Philip Yeeles

Nico: An Environment for Mathematical Expression in Schools

Computer Science Tripos

Selwyn College

May 17, 2012

Proforma

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

Original Aims of the Project

The aim of this project was to develop an application to act as an aid in mathematical education for pupils in Year 5. The software was to provide a new graphical metaphor for calculation with an accompanying working environment, with the intention of eliminating some of the problems that makes handwritten arithmetic unclear in many situations.

Work Completed

I have successfully designed and implemented an application using the Clojure programming language that allows users to express calculations using a graphical notation. The software is able to generate an abstract syntax tree from the graphical notation, evaluate it and pass the results back to the application in approximately

[1] This word count was computed using the web interface to `texcount`, which can be found at `http://app.uio.no/ifi/texcount/online.php`.

10ms. As an extension to the project, a user study was conducted to evaluate the utility of the software.

Special Difficulties

- Learning the Clojure programming language
- Learning to choose appropriate mutable data structures of those that Clojure provides
- Learning to incorporate tests into the development cycle

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 17, 2012

Contents

List of Figures

Acknowledgements

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

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 in schools. Using Petre and Green's 'Cognitive Dimensions' framework as a means of evaluation, the project aims to increase the *visibility* of the notation, to reduce the number of *hidden dependencies* between units of notation, to reduce the *viscosity* of the notation, and to make the flow of data obvious to the user [12]. 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 have observed both in my own learning and in my own teaching experience. What follows is an assessment of the technical challenges involved in this project, followed by a discussion of the ideal properties of a useful alternative notation.

1.2 Technical Challenges

Developing such an application comprises three main challenges. Firstly, developing an infrastructure that is capable of creating, storing, editing, deleting, evaluating and nesting calculations. Secondly, developing an interaction-handler layer that is able to interpret user input and events occurring in the user interface. Finally, implementing a graphical user interface 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. Such an interface should also be able to display and update the results of a calculation with low latency (a limit of 300ms is set and discussed in footnote 1, *Sec. 2.1.2*).

The application described above must be portable, to account for the diversity of computing environments available to the target audience. It must also provide a standardised user experience across the platforms on which it is deployed. A Java-based application satisfies both of these requirements, as any platform on which the Java Virtual Machine (JVM) can be installed is able to run it. Java also makes GUI libraries available, such as JavaFX and Swing, that are explicitly designed to provide similar GUI functionality on many platforms.

The application would also benefit from functional programming paradigms such as immutability, which results in functions that behave predictably and are easy to test; and homoiconicity, in which data structures can be evaluated as code and code as data structures, allowing mathematical method to be represented in a form that can be interpreted and manipulated by the application.

The Clojure language meets both of the above requirements, being a dialect of LISP that compiles to JVM bytecode. This gives the developer access to the Java standard library and offers the portability afforded by Java, whilst also providing the benefits of a homoiconic, functional language as described above. It also has the added benefit of being efficient, as it compiles directly to JVM bytecode.

1.3 Previous Work

There already exists a wide variety of educational software for mathematics, which often present mathematical problems in the context of a computer game. Although such games have repeatedly been shown to be of benefit to the learner, a barrier to their widespread use in education can be the users' preconceptions about the educational value of computer games [27] [9]. Indeed, Bragg notes that

> "One barrier [...] may be possible negative attitudes held by students towards the likelihood of the games' effectiveness in assisting mathematical learning." [6]

This is reinforced in a later paper by the same author, where Bragg concludes that

> "It appeared that game-playing negatively affected attitudes" [7]

Although it was also concluded that such problems could be addressed by dedicating lesson time to explaining the educational benefits of game-playing, an alternative approach is to introduce software into the classroom as a **tool**, rather than 'just a game'. In the same way that calculators are used in mathematics lessons, it is the intention of this project to introduce a new tool to complement learning.

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. One such example is *Pi Cubed*, which tries to make complex calculations appear as they would be written in an exam or exercise book [17]. *Soulver*, conversely, tries to achieve this by simulating 'back-of-the-envelope' calculations, whereby notes in English augment the calculation [18].

Another approach is that of the *Scrubbing Calculator*, which extends the *Soulver*-style environment by helping the user to solve equations by dragging values to edit them, showing how changing a value affects the overall result [29]. Values can be linked by dragging a line between them, which means that they are two instances of the same value: 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 an equation. The *Scrubbing Calculator* is more a tool for facilitating algebraic, rather than arithmetic, understanding; indeed, it is inherently a **calculator**, and so does not encourage thinking about a method of manual calculation.

1.4 Summary

Existing educational games for mathematics, although valid as a pedagogical tool, face obstacles from teachers' and pupils' attitudes toward them, as a result of their status as 'just games'. There exists software to aid in calculation and arithmetic by representing it clearly, but it is not intended for classroom use, and often its purpose is to make on-screen calculations simulate handwritten ones. An application is proposed to complement teaching, as one would use a calculator in the classroom. There is a niche for a tool for use in education that represents calculations in a visual manner, with a particular focus on making mathematical **method** clear. The application is to provide such 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 development. It comprises a requirements analysis, followed by a discussion of the prototyping of the graphical notation that was to be implemented in the final application. A list of third-party tools used can be found in *Appendix A*.

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. This section includes an analysis of the handwritten method according to the 'Cognitive Dimensions' framework.

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

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

2.1.1.1 Disadvantages

First of all, the fact that the traditional notation for arithmetic is handwritten entails a high level of *viscosity*: it is difficult to make changes to a calculation without sacrificing clarity. In particular, there is a lot of *repetition viscosity* involved in modification: if a number is changed that is reused, 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 making modifications. 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 notation. 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*, where decisions regarding the placement of components have to be made before the required information is available. 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*. It can actually be quicker for the user to start over again, as opposed to simply amending their calculations.

2.1.1.2 Abstraction

Handwritten arithmetic does have one key advantage: it has a very low initial *abstraction barrier*. Once the symbols and rules for each operation and digit have been learnt, the traditional system of arithmetic allows a learner to begin using it almost immediately. 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. Thus algebra, as an example, is an *abstraction-hungry* system, requiring the concept of a variable rather than a set quantity; whereas arithmetic is an *abstraction-hating* system. Without abstractions, arithmetic is easy to begin using, but can be very verbose and *viscous*, with low *visibility*.

To improve upon the standard approach of listing the steps comprising a calculation, a notational system must acknowledge and try to overcome the drawbacks listed above. To this end, I have designed and developed such a system and accompanying application, with the intention of reducing *viscosity*, increasing *visibility* and removing many of the *hidden dependencies* that beleaguer arithmetic. The functional and non-functional requirements of this system follow.

2.1.2 Functional Requirements

The new system must satisfy the following functional requirements to be considered acceptable. It must be able to:

- Present the user with a series of problems to solve
- Accept a file containing the set of questions to be answered
- Display the current question being answered
- Determine whether or not the user's solution is correct
- Progress through the current question set as the user answers each question correctly
- Interpret the user's work and display the result interactively [1]

2.1.3 Non-Functional Requirements

The system must also satisfy a number of non-functional requirements. It must, compared to handwritten arithmetic:

- Offer an improvement in *visibility*
- Offer an improvement in *juxtaposability*
- Reduce *premature commitment*
- Reduce *hidden dependencies*

And also:

- Have a visually simple interface whilst remaining accessible to older users

[1] Roca and Rousseau note that: "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." [24]

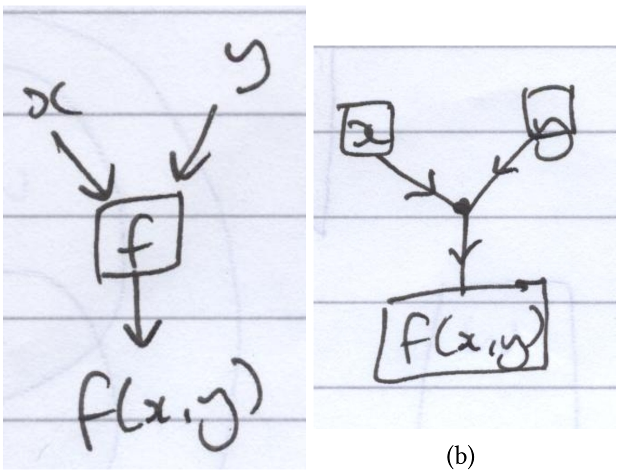
2.2 User Interface

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

2.2.1 Prototyping

2.2.1.1 Low-Fidelity Prototyping

To devise initial ideas for what could become the calculation metaphor, a target was set of devising at least twenty significantly-different potential designs. These were recorded as rough sketches, three of which were warranted an application mockup, as detailed in *Sec. 2.2.1.2*.



(a) (b)

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

The designs in *Fig. 2.2* use a flowgraph-based metaphor, in which variables and functions are represented as nodes in the graph. Dataflow is extremely clear in such diagrams, giving these notations high *visibility*. These designs are similar to the original one illustrated in the project proposal (*Appendix D*). The boxes and arrows used in such a metaphor also occupy relatively little screen-estate, which is advantageous in cases where a question necessitates many subcalculations. *Fig. 2.3* shows a refined version of a flowgraph notation, with a potential environment in which to utilise it.

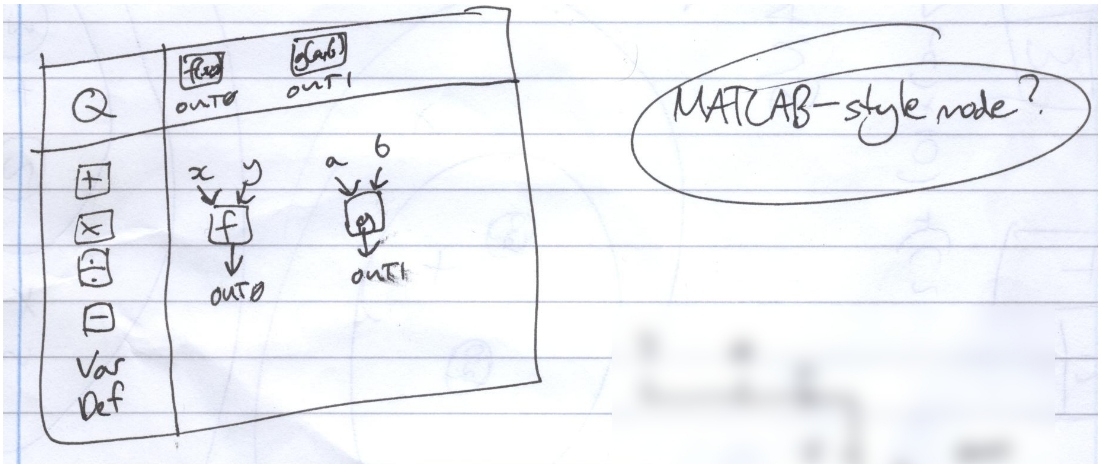


Figure 2.3: A refined sketch of a flowgraph-based language and application.

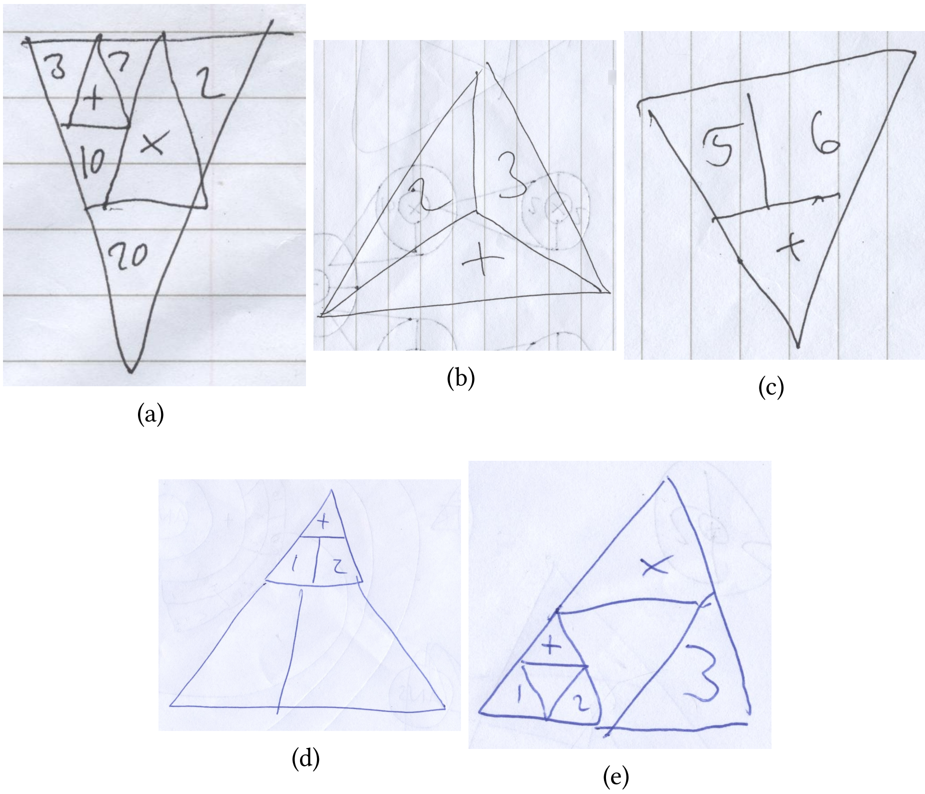


Figure 2.4: Early designs for triangle-based calculation metaphors.

Triangle-based notations were also considered, drawing inspiration from the formula triangles used in science education (*Fig. 2.4, c)* and *d)*) and from the

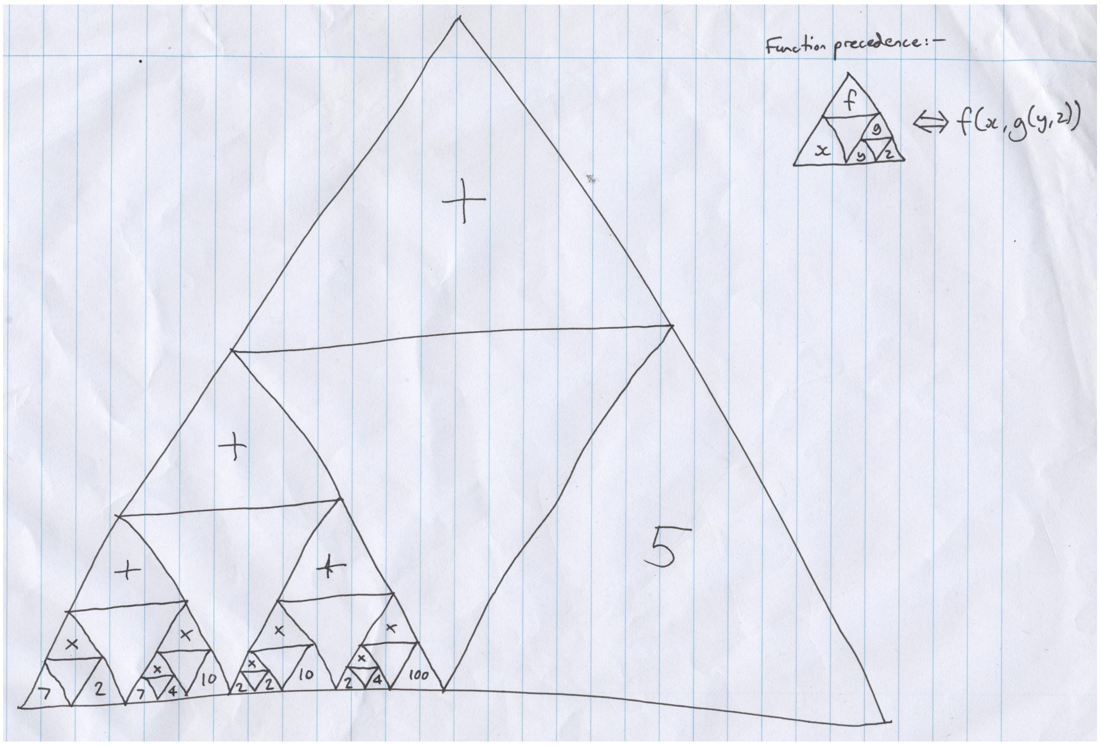
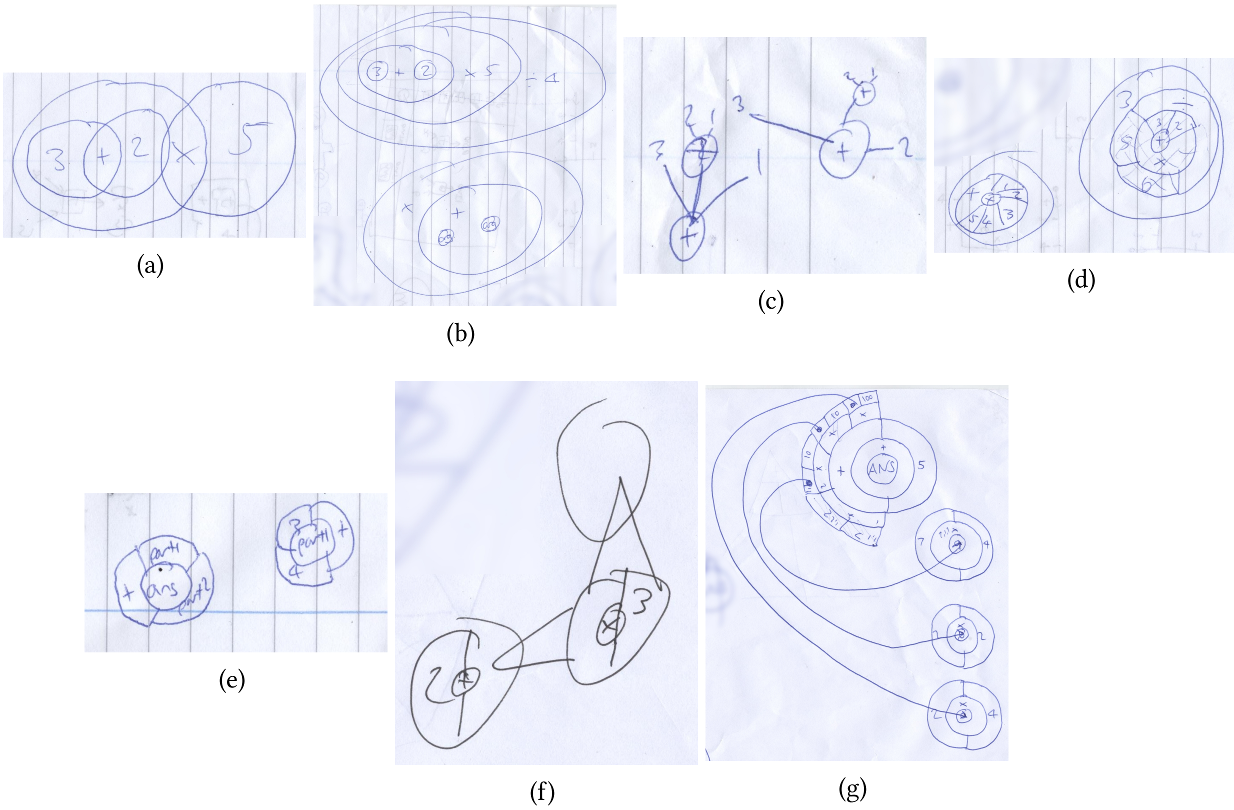


Figure 2.5: A refined sketch of a triangle-based language.

