FUNdamentals of CS: Designing and Evaluating Computer Science Activities for Kids

by

Katherine Gunion
B.Sc., University of British Columbia, 2008

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

FUNdamentals of CS: Designing and Evaluating Computer Science Activities for Kids

by

Katherine Gunion
B.Sc., University of British Columbia, 2008

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Patrick McGeer, Departmental Member
(Department of Computer Science)

Dr. Ulrike Stege, Departmental Member
(Department of Computer Science)

Dr. Tim Pelton, Outside Member
(Department of Curriculum and Instruction)

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Patrick McGeer, Departmental Member
(Department of Computer Science)

Dr. Ulrike Stege, Departmental Member
(Department of Computer Science)

Dr. Tim Pelton, Outside Member
(Department of Curriculum and Instruction)

ABSTRACT

Computer Science is not included in high school or middle school education in British Columbia. Young students are not exposed to Computer Science when they are learning their fundamentals. Given the correct abstractions like kinesthetic learning activities and graphical programming languages, elementary school students can be exposed to computer science and can understand sophisticated topics like recursion and concurrency. This means that more students' interest will be piqued and they will be exposed to sophisticated concepts before first year computer science.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computer science education (CSEd) is a young field that is comprised of numerous established disciplines such as science, mathematics and psychology. Because of its relative youth and shared background, it is common for researchers in educational computer science to look to other disciplines for theory to help in answering questions. Fincher and Petra [31], in their seminal text on computer science education, suggest that for CSEd to become its own field and not remain situated within these other disciplines, researchers must begin to ask questions that may only be answered through computer science.

CSEd is also becoming increasingly important as the prevalence of computers in society increases and the need for members of industry follows suit.

This thesis explores methods of introducing young students to important computer science concepts using fun and engaging activities. Two concepts are used as case studies, recursion and concurrency. These topics were chosen because of the controversy of when and how they are supposed to be taught.

1.1 Motivation

It is hypothesized that the declining enrolment in computer science programs at post secondary institutions since 2001 is the result of a perception of decreased number of jobs in the industry, incorrect perceptions of what computer scientists do and general unfamiliarity with the discipline [20]. Although there is potentially little we can do to address the first reason, interventions to adjust and improve the perceptions that computer scientists spend their days in isolation staring into a computer monitor

cloistered away in a windowless basement or the common belief that the content of the discipline of CS is something foreign and inaccessible to ordinary individuals are more than possible to achieve.

It is best to approach these myths early in a student's education. When students are young they are ready to learn and interested in new concepts. When one is interacting with youth and adolescents, it is easy to observe that they are like 'sponges' soaking up all of the information around them.

The critical period hypothesis claims that when children are young their brains are still actively tuned to learn and create synaptic connections. After this age, new concepts such as languages and concepts become much more difficult to absorb as the connections solidify [59]. Work relating to this hypothesis is mostly done looking at first and second language langauge acquisition.

In the current public school curriculum in British Columbia, CS is not considered a teachable subject for teachers. A secondary school teacher must have two 'teachables' to teach in the public school system this means that they have a sufficient number of undergraduate credits from these disciplines. Additionally, computer science does not have its own provincial exam[1] .

Currently students are not introduced to CS as a subject until their first year of university where students are mostly between the ages of 17 and 19, long after the 'critical developmental' years.

Problem solving in first year computer science has proven to be difficult to teach. This especially applies to recursion. It seems as though when students are given recursive functions in mathematics class, students are able to feel comfortable and can swallow the material. When the same concepts are given in the programming environment of CS1[2], the students seem to become frustrated and confused [68].

In addition to the students' need to be able to problem solve effectively using various techniques—including recursion—concurrency has recently become a key concept in CS. There is a new trend in the design of microprocessors that requires students to be prepared to deal with parallelism at various in depth levels. The current curriculum is beginning to change to accommodate this need, but a unanimously agreed upon curriculum has not been fully addressed, nor has the question of when and how to introduce concurrency [30].

[1]The standardized exam for a student's final year of high school where most students' entrance marks for university are derived.

[2]CS1 is the first programming course that an undergraduate student takes.

Various techniques have been used to attempt to make the introduction to first year computer science as digestible as possible. One popular technique is the design of educational programming languages. Some of these languages have a simple syntax, some manipulate graphics and some have clip together puzzle pieces that create an interactive program. Although there are many educational programming languages there has been some debate on how effective they are at helping students transition to a traditional text based language like Java. Various case studies explore which concepts can be demonstrated by first year students (and younger) using these programming languages. Concepts include looping and branching, variables functions and methods. Many of these studies also highlight the impressive graphics that students are able to create using only a few 'blocks' of code [23]. After the students have finished their first programming class in Alice they must inevitably learn a traditional language that will prepare them for industry or research. Powers et al. found that though their students were demonstrating understanding of the traditional control structures while using Alice, when they started using Java, the instructors saw no difference in understanding between the students who had taken a CS0[3] class in Alice and those who had not [61]. They also found that when students see a traditional programming language for the first time—and its minimal text output—they are disappointed with their limited visual capabilities.

Alice 3 will have an integrated 'see java code option' [2]. From this the students will be able to flip back and forth between the Alice representation and the Java representation. Students will be able to edit in both views, but once the code is edited in Java, it may or may not display properly in the Alice view. There are a few foreseeable problems with this design. The first is that the students are unable to look at both the visual block representation and the Java text representation simultaneously. This is a problem because it means that students must remember what their code looks like in the Alice block when exploring the Java code. This will make it very hard to map the program one to one. Secondly, Alice has built in concurrency with the *doTogether* block which does not have a simple conversion to Java. Some sort of an exception will need to be made for this special case.

The STEM[4] disciplines have had a significant decrease in enrollment over the past decade [35]. There are many contributing factors to this decline. One is the lack

[3]CS0 is a non-required pre-programming class.

[4]Science Technology Engineering and Math.

of resources in high schools. This includes lack of software, hardware and qualified instructors [56].

Beyond these issues, once students do choose to take their first programming course, we are seeing both high course drop rates and bimodal distribution. At the University of Victoria in the previous installment of CSC110 (Our cs1 computer science class), 150 students were registered in the class at its pique. At the time of the final, there were only 115 students (23% drop rate) still registered at the by the course and of those 25 did not write the final exam. So in the end, there were 90 students left out of 150, a 60% completion rate.

Another problem is that a bimodal distribution is often common in first year CS classes [54]. The bimodal distribution is when the distribution of classroom final grades have two distinct piques in their curve. One of these peaks is at the left side of the scale and represents the large portion of students who have limited understanding of the material, and the other peak represents the portion of students who have extensive understanding and sits on the right hand side of the scale.

Math, physics, science, chemistry and many other first year classes are all a part of the K-12 curriculum, so students have been introduced to these topics before arriving in first year. CS is not introduced in most high schools and this lack of introduction makes students are unaware of what computer scientists do and their impact on society. This also makes the students weary of the concepts when they are first introduced to them when they are past the age of 18.

When looking at the cohort of students in computer science, the population is very homogeneous. In the United States, only 17 percent of undergraduate computer-science degrees were awarded to women in 2004 [74].

1.2 Literature Survey

The following Section covers some of the work previously done in the field of computer science. Section 1.2.1 describes the progression of educational programming languages over the last 40 years and Section 1.2.2 describes some learning theories that align with this work.

1.2.1 History of Educational Programming Languages

It is not uncommon to see elementary school students take to programming like ducks to water. New educational programming environments are springing up every year. Designed to pique interest and teach the basics in the software field simultaneously, they are arguably being adopted at an impressive rate. The members of the Lifelong Kindergarten Group from the MIT Media Lab have been working with non textual programming representations such as Scratch, Pico Crickets, Hook Ups, Systems Thinking Blocks and other physical and graphical languages [65].

Logo - Seymour Papert [27] Logo was one of the first educational programming languages designed for kids. In the beginning it was a triangle on a screen called a turtle. the turtle could be given commands on how it moved about the screen. The turtle also held a pen that could either be down (drawing) or up, not drawing, and the pen could be multiple colours. The syntax for this language is very reduced and simple so kids can make easy functions with simple parameters to manipulate.

Microworlds EX [9] MicroWorlds EX is based on Seymour Papert's Logo, but is different because it allows for multiple turtles to be on the screen at once working concurrently. MultiLogo also has many different function calls that are designed to manage the control flow of concurrent tasks. It is a text-based language with a simple syntax and 'kid friendly' error messages. The program is compiled at runtime, but the students are also given the opportunity to work with a console to execute single tasks. The students are able to draw and easily manipulate the artwork for their turtles (characters) and backgrounds.

MultiLogo - Mitch Resnick [64] MultiLogo was one of the first 'concurrent educational programming languages' designed by Mitch Resnick, the founder of the Life Long Kindergarten (LLK) Group in the late 1980's. MultiLogo is an extension of Logo, it uses the same basic ideas (turtles, graphics and movement), and is also extendible to LEGO/Logo which controls Lego robots similar to the modern day Lego Mindstorms explained below. MultiLogo differs from Logo because of its capability to run multiple turtles/agents simultaneously. This program is introduces the user to many of the error prone situations that can occur in concurrent programming.

***Logo/StarLogo** [12] From these experiments with MultiLogo, Mitch created a product called *Logo [64]. *Logo had the concurrent capabilities of multi logo but is designed to make simulations involving hundreds or thousands of concurrent processes. From *Logo, StarLogo TNG was born.

StarLogo TNG is a 3 dimensional version of StarLogo. Its gives the appearance of concurrency allowing the user to control multiple types of turtles and multiple instances of each turtle. This program has many of the non-deterministic features of MultiLogo. For example, if you ask two turtles to make a different sound each at the same time, the user will hear the sounds simultaneously, but if you ask the turtles to change a variable simultaneously (e.g. one to a 2, one to a 3), then the value of the variable after the turtles have both run is non-deterministic.

Mindstorms [8] Another Life Long Kindergarten invention, Lego Mindstorms are Lego Robots with wheels, buttons, levers and other 'robotic' components. These components are snapped together like normal Lego and are programmed with many different languages. Some of these languages are text based (usually derivatives of common programming languages like Python or C), and some are graphical drag and drop. The most common language used is the proprietary language that is included with the kit. In this language users are able to use looping and branching, make function calls and have recursion and have multiple tracks running simultaneously. After the program is written the students download their code onto the robot via USB or infrared and the robot then executes the code. Syntax errors do not exist in this language, so the students simply must debug runtime errors.

Scratch [11] Arguably the Life Long Kindergarten group's greatest accomplishment has been their graphical block-centric programming language Scratch. Their tag line for this language is *low floor, high ceiling*. The students are able to drag and drop graphical puzzle pieces that animate to clip together. These puzzle pieces manipulate the characters on the screen called *sprites*. This language is event driven and gives the illusion of concurrency. The students are able to tell multiple instructions to execute upon the firing of one event.

Alice [58] Alice is designed by Randy Pauche's group at Carnegie Melon University. It is another graphical language that is designed to teach object oriented

programming. This programming environment allows its users to manipulate 3 dimensional graphics on a screen. It is also highly event driven. The users see looping and branching or they can use recursion or program concurrently using *Do Together* commands. The goal for Alice's approach is to eliminate syntax errors. The developers of Alice say that syntax errors (semi colons, curly braces and spelling) are the major hurdle for first year students [24]. Another design motivation was that students require meaningful context in programming. This adheres to the notion that students are no longer impressed by the textual "hello world! ."

This language is designed to be a stepping stone into Java and is frequently used in CS0 courses to introduce non-majors to Computer Science.

Pico Crickets[22] Yet another invention from the LLK group. This language is designed to make programs for students to load onto a 'Pico Cricket.' Pico Cricket's use sensors, buttons and motors to interact with the outside world. The goal of this project is to give students the ability to be very inventive and create robots, arts or any kind of craft that they can imagine.

This language is unique because is has the ability to go back and forth between graphical puzzle blocks and a logo based text language. The language on the puzzle piece blocks is very similar to the text language that is used so it is an easy transition between the two. The students are allowed to make their own functions and use them in multiple parts of the program.

Greenfoot[46] Greenfoot is a framework and environment that allows students to program in pure java, but create visual games. The Greenfoot framework provides an Actor class and a World class. The students then extend these classes to make their own Actors and Worlds. The worlds are divided by X and Y coordinated cells and has a background image that is easily set using the GUI environment. The Actor class has various methods that allows the actor to rotate, move and change images.

Greenfoot is also frequently used in first year CS classes. The transition from Greenfoot to Java is very easy because the students are programming in pure Java within the Greenfoot environment. Greenfoot is specifically designed for easy transition into Netbeans.

At the moment, all of the programming languages have their limitations with regards to the constructs and concepts that they support. They also all have their advantages and drawbacks as programming languages. Some of them are very unstable but allow the user to perform complex tasks. Others are very stable and never crash, but there are boundaries as to the depth and complexity of the students' code. So as can be seen, current flavours of educational programming languages fall short and require some support in the form of kinesthetic learning activities to be useful as educational tools.

1.2.2 Learning Background

This section explores various background in learning and teaching research. Many theories exist as to what might be the best way of learning and how information is best absorbed. Along with much of this work in how people learn, the different types of evaluation and analysis have also been explored over the years.

Learning Theories

Mayer and Monroe have outlined many important aspects of multimedia learning [53]. They show that it is better to represent a topic using at two modes (eg., visual and textual) rather than one. In their study on multiple representation theory they consider two representation systems, verbal (written and spoken) and visual (colours and shapes). Students who used both of these models were able to select information from both, organize them effectively, and integrate between the two. When students were given both verbal and visual information, they naturally do these tasks and undergo what is known as the multimedia effect.

Constructivist theory [60][5] explains that when students create their own knowledge, rather than being lectured or explicitly taught, they retain the information and gain a deeper understanding. This means that exploratory projects with guidance are more effective than lectures. Constructivist theory also has synergy with Vygotsky's Sociocultural Theory [41]. This theory states that students learn better when working with others. Students are able to discuss possible options or solutions, and with proposed solutions to a problem, they are then able to discuss what might be the best solutions and why. This allows students to articulate their thoughts and to

[5]Paiget and Papert worked very closely together.

Language	Textual/Graphical	Comments
Logo	Text	Simple text language, with a reduced syntax, to manipulate a 'turtle' on the screen.
MultiLogo	Text	Concurrent educational programming language. Designed to control either turtles or robots.
StarLogo	Graphical	Designed to control simulations of hundreds or thousands of agents on the screen. New versions are now 3 dimensional.
Mindstorms	Graphical	Intended to control a robot. Blocks that snap together. Includes: Looping, branching and event driven.
Scratch	Graphical	Controls 'sprites' on the screen. Concurrent, and event driven.
Alice	Graphical	3D characters act as objects. Very object oriented language. Allows for concurrency and recursion.
Pico Crickets	Both	A language that is used to manipulate sensors and motors. Can switch between graphical and a logo based text language with the click of a button
Greenfoot	Text	A framework that extends Java. Students program in pure Java to manipulate graphical characters

Table 1.1: Table summarizing Educational Programming Languages.

foster a deeper understanding of the task.

Unplugged Activities

The name "unplugged activities" stems from a set of learning activities developed by Mike Fellows and Tim Bell entitled Computer Science Unplugged (CS Unplugged) [15]. These activities are hands-on group problem solving activities that teach the basics of computer science without the direct use of technology. Popular exercises include those that teach sorting, searching, parody bits or binary numbers. All of these activities are readily available to both educators and parents and often come with simple print outs or ask the educator to use simple props that are easily found around the house.

Another appropriate name for these activities is kinesthetic learning exercises. These exercises are not only used for teaching concepts in technology and computer science; kinesthetic learning exercises are useful in teaching other applications as well.

Problem Based Learning

One current problem in the classroom is that students simply memorize the solutions and tasks presented to them. Modern students do not create their own solutions. When students create the solutions themselves they begin to own their answers and their knowledge. Once a student owns their knowledge, they gain the ability to apply it to real world variant problems [25]. Problem based learning (PBL) began through the need to create dynamic curriculum for first and second year medical students.

Learning requires three things: an essential body of knowledge, the ability to use the knowledge effectively and the ability to extend and improve the knowledge. PBL covers these three things because the students are required to take the body of knowledge and learn how to use and extend the knowledge in the classroom. Traditionally the last two aspects have not been touched.

The educator is required to engage the students by creating a desire for the students to know the solution. PBL approaches many of the aforementioned concerns and requests by asking the students to engage in problems and think through them from beginning to end [25]. When using PBL the problems should be very realistic for the students and instantly capture their attention. An interdisciplinary problem is favourable because it allows the students to see how aspects of their education are related. It also allows each student to approach the problem from their desired angle thus allowing the students to take control of their education. PBL should deal with

problems where there is not one correct solution. Students can explore the solution space and each individual or group will take a different path to arrive at distinct solutions. Not only should there be more than one solution, but beginning with the first problems that are introduced, the problems should be very large and ill formed so that the problem is more authentic [69]. Traditionally in education students are taught the skills and at the end of a semester they are finally given a problem where they are able to apply their knowledge. Instead, simple problems should be given throughout the semester so that students gain the skills through their application [67].

Qualitative Analysis

The definition of qualitative research has been debated and altered since its conception. Some definitions are very broad and some are very narrow [39]. Qualitative research is done with live people in real settings. This sort of analysis is done using *inductive data analysis*. A collection of data is acquired and then the researcher looks for patterns within his or her data.

Qualitative researchers become involved with the students and perform case studies to follow the patterns of individual students. Case studies monitor a group or an individual and report how they, him/her interact in the given situations. Qualitative research depends on interpretivism where aspects of the human environment are constructed by the individuals who participate in that environment. In interview format, which is common in qualitative analysis, an interview protocol should be developed. This protocol will make the various interviews adhere to a specific standard. The questions asked can then be compared qualitatively [33]. The goal of qualitative research is to increase the researcher's understanding of the phenomena being studied and guide futher research.

1.3 Related Research: Sophistication Early

What to teach an how to teach it is often debated in staff rooms and faculty meetings. In this section, two seemingly sophisticated topics are explored, recursion and concurrency. Each topic is defined and explained, and the related work in the field is described. Both of these topics and their educational practices have been researched in various high schools and universities across the globe.

Concept	**Summary**
Multimedia Learning	Best to represent a concept using two modes of sensing
Constructivist Theory	When students explore to create their own knowledge, they learn better than when they are explicitly taught.
Unplugged Activities and Kinesthetic Learning	Learning activities where students role play or problem solve without a computer in an interactive setting.
Problem Based Learning	In problem based learning, there isn't necessarily a right or a wrong answer. The students are to explore the problem space and find creative solutions.
Qualitative Analysis	Analysis of data that is not statistical. The analysis looks for trends in the (often human) data.

Table 1.2: Table summarizing learning background.

1.3.1 What is Recursion?

Recursion is a mathematical concept as well as a computer science programming construct that is well known in the field. It is a method commonly used to approach and solve numerous problems (e.g., searching and sorting tasks) and consists of defining functions where the function being defined is applied within its own definition. Because of its ubiquity and usefulness, recursion is usually taught in its most basic form in the first two years of undergraduate programming classes [6, 7].

To better explain the recursion paradigm, we can look at an example in programming. A recursive method consists of a *recursive call* and a *base case*. In a recursive call, the named method itself is called within the method. A base case is the point at which the program stops calling itself [63]. As a simple example of recursion written in the programming language Java consider:

```
public void hello(int i){
    if(i == 1)
        System.out.println(Hello + i + The End);
    else{
        System.out.println(still at + i);
        hello(i-1);
         }
    }
```

In this example, the recursive call is `hello(i-1)` and the base case is `if(i==1)`. The function outputs for input i set to 3 is:

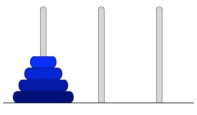
```
still at 3
still at 2
Hello 1 The End
```

Recursion can also be used as a problem solving strategy. For example, an optimal strategy for the puzzle *Towers of Hanoi*[6] [17] recursively steps through each level of complexity (i.e., the number of disks): For n disks, the problem is simplified by solving it for $n-1$ disks recursively. The base case occurs when one is simply solving a one disk Towers of Hanoi puzzle.

[6]This famous puzzle is depicted in Figure 1.1a and described in Activity 5.

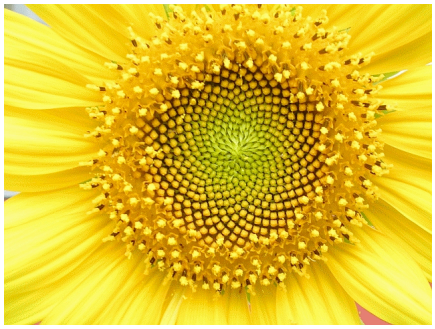
Recursion can also be effectively applied to computational problems such as searching and sorting by breaking the given data into smaller and smaller subsets (e.g., divide and conquer strategies). The base case in this situation is usually reached when there is only one element left in each subset.

Recursion can even be observed in the built and natural world [16]. Examples from the built world include virtual advertisements such as the Borax Soap Box [4]—a picture within a picture (a phenomenon referred to as the *Droste effect*). Other examples include fractals such as Koch's Snowflake [75] and Fibonacci numbers (see also Figure 1.3) [76]. Examples from the natural environment include trees (see e.g. [62]), sun flowers (see Figure 1.1c), pine cones (see Figure 1.2b), and sea shells (see Figure 1.2c). These common occurrences of recursion within the day to day experiences that many people have, adds to the appeal and application of the concept. One additional advantage of the recursion paradigm is that it can replace looping in the solution of some problems. Such a loop-free solution becomes, for many problems, much simpler to explain and describe. Also the number of lines needed for computer code when implemented is often reduced. This simplicity is illustrated in Figure 1.3. Here, two different methods are demonstrated; both calculate the i^{th}-Fibonacci number. Figure 1.3a is the iterative version containing a for-loop, and Figure 1.3b is the recursive (loop-free) version. There are two recursive calls in Figure 1.3b: both occur in the `else` statement: `fib(i-1) + fib(i-2)`. There are also two base cases, namely `i == 0` and `i == 1`.



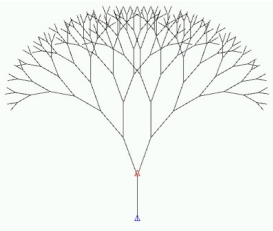
(a) The Puzzle Towers of Hanoi [71]

(b) Koch's Snowflake [1]



(c) Sunflower [50]

Figure 1.1: Some recursive pictures & puzzles.



(a) Recursive Structure of Trees

(b) Pine Cones [3]

(c) Sea Shell [51]

(d) Escher:Woodcut II, strip 3 [72]

Figure 1.2: Some other recursive figures and puzzles.

Teaching Recursion

Recursion is a difficult concept to teach to first year students. It has been hypthesised that teaching recursion is difficult because most of the time students are taught in naturally iterative languages [32]. This makes it very difficult to motivate many simple recursive functions where there is a much simpler version using a loop.

Most textbooks for teaching novice programming techniques discuss the classic examples for teaching and understanding recursion, the Fibonacci series, searching, sorting or Towers of Hanoi [77]. Michael Wirth from the University of Guelph suggests a novel problem for novices involving parking cars automatically. The technique of using non technical examples works because the students are able to relate it to their everyday lives. The teacher is not required to teach anything extra information about the problem. When teaching the Fibonacci numbers, often the teacher must tell the class what the numbers are before he or she can introduce the recursion in the problem.

