



University of Glasgow | School of
Computing Science

Register Machine Simulator

Jean Power

2106494

Supervisor: Dr Simon Rogers

A dissertation submitted in part fulfilment
of the requirement of the Degree of
Master of Science at the University of Glasgow
September 2014

Abstract

Educational Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name:

Signature:

Contents

Introduction

In 1936, Turing described a hypothetical computing machine with infinite memory and a scanner that moved along this memory, an unending strip of tape holding symbols, usually presented as binary (Turing, 1936). The machine follows a predefined list of instructions, reading a symbol from memory and performing an action, dependent on the symbol – erase, or write, then move the scanner right or left. Depending on the symbol, the machine may branch to another instruction, or move to the next in the list. Given enough time, a Turing machine can compute any function, in part due to the conditional branching ability. In this way, it can simulate any real-life computing device.

The principles behind the Turing machine underlie modern computing, and are fundamental for understanding the CPU functionality. However, it cannot be fully simulated, due to the requirement for infinite memory. This gave rise to the register machine, defined by Wang in 1957. These are similar to Turing machines with simple instruction sets of arithmetic and conditional operations, but have finite memory, lending themselves well to being simulated on modern computers.

Importantly, register machine functionality also corresponds at a base level to how a CPU works; Dennett states that “Turing Machine[s] and... Von Neumann Machine[s] ... are ‘just’ register machines with speed ups and more memory” (2008).

RodRego is a counter register machine simulator, originally developed in 1986, to demonstrate the computational capabilities of an assembly language containing three instructions, and an allocation of ten data registers (Dennett, no date, Dennett 2008). The instruction set of increment, decrement/branch, and end can combine in surprisingly complex ways, laying a foundation of understanding for CPU functionality. Unfortunately, despite some strong positives, RodRego is no longer fit for purpose, as it does not run on many modern operating systems. These types of educational tools are core to understanding how computers work, but RodRego comes equipped with no help file. It is therefore unintuitive, and inaccessible for users without prior knowledge of the register machine concept. The underlying theories, although logical, can be difficult to understand, and the interface does not reflect this.

This project aims to transfer RodRego to a more accessible medium, keeping the positive aspects and improving on the negative. Importantly, the user interface and tutorials will be focused on, to aid in accessibility, and conceptual understanding. Therefore, the user evaluation will be key to the success of this project.

First, the problem statement will be clearly defined to provide a rational path of research. The background survey follows on from this, structuring an approach to develop solutions to these problems. This includes validating solutions to similar obstacles. Finally, a delineation of the aims of this project, ensuring satisfaction of the problem statement, and the proposed work approach will be defined.

Statement of problem

Educational tools like RodRego are essential; computers impact almost every aspect of human life, and only a handful of people fully understand their inner workings and logic. The concept of a register machine is fundamental to understanding how modern computers work, without being overwhelming in its complexity. If a simulator is appropriately designed, developing simple programs will help users, with no prior understanding, learn basic computing theories.

Appropriate design and feedback is important as, together, they can create a positive learning experience. It has been shown that these experiences shape users' self-image, attitude towards computers, and perceived ability. Impacting this provides motivation to explore and discover further computing techniques and processes (Schulte & Knobelsdorf, 2007).

RodRego V1.3 was developed in 2001 using Visual Basic 5 (Dennet, no date). It is difficult, and in some cases impossible, to run on a modern computer. There are other major flaws; there is no help documentation, the interface is unintuitive, and instructions must be typed in. All of these are stumbling blocks for naïve users. It was clearly intended that RodRego would be used with prior knowledge of register machine functionality.

This project seeks to create an educational tool, using the concept of the register machine that RodRego is based on, with intended users having no prior knowledge of the theory of register machines. It will connect this to CPU functionality, with a goal of improving the user's understanding of how computers work.

Background Survey

In the development of this educational tool, many aspects affect the design and implementation. Primarily, the concept of a register machine and its functions must be preserved, and the theory of this conveyed to the user. This is the central goal of the project; it impacts the user interface, as well as the choice of technology, and the avenues of research in the development of this proposal.

Further to this, it is important to consider when a user will interact with the tool, and how the context of the situation will impact on learning outcomes. The aptitude level of the average user must be taken into account to ensure the information is appropriate, it can successfully convey the theory of a register machine, and how this relates to CPU functionality.

What follows is a brief overview of “The Secrets of Computer Power Revealed”, the paper which RodRego is based on, and a literature review which explores the above facets and their impact on this project.

The Secrets of Computer Power Revealed

Dennett puts forth that, even though it is universally known that a Turing-complete machine (one which can simulate a Turing machine) can compute any algorithmic function, it is not universally understood. To improve this understanding, he describes a register machine with an instruction set of increment, decrement/branch, and end, described in Table x.

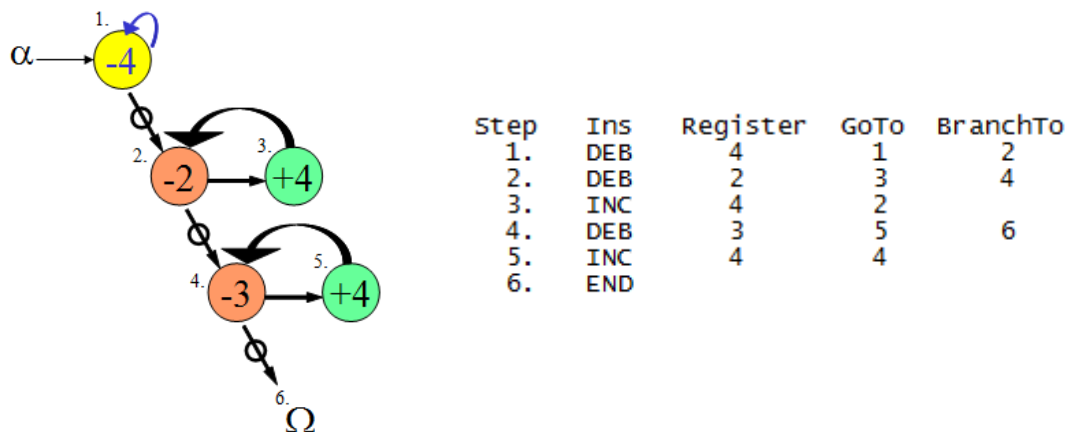
Table x – Dennett Instruction Set

Instruction	Example	Explanation
Increment (INC)	1. INC, 1, 2	Increment Register 1, move to Step 2.
Decrement/Branch (DEB)	2. DEB, 2, 1, 3	Decrement Register 2, move to Step 1, unless Register 2 contains zero; then branch to Step 3 instead
End (END)	3. END	End

He creates an Add program, taking the data from Register 2 and Register 3, placing this data in Register 4. This is demonstrated in Figure 1 as both a flow graph, and in assembly language. The first step clears Register 4 by repeatedly decrementing until zero, when the program branches to the second step. A loop of decrementing Register 2 and incrementing Register 4 follows, with a similar loop for Register 3. Finally, the program branches to end when Register 3 contains zero. This leaves the contents of Register 2 and Register 3 in Register 4.

Dennett builds on this simple example, showing how moving, copying, adding, multiplying, and so on, can be completed. Division, for example, is achieved by a loop decrementing the divisor and dividend, keeping track of each full decrement, restoring the divisor and repeating the process.

Figure 1 – Add Program



With conditional branching, this language contains the power of subroutines; Dennett asserts a program can be written to fully simulate a calculator using this hypothetical register machine and instruction set.

A series of subroutines would be required, one for each mathematical computation. The user would enter the operation in the first register – perhaps 0 for ADD or 2 for MULTIPLY – followed by the operands, their signs indicated by a flag in the succeeding registers. The register machine would decrement the first register and the first in a series of stored numbers, the match of the operation numbers. When the first register reached zero, the program would branch to the stored number; if this is not also zero, it is the incorrect operation. The program then branches to a helper subroutine, restoring the first register and the stored number, before repeating the process with the next stored number. At some point, these would match, and the program would branch to the start of the correct subroutine, performing a computation with the operands. Error flags could also be held for issues like attempting to divide by zero. The only limit to the number and complexity of the operations is the amount of memory locations the register machine has been assigned.

Dennett reminds the reader that, in this example, the data in the registers is being used for different purposes; a number, an error flag, signs, and an operation. It is the programmer who gives meaning to these numbers to interpret them, a core concept in CPU functionality. The set of subroutines, accessed by input from the user, is the basis of a stored program computer.

The register machine works with short, simple instructions and relatively small amounts of memory. Despite the fact that today's computers can work with 32/64bit instructions and billions of memory locations the basis of reading, executing and writing is the same as in a register machine.

Learners are often presented with basic concepts and principles, prior to being taught a subject in depth. This method develops baseline knowledge which supports more complex concepts. Dennett uses the register machine to improve understanding of the Turing Machine and Turing-completeness. The tool to be developed will use this same tactic, with the goal of improving understanding of CPU functionality.

Learning Environment

It has long been suggested that learning outcomes are influenced and constrained by the context of the learning experience (Brown *et al.* 1989). This is similar to computer use, defined and shaped by social and cultural situations (Sutherland, 2000). Both of these are important to consider when developing educational applications, as formal, non-formal and informal learning environments have differing impacts on cognition. Therefore, the 'Where' and 'How' of this tool is critical to consider, as it has consequences on learning.

Separation of learning environments is difficult, as boundaries are blurred by each containing attributes of all environments, but a broad definition can be applied by reviewing the learning itself. Formal learning is compulsory, intentional study leading to accreditation, with the learner making choices within a predetermined, prescribed syllabus. It is most closely associated with traditional classroom education. Non-formal learning is similar; the learner takes part in structured schooling, but they have chosen to partake, and the learning is adaptive to their needs. Swimming lessons or field trips can be considered non-formal learning environments. Finally, informal learning comes about spontaneously, in an unstructured way in everyday life, within a familial or social context. The learning is often led by the learner, and is intrinsically linked with their own motivations (Mocker & Spear, 1982, Vadeboncoeur *et al.* 2006, Eshach, 2007).

There are benefits to non-formal and informal learning. They are effective, with long-term impacts, and improved learning outcomes over formal learning alone. It has been suggested that the majority of science learning takes place outside the classroom, with real world experiences increasing scientific reasoning and, in turn, learning outcomes in the formal environment (Gerber *et al.* 2001). Students who took part in field trips significantly outperformed classmates on scientific attitude tests, a measure of the qualities of mind required for proper scientific investigation (Harvey, 1951). A review of six studies by Falk (1983) found that significant long-term learning occurs on field trips, and this was confirmed in further studies by Lam-Kan (1985) and Finson & Enochs (1987) who corroborated the link between participation, and improvements in scientific concept attainment and attitude towards science. After-school programs, a non-formal learning environment, have also been shown to improve mathematics test results, attitudes towards formal school structure, and attendance (Reisner *et al.* 2004).

Taking this into account, this tool will be developed for use in an informal environment. It is hoped that the user will choose to partake and will interact with it informally, rather than it be disseminated for use in a structured, formal environment. It is believed that this will lead to improved cognitive outcomes.

PC versus Mobile Phones

Mobiles devices are ubiquitous; the number of smartphones doubled, and data traffic increased almost 300% in only two years (CTIA, 2012). This has affected how we communicate, how we entertain ourselves, and how we learn. There are strong motivations to participate in mobile learning. It is believed this is due to a sense of control, and ownership

that users feel over a device, as well as the ability to access it in any context with continuity of material between these contexts (Issroff *et al.* 2007).

Other studies support these outcomes. A study in 2005 found that an overwhelming majority of students preferred accessing e-mail on their mobile phones (Thornton & Houser, 2005). This same study showed that students who received English lesson material over mobile e-mail performed better, in comparison to those who studied it online. Being able to access the learning material everywhere was seen as important, as the students often studied it on their commute home.

This is not an isolated effect. Over a three year period, despite taking additional time to complete, increasing numbers of students opted to use a mobile device to access English vocabulary lessons (Stockwell, 2010). The perception of mobile learning is changing, as it allows users to add productive tasks to otherwise wasted time, like commuting. The ability to integrate these learning activities into everyday life, rather than it being a separate task, gave learners freedom in both time and space. This type of independent learning, in non-formal environments, links in with the previous discussions of improved cognitive outcomes in these settings.

