragging, a line is drawn from the circle to the current location of the mouse cursor, indicating that two circles are to be joined. When the mouse passes over the argument slot of another circle, that line is fixed, and the target circle is updated such that the argument now has the value of a reference to the source circle. When the mouse is released, the information in the agent is removed. The drag-and-drop functionality was implemented thus as a drag-and-drop action constitutes a press event, followed by a drag event, followed by a release event. By storing information about the circle being dragged in an agent upon detecting a mouse press, the function called when dragging is detected can use and modify this information easily, without encountering the problems of defining new variables at runtime, as mentioned in the discussion of the application back-end. This does not impact upon other mouse events involving presses and releases (e.g. clicking), as the agent is reinitialised upon every mouse release, regardless of whether or not the drag-and-drop agent was being used.

## 3.2 User Interface

The user interface comprises the parts of the application which are user-facing; that is, they are that which the user directly interacts with. The development of *Nico*'s user interface was key to the success of the overall project, being primarily an experiment in human-computer interaction.

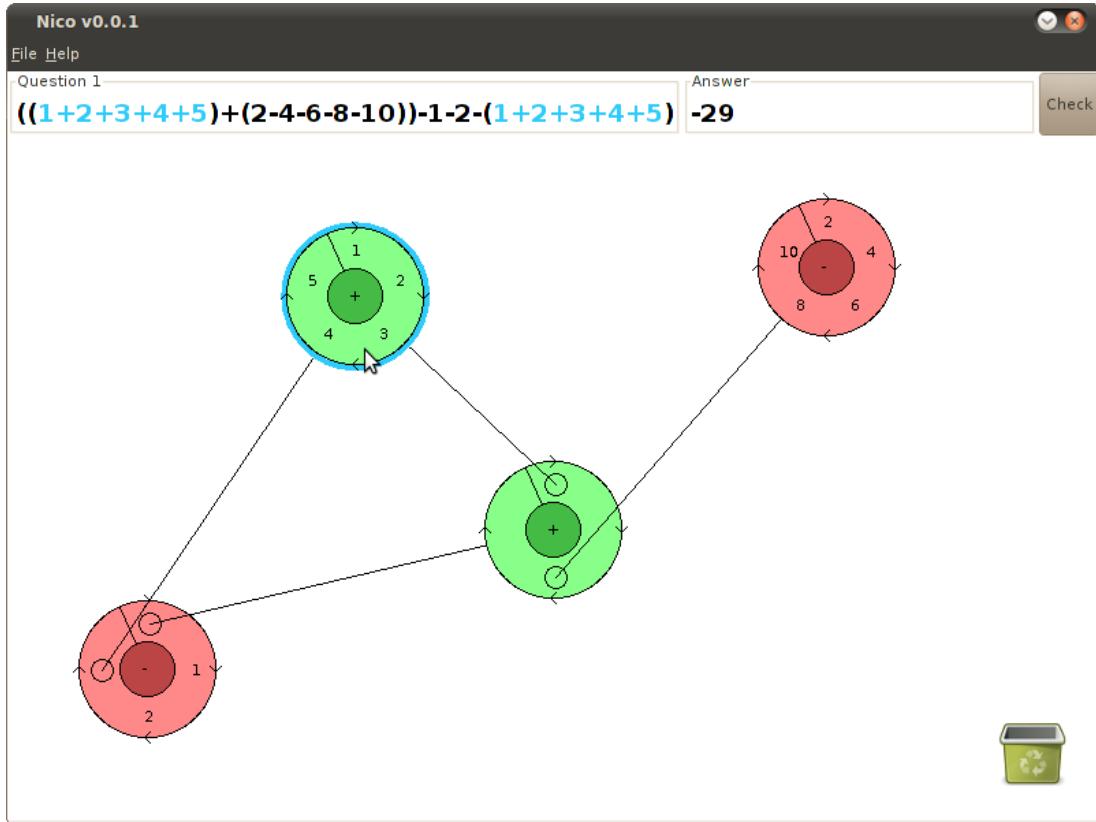


Figure 3.5: A screenshot of a *Nico* session.

The application itself was kept fairly minimal: the aim was to emphasise the notation as the most important aspect of the software. As such, the main application window, as shown above in *Fig. x.y*, is mostly dedicated to the canvas area in which the user answers the question. The main application window is split up into two main sections, the information panels that display the question being answered and the current total of the user’s calculation, as well as the Check button to submit the current total as the answer, and the large canvas area in which the user is able to construct diagrams with which to answer the question.

*Nico* uses a predominantly mouse-based control system for the construction of calculations in the application’s notation. Left-clicking on a blank patch of canvas creates a new circle, initialised with two arguments, ? and ?, and with operator ? – this approach was chosen, rather than providing the user with a null circle such as  $(0+0)$ , to make it clear to the user that the new circle required attention, and so that it wouldn’t interfere with the evaluation of the diagram; a new circle on its own could be found to be a root circle, leading to the possibility that the user’s total could suddenly drop to 0, which would be confusing. Right-clicking on a circle

increments its number of arguments by one, up to a maximum of eight. Holding the Ctrl key and right-clicking on a circle decrements its number of arguments by one, down to a minimum of two. The user is warned by a popup if the exceed either of these boundaries. The limit of two to eight arguments was decided upon as fewer than two arguments is an unfamiliar and not often useful concept from the perspective of the target audience. Although Clojure is able to evaluate, for example, `(+ 1)` (returning 1), this is liable to lead to confusion on the part of the user. The upper limit of eight arguments was set to prevent circles from becoming overcrowded and harder to read. Circles can be moved around the canvas by left-clicking and dragging the circle to the desired location. If a circle is dragged to the bin icon in the bottom left-hand corner of the screen, then it is removed. Originally, deletion was mapped to a mouse control, but in the interest of a simpler control scheme and with a limited number of mouse buttons, it was decided that an icon should be used instead. This also has the advantage of demonstrating to the new user that work can be easily undone, encouraging exploration. Left-clicking on a circle's operator brings up a dialogue box with a selection of operators in it, with a radio button for each (see Fig. x.y). Similarly, left-clicking on a circle's argument brings up a dialogue box containing a spinner, which can be used to set a new value for that argument, between -10 and 10. The spinner is limited to this particular range as this prevents the user from effectively using the software as a simple calculator. The limited range of numbers available means that the user must still consider strategies such as partitioning or long multiplication, focussing on how to solve the problem, rather than simply transcribing the question into one circle that will complete the calculation for them. The user is not required to calculate manually expressions using such small numbers, as it is assumed that the target audience are familiar with simple arithmetic. To use one circle as an argument to another, right-clicking and dragging allows the user to drag a line ending in a circle to the desired slot on the recipient circle.

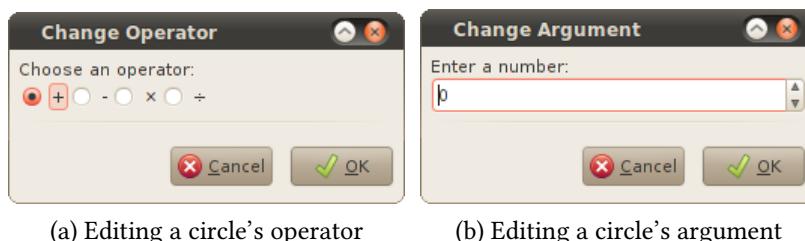


Figure 3.6: The dialogue boxes used to modify existing circles in the application.

*Nico* also uses colour-coding in various areas of the application, to draw attention to important elements of the application. Colour-coding has been shown to facilitate more effective learning. Indeed, Lamberski and Dwyer note that

“[...] younger learners, for whom color in passive materials has been found to facilitate performance in less complex concept attainment tasks because of color’s attention of motivation elements. However, when younger learners have been given self-paced instructional materials containing color-coded reading and phonic concepts, similar positive results have been attained.” [7]

Circles are coloured according to their operators, clearly separating different kinds of operations and making the notation more readable. An alternative method of colour-coding by the level of nesting was considered, but rejected on the grounds that it could become confusing if a circle were reused (see Fig. x.y).

