The Paradigm Recursion: Is It More Accessible When Introduced in Middle School?

Katherine Gunion1, Todd Milford1, and Ulrike Stege1

Abstract

Recursion is a programming paradigm as well as a problem solving strategy thought to be very challenging to grasp for university students. This article outlines a pilot study, which expands the age range of students exposed to the concept of recursion in computer science through instruction in a series of interesting and engaging activities. In this study, a small number of students (n = 9) aged 11 to 13 years, were presented with a new and unique recursion curriculum involving hands-on experiences over a seven-week period at the University of Victoria, Canada. The curriculum was comprised of a series of progressively challenging recursion activities—roughly based upon the ideas of ‘Computer Science Unplugged’ (Bell, Witten, & Fellows, 2009)—and included programming applications with MicroWorlds EX, a programming language based on LOGO. Through this engagement, an increased number of students recognized and understood the concepts covered. We hypothesize that through experiences for youth with activities such as those outlined here, the number of students who understand fundamental computer science applications and who might potentially pursue computer science in post-secondary education will increase. We hypothesis further that through an earlier encounter of “challenging” concepts the learning and understanding of those will become easier at the university level. In this paper, the curriculum, classroom experiences, preliminary, largely descriptive and qualitative results and next steps in the research are discussed.

Keywords

recursion, computer science education, kineasthetic learning activities

1University of Victoria
Computer science education is a young field that is compromised 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 Petre (2004), in their seminal text on computer science education, suggest that for the field to become solely computer science and not remain situated within these other disciplines, researchers must begin to ask questions that may only be answered through computer science. With this suggestion freshly in mind, we report here on a pilot study at the University of Victoria, Canada, that sought to find information on an appropriate and successful set of engaging hands-on lessons in recursion for a small group of students \((n = 9)\) aged 11 to 13 years.

Recursion is a mathematical concept as well as a computer science programming construct 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” (Wikipedia, 2009). Because of this ubiquity and usefulness, it is usually taught in its most basic form in the first two years of programming (Bell, Witten, & Fellows, 2009; Hsin, 2008).

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 (Reges & Stepp, 2007). As a simple example of recursion written in the programming language Java consider:

```java
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 the statement `i == 1` is true. The function outputs for input `i` set to 3:

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\(^1\) (Buneman & Levy, 1980) 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.
Recursion is also 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 is even observed in the built and natural world (Briggs, 1992). Examples from the built world include virtual advertisements such as the Borax Soap Box (Borax-Box, 2009)—a picture within a picture (a phenomenon referred to as the Droste effect). Other examples include fractals such as Koch's Snowflake (Weisstein, 2009) and Fibonacci numbers (Figure 2; Chandra & Weisstein, 2009). Examples from the natural environment are plants and trees (Figure 1(d); Prusinkiewicz & Lindenmayer, 1996), sunflowers (see Figure 1(c)), pine cones (see Figure 1(e)), and sea shells (see Figure 1(f)). As Koch's snowflake, the picture of the tree in Figure 1(d) can be created recursively by a simple L-system (Prusinkiewicz & Lindenmayer, 1996). Similar recursive descriptions exist for sunflower, pine cone, and the other examples.

**Figure 1.** Some recursive figures and puzzles.
This common occurrence of recursion within the day-to-day experiences that many people have—including middle school students such as those in this study—adds to the appeal and application of the concept. One additional advantage of the recursion paradigm is that it allows the avoidance of 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 2. Here, two different methods are demonstrated; both calculate the $i^{th}$-Fibonacci number. Figure 2(a) is the iterative version containing a for-loop, and Figure 2(b) is the recursive (loop-free) version. There are two recursive calls in Figure 2(b): both occur in the else statement: $\text{fib}(i - 1) + \text{fib}(i - 2)$. There are also two base cases, namely $i == 0$ and $i == 1$.

In this paper, we report on a recent hands-on, active approach to teaching recursion to youth. It is anticipated that the activities described here have the potential to improve students’ understanding of recursion as well as to help them gain insight into what it is that one learns when studying computer science. We also anticipate that if students are introduced to “complex” topics at a younger age, then they will not be afraid of them when they are older. If we can find ways to effectively teach traditionally difficult concepts at a younger age through active and engaging activities, the potential for better understanding of the discipline of computer science as well as for increased enrolment in computer science is a possible and most beneficial outcome.