Sierpiński triangle fractal (*Fig. 2.3*, *a)* and *e)*). Such notations not only appear in a form familiar to many users, but also have high *visibility*. Their *juxtaposability*, however, is low, as a rigid structure is imposed upon calculations. A more refined sketch is shown in *Fig. 2.5*.



(a) (b) (c) (d) (e) (f) (g)

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

Several circle-based notations were also considered, the focus here being on subexpressions, providing high *juxtaposability*, as whole units can easily be relocated. This is opposed to using separate numbers and operations as the basic units of a large calculation. These also offer an improvement in *visibility* over handwritten arithmetic, as discussed in *Sec. 2.2.1.2.3*. A more detailed sketch is shown below.

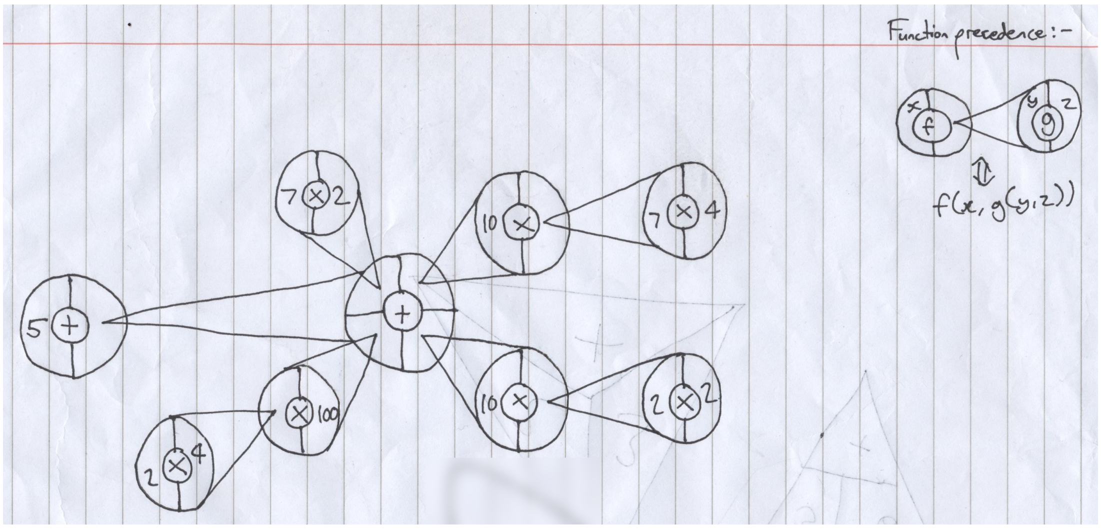


Figure 2.7: A refined sketch of a circle-based language.

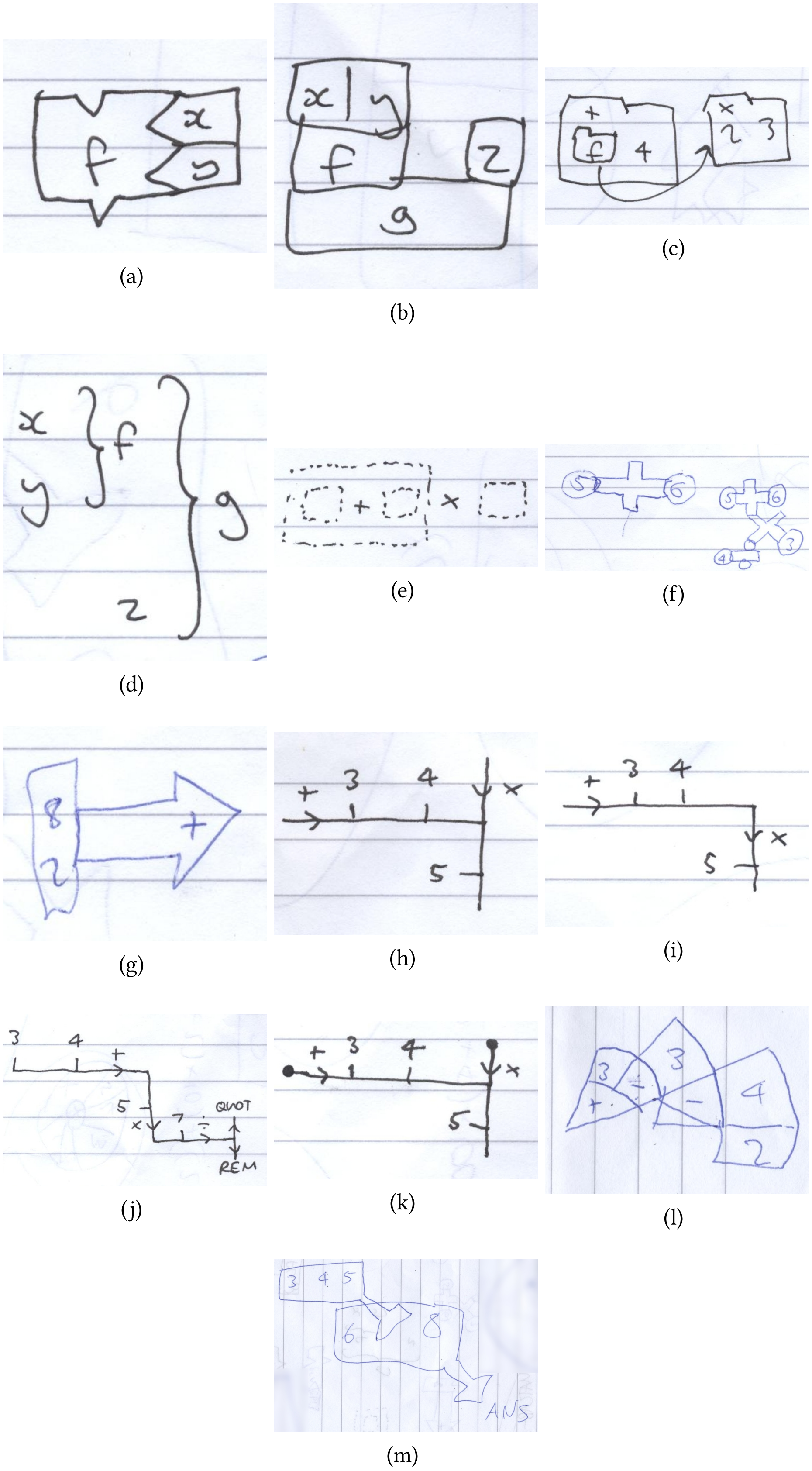


Figure 2.8: Assorted early designs for other calculation metaphors.

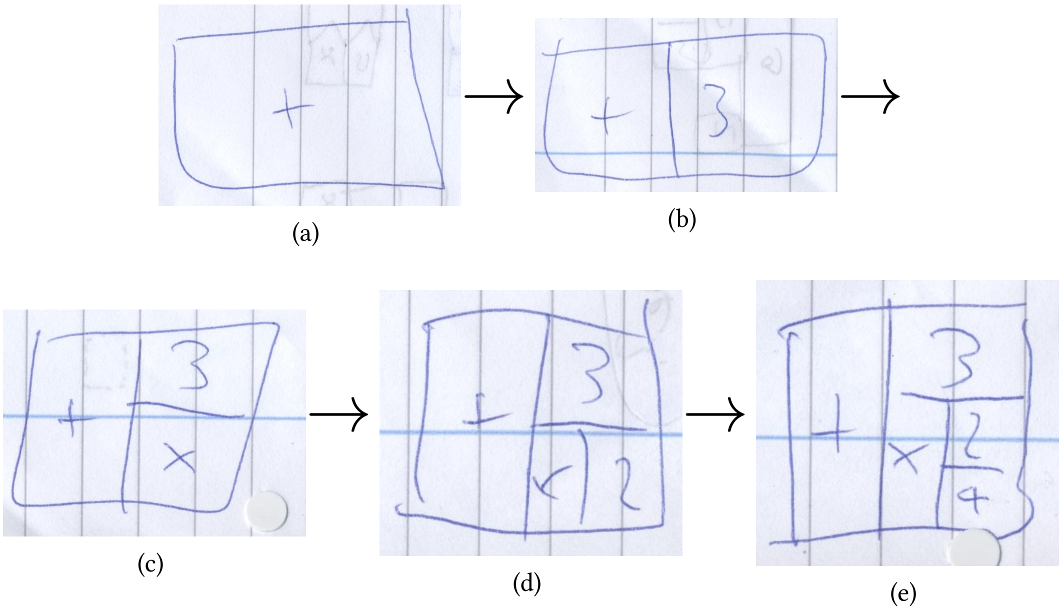


Figure 2.9: A sample progression of a calculation using a language based around an infinitely-subdivided rectangle.

Various other metaphors for calculation were also considered, as illustrated in *Figs. 2.8 and 2.9*. *Fig. 2.8 a)* was considered too similar to Google/MIT's *App Inventor* software. Sketches *c)*, *f)*, *l)* and *m)* were deemed too cumbersome, and *d)*, *e)* and *h)* to *k)* were not considered visually appealing, appearing too similar to existing mathematical notations and constructs. Items *b)* and *g)* were inflexible, and would have scaled poorly to larger calculations. Finally *Fig. 2.9* was rejected as it was too similar to the Sierpiński triangle metaphor shown in *Fig. 2.5*.

2.2.1.2 Detailed Mockups

More mature designs for the application and graphical language follow. First is a combined flowgraph representation and Read-Evaluate-Print-Loop (REPL) design. The second uses the Sierpiński triangle as a basis for the visual metaphor. The final design is a circle-based language, similar to *Fig. 2.7*.

2.2.1.2.1 Flowgraph Metaphor

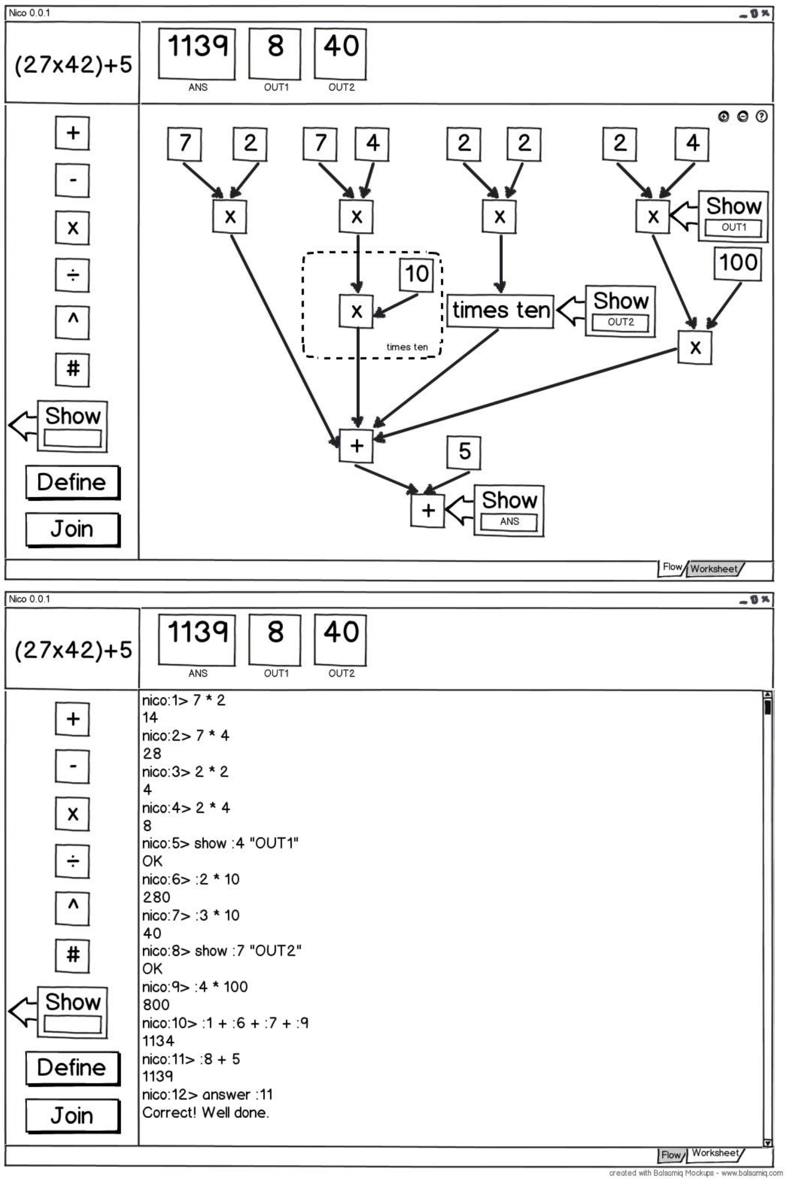


Figure 2.10: The prototype flowgraph-style language and accompanying application.

This layout expands the application prototyped in the project proposal. The application now comprises one window containing information panels, a 'toolbox' of available functions that can be dragged on to the canvas, and a canvas area upon which calculations are constructed. Zoom and help buttons are provided. The user is able to use the Flow and Worksheet tabs to switch between the graphical mode and the REPL mode respectively.

The top screen in *Fig. 2.10* shows manipulation of the graphical language, whilst the bottom screen shows a REPL utilising a simple text-based language. The flowgraph notation was used as it is *visible*, making the flow of data very clear to the user and allowing them to see much of their structure at once. The notation also occupies little screen-estate, allowing it to accommodate large calculations.

The toolbox was included to exhibit available functions. In the original design, no control scheme was specified; the user was expected to have control over many different functions, meaning that unless it were simple, the user may require guidance. There are two buttons in the toolbox: Join, which changes the function of the mouse to join boxes with arrows; and Define, which similarly allows the user to draw a box around a subsection of the diagram to define a function. This was introduced to allow for the creation of abstractions for repeated tasks, making the language more compact. It was decided to use these modes, combined with the placement of boxes in the default mode, so that the entire control scheme could be implemented using one mouse button.

The zoom buttons were included to accommodate large diagrams, and to allow the user to view their work at a comfortable size. The help button was included to provide assistance for new users. Activating help would provide information explaining how the application works.

The Worksheet mode was included for users who may feel more comfortable working in a written environment, rather than using the graphical metaphor.

This design was abandoned as it was deemed to be too complex for the target audience (Year 5). The intention of this project was not to implement an educational programming language, and by including functionality such as the REPL and Define, the system overprovides. The REPL itself is superfluous, as users comfortable working in a textual environment are likely to be comfortable using the handwritten method.

In addition to this, the flowgraph representation had a number of more general shortcomings. For example, the evaluation order of arguments is unclear, which is confusing for non-commutative functions such as subtraction. Furthermore, the language is verbose. This is not inherently a disadvantage but a more compact representation would be preferable in order to make expressions more *visible*, and to decrease time spent by the user searching for information within the notation. It is also *viscous* when replacing whole subcalculations, rather than just numbers or operators, though less so than handwritten arithmetic.

2.2.1.2.2 Sierpiński Triangle Metaphor

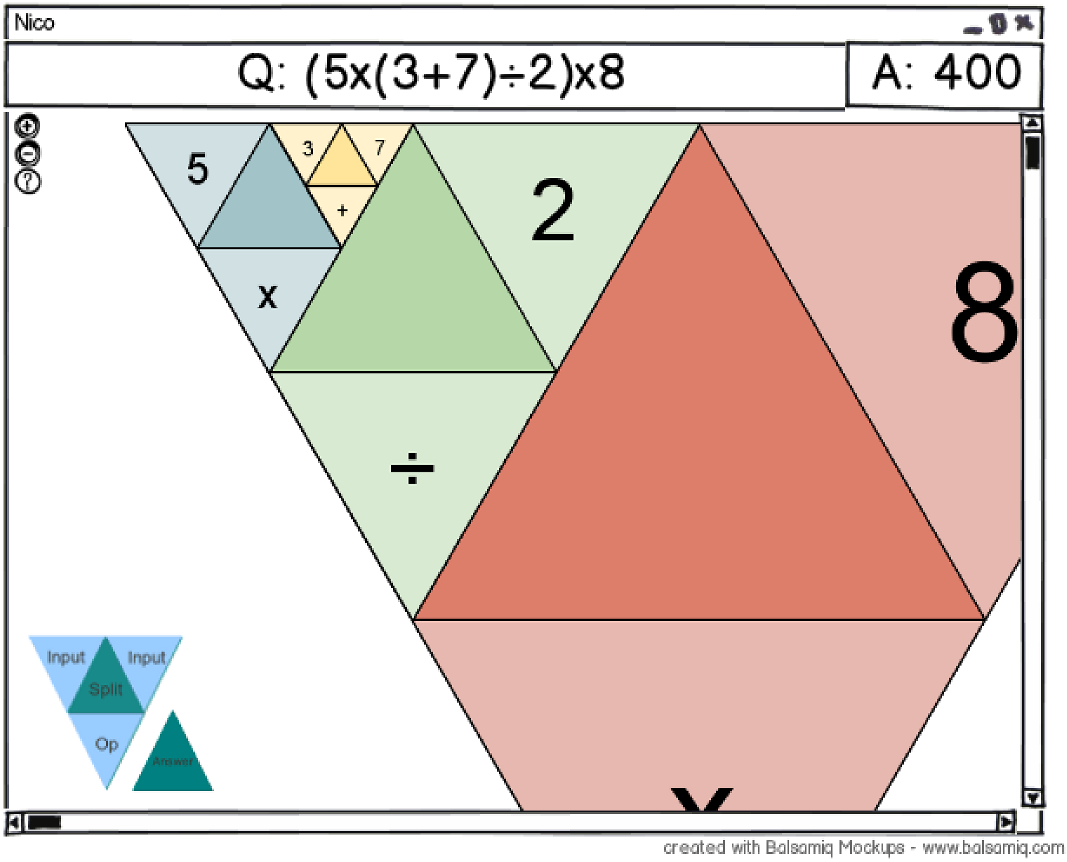


Figure 2.11: The prototype Sierpiński-triangle-based language and accompanying application.

This prototype uses a zooming instance of the Sierpiński triangle fractal as the basis of its notation, as shown in *Fig. 2.11.* The application consists of a single window with information panels and a canvas area, featuring zoom and help buttons, as well as control buttons for diagram construction.

This prototype marks a significant departure from the flowgraph metaphor outlined above and in the project proposal to address the above problems with the flowgraph notation. The application also had to change to suit the new notation: in this case, a toolbox would not be suitable, as the user is limited to one structure.

The triangle notation was chosen as it is more *visible* than the handwritten method, and places considerable emphasis on structure, making it clear how smaller calculations can combine to form larger ones by using a fractal-based notation. It is colour-coded by the level of nesting of each triangle, clarifying at which level the user is working and allowing the user to abstract away smaller triangles as arguments to encompassing calculations.

Zoom buttons were included as, although the size of the overall structure always remains the same in this notation, it becomes necessary to read very small sections quite quickly, as nested calculations are built up. Indeed, each section constitutes a quarter of the area of the triangle that contains it, as illustrated in *Fig. 2.11*.

As in the previous prototype, a help button was included to ease initiation.