In particular, lower aptitude users had improved test scores when they received text messages, suggesting topics and quizzes that could be accessed through their mobile device. This was in combination with an online learning system, accessible to all students. Users, who received messages through their mobile device, rather than on the website itself, spent twice as long on the learning system and produced improved test scores (Chen *et al.* 2008).

Repetition is the cornerstone of learning, and the spacing of this is particularly important; performance is improved if the same amount of repetition is distributed across time, instead of massed in a single session. This effect has been researched on over 300 separate occasions, and improvements in outcome have been shown in many disparate areas; physical activities like typing, contextual syntax of English grammar, and fact based learning of history (Baddeley & Longman, 1978, Bird, 2010, Cepeda *et al.* 2006, Carpenter *et al.* 2009). It has also been shown to boost test scores in mathematical tasks (Rohrer & Taylor, 2006). Increasing access to learning material should take advantage of this spacing effect, improving learning outcomes.

This all suggests implementation for a mobile device, but a PC based system is still a valid option, as mobile learning will rarely afford a fully immersive experience. A few minutes interacting with a mobile device whilst waiting for a bus could simply never impart as much information as a PC application; large passages of text are difficult to view on small screens, and mobile data connection problems can hamper video viewing and internet connectivity. When learning a language, students found authentic materials, such as novels and TV shows important, as well as scaffolding – hearing the language as well as seeing it written down (Fallahkhair *et al.* 2004). These are difficult to impart solely through the mobile medium.

However, choosing to deploy as a mobile application is sound. Most importantly, the motivational factors should increase the amount of user interaction. It provides more access opportunities, taking advantage of the spacing effect. It could also improve as the users are

naïve to the concept of a register machine. Due to the limited amount of information to be conveyed, the system does not need to be fully immersive. The implementation itself will benefit from the arguably more flexible, interactive nature of a mobile device, with intuitive gesture capture and portability. This choice also reflects the increasing user preference for mobile learning, and should improve cognitive outcomes by reducing the formality of the experience. Finally, there is strong evidence to suggest that users are more likely to download unbranded, unknown applications to their mobile phones than onto their PC or laptop (Chin *et al.* 2012). This would increase both the number and range of users.

Teaching Complex Concepts

Complex concepts, such as the inner workings of the CPU, have specific requirements for teaching the theories fully and correctly. The medium and context of the message and the level of engaged thinking play a major role in imparting this information properly.

In teaching mathematics to children, it is important to use numbers in a meaningful way. If it is only taught in a computational sense, with one right answer and no real world connection, the knowledge can only be applied directly. Encouraging a class to connect normal procedural knowledge with intuitive knowledge, through debate, led a majority of students to deduce mathematically sound, general observations in line with mathematical laws (Lampert, 1990). In this way, complicated theories were organically discovered and understood. Introducing people to register machine simulators, ensuring they have the procedural knowledge, and allowing them to problem solve within this should allow for this natural progression of thought. This developed conceptual knowledge is then relevant for understanding CPU functionality.

Visualising examples, ensuring symbolic expressions are linked with real world phenomena, allows students to develop a more complete and consistent mental model of concepts (Brasell, 1987). This can be assisted by technology; chemistry students using software that had synchronised animated, graphical, symbolic and video representations of virtual experiments, controlled by the student, decreased misconceptions and increased accuracy (Russell *et al.* 1997). Importantly, the students enjoyed these lessons; in other studies, combining audio, animation and video, the tasks were considered fun, evocative and transparent (Thornton & Houser, 2005). The students also learned effectively. Even though the register machine representation is symbolic, combining it with the real world function of computing results will lead to a better overall understanding of how a CPU works.

Humans are natural storytellers and tapping into this enhances learning outcomes; teaching complex medical concepts through storytelling gave a statistically better result over classroom teaching (Vali, 2007). This links in with overlying a meaning to improve learning, when it comes to teaching difficult theories, and implies that story based metaphors are particularly effective.

An important use of technology is gaming, with it being the most common computer operation amongst children (Mumtaz, 2001). A feeling of control and being challenged increase involvement with educational games, leading to active learning; this, in turn,

improves learning outcomes (Wishart, 1990). Interactive media, like games and simulators, significantly outperform traditional teaching methods, especially where the learner controls their own navigation through the system, personalising their experience (Vogel *et al.* 2006). Therefore, even with a difficult concept like CPU functionality, a balanced combination of educational content and a challenging game should appeal to users, and lead to improved cognitive outcomes, over classroom teaching.

Impact of Design

It is clear that mobile learning is an important tool, and one that will be relied upon as education moves outside the classroom. Massive open online courses (MOOC) continue to increase in size and number and, as mentioned previously, integrating technology into education leads to better results (Pappano, 2012). The next step is to convey the information appropriately. To ensure this, the content and design of the tool is critical. If the interface is not accessible and user friendly, if the content does not capture and transmit the information, the user will not gain anything, and the tool has failed.

Below is a table adapted from Najjar (1998), outlining the principles of educational multimedia user interface design. These design principles suggest ways in which the tools can be developed to improve learning.

Table 1 – Principles of Educational Multimedia User Interface Design, adapted from Najjar (1998)

Characteristics of the materials
Use the medium which best communicates the information
Use multimedia in a supportive, not decorative way
Present multimedia synchronously
Use elaborative media
Make the user interface interactive
Characteristics of the learner
Use multimedia with naive and lower aptitude learners
Present educational multimedia to motivated learners
To avoid developmental effects, use educational multimedia with adults and older children
Characteristics of the learning task
Use multimedia to focus a learner's attention
Encourage learners to actively process the information
Characteristics of the test of learning
Match the type of information tested to the type of information learned

With regards to the design of this simulator, some of these principles are particularly relevant. In general, pictures are far superior to text and motion based information should be displayed as a video or animation to fully convey the concept. The display must only contain relevant information; unrelated media distracts the user and decreases learning. Elaborating, for example, text with other media has a particularly positive effect on learning; multimedia encourages the user to engage multiple cognitive channels for processing the material. However, it is important to not overload input channels; text with accompanied audio is

redundant and reduces learning (Kalyuga *et al.* 1999). In a choice between audio and text, Clark & Mayer (2011) suggest using text when a learner is naïve and has the time to process both pictures and text. Interaction with the material pushes users to actively process information, as long as it is engaging; they learn faster, have a better attitude to the material, and retain the information for longer.

This type of multimedia experience has better outcomes with naïve users. It is most effective when the user has no prior experience in the domain. However, they must be intrinsically motivated to take part – naturally, motivated learners will learn more than unmotivated. This motivation can be increased by relating the content to familiar, analogies and situations, in an informal manner, and providing immediate positive feedback. However, informality must be carefully handled, as humour distracts from the core concepts that are being conveyed.

Another aspect that will impact the design is the choice of colour; to increase accessibility, the colours should be chosen to prevent confusion for the colour blind. Without this, the 8% of men who have some degree of colour blindness would be excluded from using the application (Colour Blind Awareness, no date). There are websites which suggest the best colours to use and others to increase the red/green contrast of images; this improves the ability of the colour blind to discriminate between colour variations. (Vischeck, no date, Safe Web Colours, no date)

Studies have been completed on design, and the importance of good design for learning. Having unity, a focal point and balance are important for good design. Unity can be achieved by having elements grouped; the space between each object should be less than the size of the object. The centre of interest should be clear to the user, to grab and hold their attention. This can be created by making an element different or placing it outside a group. Finally, a balance on both the x and y axis of a screen promotes comfort in users. By changing a design to adhere to, or not, these three principles, researchers altered completion rates and time to completion in e-learning (Szabo & Kanuka, 1999). When there is no instructor, these aspects are important for the success of learning outcomes. Luckily, it did not affect achievement scores, which is encouraging information if any incorrect design decisions are made.

Evaluation of current solutions

An essential aspect of project design is evaluation of existing applications to ensure that bad design decisions are not repeated, and positive aspects are incorporated. To cover both possible outcomes of the project, a mobile and PC based version of a register emulator were chosen. It is difficult to objectively define what constitutes a usable application, as each user will have specific requirements and different aptitude levels. To ensure consistency in evaluation, usability heuristics were used as a basis for review.

The usability heuristics chosen were those put forth by Neilson (1995):

Visibility of system status: System operation feedback should be appropriate to the user, and be displayed at appropriate times.

Match between system and the real world: The system should use natural language.

User control and freedom: Undo and redo should be simple, and easy to complete.

Consistency and standards: Follow conventions for the platform, ensure consistency within the application.

Error prevention: Reduce possibility of user errors.

Recognition rather than recall: Instructions, possible actions, current data should be visible and easily recognisable.

Flexibility and efficiency of use: Tailor experience for novice and advanced users, by allowing advanced users to save frequency actions or adjust settings, for example.

Aesthetic and minimalist design: The screen should only contain relevant information.

Help users recognise, diagnose, and recover from errors: Error messages should have natural language, and suggest a solution.

Help and documentation: Help documentation should be easy to find, clear, short and list concrete steps to complete a task.

In addition, for the mobile application, an additional heuristic of **context of use** was taken into account, as this was recently recognised as playing a critical role in usability studies for mobile device software (Harrison *et al.* 2013). And, due to the educational nature of this project, the outcome of application use was documented; **was the educational concept clear?**

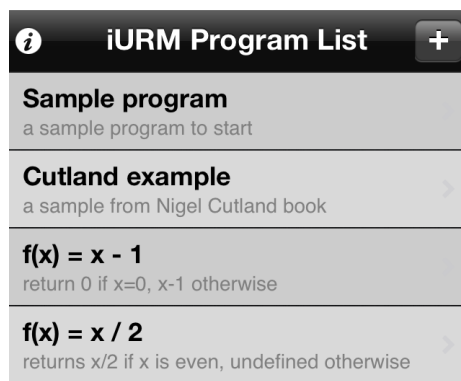
A user persona was developed to assist, as this further deepens the understanding of user requirements, whilst limiting their scope, during evaluation. The starting point for the user persona was the assumption that the application is not intended as an introduction to computers, but as a tool to enhance the understanding of the inner mechanics of a computer. An overview of the user persona developed follows.

The user will be comfortable with technology and using applications. They will find computers and mobile devices relatively easy and enjoyable to use, and understand they can be customised with software. The user will have experience with educational applications. The user will have no understanding of how a CPU works, and will look for a hands on application to develop an awareness of this concept. The user will have motivation to seek out the application and, after use, will expect to have grasped the concept. There is no specific age range for this average user.

IURM (Unlimited Register Machine) (De Bortoli, 2010)

Created by De Bortoli in 2010, this register machine has a limited instruction set of Z (zero), S (increment), T (transfer contents) and J (compare and jump). It was developed for iOS.

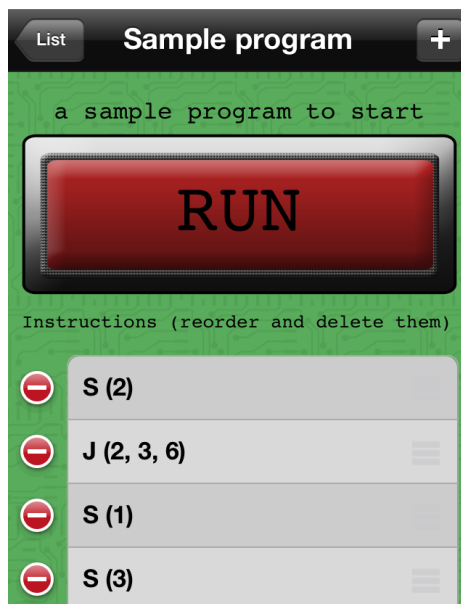
Figure 2 – Program Screen



The first screen is clearly labelled as the program list, and has an obvious 'Help' button, as well as an 'Add' button. Each program has a brief explanation. Between sessions, user saved programs can be stored.

However, there is no tutorial and only high level, sparse help. This is likely due to the intended audience already having experience with the theory of a register machine, as indicated by the reference Nigel Cutland book; this is a university level text.