The remainder of this paper is outlined as follows. Section 2 discusses previous work and our contributions to the area. Section 3 discusses the methods designed to teach recursion and details the lessons plans. Section 4 outlines the results of our preliminary study, followed by the evaluation of the program in Section 5. Section 6 suggests some future directions for research.

Prior Research

It is hypothesized that the declining enrollment in computer science programs at post secondary institutions since 2001 is the result of a perception of a decreased number of jobs in the industry, incorrect perceptions of what computer scientists do, and general unfamiliarity with the content of the discipline (Carter, 2006; Klein, 2006). 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 computer science is something foreign and inaccessible to ordinary individuals are more than possible to achieve. To address these concerns, we propose the idea of teaching recursion to middle school students with a focus on active, hands-on, and engaging activities.
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 (Haberman & Averbuch, 2002). Because of this combination of central focus and challenge with understanding, recursion is covered as a topic in early university—mostly in the first or second year of post secondary study. This time of instruction 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, Turbak, Royden, Stephan, and Herbst (1999) describe a successful course taught in Java using a recursion-first approach.

Regardless of when or how recursion is taught, few disagree that recursion is a difficult topic for novice computer science students to fully understand and master (Haberman & Averbuch, 2002; Sooriamurthi, 2001). Once recursion is understood it may be applied to many traditional computer science problems. Sorting, searching, or tree traversal algorithms are often programmed using a recursive divide and conquer approach. In many cases, a recursive solution appears much simpler and requires far fewer lines of code.

There is a rich literature base documenting efforts to improve the teaching of recur-
sion at the elementary level of computer science (i.e., first or second year of university). For example, Kruse suggests the use of tree diagrams for aiding in the teaching of recursion, and Ford discusses recursion as a generalization of control structures (Ford, 1982; Kruse, 1982). Early papers often describe recursion as just a programming concept instead of a larger problem solving concept occurring in arts and nature. Some of the more recent work views recursion as embedded in the larger picture of problem solving including problems in computer science, mathematics, and the arts. These works suggest adjustments to instruction, which address the embedding of recursion in larger problems. Ginat and Shifroni (1999) demonstrate that instruction in recursion emphasizing the declarative abstract level, as opposed to the concrete level, significantly improves a recursive solution formulation ability. In this case, the abstract level refers to concepts such as divide and conquer, whereas the concrete level refers to code tracing and understanding of the stack and execution. These authors further suggest that some of the challenges faced by students when learning recursion is the lack of a basic understanding of the computing model with regards to recursion. Haberman and Averbuch (2002) suggest that teachers place emphasis on both the declarative and procedural aspects of the base case. Similarly, Valázquez-Iturbide (2000) highlights the need for a distinction between the abstract concepts and the syntax (i.e., the written code).

Additionally, some researchers in the field of computer science suggest that expanding the instructional approaches toward a non-traditional but more active and inclusive paradigm is a worthwhile endeavor and that providing outreach type activities in computer science based outside of university campuses have the added benefit of appealing to non-traditional and underrepresented populations in computer science (Ford, 1982; Goode, 2008). This approach to instruction and location is followed in our study. We taught the concept of recursion at a declarative abstract level and concurrently taught the basic programming constructs involved. The intention was to have the students draw parallels between these two representations to strengthen their understanding of the fundamental concepts.

The questions we address within this study are: 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 toward computer science in general and recursion specifically?

**Methodology**

**Materials**

We focused our work on three basic types of recursion: namely 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) (Goodrich & Tamassia, 2001). 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 (i.e., the sets can be considered as glued back together).