A control scheme using buttons was again chosen so that only one mouse-button was required. The arrangement of the control buttons maps to the arrangement of triangles within the diagram. This design was chosen to clarify the consequences of each control upon the structure: the `Input` and `Op` buttons allow the user to modify the corresponding section of the triangle. The user is able to submit answers using the `Answer` button, allowing them to check and change their solutions before submission.

This design was ultimately abandoned as it was awkward. It is based 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 deduce that 1+2+3=6 and then calculate 6+4, which is unacceptable for a system designed to aid arithmetic learning.

The severe lack of *juxtaposability* is also problematic: as each expression must fit into the larger structure, it is not possible to reorder them without fundamentally altering the calculation being performed. It also enforces a top-down approach to calculation: one is required to begin with the outermost calculation and work inwards. There is no provision for encapsulating an existing calculation within another triangle without significant *premature commitment*.

In addition to this, an informal survey regarding potential notations was conducted, asking approximately ten participants to answer some sample questions in this notation and the circle-based notation below. Participants found this notation less clear, and were able to complete more questions correctly using the circle notation.

2.2.1.2.3 Circle Metaphor

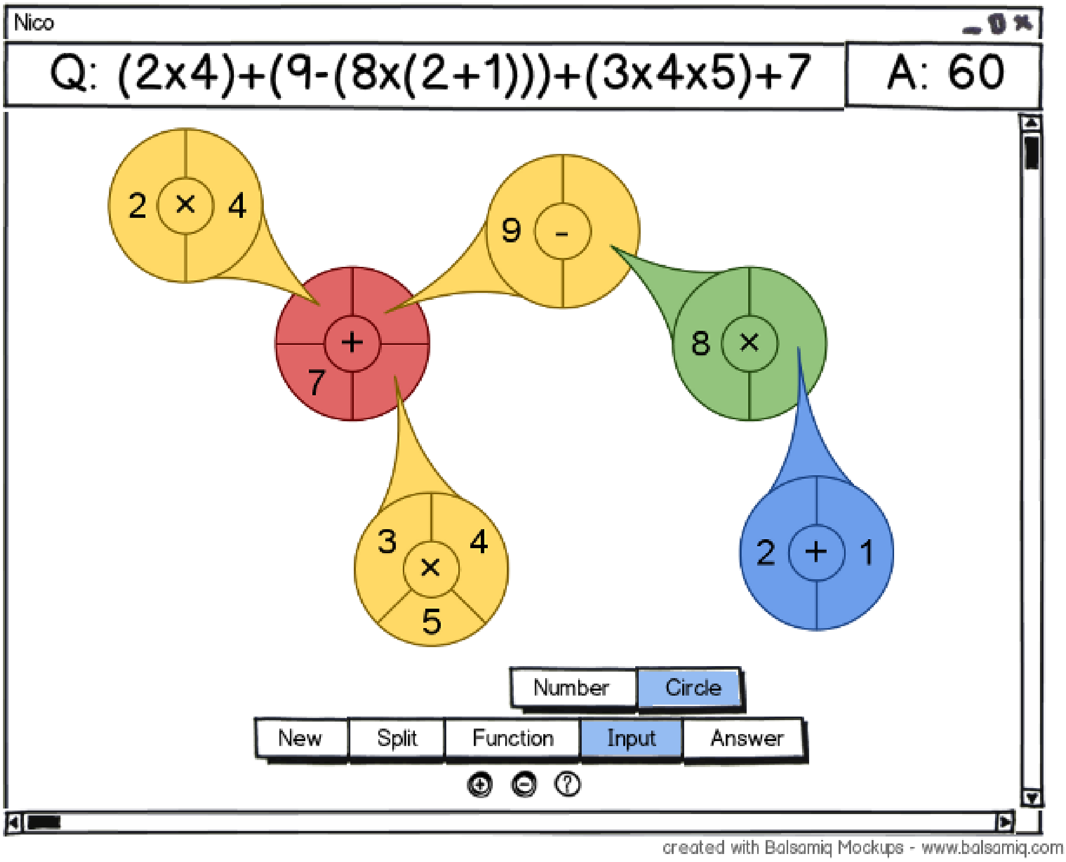


Figure 2.12: The prototype circle-based language and accompanying application.

In this prototype, circles constitute individual units of calculation, linked to indicate the flow of information. Each circle consists of an operator, around which arguments orbit, which are evaluated clockwise from 0°. It is related to the flowgraph metaphor above, but has a number of key differences.

The application comprises a single window with information panels and a canvas, which includes control, zoom and help buttons, similar to the previous two models. Zoom buttons accomodate large diagrams, again allowing the user to focus on small sections and to work at a scale that is comfortable. The help button eases initiation.

Similar to the Define and Join buttons in *Fig. 2.10*, the buttons specify mouse modes: New causes clicking to create new circles, Split causes clicked circles to gain an argument, and so on. The motivations for this were similar to that for the modes in the flowgraph model, including the ability to control the software using one mouse button. The user is able to submit answers using the Answer button, to allow them to check and change their answers before submission.

Again, this prototype is markedly different to the idea originally proposed. The application has changed to accommodate the new notation: a toolbox is unsuitable as the notation uses only one fundamental structure, a circle, and the controls in the Sierpiński model (*Fig. 2.11*) are inappropriate.

Unlike the Sierpiński metaphor, the user is not restricted to one order or pattern of constructing a calculation. The circle 'tails' make the flow of information through the diagram clear, and are colour-coded by their level of nesting, to further clarify dataflow and to eliminate *hidden dependencies*.

The circle metaphor was chosen for similar reasons to the triangle model: it emphasises the idea of subcalculations as individual units to be abstracted away and treated as any other argument. This notation has greater *visibility* than handwritten arithmetic, and offers more flexibility than the previous two models, decreasing *viscosity* by allowing the user to edit whole expressions within a calculation and exchange them as needed. *Juxtaposability* is increased as whole subcalculations are easily repositioned. The system requires little *premature commitment*, as any piece of the diagram can be modified at will. This 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 and transcribing the result.

This design was ultimately chosen as it solved, or at least ameliorated, many of the problems listed with handwritten arithmetic, without being too complex for the target audience, and without being too restrictive with regard to mathematical expression. Furthermore, the results of the survey mentioned above also indicated that participants found this notation more intuitive, and made fewer mistakes.

2.3 Summary

Prior to the implementation of the project, a thorough requirements analysis was conducted to outline the expected behaviour of the application, contrasting it with the existing system of handwritten calculations. In *Sec. 2.2*, prototypes for a suitable graphical notation were developed, and three were explored in detail. Two of these were offered as alternatives to the flawed original design. The final choice of the circle-based metaphor has many advantanges over the original flowgraph notation.

Chapter 3

Implementation

This chapter includes a discussion of the system architecture, detailing the implementation of its infrastructure, management of mutable state, administration of circles, interpretation and evaluation of the notation, and handling of the questions posed to the user. This is followed by a discussion of the interaction handler, the libraries used, the underlying structure of the user interface and the control scheme. Next, there is an analysis of the development of the user interface, considering the application itself and its control scheme, use of colour-coding and notation. Finally there is a discussion of testing methods used.

The code itself totals approximately 1,500 lines of Clojure, and was developed in a bottom-up manner, using the iterative and incremental method of software development, in which components of the application are planned, implemented, tested and evaluated before being integrated with larger components. The application satisfies the success criteria 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 user study.

3.1 System Architecture

The system comprises three layers of operation: the user interface, the interaction handler and the infrastructure. Events occurring in the user interface are processed by the interaction handler, with the resultant data being passed to the infrastructure. This handles the interpretation of such data, as well as the memory-management operations. Data can also flow from the infrastructure to the interaction handler, with results being passed to the interaction handler, causing effects in the user interface. A detailed analysis of each layer follows.

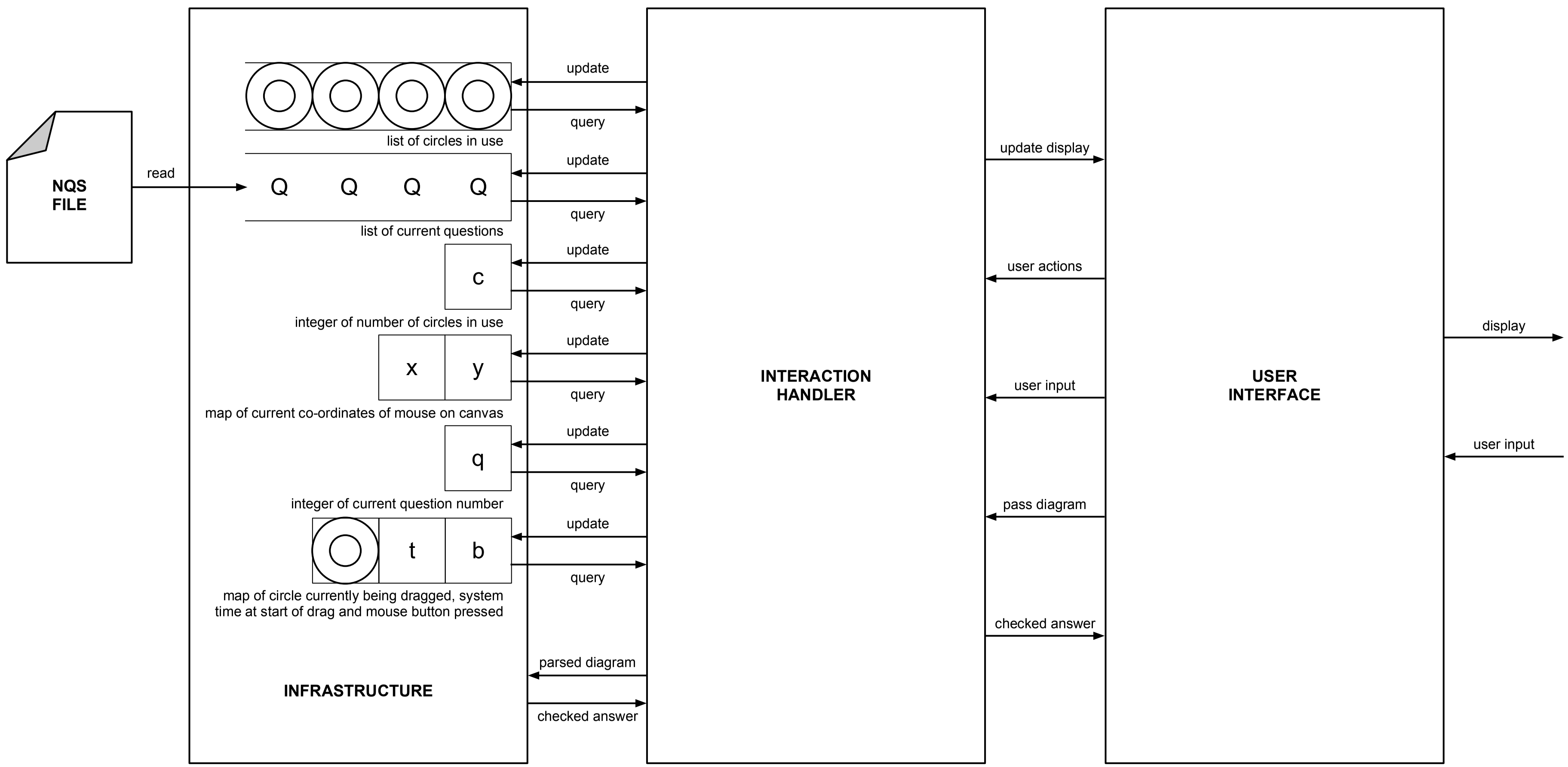


Figure 3.1: A model of *Nico*'s three-layer architecture.

3.1.1 Infrastructure

The application infrastructure handles mathematical operations; that is, it is able to interpret data from the user's input and process it as a calculation.

Circles could not simply be represented in the application as S-expression versions of the calculations they stood for, as they are not defined solely by their expressions.[1] Data were needed regarding their position on the canvas, so it was decided to use an associative map to represent a circle, containing co-ordinate data as well as the calculation S-expression.

Originally, the program was to use two macros, `defcircle` and `letcircle`, to be able to define circles at runtime. It became apparent that circles defined in this way were not accessible by the interaction handler whilst the program was running. To solve this problem, 'name' data was added to the circle format, and a mutable data structure was chosen to manage a list of circles currently in use. This allowed the application to access and modify the circles as needed, and also to discard unused circles.

3.1.1.1 Management of Mutable State

Nico's infrastructure stores and manages circles and questions using just six mutable data structures:

- The list of circles, as detailed above
- The current question set: a list of maps containing the question number and the S-expression that constitutes the question to be answered
- An integer counting how many circles have previously been created
- A map containing the current co-ordinates of the mouse cursor
- An integer indicating the number of the question currently being answered
- A temporary store for information about circles that are currently being repositioned by the user

[1] "S-expression", meaning "symbolic expression", refers to nested list data structures that are used in LISP and its derivatives to represent both source code and data. S-expressions are defined by McCarthy as a single atom, or as an expression of the form `(a . b)`, where `a` and `b` are S-expressions, thus creating a tree structure [20].

Mutable data structures were chosen as these elements needed to be continually updated whilst the program was running, and having so few is advantageous. By reducing the number of elements able to change during the execution of the program, the application is made more stable. The structure of the application is also easier to test and to comprehend: functions called upon immutable objects will not vary in their output, thus those parts of the program that do not interact with the mutable elements behave predictably.

3.1.1.2 Circle Management

Two main functions for the management of circles have been implemented: creation and deletion.[2] New circles are initialised such that they are centred on the current co-ordinates of the mouse cursor, with a name of the format **"**`c`*n***"**, where *n* is the number of circles previously created. Upon creating a circle, its expression is initialised to `(\? \? \?)`, a meaningless expression using the question mark character `\?` as a placeholder value for both the operator and the two arguments.[3]

Placeholder expressions are used so that the user is able to create a circle without having to specify parameters for it beforehand, which would entail *premature commitment*, as well as the *hard mental operation* of having to visualise a circle before committing to creating it. Such an approach was exhibited with a previous revision of the software, using a complex dialogue box (*Fig. 3.7*).

It was also decided that the circle should use explicitly **placeholder** values, rather than an expression of value 0 (such as (0+0)) so that new circles did not affect the current total displayed by becoming the new root circle (see *Sec. 3.1.1.3* for a full discussion of diagram evaluation). The placeholders also indicate that action is needed, whereas an expression of value 0 could appear to be a deliberate inclusion in the calculation.

Deletion removes a circle from the list by searching through the main circle list and returning a version which lacks the circle matching either the name or the co-ordinates specified. The circle list is then updated.

This naïve implementation of deletion led to many problems: it does not deal well with circles that are used as arguments to other circles. When such a circle is removed from the main list, references to it remain in the circles that were using it,

[2]Editing of circles is accomplished by deleting a circle and recreating a modified version of it.

[3]In this dissertation, the placeholder question mark character is represented as `\?` when referring to the question mark character literal in Clojure code, and by `?` when referring to the question mark that is displayed for placeholders in the application.

causing the application to throw `java.lang.NullPointerExceptions`. As such, an alternative version of deletion was implemented, to preserve referential integrity.

The alternative 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 do, the function determines if that circle is a root circle (*Sec. 3.1.1.3*). If so, the root circle is removed using the previous method and replaced, with null references changed to placeholders. If the circle is non-root, the deletion function recursively calls itself upon the circle, thus removing all circles between the circle to be deleted and the root. The circle can then be deleted using the original method, and the intermediate circles are replaced.

The alternative method of deletion solved the problem of maintaining referential integrity, but it has also given rise to a new problem: as the alternative 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 in the chain between the target and the root. On reflection, a better way to implement deletion would have been to make the alternative deletion function inspect the main circle list after deletion, checking for any invalid references and replacing them with \?s, rather than doing so 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 S-expression.

The root circle is the one circle in a finished diagram that is not used as an argument to any others. 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 other circles refer to it. As stated, in a **finished** diagram, there can only be one root circle, so this method is sufficient for use in marking answers.

It is possible that the user may construct two or more calculation structures in parallel, intending 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 encounters. It would have been possible to, for example, add an extra answer display for each root and to maintain a total for each, but it was decided that this would be confusing, as the user may not expect the structure of the user interface to change. This would also require reintroducing a labelling system for circles and structures, which was abandoned as it is an unfamiliar concept to non-programmers and those without algebra (the majority 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, the references to other circles can be resolved, creating a nested S-expression that can have Clojure's `eval` called on it to return an answer.

This was, in fact, a significant reason for choosing Clojure; 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 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 display (*Sec. 3.1.2.2*), can be performed.

3.1.1.4 Question Management

Sets of questions are written as external, plain-text files contaning S-expressions corresponding to questions, separated by newlines. This approach was chosen so that questions can be read into a list using Clojure's standard library, meaning that it is possible to navigate between them by setting 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 problems into S-expressions.

3.1.2 Interaction Handler

The interaction handler forms the 'middle layer' of the application (*Fig. 3.1*) that arbitrates between the infrastructure and the user interface.

The rendering function is called every time the mouse is moved, or when the canvas needs to be updated. It paints a white rectangle over the canvas area, draws a bin icon in the corner (see *Sec. 3.2.2.2*) and iterates across the circle list, drawing corresponding circles. References to other circles are visualised as lines extending from the source circle to the appropriate argument slot on the recipient. By re-rendering every time the mouse is moved, rather than continually, the application does not render when the canvas is not changing. 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 the primary development environment. Installation was attempted on a Windows machine, but caused a number of problems. It was decided to proceed using the Eclipse Foundation's SWT toolkit and Szymon Witamborski's corresponding Clojure bindings [30].

During the development of the user interface, many problems were encountered with SWT, particularly regarding drawing arbitrary shapes on the canvas and extracting user input from dialogue boxes. To keep the project on-schedule, the application was migrated to Java Swing, using the *Seesaw* library to provide bindings for Clojure [23]. Development efficiency was increased as a result of the author's previous experience using Swing.

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 displayed and responds to certain conditions. It listens for certain events (i.e. user input), and prompts the user for input when required. Seesaw requires interfaces to be structured such that each window is defined as one large structure, with appearance, listeners and other parameters set as options. Other structures, such as panels and canvases, can then be used as content.

Embedded into the main window are several mouse listeners. While the application is running, five mouse-based events are listened for: motion, dragging, button presses, button releasees and button clicks.[4] When motion is detected, the canvas is cleared and re-rendered. Motion also triggers a check to determine if the mouse is over a circle; if so, a highlighting function is called, which draws a ring around the circle, and also performs a string comparison. Using the same function that converts 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 allow the user to check their work. It was decided to implement this

[4]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 [8].

as a string, rather than S-expression, comparison as the string search is able to return character indices in the question string, indicating the substring to highlight.

3.1.2.3 Control Scheme

Upon detection of a mouse click, the software determines the 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 which mouse button was pressed and any modifier keys that were active, are used to determine the correct action to take, as illustrated in *Fig. 3.2*.

