Modelling Recursion

by

Mojtaba Ammari-Allahyari

A thesis submitted for the degree of Doctor of Philosophy in Mathematics Education

University of Warwick, Institute of Education
March 2008

In memory of my Grandfather,

A great man who taught me the language of nature!

Declaration

I declare that this thesis was carried on in accordance with the regulations of the University of Warwick. This research is my own work and it has not been submitted for any other degree.

This thesis has not been presented to any other University for examination either in the United Kingdom or overseas.

Signed: Mojtaba Ammari-Allahyari Date: 12th July 2010

Acknowledgment

First and foremost I would like to acknowledge the debt I owe to Professor Dave Pratt who, as both friend and supervisor, gave me the encouragement and direction to construct a thesis. The outcome owes much to his inspiration. Thanks also to Dr Peter Johnston-Wilder and the members and my colleagues at Warwick Institute of Education. I am grateful for your help and concern.

I am grateful for the bursary that enabled me to embark on this PhD. It was awarded by the National office of “World Mathematical Year 2000”, which was assigned by the International Mathematical Union (*IMU*) and supported by *UNESCO* in Iran and *MSRT* (Ministry of Science, Research, & Technology of Iran).

Thanks to all the students who worked with me.

Thanks also to my dear friends for their support during my journey towards this PhD, in particular Miss Katia Merine and Mr Reza Shahrjerdi.

Special thanks to my mother, father, and my sister without whose continual support and assurance I would not have been able to finish this journey.

Finally, a short acknowledgement in Persian, my mother tongue:

بنام خالق هستی، مبدع بزرگ و به وجود آورنده ازلی و ابدی دانش

تشکر میکنم از ستاد ملی سال جهانی ریاضیات وابسته به دفتر یونسکودر ایران و وزارت علوم،

تحقیقات و فناوری ایران که با برگزاری کنکوراعزام دوره دکترای آموزش ریاضی در ایران به

پیشرفت این دانش در ایران کمک نمودند.

Table of Contents

Table of Figures

Table of Tables

Table of Programs

Table of Flowcharts

Abstract

The purpose of my research is to examine and explore the ways that undergraduate students understand the concept of recursion. In order to do this, I have designed computer-based software, which provides students with a virtual and interactive environment where they can explore the concept of recursion, and demonstrate and develop their knowledge of recursion through active engagement. I have designed this computer-based software environment with the aim of investigating how students think about recursion. My approach is to design digital tools to facilitate students' understanding of recursion and to expose that thinking.

My research investigates students' understanding of the hidden layers and inherent complexity of recursion, including how they apply it within relevant contexts. The software design embedded the idea of functional abstraction around two basic principles of: '*functioning*' and '*functionality*'. The functionality principle focuses on *what* recursion achieve, and the functioning dimension concerns *how* recursion is operationalised. I wanted to answer the following crucial question: *How does the recursive thinking of university students evolve through using carefully designed digital tools?*

In the process of exploring this main question, other questions emerged:

1. Do students understand the difference between recursion and iteration?
2. How is tail and embedded recursion understood by the students?
3. To what extent does prior knowledge of the concept of iteration

influence students’ understanding of tail and embedded recursion?

4. Why is it important to have a clear understanding of the control passing mechanisms in order to understand recursion?
5. What is the role of functional abstraction in both, the design of computer-based tools and the students’ understanding of recursion?
6. How are students’ mental models of recursion shaped by their engagement with computer-based tools?

From a functional abstraction point of view almost all previous research into the concept of recursion has focused on the functionality dimension. Typically, it has focused on procedures for the calculation of the factorial of a natural number, and students were tested to see if they are able to work out the values of the a function recursively (Wiedenbeck, 1988; Anazi and Uesato, 1982) or if they are able to recognize a recursive structure (Sooriamurthi, 2001; Kurland and Pea, 1985). Also, I invented the *Animative Visualisation in the Domain of Abstraction* (AVDA) which combines the functioning and functionality principles regarding the concept of recursion. In the AVDA environment, students are given the opportunity to explore the hidden layers and the complicated behaviour of the control passing mechanisms of the concept of recursion.

In addition, most of the textbooks in mathematics and computer sciences usually fail to explain how to use recursion to solve a problem. Although it is also true that text books do not typically explain how to use iteration to solve problems, students are able to draw on to facilitate solving iterative problems (Pirolli *et al*, 1988).

My approach is inspired by how recursion can be found in everyday life and in real world phenomena, such as fractal-shaped objects like trees and spirals.

This research strictly adheres to a *Design Based Research methodology* (DBR), which is founded on the principle of the cycle of designing, testing (observing the students' experiments with the design), analysing, and modifying (Barab and Squire, 2004; Cobb and diSessa, 2003). My study was implemented throughout three iterations. The results showed that in the AVDA (Animative Visualisation in the Domain of Abstraction) environment students' thinking about the concept of recursion changed significantly. In the AVDA environment they were able to see and experience the complicated control passing mechanism of the tail and embedded recursion, referred to a *delegatory control passing*. This complicated control passing mechanism is a kind of generalization of flow in the iterative procedures, which is discussed later in the thesis.

My results show that, to model a spiral, students prefer to use iterative techniques, rather than tail recursion. The AVDA environment helped students to appreciate the delegatory control passing for tail recursive procedures. However, they still demonstrated difficulties in understanding embedded recursive procedures in modelling binary and ternary trees, particularly regarding the transition of flow between recursive calls.

Based on the results of my research, I have devised a model of the evolution of students' mental model of recursion which I have called – *the quasi-pyramid model*. This model was derived from applying functional abstraction including both functionality and functioning principles. Pedagogic implications are discussed. For example, the teaching of recursion might adopt 'animative' visualization, which is of vitally important for students' understanding of latent layers of recursion.

1. Introduction

1.1. Overview

This chapter begins with an itinerary perspective, which explains the history of how I became interested in undertaking research into the concept of recursion. It then shows the way that the initial ideas of this research program were shaped, and offers an action plan. This chapter also looks back upon my previous experiences as a mathematics student – during the time that I was completing my B.Sc. and M.Sc. in mathematics – as well as my many years of teaching mathematical courses at university level. The primitive ideas available during those years were mainly environs of how we can use graphical presentation and related electronic gadgets to develop our learning skills as students. In other words, to what extent and to what degree would we be able to use, computers and digital gadgets to facilitate thinking about and understanding of mathematical concepts?

My own personal statements here will be followed by a more pedagogical and epistemological point of view relating to the concept of recursion.

1.2. My story toward recursion

Everything started by seeing a picture...

My journey into the world of recursion was triggered by a book called *Chaos and Fractals* by Peitgen, Jurgens and Saupe (1992). At the time I had just finished my M.Sc. in Pure Mathematics. There was a fascinating image on the front cover of the book that attracted me to delve further into it.

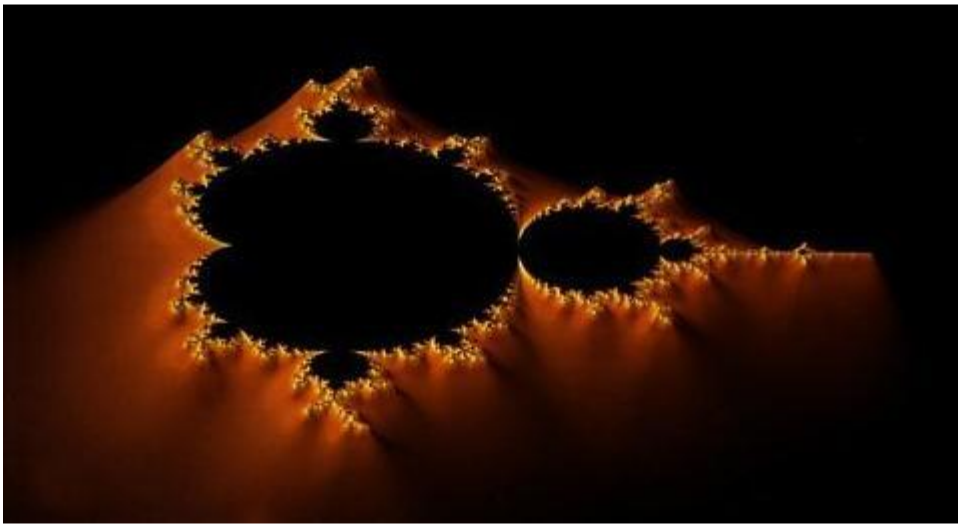


Figure 1- A beautiful image of the Mandelbrot set on the front cover of *Chaos and Fractals*

Later, I was to realise that the image is called the *Mandelbrot set* after Mandelbrot, who first introduced them in 1974. Whilst skimming quickly over the content I noticed that a very serious and strong type of mathematics was introduced in a simple and tangible style. When I was studying mathematics during my first degree, and then later during my M.Sc., I always believed that using computers and digital tools would help me to have a lucid intuition about the concepts which were being studied. For instance, as a student on a calculus course I noticed that those calculus books which contained graphical images to illustrate the limits, integrals, derivatives, etc. played a significant role in the accretion of my ability in meaning-making and understanding concepts.

This realisation persuaded me to consider applying computer and digital tools and using fractals to facilitate the teaching and learning of mathematical concepts.

There were two reasons behind this idea, the first being that computers can act as a facilitator for constructing new knowledge by providing tangible presentation in an interactive environment. The second was based on using fractals as convincing and persuading objects for students to work with mathematical concepts before knowing them. Moreover, my teaching experiments with university level students convinced me that using computers and digital tools could play a significant role in increasing the students' ability to learn mathematical concepts. For example, when they were studying calculus, some mathematical software packages like *MAPLE* were very helpful to give them an appropriate intuition of the functions when they wanted to calculate limits, integrals or derivatives.

When I encountered these fascinating and wonderful fractal images, it became apparent to me that only an elementary knowledge about mathematics was needed to be able to work with them. I believed that they might attract students because of their amazing beauty and could have the potential to persuade students to work with mathematical objects. However, this raised an important question, *how* can we use these potentials in the world of mathematics.

In considering the solution to the above question I noticed that fractals strongly depend on the concept of *recursion*. In fact, they can only be defined recursively. So, if you want students to use fractals as a facilitator to learning other mathematical concepts, perhaps you need to make sure that they have no problem understanding them at the first stage. Therefore, my focus of attention switched to the concept of recursion and investigating how students understand this concept. At first glance, it seems that the concept of recursion has no problematic part, but I soon realised that the complicated mechanism of the concept is not recognised easily by students. The next section concentrates on the pedagogical issues related to the concept of recursion.

1.3. Pedagogical account

The concept of recursion is one of the subjects, which is fundamentally difficult in mathematics and related disciplines. The reasons for this difficulty can be seen from two perspectives. The first is the lack of analogies in everyday experiments and intuitions of the concept of recursion. The second is rooted in its inherent complicated mechanism of recursion. The gap between formal mathematical concepts and everyday life analogies is of great importance from a pedagogical perspective. It is particularly visible for the concept of recursion as a consequence of the aforementioned problems with the concept of recursion. Finding links and bridges between formal explanations of recursion and informal analogies in everyday life seems to be a difficult task.

It is also hard to find a definition of the concept of recursion which is acceptable for most researchers and authors in text books. In addition, traditional teaching methods of recursion, teaching it with emphasis on examples, rather than explaining its main components, also make it a difficult concept for the students to learn.

If we argue that the difficulty of the concept of recursion is due to its inherently complicated mechanism, which underpins the challenges of mathematical pedagogy, the next important issue is which specific meanings we would like students to acquire. Perhaps these difficulties will be significantly reduced if we can find an appropriate way to uncover the roots of these difficulties. This might be considered as a starting point in employing the digital tools and computers to reveal the hidden layers of complexity of the concept of recursion. The next section of this introduction will describe the epistemological issues of the concept of recursion.

1.4. Epistemological issues

From an epistemological point of view, the concept of recursion lies in the intersection of different disciplines. Basically, it is widely used in mathematics and computer sciences. The concept of recursion is considered as an interdisciplinary concept. The first danger for interdisciplinary concepts is the challenge for a comprehensive definition. Lack of any commonly accepted definition of recursion for each of the disciplines is one of the biggest epistemological challenges for interdisciplinary concepts like recursion. In particular, the lack of a clear and comprehensive definition of the concept of recursion, which distinguishes the characteristics of mathematical and computer sciences, is a major difficulty in introducing this concept in both disciplines.

In both mathematical and computer science disciplines, recursion is usually defined in a stereotypical way. Most mathematical text books define recursion by introducing a recursive sequence or function. Similarly, the computer sciences mainly define it by presenting the classical example of a factorial of a natural number. A lack of a clear definition in both disciplines is noticeable. The nature of recursion, and its components in mathematical topics, is also different from computer sciences. Therefore, it seems that each of these disciplines declares its own epistemology for the concept of recursion. On the one hand, mathematics (a *conceptual* view) regards recursion as a function[1] which is applied within its own definition. This application has an end point if and only if the function has a limit. On the other hand, computer sciences (a *procedural* view) consider recursion as a procedure which starts with an initial program and uses itself as its sub-procedure. And the calling process will be terminated when the procedure meets its *stopping condition*. The starting and terminating processes in mathematics and computer sciences are different and they need distinct epistemological perspectives for the concept of recursion.

Yet, at the higher and more advanced level of mathematics, we find more exact formal and symbolic definitions of recursion for one and more independent variable functions[2]. In both procedural and conceptual views, recursion is defined by introducing two main components, base case(s) and recursive call(s). There is a subtle difference between these components in the mathematical and computer science disciplines. The base case in mathematics

[1] The term function in this sentence is used as a mathematical concept.

[2] Bloch (2003) is a good reference for more information about higher level mathematics and recursion.

is mainly considered as the *starting point* or *initial value*. However, in computer sciences, it is actually the *stopping condition*, and without having it the recursive procedure will never stop and the result would be an infinite loop. For instance, the Fibonacci sequence is a recursive function, starting with two initial values (base cases):

$$a_1 = 1, a_2 = 1$$

The recursive part of the Fibonacci sequence is:

$$a_n = a_{(n-1)} + a_{(n-2)} \text{ for any } n \geq 3.$$

Most of the text books in both mathematics and computer sciences define and introduce the concept of recursion by the providing examples of recursive functions or procedures rather than a certain and clear definition of the concept and its components.

The main focus in this thesis is the procedural view of the concept of recursion and the reason is again rooted in my interest in employing fractals in mathematics education. From this perspective recursion is considered to have two main parts, the base case and the recursive call. The base case in this approach operates as the stopping of the procedure and the procedure will be terminated when all the recursive calls as well as the original procedure meet the base case.

The following section will concentrate on the research themes of this study.

1.5. Research Themes

This study addresses the themes presented in this chapter. Focus is placed on the design of an environment in which students can articulate their own knowledge of the concept of recursion by active engagement with computer–based tools. In this context, computer-based tools are defined as, using computers and designing dynamic environments or situations to provide students with an interactive window[1] with which to build their own new knowledge based on their previous quasi-shaped knowledge. By quasi-shaped knowledge I refer to the previously gained or understood and accepted information about the concept which is being learnt, which obviously might not be true. The design process of computer–based tools and digital gadgets in this study is based principally on Papert's idea of computer-based tools and their potential for presenting mathematical concepts, that is, recursion in the case of this study.

My interest in fractals and using them in mathematics education led me to use fractal-shaped objects such as trees and spirals for this research. These objects were also chosen for two more reasons. Firstly, they can only be defined recursively. Secondly, they can be used to bridge formal and informal mathematics. My experiences in teaching mathematical courses at university level revealed a huge gap between the formal mathematical concepts and the informal mathematics that students were using in their everyday lives. Nunez *et*

[1] I used this term as Prof. Pratt has used it in 1989 "as a metaphor to describe the way in which the computer screen offers insights" into the student's meaning-making as they use the tools. In this study these computer-based environments are called domain f abstraction for abstracting the concept of recursion by some on screen objects.

al, (1993) term this “street mathematics versus school mathematics”. Thus I decided to use trees and spirals as everyday examples of recursion to phenomenalize the concept of recursion and bridge the gap between formal and informal mathematics. The term *phenomenalize* is adopted from Pratt (1998) and by that I mean the explanation of the students’ understanding of the contextualisation of a concept within a computer-based environment. Research supporting this view will be presented in the next chapter when I review the literature related to the concept of recursion.

The main conjecture that is examined in this PhD research program is that students can become aware of the hidden layers and the inherent complexities of the concept of recursion through active engagement with interactive computer–based tools. In other words, *how does the recursive thinking of university students evolve through the use of carefully designed digital tools? And what is the role of computer-based tools in this thinking evolution?*
Therefore, the general approach of the study will be to design the computer–based tools as a window to find out how students think about recursive procedures and their indispensable components. Having set out the above issues concerning students’ perceptions of the concept of recursion, I also intend to investigate how their perceptions are shaped and formed by the tools that were designed and provided for them.

1.6. Structure of the Thesis

This section explains the structure of the thesis. Chapter One provides a brief overview. Chapter Two reviews the previous research in the domain of recursion in further detail. It identifies some important gaps in the knowledge of the concept of recursion. It also presents the aims of the research which will emerge from a detailed review of the literature. Chapter Three outlines the approach to and the aims of the study. The method and methodological issues are discussed in Chapter Four. Chapter Five mainly investigates the design of computer-based tools for studying students' thinking when they articulate meanings about the concept of recursion through active engagement with computer-based tools for the first and second iterations. It also reports on the data of early phases of the study in iteration one (the *Treemenders*). This section will be followed by the emergent issues from the data for the first iteration and conjecture which will be reported to the second iteration's domain (the *Spirals*). In defining the term *domain* I reference what is called the domain of abstraction which is used for interactive computer-based tools. Similarly, the second iteration will also be discussed from both design and the students' thinking in this chapter. This will be followed by the emergent issues and the conjectures, which will be reported in the final stage.

The final phase of the research will be presented in Chapters Six and Seven. Chapter Six focuses largely on the tool design of the third iteration – the *Treebuilder*. The main design features of the modules and tasks of the *Treebuilder* are also discussed. Chapter Seven examines a number of student accounts of the final phase of the computer-based tool. It reveals how they construct and evolve their own mental models of the concept of recursion. Chapter Seven is divided into two major parts. Part One is basically a re-consideration of the *Spirals* tool and focuses for the most part on the students' accounts of its three tasks. The second part of Chapter Seven is predominantly about the students' accounts of the four modules of the *Treebuilder*.

Finally, Chapter Eight summarises the findings of Chapter Seven and develops these into new theoretical perspectives. It also discusses the wider implications of this research, and finishes by presenting some pedagogical implications and future steps.

2. Review of the Literature

2.1. Overview

This chapter reviews the literature in the domain of the concept of recursion. The literature review is divided into three parts. In the first part, I review the research that has been undertaken into the essential components of recursion. The second part addresses the difficulties students have in understanding and applying the concept of recursion. Finally, the third part, examines the cognitive science approach on how students understand recursion and its crucial components. My investigations into the research that has been carried out regarding the idea of mental models of recursion are also included.

PART ONE

2.2. Recursion and its Essential Components

Recursion is an interdisciplinary concept between mathematics and computer sciences. It is not only a mathematical concept but also a programming technique. It can also be considered as a problem-solving strategy both in programming and mathematical modelling (Sooriamurthi, 2001; McCracken, 1987). As a problem-solving strategy, recursion falls into the category of the Divide and Conquer (D&C) problem solving strategy. This strategy is a top down approach or top down strategy of problem-solving, which means that to solve a problem it needs to be broken down into a number of smaller sub-problems. The final answer to the original problem is the combination of the solutions of all the simpler sub-problems. For instance, to plan to go on holiday, first we need to tackle a number of small sub-problems in order to achieve the main goal, e.g. choosing the place to go, type of transport (personal car, public transport, air, coach, etc.), booking hotels, and the cost, etc.

The following diagram shows how an original problem is connected to its sub-problems. It is clear that some of the sub-problems of ‘going on holiday’ are different to the original problem itself. Which means the nature and structure of booking a room is different from choosing a place to go or estimating the cost of holiday.

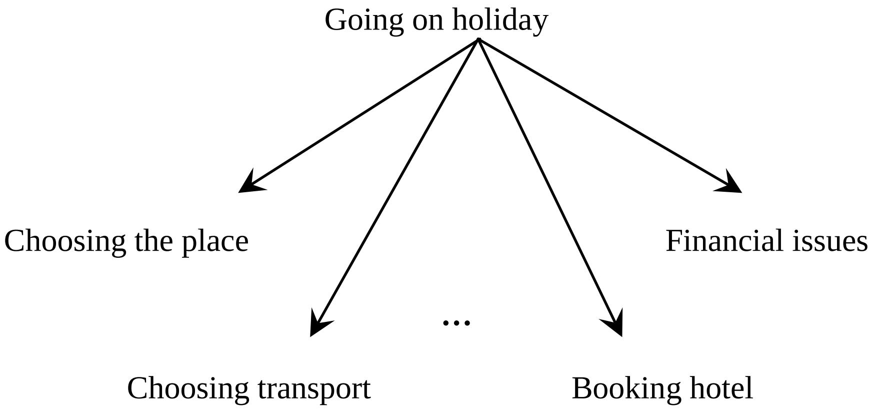


Figure 2- Divide and Conquer strategy for going on holiday

However, in recursion the sub-problems have the same nature and structure of the original problem on a smaller scale. This makes recursion a special case of D&C problem solving strategy. In the case of recursion, the sub-problems of the original problem have the same structure as the original problem.

The following two examples show this special form D&C strategy in the case of recursion. The first example is to find ‘n!’ for a natural number ‘n’. To solve this problem we need to find ‘(n – 1)!’ and to do that, we need to find ‘(n – 2)!’ and so forth.

$$n! = n \times (n-1)!$$

$$(n-1)! = (n-1) \times (n-2)!$$

.

.

.

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

$$1! = 1$$

Figure 3- Divide and Conquer strategy for factorial function

It is clear that all the sub-problems have the same structural nature of the original problem, but with smaller natural numbers. The second example is computing the determinant of an ‘n × n’ matrix. The D&C process is shown below:

Computing the determinant of a given matrix of order ‘n × n’

Computing the determinant of ‘n’ sub-matrices of order ‘(n – 1) × (n – 1)’

Computing the determinant of ‘n . (n – 1)’ sub-matrices of order ‘(n – 2) × (n – 2)’

.
.
.

Computing ‘n.(n – 1).(n – 2). … . 4.3’ sub-matrices of order ‘2 × 2’

Figure 4- D&C strategy for determinant of a matrix of order ‘n’

A number of researchers in mathematics and computer science have tried to provide a satisfactory and convincing definition of the concept of recursion (Gersting, J. L., 2007; Harvey, 1997; Wiedenbeck, 1988; Kessler and Anderson, 1986). Unfortunately, we do not have a comprehensive and perfect definition for recursion yet. The interdisciplinary nature of the concept of recursion makes it more difficult to attain a transparent definition of this concept in both mathematics and computer science. From a mathematical point of view, recursion can be considered as an inductive process with no stopping condition. In contrast, from a computer science perspective, recursion is a computational technique, which needs to be stopped at some stage to avoid infinite loops. This highlights the major difference between the structure of recursion in mathematics and computer science.

Some researchers like Harvey (1997) and Kahney (1985) defined recursion from a computational perspective. Harvey (1997) defined it as a process or function which is able to recall itself, or use itself as its sub-process. The struggle of researchers and authors can be easily seen. Kahney (1985) defines it as follows:

> *"A process that is capable of triggering new instantiations and back from terminated ones."* (p. 235)

Whereas some other researchers for example, Gersting (2007) tried to provide a more universal picture of the concept. Gersting (2007) defines it as:

> *"A definition in which the item being defined appears as part of the definition is called an* ***inductive definition*** *or a* ***recursive definition***[1]*."* (p. 129)

She was aware of the fact that this definition is not a satisfactory definition and hence adds:

> *"At first this seems like nonsense – how can we define something in terms of itself?"* (p. 129)

This statement shows that Gersting (along with many other researchers) struggles to formulate an articulate definition. Her efforts to make the definition as clear as possible cause more shortcomings regarding the definition in those two mathematical and computational aspects. Gersting (2007) continues:

> *"This works because there are two parts to a recursive definition:*
>
> 1. *A basis, where some simple cases of the item being defined are explicitly given*
> 2. *An inductive or recursive step, where new cases of the item being defined are given in terms of previous cases*
>
> *Part 1 gives us a place to start by providing some simple, concrete cases; part 2 allows us to construct new cases from*

[1] Emphasis in original

> *these simple ones and then to construct still other cases from these new ones, and so forth."* (ibid, p. 129)

In the latter part of the definition Gersting (2007) stated that 'the base' in recursion provides us a starting point. It indicates her tendency towards a mathematical approach rather than a computational one. In a mathematical approach there is no concern of moving towards infinity. However, in a computational programming approach having a stopping condition is a must, to avoid infinite loops.

Most mathematical text books define recursion in a stereotypical way. Predominantly they use some stereotype examples like factorial of a natural number or the Fibonacci sequence to define recursion. These examples are explained in the following.

- Factorial function:

1! = 1	the base case
n! = n × (n – 1)!	the recursive call

- Fibonacci sequence:

$$a_1 = a_2 = 1$$

$$a_n = a_{(n-1)} + a_{(n-2)} \text{ for } n \geq 3.$$

$$1, 1, 2, 3, 5, 8, 13, 21, ..., a_n = (a_{(n-1)} + a_{(n-2)}), \ldots$$

There is no transparent explanation of the crucial components of recursion and also the differences between two mathematical and computational approaches for recursion. The differences are almost overlooked by many authors and researchers.

Tung *et al*, (2001) described this stereotypical approach to recursion by distinguishing between understanding of the semantics of recursion and applying it. They suggested that using examples and applying the concept facilitate understanding of the semantics of it.

> *"Recursion seems to be an exception. Most learners are required to write recursive programs before fully understanding their behaviours. Due to the stereotypical nature of many recursive programming problems, instructors usually use example of recursive programming problems augmented with 'canned' problem-solving strategies to teach students*" (p. 286).

Separation of 'applying recursion' and 'understanding recursion' is what is called functional abstraction. Functional abstraction is about the difference between 'what' and 'how' it will be done. This is discussed in the second part of this chapter. This separation seems to have been neglected in the literature that is available and will be focused on later in this thesis.

The next section of the first part addresses the literature on the main components of recursion.

2.2.1. Essential Components of Recursion

Each recursive process or function whether in mathematics or computer science (computational aspect) has two main components, base case(s) and recursive call(s). Some recursive processes might have more than one base case and recursive call. It has been mentioned above that base case has different interpretation in mathematics and computer science. This makes this component of recursion an important element of the definition of recursion in both mathematics and computer science disciplines. Theoretically, it is called the simplest form of the problem.

Broadly speaking, the base case is another dilemma in defining recursion in mathematical and computational perspectives. As has already been mentioned above, the base case is the first and simplest step in the process, which reduces the problem to a manageable form that can be directly solved. In this style, the base case can be considered as a trivial form of the problem. The problematic issue is that the base case is not necessarily a starting point from a computational view. For instance, in the Fibonacci sequence above, the base cases are the starting point and they are the first two terms of the sequence. The Fibonacci sequence is not a convergent sequence – which means that it does not have any limit; this is quite natural from a mathematical point of view. However, from a computational perspective, we need to consider a stopping condition to avoid an infinite loop. Ginat and Shifroni (1999) argue that

> *"Recursion is an essential and unique tool for computational problem solving. It encapsulates decomposition of a problem*

> *into sub-problems of the same kind. Although such decompositions logically sound, it is not easily comprehended. The problem solver has to carefully specify decomposition to sub-problems and composition of the sub-problem solutions.”* (p. 127)

They also add that

> “[…] *the key emphasis in enhancing recursion formulation should be at the abstract level of problem decomposition. That is, divide – and – conquer at ‘the problem level’ irrespective of the machine implementation.”* (p. 128)

From a computational point of view, after each time calling of the recursive call the new sub-problem should approach nearer to the base case – which is going to operate as a stopping condition. For instance, to calculate 4! one needs to track the following steps:

$4! = 4 \times 3!$

$3! = 3 \times 2!$

$2! = 2 \times 1!$

$1! = 1 \times 0!$

$0! = 1$ the base case is touched so

$1! = 1 \times 1 = 1$

$2! = 2 \times 1! = 2 \times 1 = 2$

$3! = 3 \times 2! = 3 \times 2 = 6$

And finally

$4! = 4 \times 3! = 4 \times 6 = 24$

Figure 5- The process of approaching and touching base case in computational view

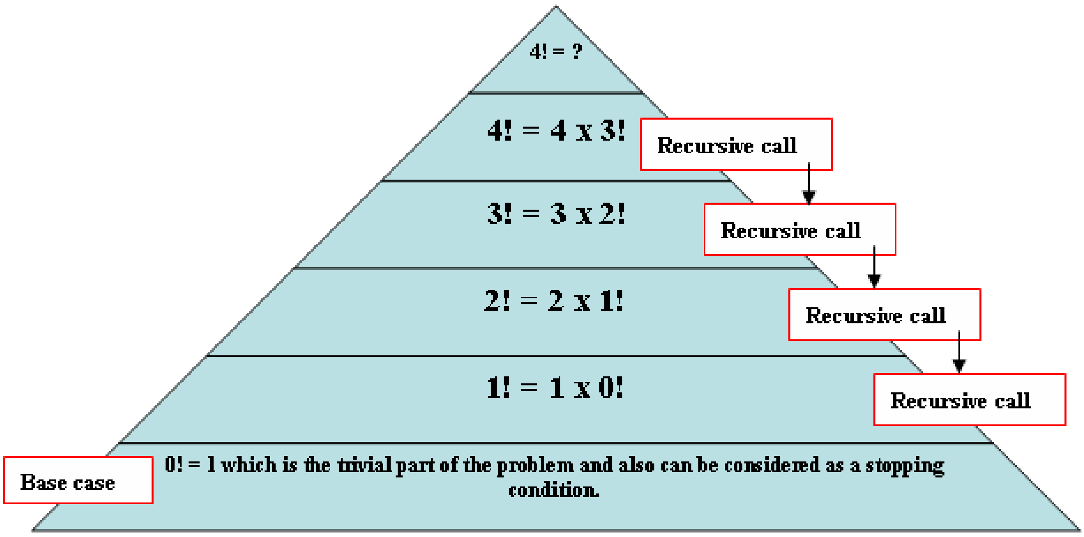


Figure 6- Visual aspect of meeting base case 0! = 1

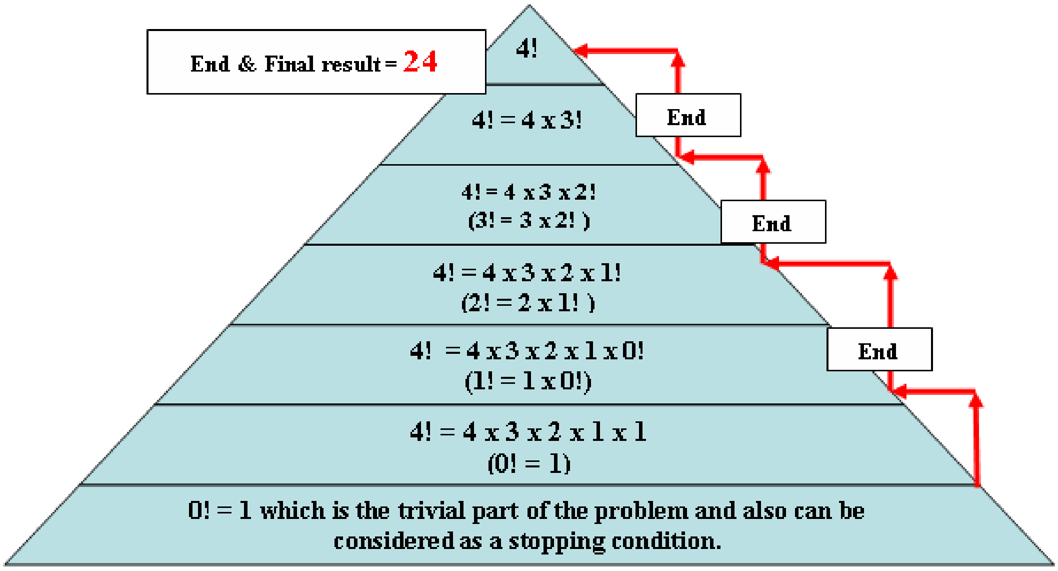
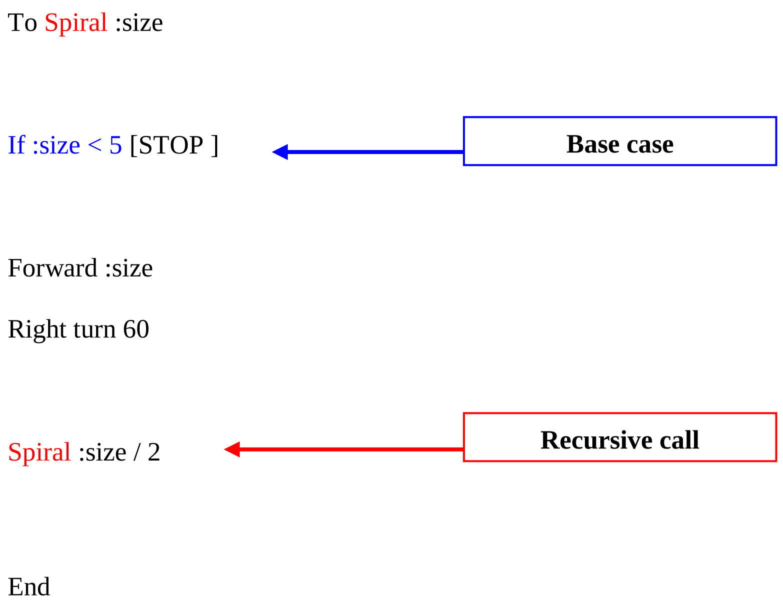


Figure 7- Final value for 4! after reaching the base case

2.2.2. Tail and Embedded Recursion

As mentioned above, the recursive call(s) is one of the indispensable components of recursive procedures. There are two sorts of recursive processes regarding the location of the recursive call(s) in recursive procedures. They are called tail and embedded recursive processes. In tail recursion, the recursive call appears in the last line of the procedure, before the ‘end’, whereas in embedded recursion, the recursive call(s) can be located at any other line of the procedure. The following two Logo programs described these two types of recursive procedures by making spiral and binary trees. The first program is a tail recursive procedure to create a spiral.



```
To Spiral :size

If :size < 5 [STOP ]

Forward :size
Right turn 60

Spiral :size / 2

End
```

Program 1- A tail recursive Logo procedure to make a spiral

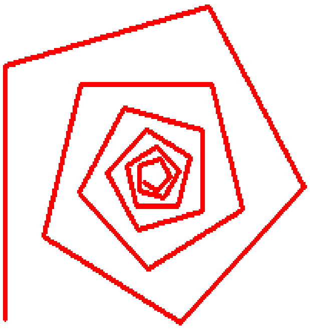


Figure 8- The outcome of the above procedure in Logo programming environment

The next program is an embedded recursive procedure with two recursive calls to make a binary tree.

```
To Tree :size

If :size < 5 [ STOP ]                    Base case-Stopping condition

Forward :size
Left turn 30

Tree :size / 2                           First recursive call
Right turn 60

Tree :size / 2                           Second recursive call

Left turn 30
Back :size
End
```

Program 2- An embedded recursive procedure in Logo programming language

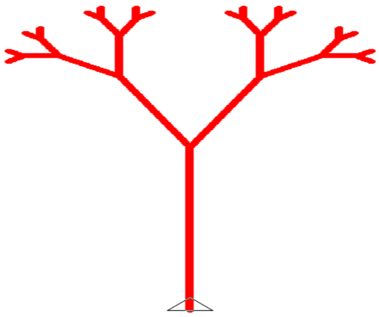


Figure 9- A binary tree, the output of the above procedure in Logo environment

Thus far, I have introduced the concept of recursion and its essential components. In addition, some of the important shortcomings of the definition of the concept of recursion as well as the interpretation of the base case in two mathematical and computational perspectives have been described. In the next part of this chapter I review the literature with regard to students' difficulties in understanding the concept of recursion. This review for the most part concentrates on the computational perspective of the concept of recursion. The predominant reason behind this tendency, anchored in designing and using computer-based tools for the concept of recursion, is the central theme of this thesis.

PART TWO

2.3. Students' Difficulties with Understanding Recursion

It is widely acknowledged that recursion is one of the most difficult concepts for students. Many students find recursion difficult to understand and to apply in their problem-solving activities (Levy and Lapidot, 2002; Sooriamurthi, 2001; Wanda, 2001; Segal, 1995; Harvey, 1993; Wiedenbeck, 1988; Kurland and Pea, 1985; Anazi and Uesato, 1983). Levy and Lapidot (2002) summarise this view:

> *"It is generally accepted that recursion is one of the most complicated and difficult-to-learn concepts for novice programmers"* (p. 89)

Henderson and Romero (1989) also argue that

> "[…] *we have found that recursion is a very difficult concept for student to learn.*" (p. 27)

Students' difficulties with this concept are due to several reasons, which are categorised as follows:

- Declarative vs. Imperative thinking;
- Inherent complexity of the concept;
- Need for a comprehensive definition and its components;
- Lack of everyday analogies;
- Recursion vs. Iteration;
- Flow of control;
- Functional abstraction.

There are a number of articles concerning the concept of recursion and its structure – base case, recursive call, flow of control, etc. – which will be discussed in more depth in this chapter (Sooriamurthi, 2001; Muramatsu and

Pratt, 2001; Segal, 1995; Wiedenbeck, 1988; Kurland and Pea, 1985; Anazi and Uesato, 1983). However, there are only few articles concerning the definition of the concept of recursion (Harvey, 1997).

2.3.1. Declarative vs. Imperative programming

In the computer sciences, *Declarative* is used as opposed to *Imperative* (also referred to as procedural) programming. Imperative programming is a sequence of instructions for the computer to be executed one by one. In contrast, declarative programming describes *what* something is like, rather than *how* it is going to be created. In other words, in imperative programming, a program specifies an algorithm to reach a goal in an explicit way. In contrast, in declarative programming a program specifies the goal or state that needs to be achieved and leaves the implementation of the appropriate algorithm to the support software. The complex mechanism of recursion is more like declarative routine than procedural style. For instance, a spreadsheet is declarative while Logo procedural programs are imperative. Sooriamurthi (2001) argues that:

> *"It is important to emphasize that a recursive routine is to be understood in one declarative reading of the routine. If one starts to manually unravel the recursion then one is doing too much work. The key is not to think too hard!*" (p. 28)

He also argues that the problems with understanding the concept of recursion is rooted in "[*...*] *insufficient exposure to declarative thinking in programming context*" (p. 25).

2.3.2. Inherent complexity of Recursion

Broadly speaking, due to its complex and uncommon structure, recursion is a difficult concept to apprehend. Velazquez (2000) states that the

> *"*[*...*] *difficulty in learning recursion does not come from the recursion concept itself, but from its interaction with other mechanisms of imperative programming*." (p 310)

Sooriamurthi (2001), meanwhile, argues that the difference between declarative and imperative strategies is one of the reasons for the complicated nature of the concept of recursion particularly for programming as a new way of thinking. He also adds that:

> *"In the world of mathematics, students are normally concerned about the declarative "what part" and not so much (if not at all) on an imperative "how part."* (Sooriamurthi, 2001)

This point, again, raises the necessity of distinguishing between mathematical and computational dimensions of the concept of recursion.

2.3.3. Need for a comprehensive definition for Recursion

Although many researchers and educators have researched around this concept on a very wide range of issues, there is still no all-inclusive definition of recursion. Leron and Zazkis (1985) distinguished between mathematical and computational aspects of the concepts of recursion. He states that from the mathematical view recursion is very close to mathematical induction and from the computational view he considered recursion as a programming technique. Ginat and Shifroni (1999), meanwhile, explain that:

> *"Recursion is an essential and unique tool for computational problem solving. It encapsulates decomposition of a problem into sub – problems of the same kind. Although such decomposition is logically sound, it is not easily comprehended."* (p. 127)

Wiedenbeck (1988) has further argued that a lack of lucid separation between mathematical and programming forms of recursion confused students.

With regard to the appreciation of the essential components of recursion, the base case is one of the most difficult parts of any recursive process or function. Recognition of the base case is one of the most problematic aspects of recursion (Haberman and Averbuch, 2002; Sooriamurthi, 2001; Kurland and Pea, 1985).