As mentioned previously, two important parts of using recursion are the base case and recursive call. Thus, we decided these two criteria were the focus of our lessons. Additionally, we use base case and stopping condition synonymously. To design activities that teach recursion to our group of youth in grades 6 through 8 (approximately age 11 to 13) we investigated a combination of kinesthetic learning (“Unplugged”) as well as programming activities. The name “Unplugged” stems from a set of learning activities developed by Tim Bell, Ian Witten, and Michael Fellows entitled “Computer Science Unplugged” (Bell, Witten, & Fellows, 2009). These activities are hands-on group problem solving activities that teach the basics of computer science, without the direct use of technology. We developed a number of kinesthetic activities to convey the mental models needed. We further elaborated some familiar activities such as Towers of Hanoi (Buneman & Levy, 1980) and the Borax Box (BoraxBox, 2009) and then extended them to map to an educational programming language.

Activity 1: Genome Sorting. This sorting activity uses genomics as motivation. The students’ objective is to sort a number of objects in the comparison based sorting model (Goodrich & Tamassia, 2001).

Students are divided into groups of two to four. Each group is given a set of colored envelopes, with folded pieces of paper inside numbered from 1 to 14. Each envelope represents a piece of an organism’s DNA sequence, and each color represents one of the four nucleotides (i.e., Adenine, Thymine, Guanine, and Cytosine). The students are instructed to sort these envelopes 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.

While there are a few ways to perform this activity, the most efficient way is to use recursion. There are various algorithms that have been developed for sorting using
recursion; mergesort and quicksort are two examples. Both methods use a technique called divide and conquer: both algorithms recursively divide the data until the base case is met and then put the data back together. In quicksort, the sorting step is performed before the recursive call, in mergesort after.

Activity 2: Visual Recursion. Visual recursion demonstrates an image that contains itself. An excellent and common example includes the Borax Box (BoraxBox, 2009) or the Droste Box (See Figures 6(a) and 6(b)). Other good examples are the Mandelbrot set (Briggs, 1992) or Koch’s Snowflake (Weisstein, 2009). See Figures 3(a) and 1(b) for illustrations of these examples. To deepen the student’s knowledge, a more tactile approach involves turning a video camera toward the screen on which the output is being projected.

Figure 3. Some recursive figures and puzzles.

In this activity, the students are shown some examples of recursive pictures (Figure 3), and the aspects of the pictures that make them recursive are highlighted. After this discussion, the students are given a collection of pictures, some are recursive and some are simply patterns. 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.

This activity relates to recursion by attempting to have the students recognize that recursion is a picture within a picture or a picture that can be described recursively. The intention here is that students can relate the pictures to functions when they begin programming.

Activity 3: The Sierpinski Carpet. Sierpinski’s Carpet, a particular instance of visual recursion (Devaney, 1995), 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 color 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.

This activity relates to recursion because it is easily described using a recursive process. The procedure is able to be written out in a recursive format as a group and allows the students to see the difference in efficiencies between the iterative and recursive formats.

Activity 4: Line-Up. In a grocery store, cafeteria, or movie theatre line-up (queue), the number of people in front of you 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 which position they are in the line is to ask the person in front of them for their position. This question continues recursively toward the front of the line until someone discovers that there is no one in front. This person then knows that they are in front and can respond to the person behind “I am number 1.” The second person in line is then able to add one to this answer and respond to the person behind. 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 often the students’ group size is small and it is then easy to see to the front of the line, 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 neighbors.

This is the first activity where the students are able to apply the recursive procedures to a typical, real-life scenario. The recursive aspect of this function is that there is a base case, which each individual must check. If the base case is true then they will respond back to the person behind them. If they have not reached the base case, then they will ask the person in front of them (calling the recursive call), and then wait until that recursive call returns. They will use that return value to calculate their own return value and then return to the person behind them.

Activity 5: Towers of Hanoi. Towers of Hanoi is an ancient Vietnamese puzzle involving three pegs and N disks of different sizes (Buneman & Levy, 1980). 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) with a few restrictions: (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 disks moved should be as small as possible.
The inherent solution to this problem is recursive. To solve a Towers of Hanoi puzzle with 5 disks, one 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 one 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 amounts of disks and to problem solve as a group or as an individual. Afterwards, the students make their solutions concrete through verbal or written responses.

Activity 6: Matchbox Sorting. Matchbox sorting is, like genome sorting (Activity 1), a sorting activity using the comparison-based sorting model and shares many of the same instructions. Students are given varying numbers of matchboxes, with the matches removed and numbers written on the inside of the boxes. Students are not allowed to open a matchbox. 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 are then required to problem solve a solution while keeping track of the number of comparisons required.