Figure 3 – Edit program screen



Clicking on a program brings the user to a screen dominated by a 'Run' button. Again, a clear 'Add' button for instructions is visible, accompanied by a short explanation of what is expected of the user. Undoing or moving an instruction is simple through the platform consistent symbols for delete/move.

There are issues for novice users. The Jump instruction requires a parameter of the number instruction that is jumped to, but there is no indication of what number each instruction is.

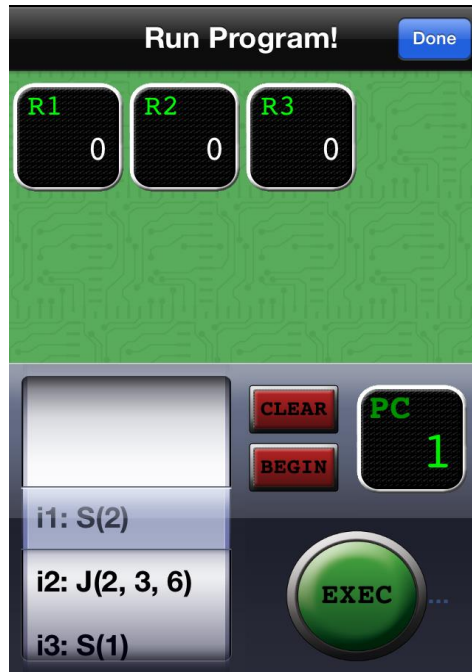
Figure 4 - Add/Edit instruction screen



When adding an instruction, the correct number of textfields for arguments dynamically appears, reducing the possibility of errors. Cancel or save are both clear options.

But, if an argument is not entered, it is defaulted to '0', instead of alerting the user to the issue, leading to unexpected effects. The user is required to remember what each instruction means, and how the arguments are ordered, as there is no link to the help text.

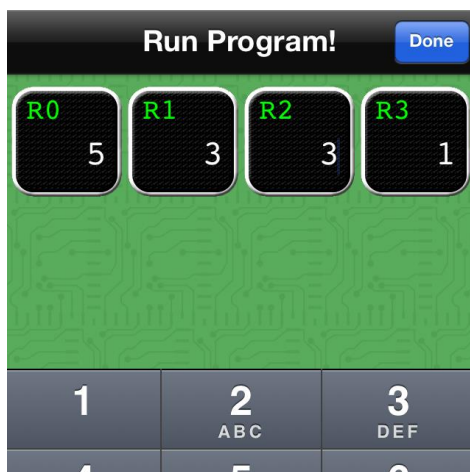
Figure 5 – Run Program screen



Clicking on 'Run', a 'Run Program' screen appears. This only displays registers affected by the program, reducing confusion. Any 5 digit number can be entered into the registers, giving flexibility. As each instruction is executed, it is highlighted on the scroller and on the PC (program counter) indicator. The register icons also flash when they are impacted by an instruction, and slide to the next register when they transfer data.

However, the program does not run automatically when 'Run' is clicked. The user must then 'Exec' the program, which can be easily confused with 'Begin' (a command to return to the first instruction). The ability to edit the registers and enter data is unclear, even though this is often necessary to see the effects of running a program.

Figure 6 – Done button issue



Finally, the 'Done' button returns the user to the instruction screen, which causes problems when parameters are being entered into the registers – users automatically select 'Done' when they have completed editing the registers, losing their edits. Even for users with experience of register machines, this screen is confusing.

In some ways, this solution is elegant. It does not remove itself with metaphors from what it is trying to teach. This ensures there is no confusion, and the message is clear. In many places, the interface is intuitive, using the platform specific conventions. The design is minimalist, with no unnecessary information and undoing is simple. Only allowing the correct number of arguments to be added to an instruction, and the instruction itself selected from a list, vastly reduces errors by the user. Simple animations as instructions are completed add to the experience of use and improve clarity, while showing the system status. The context in which this application will be used has been taken into account; an uncluttered, clear, platform specific design allows for successful use in environments where a user is not paying full attention.

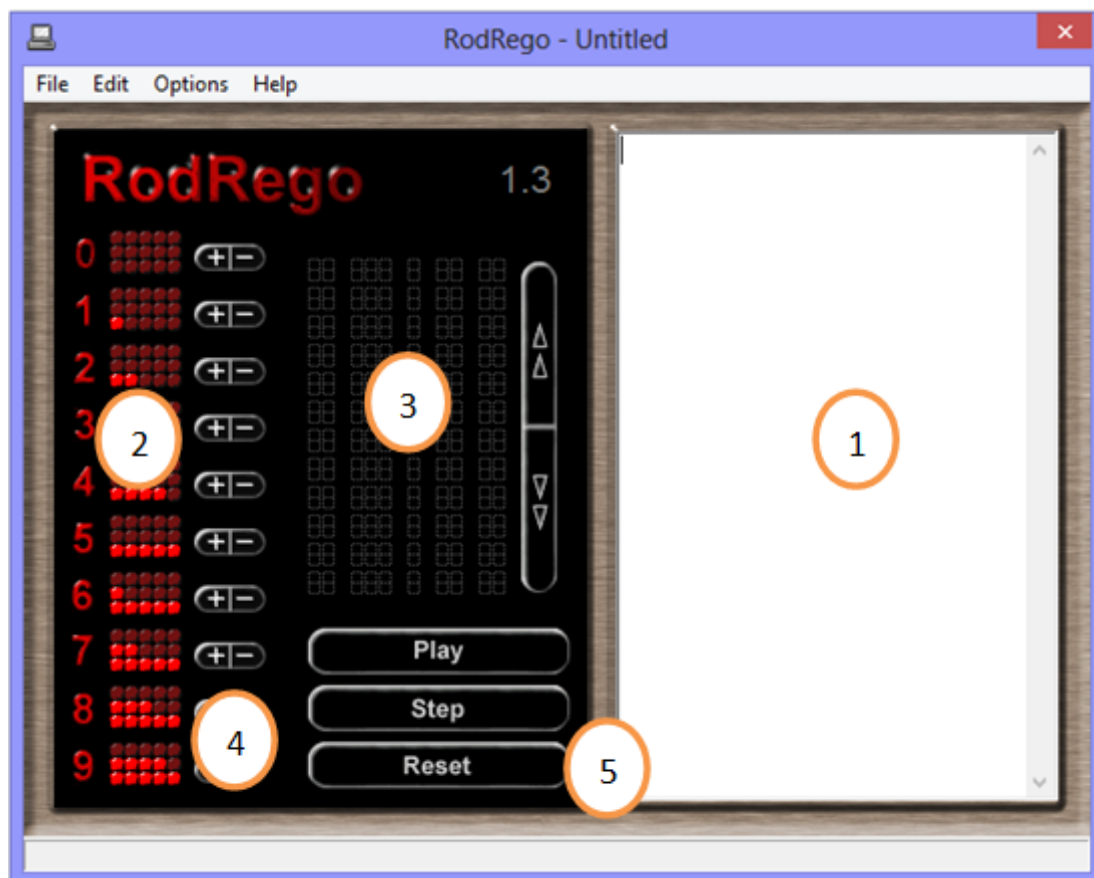
However, this does not carry across the entire application. Having an 'Exec' and a 'Begin' button is confusing. There is no feedback if an instruction does not contain the correct number of arguments. This is compounded by scant help text that is inaccessible from screens other than the main view. If a user was not already aware of how iURM should correspond to CPU function and computer memory, they would not develop this understanding from usage as this connection is not documented. This makes it inappropriate for a novice user.

RodRego V1.3 (Dennet, no date)

As previously described, RodRego is a PC based register machine simulator with an instruction set of INC (increment), DEB (decrement or branch) and END.

There is one GUI in RodRego; this evaluation will refer to specific elements, as numbered in Figure 7.

Figure 7 – RodRego V1.3

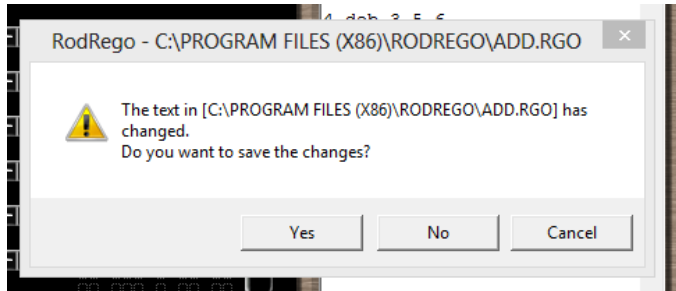


The interface for RodRego is minimalist, with no unnecessary media. It follows platform specific conventions, like Save/Load being present in the File menu, and many of the buttons have clearly associated actions.

Writing, Loading, Saving Programs

RodRego comes with example programs which can be loaded. Instructions can also be written or copy-pasted into (1). The user has the ability to save their creations. Within the program, each instruction has its own number, making it obvious where a DEB instruction will branch to (Fig. 10).

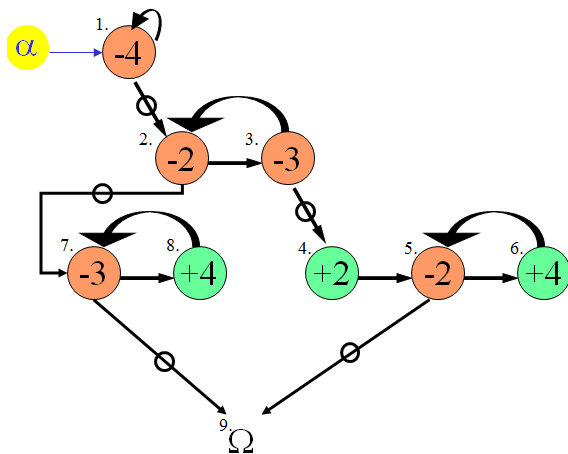
Figure 8 – Error Message



The system also warns the user to save their data if they attempt to exit without saving, reducing accidental data loss.

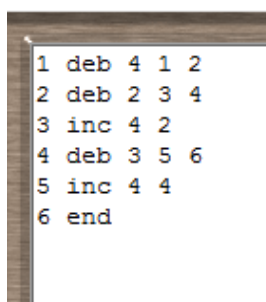
However, RodRego has no associated help or tutorial. Users must read “The Secrets of Computer Power Revealed” to understand the instruction set (Dennet, 2008). PowerPoint presentations from the website of program representations are aimed at users who already have an understanding of register machines and the flow graph notation used to denote a program.

Figure 9 - Flow graph of subtraction of 2 registers (Dennet, 2008).



For example, the flow graph in Figure 9 shows a subtraction, taking two scenarios into account – where the data in Register 2 is greater than or less than in Register 3.

Figure 10 – Entered program

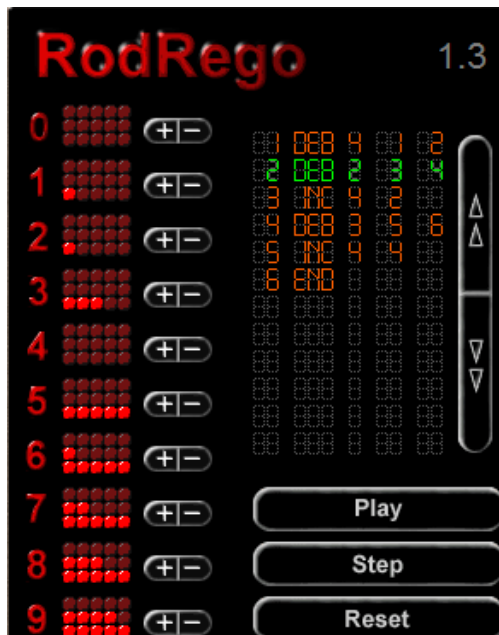


This confusion is exacerbated by users having to enter programs in (1) without indication of what each column corresponds to. With no help, it is easy to mix up which numbers relate to instruction numbers and which to register numbers. It is also not clear that each instruction should have a number in the first column

Finally, the saved file format (.rgo) is specific to RodRego. Despite it containing a text based program, it cannot be opened in a text editor. This reduces flexibility; the files cannot be printed, and are not compatible with other programs, limiting sharing.