The base case can be considered as a stopping condition and also as a trivial part of the problem to be solved. The term "trivial part" is the simplest form of the problem in the Divide and Conquer problem–solving strategy. Haberman and Averbuch (2002) distinguish between how the base case informs mathematical and computational points of views as follows:

> *"There are two aspects of base cases. The first is based on a declarative, abstract approach that treats base cases as the smallest instances (in terms of problem size) of the problem for which we know the answer immediately, without any efforts. It may be the smallest concrete entity, a boundary value, or a degenerated case. It also presents the "smallest" possible input of the problem. The second aspect is based on the procedural approach, and refers to the base case as a stopping condition. In this sense, it represents the end of decomposing the problem to smaller similar problems. In order to get a comprehensive view of the role of base cases in recursion formulation, one should adopt both the declarative and the procedural approaches."* (p. 84)

Distinguishing between these two aspects of the base case, Haberman and Averbuch (2002) ascertained the difficulty of this component of the concept of recursion. However, this matter needs to be further investigated. The base case, even as a stopping condition, can be considered in declarative routine. It is not necessary to see the base case as a stopping condition in procedural programming. Our example of the factorial of a natural number shows that in a declarative perspective, the base case operates as a stopping condition. The base case in that declarative approach represents the end of decomposing the problem into smaller and similar problems which can be considered as a stopping condition. In this case, in each step of decomposition, one should ensure that each recursive call of the problem approaches the base case. As soon as the condition $0! = 1$ is reached, the process stops all the previous instantiations (Figures 6 and 8).

This process of approaching the base case after each time the recursive call is called shows the subtle inter-relationships between the base case and the recursive call within recursive procedures. These hidden internal mechanisms make the concept of recursion hard to understand. The students' difficulty with the other indispensable component of the concept of recursion, recursive call(s), is anchored in their mental model of the recursion or mixing recursion with iteration. Ginat and Shifroni (1999) describe this phenomenon as follows:

> *"The difficulties revealed in our study demonstrate that students adhere to the iterative pattern of "forward accumulation", due to their confidence with the iteration construct, but lack of trust and full understanding of the recursion mechanism."* (p. 130)

They pointed to the students' confidence with the iteration construct and their lack of trust with recursion's mechanism. However, in their research there is insufficient evidence regarding this natural tendency of the students. Anderson *et al,* (1988) argue that the duality feature of the recursive call(s) is another problematic aspect of the concept of recursion:

> *"Another source of difficulty (especially in LISP) is the duality of meaning in recursive procedure call. On the one hand the call produces some resultant data; on the other hand it specifies that an operation be carried out repeatedly.* [...] *It can be data or complex operation, depending on your view."* (pp. 162-63)

They also add that:

> *"Because students often perseverate on one view of recursion, they are often blinded to solutions that could be easily attained from the other view."* (ibid, p. 163)

It seems that one of the reasons for students' confidence with iterative processes compared to recursive ones is their intuitions and everyday life experiments, which are described in the next section.

2.3.4. Everyday analogies and Recursion

Finding everyday life analogies for the concept of recursion is a very difficult task. There are some researchers who have taken into account the role of lack of everyday analogies in learning and applying recursion (Levy and Lapidot, 2002; Wiedenbeck, 1988; Pirolli and Anderson, 1985; Kurland and Pea, 1985). Wiedenbeck (1988) argues that students can run and perform iteration easily because in their everyday life they have seen and experienced plenty of iterative analogies. However:

> *"Part of the problem may be that students gain some initial understanding of programming concepts from analogies to everyday activities and their knowledge of the use of language, whereas in the case of recursion few everyday analogies exist. Those which are frequently used (seeing one's image reflected in a row or mirrors, the painter painting a picture of himself painting a picture of himself, etc.) ... Thus, students are unlikely to learn about recursion from analogy unless the analogies come from programming itself.* (p. 275)

Wiedenbeck argued that it is unlikely that students learn about recursion from analogy. Yet, in my opinion, fractal-shaped objects – like spirals and trees – can be considered and used as everyday analogies to facilitate student learning of the concept of recursion. Wiedenbeck also notes that Anderson *et al,* (1984) point out that although analogies might help students to learn about recursion, they must be introduced carefully as they might cause a wrong mental model of the concept of recursion. This is a very interesting point as it relates the learning issues about the concept of recursion to the mental model of the concept. "Anderson *et al*, [1984] found that novices often learn to compose recursive programs by analogies to worked out recursive examples. However,

> *"[…] in relying on analogy to examples there is the inherent danger that students may develop inadequate or incorrect mental models of what recursion does"* (ibid, p. 275).

Therefore, although the use of everyday analogies is helpful in assisting learners to master recursive procedures, if the analogy is not correct, this might cause more problems by allowing students to develop incorrect interpretations and form wrong mental models of the concept of recursion. This is a very important point that needs to be considered carefully and is discussed in the third part of this chapter. My conjecture is that using fractal-shaped objects might not only develop the students' ability in understanding and programming recursion, but also assist them to form a viable mental model of the concept of recursion. The next section of this part presents a brief introduction of fractals and fractal-shaped objects.

2.3.5. Fractals

These mathematical objects were discovered by Mandelbrot in the 1970s. Computers played a very important role in the discovery of fractals. They provide us with an efficient work place to figure out patterns which had been hidden before. Perhaps the question of "What really is a fractal?" is the most challenging question about fractals since their conception in the 1970s by Mandelbrot. In fact, there is no comprehensive definition for fractals, but luckily they have some common characteristics which are accepted by almost all experts in this realm. An object is called a fractal when:

- It is self-similar – it means that the object can be divided into certain pieces, such that each of those small pieces are a copy of the original one but in the smaller scale;

- The object has a complex and multifaceted structure in the microscopic scale;
- The object has a non-integer dimension – which means that despite Euclidean geometry in which we have integer dimensions for the objects – Line is one-dimensional, plane is 2-dimensional, space is 3-dimensional, etc., fractals have non-integer dimensions which, shows the degree of their complexity (Mandelbrot, 1982).

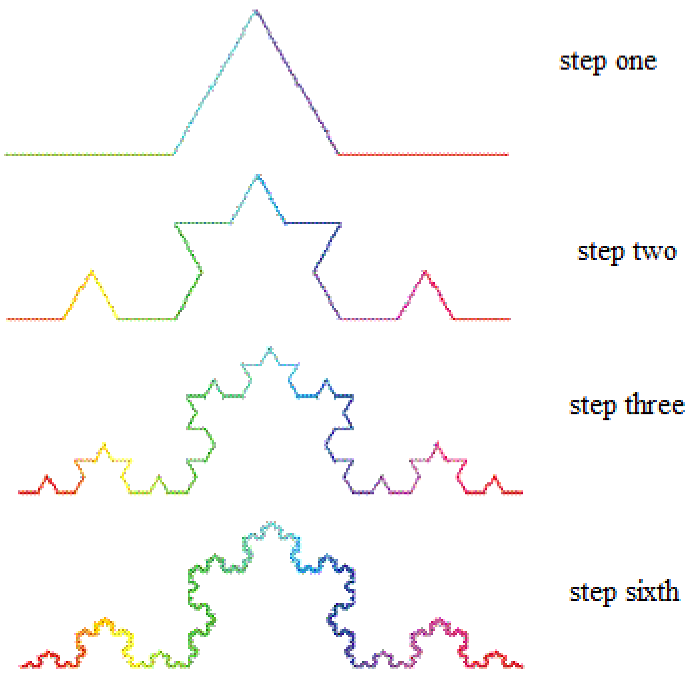


Figure 10- Koch curve fractal

For example, the Koch curve fractal (Figure 10) that was used in the first iteration of this research is a fractal and its dimension is 1.26. From a complexity perspective this number shows that this geometrical object is somewhere between a line and a plane, because a line is a one-dimensional and plane is a 2-dimensional and 1 < 1.26 < 2.

Nowadays, you can easily see the footprints of fractals everywhere. This is evident when looking at natural pattern, the complicated electronic patterns of Internet networks, astronomical research on the distribution of galaxies, the structure of DNA, and the shape of coastlines; clearly, fractals have a significant role in modelling nature and so provide an appropriate situation for learning by exploration. Fegers and Jonson (2002) have stated that fractals are

> *"*[...] *visual, relevant to many disciplines, very naturally lend themselves to computer supported activities, and can be understood (at some level) by students with relatively little mathematical background."* (p. 70)

Recursion is one of the indispensable cornerstones of fractal geometry. Fundamental elements of fractal geometry are recursions and self-similarities instead of lines and circles, which are the basic elements of Euclid's geometry. This is the main bridge between the subject of this study and fractals.

2.3.6. Recursion vs. Iteration

The relationship between recursion and the concept of iteration is another crucial aspect of the concept of recursion which is going to be discussed in this section. Distinguishing between iteration and recursion is one of the most common difficulties that students have in understanding and applying recursion and it is one that many researchers have studied (Wanda, 2001; Ginat and Shifroni, 1999; Turbak *et al*, 1999; Harvey, 1997; Wiedenbeck, 1988; Anazi and Uesato, 1982; Kurland and Pea, 1985). An *iterative* process is an accumulation process; one stage starts after the previous stage ends. In contrast, *recursion* is a process where one procedure (as a sub-procedure of itself) begins and ends before its previous procedure ends. Turbak *et al*, (1999) also explain a syntactical difference between loops and iterations:

> *"We use 'iteration' to describe a step-by-step computational process that determines the next values of a set of state variables from their previous value. A 'loop' is a particular control structure, denoted by special syntax, for expressing iteration, such as Java's WHILE and FOR constructs.*" (p. 86)

The following programs calculate the factorial of a natural number 'n' first iteratively and then recursively.

```
To iterative-factorial :n
Make "I 1
Make "n! 1

While [ :I < n + 1 ][ make "I I + 1 :n! = :n! * I ]    (The repeating part)

Output :n!
End
```

Program 3- An iterative Logo program to calculate factorial of a natural number 'n'

```
To recursive-factorial :n

If :n = 0 [ output 1]


Output :n * recursive-factorial :(n – 1)   (The recursive call)


End
```

Program 4- A recursive Logo program to calculate factorial of a natural number ‘n’

They also stated that iteration is a tail recursion. For them, iteration is a particular pattern of recursion:

> *“[…] all iterations are expressed via tail recursion, a particular form of recursion.”* (p. 88)

The influences of iteration and recursion have been of interest to many researchers. Anazi and Uesato (1982) argue that having a prior understanding of iteration facilitates a deeper understanding of recursion. They worked with 88 students in two groups of iterative–recursive and recursive–iterative. Their research shows that 64% of the students who had prior experiments with iteration were able to formulate the factorial function recursively. However, only 33% of the students with no prior experiments with iteration were able to implement the factorial function recursively. They therefore conclude that:

> *“Recursive procedures may be acquired based on learning of the corresponding iterative procedure.”* (p. 100)

They also make the point that the above conclusion is based only on an appreciation of factorial function as a mathematical definition, not as a computer program:

> *"We should be cautious when we try to extend the consideration to more complex domains such as computer programs."* (p. 102)

Wiedenbeck (1988) criticized their work by using the factorial function (mathematical view). She repeated their study by adding two more groups of iterative – iterative and recursive – recursive. Wiedenbeck's results did not support Anazi and Uesato's study. However, she carried out another study using computer programs instead of a mathematical approach. Wiedenbeck concluded in the case of computation that having prior experience of iteration facilitated understanding recursion. Further support for Wiedenbeck's results has been presented in Kessler and Anderson (1986). They focused on transferring skills between performing iterative and recursive procedures. They conclude that although writing procedures on iterative and recursive functions does not facilitate writing procedures on recursion functions, having prior experience of similar iterative procedures enabled increased sophistication in dealing with "flow of control", which is needed for understanding recursion. The flow of control is deliberated in the next section. Kessler and Anderson (1986) also claim that this result occurs because students have developed a weak mental model of recursion, and this poor mental model of recursion hampers their study of iteration. Mental models of recursion will also be discussed later in this chapter.

Ginat and Shifroni (1999) stated that discovering iteration is much easier than recursion. They also explained that although decomposition of a problem into its sub-problems of the same kind is logically coherent, it is not easily understood by learners. Another learning difficulty which is related to the composition of solutions to sub-problems is to achieve a global solution for the original problem. Kurland and Pea (1985) explained that most of the students view all forms of recursion as iteration. Some researchers believe that the functioning of tail recursion is easier than embedded recursion (Leron and Zazkis, 1985; Wiedenbeck, 1988; Turbak et. al, 1999):

> *"Recursion may be learned gradually, by bits, starting from graphics-based tail-recursion."* (Leron and Zazkis, p. 28)

Turbak *et al.* (1999) also pointed that iteration is easier than recursion:

> *"Iterations expressed via tail recursion are often easier to read, write, and reason about than loops. The rigid structure of looping constructs makes it tricky to express iterations that may terminate under multiple conditions, especially if some of the conditions occur in the middle of a loop body or require finalization actions".* (Turbak *et al*, p. 89)

Turbak *et al,* also add that:

> *"The tail recursive approach is expressible in all general-purpose programming language."* (ibid., p. 90)

Harvey (1997) points out that in iteration the procedure always repeats a certain number of commands without any changes, whereas in recursion, the procedure itself is called by one of the recursive calls with some new initial values.

The literature that has been reviewed thus far demonstrates the problematic inter-linkage between the two concepts of iteration and recursion, which needs to be focused on from a closer perspective. My conjecture is that by comparing and testing the similarities and differences in tail recursion and iteration, we can reduce students' problems with embedded recursive procedures.

2.3.7. Flow of Control

The process of control passing is one of the most important factors in understanding recursive procedures. In the above section, it was mentioned that the flow of control is essential to understanding recursion (Sooriamurthi, 2001; Kessler and Anderson, 1986; Kurland and Pea, 1985). A clear understanding of the flow of control in recursion has a direct relationship with the functioning aspect of recursive calls. Having a coherent understanding of the flow of control requires a lucid understanding of the concept of functional abstraction. Functional abstraction, in short, is the separation between what needs to be done and how it will be done. This concept will be discussed in more detail in the next section. The term flow of control can be looked at both syntactically and semantically.

From a syntactical viewpoint, it is a control structure in the procedure. For instance, as mentioned before, "[...] *loop is a particular control structure, denoted by special syntax"* (Turbak *et al*, 2001). Semantically, this explains the order of execution of the commands within a given procedure. In Logo, programs will be executed line by line. However, it has the ability to run recursion elegantly. As soon as the recursive call is encountered, the computer inserts all the lines that have been called and suspends the execution of the rest of the lines until the base case is reached. After reaching the base case, the computer resumes execution of the lines that had been suspended.

These instantiations of going forward and halting the execution and going back to execute the lines which have been called by the recursive call and then resuming the lines that had been halted is called a 'passive' flow of control over the procedure by Kurland and Pea (1985). They performed a study with students who had one year of Logo programming experience, noting that:

> *"When a Logo program is run, if a procedure references itself, execution of that procedure is temporarily suspended, and control is passed to a copy of the named procedure. Passing control is "active" in the sense that the programmer is*

> *explicitly directing the program to execute a specific procedure. However, when the execution of this instantiation of the procedure is finished, control is automatically passed back to the suspended procedure, and execution resumes at the point where it left off. Passing of control in this case is "passive" since the programmer did not need to specify where control should be passed in the program."* (p. 237)

The situation in an embedded recursive procedure is almost the same. However, there is a subtle difference in passing the control between the recursive calls. Kurland and Pea have stated that

> *"*[…] *When a procedure is executed, if there are no further calls to other procedures or to itself, execution proceeds line by line to the end of the procedure. The last command of all procedures is the END command. END signifies that the execution of the current procedure has been completed and that control is now passed to the procedure from which the current one was called. END thus 1) signals the completion of the execution of one logical unit in the program, and 2) directs the flow of control back to the calling procedure so the program can carry on."* (ibid, p.237)

They conclude that students have difficulty in running embedded recursive procedures because they have a tendency to think in terms of iteration, rather than recursion. They state that "*[t]he children were fundamentally misled by thinking of recursion as looping*" (ibid, p. 240). Students become more confused when they notice that having a looping strategy is adequate and satisfactory to work with "*active*" tail recursion, while it is not suitable

> "[...] *for embedded recursion, which requires an understanding of both active and passive flow of control. The most pervasive problem for the all children was this tendency to view all forms of recursion as iteration.*" (ibid, p. 240)

The control passing mechanism in the recursive procedures was referred to as *passive control passing* by Kurland and Pea (1985). The term *passive* has been used by them to describe the continuous moving back and forward between the recursive calls and the stopping condition. I think this is a significant movement in separating and distinguishing between the two different mechanisms of control passing between iterative and recursive procedures. However, I personally do not think that the term *passive* is a good choice for this advanced, complicated control passing in recursive procedures. The term passive seems to have a negative implication, rather than showing this advanced control passing system.

The more the literature is reviewed, the greater the need for access to the latent and hidden layers of the complicated mechanism of recursive procedures becomes apparent. To understand this, we need to present and introduce it in more sophisticated and highly developed strategies. To do so, having a high level of knowledge of the concept of functional abstraction is absolutely necessary. In the next section I focus on the concept of functional abstraction and its role in understanding recursion.

2.3.8. Functional abstraction

Functional abstraction is a very subtle concept which is considered a vital part of any design task by many researchers (e.g. Sooriamurthi, 2001). In short, it is about the ability to distinguish between functioning and functionality levels. Papert (1985) in *Mindstorms* describes it as the difference between the ability to drive a car and knowing how the engine works. The concept of functional abstraction is central to both understanding and applying the recursive procedures and functions. It is also vital in any design process (Sooriamurthi, 2001; Muramatsu and Pratt, 2001; Ginat and Shifroni, 1999, Kurland and Pea, 1985).

Sooriamurthi (2001) has studied the difficulties of undergraduate students in understanding recursion. He attributes this to an"… *inadequate appreciation of the concept of functional abstraction*" (p. 25).The key idea is that programming is a new way of thinking, and it is more about design and problem-solving than the syntactical perspectives of programming languages. Moreover, in programming, the management of complexities is vital. Sooriamurthi (2001) argues that when you cannot master the complexity, you need to handle it by using a *divide* and *conquer* strategy or abstraction. Abstraction is simply focusing on what needs to be done and, for the short-term, suspending how it is going to be done. Most students have difficulty in distinguishing the “what” part and the “how” part. Sooriamurthi (2001) argues that:

> *“The issue is simply separation of concerns: the separation of what needs to be done from how it will be done. We observed that students normally have a hard time comprehending recursion because they don’t clearly differentiate between these two forms of knowledge (the what vs. the how) and worse, tend to focus on the latter – the how. The key to comprehending any form of abstraction including recursion is to focus on the what and down play the how.”* (p. 25)

From this perspective, we can make a link between functional abstraction and declarative – imperative programming paradigms that have already been mentioned in this chapter.

The aforementioned declarative programming paradigm is about ‘what something is like’ rather than how it is going to be created. Anderson, Pirolli and Farrell (1988) have also pointed to this important issue in tracking the flow between the recursive calls. They suggest that “[...] *it is often useful to determine what has to be done to the result produced by a recursive call in order to get a result for the current function call*” (p. 163). The difference between these two levels (the ‘what’ vs. the ‘how’) is vital and important. I think focusing on this concept from the point of view of the learning and design of computer-based tools needs to be further pondered. What I am trying to say is that to drive a car, a learner does not necessarily need to know anything about the engine and its function. However, knowing about the functioning of the engine might help the learner to develop his driving skill regarding the engine’s response in different driving situations. Therefore, to acquire a certain skill in general, it is not necessary to know about the *how*. Instead, one needs to know about the *what*.

When it comes to learning a particular mathematical concept, we are not generally able to understand the concept by focusing only on the *what*. This is the case with some mathematical concepts like dividing fractions (“turn upside down and multiply”), or mathematical induction (“check that p(n) is true for some natural n, now if for any natural n, p(n) implies p(n+1) then p(n) is always true”). From a problem-solving perspective, one should bear in mind whether our purpose is the final answer or whether the process of reaching it is the key issue. Nevertheless, if finding the final answer to the question is the *purpose* of a problem-solving strategy, there would be no place for the functioning dimension of functional abstraction. For instance, acquiring the skill of dividing two fractions can be grasped instrumentally (Skemp, 1976) in a very quick and exact way i.e. turning it upside down and multiplying it. It is very difficult and hard to say that the learner will have a clear understanding about (½) ÷ (¼), which equals two. It may also be that finding the final answer is not as easy as the above example.

Solving the questions by using recursion as a problem-solving strategy is one of those situations in which reaching a final answer without having a proficient knowledge of the functioning level is not an easy task. Thus, the major point regarding the concept of functional abstraction is *to what extent* one should focus on the functioning level and to what extent the focus should be on functionality in the learning and teaching of mathematical concepts. Concentrating on this query is beyond the scope of this research. However, with regard to the concept of recursion, due to its complex character, like its complicated control passing process, one needs to ponder on the functioning level, as well as its functionality.

Another essential issue in this realm is distinguishing between the design of the computer-based tools and the learning and understanding of the concept of recursion (Table 1).

	functioning	**functionality**
Computer-based tool design	How it is designed	What is it going to do
Concept of recursion	How it works	What it does

Table 1- Separation of tool design and recursion from functional abstraction view

One might consider it as an obvious and trivial issue, but there are vital differences between them. From a computer-based design stance there is usually no need for the students to know anything about the functioning aspects of the tool design, or, how the tool works, but inevitably it is important that the student learns its functionality; what the tool does. Sooriamurthi (2001) stated that:

> *Functional abstraction is a corner stone strategy in good software design. To master recursion is to master and acquire a fundamental understanding of functional abstraction.* (p. 25)

At this stage I would like to consider the literature from a wider perspective. This helps me to analyze the students' thinking about the concept of recursion. Also, it enables me to investigate the role and impact of computer-based tools in teaching, learning recursion. In doing so, it is necessary that I review the literature on students' mental models of recursion, situated cognition and conceptual changes. These are the issues that are explained in the next part of this chapter.

PART THREE

This part of the review of the literature is divided into two sections. The first section concentrates on a brief review of the history and research that has been taken on mental models in general. The second section focuses on the research that undertaken on the mental models of recursion in particular.

2.4. A Brief Introduction to Mental Models

The term ‘mental model’ was cited as early as 1943 when Craik published “The Nature of Explanation”. Craik (1943) recognized that knowledge and understanding can be thought to operate as the application of “working models” of particular phenomena in an individual’s mind. A few years after, the cognitive scientist, Johnson-Laird (1983) used and developed the concept of “working models” as a small-scale model of reality. Basically, Craik (1943) and the few other contemporary researchers at that time were using the concept of recursion as a concept which had been accepted and used mainly based on an intuitive feeling by the researchers rather than a strong epistemological view. Therefore, for a long period of time there was no sign of any attempt towards introducing a comprehensive and explicit definition of the concept of mental model.

A few years after as a result of the symbioses combination of cognitive psychology and computer sciences, some cognitive scientists like Johnson-Laird, Stevens, and Gentner in 1983 produced two books which were mainly focused on mental models from a cognitive science perspective. This shows that although precedent researchers were using and working with mental models over the twenty years following 1943, the mental model was first introduced by Craik (1943) but there were no substantial movements on the concept until the 1980s when the cognitive scientists began to use the concept. However, it seems that despite many valuable efforts, the dilemma of defining the concept of the mental model and its borders and intersections with some other similar concepts is still open. To achieve an acceptable and practical definition of what the concept of a mental model is, and then to move towards its interpretation and utility for recursion as a mathematical concept, the rest of the section concentrates on introducing the pivotal characteristics of the mental models from a cognitive science point of view. This will be followed by the presentation of an almost plenary and precise definition of the concept of the mental model.

2.4.1. Main Characteristics and Definition of Mental Models

The main focus of study of the concept of the mental model from a cognitive perspective is to see how people interact and understand the world or the system that they encounter. In order to present a definition of a mental model which encapsulates the wide range of situations that the term 'mental model' will be used to describe, the first objective is to find the major characteristics of the concept from a cognitive perspective.

Norman (1983) characterised the human's mental model characteristics as sloppy, indistinct knowledge, incomplete, and messy. His characteristics seem to present a very clear image of the inaccuracy of mental models. However, the predictability of these sorts of mental structures is another major characteristic of humans' mental models that has apparently been overlooked in Norman's explanation. He has also added that due to the abovementioned characteristics, these models are more likely to be deficient because they might contain some flawed, probably contradictory, and perhaps unnecessary concepts.

Medin *et al,* (1990) describe mental models as a kind of knowledge structure which will be employed by people to understand the world. Generally speaking, one can sum up the main characteristics of students' mental models as follows:

- They are inaccurate, incomplete, and messy interpretations of reality, which means that based on these mental models whatever the student thought is true might not be necessarily true in reality;
- These models are much simpler than the concept they present;
- These mental structures have to be predictable, which means that they allow students to predict the possible future phases of the system at hand.

People interact with systems and the world using their mental models. However, accepting this logical necessity of the existence of mental models does not eliminate conceptual and practical difficulties. Therefore, in studying mental models, one must answer some fundamental questions like: what forms do mental models take? How does their form affect their usage? Is guidance in the use of models as important as their form? How can and should designers and researchers attempt to affect and find out more about the student's mental model?

Answering these questions illuminates the components of students' mental model about the systems that are trying to learn, observe or study. However, available research and literature does not seem to be able to adequately answer these questions.

In moving forward towards definition of a mental model Rasmussen (1979) states that students generate and form mental models of systems to describe why a system exists and what it looks like. These models also enable students to explain the present state of the system. Using these models, students can also predict the possible future states of a system. The following model shows the connections between the three characteristics of mental models and their purpose, state, function, and form.

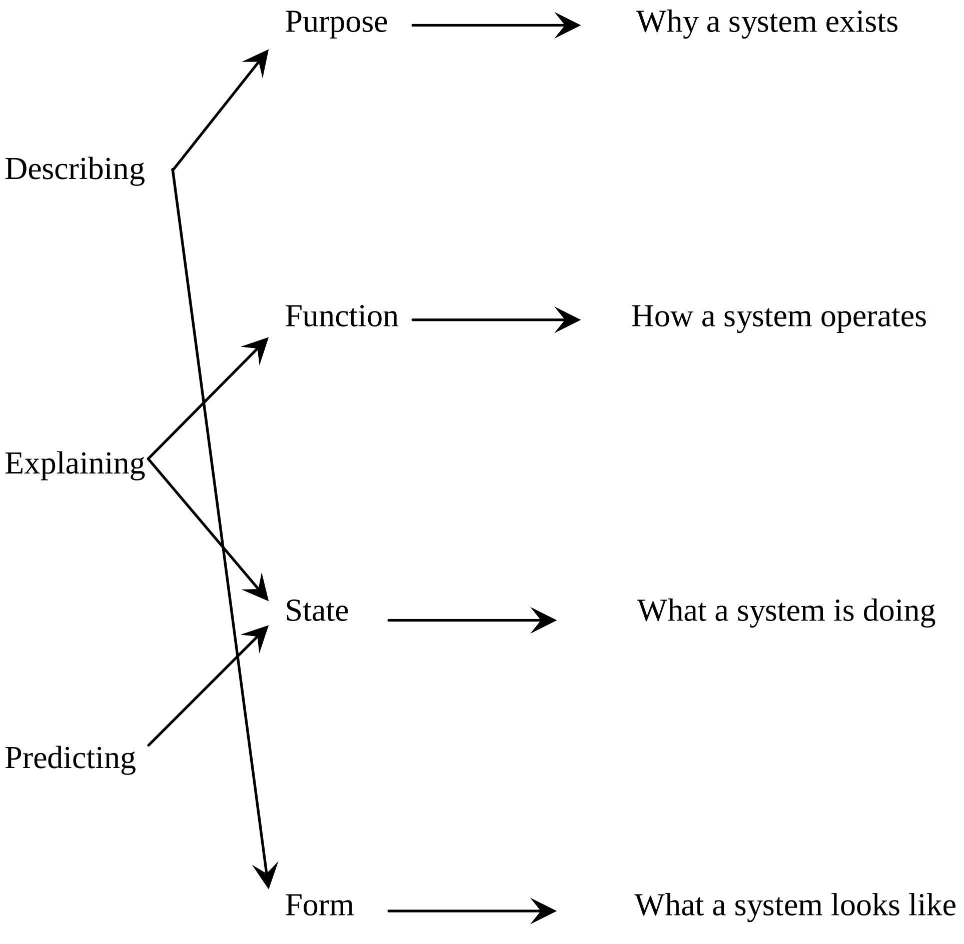


Figure 11- Rasmussen's taxonomy of the purpose of mental models (Rasmussen 1979)

Kim (1993) also points to the difference between knowing 'how' and knowing 'why'. People acquire knowledge about the world by making their own mental models. Kim (1993) distinguished between two forms of acquiring knowledge,

> "*(1) the acquisition of skill or know-how, which implies the physical ability to produce some action and (2) the acquisition of know-why*[1]*, which implies the ability to articulate a conceptual understanding of an experience.*" (p. 38)

Knowing why a system exists enables students to understand and apply their learning about the system; knowing how it operates shows what they learnt about the system. Kim (1993) respectively termed these two levels of acquisition of knowledge operational and conceptual learning. At the operational level, students are basically involved with implementations and observation of the system. However, at the conceptual level, they are mainly focused on assessments of the implementations and designing new approaches to be implemented and observed.

Kim (1993) manipulated the connection between these two levels of knowledge acquisition with two components of mental models: frameworks and routines. For Kim, the operational level represents procedural learning in which students learn the steps to complete a specific task. This is the know-how level and will be routinely captured in students' mental models.

> "*Filling out entry forms, operating a piece of machinery, handling a switchboard, and retooling a machine" can be considered as*

[1] Italics in original

> *some examples of routines that form part of a part of student's mental model.*" (p. 40)

At the conceptual level, however, is

> "*Thinking about why things are done in the first place […] [leads] to new frameworks in the mental models. The new frameworks in turn can open up opportunities for discontinuous steps of improvements by reframing a problem in radically different ways.*" (ibid, p. 40)

Kim's model of the relations between the knowledge acquisition levels and the components of mental models is demonstrated in the following table.

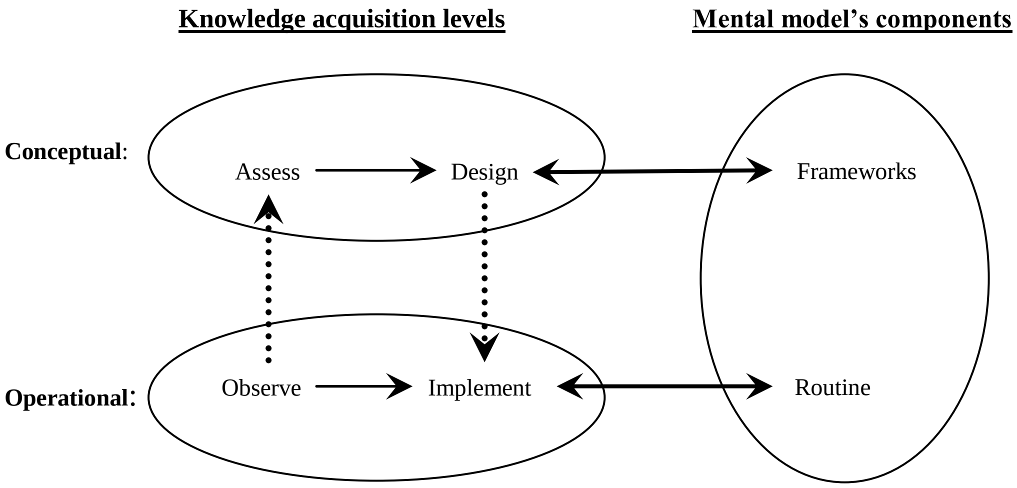


Figure 12- Kim's model of the relationship between knowledge acquisition and mental models

Kim's (1993) separation of know-why and know-how, and also Rasmussen's (1979) taxonomy of mental models, illustrate the role and importance of the concept of functional abstraction in the study of students' mental models. In other words, distinguishing between the functionality and functioning of the components of mental models holds great importance in the search for an integrated form of the mental model of a system and in this thesis an integrated model of mental models of the concept of recursion.

Mental models are not necessarily wholly accurate and they are also not complete, but at the same time they are still useful for understanding processes. Mental models which have formed in individual's minds are strongly based on their beliefs. They will only be changed when new knowledge which ultimately changes people's beliefs and understanding comes to light. Researchers in various fields place differing interpretations on the concept of the mental model. For example, Senge (1990) describes mental models as

> "*Deeply ingrained assumptions, generalizations or even pictures or images that influence how we understand the world.*" (p. 8)

In this way, Senge asserts that the individual's understanding of their environment is made up of their knowledge, beliefs, experiences and perceptions, and is also affected by their political, economical, social and cultural backgrounds.

In the context of understanding physical systems, Gentner and Stevens (1987) avoided directly defining mental models, but they explained that mental model research is

> “[...] *being characterized by careful examination of the way people understand some domain of knowledge*” (p. 1)

Johnson-Laird (1983) considered mental models as a representation of understating. In this way, Johnson-Laird (1983) stated that

> “*[a] mental model can vary from a simple image or picture to a very complex abstract or conceptual archetype built through more detailed understanding.*” (p. 8)

Two years later, Johnson –Laird (1983) mentioned that each individual mental model is only one of a number of possible models which could be, and are used in a particular context. As Spicer (1998) notes, Mantovani (1996) makes the point that differences between mental models can occur at different levels of context. That is, two people can observe the same event with different mental models and describe it differently because they have noticed different details. Evans (1989) recognises this as “*selective perception*”.

To some extent, the above explanation explains the function and structure of mental models. Using these characteristics and functions, it is possible to present a definition of a mental model as follows. Mental models are mental mechanisms that people make to describe the purposes and forms of a system, to explain the functioning and observing the state of it, and finally, to predict the future state of the system which is being studied, learnt, or observed.

Rasmussen’s categorisation of mental models (1979), did not distinguish between the researchers’ and students’ mental models. However, Norman (1983) developed a new taxonomy of mental models by introducing the term ‘conceptual model’ for the researcher’s mental model of the system being studied. Norman (1983) asserted that students’ view of the world, of themselves, of their own capabilities, and of the tasks that they are asked to perform, or topics they are asked to learn, depends heavily on the conceptualizations that they bring to the task. But, how did they conceptualize a system and how did they realize this process of conceptualization of a particular system in the first place?

Norman (1983) introduced the idea of conceptualization. He differentiates between the mental models of the expert and the novice. Norman (1983) considered four things to model people’s mental models: the target system (t); the conceptual model of the target system (C(t)); the user’s/ student’s mental model of the target system (M(t)); and, the researcher’s conceptualization of that model (C(M(t))). A conceptual model is the model which is invented by the researcher as a supposedly accurate and consistent representation of the system which is being studied, observed, or learnt – the target system. Mental models are by nature evolving models in the mind of the user/ learner. They evolve through interactions with the target system. Norman (1983) employed the term ‘conceptual model’ to delineate the model which is made by the researcher.

Norman (1983) also articulated that student's mental models will continuously be modified and evolved towards an integrated and workable state through interactions with the target system (t) and the conceptual model it (C(t)). Though it seems that the terms 'conceptual model' and 'mental model' are synonymous, Norman (1983) distinguished between them by separating educational purposes and everyday life activities.

> "*[c]onceptual models are devised as tools for the understanding or teaching of physical systems. Mental models are, what people really have in their heads and what guide their use of things.*" (p. 12)

He also introduced the term 'conceptualization of a mental model', by which he meant a model of a mental model. Thus, the researcher's conceptualization of the student's mental model is the model of the student's mental model of the target system.

Although it seems that there should be a direct relationship between the conceptual and mental models, all too often there is not. Obviously, a student's mental model reflects his/her beliefs about the system. Yet, what is not readily seen is that sometimes the student's beliefs about the system do not necessarily correspond with the conceptual model of the designers. Norman's model of modelling of a mental model can be seen as follows:

t: the target system;

C(t): researcher's conceptual model of the target system;

M(t): student’s mental model of the target system;

C(M(t)): conceptualisation of the student’s mental model of the target system

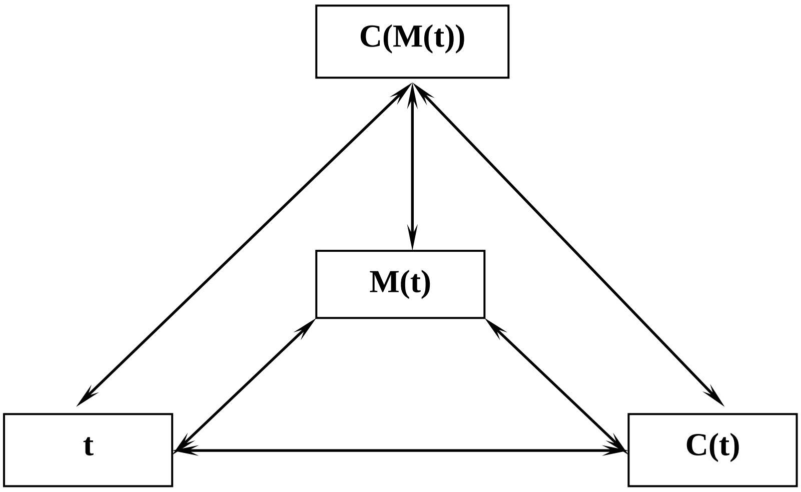


Figure 13-Modelling of a mental model

To understand students’ mental models, one needs to observe their experiments and interactions with the system and the conceptual models of the system. To do so, Norman (1983) introduced three functional factors that can apply to both mental and conceptual models: belief system, observability, and predictive power. These factors are used to distinguish the components of the student’s mental models of the target system (M(t)) and the researcher’s conceptualisation of those mental models (C(t)). This separation is a direct consequence of distinguishing C(t) – the conceptual model of the target system – and M(t). Conceptualization of the student’s mental model of the target system is actually a model of a model. The following table describes the differences between M(t) and C(M(t)) by using of the abovementioned three functional factors (Norman, 1983, pp. 10-12).

	M(t)	**C(M(t))**
Belief system	student's beliefs about the target system acquired through observation, instruction, or inference.	Should contain a model of the relevant parts of the student's belief system.
Observability	There should be a correspondence between the components and states of the mental model and the aspects and state of the system that the student can observe.	There should be a correspondence between components and observable states of the C(M(t)) and the observable aspects and states of the target system.
Predictive power	Model must have predictive power either by applying rules of inference or by procedural derivation (in whatever manner these properties may be realized in the student)	Must include a knowledge structure that makes it possible for the person to use a mental model to predict and understand the physical system.

Table 2-Functional issues to distinguish the student's mental model and its conceptualisation by the researcher

Norman (1983) summarises that:

> "*[p]eople's mental models are apt to deficient in a number of ways, perhaps including contradictory, erroneous, and unnecessary concepts. As designers, it is our duty to develop more coherent, useable mental models. …we must develop appropriate experimental methods and discard our hopes of finding neat, elegant mental models, but instead learn to understand the messy, sloppy, incomplete, and distinct structures that people actually have.*" (p.14)

This leads me on to explaining how and why mental models are relevant to understanding of recursion. Study of students' mental model of recursion provides me the way they think about recursion and would apply it in different problem solving situation. The next section of this chapter focuses on the research undertaken on the mental models of recursion.

2.5. Mental Models of Recursion

Recursion is one of the mental activities which is categorised as a highly unfamiliar activity for students. This mental unfamiliarity causes students to have difficulty in understanding it as a mathematical concept and applying it as a problem-solving technique (Gotschi, *et al*, 2003). It has earlier been mentioned that students/novices and researchers/experts differ in their mental models of a system (Norman, 1983). Particularly, research on mental models of recursion shows a significant difference between students' models and researchers' models (Kahney, 1983, Gotschi, *et al*, 2003). Students show possession of various inadequate models of recursion which mainly tends towards an iterative/loop model. Kahney (1983) in his seminal work on mental models of recursion asserted that novices and experts substantially differ in their own models of the concept of recursion. Kahney (1983) defined a model of recursion as

> "*A process that is capable of triggering new instantiations of itself, which control passing forward to successive instantiations and back from terminated ones.*" (p. 235)

He showed that novices' models of recursion mostly differ from experts' viable models of recursion, and he termed this a '*copies model*'. Kahney (1983) mentioned that it is not necessary for students to have a correct and viable model of recursion, instead it is important that a student possesses a model, even if it is an inadequate one, because this model can be considered as a base that can be debugged to form a correct model.