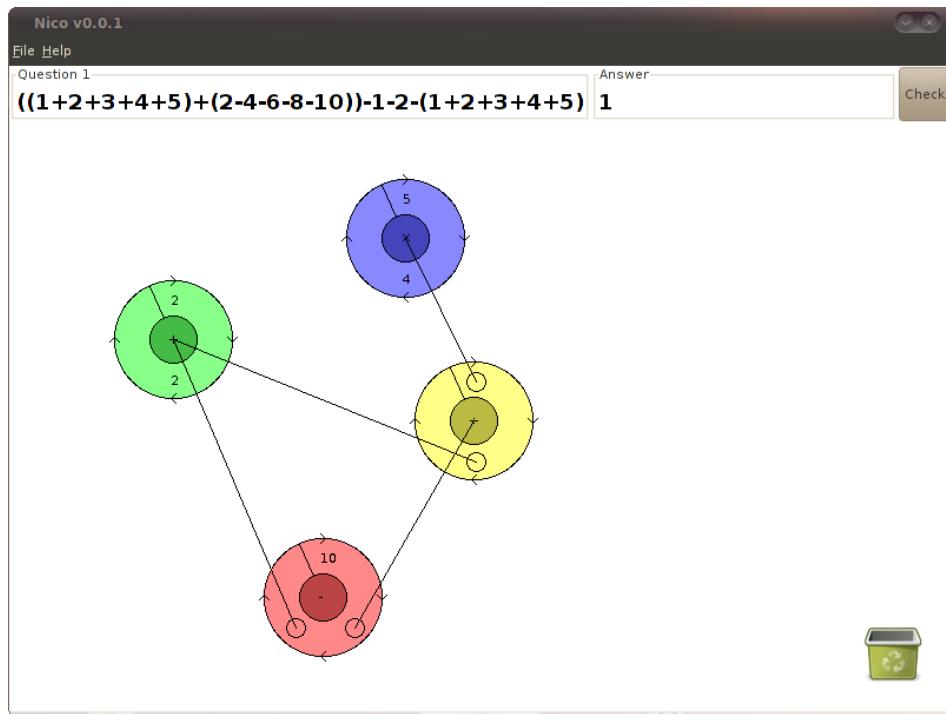
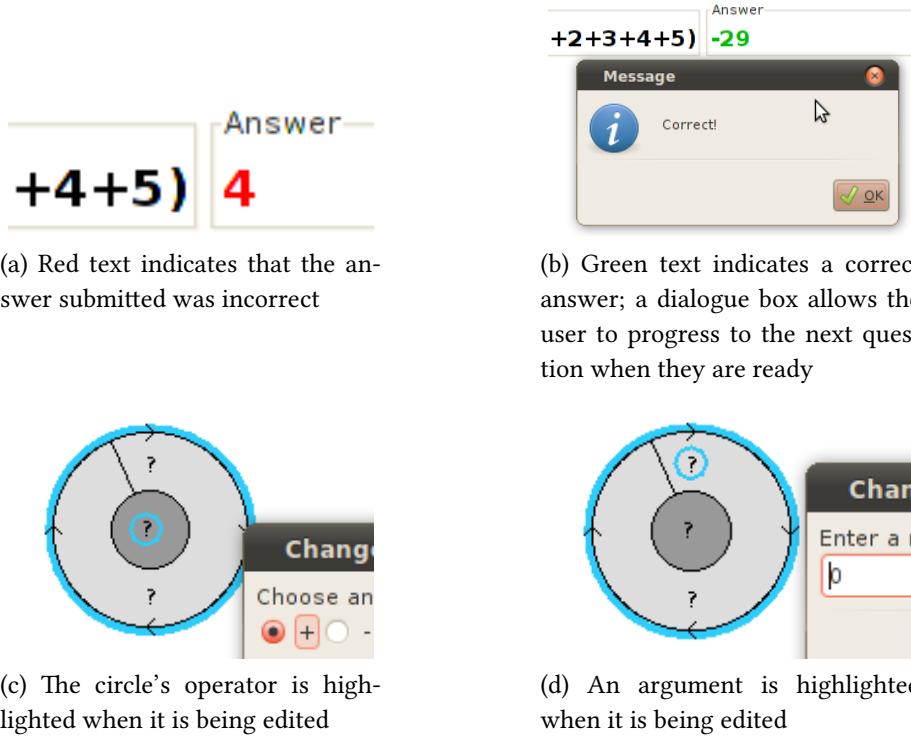


Figure 3.7: An example of where nesting-based colour-coding would be confusing. The circle containing  $(2+2)$  can be considered to be both one and two circles removed from the root circle.

Colour-coding is also used in the information panels: positioning the mouse cursor over a circle highlights the part of the question that it is likely to represent in blue, and also highlights the circle with a blue outline. When an answer is submitted, the text in the answer panel turns red if the answer was incorrect, or green if the answer was correct, providing an unintrusive form of feedback, allowing the user to continue uninterrupted (i.e. no input is required) if their answer is wrong.

Figure 3.8: Examples of colour-coding in *Nico*.

During its development, *Nico*'s user interface went through a number of revisions. Details of how the UI design developed over time follow.

### 3.2.0.4 First Revision

Initially, the application was intended to use more buttons (see the prototype in Fig. x.y, above), having more controls immediately visible to the user. In the first usable version of the interface, there were a number of buttons that appeared down the side of the application. Each button (New, Edit, Delete, etc.) would cause a dialogue box to appear to the user, which required a circle to be specified by name (at this point in the development of the project, it was required that the user name each of their circles) before any operations could be performed upon it. The operations themselves were specified by typing the circle's name into a text field, followed, in the case of the creation and editing of circles, an S-expression specifying the mathematical expression that the circle was to represent. Circles were all the same colour, and the text-highlighting and colour-coding operations had not yet been implemented. It was also the case that the links between circles did not end in

argument slots, but rather fed into the centre of the target circle. This design was abandoned as the separation between the user and the diagram itself was too great; creating circles by specifying options in many only slightly different dialogue boxes and watching the results appear on the canvas did not feel like a natural way to interact with one's calculation, and hence would have been inappropriate for the target audience. Indeed, forcing the user to commit to a design before even seeing it entailed much *premature commitment*. Requiring the user to use LISP S-expression syntax was also far from ideal, as the user then had to learn this syntax in order to use the program. The intention of the project is not to require the user to spend a significant amount of time learning how to use the system before it becomes a useful tool to them. Quite apart from this, a user familiar with LISP is also unlikely to need the aid of the circle metaphor to visualise expressions.

### 3.2.0.5 Later Revisions

Later revisions of the software removed the buttons and implemented a system of context menus with which to construct answers. Right-clicking on a blank patch of the canvas gave the user the option to create a new circle, and right-clicking on an existing circle presented the user with the option to either edit or delete that circle. Choosing to either create or edit a circle displayed a dialogue box (Fig. x.y), with many options for the user to choose from: the name and operator of a circle could be set, with up to eight arguments could be enabled, with a choice of either a number (set by a slider) or another circle (chosen using a list of radio buttons that was generated upon opening the dialogue box) as the value of that argument. Answer-checking was also made automatic: as soon as the total reaches a value equal to that of the answer to the question, the user is immediately told so, and moved on to the next question. The circles at this point were still one colour only, and there was no expression-highlighting. The colour-coded totals had been implemented by this point. This design was ultimately abandoned as, again, it separated the user from their own work too much. Although this design was an improvement, allowing the user to interact directly with the canvas to perform operations upon circles that were able to specified by pointing with the mouse, rather than by typing out a name, the grand unified dialogue box still required the user to perform operations without it being entirely obvious what the consequences would be, thus requiring an unnecessary amount of *premature commitment* from the user. In addition to this, the act of visualising the circle to be created whilst setting the many options available in the dialogue box constituted an unnecessary *hard mental operation*. The automatic answer-checking was also a problem, as it did

not give the user a chance to review their answer before proceeding. Indeed, it did not give the user a chance to learn from their mistakes, say for example if they had accidentally reached the right answer as part of a larger, incorrect calculation that they had previously intended to make.

