

AN ENVIRONMENT FOR TEACHING COMPUTATIONAL THINKING TO NON-MAJORS

by

Irfan Khan

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2014

© Irfan Khan 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public for the purpose of scholarly research only.

Abstract

An Environment for Teaching Computational Thinking to Non-Majors

Irfan Khan

Master of Science, Computer Science

Ryerson University, 2014

The fusion of computers and computational thinking is a key part in our transportation systems, communication systems, security systems, financial systems, and in our social and political instruments. These systems have become part of everyday life and the use of computational devices are rising. Despite being dependent on these computational devices, the majority of users remain oblivious to the inner workings of these devices.

We designed a course called CPS650 – Computational Thinking In Our World to illuminate the social, historical and technical context of these systems to students and citizens so they can better understand the technologies that are underpinned by computer science. The course is designed for non-major students and an audience with no previous programming experience.

Table of Contents

1	Introduction	1
1.1	Thesis Statement	1
1.2	Background	1
1.3	Problem Statement	3
1.4	Our Approach	4
1.5	Contribution	5
1.6	Structure of the Dissertation	6
2	Related Work	7
2.1	Introducing Programming Languages	8
2.2	Computational Thinking	8
2.3	Courses with 2D and 3D Animations	9
2.4	Visual Programming Languages & Environments . .	10
2.4.1	Alice Programming Environment	11

2.4.2	StarLogo Programming Environment	12
2.5	Generating Interest In Computer Science	12
3	Experimental Design	17
3.1	Our expectations from the course	17
3.2	Programming Environment	18
3.3	Topics	20
3.3.1	Communication	20
3.3.2	Social Networks	23
3.3.3	Location and GPS	26
3.3.4	Robotics	29
3.3.5	Secrets and Intellectual Property	34
3.3.6	Voting, Finance and Politics	37
3.3.7	Security and Military Computation	40
3.3.8	Transportation and Medical	43
3.4	Qualitative Research	47
3.4.1	Characteristics desirable for a survey	47
4	Results	49
4.1	Communication	50
4.2	Social Networks	53

4.3	Location and GPS	55
4.4	Robotics	58
4.5	Secrets and Intellectual Property	61
4.6	Voting, Finance and Politics	63
4.7	Security and Military Computation	65
4.8	Transportation and Medical	68
4.9	Summary	71
4.10	Survey Justifications and Limitations	74
5	Conclusions	79
5.1	Contributions	80
5.2	Future Work	80
5.3	Limitations	82
A	Lab Descriptions	83
A.1	Communication Lab Description	83
A.2	Social Networks Lab Description	89
A.3	Location and GPS Lab Description	96
A.4	Robotics Lab Description	104
A.5	Secrets and Intellectual Property Lab Description	111
A.6	Voting, Finance and Politics Lab Description	116

A.7	Security and Military Computation Lab Description .	123
A.8	Transportation and Medical Lab Description	128
B	Survey Questionnaire	137

List of Tables

4.1	Communication Lab Survey Results	52
4.2	Communication Lab Survey Results for Duration . .	52
4.3	Social Networks Lab Survey Results	53
4.4	Social Networks Lab Survey Results for Duration . .	53
4.5	Location and GPS Lab Survey Results	57
4.6	Location and GPS Lab Survey Results for Duration .	57
4.7	Robotics Lab Survey Results	58
4.8	Robotics Lab Survey Results for Duration	58
4.9	Secrets and Intellectual Property Lab Survey Results	61
4.10	Secrets and Intellectual Property Lab Survey Results for Duration	61
4.11	Voting, Finance and Politics Lab Survey Results . . .	65
4.12	Voting, Finance and Politics Lab Survey Results for Duration	65
4.13	Security and Military Computation Lab Survey Results	66
4.14	Security and Military Computation Lab Survey Re- sults for Duration	66
4.15	Transportation and Medical Lab Survey Results . . .	70
4.16	Transportation and Medical Lab Survey Results for Duration	70

List of Figures

3.1	Phratch/Scratch programming environment.	19
3.2	RFC 822 - Internet standard format for electronic mail message header.	21
3.3	Script for sending an email.	21
3.4	Sending an email from Phratch.	22
3.5	Script to get twitter followers IDs.	24
3.6	Script to get twitter followers of followers IDs.	25
3.7	Script to converts twitter user_IDs into twitter screen_names	25
3.8	GPS System Scene Setup	26
3.9	Virtual GPS System Locating Mouse-Pointer	27
3.10	Block getting the Geolocation	27
3.11	Small maze with any given rows and columns	30
3.12	Maze generated with maze setup block	31
3.13	The Wall follower or Left-hand rule Algorithm	31
3.14	Solving a maze with Tremaux's Algorithm	32
3.15	One-time pad Encryption Algorithm Script	36
3.16	One-time pad Encryption Algorithm Result	36
3.17	Ranking Candidates In Order of Preference Block	37
3.18	Condorcet Method Script	38
3.19	Yahoo Finance API to Pull Stocks	39
3.20	Stage Setup for Tracking Locations	40
3.21	Visualization for Location-Tracking	41
3.22	Line Follower Autonomous Vehicle	44
3.23	Script to perform a CAT-scan	46
4.1	Results of Communication Lab	51

4.2	Results of the Duration of Communication Lab	51
4.3	Results of Social Networks Lab	54
4.4	Results for the Duration of the Social Networks Lab .	54
4.5	Results of Location and GPS Lab	56
4.6	Results for the Duration of the Location and GPS Lab	56
4.7	Results of the Robotics Lab	59
4.8	Results for the Duration of the Robotics Lab	59
4.9	Results of Secrets and Intellectual Property Lab	62
4.10	Results for Duration of Secrets and Intellectual Prop- erty Lab	62
4.11	Results of the Voting, Finance and Politics Lab	64
4.12	Results for the Duration of the Voting, Finance and Politics Lab	64
4.13	Results of the Security and Military Computation Lab	67
4.14	Results for the Duration of the Security and Military Computation Lab	67
4.15	Results of the Transportation and Medical Lab	69
4.16	Results for the Duration of the Transportation and Medical Lab	69
4.17	Results summary of guided steps for all labs	73
4.18	Results summary of duration for all labs	73
4.19	Results summary of prior knowledge of participants for all labs	75
4.20	Results summary of understanding for all labs	75
4.21	Results summary of visualization for all labs	76
4.22	Results summary of importance for all labs	76

Chapter 1

Introduction

1.1 Thesis Statement

My thesis is that guided labs in a visual programming environment can help non-programmers understand computer science concepts and the technologies that are underpinned by computer science.

1.2 Background

John Lowther professor at Michigan Technological University gathered definitions on computer science from various resources. One of the definitions of computer science found on Lowther's webpage is "Computer Science is the study of the principles, applications, and technologies of computing and computers. It involves the study of data and data structures and the algorithms to process these structures; of principles of computer architecture-both hardware and software; of problem-solving and design methodologies; of computer-related topics such as numerical analysis, operations research, and artificial intelligence; and of language design, structure, and transla-

tion technique.”[42]

Computer science experts are connecting with other disciplines to solve problems related to health care, business, engineering. Computer science concepts such as algorithms, data and data structures are part of Communication systems, Transportation systems and Computing devices, etc. These concepts are designed to solve the problems experienced by everyday users. It is important that the data, data structure and the algorithms processing these structures within communications systems, Transportation systems, etc. and the design methodologies used in these systems should be brought into the light for everyone.

The term computational thinking as introduced by Jeannette M. Wing[43] is a “way of solving problems, designing systems and understanding human behavior, by drawing on the concepts fundamental to computer science.” Computational thinking allows for an average person to understand the fundamental principles of computer science. Jeannette M. Wing highlights the importance of computer science concepts in an average person’s everyday life by some of the examples in her article as described below:

“When your daughter goes to school in the morning, she puts in her backpack the things she needs for the day; that’s prefetching and caching. When your son loses his mittens, you suggest he retrace his steps; that’s backtracking.”[43]

1.3 Problem Statement

The fusion of computers and computational thinking is a key part in transportation systems, communication systems, security systems, financial systems, and in social and political instruments. These systems have become part of the general population's everyday life, and the use of computational devices by individuals is rising. Despite being dependent on these computational devices, the majority of users remain oblivious to the inner workings of these devices.

The lack of awareness of these technologies is partly due to the computer science departments not reaching out to broader audiences. Students from non-majors often find computer science boring because introductory courses offered to non-majors are taught using text-based programming languages. Some have speculated that computer science departments have also been unsuccessful in increasing non-major student's ability to think algorithmically. Students with no prior programming experience often find it hard to understand these introductory courses. Previously, the emphasis was put on teaching non-major students programming but not computer science concepts. In traditional programming courses, a student is given a problem statement and is expected to code a working solution of the problem; while this may be workable for computer science students, it is usually a tough task for non-major students with no guidance. Few (if any) students end up completing the lab within the lab period.

There have also been reports that there is a high rate of failure for introductory courses. The number of computer science degrees awarded to women has decreased, and women consider introductory

computer science courses as boring, asocial and lacking real world application.

1.4 Our Approach

In order to fully take advantage of computational devices, we think that the inner workings of, and the underpinning principles of these devices should become common knowledge to the general population. We want to give students an understanding and working knowledge of systems such as Location and GPS, Communications, Robotics, Social Networks, Transportation, and also provide insight on the principles governing these devices. We came up with the idea of introducing a new course at Ryerson University for non-majors. The course is “CPS650 – Computational Thinking In our World,” designed to illuminate the social, historical and technical context of the aforementioned systems. The challenges we had while designing the labs for the course were as follows:

- Design a virtual system for each week’s topic and create an effective visualization depicting the real-time systems.
- Design the labs such that students can successfully complete each lab within a 1-hour session while maximizing comprehension.
- Make the design simple by keeping the mathematics content low, remembering the diversity of our student’s background.

The goal is to give students a better understanding of the computer science concepts behind each week’s topic; teaching them program-

ming is secondary. To overcome these challenges we designed the labs to guide students through a problem by exercising strategic methods in order to arrive at a solution. These challenges also led to the changes of the programming environment Section 3.2.

1.5 Contribution

The contributions of this thesis are:

- We created guided labs in a visual programming environment which introduce non-CS-major students to concepts in programming, while making their task easier so that they can successfully complete a lab within a 1-hour session.
- We extended Scratch visual programming environment to communicate the inner workings (i.e. algorithms, data and data structures) of the systems such as Location and GPS, Communications, Robotics, Social Networks, etc. by keeping the mathematics content low and eliminating the complex programming syntax.
- We provided non-major students guided lab description, and we created visualization's supporting the systems such as Location and GPS, Communications, Robotics, Social Networks, etc., to give students a better understanding of the systems in a more stimulating fashion.

1.6 Structure of the Dissertation

This dissertation has the following structure:

- Chapter 2 presents background and the related work closest to this thesis. Chapter 2 discusses the introductory courses offered to non-majors at various universities.
- Chapter 3 discusses each week's labs designed for the course CPS650 – Computational Thinking In Our World.
- Chapter 4 discusses the results obtained from the surveys conducted for each week's topic.
- The dissertation closes with a Chapter on future direction and conclusions.

Chapter 2

Related Work

Computer science departments have offered introductory courses for over a decade in an effort to attract students from other disciplines. While no other work we could find directly similar to our research, the rest of this Chapter details some of the work closest to our research. The success of introductory courses to non-majors was attributed to the following factors:

- Mark Guzdial[15] reported the Python programming language was helpful in giving students from non-majors better understanding of programming concepts and algorithms as it freed the students from detailed language syntax.
- Visual programming languages or visual programming environments have been reported to give students from non-majors better understanding of computer science concepts and make introductory courses interesting.

The goals of the introductory courses offered to non-majors were, to introduce programming languages, to generate interest of non-major

students in computer science, and to teach computational thinking. I classified the introductory courses based on these goals.

2.1 Introducing Programming Languages

Mark Guzdial introduced a Media Computation course for non-majors[15] offered at Georgia Institute of Technology. This course used computation for communication as a guiding principle. The course used a version of the Python programming language called Jython[19], which freed students from detailed syntax and made the student's task simpler in learning how to program, and help manipulate sound, images and movies by writing small snippets of code. At the end of the Media Computation course, 97% of students agreed with the question "Are you learning to program?". From Mark Guzdial's work, I speculate that non-majors students like to learn programming if it is made simple by eliminating complex syntax and incorporating examples in the course that have direct application to the real world.

2.2 Computational Thinking

Alex Ruthman, Jesse Heines et al.[35], used the Scratch[37] visual programming environment to introduce a course "Teaching computational thinking through musical live coding." This course was an interdisciplinary general education course and was jointly offered by the computer science and music departments. This course was intended to attract students with arts and music background to computer science. The aim of the course as stated by Alex Ruthman

was to illustrate the potentiality of the Scratch programming environment by “musical-live coding”, and also explain the reason’s for a shift from the web-based environment to the Scratch programming environment. Students created small programs applying programming concepts that resulted in music as an output, and this was done with the help of Scratch sound blocks.

S. Hambrusch, C. Hoffmann et al.[16] took a multidisciplinary approach towards introducing computational thinking to science majors. The course “Introduction to Computational Thinking” was offered to science majors, using Python[30] and Python libraries to teach computational thinking through basic programming tools, visualization, data management and simulation. The course was evaluated based on the entry and exit surveys. Students were asked two questions:

- Taking another computer science course?
- Pursuing a career that requires programming skills?

Results indicated that 60% found the course to be interesting and planned on taking another CS course whereas 40% planned on taking a minor CS course.

2.3 Courses with 2D and 3D Animations

Susan Rodgers[33] work is another example of introducing computer science to non-majors where visualizations and animations play a vital role. Rodger’s goal was to teach students computer science

concepts and programming by creating simple animations and building 2D and 3D virtual worlds. In order to meet this goal, students worked with scripting languages and interactive programming environment like Alice[1], Star Logo[39], Karel++[18] and JAWAA[17]. Web pages, traffics animation, traffic simulation, robotics, and sorting algorithms were a few of the topics covered during the course. By the end of the course, 75% of the students liked the course and the feedbacks from the students were very positive. I speculate that some of the students may have been intimidated by the variety of environments they needed to learn, which could be the reason for the course not being liked by all the student.

Stephen Cooper[11] while teaching a course “Using Animated 3D Graphics To Prepare Novices for CS1”, reported that due to a lack of understanding on the part of the students with the Alice programming environments, they met runtime errors which were cryptic and left the students with no clue to where the source of such errors lay.

2.4 Visual Programming Languages & Environments

M.S. Downes and M. Boshernitsan[23] in a survey paper classified visual programming languages into purely visual languages, hybrid text visual systems, programming-by-example systems, constraint-oriented systems, and form-based systems. Their survey highlights the examples of programming systems of the 80’s and 90’s which were the predecessors of today’s visual programming languages. The Alternate reality kit (ARK)[38] designed by R.Smith at Xerox Parc,

was implemented in Smalltalk-80. ARK was designed to help users create 2D animated environments. Visual Imperative Programming (VIPR)[9] developed by Citrin et al. at the University of Colorado was designed to be an object-oriented language, and relatively easy to learn and use. Prograph[12] developed by Cox, Pietryzkowsky in 1990, and Forms/3[5] developed by Burnett in 1994 were two other examples of programming languages of the 90's. Prograph and Forms/3 languages were general purpose programming languages, where Forms/3 emphasized data abstraction.

2.4.1 Alice Programming Environment

Alice is a 3D programming environment, and it was created at Carnegie Mellon University[1]. With Alice non-programmers can create animated 3D virtual worlds[20]. The Alice programming environment freed students from writing language syntax, whereas in programming language one has to write the syntax to get the output, which was difficult for non-programmers with no prior programming experience. Programming in Alice is done by dragging graphical elements onto the code editors to perform an action, these graphical elements are functional commands and methods that help program an object. Cooper et al.[10] reported that “The mechanism for generating code relies on visual formatting rather than details of punctuation. The gain from this no-type editing mechanism is a reduction in complexity.”

2.4.2 StarLogo Programming Environment

Starlogo TNG[39] is a visual programming learning environment, similar to the Scratch programming environment. StarLogo was developed by MIT Teacher Education Program. The goals of StarLogo TNG as reported by MIT teacher education Program[39]:

- “Lower the barrier to entry for programming with a graphical interface where language elements are represented by colored blocks that fit together like puzzle pieces.”
- “Entice more young people into programming through tools that facilitate making games.”
- “Use 3D graphics to make more compelling and rich games and simulation models.”

Programming in StarLogo is done by dragging pieces of blocks onto to the scripting area for an object to perform an action.

2.5 Generating Interest In Computer Science

Mona Rizvi, Thorna Humphries[32] and a few faculty from the psychology department at Norfolk State University, introduced a CS0 course using Scratch. The goal was to improve the retention, performance and attitude of at-risk students, many of whom had weak mathematics background and struggled when placed directly in CS1. Surveys were conducted among the students from CS0 and CS1 courses. The CS0 students reported greater commitment to their major and greater programming self-confidence. This new CS0 course

managed to retain at least 70% of students in their major. We speculate that the Scratch programming environment freed students from complex syntax and gave students better understanding about the algorithms and programming concepts.

Lori Pollock, Kathleen McCoy et al.[29] designed a summer program to generate high school girls interest in computer science. The program was named Girl's **POWER** (**P**rogramming **O**f the **WE**b **R**ocks). The program had 20 participants. The goals of the summer program were as follows:

- Focus on web programming and give high school girls some idea about how social- oriented applications like electronic party rsvp's, shopping carts, chat rooms, and games are created.
- Career awareness of speakers and role models by sharing their computer science experience each week.
- Education awareness sessions and networking opportunities.

Results obtained from this program were quite positive. The survey was administered to evaluate the contribution of the POWER program in increasing high school girls knowledge. 71% responded "a lot" while 24% responded "some". In terms of confidence, 71% indicated improvement.

Casey Alt, Owen Astrachan et al.[2] reports that topics that arise from social networks can generate non-major student interest towards computer science. Casey Alt, Owen Astrachan et al. speculate that a course on social networks can generate interest in women from non-majors towards computer science. Women find introductory courses

offered by computer sciences department to be boring as reported by Fisher, Margolis, and Miller[24]. Casey Alt, Owen Astrachan et al. speculate that a course on social networks will engage women's interest in real world examples of computer science via social networking applications. The developers of the course speculate that the topic that arose from social network can be modeled to introduce computer science to non-majors. One example module reported by developers of the course is the Power Law distribution. Casey Alt and Owen Astrachan et al. report that they would like to merge these modules into existing courses in mathematics, statistics, and computer science at all levels of high school. The modules for the course are reported to be still under development.

A survey conducted in the year 2013 on the use of social networking sites by PewResearch Internet project[26] reports 73% of online adults use social networking sites.

From the related work, I speculate that students like to learn programming and computer science concepts if the introductory courses are freed from complex syntax. Visual programming languages and the Python programming language are reported to be successful in giving better understanding of the computer science concepts. Students find introductory courses interesting if they include real-time examples. I speculate Scratch programming environment to be helpful in teaching non-major students computer science concepts and computational thinking as it eliminates the problem of students making syntax errors. Moreover, there are no run-time errors reported while working on the Scratch environment. The Alice environment is reported to have run-time errors that are frustrating to students

while working on the labs. StarLogo TNG is designed to teach subjects such as biology, chemistry, physics, and not for computational multimedia projects. Scratch has an open source implementation which was key to designing the labs for the course CPS650 (Computational Thinking In Our World).

Chapter 3

Experimental Design

This Chapter describes the methodology used in and the extensions made to a visual programming environment to support a variety of technologies. The goal is to communicate the computer science behind each topic via programming. We created a simplified version of a real system based on each topic mentioned in Section 3.3 for the students to work on in each week’s lab. The students working on these labs are assumed to have no prior programming experience, no previous understanding of the technology and minimal mathematics skills. We made all the labs “guided” to help the students program their way to a successful outcome within the designated 1-hour lab, despite their lack of previous programming experience.

3.1 Our expectations from the course

By the end of the course we expect students to illustrate the key programming concepts underpinned in systems such as location and GPS, Communications, Robotics and Social Networks, etc., and being presented in each lab. We also expect students to learn the ways

in which ordinary citizens would benefit from the capacity to program. We would also like students to understand the limitations of any program such as location and GPS, Communications, Robotics and Social Networks, etc.

Note that we **do not** expect students to learn to create algorithms by the end of the course.

3.2 Programming Environment

The specific programming environment for the course is Phratch. Phratch/Scratch is a visual programming environment, based on the Scratch project from MIT[37].

The first version of Scratch was publicly launched in May 2007. Since then Scratch has become very popular among young users, with Scratchers from around the world uploading more than 1,500 new projects to the site every day[31].! The goal of Scratch stated by Resnick and Maloney of the Lifelong Kindergarten Group[21] is to help young people (ages 8 and up) develop 21st century learning skills with important mathematical and computational ideas, while also gaining a deeper understanding of the process of design. With Scratch, kids can create their interactive stories, games, music, animation, and share their creations with one another on the Web[22].

Programming is done by dragging pieces from the blocks panel into the script area. Most programs will start running by clicking the green flag in the top right of the stage. Most programs will start with “when flag checked” block, which you will find in the Control category in the left panel. Besides Control category, there

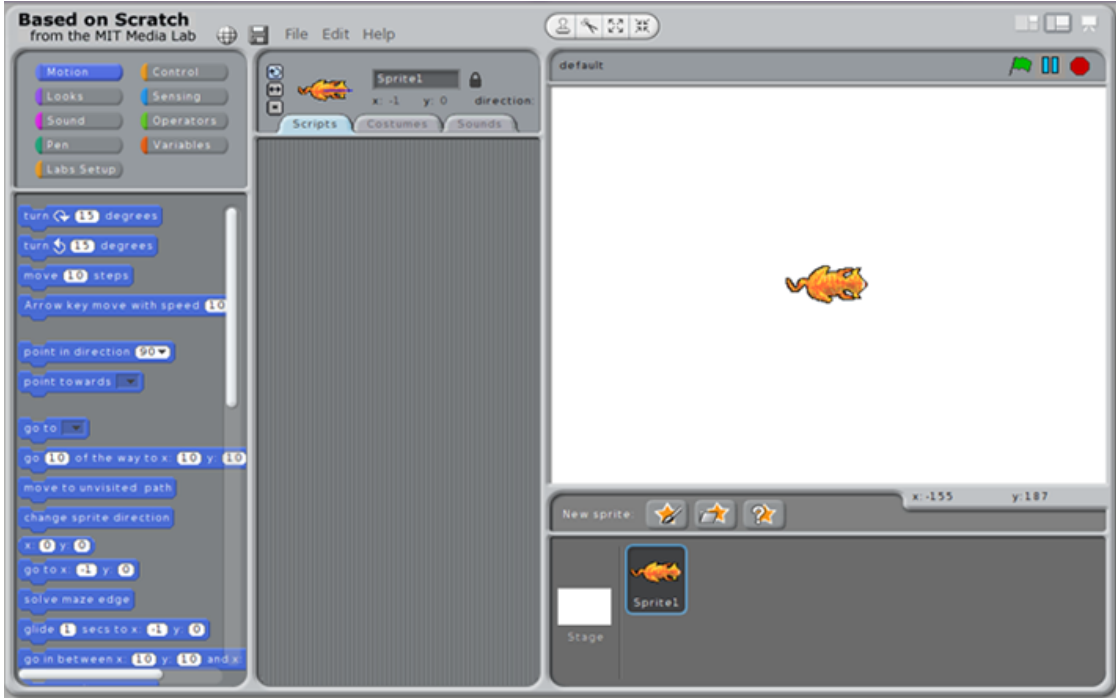


Figure 3.1: *Phratch/Scratch programming environment.*

are also Operator, Look, Sound, Pen, Sensing, Motion, and Variable categories.

Phratch[28] is the Scratch port to Pharo[27]. We chose Phratch to support the labs since Pharo was our main coding environment. We extended Phratch/Scratch by adding more blocks and a new Labs Setup category as shown in figure. 3.1. The extended blocks implement some of the trickier parts to make student tasks easier since some of the labs contain fairly complex algorithms, and we want students to complete in 1-hour. Our goal is to give the students better understanding of the topics mentioned in Section 3.3 and computer science concepts by keeping the topics as simple as possible; teaching them programming is secondary.

3.3 Topics

Each topic described below is a one-week, self-contained module in the course CPS650 – Computational Thinking In our World. Each of these topics consists of 3 hours of lecture prior to the 1- hour “directed lab.” The weekly lab allows students to work on a simplified version of a real system based on that week’s topic.

3.3.1 Communication

Before exploring a simplified version of a real-time system, the students will have some idea of various protocols such as TCP, IP, and SMTP from a 3 hour lecture.

The aim of the communication lab is to give the students some understanding of Internet email messages, the important components involved in building up a message and the protocols, to send it over the Internet. Students are supposed to build an email message and send it, by dragging “**add rfc822 header _ with _ to _**” and “**ssl send _**” or “**smtp send _ server _**” onto the scripting area of the Phratch programming environment. The functionality of the “**add rfc822 header _ with _ to _**” block for the students, is to help them build up an email message by choosing various message headers like To, From, Cc, Bcc, Subject and Body from the pulldown of the “**add rfc822 header _ with _ to _**” block, as shown in figure. 3.2.

The lab has guided steps shown in Section A.1, that students follow, in order to successfully build the script for sending an email message from Phratch environment figure. 3.3. When the script is started by clicking the green flag in the top right of the stage of

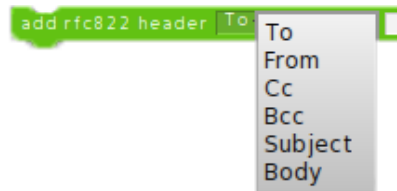


Figure 3.2: *RFC 822 - Internet standard format for electronic mail message header.*

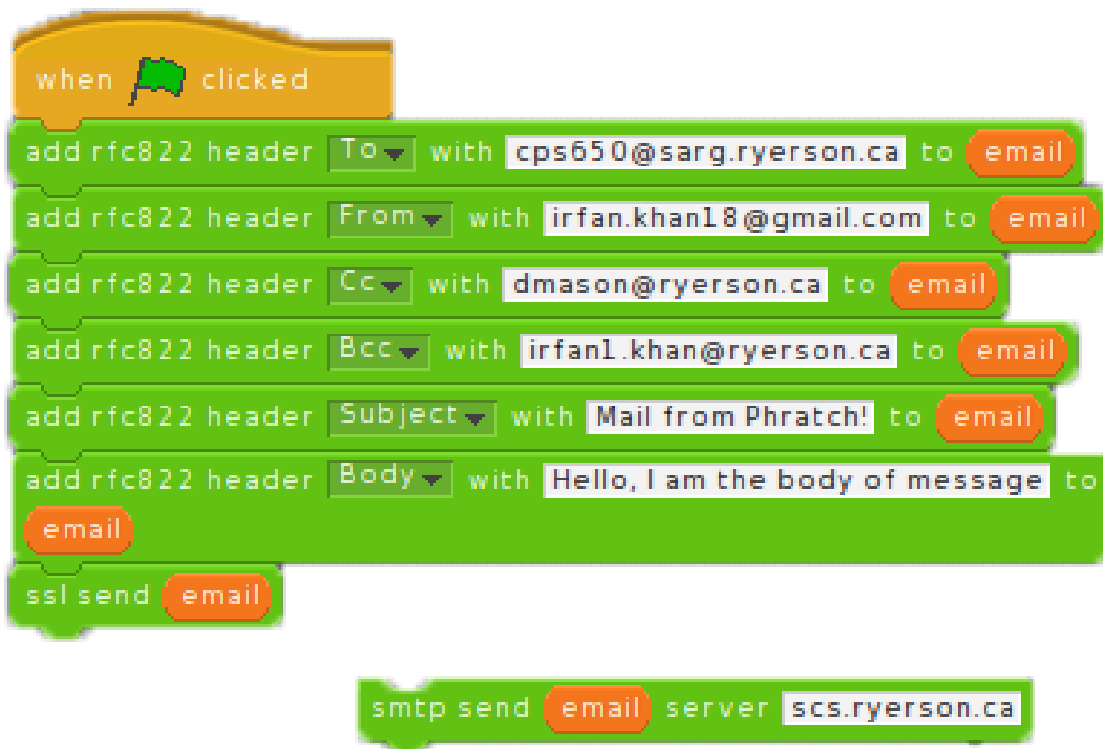


Figure 3.3: *Script for sending an email*

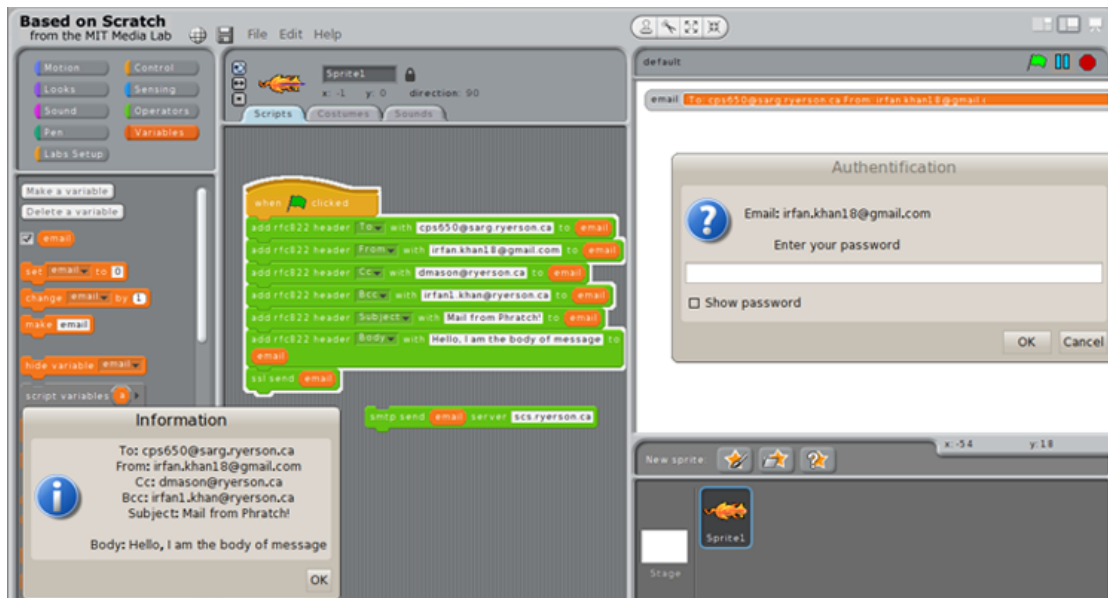


Figure 3.4: *Sending an email from Phratch.*

the Phratch environment, an information window pops up, which displays the email message built by the user, seen in figure. 3.4. If the user chooses to send an email using the “**ssl send _**” block, an authentication window pop’s up, asking the user to enter a password for the sender’s email account as shown in figure 3.4. The “**ssl send _**” block, is the secure way of sending an email message. The user can also try sending an email using “**smtp send _ server _**” block, which is a simple, but insecure, way of sending mail and doesn’t require any authentication, but only works with email servers that accept unauthenticated email messages. Blocks designed to support this lab are “**add rfc822 header _ with _ to _**”, “**ssl send _**”, “**smtp send _ server _**”.