His interpretation shows that Kahney (1983) believed in the evolving nature of mental models. This evolution can progress through active engagement with the concept and debugging the possible errors in the learners' mind. Furthermore, it has been noted that novices' models tend towards the more familiar concept of iteration (Kahney, 1983; Kurland and Pea, 1984). Kahney (1983) noticed that experts have a 'copies model' of recursion whilst novices have a 'loop model' of the concept of recursion. What he meant by the loop model was an iterative interpretation of recursion. Based on this hypothesis, Kahney focused on the student's possession of a copies model as a viable model of recursion versus iteration as an inadequate model of recursion. However, in his research, Kahney (1983) found that students have more than one deficient models of recursion. He categorized these mental models of recursion into five categories:

1. Copies model;
2. Loop model;
3. Null model;
4. Odd model;
5. Syntactic model.

And he added that the only viable and correct model of the concept of recursion is a copies model and the other models are incorrect and inaccurate models which needed to be developed. A 'copies' model of recursion is one which is always viable and enables the possessor of the model to recognise the forward flow of control in execution of the commands in the recursive procedure in a sequential way, then suspending a few commands of the procedure after each time calling of recursive call(s) which invokes new instantiations of the original procedure, and then backward control passing from the invoked copies of the procedure to their parents to terminate the execution of the procedure.

By 'loop' model, Kahney (1983) meant a deficient model of recursion which bases itself iterative interpretation and ignores the process of generation of new copies of the original procedure after each time calling of the recursive call(s). A student using a loop model disregards the backward control passing from the invoked copies to their parents. Kahney (1983) categorised students who do not show possession of any kind of model for recursion into the category of having a 'Null' model of recursion. He used the term 'odd' model to describe those students who were not able to predict the behaviour of the system, and had various misunderstandings of the recursion process. For instance, a misunderstanding of *STOP* in the stopping condition of the procedure with the *END* command which means the total termination of the procedure. The term 'syntactic' or 'magic' model categorised those students who did not know how recursion works, but they were able to recognise some segments of the recursive procedure. These students were able to predict the future of the procedure based on their recognition of the syntactical segments of the procedure, rather than having a clear understanding of mechanism of recursion.

Gotschi *et al*, tried to improve Kahney's (1983) categorisation of mental models of recursion in 2003. They defined a student's mental model of recursion as his/ her knowledge of recursion, and that this mental model is feasible and practicable if enables students to follow the recursion procedure truthfully and consistently. However, Gotschi *et al*, (2003) did not take the forward and backward flow into account in their definition of the viable copies model of recursion, by using the terms 'truthfully and consistently', they tried to cover those vital aspects and characteristics of a correct model of recursion. Gotschi *et al*, (2003) acknowledged Kahney's categorisation of mental models of recursion, but they added a few more models by distinguishing the nature and mechanism of the active and passive control passing in the recursive procedures. They identified further mental models employed in understanding the concept of recursion as follows:

1. Step model;
2. Return-value model;
3. Algebraic model.

Those students who demonstrate possession of a 'step model' evaluate recursion as *IF-THEN-ELSE*. They have no idea of the mechanism of control passing in the recursive procedures. Students with a 'return-value model' consider the recursive call(s) as the instantiations to generate values which are going to be evaluated and stored, and then combined to give the final answer. Finally, those who demonstrate possession of the 'algebraic model' manipulate the recursive procedure as an algebraic problem. Gotschi *et al*, (2003) asserted that students with ‘syntactic/magic’ and ‘active’ models need only a little help to be able to construct a viable copies model of recursion, whereas those with the ‘step’ and ‘return-value’ models had many misconceptions about the essential components and characteristics of recursion.

Ultimately, Gotschi *et al*, (2003) developed Kahney’s study, and furthermore they tried to measure the distance of knowledge of non-viable models of recursion from the viable models of recursion. They defined a viable model of recursion as follows:

> *“A student’s mental model is viable if it allows them to accurately represent the mechanics of recursion. Non-viable mental models are constructed if students have misconceptions about the mechanisms of recursion or have misconceptions about concepts fundamental to recursion.”* (p. 349)

What none of these researchers considered, however, is the ‘order’ and frequency of occurrence of these different models in the mind of students, or the hierarchy of predominance of certain models over others in students’ thinking. What has been ignored is which none-viable model will be formed in the mind of students first, and then how does it evolve into an integrated viable model of recursion.

Tung *et al*, (2001) supported Kahney's idea that having mental models – even incomplete or deficient one, is better than having no model, because an incomplete or deficient mental model of recursion has the potential to evolve and change during experimentation and the debugging. Based on the potential evolving nature of the mental models, Tung *et al*, (2001) tried to present an explanation, albeit an imperfect one, of the hierarchy of forming mental models in the minds of students as follows:

> *"Successful learners can acquire better problem solving skills and advance gradually from the naive loop model, to the intermediate syntactic, and finally to more sophisticated analytic or analytic/synthesis models."* (p. 292)

What they meant by an analytic model derived from the idea of "*[t]he solution for a programming problem by analysing its input – output behaviour*" (p. 292). This provides us with a primitive model of evolution of mental models of the concept of recursion from the phase of the looping model to what they called an 'intermediate' syntactic model and then towards more sophisticated models. The idea of the hierarchical evolution of mental models of recursion will be elaborated later on in this thesis within a computer-based domains environment.

Wu *et al*, (1998) even further developed Norman's (1983) idea of the conceptualisation of a model of recursion. They differentiate between the abstract and concrete conceptual models and try

> "[…] *to understand how different types of conceptual models and cognitive learning styles influence novice programmers when learning recursion*." (p. 292).

They conclude that, "*[c]oncrete conceptual models are better than abstract conceptual models*" for teaching recursion to novice programmers (p. 292). They also conclude that

> "*[i]ndividuals with an abstract learning style tend to perform better in learning programming*." (p 295)

Having reviewed the above literature on mental models of the concept of recursion, from a functional abstraction point of view, it becomes apparent that there is also a need to address the 'functioning' dimension of the recursion. The review revealed a big gap in the literature on the functioning aspect of recursion.

It is clear that researchers have categorised the students' mental model of recursion from the exclusive viewpoint of 'functionality'. Although this categorisation is an appropriate base point from which start research the functionality aspect of the students' mental model of recursion, there is potential for further analysis and research to delve into this area from a functioning dimension.

In the final section of this chapter, I would like to review previous research which has been undertaken in the area of using computer-based tools in the monitoring of students' mental models of recursion. This is important to my research and my design of a computer-based domain to monitor students' thinking about recursion.

2.6. Computer-based Approach to Recursion

This section focuses on the idea of using interactive computer-based tools to introduce mathematical concepts. Schon (1983) stated that the interactive computer-based tools are constructed to represent a virtual version of the real world. In the computer-based conceptualisation, students are not only able to develop their understanding of the concept which is being studied, but they can also view and reflect on their work through the computer screen as a window into the components of the concept.

Schon (1983) asserted that students' understanding of the concept could be improved in interaction between their actions with the tasks in the computer-based tools and the act of reflecting on their work. Reflecting on tasks they have been involved in enables students to think about their attitudes and assumptions, as well as their failures, which helps them to develop their knowledge about the concept. In *Mindstorms*, Papert (1980) stated that computer-based tools are appropriate environments in which students can learn from their failures, and in which they are able to build up their knowledge about the concept in a gradual style. These environments also enable students to link formal and informal knowledge. Papert (1980) explained that students

> "*Learn to transfer habits of exploration from their personal lives to the formal domain of scientific theory construction.*" (p. 117)

Papert (1980) considered theses tools as "*incubators for knowledge"* (p. 121). In this environment, learners are able to acquire knowledge through their own efforts. Lakoff and Nunez (2000) argued that embodiment saturates all human thinking. Papert (1980; 1993; 1996) continually asserts that computer-based tools are appropriate tools for embodying mathematical concepts. A computer-based tool puts students in a situation in which they can learn from their experiences.

In her seminal work on the relationship between context and cognition, Lave (1988) introduced the concept of 'situated-ness'. She elaborates on the relationship between context and cognition, suggesting that cognition is a socially situated activity. Therefore, artificial laboratory settings are not appropriate for the study of cognition because they are separated from the everyday context. For Lave, interactions with everyday situations deeply shape the learning process. Situated cognition is a major shift from an individualistic approach to a heuristic interaction within the context. In this study in particular, this contextual interaction is designed in a computer-based tool environment.

Lave's point of view led us to the idea that a student's knowledge of a mathematical concept is an integrated form of small elements which have already been gained in different situations. This important idea has been developed by diSessa (1998). He called these pre-existing elements of knowledge, which are acquired in different situations, 'p-prims', a short abbreviation for phenomenological primitives. In his seminal work '*What change in Conceptual change?*' diSessa (1998) stated that a student's knowledge of a phenomenon is based on well-structured pieces of knowledge – or p-prims.

Collins (1988) summarised the benefits of acknowledging the fundamental role of situation in cognition for the learning process as follows:

- Students learn in what conditions and situations they can apply the knowledge;
- Various situations and settings put students in a creative problem-solving state;
- Students will be able to see the implications and logical relationships between the concepts in different situations;
- Students build their own knowledge and work with it in a structural way. This way of building knowledge will allow them to apply and modify their knowledge in later use. (Collins, 1988, pp. 1-3)

Computer-based tools can provide us with a valuable environment which can be helpful in examining these sorts of heuristic issues. Papert (1980) stated that Logo as an educational programming language provides an outstanding environment in which to investigate problem-solving strategies. For him, Logo-based tools proved exceptionally advantageous in examining a wrong answer to a problem or incorrect approaches towards new knowledge:

> *“Typically in math class, a child’s reaction to a wrong answer is to try to forget it as fast as possible.”* (p. 61)

In contrast, in the computer-based tools environments, students are given the opportunity of learning from their mistakes. This process can take place by constructively using the error messages or unexpected feedback. The process of debugging is a fundamental part of problem-solving and understanding a procedure. Students in Papert’s Logo-based tools are able to work with ‘the new’ subject to be learned or ‘the new’ problem to be solved, and make connections to ‘the old’ subjects, which have already been acquired or ‘the old’ problems that have already been solved. Thereby, in a progressive way forward ‘the new’ anchors in the mind and, in turn, becomes the ‘old’ when you want to move on to explore other new problems.

2.6.1. Pedagogical Aspects of Computer-based Tools

Edwards (1995) asserted that computer-based tools can also be used as representational systems to embody particular mathematical concepts. In this research I intend to use computer-based domains as a window to embody and introduce the concept of recursion to the students by designing appropriate tools.

Noss and Hoyles (1996) stated that computer-based tools act as a window through which students are able to look at their own thinking about mathematical concepts. This window provides the researcher with the ability to observe and investigate the students thinking, mental models, and construction of meaning.

> *"*[…] *the computers, as we shall see, not only afford us a particular sharp picture of mathematical meaning making; they can also shape and remould the mathematical knowledge and activity on view.*" (p. 5)

This window provides the researcher an opportunity to observe the process and evolution of 'meaning-making', and see how student thinking forms and is shaped through interaction with the computer-based tools. Computer-based tools can be used to embody the concept of recursion and act as a window for students when they are thinking about recursion through:

- **Using before knowing** – this gives the opportunity to experience a concept and to work with its components in an interactive environment before knowing its semantics. Thus, students have the opportunity to re-create and re-build the knowledge; this is what Papert (1996) referred as the *power principle*:

  > *"The principle is called the power principle or "what comes first, using it or 'getting it'?" The natural mode of acquiring most*

knowledge is through use leading to progressively deepening understanding. [...] *The power principle re-inverts the inversion."* (p. 98)

Students are able to use the concept before knowing it and, therefore, they are able to construct and re-construct their own knowledge of mathematical concepts through interaction with the digital environments;

- **Tools to think with** – Papert (1980) asserted that computer-based tools can be considered as tools that students can use to think about mathematical concepts in depth. He also called the environment 'math-land'; a space in which students are able to live with mathematical ideas and objects, and by experiencing and thinking about them, develop their knowledge about the mathematical concepts;

- **Bridges formal and informal** – by appropriately de-contextualizing the formal mathematical knowledge and "*phenomenalizing*" (Pratt, 1998). By incorporating appropriate examples of everyday phenomena, computer-based tools can provide an environment in which students can bridge the gap between the formal and informal. In other words, computer-based tools can be employed to make a bridge between the abstract and concrete. Pratt's (1998) idea of 'phenomenalizing' concerns designing meaningful tasks in which the computer-based environments not only affect the representation of the mathematical concept, but also the process of interacting with it. I will develop this idea more thoroughly below, when I talk about Purpose and Utility. The most common interpretation of mathematical abstraction is the de-contextualisation of mathematical concepts. An example of this can be found in working with 3-D spaces in Linear algebra as triples, and defining binary operations to add and product them. This interpretation of abstraction serves to highlight the huge gap between formal mathematics and informal mathematics. Pratt (1998) points to the need for thinking about this gap by 'phenomenalizing' the mathematical concepts. Noss and Hoyles (1996) also mention distinguishing between the process and the final product. Computer-based tools which provide an interactive environment in which students evolve their understanding of mathematical concepts gives the researcher tools with which to observe this process, as well as the final result.

Wilensky (1993) considered abstraction as akin to 'connections'. According to him, concrete-ness is not a property of a concept, but rather a property of a person's relationship to a concept. Therefore, the degree of concrete-ness of a concept depends on the number of connections made between it and other concepts; the less connections made by a student, results in the formation of a more abstract concept in their mind. Noss and Hoyles (1996) develop Wilensky's idea of abstraction when they refer to the term '*web of connections*' (p. 105). This term is taken from a well known term, in the world of the Internet, 'World Wide Web' to articulate the idea of the webbing of connections in mathematical concepts. Noss and Hoyles (1990) state that students construct mathematical ideas through making connections by using the computer-based tools and webbing ideas together (pp. 220-227).

- **Purpose and utility** – it has been mentioned above that Pratt (1998) explored the idea of 'phenomenalizing' to design meaningful computer-based tools and tasks through which we can make a bridge between the utility provided and the purpose of doing those tasks. Ainley and Pratt (2002) argue that when students engage in interaction with a purposeful computer task, they can build and re-build their own knowledge, and see the catalyst components of knowledge. Therefore, the engagement factor is very important in the learning process. Computer-based tools enable the researcher not only to provide students with an environment in which they can engage and interact with the concept to be learnt/studied through the utility that has been provided, but in order that they can also see a glimpse of the purpose of doing that activity. Ainley, Pratt, and Hansen (2006) assert that the purpose-utility is an important factor in the designing of computer-based tools. They state that engaging with purposeful computer-based tasks allows students

> "*to understand not simply how to carry out a technical, but how and why that idea is useful, by applying it in a purposeful context.*" (p. 20)

- **Thinking-in-change** – Pratt (1998) asserts that students can shape and form their understanding of a concept through interaction with the computer-based environments. This process is called '*thinking-in-change*" by Noss and Hoyles (1996). They state that the thinking-in-change process

  > "*demands that we devote at least equal attention to what is to be learnt, as well as the meanings the learner draws from the educational experience*." (p. 10)

  Computer-based tools provide the environment for the researcher to investigate through the thinking-in-change process and observe how students evolve their thinking about mathematical concepts.

2.7. Summary

In this chapter, I reviewed literature in three parts. The first part of the chapter relates to the concept of recursion and its indispensable components. The second part of the chapter concerns students' difficulties with the concept of recursion and examines the different interpretation of recursion in mathematical and computer science disciplines. It reveals that an appreciation of the flow of control has a central role in understanding the complicated mechanism of this concept. The literature mainly looks at the 'functioning' aspects of the concept of recursion and its components. Consequently, the need for focusing on the 'functionality' aspects of recursion was one of the major gaps that have been revealed. The last part of the literature review mainly concentrates on mental models and mental models of recursion. The categorization of mental models of recursion is reviewed in this section, and the chapter finishes by reviewing the literature on the role and importance of computer-based tools in a students' learning process. This section discusses on the role of computer-based tools to embody the mathematical concept through using-before-knowing, bridging formal and informal by phenomenalization of mathematical concepts with some on screen objects.

In using and designing computer-based tools as a way of examining and monitoring students' thinking of recursion, I hope to develop and explore the concept of recursion from a functional abstraction standpoint by investigating both functioning and functionality aspects. I want to highlight the idea of cognition and move forward to examine students' mental modes of recursion. This will shed light on new theories and information about recursion which will contribute towards the progress of research in this area. These ideas will be developed in more detail in the following chapters.

3. Aim of Research

3.1. Overview

This chapter explains the aims and objective of this research. It begins with a brief section about the theoretical view followed by a review of the aims of this research. This research concentrates on two related overarching themes:

1. The articulation of certain principles and heuristics to describe the design of a computer-based domain for abstraction of recursion, and
2. The way that the students shape, change, and modify their thinking about the concept of recursion.

One of the major intentions of designing such computer-based domains is to see whether it is possible to plan an approach which introduces the formal interdisciplinary concept of recursion into informal computer-based tools. The observation of students' evolving thought in such a carefully designed computer-based domain may provide a better understanding of how they shape and modify their thinking about the concept of recursion by active engagement with the specific features of those domains.

A review of the literature on the computer-based tools provided the basis to suggest that such an approach might be possible. Computer-based domains provide the environment where a formal mathematical concept like recursion can be presented by informal everyday life objects like fractals and fractal-shaped objects on the screen. The term domain refers to the *domain of abstraction* (Pratt *et al*, 2008). In such domains, it was necessary to provide students with a purposefully designed computer-based environment for the concept of recursion. From now on, in this thesis the terms '*domain of abstraction*' refers to a computer-based tool for abstraction of the concept of recursion. These domains provide students an environment, in which they could think about the process of producing the final product throughout the computer screen window. In order to expand upon my approach it is useful to refer to Pratt's (1998) opinion which distinguishes between the process of design and its final product.

> *"The approach draws its inspiration from the notion that it should be easier to analyse and make sense of the design process as it is acted out rather than through an examination of the final product."* (p. 66)

A design based research methodology will allow me to develop the computer-based domains gradually and progressively, allowing me to observe how the students express their thinking about recursion when engaging with, and using the tools. Obviously, I cannot observe their actual thinking about recursion, but I am able to measure and analyse the way that they use the tools and react to certain features of the domains. and the way that they connect the process of producing the final answer. Noss & Hoyles (1996) who introduced the notion of a '*window*' to describe how computer-based tools can work to enable us to see the way things are done, stated that

> *"*[…] *the computer can help to make explicit that which is implicit, it can draw attention to that which is often left unnoticed* […] *the computer, as we shall see, not only affords us a particularly sharp picture of mathematical meaning-making; it can shape and remould the mathematical knowledge and activity on view."* (p. 5)

As the researcher, the computer-based domains acts as a *window*, into the students' thinking about the concept of recursion and its indispensable components. Simultaneously, these computer-based domains can act as a window through which the concept of recursion can be observed by the students. My aim is to design a transparent window through which students can view and work with recursion and in doing so, encourage them to express their thinking. It will assist me to gain more meaningful and detailed data. Furthermore, throughout the active engagement with the computer-based domains, the students are able to make new connections with their previous knowledge about recursion. In this sense, my research is based on the idea of '*webbing*', as described by Noss & Hoyles (1996). I have explored and analysed the responses students make about the structures which they consider to be useful for expressing the concept of recursion.

Taking into account the literature that has been reviewed, it can be seen that students already possess certain heuristics and initiatives regarding recursion and its components, for instance, iteration. Thus, one of the aims of my study is to explore how students' initial intuitions, even incorrect ones, emerge, develop, and change when they are placed in the interactive environment within computer-based domains. I want to focus on the students' thinking-in-change process and how their own experiments and feedback shape their own mental models of recursion. Distinguishing students' mental models of recursion within computer-based domains enabled me to develop a taxonomical model of students' evolution of their mental models from both functioning and functionality dimensions.

3.2. Theoretical stance

This research focuses on the students' thinking about the concept of recursion and its components. It is also aimed to examine how they change their thinking and evolve their mental models of recursion in a computer-based environment. Given the complexity of recursion and its two interrelated dimensions of functionality and functioning, it was necessary to offer students some ideas about the mechanism of control passing in recursion. Constructionism offers some ideas about how such tools might be designed.

- Using – before – knowing:

  To employ this idea in the research, I envisaged students writing programming code or designing animated representations (*Using*) in order to gain new insights into those functioning and functionality dimensions (*Knowing*);

- Phenomenalizing:

To phenomenalize the concept of recursion, I will need to create on screen instantiations, so that the students might uncover the hidden layers of the concept of recursion;

- Purpose & Utility:

  Ainley & Pratt (2002) recognise that the computer-based tool acts as a window in which the students find the task *purposeful*, and in turn, this might lead to the students' appreciation of the *utility* of recursion.

Although my design ideas are heavily shaped by constructionist literature, I do not propose that this study is constructionist per se. Constructionist theory advocates an approach towards teaching and learning, in which students are encouraged to be in control of their learning and to take ownership of that process.

Given the constraints of my research study, my intention is to research how students' thinking about recursion changes and how this impacts on my thoughts regarding the design of the computer-based domains. The lessons I learned from the constructionist literature will enable me to develop an effective window on that thinking-in-change, but my overall aspiration does not lean towards a programme for teaching and learning.

3.3. Main Themes

The literature reviewed I have reviewed for this research shows that using everyday analogies to explain recursion has been neglected. For instance, Wiedenbeck (1988) in her seminal work pointed that one of the reasons for students' difficulty with the recursion is lack of everyday analogies. This gap in the literature inspired me to fractals and fractal-shaped objects in order to conceptualise the concept of recursion in a computer-based environment. Having a clear understanding of the main components of recursion is an essential factor in understanding and applying recursion (Kurland & Pea, 1984; Anazi & Uesato, 1982; Sooriamurthi, 2001). My aim is enhance students' knowledge of these components in a computer-based domain.

As it mentioned before, the literature I have reviewed concentrates on the functioning aspect, which reveals a big gap regarding the functionality of recursion. This has convinced me to pay special attention to the functionality aspect of recursion. Also, using computer-based domains to monitor and study on the structure and evolution of mental models is another area that has been overlooked in the literature. Gotschi *et al*, (2003) briefly mentioned the advantages of having some non-viable models of recursion in the process forming and shaping a viable model, but apart from this, there has not been a serious attempt at using a computer-based tool to study student's mental models of recursion, and to identify a hierarchical method of forming and shaping mental models in students' minds.

The other principal issue in my design is the role and importance of visualisation techniques, like visual codes and animation. Providing students with a visual platform for understanding the complex mechanism of recursion will assist in the development of their knowledge and provide a suitable method for me to analyse their experiments. Although some researchers like Tung *et al*, (2001) and George (2000) worked on the visualisation of the concept, there was no evidence of them investigating the functionality and functioning aspect of recursion, by designing computer-based domains and distinguishing between tail and embedded recursion.

3.4. Specific Aims

The explicit aims of my research are as follows:

1. How can design of computer-based domains reveal the latent layers of the concept of recursion?
2. How can my tool design operate as a bridge between formal and informal mathematical concepts, and recursion in particular?
3. Does the design of computer-based tools support students' perceptions of the concept of recursion and its components?
4. How will the student's engagements with purposeful computer-based domains allow them to shape, modify, and evolve their mental models of the concept of recursion?

5. To what extent can computer-based domains simplify and support students' appreciation of recursion's functionality aspect regarding control passing mechanism?

3.5. My Approach

The approach is to design a specific and purposeful computer-based domain to act as a double opening faced window (Pratt 1998; Noss & Hoyles, 1996). Through this window, I will be able to look at the process of the students' thinking about the concept of recursion.

I intend to use computer-based tools for probing university level students' thinking about recursion. The computer-based domains will be tested, modified and re-designed using design based research (DBR) methodology (Cobb *et al*, 2003). DBR and its main characteristics will be discussed thoroughly in the next chapter. However, broadly speaking, it is a process of designing, testing, modifying and retesting the computer-based domain over a few stages – each of these stages called iteration.

Each stage of the computer-based domain is based on how well the design worked in the previous iteration. The first stage was based on the insights gained from the literature and my own conjectures about the concept of recursion. The next stages of tool design will be shaped by observation of students using the tools, alongside insights gained from reflecting on the whole design effort up to that point. I presented the initial results of each of the iterations at conferences and departmental seminars at Warwick University. During these presentations, I had the opportunity to discuss the results with other researchers. Some parts of the results have been, or are being, published by the research conferences or in journals.

Having stated the major aims of the study, the next chapter of the thesis focuses on the methodology that has been employed to implement this study.

4. Methodology

4.1. Overview

This chapter explains the methods and methodologies that have been employed to implement this research. The chapter is divided into nine sections. Sections two, three, and four are dedicated to examining design based research methodology; its history, and the way that I have employed it in this study. In section five, I explain how I have also used qualitative research methodology. These sections are followed by the research setting section and the methods of collecting and analysing the data. The chapter concludes with a summary of its sections, which leads to the next chapter which concentrates on the evolution of the computer-based domains that have been invented and employed in this research.

4.2.History of Design-Based-Research (DBR) in a Nutshell

Design Based Research (in short DBR) has recently received considerable attention from many researchers in educational studies (Brown, 1992; Collins, 1992; Cobb *et al*, 2003; Design-based collective, 2003). It is considered as an emerging framework that is able to lead to better educational research. The fundamental assumption here is that cognition will occur during an interaction between students' activities and a computer-based domain as the environment in which the learning process takes place. Ann Brown (1992) and Allan Collins (1992) referred to DBR as *design experiments* research methodology. diSessa and Cobb (2004) described DBR as

> "*iterative, situated, and theory-based attempts simultaneously to understand and improve educational processes*" (p. 80).

DBR can be described as a continuous/ongoing cycle of design, testing, analysis, modification, and re-design.

Design based research provides the researcher with an environment in which he/she is able to study the student's learning process in a practical and realistic learning situation. The researcher's involvement with the situation allows him/her to track an evolving set of evidence in a systematic way. DBR has both pragmatic and theoretical orientations, but it is mainly a dynamic, collaborative approach. Cobb *et al*, (2003) have described it as follows:

> *"Design experiments are pragmatic as well as theoretical in orientation in that the study of function – both of the design and of the resulting ecology of learning – is at the heart of the methodology"*. (p. 9)

According to Ann Brown (1992) and Alan Collins (1992), one of the salient characteristics of DBR is its provision for allowing the design and contextualisation of research in practical situations, in collaboration with the participants. Cobb *et al*, (2003) state that,

> *"Design experiments entail both 'engineering' particular forms of learning and systematically studying those forms of learning*

> *within the context defined by the means of supporting them. This designed context is subject to test and revision, and the successive iterations that result play a role similar to that of systematic variation in experiment.*" (p. 9)

Cobb *et al*, (2003) identify five interweaving characteristics of DBR. The *first characteristic* is that,

> "*The purpose of design experiments is to develop a class of theories about the process of learning and the means that are designed to support that learning.*" (p. 10)

The learning process is not only about absorbing knowledge, but it is about the way that students construct and evolve their mental models about the concept which is being studied. The *second characteristic* of DBR describes it as a

> "*highly interventionist nature of the methodology. Design studies are typically test-beds for innovations. The intent is to investigate the possibilities for educational improvement by bringing about new forms of learning in order to study them.*" (p. 10)

The *third characteristic* of DBR is that,

> "*[d]esign experiments create the conditions for developing theories* [*…*] *thus design experiments always have two faces: prospective and reflective*" (p. 10).

DBR is prospective in the sense that design implementation begins,

> "*with a hypothesized learning process and the means of supporting it in mind in order to expose the details of that process to scrutiny*" (p. 10).

DBR is reflective because it is a *conjecture-driven* method.

> "*The initial design is a conjecture about the means of supporting a particular form of learning that is to be tested. During the conduct of the design study, however, more specialized conjecture are typically framed and tested*" (p. 10).

The *forth characteristic* of DBR is a result of its prospective-reflective feature, which makes it as an *iterative design*. The iterative process requires the researcher to be alert to observing and understanding evidence in a systematic way. Finally, Cobb *et al*, (2003) stated that,

> "*theories developed during the process of experiment are humble not merely in the sense that they are concerned with*

> *domain-specific learning process, but also because they are accountable to the activity design*" (p. 10).

And this point can be interpreted as the *fifth characteristic* of DBR. Reeves (2006) outlines three underpinning principles of DBR as follows:

> *"Addressing complex problems in real contexts in collaboration with practitioners; integrating known and hypothetical design principles with technological advances to render plausible solutions to those complex problems; and conducting rigorous and reflective inquiry to test and refine innovative learning environments as well as to define new design principles*." (p. 58)

Therefore, the principle aim of DBR is to create a strong bridge between the real world and educational research. I compared DBR with more traditional research methods, and favoured to employ DBR in my research.

4.3.DBR and Traditional Methods

In 1992 Collins explained that in DBR students are not treated as subjects that need to be directed, but instead, they are treated as co-designers and participants. Students can be considered to be co-designers in DBR because their interactions with the design create further ideas either for forming new conjectures or modifying existing ones. Students are considered as participants, because DBR provides an environment in which the researcher-student, student-student, student-computer, and finally, student-computer-researcher interact and make a contribution to the final results.

One difference between DBR and traditional research methods is the ability of DBR to focus and concentrate on the complex situations in real world experiments. DBR is flexible, and due to its iterative nature, the researcher is able to test and modify the possible errors and inadequacies, as well as conducting in-depth observation through the iterations, which helps him/her to surmount the complicated nature of the real situations. Reeves (2000) stated that DBR is an action research oriented method, in the sense that the researcher has to make changes throughout the iterations. The other thing is that DBR is situated, which means that it mainly involves researching in *naturalistic contexts* (Barab and Squire, 2004). Cobb *et al*, (2003) mention that,

> "*Prototypically,* design *experiments entail both 'engineering' particular forms of learning and systematically studying those forms of learning within the context defined by the means of supporting them.*" (p. 9)

However, DBR can often produce contextual output which is not necessarily appropriate in broader contexts, and these theories need to be confirmed with the other traditional research methods. Succinctly, DBR produces *ontological innovation* (diSessa and Cobb, 2004), local instructional theories (Cobb *et al*, 2003), and design knowledge (Edelson, 2002).

It has been mentioned above that the ultimate goal of DBR is to make a link between the real world and educational research. Using DBR and its iterative research cycles, not only enables the researcher to evaluate an innovative design and intervention, but he/she can move forward to run systematic attempts to improve the innovative design. This is cyclical. The next section of this chapter focuses on the conjectures and how they are embodied in the DBR approach.

4.4.DBR in This Study

This research concentrates on the student's understanding of the concept of recursion and the way that they construct their mental model of this concept. Recursion is an interdisciplinary concept and research on recursion is also interdisciplinary research. It needs to draw on multiple theoretical perspectives, which helps me as researcher and designer to build my understanding and insights into the nature of students' learning process and the way they construct and develop meanings for the concept of recursion. DBR fits into my research perfectly because it encapsulates a series of approaches within a practical learning based setting, rather than a single fixed approach. It is an appropriate framework to connect real world phenomena and mathematical concepts, bridging formal and informal. I decided to use fractals and fractal-shaped objects to contextualize the concept of recursion in a DBR framework.

Through the collaboration of contextual practices and theory, DBR allows me to move beyond merely observing, to become involved with the student's learning activity. Pratt *et al*, (2008) describe this process as *design for abstraction*. They state that design for abstraction can be viewed from three angles: the designer and his/her design perspective; the design process and abstraction of the mathematical concept; and, finally, students and their interaction and interpretation of the design. The figure below shows the interrelations between the crucial components of design for abstraction.

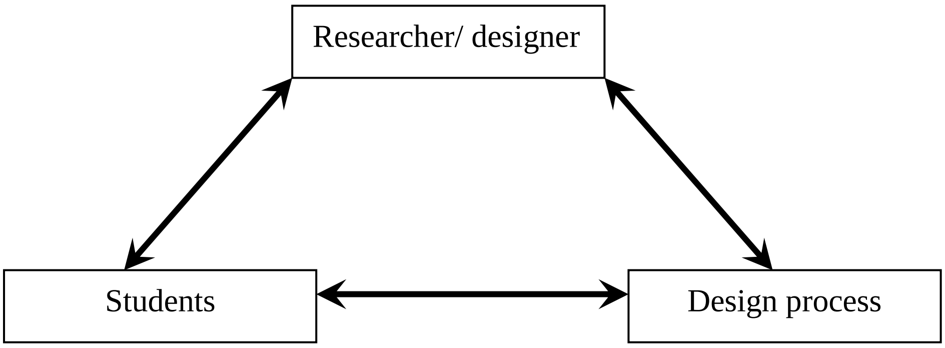


Figure 14- The interaction between major components of design for abstraction

The figure below shows the process of meaning-making of a mathematical concept through design for abstraction and phenomenalization of that concept within the design.

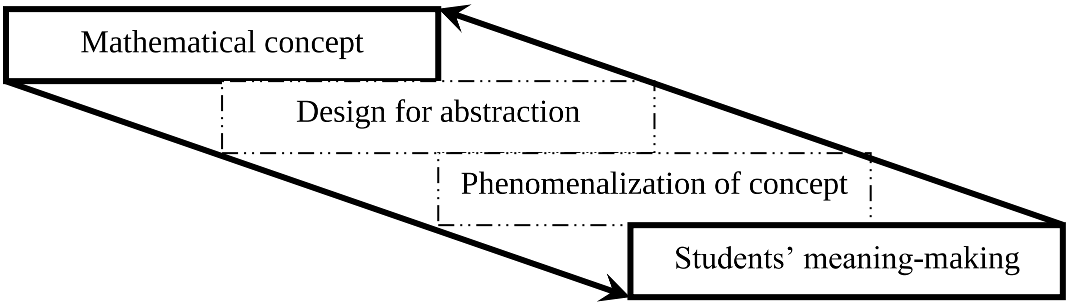


Figure 15- The process of conceptualization of a concept by researcher and phenomenalization of it by students

Based on the above plan, I decided to design a computer-based domain approach using a DBR framework research method. My research aims to investigate students' understanding of recursion through active engagement with the computer-based domains. To do so, based on the above scheme, I decided to design a domain to abstract and phenomenalize the concept of recursion. This will give the opportunity for students to engage with the crucial components of recursion. Also the design provides me with the opportunity to observe the way they think about recursion and form their own mental models of the concept. This takes place within a DBR framework.   My approach was prospective as I started the research with some assumptions and hypotheses about student's difficulties with the concept of recursion, based on the literature that I had reviewed. My other motive for employing DBR in this research was the prospective and reflective nature of it, which is reasonably fitted to the progressive modification of the computer-based domains in different stages.

The next stages of the research were carried out based on the additional conjectures that emerged through student-tools-student, students-tools-researcher interactions. The conjectures that emerged from each of the iterations were embodied in the design of the next iteration, and were modified and developed based on the reflective results of the design. Therefore, my approach is aligned with the nature of DBR because it is prospective – in the sense of starting with a hypothesis, and reflective, in the sense of employing a conjecture driven method.

Within the DBR framework I also employed several qualitative methods to delve into the inner levels of students' thinking about the concept of recursion and its indispensable components. These qualitative methods are described from both generic and particular aspects later on. However, in the next two sections of this chapter, I describe the *developmental* and *using* dimensions of the iterations within the DBR framework.

4.4.1. Developmental Dimension (Tool design)

The developmental dimension of each one of the iterations, apart from the first iteration, shows the process of designing a new iteration by modification and development of the previous iteration based on the issues that emerged from the students’ usage of it. The first iteration that was an exploratory stage was designed based on the assumptions and hypothesis that arose from the review of the literature and my own conjectures. The second and third iterations were designed based on the emerging and additional conjectures.

4.4.2. Usage Dimension (Tool use)

This dimension is more about the interactions of the student-tool-student, student-tool-researcher within each one of the iterations. The connections that they make, the way that they use the tool, the results and their reaction to those results, their explanations and utterances about the concept of recursion are all part of the usage dimension. The usage dimension is the data for the research that needed to be collected and analysed.

4.5. Qualitative Research Methodology

This study is conducted in a qualitative paradigm. Basically, in this study, due to the nature of DBR methodology in computer-based approach, I did not encounter solid and certain data like fixed numbers or structured interviews to be quantified to work with. The need for in-depth interviews alongside observing students working with the domains of abstraction, to find out how they think about the concept of recursion and how they change their thinking (thinking-in-change process), convinced me that a qualitative method was most appropriate for my research plan.

Qualitative research usually begins in a relatively open-ended way and gradually narrows down to the research questions. This method usually involves a range of methods: informal interviews (semi-structured), direct observation, participation in the students' activity, collective discussions, analyses of the personal documents produced by the students, and self-analysis. Thus, although the method is generally characterized as qualitative research, it can (and often does) include quantitative dimensions. Bryman (2001) states that in a qualitative approach we start with a general research question, then by choosing appropriate subjects we move towards collecting and interpreting the relevant data, and this is followed by the theoretical and conceptual stages. The conceptual and theoretical work will provide us with a "*tighter specification of the research question(s)*" which might demand the collection of further data and then a return to the interpretation level. After doing this cycle of collecting, interpreting, theorizing, and specification of research questions, we can go onto the final writing of the findings and conclusion (Bryman, 2001, pp. 267-269).

Therefore, a qualitative research approach is a cyclic progressive process. A process of observing and collecting things, thinking about them, interpreting them, coding and theorizing them, and finally specifying how they meet the research questions. Showing how events and patterns unfold over time is often a concern when using a qualitative approach. Pettigrew (1997) states that qualitative research methodology tends to view the students' responses in terms of process. He describes a *process* as

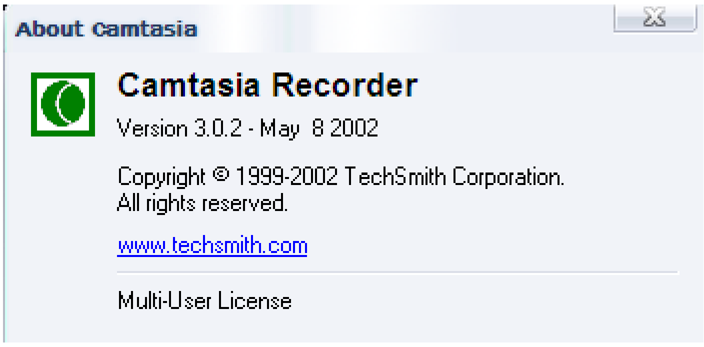
> "*a sequence of individual and collective events, actions, and activities unfolding over time in context*" (p. 338).

This emphasis on the process can be chromatically seen in a DBR framework. Bryman's (2001) qualitative research steps are reminiscent of iterations of the DBR methodology. In fact, we need to do the above steps in each one of the iterations of DBR. The next issues are the means of collecting data and the interpretation of it, which are going to be explained in this chapter.

According to Bryman (2001) among all the methods of collecting data in qualitative research methodology, interviewing and participation are the most well-known and commonly used ones. Regarding the nature of my research questions which are about the way in which students think about the concept of recursion, as a researcher I wanted to create an active situation and atmosphere to persuade the students to talk; the more they talk about their understanding of the concept the more clear I would become about what they really think about the concept. In doing so, *interviewing* is a very appropriate qualitative method and has been used by many researchers. The other dimension of this research which has to be carefully observed is the way in which students work and engage with the designed computer-based domains. Observing students' interaction with the domains of abstraction provides a view from their perspective:

> *"*[...] *many qualitative researchers express a commitment to viewing events and the social world through the eyes of the people that they study."* (Bryman, 2001, p. 277)

This provided a great opportunity for me as a researcher to ponder on the deepest parts of the students' minds to see how they think about the concept and how they change their strategies in their learning-understanding process, and finally how they make meanings for the concept being studied. To observe the students' interactions with the computer-based domains, I decided to use *participant observation*, which I describe as a qualitative data collection method later on in this section. I also recorded the students' actions and utterances by using a portable tape recorder as well as using *Camtasia* recorder software (Figure 16).



Camtasia Recorder
Version 3.0.2 - May 8 2002

Copyright © 1999-2002 TechSmith Corporation.
All rights reserved.

www.techsmith.com

Multi-User License

Figure 16- The Camtasia recorder version that I have employed in this research

When students work with the tools, they do not always give any verbal explanation; they sometimes just make some movements with the mouse or type something on the screen, modify it or even delete it. Recording and reporting these moments is one of the most challenging parts of data collection tasks in this research. However, in such situations, Camtasia is of great help as this software records every one of the students' actions on the screen as well as recording their utterances.

The data related to this research were mainly collected through semi-structured interviews and participant observation. The following two sections of this chapter describe the qualitative data collection techniques that were employed in this research in a generic style.