When learning and working with recursion students are required to understand the control flow of the program as a whole rather than just the individual function.

The programming language Pascal has been used to teach recursion by using the language along with graphics programs to draw fractals [29]. Code is used to tell 'turtles' to draw different shapes on the screen. These shapes are drawn using recursion as it is one of the only control structures in Pascal. This allows the students to analyse the recursion at three different levels; the graphical level, the analytical level and the source code level.

Mental models of recursion have been outlined by Kahney and Levenick [43, 48]. These mental models represent both viable and non viable mental representations of recursion. They include the copies model, the looping model, the active model, the

```
int
fib(int i){
        int num1 = 0;
        int num2 = 1;
        int num3, j;
        for
        (j = 1; j < i;j++)
        {
                num3 = num1
                    + num2;

                num1 = num2;
                num2 = num3;

        }
        return num1;
}
```

(a) Iterative

```
int
fib(int i){
        if(i == 1)
                return 0;
        if(i == 2)
                return 1;

        return fib(i-1) +
                    fib(i-2);
}
```

.

(b) Recursive

Figure 1.3: Two ways to compute the i^{th}-Fibonacci number, programmed in C.

Mental Model	Viable?	Description
Copies	Always	The active flow of recursion occurs then the passive flow is followed once the base case is reached
Looping	Viable when the solution can be evaluated at the base case	Recursion seen as a form of iteration(Tail Recursion).
Active	Sometimes	Demonstrates the active flow but not the passive flow.
Step	Non-Viable	No understanding of recursive call. Demonstration of the recursive condition once, or recursive condition once then base case.
Return Value	Non-Viable	An understanding that recursion is the combination of all of the return values that are stored after each call.

Table 1.3: Description of the mental models of recursion.

step model, the return value model, the magic model, the algebraic model and the odd model (see Table 1.3).

Experiments exploring children's mental models of recursion using the Logo programming language were performed by Kurland and Pea in the 80s [47]. Their research involved giving 'expert' Logo programmers between the ages of 10 and 12 pieces of Logo code, and asking the students to describe the expected result. They found that students often model recursion as looping. They create this model when taught tail recursion as the first type of recursion, then maintain this model even though it fails for the more complex versions (head and divide and conquer). This reflects Kahney's looping model.

1.3.2 What is Concurrency?

As defined by an upper level Unix programming textbook:

> *Concurrency is the sharing of resources in the same time frame. When two programs execute on the same system so that their execution is interleaved in time, they share processor resources. Programs can also share data, code and devices...Concurrent and asynchronous operations share the same problems—they cause bugs that are often hard to reproduce and create unexpected side effects.*[66]

Concurrency can happen when there are multiple threads running on one processor, multiple processors, or multiple threads running on multiple processors. The ubiquity of concurrency has been exacerbated by distributed computing with thousands of computers attempting to coordinate and communicate while accessing shared resources and data.

Many severe problems can unexpectedly occur when working with concurrent programs. Examples include: race conditions, deadlock and non-determinism.

Race conditions happen when the order in which events occur affects the final outcome of the program. A simple example is one thread adding a number to a variable, and another thread multiplying the variable. If X is 10, and 2 first added to X then it is multiplied by 12, the result is 144. If the multiplication occurs first, the answer is 122.

Structures called semaphores or locks are designed to protect data from being changed while someone else is reading/writing it. Examples might include a variable such as a bank balance or a password. If these locks are programmed incorrectly, then the program can reach a state called deadlock. Deadlock occurs when two processes are waiting for each other to release (unlock) a chunk of data, and neither of them are able to proceed until the other unlocks the resource.

Race conditions can often lead to non-determinism. Non-determinism is when you cannot predict for sure what a program will do with a given set of inputs. Figure 1.4a gives an excellent example of this. This piece of StarLogo code has two turtles [7]—Chubaka and Turtles—running concurrently. They both change a variable at seemingly the same time. The monitor function displays the value of the variable at all times. When this program runs, each thread should run once. This code is non-deterministic (and a race condition) because you can never predict the final outcome of the program.

[7] StarLogo's version of threads.

(a) Variable (b) Sound

Figure 1.4: StarLogo TNG code performing concurrent tasks.

Teaching Concurrency

Some of the most ground breaking work in the arena of computer science education and concurrency was done by the founder of the Lifelong Kindergarten Group Mitchel Resnick[65] at the MIT Media lab. In the early 80s he created a programming language called Multilogo. Multilogo was the first implementation of Logo that was able to have multiple turtles running together [64].

Dr. Resnick did one-on-one testing with a group of eight children. The children were given different concurrent tasks to implement in Multilogo. Tasks included making a car drive while flashing a light or simply making the car turn by having the wheels do two different tasks. When the students were stuck, Resnick would work through related problem solving activities that highlighted aspects of concurrency with out the use of a computer. Example activities are: two (or more) people making lemonade, sweeping the floor, or washing dishes. He observed, "pressing students to think about real life models proved to be a good way to nudge them towards a parallel programming style" [64].

This study revealed three common programming errors among the participants. The first was problem-decomposition bugs. This was when students seemed to be were 'stuck' in the sequential paradigm and could not decompose the problem into concurrent tasks. The second was after the student successfully identified the different tasks for each turtle, but unintended sequentially or concurrency occurred. The final error that Dr. Resnick identified was Multilogo specific and is not applicable to other programming languages.

Kolikant et al. found through work with high school students that a common tendency exists for students to confuse the problem of synchronization and concurrency with the actual mechanisms for the solution [44, 45]. They suggest students should be very aware of the problems associated with concurrency before they are introduced to possible solutions. Additionally, the students with whom they were working were not university students, but students in their last year of high school. They were doing a full course on concurrent distributed computing. Though the students were not beginner programmers, they were far from expert. Their study followed three iterations of a course on concurrent and distributed computing. They ask students to solve problems using pseudo-code, natural language and concurrent Pascal. They found that students create two varieties of solutions, *centralized* and *decentralized* solutions. Centralized solutions are solutions that "force specific interleavings". Centralized solutions come in one of three flavours: 1) having a manger that communicates to organize interleavings, 2) assuming an execution order and 3) creating private resources. Decentralized solutions classify solutions whose entities are all equal. No one item has control over any of the others and the entities communicate effectively amongst themselves. This solution can be further decomposed into implicit (where the student does not discuss communication) and explicit (where communication paths are clear) communication. Kolikant's paper concludes that "teaching concurrency to novices must be done using authentic problems from the world of computerized systems, rather than the clever stories" citeKolikant:2001.

The cinema theatre experiment, from Kolikant et al.'s study listed above, was then modified and recreated by Lewandowski et al. [49] using first year CS students in their beginning week of CS1. In this study they used a ticket selling scenario to determine whether students could recognize issues with race conditions contention and non-determinism. They highlight many themes and errors that are present in their students' qualitative responses. They also support Ben-David Kolikant findings that dealing correctly and efficiently with concurrency is not completely innate knowledge, but problem identification can be of great use when introducing problems to beginner parallel programmers.

Ernst and Stevenson from the University of Wisconsin have been working on introducing parallel concepts into CS1. They highlight various steps in the learning process that will help the students excel once‘ they begin to program using complex algorithms and techniques. The first is decomposition involved, "searching for the parallelism in a program and breaking it down into tasks" [70]. The next assignment

asks students to explore using threads with little exposure to mechanism. Finally, orchestration is required when students begin to work with dependencies between the tasks and shared data.

The authors found that students begin to think about all of their programs with parallelism in mind at a very sophisticated level relatively early in their education. Many of their projects hinge on using Java and implementing many of the concepts in concurrency they have discovered; however their method of partitioning an approach to concurrency and parallelism is similar to the original Resnick study with children cited above.

Herb Sutter wrote a recent article on the "evils" of sharing [28]. In this article, he gracefully points out many real life scenarios that were inherently concurrent and involved resource sharing and contention. Examples include drawing with crayons and adding cream to ones coffee at a busy Starbucks. We are able to deal with these issues in every day life, but when forced to reason about similar issues in programming and software, we tend to fall back on previous tried and true software solutions that heavily introduce sequential solutions that ultimately slow down the overall process.

Phase	Recursion	Concurrency I	Concurrency II
Type of Activity: Number in Set	Kinesthetic : 6 Programming: 3	Kinesthetic: 4	Kinesthetic: 4
Evaluation	Multiple Choice (clicker) Written Responses Audio Video	Multiple Choice (written)	Multiple Choice (written) Written Responses Audio Video and Interviews
Students	9 students Grade 6 though 8	37 students Grade 7	19 students Grade 7
Interaction	1.5 hours per week for 7 weeks.	2 hours of interaction. Teacher gave subsequent questionnaires.	2 hours of interaction. Six, 10 minute interviews.

Table 1.4: Table outlining the three phases of the experimental study.

1.4 Proposed Solution & Thesis Direction

From the evidence described so far, one can infer that there are many issues that students must tackle in early years of Computer Science. Secondly, even though

there are many difficult topics, there is a lack of effective methods to teach Computer Science to the current generation of students. The most effective methods include unplugged activities that the CSUnplugged group have designed and the work where students need to relate programming to the outside world (Mitch and the Lemonade).

This work involves designing and creating activities to teach grade 7 students difficult key concepts in CS. Specifically, this work looks for methods to teach concurrency and recursion using interactive games that allow students to problem solve as teams and develop an understanding for key CS concepts. The activities that are created are evaluated using questionnaires, audio and video data.

The activities will be easily available online for teachers and educators to use in their classrooms with limited resources and minimal preparation and be integrated into current curriculum. Furthermore, these activities will be accompanied by a grading scheme and rubric to allow for teachers to easily asses their students' understanding of the material.

Through the activities outlined in this thesis, the students become engaged in computer science activities and true computer science concepts (recursion and concurrency). The improvements between the three phases of the experiments are a result of the shortcomings and the successes of each of the previous phases.

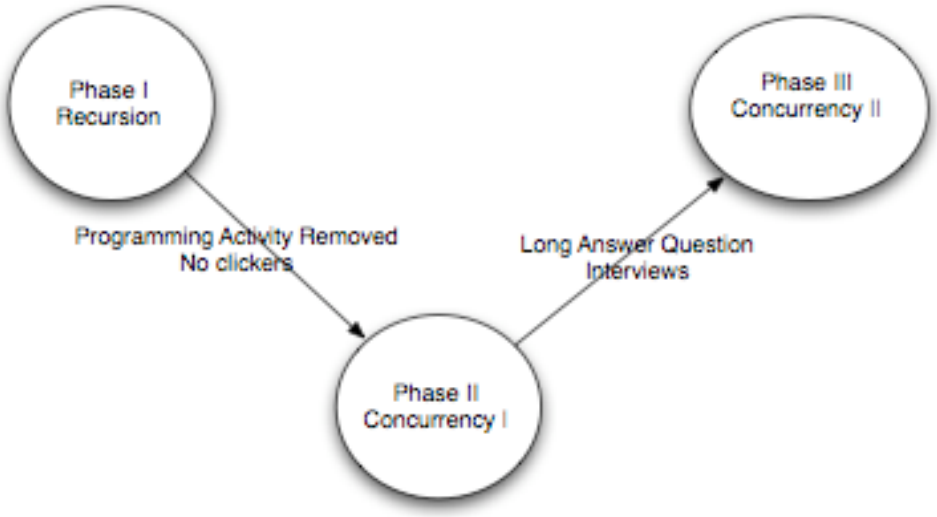


Figure 1.5: Movement between phases.

In the first experiment, we used clickers for the multiple choice questions. When looking at the raw data from the multiple choice questions between the first through the seventh week, it seems as though the students' understanding decreased. After

watching the video data that corresponds with the clicker data, it became obvious that the students were 'horsing around' with the medium. The students were playing to see how skewed they could make the results. Because of these findings, when planning and executing the second phase, we decided to use multiple choice questionnaires in paper format. For this phase we had only multiple choice questions for simplicity and analysis of the data. We allowed for the students to write comments in the margins, but we didn't require them to respond to any long answers. Here again, the difference between what was portrayed through the student's multiple choice questions and their verbal communication with the facilitators was apparent.

In the final stage of the experiments we changed our evaluation methods significantly. We had both multiple choice and long answer questionnaires. The whole classroom was audio and video recorded and we held one on one interviews. The one on one interviews and the written questionnaires allowed us to see the students' ability to understand the knowledge themselves instead of just recognizing scenarios (what we got from multiple choice). These forms of evaluation also allowed us to see variations between multiple students' understanding.

This thesis outlines the procedures and evaluation methods designed and executed throughout this research. Throughout the three phases effective methods for performing computer science activities with middle school children were developed and improved upon with the goal of introducing this group to computer science with enthusiasm and relevant information.

Chapter 2

Case Study 1: Recursion

This section will outline recursion as a sophisticated topic and the challenges that are faced when teaching recursion as well as introduce the age old debate of "when to teach recursion". We include a 7 week long case study that was developed as an afterschool club from grade 6 through 8 students [37].

2.1 The Age Old Debate

When and how to teach recursion has long been a topic of debate within computer science as recursion is both central in the discipline and thought to be difficult to learn and comprehend [38]. The argument of when to teach recursion is further complicated by debates amongst advocates for 'looping first' and advocates for 'recursion first.' Functional languages use a specific type of recursion to perform iteration (*tail recursion*); in these languages recursion is taught before looping out of necessity. Although Java is not a functional language, in a paper by Turbak et al., they describe a successful course taught using Java using a 'recursion first' approach [73].

2.2 Teaching Recursion to Kids

We limit our work to three basic types of recursion: head recursion, tail recursion and divide and conquer. In head recursion the recursive call is placed before the other tasks in the method. By doing so, the versions of the method on the stack will execute in reverse order. In tail recursion the recursive call is the last call in the method. The third method, divide and conquer, an algorithm design paradigm, is one of the most

successful of these techniques. Divide and conquer is done on an input set that can be divided, such as an array of words or integers. Here, the input is divided into smaller parts (often two), and then the method is recursively applied to each part. Analogously to head and tail recursion, the main tasks, which also merge the solution for the different parts together to one solution for the given input, can be done either before (e.g., quicksort) or after the placement of recursive calls (e.g., mergesort) [36]. The method is first performed on the set as a whole, and then performed on the two sets, which divides these two sets again. This pattern continues until the base case is reached, at which point the sets are merged (glued back together).

2.3 Kinesthetic Learning Activities, Games and Toys

As mentioned previously, two important parts of using recursion are the base case and recursive call. Thus, we decided these two criteria should be the focus of our lessons. To design activities that teach recursion to our group of youth in grades 6–8 (approximately age 11 to 13) we investigated a combination of kinesthetic learning ("unplugged") and programming activities. We developed a number of kinesthetic activities to teach the mental models needed. We built on some familiar activities such as *Towers of Hanoi* [17] and the *Borax Box* [4] and then extended these activities to map to an educational programming language.

Activity 1: Genome Sorting
The students' objective is to sort a number of objects in the comparison based sorting model [36].

Students are divided into groups of two to four. Each group is given a set of coloured envelopes, with folded pieces of paper inside numbered from $1 - 14$. Each envelope represents a piece of an organism's DNA sequence, and each colour represents one of the four nucleotides (Adenine, Thymine, Guanine, Cytosine). The students are instructed to sort these envelops into a correct order without looking inside of them. One facilitator is identified as the "comparator" and students show two envelopes at a time to this comparator to determine (comparing the numbers) which envelope comes first in order. Each comparison costs one CPSC Buck (short for ComPuter SCience Buck). The students' goal is to determine the genome sequence

with as few CPSC Bucks spent as possible.

Activity 2: Visual Recursion

Visual recursion is an image that contains itself. Two excellent and common examples are the Borax Box [4] and the Droste Box (See Figures 2.3a and 2.3b). Other good examples include the Mandelbrot set [16] and Koch's Snowflake [75] (See Figure 1.1b). To deepen the student's knowledge, a teacher can take a more tactile approach by turning a video camera toward a screen on which the output is being projected.

In this activity, the students are shown some examples of recursive pictures and the aspects of the pictures that make them recursive are highlighted. After this discussion, the students are given a collection of pictures, some recursive and some simply patterned. Students are asked to mark on the back of the images whether they do or do not think the image has the recursive property. After the students have finished, the students' answers are tallied and the pictures (and perceptions) are discussed as a group.

Activity 3: Sierpinski's Carpet

Sierpinski's Carpet [26], is a recursive image. The image includes nine smaller instances of the carpet inside the big one, and each of those again contains nine smaller carpets etc. A picture of the carpet is more difficult to describe without using recursion. In the recursive description, one simply indicates "draw a square, then divide it into nine squares, then colour in the centre square. Do the same thing to all the little squares".

In this activity, students are divided into pairs. One student in each pair is the 'drawer' and the other one is the 'describer'. The drawer is given a sheet of paper and a pen or pencil. The describer is given a picture of Sierpinski's carpet; the describer's job is to verbally tell the drawer how to draw Sierpinski's carpet.

After the students have completed the task, the group comes together and discusses solutions to the problem.

Activity 4: Line-Up

In a grocery store, cafeteria or movie theatre line-up (queue), the number of people in the line can become quite large. The person at the end of the line wants to know how many people are in front of them without leaving the line and losing their spot.

The objective is to come up with a solution where everybody is allowed to only talk to the individuals within immediate proximity.

An efficient solution for any person to figure out their position in the line is to ask the person in front of them their position. This question continues recursively towards the front of the line until someone discovers that there is no one in front of them. This person then

knows that they are in front and can respond to the person behind them "I am number 1." The second person in line is then able to add one to this answer and respond to the next person. The message is propagated until the initiator of the question receives a response.

In this activity all the students line up in a row, as though they were standing in line at a grocery store. Because there is often a small number of students, making it too easy to see ahead of you, all of the students are blindfolded and shuffled in a line. They then need to create an algorithm to discover the number of people that are in front of them without leaving the line-up and only whispering to their neighbours.

Activity 5: Towers of Hanoi

Towers of Hanoi is an ancient Vietnamese puzzle involving three pegs and N disks of different sizes [17]. All of the disks start piled on the one peg (*source*) with the largest on the bottom, and all of the other smaller ones piled incrementally on top. The goal of the game is to move the disks from the source to one of the other pegs (*destination*) while following these rules: (1) Only one disk can be moved at a time; (2) at no time can a larger disk be placed on top of a smaller disk; and (3) the number of disk moves should be as small as possible.

The solution to this problem is recursive. To solve a Towers of Hanoi puzzle with 5 disks, you must move the four disks on top to the third *auxiliary* peg, then move the largest disk to the destination, and finally move the four smaller disks on top of the big one from the auxiliary peg to the destination.

The same technique (recursive call) is employed to move the four smaller disks from the source to the auxiliary peg and later to move the four smaller disks from the auxiliary peg to the destination. This pattern continues until only the smallest disk needs to be moved (base case).

In this activity, students are given the opportunity to play Towers of Hanoi with different numbers of disks and to problem solve as a group or as individually. After-

Activity	Description	Concepts
Genome	A comparison based sorting activity. Students must blindly sort numbers.	Application of recursion in problem solving.
Visual	Interact with images that are either recursive or non recursive and identify the differences.	Identification of the recursive call.
Sierpinski	One student must describe to the other how to draw Sierpinski's carpet	Identifying and applying the base case and recursive call.
Line Up	Students line up blindfolded. The last person must find out how many people are in front of them in the queue. Everyone must only talk to their direct neighbours.	Base case and recursive call.
Towers	Play with physical puzzles and create an algorithm.	Problem solving with recursion. Application of the recursive call and base case
Match-Box	Comparison based sort. Numbers inside of match boxes.	Application of recursion in problem solving

Table 2.1: Synopsis of unplugged activities and their related concepts.

wards the students make their solutions concrete with verbal or written responses.

Activity 6: Match-Box Sorting

Match-box sorting is, like genome sorting (Activity 1), a sorting activity using the comparison based sorting model, so they share many of the same instructions. Students are given varying numbers of match boxes with the matches removed and numbers written on the inside of the boxes. Students are not allowed to open the match boxes. An instructor acts as a 'scale' and the students are able to give two boxes at a time to the instructor. The instructor tells the students which box contains the number that is higher, lower, or neither (if both contain the same number). Students must then sort the boxes while keeping track of the number of comparisons required.

2.4 Educational Programming Languages and Activities

There are various programming languages that were developed with children in mind. Many of them have either a simplified syntax, or use drag and drop puzzle pieces that clip together to create code. They focus on animation and graphics to appeal to a younger audience. Two popular educational programming languages that support recursion are Alice and MicroWorlds EX [9][1].

Alice

Alice is a graphical programming language that was developed at Carnegie Mellon University [57]. It has a drag and drop interface that is supposed to replicate syntax similar to Java's. The code allows students to animate 3D characters in a scene or to create 3D games and scenarios. When a student makes a new method, a new block with the name of the method is created in a side panel. Alice allows for these functions to be called recursively. When a student drags a function into its own definition—creating a recursive call—a pop-up window appears:

> The code you just dropped in creates a 'recursive method call'.
> We recommend that you understand what recursion is before making a call like this. Are you sure you want to do this?
> **Yes** I understand what I am doing.
> **No** I made this call accidentally.

After a sufficient amount of work, Alice's graphics look very professional and have a 'finished' look; however, the complex graphics can be intimidating and difficult to manipulate for beginners. When we were testing to see which language we would use, often the recursion would become overbearing for the Alice system and would even crash the program.

Logo and MicroWorlds EX

[1] There are many other languages, some support recursion, others do not. Alice and MicroWorlds EX are discussed here simply because they are the languages that the authors had the most experience with.

Logo is a programming language that was developed in the 60s [27]. It controls a turtle on the screen. The turtle goes forward, backward, left and right and is able to hold a pen of varying colours. There are multiple commands that manipulate these settings. Logo's syntax is simplified, omitting semicolons and curly braces, enclosing functions between *To..* and *..End* and logical statements with square brackets. Logo supports looping, branching, functions and most importantly recursion.

MicroWorlds EX is a derivative of Logo with similar syntax and commands, but more interesting visuals with multiple turtles and a more intricate IDE [9] (Figure 2.1). There is a very explicit API and the students are able to make their own functions harnessing existing power within the language with a few simple calls. Students can also be creative by making their own characters and sounds. After some review, we chose MicroWorlds EX for our after school program because of its versatility and cleanliness. MicroWorlds EX also has a quick response time and does not slow the machine down. We were able to implement programs to teach the concepts without the overhead of dealing with 3D graphics. Because of this, students were free to be creative and draw their own characters. Although because we chose this platform, the students were required to overcome an initial learning curve caused by syntax and precision errors (e.g., whitespace, typos, consistent naming).

Programming Activity 1: Drawing Squares

Using Microworlds, the concept of a recursive call, is easily introduced to students using turtle art. When programming using recursion, the students must understand the concept of a function/method. Return value and parameters are also assets in furthering their understanding. To teach these concepts this using turtle art, the teachers can ask students to make a function that draws a square. A square is made by first telling the turtle to *put its pen down*, then telling the turtle to move forward, turn 90°, and repeat 3 more times. The addition of a simple parameter allows the students to change the size of the square by changing the amount that the turtle moves forward. The students were able to observe through experimentation the effects firstly of the addition of a recursive call (as well as what it looks like), and secondly of its different placements inside the function. Finally in this activity, the students add a base case to change how many squares are made, and what might happen when the base case is reached.

Figure 2.1 demonstrates a piece of code (top) that draws multiple squares using the square function (bottom).