### 3.2.0.6 Current Revision

The final, current revision of the software reintroduces the Check button to address the issues with the automatic answer-checking in the previous version of the application. It also introduces the control scheme detailed above, allowing the user to feel more like they are directly manipulating their work in progress, with each action having clear consequences. A colour scheme for the circles was implemented, to make the notation easier to read, and the expression-highlighting functionality was also introduced, along with the other means of highlighting, as shown above. Finally, the bin icon was introduced after deciding that a control for deletion was not an ideal approach, as detailed above.

## OLD VERSIONS OF THE APP LOL

Figure 3.9: Nico’s user interface went through a number of revisions before reaching its current state.

### 3.2.1 Notation

## OLD VERSIONS OF THE NOTATION LOL

Figure 3.10: The notation used also went through a number of revisions before reaching its current state.

The notation itself was also revised several times during the development of the application. The original vision of the notation can be seen in *Fig. x.y*. The first implementation of the notation consisted of uniformly-coloured circles connected by lines feeding into the centre of the recipient circle. There were no placeholder values; these were unnecessary as the system of dialogue boxes meant that at no point could a circle have any aspect of itself in an undefined state. There was also no indication of argument ordering, which is acceptable when the operator is commutative (i.e. addition or multiplication), but for operations where this is not the case, this can quickly become confusing. Although the ordering and placement of arguments was defined, with no indication of this, the user is required to figure

this out through experience, which is counter to the intentions of the software. As mentioned above, it should not be required that the user spend a significant amount of time learning how to use the system, especially due to deficiencies in the notation. Another, greater, problem related to argument ordering was that of the circles that linked to other circles. With all input circles pointing to the operator at the centre of their target circle, it was not at all clear in which order the input circles evaluated, neither relative to each other nor to the other arguments in the circle.

The notation changed somewhat to address the problems listed above. In its current revision, the notation has had a number of features added. Firstly, the circles are now colour-coded by operator, making it easier for the user to distinguish between different kinds of circles. The circles themselves have also been embellished slightly with the addition of a line denoting the star of the expression, just to the side of the first argument at the top of the circle. Small arrows have also been added to the edge of the circles to indicate in which direction the arguments should be read. Finally, the links between circles have been changed such that they now occupy an argument slot that could otherwise be taken by a number, with a circle on the end of the link line to clearly show where the link terminates.

### 3.3 Summary

In this chapter, the implementation of the project was discussed in some detail, as well as the challenges that were faced during this development phase. The software itself fulfils and exceeds the criteria for success outlined in the project proposal, being a complete and usable system that improves greatly, in terms of cognitive dimensions, upon the traditional method of handwritten arithmetic. Third-party libraries, in particular Seesaw, were used where appropriate. The application divides into three main sections: the back-end, handling mathematical operations and memory management, the interaction handler, helping the user interface to interface with the back-end, and the user interface itself. The user interface was revised many times to correct deficiencies in the design that would have led to a poor user experience, and the back-end and interaction handler were also both restructured as new demands upon the software became apparent.

In the next chapter, the software will be evaluated, discussing test results and the user study that was conducted as an extension to the project.



# **Chapter 4**

## **Evaluation**

This chapter concerns the testing and evaluation of *Nico*; carried out to determine the quality and utility of the software.

### **4.1 Testing**

Throughout the development of the software, I performed unit tests to ensure the integrity of my code. My development environment of choice, GNU Emacs using SLIME/SWANK and clojure-mode, includes functionality such that commented-out lines of code can be evaluated at will. As such, it was possible for me to write unit tests into the main body of my code, evaluating them whenever a change may have affected how a function behaved. In the interests of tidiness and code readability, many tests were removed as they became unnecessary later on in the development in the project, but *Fig. x.y* shows one of the test blocks that were left in the most recent revision of the source code.

```

1  (defn detect-subs
2    "Returns a list of maps containing start and end indices showing where a substring c appears in the superstring q."
3    [c q]
4    (loop [s (cond (= (subs q
5      (- (count q) (count c)))
6      (count q)) c) (split q
7        (re-pattern
8          (escape
9            c
10           {\+ "\\\\"}
11           \C "\\\\""
12           \O "\\\\"})))
13      :else (butlast (split q
14        (re-pattern
15          (escape
16            c
17           {\+ "\\\\"}
18           \C "\\\\""
19           \O "\\\\"}))))
20      i 0
21      l '[])
22    (cond (empty? s) (reverse l)
23      (= c (subs q
24        (+ i (count (first s)))
25        (+ i (count (first s)) (count c)))) (recur (rest s)
26          (inc i)
27          (cons {:s (+ i (count (first s)))
28            :e (+ i (count (first s)) (count c))})
29          l))
30      :else (recur s (inc i) l))))
31
32  ;;= (def tq (lisp-to-maths (eval (:q (first @current-qset)))))
33  ;;= (subs tq (:s (nth (detect-subs "1+2" tq) 0)) (:e (nth (detect-subs "1+2" tq) 0)))
34  ;;= (subs tq (:s (nth (detect-subs "1+2" tq) 1)) (:e (nth (detect-subs "1+2" tq) 1)))
35  ;;= (subs tq (:s (nth (detect-subs "1+2" tq) 2)) (:e (nth (detect-subs "1+2" tq) 2)))
36  ;;= (let [s "roflomaomglolwtf" i (detect-subs "lol" s)] (subs s (:s (nth i 0)) (:e (nth i 0))))
37  ;;= (let [s "roflomaomglolwtf" i (detect-subs "lol" s)] (subs s (:s (nth i 1)) (:e (nth i 1))))
38  ;;= (let [s "roflomaomglolwtf" i (detect-subs "lol" s)] (subs s (:s (nth i 2)) (:e (nth i 2))))
39
40  ;;= (let [s "0123456789" i (detect-subs "456" s)] (subs s (:s (first i)) (:e (first i))))
41  ;;= (let [s "roflomaomglolwtf" i (detect-subs "lol" s)] (subs s (:s (first i)) (:e (first i))))
42  ;;= (let [s "fagaha" i (detect-subs "a" s)] (subs s (:s (first i)) (:e (first i))))
43  ;;= (let [s "lalala" i (detect-subs "a" s)] (subs s (:s (first i)) (:e (first i))))
44  ;;= (detect-subs (lisp-to-maths (eval-circle (find-circle "c0")))) (lisp-to-maths (eval (:q (first @current-qset)))))
45  ;;= (let [s "dcbabcd" i (detect-subs "a" s)] (subs s (:s (first i)) (:e (first i))))
46  ;;= (let [s "dcbabcd" i (detect-subs "a" s)] (subs s (:s (nth i 1)) (:e (nth i 1))))
47

```

Figure 4.1: An example of a block of inline tests, taken from `core.clj`, showing some tests to be run to test the implementation of `detect-subs`.

## 4.2 UI Evaluation

### 4.2.1 Notation Evaluation

## 4.3 User Study

To assess the utility of the software relative to handwritten arithmetic, a user study was conducted, allowing real users to get to grips with the system, and provide feedback on it. To this end, users were given a tutorial video and a questionnaire (see *Appendix A* for the full questionnaire) to complete whilst using the software.

Originally, it was the intention of this project to perform a user study upon a ‘test class’ of Year 5 pupils at an actual school, but the ethical complications involved with working with vulnerable individuals were so great as to make such a study impractical, given the time constraints of this dissertation; in addition to obtaining the approval of the Ethics Committee and the permission of a school to conduct the study, consent forms would have to be distributed to the test class, taken home, signed by the pupils’ guardians, returned to the school and returned to myself. It was, therefore, decided that the software should be tested upon a group of students not currently reading a ‘mathematical’ subject, such as the Natural Sciences Tripos, the Mathematical Tripos or the Computer Science Tripos, which necessitate a high level of mathematical aptitude and experience.