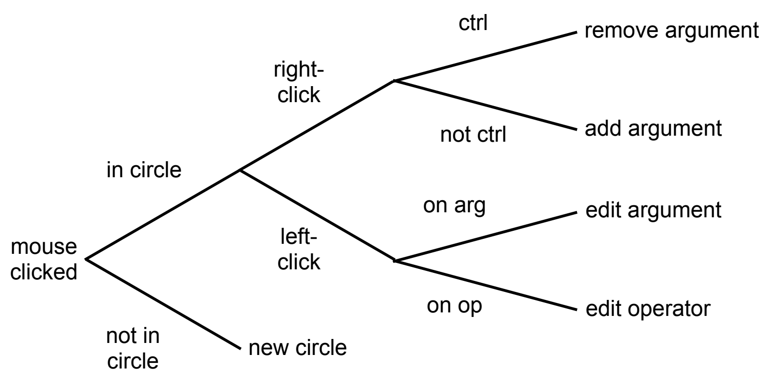


Figure 3.2: Decision tree to determine the appropriate response to a mouse click event.

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, using a spinner to set its value.
    * Mouse on operator

      · A small blue circle is drawn around the operator and a dialogue box is launched, using four radio buttons to set its value.

- Right-click
  * Ctrl key pressed
    · The final argument is removed from the circle. There is a minimum of two arguments: an alert box warns the user if they try to use fewer.
  * Ctrl key not pressed
    · A placeholder argument is added to the circle. There is a maximum of eight arguments: an alert box warns the user if they try to use more.

Drag-and-drop functionality is also included in the 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 is stored with the current system time and the button that was pressed in a mutable data structure. When a drag event is detected, that information is then used to perform one of two operations upon the circle.

If the left mouse button is held whilst dragging, the circle's co-ordinates are updated to the current co-ordinates of the cursor. If the right mouse button is held whilst dragging, a line is drawn from the circle to the current location of the cursor. 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 is now a reference to the source circle.

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 when the mouse is pressed, the function called upon dragging can use and modify this information easily. This does not impact upon other mouse events involving presses and releases (e.g. clicking), as this information is discarded upon every mouse release, regardless of whether or not it was being used.

The control scheme is almost entirely mouse-based as the mouse is a simpler interface than the keyboard, and is familiar to most computer users in a school environment, regardless of ability (the use of applications that require mouse control, such as many word processors, is mandated by the National Curriculum). By limiting control to the mouse, the user is required to remember fewer button locations, and is likely to be able to guess controls correctly if forgotten. A caveat to this is that the use of the Ctrl key is required, as this was preferable to mouse chording (unfamiliar to novices), or to requiring a three-button mouse (not always available).

The design of the control scheme is discussed further in *Sec. 3.2.2*.

3.2 User Interface

The user interface comprises the parts of the application that 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.

During its development, *Nico*'s user interface went through a number of revisions, which are detailed below.

3.2.1 Application

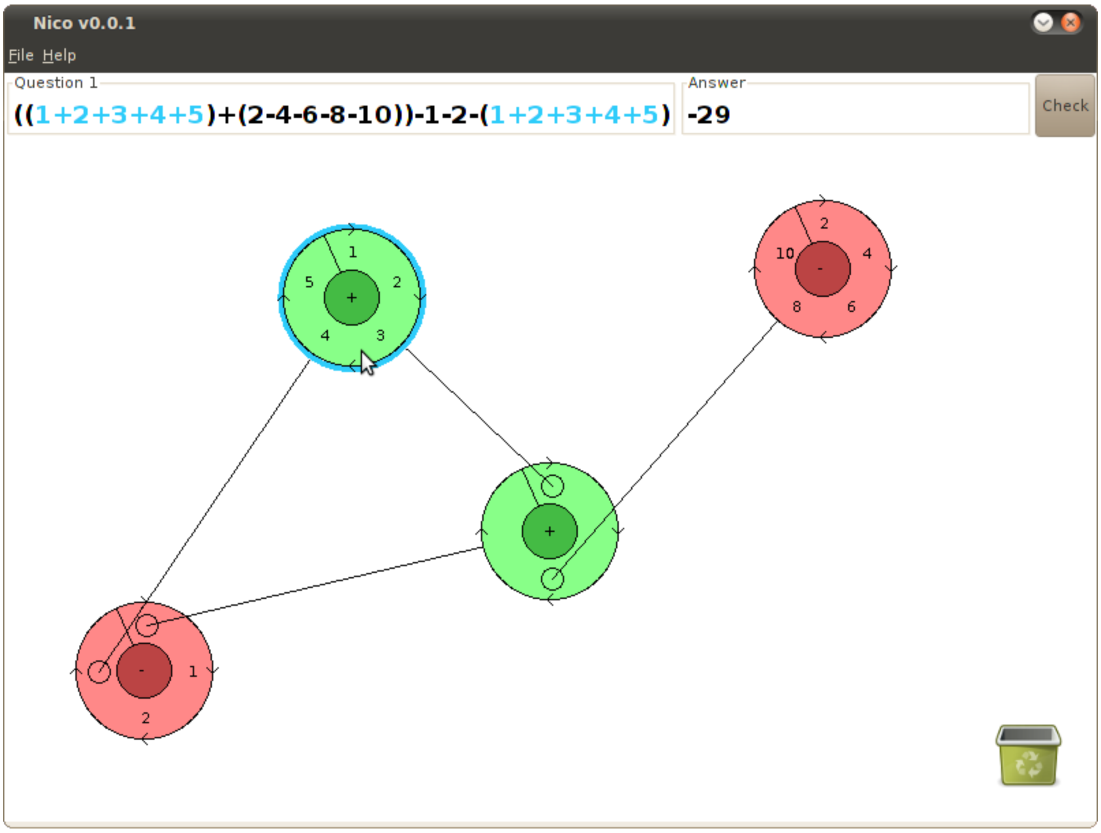


Figure 3.3: A screenshot of a *Nico* session.

The main application window has two main sections: the information panels that display the question and the current value of the calculation, as well as the `Check` button to submit the current total as the answer; and the canvas, upon which the user constructs diagrams. This canvas occupies most of the main window (*Fig. 3.3*), and the overall design of the application is minimal, with the aim of emphasising the diagram above all else.

3.2.1.1 Initial Revision and Control Buttons

Initially, the application was intended to use more buttons (*Fig. 2.12*), having more controls immediately visible to the user.

In the first usable version of the interface, there were a number of buttons that appeared to the right of the canvas (*Fig. 3.4*). These were relocated to save screen estate for the diagram. Each circle, at this point, was required to be labelled. Each button would cause a dialogue box to appear, which required a circle to be specified by name before any operations could be performed.

This design was abandoned as the separation between the user and their work was too great; creating circles by specifying options in 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. Forcing the user to commit to a design before even seeing it entailed *premature commitment*, and having to re-express circles to edit them greatly increased *viscosity*.

3.2.1.2 Context Menus and Automated Marking

Later revisions of the software removed the buttons and implemented a system of context menus with which to construct answers, further increasing the space available for the user's answer, making the consequences of actions more evident. For example, right-clicking on a circle and seleting 'Delete', rather than choosing to delete a circle and specifying the target's name. Hence, the user was able to interact with the canvas itself. The many dialogue boxes were combined into one (*Fig. 3.7*), which was used for both creation and modification.

Answer-checking was also automated: when the total reached a value equal to that of the answer to the question, the user was immediately moved on to the next question. The intention here was to streamline the user experience, increasing the speed with which question sets were completed.

This design was abandoned as, again, it separated the user from their work. Although this design was an improvement, allowing the user to interact directly with the canvas to perform operations rather than by specifying an action and name, the dialogue box was too complex, and still entailed *premature commitment*. The automatic answer-checking was also a problem, as it did not give the user a chance to review their answer before proceeding. The user could not learn from their mistakes, for example if they had accidentally reached the right answer as part of a larger, incorrect calculation.

3.2.1.3 Final Revision

The final revision of the software reintroduced the `Check` button to address the issues with the automatic answer-checking in the previous version. The system of context menus was replaced with a new control scheme (*Sec. 3.2.2*).

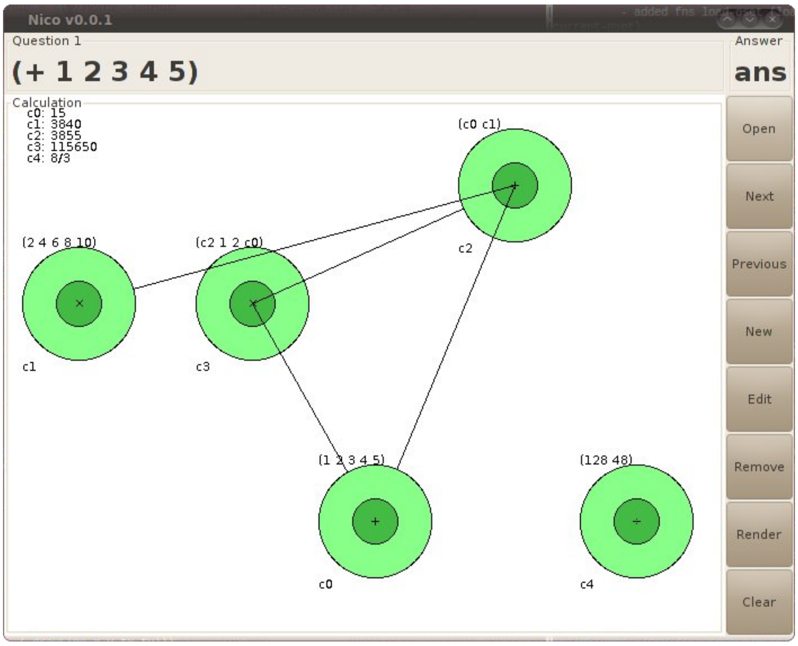


Figure 3.4: A very early version of the user interface. Many features are yet to be implemented, but the button-based layout and the early revision of the notation is evident.

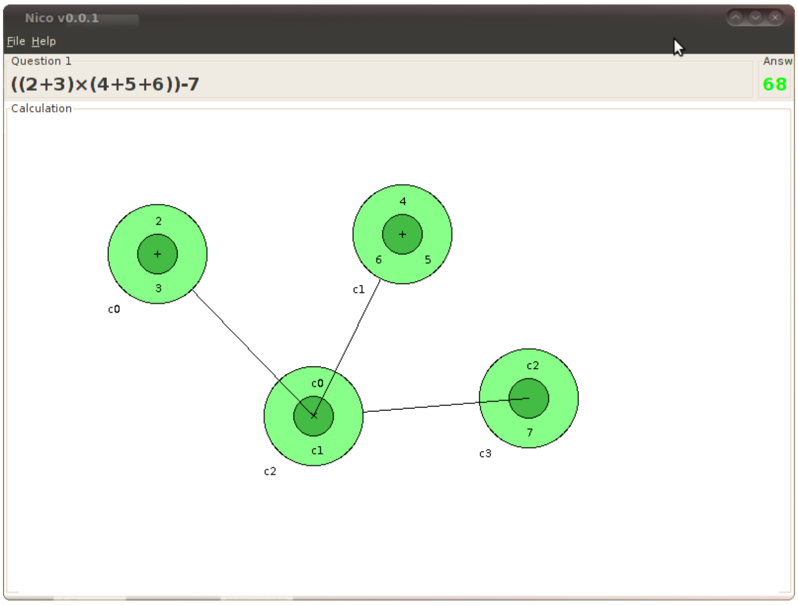


Figure 3.5: A later revision of the user interface. The screen has been cleared, allowing context menus to replace buttons. The facility for checking an answer has been removed, due to its automation in this version. The notation has now been fully implemented, but still lacks many features of the final version.

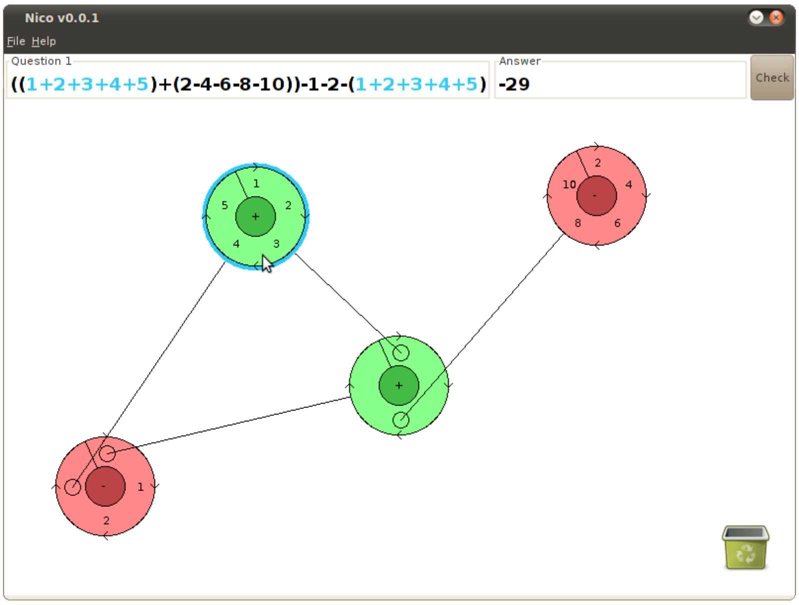


Figure 3.6: The final revision of the user interface. The layout is similar to the previous version, but now includes highlighting and a bin icon for deletion. The notation is now colour-coded, and the answer-checking button has been reintroduced.

3.2.1.4 Dialogue Boxes

Dialogue boxes have been a consistent feature of the user interface throughout development. Initially, dialogue boxes were triggered by the buttons to the right of the canvas (*Fig. 3.4*), and consisted of a text field, into which the user typed a circle's name, followed by, in the case of the creation and modification, an S-expression specifying the calculation that the circle was to represent.

This was never intended to be the final version, but was useful for testing the rest of the user interface and interaction handler at the time. Requiring a user to be familiar with LISP syntax ran somewhat counter to the intentions of this project, being to provide an accessible alternative notation for arithmetic for the non-programmer, pre-algebra user.

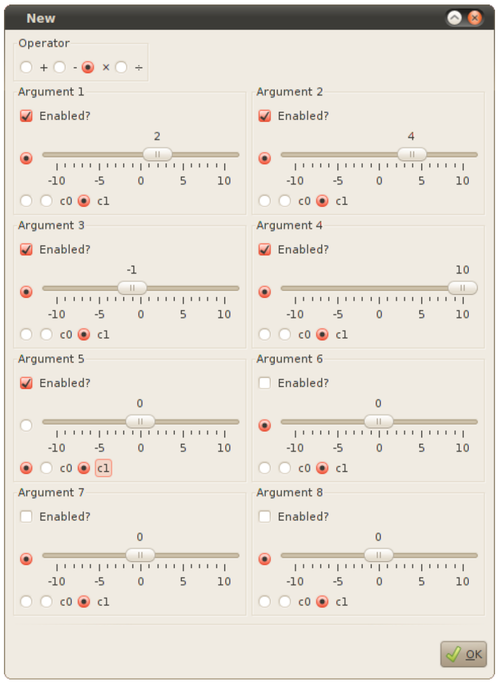
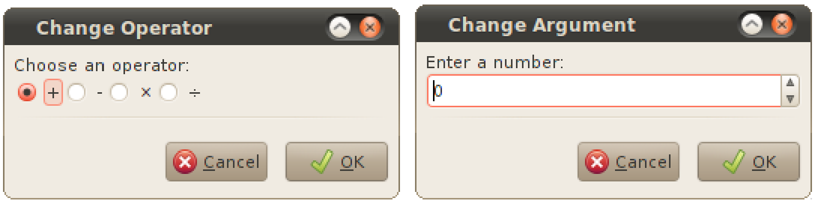


Figure 3.7: The unified dialogue box.

In later revisions of the software, as mentioned in the previous section, the many dialogue boxes were combined into one, with many options available to specify the parameters of a circle. This decision was made with the intention of streamlining the user experience by providing all available controls in one location, dynamically generating its layout to make new circles available as arguments (*Fig. 3.7*). However, the unified dialogue box still required the user to perform operations without the consequences being obvious. This required unnecessary *premature commitment* from the user. Additionally, the act of visualising the circle whilst setting the many options available in the dialogue box constituted a superfluous *hard mental operation*. Finally, the unified dialogue box required **more** time to search for information than several simpler dialogue boxes, in which less information is presented at once.

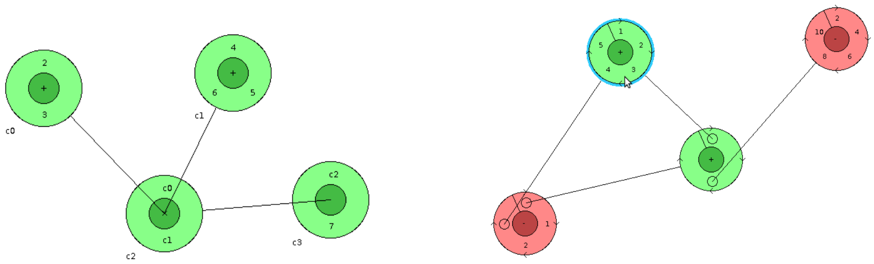
In the final revision of the software, the unified dialogue box was reduced to two much simpler ones (*Fig. 3.8*), used in combination with the new control scheme. The large dialogue box was unwieldy and dense, requiring the user to search for controls within it and increasing the *viscosity* of the notation by requiring all parameters to be set every time one needed to be edited. Two simple dialogue boxes serves to greatly decrease this *viscosity*, making it so that single arguments or operators could be edited independently of the rest of their circles. In combination with the improved control scheme, this helped to eliminate much of the *premature* *commitment* of previous designs, by making the creation of a circle an incremental process, rather than a single event.



(a) Editing a circle's operator (b) Editing a circle's argument

Figure 3.8: The dialogue boxes used to modify existing circles in the application.

3.2.2 Notation



(a) An old version of the notation. Circles are uniformly coloured, and links feed into the centre of their target circles. There is no indication of argument evaluation order.

(b) The final version of the notation. Circles are colour-coded by operation, highlighting has been implemented and links fill argument slots, ending in a circle to indicate their destination. The diagonal line and arrows have been added to circles to indicate evaluation order.

Figure 3.9: The notation used went through a number of revisions before reaching its current state.

The notation itself was revised several times. The original vision of the notation can be seen in *Fig. 2.12*. It was decided to move away from this model: the 'tails' occupied too much screen estate, and were thus replaced with simple lines, allowing many linked circles to be closer together but still *visible*.

3.2.2.1 Early Revisions

The first implementation of the notation consisted of uniformly-coloured circles connected by lines feeding into the centre of the recipient circle. This was so that links to circles would not restrict further the number of available arguments per circle. A colour scheme had not yet been implemented.

Placeholder values were unnecessary as the dialogue boxes meant that at no point could a circle have any aspect of itself undefined. There was also no indication of argument ordering, though it was defined. This was only acceptable when the operator was commutative, but became confusing for operations where this was not the case. The user was required to determine the order through experience, which was counter to the intentions of this project. 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 links between circles. With all links pointing to the operator at the centre of their targets, it was not at all clear in which order the input circles evaluated, neither relative to each other nor to the other arguments. This was problematic in situations where one circle was used as an argument to several others.