Programming Activity 2: Animal Chase

To encourage students' use of recursion and base case we had them make an animation that involves two turtles on the screen, one chasing the other. The students are shown an example in which the animal that was chasing came in close proximity to the animal that was being chased. Eventually, the first animal would hug the second animal.

In this example the students saw a base case of *within a certain proximity*. When this was true, an interaction would occur between the animals. When the base case was not true, one animal turns towards the other and moves x steps.

Programming Activity 3: Towers of Hanoi

This programming activity is a follow up to the Towers of Hanoi unplugged activity (Activity 5 from above). After the group has developed an algorithm for solving the Towers of Hanoi problem, the students can use this algorithm can then to make their own Towers of Hanoi solver.

When using Microworlds EX, the students are given some 'helper functions'. The helper functions given are *move* and *which*. These two functions identify which peg and which disk the students are accessing. These functions are given so the students can explore the Towers of Hanoi Solving function without worrying about the graphics.

```
to sis :num
if :num > 0
[
sis :num - 10
square :num
wait 3
]
end


to square :num
repeat 4
[
fd :num
rt 90
]
end
```

Figure 2.1: A piece of recursive code that had a turtle drawing big squares on the screen.

Activity	Description	Concepts
Drawing Squares	Students program a turtle to draw squares (or any shape) recursively getting smaller or larger.	Exploration of recursive call and base case
Animal Chase	Create a scenario where one animal chases the other. When the predator touches the prey, the predator hugs the prey.	Application of the recursive call and base case
Towers of Hanoi	After the students create an algorithm in pseudo code, they program a solution to Towers of Hanoi.	Application of recursion and transfer from pseudo code.

Table 2.2: Synopsis of programming activities and their related concepts.

2.4.1 Procedures

In the Fall of 2008, we ran a weekly after-school club[2] on Monday evenings lasting 1.5 hours for students in grades 5 through 8 (age 11 to 13) over a seven week period. We advertised our program in the neighbouring schools to staff, faculty, and students in the department of Computer Science, looking for a maximum group size of 15. In the end there were 9 students, 7 girls and 2 boys. Not all of the students were present for all of the weeks, and some friends of the students participated for a single session. The program was divided into two sections each week. The first was entitled *Recursion Aversion*, where one or more of the activities listed above was executed, and the second half was reserved for programming with MicroWorlds EX.

Each week built upon topics that were covered in the previous weeks. Each of the weeks also attempted to explore at least one of these main questions:

1. Can the students identify recursion?
2. Can the students understand and apply recursion?
3. What is the effect on attitudes towards computer science?

Week 1

In the first week, the students completed the pre-test (Activity 1, Genome Sorting) so we could evaluate their current understanding of the material. We were interested

[2]SPARCS After-School Club at the University of Victoria.

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
	The students were given the pre-test activity.	Visual recursion was introduced.	Problem solving with Sierpinki and nested squares.	Stopping condition. Line up and animal chase.	Problem solving with Towers of Hanoi.	Free Time	Post-test sorting activity.
Recognize							
base case							
recursive call							
Understand							
base case							
recursive call							
Enjoyment							

Table 2.3: Explains the target questions and activities for each of the weeks.

in the strategies the students would use to solve this problem. This activity was designed to determine whether students understood and could apply recursion prior to having been taught the concept. We had four groups, two with three students and two with two and the activity was executed without any prior discussion. After the pre-test, the students became acquainted with MicroWorlds EX. The students were given the task to animate their name on the screen. This allowed them to learn how to write functions, use simple commands and draw their own figures.

Week 2

This week the concept of visual recursion was introduced. The intention was to have students understand the concept of a picture containing a picture of itself and learn to recognize this phenomenon when it was presented. We began by teaching the students to recognize recursion in images and then tested to see if they were able to identify it in other examples. We first showed them a few recursive images and a video of the Mandelbrot Set[3]. We then handed out pictures, some being examples of visual recursion, and others not. After the students voted, we discussed using these examples the requirements of a recursive image.

When using recursion as a programming technique, additional concepts that the students need to understand are procedures and parameters. Procedures are the basis of the recursive call and the base case is tested against the parameters. This week we taught the students what a procedure is in MicroWorlds EX by designing procedures to instruct the turtles to draw squares of different sizes. The students then created

[3]A part of the Mandelbrot picture is selected and zoomed in, and that process is repeated for the duration of the movie [13].

procedures to draw other shapes.

Week 3

Our goal this week was to teach problem solving using recursion and see whether the students were able to naturally decompose problems with this strategy. For this, we introduced the Sierpinski Carpet activity in pairs.

This week we introduced a programming activity, having being introduced to function calls and parameters the week before, we taught the students the concept of a recursive call. The students used their square drawing functions from the week before to draw nested squares.

Both of the activities from this week were designed to evaluate students' ability to recognize, understand and apply recursion. When the students were faced with describing an instance of the Sierpinski's Carpet[4], if it is understood successfully, they recognize the recursion and attempt to describe the image using a recursive technique. This would demonstrate their ability to recursion.

This week was mostly about understanding the recursive call, and the concept of an entity containing itself.

Week 4

In Week 4, the students took the recursive call one step further by adding a stopping condition (the base case). We introduced it by having them act out the line-up scenario, discovering the stopping condition of *no one in front of me*. The students then programmed this concept by having one turtlxe chase another turtle until it caught and hugged it. The function that they were to create stated that the chasing turtle would move towards the other turtle until it reached the base case of touching the turtle.

Here the students demonstrate their knowledge and understanding of base case integrated with recursive call. This lesson consisted of the students utilizing tools and code collaboratively written as a group.

[4]In the Sierpinki Carpet activity the students may face more difficulty in to identifying the Droste Effect than for the picture we chose in the Visual Recursion Activity (Activity 2), as the presentation of the carpet does not easily allow the identification of a carpet instance of a certain size due to the many same-size carpet instances next to each other.

Week 5

This week's goal was to give the students an interesting task encompassing the different topics introduced up to this point. We chose Towers of Hanoi (Activity 5) because it includes has good problem solving logic and includes many of the concepts already covered, the solution to this problem is interesting, hands on and includes a base case and two recursive calls. Multiple hands-on versions of the puzzle were present in the room. The students worked on solving the puzzle and working out strategies. Once the group had come up with a general algorithm, we transferred this to MicroWorlds EX and each of the students then programmed their own version of Towers of Hanoi (Figure. 2.2).

Week 6

In Week 6, the students were given free time to work on something that they had seen so far in the session. Many of them chose to continue playing with the Towers of Hanoi physical puzzle. Through the choices they made in this session, their enjoyment of the physical puzzles was evident.

Week 7

In the final week of the after-school club we administered the post-test: Activity 6 (match-box sorting). This test was designed to determine whether or not the students were able to apply recursion appropriately.

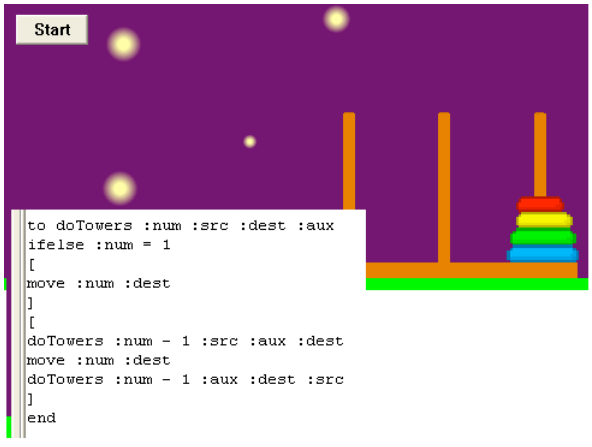


Figure 2.2: One student's screen for Towers of Hanoi.

(a) Droste Box [18]

(b) Borax Box [5]

(c) Escher: Hand With Reflecting Sphere

Figure 2.3: Images used for questions 1-3.

2.5 Evaluation

In efforts to triangulate our findings, we decided to use various methods of data collection to help us evaluate the students' understanding of recursion (i.e. video recording, audio recording, note taking). We also used the clicker system i-clicker^{TM} [5]. Clickers are remote controls with buttons A to E that facilitate data gathering for multiple choice questions. When a multiple choice question is projected overhead, the students choose their answer by pressing the corresponding button on their remote. A base station at the teachers desk is connected to a computer that records the students' answers. Each week we collected data using four to eight clicker questions about recursion, MicroWorlds and general concepts we taught thus far.

2.6 Clicker Data

Table B.1 (in Appendix B) lists the questions that were asked throughout the seven weeks of this study. Some questions were asked multiple times, and some were only asked once. Problems that asked the same question, but using different variables are grouped together (e.g. we asked *Is this code recursive?* for more than one method). In the tabulations, invalid answers (outliers) are categorized as incorrect.

Visual Recursion

[5] www.iclicker.com

The first three questions were surveyed in week two after the concept of visual recursion had been explored. The first three questions had accompanying images (see Figure 2.3). Question two was revisited in weeks 5 and 7 (see Figure 2.6)

Question 1	**Question 2**
Is the Droste Box Recursive?(Fig. 2.3a) *A*) Yes *B*) No	*What is Recursive about the Borax box?*(Fig. 2.3b) *A*) Horses *B*) Lettering *C*) Not recursive *D*) The Box

Question 3

Is the Hand With Reflecting Sphere Picture Recursive? (Fig. 2.3c)
A) Yes
B) No

Table 2.4: Multiple choice questions about visual recursion.

Coding questions

In the following questions the students were shown pieces of MicroWorlds code. They were then asked questions about the base case, the output or the functionality of the code. Some of the code was recursive, while other pieces were not recursive.

Question 4	**Question 5**
In one question, the students were presented code that has a turtle drawing squares recursively but lacks a base case. The students were then given the option of filling in the missing line with MicroWorlds code *A*) `repeat 5` *B*) `if (i < 5)` *C*) (the function call) `doSquares(5)`.	The students were shown a snip-it of code where the base case was `if(i<5)` and the four options *A*) 6 *B*) 8 *C*) 1 *D*) 5

2.6.1 Written Responses

Other forms of data collection used in this study was written responses and long answer questions. These responses are also organized in order of week of collection. Some weeks did not have any written responses; therefore, each week is not represented in this section. Table 2.6 outlines the questions the students were asked.

Question 6	**Question 7**	**Question 8**
In another question, a recursive function was placed on the overhead with a turtle drawing squares. The students were asked how many squares the turtle would draw. Further questions targeted the recursive call, in particular, the students were required to figure out the order the squares were drawn. For this the students were given code and 4 different images and they were required to choose which one was the proper output.	In the non recursive procedures, the students would see `repeat n` or `forever` loops. They were then given the binary choice Yes this is recursive or No it is not.	In the recursive procedures, the students would see procedures that were clearly recursive with a recursive call and a base case.

Table 2.5: Questions asking the students about pieces of code.

	Topic
Question 1	Is a certain Picture Recursive?
Question 2	From all of the pictures we just looked at, how would you describe recursion?
Question 3	Drawing the output of a function
Question 4	After Studying Recursion, what would you say it is? What is important about it?

Table 2.6: The written questions that were asked throughout the workshop.

Question 1

In the second Week we had the students vote on whether they thought certain images were recursive. This activity was very early in the week. Prior to this activity the students had seen only two other recursive pictures and heard the definition of *"An image that contains itself"*. The images were printed on paper and handed out to the group. The students indicated their responses anonymously on the back of the images. Table 2.7 shows the students perceptions of each of the images.

Image	Correct	Incorrect	N
1. Mandelbrot Set	3	2	5
2. Mona Lisa	6	0	6
3. Spoof	6	1	7
4. Circle Limit	0	5	5
5. Division of Plane	1	2	3
6. Koch's Snowflake	7	0	7

Table 2.7: Results from voting whether images were recursive or not.

Question 2

In the second week the students were asked *From all of the pictures we just looked at, how would you describe recursion?*. Typical answers were "a picture within a picture, forever", "When a whole picture is repeated inside itself" or "It is when a picture is repeated inside itself infinitely."

Question 3

In week 7 the students were given handouts with 3 different procedures written in MicroWorlds (see figure 2.4. The students were asked to draw the output from one of the three procedures and indicate on the handout which procedure they drew. The first procedure was recursive(Figure 2.4a), the second used a `repeat` loop (Figure 2.4b) and the final called `drawSquare :size` 5 times with the value of the parameter decreasing (Figure 2.4c). The first and second also had the function call `Squares 24`All of the procedures had the same output (5 squares getting smaller and smaller inside one another) and used a method called `drawSquare :Size` that would draw a square the size of *size*. 5 students handed in this activity (6 were present that day). Of the five students, four drew correct images representing the output. One student circled the recursive procedure, one student circled the `repeat` loop and the final two circled the procedure that simply had 5 method calls. The students had seen drawSquares before and were reminded of what this procedure accomplished.

Question 4

At the end of the 7 week session, we asked the students to answer the question *After studying recursion for the last few weeks, what would you say it is? What is important about it?* Many of their answers described that they felt that recursion simplified solutions: "It makes things easier" , " It's cool, it makes your brain work". They also indicated information about repetition, and discussed pictures: "For towers of Hanoi instead of just doing it randomly just keep doing the same thing.", "Recursion is basically a picture inside a picture....", "It is something done over and over getting smaller". None of the students defined recursion perfectly; however their answers allow us to see some basic understanding and what they took away from the experience. The expression 'picture within a picture' surfaced frequently, as well as repetition.

```
to Squares :size

ifelse ( :size < 3 )
[

    drawSquare :size
]
[
    drawSquare :size
    Squares ( :size / 2 )
]

end
```

(a) Recursive

```
to Squares :size

make ``howmany (:size / 6)

repeat :howmany
[
    drawSquare :size
    make ``size (:size / 2)
]

drawSquare 2

end
```

(b) Iterative

```
drawSquare 24

drawSquare 12

drawSquare 6

drawSquare 3

drawSquare 2      .


.
```

(c) Function Calls

Figure 2.4: Students were given these three chunks of code and asked to draw one of them.

2.7 Video Data and Observations

The video and audio recordings of all sessions in this study provided a rich and useful data source from which information was extracted and analysis was performed. According to the guidelines established by Barron and Engle [14] we allowed our three overarching questions (i.e., students recognizing, understanding and enjoying recursion) to guide the direction of data collection. Further to this, we chose a manifest content approach – where interactions among the students and instructor focusing on these specific questions were selected and examined – and focused on each session in sequential order according to what questions we believe offered the best evidence.

Week One

In our week one pre-activity (genome sorting), a number of groups became discouraged almost immediately claiming the task was impossible. After considering the task, most groups described a working algorithm (not necessarily the most efficient). None of their suggested strategies for solving this task was recursive.

Week Two

The students were very engaged with the visual recursion activity this week and were actively debating amongst one another about whether the pictures in the activity were recursive. The students successfully identified the recursion in the presented pictures with the Droste Effect[6], but became unsure when Koch's Snowflake and the Mandelbrot Set were presented because the recursion is not as obvious. After debating amongst themselves, the students could not come to an agreement[7]. When asked what aspect of the Borax Box was recursive, one student – who defended that the lettering was recursive – stated that *"If the box is recursive, then everything on the box is recursive."*

With regards to Escher's *Hand With Reflecting Sphere* students had mixed responses. The correct answer is that the image is non-recursive. One student felt that it was recursive because *"The ball is a reflection"*. An incorrect reason for that the image is non-recursive: *"[It] reoccurs like once, but doesn't contain the picture as a whole. It doesn't go over and over again."*. A final response to this discussion was

[6]Where an entire image is contained within the image like the Borax Box.

[7]We were told that after this week (it was just after Halloween), one of the student's told her mother that next year she'd like to make a "recurjik" pumpkin, with little pumpkins inside big pumpkins.

that *'"If it was recursive it would show the hand with the ball inside the ball".*

Week Three

The first half of the third week focused on The Sierpinski Carpet. In pairs, one student was required to provide enough verbal detail to the other student so that this second student could reproduce an image of The Sierpinski Carpet. The video and audio analysis for this section focused on the students' recognition and if students were able to see that The Sierpinski Carpet could be explained to their peer via recursion. We feel that the recursive solution is the simplest. When given The Sierpinski Carpet to draw none of the students described the picture using recursion. They all started from one corner and told their partners to draw a square grid and colour in certain squares black.

A follow up discussion—to consolidate learning among all the groups in the class—involved students identifying the strategies employed for The Sierpinski Carpet activity as well as connecting this activity to the previous week. As a group, students were able to explain to the instructor how The Sierpinski Carpet was recursive and subsequently directed the instructor —with some encouragement and direct questioning on the part of the instructor—how to recreate it in a recursive form (something that none of them had done initially). Examples of direction made by the students on how to draw The Sierpinski Carpet included: "draw a three by three grid" and "over and over again". They needed further guidance in recognizing what to do to stop the procedure (the base case) and then in recognizing that this represented a "procedure inside a procedure."

Although the students did not employ recursion to help their peers recreate The Sierpinski Carpet and later had difficulty recognizing the "procedure within a procedure" aspects of their group direction, there is evidence that re-engaging with The Sierpinski Carpet activity and connecting this learning to the previous weeks exercise helped them to recognize the recursive elements involved. They were quick to contribute and correct to identify that by breaking this down into a repetitive series of directions, the Sierpinski carpet could be replicated reliably. This moment of recognition—bookended by some challenges with full understanding—is further validated by observations made in subsequent sessions.

Week Four

The fourth week focused on a MicroWorlds exercise where the task was to let one character (say a person) chase another character (say a bird) until the first character is able to hug the second one. This task (from the view of the person) can be solved easily using tail recursion or using an iterative approach (e.g., a while loop or a repeat until loop).

Listing 2.1: Recursive Hug Solution

```
Algorithm AnimalChase(bird)

    if in huggable distance of the bird then
          hug
    else
        Move a few steps towards the bird.
        AnimalChase(bird)
```

Listing 2.2: Iterative Hug Solution

```
Algorithm AnimalChase

      while  (the person is) not in huggable distance of the bird

      do

           move a few steps towards the bird
           hug the bird.
      end
```

Listing 2.1 shows a possible approach using tail recursion while listing 2.2 shows a possible iterative approach. The task was first shown as an animation and then analyzed with the group. Elements on the video and audio analysis of this discussion focused on first understanding the task, then developing a strategy with focus on afterwards realizing the discussed algorithm in MicroWorlds. As mentioned above, this task may be solved recursively as well as iteratively. The instructor asked targeted questions to deepen the understanding of the tasks. The group identified with aiding questions the stopping condition or base case of while loop or recursion respectively, namely the person will hug the bird only when (s)he is very near the bird. Easier for

the group was to identify the repeated element, namely to move towards the bird if no hug is possible. When asked how to program the procedure, a participant suggests a loop and thus the iterative approach.

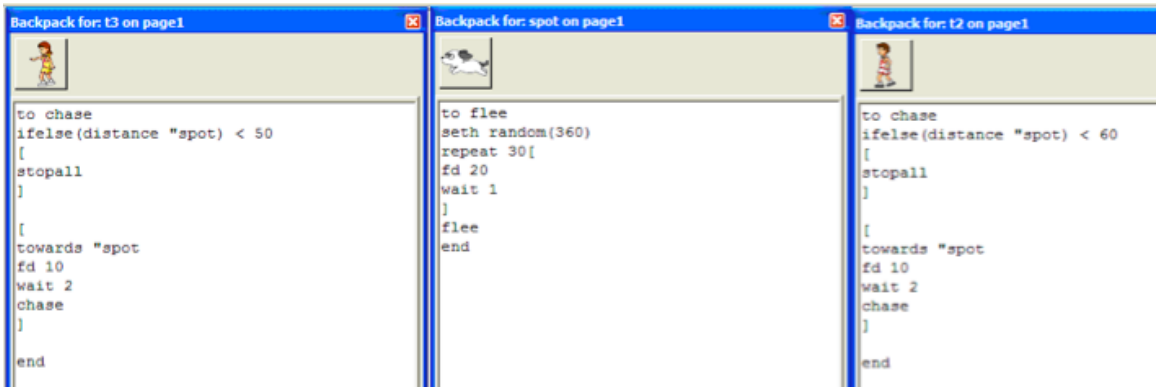


Figure 2.5: One student's Code for the Hug activity with three characters

The instructor next wrote a simple recursive procedure on the board. The students had seen this procedure the Week before (draw squares inside a square). The instructor also targets questions to draw parallels between the two tasks. The students do not bring up by themselves any discussion on the recursive element in either last Weeks procedure on the current one.

Although students did not actively develop a recursive strategy for the problem, the programming part show evidence that the students could apply recursion to realize the task as they all (with different amount of support) were able to write their recursive AnimalChase procedure.

Week Five

In Week 5, students were presented with hands-on towers of Hanoi puzzles. This activity was the most popular among the participants. The overall goal was to explore a strategy which lets you solve the puzzle for any number of disks. As this is best described recursively, the activity was thought to gain insight into the students' ability to apply recursion.

Initially, the students worked either individually or in pairs with a physical version of Towers of Hanoi. They were given the rules of the game and then allowed 30 minutes of free time to solve the puzzle. The facilitators went around the room helping the students verbalize their strategies. When the students were consolidated as a group, they discussed the number of moves that they achieved and what they had found

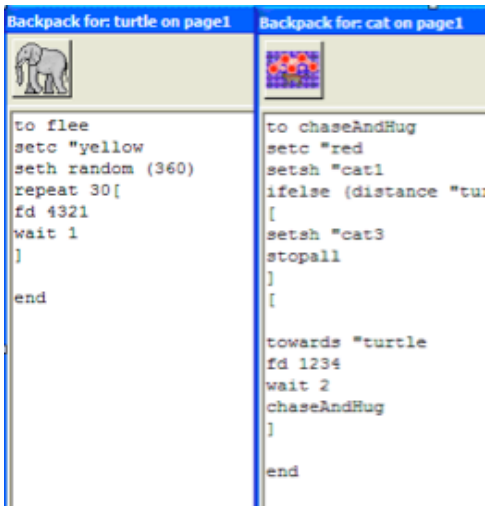


Figure 2.6: Code for the hug activity, Two Characters

to be their optimal solution. We observed that by the end of the 30 minutes, most of them had the correct mechanics when working with 4 disks that are the same as the ones when the problem is solved recursively. We remark that the whole group remained enthusiastic and on task for this entire 30 minutes.

In front of the class, one of the students drew a 6 step image on the board describing how she solved towers of Hanoi for six disks (See Figure 2.7). She had previously worked this solution out on paper. Using this student's diagram the instructor then scaffolded the experience and allowed them to demonstrate more than they could on their own. The students correctly produced the recursive calls and the base case. The students then used this pseudo code—and some given functions – to create their own Towers of Hanoi using Microworlds (Figure 2.2). At the end of the session with some help from the instructors, four students' code worked completely, one required some simple syntax corrections (white space and spelling) and two were not correct.

Week Seven

In the culminating Week of this series of activities, the students were presented with a matchbook sorting activity—based upon the same principals as the genome sorting activity from the first Week. This exercise was deliberately designed to gather evidence of both student recognition and understanding of recursion. It was anticipated that students would be able to draw upon their prior experience with recursion (i.e., through the genome sorting exercise among others) and generalize this under-

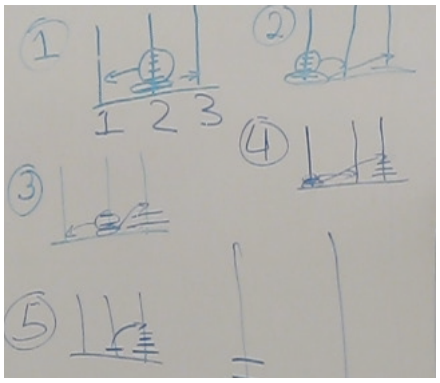


Figure 2.7: One of the student's interpretations of how to solve Towers of Hanoi.

standing to the matchbook problem and in so demonstrate their understanding of recursion.