Running Programs

Figure 11 - Loaded, running program



When ‘Play’ is clicked, the program is loaded into (3) and, as program runs, the current instruction is highlighted in green. This feedback makes progress through the program clear to the user. The lights (2), representing data, flick on and off as the register is incremented or decremented. In this way, the system status is obvious to the user at all times.

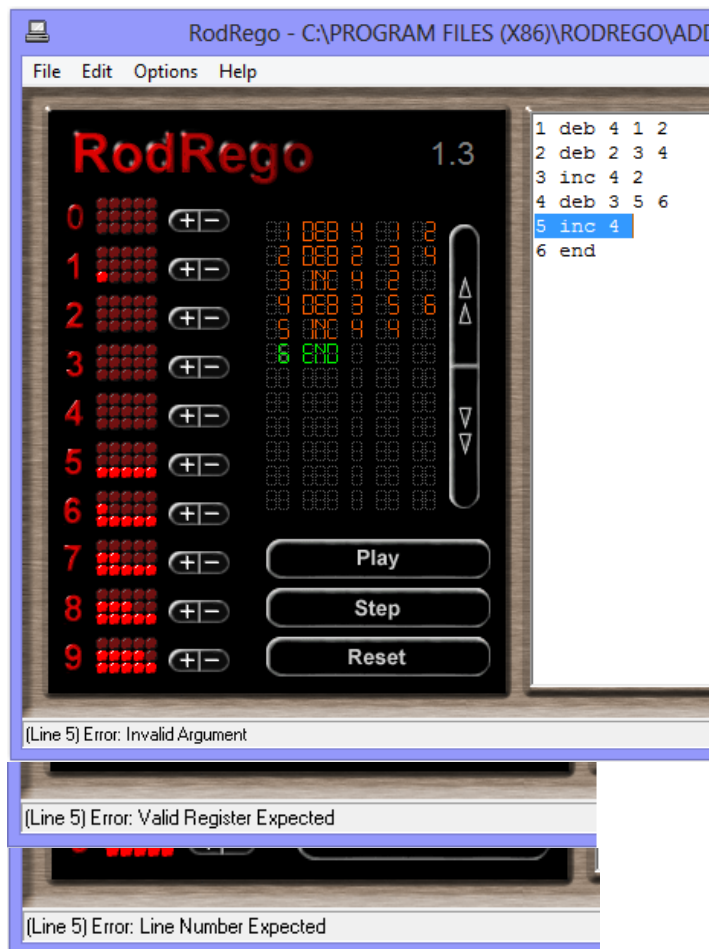
The user can choose to step through the program, or run it in its entirety; this eases debugging.

But the effects of running a program can be difficult to follow. There are ten register representations on the interface and new data is displayed with the same red light as data already in the register. Decrementated registers are not highlighted to show data was removed. There is no clear connection between the program in (3) and the registers in (2); there are no column headings to explain whether a register or an instruction is impacted. Even though the running program itself is easy to follow, its impacts are not. Another limitation of the register representation, not necessarily as apparent to users, is that numbers over 15 are lost. No error message alerts users that a result is incorrect due to this.

There is no easy way to reset the registers, and the user must manually add or remove data with the plus/minus buttons (4), one click at a time. The reset button (5) resets the program back to the first instruction, and not the registers, which is frustrating; to run the same program multiple times, with the same outcome, the user must update the appropriate registers each time. This makes troubleshooting an arduous task.

If there are any errors in the program, error messages are displayed. Some of these are clear, pointing directly to the affected line and stating the issue; “(Line 5) Error: Valid Register Expected”. However, in other cases, they are not as straightforward; “Invalid Argument” does not provide any specification for what a valid argument is. Examples of these error messages are noted in Figure 12.

Figure 12 – Error Messages



If an instruction is entered incorrectly, a blue highlight of the instruction in (1) shows where the error lies.

Some of the error messages are clear, but none have suggestions about how to fix the error.

There is some functionality which must be retained in an updated RodRego system; the ability to save and load programs, the clear step-by-step progress through a program, and the relatively minimalist interface.

But, there is much that needs to be changed. The design must be updated to conform to usability standards. Not all registers need to be displayed at once as this is unnecessarily distracting. A direct connection between the instruction being completed and the impact on the registers would much clarify the action of the simulator. Tutorial and help sections would improve user understanding without having to refer to the academic paper, and labelling what is expected for each argument of an instruction is essential. Manually typing in instructions is prone to errors; this could be improved by an interface similar to iURM, and clearer error messages that provide solutions. How difficult it is to undo actions in RodRego dissuades users from trouble shooting and trying different tasks, this must be changed. These updates would expand the intended audience to include novice users.

Requirements

The overall aim of the project was to take RodRego and implement it on a modern platform. A narrowing of focus was required to ensure a well-defined project with clear requirements. These were gathered through discussions with Dr. Simon Rogers, a review of background literature and current solutions, and evaluation of prototype designs with experienced users. A full list of the requirements is in Appendix (X), arranged as per the MoSCoW method, a tool to prioritise tasks. These are divided into priority groups, indicated by the assigned letter; Must (M) is absolutely critical to the success of the project, Should (S) are important, but a solution would still be viable without them, Could (C) are desirable, and do not have a major impact if left out and finally, Would (W) are requirements that future iterations of the solution may have, but they are not a part of the current process. (reference). This allows for the definition of scope and the correct assignment of time and effort in a project.

Apart from the fundamental requirement of modernisation, whilst maintaining programming functionality, RodRego is an educational tool and this shaped the avenues of research. This led to the decision of deployment to a mobile platform, and heavily influenced the interface. Many of the “should” requirements stem from the needs of naïve users, including in-depth tutorials, measures to reduce user error and parameterised error messages. These were sourced from design and usability heuristics for products with naïve users.

The evaluations that were carried out on current solutions also highlighted the need for this, as confusion and inconsistency negatively affected the learning outcomes of the applications. The lack of help, inability to undo, having to manually type programs, unintuitive interfaces, unclear error messages and lack of labelling were issues which had to be resolved, while keeping positive functionality such as clearly stepping through a program and saving/loading.

Finally the prototype design review cemented the tutorial requirement, with multiple evaluators mentioning it. The design presented was, in general, well received; this was probably due to it incorporating design and usability guidelines.

It was decided to take advantage of research indicating the effectiveness of gamification elements on the motivation and learning outcomes of users, and merge some of these elements into the application. Although not necessary for the success of the project, this was high priority. A register machine simulator is not innately compelling to most, and encouraging multiple visits through challenges, whilst providing positive reinforcement, was expected to make it more captivating.

Despite investigations pointing to storytelling as a powerful way to convey complex concepts, it was decided that applying a metaphor or story to the tool could distract, removing the visualisation even further from the theory of a register machine. Simulating a register machine, with symbols for instructions and memory locations and allowing a user to create programs with these symbols, is already an analogy in itself, a step away from writing assembly language code. This was compounded by the language that needed to be used in the

tutorial, as highly technical terms had to be avoided to prevent confusion, again distancing the tool from core concepts.

Keeping in mind users should progress, leading to different expectations of application functionality, requirements to fulfil their changing needs were considered. Subroutines and the ability to comment specific instructions were the two main functions to be integrated. However, due to the possibility of time constraints and the overarching “educational tool” requirement, these were marked as desirable, rather than critical.

Saving and loading of files, which allows users to continue working on unfinished programs, added the additional advanced functionality of letting users manually create their own text programs. The requirements were drafted to ensure the files were created in a widely used format and defined structure, with the programs written in Dennett’s assembly language (seen in Figure X). This meant they could be edited outside of the application, expanding the user experience, and further deepening their understanding of register machines.

Design and Implementation

After the requirements were defined, class diagrams were designed, along with a prototype of the interface. An evaluation was carried out on this prototype, with feedback integrated, before the interface was implemented. At the same time, the Java classes were developed. Although it was expected that the interface would not have major impacts on the Java code, some design decisions did have effects. Next, further evaluation was carried out with naïve and experienced users; this focused mainly on the ease of use of the interface. After the captured observations were updated, a small number of long term evaluations took place to assess the aspect of learning; a requirement of the tool is to convey register machine theory, and connect this with CPU functionality.

Programming was completed in Eclipse, with GitHub version control. An Xperia Ray, Samsung Galaxy 4 and Motorola Xoom tablet were used in development testing. Tools such as HierarchyViewer were used to assess Android Layouts.

What follows is a discussion of the interface design and implementation with the evaluations carried out on the prototype design, Java design and implementation and, finally, an overview of the unit and system testing which was completed.

Interface Design

Due to the importance of the interface in the success of this project, much research in guidelines and evaluation was completed. The visualisation of the instructions and how they would be connected, allowing for many different kinds of programs, was a complicated decision. The first thought was to use Dennett's flow graphs, the user dragging and dropping increment/decrement instructions on screen and connecting these with loop and branches. However, it was felt that this would not sufficiently convey the sequential nature of a program to users without a computing background. Also, with three different connections required (next, loop and branch), all crossing over and under each other, the program on screen could become unintuitive and confusing; literal "spaghetti" code. The user should be assisted to keep the program as visually clear as possible. Setting instructions consecutively in a line or table reduced the connections required to two; loop and branch.

The first prototype is shown in Figure 13. This concept incorporated a storyline, with the user "Giving" and "Taking" cookies from coloured houses (registers). Each row was a step; the coloured box is the connection to the register, and the background colour indicates give or take. The program would complete the give/take and move to the step indicated in the "Then" column, or branch to the step indicated in the "0?" column, if the register was zero. The program in Figure 13 adds by moving the data from Register 1 and Register 2 into Register 3.

Figure 13 – First prototype design, Cookie Application



This integrated many of the design and usability guidelines, as well as some positive aspects noted in the evaluation of current solutions; there is minimal text, with only the required options on screen. The interface is visual interactive, and balanced, with reminders as to the

actions of the buttons. By using a table, errors are reduced; no instruction could have two next steps.

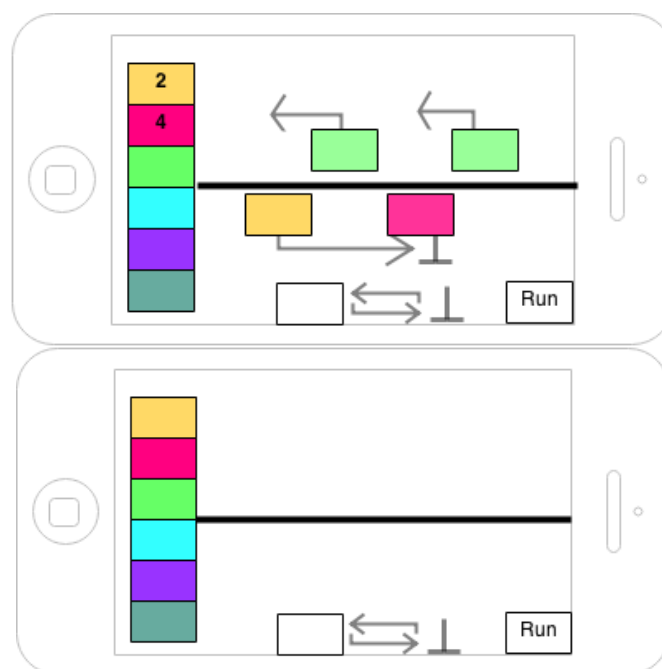
However, prior to evaluation, issues were noted. If a register is affected by two steps, it is not clear to the user which of these two steps is being branched or looped to by other instructions. A transparent overlay of connecting lines was a possible solution but, similar to flow charts, could lead to confusion. Also, the user would have to specify the next step in the “Then” column for all rows. This additional procedure could frustrate users, as it is not intuitive to have to state which step to go to if it is next in the table.

For the evaluation, a different prototype was designed. This is shown in Figure 14, and is similar to the final design of the application.

This did not have a story line, for reasons previously discussed, and builds programs along the line across the centre of the screen. The box icon places an increment or decrement instruction on the line, depending if it is above or below, and the arrows represent the looping or branching. An arrow above the line loops back, and an arrow below the line branches forward. The program shown is the same addition program as in Figure 13. Tapping the registers on the left adds number data; the operands for the program.