3.2.2.2 Final Revision

The design of the notation was changed to address these problems. Firstly, the circles are now colour-coded by operator, making it easier to distinguish between different kinds of circles. The circles themselves have also been embellished with the addition of a line to the left of the first argument, denoting its place at the top of the circle. 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 to occupy an argument slot that could otherwise be taken by a number, with a marker on the end of the link line to clearly show where it terminates.

3.2.3 Control Scheme

3.2.3.1 Development

Initially, the application used a collection of buttons and dialogue boxes, as detailed above, to control the circles that appeared on the canvas. This was implemented to ensure that the user interface was functional, and so that circle operations could be tested. It was abandoned as it required too much *premature commitment* from the user, who would have to consider the consequences of performing an action before it was carried out.

In later revisions, a system of context menus was implemented to decrease the separation between the user and their work. 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. This required less forethought on the part of the user, but the use of the unified dialogue box (*Fig. 3.7*) still entailed *premature commitment*.

The control system is, in its final revision, predominantly mouse-based for the construction of calculations in the application's notation.

3.2.3.2 Control Design

Left-clicking on a blank patch of canvas creates a new circle, initialised with a placeholder expression. This approach was chosen 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, as detailed in *Sec. 3.1.2.3*.

Left-clicking on a circle's operator brings up a dialogue box with a radio button for each operator (*Fig. 3.8*). Similarly, left-clicking on a circle's argument brings up a dialogue box containing a spinner to set a new value for that argument, between -10 and 10. The spinner is limited to this particular range to prevent the application from being used as a 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 using such small numbers, as it is assumed that the target audience is familiar with very simple arithmetic.

Right-clicking on a circle increments its number of arguments by one, to a maximum of eight. Holding the Ctrl key and right-clicking on a circle decrements its number of arguments by one, to a minimum of two. The user is warned by an alert box if these boundaries are exceeded. The limit of two to eight arguments was decided upon as fewer than two arguments is an unfamiliar concept that is rarely of use to 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 hard to read.

Circles can be repositioned by left-clicking and dragging them to the desired location. If a circle is dragged to the bin icon in the bottom left-hand corner of the screen, it is removed. Originally, deletion was mapped to a mouse control, but to simplify the control scheme, it was decided that an icon should be used instead. This also has the advantage of demonstrating to the new user that mistakes can be erased, encouraging exploration.

To use one circle as an argument to another, right-clicking and dragging allows the user to draw a line to the desired slot on the recipient circle. By having the user pull links out from existing circles, as opposed to specifying references and links consequently appearing, the direction of dataflow is emphasised.

This new control scheme allows the user to directly manipulate their work in progress, with each action having clear consequences. It reduces the *viscosity* of editing circles from the previous control schemes that used buttons and context menus, and greatly reduces the *premature commitment* required when creating circles. The environment encourages exploration, and does not require the user to spend time searching for information within parts of the interface.

3.2.4 Colour-Coding

Nico uses colour-coding in various areas of the application, to draw attention to important elements. Colour-coding has been shown to faciliate more effective learning. Lamberski and Dwyer conclude that, for younger learners,

> "color in passive materials has been found to facilitate performance in less complex concept attainment tasks because of color's attention or 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." [16]

Nico's colour scheme was implemented incrementally over the course of the project. It was originally intended to be based on how nested a circle was (*Fig. 2.12*). This was rejected on the grounds that it could become confusing if circles were reused (see *Fig. 3.10*). Consequently, circles are coloured according to their operators, clearly separating different kinds of operations and making the notation more *visible*.

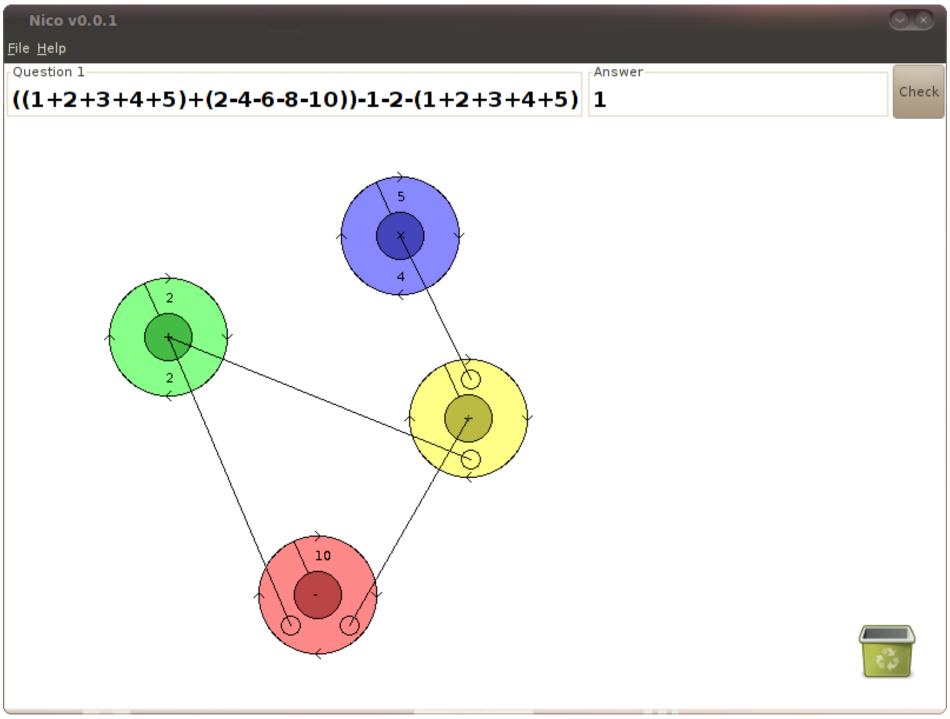


Figure 3.10: An example of where nesting-based colour-coding would be confusing. The circle containing (2+2) can be considered to be both one and two circles removed from the root circle.

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 (see *Fig. 3.11*). This helps the user to assess their progress towards the goal of solving the question, and also allows them to focus their attention upon a single circle.

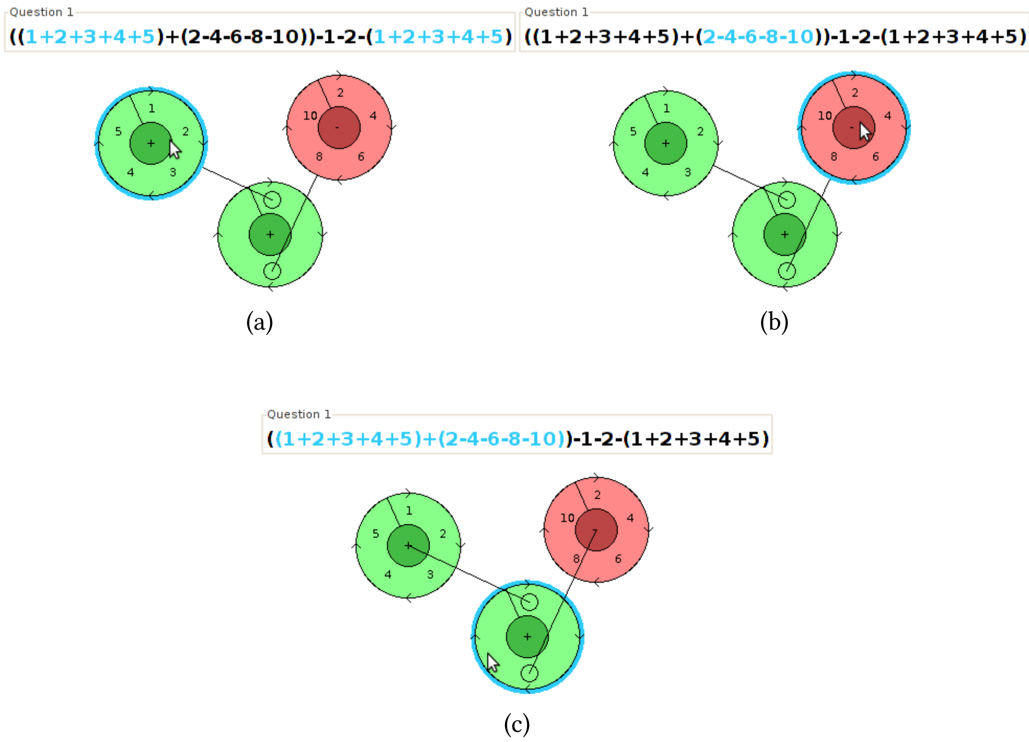
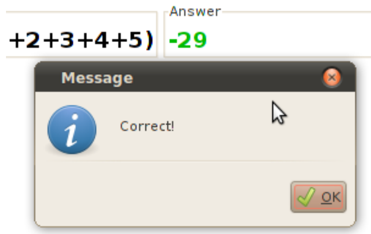


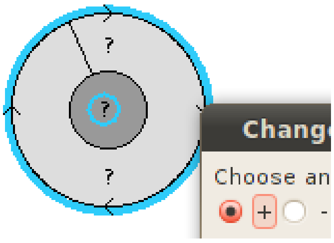
Figure 3.11: Three instances of the question text being highlighted as a corresponding circle is moused-over.

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 if their answer is wrong.

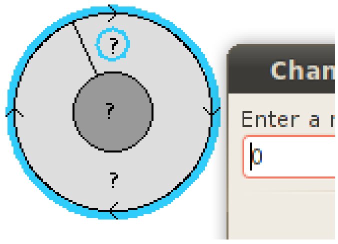
(a) Red text indicates that the answer submitted was incorrect



(b) Green text indicates a correct answer; a dialogue box allows the user to progress to the next question when they are ready



(c) The circle's operator is highlighted when it is being edited



(d) An argument is highlighted when it is being edited

Figure 3.12: Examples of colour-coding in *Nico*.

3.2.4.1 Colour Blindness

As *Nico*'s notation and user interface are reliant upon colour to express or emphasise information, colour-blind users are liable to find some parts of the application inaccessible. Although no functionality to address this was included in the application itself, the colours are easily replaced in the source code. They are defined early on as variables, and are referred to throughout. By changing the definition of each colour variable, it is possible to create customised versions of the software to suit individual cases of colour blindness. For example, a person with total colour blindness may need a version of the software that uses shades of grey, rather than colours, people with other forms of colour blindness, such as protanopia or tritanomaly may only need to replace certain colours in the colour scheme.

3.3 Testing

Development of the software was primarily REPL-driven, allowing for rapid development and testing whilst the application was running. A suite of unit tests were derived from assertions made in the REPL, using Leiningen’s testing framework and the `clojure.test` library. These constituted a number of tests containing several assertions that were able to be run all at once by using the command `lein test` in a terminal. This was advantageous as it allowed for quick assessment of many functions within the software.

Unit tests were performed on many infrastructure functions, especially those that did not require access to mutable state. Such functions behaved very predictably; as pure functions with no side effects and being called upon immutable data, once unit tested, these did not require further testing. An example of a unit test is shown in *Fig. 3.13*.

```
(deftest lisp-maths
  (is (= (lisp-to-maths ’(+ 2 3)) ”2+3”))
  (is (= (lisp-to-maths ’(- 2 3)) ”2-3”))
  (is (= (lisp-to-maths ’(* 2 3)) ”2×3”))
  (is (= (lisp-to-maths ’(/ 2 3)) ”2÷3”))
  (is (= (lisp-to-maths ’(+ 19 28 30 1 -4)) ”19+28+30+1+-4”))
  (is (= (lisp-to-maths ’(+ (- 2 3) (* 4 5))) ”(2-3)+(4×5)”))
  (is (= (lisp-to-maths ’(+ 3 4 (- 2 1) 1)) ”3+4+(2-1)+1”)))
```

Figure 3.13: A unit test taken from the test file. Several assertions are made within the body of a test. The command `lein test` causes all such tests to be run, providing the developer with the number of passes and failures, and details of any failures that occurred.

Integration testing was required for larger components, and especially those that accessed mutable data structures. The REPL-driven nature of development meant that it was possible to evaluate certain statements while the application was running, for example to compare the updated value of a circle after a modification operation (*Fig. 3.14*).

```
nico.core> (:circ (find-circle ”c0”))
(#<core$_PLUS_ clojure.core$_PLUS_@24de1f47> 1 2 3)
nico.core> (remove-last-arg (find-circle ”c0”))
[redacted: output stating successful update of agent]
nico.core> (:circ (find-circle ”c0”))
(#<core$_PLUS_ clojure.core$_PLUS_@24de1f47> 1 2)
```

Figure 3.14: A REPL-based integration test for a function to remove the final argument of a circle. The expression contained within a circle is checked, the removal function is called, and the expression is checked again. The effect is visible; the final argument has been removed.

Testing was made much easier by the small number of mutable data structures. Fewer integration tests were required, as after it was established that a pure function that did not interact with mutable objects behaved as expected, it could be assumed that it would behave as expected outside of the unit-testing situation.

3.4 Summary

The software fulfils and exceeds the criteria for success outlined in the project proposal, being a complete and usable system that improves upon the traditional method of handwritten arithmetic.

Third-party libraries, in particular Seesaw, were used where appropriate.

The application comprises three sections: the infrastructure, handling mathematical operations and memory management; the interaction handler, acting as an interface between the UI and the infrastructure; and the user interface itself.

Mutable state was introduced where necessary, but kept to a minimum. The application uses just six mutable data structures, meaning that functions not interacting with these objects will behave predictably, making the software stabler, and easier to comprehend and test.

The user interface was revised many times to correct deficiencies in the design that would have led to a poor user experience, and the infrastructure and interaction handler were also both restructured as new demands upon the software were made.

Chapter 4

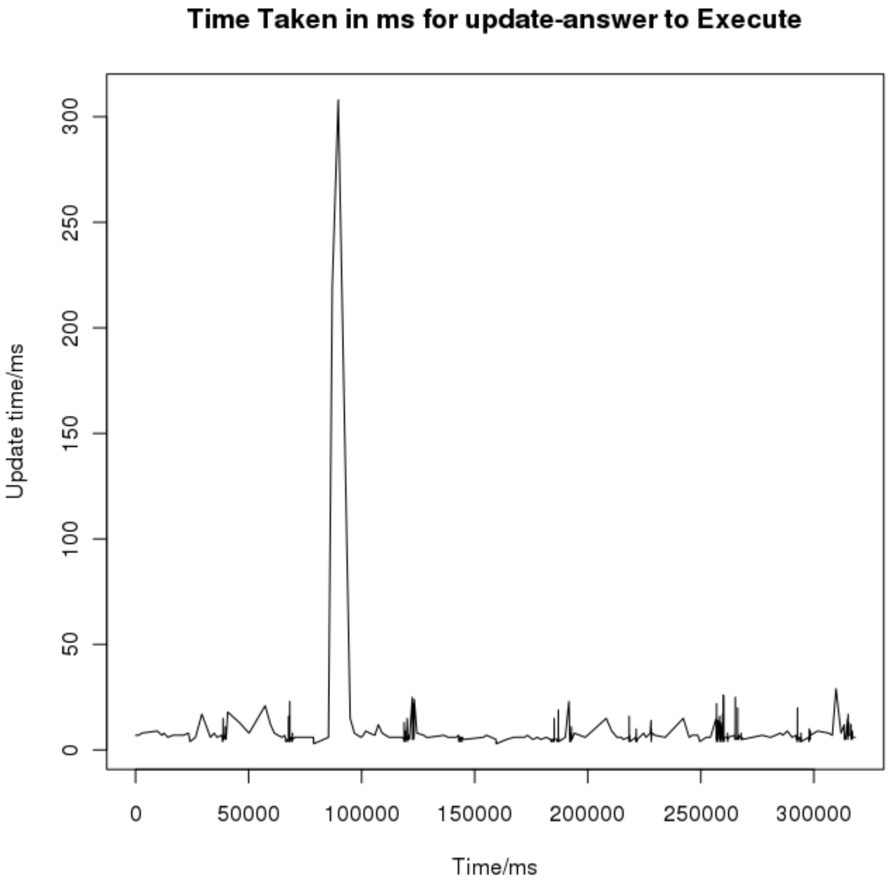
Evaluation

This chapter details the results of tests carried out to evaluate the software. First is an analysis of the time taken to update the display, followed by the results of a user study.

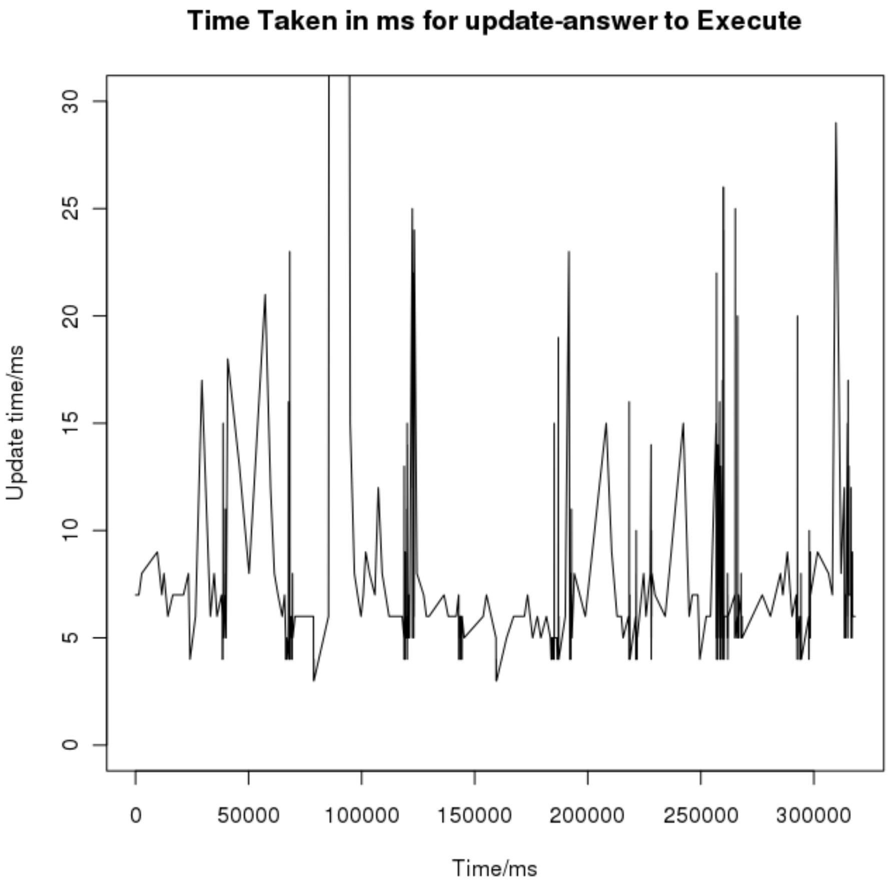
4.1 UI Evaluation

The time taken to generate an S-expression, evaluate it and display the result to the user, implemented in the function `update-answer`, was required by the project proposal to be less than 300ms. To test this, a *Nico* session was completed, using the same questions as the user study (below), but with a modified version of `update-answer` that recorded the system time upon entering and exiting the function in an external log file.

A plot of the time taken for each call to `update-answer` to complete over time is shown below.