The initial problem was presented briefly by the instructor and then students were tasked with solving the problem. The first student suggestions revolved around splitting the matchbooks up between bigger and smaller and asking the instructor to tell them which was bigger between select pairs of matchbooks and then grouping accordingly. They continued this approach with all the matchboxes ultimately realizing that once they had this information they did not know where to go next (e.g. "I don't know what to do now?"). The students had forgotten the larger task and appeared lost in the detail. After asking again what the goal of the exercise was, they continued to break each of the initial groups into larger and smaller using the same approach.

At this point, one student commented that all the matchbooks in the bigger pile were larger than all those in the smaller pile causing another to respond "not necessarily". This caused some consternation among the group, with agreement from one who mentioned that "we did not compare all matchbooks in the set" and another suggesting that we restart.

Some instructor directions and refocusing questions (i.e. "Remind me what are we trying to do here?" and asking some students to restate their observations from the initial exercise) helped the group to refocus on the task and to redesign their efforts to solve the problem. One student clearly articulated—through demonstration—that choosing one matchbox as a reference point and then comparing all other match boxes to this one would indicate the relative positioning[8]. After two iterations of this

[8]Note that this together with the recursive method initiated above results in quicksort.

procedure, this student was asked to pass the task on to another who continued the procedure clearly replicating what the first student had set out. This was continued allowing all but one—who was obviously interested but not engaged with the group—to apply this procedure.

In the end, they were able to apply the procedure to all the matchbook pairings and successfully solve the problem. When asked if this was recursion—as one had mentioned that it was—they had to be reminded just what recursion was. After this, one student (the same one who demonstrated the procedure used to solve the problem) clearly articulated how his procedure was in fact recursion with the base case as the point where all matchboxes have been compared.

This experience provided clear evidence of students applying recursion to a novel problem. Although students were obviously at different stages in their understanding they did apply a recursive procedure two times within the activity. The initial, although recursive, was unsuccessful at solving the problem. They then reengaged—and from help with a particular student—were able to recursively solve the problem through quick sort.

2.8 Summary of Results

Described here is a small pre-test post-test study to examine he effectiveness of teaching recursion to middle school students through a series on hands-on interactive unplugged and plugged in activities. Using a variety of data collection tools (i.e., clickers, written answers and video and audio recordings), results point to the students increasing their recognition of recursion and enjoyment of these activities, with some indications that understanding has also improved. We present the discussion in the context of the original questions asked at the onset of this study: do the activities outlined here increase students abilities to (1) recognize recursion, (2) understand recursion, and (3) leave our sessions with a more positive attitude towards Computer Science and recursion? These questions are further explored below.

With regards to the first question (recognizing recursion), there is ample evidence that all the students in this study were able to demonstrate recognition by the conclusions of the seven Week session. In the clicker questions, one student persistently answered incorrectly to the Borax Box question. When asked why, she defended herself saying that if the picture was recursive, then all aspects of the picture are

recursive (including the lettering). This would suggest that this wrong answer does not indicate complete misunderstanding rather the student over applying the term.

Written responses also pointed to a recognition and ability to offer definitions such as 'a picture within a picture forever'. This suggests that the visual image of recursion is a powerful teaching tool that could be exploited extensively to aid with the initial introduction to recursion.

Video and audio data clearly indicates a progression in the recognition of recursion through the activities in this study. Although the students did not consistently apply recursion as a first approach to solving problems presented to them (e.g., the Serpinski Carpet) they did use recursion later in the sessions (e.g., match box sorting) and were able to recognize the recursive elements of all activities once discussions with the facilitators took place. This suggests that their motivation inside sessions was to solve the problems, not just using recursion. However, they did—after discussion with the facilitator—consistently demonstrate a recognition of the problem.

In support of the second question (understanding or applying recursion) had two activities that offered data. The first involves the Towers of Hanoi problem where one student was able to articulate the algorithm for her solution even on the white board. With the guidance of one facilitator, the group developed the MicroWorlds code with the recursive calls and the base case (see Figure 2.7). The other instance involves a different student and the match box sorting activity who, after the groups initial dead end solution (that was recursive we might add), was able to independently demonstrate a recursive procedure that could solve the problem. The student required was the aid in the articulation of the base case and the final verbalization that this was in fact recursive. These two scenarios offer some evidence that certain students were able to demonstrate their understanding of recursion by applying in to new situations.

Additional information could have been elicited through individual student interviews at key points in the lesson to probe a little deeper into their ideas behind these solutions.

The third research question explored the students enthusiasm and enjoyment towards Computer Science activities. Researchers commented independently that there was total 'buy in' amongst all of the participants across all of the activity where data was recorded. Throughout the entire session the students were very enthusiastic and over one half of the group returned for the next offering of SPARCS After-School Club. The students took ownership of their work in several activities, such as animal

chase and Towers of Hanoi. In particular in Week six, we allowed the students to play on their own. All of the students found an activity to play with, going way beyond previous experiences. This outcome leads the researchers to believe that there is promise in this type of exercise.

Chapter 3

Case Study 2: Concurrency

This section will outline concurrency as a sophisticated topic and the challenges that are faced when teaching concurrency. Concurrency is commonly taught in upper years of university but recently there has been a push to teach it earlier in the education stream because of its prevalence in industry.

3.1 Why Teach Concurrency (the new debate)

Over the past few years the topic of teaching concurrency in university computer science classes has become very prevalent. Workshops such as Principles and Practice of Parallel Programming at HPCA '09 and Curricula For Concurrency and Parallelism located at OOPSLA '09. These workshops discuss what to teach, when to teach it, and why to teach concurrency.

An example of a school that has decided to heavily integrate concurrency into their CS curriculum is UC Berkeley. UC Berkeley has decided to introduce concurrency with Map Reduce and Scheme in their first year intro to CS. They then maintain parallelism and concurrency as major themes throughout the rest of the curriculum.

3.2 Teaching Concurrency to Kids

When teaching concurrency, it is challenging to connect concurrency with relevent scenarios that demonstrate the coordiation and communication issues. Students mostly understand how in the real world, agents (humans) are able to do more than one thing in parallel (one human sweeps the floor, while the other sets the table). Race

conditions, deadlock and other challenges in concurrent programming are difficult to motivate because humans are able to take in much more information at a time than a computer can. Humans can use many of their senses concurrently.

3.2.1 Kinesthetic Learning Activities, Games and Toys

In designing kinesthetic learning activities related to parallelism, breaking a task into subtasks that execute concurrently is easily mapped into a setting where participants themselves are independent subtasks. In each of the activities presented here, we draw from examples of real world parallelism—going to a movie, washing dishes, maneuvering a maze, and the classic dining philosophers. Here we provide an overview of these four activities, and how each provides potential insight with respect to natural models for parallelism. Table 3.1 summarizes this overview.

Activity 1: Movie Theatre

In this first activity, students consider a situation where two people arrive at a movie theater with two cashiers, each positioned in such a way that people in different lineups cannot see each other. The movie has already started, and the students need to decide if they will split up to try to buy tickets. The consequences of splitting are that they may end up with too many tickets—we additionally made this a nonrefundable scenario to steepen the consequences.

Each of the subtasks in this scenario are the same: the job is to stand in a line, and purchase two tickets. The speed of the lineups and lack of synchronization make the outcome of this scenario nondeterministic. This activity allows students to reason about the race condition at the heart of the problem, and try to avoid the outcome where four tickets are purchased.

Activity 2: Washing Dishes

The second scenario involves the after dinner task of washing the dishes. Again, there are two participants, and a sense of urgency to get the job done as fast as possible. The elements of work that need to be divided in this case are the subtasks (1) washing each dish in soap and water, (2) drying each dish and (3) putting each dish away in the cupboard.

Unlike the first activity, the students must subtask according to different activities that depend on one another. Additionally, though unintentional on our part, there is

a decoupling that would be possible given a drying tray. Nevertheless, the students are faced with issues of heterogeneous subtasks, communication and workflow.

Activity 3: Maze of Doom

After two such seemingly innocuous activities, we wanted to try to be a little more thrilling for the Grade 7 subjects. In this activity, the students were given the scenario of being hidden somewhere deep in a pitch black maze with corridors lined with spikes and dripping with poisonous goo. The tricky corridors go forward, backward, side to side and in diagonals. Unfortunately, they are intended to seal one in the dungeon forever, unless they can carefully manoever to an exit. Luckily, there two cats that can see in the dark. Due to their lazy nature, one cat can look forward and backward and the other can look side to side. Each indicate which direction you should go.

Simulating this scenario was a little more challenging. The students were split into groups of three and given a printout of a maze and a marker. One student from each group was blindfolded as the person stuck in the maze, and two others were the cats to help navigate the maze. One of the directing students was responsible for indicating when to go forward and backward, and the second was responsible for indicating when to go left or right. The portions of the maze that were on a diagonal required simultaneous instructions to be navigated successfully. Some of the passages were large enough that they could be navigated by creating a step pattern (one step forward/backward, one step to the side) with interleaving of instructions. Before starting the activity the students were required to decide on instructions to indicate a diagonal movement.

In contrast to the previous scenarios, being trapped with magic cats was not an everyday occurrence. Additionally, this activity had urgency associated with "doom" as opposed to a financial penalty or a slight delay. Here we were looking to understand natural tendencies with respect to the simultaneous updating of shared state. Deliberate coordination with simultaneous activity, allowing for a combination of inputs as opposed to an interleaving, was at the heart of this exercise.

Activity 4: Knights and Forks

In this activity, the students became five knights at King Arthur's round table all trying to have dinner, with one fork between each knight. The students were divided

Activity	Description	Concepts
Movie	Two line ups to choose from. Students need to buy two tickets. Should they split up? Or stay together?	non-determinism, communication synchronization, race conditions
Dishes	There are three tasks. Find the most efficient way to divide the tasks and do the dishes with two people	work flow, subtasks, communication
Maze	Escape a maze. Can be told 4 directions to move:Forward, backward, left and right. How to go diagonal?	communication, coordination, shared entity, simultaneous update
Knights	5 people, 5 forks, but you need 2 forks to eat	deadlock, sharing = bad, synchronization, starvation

Table 3.1: Synopsis of activities and their related concepts.

into groups of 5, assumed the role of Knights and attempted to pick up forks inserted between them.

As with the classic dining philosophers problem, this scenario can result in several undesirable interleavings, resulting in starvation of some knights, or even deadlock of all knights.

3.2.2 Procedures

In the spring of 2009 the concurrency activities and question were performed in two consecutive experiments first with 37 students, then an updated version was run with 19 students two months later.

Experiment One

Our experimental group consisted of 37 Grade 7 students from local middle school. This group of students had limited to no programming experience. A subset of the students have seen educational programming languages like Scratch [11], Alice, [58] and MicroWorlds EX [9].

A synopsis of the layout is presented in Figure 3.1. The overall flow was two activities, an initial checkpoint for evaluation (CP1A), two more activities, a follow-

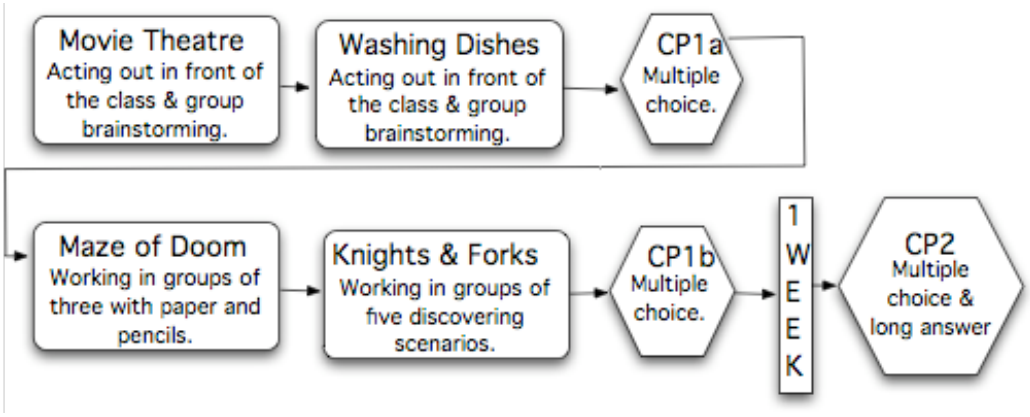


Figure 3.1: Timeline for the first experiment group.

up checkpoint (CP1B), a one Week delay, followed by a final evaluation (CP2). The first two checkpoints (1A and 1B) were the same set of multiple choice questions, and the final checkpoint was both multiple choice and long answer questions.

The facilitators entered the classroom on a Friday afternoon. Two classes were brought together to participate in the activities in the cafeteria. There were five adults present, three facilitators as well as the students' teachers. This group of students had a very intricate set of activities. The first two scenarios (movie theatre and dishwashers) were portrayed by students and facilitators at the front of the classroom. The audience would yell out answers and scenarios that might arise and all students were encouraged to participate.

The second two (Maze of Doom and Knights and Forks) were acted out by the students in groups. There were two evaluation points on this day. One was after the dishwasher activity, and another was at the end of the afternoon. The final evaluation point was administered by the classroom teachers one Week after the workshop.

Experiment Two

This experimental group consisted of 19 grade 7 students from the same local middle school with similar programming experience. The layout of this activity was very similar to experiment one. In experiment two, the students were given two activities an initial checkpoint then another two activities. Two days later, the students were given a different questionnaire, and 6 were randomly selected for interviews (see Figure 3.2).

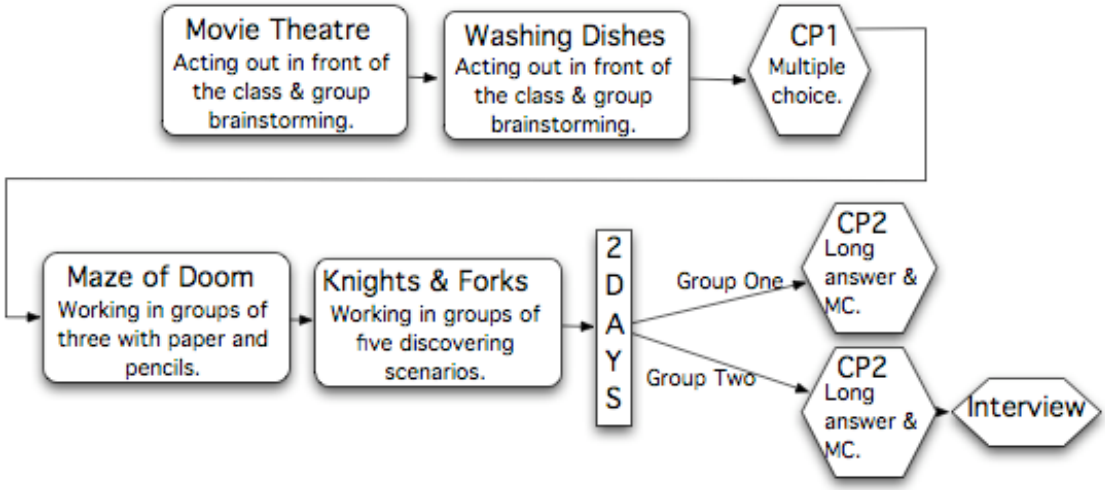


Figure 3.2: Timeline for the second experiment group.
In this iteration, one class of students were taught with just three facilitators present.

3.2.3 Evaluation

Experiment One

To evaluate the participants' understanding of the problems, three questionnaires were collected. The first questionnaire was an 8 question multiple choice questionnaire that asked the students questions regarding the movie theatre and the dish washing activities. The questionnaires asked questions regarding the scenarios given in class, as well as questions exploring the students' understanding of specific problems. This questionnaire was given to the students twice. First as Checkpoint 1A, given to the students right after seeing the problems. Then, at the end of the afternoon, as Checkpoint 1B, after the students had participated in the other two activities.

The final questionnaire was given to the students by their teachers as a reflection questionnaire (Checkpoint 2) one Week after we visited the school. This reflection questionnaire asked the students 4 multiple choice questions and two open ended questions. These questions pertained to the knights and forks, and maze activities. The questions asked the students to evaluate how they solved the problems. The specific problems from all checkpoints are outlined in section 3.3.

In addition to the use of questionnaires the facilitators of the activities were asked to make notes on the participants solutions and problems encountered while completing these activities.

Experiment Two

In this experiment, the participants' understanding of the problems was evaluated with three questionnaires. Most of the questions from this experiment were either the same as, or modified versions of the questions from experiment one. Many of the changes were to convert the multiple choice questions into long answer questions. We did this so that the students were able to demonstrate their knowledge and understanding instead of their recognition of answers. The addition of *Why?* after a few of the multiple choice also attempted to allow the students to be creative and defend their decisions.

The first questionnaire was comprised of one multiple choice question, and three long answer questions. Here we were evaluating the Movie Theatre problem and their understanding of race conditions and non-determinism.

The next questionnaire asked the students to discuss the dish washing scenario. Here there were five multiple choice questions and three long answer questions.

The final questionnaire was administered two days later repeated a few of the questions from questionnaire one and two, but changed most of them into multiple choice questions. This questionnaire had seven multiple choice questions and two long answer questions. These questions were all related to the Movie Theatre problem and the Dish Washing Scenario.

Six students were randomly selected for video recorded interviews. These interviews were one on one and had the students discuss the Knights and Forks problem and a new problem discussing and Etch-A-Sketch. The questions here all had the students reflect on the activities and to describe the solutions them and their group came up with. Beyond this, the students were asked to discuss how these activities were related to each other and the outside world.

3.3 Questions and Answers

Experiment One

The questions were administered in two checkpoints. In check point 1A there were a maximum of 37 students and a minimum of 34. This variance comes from students choosing not to answer questions or answering incorrectly[1].

[1] An example of an incorrect answer, would be if the student responded to multiple answers when the question clearly asked the student to circle one option.

Movie Theatre Scenario								
	1.1		1.2		1.3		1.4	
	#	%	#	%	#	%	#	%
Checkpoint 1A								
a	21	56.8%	3	9.4%	16	50%	0	0%
b	16	43.2%	5	15.6%	16	50%	3	9.4%
c	-	-	24	75%	0	0%	0	0%
d	-	-	-	-	0	0%	29	90.6%
Checkpoint 1B								
a	22	64.7%	2	6.3%	20	60.6%	0	0%
b	12	32.3%	8	25%	12	37.5%	1	3.1%
c	-	-	22	68.8%	0	0%	1	3.1%
d	-	-	-	-	1	3.1%	30	93.8%

Table 3.2: Synopsis of multiple choice questions for The Movie Scenario, see Section 3.3.1.

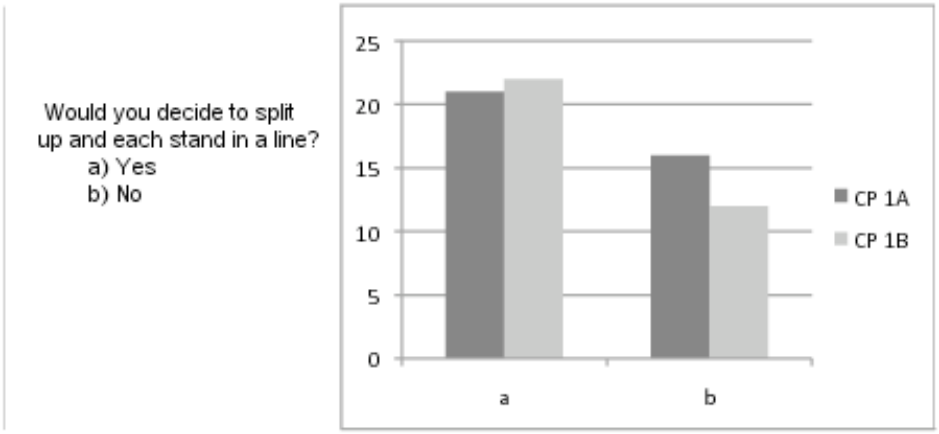
The second half of the first Checkpoint was administered at the end of the workshop. The same questions as in Checkpoint 1A were asked, but the students had gone over many similar concepts through out the workshop. The students were also 'polluted' with a considerable amount of extra information (dead lock, variable manipulation etc.). We had a maximum of 34 students participate in the questions at Checkpoint 1B, and a minimum of 32.

3.3.1 Combined Questions

Movie Theatre

The first two movie theatre questions are designed to discover whether the students understand the non deterministic properties of this scenario. The second half is meant to see what mechanisms they attempt to use to minimize the probability of error. The answer to these questions are represented in table 3.3.1. Table 3.3.1's columns represent the questions that were asked and the rows represent the answers from the students. The columns are broken down between checkpoint 1A and 1B. The checkpoints represent the same questionnaire asked at two different times. Each of the responses is broken up by raw number and percentage of students who answered each response.

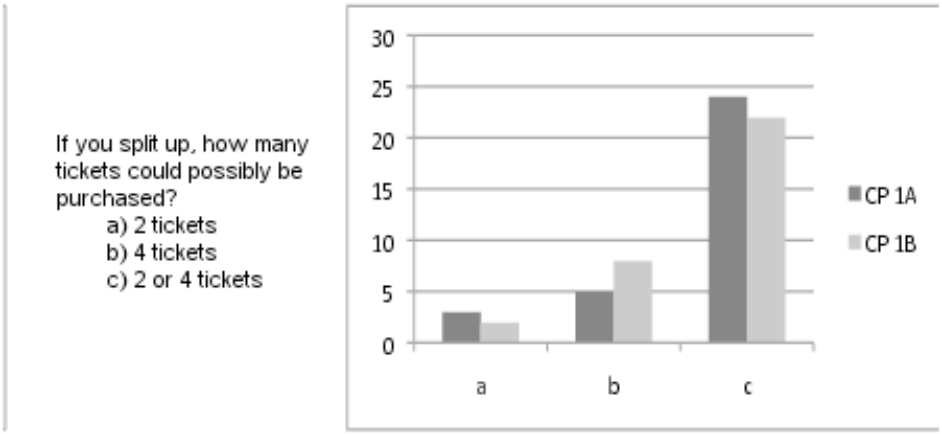
Question 1.1



In question 1.1 during our first checkpoint, 21 students identified that they would split up so each stood in a line (creating a race condition), and 16 identified that they would remain together.

In Checkpoint 1B, we had a total of 34 students answer this question. We had one extra student say that they would split up, to bring that total to 22, and only 12 indicate that they would stick together.