Educational Programming Languages

There are various programming languages that are developed with children in mind. Many of these programming languages have either a simplified syntax or use drag and drop puzzle pieces that clip together to create code. Further, they focus on animation and graphics that especially appeal to a younger audience. Two popular educational programming languages that support recursion are Alice and MicroWorlds EX² (LSCI, 2009; Pausch, 2008).

Alice. Alice is a graphical programming language developed at Carnegie Mellon University (Pausch, 2008). It has a drag and drop interface that is supposed to replicate syntax similar to Java. The code animates 3-D characters on a scene or allows the students to create 3-D games and scenarios. When a student makes a new method, a new block with this name 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—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 can look very professional and have a “finished” look; however, the complex graphics can be intimidating and difficult to manipulate for beginners. When testing, often the recursion would become overbearing for the Alice system and would even crash the program.

**LOGO and MicroWorlds EX.** LOGO is a programming language that was developed in the 1960s (Papert, Watt, diSessa, & Weir, 1979). It controls a turtle on the screen. The turtle goes forward, backward, left, right, and is able to hold a pen with varying colors. There are multiple commands that manipulate these settings. LOGO’s syntax is simplified omitting semicolons and curly braces, enclosing functions with To..End surrounding the functions and square brackets enclosing logical statements. 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, the ability to have multiple turtles, and a more intricate IDE (LSCI, 2009; Figure 4). 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. There is also the ability for the students to be creative through their own characters and sounds.

After some review, we chose MicroWorlds EX 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 we wanted without the overhead of dealing with 3-D graphics. Because of this, students were free to be creative and draw their own tax and precision errors (e.g., whitespace, typos, consistent naming).

**Figure 4.** A piece of recursive MicroWorlds EX code that makes a turtle draw big squares on the screen.

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

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

In the fall of 2008, we ran a weekly after-school club in 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 neighboring schools and with staff, faculty, and students in the department, looking for a maximum group size of 15. In the end, there were 9 students, 7 girls and 2 boys. The experiments were performed with student and parent understanding and consent. Not all of the students were present for all of the weeks, and some friends 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. 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 the main questions:

1. Can the students identify recursion?
2. Can the students understand and apply recursion?

Identifying and understanding targets the students’ conceptual knowledge with respect to recursion, while the ability to apply recursion considers their procedural knowledge of recursion. Data on the additional question of “What is the effect on attitudes toward computer science?” was also collected more informally throughout these sessions. Table 1 provides the timeline for the instructional focus of the weekly session. In the first three weeks, the focus was on helping students to recognize recursion, and in the last four weeks, the focus was on helping them to understand. Both understanding and enjoyment were observed over the entire seven weeks.

Table 1. Explains the target questions for each of the weeks
Week 1. In the first week, the students completed the pre-test (Activity 1) to evaluate their current understanding of the material. We were interested in the strategies the students would use to solve this novel problem. This activity was designed to analyze whether students understood and could apply recursion prior to having been taught the concept. The pre-test consisted of the genome sorting activity described previously. 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 test to see if they are able to identify it in other examples. We first showed them a few images that contained an image within an image and a video of the Mandelbrot Set where a part of the Mandelbrot picture is selected and zoomed in, and that process is repeated for the duration of the movie. We then handed out pictures, some being examples of visual recursion, and others were not recursive. After the students voted, we discussed through these examples the requirements of a recursive image.

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

Week 3. Our goal in this week was to highlight problem solving with 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.

In the programming activity, having been introduced to function calls and parameters the week before, the students were now introduced to 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 are faced with describing an instance of Sierpinski’s Carpet, if it is understood successfully, they recognize the recursion and attempt to describe the image using a recursive technique. This would demonstrate their understanding in applying 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 turtle chase another turtle until it caught and hugged it. The function that they were to create stated that the chasing turtle would move toward the other turtle until it reached the base case of touching the turtle. At this point, the chaser would hug the other animal.

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.

Week 5. This week’s goal was to give the students an interesting task encompassing the different topics introduced up to this point. The solution to the Towers of Hanoi puzzle (Activity 5) 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, and 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 5).