4.5.1. Semi-structured Interview

Basically, an interview is a conversation between two or more people. The questions are asked by the interviewer and the answers are given by the interviewees. Interviews can also be employed as a research instrument in which the interviewee and interviewer interact with each other. Research interviews are conducted based on certain goals which have to be clear before embarking on the interview.

I mentioned before that interviewing is one - and probably the most widely employed - of the data collection methods in qualitative research. Bryman (2001) distinguishes between two sorts of interviewing methods in qualitative research, the *unstructured interview* and the *semi-structured* interview. He describes the unstructured interview as follows:

> *"There may be just a single question that the interviewer asks and the interviewee is then allowed to respond freely, with the interviewer simply responding to points that seem worthy of being followed up."* (p. 314)

He continues to describe the semi-structured interview as:

> *"The researcher has a list of questions or fairly specific topics to be covered, often referred to as an 'interview guide', but the interviewee has a great deal of leeway in how to reply"*. (p. 314)

In the semi-structured interview, there is no need to follow the questions in order. In addition, the interviewer might ask some questions which are not even in his/her guide. Therefore, it is possible to adjust the emphasis of the research as a result of significant issues which might emerge during the interviews or observations. However, in the case of both the unstructured and semi-structured interview, the atmosphere of the interview is flexible. Qualitative research generally tends to be more unstructured and adaptable and seeks rich and detailed answers. Questions in the qualitative research methods are normally open-ended, probing questions. In this research I use semi-structured interviews, which will be explained further on in this chapter.

4.5.2. Participant Observation

I mentioned in 4.5 that participant observation is another important and widely employed method of collecting data in qualitative research. It is also one of the most demanding methods of data collection in qualitative research. Participant observation can be defined as a method of collecting information and data simply by participating in people's everyday lives and activities. Gold (1969) bases the degree and extent of the participation of the observer as a researcher and the peoples' activities, classified participant observation into the four classes of *complete participant*, *participant-as-observer*, *observer-as-participant*, and *complete observer* (Figure 17).

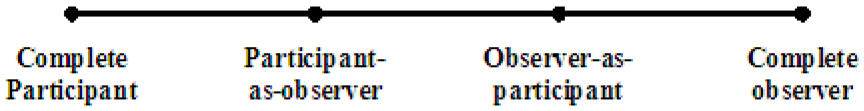


Figure 17- The participation classification of Gold (1969)

Gans (1968) categorises them in the three following classes:

- Total participant;
- Researcher – participant;
- Total observer.

By *total participant* he meant that the researcher "*is completely involved in a certain situation and has to resume a researcher stance once the situation has unfolded*" (Bryman 2001, p. 300). This can be considered as the *complete participant* of Gold. By *research-participant* he meant that the researcher has a dual role in certain situations. Gans (1968) also refers to this as a *semi-involved* role, which is a sort of combination of participant-as-observer and observer-as-participant of Gold's classification. Finally, Gans's third class is *total observer,* in which the researcher has no involvement in the situation. This is almost the same as Gold's last class, *complete observer*. By *total participation* he meant that the researcher is "*in a certain situation and has to resume a researcher stance once the situation has unfolded*" (p. 300). Gold also added that by using participant observation, the researcher immerses his/herself in the subject to be studied. In this way, they can perceive the subject more deeply than through other methods like questionnaires. This method, through a concentrated involvement with subjects in their natural environment, allows researchers to gain a close awareness of the students and their practices. In this method, the researcher becomes a participant in the context being observed. So, it implies an immersive experience in a real world. On the other hand, the researcher must observe the subject to be studied, which needs a scientific approach to knowledge.

4.6.Research Setting

One of the main cornerstones of the design of the computer-based *domain* in this research was some of the constructionist ideas. For instance, to design the tools for this research purposes, Papert's power principle, purpose and utility, thinking-in-change, and bridging formal and informal mathematics with regards to the concept of recursion were considered (Pratt *et al*, 2008).

I approached such a setting by designing a computer-based tool using *Imagine Logo,* a powerful version of the *Logo* educational programming language, published by Logotron. This computer-based tool is designed to model binary, ternary trees and spirals as examples of fractals and fractal-shaped objects. These everyday life examples were employed in this research with the purpose of conceptualising the concept of recursion. The purpose of the computer-based tool was to uncover the latent layers of the concept of recursion for the students. The other purpose was to provide me as a researcher with a window into the way that students shape and form their own mental models of the concept of recursion.

4.6.1.Participants

The first iteration of this research was implemented during August 2005 using five student volunteers. The volunteers were studying in their first and second years of mathematics and computer sciences degrees at the University of Warwick. They were aged 18-20 and were tutored by me. I interviewed them in groups of two pairs and one individual. The second iteration was performed during February and March 2006 with seven volunteer mathematics students aged 18-22. They were interviewed in groups of three pairs and one individual. And finally the last iteration of the research was implemented in October and November 2006 using 17 volunteer students aged 18-22, who were studying at the University of Warwick in their first and second year of a mathematics and computer sciences degrees. They were interviewed in groups of seven pairs and three individuals.

Selection of students at university level was based on the following reasons. The first reason is, there is an advantage of working with university level students which is anchored in a pedagogic context. In spite of the importance of the concept of recursion in mathematics and computer sciences, there is almost no place for the concept of recursion in curricular material and text books used within university courses in mathematics and computer sciences. The second aspect is fixed from a pragmatic and technical perspective. The research needed to examine students' ability in understanding and applying the concept of recursion in a computer-based environment. So, to challenge their ability and knowledge in employing recursive strategies required them to have a basic knowledge of mathematics and programming at an undergraduate level.

4.6.2. Implementation

This research was implemented within the DBR framework throughout three iterations. The first iteration was designed during Jan-Feb 2005 and tested with the students in August 2005. Iteration two was designed during Sep-Dec 2005 and tested in Feb-Mar 2006. Finally, the third iteration was designed during Aug-Sep 2006 and tested with the students in Oct-Nov 2006. Students who volunteered to participate in this study were working with the computer-based tasks either in groups of two or individually. Attention was essentially paid to the collaboration of student-tool-student, student-tool-me as researcher. The domains of abstraction for the three iterations are called *Treemenders, Spirals,* and *Treebuilder* respectively.

These computer-based tools were designed and programmed by me as researcher and designer and Professor Pratt as co-designer. The design of the *Treemenders* – for the first iteration - was a result of regular meetings and discussions between myself and Professor Pratt on the possible ways to embody the initial conjectures of the computer-based tool *Treemenders*. The initial conjectures were largely based on the extensive literature that I had reviewed and my idea about employing the binary trees as an everyday life example as well as employing fractals to reveal the hidden layers of the concept of recursion. Additionally, the design of the other two domains – *Spirals* and *Treebuilder* – for the second and third iteration was also a result of the regular meetings between me as researcher/designer and Professor Pratt as co-designer.

However, the designs of these two domains of abstraction were chiefly based on the emerging conjectures from the previous iteration(s) as a result of the student-tool-student and student-tool-me as researcher interactions. I carefully discussed these new ideas and conjectures with Professor Pratt for validity issues and also concerning ways of embodying them into the design of the next iteration. Inviting colleagues and departmental members to share and discuss the results and emerging issues of the iteration provided me with more ideas and insights. I also presented the results to the public in a number of seminars and conferences, and a journal paper which made the results more accessible for many researchers (Ammari-Allahyari, 2005, 2006; and a journal paper in the Journal of Learning (in press)). The following table presents a sketch of the developmental and usage aspects of the three iterations in this research within the DBR framework.

Developmental aspect			
	Iteration 1	**Iteration 2**	**Iteration 3**
Time **(design and test)**[6]	Jan-Aug 2005	Sep 2005- March2006	Aug-Nov2006
Usage aspect			
Pre – questionnaire	Yes	Yes	Yes
Interview	3 x 1.5 hours	4 x 1.5 hours	10 x 1.5 hours
Number of students	5 students (2 pairs and one individual)	7 students (3 pairs and one individual)	17 students (7 pairs and 3 individual)

Table 3- An outline of the developmental and usage of the three iterations

[6] This time does not include the period of the data analysis.

4.7.Methods for Data Collection

As a qualitative researcher, I was searching for an understanding of: students' behaviour when they engaged with the domains, the way in which they think about the concept of recursion, and the way in which they develop their knowledge through active interaction with the tools. Therefore, as mentioned above, the main data collection methods that I needed to employ were semi-structured interviewing, participant observation, and a combination of both techniques.

The rationale for combining these two methods was that there were some aspects of the students' behaviour that could not be revealed by only observing their interaction with the computer. There were also some issues that could not be uncovered by merely interviewing the students. For instance, their interpretations and reactions after seeing the animations used in the domains of abstraction cannot be grasped even by semi-structured interview; this was totally dependent on the impact of the visual presentations on the students' interpretation of the concept, which was not accessible in the interview sections.

The study was implemented in three iterations in a DBR methodology framework. The first iteration was at an exploratory level, so the tool was designed based on the primitive conjectures and some issues regarding students' difficulties with the concept of recursion. Due to the lack of clear understanding about possible student difficulties, I decided to employ the following stages for collecting the data. First, a semi-structured interview using a semi-structured pre-questionnaire to see how the students think about the recursion by seeing the pictures of trees and Koch curve – a mathematical fractal – the interview guide for this semi-structured interview is located in appendix A. Then as a participant observer I asked the students to start work with the domain immediately after finishing the pre-questionnaire. The role of participant observer and semi-structured interviewer helped me to examine the deeper layers of students' knowledge of the concept of recursion and its essential components. This also enabled me to provide considerable emphasis on the contextual understanding of the students' behaviour.

Regarding the above mentioned classifications of participant observation, I can identify my role as a *researcher-participant*. This dual role allowed me to observe the students' interaction with the computer-based tool as well as interjecting if a student was pondering about some unexpected issue, or when the students seemed to be stuck and there would not be any progress if I didn't intercede. Being a participant observer provided me with access to gathering information regarding the students' experiments with the concept of recursion through closely working with the tool. This method has also allowed me the privilege of flexibility for deliberation and consequently deepening the students' way of thinking about recursion. As a participant observer I encountered and was involved "*in a continual process of reflection and alteration of the focus of observations in accordance with analytic developments*" (May, 2001, p. 159).

The data collection methods in the second iteration were almost the same as in the first iteration. The only difference was related to the design of the tool. Having done the first iteration, there were some issues that emerged from the students' utterances and working with the first iteration's tool which informed the design of the second iteration's tool. Apart from this slight difference in design aspect, the data collection methods were the same as those in the first iteration. The students were asked to answer the semi-structured pre-questionnaire first and then immediately after finishing it start their experiment with the computer-based domain.

Collecting data in the final iteration was implemented only by participant observation of the students' engagement with the tools. Thus, by adopting the role of participant observer, I was able to observe and track the students' actions in different situations during they work with computer-based tools. Sometimes they were quite silent and just worked with the buttons and cursor on the screen. By recording their movements on the screen by Camtasia recording software along with the qualitative questioning techniques and asking such questions as *Why did it happen? What do you think about it? Why have they done a certain action? What would happen if something different happened? What do they mean and how they relate to particular relationships and actions?* I was able to record those silent moments to be analysed (May, 2001, pp. 156-165).

In all these three iterations, the semi-structured interviews with the students and participant observing of the students' interaction with the tools were recorded both on the tape and digitally using Camtasia recording software. This recorded data and my own hand written notes were presented as the data for this study to be analysed and discussed. I wrote intensive field notes including important and crucial moments in student-tool-student and student-tool-me as researcher collaborations as well as the new ideas and insights which were triggered in my mind for the next stage of design while I was working with my participants. These notes and the transcripts of the semi-structured interviews and the results of the pre-questionnaires were all taken into account as the unprocessed material to be coded and considered as the data which needed to be analysed for the next stage of the research. The next section of this chapter concentrates on the methods used to analyse the data in this study.

4.8.Methods for Data Analysis

Qualitative data analysis is a process which is similar to the DBR approach in the sense that it has a cyclic nature. Qualitative data analysis by nature is a progressive process. All the interview sessions concerning the iterations were tape-recorded as well as using Camtasia recording software. The latter allowed me to re-observe the scenes in accordance with the students' utterances after finishing the interview sessions. All the recorded data was transcribed. The interview transcriptions and the students' experiments with the tools were treated as the data to be analysed for this research. Analysing the data focused on two prevalent qualitative analysis methods, *progressive focusing* and *coding*. Progressive focussing guarded me against pre-assumptions. It can be considered as a method of interaction between research issues and the field activities. Progressive focussing allowed me to focus on the research issues gradually as they were emerging through each one of the iterations under observation. Stake (1981) states that

> *"Progressive focusing requires that the researcher be well acquainted with the complexities of the problem before going to the field, but not too committed to a study plan. It is accomplished in multiple stages: first observation of the site, the further inquiry, beginning to focus on the relevant issues, and then seeking to explain."* (p. 14)

I mixed the coding analysis method with the progressive focusing. This method of analysing the data allowed me to have reliable results based on the students' utterances.

In his seminal work 'Analysing Qualitative Data', Gibbs (2007) categorises the codes into two categories of *descriptive* and *analytic* codes. This categorization actually allowed me to form my thinking about the data and analyse them. Descriptive codes are more concentrated on the detailed transcriptions. I also sought to classify the students' descriptions. However, the analytic codes are focused more on a wider perspective. Therefore, compared with the descriptive codes, the analytic codes are more generic and defined in such a way so as to contain a broad-spectrum of the descriptive codes. The progressive focussing acted like a transitional catalyser between the descriptive and analytic codes.

4.9.Summary

In summary, this chapter is comprised of seven key sections. The first section concentrates on introducing the DBR; its history and characteristics, and the way in which it has been employed in this research. Then I discuss qualitative research methods, followed by the research setting and the way the design of the computer-based domain of abstraction has been implemented in this research as well as how I collected the data for this research. The latter in particular describes how the domains were implemented into the different stages of this research. Having explained these methodological issues, the next chapter of the thesis presents the path of the evolution of the computer-based tools from the exploratory tool of the first iteration into the more sophisticated domain of abstraction of the second iteration. The above mentioned techniques were practically applied for this transition as well as collecting data about and analysis of the final iteration, which is discussed through Chapters Six to Eight.

5. The Evolution of the Computer-Based Domain

5.1. Overview

This chapter outlines the different stages of the development of the computer-based domain of abstraction throughout three iterations within the DBR framework in this study.

This outline is developed from two major perspectives: the design development of the computer-based tools and the usage of those tools by the participants. Thus, the chapter starts by describing the path of designing the computer-based domains, based on some constructionist's ideas of bridging formal and informal, phenomenalizing of the concept by fractals and putting the students in the situation of using recursion before knowing its mechanism, at the same time aims to uncover the students' responses and thinking about recursion in the domain of abstractions for recursion.

The domain of abstraction design in this research is developed throughout three stages, and is here referred to as three iterations. These iterations were designed, tested, and modified within the DBR framework. The domain of abstractions in these three iterations are called the *Treemenders*, the *Spirals*, and the *Treebuilder* respectively for the first, second and third iterations. The first two iterations (the *Treemenders* and *Spirals* domains) of my research are thoroughly discussed in this chapter. The third iteration, the *Treebuilder* domain, will be discussed in the next chapters. The emerging issues and the conjecture which inform the next iteration are also described after each one of those iterations. The chapter finishes by reporting the emerging issues and conjecture(s) for the final iteration, which is explained in Chapters Six and Seven. The following diagram sketches out how my study is performed throughout three iterations within a DBR framework.

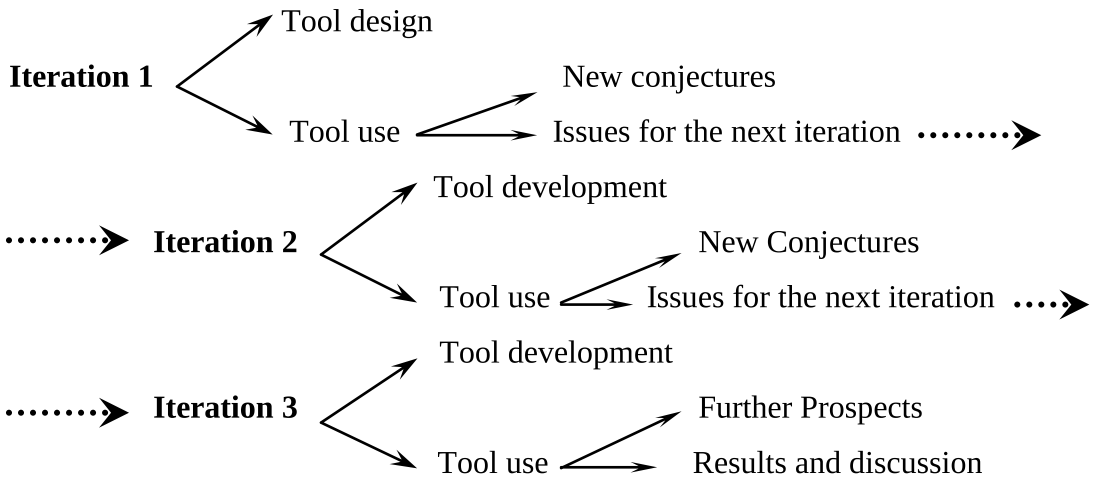


Figure 18- Three iterations of this research in the DBR framework

The figure below shows the cycle between the tool design, tool use, and the issues which emerge to modify and design the next iteration.

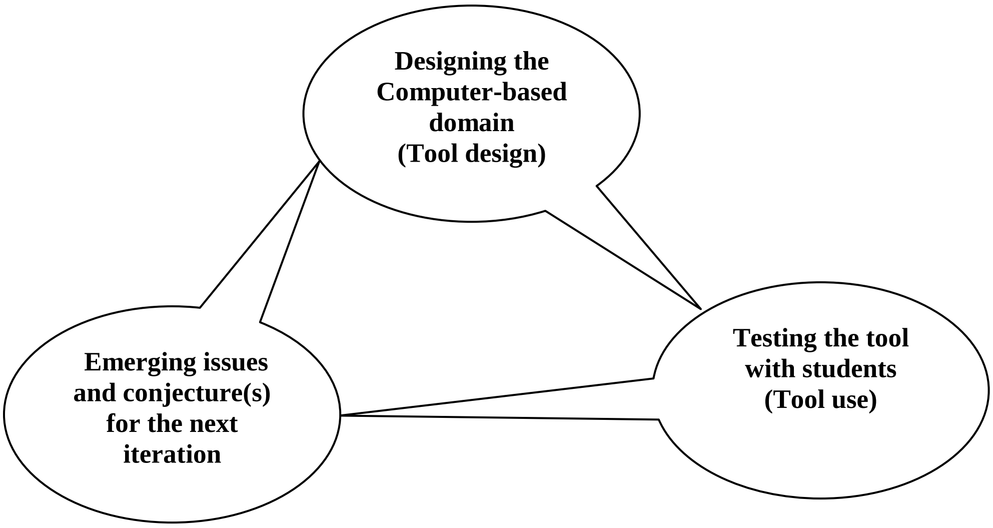

Designing the Computer-based domain (Tool design)
Testing the tool with students (Tool use)
Emerging issues and conjecture(s) for the next iteration

Figure 19- The cycle of design-test-modify and emerging issues-design

5.2. The Development of the Domains of Abstraction in a Nutshell

The first iteration, the design of the *Treemenders* domain, was based on the initial hypotheses and conjectures that arose from the literature and on my own instinctive feeling about the concept of recursion. The second iteration, the *Spirals* was a direct result of my new conjectures based on the emerging issues received through testing and analyzing the first iteration. Finally, the third iteration's computer-based domain, the *Treebuilder*, was a combination of the emerging issues all the way through the previous two iterations, which allowed me to frame a few additional conjectures and embody them in the *Treebuilder*. The following table summarises the main themes which were considered through developing the tool in the three iterations within the DBR framework.

	Iteration 1	**Iteration 2**	**Iteration 3**
Computer-based tool	*Treemenders*	*Spirals*	*Treebuilder*
Participants	2 pairs and 1 individual (2 Maths and 3 Computer)	3 pairs and 1 individual (all Mathematics)	7 pairs and 3 individuals (all Mathematics)
Main focus	Embedded recursion	Tail recursion	Embedded and Tail recursion
Functional abstraction	Functionality	Functioning and functionality	Functioning and functionality
Technical eminent	Interactive computer environment	Animative Visualisation	Animative Visualisation
Underpinning ideas	My initial conjectures and literature	My additional conjectures and ideas + the emerging issues from the previous iteration	My additional conjectures and ideas + the emerging issues from the previous iteration
Pre-questionnaire before working with the tool	Yes	Yes	No

Table 4- The major issues were considered in development of tools in three iterations

The next section of this chapter concentrates on the design development and tool-use perspectives of the first iteration of my research.

5.3. Iteration One – *Treemenders*

The *Treemenders* domain was designed based on what arose from the review of the literature which focused on students' difficulties in understanding and utilizing recursion. This iteration focused on discovering the students' actual reaction in working with recursion and to evaluate it with the issues that came out of the literature I had reviewed. The literature provided me with a first impression and insights into how mathematics and computer sciences students might work with recursion. The main attention in this iteration was paid to inspecting students' difficulties in understanding and applying the concept of recursion and their appreciation of the crucial components of the concept of recursion, the base case(s), recursive call(s), and flow of control.

These ideas were embodied in the *Treemenders* domain by designing and programming an environment in which students were able to generate binary trees by using an embedded recursive procedure. It has already been mentioned that, binary trees were chosen in this research as examples of fractals to provide a natural learning environment for students. From a functional abstraction view point, the focus in *Treemenders* was based mainly on the functionality aspect. The students who participated in the first iteration were able to change a few parameters in the procedure to generate their own binary trees. But, the domain failed to inform the students about the functionality aspect of recursion, to show them how the tree was being generated.

The parameters were purposefully chosen to allow the students to track and recognize the main components of the recursive procedures, such as the base case, stopping condition, and the recursive calls. In this way, they were able to make different binary trees, and by observing the shape and structure of those trees to work out the mechanism of a recursive procedure. However, after and during the testing of *Treemenders,* I was convinced that this version of the software did not have much to offer in terms of the functioning aspect of the learning. In other words, the participants were able to see *what* they were looking for and *what* was the result of the procedure by changing those parameters, but the software was not equipped with appropriate devices and design techniques to provide them with an understanding of *how* the recursive procedure works.

At this point it is possible to explain the main conjecture that was intended to investigate in the first iteration of my research as follows: By exploring key parameters in a recursive procedure, students will be able to contact the visual output from the procedure to procedure's code.

The next sections of this chapter discuss my approach in this iteration, tool development, and tool use of *Treemenders* domain to investigate the aforementioned.

5.3.1. My Approach - *Treemenders*

Theoretically, the main approach to the design of the domain of abstraction in this research is to provide a window by which one can understand the participant's thinking and the way that frames and shapes their mental model of the concept of recursion. Taking that into account, the approach of collecting data in iteration one was through using two common qualitative research methods, participant observation and semi-structured interviews. It has been mentioned in Chapter Four that these two qualitative data collection methods allowed me as the researcher to work out the participant's understanding of the concept of recursion and its essential components by looking through the window of the *Treemenders* (Bryman, 2001; Wilkinson, 2000; Barab and Squire, 2004).

Five volunteer students participated in this iteration, two first year mathematics students and three second year computer sciences students. Except one of the mathematics students, the rest were familiar with Logo programming. In the case of that mathematics student who had no familiarity with Logo programming language, I gave them a brief instruction about the commands that were used in the embedded recursive procedure. As a participant observer I avoided as far as possible any judgment based on my understanding of the utterances of interviewees, and just encouraged the students to give clear and transparent reasons for their decisions by asking open-ended questions as well as some questions about what they were thinking to figure out the way that they shape and form their understanding of the concept of recursion. All the interviews were audio taped.

Each of the interview sessions lasted about one hour. The interview sessions were started by giving students a pen and paper task (the pre-questionnaire) in the format of a semi-structured interview[1]. In those tasks, students were given photographs of two trees (Figure 20) and a mathematical fractal, Koch curve (Figure 21), and were asked to describe the shapes and their structures. The purpose and objective of this pre-questionnaire was to see whether they could see any structural parameters like symmetry or self-similarity in those pictures. I also asked them how they would have gone about modelling and making a tree if they wanted to program it. This open-ended question was designed to evaluate the student's mentality about the concept of recursion in the fractal structures – because these structures can only be defined recursively – and also the student's ability to apply recursive procedures in problem-solving situations. Regarding the Koch curve task in this module, the students were expected to describe how such a shape can be constructed and if they wanted to program it how would they do it and what are the essential components for such a computer-based task.

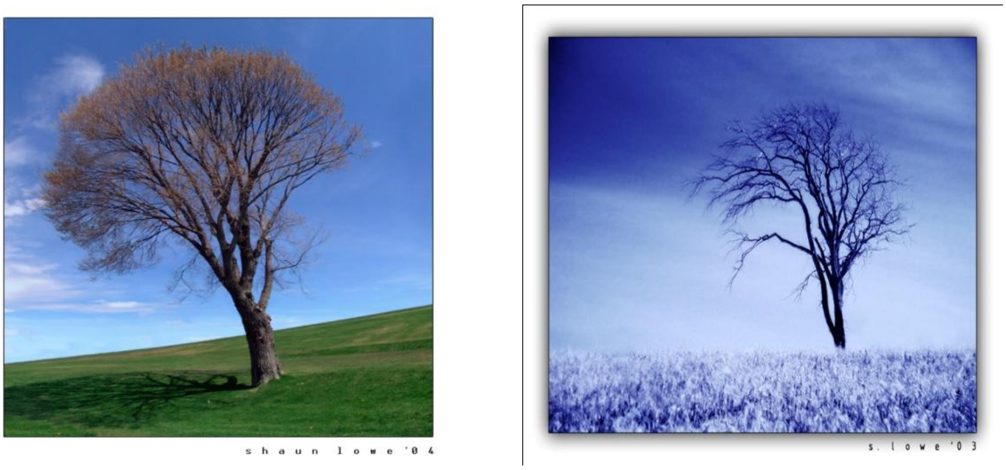


Figure 20- The images of the trees ((a) on the left, (b) on the right side of the page) in iteration one

[1] Pen and paper tasks are located in appendix A

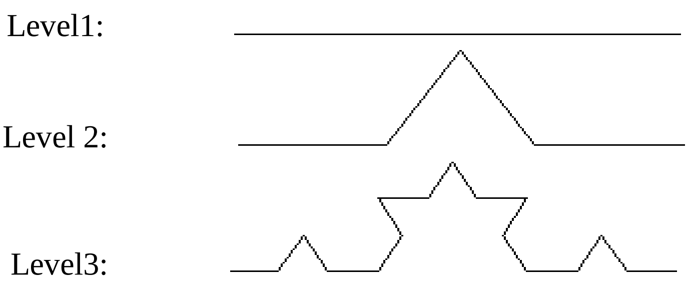
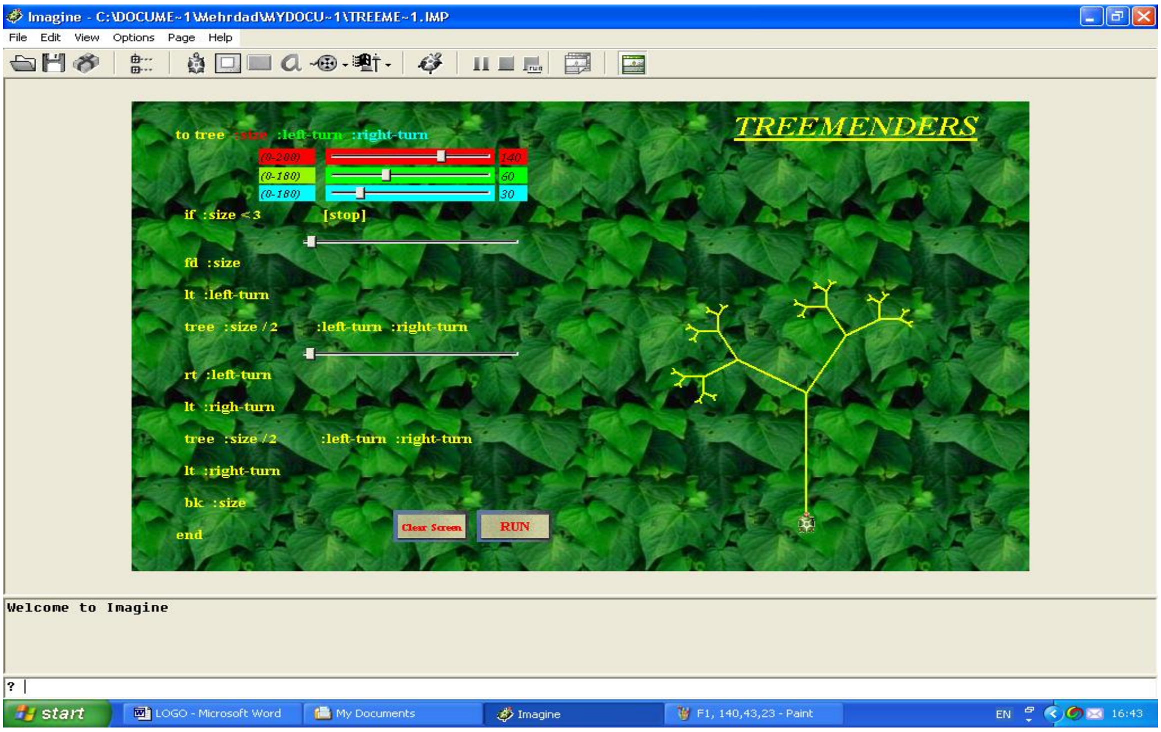


Figure 21- Image of the Koch curve in iteration one

Then the students were asked to start to work with the domain of abstraction – *Treemenders* – the main interface of the *Treemenders* is shown in Figure 22. The other rationale for using fractals stemmed from Wiedenbeck’s (1988) study of the role of everyday life analogies in students’ understanding of the concept of recursion. Wiedenbeck (1988) and Harvey (1997) mention that having everyday analogies might facilitate students’ understanding of the concept of recursion.



```
to tree :size :left-turn :right-turn
  if :size < 3          [stop]
  fd :size
  lt :left-turn
  tree :size / 2      :left-turn :right-turn
  rt :left-turn
  lt :righ-turn
  tree :size /2       :left-turn :right-turn
  lt :right-turn
  bk :size
end
```

```
Welcome to Imagine
```

Figure 22-The main interface of the *Treemenders*

The embedded procedure that was given to the students in *Treemenders* is shown below.

```
To Tree :size :left-turn :right-turn
If :size < 3  [ STOP ]
Forward :size
Left turn :left-turn

Tree :size / 2 :left-turn :right-turn        (the first recursive call)

Right turn :left-turn
Right turn :right-turn

Tree :size / 2 :left-turn :right-turn        (the second recursive call)

Left turn :right-turn
Back :size
End
```

Program 5-The embedded recursive procedure in the *Treemenders*

The next section of this chapter concentrates on the tool design issues of the *Treemenders* domain.

5.3.2. Pen & paper task – Iteration One

In the pen & paper task (the pre-questionnaire) the students showed strong evidence of a tendency to use iterative thinking to describe the given images of two natural trees and Koch curve (figures 20 & 21). Their prior knowledge and experiments with iterative procedures caused them to see recursion as iteration.

For instance, the second question on the pre-questionnaire was designed to see whether the students recognised the recursive structure of the Koch curve. Students were given the image of the first three levels of making a Koch curve. This question had two parts: the first part asked about the construction of the Koch curve, and the second part asked about the how to move from one level to another (figure 21). First year mathematics students, Sarah and Jin (who participated as a pair) both responded as follows:

1. I asked: How are these levels related to each other?
2. Sarah: Each level becomes one of the parts of the others. Umm, level one repeated four times to build the structure of level two, and level two builds level three, and so on.
3. Jin: If we call level one X, then level four will be four times X and the other levels can be constructed similarly.
4. Sarah interjected: We can go from level two to level four directly by repeating the whole of level two on each of these little pieces to make level four.

Sarah in line 4 directly pointed that level four is twice repetition of level two. Similarly, in the line 3, Jin described that level four is four times level one!

A second year computer sciences student, Feng (who participated individually) evidenced a stereotype understanding of the concept of recursion. He initially, pointed to the complexity of the Koch curve fractal. Feng's explanation of the Koch curve seemed to be based on a naive mathematical analysis. For him every sharp corner on the Koch curve was a vertex.

5. Feng: It is becoming more complex as we are going down. The number of vertices is increasing.
6. I asked: What do you mean by vertices?
7. Feng: The sharp points on the curve; this shape is tending to become a curve with no sharp points - a smooth curve with no vertices.

When Feng was asked to program the Koch curve, as mentioned above, he did not use any recursive techniques and he was trying to find a way to draw a smooth curve. Feng's explanation did not give much to see how he thinks about the concept of recursion. So, I decided to ask him to explain how he would program the factorial of a given number. His response showed the stereotypical picture of recursion in his mind as he immediately replied: "*the factorial of a natural number is recursive*". He was not able to recognise any recursive structure in the Koch curve even though he stated that factorial programming is recursive:

8. I asked: How would you program the factorial of a natural number?
9. Feng immediately answered: the factorial of a natural number is recursive.
10. I asked: What do you mean by recursive?
11. Feng: It calls itself each time.
12. I asked: Do you think we can use recursive here for the Koch curve?
13. Feng: Umm, I don't think so.

I showed him a picture, in which the recursive parts of the Koch fractal were shown with different colours (Appendix A)[1]. Feng's response showed that his difficulties to distinguish between iteration and recursion as he described the Koch curve as a FOR loop.

14. I added: What do you do to program it?
15. Feng: It is a FOR loop.
16. I asked: What do you mean by that?
17. Feng: It is repeating the same thing.
18. I asked: What is repeating?
19. Feng: It is like repeating one level in another level and another level repeats in the next one and so on.

It can be seen from the students' utterances that first year mathematics students (Sarah and Jin) were trying to describe the recursive structure of the Koch curve as an iterative, repetitive, structure (lines 1-4). Also, the second year computer sciences student (Feng) also described it as repetition (lines 15-19 and also lines 12-13). However, the situation for the other two second year

[1] It was conjectured that it can be used as a visual presentation of the Koch curve to facilitate students to recognise the structure of it.

students in computer sciences, Koroush and Yasaman (who participated in a pair) was a bit different. Koroush and Yasaman indirectly pointed to the calling process of the recursive call in a recursive structure (Lines: 20, 21, and 24 below):

20. Koroush: You start with the first level and reproduce the whole thing on each of the smaller segments. And on each edge you do the same.
21. Koroush also added that: You start with the straight line and split it into three parts recursively repeating it into new parts.
22. I asked: What do you mean by recursively repeating?
23. Koroush: I mean you can do the same level again but a bit smaller into another level.
24. Yasaman interjected and added that: Yes, I agree with Koroush about that. I think it's doing the same thing but each time a bit smaller than the previous one.

The comments made by Koroush and Yasaman above show that they recognised the recursive structure of the Koch curve. However, they were also describing the Koch curve by using the term "*repeating*" (Line 21). I was not yet convinced whether or not they were distinguishing between repetition (iteration) and recursion.). To make the distinction more perceptible, I asked them questions about the features and structure of the images of the natural trees, which they were given in the pre-questionnaire (figure 20a and 20b) and asked them to describe them. Initially, they explained the features of each tree in a descriptive manner.

25. I asked: What are the essential features of these trees if you want to model them?
26. Yasaman: A tree in the desert, lots of branches, with a big trunk. The other tree's branches look well ordered and a bit lopsided.
27. Koroush interjected: They have no leaves!
28. I asked: What about the structure of the tree?
29. Yasaman: It starts with a big trunk, but gradually become thinner and thinner at the top. There is a branching structure.
30. Koroush added that: It is symmetric. It looks symmetric from far away but not exactly. The general shape looks symmetric.

31. Yasaman interjected: It is almost symmetric, but not exactly. It seems that it has two parts, which are almost similar from far away.
32. I asked: Do you think these shapes are self-similar as well?
33. Yasaman: What do you mean by self-similar?
34. I answered: Something like a cauliflower, when you divide it into two parts, each one of those two pieces looks like a whole cauliflower but a bit smaller, so we can say this object is self similar.
35. Yasaman continued: I think it is self-similar, because of its branching system. Each time it makes two new branches and each of these new branches are going to have another two new branches, etc.
36. I asked: What if we have three or more new branches at each one of the branching points?
37. Yasaman: No difference, it is going to generate three or more new branches each time.
38. Koroush interjected: The main things are the main trunk, branches and the further branches - and bearing in mind that they are getting smaller and smaller on top.

At this moment Yasaman’s remark shows evidence of thinking-in-change about the structure of the branches and the possible strategy for programming that tree. Her thinking about the structure of the tree and the branching system began to be framed earlier, as can be seen in lines 29 and 35-37.

39. Yasaman: The categorization of branches is a bit difficult. A set of branches and a subset of tree are going to be made again and again.
40. I asked: What do you mean by a set of branches and its subset?
41. Yasaman replied: Each branch has some new branches and each of those new branches have some new branches again. So, they can be considered as a set of branches and the subset of the new branches.
42. I asked: How do you program it?
43. Yasaman: I would probably use the technique for sorting of a set of numbers.
44. I asked: Why?
45. Yasaman replied: I would like to sort this set of branches and its subset of new branches.
46. Koroush interjected: Draw one line here, and draw another line as its branch. Obviously you can choose a random point on the line to draw up the new branches and then repeat the whole process from the end to these branches with different angles.

47. I asked: What about the size of these branches? Are you going to keep them fixed?
48. Koroush added that: Ok, perhaps decreasing the size of the branches.

At that moment Yasaman, interjected and pointed to the base case.

49. Yasaman said: I think the base should be one branch and then make two new branches and then repeat the same process on each of the new branches. Basically, recursion is different from iteration, because of the base case. In recursion we start the process and then we get to the base case at the end. But in iteration we start with the base case!
50. I asked: Koroush did you consider any stopping condition for your algorithm?
51. Yasaman interjected: I think considering a base could act as a stopping condition.
52. Koroush: Umm, it could be. Also, we can put a condition on the length of size for stopping the algorithm.

It can be seen above that, Yasaman and Koroush explained an algorithm to model and program the trees. Principally, their algorithm was a recursive algorithm. However, working with these pen and paper (pre-questionnaire) tasks did not give me more information about their thinking about the interrelations of a recursive procedure and the way that control passes throughout calling recursive call(s) and reaching the base case(s). They also pointed to one of the crucial components of the concept of recursion, the base case(s), and its role as a starting point or a stopping condition (lines 49-52). Of interest point was Yasaman's idea of the differentiation between recursion and iteration based on the different functions of the base case as a stopping condition or as a starting point respectively.

Feng, another second year computer sciences student, responded to the same task with two images of trees, explaining them as follows:

53. Feng: This is a tree, it has no leaves, there are quite a lot of branches, and it's only one tree situated in the desert. The other one tends to the left side. It has smaller branches on the top. It is not high. The body is quite big.
54. I asked: Is there any symmetry in these trees?
55. Feng: Umm, not really, it's like an ellipse.
56. I asked: What are the main components of these trees if you want to program them?
57. Feng: The first thing is the root. It has the main body. From this main body you have at least one branch. Also, you can have more branches.
58. Feng also added that: If I want to make a computer program, I think of this shape because the tree will grow differently in any direction. So, in my program the majority of the branches should tend to the east side. The probability of the branching to the east should be bigger than in other directions. We need some coordinates on this graph.
59. I asked: What do you mean by coordinates?
60. Feng: A fixed point to measure the distance to other points on the tree.
61. I asked: Do you mean something like Cartesian co-ordinations?
62. Feng: Yes, and we also need some angles for branching. For the branches on the right side we might have the same process with different angles. Angle is the most common variable. I think we can chose random points on the main body for branching.

It can be seen that Feng never directly pointed to any recursive structure to model a tree in his comments. In addition, his remarks show evidence of taking recursion as iteration in lines. However, in line 62, there is some fragile evidence of the semantics of a recursive call being used to produce branches to the left and right by explaining the term '*same process*'. Lines 57-59 can be interpreted as his appreciation of the base case in the recursive structure as a starting point. However, the pen and paper task, as in the case of Yasaman and Koroush, cannot give much information about Feng's appreciation and thinking about the process of control passing.