The icons are recognisable, without needing text, and the interface is minimal, but interactive. Only the required media is on screen, reducing distractions. User error is reduced by only being able to add instructions with the icons. Unlike the Cookie prototype, the instructions within the program have an obvious successor; the next instruction on the line. The program itself is clearly differentiated from the registers and icons. Both of these make this program visually clearer. The user does not have to manually set the successor, as it is automatic; fewer steps mean fewer chances for error. Finally, by not using an overlying metaphor, the tutorial can be less obfuscated.

Figure 14 – Second prototype design, Line Application



Evaluation of Prototype

Six users evaluated the prototype design in Figure 14. Five of these were MSc. IT students who had completed Systems and Networks, a course which covers CPU functionality in depth, as well as assembly language programming, and one was an MSc. CS student, with a background in computing science. On a scale of 1-5, most considered their knowledge of CPU function and register machines to be a 3.

The aims of this evaluation were two-fold. First, feedback on the design and program building functionality would indicate if the interface was useable. Secondly, open ended questions about register machines and the proposed design could gather additional requirements.

The surveys are found in Appendix E.

Methodology

These users were given an overview of the project and register machine functionality, followed by an in depth description of the prototype. They were then asked to complete a questionnaire, and were encouraged to “think out loud”. They were prompted to comment on various aspects of the design and functionality, rather than providing short answers on the questionnaire

Results and Feedback

No additional requirements were noted from questions probing the volunteers for good and bad aspects of register machine simulators they had used.

Good feedback was received overall; each evaluator agreed that the application conveyed the concept of a register machine, the steps to build programs were straight forward and that the complex concepts were simplified. One user liked the fact that the application “Uses simple concepts such as shape, colour and movement to represent more complex ones, making it easier to understand how a register machine works”

The layout was considered to be clear, colourful and simple. Multiple evaluators felt the application would be useful, with one specifically stating “Wish we’d had something like it in class”. All thought the level of difficulty was appropriate for naïve users, and that the application would increase their knowledge.

Some feedback was received:

- 1) Intensive tutorial/help required.
- 2) Gaming aspect required – challenges, for example.
- 3) Character to help users through tutorial.
- 4) Colour arrows to clarify where they come from and point to.

- 5) Clear distinction between the arrows (branch/loop).
- 6) Motion of program unclear – animate the line from left to right/use chevrons.
- 7) Possibility to add additional registers.
- 8) Add breakpoints.
- 9) Sound cues.

Feedback on the tutorial and gaming aspects strengthened the requirement for these (1-3). The feedback for the arrows was not incorporated immediately, as it was felt that this might add additional complexity to the interface, and another tester believed the arrows were already too emphasised (4-5). It was decided that updates to the arrows would not take place until user testing of the developed application was completed.

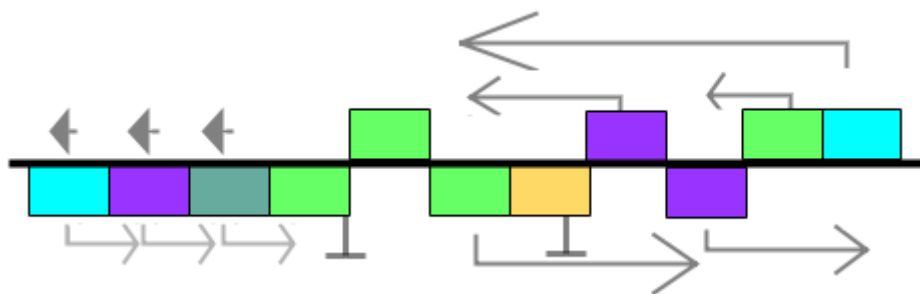
It was thought that animation of the central line would prioritise the sequential nature of the program visualisation too much, taking away from the major role that arrows play. Therefore, this feedback was not used, but it was noted that the tutorial should stress the left to right movement of the program (6).

Adding additional registers and breakpoints are functionalities which would appeal to advanced users (7-8). As this project's main aim is to develop an educational tool, within a limited timescale, these requirements were considered low priority.

Finally, as research showed that text, pictures and audio together can actually reduce learning, the suggestion of sound cues was not integrated (9). (reference)

The positive feedback and ability to create visually clear programs, even complex ones like division in Figure 15, indicated that this would be a reasonable design to implement. The decision was made to move forward with this design.

Figure 15– Division program, Line Application



This program ensures the divisor (Register 3, green) is not zero, and then decrements the divisor and dividend until the divisor is zero. It then restores the divisor using the information in Register 5 (purple), captures the full decrement in Register 4 (turquoise), and returns back to the start of the loop.

Interface Implementation

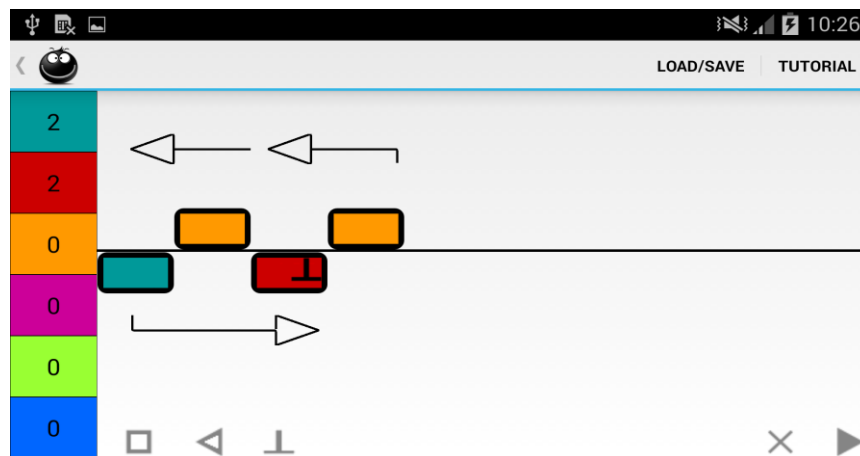
The user interface of an Android application is composed of both XML layouts and dynamically added elements. When the application is launched, `onCreate` is called from the first Activity, the Launcher. This method inflates, or renders, a layout on the screen.

The XML layout defines nested ViewGroups and Views. ViewGroups are containers, holding child Views. They can apply attributes to these children, and the children themselves can also have attributes. ViewGroups and Views can be assigned unique IDs. Anywhere in an Activity, a ViewGroup container can be called with this unique ID, and a new View added to it with its own layout parameters.

Android layouts occur in two passes; the first measures ViewGroups and Views, a top down traversal, where parent ViewGroups push measurements to child Views. The child View then sets its own size, as per its attributes, and the measurements it received. For example, a child View with the attribute of `android:layout_height="match_parent"` will size its height to the largest possible allowed size. The second pass actually places the ViewGroups and Views on screen. (Reference)

The developed user interface is in Figure 15 with an add program. This is prior to user evaluation of the implemented application.

Figure 15 – Developed Interface, prior to user evaluation



The overall structure of the application interface is a `RelativeLayout`, containing two ViewGroups. To the left, there is a `ScrollView` for the registers. Each register is a `Button`, their colours and data populated dynamically in code. Across the top of the screen is the standard Android `ActionBar`. The rest of the screen is taken up with a `LinearLayout`, to allow a consistent 85/15 split between the command bar at the bottom of the screen and the central area for programming when it expands and requires scrolling. This is achieved using `android:layout_weight` attributes.

The command bar contains the add instruction icons, clear/bin and run, each a static `ImageButton`. These are invisible or visible, depending on the system status. This assists the user, as the arrow, end, clear/bin and run icons appear only when they are actually applicable. To reduce the elements on screen, the clear icon becomes the bin icon when an instruction is dragged.

The programming area, or `ActionFrame`, created some interesting programming issues. This area is a `RelativeLayout`; each instruction must be placed relative to other instructions, and the horizontal line, using relative layout attributes. Iteration through the list of instructions, in order of the steps of the program, places each instruction on screen separately. In order for a `View` to be given a relative layout attribute, the relative `View` must be on screen. Two major problems were dealt with when it came to the arrow instructions.

The initial box (increment/decrement instruction) is always placed relative to the `ActionFrame`; left of this parent `ViewGroup`. It does not rely on other instructions for its placement. For the next box, adding the layout parameter of `(RelativeLayout.RIGHT_OF, previousBoxID)` is sufficient because this initial box is present before the next box is placed on screen. Arrows (loop/branch) must be placed relative to these boxes; spanning the correct amount, and also placing themselves relative to each other. However, as the arrows may be pointing to a box or overlapping other arrows that are not yet on screen, errors can occur. This was the first problem to resolve.

Secondly, a custom `View` was originally used for these arrows. However, custom `Views` must override `onMeasure`, a method which is called by its `ViewGroup` parent during the first layout pass, allowing the custom `View` to set its measurements. The size of the arrow custom `View` depended on the boxes it was interacting with. But, the boxes were not yet on screen, as this is completed during the second pass and, so, the custom `View` would return a size of (0, 0). This made the arrows invisible. Performing multiple iterations to add boxes first and arrows second, for example, was not a solution as all child `Views` are traversed for all layout passes.

These issues were resolved in two ways. The first was to use a standard `ImageButton` for each arrow and include a `Sizer` on screen; an invisible `ImageButton` which matches the box size. The `Arrow.class` checks the number of boxes it spans by iterating over the list of instructions. The `DrawArrow.class` uses this to create a `Bitmap` of the correct size, multiplying the number of boxes by the `Sizer` width, and draws an arrow onto this `Bitmap`. It is placed on the `ImageButton` which can, in turn, determine its size from the `Bitmap`. The `ImageButton` is then aligned relative to the arrow's preceding box, which is always on screen prior the arrow. This resolved sizing issue and the horizontal layout relative to the boxes. However, it gave rise to an error where the loop arrows cut through increment boxes (above the line) if they were associated with a decrement box and spanned multiple boxes.

To align the arrows correctly was resolved, again, with the `Sizer`. `Arrow` instructions have the height of the `Sizer`. Iteration over the list of instructions determines if there are any increment and decrement instructions; each arrow is then set with a margin of the `Sizer` height, to prevent cutting through box instructions.

Then, by iterating over the list of instructions, the arrow can determine if it spans the same boxes as another arrow. If it does, the margin of the other arrow is determined, increased and set as the margin for the new arrow. These were sufficient solutions,

Another considerable issue occurred when the custom `ViewGroup`, `TwoDScrollView.class`, was wrapped around `ActionFrame`. (Reference) For longer programs, the box instructions become too wide for the screen, and for complex programs, the arrows stack vertically and become too tall. Without an integrated 2D scrolling layout in Android, `TwoDScrollView` was used. However, `TwoDScrollView.class`, although extremely useful, extends `FrameLayout`, and not `ScrollView`.

If a child of a `ScrollView` is set to match its parent, the `ScrollView` will not activate; the child `View` will constantly expand to the height of the `ScrollView`, rendering it useless. Data in the child `View` will be pushed off screen. `ScrollView` has an attribute, `android:fillViewport`, which forces the child to fill the viewport, and not the parent `ScrollView`.

`ActionFrame` is required to fill the screen; `theLine`, extending across the screen, must be horizontally central. This is impossible without its parent knowing how large the screen is.

As `TwoDScrollView` did not have this useful attribute, all children of `ActionFrame` were impacted; they were placed relative to each other, but off screen, as `ActionFrame` could not determine the central point for `theLine`.

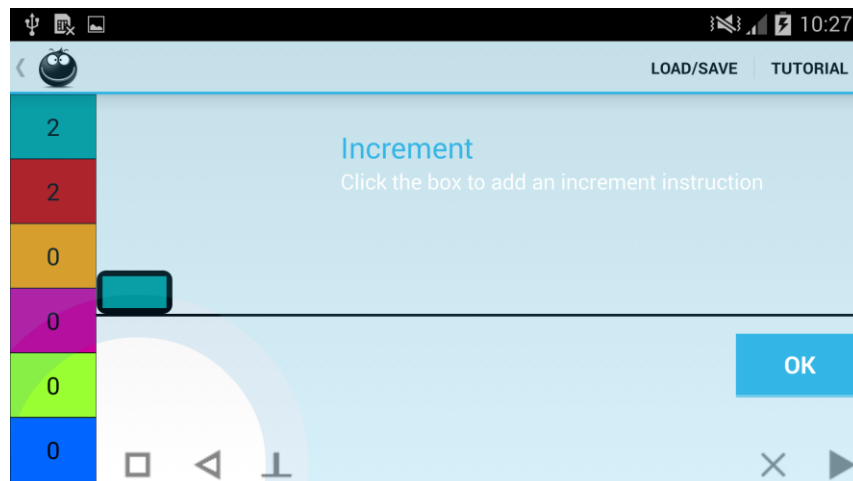
A discussion with Dr. John Williamson of the Glasgow University, School of Computing Science uncovered three options; implement a different 2D scrollview, use an entirely custom `ViewGroup` for the interface or implement the `ActionFrame` with `AbsoluteLayout`, with each child set with exact coordinates on screen.