This week the students were able to demonstrate their level of enjoyment for the activity, their understanding of recursion, and their ability to apply the concept.

Figure 5. One student’s screen of her MicroWorlds EX code and graphics for Towers of Hanoi.
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 (matchbox sorting). This test was designed to answer the question, whether or not the students are able to apply the concept of recursion.

Results

In efforts to triangulate our findings, we decided to use various methods of data collection to help us determine the students' understanding (i.e., video recording, audio recording, note taking, and clicker questions). The clicker system used was i-clicker. Clickers are remote controls with buttons from A to E to 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 and connected to a computer records the students' answers. Most weeks, we collected data on between four and eight clicker questions about recursion, MicroWorlds EX, and general understanding of the activities thus far.

Clicker Data

Throughout the seven weeks of this study, the students were presented with different scenarios (such as images and MicroWorlds EX programming code) and questions (such as “is the image you see recursive?” and “when is the base case met?”). For each question, up to 5 choices (i.e., A to E) of possible answers were presented to be selected by using the clicker. Table 2 illustrates the different questions that were asked. Some questions were asked up to three times. Note that questions 1 and 3 as well as 7 and 8 are the same. However, for 1 and 3 the presented images were different with one being recursive and the other one not. The same was done for the questions 7 and 8.

The questions are grouped under two categories: visual recursion (questions 1 to 3) and coding questions (questions 4 to 8).

Visual Recursion. The first three groups were surveyed in Week 2 after the concept of visual recursion had been explored. The first three questions (see below) also had accompanying images (Figure 6). The Borax box question was then revisited in both Week 5 and Week 7.

Question 1. Is the Droste Box Recursive? (Figure 6(a))
A) Yes
B) No
Table 2. Summary of the i>clicker data.

<table>
<thead>
<tr>
<th>Questions</th>
<th>1st Iteration</th>
<th>2nd Iteration</th>
<th>3rd Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C*</td>
<td>I**</td>
<td>n</td>
</tr>
<tr>
<td>1. Is the image recursive? (recursive image presented)</td>
<td>9</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>2. What is recursive about the Borax Box?</td>
<td>8</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>3. Is the image recursive? (non-recursive image presented)</td>
<td>5</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4. Base case missing – what line of code?</td>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>5. What is the output of a recursive function?</td>
<td>3</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>6. When is the base case met?</td>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>7. Is this procedure recursive? (non-recursive procedure presented)</td>
<td>5</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>8. Is this procedure recursive? (recursive procedure presented)</td>
<td>4</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

*Correct **Incorrect

Figure 6. Images used for Questions 1–3..

(a) Droste Box  (b) Borax Box  (c) Escher: Hand With Reflecting Sphere
Question 2. What is Recursive about the Borax Box? (Figure 6(b))
A) Horses
B) Lettering
C) Not recursive
D) The Box

Question 3. Is the Hand With Reflecting Sphere Picture Recursive? (Figure 6(c))
A) Yes
B) No

Coding Questions. In the remaining questions, the students were shown pieces of MicroWorlds EX 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. 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 EX code.
A) repeat 5
B) if (i < 5)
C) (the function call) doSquares 5

Question 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

In the final two categories, the students were shown full procedures that were either recursive, or would use looping.

Question 6. 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 three different images and they were required to choose which one was the proper output. The success on this last question was minimal. We note, however, that this question was a very difficult one requiring the students to understand head versus tail recursion.

Question 7. In the non-recursive procedures, the students would see repeat or forever loops. They were then given the binary choice “Yes” this is recursive or “No” it is not.

Question 8. In the recursive procedures, the students would see procedures that were clearly recursive with a recursive call and a base case.
Written Responses

Another form of data collection we utilized in this study was written responses. These responses are also organized in order of weeks collected. The students were not required to give a written response each week; therefore, not every week is represented in this section. Table 3 outlines the questions covered.

Table 3. The written questions that were asked throughout the workshop.