Test subjects were first asked to sign a statement of informed consent, giving their consent to participate in the study, and were then asked if they had previously used *Nico*, and to indicate their confidence in their ability to solve simple mathematical problems on a five-point Likert item. Users were then divided into two groups. The first watched the tutorial video and were allowed to explore the application in a ‘sandbox’ environment for five minutes. They then completed one question set using *Nico*, and answered the same questions by hand. The second group completed the questions by hand before watching the video and using the software. The time taken to answer each question was recorded, with *Nico* including functionality to query the system time as the user progressed and the handwritten questions being timed using a stopwatch. Both groups were then asked to complete a series of questions to provide feedback on the software, taking questions from Blackwell and Green’s sample questionnaire. [4]

### 4.3.1 Pilot Study

A pilot study was conducted with one test subject, to assess the feasibility of the experiment that had been designed. The outcome of this study was to make changes to several areas of the test, and a small change to the control scheme of the application itself. Firstly, the pilot test subject noted that the original version of the tutorial video for *Nico* was hard to follow; it used subtitles to convey information, which the test subject felt were hard to read and understand in the time given. The subject also noted that some of the questions in the feedback section were confusing or seemingly irrelevant to this particular application, and was unsure of how to answer them. The final piece of feedback that the subject provided was that the control scheme of the application seemed counterintuitively focussed upon the right

mouse button, where it would have felt more natural to the user to have used the left.

In the interests of improving the study, I acted upon the subject's advice. The video was reproduced using a voiceover rather than subtitles, making it easier to follow. Some of the less relevant questions were removed from the feedback section, leaving those more applicable to *Nico*. Finally, the control scheme was revised, assigning the left mouse button to primary functions such as the creation and editing of circles, and the right to functions such as changing the number of arguments.

### 4.3.2 Results

The results of the tests are shown in *Table x.y*, listing the times taken for each subject to answer the questions both by hand and using *Nico*.

Question	Group 1							
	Subject 1		Subject 2		Subject 3		Subject 4	
	Hand	Nico	Hand	Nico	Hand	Nico	Hand	Nico
Q1	4600	19547	3100	34094	1800	24657	1300	38891
Q2	8800	23313	7000	24329	6300	17437	5400	12031
Q3	20600	53500	13600	36360	10100	40798	10700	31813
Q4	36200	54375	26400	60156	14200	79704	23800	50094
Q5	61900	103344	39600	79844	26500	103221	33400	61064
Q6	213400	224406	190300	215376	84900	172940	181700	154691
Q7	235200	165813	1996000	103376	909000	48922	194200	52595
Q8	259500	306625	282500	158094	147200	138581	256700	155347
Q9	304400	230953	313500	123626	159800	183722	271700	196144
Q10	416500	307734	357700	213642	197100	245161	322200	235364

Question	Group 2							
	Subject 1		Subject 2		Subject 3		Subject 4	
	Hand	Nico	Hand	Nico	Hand	Nico	Hand	Nico
Q1	1200	22299	2100	23516	2600	16142	2600	21985
Q2	2200	24530	5600	19015	7200	16875	7200	23422
Q3	17300	47620	7600	29735	18400	27923	15900	48470
Q4	23000	50704	20700	43094	25300	37516	38600	174160
Q5	46600	134828	33300	70313	35500	58954	72400	98846
Q6	199900	629835	136400	164891	99000	113659	266600	252708
Q7	202400	80277	142200	48890	1106000	52954	272900	72767
Q8	295900	248736	182000	107375	184700	78001	348900	257224
Q9	312900	152185	204100	118594	200500	67485	384500	144753
Q10	357300	238300	261600	203110	237500	324581	472900	270177

Table 4.1: Table showing the time in milliseconds taken by each test subject to complete each question, by hand and using the software.

This data can be visualised using a series of plots, as follows.

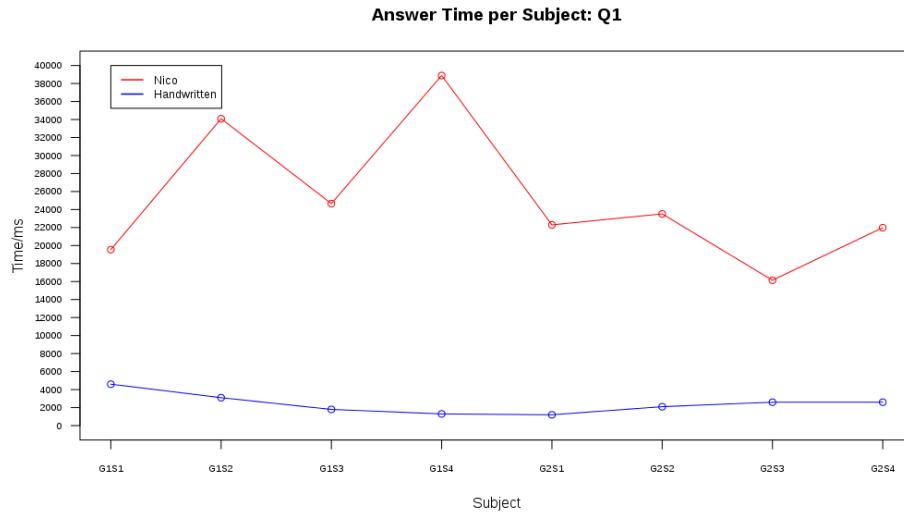


Figure 4.2: Time taken in milliseconds per subject to answer Question 1.

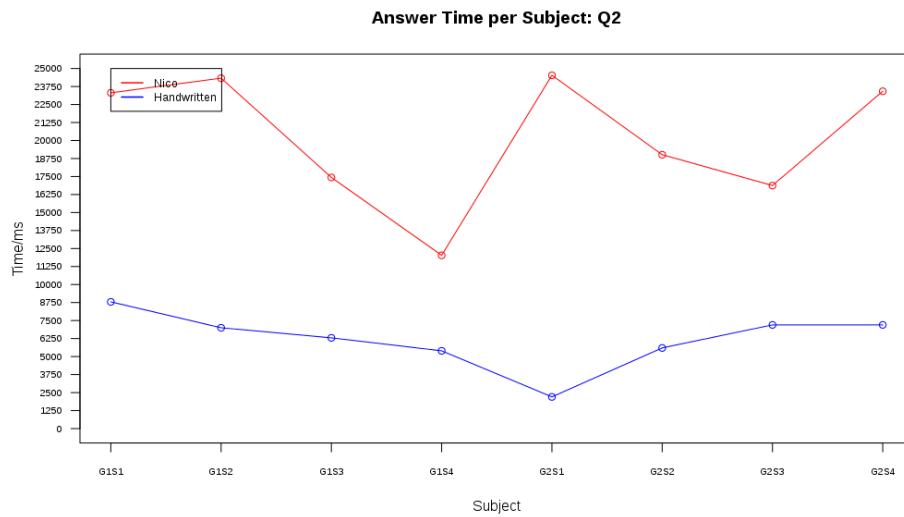


Figure 4.3: Time taken in milliseconds per subject to answer Question 2.