3.3.2 Social Networks

The aim of this lab is to give student's some understanding about web Application Programming Interfaces(API), and how APIs can be used to share, or pull information from Social Networking websites. The students had guided steps, to help them achieve three tasks for this lab:

- Use Twitter REST “Get followers/IDs” API v1.1, to get the followers IDs of their Twitter account, add them to a list, and then iterate over the list, to get the number of followers (i.e. count the number of IDs).
- Get followers of followers IDs into a new list.
- Use the Twitter REST “user lookup” API v1.1, to convert IDs into `screen_names`.

To achieve the above tasks from Phratch environment, we designed “**Json extract _ IDs into _**”, “**Json extract screen name for id _**”, “**iterate_**”, “**replace old item of _ with new _**” and “**current item _**” blocks. The functionality of each of these blocks is described below:

- **Json extract _ IDs into _** block: The functionality of this block was to communicate with Twitter “Get followers/IDs” API, and get the followers IDs into a list.
- **Json extract screen name for ID _** block: This block was designed to communicate with Twitter “user lookup” API, it converts any given twitter `user_id` into the twitter `screen_name`.

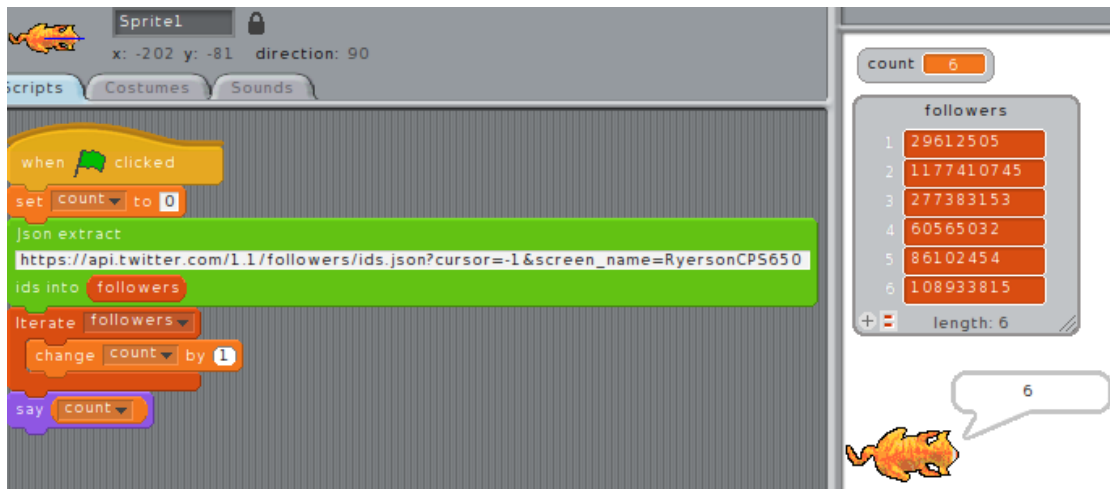


Figure 3.5: Script to get twitter followers IDs.

- **iterate_** block: The functionality of this block was to iterate the list chosen by the user from the pulldown.
- **replace old item of _ with new _** block: The functionality of this block was to replace the old item of the list with a new item.
- **current item _** block: The functionality of this block was to answer the current item being iterated from the list under the iterate block.

The script that students were supposed to build with the help of guided steps shown in Section A.2 to achieve task 1, can be seen in figure 3.5. The script to achieve followers of followers IDs is shown in figure 3.6, and you can also see, a new list named “foo” which has followers of followers of CPS650 course twitter account. The script to convert twitter `user_IDs` into twitter `screen_names`, by using twitter “user lookup” API is shown in figure 3.7. The list that had followers

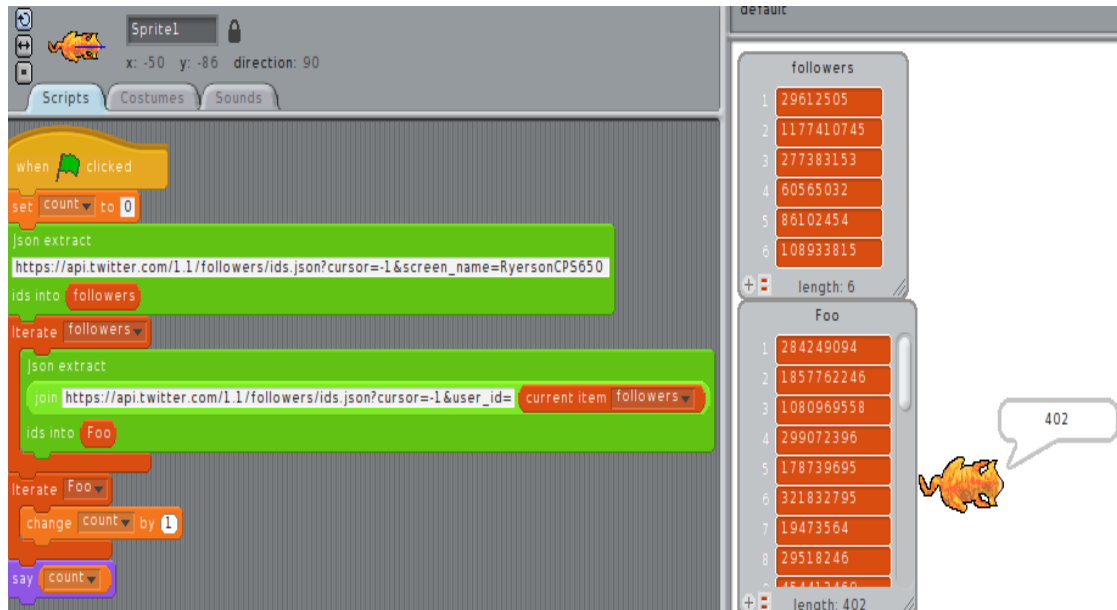


Figure 3.6: Script to get twitter followers of followers IDs.

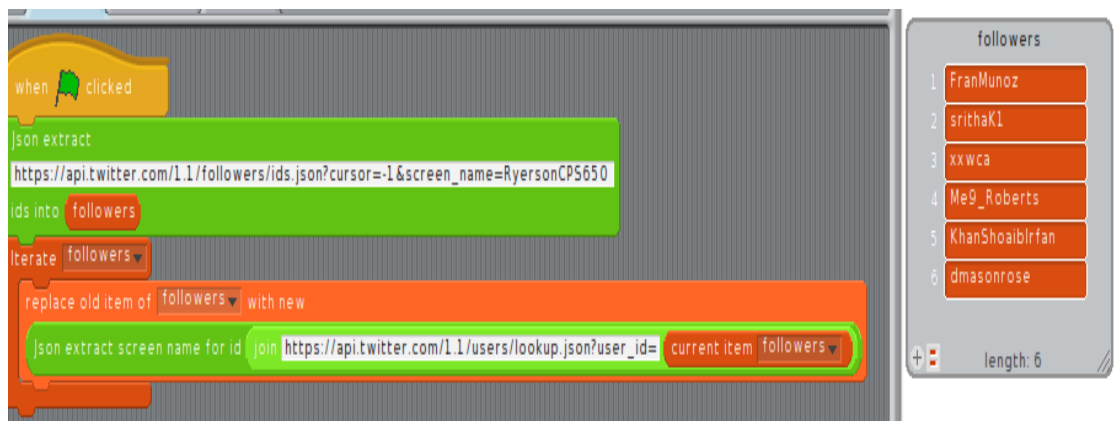


Figure 3.7: Script to convert twitter *user IDs* into twitter *screen names*

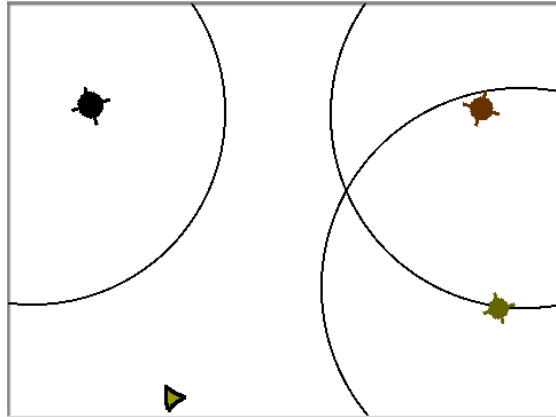


Figure 3.8: *GPS System Scene Setup*

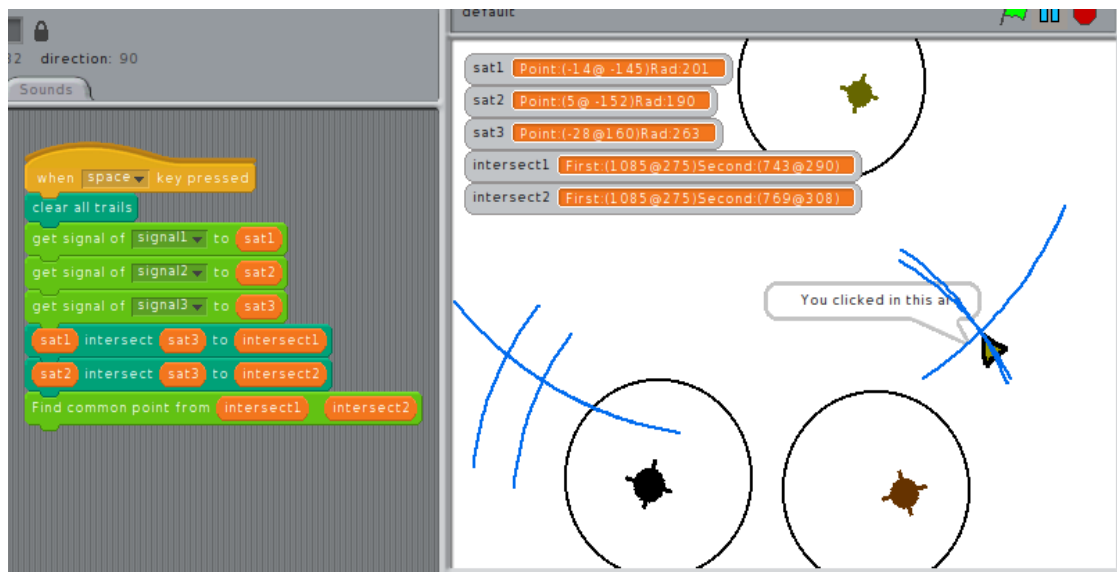
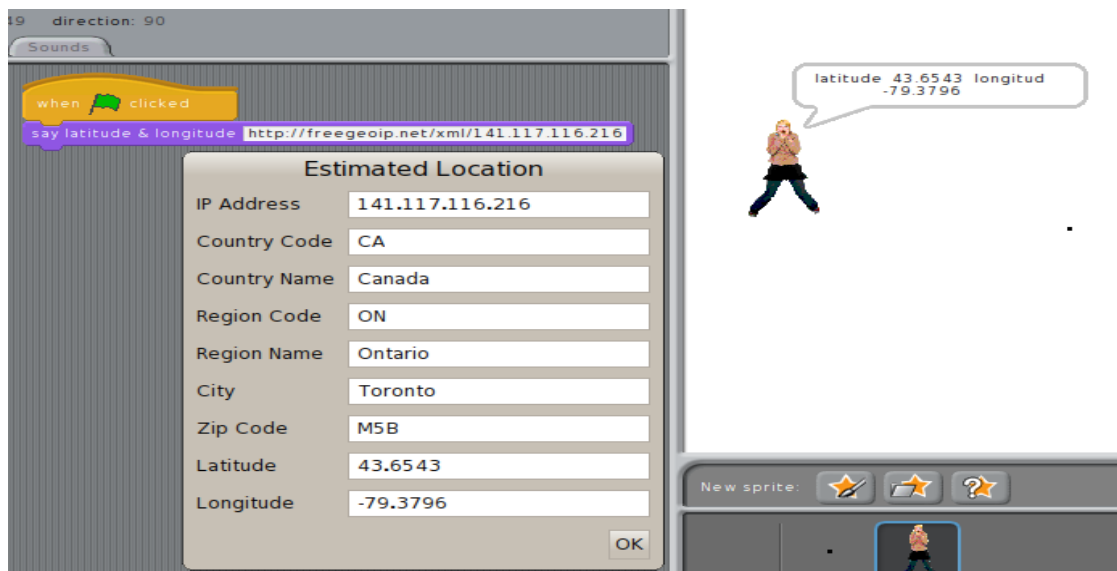
IDs earlier has been replaced by the twitter `screen_names`.

3.3.3 Location and GPS

The aim of this lab was to give student's an understanding of how the Global Positioning System (GPS) works. The Global positioning system (GPS) is used on computing devices like mobile phones, and it has also become part of the general population's daily life for travel purposes. GPS helps in preventing transportation accidents and helps in search and rescue efforts[14].

We wanted to give students an idea of how GPS works to provide location. For the purpose of this lab, we created a 2D virtual GPS system scene which was part of the setup. We designed an animation with three satellite images and three signal images (i.e. sprites), that drift across the stage of the Phratch environment. The three signal images follow the satellite images and leave a circle - like visualization.

Students are supposed to create a script in order to locate the

Figure 3.9: *Virtual GPS System Locating Mouse-Pointer*Figure 3.10: *Block getting the Geolocation*

mouse pointer on the stage of the Phratch environment with the help of guided steps shown in Section A.3. The idea was to create a block that would get the location (i.e. center) and distance of three satellites from the mouse pointer. Once we have a location and distance of three satellites, students will draw three circles using the location and distance as the radius to find the intersecting points. The common point among the intersecting points would be the resulting location of the mouse pointer. The blocks we designed to support this topic are as follows:

- **GPS setup** block: The functionality of this block was to create an animation of virtual GPS system. This block added three satellite images (i.e. Sprites) and signal images (i.e. Sprites), these sprites have some predefined script that drifts the satellite images and leaves a circle-like visualization while drifting on the stage shown in figure 3.8.
- **signal _ with radius _** block: The functionality of this block is to follow the drifting satellite across the stage of the Phratch environment and leave a circle-like visualization with any given radius shown in figure 3.8.
- **get signal of _ into _** block: The functionality of this block is to get the location (i.e. center) of the satellite and the distance (i.e. radius) to the mouse-pointer.
- **_ intersection _ to _** block: The functionality of this block is to get the set of intersection of points of the 2 given locations and draw the arcs onto the stage of Phratch environment.

- **find the common point from** _ _ block: The functionality of this block is to display a common intersecting point on the stage of the Phratch environment.

The script that, students are supposed to achieve in locating the mouse pointer is shown in figure 3.9.

We also designed **say latitude & longitude** _ block that answer's the Geolocation as shown in figure 3.10. It uses the Freegeoip REST API[13] to locate the Geolocations of the IP addresses.

3.3.4 Robotics

The aim of this lab is to give students an understanding of how robots can be programmed to solve problems. For the purpose of this lab, students are supposed to script the robot (i.e. sprite) to solve a Maze, using the Wall follower or Left-hand rule algorithm[25] and Tremauxs Algorithm[25].

Maze Generation

The blocks, that we designed to generate the Maze, using depth-first search algorithm were, “**small maze rows** _ **columns** _” and “**maze setup**.” The functionality of both of these blocks is as follows:

- **small maze rows** _ **columns** _ block: The functionality of this block is that it can generate a random smaller maze on the stage of the Phratch environment with any given number of rows and columns chosen. Figure 3.11 shows a smaller maze generated with four rows and four columns.

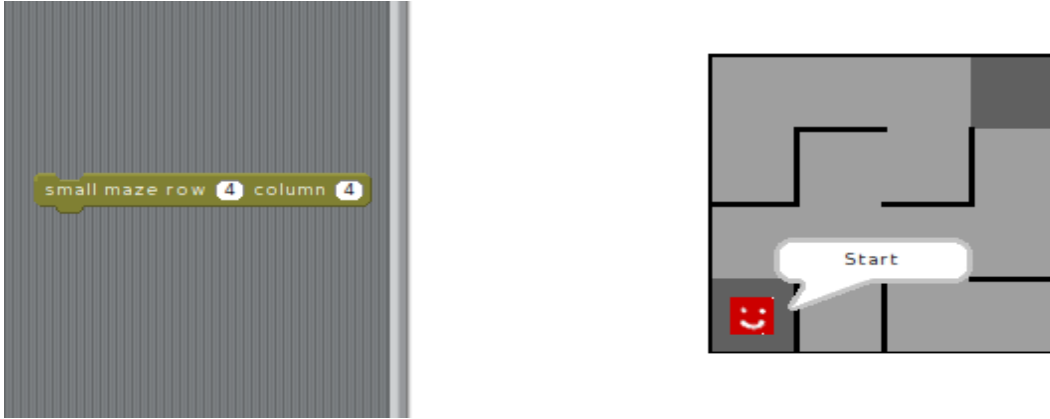


Figure 3.11: *Small maze with any given rows and columns*

- **maze setup** block: The functionality of this block was to generate a random maze each time. The maze generated using this block covers the entire stage of Phratch environment, as shown in figure 3.12.

We want students to try solving the smaller maze first, before solving the main maze, generated by the **maze setup** block.

Maze solving

The blocks that we designed for students to help solve the Maze using the Wall follower or Left-hand rule algorithm, and Tremauxs Algorithm are described below:

- The Wall-follower or Left-hand-rule Algorithm

The wall follower algorithm is very simple. According to the wall follower algorithm every time the robot moves on the maze, it has to follow the wall on the left-hand side of it. The blocks we designed to support this algorithm are as follows:

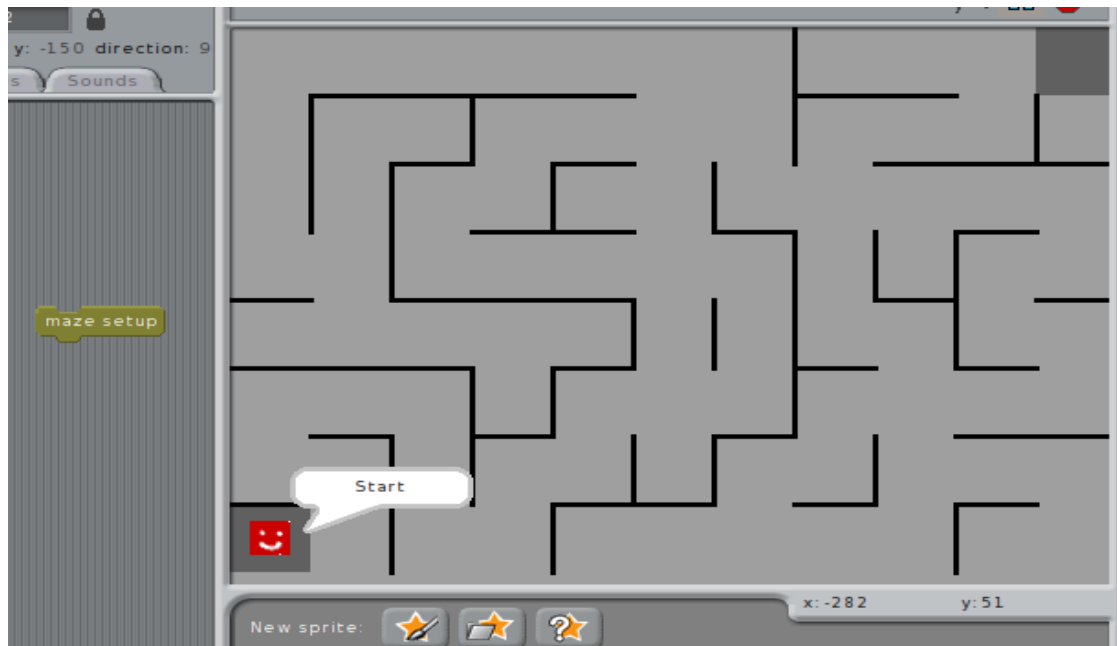


Figure 3.12: Maze generated with **maze setup** block

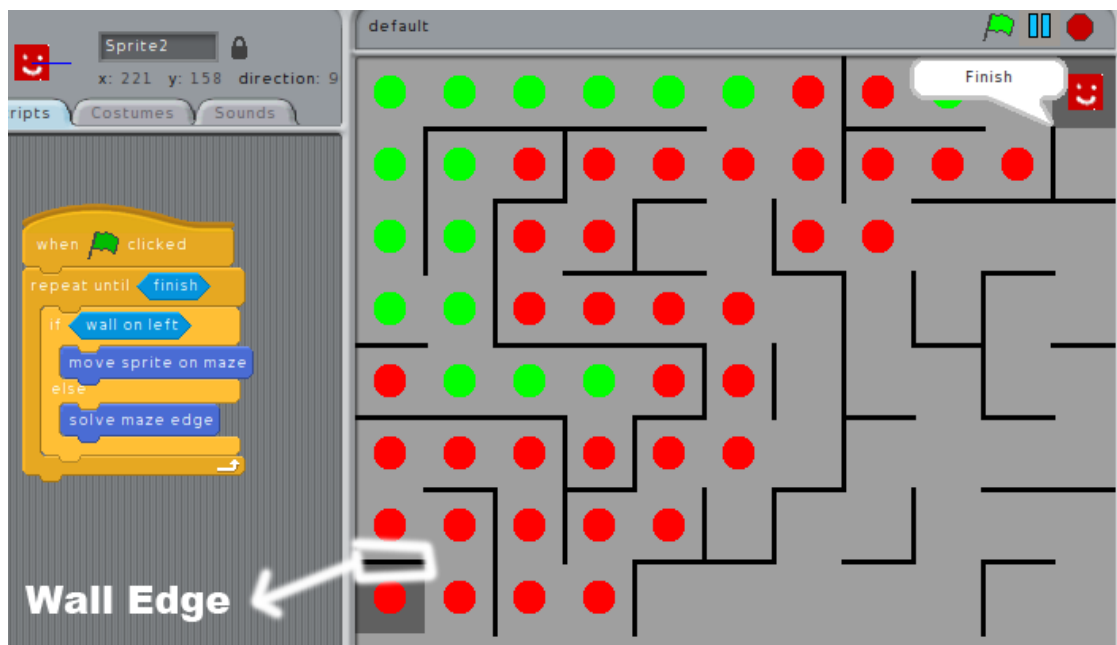


Figure 3.13: The Wall follower or Left-hand rule Algorithm

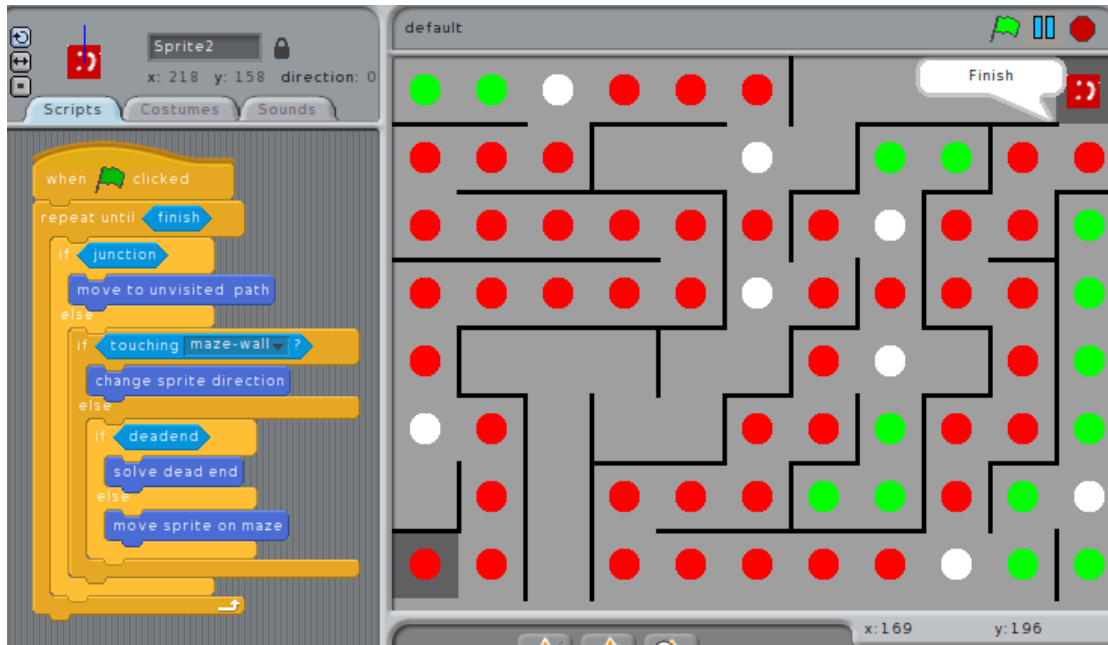


Figure 3.14: *Solving a maze with Tremaux's Algorithm*

- **finish** block: The functionality of this block is to sense if the robot (i.e. sprite) has reached the finish point of the maze.
- **wall on left** block: The functionality of this block is to sense if there is a wall on the left hand side of the robot (i.e. sprite).
- **move sprite on maze** block: The functionality of this block is to move the robot (i.e. sprite) on the maze. This block will leave a red circle on the maze indicating the cell is visited once, otherwise green if the cell is visited twice by the robot.
- **solve maze edge** block: The functionality of this block is to move the robot (i.e. sprite) forward and turn left if the

robot reaches the wall edge of the maze. One of the wall edge is shown in figure 3.13.

The script that students were supposed to achieve with the help of guided steps is shown in Section A.4 and figure 3.13. In order to give more clarity to the students on the path taken by the robot in figure 3.13 the red circles on the maze indicate the cells visited once by the robot, and the green circles indicate the cells visited twice by the robot.

- **Tremaux's Algorithm**

Tremaux's Algorithm was invented by Charles Pierre Tremaux. The rules for Tremaux's algorithm are as follows:

- Leave a trail behind, while moving on the maze.
- No cells visited more than twice.
- If reached a junction(i.e. cell where robot has at least two unvisited paths), mark the junction and take a random unvisited path.
- If no unvisited path exists travel back to the previous junction to take another path.

The blocks we designed to support this algorithm are as follows:

- **Junction** block: The functionality of this block is to sense if the robot is at the junction (check if the robot has at least, two unvisited paths). This block leave's a white circle, identifying the cell as the junction.