Sarah and Jin, two first year mathematics students who at the start described the image of trees in slightly descriptive manner, like Koroush and Yasaman. However, gradually they tried to be more analytical about some structural aspects, for example, the direction and size of the branches and their angles. I asked them about the main features that needed to be considered to program the trees.

63. Sarah: Umm, construct the trunk, and the process of branching. Also the size of new branches.
64. Jin: The same things.
65. I asked: How about the angle of the branches?
66. Jin: I think for the first branch the angle should be less than 90 degrees and the second branch less than the first branch and so on.
67. Sarah: Oh yes, I agree with that.
68. I asked: Why do you think so?
69. Jin: The purpose of the tree is growing up and up.
70. Sarah interjected: Yes, in this way the branches growing upwards.

Line 63 shows that for Sarah and Jin the first challenging part of modelling a tree was branching. Sarah also pointed to the picture of the tree (a), the picture below, and added:

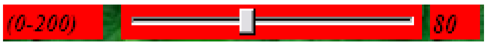
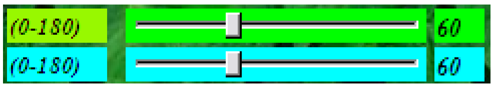
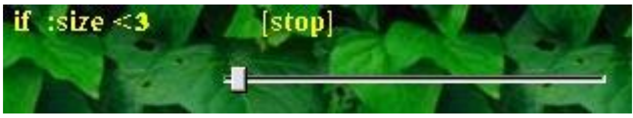
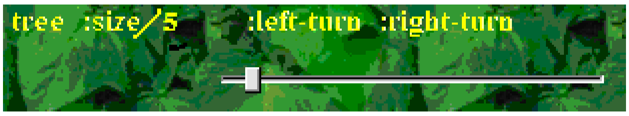
71. Sarah: I think this one is a little bit harder to program.
72. I asked: Why do you think so?
73. Sarah: Because the first one has a trunk and then branches. Whereas, in this photo we have not got trunk and just branching.

Lines 71-72 show a small amount of evidence of Sarah's thinking about the base case as the starting point to draw a tree. Sarah pointed to the main trunk in the tree in figures 20a and mentioned that the tree in figure 20b has no main trunk. She also added '*this one is a little bit harder to program'*, this gave me an initial insight into the importance of the base case in dealing with the concept of recursion for the students. Similar to the other participants, I did not find very much evidence about flow of control in any algorithm to generate the tree in the pre-questionnaire task. The next section of the chapter discusses on the students' accounts on the domain of abstraction (the *Treemenders*).

5.3.3. Tool design – *Treemenders*

The *Treemenders* domain was designed to model the binary trees, to bridge formal and informal. There were some sliders for setting the initial values for the size of new stems, angles to the right and left for making new stems, the size of the main trunk of the tree, and finally a minimum value for the size of new stems as the stopping condition. Based on the main conjecture of the first iteration, the components of recursion were phenomenalized using these on screen objects. It was conjectured that having these sorts of control over these parameters allow students to appreciate the interrelations between components of recursion.

For instance, by increasing/decreasing the minimum size of the new branches in the stopping condition, the number of new stems will be decreased/ increased respectively. Having a new stem means the procedure has called one of its recursive calls. Therefore, the minimum length of the size of the new stems as the stopping condition has a reverse relation with the number of times that the procedure calls its recursive calls. It was conjectured that, providing this level of playfulness will help students to see the hidden mechanism of the flow in a recursive procedure.

The *Treemenders* mainly presented an embedded recursive procedure to generate binary trees. The embedded recursive procedure contrived in *Treemenders* had five variables of: *size* , *right* and *left-turn* (for angle to the right and left) , *stopping condition*  if :size <3 [stop], and finally *the change rate* of the size of the new stems  tree :size/5 :left-turn :right-turn.

The pre-made program was depicted on the left side of the screen. The students were able to see the output of the program on the right side of the screen. The students were able to generate their own binary trees by setting those initial values and see the output on the screen immediately after running the procedure using the tools contrived in the *Treemenders* domain.

The students were not able to change the programming codes and syntax. However, they were offered to have control over the above mentioned parameters. The students were required to generate their own binary trees by changing those parameters. The sliders were designed and contrived in the software to point to the essential components of a recursive procedure. The sliders for the size of the main branch and the value for the stopping condition were contrived to evaluate the students' appreciation of the base case as one of the essential components of the concept of recursion. Haberman and Averbuch's (2002) research ascertained that students have difficulties with base case(s), whether it is a starting point (the simplest form of the problem), or a stopping point. The students were expected to realize that changing the stopping condition value causes a smaller number of new stems and, as a result, a smaller number of calling of the recursive calls. It was also conjectured that it might help them to track the flow of control over the procedure by combining those syntactical commands with the picture of the binary tree and its structure on the *Treemenders* window.

The other important issue about student's difficulties with the concept of recursion that was mentioned in the review of the literature was flow of control. Kurland and Pea (1985) introduced active and passive flow of control, depending on forward flow, when the procedure is calling the recursive calls, and backward flow, when the procedure was terminating the already generated copies of the original procedure. It was conjectured that the students might appreciate the complicated control passing process in the recursive procedure by embodying the idea of the connection between the stopping conditions and seeing the final outcome on the screen. This idea was put into *Treemenders* by designing two sliders for the angles to the left and to the right alongside the stopping condition and the change rate of the new stem sliders. The other design issue in this iteration was that the students had the opportunity to observe and scrutinise the interrelationships of those parameters with each other by changing the sliders.

Using-before-knowing as one of the constructionist ideas in design was also employed in the design of the first iteration's domain. The students have not been told that they are working with a recursive procedure. They were expected to work with the phenomenalized tools which enable them to work with the concept of recursion even though they do not as yet have a good understanding of it. The next section of this chapter focuses on the usage aspect of the *Treemenders* domain. This section discussed the students' engagement with the domain.

5.3.4. Tool use – *Treemenders*

This section concentrates on the usage aspect (tool use) of the *Treemenders*. Working with the *Treemenders* offered the students the opportunity to work interactively with a recursive procedure and produce their own binary trees. It has been mentioned above that they were given control over a few parameters and variables through sliders, which were designed to attract the students' attention to some key components like base case and flow of control. It was conjectured that an appreciation of these issues is essentially important for understanding a recursive procedure.

Since the first iteration was an exploratory iteration, the main focus and emphasis was putting the computer-based approach into action to compare the reality and the issues from the literature and my conjectures about students' difficulties with the concept of recursion. In this way, I was able to work out the future design issues and the problems on which I needed to concentrate.

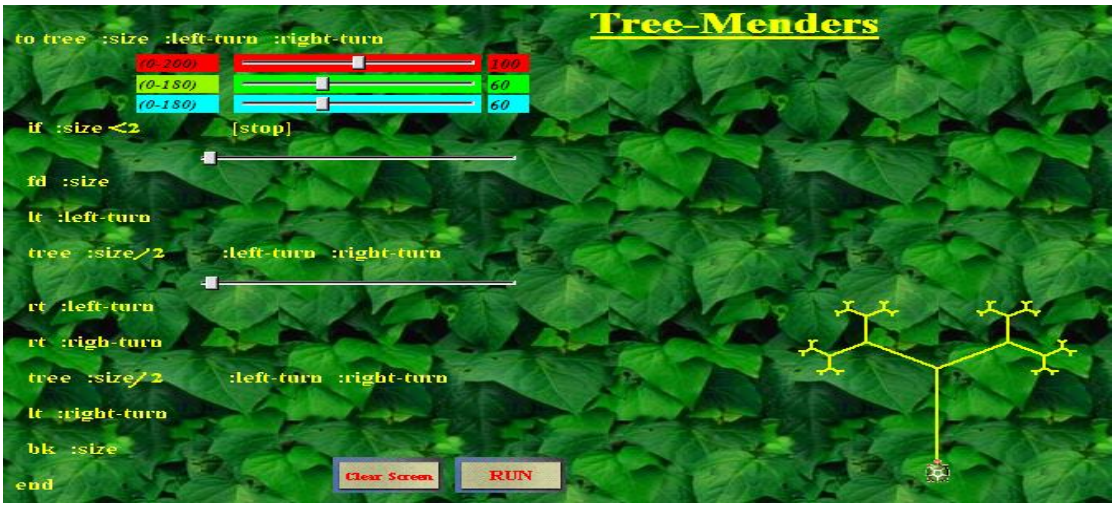
In this iteration, the students were given a pre-made embedded recursive procedure to generate a binary tree, so they had no opportunity to create their own algorithms for generating a binary tree. Therefore, the design did not give me much information about the student's understanding and thinking about the way that the trees were being generated by the recursive procedure and also their ability to apply the concept of recursion as a problem-solving strategy. These points will inform the next iteration of the design. Through working with the control sliders, the students succeeded in finding the interrelations between the crucial parts of a recursive process. However, they still had some difficulties with recognition of the flow of control, the role of base case, the position of the recursive calls in the program and their functions.

The next parts of this chapter focus on the particular aspects of the tool use aspect of *Treemenders*. Tool use aspect is discussed in two parts, part one focuses on the students' results in the pre-questionnaire and the second part examines the students' experiment with the *Treemenders* domain of recursion abstraction. In accordance with the aims of the study in both those parts the main attention is on the structure of the concept of recursion. This strongly depends on the students' difficulties with recognition of iteration and recursion. The students' understanding of recursion is classified into two categories of recursion vs. iteration and the flow of control. These categories are thoroughly discussed in the next sections of the chapter.

In all three cases, a pair from mathematics, a pair and one individual from computer sciences, after finishing the semi-structured pre-questionnaire (pen and paper task), we moved to work with the computer-based tool (the *Treemenders*). I began the task by giving them a short and succinct introduction to the software and the tools. Focus of attention was directed at the student's appreciation of dispensable components of recursion and their ability to track the control passing process over the given procedure. Whilst working with the pen and paper, the second year Feng stated that the Koch fractal is a FOR-loop because '*it is repeating the same thing*' (lines 15-19) and it is not a recursive structure (line 13). He also displayed a stereotypical image of the concept of recursion (lines 9-11), when he stated that the factorial of a natural number is recursive. Having given all these explanations, he started to work with the explanatory domain of abstraction for the first iteration, the *Treemenders*. He had difficulty understanding and applying the recursive calls in the procedure. So he started with changing the angles which seemed to be more tangible parameters.

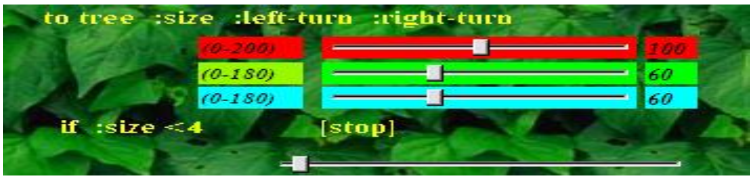
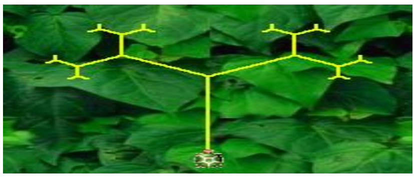
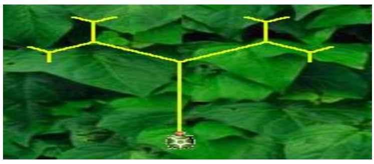
74. Feng: I would like to start with changing the angles.

Both left and right angles equal 50, the stopping condition slider was already fixed on two, and the rate of change of the new stems was also fixed on two as the default value (Figure 23).



```
to tree :size :left-turn :right-turn
      (0-200)  100
      (0-180)  60
      (0-180)  60
 if :size <2          [stop]
 fd :size
 lt :left-turn
 tree :size/2      :left-turn :right-turn
 rt :left-turn
 rt :righ-turn
 tree :size/2      :left-turn :right-turn
 lt :right-turn
 bk :size
end
```

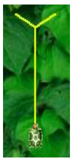
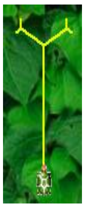
Figure 23- Feng's experiment with angles (both 60 degrees)

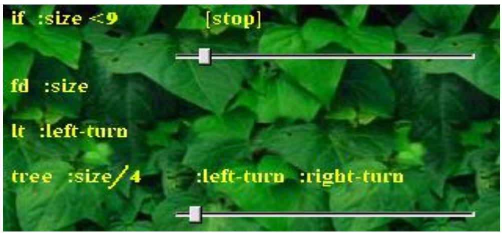
Then Feng tried to change the value of the stopping condition from less than two to less than four , the result was a tree with fewer branches . Then he tried it with a size less than nine, and the final result of the binary tree showed that the number of the branches decreased . He kept silent and worked carefully with different values for the stopping condition.

```
to tree :size :left-turn :right-turn
      (0-200)  100
      (0-180)  60
      (0-180)  60
 if :size <4          [stop]
```

The explanatory domain in the first iteration assisted Feng to recognise some level of connections between the components of the recursion. The following lines show how Feng appreciates the connections between the size of the stems in the stopping condition and number of new stems in engagement with the *Treemenders* domain.

75. Feng: I think the bigger stopping condition makes a lesser number of new branches.
76. I asked: Why do you think so?
77. Feng: I set the angles both equal to 60, when the stopping condition was two I had five new branches. For a size of less than four, I had four new branches and now when I set the stopping condition to less than nine, I only have three new branches.
78. I asked: Do you think the size of the new branches is important as well?
79. Feng: Umm, It could be.

In the line 77 Feng directly pointed to that relationship. Then, he tried to work with the slider of the rate of change of the new stems. He changed it to 4, the angles to the left and right were fixed at equal to 60 and the stopping condition was nine. The final output of the tree only had one new stem . Then he changed the value of the stopping condition to 4, and the final result had one more stem . Feng continued to try a few different values for the size of the main trunk and stopping condition. The following lines show Feng's appreciation of the size of tree and the initial value of the size.



```
if :size <9          [stop]
fd :size
lt :left-turn
tree :size/4         :left-turn :right-turn
```

80. Feng: I think by increasing the size to 200 I have bigger branches and by decreasing the stopping condition I have more branches, and if I increase the ratio of the size it decreases the number of branches.
81. I asked: Can you tell me the order of execution of the lines in the program? [see program 1]
82. Feng: It begins with those three values, the size and the angles. Then it checks if the size is less than 2 or not, if the size is 100 so it does the rest of the lines.
83. I asked: How about '*Tree :size/2 :left-turn :right-turn*'?
84. Feng: Umm, it is just changes the length of the new branches.

Lines 83 and 84 showed that Fend considered the recursive call as a variable to change the length of the size for the new stems. That can be considered as a sign of iterative thinking rather than recursive. Feng's explanation in lines 81-84 shows that he has a sequential interpretation of control passing in the given recursive procedure. It has already been mentioned above that Feng said that the factorial of a natural number is recursive, but that a Koch fractal is a FOR-loop. To illuminate his idea about recursion and loops I tried to understand whether he could see any recursive structure in the given procedure.

85. I asked: You said the factorial of a natural number is recursive.
86. Feng immediate answered: Yes!
87. I continued: Why do you think so? What makes the factorial recursive?
88. Feng: It calls itself again and again.
89. I asked: What's the difference between recursion and a FOR-loop?
90. Feng: Recursion is a WHILE-loop!

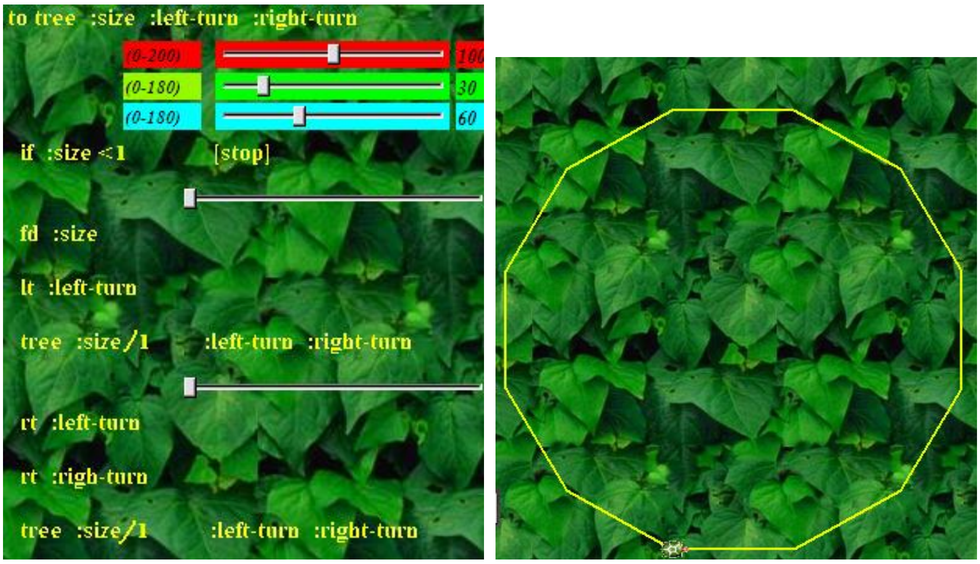
Feng's description of the recursion in line 88 and line 90 is evidence that he has a stereotypical understanding of the concept of recursion. Also, the design in this iteration was a kind of exploratory domain so it did not allow him to see the latent layers of the complicated control passing in a recursive procedure.

Sarah and Jin, the pair of Mathematics students, also started to work with *Treemenders* in a similar way to Feng. They began by changing the values for the angles to the left and right and observed the final output with scrutiny. The interesting point with them happened when they made an infinite loop.

91. Jin: I would like to change the stopping condition to one.
92. Jin: Ok, let's change the length of the new branches equal to the original one. By setting it to *size / 1*.
93. I said: Let's see what the result will be.
94. Sarah and Jin: Oh! It is not stopping at all.
95. Sarah: What if I increase the stopping condition value.
96. Jin: I agree.
97. Sarah: No difference!
98. Jin: When we have *size / 1*, changing the other values like stopping condition doesn't seem to make any difference.
99. I asked: How about the size?
100. Sarah immediately answered: It makes the bigger circle shape
101. Jin: Yes, let's change the angles to see whether they make any difference?
102. Sarah: Let's take size 60, left angle say 120, and right angle 90
103. Sarah and Jin: Oh! That's interesting.

Jin's innovative idea to make the rate of change of the new stems equal to the size of the main trunk created a new line to attack the complexity of the control passing process in the recursive procedures. The idea that I wanted to use at this stage was that, whether or not they reached the point in the procedure where these initial values will never call the second recursive call, I wanted to look into their work to see whether they noticed that by making the size of the new stems in a recursive procedure the same as the size of the main trunk, the first recursive call is going to call the whole procedure with the same initial value again and again.

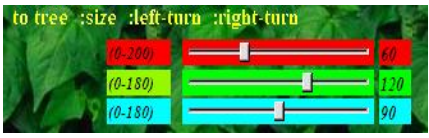
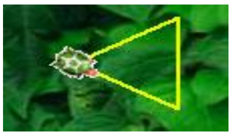
In fact, I wanted to check whether they realised that control will never pass to the second recursive call. In lines 92-94, they made the main size equal to 100, the angle to the left 30, and the angle to the right equal to 60. They also set the minimum length of the stems equal to one as the stopping condition. The change rate of the length of the new stems was set the same as the length of the main trunk. The result can be seen in the below picture.



```
to tree  :size  :left-turn  :right-turn
            (0-200)    100
            (0-180)    30
            (0-180)    60
if  :size <1           [stop]

fd  :size
lt  :left-turn
tree  :size/1      :left-turn  :right-turn

rt  :left-turn
rt  :righ-turn
tree  :size/1       :left-turn  :right-turn
```

Figure 24- Infinite loop by taking the length of the size of new stems equal to the length of the main trunk (lines 92-94 Sarah and Jin)

Then they changed the value of the angles to the left and right. That was a crucial moment to see whether they recognized the flow of control over the procedure or not.

The given procedure had two recursive calls to produce the branches to the left and right. The first recursive call in the procedure was programmed to generate branches to the left. By those initial values, which were chosen by Sarah & Jin, the procedure never calls the second recursive call, which means they had not have any branches to the right. And that is the main reason for having those anticlockwise polygon-shapes. I looked into their experiment to discover this point. In lines 102-103, by taking the main size 60, and the angles to the left and right equal to 120 and 90 respectively, Sarah and Jin achieved an anticlockwise infinite loop over a triangle  to tree :size :left-turn :right-turn (0-200) 60 (0-180) 120 (0-180) 90, . This result, and also their mathematical background, persuaded them to generate a few more different polygons. The outline of a few interesting trial and errors made by Sarah and Jin is presented in the below table.

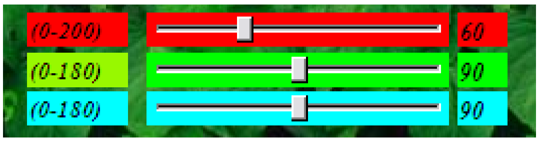
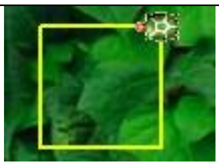
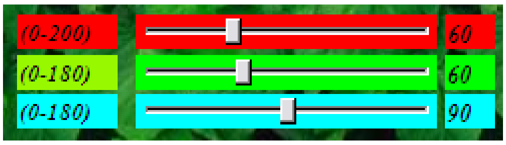
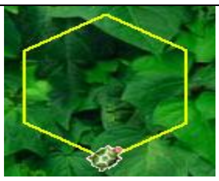
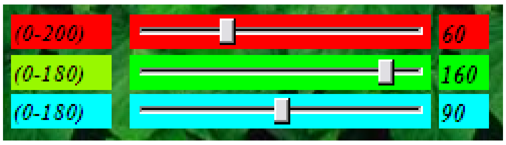
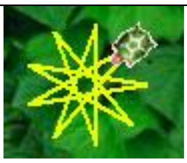
Initial values	Output
 (0-200) 60 (0-180) 90 (0-180) 90 Size 60, angle to the left 90, angle to the right 90	 Anticlockwise infinite loop over a square
 (0-200) 60 (0-180) 60 (0-180) 90 Size 60, angle to the left 60, angle to the right 90	 Anticlockwise infinite loop over a hexagon
 (0-200) 60 (0-180) 160 (0-180) 90 Size 60, angle to the left 160, angle to the right 90	 Anticlockwise infinite loop over a star

Table 5-Some interesting result from Sarah and Jin's experiment with *Treemenders*

104. I asked: What do you think about the value of angles in these shapes?
105. Sarah: Umm, I don't know!
106. Jin interjected: If you consider the square one, the angle is 90
107. I asked: How about the hexagon?
108. Sarah: Well, the angle to the right has no effect on the result.
109. I asked: Why do you think so?

I tried to attract their attention to the role and effect of the angles to the left and right and to see whether or not they would find that the program never goes to the right. So, I asked them to explain the role of the angles for me (line 104). Their immediate answer showed me the square with the degree 90 at all its vertices (line 106). I referred to the hexagon and Sarah replied that the "*angle to the right has no effect*" (line 108), and to justify her answer they moved back to the square one and tried to change the angle to the right to see the difference.

110. Jin: The result hasn't changed at all!
111. Sarah: Yes, I think it doesn't make any difference to the result!
112. I asked: Why is it like that?
113. Sarah: Umm, I have no idea!
114. Jin: It has never turned to the right at all! Why is it like that?
115. Sarah: Yes, that is why. But why doesn't it turn to the right then?
116. I replied: Well, that's what we are thinking about. What do you think?
117. Sarah: No idea why it doesn't go to the right side by taking that.
118. Jin: Umm, no idea.

They were stuck and were not able to explain and describe the strange behaviour of the procedure, which was a direct result of the complicated control passing process in the recursive procedures.

The following lines show their difficulty in recognition of the recursive call and the functionality aspect of it. In the line 121 Sarah pointed to the syntax of the recursive call and that she cannot understand it clearly.

119. I asked: What are the most important parts of this procedure?
120. Jim: I think the size of branching is really important.
121. Sarah: I think this line is dodgy.

Therefore, for Sarah and Jin the most difficult part to pass was to understand the recursive calls and their function. In lines 110-118, one can see that they had no idea about the suspension of the execution and jumping back to the beginning of the procedure, again and again, due to the control passing mechanism of the recursive procedures. However, lines 119-121 show evidence that they were aware of their inability to understand the function and functionality of the recursive call '*Tree :size/2 :left-angle :right-angle*' and that was the reason that Sarah called this command of the procedure the *dodgy* one (line 121). Jin also pointed out that the most important part of the procedure to be understood is the size of branching (line 120), which is exactly the recursive call.

In contrast to the other three students, Yasaman and Koroush, a second year computer science pair, recognised that the given procedure is a recursive procedure. Semantically, they explained that there are two recursive calls (lines 122-124).

The following lines demonstrate that the *Treemenders* domain offered Koroush & Yasaman the environment, in which they could see the connection s and interrelations between the stopping condition and number of new stems in their binary tree.

122. Koroush: This is recursive procedure.
123. I asked: What do you mean by that?
124. Koroush: Because it is calling itself.
125. Yasaman interjected: Yes, this procedure is calling itself. I will go for the angle first. Because, for making branches we should know how much we must turn to the right or left. So, branching is very important for me. The other point is that it cannot work

forever and we should consider a limit for it. Therefore, I am going to consider this limit as a stopping condition.

126. Koroush added that: I would like to get more iteration.
127. I asked: what do you mean by iteration?
128. Koroush immediately answered: By making the stopping condition small, it will iterate more and basically it gives us more branches.

Lines 125 and 128 give evidence of their ability to make the connection between the stopping condition, recursive calls, and the number of new stems. That was one of the beneficial points of employing the computer-based domain, which provided them with the opportunity of observing the syntactical commands and the output simultaneously.

The next section of this chapter discusses the result of the first iteration. Following that, is the section which introduces the issues which need to be reported on in the second iteration.

5.3.5. Discussion – *Treemenders*

I would like to discuss the results of the *Treemenders* from two parallel perspectives of design development and the tool use.

1) Discussion on the design development:

Basically, the first iteration was an exploratory level of design. The main attention in this exploration was paid to the students' difficulties in understanding the concept of recursion and its indispensable components. The initial hypotheses and conjectures were made based on the extensive literature reviewed and my own ideas about tackling such a research idea. Although the computer-based domain in this iteration facilitates the participant's recognition of some interrelationships between the internal parameters of the embedded recursive procedure, for instance see Sarah and Jin, lines (98-103) and Yasaman and Koroush, lines (126-128).

However, I was also convinced that *Treemenders* did not have the material abilities to resolve students' problems with the complicated control passing process in the recursive procedures. For instance, in Sarah and Jin's account (lines 110-118) they had no idea why the program never drew any branches to the right, in the other words why the procedure did not call the second recursive call at all. From a functional abstraction point of view, *Treemenders* was mainly based and designed from the functionality (what) part rather than the functioning (how) part. Hence, the students were able to see what was going to be done by the embedded procedure, the binary tree, but they were not able to see and find out how it was going to be done.

That was one of my main concerns in any effort to design the next iteration. It was also the most challenging part of the design of the next iteration. On the one hand, I had to improve the weaknesses of the *Treemenders*, and on the other hand was my interest in using fractals or fractal-shaped objects as the everyday life analogies to design an appropriate domain of abstraction. I had to consider the students and types of material that needed to be contrived into the software to provide them with an appropriate environment to work with the concept of recursion. The software needed to provide me as a researcher a window in which I can see how students think about the concept of recursion and the way that they form and build their mental model of this concept and its essential components. Overall, in the *Treemenders* domain, the concept of recursion was phenomenalized by modelling a binary tree. Trees were chosen to bridge a formal mathematical concept with informal fractal objects. This domain only provided the students with a limited level of playfulness and control over some parameters. Although it had weaknesses in presenting a deeper level insights to the students, however, it had a few promising strengths, the ability to show the connections between the different parameters of the procedure. It was this that persuaded me to improve it for the next iteration.

2) Discussion on the tool use:

From the tool use perspective, based on the aims of my research the main attention in this section is paid to the following issues: 1) The students' ability in distinguishing recursion from the familiar iteration concept. This issue is strongly dependents on their understanding of the mechanism of flow of control in the pre-made recursive procedure. 2) The students' appreciation of the main components of the concept of recursion.

Regarding the students' differentiation between the iteration and recursion, when they were working with pen and paper tasks (the pre-questionnaire), except the second year computer sciences pair (Yasaman and Koroush), the other students considered recursion as iteration. For instance, Sarah interpreted the Koch curve structure as an iterative structure by saying it is '*repeating*' the same thing (line 2 and line 4). Also, Feng considered the Koch curve as an iterative structure rather than a recursive one (lines 16-19).

Feng’s account showed a kind of stereotypical understanding about the concept of recursion by saying that the ‘*factorial of a natural number is recursive*’ (line 9). When I asked him what he meant by recursion, he answered ‘*it is calling itself each time*’ (line 11) and then he continued that the Koch curve does not have a recursive structure (line 13). His explanations convinced me that based on previous knowledge, he had some stereotypical idea about recursion – what is factorial? It is recursion! However, his understanding of the concept of recursion was not deep. In line 15, Feng added that the Koch curve is a *FOR*-loop.

Regarding the crucial components of the concept of recursion almost all of the students, who participated in the first iteration, appreciated the base case, although they did not consider it as a crucial component of recursion.

In the account of Sarah and Jin (both year two mathematics), in lines 71-73, Sarah states that modelling of the image of a tree in figure 20(b) is harder than modelling the image of a tree in figure 20(a) because it has no main trunk and starts with branching. Feng (computer sciences year two) also in lines 57 and 60 pointed to the need for having a base case. In both cases, the participants pointed to the base case indirectly, and also they did not consider it as a component of a recursive process.

However, in the case of Yasaman and Koroush (working in a pair, computer sciences year two), the story was a bit different. In line 20 Koroush, and Yasaman in line 29, pointed to the need for the base indirectly by saying that to model a tree you need to start with a main trunk, for me that was initial thinking about the existence of a base. Then, Yasaman in lines 49-51 made an algorithm for generating a tree and directly pointed to the base case by using the term '*base*'. Moreover, she distinguished between recursion and iteration through a different interpretation of base case. Yasaman stated that: *"[i]n recursion we start the process and then we get to the base case at the end. But in iteration we start with the base case*".

Her explanation was important for me in four directions. The first direction was, for her, iterative procedures and recursive procedures were different. The second direction was that she had a measure for that separation which was base case. The third and most important direction was that she considered the base case as a component of recursive procedures. And the final direction was her initial idea about the base case in recursive procedure, as she thought that the recursion ends with the base case. When she stated that "*I think considering a base case could act as a stopping condition*", Koroush replied that "*It could be. Also we can put a condition on the length of size for stopping the algorithm*" (lines 51-52). These explanations also showed me that although they agreed that in a recursive procedure the base case can act as a stopping condition, however, they were not sure whether putting a limit on the length of the size is stopping condition or base case. Their explanations provided me with valuable insights for the design of the next iteration, which will be discussed in the next section of this chapter.

The pen and paper task, as already mentioned, said little about the process of flow of control. The students' experiments with the domain of abstraction (*Treemenders*) provided me with new intuitions and insights to find an appropriate embodiment and phenomenalization for the control passing process in recursive processes in the next iteration. The students had great difficulty in recognizing the flow in the given embedded recursive procedure. Only Yasaman and Koroush showed an initial understanding about the flow in the recursive procedures. The other students considered a sort of sequential flow over the embedded recursive procedure.

Overall, at this exploratory level, I achieved some insights and conjectures about the embodiment of the complicated control passing flow in the recursive procedures and thinking about the functioning aspect (*how* part) of the design for designing the next iteration, which is going to be explained in the next section.

5.3.6. Issues & conjecture(s) for the next Iteration

The results which were achieved by the students through using the *Treemenders* were promising for the future work in the next iteration. Because of the nature of the query, achieving these initial results by using traditional methods at this stage was difficult, if not impossible.

The design of this iteration supports the accuracy of many issues about students' difficulties with understanding and applying the concept of recursion that had been reviewed in the literature. Issues include students' difficulties in recognition of the base case, flow of control, and the function of recursive calls. However, the computer-based tool requires the question of '*how'* the complicated mechanism of flow in recursive procedures can be visualized and embodied in the design. How the design can be improved, to allow students in a domain of abstraction environment, to enhance and develop their understanding of the concept of recursion. How this understanding can assist them to frame and shape their own mental models of this concept through interaction with the tasks contrived in the domain. It was visibly apparent, and I was convinced that the current domain in the first iteration (*Treemenders*) did not guide the students' attention towards tracking the complex control passing process in an embedded recursive procedure.

I decided to concentrate on a more visible visualization of the concept of recursion. Based on the *Treemenders* early results relating to the students' difficulties in recognition of the relation between recursion and iteration, which directly stemmed in inadequate knowledge of control passing mechanism in recursive procedure, I decided to use some fractal-shaped objects, which can be described both iteratively and recursively, to scrutinize the students' appreciation of recursion and iteration. I use the term 'fractal-shaped object' in order to distinguish them from fractals. Fractals can only be defined recursively and that was the reason that I was looking for objects that can be programmed both iteratively and recursively.

That was one of the crucial challenges of the design of the next iteration. Based on my initial interest on using fractals and fractal-shaped objects I decided to use spirals. They look like fractals, but they are not because they can be described both iteratively and recursively. Students' difficulty with tracking the flow in the embedded recursive procedure was the other major challenge which needed to be improved in the second iteration. In the case of Yasaman and Koroush, I noticed that they generally were able to describe the mechanism of the recursive procedure (lines 122-127). However, when it came to describing the functioning of the recursive calls in particular – the first and second recursive calls – they did not have much to say in the first iteration. Consequently, the new conjecture was the separation of tail and embedded recursive calls for two reasons. The first reason was that a tail recursive procedure is very similar to an iterative procedure. So, it would help me to look through the student's mental model about iteration and recursion from a closer perspective.

Also, based on the initial result of *Treemenders* it was apparent that students have less difficulties in working with trees as they already have a kind of pre-made mentality about them as example of the objects in our everyday life. It persuaded me to phenomenalize the concept by using spirals, to bridge formal and informal, and embody and visualise the tail recursive procedures in the next iteration.

In addition, functional abstraction is a key concept in dealing with recursion. From this perspective, the other major issue which needed to be considered in the next iteration was the lack of enough attention on the functioning (*how part*) in the design process of the first iteration. I realized that I needed to employ a more dynamic and interactive domain of abstraction environment to provide the students with the opportunity of seeing the latent layers of the complicated control passing process in the tail and embedded recursive procedures.

I decided to put more focus on the functioning alongside the functionality dimensions. In light of these results, the next domain, which is called *Spirals*, was designed in such a way that students were able to engage with recursion at two levels, namely 'functionality' and 'functioning'. It means that, in this domain of abstraction, the students were able to switch their thinking between functionality and functioning levels of recursion.

It has been mentioned before that, according to Sooriamurthi (2002), one of the major difficulties students face concerning the concept of recursion arises from focusing on the *how*. In contrast, he believes that they should focus on the *what* first, and then focus on the *how*. The primary result of *Treemenders* reveals that only focusing on the *what* part and ignoring the *how* part did not give much chance for the participants to track the flow of control. Hence, I decided to employ them shoulder to shoulder in the next design of the software. Therefore, the main issues for the next iteration can be summarized in the three major issues of: separation of tail and embedded recursion, focus on the functioning aspect of functional abstraction as well as functionality aspect, and using spirals for the embodiment of tail recursive procedures in the computer-based domain of abstraction.

The first design was mainly based on drawing a binary tree using some parameters. In order to design and make a computer-based domain to act as a window to look into student's thinking and thinking-in-change about the concept of recursion and its components. Therefore, my main focus was on creating an appropriate computer-based design which is both purposeful and dynamic. This responsibility drew my attention to the contextualization of a tail recursive procedure by spirals. I also designed a new task in which students have the opportunity to engage with the software by entering their own commands into the task and observing the results.

Having said the issues above, the conjecture which was reported into the second iteration can be explained, as follows: By exploring the similarities and differences between iteration and tail recursion, students will recognise the flow of control in tail recursion. The aspiration is that awareness of flow of control in tail recursion may later support appreciation of the flow of control in embedded recursion.

I designed and coded the *Spirals* as the domain of abstraction for the second iteration. The next section of this chapter concentrates on the second iteration.

5.4. Iteration Two – *Spirals* and *Blank box*

5.4.1. Overview

The domain of abstraction in this iteration is called *Spirals*. In this iteration, the main attention was paid to the modelling of spirals. The rationale for choosing spirals stemmed from the initial interest in using fractals and fractal-shaped objects and also the challenge to find objects that can be modelled both iteratively and recursively.

I used these fractal-shaped objects to embody the conjectures on the relationships of the iteration and tail recursion in the computer-based domain of abstraction called *Spirals* in the second iteration. I also intended to examine the possible influence of previous experiments with iteration on the students' thinking-in-change process about the concept of tail recursion. This domain of abstraction acts like a window for me as a researcher to look through the process of student's thinking-in-change and also for the students as participants to work with this domain and try to frame and shape their thinking and mental models of recursion through this window. The *Spirals* domain was designed in such a way that the students were able to create their own spirals using two iterative and recursive techniques. These techniques were called *blue* and *red* techniques respectively (Figures 26 and 27), to avoid the creation of any sort of prejudgments by the participants.

The *Spirals* domain of abstraction also allowed the students the opportunity to compare these two techniques in a *comparison* module (Figure 28). I employed a few visualisation techniques in the *comparison* module, like colour-coding and animation. The main reason to go for this challenging part of the design was the noticeable need to pay attention to the functioning aspect of the mechanism of recursion as a result of the first iteration. It was conjectured that it might provide a better window for the participants to look through the complicated mechanism of the concept of recursion and its crucial components.

I called this innovative approach towards investigation of the student's thinking-in-change and their own mental models of the concept of recursion, ***A**nimative **V**isualisation in the **D**omain of **A**bstraction,* abbreviated to the AVDA approach. AVDA provided the students a dynamic visual presentation of the viable *copies* model of the tail-recursive procedure. Computer-based environments are appropriate domains of abstractions (Pratt *et al,* 2008) to use with the advantage of providing a window to look through the students' thinking process about different mathematical concepts. The intention was that, by designing such windows, I could as a researcher probe the students' thinking and thinking-in-change process more easily. Moreover, it might facilitate students to frame and shape their thinking about the concept through this window.

The results of this iteration provided me with rich insights into the role of dynamic visualisations, AVDA, in this study. These results are discussed in the issues for the next iteration later in this chapter. The pictures below (Figures 25-28) show the main interface of the *Spirals* computer-based domain of abstraction for the modelling of spirals.