Investigation yielded no other 2D scrollviews and, with time constraints, design and implementation of a custom `ViewGroup` was not feasible. The suggestion of `AbsoluteLayout`, initially thought to not be good programming practice, was a reasonable solution for this type of interface. The elements are placed relative to each other and the parent layout. As long as the coordinates are determined each time the program is drawn, as opposed to being hard coded, the interface remains accessible.

`AbsoluteLayout` is, unfortunately, deprecated but a resolution in the form of margins was achieved. The elements are set with layout attributes of above or below `theLine`. Using the `Sizer`, the margins of the elements, relative to `theLine`, are also set. The `ActionFrame` can then use these two attributes to accurately place the child `View` on screen.

Another aspect of the interface is the tutorial. This has a simple XML layout, consisting of an image, some text, and buttons. It integrates the library of `ShowcaseView v5.0` (reference), a tool to easily highlight important parts of an application to the user. Figure 16 show this in action.

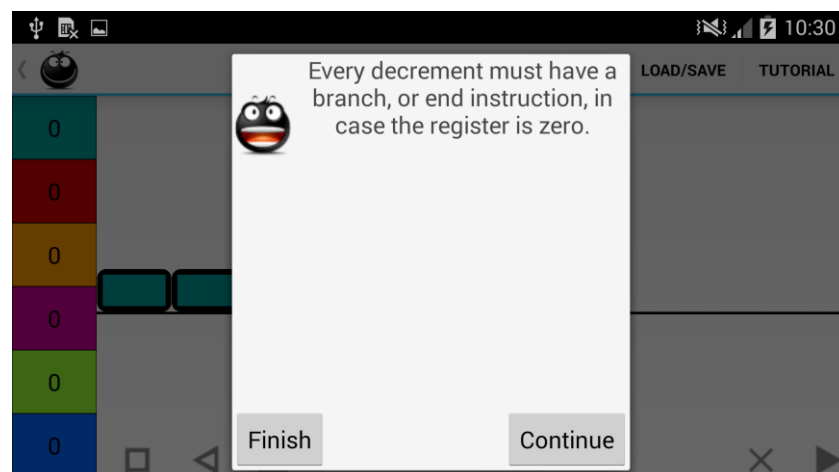
Figure 16 – ShowcaseView



ShowcaseView uses the Builder pattern to implement this screen. It is provided with the target for the focus circle, the title and text, and a click listener for the “Ok” button. In the case of this application, it was further configured to prevent users pressing outside the focus circle, and moving the “Ok” button position so it did not cover the clear/bin and run icons.

As the user moves through the tutorial, dialogs are presented. These contain an image of the character, “Reggie”, and some text to guide the user. The tutorial then displays ShowcaseView to invite the user to interact with the application, an important part of active learning. The tutorial texts explain a register machine, link this with CPU functionality, define the instruction set, and walk the user through the steps required to build a program. It also provides an explanation where something may be unintuitive, as in Figure 17; to a naïve user, the requirement for a branch/end may not make sense.

Figure 17 – Tutorial



This in depth tutorial consolidates the background research and requirements for the application, in the hope it provides an introduction to CPU functionality, and assists learners.

Another requirement was useful error messages with steps to resolve the error. These are throughout the application; file import, file export, building programs and running programs. Each is parameterised either by virtue of its position within the program, or dynamically from the code and helps the user to deal with the error. Two examples are seen in Figure 18. The first warns the user about having no branch or end for a decrement instruction, and the second occurs if a row on a program file import has a number which is not in order, as the step numbers must be consecutive. This is parameterised by the expected step number.

Figure 18 – Error message examples

All decrement (under the line) must have a branch arrow (under the line), or end, so I know what to do when I try decrement a register with zero.

Incorrect data for step 3. Steps must be consecutive numbers, starting at 0.

As documented as a risk in the proposal for the project, the Android programming took longer than expected. This was due to the inexperience of the author but also, in part, due to the complexity of the visualisation of the program. Android layouts were not designed for constant, dynamic updates to a 2D scrolling screen, due to the two pass layout process and the lack of a 2D scrollview. This type of update is usually implemented using a Canvas, capturing the velocity and direction as a user scrolls, and redrawing the elements, or, as was suggested by Dr. John Williamson, a completely custom ViewGroup.

More research prior to programming would not necessarily have highlighted the planned approach as an issue, as RelativeLayout works well on a static screen that does not need to scroll, once the two pass layout process is understood, and TwoDScrollView.class is, at first glance a fully working solution.

Unfortunately, this led to the removal of Gamification requirements from the project, as these were not interlinked with other requirements, and were not critical for success. Attempting to implement these would have had a negative impact on the critical success factors, including the essential user evaluation.

Evaluation of Application

Throughout the development process, there was regular evaluation of the application. The interface and logic were tested with both black-box and white-box testing by the author, and implementation issues and resolutions were discussed with the project supervisor on a weekly basis. Then, after the development of the application, two types of user evaluation were completed.

The first, short-term, evaluation was similar to the prototype evaluation. Users, both experienced and naïve, were asked complete the tutorial, interact with the application and fill in a questionnaire. The users were also watched, their interactions with the application noted.

The aims of this evaluation were to:

- 1) Test the interface design for clarity
- 2) Test the interface design for usability
- 3) Test if programs on screen are readable
- 4) Capture bugs not noticed during implementation
- 5) Test the tutorial's ability to convey the concept of the application, and how to interact with it.

Due to the short amount of time these evaluators were able to spend testing the application, it was felt that the learning outcomes would not be fully assessed. As an important requirement for this project is to deliver an educational tool, the second, long-term, evaluation was completed. In this, users were provided with the application to download to their own devices. They were asked to use the application over the space of a week, and were provided with some suggestions for programs they could develop. This group also completed a questionnaire. The aims of this evaluation were to assess the learning outcomes after repeated uses of the application, as well as the four short-term aims.

Short Term Evaluation

Nine evaluators took part in the short term evaluation, using a Samsung Galaxy s4 (5", 1080p display). The average time using the application was 20 minutes. This included completing the tutorial.

Their responses to the question, "What is your level of knowledge of how a CPU functions" are detailed in Table 2.

Table 2 – Responses to “What is your level of knowledge of how a CPU functions”

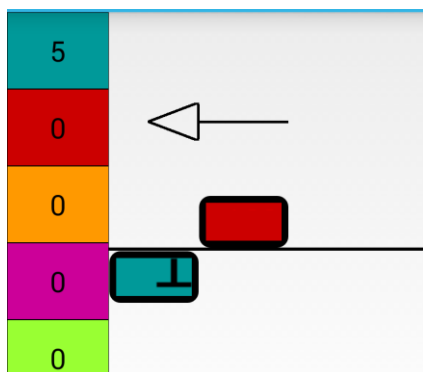
Scale	Users
1 – None	0
2 – Slight	3
3 – Some	5
4 – A lot	1
5 – Full	0

When questioned on how a CPU completes an “ADD” function, most were able to give a brief overview of the CPU taking the data from two registers, adding them, and placing the result in a third register. Some went into more depth, mentioning the use of adders, logic gates, binary and bits. One user in was unable to explain, simply writing “No idea”.

Some users had never used a register machine simulator. Those that had highlighted inadequate or absent tutorials and the lack of a visual interface as frustrations, but did feel it was useful for learning assembly language programming.

After the baseline questions, the users completed the tutorial. At the end of the tutorial, a move program is displayed on screen, as shown in Figure 19.

Figure 19 – Move program



This program decrements the first register (teal), which originally has 5, and increments the second register (red), until the application attempts to decrement the first register and it contains zero. The program then stops.

The users were asked what actions this program took, and their responses (paraphrased) are shown in Table 3.

Table 3 – User responses for action of program

Response	Users
Correct action and outcome determined	3
Decrements the 5 to 0, and ends	1
Decrements and stops immediately	2
Decrements first register, increments second register, decrements first register and ends	1
Adds 5 to itself	1
No answer	1

Despite the majority of users not fully understanding the program, most mentioned decrementing. Some, especially the users who noted that the 5 contained in Register 1 was impacted would, perhaps, have determined the solution if given more time. All were able to follow the program when it was actually run.

Interestingly, the user who specified “No idea” when queried on CPU functionality, correctly answered this question. This was a particularly positive outcome, as the application is aimed at completely novice users.

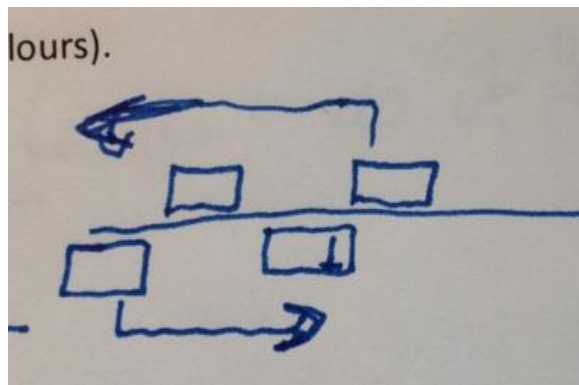
Overall, this suggested the visualisation was appropriate with a readable program, which conveyed the concept of the program itself, decrementing, incrementing and looping.

Next, the users were invited to spend some time playing with the application. It was suggested they attempt to build an “Add” program, as this was another question to be completed. They could also draw this, if they felt more comfortable.

Despite the short amount of time, two were able to create an “Add” program, with two others coming very close, needing only one additional instruction. An example is displayed in Figure 20. Again, this was a good outcome; the users understood the concept of adding using the limited instruction set, and were able to create this program using the application.

Figure 20 – User created “Add” program, with

Reggie instructions; requires one additional arrow.



This program would work if the second register being decremented is larger than the first, as the increment of the third register would always be skipped when the first register reached zero.

If the first register is larger, the program would stop prematurely when the second register reached zero, never looping back to continue the loop of decrementing the first register and incrementing the third.

Finally, the users completed the questionnaire, which looked for feedback on all aspects of the application.

After using the application, most users were able to state that a register machine is a simple CPU, with some describing the process in more depth. The user who originally did not know how a CPU worked responded with “Stores information and becomes the ingredients for functions [sic]”, a definite improvement in their knowledge.

The tutorial was considered to be very clear by all. Some felt that the explanation of arrows and branching needed more description, and more than one evaluator requested examples of programs in the tutorial. These improvements were implemented, but not to the suggested, in depth level of walking the user through creating a program. There were two main reasons for this; first, some users felt the tutorial was already long, with one stating “I forgot how to do

things” due to its length. Second, it is important that the tool challenge users, rather than provide all answers. This is essential in active learning.

In general, users did feel that they needed more time to play with the application to get full benefit, especially as instructions such as branch and loop were complicated concepts. A suggestion that came up multiple times was to have a “help mode”, where users would be provided with extra assistance when adding or dragging an instruction; for example, if the Box icon was selected, a message would remind the user that they can drag the instruction above and below the line which means increment or decrement, and tap the instruction to change the associated register. This would have a short animation demonstrating this.

This concept was too time-consuming to implement in this iteration of the application. However, it is an interesting and useful suggestion, allowing the user to tailor their experience to their own knowledge. Due to the design of the application code, it would be a relatively simple integration; `MainActivity.class` would instantiate a `Help.class`, which holds the animations and pulls the correct String messages from the Android resource folder. `SharedPreferences`, a framework which stores persistent application data, would contain a `boolean` value indicating if the user has the help mode switched on. When elements on screen are clicked, `MainActivity.class` would capture the event and route it to the `Help.class` to display the media.