- **move to unvisited path** block: The functionality of this block is to turn the robot towards the unvisited path. If there are no unvisited paths this block will turn the robot towards the path not taken more than twice.
- **touching maze-wall** block: The functionality of this block is to sense if the robot is touching any wall of the maze.
- **change sprite direction** block: The functionality of this block is to change the direction of the robot if it is touching any of the maze walls.
- **dead end** block: The functionality of this block is to sense if the robot has walls on the front, left, and right hand sides of it.
- **solve dead end** block: The functionality of this block is to turn the robot back if it has reached a dead end.

The script that students were supposed to achieve with the help of guided steps is shown in figure 3.14. In figure 3.14, the white circles on the maze indicate the junctions.

3.3.5 Secrets and Intellectual Property

The aim of this lab is to give students some understanding of how Encryption is done. Encryption is the process of transforming the message or data into an unreadable form, such that only intended recipients can read it. For the purpose of this lab, we want to focus on the one-time pad encryption algorithm[34]. The blocks we designed to support this algorithm are as follows:

- **Enter Message – Create secret key –** block: The functionality of this block is to take the message that needs to be encrypted and the secret key generated by the user.
- **message** block: The functionality of this block is to answer the length of characters in the message that needs to be encrypted.
- **pick letter from message** block: The functionality of this block is to pick letters one-by-one from the message that needs to be encrypted.
- **pick letter from secret key** block: The functionality of this block is to pick letters one-by-one from the secret key.
- **encrypt and push result –** block: The functionality of this block is to get the corresponding letter when the numerical value of the message character is added, with the numerical value of the secret key character.

Example: if letter C's numerical value is 2 and letter B's numerical value is 1, the sum of C+B would result in 3, so this block will answer D as the resulting encrypted letter of the numerical value 3.

The script that students are supposed to achieve with the help of guided steps shown in Section A.5 and figure 3.15. Figure 3.15 shows the user is being asked to enter the message for encryption. Figure 3.16 shows the encrypted result and a small summary in the information window providing details about the message entered, the secret key set and the encryption result.

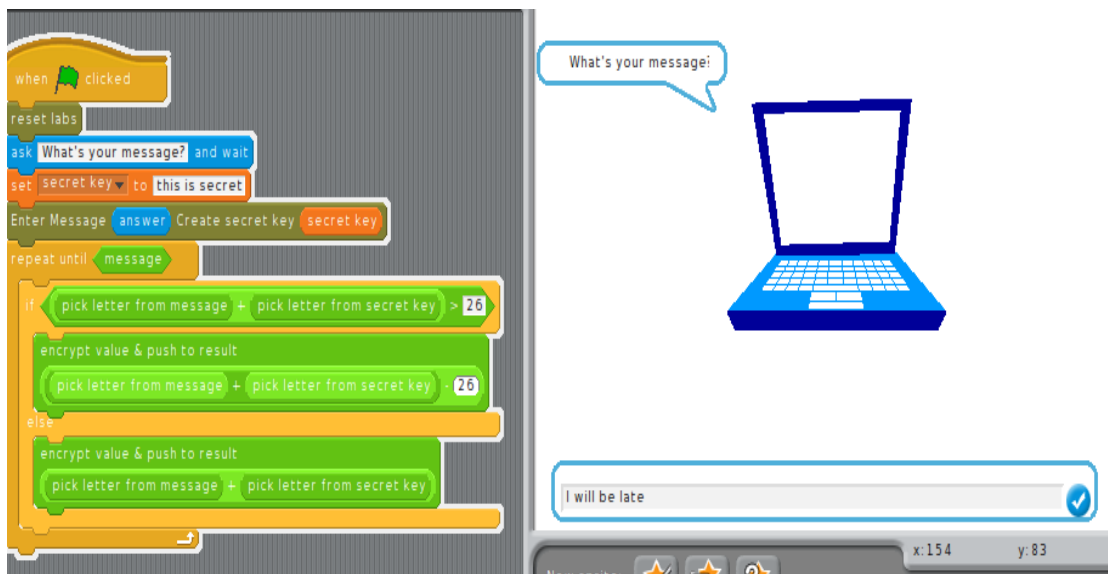


Figure 3.15: *One-time pad Encryption Algorithm Script*

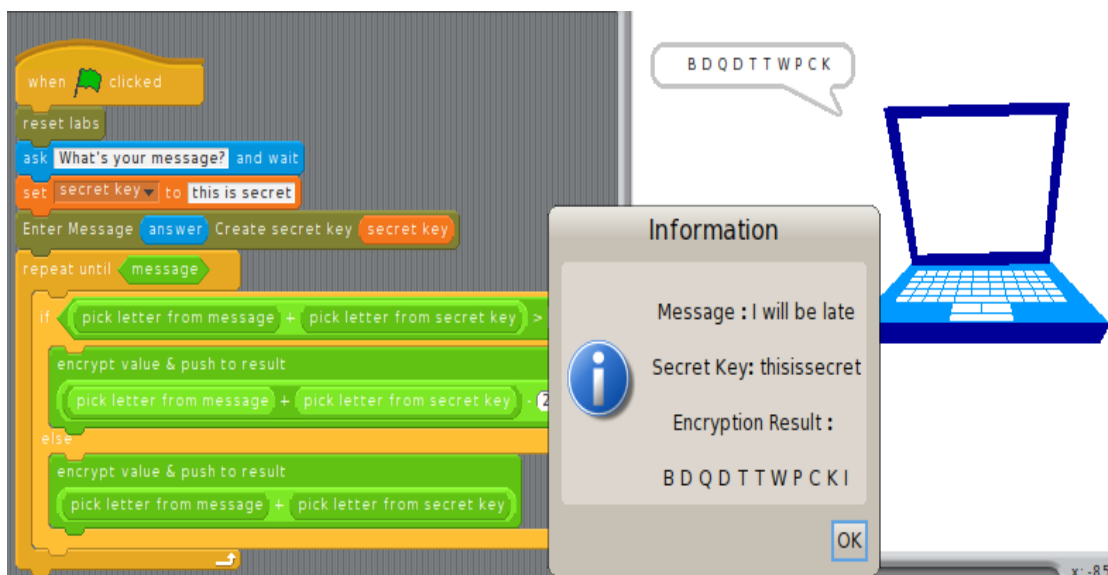


Figure 3.16: *One-time pad Encryption Algorithm Result*

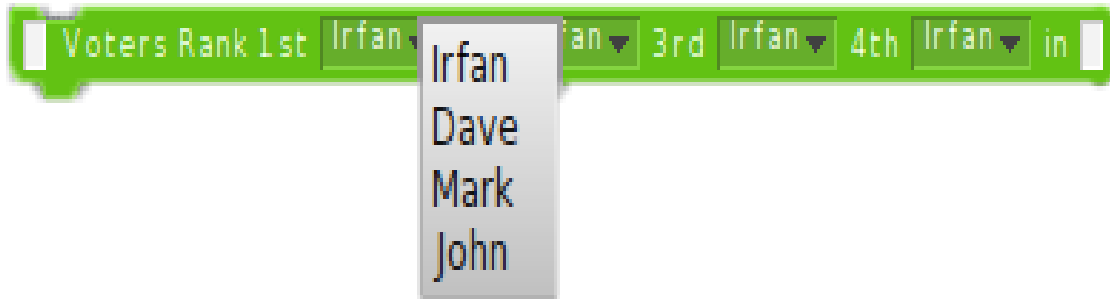


Figure 3.17: *Ranking Candidates In Order of Preference Block*

3.3.6 Voting, Finance and Politics

The aim of this lab is to give students some understanding about Voting systems and Finance (the stock market).

Voting

The goal for this topic is to give students understanding about how Condorcet voting methods work in choosing a winner of the election. Condorcet methods are used when voters rank the candidates in order of their preference. A Condorcet method does a one-on-one match up among all the other candidates, and the candidate with the majority preference is declared the Condorcet winner. The blocks we designed to support this lab are as follows:

- **_ Voters Rank 1st _ 2nd _ 3rd _ 4th_ in _** block: The functionality of this block is to generate voting data by taking the number of votes and the choice of candidate preference from the pulldown and store everything in a variable. The pulldown has four candidate's names, as shown in figure 3.17.
- **display voters data from _** block: The functionality of this

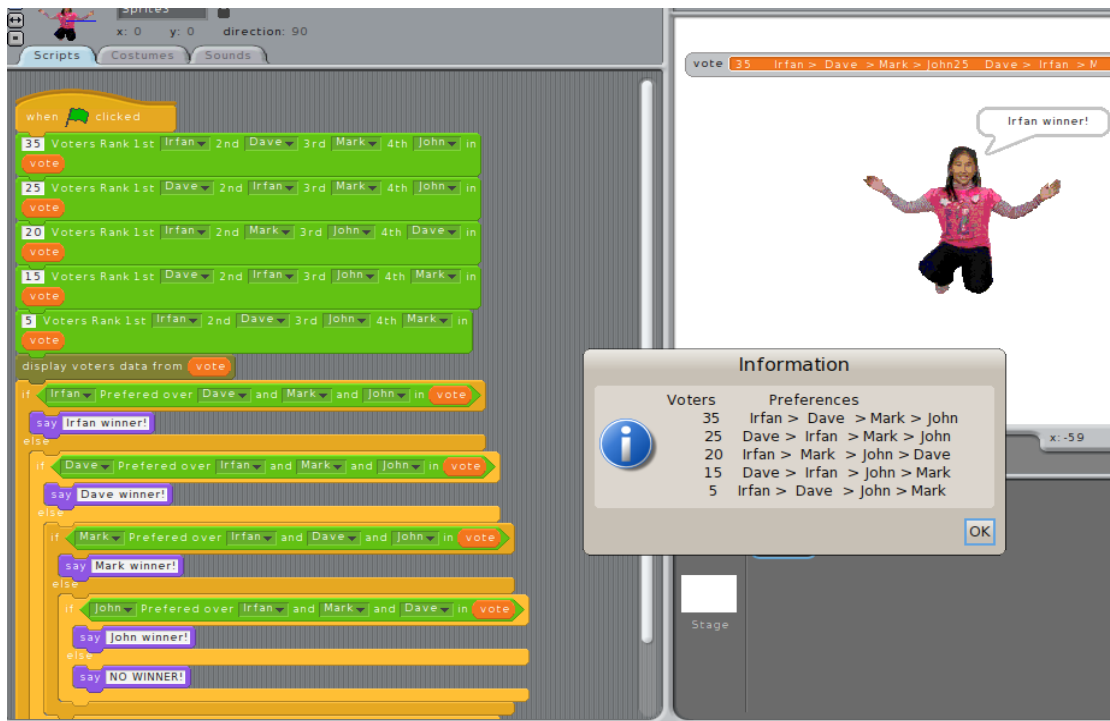
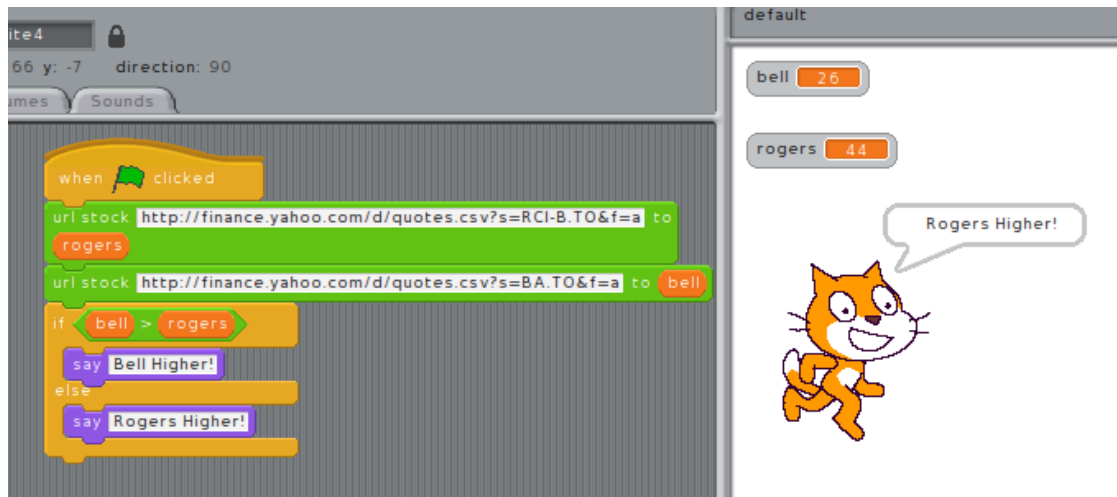


Figure 3.18: *Condorcet Method Script*

Figure 3.19: *Yahoo Finance API to Pull Stocks*

block is to display the voter data generated as shown in the information window of figure 3.18.

- **_ preferred over _ and _ and _ in _** block: The functionality of this block is to perform a one-on-one matchup of the preferred candidate over the other candidates and check if the preferred candidate is the Condorcet winner.

The script that students are supposed to achieve with the help of guided steps is shown in Section A.6 and figure 3.18.

Finance

For the finance topic, we have students focus on stock markets. We want to give students an idea about how web applications can pull stock information from any stock exchange using stocks API.

We used **YAHOO FINANCE API**, which can help users get Toronto Stock Exchange (TSX) rates. We designed a block, **url**

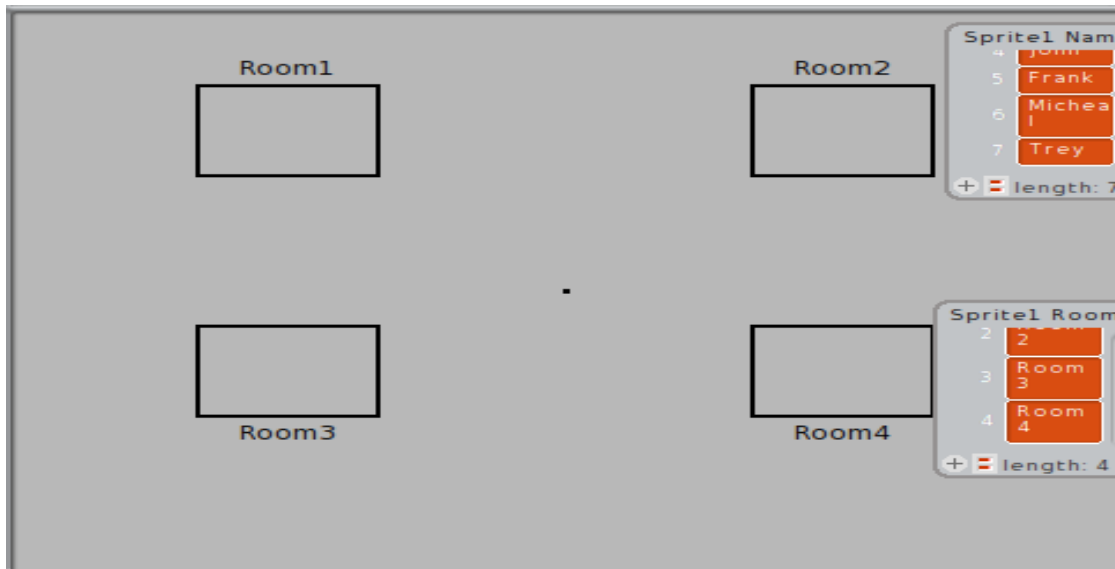


Figure 3.20: *Stage Setup for Tracking Locations*

stock _ to _ block. The functionality of this block is to get the stock rates of any company. Students have the freedom to design their own task. For example, students can get the asking price of the BELL and Rogers companies, using the Yahoo finance API and check which company has the higher asking price, the script is shown in figure 3.19.

3.3.7 Security and Military Computation

The aim of this lab is to give student's an understanding of how location tracking works. Government officials uses tracking for surveillance purposes. It is also used by the general population to track their shipments, or tracking the transactions of a bank account. Location-tracking is the combination of Geographic Information Systems (GIS)[3], Global Positioning System (GPS)[14], Radio Frequency

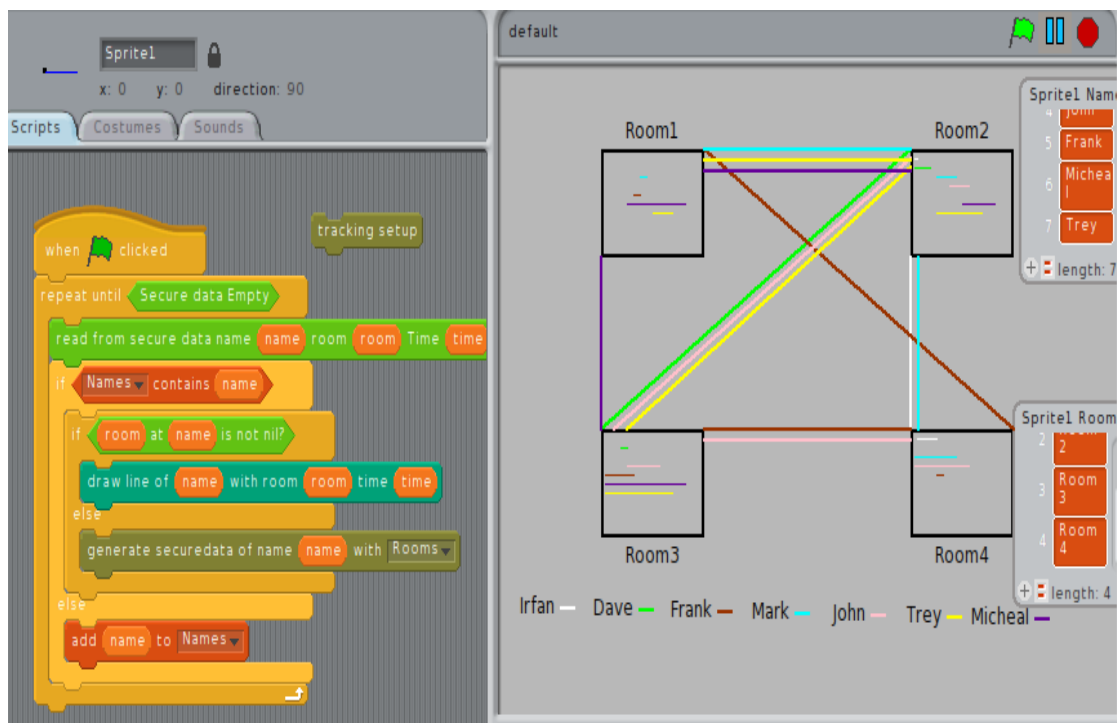


Figure 3.21: Visualization for Location-Tracking

Identification (RFID)[4] and Wireless Local Area Network (WLAN)[3].

The goal for this lab is to provide students with an understanding of how, security tracking software is helpful in creating a visualization of secure data. Students will access fabricated secure data, which contains information about fabricated persons. The information includes the name of the person, rooms that a person has been to and the time spent in each room. Students will use blocks to access the secure information and create a visualization, which will help students track each person one-by-one.

The blocks we designed to support this topic are as follows:

- **tracking setup** block: The functionality of this block is to generate fabricated secure data and add an image with rooms to the stage of the Phratch environment as shown in figure 3.20.
- **secure data is empty** block: The functionality of this block is to check if the security data is empty or to make sure there is no data left which is unread.
- **rooms_ at name_ is not nil?** block: The functionality of this block is to check if the data of the person who is being tracked contains the room data.
- **read from secure data name_room_time_** block: The functionality of this block is to read the fabricated secure data and get the name of the person, rooms the person has been to and the time spent by a person in each room.
- **draw lines for _with room_and time_** block: The functionality of this block is to create visualizations for the persons name

being tracked. This block chooses a random color to draw the lines connecting different rooms and add a timeline on the stage of Phratch environment. The connecting lines depict the moments of the person and the time spent in each room.

- **generate the secure data of name_with_** block: The functionality of this block is to generate the fabricated secure data for cases in which the person who is being tracked has no room locations.

The script that students achieve with the help of the guided steps is shown in Section A.7 and figure 3.21

3.3.8 Transportation and Medical

The aim of this lab is to give students some understanding about Transportation (autonomous vehicle) and Medical (CAT-scan).

Transportation

We had students focus on how autonomous vehicles work and the importance of sensors in helping autonomous vehicles detect their environment. To give students better understanding of the autonomous vehicle, we had students focus on the line-following autonomous vehicles.

The block that we designed to support this lab is a **transport setup** block. The functionality of this block is to let users freely draw a path on the stage of Phratch environment.

We designed a sprite to act as an autonomous vehicle. The sprite has red colour on the left and green colour on the right side of it.

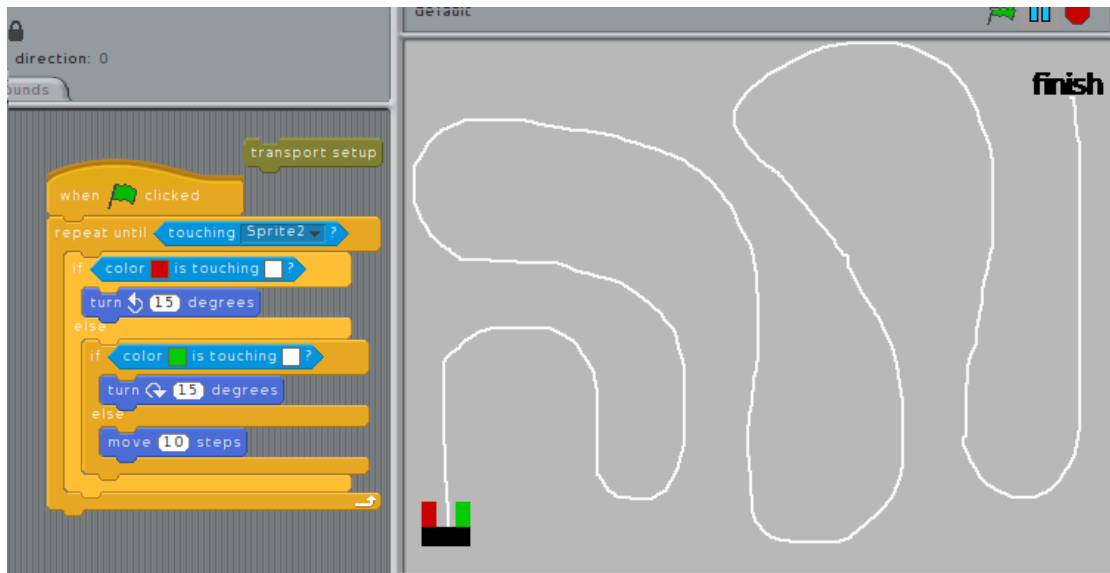


Figure 3.22: *Line Follower Autonomous Vehicle*

These colour's act as the sensors in detecting the path as shown in figure 3.22. The script that students are supposed to achieve with the help of guided steps is shown in Section A.8, and figure 3.22.

Medical

Our aim is to give students some understanding about how CAT-scan works. CAT-scan or CT scanner according to the article[7], “generates a 3-dimensional (3-D) image of the inside of an object. The 3-D image is made after many 2-dimensional (2-D) X-ray images are taken around a single axis of rotation - in other words, many pictures of the same area are taken from many angles and then combined to produce a 3-D image.” A detailed explanation of how CAT-scan works can be found in the article[7].

For the purpose of this lab, we give students an understanding of

how CAT- scan software uses density values to generate an image of the scanned object. Students have two tasks to achieve in the lab, which are as follows:

- Generate density values of any object scanned and add them to the list (i.e. we have students assume that, there is a CAT-scan at the background that generates the density values of the object scanned).
- Perform the scans, using the density values generated, to get some vague image of the object on the stage of Phratch environment.

The blocks, we designed to support this lab are:

- **get density values for CAT-scan** _ block: The functionality of this block is to generate the density values of the object scanned at six angles at the background and add the density values with an appropriate angle to the list.
- **angles from** _ block: The functionality of this block is to check the number of angles in the list.
- **perform scan at angle** block: The functionality of this block is to project the gray colour rays onto the stage; the amount of grayness in the rays is calculated by the percentage of the scanned object covered by the ray. If the ray covers a maximum amount of the scanned object, the ray will be maximum gray (i.e. 0.5 RGB colour value).

Figure 3.23 shows a vague image of diamond shape one of the scanned objects and the script students wrote with the help of the

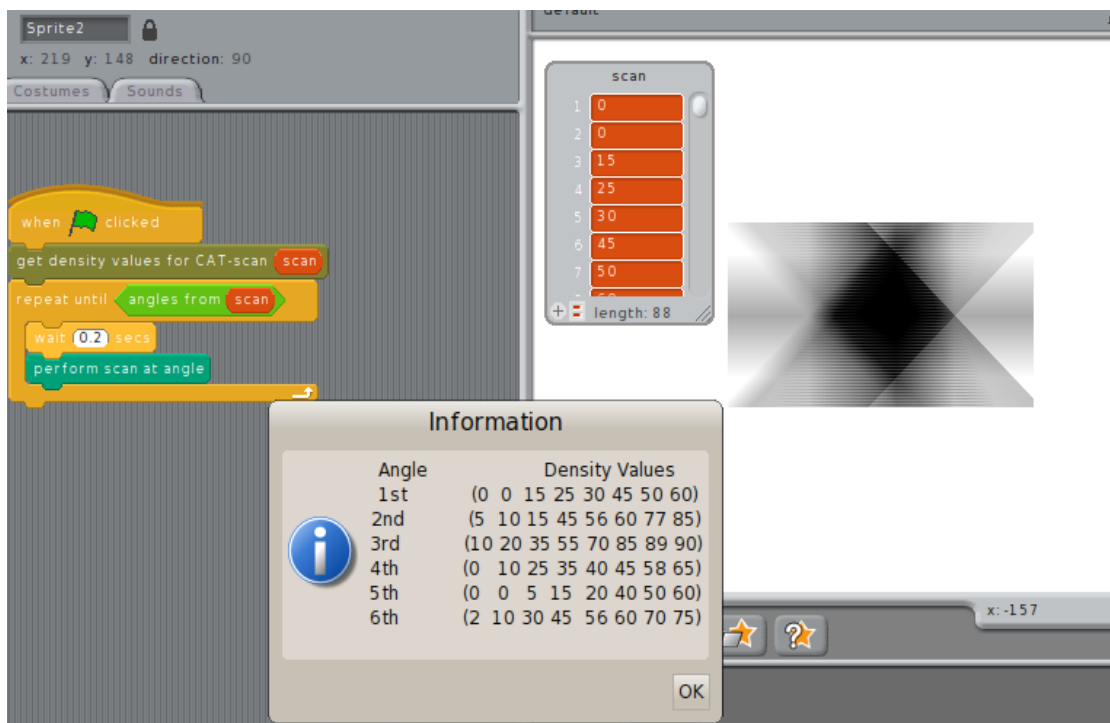


Figure 3.23: Script to perform a CAT-scan

guided steps shown in Section A.8. Figure 3.23 also shows an information window with density values. For this lab we have a diamond, circle, triangle and rectangle shapes as the scanned objects.

3.4 Qualitative Research

We consider surveys to be a good option in evaluating the experiments. Surveys help communicate with the respondent individually and provide a better insight of the topic being tested. Most of the introductory courses to non-programmers mentioned in Chapter 2 used surveys to evaluate the introductory course. Survey questions asked to the students at the end of each lab can be seen in Appendix B. The questions asked to the students at the end of the each lab are constructed by us. Questions were constructed based on thesis goals such as “Do guided labs help students complete the lab successfully?” and “Were we successful in providing student understanding about the systems being explored in the lab?”. The Questions 3 and 7 as shown in Appendix B were constructed from the related work mentioned in Chapter 2.

3.4.1 Characteristics desirable for a survey

Below are the characteristics of the surveys in terms of sample size, self-selection and control groups that we consider important and desirable for any survey.

Sample Size: Sample size is the determination of a number of samples to be included in the study. It is important for any survey to have an appropriate and representative sample size. The general

rule as reported by the StatsCan website[40] is that for any survey to have a reduced sampling error it is important to increase sample size. Increase in sample size might also increase the cost.

Self Selection: Self-selection is a survey method targeted to a selected population. This can also lead to sampling bias. If the survey is sample biased, the samples collected will be less representative of the targeted population. The results can be manipulated to be favorable.

Control Groups: C.J Peng & M.B. Ziskin[8] defined control group as “untreated group with which the experimental group (or treatment group) is contrasted. It consists of units of study that did not receive the treatment whose effect is under investigation.”

Our survey applied to students registered in the course CPS650 (Computational Thinking In Our World) compromising a representative sample size since this group had no prior programming knowledge. Moreover to survey the students from the course we had to get Ryerson ethics board approval (REB)[36]. Since our surveys were voluntary and included a self - selected audience, it might seem that the samples are biased. Justification and reasons for samples not being biased and an explanation for the difference in sample size for each lab is mentioned in Section 4.10.