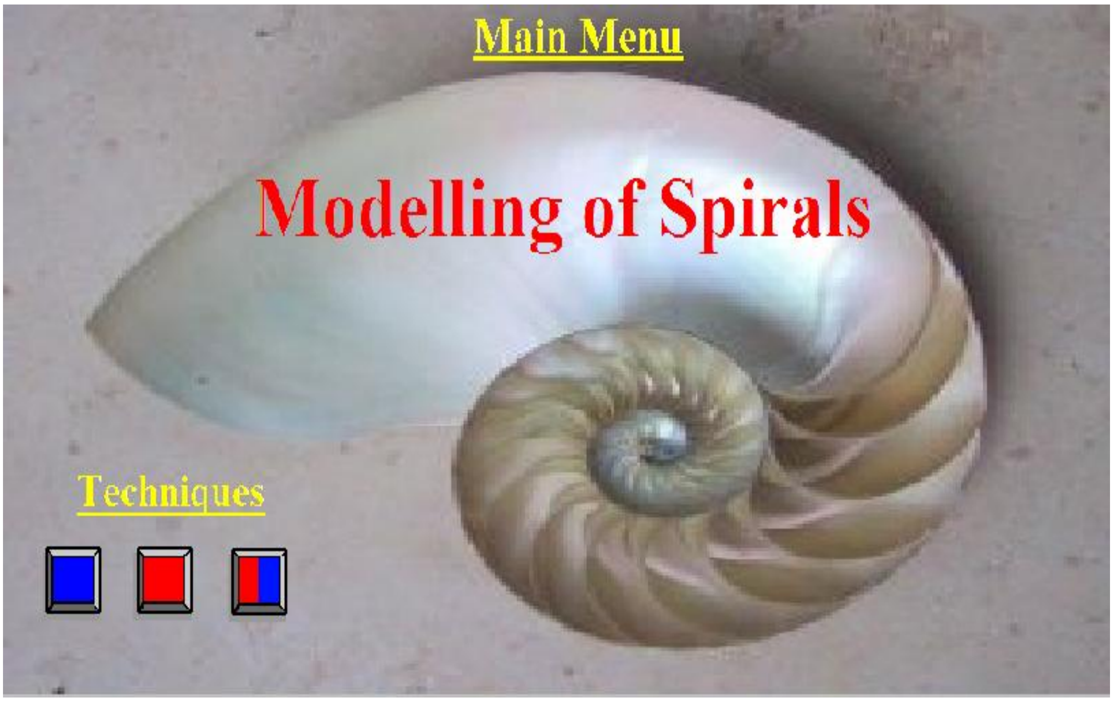
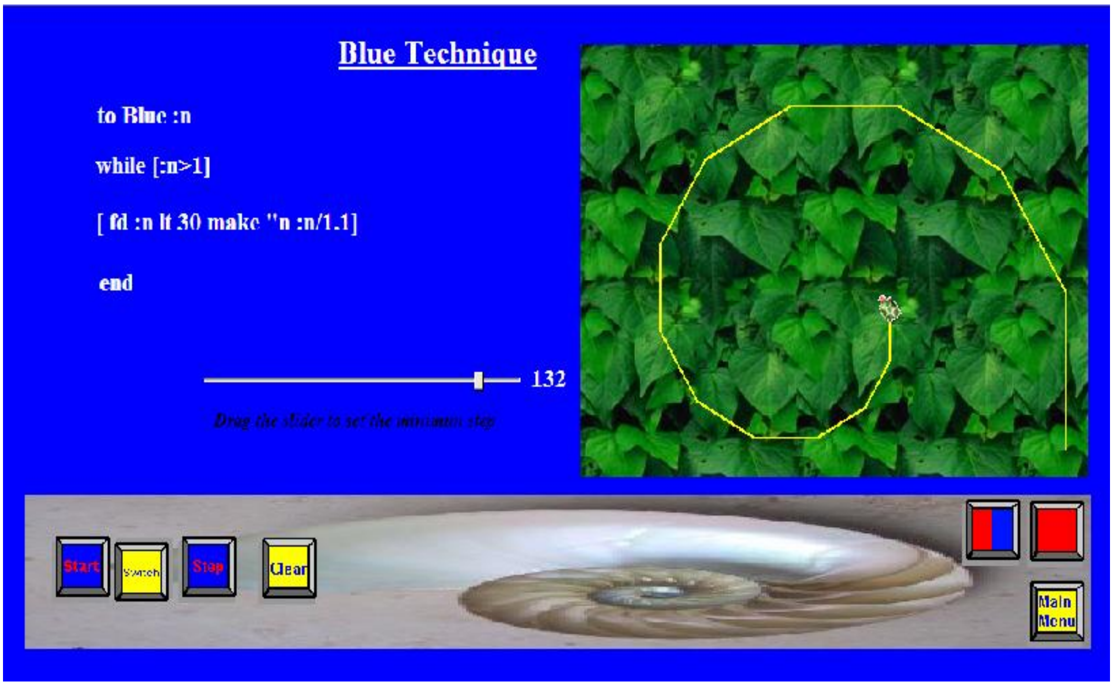


Figure 25-The main interface of *Spirals* (AVDA approach)

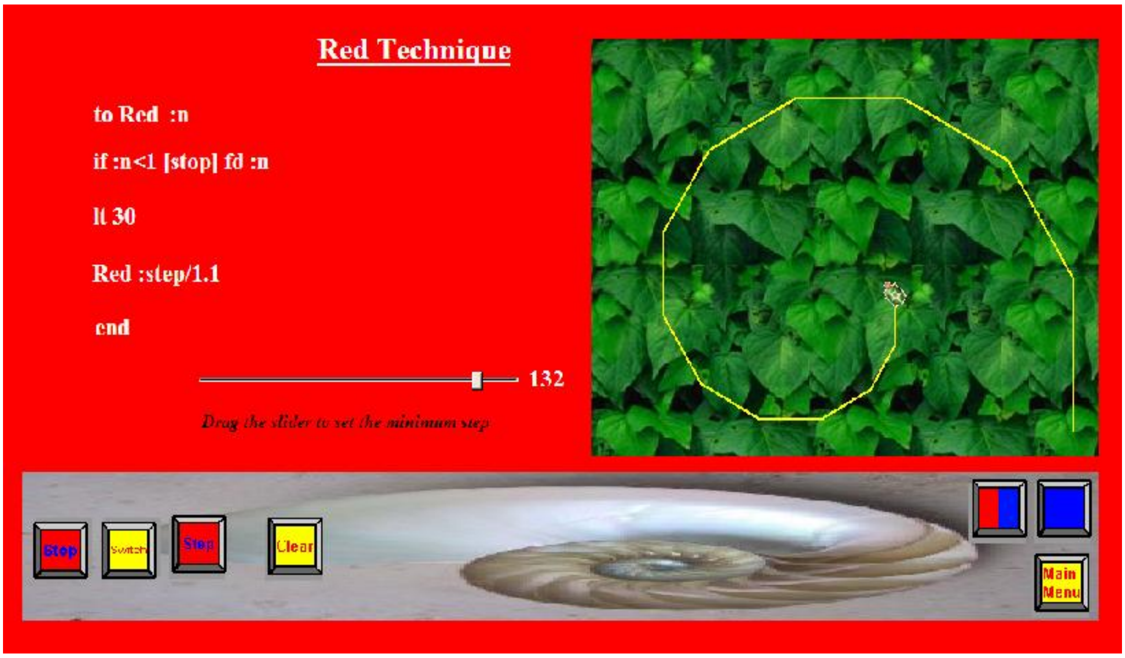


```
to Blue :n
while [:n>1]
[ fd :n lt 30 make "n :n/1.1]
end
```

132

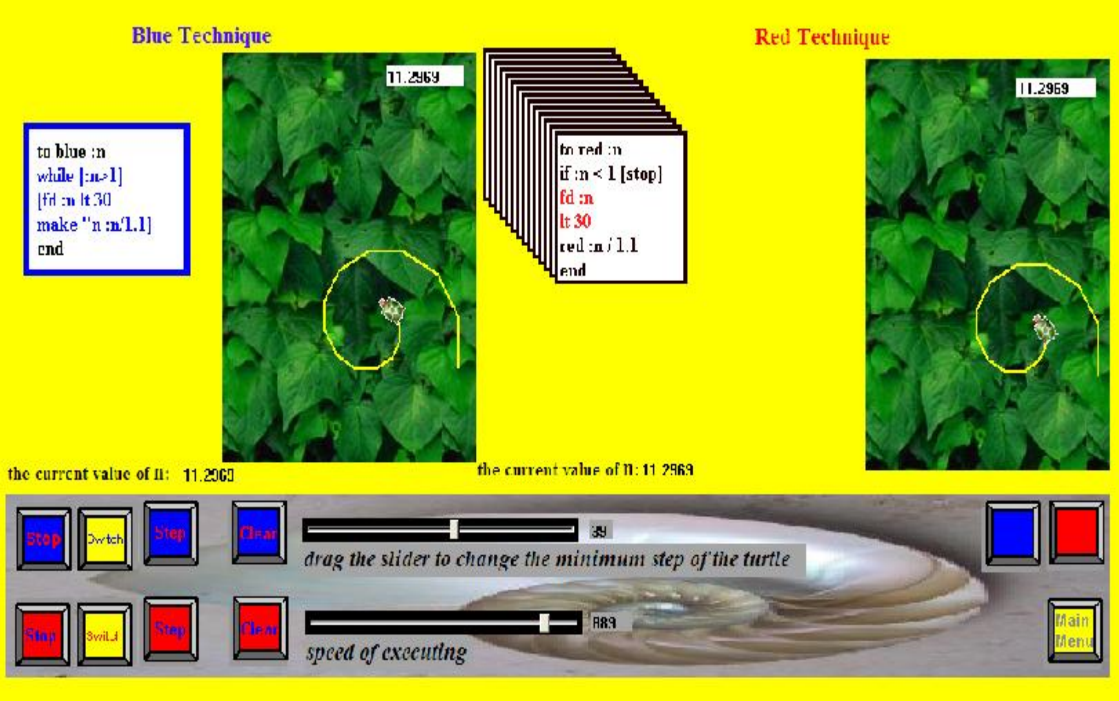
Drag the slider to set the minimum step

Figure 26- The blue technique (iterative procedure)



```
to Red :n
if :n<1 [stop] fd :n
lt 30
Red :step/1.1
end
```

Figure 27-The red technique (recursive procedure)


Blue Technique
Red Technique
11.2969
11.2969
the current value of n: 11.2969
the current value of n: 11.2969
Stop
Switch
Step
Clear
39
drag the slider to change the minimum step of the turtle
Stop
Switch
Step
Clear
889
speed of executing
Main Menu

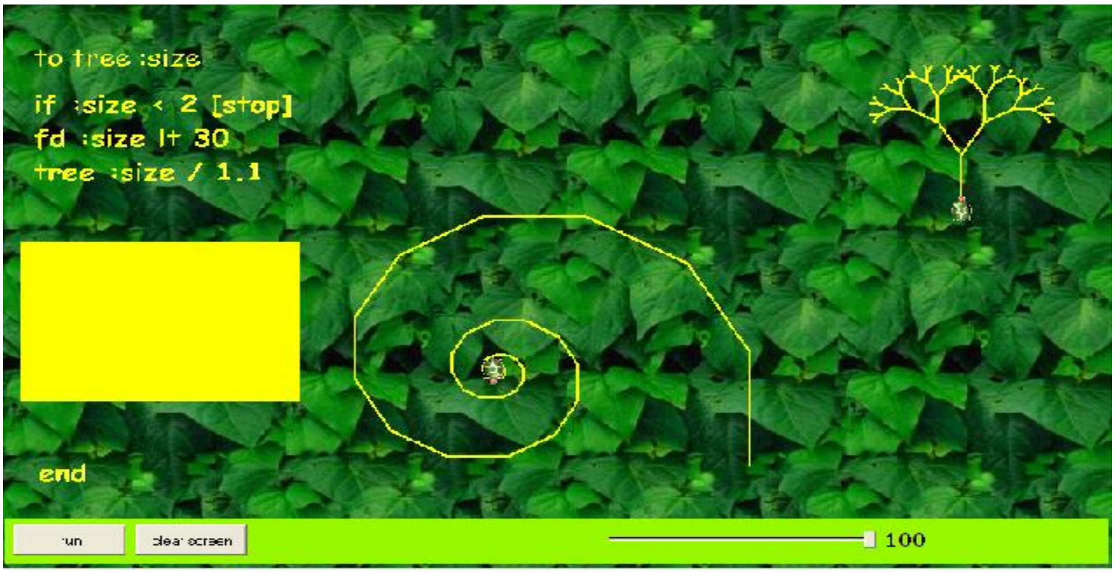
```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

Figure 28-The comparison module (AVDA approach)

As mentioned above, one of the major concerns in the innovation of AVDA was to focus on the functioning aspect of the concept of recursion. To do so, I also designed another domain called the *blank-box* task. In this domain, the students were able to program and engage with the computer a bit more interactively than with *Spirals*. It was conjectured that the *blank-box* module may provide more playfulness to the students. In this module, the students were given an incomplete recursive procedure to generate a binary tree.

The *blank-box* module was a relatively demanding task in the sense that the students were asked to fill a given empty box to complete the recursive procedure to produce a binary tree. Performing this task requires a clear mental model and perception of the recursive procedure and its essential components like the functionality and functioning of the recursive call(s). So, it was conjectured that the task would provide me with an appropriate window into the students' thinking and beliefs about the recursive procedures.



```
to tree :size

if :size < 2 [stop]
fd :size lt 30
tree :size / 1.1



end
```

Figure 29- The *blank-box* module

The other issue was that the task was designed to probe whether students were able to make any sort of connection between the tail and recursive procedures. In other words, the task was designed in such a way that a few given commands in the incomplete recursive procedure were a tail recursive procedure to draw a spiral. It was also conjectured that the students might be able to make some kind of connection to make two spirals in opposite directions to generate a tree. The main interface of the *blank-box* task is shown in the above picture (Figure 29).

My approach to implementing the second iteration, tool design, and tool use aspects are discussed in the next sections of this chapter.

5.4.2. My Approach – *Spirals*

I worked with seven volunteer students. They were mathematics specialists who were studying on a four-year degree program, and were training to be primary school teachers. The students attended the interviews and participated in the tasks in three pairs and one individual. Each interview session lasted 1.5 hours. The tasks in this iteration were implemented in a particular order which is shown in the following table.

Task	Order of implementation	Purpose
Pre-questionnaire	First task	Finding out the students' beliefs through a few open ended questions.
blank-box	Second task	Evaluating the students' thinking and thinking-in-change through the window of the tool.
Spirals	Third task	Evaluating the students' thinking and thinking-in-change through the window of AVDV environment.
blank-box	Fourth task	Evaluating the effect of working in the AVDA environment on the students' thinking and thinking-in-change through the window of the tool.

Table 6- The order of the tasks in the second iteration

In the first place, students were asked to answer the pre-questionnaire task, in which I was able to see what they believe and think about the natural spiral structures and binary trees. Then, it was the turn of the *blank-box* module. I asked the students to work with this module twice. The first time was when they finished work with the pre-questionnaire and the second time was when they finished their experiment with the *Spirals* computer-based domain (AVDA approach).

This order of implementing the *blank-box* module provided me with multiple windows into the students' thinking and thinking-in-change process before and after their experiments within the AVDA environment. Looking into those windows enabled me to investigate how the students framed, shaped and developed their thinking and mental models of the concept of tail recursion. I was also able to track their thinking-in-change process through the windows provided by the *blank-box* module and AVDA.

The students' responses to the pre-questionnaire task and also their experiments with the *blank-box* task and AVDA environment were recorded both using an audio tape-recorder and a Camtasia screen recorder. Using a Camtasia screen recorder allowed me to record every single movement of the students on the screen while they were working within the AVDA environment.

The advantage of using such recording software was that, due to the nature of the tasks, I had to take many research notes while they were working with the software. Writing the notes, was very difficult and complicated because the students said nothing and were just moving the cursor on the screen to point to certain commands or typing something on the screen. The reason for the complexity of the note-taking in these sorts of situations is that, as a participant observer, I had to observe and scrutinise every movement. Switching my attention to write about those non-verbal situations could increase the risk of losing some valuable observation of the students' explanations or movements.

Using Camtasia the screen recorder software helped me to record those non-verbal situations. Similarly to the first iteration, all the interviews were transcribed and saved in two stages of plain and interpretive accounts. The interpretive accounts were treated as data and coded in two stages of descriptive codes and analytic codes[1] to be analysed. The analysed data provided me with some new conjectures and issues which were reported to the final iteration of this research. The pre-questionnaire task in the second iteration is discussed in the next session of this chapter. It is followed by the tool design and tool use sections.

5.4.3. Pen & paper task – Iteration Two

Similar to the previous iteration, I asked the students to start the interview by answering the pre-questionnaire task. The task contained two open ended questions. The first question was about modelling the Joshua tree (Figure 30).

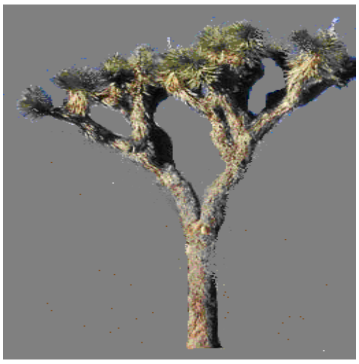


Figure 30- Joshua tree in the pre-questionnaire task

The reason for choosing a Joshua tree was the branching structure of these trees. Joshua trees' structure is almost similar to binary and ternary trees. The second question was about the spiral structures in nature. Students were given two natural spiral patterns – a snail shell and a flower plant – and a paper made spiral (Figure 31).

[1] Please see appendix B to see the table of Codes.

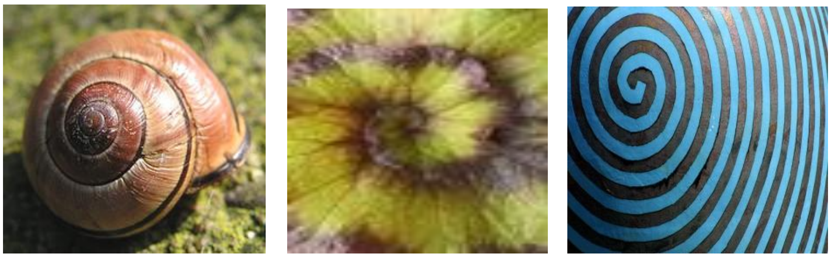


Figure 31-The spiral shape patterns in the pre-questionnaire task

Similar to the pre-questionnaire task in the first iteration, the students were asked to describe the images and explain how they would write an algorithm to produce these objects. The students' responses to these questions are discussed in the tool use section later in this chapter. The next section of this chapter concentrates on the key design features of the tool design in the second iteration.

5.4.4. Tool Design – The *Blank-Box* and the AVDA environment in *Spirals*

From the design perspective, the second iteration had a substantial difference from the first iteration. It has been mentioned above that from the functional abstraction point of view the computer-based tool for the first iteration had some limitations. The *Treemenders* was designed mainly from a functionality (what needs to be done) point of view rather than functioning (how it will be done) point of view. Therefore, as shown in the students' accounts, for instance Sarah and Jin lines 104-108 and lines 110-118, the need for focusing on the *how* part was a major issue for the next iterations.

One of the main challenges for me was to find a strategy for designing the domain of abstraction from both functionality and functioning aspects. To do so, I designed two computer-based tools for the second iteration. The first one, as mentioned above, is the *blank-box* module and the second is the AVDA domain of abstraction. Hence, this section is divided into two major parts. The first part focuses on the design features of the *blank-box* task. It is followed by the key design features of the AVDA environment, which includes three modules of the *red* and *blue* techniques and the *comparison* module. Table 7 outlines the key design features in the tasks which were designed for the second iteration.

Computer-based domain		**Design features**	**Purpose**	
The *blank-box* module		An incomplete recursive procedure with a given blank-box to be filled by the students.	**For the students**	Creating a window to investigate both the functioning and functionality aspects of recursion
			For me as a Researcher	Figuring out the students' thinking and thinking-in-change about recursion and its components.
AVDA environment	***Blue* technique**	Designing two modes for executing an iterative procedure to produce a spiral – normal mode and stepwise mode	**For the Students**	To provide students with the opportunity for an in-depth investigation of the control passing in an iterative procedure by reflecting the steps and the correspondence output in one window
			For me as a Researcher	Figuring out the students' thinking and thinking-in-change about iterative procedures.
	***Red* technique**	Designing two modes for executing a tail recursive procedure to produce a spiral – normal mode and stepwise mode	**For the students**	To provide students with the opportunity for in-depth investigation of the control passing in a tail recursive procedure by reflecting the steps and the correspondence output in one window
			For me as a researcher	Figuring out the students' thinking and thinking-in-change about tail recursion and the influence of their previous experiments with iteration.
	Comparison module	Designing a representation of a viable copies model for the tail recursion and comparing it with the iteration, also using colour-codes for each command of the procedures.	**For the Students**	To provide students with the opportunity for in-depth investigation of the control passing in a tail recursive and iterative procedures and correspondence output in the AVDA approach in one window.
			For me as a Researcher	Figuring out the students' thinking and thinking-in-change about tail recursion and the influence of the AVDA approach on their thinking style.

Table 7-The outline of the design-purpose features of the task in the second iteration

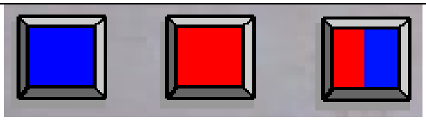
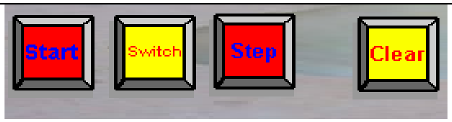
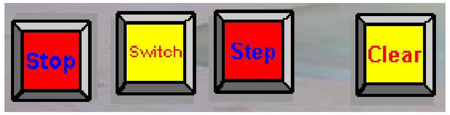
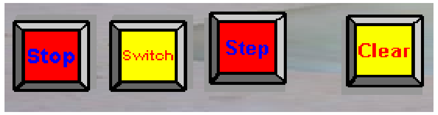
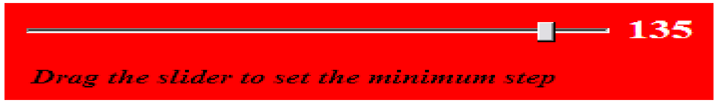
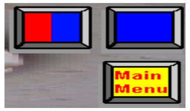
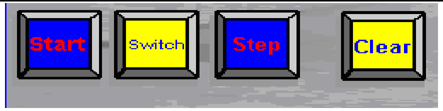
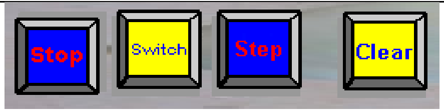
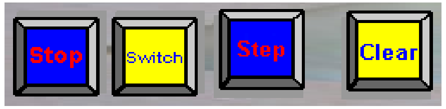
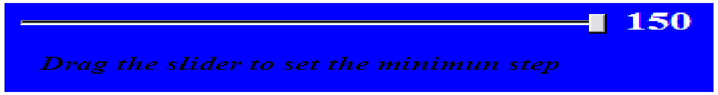
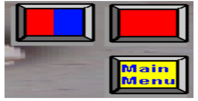
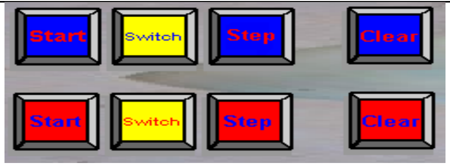
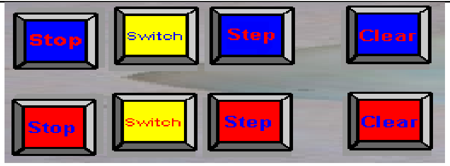
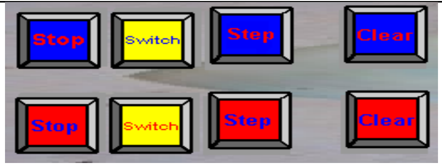
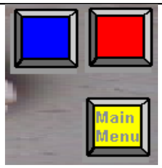
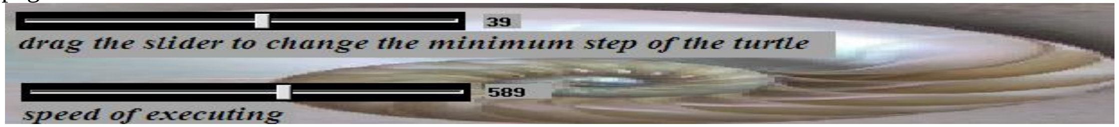
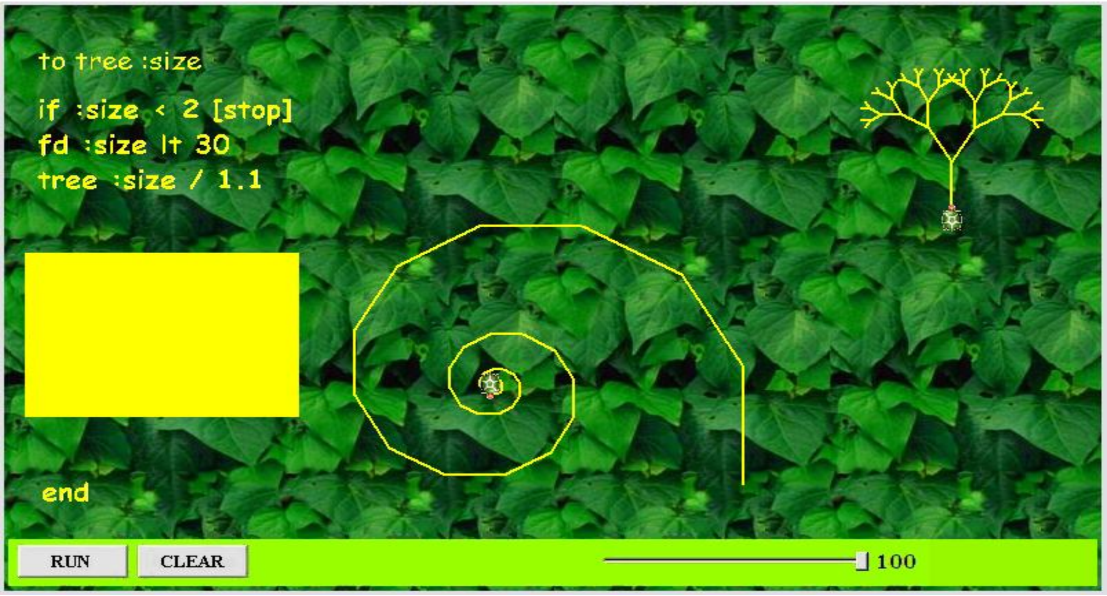
Module	Design features
Main interface	 1) *blue* technique, *red* technique, and *comparison* module
The *red* technique	 1)The control box of the *red* technique  2)After activating the *start* button  3)After activating both the *start* and *switch* buttons  4)The slider for setting the initial step  5)Move to other pages
The *blue* technique	 1)The control box of the *blue* technique  2)After activating the *start* button  3)After activating both the *start* and *switch* buttons  4)The slider for setting the initial step  5)Move to other pages
The *Comparison* module (AVDA)	 Start Switch Step Clear Start Switch Step Clear 1)The control box of the *comparison* task  Stop Switch Step Clear Stop Switch Step Clear 2)After activating the *start* buttons  Stop Switch Step Clear Stop Switch Step Clear 3)After activating both the *start* and *switch*  4)Move to other pages  39 drag the slider to change the minimum step of the turtle 589 speed of executing 5)The sliders for the initial size and speed the control of execution of the procedure

Table 8-The design features of the three modules of the *Spirals* domain of abstraction

The *blank-box* module

This module was designed based on two design techniques; Using-before-knowing, by modelling binary and ternary trees. In the *blank-box* task, the concepts of tail and embedded recursion were phenomenalized by using spirals and trees. The interesting design point in this module was the link between the tail and embedded recursion (see program 6). The link was embodied by giving the students an incomplete embedded recursive procedure such that the output of the given codes was a spiral – a tail recursive part – and they were asked to complete the procedure to make a tree.

This module was designed to evaluate the students' ability to use and apply recursive procedures in the problem-solving situation. The *blank-box* module also assessed their recognition and understanding of the functioning aspects of the main components of a recursive procedure – recursive call(s) and base case. The main interface of the *blank-box* module is shown in the below picture.



```
to tree :size

if :size < 2 [stop]
fd :size lt 30
tree :size / 1.1



end
```

Figure 32-The interface of the *blank-box* module

As shown in picture 33, the key feature of this module was the incomplete recursive procedure on the left side of the picture. The procedure is demonstrated below.

```
To tree :size
If :size < 2  [ STOP ]
Forward :size
Left turn 30
Tree :size / 1.1
 ________________________
| The blank box to be    |
| filled with appropriate|
| commands by the        |
| students               |
|________________________|

End
```

Program 6-The incomplete recursive procedure and the *blank-box*

The students were given a control box – as shown in picture 33 – including two buttons and one slider (Figure 33). They could run their program by pressing the run RUN button and clear the screen by pressing the clear CLEAR button to amend their approach. The students also were able to set the initial size of the main trunk of the tree by the slider which was contrived in the control box (Figure 33).

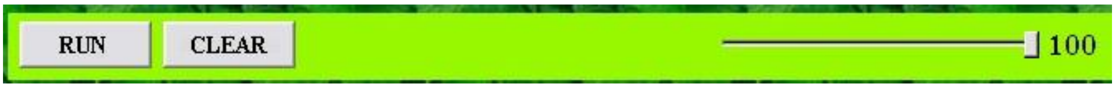


Figure 33-Control box in the *blank-box* module

In this module, the students were able to enter their own commands into the given box in an incomplete recursive procedure to generate a binary tree. They were expected to use and work with the recursive procedure before knowing about it. The first part of the procedure, which was given to them (shown using the red colour in Program 2 above) as shown in the above picture, produces a spiral in the left direction. It was conjectured that the way that the students try to complete the task would provide me with a rich window towards their understanding and thinking about the recursion and its crucial components.

One of the key design features of this module was that the students were able to enter their own commands, which had to include one additional recursive call, into the given blank box and by seeing the output on the screen reflect on their work and possibly make amendments and modifications.

This module seemed to be an easy task. However, it was designed in such a sophisticated way that the students were able to complete the task only if they had a visible model of recursion and a clear understanding of the functioning of the recursive call as a vital component of it as well as the functionality of the components of a recursive procedure. It was conjectured that they would realize that the procedure needs another recursive call only if they have a clear understanding of the functioning of recursive call, to generate the branches in the right direction. It was also conjectured that if the students put the recursive call in the right place, this would show that they have a clear understanding about the functioning of the recursive call and the mechanism of control passing in the recursive procedures. That was the reason which made this module a challenging module for the students. It provided a rich window for me to probe students' thinking about the concept of recursion. Students' results with this module provided me some promising results, which are discussed later on in this chapter in the tool use section.

The *Spirals* computer-based domain and AVDA approach

Figure 34-Main page of the *Spirals* domain

As earlier mentioned, the AVDA (Animative Visualisation in the Domain of Abstraction) innovation was designed within the *Spirals* computer-based tool, based on the results of the first iteration and my initial interest in using fractal and fractal-shaped objects to embody the concept of recursion. This domain of abstraction and AVDA approach were carefully designed to provide me as a researcher with a window into the students' thinking about the tail recursive procedures and also a window for the students to form and develop their thinking about the latent layers of the tail recursive procedures.

It was conjectured that the animation technique employed in this domain provides the students the situation in which they could see and experience the latent layers of the concept of recursion and its control passing mechanism. Therefore, AVDA provides students some on screen objects to work with and investigate on the main components of tail recursion such as functioning of the recursive call in the *comparison* module and animation and colour-codes. It also helped students to see the control passing mechanism of the tail recursive procedures.

Tail recursion was phenomenalized by using informal fractal-shaped objects and animative visualisation throughout three modules in the *Spirals* domain of abstraction which are discussed later on in this chapter.

Spirals domain contains three modules and a main page (Figure 34). The students have three choices in the main menu of the tool: the *blue* technique, the *red* technique, and the *comparison* module. Table 4 outlines the main design features in these three modules. The students were able to move to each one of these three modules from the main menu by pressing on each one of the three buttons, which were contrived in the main page – see the first row of Table 4. To go to the *blue* technique, they needed to press the blue 🟦 button, to go to the *red* technique, they needed to press the red 🟥 button, and by pressing the red-blue 🟥🟦 button they were able to move onto the *comparison* module. The students were asked to implement the modules in a hierarchical way as shown in the below diagram.

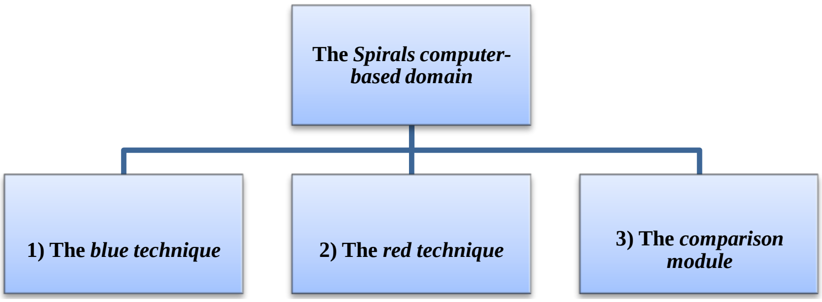
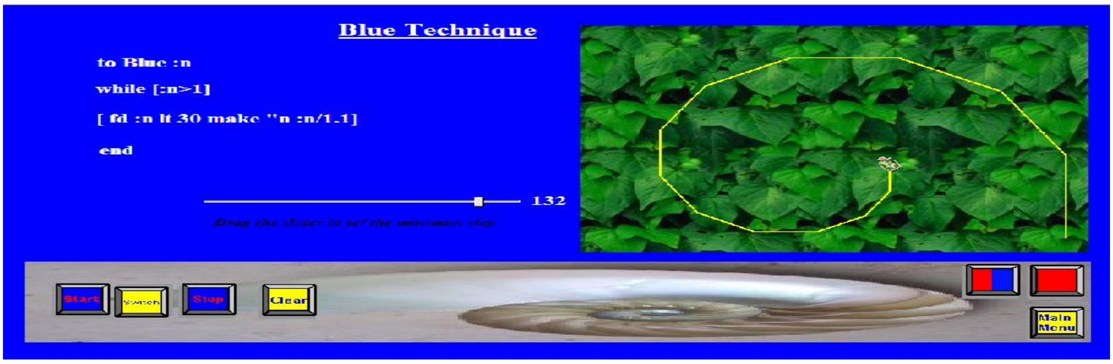

The *Spirals* computer-based domain
1) The *blue technique*
2) The *red technique*
3) The *comparison module*

Figure 35- The order of execution of the modules in the *Spirals* domain by the students

As shown above, the first task that the students were asked to work with was the *blue* technique. Then the students were asked to work with the *red* technique, and finally with the AVDA innovation in the *comparison* module. The rationale for designing these three tasks was to see how the students think about the iteration and tail recursive procedures. Moreover, to see to what extent they change and develop their thinking about these concepts within the AVDA approach in the *comparison* module. In the *comparison* module, I created an animative innovation towards functioning aspect of the concept of tail recursion in the *Spirals* domain.

Despite the *blue* and *red* techniques, which were mainly designed to see how students think about the iterative and recursive (tail recursive in particular in the second iteration) procedures, the *comparison* module was mainly designed to phenomenalize and embody the viable copies model of the tail recursive procedure by employing animation and colour-coding. The underlying principle was to investigate how the students' thinking and mental models about tail recursion is evolved and changed by working in that domain.

The *blue* technique (iterative)



```
to Blue :n
while [:n>1]
[ fd :n lt 30 make "n :n/1.1]
end
```

Figure 36-The *blue* technique (iterative)

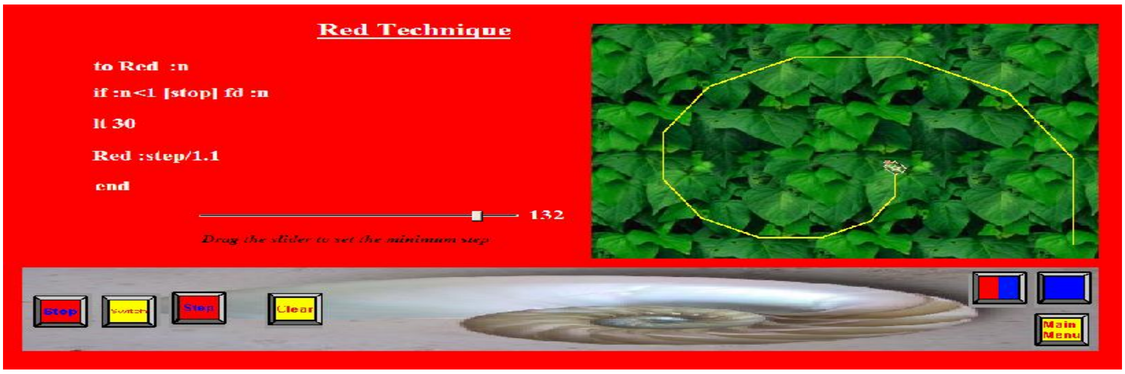
The *blue* module was the first module with which the students were asked to work (Figure 36). The *blue* technique was based on an iterative procedure to generate a spiral. They students have two choices of executing the procedure in normal mode or step-wise mode. They could switch between these modes by pressing the appropriate buttons in the control box – see the third row of Table 8. In the step-wise mode of execution, they needed to press the *switch* button Switch first, and then press the *start* button Start, and finally keep pressing the *step* button Step to generate the spiral step by step. They were also given a slider to set the initial size of the first step, shown in Table 8.

A clear button Clear was also contrived in the control box to allow students to clear the screen to test and check a new spiral. In the third row of Table 8, you can see another set of control buttons, number (5), which allow the students to move onto the other pages by pressing the appropriate buttons. The iterative procedure that was written to generate the spiral iteratively is shown below.

```
To Blue :n

While [:n > 1]
[Forward :n Left turn 30 Make “n :n / 1.1]

End
```

Program 7-The iterative procedure to generate a spiral in the *blue* technique

The *red* technique (recursive)



```
to Red  :n
if :n<1 [stop] fd :n
lt 30
Red :step/1.1
end
```

Figure 37-The *red* technique (recursive)

The *red* technique is similar to the *blue* technique. The major difference between these two is the programming technique. The program in the *red* technique was a tail recursive to generate a spiral which is shown below. The program has one recursive call – *Red :n / 1.1* – and a base case – *If :n < 1 [Stop]* – which operates as the stopping condition.

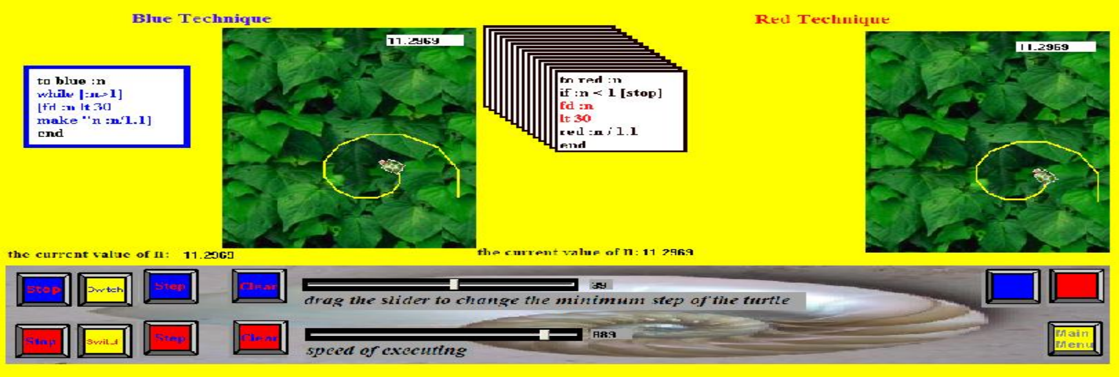
```
To Red :n
If :n < 1 [stop]  Forward :n
Left turn 30

Red :n / 1.1

End
```

Program 8-The tail recursive program to generate a spiral in the *red* technique

In picture 37 it can be seen that the students were able to reflect on their work by observing the output of the procedure on the screen.

Similar to the *blue* technique, which is explained above, the students have two choices of running the procedure in the normal or step-wise mode by pressing the appropriate buttons and a slider to set the size of the initial step – shown in the second row of the Table 8.

The *comparison* module and AVDA approach



```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

the current value of n: 11.2969

the current value of n: 11.2969

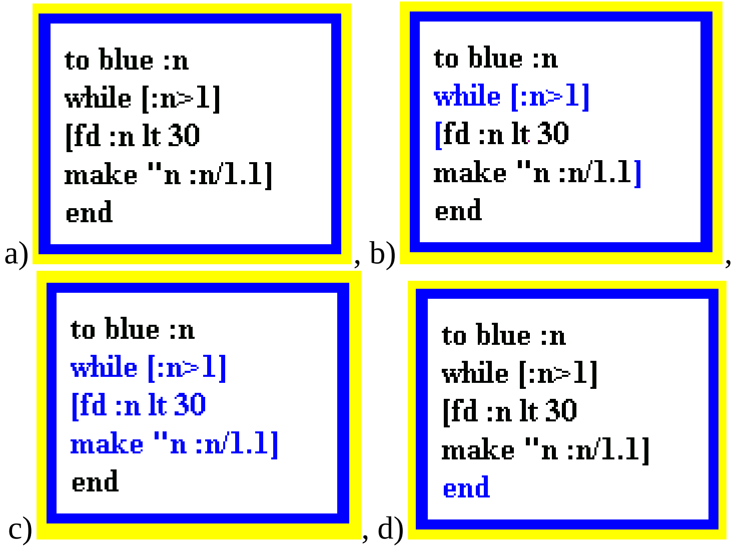
drag the slider to change the minimum step of the turtle

speed of executing

Figure 38- The *comparison* page and AVDA innovation in the *Spirals* domain

The AVDA innovative approach was employed in this module. As shown in Figure 38, the students were provided with the opportunity to compare the iterative technique with an embodiment of the viable copies model of the tail recursion.