Feedback and Action Taken

- 1) Arrows become |<<<|
- 2) Flattened screen, as per android expectations (reference)
- 3) Icons have (+) to indicate add.
- 4) Errors with dragging tail of arrow
- 5) Dragging makes views disappear, to be clearer
- 6) images in tutorial
- 7) darker showcase view
- 8) branch appears different
- 9) minor text updates + last showcase taken out
- 10) clear removes register data
- 11) change icons to green (+)
- 12) arrow points down

- Final design screenshots
- Sequence diagram, to show flow of application

Java Design and Implementation

CLASS DIAGRAM TO GO HERE

The class structure followed the Model-View-Presenter (MVP), an iteration of Model-View-Controller (MVC) which is better suited to Android development. Central to both is the requirement to keep presentation separate from logic, allowing for different interfaces, but there are some differences.

The Model is the same in both, holding the structure of the system data; this application has `Instruction` Model classes.

The View differs; activities are the “gateway” to an application, which would make them MVC controllers. However, classes which extend `Activity` must call `setContentView`. This method is parameterised with an XML layout, presenting it on screen, tightly coupling activities and layout. An activity must also deal with events such as `onCreate` and `onPause`. This naturally leads to localisation of all user event routing through the view. `MainActivity` is the central View.

Finally, there is a presenter, which is often referred to as the man in the middle – `Game` in this application. This retrieves data from the model, formats it, and provides it to the view. This decouples presentation from data, allowing for flexibility; a new GUI module can slot in, extending application use across multiple devices. It also has control over the effects of user interaction.

Apart from decoupling and separation of concerns implemented through the MVP pattern, best practices of object orientated program were also followed.

Generalising all instructions under one interface, `Instruction`, allowed the `Game` class to quickly iterate over and pull information from the instructions without needing to interact with each instruction differently, depending on its type. As much as possible, the application refers to `Instruction`, rather than the specific type. This allows for future extension; perhaps to include a `Jump` or `Transfer` instruction.

Getters and setters were also used, ensuring if any changes in the data structure were required, the contract for the class would not change.

Classes were decoupled; content coupling was reduced by keeping instance variables private, accessed only through getters/setters. Examples of this are noted throughout all classes. Methods get data required to complete their actions using these getters to prevent control coupling; this can be seen in particular between the `Arrow` and `Box` classes, as the actions of `Arrow.doWork` depends on what action the `Box` instruction was able to complete. Some common coupling was required, as constants like the maximum number of registers impacts multiple classes.

Best practice for Android is to keep all resources external to the application, including images, text and colours. This allows for independent updates, and easy maintainability, as well as catering to different device sizes, and languages. Due to time constraints, this application does not currently have these provisions. However, because the resource directory was used, future extensibility is merely a matter of adding the larger icons, or the translated text, into the correct folder.

Some coupling was required; although all updates/access to the data of the models is through Game, at certain points, MainActivity needs to know the type of instruction. That is generally dragging/touching, where the MA needs to change its behaviour, depending on what is being dragged or touched, and where the updates need to be quickly done on screen – it is efficient for MainActivity to make these decisions itself, rather than passing responsibility to Game and waiting for a response, or updating arrays (and iterating through) of each type of instruction.

Tutorial is completely decoupled from instructions, accessing information/adding instructions solely through Game. However, it does also interact with SaveLoad. This is because of the file loading ability of SaveLoad. It allows for any file to be populated at the end of the tutorial, instead of hardcoding the program on screen.

SaveLoad is a Presenter class, a helper to Game, but also holds responsibility for showing the file dialog for capturing the user input. The reason for this is to allow the file dialog to be changed easily (maintainable), with all actions required of saving and loading together in one class.

Error check is model class, holding error checking logic, interacting directly with model data, another helper to game. This does not interact with activity.

The instructions – Box, End and Arrow – due to the structure chosen (Doubly linked list) are coupled through predecessor/successor information. They also provide the called (Game) with the new instruction to be completed, as they set the next instruction. They also pull a boolean from game to see if a decrement was possible. This affects their actions.

- Final class diagram (as do not foresee user evaluation having major impacts on Java, so iterations of the previous class diagrams not required)

- Design discussion

- OO best practices which would be used

- Singleton Util class to give unique IDs.

- Push and Pull of model data (update currPos, etc)

- Threading – discussion of threads needed, and runOnUiThread

- MainActivity brief overview

- Static Layout separate (XML) from dynamic (Java)

- UpdateDisplay() method
- onTouch() and onDrag() methods
- Extensibility decision - models are connected to Android components, instead of extending Android components
- Instruction interface
- Allows for MainActivity and Game to interact with the instructions, without having to know their exact type
- Brief overview of Box/Arrow/End
- Game
 - UpdateTail() method + associated helper method
 - Decision re. making pred == goTo if switch.
 - UpdateHead() method + associated helper method
 - RunGame() method + how screen is updated using threads
- File Input/Output
 - aFileDialog -
- Help + Error Checking/User assistance
 - Game error checking (head cannot go behind tail, etc) with dialogs to inform user
 - File output ignores multiple arrows out of same box
 - Tutorial

Testing

Throughout the development, as new code was completed, informal unit testing was completed on each method. This included white-box testing which followed the path of data through the methods, and black-box testing where the author interacting directly with the application, keeping note of any bugs, and updating them. Towards the end of the project,

formal testing was completed. In some cases JUnit testing was completed, and in other cases, black box testing was more appropriate. Due to time constraints, the formal testing was not as in depth as would be expected of an application to be deployed. However, it is believed that good coverage was achieved by the regular informal testing, wide ranging user evaluation, and the series of formal, documented tests which follow.

Testing of SaveLoad.class was black box. This is due to the previous testing of MainActivity.class and Game.class, which had proven both the correct visualisation of the program, and running the program, from the list of instructions. The error checking prior to file creation is completed by ErrorCheck.class, also previously tested.

The files saved by this class take the list of instructions from Game, and create a file written in Dennett's assembly language. Files of the same structure can also be loaded into the application.

Four programs were created (Add, Move, Copy and Subtract with a flag to indicate if the answer is a minus). These were saved as .txt format by the application. The data in these files was manually confirmed to be correct; in Dennett's assembly language, specifying the correct operations.

The files were then loaded back into the application. Screenshots from before and after confirmed the correct program was displayed on screen. The programs performed correctly, as was expected from MainActivity.class and Game.class testing.

Next, three malformed files were created to ensure all error messages were firing correctly. These were loaded into the application, and the error messages checked.

An attempt was made to create malformed programs on screen, to determine how SaveLoad.class would cope with this for file output. It was decided to create more error checks on programs, rather than allow a user to export a program with errors such as multiple arrows coming from the same Box. As the application is an educational tool, it should direct users to create correct programs, as they may not have the knowledge to debug without help from the application.

Conclusions

- Success Parameters
- Discussion of risks
- Outstanding issues
 - Automatic scaling of screen for program size
 - Tested on x number of devices, but needs more
 - Showcaseview has GC issues, which despite update, did not solve. Causes program to crash if multiple tutorials completed
- Future extensibility

- Allowing for breakpoints - when register reaches certain value, or specific instruction - and "stepping"
- Many registers
- How to display in the most efficient/pretty way
- Entire layout should be custom view
- Subroutines which can be loaded in
- Help mode – each new instruction shows short help, to remind users
- sort instructionlist, rather than iterating through 3 times
- Social sharing – FB/Twitter

References

- Android. (No date). Using Hardware Devices. [Online]. [Accessed 26Mar2014]. Available at: <http://developer.android.com/tools/device.html>
- Apple. (2014). iOS Developer Program. [Online]. [Accessed 26Mar2014]. Available at: <https://developer.apple.com/programs/ios/>
- Baddeley, A. D., & Longman, D. J. A. (1978). The influence of length and frequency of training session on the rate of learning to type. *Ergonomics*, **21**(8), pp627-635
- Bird, S. (2010). Effects of distributed practice on the acquisition of second language English syntax. *Applied Psycholinguistics*, **31**(4), pp635-650.
- Bort, J. (2013). Google: There Are 900 Million Android Devices Activated. *Business Insider*. [Online]. 15May. [Accessed: 26Mar2014]. Available at: <http://www.businessinsider.com/900-million-android-devices-in-2013-2013-5>
- Brasell, H. (1987). The effect of real- time laboratory graphing on learning graphic representations of distance and velocity. *Journal of research in science teaching*, **24**(4), pp385-395
- Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational researcher*, **18**(1), pp32-42
- Carpenter, S. K., Pashler, H., & Cepeda, N. J. (2009). Using tests to enhance 8th grade students' retention of US history facts. *Applied Cognitive Psychology*, **23**(6), pp760-771
- Cepeda, N. J., Pashler, H., Vul, E., Wixted, J. T., & Rohrer, D. (2006). Distributed practice in verbal recall tasks: A review and quantitative synthesis. *Psychological bulletin*, **132**(3), pp354

- Chen, G. D., Chang, C. K., & Wang, C. Y. (2008). Ubiquitous learning website: Scaffold learners by mobile devices with information-aware techniques. *Computers & Education*, **50**(1), pp77-90
- Chin, E., Felt, A. P., Sekar, V., & Wagner, D. (2012). Measuring user confidence in smartphone security and privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security* (p. 1). ACM
- Clark, R. C., & Mayer, R. E. (2011). *E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning*. John Wiley & Sons. pp133 – 142
- Colour Blind Awareness. (No Date). What is colour blindness? [Online]. [Accessed 26Mar2014]. Available at: <http://www.colourblindawareness.org/colour-blindness>
- CTIA. 2012. CTIA's Semi-Annual Wireless Industry Survey Results. [Online]. [Accessed: 26Mar2014]. Available at: http://files.ctia.org/pdf/CTIA_Survey_YE_2012_Graphics-FINAL.pdf
- De Bortoli, A. (2010). IURM (Unlimited Register Machine). [Online]. [Accessed: 26Mar2014]. Available at: <https://itunes.apple.com/us/app/iurm/id386981571?mt=8>
- Dennet, D.C. (No Date). RodRego - Register Machine Simulation (Ver 1.3). [Online]. [Accessed: 26Mar2014]. Available at: <http://sites.tufts.edu/roddreg>
- Dennet, D.C. (2008). The Secrets of Computer Power Revealed. [Online]. [Accessed: 26Mar2014]. Available at: <http://sites.tufts.edu/roddreg/files/2011/03/Secrets-of-Computer-Power-Revealed-2008.pdf>
- Eshach, H. (2007). Bridging in-school and out-of-school learning: Formal, non-formal, and informal education. *Journal of Science Education and Technology*, **16**(2), pp171-190
- Falk, J. H. (1983). Field trips: A look at environmental effects on learning. *Journal of Biological Education*, **17**(2), pp137-142
- Fallahkhair, S., Pemberton, L., & Masthoff, J. (2004). A dual device scenario for informal language learning: interactive television meets the mobile phone. In *Proceedings of the IEEE International Conference on Advanced Learning Technologies* (pp. 16-20). IEEE Computer Society
- Finson, K. D., & Enochs, L. G. (1987). Student attitudes toward science- technology- society resulting from visitation to a science- technology museum. *Journal of Research in Science Teaching*, **24**(7), pp593-609
- Gerber, B. L., Cavallo, A. M., & Marek, E. A. (2001). Relationships among informal learning environments, teaching procedures and scientific reasoning ability. *International Journal of Science Education*, **23**(5), pp535-549
- Harrison, R., Flood, D., & Duce, D. (2013). Usability of mobile applications: literature review and rationale for a new usability model. *Journal of Interaction Science*, **1**(1), pp1-16.