<table>
<thead>
<tr>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1 Is a certain picture recursive?</td>
</tr>
<tr>
<td>Question 2 From all of the pictures we have just looked at, how</td>
</tr>
<tr>
<td>would you describe recursion?</td>
</tr>
<tr>
<td>Question 3 Drawing the output of a function.</td>
</tr>
<tr>
<td>Question 4 After studying recursion, what would you say it is? What</td>
</tr>
<tr>
<td>is important about it?</td>
</tr>
</tbody>
</table>

Table 4. Results of voting on whether images were recursive or not.

<table>
<thead>
<tr>
<th>Image</th>
<th>Correct</th>
<th>Incorrect</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Mandelbrot Set</td>
<td>(Fig 3(a))</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2. Mona Lisa</td>
<td>(Fig 3(b))</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>3. Spoof</td>
<td>(Fig 3(c))</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>4. Circle Limit</td>
<td>(Fig 1(g))</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5. Division of Plane</td>
<td>(Fig 3(d))</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6. Koch’s Snowflake</td>
<td>(Fig 1(b))</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Question 1. In the second week, students were asked to vote on whether they thought presented images were recursive. 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 4 shows the students’ perceptions of each of the images. This data is also provided in Figure 7.

Question 2. In the second week, we asked the question: “From all of the pictures we just looked at, how would you describe recursion?” Typical student 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.”
**Figure 7.** Descriptive results from written questions.

![Bar chart showing correct and incorrect responses for different code chunks.](image)

**Figure 8.** Students were given these three chunks of code and asked to draw one of them.

(a) Recursive

```plaintext
to Squares :size
  ifelse (:size < 3)
    [ drawSquare :size
    [ repeat :howmany
      [ drawSquare :size
        Squares (:size / 2)
      ]
    ]
  end
end
```

(b) Iterative

```plaintext
to Squares :size
  make "" :howmany (:size / 6 )
  drawSquare 24
  make "" :howmany (:size / 6 )
  drawSquare 12
  make "" :howmany (:size / 6 )
  drawSquare 6
  make "" :howmany (:size / 6 )
  drawSquare 3
  make "" :howmany (:size / 6 )
  drawSquare 2
end
```

(c) Function Calls

```plaintext
to Squares :size
  make "" :howmany (:size / 6 )
  drawSquare 24
  drawSquare 12
  drawSquare 6
  drawSquare 3
  drawSquare 2
end
```
Question 3. In Week 7, we asked the students to draw one of three procedures written in MicroWorlds EX given to them on handouts (Figure 8). The students got to choose which procedure they drew and were instructed to indicate which they chose (i.e., recursive in Figure 8(a) and iterative in Figure 8(b)). The first procedure was recursive (Figure 8(a)), and the second used a repeat loop (Figure 8(b)). The third code segment (Figure 8(c)) consists of five function calls with the value of the parameter decreasing. 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. The students were reminded of what the drawSquare procedure did and had seen it multiple times throughout the workshops. Five students handed in this activity (six students were present that day). Of the five students, four drew correct images. One student circled the recursive procedure, one student circled the iterative option, and the final two circled the option “Function Calls.”

Question 4. At the end of the seven-week session, we asked the students to write out answers to 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 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 did “repetition.”

Video and Audio Data

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 (2005), 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 1. 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 came up with a correct algorithm (not necessarily the most efficient). None of their suggested strategies for solving this task was recursive.

Week 2. The students were very engaged with the visual recursion activity in Week 2
and were actively debating amongst one another about whether the pictures we handed out were recursive. The students successfully identified the recursion in the presented pictures with the Droste effect, but became unsure when Koch’s Snowflake and the Mandelbrot Set were presented. After debating amongst themselves, they could not come to an agreement. When asked what aspect of the Borax Box was recursive, one student—who defended that the lettering was non-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 to this question is that the image is non-recursive. One student felt that it was recursive because “The ball is a reflection.” Another incorrect statement: “It re-occurs 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 that “If it was recursive it would show the hand with the ball inside the ball.”

Week 3. 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 was primarily focused on the question of recognition and if students would be 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 color in certain squares black.

A follow up discussion—to consolidate learning among all the groups in the class—involves 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 is 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.”