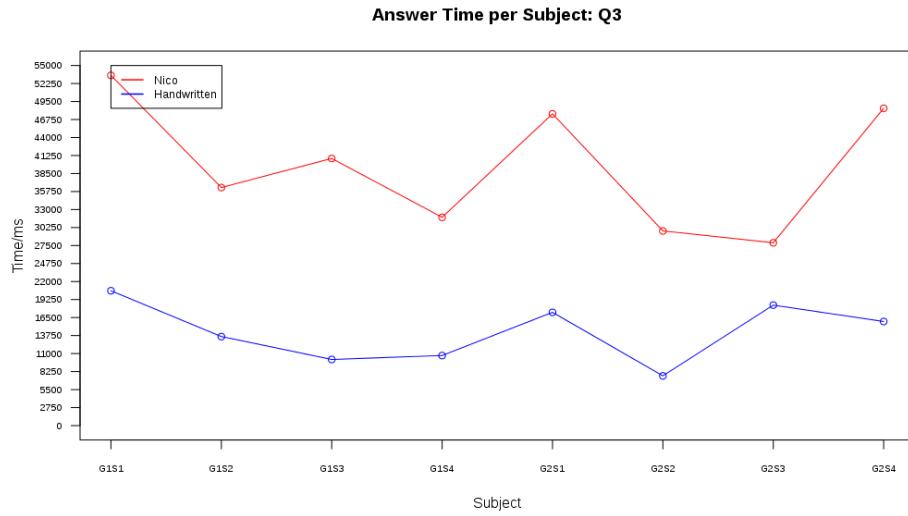


Figure 4.4: Time taken in milliseconds per subject to answer Question 3.

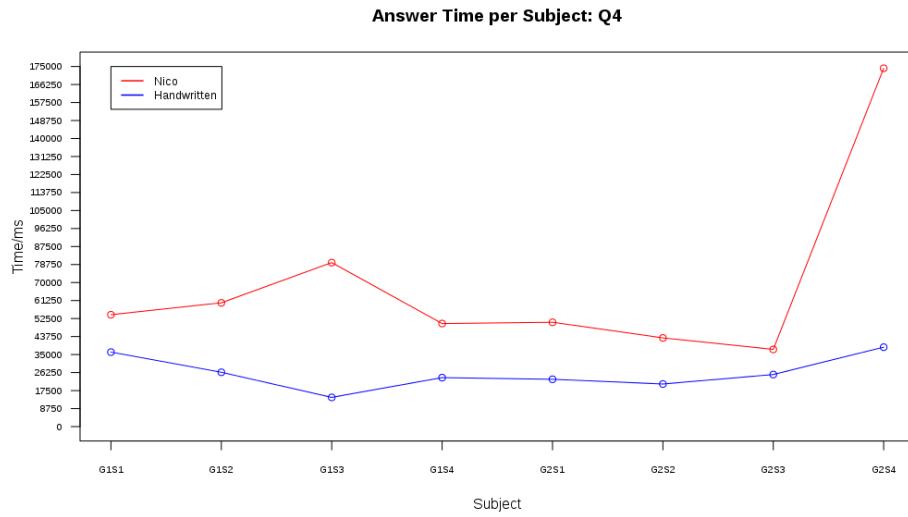


Figure 4.5: Time taken in milliseconds per subject to answer Question 4.

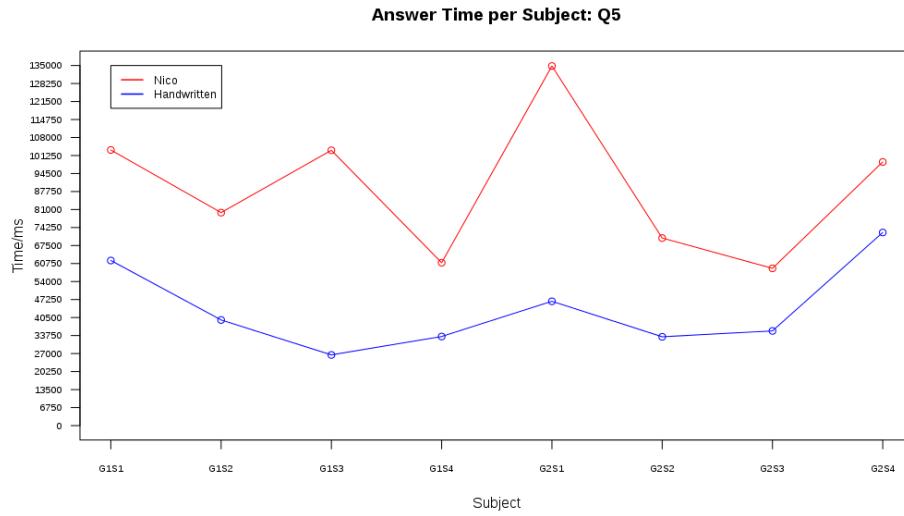


Figure 4.6: Time taken in milliseconds per subject to answer Question 5.

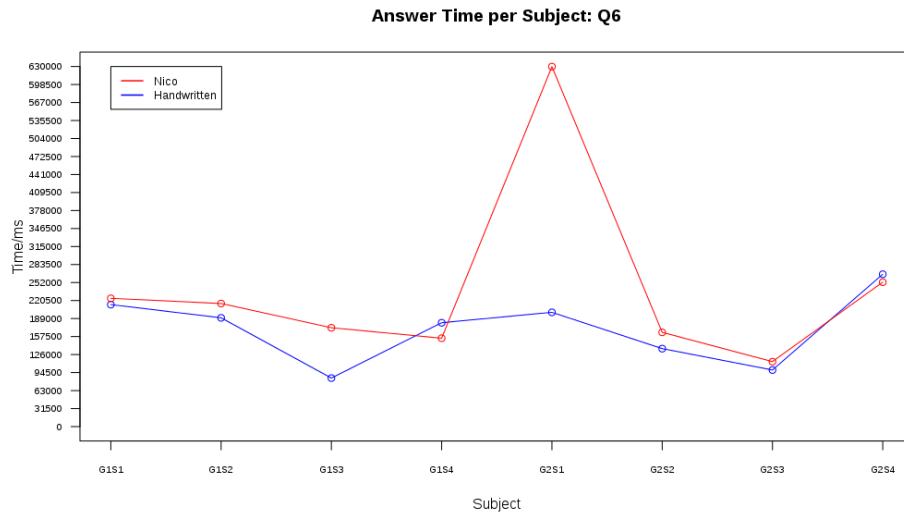


Figure 4.7: Time taken in milliseconds per subject to answer Question 6.

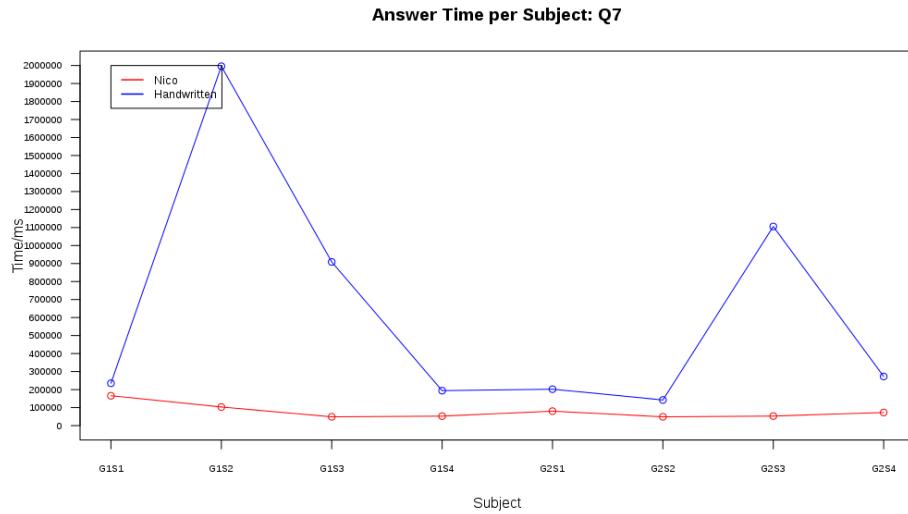


Figure 4.8: Time taken in milliseconds per subject to answer Question 7.