Time Taken in ms for update-answer to Execute
Update time/ms
0
50
100
150
200
250
300
0
50000
100000
150000
200000
250000
300000
Time/ms

(a)


Time Taken in ms for update-answer to Execute
Update time/ms
0
5
10
15
20
25
30
0
50000
100000
150000
200000
250000
300000
Time/ms

`update-answer` was, in almost all cases, able to complete in significantly less than the upper limit of 300ms, with a mean time of 6.751852ms. Of the 810 samples taken, there was one anomalous time of 308ms.

4.2 User Study

To assess the utility of the software relative to handwritten arithmetic, a user study was conducted, allowing users to test the system and provide feedback. Users were given a tutorial video and a questionnaire (*Appendix C*) to complete whilst using the software.

4.2.1 Experimental Design

Originally, it was the intention of this project to perform a user study upon a 'test class' of Year 5 pupils, but working with vulnerable individuals involved ethical complications that were diffcult to overcome, 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 to be signed by the pupils' guardians and returned via the school to the author. It was therefore decided that students not currently reading a 'mathematical' subject, such as the Computer Science Tripos which necessitates a high level of mathematical aptitude and experience, were suitable alternatives.

Participants were first asked to sign a statement of informed consent to participate in the study, and were asked if they had previously used *Nico*. They were required to indicate their confidence in their ability to solve simple mathematical problems on a five-point Likert item. The meaning of each point in the Likert item was explained to the users beforehand, as it is well-established that people tend to overestimate their abilities [21].

Users were divided into two groups. The first watched the tutorial video and explored the application in a 'sandbox' environment for five minutes. They then completed one question set using *Nico*, and then answered the same questions by hand. The second group completed the same questions by hand, before watching the video and using the software in the same way as the first group. This reversal was to eliminate any advantage exhibited by having completed the questions once, before doing so again using a different method.

The time taken to answer each question was recorded. Functionality was implemented to log the system time at significant points in the user's progress using *Nico*, and the handwritten questions were timed using a stopwatch.

Both groups were then asked to complete a series of questions to provide feedback on the software, using questions from Blackwell and Green [4].

This approach was chosen for a number of reasons. *Nico* was compared to questions answered by hand, rather than a calculator, as its intention is not to calculate answers for the user, but to be used as a technique of visualising method. To compare *Nico* to questions answered using a calculator would not be a useful comparison, as the use of a calculator entails different methods to those that would be used when calculating by hand.

The decision to use two methods of evaluation of the software (timing users and the 'Cognitive Dimensions' questionnaire) allows several aspects of the software to be assessed. By timing users' responses to questions both by hand and using the software, the time taken by the participant to think about how to answer a question is gauged. It is expected that *Nico* should decrease the amount of time taken to work through complex problems involving several subcalculations.

The 'Cognitive Dimensions' questionnaire also allows users to critique the design of the user interface, gauging how well the software is received. With regard to the design of the questionnaire itself, it was decided to use a Likert item to assess the users' initial level of confidence in their arithmetic ability as this provides a simple approach that is quantifiable, and thus is analysed more easily than a verbal response.

A tutorial video was decided upon, rather than a textual approach or a live demonstration, as it provides a means of demonstrating an example session in a standardised way. In a live demonstration, important elements could potentially be omitted by the demonstrator. A textual description may not make certain processes clear, and may use unfamiliar terminology.

4.2.2 Pilot Study

A pilot study was conducted with one participant, 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 participant noted that the original version of the tutorial video was hard to follow; it used subtitles to convey information, which were felt to be hard to read and understand in the time given. The participant suggested using a voiceover instead.

The participant also noted that some of the questions in the feedback section were confusing or irrelevant to this particular application, and was unsure of how to answer them.

Finally, the participant felt 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.

This feedback was acted upon. The video was reproduced using a voiceover rather than subtitles. Irrelevant questions were removed from the feedback section. 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.

4.2.3 Results

The results of the tests are shown in *Figs. 4.2* and *4.3*, illustrating the times taken the participants in each group respectively to answer the questions both by hand and using *Nico*.

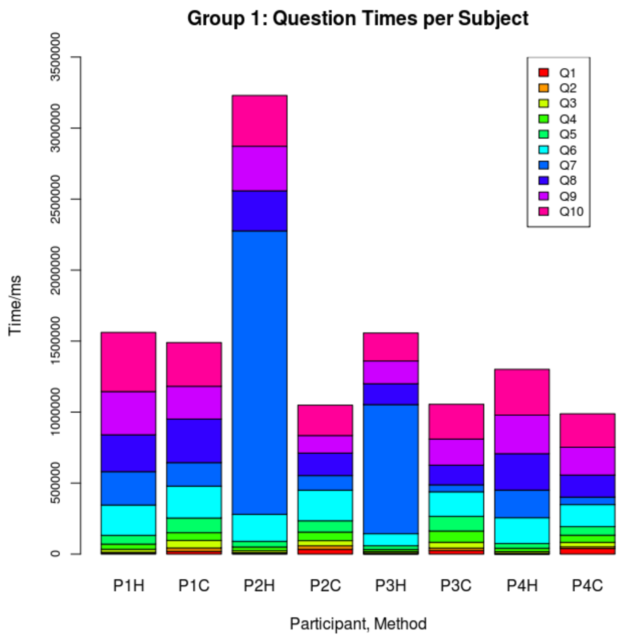


Figure 4.2: Time taken in milliseconds for each participant in Group 1 to complete the questions, both by hand and using the software. Names are formatted as Pn[HC], where Pn is Participant n, H denotes Handwritten and C denotes Computer.

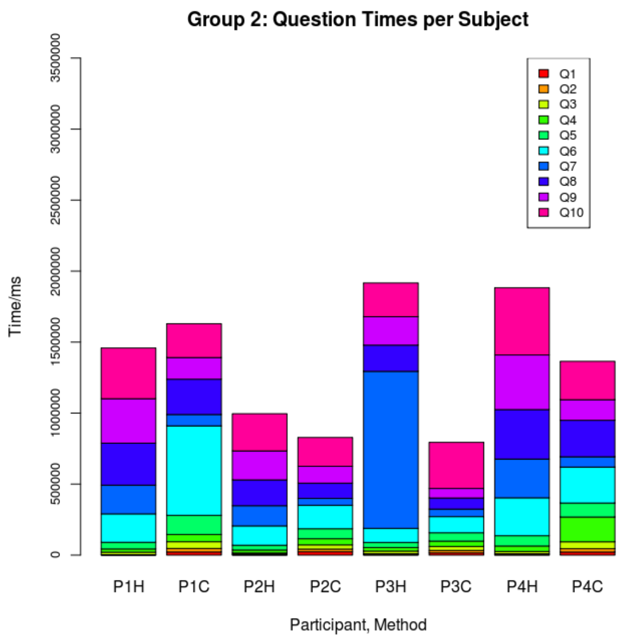


Figure 4.3: Time taken in milliseconds for each participant in Group 2 to complete the questions, both by hand and using the software. Names are formatted as in *Fig. 4.2*.

This data can be visualised using a series of plots, as below. The complete set of plots and full table of times can be found in *Appendix B*.

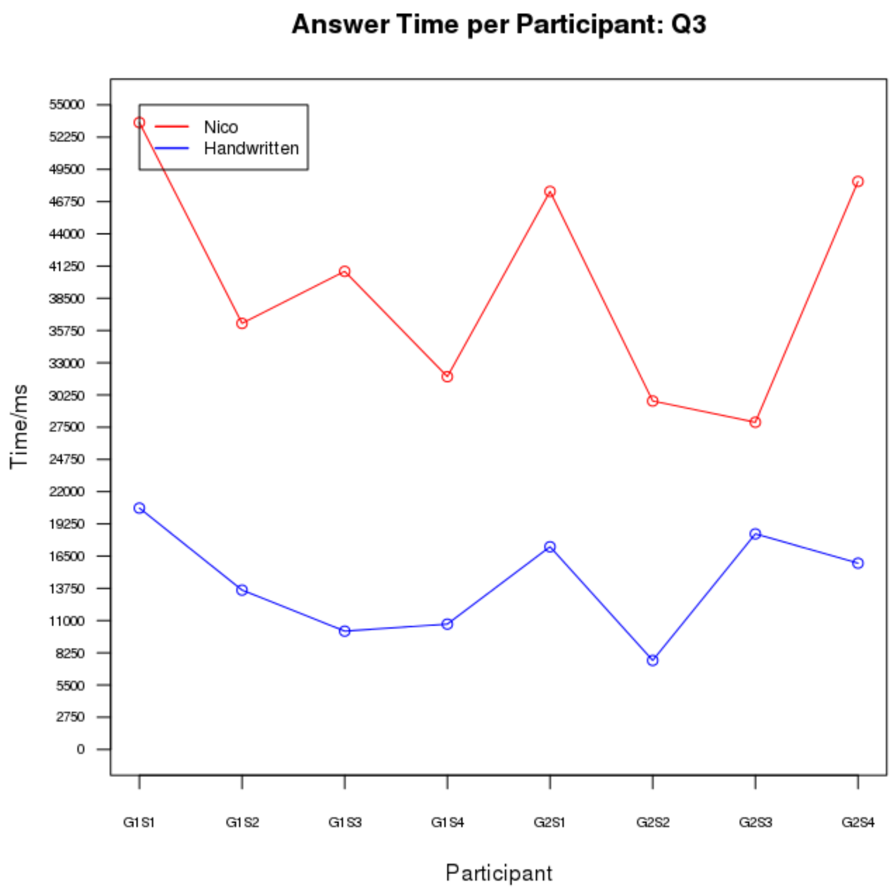


Figure 4.4: Time taken in milliseconds per participant to answer Question 3: 1+2+3+4+5.

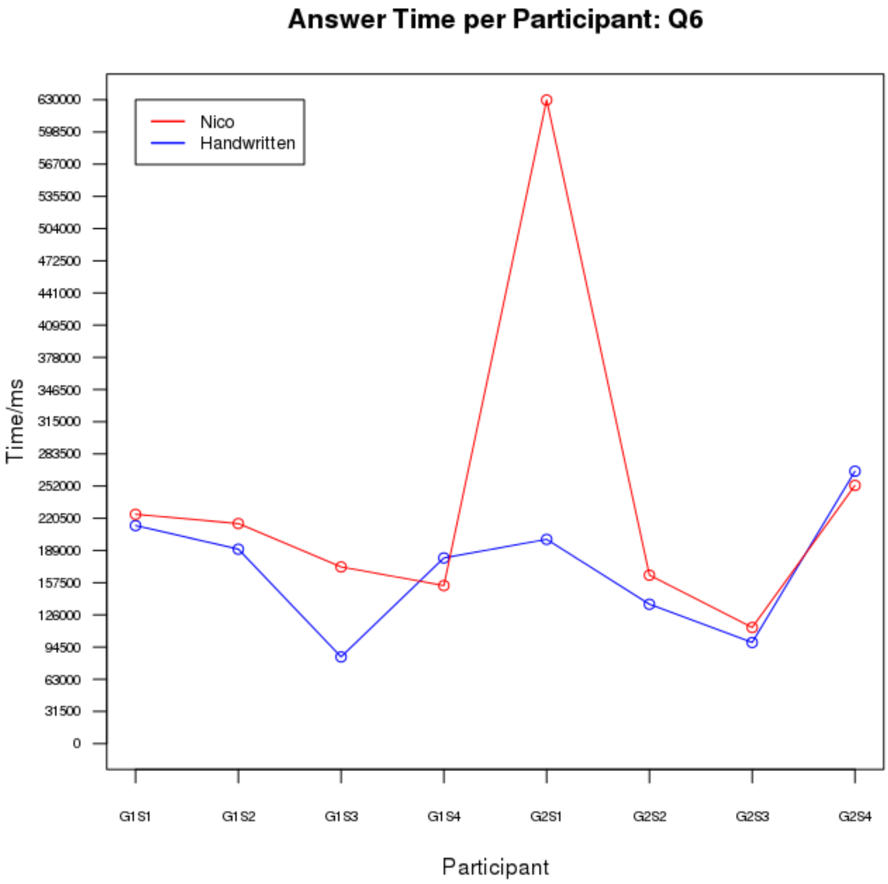


Figure 4.5: Time taken in milliseconds per participant to answer Question 6: $((1+2+3+4+5)+(2\times4\times6\times8\times10))\times1\times2\times(1+2+3+4+5)$.

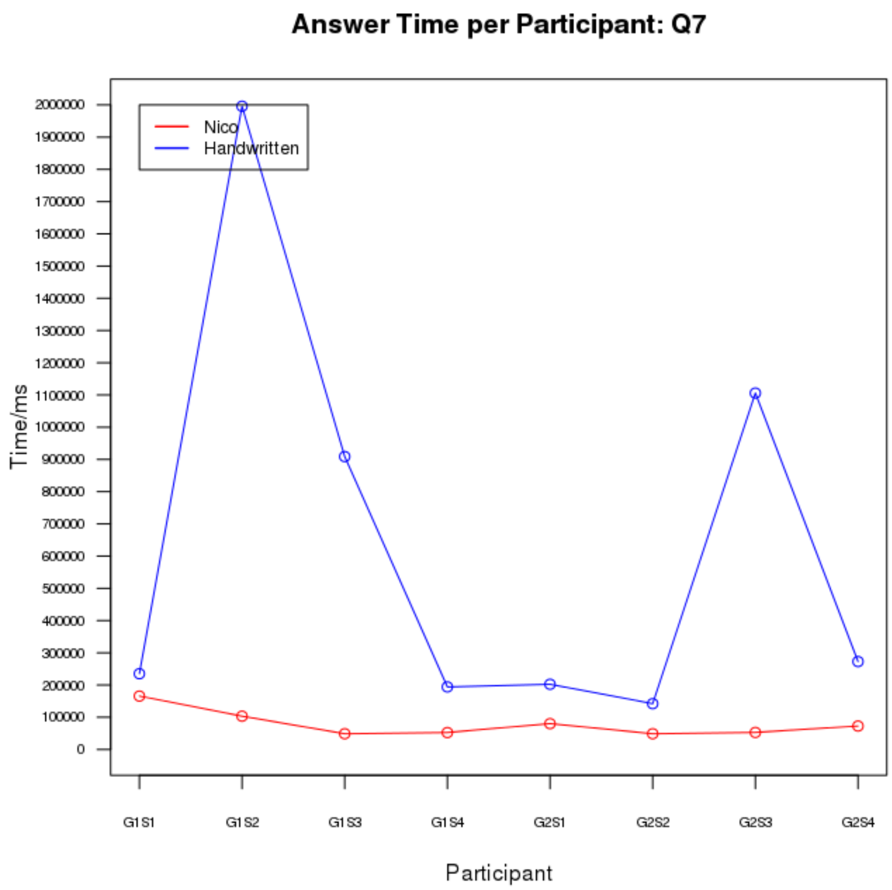


Figure 4.6: Time taken in milliseconds per participant to answer Question 7: 12+14.

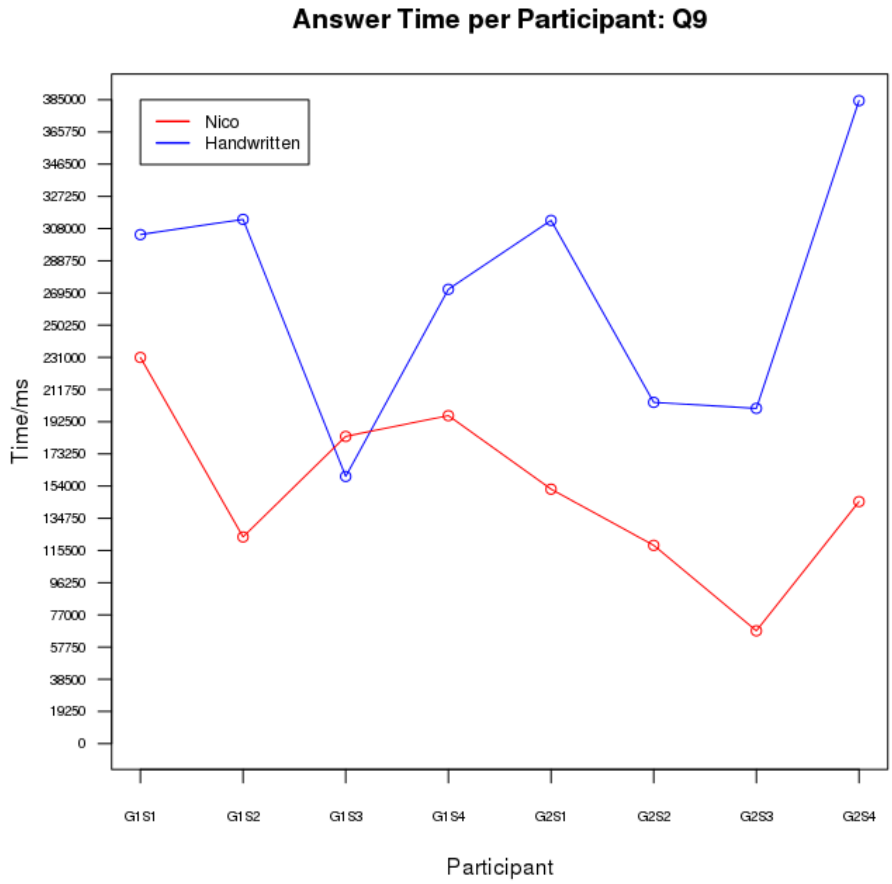


Figure 4.7: Time taken in milliseconds per participant to answer Question 9: 120÷((2×10)+5+5).

A two-sample, paired t-test using an α 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 4.1*. The data is assumed to fit a normal distribution. A Shapiro-Wilk test was considered to test each dataset for such a distribution, but the sample size of eight data points per set (i.e. eight participants for each question) was deemed to be too small for such a test to be meaningful.

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

Table 4.1: 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 α 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 completion time was markedly increased by using the software. This is somewhat to be expected; these questions were included to allow the user to get used to solving simple problems using *Nico*. They each required a maximum of three separate calculations, and constructed from numbers less than ten.

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 used numbers less than ten, but required several more nested calculations to solve, suggesting that *Nico* offered some improvement over handwriting. However, 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, and included many subcalculations. For the seventh, eighth and ninth questions, *Nico* offered a statistically-significant improvement. These questions required 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 clarify for the user, and the results demonstrate an obvious decrease in the time taken to solve such questions.

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, this question was not challenging to solve by hand, and as such the lack of difference between the twoconditions was not unexpected. The simple subcalculations are easier to perform by hand, but combining them is faster using *Nico*.

This conclusion is borne out by the plots shown above: *Fig. 4.4* shows a clear advantage to performing a simple calculation by hand. Question 6 (*Fig. 4.5*), implies that there is no clear advantage to using either method for this type of question. Questions 7 and 9 (*Fig. 4.6* and *Fig. 4.7*), which are 'complex' questions, show *Nico* to be a faster means of calculation in such cases. This is especially the case with Question 7. Data were also collected regarding how well the questions were answered by each user, which follow.

	Group 1							
	Participant 1		Participant 2		Participant 3		Participant 4	
Question	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

	Group 2							
	Participant 1		Participant 2		Participant 3		Participant 4	
Question	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 4.2: 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*.