There is evidence provided amongst the group that re-engaging with the Sierpinski Carpet activity and connecting this learning to the previous week’s exercise helped them to recognize the recursive elements involved. The students did not employ recursion to help their peers recreate the Sierpinski Carpet, and later they had difficulty recognizing the “procedure within a procedure” aspects of their group direction. However, 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 4.** The fourth week focused on a MicroWorlds EX 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 viewpoint 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).

The first piece of code that follows shows a possible approach using tail recursion.

```
Algorithm AnimalChase (bird)
if in huggable distance of the bird then hug
else
  move a few steps towards the bird
  AnimalChase (bird)
Next we see a piece of code that shows a possible iterative approach.
Algorithm AnimalChase (bird)
while (the person is) not in huggable distance of the bird do
  move a few steps towards the bird
  hug the bird.
end
```

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 and then on developing a strategy in MicroWorlds EX that was based upon the discussed algorithm. 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 guidance) the stopping condition or base case of a while loop or recursion, respectively. Namely, the person will hug the bird only when (s)he is near the bird. An easier task for the group was to identify the repeated element, namely to move toward the bird if no hug is possible. When asked how to program the procedure, a participant suggests a loop and thus the iterative process.

The instructor next used the board to write a simple recursive procedure, seen the week before (draw squares inside a square). They then used targeted questions to draw parallels between the two tasks. The students did not independently suggest any discussion on the recursive element in either last week’s procedure or in the current one. However, the AnimalChase algorithm was not solved as this was transferred to a task in MicroWorlds EX.

Although students did not actively develop a recursive strategy for the problem, the programming part demonstrated evidence that the students could apply recursion to realize the task as they all (with different amount of support) were able to write their
Figure 9. One student’s code for the hug activity with two characters.

```plaintext
to flee
setc "yellow
seth random (360)
repeat 30
fd 4321
wait 1
]
end
```

```plaintext
to chaseAndHug
setc "red
seth "cat1
ifelse (distance "tu:
[
seth "cat3
stopall
]
[
towards "turtle
fd 1234
wait 2
chaseAndHug
]
end
```

Figure 10. One student’s code for the hug activity with three characters.

recursive AnimalChase procedure. Figures 9 and 10 provide evidence of student code.

Week 5. 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 to solve the problem and what they had found to be their optimal solution. We observed that by the end of the 30 minutes, most of them had a recursive solution for 4 disks. We remark that the whole group remained enthusiastic and on task for the entire 30 minutes.

**Figure 11.** One of the student’s interpretations of how to solve the Towers of Hanoi.

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 (Figure 11). 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 program using MicroWorlds EX (Figure 5). At the end of the session with some help from the instructors, four student codes worked completely, one required some simple syntax corrections, (white space and spelling) and two were incorrect.

**Week 7.** In the culminating week of this series of activities, the students were presented with a matchbox sorting activity—based upon the same principals as the genome sorting activity in lesson one. 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 understanding to the novel matchbox problem and demonstrate their understanding of recursion.

The instructor presented the initial problem and then students were tasked with solving the problem. The first student suggestions revolved around splitting the matchboxes up between bigger and smaller ones. The students were to ask the instructor to tell them
which was bigger between select pairs of matchboxes. Then the students grouped them 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 matchboxes in the bigger pile contained larger values 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 matchboxes in the set” and another suggesting that we restart.

Directions and 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 matchboxes to this one would indicate the relative positioning. After two iterations of this 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 to apply this procedure.

In the end, they were able to apply the procedure to all the matchbox pairings and successfully solve the problem. When asked if this was recursion—as one had mentioned—they had to be reminded just what recursion is. After this, one student (the same one who demonstrated the procedure that was used to solve the problem) clearly articulated how his procedure was in fact recursion, the base case being the moment 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 effort by the students, although recursive, was unsuccessful at solving the problem; however, they then reengaged—and with help by a particular student—were able to recursively solve the problem using quicksort.
(i.e., clickers, written answers, and video and audio recordings), results point to middle school 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 toward computer science and recursion? These questions are further explored below.

With regards to the first question (recognizing recursion; conceptual knowledge), 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 the Borax Box question incorrectly. When asked why, she defended herself, saying that if the picture was recursive, then all aspects of the picture have to be recursive as well (including the lettering). This would suggest that this wrong answer does not indicate misunderstanding. 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 Sierpinski Carpet), they did use recursion later in the sessions (e.g., matchbox 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 recognition of the problem.