Chapter 4

Results

Chapter 4 describes the results of the experiments described in Chapter 3. Surveys were conducted at the end of each 1-hour directed lab to evaluate each various characteristic of the lab. Students had 10 minutes to complete the survey. The survey questions were the same for each week's lab; 36 students registered for the course CPS650 – Computational Thinking In Our World and participation in the surveys was entirely voluntary.

We evaluated each lab based on the following attributes:

- **Prior knowledge:** we wanted to know if students had any prior knowledge about the topic presented in the lab.
- **Visualization:** we wanted to know if the visualization was useful in giving the students better understanding of the topic of the lab.
- **Guided steps:** we wanted to know if the lab description given to students, was helpful in successfully completing the lab.
- **Understanding:** we wanted to know if the lab gave students a

better understanding of the topic.

- **Importance:** we wanted to know if the lab was important in learning the course material for CPS650 – Computational Thinking In Our World.
- **Duration:** we wanted to know if the duration of the lab was appropriate.

The 4 choices that students were given to evaluate each question of the lab except duration were **strongly agree**, **slightly agree**, **slightly disagree** and **strongly disagree**. To evaluate the **Duration** of the lab, students were given different options to choose from which were **too short**, **short**, **right**, **long**, **too long**.

4.1 Communication

Students liked sending an email from the Phratch environment; all 36 students successfully completed the lab within 1-hour with the help of the guided steps. Only 12 students participated in the survey, among these 12 students three students suggested that, more tasks be added to the lab.

The survey results of the 12 participants for the communication lab are shown in Tables 4.1 & 4.2. The tabular data can be visualized by the figures 4.1 & 4.2.

The results of 12 participants for the communication lab are discussed below:

67% of the respondents had no prior understanding or knowledge about the topic. Looking at the results of the understanding attribute

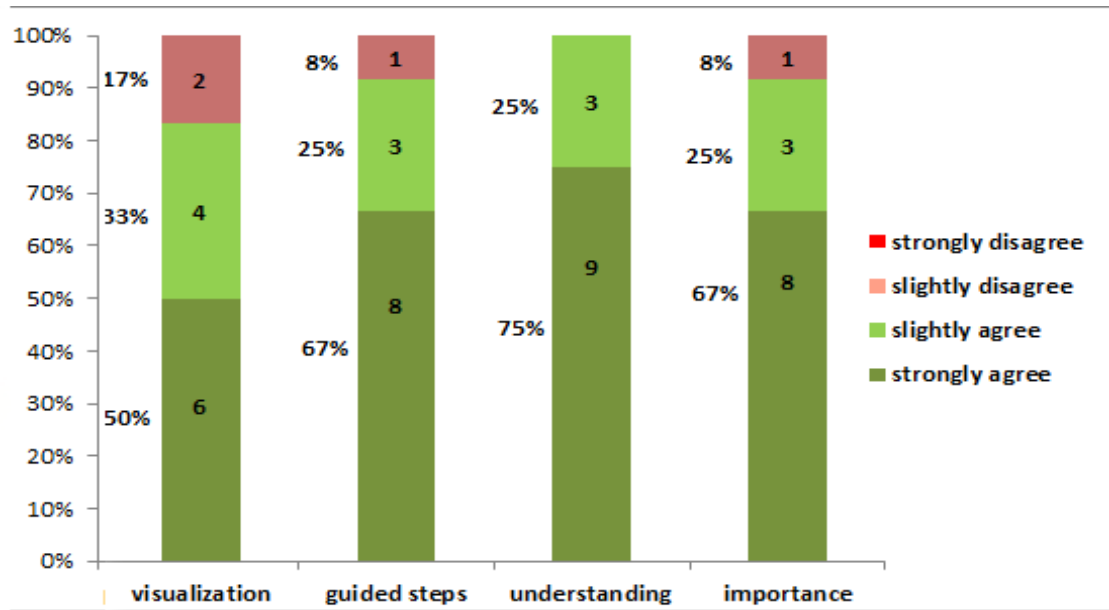


Figure 4.1: Results of Communication Lab
12 participants results shown in % for Communication Lab

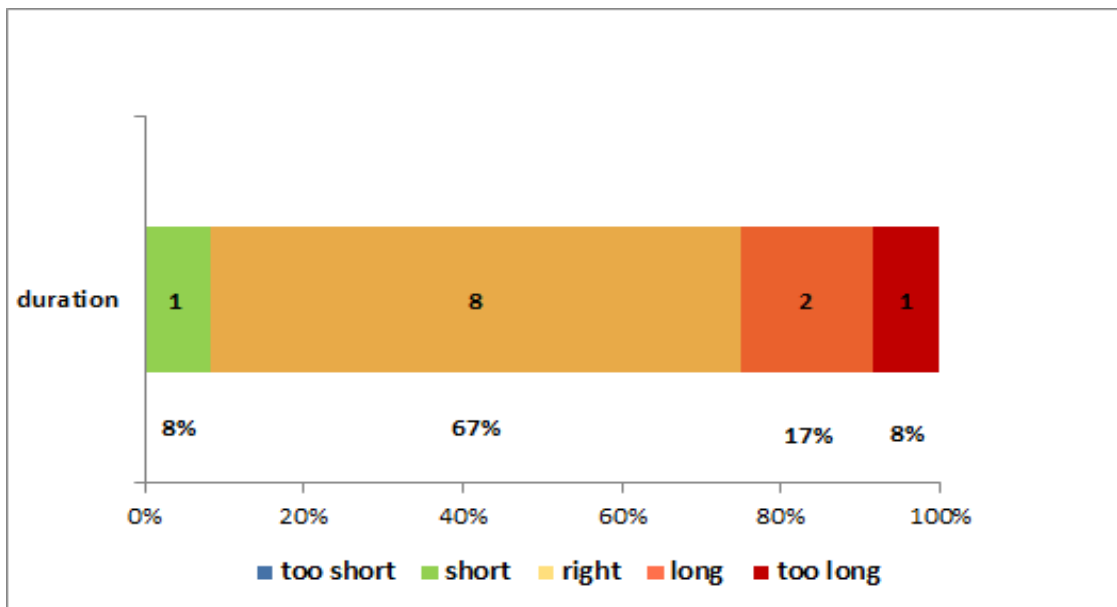


Figure 4.2: Results of the Duration of Communication Lab
12 participants results shown in % for the duration of Communication Lab

Table 4.1: Communication Lab Survey Results

	strongly agree	slightly agree	slightly disagree	strongly disagree
prior knowledge	1	3	7	1
visualization	6	4	2	0
guided steps	8	3	1	0
understanding	9	3	0	0
importance	8	3	1	0

Table 4.2: Communication Lab Survey Results for Duration

	too short	short	right	long	too long
duration	0	1	8	2	1

where 75% of the respondents “strongly agreed” on understanding the topic presented in the lab, we speculate that the lab provided students with a good understanding of the Internet email messages and the important components involved in building up a message and the protocols.

67% of the respondents “strongly agreed” and 25% of the respondents “slightly agreed” on guided steps as helpful in successfully completing the lab within 1-hour. From this, we speculate that guided steps also played a major role in providing students with an understanding of the topic presented in the lab.

8% of the respondents reported the duration of the lab was short. We speculate that since this was an early stage for these students, and they were not yet comfortable with the working of the Scratch programming environment, this could be the reason for 8% of the respondents reporting the duration of the lab to be short.

4.2 Social Networks

Students found the Social Networks lab to be interesting in giving them an understanding about Twitter’s REST API. Twitter’s API’s rate limiting in v1.1 allows only 15 requests per twitter developer account[41].

This discouraged the students as they had to wait another 15 minutes to access the data using the Twitter API. To solve the Twitter API rate limiting problem, we made multiple twitter developer accounts. 12 students opted to participate in the survey. The survey results for the 12 participants in the Social Networks lab are shown in Tables 4.3 & 4.4. The tabular data can be visualized by the figures 4.3 & 4.4.

Table 4.3: Social Networks Lab Survey Results

	strongly agree	slightly agree	slightly disagree	strongly disagree
prior knowledge	0	3	3	6
visualization	6	5	1	0
guided steps	7	4	1	0
understanding	7	3	2	0
importance	8	4	0	0

Table 4.4: Social Networks Lab Survey Results for Duration

	too short	short	right	long	too long
duration	0	4	8	0	0

The results of 12 participants for the Social Network lab are discussed below:

75% of the respondents had no prior understanding or knowledge of the topic presented in the lab. 59% of the respondents “strongly

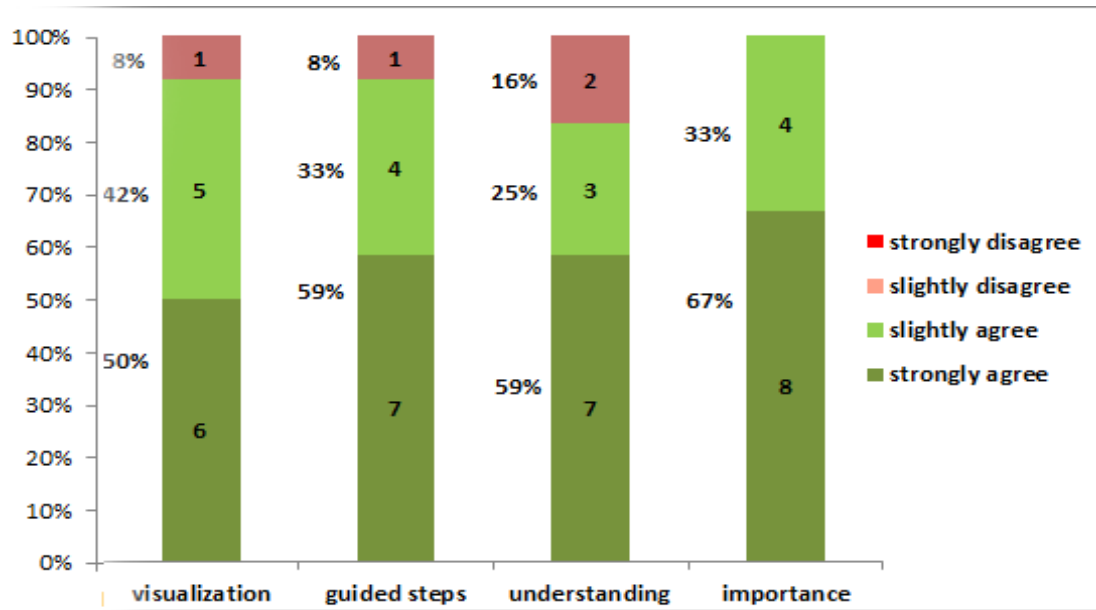


Figure 4.3: Results of Social Networks Lab
12 participants results shown in % for Social Networks Lab

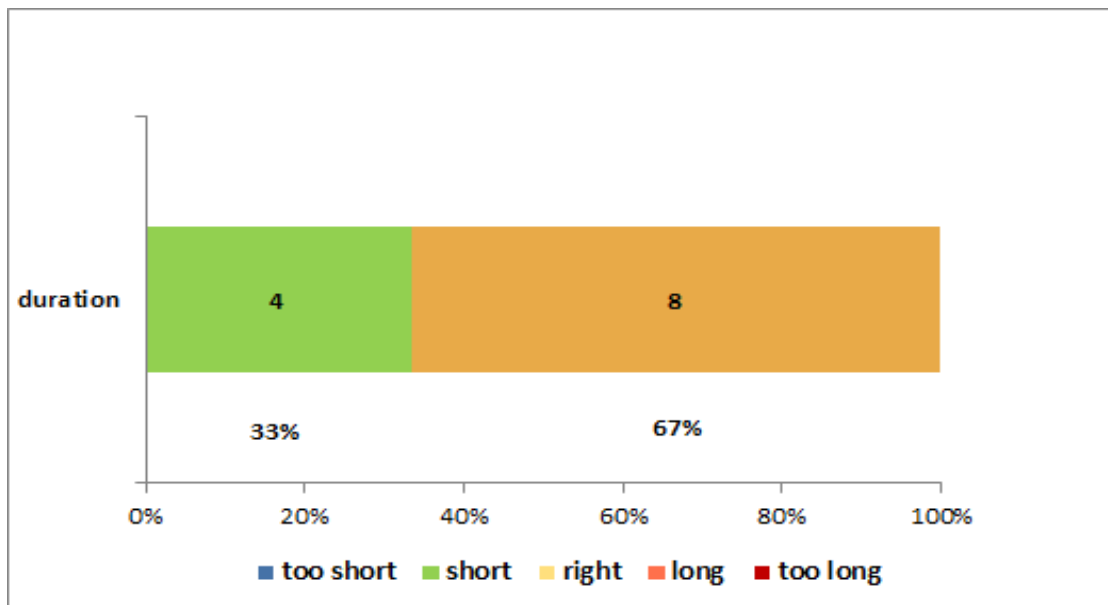


Figure 4.4: Results for the Duration of the Social Networks Lab
12 participants results shown in % for the Duration of Social Networks Lab

agreed,” 25% of the respondents “slightly agreed”, and 16% of the respondents “slightly disagreed” on the lab facilitating an understanding of the web API’s. We speculate that 16% slight disagreement in understanding was due to the rate limiting of the Twitter API, which might have discouraged students from concentrating on the lab.

59% of the respondents “strongly agreed” and 33% of the respondents “slightly agreed” on guided steps helping them complete the lab within 1-hour.

67% of the respondents “strongly agreed” and 33% of the respondents “slightly agreed” on the topic being presented in the lab is important in learning the material of the course CPS650 – Computational Thinking In Our World.

33% of the respondents reported the duration of the lab was “short.”

4.3 Location and GPS

Students reported that, they enjoyed working on the lab. 14 students opted to participate in the survey. The survey results of the 14 participants for the Location and GPS lab are shown in Tables 4.5 & 4.6. The tabular data can be visualized by the figures 4.5 & 4.6.

The results of 14 participants for the Location and GPS lab are discussed below:

42% of the respondents “strongly disagreed,” 29% of the respondents “slightly disagreed” and 29% of the respondents “slightly agreed” on having prior knowledge or understanding about the topic presented. 86% of the respondents “strongly agreed,” 14% of the re-

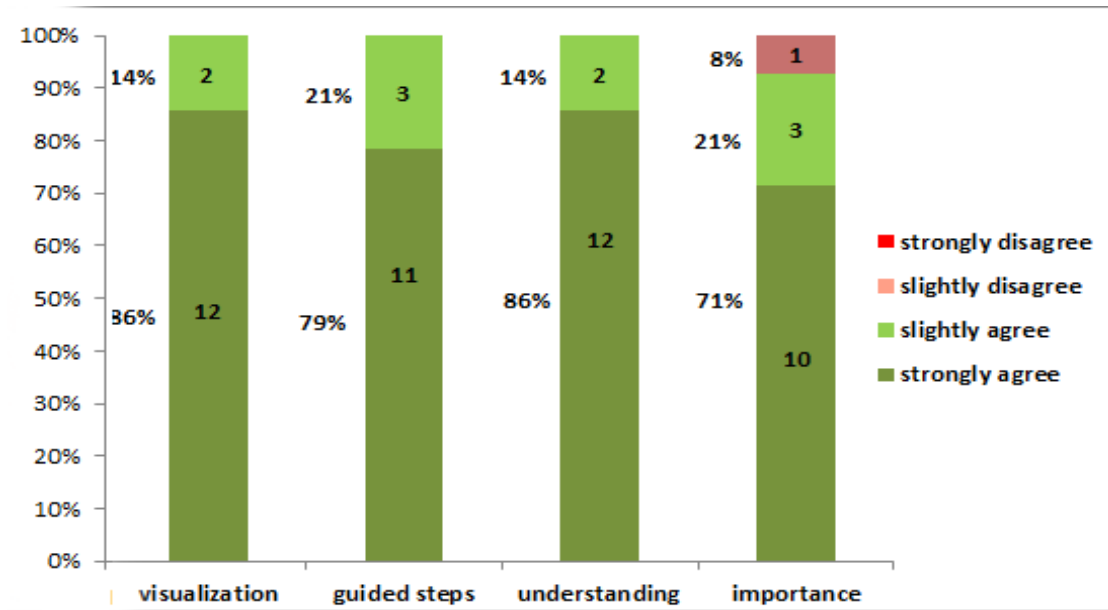


Figure 4.5: Results of Location and GPS Lab
14 participants results shown in % of Location and GPS Lab

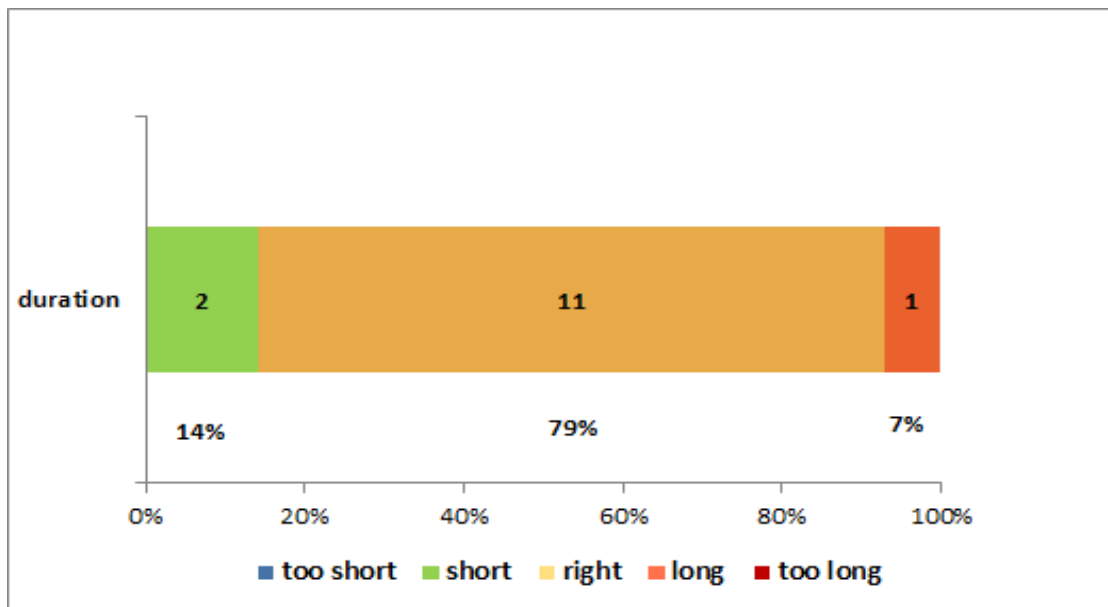


Figure 4.6: Results for the Duration of the Location and GPS Lab
14 participants results shown in % for the Duration of Location and GPS Lab

Table 4.5: Location and GPS Lab Survey Results

	strongly agree	slightly agree	slightly disagree	strongly disagree
prior knowledge	0	4	4	6
visualization	12	2	0	0
guided steps	11	3	0	0
understanding	12	2	0	0
importance	10	3	1	0

Table 4.6: Location and GPS Lab Survey Results for Duration

	too short	short	right	long	too long
duration	0	2	11	1	0

spondents “slightly agreed” on understanding how GPS works to provide location. From the results, we speculate that our design was successful in giving students understanding about this topic.

86% of the respondents “strongly agreed,” 14% of the respondents “slightly agreed” on visualization being helpful in giving them understanding. Visualization was one of the important part of our design in depicting the real-time scenario.

79% of the respondents “strongly agreed” and 21% of the respondents “slightly agreed” on guided steps helping them complete the lab within 1-hour.

71% of the respondents “strongly agreed,” 21% of the respondents “slightly agreed” and 8% of the respondents “slightly disagreed” on the topic being presented is important in learning the material of the course CPS650 – Computational Thinking In Our World.

14% of the respondents reported the duration of the lab was “short,” 79% of the respondents reported the duration of the lab was “right” and 7% of the respondents reported the duration of the

lab was “long.”

4.4 Robotics

Three students got confused with the names of the two blocks **solve maze edge** and **solve dead end**. The three students reported that the functionality of the two blocks was not clear. Considering the suggestions of these three students, we changed the names of the blocks based on their functionality.

solve maze edge was renamed to **move forward & turn left**.

solve dead end block was renamed to **turn back**.

12 students opted to participate in the survey. The survey results of the 12 participants for the Robotics lab are shown in Tables 4.7 & 4.8. The tabular data can be visualized by the figures 4.7 & 4.8

Table 4.7: Robotics Lab Survey Results

	strongly agree	slightly agree	slightly disagree	strongly disagree
prior knowledge	1	6	4	1
visualization	7	4	1	0
guided steps	9	2	1	0
understanding	6	3	2	1
importance	4	5	3	0

Table 4.8: Robotics Lab Survey Results for Duration

	too short	short	right	long	too long
duration	0	1	10	1	0

The results of 12 participants for the Robotics lab are discussed below:

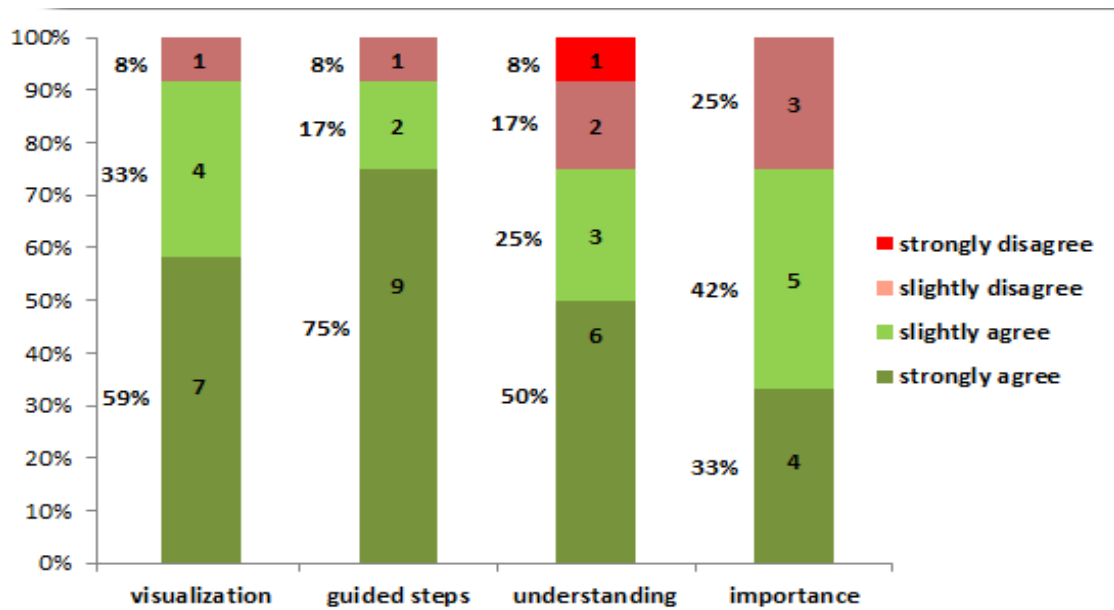


Figure 4.7: Results of the Robotics Lab
 12 participants results shown in % the Robotics Lab Lab

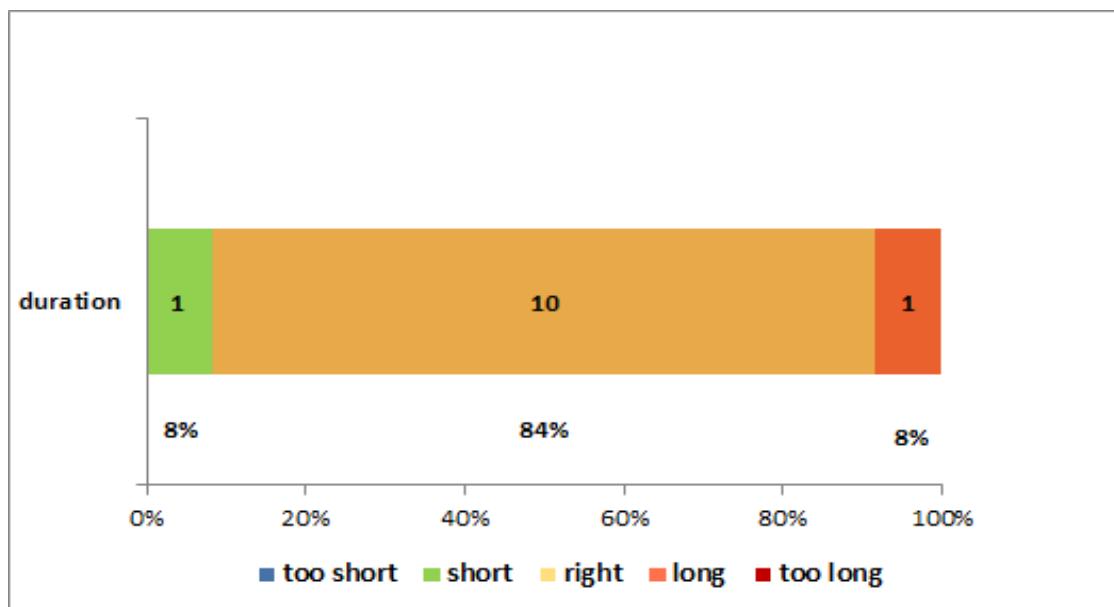


Figure 4.8: Results for the Duration of the Robotics Lab
 12 participants results shown in %for the Duration of the Robotics Lab

42% of the respondents disagreed, 50% of the respondents “slightly agreed” and 8% of the respondents “strongly agreed” on having prior knowledge or understanding about the topic presented. 50% of the respondents “strongly agreed”, 25% of the respondents “slightly agreed”, while 25% disagreed on understanding how robots are programmed to solve problems. We speculate that the 25% disagreement on understanding was due to the names of the two blocks reported by the students as not being clear.

59% of the respondents “strongly agreed,” 33% of the respondents “slightly agreed” on visualization being helpful in facilitating them understanding. From the results, we speculate that the visualization helped students in keeping track of the robots movements.

75% of the respondents “strongly agreed” and 17% of the respondents “slightly agreed” on guided steps helping them complete the lab within 1-hour.

33% of the respondents “strongly agreed,” 42% of the respondents “slightly agreed” and 25% of the respondents “slightly disagreed” on the topic presented is important in learning the material of the course CPS650 – Computational Thinking In Our World.

8% of the respondents reported the duration of the lab was “short,” 84% of the respondents reported the duration of the lab was “right” and 8% of the respondents reported the duration of the lab was “long.”

4.5 Secrets and Intellectual Property

Students found the guided steps helpful in successfully completing the lab. Four students also tried decryption although it was not part of the lab. Students found the lab interesting and helpful enough in giving them better understanding. 14 students opted to participate in the survey. The survey results of the 14 participants for the Secrets and Intellectual Property lab are shown in Tables 4.9 & 4.10. The figures 4.9 & 4.10 help visualize the tabular data.