Question 1.2

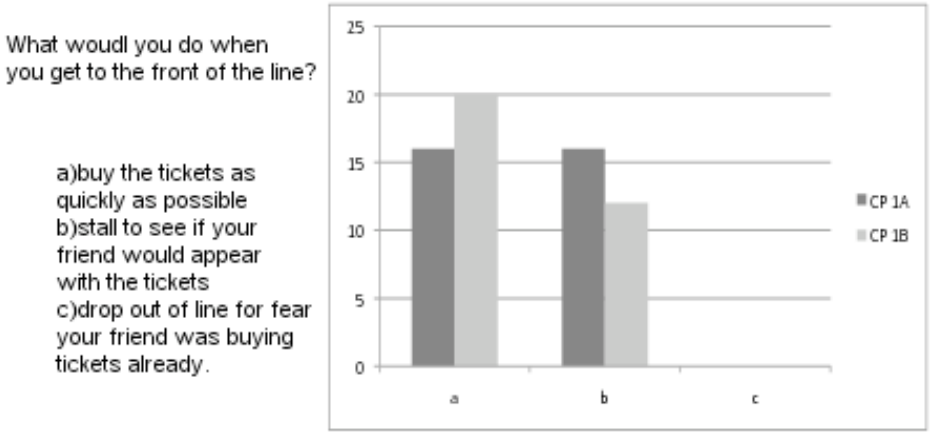


Here 8 students gave deterministic answers (3 selected a, and 5 selected b). The remaining 22 students correctly identified that there was a non deterministic number of tickets that would be bought.

After the days activities, 10 students chose the deterministic options for 1.2. Two indicated a, saying that they anticipated getting 2 tickets, while 8 expected to receive 4. The number of students who expected a non-deterministic amount of tickets (2 or

4) decreased to 22.

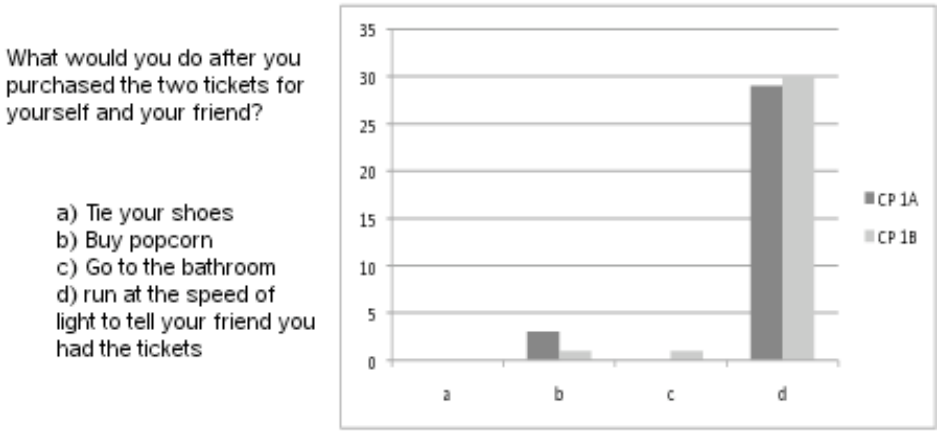
Question 1.3



This question outlines three very different possibilities. Half of the students (16) wanted to buy the tickets as quickly as possible, and the other half wanted to stall to see if their friend would buy the tickets. In this check point, no students were interested in dropping out.

In Checkpoint 1B, the percentage of students who would stall here decreased, while the number of students who would either go very fast or dropped out increased. Twenty students indicated that they would buy the tickets as quickly as possible, while12 students indicated that they would stall.

Question 1.4



The multiple choice answers to this question can be broken down into two categories. First, answers a through c, all give a greater room for error than answer d which attempts to minimize the chance for error. Three students answered b and the

remaining 29 students attempted to minimize the error by running as fast as they could to their friend.

When asked how many students would either stall after buying the tickets (answers a-c), or move as quickly as possible(d), two indicated that they would stall and 30 indicated that they would move as quickly as possible.

Dish Washing Scenario								
	2.1		2.2		2.3		2.4	
	#	%	#	%	#	%	#	%
Checkpoint 1A								
a	3	8.6%	20	58.8%	1	2.7%	6	16.7%
b	11	31.4%	6	17.6%	35	94.6%	19	52.8%
c	15	42.9%	2	5.9%	1	2.7%	8	22.2%
d	6	17.1%	6	17.6%	0	0%	3	8.3%
Checkpoint 1B								
a	3	9.1%	21	65.7%	0	0%	7	20.6%
b	11	33.3%	4	12.1%	32	91.4%	14	41.2%
c	14	42.4%	2	6.3%	3	8.6%	9	26.5%
d	5	15.1%	5	15.6%	0	0%	4	11.8%

Table 3.3: Synopsis of Dish Washing Scenario, see Section 3.3.1.

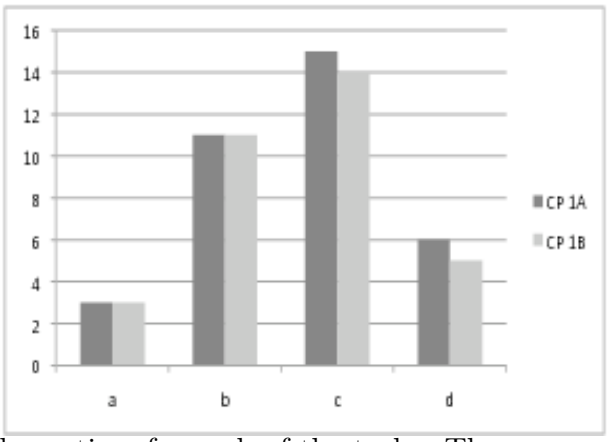
Dish Washing Scenario

This section reports the questions asked and the answers provided after the students helped the facilitators act out the Dish Washing Scenario. These questions were divided so the first three forced the students to think about time management, then Question 2.4 asked the students to create a final plan.

Question 2.1

Which of the following tasks is the fastest for a single dish?

a) washing the dish
b) drying the dish
c) putting the dish away
d) each task takes the same amount of time



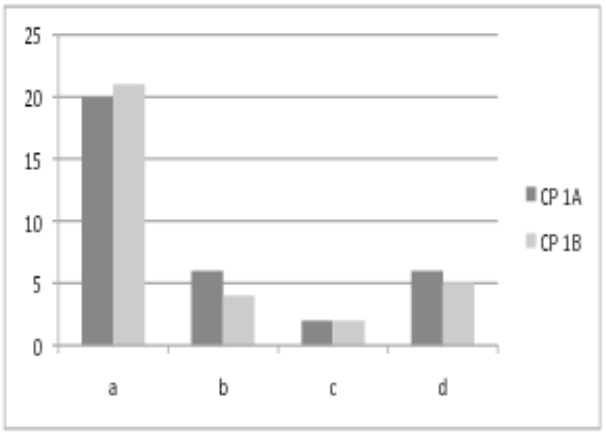
Here students analyzed expected run time for each of the tasks. The answers a b and c can be amalgamated as answers that identify a difference in speed. D, is the only answer that indicates equivalency. Twenty nine students chose one of a,b or c and 6 chose d.

There was very little change in the students perceptions of how long each task would take between Checkpoints a and b. Twenty eight students gave answers that one task would be faster than the others while 5 thought they would all take the same.

2.2

Which of the following tasks is the slowest for a single dish?

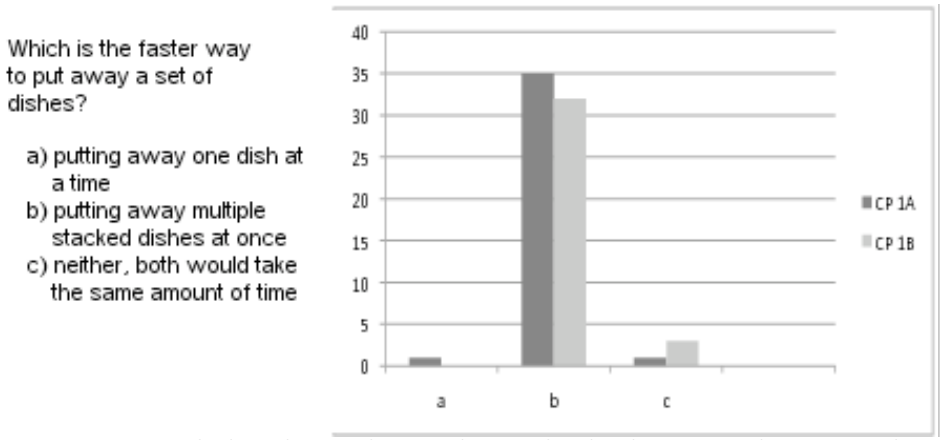
a) washing the dish
b) drying the dish
c) putting the dish away
d) each task takes the same amount of time



In Question 2.2, answers a through c can be amalgamated using the same rational as in 2.1. Most of the students said that the tasks are all accomplished at different speeds. Twenty eight students answered a b or c, while 6 indicated that they would run and the same speed.

In Checkpoint 1B, 27 students felt that there would be a slower task, while 5 thought the tasks would all take the same amount of time.

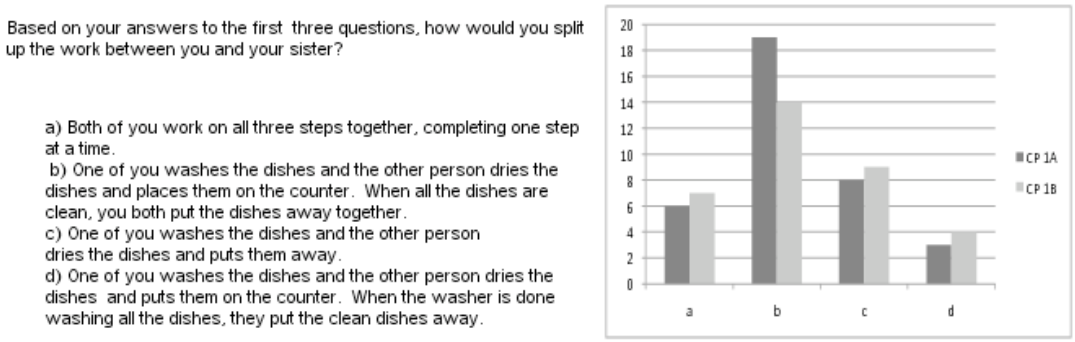
2.3



This question is intended to have the students think about combining tasks. One student answered that putting one dish away at a time would be faster, 35 students answered b (multiple dishes), and 1 thought there would be no difference.

Thirty two students thought that stacking would be faster, while 3 thought that putting multiple dishes away simultaneously would take the same amount of time as a single dish.

Question 2.4



In the last question students were asked how the final division of tasks should work. 6 thought that a would be the best solution, 19 thought solution b would work the best, 8 thought c and the remaining 3 thought d.

The final question, where the students needed to choose a final task division, indicated minor changes between Checkpoints. 7 students thought that completing all

three tasks together was the fastest, 14 thought one washer, one dryer and both put away was the fastest, 9 thought 1 washer 1 dryer-put away and 4 thought 1 washer-put away and 1 dryer was the most efficient.

Checkpoint 2

Checkpoint two, given a Week after the workshop, contained questions pertaining to the knights and forks and the maze activities. The first question gave the students a scenario where they needed to operate two knobs of an extremely large Etch-A-Sketch with a friend. The question asked the students how they would effectively draw a square. The options were as follows:

- You draw the first two lines
- You can control one knob, and your friend controls the other knob
- You draw all the lines of the rectangle yourself
- Your friend draws all the lines of the rectangle herself.

In this situation, the students are required to analyze the best way to split up the tasks. The students should have had some practice with this when discussing how they were going to guide their group member through the maze, and also in their division of labour with the dish-washing activity.

Etch-a-Sketch

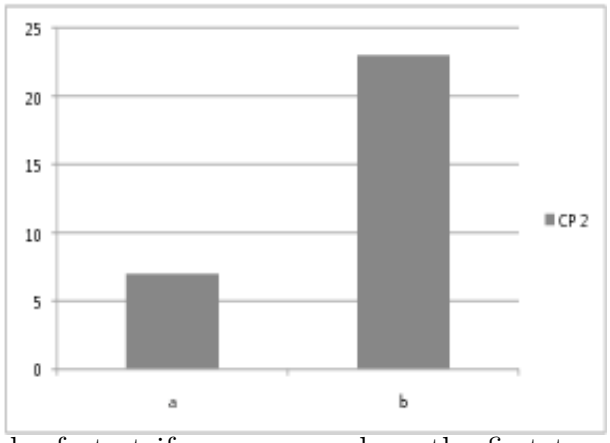
The students were given instructions to read the following scenario and answer Questions 3.1 to 3.4 according to their intuitions.

"You and a friend have found a GIANT etch-a-sketch. There are two knobs on the etch-a-sketch. The left knob draws a vertical line (up when turned left and down when turned right) and the right knob draws a horizontal line (left when turned left and right when turned right). BUT, this etch-a-sketch is SO big that you can't reach both knobs at the same time on your own. You and your friend want to draw a rectangle."

Question 3.1

What is an efficient way to perform this task withyour friend?

a) You draw the first two lines of the rectangle and then your friend draws the other two lines.
b)You can control one knob, and your friend controls the other knob.
c)You draw all the lines of the rectangle yourself.
d) Your friend draws all the lines of the rectangle herself.

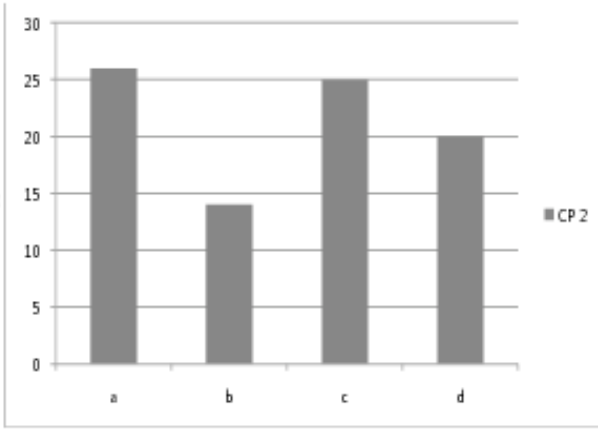


Seven students stated that it would be fastest if one person drew the first two lines, then the other person drew the second two. Twenty three of the students identified it would be fastest if you alternated which lines you drew. None of the students thought that it would be fastest to draw the entire rectangle yourself.

Question 3.2

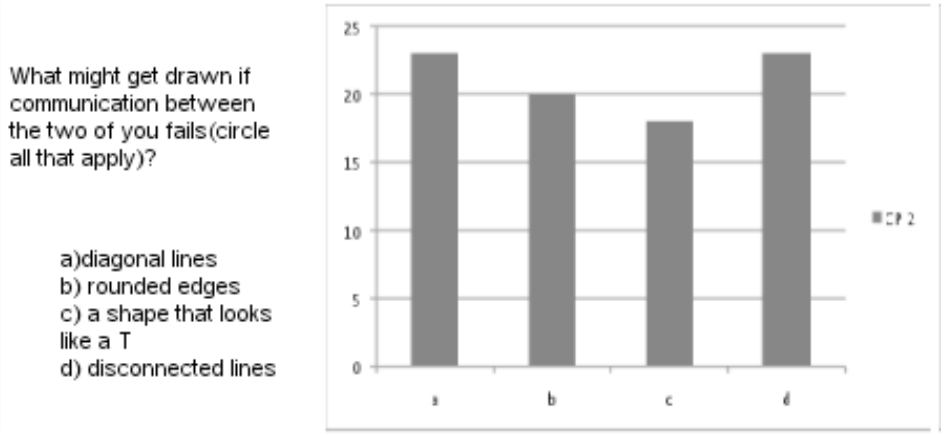
What information will need to be communicated between you and your friend in order for you to work together? (Circle all that apply)

a) Which one of you will start to c first.
b)How fast the knob is being turned.
c)Notification of when each of you finishes your portion of the task.
d) Which side of the rectangle each of you is drawing.



Here most students were able to identify that a lot of communication was necessary. The indication of *how fast each knob was turning* seemed to be the task that most people felt they did not need to communicate. This is possibly a cause of humans having many extra senses that make some of these scenarios difficult to understand as is discussed in the following sections.

Question 3.3



Thirty students responded to this questions. Of these 30, 23 thought that diagonal lines were possible, 20 thought rounded edges might be possible, 18 answered that they might see a T, and 23 thought disconnected lines.

Eight students said that all of the above options were possible. The most surprising outcome, was that there were 7 students who did not expect diagonal lines to be drawn if communication wires were crossed. This was surprising because of the maze activity.

Question 3.4 How would you draw a diagonal line?
When the students answered questions regarding the maze problem and their approach to dealing with the diagonal sections of the maze many of them described drawing with both knobs at the same time:

> *"You turn both knobs at the same time."*
> *"Turn the vertical knob left while you turn the horizontal knob left or right."*
> *"Both people turn their knob at the same time."*

Only one student indicated that speed would play a part in drawing a good diagonal line:

> *" Turn both knobs at the same time and* ***speed****"*[2]

Only four students out of 30 identified that making a staircase or *"Up, side, up, side"* would be an efficient solution.

[2]Emphasis was on the student's sheet

King Arthur's table

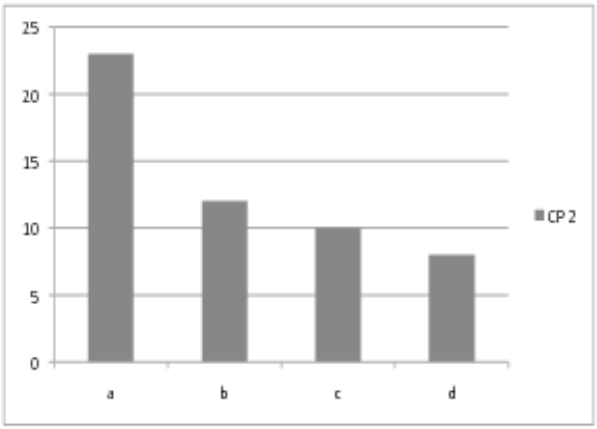
The second section of questions that the students saw in Checkpoint 2 related to their experience with knights and forks the Week before. These questions (4.1 and 4.2) ask the students to highlight the problems that they saw and identify solutions that they decided upon.

Remember the Knights and Forks problem? Five hungry knights are sitting around King Arthur's Round Table, they all have a GIANT plate of spaghetti in front of them and each person needs two forks to be able to eat the spaghetti. There was one problem: there was only one fork between each of the knights.

Question 4.1

Did your Group come across any of the following situations? Circle all that apply.

a) everyone in the group was holding only one fork
b) everyone only got to eat at the same time
c) no one got to eat for a long time
d) everyone ate with their hands



Disregarding the 'silly' answer (d), the viable answers would be a and c. Answer a represents deadlock, while c represents starvation.
We had 27 students respond to this question. Twelve students incorrectly remembered that they could all eat at the same time. Twenty three said they reached a state of 'deadlock', and 10 indicated they reached starvation.

Question 4.2 What solutions did your group come up with? Most of the students gave great answers to this questions. Most indicated they needed to take turns:

> *"Everyone Shared the forks. 2 people ate then gave the forks to 2 other people and after they were finished there was only one person left"*
> *"First 2 people ate, then the other two ate, we shared the forks"*

Another took another approach to *"Chuck people out"*.

Movie Theatre Scenario								
	1.1		1.2		1.3		1.4	
	#	%	#	%	#	%	#	%
Checkpoint 1								
a	16	80%	7	35%	-	-	-	-
b	4	20%	9	45%	-	-	-	-
Junk	-	-	4	20%	-	-	-	-
Checkpoint 2								
a	11	55%	8	40%	1	5%	0	0%
b	9	45%	10	50%	0	0%	1	5%
c	-	-	-	-	0	0%	0	0%
d	-	-	-	-	17	85%	16	80%
Junk	-	-	2	10%	2	10%	3	15%

Table 3.4: Movie theatre problem: a is Yes/Deterministic b is No/Non-Deterministic.

We had some students who did not identify that there was a problem or they did they give a reasonable solution.

> *"We all Eat at the same time"*
> *"Break the forks in half"*
> *"That everyone would pick up the fork on the right side"*

One interesting student looked beyond the scope of the problem and decided to make this problem into a task distribution problem:

> *"sharing the forks, eating with your hands, one person puts meat on the plate for each person"*

Experiment Two

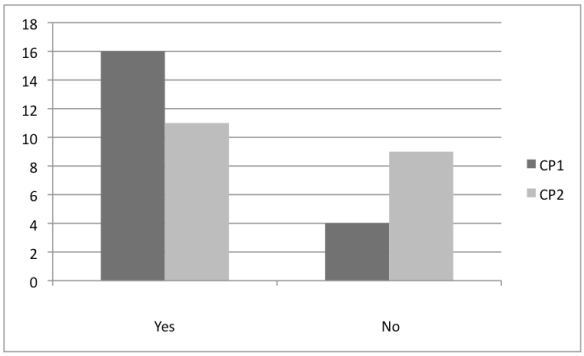
Experiment two was performed a few months after experiment one. This was a different group of grade 7 students at the same school. Much of what changed was in the evaluation questions.

Movie Theatre

Question 1.1

This is the same question as was asked twice in experiment one. In the reflection questionnaire, the students were asked this question again.

Would You decide to split up and each stand in a line?
a) yes
b) no



Some students also added comments:

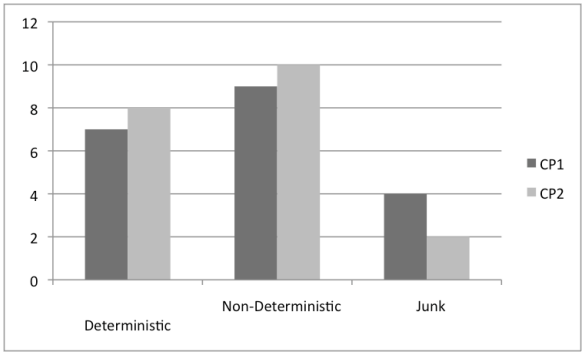
> *"No, because you both might end up with 2 tickets and no money to buy food"*
> *"No, Because you might get lost, and you might both buy tickets because neither of you have a form of communication."*

Question 1.2

What are the possible number of tickets that could be purchased?

In this question, the students gave numbers as answers. The graph below shows the students that gave deterministic answers (either 2 or 4), and the number of students that gave non deterministic answers (multiple possibilities).

What are the possible number of tickets that could be purchased?



Some students had little understanding of the problem, let alone the nuances of the problem space:

> *"It depends on how many friends and how much money you have."*
> "∞"

Dissimilarly, some students had excellent understanding of the problem, and some even the non-determinism of the problem:

> *"wait for your friend"*. (Creates a Deadlock).

Dish Washing Scenario						
	2.1		2.2		2.3	
	#	%	#	%	#	%
Checkpoint 1						
a	1	5%	14	70%	0	0%
b	3	15%	1	5%	15	75%
c	11	55%	3	15%	4	20%
d	3	15%	2	10%	0	0%
Junk	2	10%	0	0%	1	5%
Checkpoint 2						
a	1	5%	15	75%	-	-
b	3	15%	1	5%	-	-
c	12	60%	2	10%	-	-
d	2	10%	1	5%	-	-
Junk	1	5%	1	5%	-	-

Table 3.5: Students' Responses to Washing Dishes

"1,2,3 or 4: You could buy one for yourself thinking your friend would buy her own(and she didn't). Or two because you could each buy one, three because you could buy two and your friend could buy one or four you could both buy two"

Question 1.3

What are your choices when you get to the front of the line? And how will these choices affect your friend?