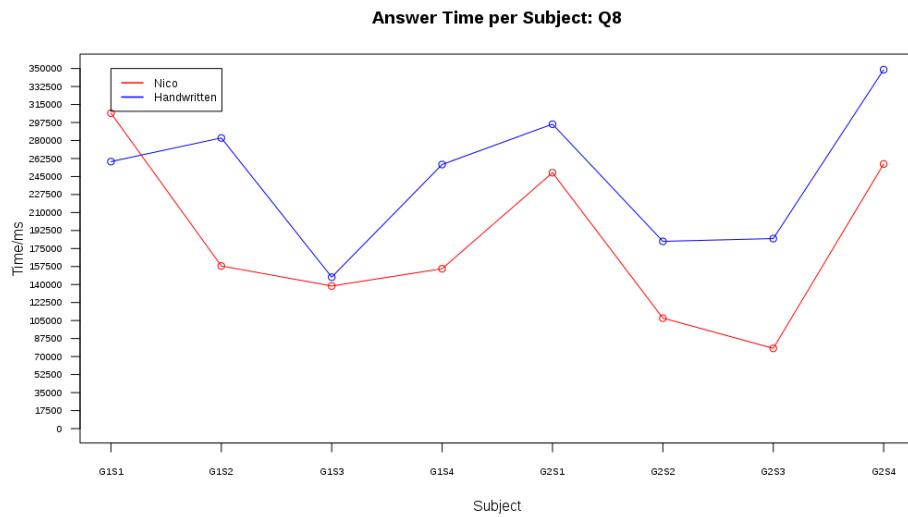


Figure 4.9: Time taken in milliseconds per subject to answer Question 8.

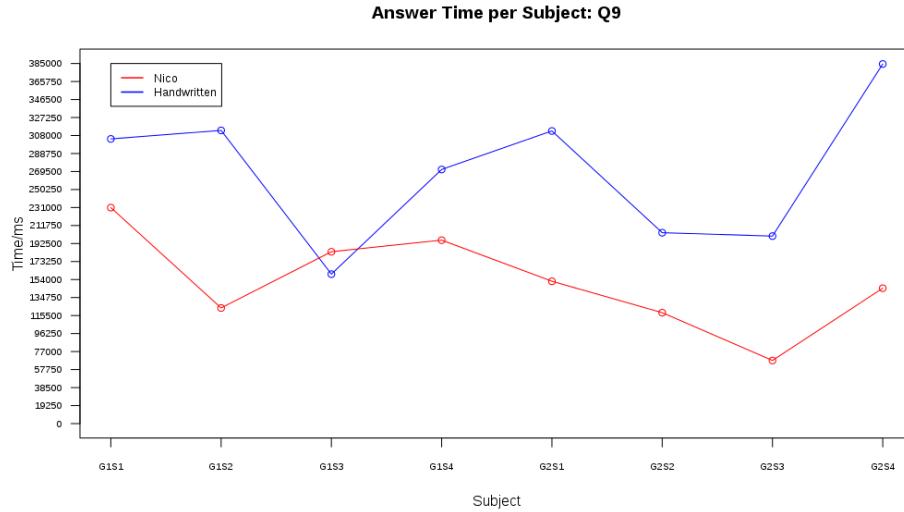


Figure 4.10: Time taken in milliseconds per subject to answer Question 9.

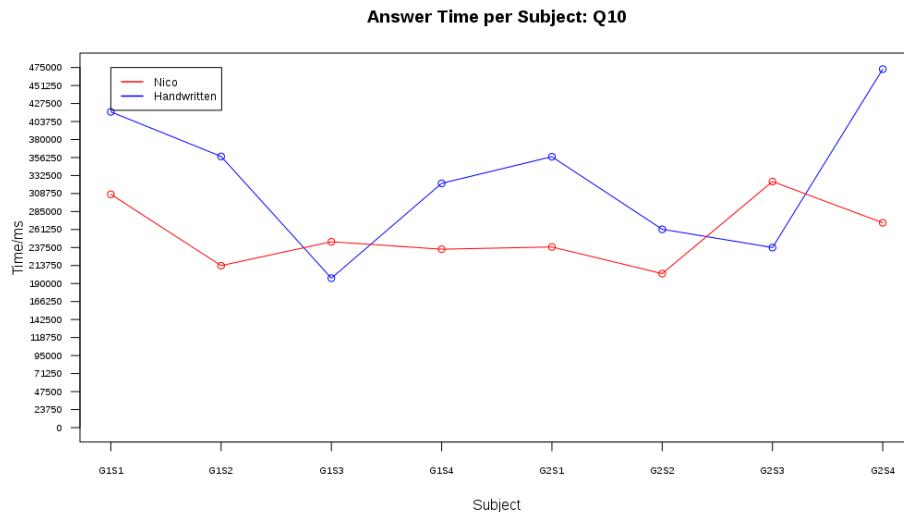


Figure 4.11: Time taken in milliseconds per subject to answer Question 10.

A two-sample, paired  $t$ -test using an  $\alpha$  value of 0.05 was used to determine whether or not the difference between the mean time taken to answer each question by hand and using *Nico* was statistically-significant. The paired  $t$ -test was chosen as the samples are not independent. The results of the  $t$ -tests on each question's datasets are shown in *Table x.y*. The data is assumed to fit a normal distribution. A Shapiro-Wilk test was considered to test each dataset for normality, but the sample size of eight data points per set (i.e. eight subjects for each question) was deemed to be too small for such a test to have meaningful results.

Question	<i>t</i>	<i>df</i>	<i>p</i>	Significant?
Q1	-8.0204	7	0.00008968	Yes
Q2	-8.0493	7	0.00008764	Yes
Q3	-8.9296	7	0.00004489	Yes
Q4	-2.9592	7	0.02113	Yes
Q5	-5.288	7	0.001138	Yes
Q6	-1.3149	7	0.2300	No
Q7	2.3688	7	0.0497	Yes
Q8	3.1	7	0.01732	Yes
Q9	4.0293	7	0.005	Yes
Q10	2.1298	7	0.0707	No

Table 4.2: Table showing the results of the paired *t*-test comparing the distribution of times taken to answer each question by hand and using the software, using an  $\alpha$  value of 0.05.

As the *p*-values in the above table indicate, there was a significant difference in performance between using *Nico* and completing the questions by hand. For the first five questions, the time taken to complete the questions is markedly increased by using the software. This is somewhat to be expected; the first five questions were ‘warm-up’ questions, requiring a maximum of three separate calculations to solve and with all numbers involved in each calculation being less than ten. Such questions were included to give the user a chance to get used to solving simple problems using *Nico*, before moving on to more complex tasks. By the sixth question, there is no longer a statistically-significant difference between using the software and completing the questions by hand. This particular question still uses numbers less than ten, but requires several more nested calculations to solve, suggesting that *Nico* offers some improvement where the handwritten method is lacking, but also that this is offset by the greater time required to input very simple expressions into the application. The final four questions introduced numbers greater than ten, include many subcalculations and large numbers. For the seventh, eighth and ninth questions, *Nico* offers a statistically-significant improvement. These questions involve the manipulation of larger numbers, requiring strategies that go beyond simply recalling number bonds, such as partitioning or long multiplication. It is these types of questions that *Nico* is intended to make easier for the user, and the often-dramatic decrease in the time taken to solve such questions is shown to great effect here. The tenth question, on reflection, was not well-chosen. It is comparable in difficulty to the fifth question, being a series of simple subcalculations with numbers less than ten. Though there are more calculations to perform, and one operation involving a large number at the end, this question is not

especially challenging to solve by hand, and as such the lack of a significant difference between using the software and not doing so is not entirely unexpected. The simple subcalculations are easier to perform by hand, and combining them is faster using *Nico*.

Data were also collected regarding how well the questions were answered by each user, which follow.