Table 4.9: Secrets and Intellectual Property Lab Survey Results

	strongly agree	slightly agree	slightly disagree	strongly disagree
prior knowledge	2	3	5	4
visualization	6	8	0	0
guided steps	11	3	0	0
understanding	10	3	1	0
importance	10	4	0	0

Table 4.10: Secrets and Intellectual Property Lab Survey Results for Duration

	too short	short	right	long	too long
duration	0	0	14	0	0

The results of 14 participants for the Secrets and Intellectual Property lab are discussed below:

29% of the respondents “strongly disagreed,” 36% of the respondents “slightly disagreed” and 21% of the respondents “slightly agreed” on having prior knowledge or understanding about the topic presented. 71% of the respondents “strongly agreed,” 21% of the respondents “slightly agreed” on understanding how one-time pad algorithms work in encrypting a message. From the results, we speculate

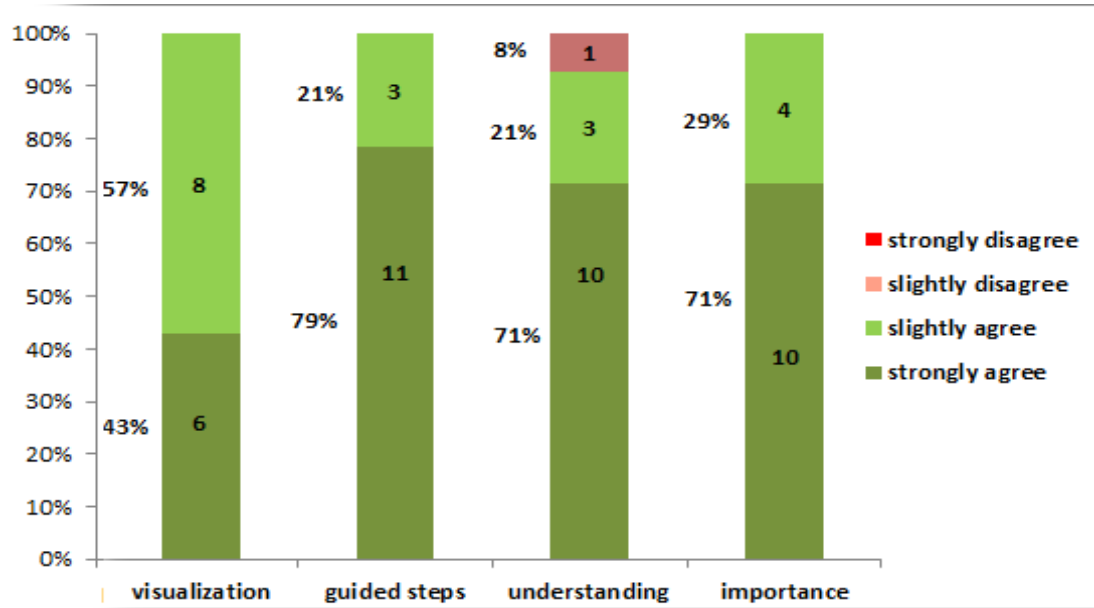


Figure 4.9: Results of Secrets and Intellectual Property Lab
 14 participants results shown in % the Secrets and Intellectual Property Lab

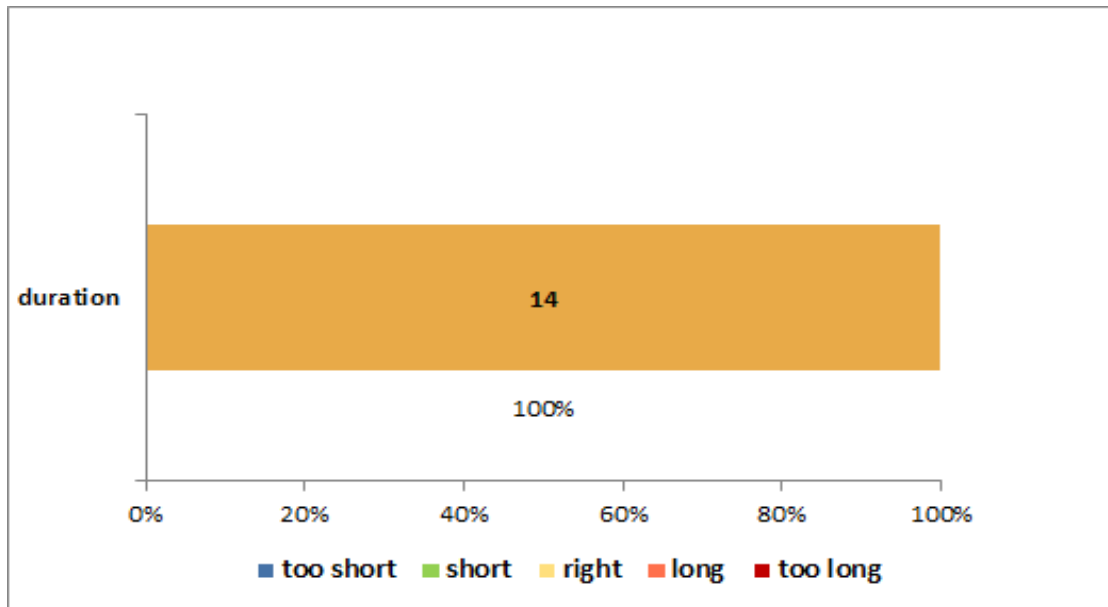


Figure 4.10: Results for Duration of Secrets and Intellectual Property Lab
 14 participants results shown in % for the duration of Secrets and Intellectual Property Lab

that our design was successful in increasing 57% of the respondents understanding about this topic.

79% of the respondents “strongly agreed” and 21% of the respondents “slightly agreed” on guided steps helping them successfully complete the lab within 1-hour.

71% of the respondents “strongly agreed,” while 29% of the respondents “slightly agreed” on the topic presented is important in learning the material of the course CPS650 – Computational Thinking In Our World.

100% of the respondents reported the duration of the lab was right.

4.6 Voting, Finance and Politics

Two students reported the duration of the lab was short since there were two topics covered in the lab. Students wanted to try to accomplish more tasks for the Finance topic because of lack of time they could not. 16 students opted for participating in the survey. The survey results of the 16 participants for the Voting, Finance and Politics lab, are shown in Tables 4.11 & 4.12. The tabular data can be visualized by the figures 4.11 & 4.12.

The results of 16 participants for the Voting, Finance and Politics lab are discussed below:

13% of the respondents “strongly agreed,” 38% of the respondents “slightly agreed” while 49% of the respondents disagreed on having prior knowledge or understanding about the topic presented.

69% of the respondents “strongly agreed,” 25% of the respondents

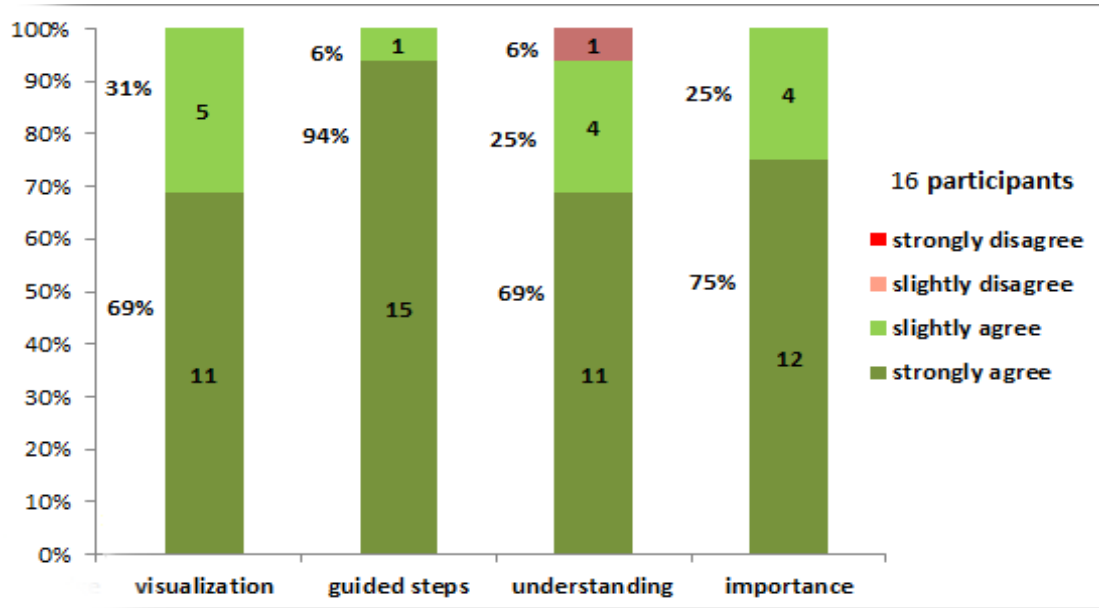


Figure 4.11: Results of the Voting, Finance and Politics Lab
16 participants results shown in % of Voting, Finance and Politics Lab

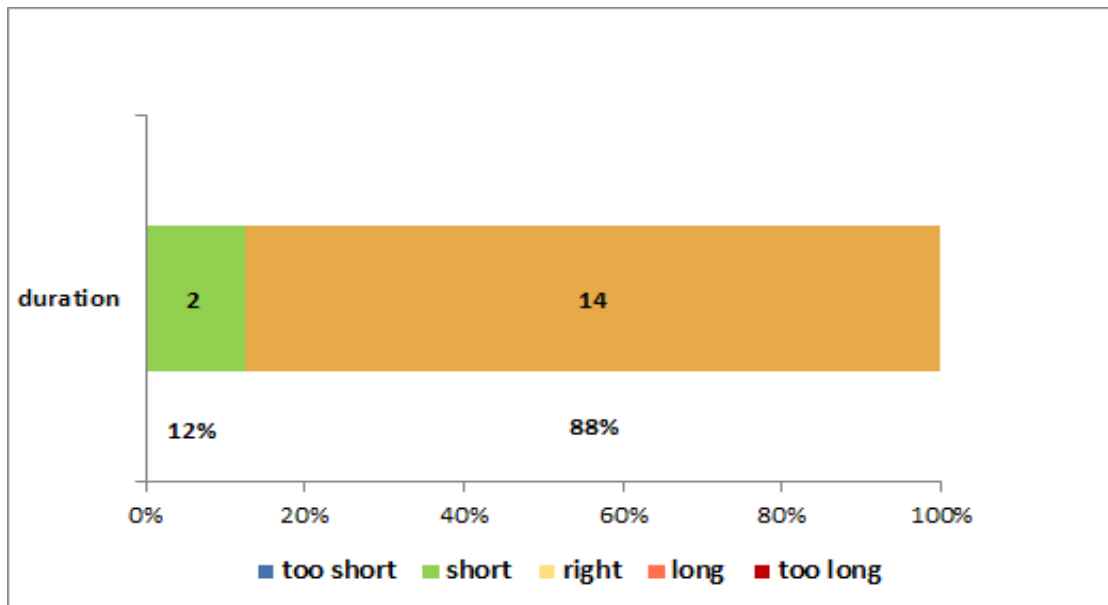


Figure 4.12: Results for the Duration of the Voting, Finance and Politics Lab
16 participants results shown in % for the Duration of the Voting, Finance and Politics Lab

Table 4.11: Voting, Finance and Politics Lab Survey Results

	strongly agree	slightly agree	slightly disagree	strongly disagree
prior knowledge	2	6	5	3
visualization	11	5	0	0
guided steps	15	1	0	0
understanding	11	4	1	0
importance	12	4	0	0

Table 4.12: Voting, Finance and Politics Lab Survey Results for Duration

	too short	short	right	long	too long
duration	0	2	14	0	0

“slightly agreed” on understanding voting systems and finance(stock markets).

94% of the respondents “strongly agreed” and 6% of the respondents “slightly agreed” on guided steps helping them successfully complete the lab within 1-hour.

75% of the respondents “strongly agreed” and 25% of the respondents “slightly agreed” on the topic presented is important in learning the material of the course CPS650 – Computational Thinking In Our World.

88% of the respondents reported the duration of the lab was “right” while 12% reported it was “short”.

4.7 Security and Military Computation

Two Students suggested that it would be helpful if they could access the real data instead of fabricated data on the grounds that it would make the lab more interesting. 22 students opted to participate in

the survey. The survey results of the 22 participants for the Security and Military Computation lab are shown in Tables 4.13 & 4.14. The tabular data can be visualized by the figures 4.13 & 4.14.

Table 4.13: Security and Military Computation Lab Survey Results

	strongly agree	slightly agree	slightly disagree	strongly disagree
prior knowledge	2	8	7	5
visualization	10	11	1	0
guided steps	18	4	0	0
understanding	13	7	2	0
importance	14	6	1	1

Table 4.14: Security and Military Computation Lab Survey Results for Duration

	too short	short	right	long	too long
duration	0	5	17	0	0

The results of 22 participants for the Security and Military Computation lab are discussed below:

55% of the respondents disagreed while 36% of the respondents “slightly agreed” and 9% of the respondents “strongly agreed” on having prior knowledge or understanding about the topic presented. 59% of the respondents “strongly agreed,” 32% of the respondents “slightly agreed,” while 9% slightly disagreed on understanding how location tracking works. We speculate that the 9% disagreement on understanding was due to the data being fabricated, students would have been more interested had we used their Ryerson University’s one card data.

45% of the respondents “strongly agreed,” 50% of the respondents “slightly agreed” on the visualization being helpful in facilitating un-

4.7. SECURITY AND MILITARY COMPUTATION

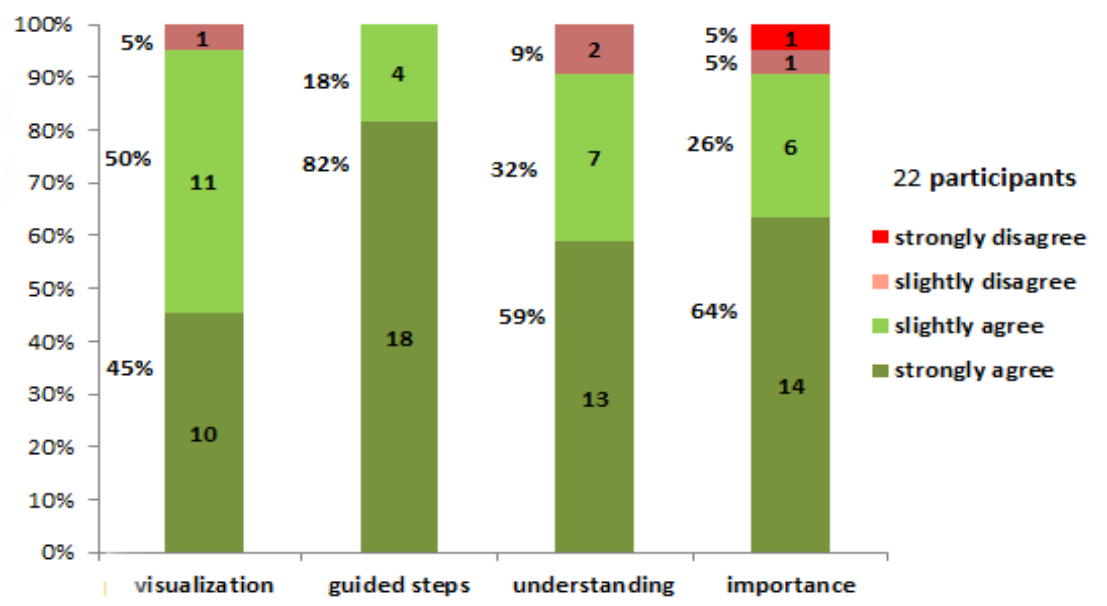


Figure 4.13: Results of the Security and Military Computation Lab
22 participants results shown in % of Security and Military Computation Lab

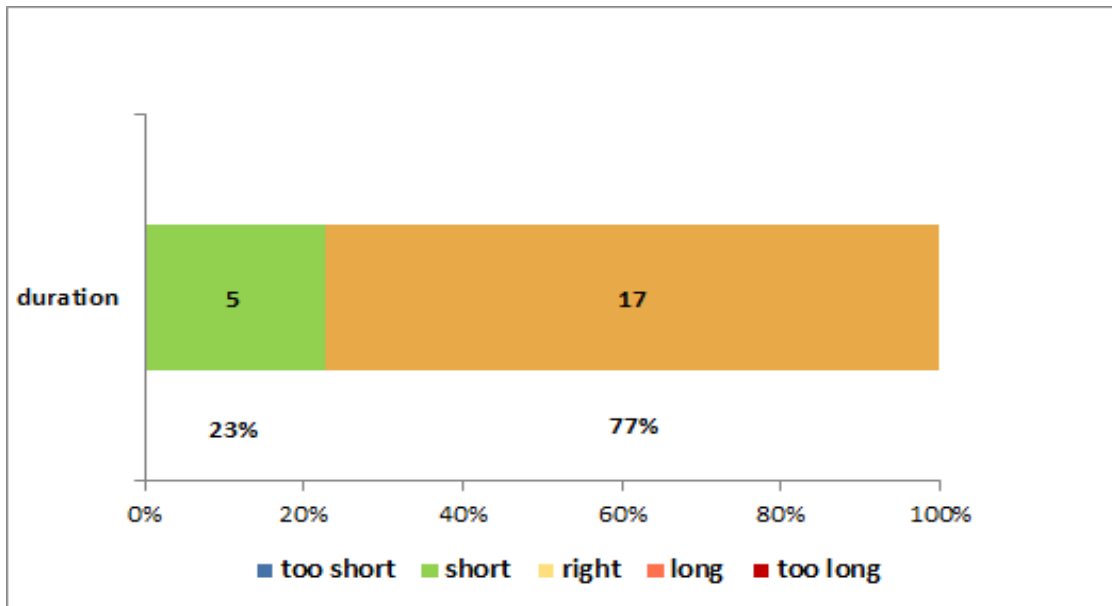


Figure 4.14: Results for the Duration of the Security and Military Computation Lab
22 participants results shown in % for the Duration of the Security and Military Computation Lab

derstanding. From the results, we speculate that we need to improve the visualization for this lab and design an animation rather than drawing simple lines.

82% of the respondents “strongly agreed” and 18% of the respondents “slightly agreed” on guided steps helping them successfully complete the lab within 1 hour.

64% of the respondents “strongly agreed,” 28% of the respondents “slightly agreed” and 4% of the respondents “slightly disagreed” on the topic being presented is important in learning the material of the course CPS650 –Computational Thinking In Our World.

23% of the respondents reported the duration of the lab was “short,” 77% reported it was “right.”

4.8 Transportation and Medical

Six students reported the lab was short. For the Medical topic three students suggested that it would be nice if they could scan their own image file instead of different shapes such as circle, triangle and rectangle. 26 students opted to participate in the survey. The survey results of the 26 participants for the Transportation and Medical lab are shown in Tables 4.15 & 4.16. The tabular data can be visualized by the figures 4.15 & 4.16.

The results of 26 participants for the Transportation and Medical lab are discussed below:

58% of the respondents disagreed while 23% of the respondents “slightly agreed” and 19% of the respondents “strongly agreed” on having prior knowledge or understanding about the topic presented.

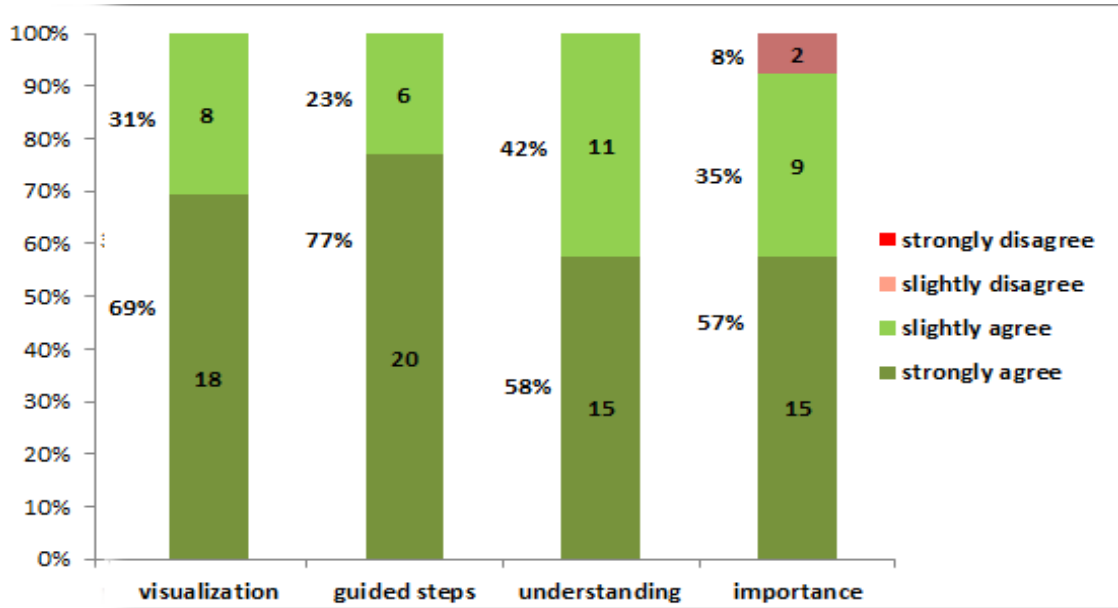


Figure 4.15: Results of the Transportation and Medical Lab
26 participants results shown in % of the Transportation and Medical Lab

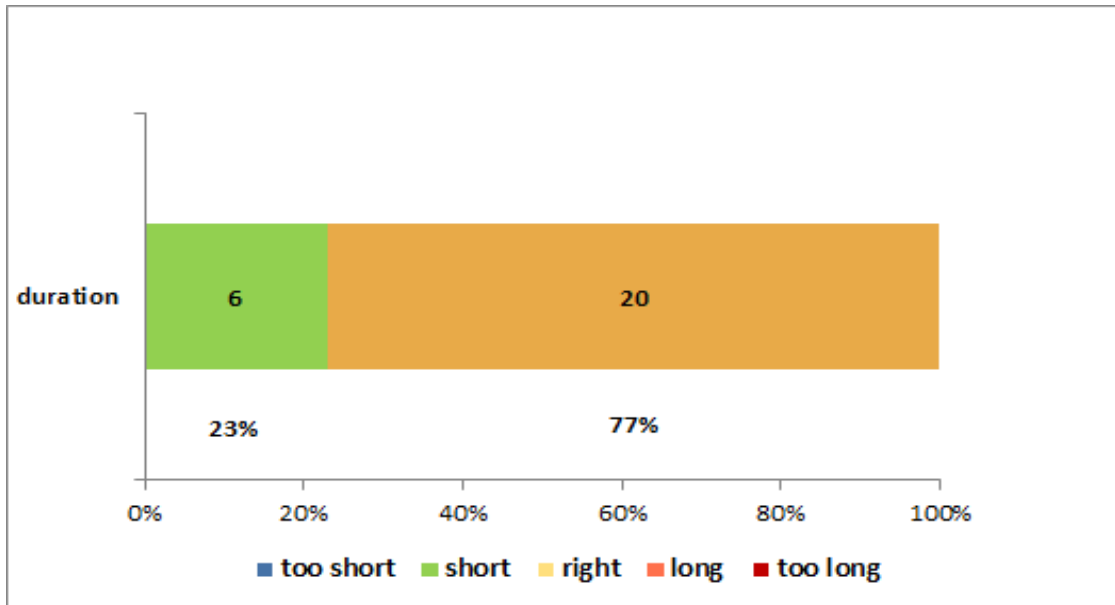


Figure 4.16: Results for the Duration of the Transportation and Medical Lab
26 participants results shown in % for the Duration of the Transportation and Medical Lab

Table 4.15: Transportation and Medical Lab Survey Results

	strongly agree	slightly agree	slightly disagree	strongly disagree
prior knowledge	5	6	8	7
visualization	18	8	0	0
guided steps	20	6	0	0
understanding	15	11	0	0
importance	15	9	2	0

Table 4.16: Transportation and Medical Lab Survey Results for Duration

	too short	short	right	long	too long
duration	0	6	20	0	0

58% of the respondents “strongly agreed” while 42% of the respondents “slightly agreed” on understanding how autonomous vehicles work and how CAT-scan works. From the results, we speculate that the rise of 39% of the respondents on “strongly agreeing” on understanding have increased their knowledge and provided clear understanding about the topic presented in the lab.

69% of the respondents “strongly agreed,” 31% of the respondents “slightly agreed” on visualization being helpful in giving them understanding.

77% of the respondents “strongly agreed” and 23% of the respondents “slightly agreed” on guided steps helping them successfully complete the lab within 1-hour.

57% of the respondents “strongly agreed,” 35% of the respondents “slightly agreed” and 8% of the respondents “slightly disagreed” on the topic being presented is important in learning the material of the course CPS650 – Computational Thinking In Our World.

23% of the respondents reported the duration of the lab was

“short,” 77% reported it was “right.”

4.9 Summary

After completing all the weekly labs, two students opted for a one-on-one interview. The question and answers of each student are discussed below.

Overall experience about working on the labs?

1st student: “overall experience was good, learned about different variety of technologies. The labs gave me better understanding of each topic.”

2nd student: “At the beginning it was difficult, but it got easier as I got used to the Phratch programming environment.”

Which week’s lab was your favorite?

1st student: “GPS lab was interesting, because it has good visualization depicting the real-time scenario.”

2nd student: “Robotics lab was my favorite, because I liked the visualization and the way the robot solved the maze.”

Which week’s lab was difficult to work on?

1st student: “Didn’t find any lab to be difficult to work on, guided steps made it easy.”

2nd student: “Voting and finance lab was difficult, due to lack of time.”

Any suggestion to help us improve the labs?

2nd student: It would be good if we have a first lab, just to explore the Phratch environment and get used to it, before working on the labs. A visual or an image of the expected output should be

added to the description of the lab.

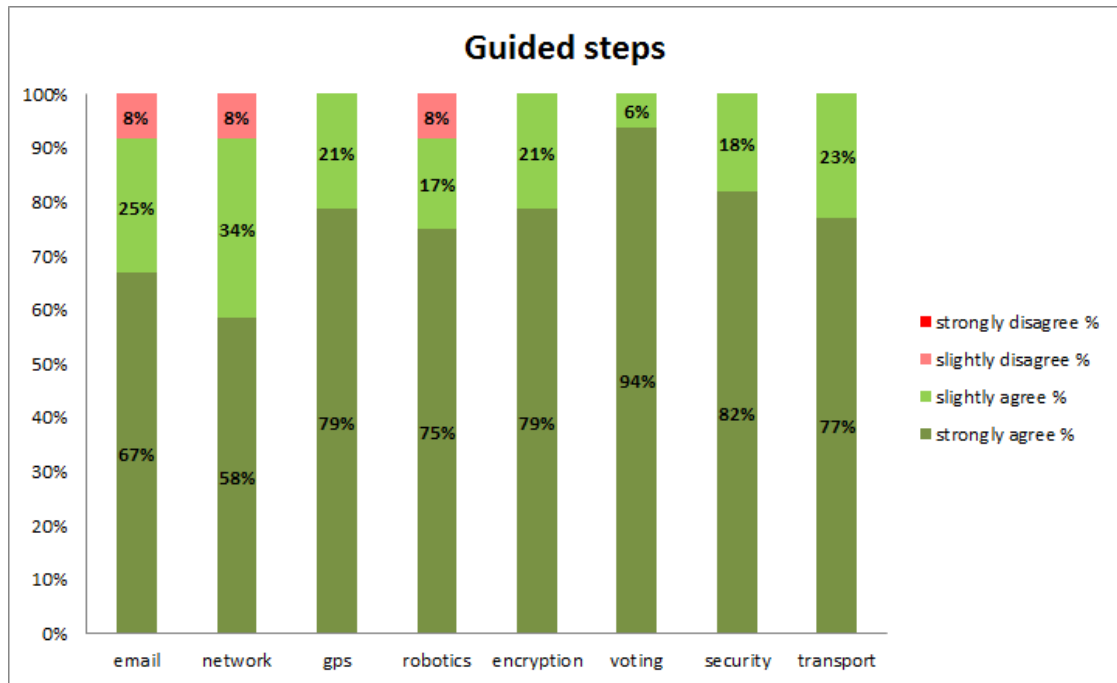


Figure 4.17: Results summary of guided steps for all labs

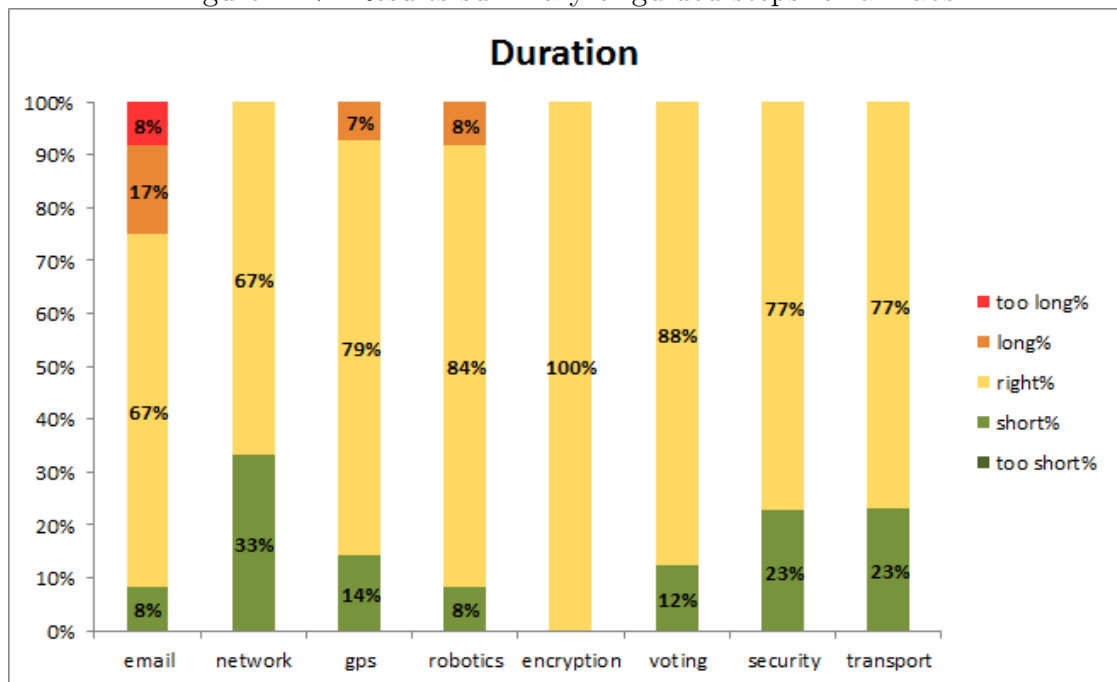


Figure 4.18: Results summary of duration for all labs

From figure 4.17 it can be speculated that the guided labs description provided for the students for each week's lab was found to be helpful. The slight disagreed of 8% for the earlier labs could be the reason for students still getting used to the guided steps and the Scratch programming environment.