Some students did not see that there was a race condition, or any time constraints that might affect the friend. Some also didn't see that there were various options:

> *"I'd ask for 2 tickets. This way my friend will have a ticket and wont have to wait in line anymore because if she hasn't come back to me yet she obviously hasn't got a ticket yet"*

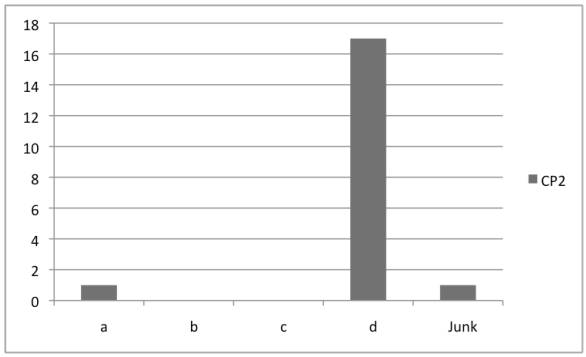
Others understood how their decisions would affect their friend:

> *"buy or don't buy. If you buy some he wont know whether to buy some or not"*
> *"your choice could affect your friend if they had already bought themselves and you a ticket..."*

As a part of the reflection questionnaire, the students were given the same question, but with multiple choice options. This is significantly different from the version from

before because of the d response. This was added because the students would then have the control of a 'lock' of sorts. But by adding this option, the race condition was lost.

a) buy the tickets as quickly as possible
b) stall to see if your friend would appear with the tickets
c) drop out of line for fear your friend was buying tickets already
d) call your friend when there is only one person left in front of you

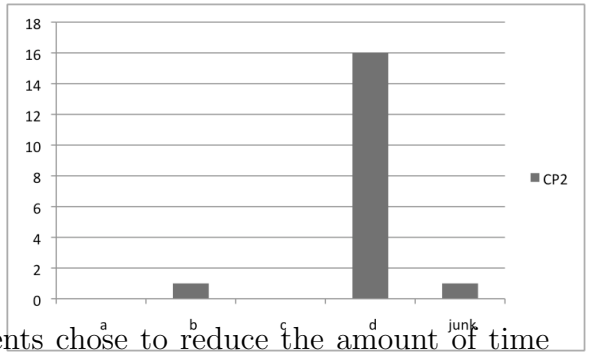


By looking at the graph, most students chose option d. This proved that they saw this as the best answer to avoid the race condition, but this addition made it so that we did not see the variances in the students' understanding.

Question 1.4 What would you do after you purchased the two tickets for yourself and your friend?

> *"Go to the theatre, buy popcorn."*
> *"I'd run to tell my friend not to buy a ticket. If we got too many, we'd sell the rest at the end of the line"*
> *"Find your friend quick so he doesn't buy two more. "*

a) tie your shoe
b) create a distraction so everyone would react and your friend would stop
c) go to the bathroom
d) run at the speed of light to tell your friend you have the tickets



Similarly to experiment two, most students chose to reduce the amount of time that passes by running at the speed of light to find their friend (answer d).

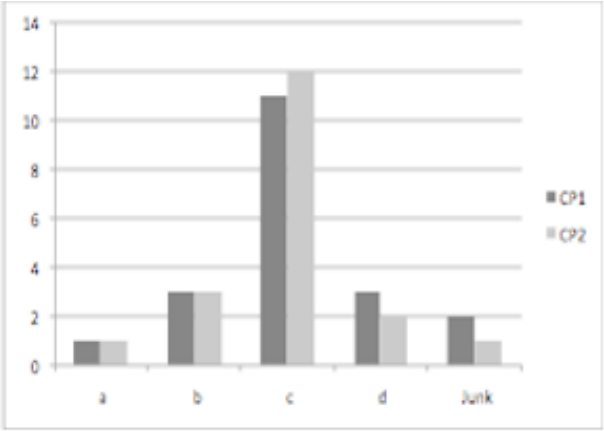
Washing Dishes

In the reflection section, the students were given the following multiple choice options. Questions 2.1 and 2.2 were asked in experiment one.

2.1

Which of the following tasks is the fastest for a single dish?

a) washing the dish
b) drying the dish
c) putting the dish away
d) each task takes the same amount of time

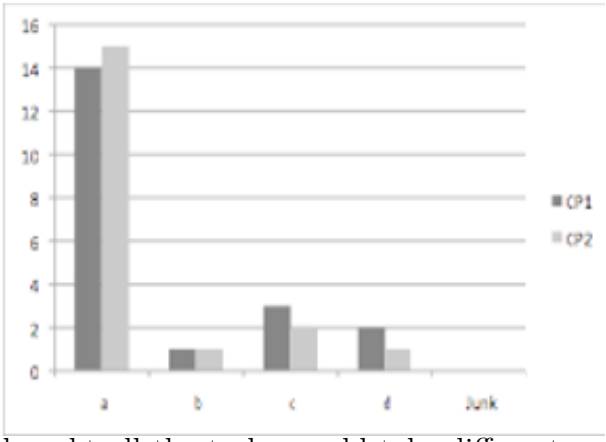


In Checkpoint 1, 25 students chose an option that had a time difference most thinking that drying was that putting a dish away was the fastest, while 3 thought that all of the tasks would take the same amount of time. In Checkpoint 26 students thought there would be a difference where only 2 thought all the tasks would take the same amount of time.

Question 2.2

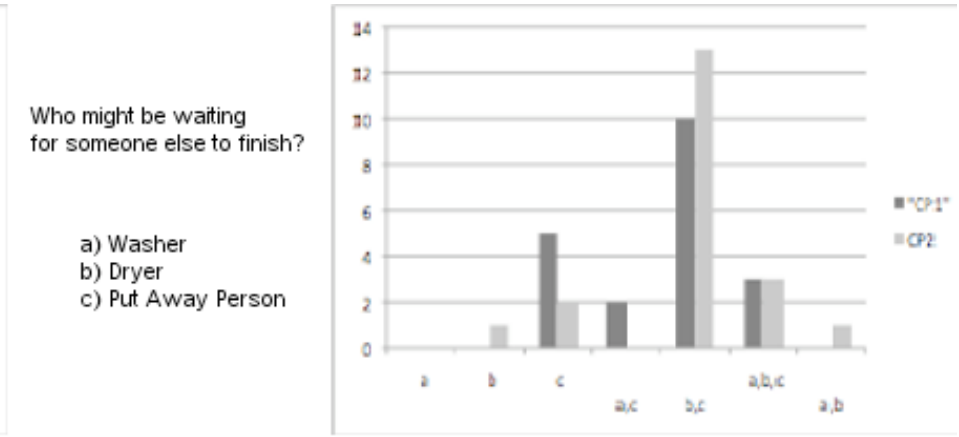
Which of the following tasks is the slowest for a single dish?

a) washing the dish
b) drying the dish
c) putting the dish away
d) each task takes the same amount of time



In Checkpoint 1, 18 students thought all the tasks would take different amounts of time(with washing being the fastest on average), and two thought that they would all take the same amount. In Checkpoint 2, the same amount of students though that the tasks took different amounts of time, where only 1 thought that it would be the same.

Question 2.3



This question attempts to see if the students notice dependencies. The correct answer to this question is that b and c will be waiting. In Checkpoint 1 10 students correctly responded to this question. In Checkpoint 2, 13 correctly identified the dependencies.

2.5 Does this change if there is nowhere to put the dish between steps? Why or Why not?

Again some answers fell short of the expectations:

> *"no"*
> *"No, Because you pass it person to person and the last person puts it away."*

While others understood the dependencies that were created with this situation:

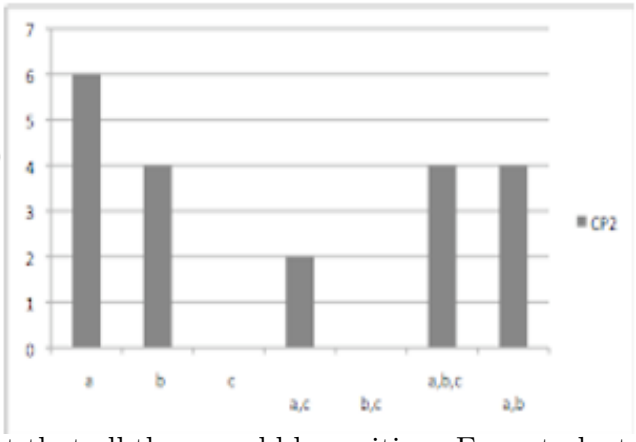
> *"YES!!!, if you had to hold the dish until the next person is finished then you couldn't start on the next dishes and then they could be waiting for you. It slows down the process. "*

In the reflection section, Questions 2.4 and 2.5 were combined into a multiple choice question.

Question 2.6

Who might be waiting for someone else to finish if there is no where to put the dish between steps(ie, no drying rack)?

a) Washer
b) Dryer
c) Put Away Person



The removal of a drying rack meant that all three could be waiting. Four students got this answer correct saying that all three would be waiting. Sixteen students did realize that this would add the fact that the washer would now be waiting which was different from last time.

Question 2.7 How From the reflection Questionnaire: would you split up the work between you and your friend?

> *"I would wash and dry the dishes and my friend would put the dishes away and the next time we wash the dishes we would switch."*
> *"Me: Wash and Rinse, Friend: Dry, Both: Put away "*

This set of answers demonstrated that some students were able to thoroughly think about the different tasks.

2.8 Why did you choose this division?

> *"I choose this division because drying is quicker than washing so if that person could do two jobs at the same pace as the person washing"*
> *"I would wash, my friend would dry and put away. Once I finish washing I'll help put away the dishes."*

Interviews

The interviews all followed a basic form with specific questions that can be seen in table 3.6. The interviews gave the researchers a chance to see how the students

processed the answers and give them some hints. The interviews also helped to understand the full potential of each of the students.

3.4 Summary of Results

Our results are limited by several factors, including our own learning curve with respect to designing kinesthetic activities. The motivation behind the activities for this study followed along with the constructivist theory to take situations that the participants may be familiar with—and interested in–and present them in a manner to highlight the issues in concurrency. Many of these problems seem simple to solve in a multisensory world, and hence potentially uninteresting. Attempting to emphasize the issues that might arise when the problems have to be solved by machines with limited sensory ability is a challenge.

With regards to experiment 1, initially we were hoping to see some difference between Checkpoint 1A and 1B. This did not occur as the students were already very aware of probable solutions before Checkpoint 1A. In experiment two we tried to give the students less information before Checkpoint 1A. By doing this, we wanted to see more of their raw intuition and see more of a change between 1A and 1B. This can hopefully be refined in future experiments. Additionally, Checkpoint 1A would also contain questions pertaining to the knights and forks and maze scenarios.

During design and pilot testing it was found that it was especially difficult to motivate the problem of race conditions. That is, race conditions were perceived to be the most illogical of all the problems presented to pilot testers as there was not a good mapping to this scenario from real world problems. In preliminary testing of some of the activities, we attempted to 'act out' the lemonade problem described by Mitch Resnick in [64]. In the lemonade problem there are two people who are attempting to make a jug of lemonade. They taste the current status of the lemonade then go get the required ingredients (sugar, water or lemon) depending on whether the lemonade is too sour, too potent or too sweet. The students had trouble understanding the problem because of the fact that with all the sensory input and processing abilities available to humans, race conditions are very hard to artificially induce. Additionally, removing the sensory input, especially sight, is difficult and potentially dangerous.

Deadlock was much easier to motivate then race conditions. It was easier to have participants disregard sensory input and follow instructions than in the race condition

Questions	Example Answers
Knights and Forks	
What was the set up for the Knights and forks problem?	Most students remembered the setup, but didn't remember the exact number of people. One student had to draw the table.*yes, deadlock!*
What is the problem in this scenario? Why?	Not enough forks for people.Everone only got one fork, or someone got no forks.
What situations did your group come across?	I got to eat, so someone else didn't. No one could eat.
What were your groups solutions? What are some other solutions?	Kill someone, this helped because when one person was done, he could share his forks. Break forks in half. Take turns, two by two. Skinniest got to eat first, fattest last.
Would this change with limited communication?	If I had no communication, I would put my fork down and hope. Everyone would be stressed out. My table was good, so I'd put my fork down.
Etch-a-Sketch	
What is the most efficient way to perform this task?	You draw the horizontal lines while your friend draws the vertical lines. We'd Each take a knob. Tell eachother when we are done!
What information needs to be communicated between you and your friend?	Stopping, starting.
What might be drawn if the communication fails?	Irregular shapes, random shapes. Kinda straight lines, but all wavey. Half circle corners.
How would you draw a diagonal line?	Turn the knobs at the same time.
General	
What do these problems have in common?	No, not similar, very different. Communication is necessary
How do they tie into computer science?	No connection. Computers need proper instructions. How computers organize their data with processors and priorities. Loading a website and communicating with servers.
Can you think of any other examples or situations with these problems?	Work together. 2 goalie gloves, 2 people, each take a goalie glove,then we have no goalie.

Table 3.6: Summary of Interview data.

scenario. In the pilot tests participants easily arrived in a deadlock and they were able to proceed to test the potential solutions to prevent or resolve deadlock conditions.

The maze activity was designed to motivate the benefit of having coordination with concurrency. The activity was modified from an Alice activity that had ladybugs moving diagonally using a concurrent execution of moving forward and sideways at the same time. Originally we transferred this activity using an Etch-A-Sketch device that limits movement to left and right directions with concurrency used to create diagonals. Due to logistics problems this was replaced with a maze activity that attempt to emulate the limitations of an Etch-A-Sketch.

A major difference between the two experimental phases was the form of evaluation. The addition of long answer questions allowed us to differentiate between the different levels of understanding. Some students that could have slipped between the cracks by guessing the multiple choice answer became very prevalent in the long answer study.

The addition of interview questions gave the students a chance to be interactive and creative. This information again demonstrated the differences between the weaker and stronger students.

These exploratory studies gave us promising feed back on students intuition and curiosity regarding these problem solving tasks. Students came up with some great ideas and were able to identify some key problems with the current implementation of concurrency in software. Based on some of their answers, we were able to predict with some error the outcome of further questions. This indicates that the students have an understanding of some of the patterns pertaining to each activity. These simple and fun kinesthetic learning activities allowed the students to explore computer science in a fun, interactive and constructivist domain.

Chapter 4

Experimental Model: Analysis and Extensions

This chapter discusses and extends the experimental model. The extensions discussed include extensions to the activities, and evaluation models. First we look at the model that we used in our case studies and how they relate to other similar models. Next different methods are discussed extending current tactics and tools for teaching recursion and concurrency including programming languages, patterns, Petri Nets and CSP. Finally this chapter discusses differences between quantitative and qualitative analysis of human data and which types were positive on our research. We then use this information to design a rubric that attempts to make evaluating a student's understanding of concurrency standardized.

4.1 The Model

The current model that we are using incorporates kinesthetic learning activities, problem based learning and some plugged in (activities including technology) activities. Our model uses multiple choice, long answer and observation as evaluation methods. This model is similar to problem based learning and Computer Science Unplugged discussed in Section 1.2.2.

4.2 Extending the Model: Activities

Modern techniques to teach CS are being explored in various venues. Some programming languages have thoughtfully included some key computer science topics and attempted to make them accessible for kids and first year students alike. This section explores some new ideas that could extend some of these models even further.

4.2.1 Recursion

As stated before, recursion is an age old debate. The kinesthetic learning activities that were created here are just the tip of the iceberg. It is very common to portray recursion visually using stack tracing, but where have the recursive languages gone with recursion?

Programming Languages: a Recursive Perspective

In the case study we did a small analysis of which of two languages we would use, Alice or MicroWorlds EX. As mentioned in this analysis, both of these languages have their shortcomings. Alice has a complicated interface with lots going on and the images are difficult to manipulate. MicroWorlds has a much simpler environment with simple images, but the syntax can add an extra layer of complexity for some students.

A very popular and successful language is Scratch (as described in Section 1.2.1). We have been exploring using Scratch to demonstrate recursion. Scratch does not have true functions, parameters or a stack. This makes it nearly impossible to do head recursion, but with the broadcast and receive blocks we are able to teach establish a sense of tail recursion. With the addition of parameters, a proper stack and function names Scratch would be a great alternative for teaching simple concepts in recursion. Figure 4.1 demonstrates a mock-up we presented at Scratch@MIT in July 2008.

4.2.2 Concurrency

Concurrency is relatively new in the CSEd debates, but there are still various ways already created to extend the model and teach it in novel ways. In this section, design patterns, Petri nets, CSP and educational programming languages are explored as interesting methods to teach concurrency.

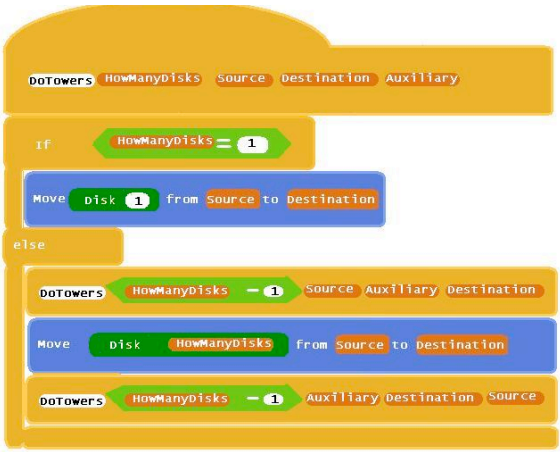


Figure 4.1: Mock-up of recursive towers of Hanoi in Scratch like blocks.

Patterns

Patterns have been a key feature in software engineering design and refactoring. A parallel pattern language is now being developed to help developers understand the contentions and techniques involved in parallel programming [10]. This language is being developed similarly to how the patterns are described in the 'Gang of Four' book *Design Patterns: Elements of Reusable Object Oriented Design* [34]. Arguably all good programs should adhere to at least one if not multiple instances of these patterns and they contain important knowledge for many student and professional programmers. The first purpose that the writers express is: "Education: Patterns have pedagogical value and help new parallel programmers more quickly master the field by learning *the bag of tools* of the experts" [34].

Although the activities described in this document involve movie theatres, cats and washing dishes, they can be described using parallel patterns—described by the members of the Berkeley ParLab—to the intended learning outcomes of the activities proposed in the body of the thesis.

Relevant patterns

Pipeline [52]: This pattern has ordered—but independent—communication stages similar to an assembly line. At the beginning of the process the end computations (or steps) are idle. When the program is in the middle of execution all stages of the pipeline are busy. When the process is near its end the beginning computations are idle.

This pattern can be used to describe the dependencies observed in the dish washing situation. In thhe dish washing situation, there are three computations: wash a

dish, dry a dish and put a dish away. There are also some dependencies that are created with this scenario (described in Section 3.2.1) that are described in the pipeline pattern.

Discrete Event Pattern: The discrete event pattern is similar to the pipeline pattern. There are multiple tasks working together passing data between them. The difference between the two patterns is because the discrete event pattern is not a linear or timed path. A unit of execution (EU) begins when it receives an event. The event contains data on which the EU is to perform some computation. Once the EU has completed, it then broadcasts an event and a 'listening' EU then receives the event and begins its own computation. Often when using this pattern there will be an event handler that manages the communication between the EUs.

The dishwashing scenario could also be described using this pattern. Another common scenario that is described by this pattern, is that of a newsroom where many bits of information are sent to various people and the people begin execution on the data when they receive it.

Mutual Exclusion [10]: Mutual exclusion is described as a pattern that coordinates the execution of threads or processes. This pattern is utilized when shared memory must be accessed for read and writes. When this pattern is not used in these situation, race conditions and non-determinism are created. This pattern's solution is that one must define sections of code or memory that can only be executed in serial. The problems that mutual exclusion address are explored in the movie theatre problem and in the Etch-A-Sketch problem[1].

Task Parallelism [21]: When looking at a problem often it can be divided into different tasks, and have a variety of different divisions. When deciding how to divide a task one must weigh the options. Dividing a problem into a large number of smaller tasks allows for better load balancing, but as the number of tasks increase, so does the overhead. The inverse is true for a small number of tasks. This pattern again comes in to play when working with the dish washing scenario. The students are required to see how the tasks can be split up, and the overhead of communication with smaller divisions becomes prevalent. The added scenario where students don't have a dish rack, adds to the increased overhead and dependencies between tasks.

[1]If you draw a Petri net for drawing straight lines with an Etch-a-Sketch, it looks the same as for mutual exclusion.

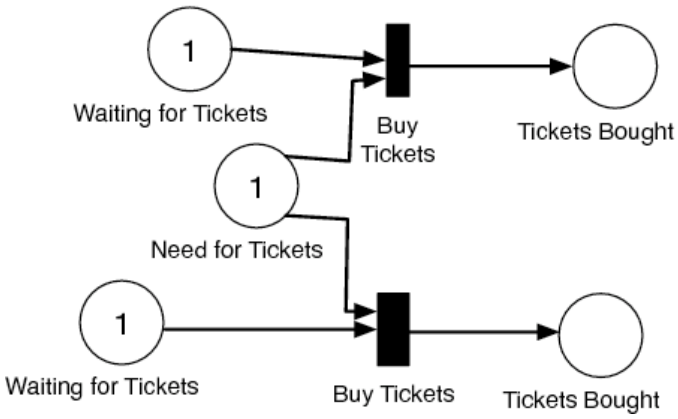


Figure 4.2: Petri Net of an ideal Ticket Scenario.

Petri Nets

Petri nets are a common way of describing coordination of concurrent programs. The circles in the Petri net represent states that the program can be in, and the rectangles represent transitions. Tokens are present to describe what state the system is in, and to transition between states their must be a token in each of the states entering the transition. Some of the edges have numbers on them that changes the number of tokens that are either required to enter the transition, or are given to the next state following the transition.

Similarly to how patterns can represent our 'non-traditional' problems, Petri nets can also give a great representation of the communication and co-ordination of our systems.

The Petri net in Figure 4.2 represents the ideal solution to the movie theatre problem. This solution has a mechanism for deciphering whether or not there is still a need to buy tickets. the students must find a real life solution for this mechanism.

Currently Petri nets are taught in upper year concurrency classes. Looking at these simplistic models that relate to real world situations, this notation (and others like it), could be transferred into earlier CS classes to give students the opportunity to explore these notations. The rest of the Petri nets for the problems described in the concurrency section, are included in Appendix A.

CSP

Communicating Sequential Process (CSP) is a language definition that demonstrates the dependencies in this system. CSP is a combination of various mathe-

Definitions and alphabet: Ticket Sales

K_i = A kid/patron that wants to buy a ticket

V_i = A teller/ Cashier

KS = The group of all of the patrons

M = The mechanism that Is used to get the attention of the other person.

$$\alpha K_i = \{wait, hear, buy, tell\}$$

$$\alpha V_i = \{sell\}$$

$$\alpha M = \{hear_n\}$$

$$K_i = wait \rightarrow ((hear_i \rightarrow SKIP)|(buy \rightarrow M(i) \rightarrow SKIP))$$

$$V_i = sell \rightarrow V_i$$

$$\text{M(i)} = hear_{(i+1 mod 2)}; \surd$$

$$\text{KS} = \{k_1 || K_2\}$$

$$\text{VEND} = \{V_1 || V_2\}$$

$$\text{MTP} = \{KS \| VEND\}$$

Language	Concurrent Aspects
StarLogo TNG	- Multiple Threads at once - Used for simulations - Can create race conditions - Programming constructs for parallel programming
MicroWorlds EX	- Turtles run in parallel - Functions for parallel programming
Scratch	- Looks parallel to the user. - All pieces of code with the same 'hat' block will begin to execute at the same time.
Greenfoot	- Runs off of actor model. - All classes extensions of actor class. - Each actor runs in a thread.(Hidden to the user)
Mindsorms	-The different 'tracks' are threads. -Students can have different components of a program running on each track.

Table 4.1: Educational Programming Languages that Support Concurrency

matical symbols that describe how the processes work together. Non-determinism, resource sharing and dependencies are easily described using these symbols [40].