The embodiment was created by using animation and colour-coding. The control box and the buttons that the students were provided to work with are shown in the fourth row of Table 8. The students were also given one more slider to control the speed of the execution, the image numbered (4) in the fourth row of Table 8. It was conjectured that by slowing down the speed of the execution, the students would have more opportunity to see the mechanism of control passing when the tail recursive procedure was being executed. The colour-code was employed in such a way that each command which was being executed was flashing or changing into the other colour. For instance, when the *blue* technique was being executed in the AVDA approach, the colour of each line of the procedure being executed transferred into the blue colour (Figure 39 a, b, c and d).



a)

```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

, b)

```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

,

c)

```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

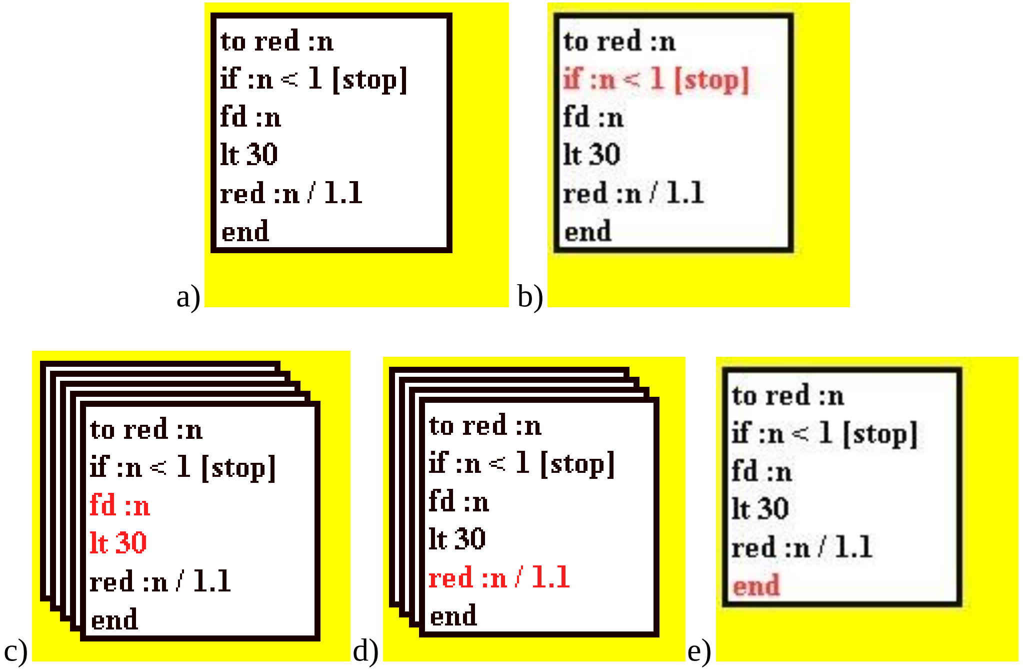
, d)

```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

Figure 39-The colour-codes in the iterative technique in the AVDA environment

Figure 39 (a) shows the iterative procedure before starting the execution. Immediately, after starting to run the procedure in Figure 39 (b) the first command of the procedure changed to blue colour when it was being executed. Then when the control passed to the next part of the procedure, Figure 39 (c) shows that the colour of the commands which were being executed changed to blue, and when the procedure finished the execution colour of the last command of the procedure changed to blue, see Figure 39 (d).

A similar process was employed in the *red* technique in the AVDA environment. The only difference was that besides the colour-coding on the lines of the procedure, the students had the opportunity of seeing the copies of the original procedure. The animation technique was designed in such a way that, after each time calling, the recursive call for a new copy, a new copy of the procedure was appeared on the screen. Simultaneously, in each of the generated instantiations I employed the colour codes, it can be seen in Figure 40 a, b, c, d, and e. Picture 40 (a) shows the procedure before execution, and trivially there is no generated copy (instantiation). The pictures numbered 40 (b)-(e) show both generating copies and also the colour codes of those commands which were being executed at the time. It was conjectured that seeing the viable copies model for the tail recursion would allow the students to develop their thinking and mental models of the concept.



```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

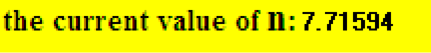
```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

Figure 40-Animation and the colour codes in the tail recursive procedure in the AVDA environment

Technically, apart from the animation and colour codes, I contrived a few more facilities into the *comparison* module to facilitate the process of tracking the mechanism of the flow of control in the tail recursive procedure by the students. For instance, in Figure 38, the interface of the *comparison* module, you can see the box in both *blue* and *red* techniques which shows the current value of the length of the segment  the current value of n: 7.71594 which was being drawn by the turtle. The value shown in this box is also shown on the screen in which the spiral was being drawn (Figure 41).

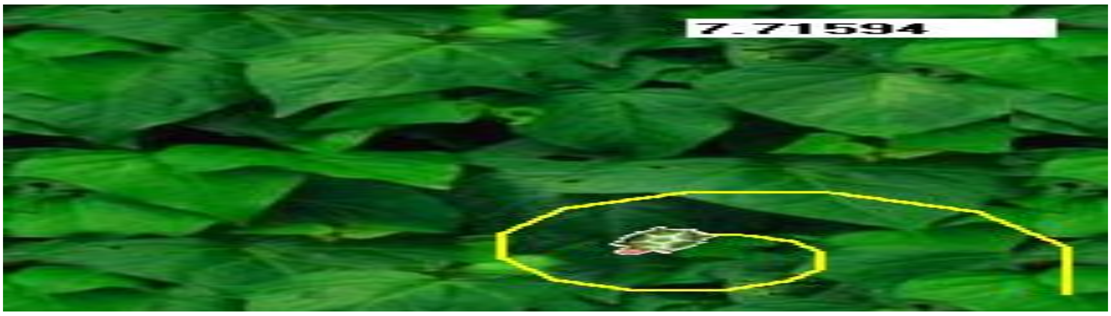


Figure 41-The box for showing the length of the last segment which was being drawn by the turtle

Similar to the *red* and *blue* modules of the *Spirals* domain, in the *Comparison* module, the students were also allowed to run the procedures in the two modes of normal and step-wise. It was conjectured that by step-wise execution of the procedures the students would be able to follow and track the flow by observing the syntax and output simultaneously. The link between the syntax and output is embodied and phenomenalized by using the colour codes of the commands and animation plus the output which was a spiral being drawn by the turtle. Therefore, it was conjectured that such a design would assist the students to bridge the functionality – *what needs to be done* – and the functioning – *how it will be done* (Sooriamurthi, 2002).

<u>Summary of tool design – the Spirals</u>

To sum up, based on the emergent issues and the conjecture from the first iteration, I designed the domain of abstraction for the second iteration. Based on the aims of this research, the design was mainly intended to probe the students' thinking about the tail recursion throughout the following issues:

- Emphasizing on functioning aspect of the design,
- Giving more control to the students (playful-ness)

The first part was contrived into the domain of abstraction by using animation and colour-coding in the AVDA environment. And the second was partly done by giving the students the opportunity of running the procedures in the step-wise mode and also adding their own commands in the *blank-box* module and see the output on the screen. Therefore, compare with the first iteration's domain, the *Treemenders*, the domain for the second iteration was more interactive and playful for the students and also has represented the latent layers of the mechanism of control passing in the tail recursive procedures.

The next section of this chapter discusses the students' use of the tool in the second iteration.

5.4.5. Tool Use – *Spirals*

This section concentrates on the usage dimension of the *blank-box* module and the *Spirals*. It is divided into two parts. The first part discussed on the students' accounts on the pen & paper task (the pre-questionnaire) and the second part is about the students' accounts on the *blank-box* module and the *Spirals* domain of abstraction.

Tool use, pen & paper task (the Pre-questionnaire) – *Iteration two*:

I worked with three pairs and one individual. Each interview lasted 1.5 hours. I start the tool use section with a brief explanation of the students' responses to the pre-questionnaire (the pen and paper task). As mentioned earlier, the pre-questionnaire task had two major parts. The first part was an open-ended question about the Joshua tree (Figure 42 (a)) and the second was about some natural spiral-shape objects (Figure 42 (b)-(d)).

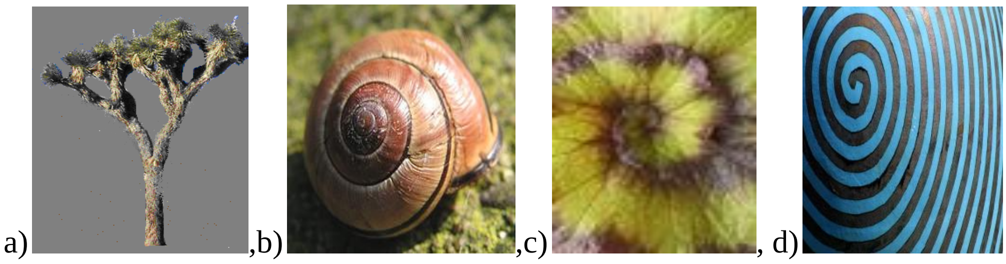


Figure 42-The images of the pre-questionnaire task

The students were first asked to describe the images, and then required to describe what they could say about their shape, structure, size, and angle. Finally, they needed to describe the essential features of those images if they wanted to draw them. In the following part of this section, I am going to present and discuss two pairs (Tabby and Akilla, and Andrew and Hayley) of the participants in detail. In this iteration, I also interviewed an individual student, Kieran, and I compared his explanations with those two pairs. I pointed to the image of the Joshua tree (Figure 42 (a)) and started the task.

1. I asked: can you describe this image?
2. Kieran: It looks like a bifurcation.
3. I asked: What do you mean by that?
4. Kieran: It is a fractal and each time it stems it is doubled.
5. I asked: What can you say about the size and angle?
6. Kieran: As I said, it is a fractal and the branching angle is 45. It seems that the length of the new branches is almost half that of the previous ones.

7. I asked: What are the essential features of the image if you want to draw it?
8. Kieran: The branching points.

Lines 2-6 show that Kieran is familiar with fractal-shaped objects. He also mentioned that if he wanted to draw a Joshua tree, the essential part is *branching* (line 8). And we know that there are two main issues with regards to the branching in terms of the components of a recursive procedure. The first one is recursive call(s), which means to create the new branches you need to consider a recursive call in your procedure, and also the complicated mechanism of control passing in the procedure. The latter allows the students to locate the recursive calls in the right place in the procedure. Regarding the spiral pictures (Figure 42 (b)-(d)), Kieran responded as follows:

9. Kieran: They are like fossils and sea and plant life objects.
10. I asked: How about the structure of them?
11. Kieran: They look like fractal structures. Regular spirals.
12. I asked: What do you mean by the regular spirals?
13. Kieran: It seems that the distances between the layers of the spirals are similar.
14. I asked: Can you describe the essential features of them?
15. Kieran: They are circular structures, the spacing between the lines is an important factor, and the angle seems to be increasing.

Kieran’s initial explanations in the pre-questionnaire task show that he recognised the fractal structure of the Joshua tree (lines 2-4). Moreover, he thought that the spirals are also fractals (line 11), as we know they are not because they can also be defined iteratively. Structurally, he described the Joshua tree as a bifurcation which always produces two branches (lines 2-6). With regard to the spirals, he was thinking about the space between the circular lines in a spiral (lines 11-15). He also considered them to be circular objects.

One interesting point was that Kieran’s first impression of the image of Joshua tree was that ‘*it looks like a bifurcation*’ (line 2) and the first impression of the image of the spirals was that ‘*they are like fossils and sea and plant life objects*’ (line 9).

Tabby and Akilla, another pair of students who took part in this iteration, responded to the pre-questionnaire module by focusing more on some quantitative structural parameters like angles and length. They described the spirals in Figure 42-((b) and (c)) as follows:

16. Tabby: The angle is becoming bigger and bigger and bigger. You know what I mean?
17. Akilla: Yeah I know, I can see it
18. Tabby: but in this one [*Figure 42 (d)*] the angle is the same.
19. Akilla interjected: Yeah it is!
20. Tabby: What do you mean by the structure?
21. I answered: What do you think?
22. Tabby: I see a pattern going all the way around. I was just thinking I can draw a circle with ‘n’ but this side slightly goes out and then carries on the same.
23. I asked: What are the essential features of the images if you want to draw them?
24. Tabby: I am just thinking of it in terms of a circle. You know to draw a circle, repeat 360 forward 1, so, maybe…
25. Akilla interjected: We might consider going right back to the beginning, it goes back.
26. Tabby added that: It goes back, but it needs to carry on to do more repeating and avoid going over itself, we need to increase the angle as well,
27. Akilla agreed by saying: Yeah!
28. Tabby continued: I don’t know if that is right or not, but that might make something like that [*Figure 42 (d)*]
29. I asked: What is the difference between a circle and a spiral?
30. Tabby: When you do a circle, you *repeat 360, forward 1,*
31. Akilla: For a spiral when you get back to where you started, you need to slightly somehow increase the angle or something to go up there and then carry on
32. I asked: At that point are you going to change the angle or do you want to change the size?

33. Tabby: I would have to change the angle in the first place, so it does not draw over the circle. The right one [*Figure 42- (b)*] would be increasing in angle each time.
34. I asked: Do you think about the size?
35. Tabby: Yes, for example, if you had *forward* 5, it would mean just drawing more like this kind of shape [*Figure 42-(c)*], you know what I mean? It's not very good, you have got more straight lines in your spiral.
36. Akilla interjected: Yeah,
37. Tabby: Because forward 1 is like the smallest amount of step you can do, it looks like a circle but you still actually draw little, little lines and put them together, the angle would be affected, but how do we repeat iteration?
38. Akilla: I don't know!

Lines 24-33 showed that Tabby and Akilla were thinking in terms of iteration about the structure of the spirals. However, their utterances about the Joshua tree (Figure 42-(a)) were slightly related to recursion (line 44).

39. Akilla: It is like an up-growing tree. The branches are getting smaller as you go higher. Do the angles become smaller?
40. Tabby: Not really, the angles are the same thing. They are just being repeated on top of it on different branches.
41. Akilla: So, the angles are the same?
42. Tabby: Yeah, they are just repeating the same thing.
43. I asked: What do you mean by the same thing?
44. Tabby: It is like the same thing. I was saying the same **"Y"** shape is repeating itself somewhere else. And the angle for the original **"Y"** is the same, I think.

In line 44, Tabby to some extent pointed to the concept of recursion semantically. However, it was too early to take this explanation as evidence to show that her thinking about the structure of a Joshua tree was recursive.

45. Akilla: Yeah, the same **"Y"** but they are just going up on each other.
46. Tabby: They go up on each other and again get smaller and smaller.
47. I asked: Is it also the same when it becomes smaller?
48. Tabby: The same shape, the same angle, just a smaller distance.
49. I asked: What are the essential parts of the shape if you wanted to draw it?

50. Akilla: The structure of it, the initial **“Y”**. And then **“Y”** over **“Y”**s becoming bigger like a big “**Y**”
51. Tabby: But it doesn’t look like a big **“Y”** – it’s just the initial **“Y”** and the other **“Y”**s on top of each other. We don’t care what the final shape would be. So, if you look at this image, are you going to look at the whole thing and then go in, or start at the smaller one and then go up, that’s the difference between the structure and the shape.

Tabby came up with the idea of “Y” shape for the Joshua tree (line 44), a shape that repeats itself. For Akilla, although she agreed with Tabby about the “Y” structure, she did think that the final product still looked like a big “Y” (line 50). Tabby did not agree with that and implicitly came up with the idea of the concept of the base case – to be a starting point or a stopping condition. In line 51, Tabby stated that ‘*you are going to look at the whole thing and go in or start at the smaller one and then go up*’. I used this idea in the design of the next iteration, which will be discussed later on in this thesis.

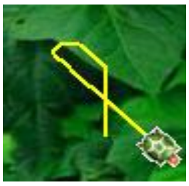
Tool use – the *Spirals*

Then the students were asked to work with the computer-based domains – the *blank-box* task and the *Spirals*. In these tasks, the main issues that I was looking at were the way that they think about the iterative and tail recursive procedures and the influence of them on each other. I also wanted to see how the students shape, form, and evolve their own mental models of tail recursive procedures in the AVDA environment. After finishing the pre-questionnaire, I asked Kieran to work with the *blank-box* module. After my brief introduction about the tools he started to work with it.

The following lines demonstrate the difficulties that Kieran showed regarding understanding the functionality of the recursive calls and the flow of control. In the line 58 he asked me about a command to send the control backward. His comment shows his incompetent knowledge about the functioning of the recursive calls and the control passing mechanism.

52. Kieran: It goes forward in size, the usual value and then left turn forty five
53. My description: [*he changed the angle from 30 to 45, paused a bit and said*][10]
54. Kieran: Can we go backwards?
55. I said: What do you think?
56. Kieran: I mean like, it has gone forwards once, turn 45, can I go backwards?
57. My description: [*paused and changed the ratio of the size of the new stems size / 2 instead of size / 1.1 and typed the following commands into the empty box*]

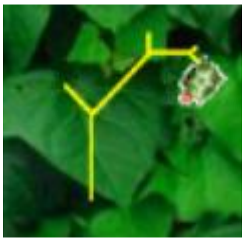
```
To tree :size
If :n < 2 [stop]
Forward :size
Left turn 45
Tree :size / 2

    Forward :size

End
```



58. Kieran: Are there any backwards commands to go back all the way around?
59. My description: [*then he removed the first recursive call and typed the following commands into the blank box*]

[10] The text inside the brackets is my descriptions of his non-verbal experiments.

```
To tree :size
If :n < 2 [stop]
Forward :size
Left turn 45

  Forward :size / 2
  Back :size / 2
  Right turn 90
  Forward :size / 2
  Tree :size / 2

End
```



The commands in the box (line 59) showed that Kieran wanted to have the command '*forward :size / 2*' repeatedly in the procedure to generate the new stems. This shows that he knew what he wanted, the new stems, but he did not have a clear understanding that it could be done by locating a recursive call. He wondered why he had no branches to the left (line 26). He had removed the recursive call, which was given outside the box, and instead he located another one in the box immediately after turning right 90 degrees. That was the reason for having all the new branches to the right and no new branches to the left. The *blank box* task had an important impact on Kieran's thinking-in-change process. Despite his difficulty with flow, in line 64 he described the procedure as a procedure which is calling itself.

60. Kieran: So that wasn't too bad. So, I've got the first two branches now.
61. My description: [*He could make the first branching point and he added that*]
62. Kieran: I was thinking it was only one stand at the beginning and each time after that I had to make two branches. And I was wondering why do this part just once!
63. My description: [*What he meant by "this part" was two commands [forward :size Left turn 45]. He was pointing to the only branch which was drawn into the left (see line 23). Then he decided to put a new recursive call into the first part of the procedure*].
64. Kieran: It is like calling itself from itself! So, oh that's good actually.

65. I asked: Why do you think so?
66. Kieran: Because we have another tree in here.

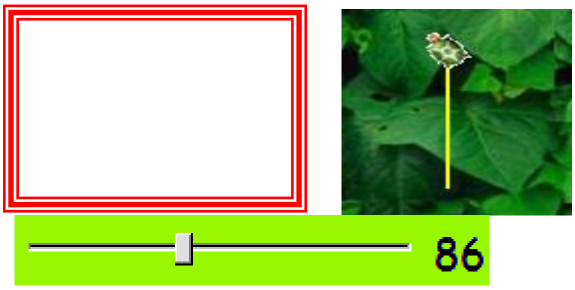
Lines 54-59 show that Kieran had difficulty in recognising and tracking the flow in the tail recursive procedure. There was also some evidence of his difficulty in recognising the functioning of the components of the recursive procedure. For instance, in line 59 he used the command '*forward :size*' three times, and he also deleted the given recursive call and put it in the box in the wrong place. His utterances in line 60 show that he knew what he was looking for. When he said "*I have got the first two branches now*" (line 60), it shows that he was looking for a binary tree – something similar to the Joshua tree in the pre-questionnaire.

However, by putting the commands in the wrong places in the procedure (see lines 57-59), Kieran shows his difficulties with both the functionality and functioning aspects of the concept of recursion. In line 64, he mentioned that the procedure is calling itself. This means that he recognized the syntax of the recursive call, but still had problems with its functioning and functionality.

Tabby and Akilla, the other pair of students who participated in the second iteration, started with working on the stopping condition in response to the *blank-box*.

67. Tabby: Let's start with a big number [*If :size <85*] to see what the difference is.
68. My description: [*They changed the stopping condition to 86 and the initial value of the size was already chosen to be 85*].

```
To tree :size
If :n < **85** [stop]
Forward :size
Left turn 30
Tree :size / 1.1
```

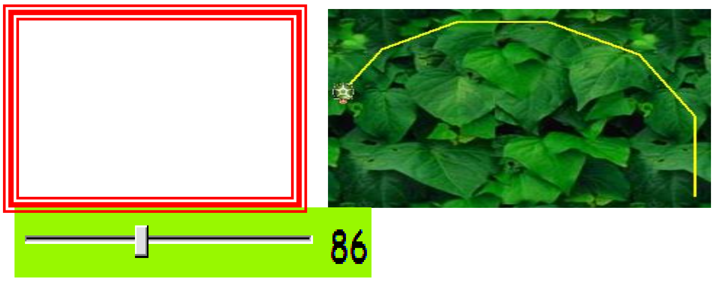


```
End
```

The following lines demonstrate that Tabby & Akilla also had difficulty in appreciation of the functioning aspect of the recursive calls and flow of control. In the lines 72-74 they were trying to send the control to the right, to have few branches on that side.

69. Akilla: Oh! It’s only a line, what happened?
70. Tabby: Umm, yes, because 86 is not less than 85, the size is not less than 85, right, then stop, if not it’s going to go forward whatever size we did, turn left 30 and then do it again, do the *tree size divided by 1.1*.
71. My description: [*Then she changed the stopping condition to 50 and got the below result*]

```
To tree :size
If :n < **50**[stop]
Forward :size
Left turn 30
Tree :size / 1.1
```



```
End
```

72. Tabby: I was thinking about a tree with the **“Y”** shape of the Joshua tree. But how can I make tree using a spiral? Oh Ok, I’ve got a spiral in the left direction [*line 71*], if you do another

one there to the right and then another one up there on the top then that becomes a tree, right?

73. Akilla: Yeah, but quite complicated. I don’t know how to do that! You know how it turns left 30, if you want another branch off it, then the angle must be to the right,

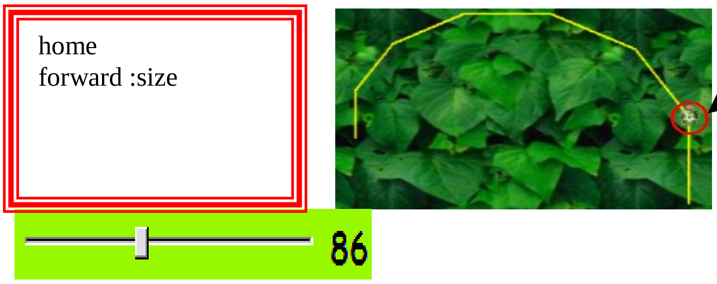
74. Tabby: and then how we are going to getting back to that point where we started? I don’t know if it’s gone *home*, or if we can do something else.

75. I asked: What do you mean by *home*?

76. Tabby: I want it to go back to the original size.

```
To tree :size
If :n < 50[stop]
Forward :size
Left turn 30
Tree :size / 1.1
```

The new position of the turtle after running the procedure with the commands in the box



```
home
forward :size
```

```
End
```

Their explanation in lines 69-71 and line 79 shows that they had difficulty in completing the task. They appreciated one of the functions of the stopping condition which can also be used as a factor to control the number of stems in the spiral. Lines 73-75 also evidenced the implementation of a recursive procedure. Working with recursive control passing (line 72 -74 and lines 78-80) in the problem-solving situation was a very difficult and ‘*complicated*’ task for them (line 73). Tabby and Akilla had difficulty in finding a way to send the turtle back to the position it started from to draw the first spiral (line 77). Akilla’s explanation in lines 78 and 80-83 showed that she had a sort of instinctive feeling about the functionality of the recursive call which was given in the program. However, Tabby was not sure about it, as in line 79 she stated that, '*I am confused!*'

77. Tabby: We need to move the turtle to another point here [*she was pointing to the end of the first trunk*]. So, if we just go forward size.
78. Akilla (interjected): Now we need another *tree*? Ah! No, we need a *tree* at the beginning.
79. Tabby: I am confused! We want *tree size* to do that spiral, and then go home, and then you wanted to go forward size, and then the *tree size* again? So, what happened then?
80. Akilla: I don't know! Probably we don't need both *trees* here!
81. I asked: What did you expect to have?
82. Akilla: Doing the first part, then the first *tree*, then going back *home*, then going up, and then turning right, and then doing another *tree*.

Working with modules of the *Spirals* domain of abstraction:

The students were then asked to work with the *Spirals* computer-based domain of abstraction. In this domain, I mainly wanted to find out how they thought about the iteration and tail recursion, the possible confluence of the iteration and tail recursion on each other, and finally the role and importance of the AVDA in their thinking about those concepts. In other words, I was looking to see how they could frame, change and evolve their thinking about the concept of tail recursion and its indispensable components in the AVDA environment.

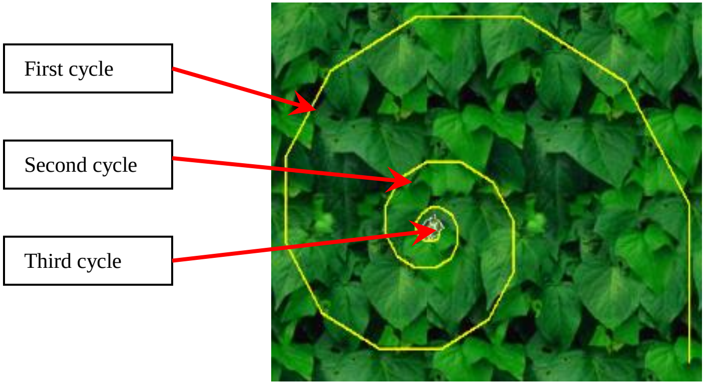
Tabby and Akilla started to work with the *Spirals* domain with the *red* technique (tail recursive). After a few minutes they moved into the *blue* technique (iterative). Their utterances in lines 83-89 show that, before using the AVDA innovation, they thought that the *blue* and the *red* techniques were the same (lines 93-96) and that there were only some syntactical differences between them (lines 84-87). Also, in line 83 they pointed to an interesting issue which was about distinguishing the process and the result (the final output).

Tabby described the *red* technique as follows after working with the *blue* technique.

83. I asked: Can you see any differences between the *red* and the *blue* techniques?
84. Tabby: The commands are different but make the same thing. Ok, *to red, if the step is less than one stop,* otherwise, *forwards* 91, and then *turns left 30*. And then it does *red step divides by 1.1,*
85. Akilla: It’s just like the one that we were doing in the previous one with a different size!
86. Tabby: Yeah!
87. My description: [*They moved back on the blue technique and Akilla said that:*]
88. Akilla: The same thing! In the *red* if ‘n’ is less than 1 it does that but let’s see what it does in the *blue*. It does *to blue ‘n’, while ‘n’ is greater than 1, same thing*! I think only *if ‘n’ is greater than one,* then it goes *forward* and then *turns left 30, make*? Is that what it means?
89. Tabby: It changes the current value of the ‘n’ with ‘n’ divided by 1.1.

To find any possible differences between those techniques Tabby and Akilla concentrated on the final output of the procedures. Akilla paid attention to counting the number of cycles of the spirals in both techniques.

90. Akilla: When you have like a big spiral, it goes on almost one, two, and three circles,



, n=135

91. Akilla added that: But when ‘n’ is small it only does one cycle.
92. Tabby: Let’s hide the turtle. I think the numbers of steps are the same, the number of actual things are the same! I’m not counting them but you know probably they have to be the same.

93. Akilla (interjected): So we have small and big sized spirals.
94. Tabby: Hang on, do the same number on both!
95. My description: [*They tried n=20 and n=150 for both techniques*]
96. Akilla: They are the same!
97. Tabby: Yeah!

The lines above show that Tabby & Akilla had difficulty in appreciating the functioning aspect of recursive calls and the control passing mechanism while they were working with the *blank box* task. However, when they started to work with the tools in the AVDA environment of the *Spirals* domain, they began to change their thinking and improve their understating of recursion.

In the AVDA environment, Tabby and Akilla slightly modified their thinking about the concept of tail recursion (lines 98-103). They described iteration and tail recursion as *"the same"* (line 96) when they were working with the *red* and *blue* techniques. However, soon after observing the animation in the AVDA environment Tabby said:

98. Tabby: What's it doing? That one [*the animation in the red technique*] is just repeating unless 'n' is less than 1. Why is it going back up? Let's do it step by step, if 'n' is less than one stop, which it's not, so it's not going to stop. It carries on forward, then left turns 30, and then it divides that value by 1.1
99. Akilla (interjected): It keeps going unless the value of 'n' is less than one.
100. Tabby: But, it's going back, isn't it?
101. I asked: What do you mean by it divides that value by 1.1?
102. Tabby: It goes forward and then left 30, then changes the value of 'n' - n divides by 1.1.
103. Akilla: The same thing as *make* in the other one [*the blue technique*]
104. Tabby: Umm, I don't know! It's different from that it is going back on itself!

The animative visualisation employed in the AVDA assisted them to picture the control passing mechanism in the recursive calls. In the line 105, Akilla pointed that the procedure is going back on itself. Her comment was very important regarding appreciation of delegatory control passing. Tabby in the line 106 also noticed this complicated control passing procedure. They were a bit confused as it was not a familiar control passing for them (line107).

105. Akilla: It’s going on until ‘n’ is less than 1 and then it goes back on itself and then stops! I want to reduce the speed to see what its doing.
106. Tabby: After stop, and when ‘n’ is less than 1, it’s going back on itself! Why doesn’t it stop then?
107. Akilla: I don’t know. I’m confused! They aren’t the same, that one [*the blue technique*] is straightforward, but I don’t know why this one is doing those steps and then, when ‘n’ is less than 1, going back on itself
108. Tabby: Yeah, that one [*the blue technique*] is much quicker than this. It’s complicated!

Tabby and Akilla changed their thinking about the tail recursion when they were working with the AVDA environment. According to Gotschi (2003), in lines 98-104, Tabby changed her thinking from a loop model – repeating – to a return-value model when she pointed to the difference between the *make* command in the *blue* technique and the recursive call in the *red* technique. Lines 104-108 show that because of the animation and the process of finishing all the generated instantiations of the original procedure by going back, they knew that something was different, but they could not see the reason and the process of control passing in the tail recursion (line 105). For them, the iterative procedure was much quicker and more straightforward (lines 107-108).

They both preferred to work with the familiar and quick iterative technique to produce a spiral rather than using tail recursive technique.

109. I asked: Which one would you prefer to use if you want to generate a spiral?
110. Tabby: The *red* one is harder than the *blue*.
111. Akilla: I like the *blue* technique because it finishes quickly. I don't particularly understand it very well, but I think it finishes the job quickly.

The other pair of students, who participated in the second iteration were Andrew and Hayley. When they were working with the *Spirals* domain, before going to the AVDA environment, they described the *red* and the *blue* technique as the same. In lines 112-116 they described the iterative procedure:

112. Andrew: To *blue*, while 'n' is greater than one you go forward 'n', left turn 30, and make 'n', a new value 'n' over 1.1.
113. Hayley (interjected): Yeah! That's right,
114. Andrew (continued): So, it has gone forward 100, and then left turn 30, at the end of each one is doing a left 30 and then stops.
115. Hayley: And whatever the length is doesn't matter.
116. Andrew: The length is reducing! Because each time 'n' is divided by 1.1.

Then they moved to the *red* technique. And in a similar way to Tabby and Akilla, they stated that *red* and the *blue* are the same. Lines 117-118 show that their conclusion was mainly based on the output and final result of the procedure rather than the process.

117. Andrew: So, that's the same thing really. The spirals are the same in both of them.
118. Hayley: Yeah they're producing the same thing!

The AVDA environment gave them a clear insight into the mechanism of control passing in both the iterative and the recursive procedures. Andrew stated that:

119. Andrew: [*In the red technique*] While ‘n’ is greater than 1 stop! Ignore that, because ‘n; is 91, it turns left 30, and it tells you to run the *red* again with step which is ‘n’ divides by 1.1, so this is ‘n’.
120. Hayley: Oh! Right.
121. Andrew: So, it’s going again!
122. I asked: Can you tell me what this line [*red :size / 1.1*] does exactly?
123. Andrew: It tells us to run this procedure again. [*He points to the first three lines of the* This is your procedure *red*

```
To Red :n
If :n < 1 [Stop]
Forward :n
Left turn 30
```

124. Andrew (continued): And this is your instruction to run *red,*

```
Red :n / 1.1
```

125. Andrew (continued): The next time it runs, it says ‘n’ divided by 1.1, so, we’ve got *red* with this value of ‘n’ and it runs it until it gets to the point that step is less than 1.

Lines 119-125 show that in the AVDA environment Andrew changed his thinking about the concept of tail recursion and paid more attention to the process rather than just focusing on the final output. Lines 123-124 evidenced that he distinguished between the procedure and the recursive call. He interpreted the recursive call as something outside the original procedure - as an “*instruction*” (line 124 and lines 135-137) that tells you to run the procedure, which is shown in line 123.

The AVDA provided them to see the latent layers of recursion. Andrew in the line 127 pointed that the control is going back to where it was started. Before, working with this, he was thinking that both iterative and recursive are the same!

126. Hayley: Why does it go back?
127. Andrew: It turns back to where it started! The ‘n’ hasn’t changed. That’s why it’s still 50.
128. My description: [*They have set the initial value of ‘n’ to be 50*]
129. Andrew (continued): Because you’re doing a procedure within a procedure within a procedure and it only finishes when it goes back to the first procedure. Do you understand?
130. Hayley: Oh yeah!
131. Andrew (added that): [*he pointed to the end command in the blue technique and said that*] This one finishes here! What you do, you start off with a procedure say A1, you do X, because it has not reached *stop* it goes to A2, and then A2 does X, it has not reached to *stop,* it goes to A3 and then A3 does X , and eventually you go to An and then An reaches *stop*, so that doesn’t carry on and goes back to A3 and that stops, it goes back to A2, stops, and A1 stops and that’s why you go back to the ‘n’ you started with. When it goes back , it starts finishing each of the previous procedures.
132. I asked: Therefore, what is the difference between these techniques?
133. Andrew: This one [*the red technique*] is using two procedures.
134. My description: [*he pointed to the procedure which is shown in line 123 and said*]
135. Andrew: This is one procedure. This is sort of saying we are going to take this here, and then we are going to run the procedure. And then we will have another procedure outside of this procedure.
136. My description: [*whilst saying that he was pointing to the recursive call and described it as another procedure outside of the red technique*]
137. Hayley (asked Andrew): You mean the *blue* is different from the *red*?
138. My description: [*He pointed to the blue technique and said*]
139. Andrew: Yeah! Actually there is nothing here to tell us to run itself it runs itself continuously until ‘n’ is less than 1.
140. Andrew: The *red* goes through, creates some and goes back, goes through, creates some and goes back. You need to keep running the procedure over and over again. The *blue* is doing the same thing but within the procedure, and by doing it in that way [*the red*] you can use this procedure [*the procedure which is shown in line 123*] somewhere else if you wanted to!

Line 131 shows that Hayley was struggling to see the differences between the two techniques. Lines 129-140 show that although Andrew described the mechanism of control passing in the tail recursion, his model of a tail recursion from a loop model transferred to the syntax model (line 138) and then a naive version of the copies model. I called his model at this stage a naive version of the copies model because he did not appreciate the recursive call as a component of the tail recursive procedure. For him, the recursive call was another procedure to call the original procedure (line 135).

Like the other students Andrew & Hayley would also preferred to use Iterative technique to make a spiral rather than recursive technique. The iterative technique for them was easier and straightforward (lines 142-144).

141. I asked: Which one would you prefer to use if you want to generate a spiral?
142. Hayley: The *blue* one I guess, because it is straightforward.
143. Andrew: I think the *blue* one is easier to understand. What is happening in the totality. In the absolute totality they're both the same! They both achieve the same result. But the way they achieve it is different. I think understanding the *blue* one is probably easier. To understand the *red* one is more difficult because you have to go back to finish the procedure that logically you thought you finished. I think this way is hard to comprehend.

5.4.6. Discussion – the *Blank-box, Spiral* and the AVDA environment

In this section of the chapter, I discuss the students' responses while they were working with the *blank-box* module and the *Spirals* and AVDA environment.

The discussion is mainly around two perspectives, the students' thinking about the concept of recursion and the design of the domain of abstraction. . The students' thinking about the concept of recursion is discussed in three categories. The first category is the appreciation of the main components of the tail recursion, and the confluences between iteration and tail recursion.

Secondly, the role and importance of the AVDA environment as a window into the students' minds and the way they think about the concepts of tail recursion and iteration to generate spirals. Finally, observing and investigating the students' thinking-in-change process within the AVDA domain.

Discussion on the students' results on pre-questionnaire:

Before working with the *Spirals* and the AVDA environment, the pre-questionnaire task described the structure of the spirals and Joshua tree in a iterative and circular way. For the students, those objects (Figure 42 (a)-(d)) had repetitive structures. For instance, in lines 22-24, Tabby and Akilla stated that they thought about those pictures "*in terms of a circle*" (line 24, Akilla). They also described the Joshua tree's branches as "[…] *just being repeated on top of different branches*" (line 40, Tabby).

From their results in the pre-questionnaire task, it appeared that the main difficulty of the students to describe and also implement the recursive structures was the functioning and functionality of the recursive calls which had a direct effect on their appreciation of the control passing mechanism. For instance, when Tabby wanted to describe spiral patterns she stated that, " [...] *it looks like a circle but you still actually draw little, little lines and put them together, the angle would be affected, but how do we repeat iteration?*" (line 37, Tabby). However, in the two cases of Kieran, and Tabby and Akilla the outcomes show a kind of naive description of the recursive structures. Kieran in lines 2-15 pointed out that the images given in the pre-questionnaire task

(Figure 42 (a)-(d)) are fractals, he also added that the Joshua tree is '*a bifurcation*' (line 2) because '*each time it stemmed double*' (line 4).

Also, in Tabby and Akilla's case, Tabby described the Joshua tree as a 'Y' shape structure which '*repeats itself somewhere else*' (line 44). Afterwards, in lines 49-51, they pointed to an immature, nevertheless important, issue. Akilla described the Joshua tree as a 'Y' shape structure, '*and then 'Y' over 'Y's becoming bigger like a big 'Y''* (line 50). Although it was not developed enough, that was an important description. In other words, Akilla intuitively saw the recursive calls as triggering the whole thing ('Y's) but in a smaller scale until reaching final output (the big 'Y') at the end. Tabby stated the Joshua tree (Figure 42-(a)) as a whole does not look like a big 'Y'. However, it has 'Y's in its structure. She distinguished between bottom-up and up-bottom approaches by saying, "*if we look at this image, are you going to look at the whole thing and then go in or start at the smaller one and then go up?*" (line 51, Tabby).

One of the key design features in the second iteration was to focus more on the functioning aspects of the components of the recursive procedures. To do so, the *blank-box* module and the AVDA innovation within the *Spirals* computer-based domain were designed and tested. As mentioned before in Table 7, the second task that the students were asked to work with was the *blank-box* module. They could only finish the module successfully if they had a clear understanding about the mechanism of control passing in the recursive procedures. In other words, they had to be aware of the functioning and functionality of the recursive calls.

The module was designed in order to provide students with the opportunity to create and implement a recursive procedure on their own rather than working with pre-made procedures. It was designed in such a way that, the students have had the opportunity to use and work with the concept of recursion and its components. Thus, it provided me as a researcher a rich window into their mind to see how they think about the concept of recursion and its essential components like the recursive call(s) and the base case(s). The results of the students' working with the *blank-box* module revealed that they had a major difficulty with tracking the flow of control in the recursive procedure. For instance, Kieran tried to send the turtle back to where it started to draw a spiral, but he did not know how to do that (lines 54-57). His second try in line 58 showed that he had a sequential model in his mind to complete the task, by repeating the commands *forward :size / 2* twice and deleting the given recursive call. Lines 60-64 also reveal that he had difficulty with the functioning (how part) of the recursive calls. The reason is he knew what he was going to generate (functionality, the what part) because in line 60 he said, "*I have got the first two branches now*". But he did not know how to make the other branches on top of it by sending the turtle back to the right stems. In line 53, he asked why the turtle drew the first two branches only once. This showed me that he did not have enough understanding about the functioning of the commands that he entered into the blank box.