The Pearson product-moment correlation coefficient r of this data was calculated, comparing the concatenated ‘marks’ and ‘tries’ data series (giving two series of 80 elements). The r value obtained was -0.2278926. Using Fisher’s critical value of 0.2172, we see that this is significant [10]. This indicates that *Nico* does not have a negative effect on a user’s ability to answer questions correctly.

4.2.3.1 Feedback

The final section of the questionnaire constituted a series of questions adapted from Blackwell and Green [4]. The users' responses were graded based upon their tone, scoring -1 for negative, 0 for neutral and 1 for positive. The total number of positive, negative and neutral response are shown in *Table 4.3*, with the full results in *Appendix C*.

	+	±	-
2.1.1	6	1	1
2.1.2	2	0	6
2.1.3	8	0	0
2.2.1	7	1	0
2.2.2	4	0	4
2.3.1	1	2	5
2.3.2	2	3	3
2.4.1	6	0	2
2.4.2	3	0	5
2.5.1	6	0	2
2.5.2	4	3	1
2.5.3	6	2	0
2.6.1	6	2	0
2.6.2	2	1	5
2.7.1	5	1	2
2.7.2	3	3	2
2.7.3	3	5	0
2.8.1	6	2	0
2.9.1	2	6	0
2.9.2	1	6	1
2.9.3	1	7	0
3.1	3	5	0
3.2	2	2	4

Table 4.3: Table showing the totals of positive, neutral and negative responses to each question.

This data can also be visualised using a diagram, as follows.

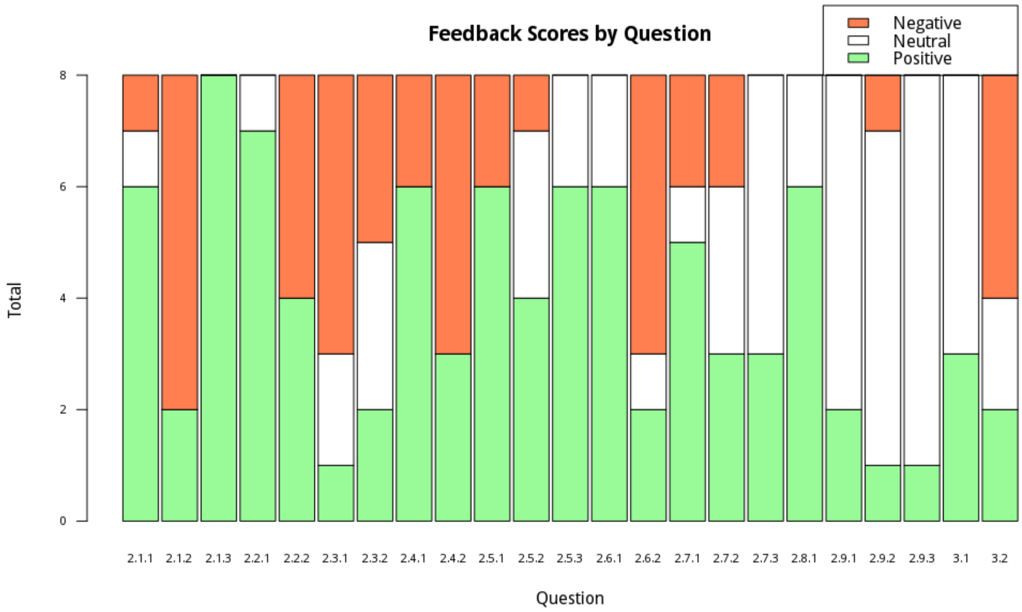


Figure 4.8: Diagram showing the proportions of positive, neutral and negative feedback per question.

The modal feedback scores both by question and by topic are shown below, to determine which was the most common tone of response for each section.

Topic	Question	Mode Score(s)
Visibility & Juxtaposability, Mode = 1	2.1.1	1
	2.1.2	-1
	2.1.3	1
Viscosity, Mode = 1	2.2.1	1
	2.2.2	1 and -1
Error Proneness, Mode = -1	2.3.1	-1
	2.3.2	0 and -1
Closeness of Mapping, Mode = 1	2.4.1	1
	2.4.2	-1
Role Expressiveness, Mode = 1	2.5.1	1
	2.5.2	1
	2.5.3	1
Hidden Dependencies, Mode = 1	2.6.1	1
	2.6.2	-1
Progressive Evaluation, Mode = 1	2.7.1	1
	2.7.2	1 and 0
	2.7.3	0
Provisionality, Mode = 1	2.8.1	1
Secondary Notation, Mode = 0	2.9.1	0
	2.9.2	0
	2.9.3	0
Feedback, Mode = 0	3.1	0
	3.2	-1

Table 4.4: Table showing the modal feedback values by question and by groups of questions.

The modal feedback score across all questions was also taken, giving a value of 1, hence the feedback was broadly positive.[1]

By considering the modal values by topic, it is possible to determine roughly how well-received the software was, with regard to each aspect of the questionnaire. Modes also varied by question, the consequences of which are noted within each section.

4.2.3.1.1 Visibility & Juxtaposability

Mode = 1

[1]The total number of responses were: 39 negative, 50 neutral, 87 positive.

Feedback in this section was broadly positive. One participant described the notation as “very intuitive to use”, although another noted that “circles +[sic] operators could be larger”. Questions 2.1.1 and 2.1.3 received mostly positive feedback; however in Question 2.2.2, several users noted that the connecting lines between circles could occasionally become confusing when numerous.

4.2.3.1.2 Viscosity

Mode = 1

Feedback on the perceived level of viscosity in the software was also positive. Question 2.2.1 received mostly positive feedback, whereas Question 2.2.2 was bimodal, having equal numbers of positive and negative responses. Users noted that it was “very easy” to make changes to previous work, and that it was “easy to do and obvious how to do so”. However, some felt that it was “more difficult to get rid of arguments [using Ctrl]”. Another user concurred with this: “deleting additional numbers [...] is the most difficult, but still simple to remember”.

4.2.3.1.3 Error Proneness

Mode = -1

Users responded to the questions in this section with negative feedback. Several users felt that the drag-and-drop mechanism for joining circles was easily misunderstood, having tried to terminate lines in invalid locations, although one user notes that “I didn’t make any mistakes!”.

4.2.3.1.4 Closeness of Mapping

Mode = 1

This section was generally met with positive feedback, but Question 2.4.2 received more negative reponses. Participants felt that the notation was “very close[ly]” related to the result that they were describing, with one user going so far as to say that it was “exactly so”.

Several users found that the limitation of available numbers to the range -10 to 10 to be a strange way of working. This limitation was a conscious design choice, and it was expected that users would appreciate the need to prevent the software from becoming used as simply a calculator. Additionally, it was difficult to explain this without imposing a method of calculation, and as such this limitation was not mentioned in the tutorial video. Some participants therefore concluded that it was a flaw in the software.

4.2.3.1.5 Role Expressiveness

Mode = 1

Feedback on this section was broadly positive, with Question 2.5.3 receiving particularly favourable responses. User opinions differed with regard to the clarity of the notation. A majority felt that it was "well laid out", and several users found the colour-coding "useful". However, some found that they were liable to forget the meaning of some parts of the notation. For example:

> "if I were looking at a completed diagram it might be confusing but when you work through a question and break it down it is easy to see what's what because you create each part of it as you get there."

Most users felt that there were no parts of the notation that were especially difficult to interpret, although a few noted that the screen can become busy for more complex calculations. One user agrees with the feedback from the 'Visibility & Juxtaposability' section in stating that connecting lines between circles can become "difficult to take in altogether" when numerous. All users responded to Question 2.5.3 with "no", "none" or equivalent answers, indicating that they only used notation that they felt was necessary.

4.2.3.1.6 Hidden Dependencies

Mode = 1

Feedback on this section was mixed, though modally positive. Question 2.6.1 received a positive response: most users found that dependencies between units of the notation were made very clear, apart from one user who found that, while dependencies were clear, it was difficult to infer the results of their calculations whilst constructing them. One user neglected to answer the question. However, the feedback on Question 2.6.2 was negative. Once again, several users found that the notation became harder to read, and dependencies harder to follow, for large numbers of circles. This suggests that either circles should be made smaller, the canvas made larger, or an alternative to linking lines, which can become "tangled", should be sought. Zoom functionality (an unimplemented feature of the prototypes) would also have solved this problem.

4.2.3.1.7 Progressive Evaluation

Mode = 1

Feedback here was modally positive, though responses to individual question were more mixed. Users found the running total in the information panel to be a useful means of tracking their progress, but noted that it was difficult to determine the value of partially-completed calculations where several groups of circles were kept separate until the end, due to the single answer-display.

4.2.3.1.8 Provisionality

Mode = 1

Response to this question was positive. Users spoke favourably of the running total displayed in the information panel, coupled with the prominent bin icon to faciliate easy deletion. One user comments:

> "I could create notations [i.e. circles] and throw them in the bin if they didn't help."

This is in accordance with the intention of the project, which is to allow users to explore potential solutions and to determine why a given method may or may not work.

4.2.3.1.9 Secondary Notation

Mode = 0

Feedback here was neutral. A means of making notes or comments is not provided in the software, though this was neither well- nor poorly-received by the users. Several users mentioned that, given the opportunity to annotate their diagrams in paper form, they would: "keep a track of the different sums I had done so far". Users commented that they did not add any extra marks in the software to clarify or emphasise the information displayed onscreen.

4.2.3.1.10 Feedback

Mode = 0

Feedback in this section was largely neutral. Question 3.1 received neutral responses, whereas Question 3.2 received negative feedback. Most users did not feel that they were using the software in a way which was 'wrong' or 'unusual', though one user did feel that having to create larger numbers from the limited range available constituted such behaviour. Furthermore, several users suggested that an improvement to the software would be to remove the limitation (see 'Closeness of Mapping'). Others felt that the circles should be larger, or that the canvas should be more capacious.

4.3 Summary

A criterion for the success of the project was to develop an application that was able to represent calculations using a novel visual metaphor, and to be able to evalute and display the results of this in under 300ms. This has been achieved, with the function `update-answer` taking an average of approximately 7ms to execute.

As an extension, a user study was conducted to assess the suitability of the software to complement traditional methods of mathematical education in the target audience of Year 5 pupils. The results of this study can be summarised thus:

- There is a statistically-significant difference between the time taken to solve questions by hand and by using the software
  - For simple questions, *Nico* increases the solution time
  - For complex questions involving large numbers and nested calculations, *Nico* decreases the solution time
- *Nico* does not negatively impact the user's ability to answer question correctly
- Feedback on the software was broadly positive, with users concluding that:
  - The notation had few hidden dependencies
  - With many circles, the canvas could sometimes become overcrowded
  - The software encourages exploration, and it is easy to try out many potential solutions
  - The limitation of numbers to the range -10 to 10 was confusing, especially as it was not explained

Chapter 5

Conclusions

Overall this project has been very successful. Not only have the success criteria for this project been fulfilled, but a user study has also been conducted as an extension. As quoted at the beginning of Chapter 3, the success criteria were to develop an application that is able to generate and evaluate expressions from a novel graphical notation, displaying results in less than 300ms. In this regard, *Nico* has met and far exceeded expectations. A functioning application and accompanying visual metaphor have been developed that allow the user to express a calculation graphically. From this, the application is able to generate a valid Clojure S-expression – the tree – that is able to be passed around and evaluated. The application takes considerably less than 300ms to interpret the notation and update the display accordingly.

The software itself comprises a three-layer structure, with an infrastructure that interprets and manipulates user-input data; an interaction handler that provides an iterface between the infrastructure and the user interface; and the user interface itself, the environment in which the user interacts with the notation to express calculations.

The user study conducted as an extension to the main project concluded that *Nico* made a statistically-significant difference to the users' speed of answering questions. This was particularly the case for more complex questions, with many subcalculations and larger numbers, the software was shown to decrease the time taken for users to answer.

User feedback was modally positive, with users finding that the software made it easy to explore many possible solutions, but also that the notation could become confusing where a large number of circles occupied much of the canvas' available area. However, it was also noted that the notation had few *hidden dependencies*, and was easy to read in most cases.

5.1 Future Work

The project suggests a number of potential improvements and extensions:

- To improve the notation in accordance with the feedback received
- To reimplement deletion, preserving both references and referential integrity
- To explore the effects of implementing a negative mark scheme, in which marks are deducted for each incorrect submission
- To develop a partner application to allow tutors to design their own question sets
- To consider other novel metaphors for calculation
- To conduct a larger-scale user study with a similar premise, involving participants from the target audience
- To conduct a study into the long-term effects of using such software in a real school environment

The application of Petre and Green's 'Cognitive Dimensions' framework to metaphors in educational software might yield further worthwhile research, extending Blackwell and Green's research into visual programming languages into the domain of educational software [12] [3].

This project will be further developed and maintained in the future, with a view to its introduction into actual learning environments.

Bibliography

[1] R. Abraham. The Trouble with Math. *Educating the Whole Child for the Whole World: The Ross School Model and Education for the Global Era*, pages 125–137, 2010.

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

[3] A. F. Blackwell and T. R. G. Green. Does metaphor increase visual language usability?, 1999.

[4] A. F. Blackwell and T. R. G. Green. A cognitive dimensions questionnaire optimised for users, 2000.

[5] A. F. Blackwell and T. R. G. Green. Notational systems – the cognitive dimensions of notations framework. *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*, 2003.

[6] L. Bragg. Students' impressions of the value of games for the learning of mathematics. In *Proceedings of the 30th conference of the international group for the psychology of mathematics education*, pages 217–224, Cape Town, South Africa, July 2006. International Group for the Psychology of Mathematics Education.

[7] L. Bragg. Students' conflicting attitudes towards games as a vehicle for learning mathematics: a methodological dilemma. *Mathematics education research journal*, 19:29–44, 2007.

[8] Oracle Corporation. Mouseevent (java platform se 7). `http://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseEvent.html`, 2012.

[9] G. C. Delacruz, G. K. .W. K. Chung, and E. L. Baker. Cresst report 773: Validity evidence for games as assessment environments, 2010.

[10] R. A. Fisher. *Statistical Methods, Experimental Design and Scientific Inference.* Oxford University Press, 1990.

[11] M. Fogus and C. Houser. *The Joy of Clojure: Thinking the Clojure Way.* Manning Publications, 2011.

[12] T. R. G. Green and M. Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework, 1996.

[13] S. Halloway. *Programming Clojure.* Pragmatic Bookshelf, 2009.

[14] Michael Hugill. *Advanced Statistics.* CollinsEducational, 1991.

[15] J. H. Kanner. *The instructional effectiveness of color in television: A review of the evidence.* Department of the Army, Office of the Assistant Chief of Staff for Communications-Electronics, Washington, D.C., 1968.

[16] R. Lamberski and F. Dwyer. The instructional effect of coding (color and black and white) on information acquisition and retrieval. *Educational Technology Research and Development*, 31:9–21, 1983. 10.1007/BF02765207.

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

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

[19] Mike Mannion and Barry Keepence. Smart requirements. *SIGSOFT Softw. Eng. Notes*, 20(2):42–47, April 1995.

[20] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Commuications of the ACM*, 3(4):184–195, April 1960.

[21] Roberta Mura. Sex-related differences in expectations of success in undergraduate mathematics. *Journal for Research in Mathematics Education*, 18(1):pp. 15–24, 1987.

[22] B. A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing*, 1:97–123, 1990.

[23] D. Ray. daveray/seesaw. `https://github.com/daveray/seesaw`, 2011.

[24] A. Roca and F. Rousseau. Interactive multimedia and next generation networks. In *MIPS 2004: Second International Workshop on Multimedia Interactive Protocols and Systems (LNCS 3311)*, Grenoble, France, November 2004. ACM.

[25] samaaron and jlr. overtone/live-coding-emacs. https://github.com/overtone/live-coding-emacs, 2011.

[26] P. Swan and L. Marshall. Mathematics games as a pedagogical tool. In *CoSMEd 2009 3rd International Conference on Science and Mathematics Education Proceedings*, pages 402–406, Penang, Malaysia, November 2009. SEAMEO RECSAM.

[27] P. Swan and L. Marshall. Mathematics games: Time wasters or time well spent? In *Proceedings of the 10th International Conference "Models in Developing Mathematics Education"*, pages 540–544, Dresden, Saxony, Germany, September 2009. University of Applied Sciences, Dresden.

[28] B. Victor. Kill Math. http://worrydream.com/KillMath/, 2011.

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

[30] S. Witamborski. santamon/guiftw. https://github.com/santamon/GUIFTW, 2011.

Appendix A

Third-Party Tools

A list of the third-party tools that were used in the development of the project follows.

- 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 [25], 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

Appendix B

User Response Times

The table and plots of the times taken for each user to answer each question follow.

	Group 1							
	Participant 1		Participant 2		Participant 3		Participant 4	
Question	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

	Group 2							
	Participant 1		Participant 2		Participant 3		Participant 4	
Question	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 B.1: Table showing the time in milliseconds taken by each participant to complete each question, by hand and using the software.

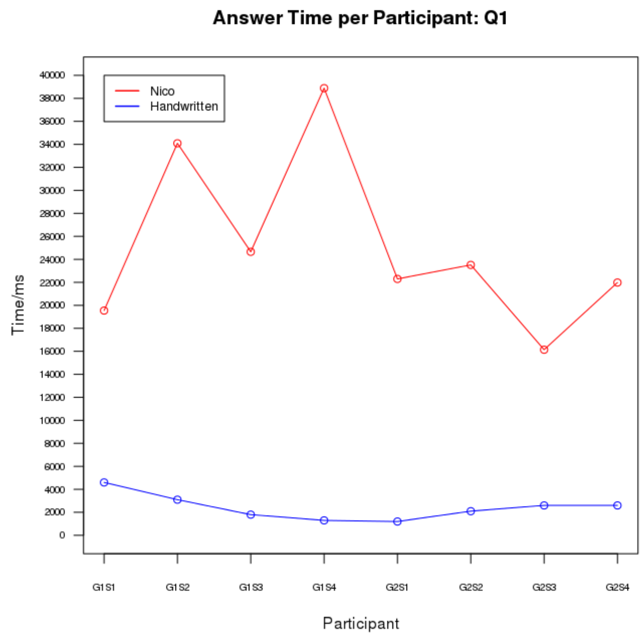


Figure B.1: Time taken in milliseconds per participant to answer Question 1.

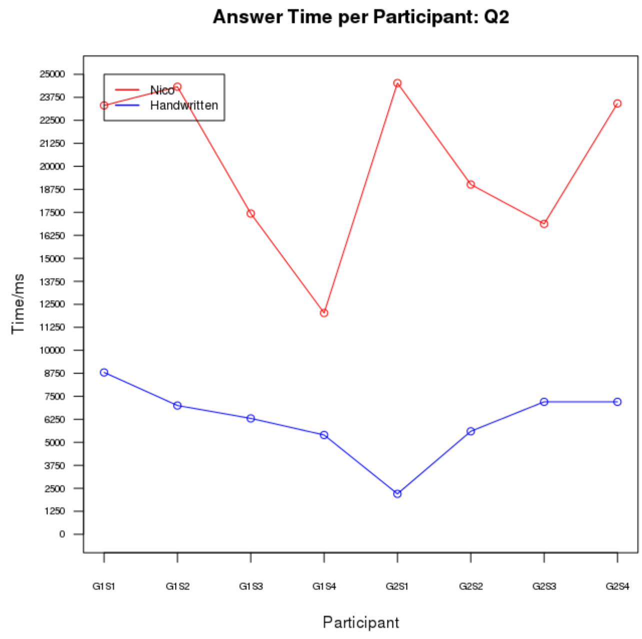


Figure B.2: Time taken in milliseconds per participant to answer Question 2.