Figure 4.18 shows that some participants for each week's lab found the duration to be short. We speculate that students might have found reading guided steps and performing the task simultaneously could be time consuming and in some cases we were covering two topics in one week, which could have also been one of the reasons for participants reporting that the duration of the lab was short. To overcome this problem we speculate that, making the guided steps available for the students in advance (prior to the lab) would be a possible solution. Figure 4.19 shows the summary of the percentage of participants who had prior knowledge about each week's topic presented in the lab. Figure 4.20 shows that we were quite successful in facilitating student understanding about each of the topics, though there was a slight disagreement on a few of the topics; possible reasons have been discussed above.

4.10 Survey Justifications and Limitations

Our surveys utilized small sample sizes because participation in the survey was not made mandatory to the students. Ryerson Ethics Board(REB) said that it would be unfair for students to participate in the survey that is not part of their lab, and REB also pointed out that students would feel inconvenienced spending the lab time

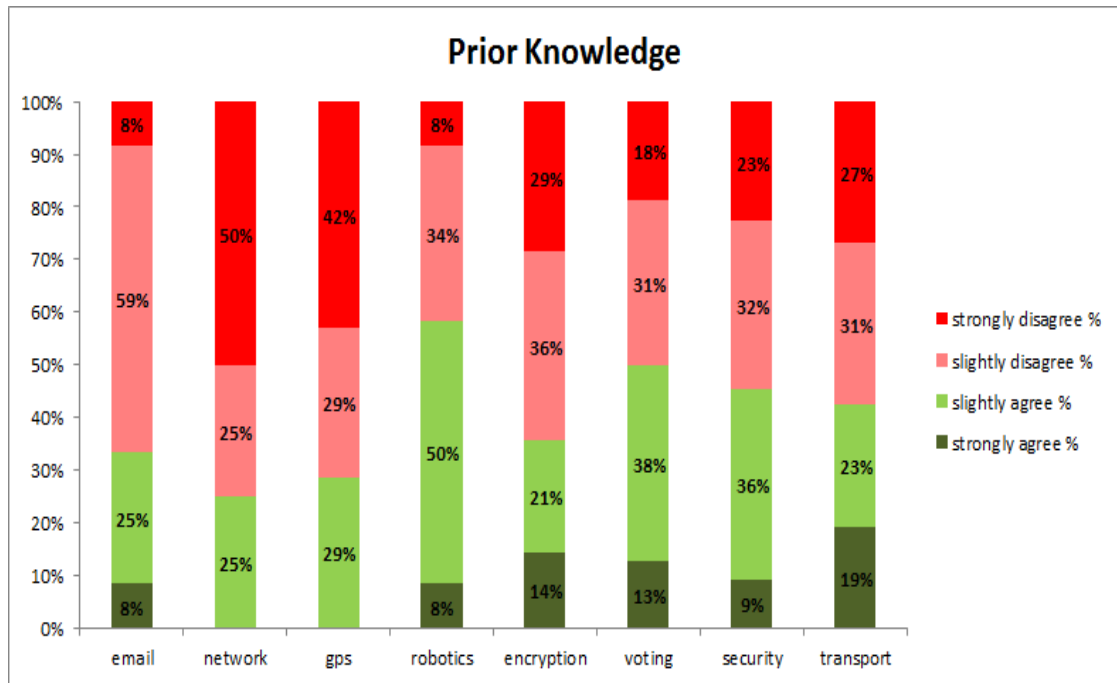


Figure 4.19: Results summary of prior knowledge of participants for all labs

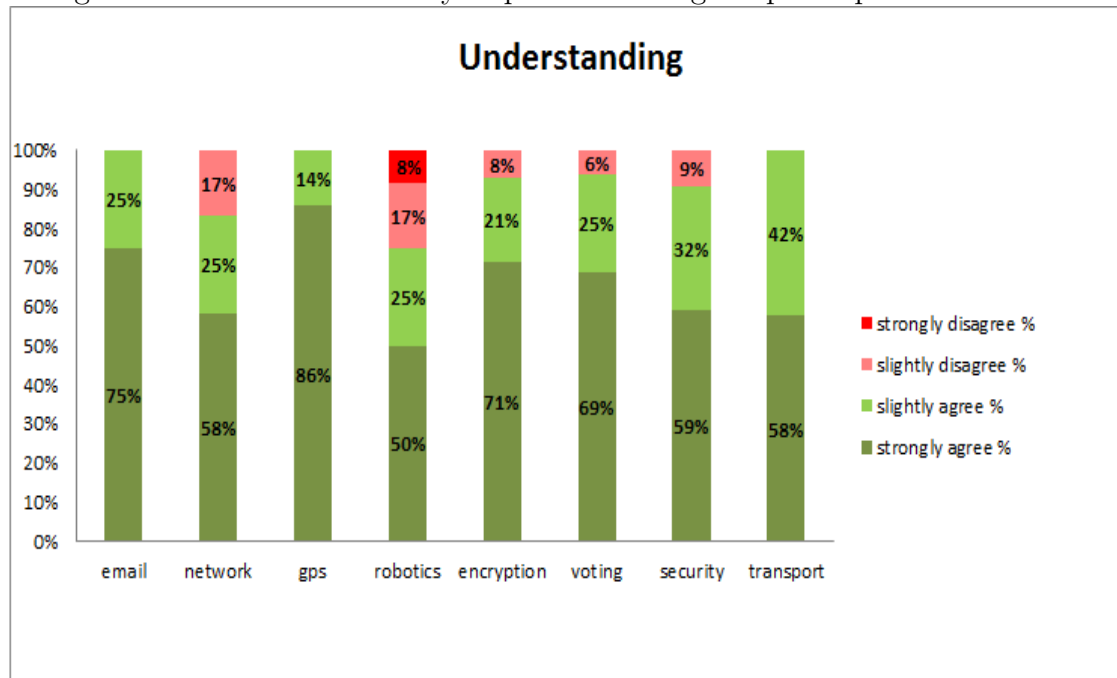


Figure 4.20: Results summary of understanding for all labs

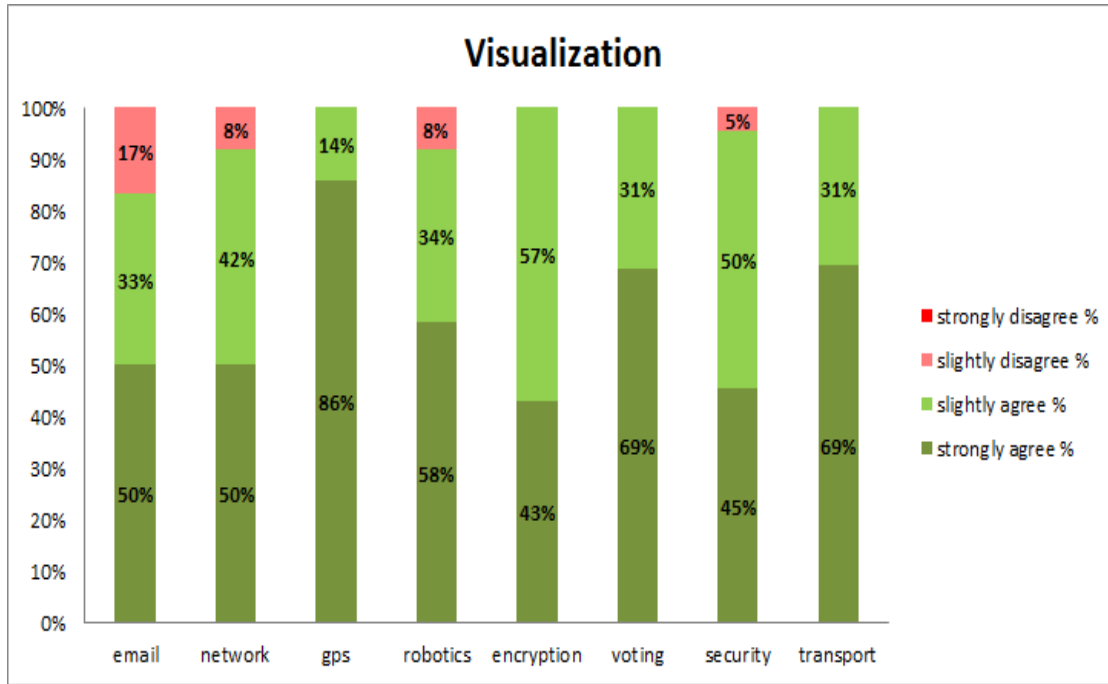


Figure 4.21: Results summary of visualization for all labs

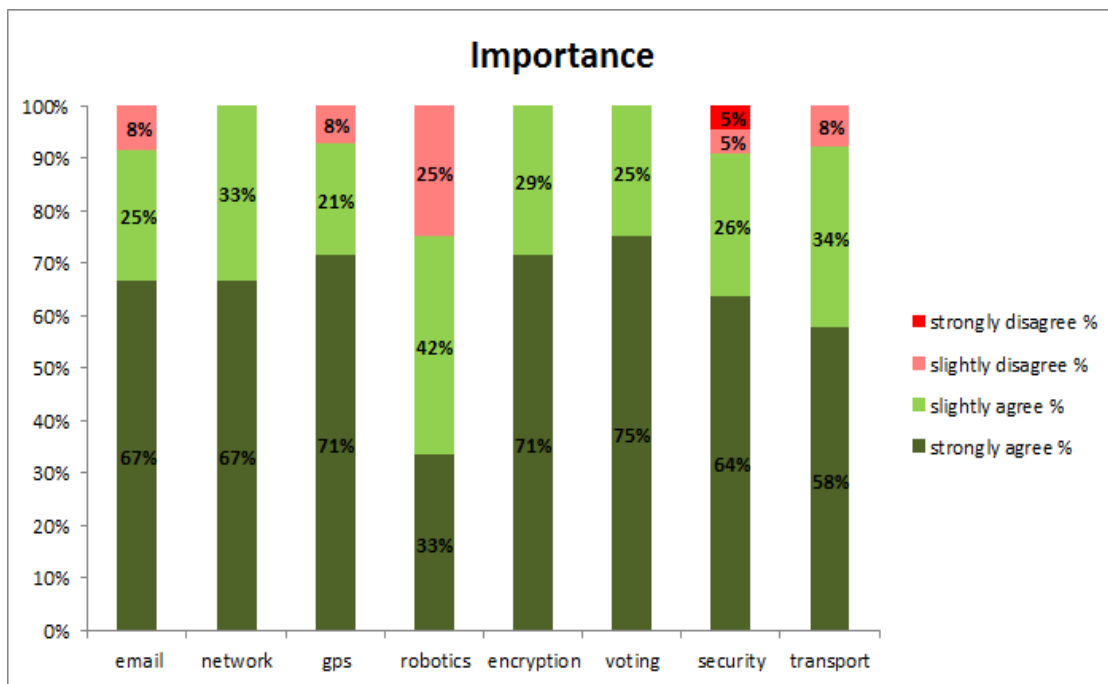


Figure 4.22: Results summary of importance for all labs

answering a survey that is not part of the lab. This was REB's explanation for not letting us make the surveys mandatory for all students registered in the course CPS650 – Computational Thinking In our World. We think that the results gathered from the smaller samples are not biased, because our surveys include results with negative feedback such as for Robotics lab, Voting, Finance and Politics lab, and Security and Military Computation lab from the participants who didn't like the lab or didn't end up achieving a better understanding about the lab. Even though we had small sample size we did get some valuable suggestions that would help in improving the labs, programming environment and the course if offered in the future. These suggestions are described in Section 5.2.

The limitations that are to be considered for our surveys

The surveys had limited sample size. The reasons are as follows:

- There were only 36 students registered in the course.
- We didn't have enough funding to hire students so that the sample size would be larger.

If we had sample sizes of several hundred, then results obtained from the survey would have increased confidence level of our research results.

We didn't analyze to see if guided labs can help non-programmers understand programming languages such as C.

Chapter 5

Conclusions

Students found the labs to be interesting and from the results we speculate that overall guided labs did help students complete the lab within 1-hour. The guided steps and the visualization designed for each week's lab were also helpful in providing students understanding about the technologies underpinned by computer science. The backbone of the Computational Thinking course is the set of examples that reflect real-time data and the visualization that were created to support these examples; that is what makes the labs interesting for students. Students like examples that connect more to their everyday life; we tried our best to reach that goal. The inner working of the systems such as Location and GPS, Communications, Robotics, Social Networks, etc. can be communicated to non-major students by keeping the mathematics content low and eliminating the complex programming syntax through visual programming environment. Guided description and the visualization supporting the systems such as Location and GPS, Communications, Robotics, Social Networks, etc. gives non-major students a better understanding of the systems

and make the labs interesting.

5.1 Contributions

The key contribution of this thesis is we created guided labs in a visual programming environment which introduce non-CS-major students to concepts in programming, while making their task easier so that they can successfully complete a lab within a 1-hour session. We extended the Scratch programming environment to create a simplified version of each of a series of real systems based on each topic mentioned in Section 3.3 for the students to work on in each week's lab. Surveys were conducted to evaluate various characteristic of each lab. From the survey results, we determined that guided labs do help non-CS-major students successfully complete a lab within a 1-hour session. These results are discussed in Chapter 4.

5.2 Future Work

In subsequent offerings of this course, we need to take into account the suggestions made by the students on how to improve our labs.

For the social networks lab, when we pull the IDs of followers of a twitter user, we should create a visualization of the web of the IDs of the followers instead of just pulling them on the list.

For the Robotics lab, instead of using the regular maze solving algorithm, we should provide students with some idea of programming physical robots through Phratch[6] to make the lab more interesting. There is an extension to Phratch, that we could use to allow students

to program physical robots.

For the Security and Military lab, we have considered improving the visualization and using real data such as Ryerson one card data.

For the Medical topic, we will allow the students to scan an image instead of shapes. This would make the lab more interesting for the students.

There are topics that we did not cover in the labs due to lack of time such as fashion and design, big data and cheap computations. We have considered presenting these topics to the students as well. Overall I think that a 1-hour lab should be dedicated to a single topic. Currently, there are two labs that involve two topics each. Students find it hard to switch between the topics and there is a lack of time for them to work.

Our labs reveal the inner workings of complicated systems. We speculate that this will help the average person with no prior programming knowledge to understand the technologies that are underpinned with computer science.

We experienced that Phratch programming environment which is the main programming environment for the course CPS650 (Computational Thinking In Our World) runs slowly while compiling long scripts. We should consider how to improve the Phratch programming environment to run fast while compiling long scripts. Currently, there is no functionality in the Phratch programming environment to support statistics or visualize raw data. Considerations on improving Phratch programming environment to support this functionality will be beneficial in teaching non-CS-major students Big data and Statistics concepts.

5.3 Limitations

When considering any conclusions on the results obtained from the surveys, the following limitations should be considered:

- Participation in the survey was not made mandatory for the students registered in the course CPS650 - Computational Thinking in Our World.
- Our results were not compared with the results of the other introductory courses offered to non-majors.
- We did not analyze to see if guided labs can help non-majors understand programming languages.

Appendix A

Lab Descriptions

Note that there are a few formatting issues with these descriptions. This is because the sections below are automatically generated from the HTML pages for the labs.

A.1 Communication Lab Description

The aim of this lab is to give you some understanding of Internet email messages, the important components involved in building up a message and the protocols to send it over the Internet.

“An Internet email message”¹ consists of three important components, the message envelope, the message header, and the message body. The message header has important information including the sender’s email address (‘From:’), recipient email address(es) (‘To:’, ‘Cc:’, ‘Bcc:’) and the subject (‘Subject:’). For a message to be acceptable to send over the internet, it must follow the internet standard format for electronic mail message header called **RFC822**. Once you have an RFC822-compliant message, you can send it to the

¹<http://en.wikipedia.org/wiki/Email>

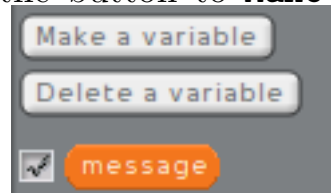
recipient(s) using the simple mail transfer protocol (**SMTP**).

In the Scratch/Phratch environment, we will be using the blocks: First to create a message. The message will be created using internet standard format for electronic mail message header, we will use RFC 822 block to create the message headers. Second, for sending this message over the internet, we will use secure socket layer **SSL** block which is secured way of sending a mail to the recipients. The SSL block will ask you to authenticate the information you provided i.e. (enter password for email address) to make a secure mail transfer.

Step 1: Creating Message

1. Creating a new variable:

In order to build up the message headers one by one, we need to create a new variable. This variable will hold the entire information of the message that can be used later for sending using SSL block. You need to click on the button to **Make a variable**



from the **Variables category**, for now we will make a **message** as a new variable.

2. Setting up the message

- Drag the **when flag clicked** block from the **Control category**,



, to initiate the program.

- You need a **set ___ to ___** block from the **Variables category**

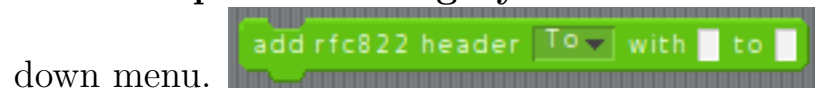


, and set **message** variable to 0 or nil, to make sure each time the program starts variable has nothing stored in it.

- Adding RFC 822 headers:

To create a message in Internet standard format, we need to use RFC 822 header block which will hold the sender and recipient information along with the subject header and message body.

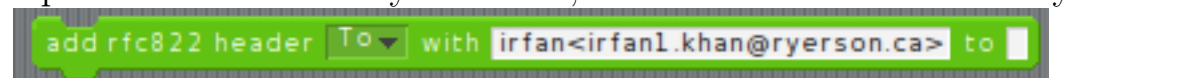
- To add recipient header, drag **add rfc 822 header ___ with ___ to ___** from the **Operators category** and select **To** from the drop-



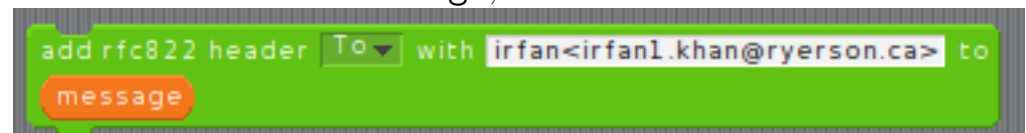
down menu.

- Fill the fields of the rfc 822 block by typing the recipient's email address which can be in any of the formats as follows:

'Irfan Khan <irfan1.khan@ryerson.ca >' or for multiple recipient 'name<name@ryerson.ca>, secondname<secondname@ryerson.ca>'



- Next drag the variable block that you have created from the **Variables category** and plug it into this block. Here we created the variable **message**, so we will use that



- Repeat the above steps to add 'from' i.e. (sender) email address, and 'subject' headers, using the same **add rfc 822 header ___**

with `---to ---` block from the **Operators category**.

- Although the body of the message is not a header, in this environment, you use the same block to add the body.

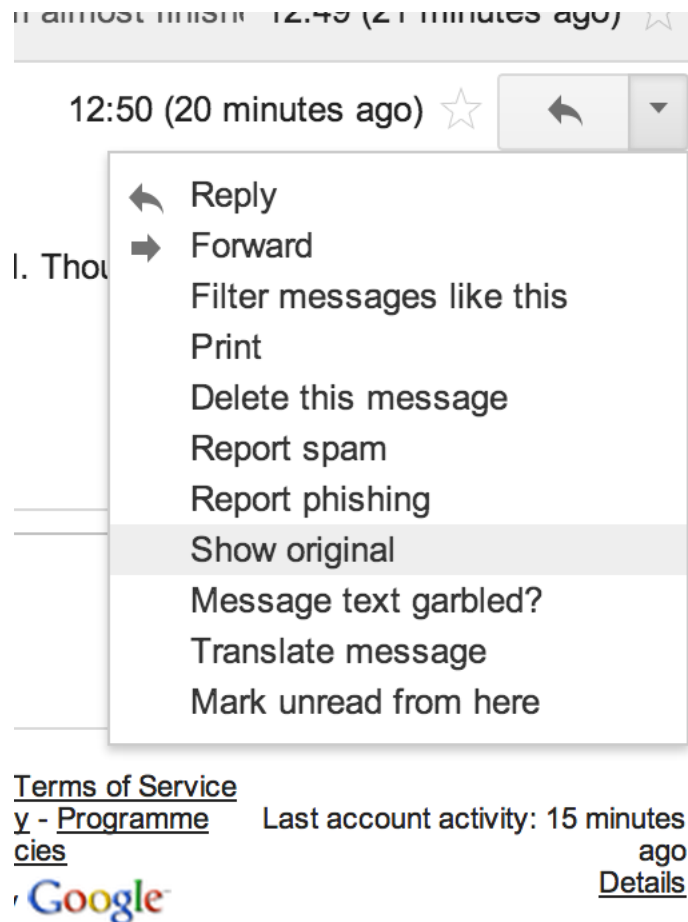
3. Now our message looks something like:

```
From: Irfan Khan <$irfan1.khan@ryerson.ca>$  
To: CPS650 Course <$cps650@sarg.ryerson.ca>$  
Subject: this is a test
```

```
This is the body  
of the  
message  
From: now on  
More body
```

Note that all the headers start in the first column and have a “: ” between the keyword and the contents. There is then an empty line and the body starts. The body may contain content that looks like a header, but it is just treated as text and is not interpreted. You can see more of the headers by looking at “Show original” for an email in the Ryerson (Google) mail reader at gmail.ryerson.ca² as shown here:

²<http://gmail.ryerson.ca>



Sending the RFC 822 message: Now that we have created the message, we need to send it in either of two ways:

- using SMTP, which is a simple, but insecure, way of sending mail and doesn't require any authentication, but only works with some email servers.
- using SSL on top of SMTP, which is a more secure way of sending mails.

Whichever you choose, the operations are very similar, but there are details that differ.

- If you choose to send it using plain SMTP, drag **smtp send** -- **server** -- from the **Operators category**. Plug in the variable that holds the entire message and type in the server name.



Note: This will work only if the server name that you enter accepts unauthenticated emails. The TA may be able to give you the name of the server that will accept your email.

- If you choose to send it using SSL, drag **SSL** -- from the **Operators category**. Plug in the variable that holds the entire message. Once you run the script it will ask you to enter a password for your gmail account.



Note: This will work only if the sender has a gmail account. The “From:” header must contain your gmail account, and you will have to validate it.

A.2 Social Networks Lab Description

The aim of this lab is to give you some understanding about web APIs- Application Programming Interfaces and how APIs can be used to share, or pull information from Social Networking websites. “Web API³ is usually nothing more than a set of Hypertext Transfer Protocol(HTTP) request messages with the response message structured in the JavaScript Object Notation (JSON) format.” Such requests can be made to most social networking websites in order to get some useful information about users.

Social media sites like Facebook and Twitter are interesting for many reasons - including the inherent value to people of what they offer - but for our purposes, they are interesting because of their size and the information you might be able to create from the vast supply of data they hold. For the purpose of this lab, we will use the Twitter API rather than the Facebook API since the Twitter API is much simpler, and more information is available without using passwords.

One of the pieces of information that is easily accessible via the Twitter API is the list of IDs that follow a particular ID. In this lab, we are going to find the number of people that follow you on Twitter, and if you tweeted something and everybody that followed you re-tweeted it, how many people would see the tweet.

In the Scratch/Phratch environment, we will be using blocks to achieve three tasks:

1. Get followers IDs of your twitter account, and add them to a list. Once we have the IDs on the list, we can iterate over the

³http://en.wikipedia.org/wiki/Application_programming_interface

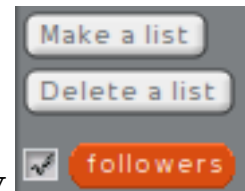
list to get the number of followers (ie count the number of IDs).

2. Get followers of followers IDs into new list.
3. If we have time left, we can use the Twitter “user lookup” API to convert IDs into usernames. This Twitter API has to be used wisely, since it has some limitations.

Step 1: Getting followers Ids

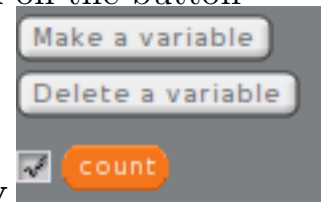
1. Creating a new variable & a new List:

In order to get the followers IDs and store these IDs on a list, we need to create a new list. This list will hold all the followers IDs of the designated twitter user. You need to click on the button



Make a list from the **Variables category**. We will call the new list **followers**.

To count the number of followers one by one from the list, we need to create a new variable. You need to click on the button



Make a variable from the **Variables category**

We will call the new variable **count**.

2. Using Twitter API to get followers IDs

(a) Drag the **when flag clicked** block from the **Control cat-**



category, , to initiate the program.

- (b) You need a **set** ___ to ___ block from the **Variables category**



, and set **count** variable to 0, to make sure each time the program starts variable has nothing stored in it.

- (c) You need a **delete** ___ of ___ block from the **Variables**



category , and select the **followers** list name, and select **all**, to make sure each time the program starts the list has nothing stored in it.

- (d) To use the Twitter API to get followers IDs, you need a **Json extract** ___ ids into ___ block from the **Operators category**

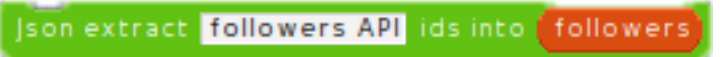


, and use the link below with your twitter screen_name to extract followers IDs (or if you're not on Twitter, use - RyersonCPS650:

https://api.twitter.com/1.1/followers/ids.json?cursor=-1&screen_name=RyersonCPS650

copy this link and paste it in the block.

- (e) Now that we have copied the link, we want to drag the **followers** list from the **Variables category** and plug it into the block to add IDs into list we created. The block will




look something like this,

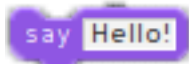
- (f) Once we have the Twitter API setup to pull the followers

IDs and add them to the list, we want to iterate over the list to count the number of followers twitter user has. Drag the **Iterate** ___loop block from the **Variables category**



, set the list name to followers, since we want to iterate over the followers list to count the followers

- Inside the **Iterate** block, in order to count the number of followers while the list is being iterated, we need a **change** ___ by ___ block from the **Variables category**
 , set the variable name to **count** and change the count value by 1.

3. For a sprite to say the result, you can use a **say** ___ block from the **Looks category**  , and drag the **count** variable block that you have created from the Variables category and plug it into say block.

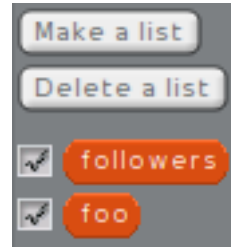
By now you have your Task 1 complete, the above script will get you the followers IDs, and will answer the number of followers the twitter user has.

Step 2: Getting followers of followers In order to get the followers of followers IDs, The script remains the same until the **Iterate** loop block, inside the iterate block. We need a small change in Twitter API link, since we will be using IDs to get the followers of followers, instead of using screen_name we will use user_id.

1. Modifying Twitter API to get followers of followers:

To get followers of followers IDs we need to create a new list, this new list will have followers of followers IDs in it.

In order to do so, you need to click on the button **Make a list**

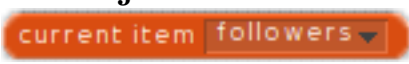


from the **Variables category**, for now we will make **Foo** as the new list.

2. Inside the **Iterate** ___ loop block, we need the following blocks:

- We want to join a modified Twitter API with the **user_id** stored in the **followers** list that we created at the beginning. You need to drag **join** ___ ___ block from the **Operators category** , set the first field of the **join** block with a modified twitter API.

`https://api.twitter.com/1.1/followers/ids.json?cursor=-1&user_id=`

- Now we need a current iterated item from the **followers** list to be joined with Twitter API. In order to do so, you need to drag **current item** ___ block from the **Variables category** and plug it into the second field of **join** block. The block might look something like this .

- By now we have our followers of followers Twitter API setup. In order to pull followers IDs from **followers** list

one by one, and add them to the new **foo** list we created. We need to drag a **Json extract** ___ ids into ___ block from the **Operators category**, plug in the **join** block



at the first field of **Json extract** block and then plug in the **foo** list at the second field of the **json extract** block, since we want to add followers of followers IDs to our new list.