In the CSP for the dishwasher problem, the sink object is required to wash the dishes and a towel is required to dry the dishes. Symbols are used as a counter for how many dishes are at what point in the system. These mechanisms are also used as a barrier. This is especially apparently in answer a. An 'empty' signal is sent once all of the dishes have been washed, a 'done' signal is sent to allow for the dishes to begin to be dried. The same situation happens when the dishes are allowed to be put away. This is also in Appendix A.

Patterns Petri nets and CSP are commonly used to describe complex systems for upper year students and professionals, but as can be seen in this section they can also be applied to these real world scenarios. This makes them an excellent educational tool–not for our 7th grade subjects—for our first year students.

Programming Languages: a Concurrent Perspective

Modern programming languages for kids often have inherent concurrent components. These languages, often graphical in nature, make it very simple for the students to have multiple characters on the screen acting and reacting at once. An overview of these languages is provided in Section 1.2, here a report on their concurrent aspects is outlined.

StarLogo TNG is designed to have multiple turtles running at once. This program has a control structure called *Run once* which has a place to clip information for each type of turtle to do at the same time. The user is also able to design the program for multiple instances of the same turtle to be cloned, and all respond to commands simultaneously.

MicroWorlds EX allows for multiple turtles to be run at once. There are a few commands that allow for this. The *when* command tells MicroWorlds to wait until something is true and then runs the next instruction in the list. *Run* takes an instruction list as a parameter, and begins each of the instructions as a parallel process.

Scratch has sprites as its 'threads' (similar to the other languages' turtles). Each sprite is able to be doing multiple actions at the same time. Each collection of blocks start with a hat block. This hat block indicates which event triggers these collections to start running. For example when the *green flag* button is clicked, then all of the green flag hat blocks will initiate and run in parallel.

The Scratch IDE is sometimes dependant on the concurrency that is naturally available. In the creation of a game of pong, one of the professors wrote his program using a polling system with if statements while the students created a concurrent version. When the two programs were run, the polling version was unable to keep up with the dynamic aspects of the system and the concurrent versions were the only ones that worked.

Greenfoot is based on objects. All of the characters in Greenfoot are objects that extend the Actor class and they all have the act method. The act method of each character is executed concurrently when the *act* button is clicked. When the *run* button is clicked, all of the actors methods are executed repeatedly in parallel. This gives the students concurrency by default.

Mindstorms makes parallel programming fairly intuitive for young students. Mindstorms blocks run on a track and execute in sequential order. To run the program with parallel components, you simply add an extra track. From observations and working with groups, this solution is fairly intuitive to young programmers, but unintuitive to older programmers. Similarly to the polling problem in Scratch, the students were given the problem to move forwards until the robot detected a black line. In one of our sessions, two young students—who had never programmed—wrote a parallel program to solve this problem with the light sensor on a separate track. The solution that commonly observed with older, experienced, programmers is a polling loop.

On one hand, these features are helpful in teaching students about concurrency and forcing the students to realize that concurrency and parallelism is an inherent part of programming. On the other hand, many of these languages abstract away the dependencies and the problems that can arise when working in parallel.

4.3 Extending the Model:Evaluation and Analysis

There are various different evaluation methods that educators use today. This Section describes our decisions with qualitative vs. quantitative analysis, and how this could be standardized with the help of a rubric.

4.3.1 Quantitative vs. Qualitative Analysis

The evaluation methods that we used for these experiments were multiple choice questions (clicker and paper), long answer questions, video analysis, interviews, pre-test and post-test and audio recording. These methods were evaluating the students understanding of the activities and their ability to apply the lessons learned to concepts related to recursion and concurrency.

If we refer back to Figure 1.4 and Table 1.4 it is observed that the evaluation methods evolved drastically between all three stages.

In the first experiment, clicker questions were the main method of evaluation. This technology makes it very easy to record students' responses with out much instructor overhead. Unfortunately the students began messing around with their responses on the clicker questions. They would give invalid answers, or change their answers multiple times to play around[2]. This resulted in the students' answers being incorrect giving the impression that their understanding was decreasing over time. In this experiment we also used two long answer questions, video data and pre and post tests for analysis. The rest of the data contradicted the students' declining understanding. In this data the most valuable information came from the videos from the pre-test and post-tests. These allowed us to see fully the difference between the students' competency levels before and after the interaction.

In the second experiment, because of our problems with clickers with the first group and our short interaction time, we continued using multiple choice questions but in paper format because of ease of analysis. Again we used a few long answer

[2]Video data demonstrates them discussing their antics.

questions, and the same questions were repeated multiple times over a period of time to observe possible effects of the interaction (acting as a pre and post test). The quantitative analysis of the data in phase two included creating cross tabulations to allow us to see the interrelationships between two or more variables. These interrelationships could point us towards causal relationships and indicators of understanding.

In phase three, we decided to use some multiple choice questions, but mostly long answer questions and video data. The students could be extremely creative with their answers and the researchers could observe the depth of the students' understanding. The video data consisted of data from the classroom as a whole and individual interviews from 6 participants. The interviews were especially helpful because it allowed the researchers to completely observe the thought process of the individual participants. It also allowed the researchers to ask leading questions so that if the student did not create the correct answer themselves, recognition of the correct answer could then be evaluated. The video data allowed us to see the activities were being effective and where they were falling short.

The most effective methods of evaluation were the methods that took the most amount of time and required the most understanding of the material by the evaluators. These stipulations make evaluation difficult when attempting to integrate this material into elementary school classrooms. In the final phase the research is extended to include the development of a rubric designed to make evaluation of some of this material easier for classroom teachers.

Self evaluation in phase one of the experiment showed us that pre-test and post-test are a very valuable method of evaluation. In phases two and three we attempted to get some information from pre-test and post-test by asking the students the same questions at the beginning of the workshop as at the end. This was not as effective as in the first phase because we were simply asking the students to remember the scenario from before. This could be made much more effective if we were able to create some parallel activities and conduct one at the beginning of the workshop, and another at the end.

Complex learning is diffucult to assess or evaluate using a single measure. Both the process of learning and the final products need to be evaluated, individually and together. Regardless of which components are evaluated, currently students only care about the grade that they receive and not the feedback. One solution to these problems is to use rubrics for evaluation and assessment.

"A rubric is a set of scales, one for each criterion that is considered important" [42], different aspects of a project or assignment that need to be assessed are identified. Once the important learning concepts have been identified, the different levels of 'understanding' are fleshed out and placed in order. It is important that these ratings are element specific with very descriptive expectations outlined along with aspects of the performance that are considered important [55]. Quite often ideas/concepts end up being clumped together when put into rubrics. If the elements are clumped together, it is much more difficult to construct ratings that target each one specifcally. The proper way to make a rubric is to separate each concept into its finest granularity [42] and expanded into elemental concepts, making sure each element is unidimensional.

A key idea is that a rubric is a *set of scales*. This highlights the necessity for concepts to each have their own distinct scale. When ideas are placed on the same scale, it is sometimes hard to distinguish between levels on the scale for each element. A good example might be a scale that describes the student's engagement might have four levels, while one that discusses their adherence to style formatting might be binary.

Additionally, there are two different types of assessment rubrics, *holistic* and *analytic* [55]. Holistic rubrics are designed to encompass the entire problem (product and process). This is beneficial when minor errors are acceptable throughout the problem as long as the summative result is the focus of the project. Analytic rubrics have a finer granularity and assess each different aspect of the scoring criteria. Students are able to get very detailed feedback, but the overhead for the grader/educator is much larger.

In phase three, a rubric was developed and explored by five research assistants. Table 4.3 shows the outline of the rubric. The left most column describes the different components that the students are supposed to understand. Each of the following columns is the levels at which the students can understand. In general, level one is when the student has limited to no understanding of the concept. Level two is when the student recognizes the component when it is given to them. Level three is when the student understands the component at the 'expected' level and level four is when the student adds some components of creativity to his answer.

The research assistants were each given data from 3 students and asked to rateeach student's understanding and engagement with the problem on a scale from 1 to 10. After this, the assistant was asked to rate the student on the rubric. The goal of this survey was to see if the rubric could make it so that multiple people would grade the

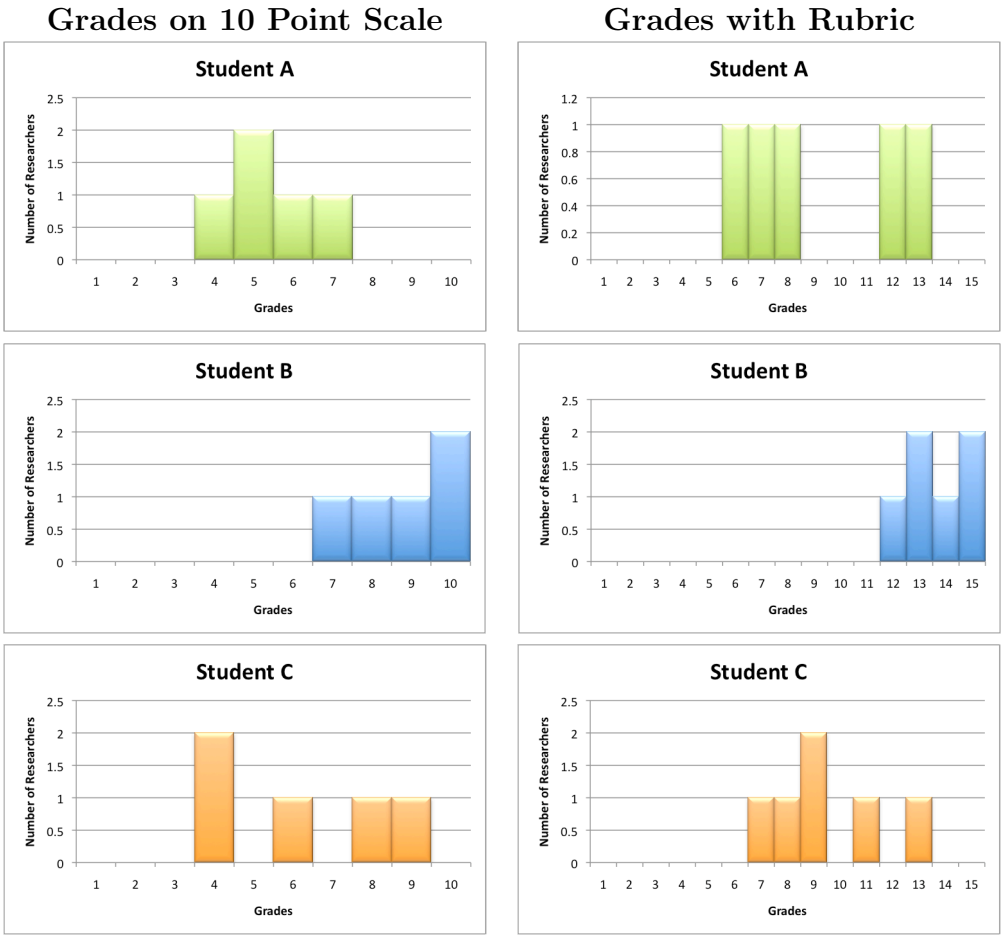


Table 4.2: Histograms from rubric analysis.

same student similarly, whereas when they graded the students on a 1 to 10 scale there would be less commonality between the grades.

The outcome of this experiment was not conclusive. The charts in Table 4.2 are histograms of the answers of our 5 researchers. Where the researchers answered between values, one was added to the two buckets that the number straddled (eg. 1.5 would add 1 to each 1 and 2).

These graphs are fairly inconclusive of the reliability and consistency of the rubric. If they supported our rubric, then the graphs in the right column would be less dispersed than the graphs on the left. This is clearly not the case. This tells us that a) we need more data points, and b) our rubric needs to be refined more.

Concept / Level	1	2	3	4
Understanding Resource Sharing	Did not see that contention existed.	Recognized some aspects of the contention or limited ability to explain. recognized aspects	Identification of all the aspects. Proper explaination of reasons for contention	Ability to create/describe a novel situation where contention exists
Solution to Resource Sharing	Did not see that there was a solution.	Recognized the solutions but had limited logic.	Came up with a viable solution.	Invented a creative solution to the problem and thought of additional solutions
Identification of a Non-Determinisic Situation a	Assumed that the situation was deterministic.	Recognized the solutions, but had limited logic.	Saw the non determinism in the problem and was able to explain the logic properly.	
Solution to Non-Determinism	Solution does not attempt to solve the problem.	Solution attempts to solve the problem, but not all problems are considered.	Solution solves the problem by giving striaght forward answers. Logic is also included in the answers.	Solution solves the problem by giving exemplary and creative answers and logic. Well constructed ideas.

Table 4.3: The rubric created for concurrency.

Chapter 5

Conclusions and Future Work

CSEd is a new subject and there is lots to explore. The basis of this research is that there are various ways to teach computer science to students of all ages. To demonstrate this, we worked with stereotypically the most difficult of age groups, pre-teens.

The results of these activities demonstrated that the students understood some of the basics of our two case studies. In the recursion case study (phase one), many students were able to identify a recursive call and the base case. In the concurrency case study (phases two and three), students could properly identify contention and non-determinism in our fun scenarios. The results also demonstrated that the students became very engaged with the problems. They worked as teams problem solving the scenarios often coming up with creative ideas. At the very least, we can say that the students were exposed to the problems. If the students did not understand all of the components of our lessons, we can hope that they remember some of the vocabulary and may remember it when they come across the terms later in life.

Phase One: Recursion

The initial findings from this study must be treated as suggestive as we employed a single group pre-test post-test design because we wanted to conduct both a pilot study and one that needed to move forward within a limited time period. We further recognize that the choices made in the initial design come with some inherent limitations. Firstly, there is a question about the internal validity. We cannot tell if the independent variable and not some extraneous or confounding one produced the observed effect. Because we had a small group size and no control group or comparison subjects, we cannot be sure that something in addition to the weekly sessions might have changed subjects understanding of recursion. Secondly, experimenter and sub-

ject effects are definitely a threat here as we saw evidence that subjects made efforts to please the researchers and at the same time the researchers suffered potentially from the halo effect (i.e., allowing initial observations to influence later interactions and observations). Both of these factors influence the generalizability of our results.

Our study has been described on the basis of what activities were covered during each weekly session and how students responded to each of these sessions. We recognize that we had a small number of students; however, due to the exploratory nature of the study and the realization that small groups enable better participation and management, we felt this was a reasonable choice at this time for a preliminary study. Notwithstanding its limitations, this study represented an important effort to evaluate the capability of middle school students to engage positively with computer science activities and to understand the ideas of recursion. We found that hands-on activities such as these can increase the engagement and learning of a small group of students. Despite the fears expressed within the discipline of computer science towards both teaching and learning recursion, we have uncovered that younger students can at the very least learn to recognize and enjoy the experience of learning recursion. We also exposed some evidence towards understanding and application of recursion.

During the pilot investigation we discovered a number of research questions that we plan to incorporate into the larger activities or to study on an individual basis with select students. For example, the activity Sierpinski's Carpet involves a 'describer' presented with a picture of Sierpinski's Carpet. After a certain number of iterations, a drawer is supposed to replicate the picture following exactly the instructions given by the describer. During the pilot study we battled with determining a 'fair' way to choose a presentation of Sierpinski's Carpet for the picture (i.e., how many iterations should be included?). These questions will be further included within the recursion activities to help us in obtaining more informative results.

Another change in the overall design and delivery of the recursion workshop may be to initially teach the basics of a programming language to the participants. Then focusing the instructional delivery on the content of the recursion paradigm and its realization, instead of having to deal with basic programming issues in parallel.

It was observed that questions and challenges with the programming aspect of the recursion paradigm actually surfaced during the activities on recursion and we felt that this might have taken away some of the focus on and potential learning. There is some potential to run tandem classes to help answer this question with one class

receiving the parallel instruction of programming and recursion and one receiving the same content in series (i.e., programming first and recursion activities second).

Overall, the large scale study that will be extended from this pilot will better focus on our research questions (and subsequent data collection): do students improve their understanding of recursion after exposure to our unplugged activities; do attitudes towards computer science change significantly after these same activities; and does there exist a relationship exists between attitudes towards computer science and enjoyment and success in these activities? The design will follow the same pre-test and post-test design. Additionally, students' attitudes towards computer science will be measured in a pre and post-session attitudinal measure. In this way, results of significant effect can be determined as can any correlation between general attitudes towards computer science and outcomes that are a result of involvement in these activities.

Phase Two: Concurrency I

This phase was the first phase where we ran a study that did not have prolonged intervention. This study was also too small to have significant numbers so must treated at a suggestive study. The conclusions from this study gave the researchers promise that there is value in this research. This study attempted to expand on some of the knowledge learned from the execution of the recursion workshop in phase one. We learned that there was great overhead when working with the clicker system so for this phase we chose to work with paper answers only.

By the end of this workshop the students seemed sufficiently comfortable with some of the concepts and the activities they were presented. They were aware of the vocabulary that was presented to them and were able to find creative ways that these activities related to computer science.

Phase Three: Concurrency II

From the first two phases of the experiment I had the opportunity to try out various exercises, techniques and methods of evaluation. From these phases, I chose to use mostly long answer questions and video data. In phase two, I had used multiple choice questions which are easier to evaluate, but the different levels of understanding were easier to distinguish using long answer questions.

The interviews turned out to be the most valuable information that we received from the students. It gave the students an opportunity to explore the answers with the instructor and also allowed the students to precisely express their level of understanding. These interviews might be very difficult for teachers to execute in the

classroom, but they are very helpful in evaluating the effectiveness of the activities. The interviews could be extended into group interviews to make marking less strenuous on the teacher, and to give the students the opportunity to express themselves with their peers.

As was learned with the recursion phase of the experiment, pre-test and post-test are a very valuable method of evaluation. With this experiment we attempted to get some information from pre test and post test by asking the students the same questions at the beginning of the workshop as at the end. This was not as effective as in the phase one, because we were simply asking the students to remember the scenario from before. This could be made much more effective if we were able to create some parallel activities and conduct one at the beginning of the workshop, and another at the end. We could then see the difference between the students' success on the activities, and what they find difficult.

In the development of the rubric, many lessons were first learned through research and reading up on rubric development. Then it was continued though a pilot user study with research assistants.

When the user study is expanded to be used with participants (instead of research assistants), the ordering will be different. The order for this activity was that the researchers graded the students on a 10 point scale then they will go back and grade each student using the rubric. This will mean that the researchers are not thinking of the rubric at all when they are grading with the 10 point scale. Also after talking to the research assistants, they felt that the video data contained much more information, and it was easier for them to give marks to the students when using this medium. Because of this, in following iterations, the interviews will be longer and the researchers who are evaluating the rubric will not see any of the multiple choice paper answers.

The Big Picture

With these case studies along with the related work in the field, we can see that this is just the beginning. The methods here are limited by the size of the groups and the rigour of the questions and experimental method. In future phases of the study, we would like to have better questions and ones that are easier on which to perform cross tabulations. Additionally further experimentation using questions in one on one and group interviews will help to give the researchers a better idea of how students process these questions and problems.

The size of the experimental groups also needs to be expanded. For the work to be significant, it would be excellent to have hundreds of kids participate in these experiments from various backgrounds. With a larger number of students participating, the option of structuring the experiment using a control group would also give greater validity to the results.

Recursion and concurrency are simply two of the case studies that are possible. Other work has been done using graph theory [19] and could further be done using any concept that is considered difficult to learn in university education.

One major goal of this research is to discover how to integrate computer science seamlessly into K-12 education in British Columbia. Further work with current requirements and feedback from teachers would allow for the activities to fit better with classroom requirements. By having proper CS education in elementary and high school, hopefully first year computer science would not be so daunting and difficult.

As we begin to look outward for ways to increase the exposure of computer science to largely untapped populations, approaches such as the ones outlined here represent an avenue towards this change. We look forward to continuing our research and to future studies that replicate and expand upon our findings. It is anticipated that we can demonstrate significant effects on knowledge gain of complex topics. We are also interested in improving general attitudes towards computer science along with an increased appreciation for the field. After all, this population includes our computer scientists of tomorrow.

Appendix A

Petri Nets and CSP

Definitions:Washing Dishes

WASH = The act of cleaning the dishes with water and placing them to be dried.
DRY = The act of drying the dishes and getting them ready to be put away.
PAWAY = Putting the dishes in the cupboard.
DISH = A dish to be cleaned.
CLEANER = a kid that is in the kitchen doing the dishes.
TOWEL= The towel need to be used to dry the dishes. Only one dish can be dried at a time.
SINK = The sink must be used to wash the dishes. Only one dish can be washed at a time.
toWash, toDry, goAway = the three mechanisms that are used to count how many dishes are still required to go through each part of the system.
WASHERS = all of the washers.
KITCHEN = The entire dish washing problem.
clean = The signal that means that there are dishes to be washed.
dishToWash = Indicates that a dish has begun to be washed.
startWash = Is when the sink is being used.
stopWashed = The sink is finished being used.
dishDoneWash = When a dish has exited the sink and is ready to be dried.
put = put a dish on the counter to be dried. Adds to the 'ready to be dried' queue.
empty = when there are no more dishes to be washed.
dry = indicates that a dish on the queue is ready to be dried.
dishToDry = an event that a single dish waits for to start being dried.
pickUpT = the event to pick up a towel and put it 'into use'.
putDownT = the event to put a towel down and make it free
dishDoneDry = The dish is now dried
none = the event that says there are no dishes left to dry.
dish = Removing a dish from the drying dishes pile.
dishAway = Putting an individual dish away.
done = when there are no dishes left to go away.