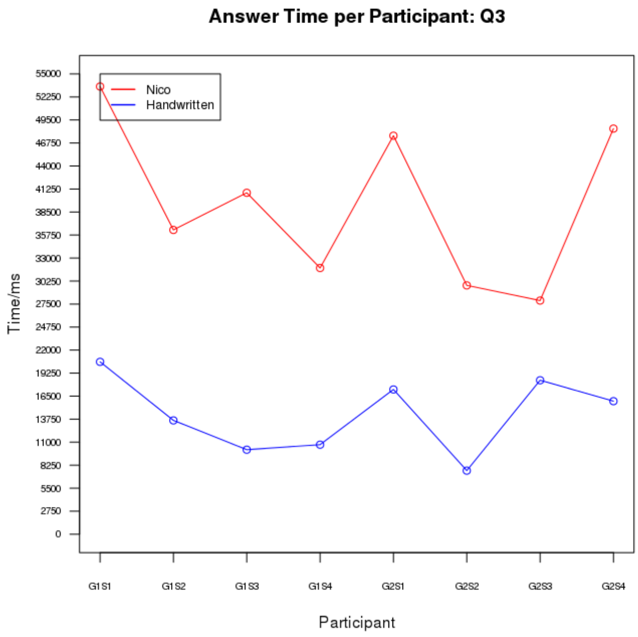


Figure B.3: Time taken in milliseconds per participant to answer Question 3.

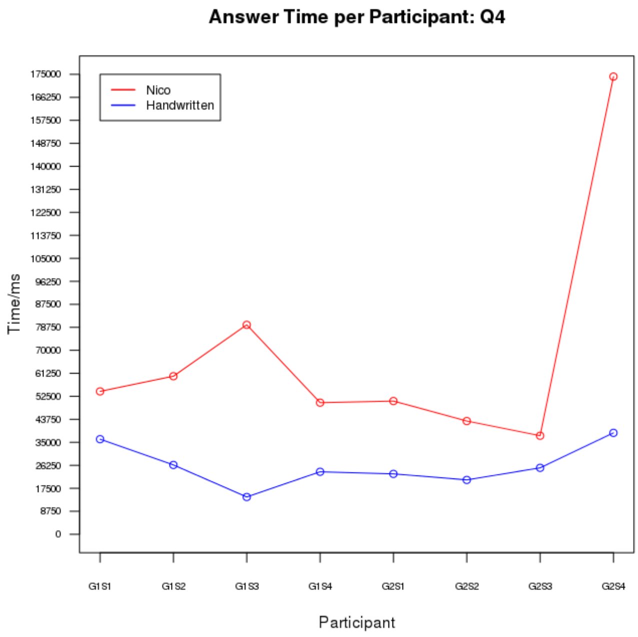


Figure B.4: Time taken in milliseconds per participant to answer Question 4.

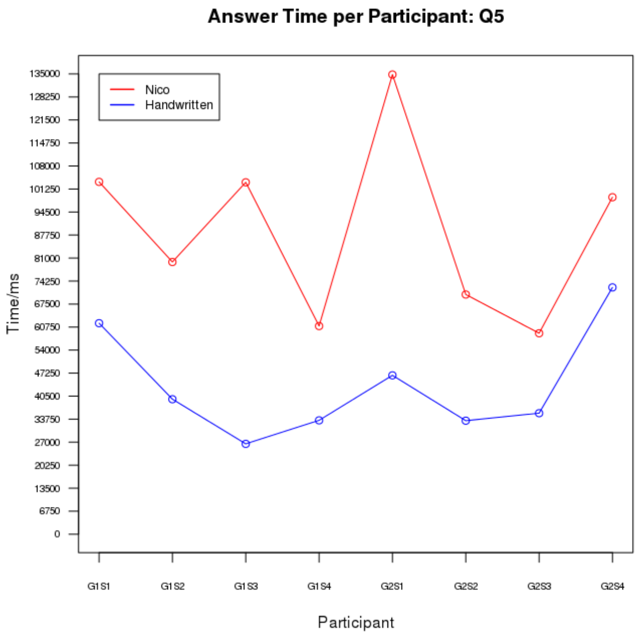


Figure B.5: Time taken in milliseconds per participant to answer Question 5.

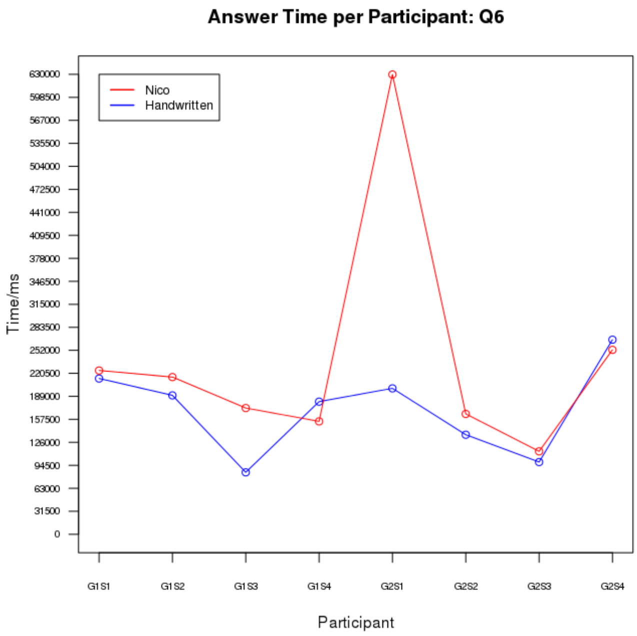


Figure B.6: Time taken in milliseconds per participant to answer Question 6.

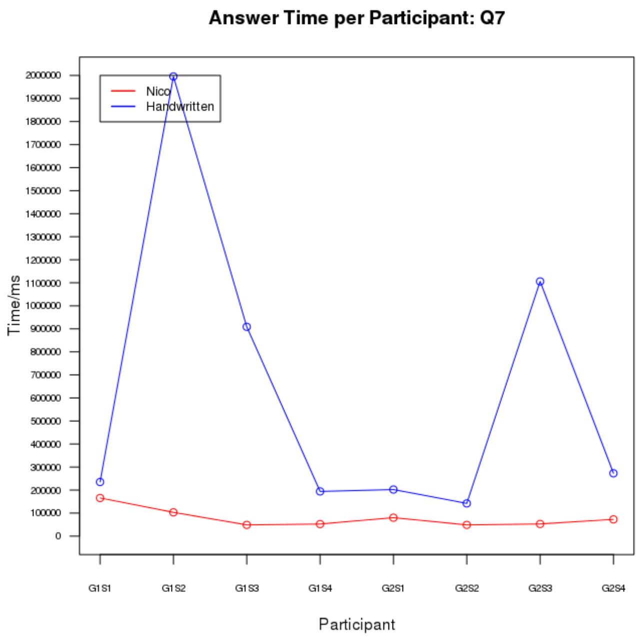


Figure B.7: Time taken in milliseconds per participant to answer Question 7.

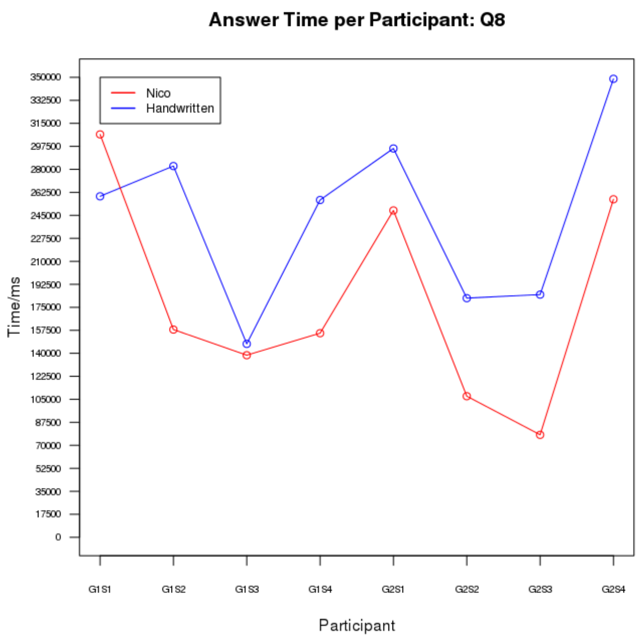


Figure B.8: Time taken in milliseconds per participant to answer Question 8.

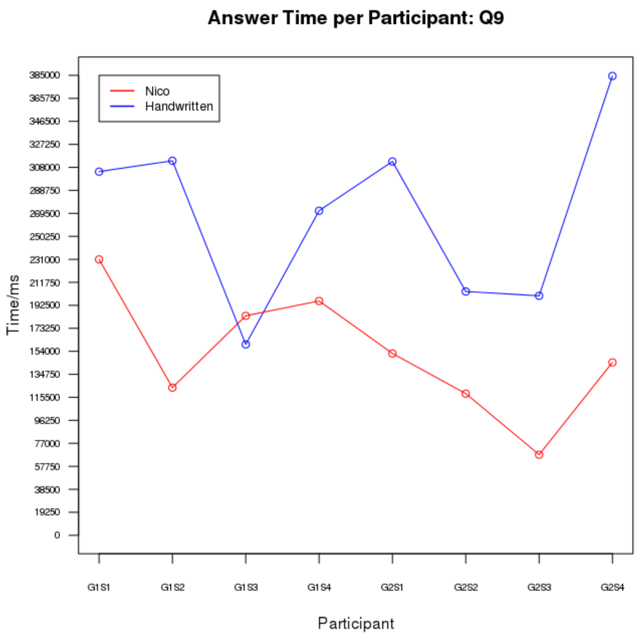


Figure B.9: Time taken in milliseconds per participant to answer Question 9.

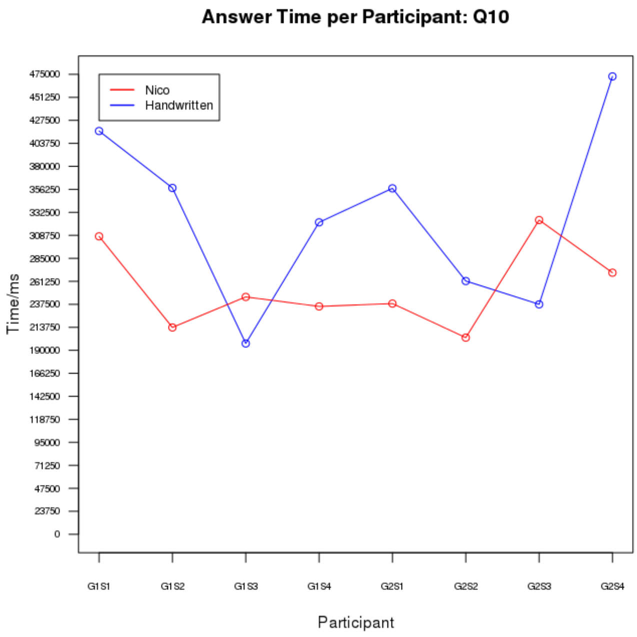


Figure B.10: Time taken in milliseconds per participant to answer Question 10.

Appendix C

User Feedback

The results of the user feedback are shown below, with -1 indicating a negative response, 0 indicating a neutral response and 1 indicating a positive response.

	Group 1				Group 2				Totals		
Question	P1	P2	P3	P4	P1	P2	P3	P4	+	±	-
2.1.1	1	1	-1	0	1	1	1	1	6	1	1
2.1.2	-1	1	-1	-1	-1	-1	1	-1	2	0	6
2.1.3	1	1	1	1	1	1	1	1	8	0	0
2.2.1	1	1	1	0	1	1	1	1	7	1	0
2.2.2	-1	1	-1	-1	-1	1	1	1	4	0	4
2.3.1	-1	-1	-1	-1	-1	1	0	0	1	2	5
2.3.2	-1	1	0	-1	0	0	-1	1	2	3	3
2.4.1	1	1	-1	1	-1	1	1	1	6	0	2
2.4.2	-1	-1	-1	-1	-1	1	1	1	3	0	5
2.5.1	-1	1	-1	1	1	1	1	1	6	0	2
2.5.2	0	1	0	0	-1	1	1	1	4	3	1
2.5.3	0	1	1	1	0	1	1	1	6	2	0
2.6.1	1	1	0	0	1	1	1	1	6	2	0
2.6.2	-1	-1	-1	-1	-1	1	0	1	2	1	5
2.7.1	-1	1	-1	1	0	1	1	1	5	1	2
2.7.2	0	-1	-1	1	1	0	1	0	3	3	2
2.7.3	0	1	0	1	0	0	1	0	3	5	0
2.8.1	1	1	1	0	1	1	1	0	6	2	0
2.9.1	1	0	0	0	0	1	0	0	2	6	0
2.9.2	0	0	-1	0	0	1	0	0	1	6	1
2.9.3	0	0	0	1	0	0	0	0	1	7	0
3.1	0	0	1	0	0	0	1	1	3	5	0
3.2	0	1	-1	0	-1	1	-1	-1	2	2	4

Table C.1: Table showing the tone of feedback by participant and question, with the totals for each type of feedback by question.

Appendix D

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.

Strongly disagree	Disagree	Neutral	Agree	Strongly agree
○	○	○	○	○

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÷3*

3. *1+2+3+4+5*

4. *(2×4)+(3-5)*

5. $((3\times4)\div(3+3))\times8$

6. $((1+2+3+4+5)+(2\times4\times6\times8\times10))\times1\times2\times(1+2+3+4+5)$

7. $12+14$

8. 247×35

9. $120\div((2\times10)+5+5)$

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 E

Project Proposal

The original project proposal follows.

Computer Science Part II Project Proposal

Nico: An Environment for Mathematical Expression in Schools

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

Special Resources Required

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

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

Introduction

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

Work that has to be done

The project breaks down into the following sections:–

1. Core system
    a. A syntax for questions and a means of loading them
    b. A set of basic functions available to the student
    c. A means of inputting an answer that can be evaluated on-the-fly
    d. A means of re-expressing the question to reflect how the student works (e.g. $12 \times 34 \Rightarrow (10 \times 34) + (2 \times 34)$)
    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

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

   b. A means of validating combinations of the drag-and-drop elements
   c. A means of defining functions
   d. A means of viewing documentation
3. Evaluation
   a. Test software on non-technical but mathematically-able subjects
   b. Evaluate the correctness of Nico's translations between diagram and code
4. Extensions
   a. Create and distribute questionnaires to test classes
   b. Collect and interpret data
   c. Create a tutorial mode for new users

Difficulties to Overcome

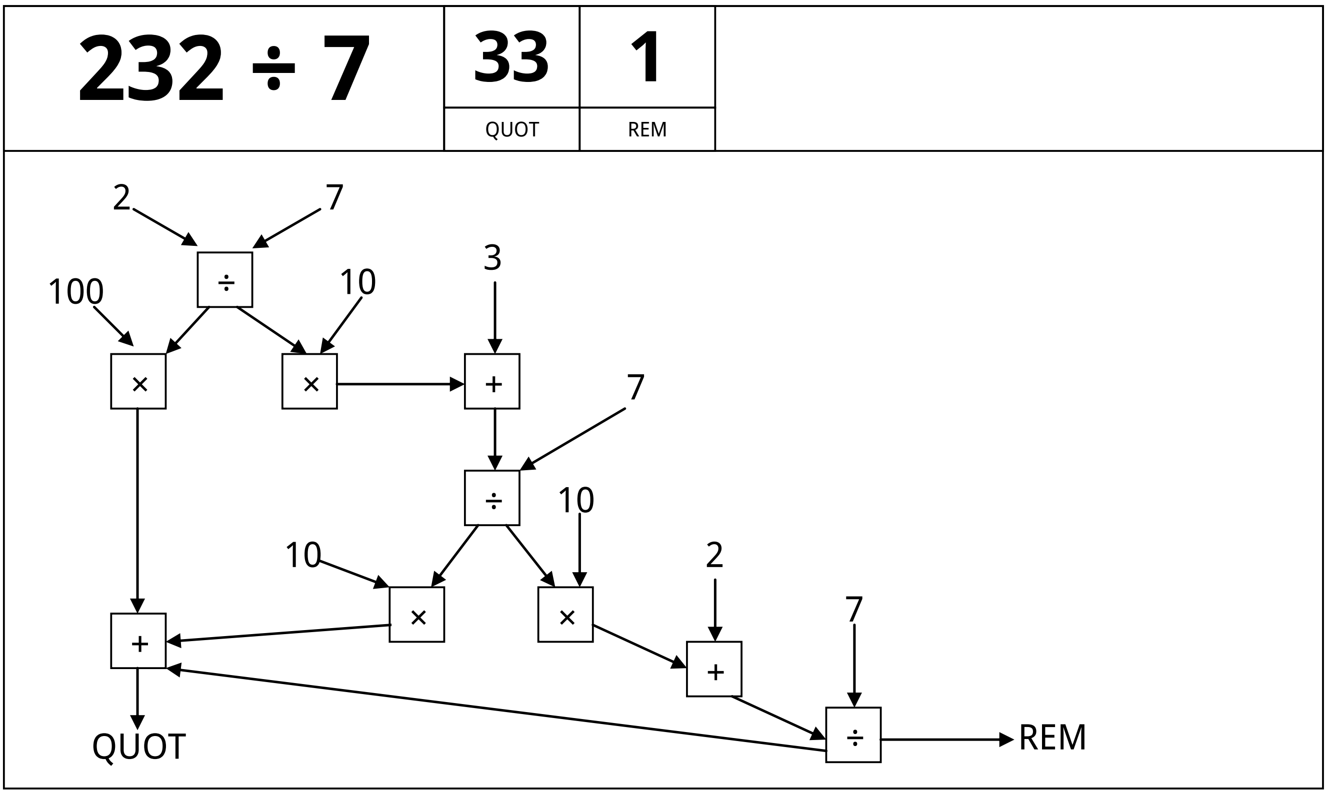
The following main learning tasks will have to be undertaken before the project can be started:

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

Starting Point

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

Below is a mockup of what I aim for Nico to look like. Notice how the method is expressed in the form of a flowchart, with outputs (in this case two) for the answer. Arrows show the direction of input and output, and the QUOT and REM boxes show the result of evaluating the functions being passed to them. Ideally, the question "232 ÷ 7" would also change to reflect how the student breaks down the question.



The above solution would auto-generate the following abstract syntax tree represented in Clojure code:–

QUOT is the output of:–

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

REM is the output of:–

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

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

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

Resources

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

Work Plan

Planned starting date is 27/10/2011.

October 2011

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

November 2011

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

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

December 2011

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

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

January 2012

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

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

February 2012

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

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

March 2012

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

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

April 2012

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

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

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