Now you have your Task 2 complete, and this script will give you followers of followers IDs in a new list.

Step 3: Convert user IDs into user names This task is very similar to that of followers of followers IDs. The only difference here is we will use Twitter user lookup API instead of followers IDs API, and once we have a user name, we will replace the user ID of the list with user name.


Note: Don't use your Twitter user name instead use **RyersonCPS650**, since Twitter allows only 350 user lookups in one hour. Just to make sure everyone gets a chance use only **RyersonCPS650** while performing this task.

1. Repeat the same steps until the **Iterate** loop block to get the followers IDs with **RyersonCPS650** as the `screen_name`.
2. Inside the **Iterate** loop block, use the following blocks:
 - You need to use the **join** ___ ___ block from the **Operators category**, set the first field of the **join** block with Twitter


user look up API:

`https://api.twitter.com/1.1/users/lookup.json?user_id=`

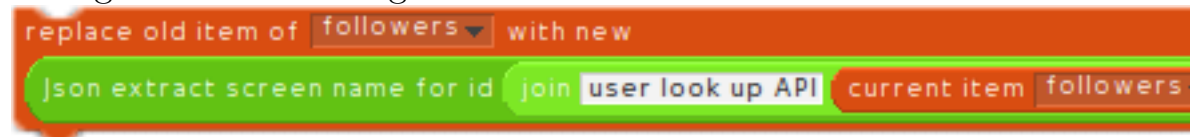
Set the next field of the `join` block with the current iterated item from the `followers` list, drag `current item` ___ block from the **Variables category** and plug it into the second field.

- Now we need a `json extract screen name for id` ___ block from the **Operators category**  to extract the screen name for the corresponding user ID.

In order to do so, plug in the `join` block into `json extract screen name` block.

- Next to replace the current iterated of the `followers` list we need a `replace old item of ___ with new ___` block from the **Variables category** , set the first field of this block with `followers` list name and in the second field choose in the `json extract screen name for id` ___ block.

It might look something like this



By now we have Task 3 complete, and you can try this task with your own Twitter `screen_name` at home.

A.3 Location and GPS Lab Description

The aim of this lab is to give you some understanding of how Location and the Global Positioning System (GPS) works. GPS is a “space-based satellite navigation system”⁴ that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.” It is maintained by the United States government. GPS has become the essential part of our daily life, and it is part of our transportation systems, aviation systems and GPS is also used for search and rescue operations.

Before we start the lab, you might want to have a look at this video⁵, This will give you some idea about how GPS works before we work on this lab.

In this lab, we are going to find the location of the mouse pointer placed anywhere on the stage with the help of satellites. We will have three satellites drifting across the stage continuously. (There are also three signal sprites associated with these satellites that follow the satellites and leave a signal (i.e. circle like visualization). We will script these signals to give us the locations of satellites and the intersecting points of these signal visualizations. The Intersection of these signals will give us a common point, this common point will help us in locating the mouse pointer on stage.

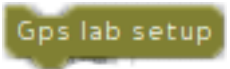
In the Scratch/Phratch environment, we will use blocks in performing the following tasks:

⁴http://en.wikipedia.org/wiki/Global_Positioning_System

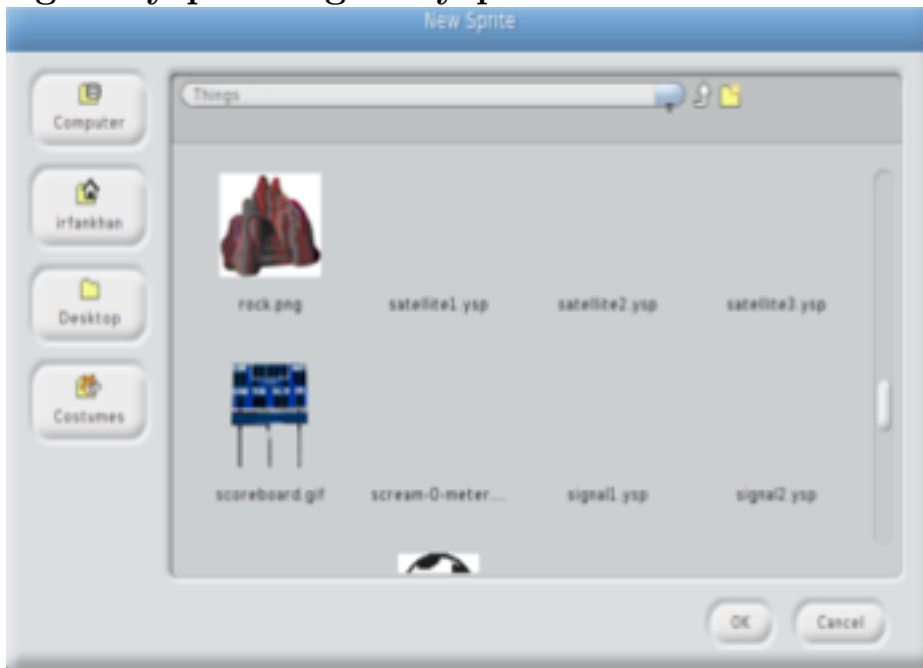
⁵<http://www.youtube.com/watch?v=PLjld-edVj8>

1. We need to setup the lab with sprite images of satellites, signals and create a new sprite (i.e. locator) that can be used for answering the location.
2. We will be creating new variables that will store the location and intersecting points of the signals.
3. We will script signal sprites to get the location of the satellites, and also script the locator sprite to get the common point of all the intersections of the signals. The common point will be the location of the mouse pointer on the stage.

Step 1: Setup GPS lab

1. In order to first setup the lab, you need to click just once on the **Gps lab setup** block located at the **LabSetup** category . This will open two windows, and the first one will guide you in selecting the sprites (i.e. satellite & signal images).
2. From the next **New sprite** window, under the **Things** folder select **satellite1.ysp**, **satellite2.ysp**, **satellite3.ysp**, **signal1.ysp**,

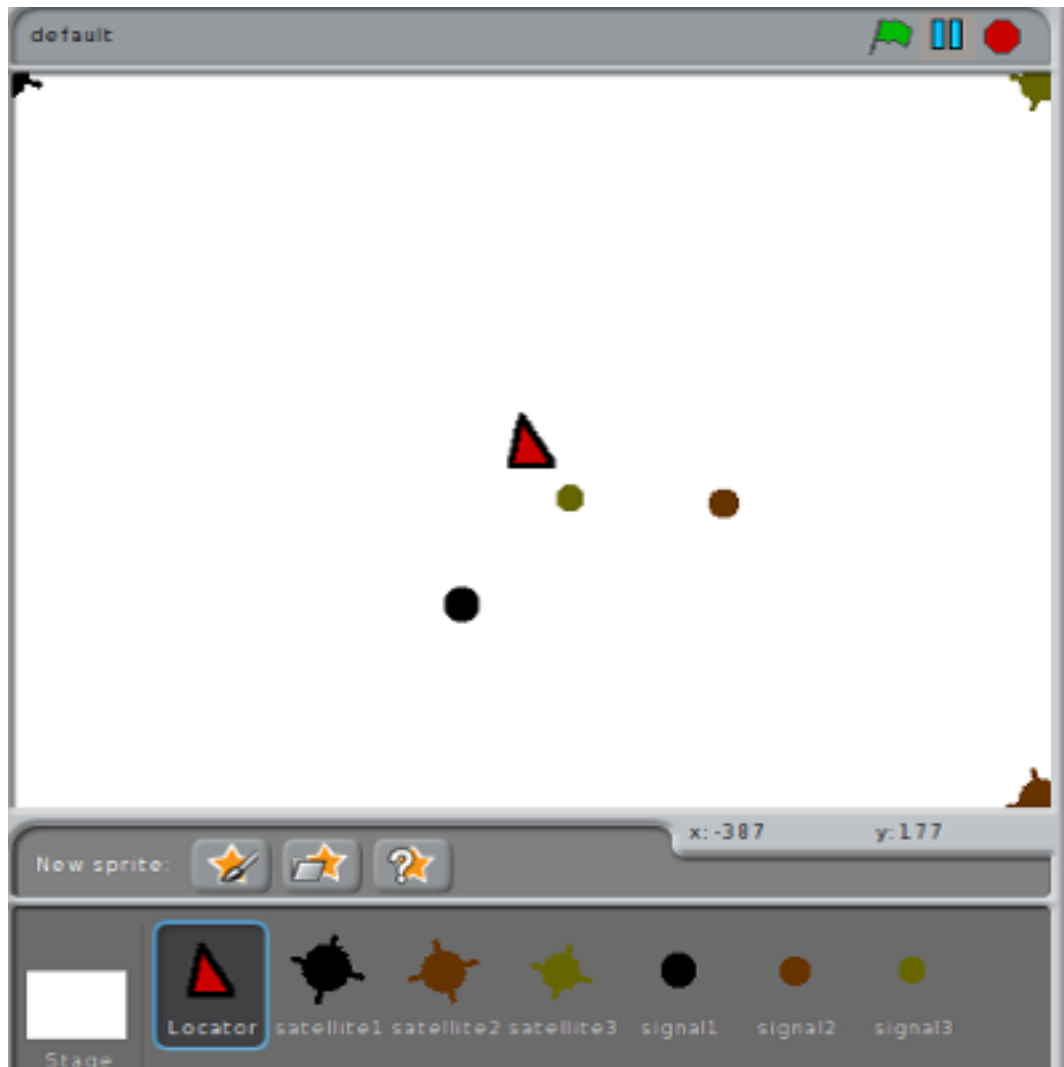
signal2.ysp and **signal3.ysp**.



This will open three satellites and signals which are already scripted to drift across the stage.

3. Now we need a sprite that will answer the location. You can either import any image from Scratch/Phratch environment or create a new sprite in the paint editor, and this sprite will act as a locator. Set the name of this sprite to **Locator**.

By now the stage will look something like this



Step 2: Creating variables

1. you need to create five variables, three variables will be used to store the location of the satellites i.e. (Point and radius) and the other two variables will store the intersecting points of the locations.
2. You need to click on the button **Make a variable** from the



Variables category, for now we will name the three variables that will store the location of the satellites as **s1**, **s2** and **s3**. Name the other two variable that will hold the intersecting point as **intersect1**, **intersect2**.

Step3 : Scripting the Sprites

1. Already scripted Satellite sprites:

If you look at all the three satellite sprites, they are already scripted to drift across the stage. We do not have to do anything more for these satellite sprites.

2. Already scripted Signal sprites:

If you look at all the three signal sprites, they are already scripted. The signal sprite is supposed to follow the corresponding satellite (i.e. signal1 follows satellite1) and leave a circle-like visualization from the position of the satellite.

3. Scripting Signals & Locator sprite :

Now we need to get the location and distance of satellites from the mouse pointer. In order to do so; click on **Signal1** sprite

from the workpane, and drag a **when ___ key pressed** block



, from the **Control category** onto the scripting area, choose the **space** from the pull-down.


- To get the location of the satellite, drag the **get signal ___ to __** block , from the **Operators category** and plug it below **when ___ key pressed** block.

- Set the first field of the **get signal ___ to __** block with the location of the signal sprite, you need **__ of __** block




, from **Sensing category**. Select x&y position of the corresponding sprite. If you are working on signal1 then, you will need x&y position of signal1.


- Since we are working on signal1 sprite, plug in the variable **s1** into the second field of the **get signal ___ to __** block. The variable **s1** will hold the location of signal1 sprite
- To make sure the variable is always empty before storing the location, use **set ___ to __** block, from the **Variables category** and set the first field with **s1** variable name and second with 0 or nil.
- Repeat this for the remaining signal2 and signal3 sprites, and store the locations in **s2** and **s3** variables.
- To let the locator sprite know when all the locations are re-

ceived, you need to add **broadcast** __ block  from the **Control category** and create a new message something like “**Location received**”. This block should be added at signal3 sprite.

By now we have three variables that hold the locations, This sprite will perform the intersection of the variables that hold the location. The Intersection will give a common point, which will be our result.

We need to perform the intersection only when all the three variable hold the locations, click on **Locator** sprite from the work-

pane, and drag a **when I receive** ___ block , from the **Control category**. Choose the message you created from pulldown.

- To perform the intersections between the two locations, you might need a __ **intersect** __ to __ block , from the **Pen category**. Set the first and second field with two variable **s1** and **s2** and store the resulting intersection point in **intersect1** variable, so plug intersect1 variable in the third field of __ **intersect** __ to __ block.
- Perform the intersection again using the same __ **intersect** __ to __ block, this time with **s2** and **s3** variable and store the result in **intersect2** variable.

- By now both the variable **intersect1** and **intersect2** will hold at least one common point which is our result. To find that use `find common point from -- -- block` from the **Operators** category. Plug in **intersect1** and **intersect2** variables for the sprite to answer the location of a mouse pointer.

Note: Make sure the mouse-pointer is always on the stage when space key is pressed.

A.4 Robotics Lab Description

The aim of this lab is to give you an understanding about how robots can be programmed to solve problems. For the purpose of this lab, you will be scripting the robot (i.e. sprite) to solve a Maze. You can use the help of algorithms to solve the problem.

“Algorithm is the step by step procedure to solve any given problem.”⁶

There are Maze generating algorithms and Maze solving algorithms. For this lab our aim is to solve a Maze, we will focus on Maze solving algorithms. There are six Maze solving algorithms, the random mouse algorithm, the wall follower or the left-hand rule, pledge algorithm, Tremaux’s algorithm, dead-end filling and shortest path algorithm.

You will be implementing only the wall follower algorithm and Tremaux’s algorithm for this lab because they are easy to understand and implement.

In the Scratch/Phratch environment, we will use blocks in performing the following tasks:

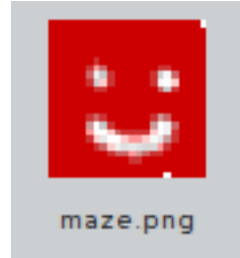
1. We need to setup the lab with a sprite image that will act like a robot in solving the Maze, then generate a Maze.
2. We need to script the robot (i.e. sprite) to implement the wall follower algorithm.
3. We will implement Tremaux’s Algorithm, that is the other way to solve a Maze.

⁶<http://en.wikipedia.org/wiki/Algorithm>

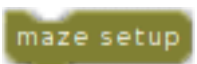
Step 1: Setup Robotics Lab

1. We need a Sprite that will act as a Robot in solving a maze.

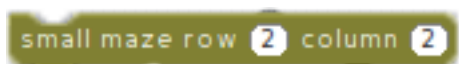
In order to do so; click on . This will open New Sprite window, select **maze.png** image from the window. Please use



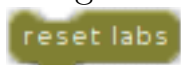

only this sprite for the lab,

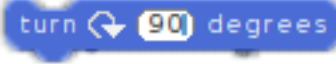
2. We need to generate a maze, and you need to click just once on the **maze setup** block located at the **LabSetup category** . This will generate a maze on the stage. The maze is generated using the depth-first search algorithm.

or

You can also create a smaller maze with a certain number of rows and columns, you need to use **small maze rows -- columns --** block, from the **LabSetup category** .

Fill the first and the second field with any number less than seven. This will generate a smaller maze at the center of the stage.

3. You can reset the maze on the stage using **reset labs** block from the **LabSetup category** , and **clear all trails** block, from the **Pens category** . These blocks will reset the labs and clear any trails on the stage.

4. Make sure sprite is always facing open wall every time you run the script. In order to do so use **turn** ___ block from the **Motion category** . The block can be either clockwise or counter clockwise and set the field with 90.

Step 2: Solving a Maze using the Wall follower algorithm

1. Wall follower algorithm or Left hand rule:

The wall follower algorithm is very simple and easy to implement. The way this algorithm works is:

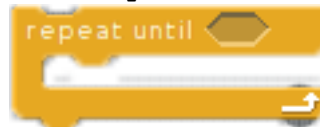
each time the robot takes certain steps, it has to follow the wall on the left-hand side of it.

- Drag the **when flag clicked** block from the **Control category**



, to initiate the program.

- You want to move the robot (i.e. sprite) until it reaches the other end of the maze (i.e. finish). In order to do so, you need a **repeat until** ___ block from the **Control**



category . Plug in the **finish** block, from the **Sensing category** 

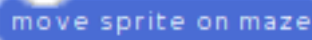
Everything below goes under the **repeat until** ___ block.

- We are going to check if there is a wall on the left. In order to do so, you need a conditional `If __ else __`



block, from the **Control category** . Plug in `wall on left` block, from the **Sensing category** .

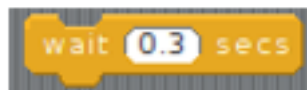
- If the above condition is true, you need to move the sprite. You need a `move sprite on maze` block, from the **Motion category** .



- If the above condition is False (i.e. `else`), this condition can only be false if sprite is next to wall edge. In order to solve this you need a `solve maze edge` block, from the **Motion category** .



- To make the sprite move a little slower on the stage, you can use `wait __ secs` block, from the **Control category**



. Set the field with anything less than 1 sec (i.e. 0.5 or 0.3). Plug in this block before the `move sprite on maze` block.

By now you have solved the maze using the wall follower algorithm.

Note: You can implement only the wall follower algorithm with small maze.

Step 3: Solving a Maze using Tremaux's algorithm "Tremaux's algorithm"⁷

is one of the most popular and successful algorithms for guaranteeing a solution to the maze, but it does involve you being able to remember what paths you have taken. This algorithm consists of a few simple rules:

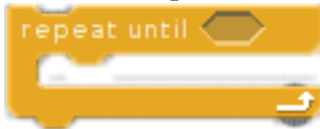
- No path is taken more than twice (ideally, although sometimes this does not apply).
- When you come to a new junction, take any path you want.
- When a new path leads to a junction you have already been at, treat it as a dead end and go back to your previous junction to take the path you have taken least."


1. Drag the **when flag clicked** block from the **Control** cate-



gory, to initiate the program.

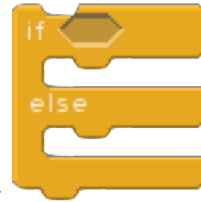
2. You want to move the robot (i.e. sprite) until it reaches the other end of the maze (i.e. finish). In order to do so, you need a **repeat until** ___ block from the **Control** category,



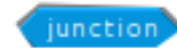
. Plug in the **finish** block, from the **Sensing** category . Everything below goes under the **repeat until** ___ block.

- Now you might want to check if there is a junction. In order to do so, you need a conditional **If** __ **else** __ block, from

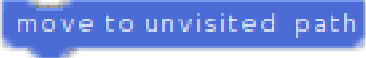
⁷<http://www.ramseyerfarms.com/blog/?p=125>



the **Control** category, . Plug in junction

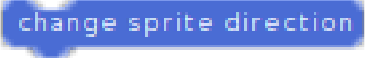


block, from the **Sensing** category .

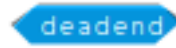
- If the above condition is true, you need to take a random path that has not be travelled before. You need a move to unvisited path block, from the **Motion** category  .

- If the above condition is false (i.e. **else**), you need to check if the sprite is touching maze wall. In order to solve this you need a you need a conditional If -- else -- block, from the **Control** category. Plug in touching -- block, from

the **Sensing** category  , choose **touching maze wall** from the pulldown.

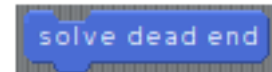
- If touching maze wall condition is true, you need to change the direction of the maze. In order to do so use change sprite direction block, from the **Motion** category  .

- If the above condition is false (i.e. **else**), you need to check if the sprite is at a dead end. In order to solve this, you need a conditional If -- else -- block from the **Control** category, and plug in dead end block, from



the **Sensing category** .

- * If dead end condition is true, you need to make the sprite turn back. In order to do so use `solve dead end`



block, from the **Motion category** .

- * If the above condition is false (i.e. **else**), then we want to move the sprite on maze. In order to do so, you need a `move sprite on maze` block, from the

Motion category  .

Note: Red circles on the maze indicate path taken once.
White circles on the maze indicate junctions.
Green circles on the maze indicate path taken twice.

This algorithm won't work on a small maze.

A.5 Secrets and Intellectual Property Lab Description

The aim of this lab is to give you some understanding of how Encryption is done. “Encryption is the process of encoding messages (or information) in such a way that only authorized parties can read it. Information or messages are encrypted using an encryption algorithm which results in an unreadable ciphertext.”⁸

For the purpose of this lab, we will focus on the one-time pad (OTP) algorithm. The “one-time pad”⁹ is a type of encryption that is impossible to crack if used correctly. Each bit or character from the plaintext is encrypted by a modular addition with a bit or character from a secret random key (or pad) of the same length as the plaintext, resulting in a ciphertext. If the key is truly random, at least as long as the plaintext, never reused in whole or part, and kept secret, the ciphertext will be impossible to decrypt or break without knowing the key.”

In the Scratch/Phratch environment, we will use blocks to perform the one-time-pad algorithm.

One-time-pad algorithm “For example”¹⁰, if Alice wants to send an encrypted message of ‘Hello’ to Bob. Assume two pads of paper containing identical random sequences of letters were somehow previously produced and securely issued to both. Alice chooses the appropriate unused page from the pad. The way to do this is nor-

⁸<http://en.wikipedia.org/wiki/Cipher>

⁹http://en.wikipedia.org/wiki/One-time_pad

¹⁰http://en.wikipedia.org/wiki/One-time_pad


mally arranged for in advance, as for instance ‘use the 12th sheet on 1 May’, or ‘use the next available sheet for the next message.’ The material on the selected sheet is the key for this message. Each letter from the pad will be combined in a predetermined way with one letter of the message. It is common, to assign each letter a numerical value: e.g. “A” is 0, “B” is 1, and so on. In this example, the technique is to combine the key and the message using modular addition. The numerical values of a corresponding message and key letters are added together, modulo 26. If the number is larger than 26, then the remainder after subtraction of 26 is taken in a modular arithmetic fashion. This simply means that if the computations “go past” Z, the sequence starts again at A. If key material begins with “BCACZ” and the message is “HELLO,” then the resulting encryption will be “IGLNN”. For more complex example with a key beginning with “XMCKL” and the same message, “HELLO,” the resulting encryption will be “EQNVZ”.”

1. Drag the **when flag clicked** block from the **Control category**



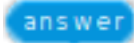
, to initiate the program.

2. You want to ask the user a message to be encrypted. In order to do so, use **ask** __ and **wait** block, from the **Sensing category** . Set the field with “what’s your message?”.
3. You need to create a secret key. In order to do so, create a variable from the **Variable category** and name it anything

you like. You need a **set** __ **to** __ block, from the **Variable category** . Set the field with your own secret key (i.e. type in any alphabetic characters you like), let's call this **secret key**.

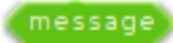
4. Once you have both secret key and message, use **Enter Message** __ **Create s**



block from the **Labs Setup category** .
Set the first field with **answer** block from **Sensing category**  since it holds the message. Next plug in the variable name block from the **Variable category** which holds the secret key in the second field.

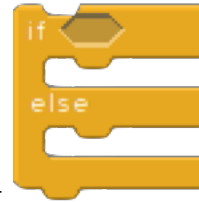
5. Now we want to encrypt the whole message by picking each letter one-by-one from it. In order to do so, you need an

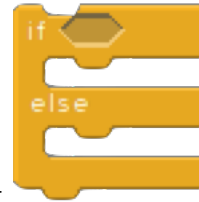






repeat until __ block from the **Control category** ,
and to perform the encryption for the entire message, plug in **message** block from the **Operator category**  , into the **repeat until** __ block.

Everything below goes inside the **repeat until** __ block.

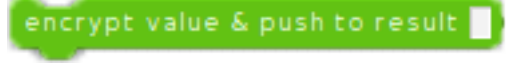

- (a) Now you need to pick characters one-by-one from the message and the secret key to check if the sum of each letter is greater than 26. In order to do so, you need an **if** __ **else** __



block from the **Control category**, . Now for checking if the sum of letters is greater than 26, set the **if** **else** block field with **>** block from the **Operators category** .

- (b) Plug in the **+** block from the **Operators category** , in the first field of **>** block.
- (c) Set the first and second field of **+** block, with **pick letter from message**  and **pick letter from secret key**  blocks, from the **Operators category**.


Since we want to check if the sum is greater than 26, set the second field of **>** block with 26.

- i. If the above condition is true, we want to subtract 26 from the sum. The subtracted value will give us the corresponding encrypted letter. In order to do so, you need an **encrypt value & push to result** block from the **Operator category** . Set the field with **-** block from the **Operators category** . Repeat the above step, set the first field of **-** block with **+** block from the **Operators category**.
- ii. Set the first and second field of **+** block, with

pick letter from message and pick letter from secret key blocks from the **Operators category**. Since we want to subtract 26 from the sum, set the second field of __ - __ block to 26.

iii. If the above condition is false (i.e. else), we want to get the sum of the letters. The sum will give us the corresponding encrypted letter. In order to do so, you need an encrypt value & push to result block from the **Operator category**. Set the field of encrypt value & push to result block with __ + __ block from the **Operators category**.

iv. Set the first and second field of __ + __ block, with pick letter from message and pick letter from secret key blocks from the **Operators category**.

(d) If you want to run the program again, use the reset labs block, from the **Labs Setup category** .

A.6 Voting, Finance and Politics Lab Description

The aim of this lab is to give you some understanding about Voting systems and Finance (stock market).

Voting For the purpose of this lab, we will be learning about Condorcet method, which is one of the methods that is used in voting systems. “The Condorcet method¹¹ is used to determine the winner of each of the possible one-on-one elections among the candidates. The Condorcet winner would be the candidate that beats all of it’s opponents. In a Condorcet election, the voter ranks the list of candidates in order of preference. For example, if there are five candidates Joe Smith, John Citizen, Jane Doe, Fred Rubble and Mary Hill. The voter can give 1st preference to John Citizen, 2nd preference to Mary Hill, 3rd preference to Jane Doe and so on.”

¹¹http://en.wikipedia.org/wiki/Condorcet_method

Rank any number of options in your order of preference.

- ☐ Joe Smith
- ☒ **1** John Citizen
- ☒ **3** Jane Doe
- ☐ Fred Rubble
- ☒ **2** Mary Hill

If 100% voters voted among the above candidates in order of their preference and the voting data generated is as follows:

35% voters 1st Joe Smith, 2nd John Citizen, 3rd Jane Doe, 4th Fred Rubble, 5th Mary Hill

25% voters 1st John Citizen , 2nd Joe Smith, 3rd Jane Doe, 4th Fred Rubble, 5th Mary Hill

20% voters 1st Joe Smith, 2nd Jane Doe, 3rd John Citizen, 4th Fred Rubble, 5th Mary Hill

15% voters 1st Fred Rubble, 2nd John Citizen, 3rd Jane Doe, 4th Joe Smith, 5th Mary Hill

5% voters 1st Mary Hill, 2nd John Citizen, 3rd Jane Doe, 4th Fred Rubble, 5th Joe Smith

From the above data, in order to find the Condorcet winner each candidate is compared with the other candidates. For example, if

I think Joe Smith is the winner, Joe smith needs to beat all of his opponents. This means how many voters preferred Joe Smith over all the other candidates. If the number of voters who prefer Joe Smith are greater in total than all the other candidate preferences then Joe Smith will be the Condorcet winner.

In the Scratch/Phratch environment, we will use blocks to generate voters data and then use that data to find out the Condorcet winner.

1. Drag the **when flag clicked** block from the **Control category**




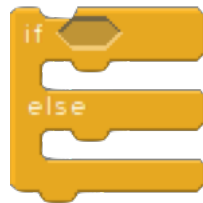
gory, to initiate the program.

2. We need a variable that will hold the voting data. In order to do so, create a new variable from the **Variables Category**.
3. Now we need to generate the voting data. You need a **--Voters Rank 1st -- 2nd -- 3rd -- 4th -- in --** block from the **Operators category**.



4. Set the first field with any number you like and it would depend on how you divide 100 voters. If I am dividing 100 into five groups of voters (i.e. 35, 25, 20, 15 and 5), then I would need five **--Voters Rank 1st -- 2nd -- 3rd -- 4th -- in --** blocks.
5. Now you need to rank the candidates in order of your preferences, choose the names from the pulldown for 1st, 2nd, 3rd and 4th preference.