Question	Group 1							
	Subject 1		Subject 2		Subject 3		Subject 4	
	Marks	Errors	Marks	Errors	Marks	Errors	Marks	Errors
Q1	1	0	1	0	1	0	1	0
Q2	1	0	1	0	1	0	1	0
Q3	1	0	1	0	1	0	0	0
Q4	1	0	1	0	1	0	1	0
Q5	1	0	1	0	1	0	1	0
Q6	0	0	0	0	0	0	0	0
Q7	1	0	1	0	1	0	1	0
Q8	0	0	1	1	1	0	1	0
Q9	0	0	1	0	1	1	1	1
Q10	1	0	0	0	1	0	1	0

Question	Group 2							
	Subject 1		Subject 2		Subject 3		Subject 4	
	Marks	Errors	Marks	Errors	Marks	Errors	Marks	Errors
Q1	1	0	1	0	1	0	1	0
Q2	1	0	1	0	1	0	1	0
Q3	1	0	1	0	1	0	1	0
Q4	1	0	1	0	1	0	1	2
Q5	0	1	1	0	1	0	1	0
Q6	0	1	1	0	0	0	0	0
Q7	1	0	1	0	1	0	1	0
Q8	0	0	1	0	1	0	1	1
Q9	1	0	1	0	1	0	1	0
Q10	0	0	1	0	0	5	1	0

Table 4.3: Table comparing the mark obtained for each question answered by hand (a correct answer scored 1, whereas an incorrect answer scored 0) to the number of attempts made to answer each question before submitting the correct answer in *Nico*.

lol

#### 4.3.2.1 Feedback

## 4.4 Goals

## 4.5 Summary



# **Chapter 5**

## **Conclusions**



# **Appendix A**

## **Questionnaire**

The questionnaire used in the user study follows.

# Nico: An Environment for Mathematical Expression in Schools

Thank you for agreeing to participate in the user study for my Part II Project. *Nico* is a piece of educational software designed to aid learners in the visualisation of mathematical problems, by separating out the constituent parts of a calculation into distinct visual units on-screen.

The purpose of this study is to ascertain how well *Nico* achieves its goal of providing a clear, accessible, interactive means of calculation, and to gather feedback on how the application could be improved. The study also aims to compare *Nico* to traditional mathematical methods.

Please review and sign the attached Statement of Informed Consent (Section 1) and please feel free to ask any questions you may have about it.

## 1 Statement of Informed Consent

### Statement of Informed Consent

I state that I am over 18 years of age and wish to participate in a program of research being conducted by Philip Yeeles at the University of Cambridge. I acknowledge that this study has been approved by the University of Cambridge Computer Laboratory Ethics Committee.

The purpose of this research is to assess the usability of a graphical notation and software application for representing mathematical calculations in a graphical manner.

The study involves the use of the application whilst being supervised. I will be asked to complete certain tasks both with and without the application, and I will also be asked open-ended questions about the application and my experience as a user thereof.

All information collected in the study is confidential, and my name will not be identified at any time. I understand that I may ask questions or terminate my involvement in the study freely and at any time without consequence.

I acknowledge that my (anonymised) responses may be published in the final report, and that this report will be made publicly available from the University of Cambridge Computer Laboratory Library and from GitHub.

Signed:

Name:

Date:

THIS PAGE INTENTIONALLY LEFT BLANK

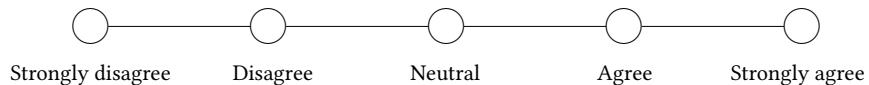
## 2 Study

Before we begin, please answer the following questions.

- Have you used *Nico* before (circle as appropriate)?

Yes      No

- How well do you agree with the following statement (cross as appropriate)?  
*I am confident in my ability to calculate answers to simple mathematical problems.*



Thank you. You will now be issued a group number.

- If you are in Group 1, please proceed to Section 2.1.
- If you are in Group 2, please proceed to Section 2.2.

If you would like to stop at any point, please don't hesitate to let me know.

### 2.1 Nico

In this section, you will use *Nico* to solve some simple mathematical problems. The purpose of this section is to evaluate how well *Nico* performs in comparison with manual calculation. We will be keeping a record of the time taken to complete each problem, but please do not let this make you feel rushed. Work at a pace that is normal and comfortable for you.

Before we begin the tasks, please watch the instructional video `tut.ogm` for a briefing on how to use *Nico* and an explanation of its controls.

Now that you have done this, please spend 5 minutes experimenting with *Nico*. Open the application and load the file `qs/blank.nqs` using the file chooser. As you explore, please tell me your thoughts about the application.

Now, let us move on to the problems. Please close the application and open it again, this time loading the file `qs/user-study.nqs`. You will be presented with a series of problems to solve using *Nico*; please solve them.

Thank you very much.

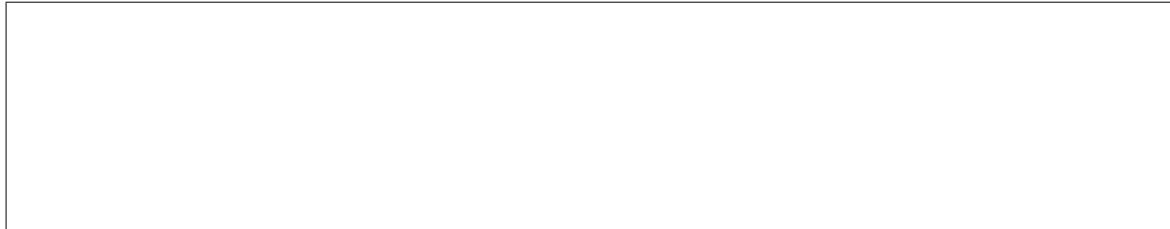
- If you are in Group 1, please continue to Section 2.2
- If you are in Group 2, please continue to Section 3

## 2.2 Manual Calculation

In this section, you will solve some simple mathematical problems using pen and paper. The purpose of this section is as a control, to compare to your results using *Nico*. Once again, we will be keeping a record of the time you take to complete each question, but please do not let this make you feel rushed. Work at a pace that is normal and comfortable for you, and don't forget to show your working.

Let us begin.

1.  $2+3$



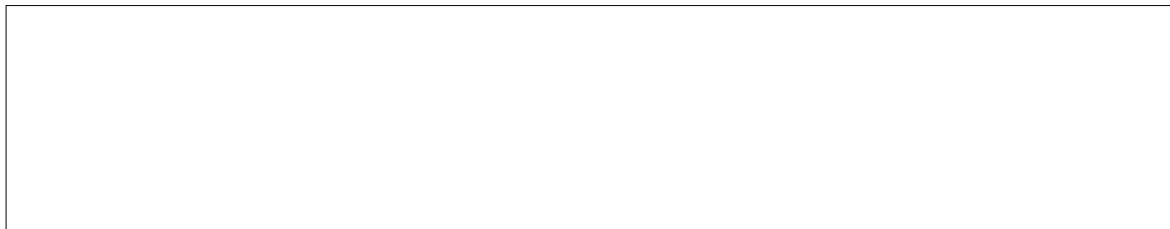
2.  $9 \div 3$



3.  $1+2+3+4+5$



4.  $(2 \times 4) + (3 - 5)$



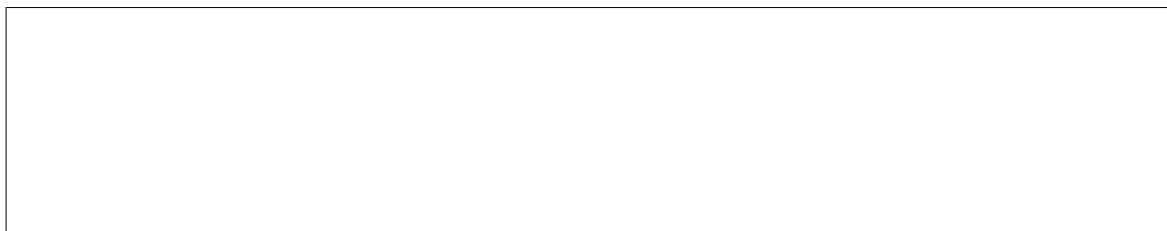
5.  $((3 \times 4) \div (3 + 3)) \times 8$



6.  $((1+2+3+4+5)+(2 \times 4 \times 6 \times 8 \times 10)) \times 1 \times 2 \times (1+2+3+4+5)$



7.  $12 + 14$



8.  $247 \times 35$



9.  $120 \div ((2 \times 10) + 5 + 5)$



10.  $((2+5)\times(6\div2)\times(9-8))+((3+4)-(5\times6))+120$



Thank you very much.