In support of the second question (understanding or applying recursion), we present data from two activities. The first involved the Towers of Hanoi puzzle where one student was able to articulate the algorithm for her solution even on the white board, thus demonstrating conceptual and procedural knowledge. With the guidance of one facilitator, the group developed the MicroWorlds EX code with the recursive calls and the base case (see Figure 5). The other instance involves a different student and the matchbox sorting activity. After the groups’ initial dead-end solution (a recursive one, we might add), the student was able to independently demonstrate a recursive procedure that solved the problem. The only thing lacking was the unaided articulation of the base case and this student actually verbalizing that this was in fact recursive. These two scenarios offer some evidence that certain students were able to demonstrate their understanding (conceptual knowledge) of recursion by applying it to new situations (procedural knowledge).

The third research question explored the students’ enthusiasm and enjoyment toward 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 was a very positive outcome from the researchers’ point of view.

Implications for Pedagogy

The implications here are potentially useful to middle school teachers and parents as well as instructors in computer science at the undergraduate level. There is a widespread agreement that recursion is a challenging topic for students beginning their studies in computer science. The literature is full of attempts to improve student success through a variety of teaching tools and approaches (e.g., Haberman & Averbuch, 2002; Sooriamurthi, 2001). However, if we can successfully expose students to recursion at the middle school level, long before they find themselves in first year computer science, we may ease the challenge that both students and instructors face when covering this topic. Additionally, if after being introduced to concepts like recursion—in ways that foster enjoyment and interest—we may soon find more undergraduates in studying computer science at universities. This outcome is one that would benefit the students, the university, and the larger society.

Limitations

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. First, 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 subject 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.
Conclusion

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. Here we outline these activities and how a group of nine middle school students (age 11 to 13) reacted to them. We recognize that this is a small number; 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 toward 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 of students understanding and application of recursion. 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 toward this change. We look forward to continuing our research and to future studies that replicate and expand upon our findings.

Future Directions

After presenting a small group of youth in an after-school club setting with activities that comprised the recursion paradigm, we are encouraged. The success of this workshop has motivated us to extend our preliminary study and take the activities to a broader audience of middle school students on Vancouver Island. This will allow us to better investigate the capabilities of a more typical and more representative population to learn and understand the concept of recursion at its most basic level. Additionally, we are planning further analysis of the relation between conceptual and procedural knowledge of recursion through this next study. We are currently planning these expanded investigations for the fall term of 2009 to coincide with the beginning of a new school year in Canada.

However, during this 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. Only then would we continue by 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 toward computer science change significantly after these same activities; and does there exist a relationship between attitudes toward computer science and enjoyment and success in these activities? The design will follow the same pre-test and post-test design. Additionally, students’ attitudes toward 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 toward computer science and outcomes that are a result of involvement in these activities. It is anticipated that we can demonstrate significant effects on knowledge gain of recursion and attitude improvement toward and appreciation of computer science. After all, this sample represents the population who will become the computer scientists of tomorrow.

Endnotes

1. This famous puzzle is depicted in Figure 1(a) and described in Activity 5.
2. There are many other languages, some support recursion, others do not. Alice and MicroWorlds EX are discussed here simply because they are the languages with which the authors had the most experience.
3. SPARCS after-school club at the University of Victoria.
4. In the Sierpinski’s Carpet activity, the students may face a more difficult challenge to identify 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.
6. We were told that after this week (it was just after Halloween) one of the students told her mother that next year she would like to make a “recurjik” pumpkin, with little pumpkins inside big pumpkins.
7. Note that this together with the recursive methods initiated above results in quick-sort.

References


Velázquez-Iturbide, J. (2000). Recursion in gradual steps (is recursion really that difficult?). In proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education (pp. 310-314). Austin, Texas.


Acknowledgments

Funding for this article was made available through an NSERC grant via Pacific CRYSTAL, University of Victoria.

Paper submitted on October 7, 2009.
The final version accepted on October 24, 2009.