Alphabet: Washing Dishes

$\alpha WASH$ = { clean, dishToWash, StartWash, StopWash, dishDoneWash, put, empty }
αDRY = {dry, dishToDry, pickUpT, putDownT, dishDoneDry, none }
$\alpha PAWAY$ = { dish, dishToAway, dishAway, done }
$\alpha DISH$ = { dishToWash, dishDoneWash, dishToDry, dishDoneDry, dishAway }
$\alpha CLEANER_i$ = { WASH, DRY, PAWAY}
$\alpha TOWEL$ = {pickUpT, putDownT }
$\alpha SINK$ = {StartWash, StopWash }
$\alpha toWash_i$ = { clean, empty}
$\alpha toDry_i$ = { dry, none}
$\alpha goAway_i$ = { dish, done}

Steps of Cleaning the Dishes

$WASH = (clean \rightarrow dishToWash \rightarrow StartWash \rightarrow StopWash \rightarrow dishDoneWash \rightarrow WASH)|(empty \rightarrow SKIP)$
$DRY = (dry \rightarrow dishToDry \rightarrow pickupT \rightarrow putdownT \rightarrow dishDoneDry \rightarrow dryRack \rightarrow DRY)|(none \rightarrow SKIP)$
$PAWAY = (dish \rightarrow dishAway \rightarrow PAWAY)|(done \rightarrow SKIP)$

Synchronization Constructs

$ToWash_{(i=1toN)} = clean \rightarrow ToWash_{i-1}$
$ToWash_{(0)} = empty \rightarrow ToWash_0$

$toDry_{(i=1toN)} = (dry \rightarrow ToDry_{i-1})|(put \rightarrow ToDry_{i+1})$
$toDry_{(0)} = (none \rightarrow ToDry_0)|(put \rightarrow ToDry_1)$

$goAway_{(i=1toN,)} = dish \rightarrow goAway(i-1)|(dryRack \rightarrow goAway_{i+1})$
$goAway_0 = done \rightarrow goAway_0|dryRack \rightarrow goAway_1)$

SINK = startWash → stopWash → SINK
TOWEL = pickUpT → putDownT →TOWEL
$DISH_i = dishtowash \rightarrow dishDoneWash \rightarrow dishtodry \rightarrow dishDoneDry \rightarrow dishAway \rightarrow SKIP$

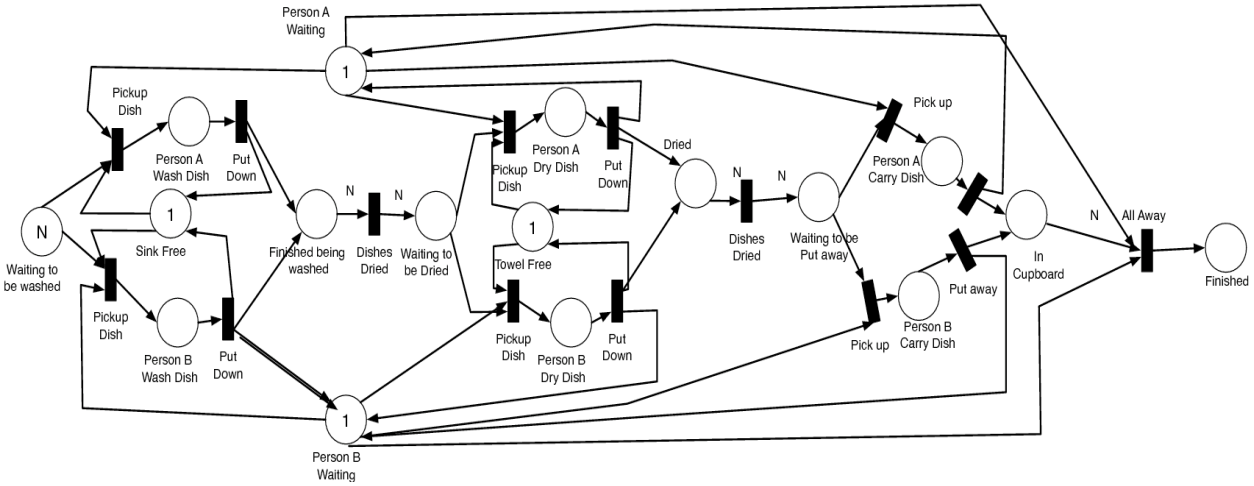


Figure A.1: PetriNet of the solution a to question 2.4

Dishwasher Solution when washers work on each task together

$Cleaner_1 = WASH; DRY; PAWAY; \surd$
$Cleaner_2 = WASH; DRY; PAWAY; \surd$
DISHES = $\{DISH_0 ||| ..... ||| DISH_N\}$
$CLEANERS = \{Cleaner_1 ||| Cleaner_2\}$

KITCHEN = {CLEANERS ‖ DISHES‖ TOWEL ‖ SINK ‖$ToWash_{(N)}$‖$ToDry_0$‖$goAway_0$}

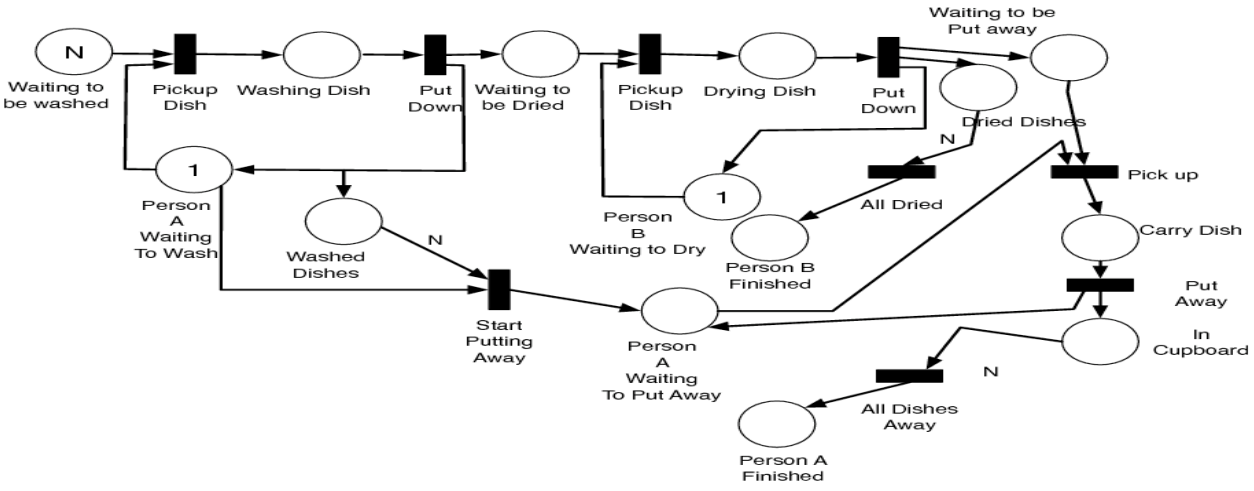


Figure A.2: PetriNet of the solution b to question 2.4

The first person Washes and puts the dishes away and the other person Dries them

$Cleaner_1 = WASH; PAWAY; \surd$
$Ceaner_2 = DRY; PAWAY; \surd$
CLEANERS = $\{CLEANER_0 ||| CLEANER_1\}$
DISHES = $\{DISH_0 ||| ..... ||| DISH_N\}$

KITCHEN = {CLEANERS ‖ TOWEL ‖ SINK ‖ DISHES ‖$ToWash_N$‖$ToDry_0$‖$goAway_0$}

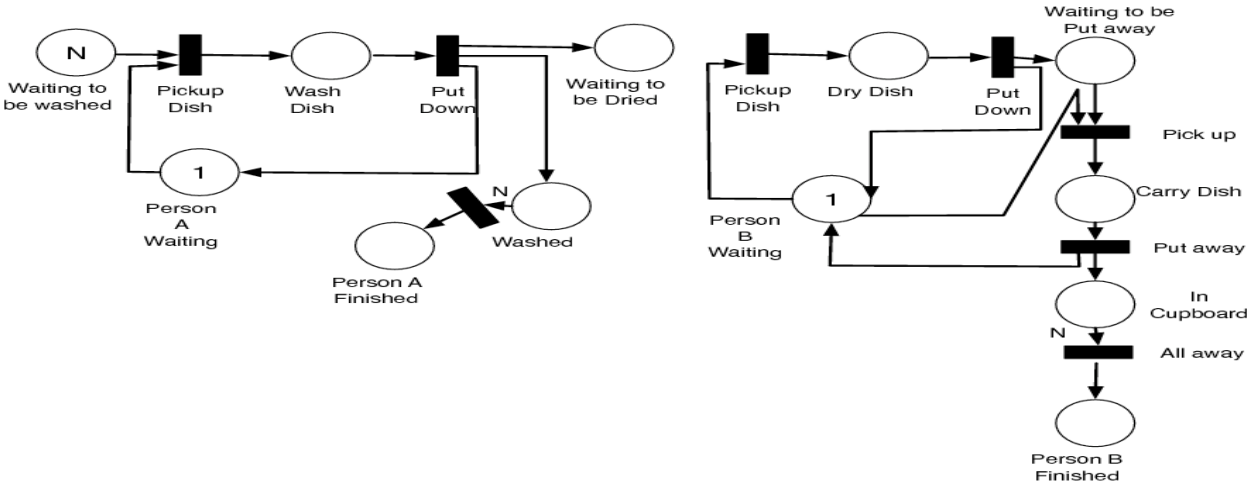


Figure A.3: PetriNet of the solution c to question 2.4

Dishwashing CSP with one person washing and the other Drying and putting away

$Cleaner_1 = WASH; \surd$
$Ceaner_2$ = {PAWAY ||| DRY }

CLEANERS = $\{CLEANER_0 ||| CLEANER_1\}$
DISHES = $\{DISH_0 ||| ..... ||| DISH_N\}$
KITCHEN = $\{CLEANERS ‖ TOWEL ‖ SINK ‖ DISHES ‖ ToWash_N ‖ ToDry_0 ‖ goAway_0\}$

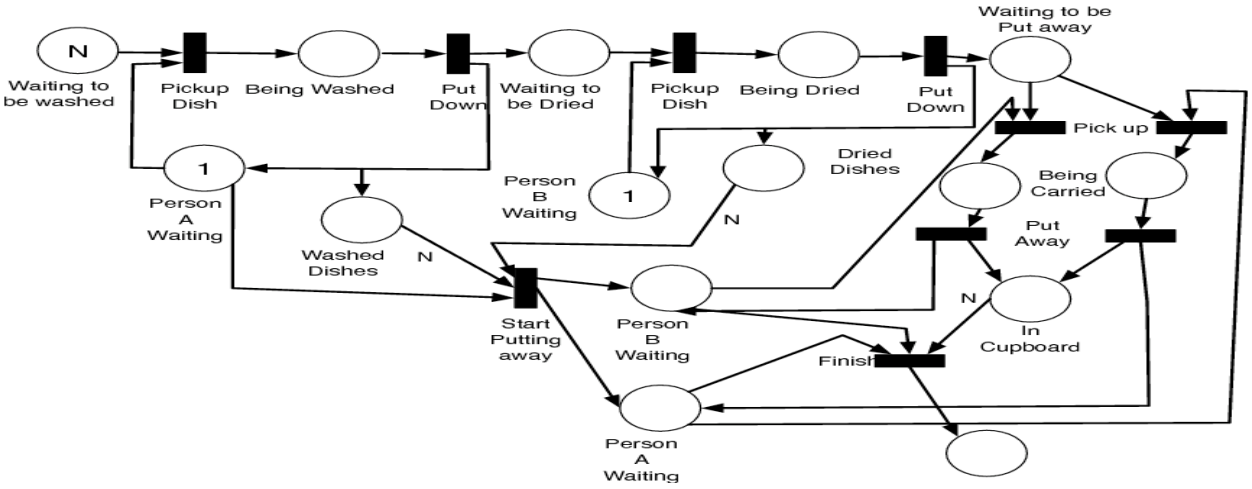


Figure A.4: Petri Net of the solution d to question 2.4

One Person Washes, the other person dries, then they both Put the dishes away

$Cleaner_1 = WASH; none \rightarrow PAWAY; \surd$
$Ceaner_2 = DRY; none \rightarrow PAWAY; \surd$

CLEANERS = $\{CLEANER_0 ||| CLEANER_1\}$
DISHES = $\{DISH_0 ||| ..... ||| DISH_N\}$

KITCHEN = {CLEANERS ‖ TOWEL ‖ SINK ‖ DISHES ‖$ToWash_N$‖$ToDry_0$‖$goAway_0$}

Appendix B

Recursion Clicker Results

Iterations	First			Second			Third		
Question	Correct	Incorrect	N	Correct	Incorrect	N	Correct	Incorrect	N
1. Is the Droste Box Recursive?	9	0	9						
2. What is recursive about the borax box?	8	1	9	5	2	7	3	2	5
3. Is Escher's Ball recursive?	5	4	9						
4. Base Case missing: what line of code?	3	4	7						
5. What is the output of a recursive function?	3	4	7						
6. When is the base case met?	2	5	7						
7. Is this procedure recursive? (non-recursive)	5	2	7	3	5	8	3	2	5
8. Is this procedure recursive? (recursive)	4	4	8	3	2	5			

Table B.1: Summary of the clicker data from the 7 weeks. The questions are in order of first occurrence in class.

Bibliography

[1] (2007) Mathgrapher—the mathematical graphing tool for students, scientists and engineers. [Online]. Available: http://www.mathgrapher.com/index.php?pagina=example_Lindenmayer

[2] (2008) Alice 3 update. [Online]. Available: http://www.alice.org/index.php?page=alice3_ progress _ report

[3] (2009) 911:geometry = wikicompany. [Online]. Available: http://www.wikicompany.org/wiki/911:Geormetry

[4] (2009) Borax box. [Online]. Available: http://piscines-apollo.com/images/borax box.jpg

[5] (2009) Borax_box.jpg. [Online]. Available: http://1.bp.blogspot.com/

[6] (2009) Cpsc 115: Course outline. [Online]. Available: http://courses.seng.engr.uvic.ca/courses/ 2009/spring/csc/115

[7] (2009) Cpsc 211: Course outline. [Online]. Available: http://www.ugrad.cs.ubc.ca/ cs211/courseInfo /courseInformation.html

[8] (2009) Lego mindstorms. [Online]. Available: http://mindstorms.lego.com [Online]

[9] (2009) Lsci::solution::microworldsex. [Online]. Available: http://www.microworlds. com/solutions/mwex.html

[10] (2009) A pattern language for parallel programming. [Online]. Available: http://www.cise.ufl.edu/research/ParallelPatterns/

[11] (2009) Scratch. [Online]. Available: http://scrath.mit.edu

[12] (2009) Starlogo tng. [Online]. Available: http://education.mit.edu/drupal/starlogo-tng

[13] (2009) Zoom into the mandelbrot set. [Online]. Available: http://video.google.com/videoplay?docid= 6460130356432628677

[14] B. Barron and R. Engle. (2009) Video research in education: Questions and guidelines for researchers and reviewers. [Online]. Available: http://drdc.uchicago.edu/what/video-research.html

[15] T. Bell. (2009) Csunplugged. [Online]. Available: www.csunplugged.org

[16] J. Briggs, *Fractals: The Patterns of Chaos*. Touchstone, 1992.

[17] P. Buneman and L. Levy, "The towers of hanoi problem," *Information Processing Letters*, vol. 10, no. 4-5, pp. 243 – 244, 1980. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0F-45GN64J-3M/2/05b2ed728c8ccbc074664fa68c466ada

[18] P. Canabarro. (2009) 50 stunning examples of the droste effect. [Online]. Available: http://www.webdesignerdepot.com/2009/09/50-stunning-examples-of-the-droste-effect/

[19] S. Carruthers, K. Gunion, and U. Stege, "Computational biology unplugged!" in *WCCCE '09: Proceedings of the 14th Western Canadian Conference on Computing Education*. New York, NY, USA: ACM, 2009, pp. 126–126.

[20] L. Carter, "Why students with and apparent aptitude for computer science don't choose to major in computer science," in *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, 2006, pp. 27–31.

[21] J. Chong. (2009) Task parallelism algorithmic strategy pattern. [Online]. Available: http://parlab.eecs.berkeley.edu/wiki/_media/patterns/paraplop_g4_3.pdf

[22] T. P. I. Company. (2009) Pico crickets. [Online]. Available: www.picocricket.com

[23] M. Conway, S. Audia, T. Burnette, D. Cosgrove, and K. Christiansen., "Alice: lessons learned from building a 3d system for novices," in *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, The Hague, The Netherlands, April 2000, pp. 486–493.

[24] S. Cooper, W. Dann, and R. Pausch, "Teaching objects-first in introductory computer science," in *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*. New York, NY, USA: ACM, 2003.

[25] R. Delisle, *How to Use Problem-Based Learning In The Classroom*. Virginia: Association for Supervision and Curriculum Development, 1997.

[26] R. L. Devaney. (2009) The sierpinski triangle. [Online]. Available: http://math.bu.edu/DYSYS/chaos-game/node2.html

[27] A. diSessa Seymour Papert, Daniel Watt and S. Weir, "Final report of the brookline LOGO project. part III: Profiles of individual student's work," MIT Artificial Intelligence Laboratory, Tech. Rep. AIM-546, Sept. 6 1979. [Online]. Available: ftp://publications.ai.mit.edu/ai-publications/500-999/AIM-546.ps; ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-546.pdf

[28] D. Dobbs. (2009) Dr.dobbs— sharing is the root of all contention. [Online]. Available: http://www.ddj.com/hpc-high-performance-computing/214100002

[29] B. S. Elenbogen and M. R. O'Kennon, "Teaching recursion using fractals in prolog," in *SIGCSE '88: Proceedings of the nineteenth SIGCSE technical symposium on Computer science education*. New York, NY, USA: ACM, 1988, pp. 263–266.

[30] D. J. Ernst and D. E. Stevenson, "Concurrent cs: preparing students for a multicore world," in *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*. New York, NY, USA: ACM, 2008, pp. 230–234.

[31] S. Fincher and M. Petre, *Computer Science Education Research*, Fincher, Ed. Taylor & Francis Ltd, 2004.

[32] G. Ford, "A framework for teaching recursion," *SIGCSE Bull.*, vol. 14, no. 2, pp. 32–39, 1982.

[33] J. Gall, M. Gall, and W. Borg, *Applying Educational Research*, N. Benevento, Ed. Toronto: Pearson Education inc., 2005.

[34] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*, B. W. Kernighan, Ed. Addison-Wesley Professional, 1995.

[35] J. Ganssle. (2009) Computer science enrollment statistics: Is the decline slowing? [Online]. Available: http://www.embedded.com/design/216200075

[36] M. T. Goodrich and R. Tamassia, *Algorithm Design: Foundations, Analysis, and Internet Examples*. Wiley, September 2001. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0471383651

[37] K. Gunion, T. Milford, and U. Stege, "Curing recursion aversion," in *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*. New York, NY, USA: ACM, 2009, pp. 124–128.

[38] B. Haberman and H. Averbuch, "The case of base cases: why are they so difficult to recognize? student difficulties with recursion," *SIGCSE Bull.*, vol. 34, no. 3, pp. 84–88, 2002.

[39] J. A. Hatch, *Doing Qualitative Research in Education Settings*. Cornell University Press, 2002.

[40] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, no. 8, pp. 666–677, 1978.

[41] J. A. Jaramillo, "Vygotsky's sociocultural theory and contributions to the development of constructivist curricula," *Education*, vol. 117, p. 1, 1996.

[42] D. H. Jonassen, J. Howland, J. Moore, and R. M. Marra, *Learning to Solve Problems with Technology: A constructivist Perspective*, 2nd ed. New Jersey: Prentice Hall, 2003.

[43] H. Kahney, "What do novice programmers know about recursion," in *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 1983, pp. 235–239.

[44] Y. Kolikant, "Learning concurrency: evolution of students' understanding of synchronization," *Internation Journal of Human Computer Studies*, vol. 60, pp. 243–268, 2004.

[45] Y. B.-D. Kolikant, M. Ben-Ari, and S. Pollack, "The anthropology semaphores," in *ITiCSE '00: Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education*, 2000, pp. 21–24.

[46] M. Kolling. (2009) Greenfoot: University of kent. [Online]. Available: www.greenfoot.org

[47] D. M. Kurland and R. D. Pea, "Children's mental models of recursive logo programs," Bank Street Coll. of Education, New York, NY. Center for Children and Technology, Tech. Rep. 10, 1985, This work was supported by a grant from the Spencer Foundation. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00190537/en/

[48] J. Levenick, "Teaching recursion before iteration," *The Computing Teacher*, pp. 12–15, 1990.

[49] G. Lewandowski, D. J. Bouver, R. McCartney, K. Saunders, and B. Simon, "Commonsense computing (episode 3): Concurrency and concert tickets," *SESSION: What are the barriers to learning computing?*, vol. 3, pp. 133–144, 2007.

[50] Lucapost. (2007) Sunflower: the fibonacci sequence, golden section. [Online]. Available: http:/www.flicker.com/photos/lucapost/694780262

[51] J. MatheMagics. (2008) Lost secrets of the phi code. [Online]. Available: http://www.jainmethemagics.com/newsletter_lookup?newsletter_id=24

[52] T. Mattson, B. Sanders, and B. Massingill. (2009) Pipelineprocessing design pattern. [Online]. Available: http://www.cise.ufl.edu/research/ParallelPatterns/PatternLanguage/AlgorithmStructure/Pip

[53] R. E. Mayer and R. Moreno, "A cognitive theory of multimedia learning: Implications for design principles," in *the annual meeting of the ACM SIGCHI Conference on Human Factors in Computing Systems*, Los Angeles, CA., 1998.

[54] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year cs students," *SIGCSE Bull.*, vol. 33, no. 4, pp. 125–180, 2001.

[55] C. A. Mertler, "Designing scoring rubrics for your classroom," *Practical assessment, research and evaluation*, vol. 7, no. 25, pp. 1–5, 2001.

[56] D. Nagel. (2009) Computer science courses on the decline – the journal. [Online]. Available: http://thejournal.com/Articles/2009/08/04/Computer-Science-Courses-on-the-Decline.aspx

[57] R. Pauche. (2008) Alice. [Online]. Available: http://www.alice.org/index

[58] ——. (2008) Alice programming language. [Online]. Available: www.alice.org

[59] W. Penfield and R. L, *Speech and Brain Mechanisms*. Princeton University Press, 1959.

[60] J. Piaget, *The Child's Conception of the World.*, J. Tomlinso, Ed. Patterson, NJ: Littlefield, Adams, 1960.

[61] K. Powers, S. Ecott, and L. M. L. M. Hirshfield, "Through the looking glass: teaching cs0 with alice," *SIGCSE Bulliten*, vol. 39, no. 1, pp. 213–217, March 2007.

[62] P. Prusinkiewicz and A. Lindenmayer, *The algorithmic beauty of plants*, D. Rand, Ed. New York, NY, USA: Springer-Verlag New York, Inc., 1990.

[63] S. Reges and M. Stepp, *Building Java Programs: A Back to Basics Approach*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2007.

[64] M. Resnick, "Multilogo: A study of children and concurrent programming," *Interactive Learning Environments*, vol. 1, no. 3, pp. 1049–4820, 1990.

[65] ——. (2008) Lifelong kindergarten group. [Online]. Available: http://llk.media.mit.edu/index.php

[66] K. A. Robbins and S. Robbins, *Unix Systems Programming: Communication Concurrency and Threads*, 5th ed., G. Doench, Ed. USA: Pearson Education LTD, 2003.

[67] J. R. Savery and T. M. Duffy, "Problem based learning: An instructional model and its constructivist framework," Indiana University, Tech. Rep., June 2001.

[68] R. Sooriamurthi, "Problems in comprehending recursion and suggested solutions," in *SIGSCE Bull.*, no. 33, 2001, pp. 25–28.

[69] W. J. Stepien, *Problem-Based Learning with the Internet: Grades 3-6.* Tucson, Arizona: Zephyr Press, 2002.

[70] D. E. D. Stevenson, "Concurrent cs: Preparing students for a multicore world," in *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, 2008, pp. 230–234.

[71] K. Touhey. (2002) The sierpinski gasket and towers of hanoi. [Online]. Available: http://www.math.ubc.ca/ cass/courses/m308-02b/projects/touhey/index.html

[72] R. Tsur. (2000) Psyart: An online journal for the psychological study of the arts. [Online]. Available: http://www.clas.ufl.edu/ipsa/journal/2000_tsur03.shtml

[73] F. Turbak, C. Royden, J. Stephan, and J. Herbst, "Teaching recursion before loops in cs1," *Journal of Computing in Small Colleges*, vol. 14, no. 4, pp. 86–101, 1999.

[74] J. Vegso. (2005) Interest in cs as a major drops among incoming freshmen. [Online]. Available: http://www.cra.org/CRN/articles/may05/vegso

[75] E. W. Weisstein. (2009) Koch snowflake. From MathWorld – A Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com/KochSnowflake.html

[76] E. W. Weisstein and P. Chandra. (2009) Fibonacci number. From MathWorld – A Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com/FibonacciNumber.html

[77] M. Wirth, "Introducing recursion by parking cars," *SIGCSE Bull.*, vol. 40, no. 4, pp. 52–55, 2008.