- If you are in Group 1, please continue to Section 3
- If you are in Group 2, please continue to Section 2.1

### 3 Questions

#### 3.1 Notation

When using the system, what proportion of your time (as a rough percentage) do you spend:

1. Searching for information within the notation  %
2. Translating substantial amounts of information from some other source into the system  %
3. Adding small bits of information to a description that you have previously created  %
4. Reorganising and restructuring descriptions that you have previously created  %
5. Playing around with new ideas in the notation, without being sure what will result  %

#### 3.2 Cognitive Dimensions

##### 3.2.1 Visibility and Juxtaposability

1. How easy is it to see or find the various parts of the notation while it is being created or changed? Why?

2. What kind of things are more difficult to see or find?

3. If you need to compare or combine different parts, can you see them at the same time? If not, why not?

##### 3.2.2 Viscosity

1. When you need to make changes to previous work, how easy is it to make the change? Why?

2. Are there particular changes that are more difficult or especially difficult to make? Which ones?

### 3.2.3 Error Proneness

1. Do some kinds of mistake seem particularly common or easy to make? Which ones?

2. Do you often find yourself making small slips that irritate you or make you feel stupid? What are some examples?

### 3.2.4 Closeness of Mapping

1. How closely related is the notation to the result that you are describing? Why?

2. Which parts seem to be a particularly strange way of doing or describing something?

### 3.2.5 Role Expressiveness

1. When reading the notation, is it easy to tell what each part is for in the overall scheme? Why?

2. Are there some parts that are particularly difficult to interpret? Which ones?

3. Are there parts that you really don't know what they mean, but you put them in just because it's always been that way? What are they?

#### 3.2.6 Hidden Dependencies

1. If the structure of the calculation means that some parts are closely related to other parts, and changes to one may affect the other, are those dependencies visible? What kind of dependencies are hidden?

2. In what ways can it get worse when you are creating a particularly large description?

#### 3.2.7 Progressive Evaluation

1. How easy is it to stop in the middle of creating some notation, and check your work so far? Can you do this any time you like? If not, why not?

2. Can you find out how much progress you have made, or check what stage in your work you are up to? If not, why not?

3. Can you try out partially-completed versions of the calculation? If not, why not?

### 3.2.8 Provisionality

1. Is it possible to sketch things out when you are playing around with ideas, or when you aren't sure which way to proceed? What features of the notation help you to do this?

### 3.2.9 Secondary Notation

1. Is it possible to make notes to yourself, or express information that is not really recognised as part of the notation?

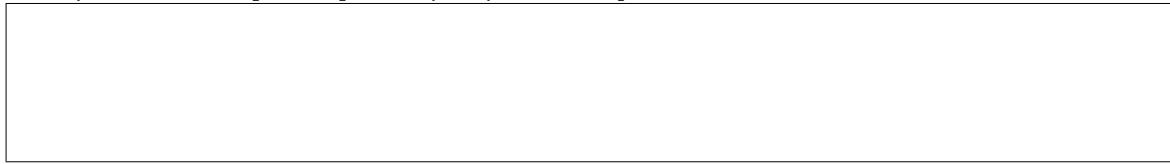
2. If it was printed on a piece of paper that you could annotate or scribble on, what would you write or draw?

3. Do you ever add extra marks (or colours or format choices) to clarify, emphasise or repeat what is there already?

### 3.3 Feedback

1. Do you find yourself using this notation in ways that are unusual, or ways that the designer might not have intended? If so, what are some examples?

2. After completing this questionnaire, can you think of obvious ways that the design of the system could be improved? What are they? Could it be improved specifically for your own requirements?



Thank you very much for your help with my project! Your responses will remain confidential, although they are liable to appear in an anonymised form in the final report, of which a copy will be retained by the University of Cambridge Computer Laboratory Library (<http://www.cl.cam.ac.uk/library/>). The final report will also be available via my repository at GitHub (<http://github.com/loomcore/nico>).

## **Appendix B**

### **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<sup>1</sup>. 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.  $12 \times 34 \Rightarrow (10 \times 34) + (2 \times 34)$ )
  - 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

<sup>1</sup> *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

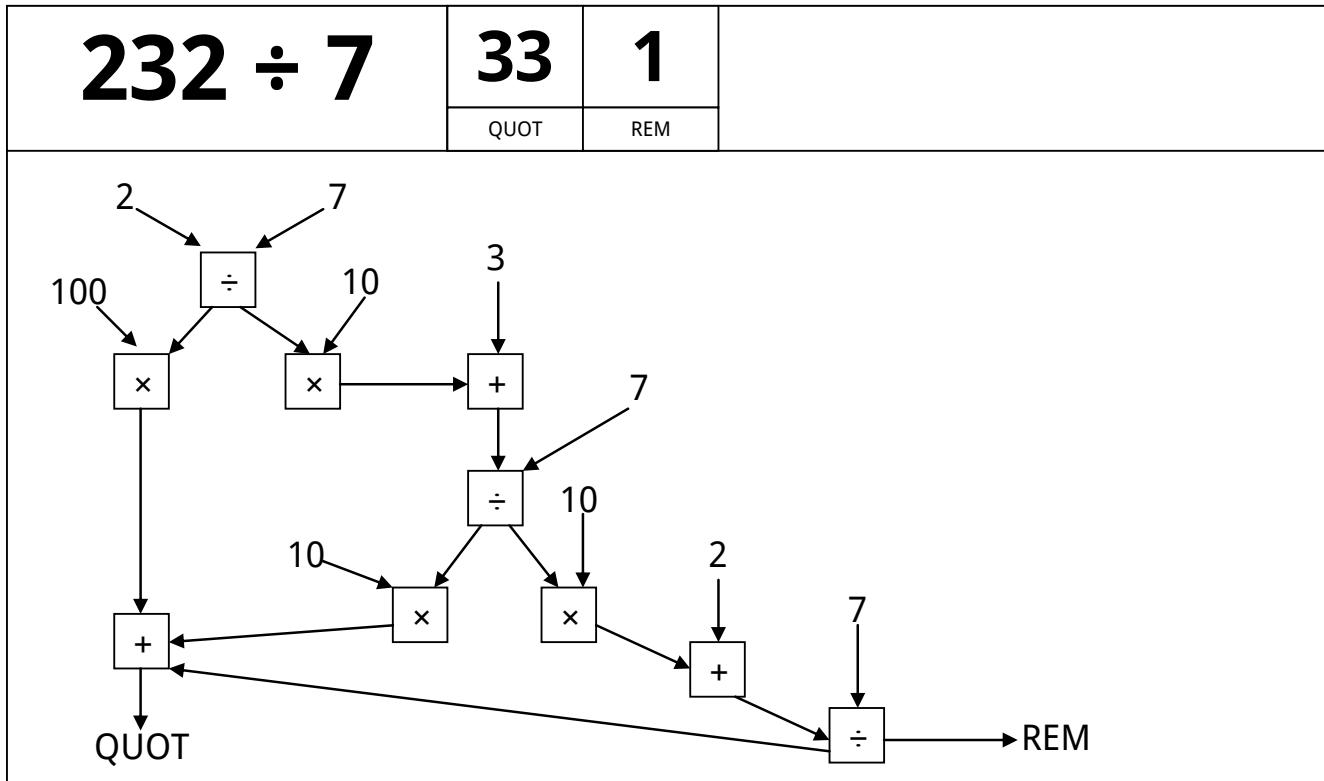
The following main learning tasks will have to be undertaken before the project can be started:

- 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
  (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.