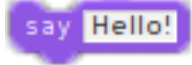
6. We want the variable to hold the voting information, drag the variable block that you have created earlier, from the **Variables category**, and plug it into the last field of `--Voters Rank 1st -- 2nd -- 3rd -- 4th -- in --` block.
7. You need four more `--Voters Rank 1st -- 2nd -- 3rd -- 4th -- in --` blocks, if you are dividing voters in five groups. Repeat the above steps with different number of voters and preferences.
8. To view the data you have generated, you can use `display voters data from --` block from the **Labs Setup category** . Plug in the variable block that you have created into it from the **Variables category**.
9. To find out the Condorcet winner, you need to check each candidate one-by-one if he is preferred over other candidates. In order to do so, you need an `If -- else` block from the **Control**



category `--`, plug in `-- Preferred over -- and -- and -- in --` block from the **Operators category**



- Choose any candidate name from the pulldown to set it as the first field. For example, if you are checking if **Irfan** is winner, set the first field as **Irfan** and set the other fields as **Dave**, **Mark** and **John**.

- Plug in the variable that holds the voting data into the last field of `-- Preferred over -- and -- and -- in --` block.
- If **Irfan** is the winner, i.e. if the above condition is true, you need a **say** `--` block from the **Looks category** . Set the field as “Irfan is the winner!”
- If the above condition is false (i.e. else), you might need to check if another candidate is the winner. Drag one more **If** `-- else` block from the **Control category** and plug in `-- Preferred over -- and -- and -- in --` block from the **Operators category**. This time you might want to check if **Dave** is the winner set the first field as **Dave** and set the other fields as **Irfan**, **Mark** and **John**.
 - If the above condition is true (i.e. Dave is the winner), you might need a **say** `--` block from the **Looks category**. Set the field as “Dave is the winner!”
 - If the above condition is false (i.e. else), You might want to repeat the steps from D, and check if Mark is the winner, if not check if John is the winner.

Finance In this lab, we will be using Yahoo Finance API to pull stocks. For the purpose of this lab, you will be pulling stocks of Toronto stock Exchange (TSX) market. You can perform your own task for this part of the lab. For example, to give you some idea, we can pull the asking price of Rogers and Bell and find out which one was Higher.

1. Drag the **when flag clicked** block from the **Control category**,



, to initiate the program.

2. You need an **url stock __ to __** block from the **Operators category**



, to pull stocks using the Yahoo Finance API.

3. Set the first field of the **url stock __ to __** block with the URL below.

<http://finance.yahoo.com/d/quotes.csv?s=RCI-B.TO&f=a>
(Rogers)

RCI-B.TO in the above URL is the symbol of the Rogers company, so if you want to pull stock for **APPLE Inc**, instead of using **RCI-B.TO** you will use **AAPL** in the above URL.

Alphabet 'a' at the end of the URL will show you the asking price for Rogers.

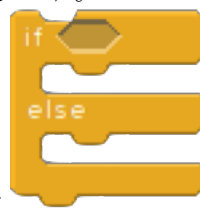
4. Set the second field of the **url stock __ to __** with a variable name that you have created from the **Variables category**, for now I have created Rogers, so plug in Rogers variable.
5. Similarly, to pull the asking price for Bell, you need one more **url stock __ to __** block from the **Operators category** and

set the first field with the URL.

http://finance.yahoo.com/d/quotes.csv?s=BA.TO&f=a
(Bell)

plug in variable named bell, at the second field of `url stock __`
`to __` block.

- Now that you have the asking price of bell and rogers, you can



use an `If __ else` block from the **Control category**,
to find out the one with higher asking price. Plug in `-->--` block
from the **Operators category**.



- Plug in the Bell variable into the first field and the Rogers variable into the second field of `-->--` block.
- If the above condition is true (i.e. Bell asking price is greater than Rogers), you might want to use `Say __` block from the **Looks category** and set the field saying “Bell is Higher.”
- If above condition is false (i.e. Bell asking price is lower than Rogers), you might want to use `Say __` block from the **Looks category** and set the field saying “Rogers is Higher.”

This one was a simple task, and you can perform your own tasks to get the averages, dates, closing price and dividends. The link below will guide you USING YAHOO FINANCE API¹².

¹²http://www.jarloo.com/yahoo_finance/

A.7 Security and Military Computation Lab Description

The aim of this lab is to give student's some understanding about how location tracking works.

"Location tracking is not one, singular technology. Rather, it is the convergence of several technologies that can be merged to create systems that track inventory, livestock or vehicle fleets. Similar systems can be created to deliver location-based services to wireless devices" Bonsor, Kevin on "How Location Tracking Works."¹³

Current technologies being used to create location-tracking and location-based systems include:

- Geographic Information Systems (GIS)¹⁴
- Global Positioning System (GPS)¹⁵
- Radio Frequency Identification (RFID)¹⁶
- Wireless Local Area Network (WLAN)¹⁷

The magnetic strip at the back of your Ryerson one card can also be used to track your location. These magnetic strips have information stored in them which is helpful in tracking, and are most commonly used in credit cards. More information on how magnetic strips work's can be found here.¹⁸

¹³<http://electronics.howstuffworks.com/everyday-tech/location-tracking1.htm>

¹⁴<http://electronics.howstuffworks.com/everyday-tech/location-tracking1.htm>

¹⁵<http://electronics.howstuffworks.com/everyday-tech/location-tracking1.htm>

¹⁶<http://electronics.howstuffworks.com/gadgets/high-tech-gadgets/rfid.htm>

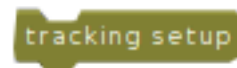
¹⁷<http://electronics.howstuffworks.com/everyday-tech/location-tracking1.htm>

¹⁸<http://money.howstuffworks.com/personal-finance/debt-management/magnetic-stripe-credit-card.htm>

For the purpose of this lab, we will access fabricated secure data, which has information about random persons. The information will include the name of the person, rooms that random person has been to and the time spent by him in each room.

In a Scratch/Phratch environment, we will be using blocks to access the secure information and create a visualization that will help us track each person one by one. The visualization will give us a clear understanding about the movements and the time spent by each person in various rooms.

1. Before you begin tracking information, you need to setup the lab and generate fabricated data of each person that we are going to



track. Click on **tracking setup** block once from the **Labs Setup category**. This will create room images on the stage and the lists of random persons names and rooms that we will be tracking in this lab.

2. Drag the **when flag clicked** block from the **Control category**



to initiate the program.

3. You need to create three variables from the **Variables category** that will hold the name, time and room information for each

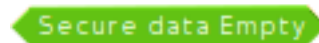


person.

- Now you want to read the data of each person one by one with the time and the room they are in. You need a **repeat until** --



block from the **Control category**, to keep reading the fabricated data until it's empty. You need to plug in



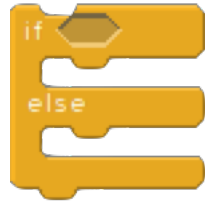
a **secure data empty** block, from the **Operators category** into **repeat until** -- block. Everything below goes inside **repeat until** -- .

- To read the fabricated data of each person one by one. You need a **read from secure data name -- room -- Time --** block



, from the **Operators category**. Set the first field with **name** block variable, second with **room** block variable, and the third with **time** block variable.

- Each variable holds a piece of information. You need to check if the **name** variable block that holds the name of the person is already in the **Names** list. In order to do so, you need an **if** --

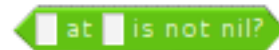


else block _____, from the **Control category**. Plug



in a __ contains __ block _____, from the **Variables category** into the if __ else block. Choose **Names** list name from the pulldown for the first field of __ contains __ block and set the second field with **name** variable block.

- If the above condition is true, you need to check if **room** variable block has information of the person that we are currently reading on. In order to do so, you need another if __ else block, from the **Control category**. Plug in a room



__ at name__ is not nil? block _____, from the **Operators category**. Set the first field with **room** variable block and the second field with **name** variable block.

- If the above condition is true, In order to track the person movements between the rooms and the time spent in each room, you need a draw line of __ with room



__ time __ block _____, from the **Pens category**. Set the first field with **name** block variable, second with **room** block variable, and the third

with **time** block variable.

- If the above condition is false (i.e. else), you need to generate data for the current person. In order to do so, you need a **generate secure data of name** __ with



__ block, from the **Labs Setup category**. Set the first field with **name** variable block and choose **Rooms** list name from the pull-down for the second field.

- If the above condition is false (i.e. else), you need to add the name of the person that the **name** variable block holds in the **Names** list. In order to do so, you need an **add** __



to __ block, from the **Variables category**. Set the first field with **name** variable block and choose **Names** list name from the pulldown for the second field.

7. You can add your name to the names list and track it. In order to do so, right below the **when flag clicked** block, drag an **add** __ to __ block, from the **Variables category**. Set the first field with your name and choose **Names** list name from the pulldown for the second field.

A.8 Transportation and Medical Lab Description

The aim of this lab is to facilitate an understanding of Transportation (autonomous vehicle) and Medical (CAT-scan).


Transportation Transportation is part of everyday life, for the purpose of this lab we will be focusing on how **autonomous vehicles** work. “Autonomous vehicle¹⁹ or autonomous car is a car with no driver, and it is also known as a self-driving car. An autonomous vehicle is capable of sensing its environment and navigating without human input. Autonomous vehicles sense their surroundings with such techniques as radar, lidar, GPS, and computer vision. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage. Some autonomous vehicles update their maps based on sensory input, allowing the vehicles to keep track of their position even when conditions change or when they enter uncharted environments.”

In this lab, we will be working on a line following autonomous car. We will draw a path on the stage of the Scratch/Phratch environment and script the autonomous car to follow the path.

Transportation Lab setup

1. Click on , to open a new sprite image. Select the sprite

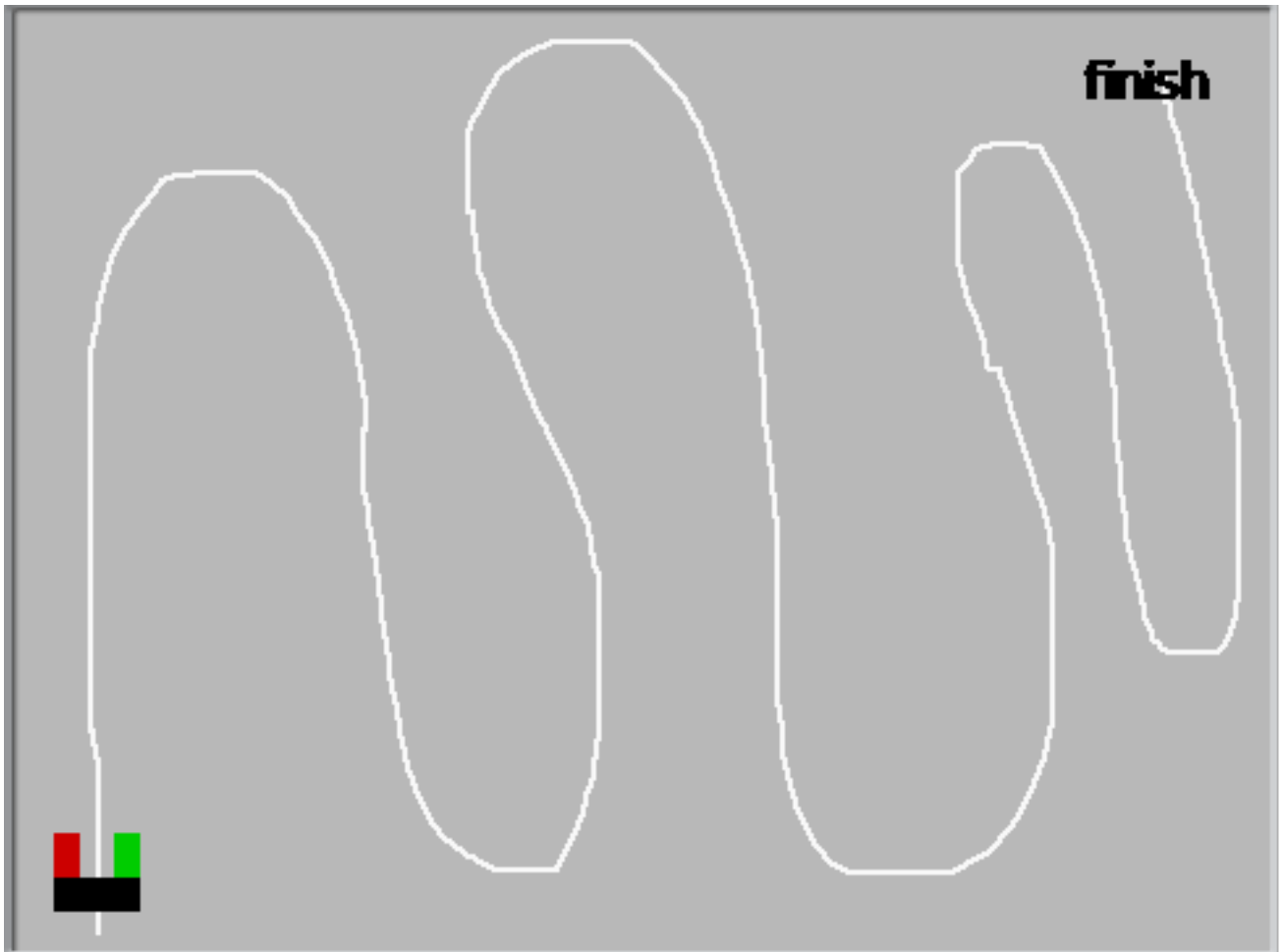
¹⁹http://en.wikipedia.org/wiki/Autonomous_car

name transport.png , from the new sprite window. The transport.png sprite will act as autonomous car.



2. Next click on **transport setup** block once , from the **Labs Setup category** this will let you freely draw path on the stage.

3. Now you can freely draw a path on the stage with the help of a mouse pointer. You might also need a new sprite that will act as a finish point for autonomous vehicle. Place the autonomous car sprite at the starting point of the path drawn and place the other sprite at the end of the path. By now the stage will look something like this as shown below.




4. You can use **reset labs** block , from the **labs Setup category** to redo the lab with different path.

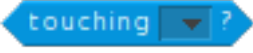
Scripting the Sprite

1. Drag the **when flag clicked** block from the **Control category**,


, to initiate the program.


2. Now you want to move the autonomous sprite until it reaches the

finish point. You need a **repeat until** __ block , from the **Control category**. Now that you have already cre-

ated a finish point sprite, plug in **touching** __ block , from the **Sensing category** into **repeat until** __ block. Set the field with the finish point sprite name. Everything below goes inside **repeat until** __ .

3. The red and green colour of the autonomous sprite will act as sensors which will sense the path. In order to do so, you need an

if __ **else** __ block , from the **Control category**. To sense if green colour of autonomous sprite is touching the path, you need to plug in **color** __ **is touching** __ block

, from the **Sensing category** into **if** __ **else** __ block . Click on the first field and set it with green colour, click on the second field and set it with white colour.

- (a) If the above condition is true, you want to turn the autonomous sprite clock-wise. In order to do so, drag **turn**

 **clock wise** __ **degree** block , from

the **Motion category**. Set the field with 15 degrees.


- (b) If the above condition is false (i.e. else), you want to check if autonomous sprite's red colour is touching the path drawn. Repeat step **3** however this time set the first and second field of colour `-- is touching --` the block with red colour and white colour.


- i. If the above condition is true, you want to turn the autonomous sprite counter clock-wise. In order to do so, drag `turn counter clock wise -- degree` block



, from the **Motion category**.

Set the field with 15 degrees.

- ii. If the above condition is false (i.e. else), you want to move the autonomous sprite. In order to do so, you need a `move -- steps` block , from the **Motion category**. Set the field with 10 or 5 , depends on how fast you want to move the sprite.

4. You can use a `reset labs` block , from the **Labs setup category** to clear the stage.

Medical (CAT- scan) “The CAT scan (Computerized axial tomography) or CT scanner uses digital geometry processing to generate a 3-dimensional (3-D) image of the inside of an object. The 3-D image is made after many 2-dimensional (2-D) X-ray images are taken around a single axis of rotation - in other words, many pictures of

the same area are taken from many angles and then placed together to produce a 3-D image.

How does a CAT-scan work? A CAT scan emits a series of narrow beams through the human body as it moves through an arc, unlike an X-ray machine which sends just one radiation beam. The final picture is far more detailed than an X-ray image.

Inside the CAT scan, there is an X-ray detector which can see hundreds of different levels of density. It can see tissues inside a solid organ. This data is transmitted to a computer, which builds up a 3-D cross-sectional picture of the part of the body and displays it on the screen.

A CAT scan uses a computer that takes data from several X-ray images of structures inside a human's or animal's body and converts them into pictures on a monitor.” Media News today Article²⁰

For the purpose of the lab, we will be generating density values of an object. Once we have the density values, we will convert the values into picture of the object scanned.

In Scratch/Phratch environment we will be using blocks to perform the following tasks:

1. We will generate density values of an object scanned and add them to the list.
2. We will use density values from a list, and perform a scan at different angles. This will lead to some vague image of the object on the stage.

²⁰<http://www.medicalnewstoday.com/articles/153201.php>

CAT- scan

1. Drag the **when flag clicked** block from the **Control category**,



to initiate the program.

2. Create a new list from the **Variables category** , and name it anything you like. This list will hold the density values of the scanned object.

3. To generate the density values and add them to the list, you need

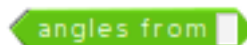


a **get density values for CAT-scan** block from the **Labs Setup category** . Plug in the list block from the **Variables category** into the **get density values for CAT-scan** block.

4. Now that we have the density values, we want to perform scans at different angles. In order to do so, you need a **repeat until**

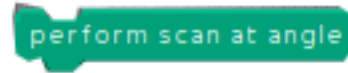


block, from the **Control category**. Plug



in **angles from** block, from the **Operators category** to perform the scan at all angles. Next, you need to plug in a list block from the **Variables category** into **angles from** block. Everything below goes inside **repeat until** block.

5. Now to perform a scan at each angle one by one, you need a



perform scan at angle block, from the **Pens category**.



6. You need a wait __ secs block, from the **Control category**. Set the field with 0.2 secs, this block will give you a clear view of each scan being performed one by one.



7. You can use a reset labs block, from the **Labs setup category** to clear the stage.

Appendix B

Survey Questionnaire

Participants code:

Lab name:

1. Do you feel like you possess prior knowledge or some vague idea about the system you explored in this lab?
 - (a) strongly agree
 - (b) slightly agree
 - (c) slightly disagree
 - (d) strongly disagree

2. I felt the visualization of the system was effective enough in providing good understanding
 - (a) strongly agree
 - (b) slightly agree
 - (c) slightly disagree
 - (d) strongly disagree

3. I felt the lab was interesting and I liked it
 - (a) strongly agree
 - (b) slightly agree
 - (c) slightly disagree
 - (d) strongly disagree
4. I think the lab description given to me was helpful in completing the lab successfully
 - (a) strongly agree
 - (b) slightly agree
 - (c) slightly disagree
 - (d) strongly disagree
5. I felt the lab was supportive enough in facilitating a better understanding of the system
 - (a) strongly agree
 - (b) slightly agree
 - (c) slightly disagree
 - (d) strongly disagree
6. The duration of this lab was
 - (a) too short
 - (b) short
 - (c) right

(d) long

(e) too long

7. I feel that I performed up to my potential in this lab.

(a) strongly agree

(b) slightly agree

(c) slightly disagree

(d) strongly disagree

8. I feel that this lab is important in learning this week's course material

(a) strongly agree

(b) slightly agree

(c) slightly disagree

(d) strongly disagree

9. Do you have any opinions or suggestions

Bibliography

- [1] Alice. Last visited: 22 April 2014. URL: <http://www.alice.org>.
- [2] Casey Alt et al. “Social Networks Generate Interest in Computer Science”. In: *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '06. Houston, Texas, USA: ACM, 2006, pp. 438–442. ISBN: 1-59593-259-3. DOI: 10.1145/1121341.1121477. URL: <http://doi.acm.org/10.1145/1121341.1121477>.
- [3] Kevin Bonsor. *How Location Tracking Works*. Last visited: 19 April 2014. May 2001. URL: [HowStuffWorks.com.%20%3Chttp://electronics.howstuffworks.com/everyday-tech/location-tracking.htm%3E](http://electronics.howstuffworks.com/everyday-tech/location-tracking.htm).
- [4] Kevin Bonsor and Wesley Fenlon. *How RFID Works*. Last visited: 22 April 2014. Nov. 2007. URL: [HowStuffWorks.com.%20%3Chttp://electronics.howstuffworks.com/gadgets/high-tech-gadgets/rfid.htm%3E](http://electronics.howstuffworks.com/gadgets/high-tech-gadgets/rfid.htm).
- [5] M.M. Burnett and AL. Ambler. “A declarative approach to event-handling in visual programming languages”. In: *Visual Languages, 1992. Proceedings., 1992 IEEE Workshop on*. Sept. 1992, pp. 34–40. DOI: 10.1109/WVL.1992.275786.
- [6] CAR: Components, Agents, and Robots with Dynamic Languages. Last visited: 22 April 2014. URL: <http://car.mines-douai.fr/category/phratch/>.
- [7] MNT Knowledge Center. *How CAT Scan Works*. Last visited: 19 April 2014. June 2009. URL: <http://www.medicalnewstoday.com/articles/153201.php>,.

- [8] Peng Chao-Ying Joanne and Ziskin Mary B. *Control Group*. Last visited: 27 May 2014. URL: <http://srmo.sagepub.com/view/encyclopedia-of-survey-research-methods/n102.xml>.
- [9] Wayne Citrin, Michael Doherty, and Benjamin Zorn. “The design of a completely visual object-oriented programming language”. In: *Visual Object-Oriented Programming: Concepts and Environments*. Prentice-Hall, New York (1995).
- [10] Stephen Cooper, Wanda Dann, and Randy Pausch. “Teaching Objects-first in Introductory Computer Science”. In: *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE ’03. Reno, Nevada, USA: ACM, 2003, pp. 191–195. ISBN: 1-58113-648-X. DOI: 10.1145/611892.611966. URL: <http://doi.acm.org/10.1145/611892.611966>.
- [11] Stephen Cooper, Wanda Dann, and Randy Pausch. “Using Animated 3D Graphics To Prepare Novices for CS1”. In: *Computer Science Education Journal* 13 (2003), pp. 28–29.
- [12] Phillip T Cox and Tomasz Pietryzkowsky. *Using a pictorial representation to combine dataflow and object-orientation in a languageindependent programming mechanism*. Visual Programming Environments: Paradigms and Systems. Los Alamitos, CA: IEEE Computer Society Press, 1990.
- [13] *FreegeoIp*. Last visited: 22 April 2014. URL: <http://freegeoip.net/>.
- [14] *GPS Applications*. Last visited: 21 April 2014. URL: <http://www.gps.gov/applications/>.
- [15] Mark Guzdial. “A Media Computation Course for Non-majors”. In: *SIGCSE Bull.* 35.3 (June 2003), pp. 104–108. ISSN: 0097-8418. DOI: 10.1145/961290.961542. URL: <http://doi.acm.org/10.1145/961290.961542>.
- [16] Susanne Hambrusch et al. “A Multidisciplinary Approach Towards Computational Thinking for Science Majors”. In: *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*. SIGCSE ’09. Chattanooga, TN, USA: ACM, 2009, pp. 183–187. ISBN: 978-1-60558-183-5. DOI: 10.1145/1508865.1508931. URL: <http://doi.acm.org/10.1145/1508865.1508931>.

- [17] JAWAA. Last visited: 22 April 2014. URL: <http://www.cs.duke.edu/csed/jawaa2>.
- [18] Bergin Joseph et al. *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming*. John Wiley and Sons Inc, 1996.
- [19] Jython. Last visited: 22 April 2014. URL: <http://www.jython.org>.
- [20] Caitlin Kelleher and Randy Pausch. “Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers”. In: *ACM Comput. Surv.* 37.2 (June 2005), pp. 83–137. ISSN: 0360-0300. DOI: 10.1145/1089733.1089734. URL: <http://doi.acm.org/10.1145/1089733.1089734>.
- [21] *LifeLong Kindergarden Group*. Last visited: 21 April 2014. URL: <http://llk.media.mit.edu/>.
- [22] John Maloney et al. “The Scratch Programming Language and Environment”. In: *Trans. Comput. Educ.* 10.4 (Nov. 2010), 16:1–16:15. ISSN: 1946-6226. DOI: 10.1145/1868358.1868363. URL: <http://doi.acm.org/10.1145/1868358.1868363>.
- [23] Boshernitsan Marat and Downes Michael.S. “Visual Programming Languages: A Survey”. In: (Dec. 2004).
- [24] P. Margolis and A. Fisher. “Unlocking the Clubhouse: Women in Computing”. In: *MIT Press* (2002).
- [25] F.P. Miller, A.F. Vandome, and M.B. John. *Maze Solving Algorithm*. VDM Publishing, 2010. ISBN: 9786131652530. URL: <http://books.google.ca/books?id=L1BOYAAACAAJ>.
- [26] *PewResearch Internet Project*. Last visited: 21 April 2014. URL: <http://www.pewinternet.org/fact-sheets/social-networking-fact-sheet/>.
- [27] *Pharo*. Last visited: 21 April 2014. URL: <http://www.pharo-project.org/home>.
- [28] *Phratch*. Last visited: 21 April 2014. URL: <http://www.phratch.com/>.

- [29] Lori Pollock et al. “Increasing High School Girls’ Self Confidence and Awareness of CS Through a Positive Summer Experience”. In: *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE ’04. Norfolk, Virginia, USA: ACM, 2004, pp. 185–189. ISBN: 1-58113-798-2. DOI: 10.1145/971300.971369. URL: <http://doi.acm.org/10.1145/971300.971369>.
- [30] *Python*. Last visited: 21 April 2014. URL: <https://www.python.org/>.
- [31] Mitchel Resnick et al. “Scratch: Programming for All”. In: *Commun. ACM* 52.11 (Nov. 2009), pp. 60–67. ISSN: 0001-0782. DOI: 10.1145/1592761.1592779. URL: <http://doi.acm.org/10.1145/1592761.1592779>.
- [32] Mona Rizvi et al. “A CS0 Course Using Scratch”. In: *J. Comput. Sci. Coll.* 26.3 (Jan. 2011), pp. 19–27. ISSN: 1937-4771. URL: <http://dl.acm.org/citation.cfm?id=1859159.1859166>.
- [33] Susan H. Rodger. “Introducing Computer Science Through Animation and Virtual Worlds”. In: *SIGCSE Bull.* 34.1 (Feb. 2002), pp. 186–190. ISSN: 0097-8418. DOI: 10.1145/563517.563411. URL: <http://doi.acm.org/10.1145/563517.563411>.
- [34] Frank Rubin. “ONE-TIME PAD CRYPTOGRAPHY”. In: *Cryptologia* 20.4 (1996), pp. 359–364. DOI: 10.1080/0161-119691885040. eprint: <http://www.tandfonline.com/doi/pdf/10.1080/0161-119691885040>. URL: <http://www.tandfonline.com/doi/abs/10.1080/0161-119691885040>.
- [35] Alex Ruthmann et al. “Teaching Computational Thinking Through Musical Live Coding in Scratch”. In: *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. SIGCSE ’10. Milwaukee, Wisconsin, USA: ACM, 2010, pp. 351–355. ISBN: 978-1-4503-0006-3. DOI: 10.1145/1734263.1734384. URL: <http://doi.acm.org/10.1145/1734263.1734384>.
- [36] *Ryerson Research Ethics Board*. Last visited: 27 May 2014. URL: <http://www.ryerson.ca/research/services/ethics/human/reb.html>.
- [37] *Scratch*. Last visited: 21 April 2014. URL: <http://scratch.mit.edu/>.

- [38] Randall B. Smith. “Experiences with the Alternate Reality Kit: An Example of the Tension Between Literalism and Magic”. In: *SIGCHI Bull.* 18.4 (May 1986), pp. 61–67. ISSN: 0736-6906. DOI: 10.1145/1165387.30861. URL: <http://doi.acm.org/10.1145/1165387.30861>.
- [39] *StarLogo TNG*. Last visited: 22 April 2014. URL: <http://education.mit.edu/starlogo/>.
- [40] *Statistics Canada*. Last visited: 27 May 2014. URL: <http://www.statcan.gc.ca/edu/power-pouvoir/ch6/sampling-echantillonnage/5214807-eng.htm>.
- [41] *Twitter Rate Limiting*. Last visited: 21 April 2014. 2014. URL: <https://dev.twitter.com/docs/rate-limiting/1.1>.
- [42] *What is Computer Science?* Last visited: 27 May 2014. URL: <http://www.cs.mtu.edu/~john/whatiscs.html>.
- [43] Jeannette M. Wing. “Computational Thinking”. In: *Commun. ACM* 49.3 (Mar. 2006), pp. 33–35. ISSN: 0001-0782. DOI: 10.1145/1118178.1118215. URL: <http://doi.acm.org/10.1145/1118178.1118215>.