- Harvey, H. W. (1951). An experimental study of the effect of field trips upon the development of scientific attitudes in a ninth grade general science class. *Science Education*, **35**(5), pp242-248
- Issroff, K., Scanlon, E., & Jones, A. (2007). Affect and mobile technologies: case studies. In *BEYOND MOBILE LEARNING WORKSHOP* (p. 18).
- Kalyuga, S., Chandler, P., & Sweller, J. (1999). Managing split-attention and redundancy in multimedia instruction. *Applied cognitive psychology*, **13**(4), pp351-371
- Lam-Kan, K. S. (1985). The contributions of enrichment activities towards science interest and science achievement. Unpublished master's thesis, National University of Singapore, Singapore. [ABSTRACT ONLY]
- Lampert, M. (1990). When the problem is not the question and the solution is not the answer: Mathematical knowing and teaching. *American educational research journal*, **27**(1), pp29-63
- Mocker, D. W., & Spear, G. E. (1982). Lifelong Learning: Formal, Nonformal, Informal, and Self-Directed. Information Series No. 241
- Mumtaz, S. (2001). Children's enjoyment and perception of computer use in the home and the school. *Computers & Education*, **36**(4), pp347-362
- Najjar, L. J. (1998). Principles of educational multimedia user interface design. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, **40**(2), pp311-323
- Nielsen, J. (1995). 10 Usability Heuristics for User Interface Design. [Online]. [Accessed 26Mar2014]. Available at: <http://www.nngroup.com/articles/ten-usability-heuristics/>
- Pappano, L. (2012). The Year of the MOOC. *The New York Times*. [Online]. 2Nov. [Accessed: 25Mar2014]. Available at: http://www.nytimes.com/2012/11/04/education/edlife/massive-open-online-courses-are-multiplying-at-a-rapid-pace.html?_r=0
- Reisner, E. R., White, R. N., Russell, C. A., & Birmingham, J. (2004). Building quality, scale and effectiveness in after-school programs. *Washington, DC: Policy Studies Associates*
- Rogowsky, M. (2014). Without Much Fanfare, Apple Has Sold Its 500 Millionth iPhone. *Forbes*. [Online]. 25Mar. [Accessed 26Mar2014]. Available at: <http://www.forbes.com/sites/markrogowsky/2014/03/25/without-much-fanfare-apple-has-sold-its-500-millionth-iphone/>
- Rohrer, D., & Taylor, K. (2006). The effects of overlearning and distributed practise on the retention of mathematics knowledge. *Applied Cognitive Psychology*, **20**(9), pp1209-1224
- Russell, J. W., Kozma, R. B., Jones, T., Wykoff, J., Marx, N. & Davis, J. (1997). Use of simultaneous-synchronised macroscopic, microscopic and symbolic representations to enhance the teaching and learning of chemical concepts. *Journal of Chemical Education*, **74**(3), pp330-334

Safe Web Colours (No Date). Safe Web Colours. [Online]. [Accessed 26Mar2014]. Available at: <http://safecolours.rigdenage.com/index.html>

Schulte, C., & Knobelsdorf, M. (2007). Attitudes towards computer science-computing experiences as a starting point and barrier to computer science. In *Proceedings of the third international workshop on Computing education research* (pp. 27-38). ACM.

Stockwell, G. (2010). Using mobile phones for vocabulary activities: Examining the effect of the platform. *Language Learning & Technology*, **14**(2), pp95-110

Sutherland, R., Facer, K., Furlong, R., & Furlong, J. (2000). A new environment for education? The computer in the home. *Computers & Education*, **34**(3), pp195-212

Szabo, M., & Kanuka, H. (1999). Effects of violating screen design principles of balance, unity, and focus on recall learning, study time, and completion rates. *Journal of Educational Multimedia and Hypermedia*, **8**(1), pp23-42

Thornton, P., & Houser, C. (2005). Using mobile phones in English education in Japan. *Journal of computer assisted learning*, **21**(3), pp217-228.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *J. of Math*, **58**, pp345-363

Vadeboncoeur, J. A., Blackburn, M., & Moje, E. B. (2006). Engaging young people: Learning in informal contexts. *Review of Research in Education*, pp30, 239

Vali, A. (2007). Storytelling: An Effective Method for Teaching Complicated Concepts in Medicine. *Iranian Journal of Medical Education*, **7**(1), pp155-160. [ABSTRACT ONLY]

Vischeck. (No date). Color blind image correction. [Online]. [Accessed 26Mar2014]. Available at: <http://www.vischeck.com/daltonize/>

Vogel, J. J., Vogel, D. S., Cannon-Bowers, J., Bowers, C. A., Muse, K., & Wright, M. (2006). Computer gaming and interactive simulations for learning: A meta-analysis. *Journal of Educational Computing Research*, **34**(3), pp229-243.

Wang, H. (1957). A variant to Turing's theory of computing machines. *Journal of the ACM (JACM)*, **4**(1), pp63-92

Wishart, J. (1990). Cognitive factors related to user involvement with computers and their effects upon learning from an educational computer game. *Computers & Education*, **15**(1), pp145-150

Appendices

Taking into account the problem statement and the background literature, a proposed solution for this project now follows. Risks to the approach are discussed and mitigation plans documented inline.

5.1 Platform and Language

This simulator will be deployed as a mobile application, due to user preference for mobile learning, improved learning outcomes, and the increased likelihood that users will download and interact with the application. Due to improved flexibility, and the open development platform, Android was chosen. To test applications on the iPhone, a costly developer license is required, while Android devices can accept an application through a USB connection (Apple, 2014, Android, no date). There are many more Android devices sold than iPhones, giving access to a larger market (Rogowsky, 2014, Bort, 2013). Android applications are developed in Java which will ease redeployment to a PC application, if there are any issues with Android programming. It also increases opportunities for future expansion; the application logic could be ported to a PC based system in future, integrating with a PC specific interface.

Risk: Author has no experience in Android programming, which will affect timelines, and may make project completion infeasible in the time allocated.

Mitigation plan: A considerable length of time has been set aside to develop the Android interface, as in Table 2 (seven weeks). Time was also spent during Semester 2 to get familiarised with basic concepts of Android development, design patterns for the platform and solutions to problems noted during project proposal discussions. From this, a simple application was created with swipe gesture capture and a basic interface. Support is available within the University of Glasgow Computer Science department if problems are encountered during the project.

However, to further mitigate the risk of unforeseen issues or underestimation of the time required, an evaluation will be made in Week 6. If the project has not progressed sufficiently,

the implementation will be switched to a PC application, completed in Java, which the author has experience with. This switch will be assisted by the system logic being kept separate to Android development code and the evaluation data being transferrable to the new platform.

5.2 Design

The concept of a register machine is, in some ways, complex. An overlying metaphor will be created, to connect it with a real life situation or story, making the theory meaningful to the user. This will improve understanding, and increase motivation. However, it is important that the user is aware it is a metaphor; the analogy must be connected with the register machine concept. This connection and, consequentially, the correlation to CPU functionality, can be made clear through the tutorial and help documentation. These must be accessible throughout the application, and should provide step-by-step instructions for all tasks a user may need to perform. How appropriate the metaphor is can be assessed during the evaluation by questioning the understanding of the register machine concept after use of the application.

Naïve users benefit from a multimedia approach; this will involve pictures and text, but no audio, as this would overload cognitive input channels. There will be some animation to create a focal point of the important actions when they are occurring. For example, the registers will animate as they are decremented or incremented. The interface will be interactive and require active problem solving, as this will encourage elaborative processing, and improve learning outcomes.

The interface will be as clean and clear as possible to reduce distraction; one way to achieve this is to display only the registers which will be affected by a program when it is run. Appropriate colours will be chosen to assure accessibility for the colour blind. Evaluation will ensure that the interface is correctly balanced, with a focal point and unity, whilst remaining usable. This will increase completion rates, and improve accessibility.

As the user has downloaded the application, they will have an intrinsic motivation to learn; however, this can be increased by providing positive feedback, and integrating some game elements into the application. This impacts the implementation of the system logic; it must be able to check a user program for correctness against a predefined requirement for the program. This will allow the system to keep track of what programs the user has correctly developed, give positive feedback, and suggest new tasks for the user to complete. It is hoped the user will be motivated to return by being given these goals. These checks can also be used to ensure the error messages are appropriately parameterised, and can assist the user in fixing any issues. To reduce the number of errors that do appear, commands will be chosen from a predefined list, and all arguments will need to be entered by the user, prior to running the program.

The user must have the ability to save and load programs. This is particularly important in the context of mobile application use; a user may not fully complete one program in the short time they interact with the application. A strong motivation for mobile learning is consistency of materials across all contexts, and this will be achieved by writing to and reading from a text file with a defined structure.

5.3 Evaluation

Before the application interface is programmed, paper prototypes will be developed to assess suitability, along with a questionnaire to document responses. The wireframes will be

evaluated with users who have previous knowledge of the register machine concept. This ensures that feedback can be sought on the appropriateness of the metaphor and design of the interface, without this being influenced by perceptions of the learning task. Assessing learning outcomes at this point would not be useful, as it would be impacted by the evaluator's explanations to the volunteer and the lack of a working prototype. It would be a traditional classroom learning experience, different from the informal mobile learning situation, and would provide skewed data.

When a working prototype is ready, this will be evaluated in naïve users who are comfortable with mobile device use. These users will have their baseline understanding of register machines and CPU functionality queried. They will then use the application, with an evaluator on hand to answer questions. These questions will be recorded, as they will highlight inconsistencies, issues and necessary improvements.

The user will complete a task on the application; develop a simple addition program. They will be asked to "think out loud". This will highlight important aspects of usability; where the user wants to go, what tasks they want to complete and any roadblocks they discover. It will also capture comments about the cohesion and balance of the interface. Confusion with regards to navigation or functionality and negative comments about the design will be particularly important data, as these can impact user persistence.

After completion, the user will be questioned again on their understanding of a register machine. They will be asked how this relates to CPU function. This can be compared to the baseline, and will assess learning outcomes and the appropriateness of the metaphor; can the user separate the concept from the metaphor, and transfer this understanding to the theories supporting CPU functionality. The user will also be questioned about all aspects of the application functionality; for example, were the tutorial, help and error messages useful, did they find the tasks challenging, did they feel they were in control and would they return to complete further tasks? This will show if the application has achieved its goal of being usable and functional and if it, in itself, is a motivational tool. Finally, a subtraction program will be shown to the user within the application, and they will be asked what the goal of the program is; this will determine if the program representation is easy to understand.

Risk: Pre-evaluation will show no design is appropriate.

Mitigation plan: Interface programming does not begin until Week 3. Further evaluation can be carried out in Week 2 to develop a more appropriate design.

Appendix X

- Must run on a modern platform
- Must be educational
 - Must maintain and convey register machine concept
 - Must link register machine concept with CPU functionality
 - Must have tutorial/help
- Must be possible to create a program
- Must be possible to visualise the program on screen
 - Instructions must be visually differentiated
 - Instructions must be visually linked to registers
- Must be possible to run a program, visualising the effect on register data
 - Program must run as expected, each instruction actioned in turn.
- Must be possible to delete a program from screen
- Should be usable by naive user
 - Should provide clear, parameterised error messages
 - Error messages should contain the instruction with the issue
 - Error messages should have steps to resolve the error
 - Should reduce the chance of user error
 - User should not be able to complete incorrect actions
 - User should select instructions from predefined list
 - User should be warned if instructions are not correct (a decrement without a branch, for example)
 - Actions of buttons should be clear
 - Should use standard icons where possible
 - Should label buttons where applicable
 - Should contain an in depth tutorial/help
 - Tutorial/help should contain text and pictures
 - Tutorial/help should detail all steps to create a program
 - Tutorial/help should not contain technical terms

Should allow for undo of action

System status should be clear to user at all times

- Status of “Running”
- Status of “Not Running”
- Affected registers
- Current instruction being completed

- Should allow save of program to a file, and loading of program from a file.

File should have defined structure

File should be widely used format

File should be editable outside application

File should not import if incorrect format

File should not export if program has errors

File should output in Dennett’s assembly language

- Should challenge users to complete tasks

Should keep track of completed challenges

Should provide motivational feedback

- Should have interactive interface

User should be able to create, edit, run and delete a program

- Should be accessible

Should use colours which can be discerned by the colour blind

- Could allow for specifying subroutines.

- Could allow comments to be added to instructions.

- Could allow for breakpoints to be added.

- Could allow for additional registers to be added.

- Would be implemented across multiple devices.

Tool would be flexible and automatically scale for larger/smaller screens.