In the case of Tabby and Akilla, it revealed that they had difficulty with the functioning of the main components of recursion. Tabby and Akilla made a good connection between the length of the last stem (the stopping condition) and the number of the stems. In line 68-69, the result that they had was only the main trunk of the tree that they wanted to make. Similar to Kieran's account, Tabby and Akilla also had difficulty tracking the flow and sending the turtle back to where it was started (lines 70-74). As an example, when Tabby (line 72) said "*I consider a 'Y' shape structure for tree, something like a Joshua tree*", Akilla responded "*But it's quite complicated*' (line 73). Altogether, the results show that this module was a very challenging module for the students as they needed to consider both functioning and functionality for the components of the recursive procedure in their mind simultaneously (lines 72-74 and 77-80). The students' thinking and mental models of the iterative and tail recursive procedure were scrutinized and observed through the window that the AVDA innovation provided me within the *Spirals* environment.

There is strong evidence to show that, at the beginning, both iterative and tail recursive procedures were considered to be the same for the students who participated in this study. For instance, in lines 83-88, by focusing on the final output instead of the process of making that output, both Tabby and Akilla stated that both techniques made the same things. By paying more attention to the commands in the *red* and *blue* procedures, Akilla pointed to some syntactical differences between those two techniques. She stated that those two techniques are the same thing and pointed to the difference between the stopping condition in the *red* and *blue* technique (line 88). Working in the

AVDA environment helped them to develop their thinking about the differences between the mechanisms of those two techniques. Their utterances in lines 98-104 showed that they considered the recursive call to be a command which generates a new value for the ‘n’. Tabby described the *red* technique in line 98 as a repetitive, or loop, mechanism. However, she was not sure about the functioning of the recursive call as a “*returning value*” model of thinking about recursion (Gotschi, 2003).

Andrew and Hayley, the other pair of students, also responded in a similar manner to Tabby and Akilla’s account. In lines 114-118, they stated that the *red* and *blue* techniques are the same. But, soon after working in the AVDA environment, they recognised some syntactical differences (lines 119-125). In line 123, Andrew described the recursive call as an instruction to run *red*. Therefore, within the AVDA environment, they developed their thinking from a loop model to the syntax model – by recognising the syntactical differences – and then eventually a sort of naive version of the copies model of recursion in lines 126-135. Lines 126-131 evidenced that Andrew’s description of the *red* technique was very close to the viable copies model of recursion. However, he did not consider the recursive call to be an essential component of the *red* procedure. Instead, he considered the recursive call as ‘*another procedure inside of the red procedure*’ (lines 132-135).

There is strong evidence in the results to show that all the students would prefer to use the iterative technique to generate a spiral rather than the tail recursive technique. The students stated that the *blue* technique (iterative) is much quicker, more straightforward, and easier to understand. On the contrary, the *red* technique (tail recursive) was characterised as harder, more complicated and more difficult to understand (for instance, see lines 108-111 and lines 141-143). Andrew, in line 143, described the tail recursive technique as follows: "*It is more difficult because you have to go back to finish the procedure that logically you thought you finished*".

The students' accounts of this iteration also provided me with a few more important insights into the design of the next (and final) iteration of this research, which is explained in the next section of this chapter. The third iteration, the *Treebuilder* computer-based domain, is thoroughly discussed in the next chapter of this thesis.

5.4.7. Issues & conjecture(s) for the next Iteration

The results obtained from this iteration were quite promising and insightful for the design of the next stage of this study within a DBR framework. The computer-based domains in the second iteration, the AVDA innovation within the *Spirals* domain of abstraction and the *blank-box* module, were designed based on the results that gained from the first iteration. The *Spirals* domain had a significant difference with its precedent in the first iteration the *Treemenders*. The above discussion on the students' explanation and experiences with the domain showed that the *Spirals* presented the hidden layers of the mechanism of the control passing in the tail recursion to the students. Although, the students showed some difficulties in handling the flow in the tail recursive procedure, but the *Spirals* helps them to change their mind and appreciate some differences between tail recursion and iteration.

The *Spirals* domain depicted the latent layers of the control passing mechanism in a tail recursive procedure by employing animation techniques and colour-coding. The analysis of data in the previous section evidenced a significant role of AVDA innovation in students' thinking-in-change process regarding tail recursion.

From functional abstraction perspective, the *Spirals* domain successfully presented both functionality and functioning aspects of the components of the tail recursive procedure by modelling spirals. Although the first two modules of the *Spirals* domain – the *red* and *blue* modules – were mainly focused on functionality aspect of the concept, but there were some signs of functioning by giving the students the opportunity of running the procedure in the step-wise mode.

The comparison module was depicting both functioning and functionality aspects together in one screen. The students could see the final output of the procedure on the screen (the functionality – *what* part). Also, they could see and observe the animative approach contrived in AVDA visualisation besides the colour-codes on the lines of the procedures to realise and appreciate *how* the control is passing around the procedure (the functioning aspect).

After doing these two iterations, the concept of embedded recursion could now be tackled. The results in the first iteration convinced me to focus more on the functioning aspect of the essential parts of the concept of recursion, and to create and design more visible visualizations to uncover the mechanism of the control passing in the recursive procedures.

Hence, I decided to focus on the tail recursive procedures. I embodied the conjectures that emerged from the first iteration by using spirals in the AVDA environment. The results of the second iteration convinced me that using animative visualization can acts as a dual window. On one hand, it provided me as a researcher with an opportunity to look into the students' minds and observe how they think about the concept of tail recursion, and also to explore their thinking-in-change process through that window. Conversely, it offered the students a window through which they could look into the latent layers of the concept of tail recursion and its complicated control passing process.

Consequently, the following conjectures can now be discussed and reported on. This leads to the final stage of the design as follows:

- By using objects that instantiate the output from a recursive procedure, students will attend to the functionality of the recursive procedure,
- Having experience of the flow of control in iteration and tail recursive procedure students will be able to recognise the flow of control in embedded recursion.

However, I decided to develop both the AVDA and the *blank-box* tasks for the next iteration of the embedded recursive procedures. My attention was drawn towards embodying and phenomenalizing the emerging conjectures by contextualization of the embedded recursive procedure by using binary and ternary trees. To maintain the focus on the functioning aspect of the concept of recursion, I also designed a developed version of the *blank-box* module for the next iteration. Having taken into account the above issues, I designed and programmed *Treebuilder* as the final computer-based domain of abstraction for the third iteration of this research within a DBR framework. There were several innovations in the *Treebuilder* computer-based domain which are discussed thoroughly in the next chapter.

6. Iteration Three – Tool design of the *Treebuilder*

6.1. Overview

The present chapter discusses the design aspects of the third (and final) iteration of this research within a DBR framework. The computer-based domain of abstraction which was designed for this iteration is called *Treebuilder*. Figure 43 (below) shows the main interface of this computer-based tool.

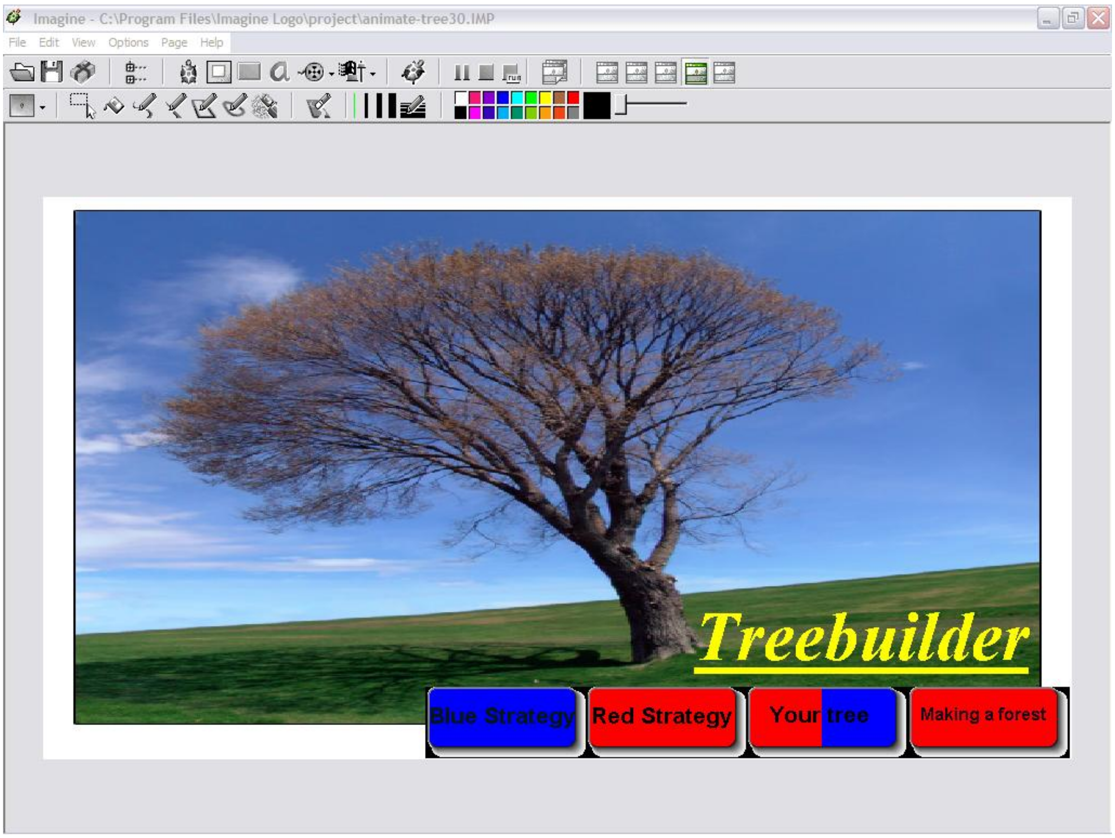

Imagine - C:\Program Files\Imagine Logo\project\animate-tree30.IMP
File Edit View Options Page Help
Treebuilder
Blue Strategy
Red Strategy
Your tree
Making a forest

Figure 43-The main interface of the *Treebuilder* domain

The first iteration – the *Treemenders* – was mainly an exploratory phase. During this phase, the primary focus was on discovering the problematic issues regarding students' thinking about the concept of recursion from a functionality perspective. The second iteration – the *Spirals* – was designed based on *post-*

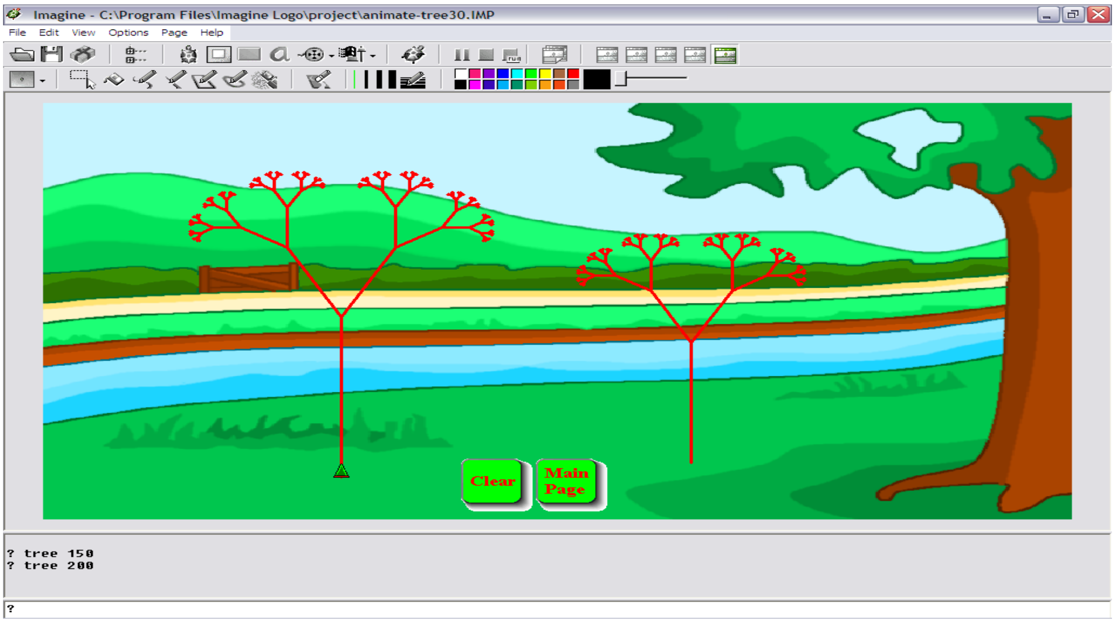
hoc issues, results, and conjectures emerging from the first iteration. At this stage the focus was for the most part placed on tail-recursive processes and their relationship with the iterative process from both functioning and functionality perspectives. Based on the vital results that emerged from the second iteration, the third iteration was designed with a special focus on the functioning aspect of the embedded recursion and its components. Further discussion of this aspect is detailed later in this chapter.

I begin this chapter by explaining my approach to this, third and final iteration. This section also includes a description of the complicated control passing mechanism in the recursive procedure. This is illustrated using flowcharts which appear later in this chapter. The chapter continues by describing the tool design of the *Treebuilder* domain of abstraction. Subsequently, the functioning aspects of those modules are discussed, and finally the chapter finishes with a summary. The tool use of the third iteration, which concentrates on explaining and discussing the students' accounts, is discussed in the next chapter.

6.2.My Approach – *Treebuilder*

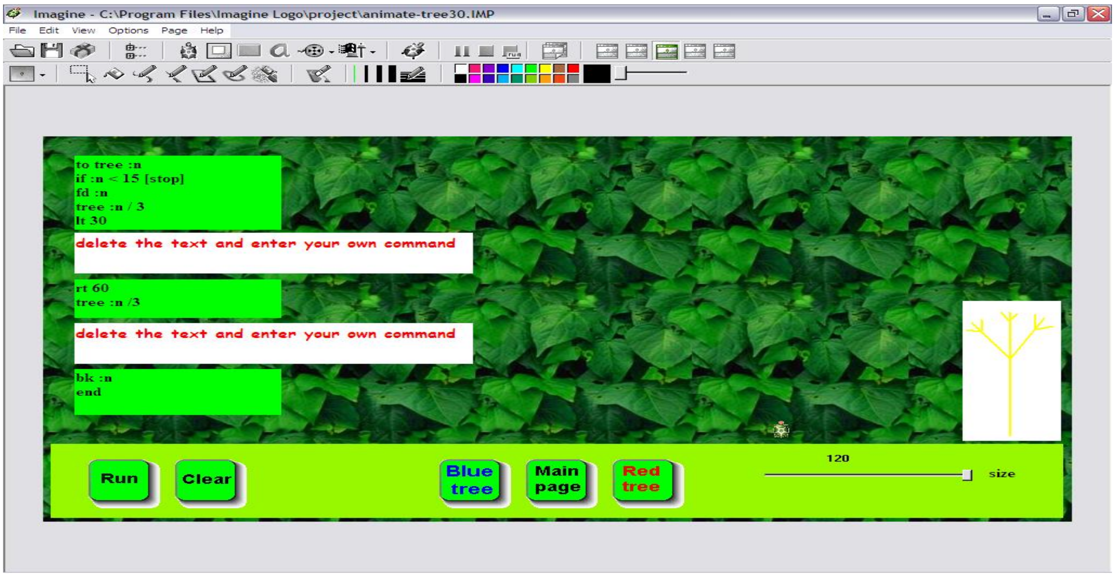
To implement the *Treebuilder*, I worked with 17 student volunteers. They were mathematics specialists who were studying on a four year degree program, and were training to be primary school teachers. They attended the interviews and participated in the tasks. There were seven pairs and three individuals. Each interview session lasted 1.5 hours.

The domain of abstraction in this iteration – the *Treebuilder* – was a direct result of the previous iteration – the *Spirals* – and the AVDA innovation. Regarding the results of the second iteration, I decided to employ and develop that approach (AVDA) for the embedded recursive procedures. The *Treebuilder* domain of abstraction was designed with four modules; *making a forest*, the *blue* strategy, the *red* strategy, and *your tree*. The pictures below (figures 44-47) show the main interface of the modules of the *Treebuilder* computer-based domain of abstraction for the modelling of binary trees.


Imagine - C:\Program Files\Imagine Logo\project\animate-tree30.IMP
File
Edit
View
Options
Page
Help
Clear
Main Page

```
? tree 150
? tree 200
?
```

Figure 44-The main interface of the *making a forest* module



Imagine - C:\Program Files\Imagine Logo\project\animate-tree30.IMP

```
to tree :n
if :n < 15 [stop]
fd :n
tree :n / 3
lt 30
```

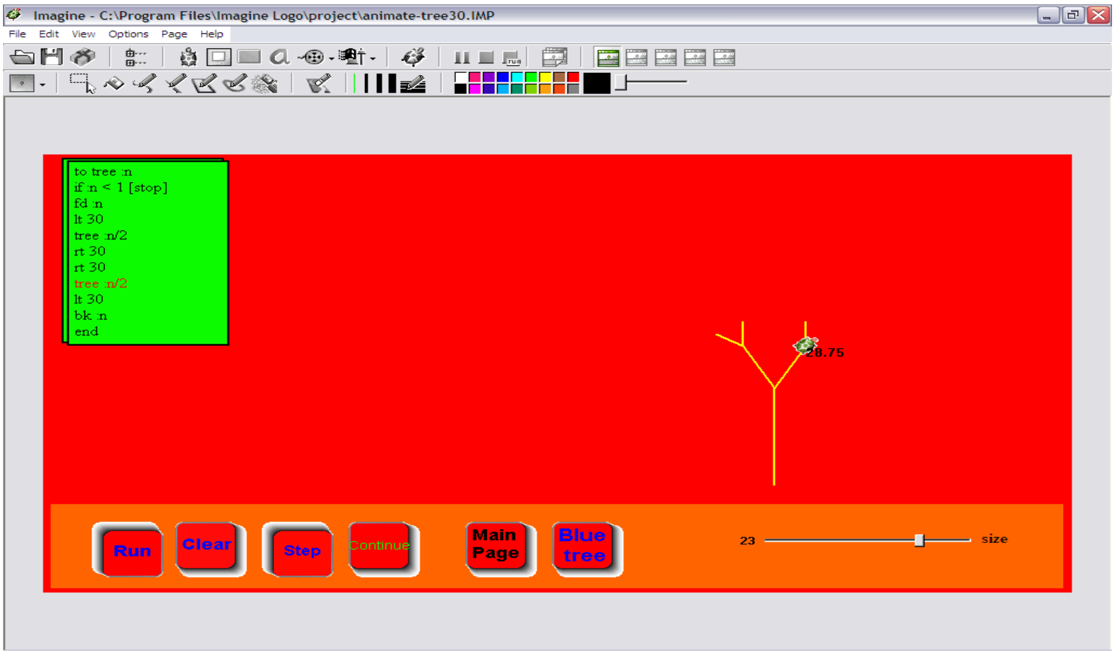
delete the text and enter your own command

```
rt 60
tree :n /3
```

delete the text and enter your own command

```
bk :n
end
```

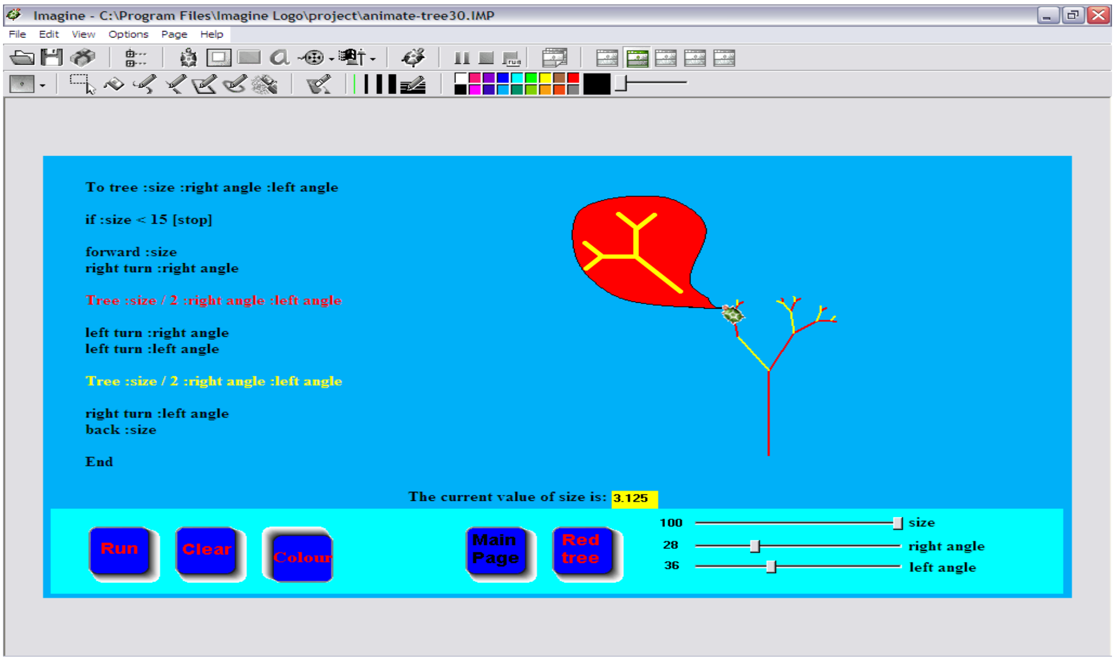
Figure 45-The main interface of the *your tree* module



Imagine - C:\Program Files\Imagine Logo\project\animate-tree30.IMP

```
to tree :n
if :n < 1 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

Figure 46-The main interface of the *red* strategy module


Imagine - C:\Program Files\Imagine Logo\project\animate-tree30.IMP
File
Edit
View
Options
Page
Help
Run
Clear
Colour
Main Page
Red tree

```
To tree :size :right angle :left angle

if :size < 15 [stop]

forward :size
right turn :right angle

Tree :size / 2 :right angle :left angle

left turn :right angle
left turn :left angle

Tree :size / 2 :right angle :left angle

right turn :left angle
back :size

End
```

The current value of size is: 3.125

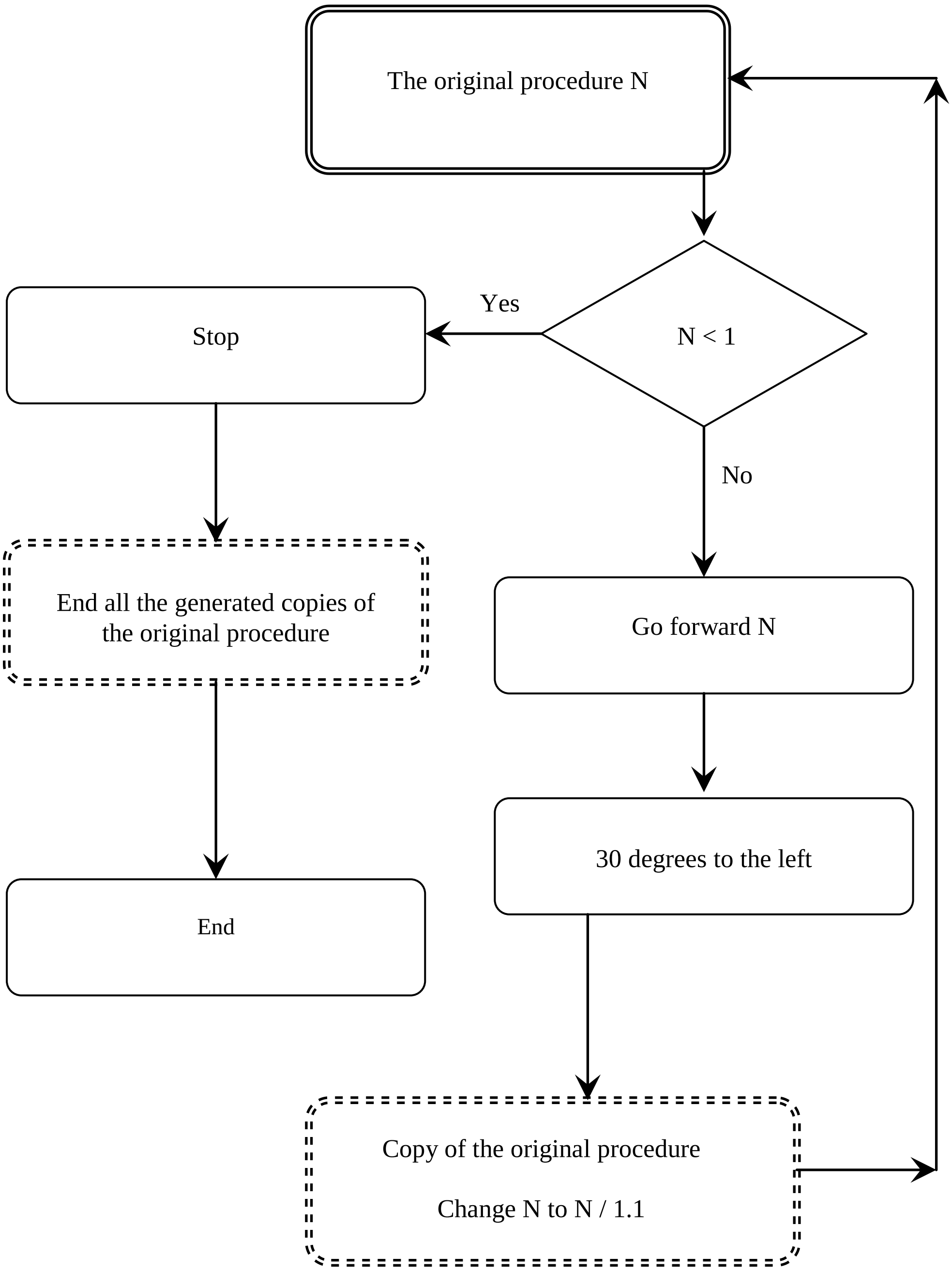
100 size
28 right angle
36 left angle

Figure 47-The main interface of the *blue* strategy module

After designing the third domain – the *Treebuilder* – I decided to combine it with the *Spirals* computer-based domain. This decision was made based on the interconnections between the tail and embedded recursion concepts. The resulting conjecture initially emerged after the first iteration, the *Treemenders*. During the activity, students showed immense difficulty in tracking the flow of control over the procedure (iteration one: Feng, lines 11-15 & Sarah & Jin, lines 110-118). This observation gave me the initial insight to study tail and embedded recursive procedures separately. Combining the *Spirals* and the *Treebuilder* in the third iteration provided me with a more concise picture of how the students' were developing their thinking in response to the tools and therefore the concepts of tail and embedded recursion.

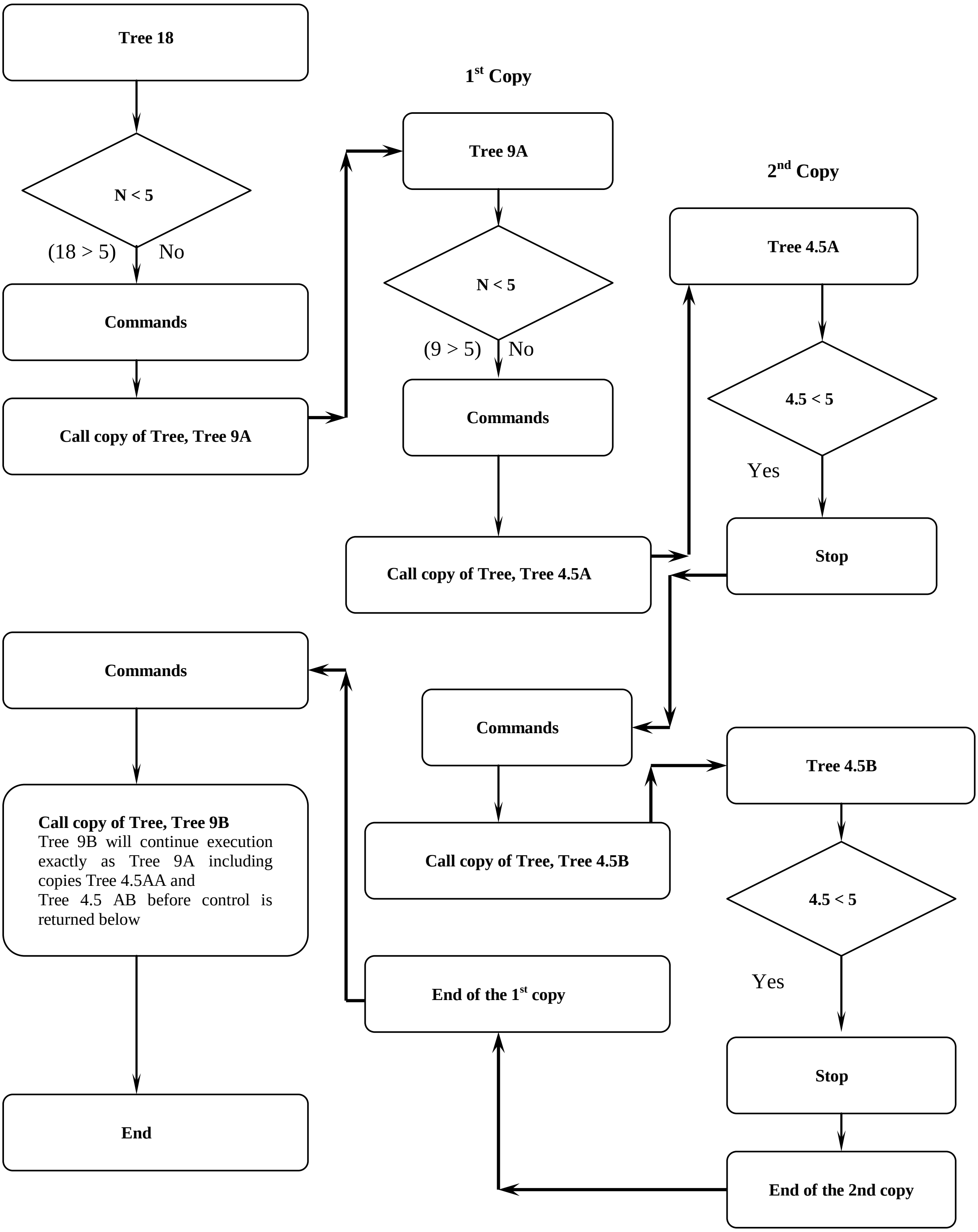
The intricate mechanism of the control passing structure of embedded and tail recursive procedures is shown in the flowcharts below. The first flowchart is designed to show the control passing mechanism in a tail recursive procedure. The procedure described by flowchart one was used in the *red* technique (recursive) within the *Spirals* domain. The second flowchart is designed to show the control passing mechanisms of the embedded recursive procedure which was used in the first and third iterations, the *Treemenders*. Finally, flowchart three describes the mechanism of passing control between the two recursive calls in an embedded recursive procedure.

Control passing process in a tail recursive procedure



Flowchart 1- The control passing mechanism in a tail recursive procedure

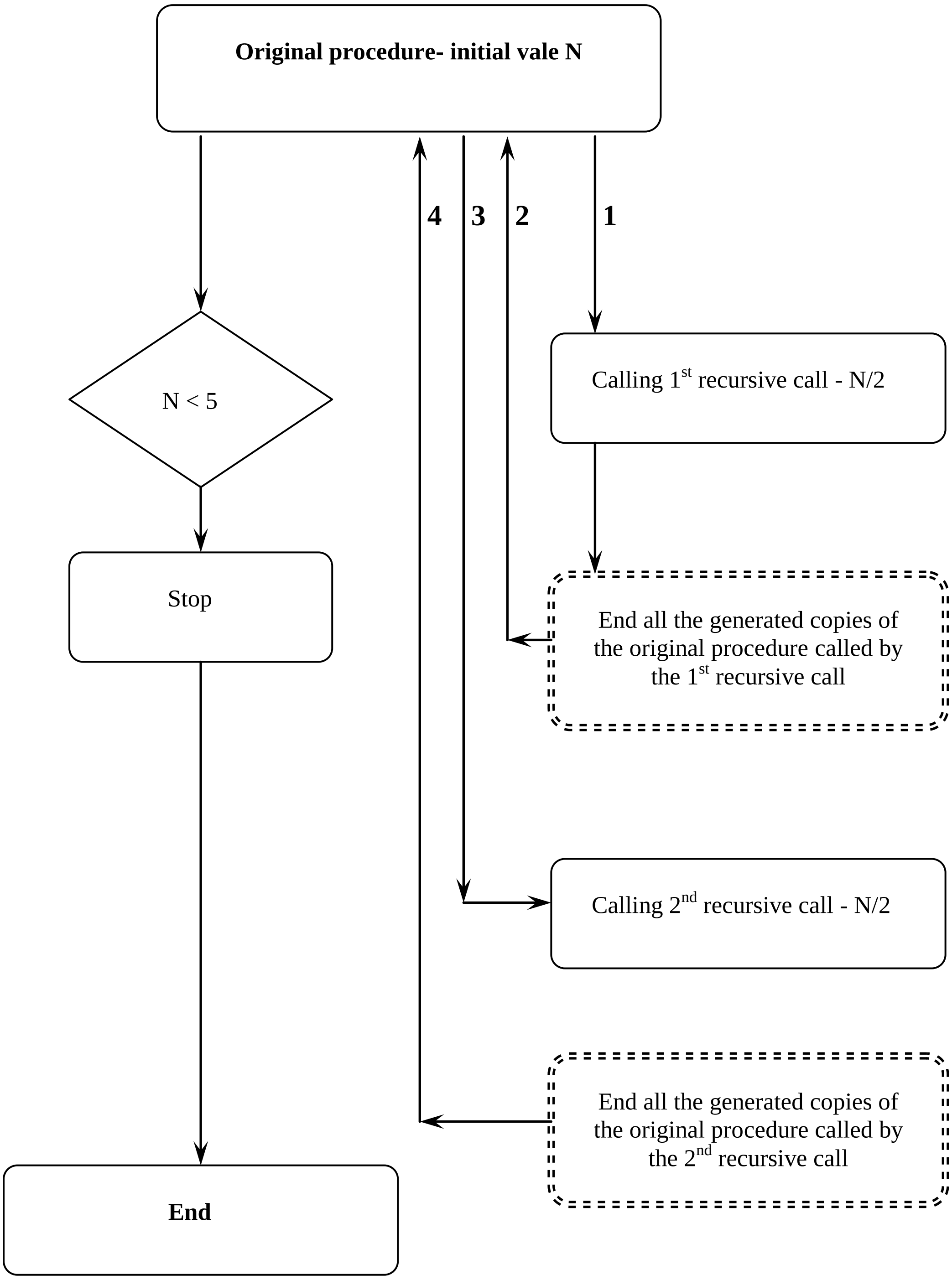
Control passing mechanism in an embedded recursive procedure



Tree 18
N < 5
(18 > 5)
No
Commands
Call copy of Tree, Tree 9A
1st Copy
Tree 9A
N < 5
(9 > 5)
No
Commands
Call copy of Tree, Tree 4.5A
2nd Copy
Tree 4.5A
4.5 < 5
Yes
Stop
Commands
Commands
Call copy of Tree, Tree 9B
Tree 9B will continue execution exactly as Tree 9A including copies Tree 4.5AA and Tree 4.5 AB before control is returned below
Call copy of Tree, Tree 4.5B
Tree 4.5B
4.5 < 5
Yes
End of the 1st copy
Stop
End
End of the 2nd copy

Flowchart 2- The mechanism of flow in an embedded recursive call

Control passing mechanism between two recursive calls



Original procedure- initial vale N
4
3
2
1
Calling 1st recursive call - N/2
N < 5
Stop
End all the generated copies of the original procedure called by the 1st recursive call
Calling 2nd recursive call - N/2
End all the generated copies of the original procedure called by the 2nd recursive call
End

Flowchart 3- the delegation of control between two recursive calls

The tasks and modules of the *Spirals* and the *Treebuilder* computer-based domains in the third iteration were implemented in the following order as a direct result of the second iteration:

Module		**Order of implementation**	**Purpose**
Spirals	***blue* technique (Iterative)**	First task	Evaluating the student's thinking and thinking-in-change about the iterative procedure in the window of the tool.
	***red* technique (Recursive)**	Second task	Evaluating the student's thinking and thinking-in-change about the tail recursive procedure in the window of the tool.
	Comparison	Third task	Evaluating the student's thinking and thinking-in-change through the window of the AVDA environment.
Treebuilder	***Making a forest***	Fourth task	To embody the conjecture about the functionality of recursion.
	***blue* strategy**	Fifth task	Evaluating the students' ability to track the flow of control in the AVDA environment over the output as well as to employ colour codes for the recursive calls – a link between functionality and functioning.
	***red* strategy**	Sixth task	Evaluating students' thinking about the embedded recursive procedure in the AVDA environment using a step-wise animative approach over the commands of the procedure.
	your tree	Seventh task	Figuring out the students' thinking and thinking-in-change about embedded recursion and its components.

Table 9- The order of the tasks in the second iteration

Table 9 is divided into two main sections. The first part is about the order of implementation and purpose of the *Spirals* domain's modules and the second part presents the order of implementation and purpose of the *Treebuilder* domain's modules in the third iteration.

In the first place, during the activity, the students were asked to work with the *Spirals* modules, which enabled me to see their thinking and mental models regarding the iterative and tail recursive procedures through the AVDA environment within the *Spirals* domain of abstraction. Then, they were asked to work with the modules of the *Treebuilder*, which were mainly designed to investigate the students' thinking and thinking-in-change process about the embedded recursive procedures. The students were asked to start working with the modules of the *Treebuilder* domain of abstraction in the order that is explained in Table 9.

The students' activities and reactions while they were working with the modules of the third iteration were recorded using a Camtasia screen recorder. Camtasia enabled me to record their utterances during working with the modules. It allowed me to record all the non-verbal moments while they were working with those modules. As a participant observer, I carefully observed their responses and reactions. I only intervened in their experiments to ask open-ended questions like: *What do you think? Why is it working like that? What if it is working like that?* The students' responses to these sorts of questions enabled me to have more of an opportunity to understand how and what they thought about the concept of recursion and also how the thinking-in-change process was shaped and framed in their mind. The next two sections of this chapter concentrate on the key features of the design development and tool use of the modules of the third iteration.

6.3.Tool design – *Treebuilder* and the AVDA environment

The key design features of the modules of the *Treebuilder* domain of abstraction resulted from the conjectures that emerged from the previous iteration. From a design perspective, the final iteration was the continuation of the substantial innovation of AVDA in the second iteration within the *Spirals* domain of abstraction. As mentioned above, the *Treebuilder* has four modules. The following diagram shows the modules of the *Treebuilder* domain of abstraction.

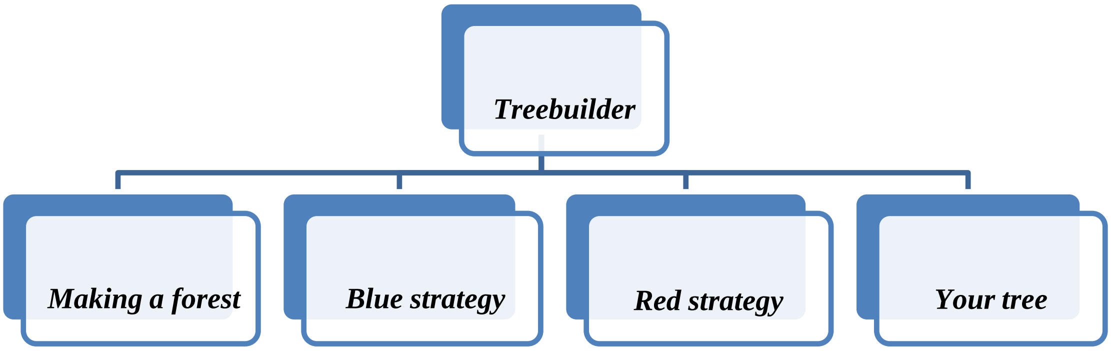


Figure 48-The modules of the *Treebuilder* domain of abstraction

The prominent feature of the *Treebuilder* domain of abstraction employs the animative techniques of AVDA in the output of the embedded recursive procedure while it is being drawn by the turtle. This technique is fully described later on in the *blue* strategy section.

Table ten summarises the design aspects of the four modules of the *Treebuilder*. The first row of the table shows the control box of the main menu of the *Treebuilder* domain (Image 1 & 2). This control box contains four buttons. On activation, each button is highlighted on screen to help the students to keep track of which button is currently activated. In the second row, the control box of the *making a forest* module is shown (Images 1-3). The third row of the table shows the control box of the *blue* strategy module (Image 1) and the sliders which were designed for the length of the initial size and the angles for branching to the left and right (Image 2). Images 3 & 4 of the third row also depict a box that shows, the current value of the size of the new stems, which were being drawn by the turtle. The background of the box (red and yellow) shows the colour codes which were used for the branches to the left and right respectively. Image five, in the third row shows the colour codes (red and yellow) which were used for the first and second recursive calls in the given embedded recursive procedure. Images 1, 2, & 3 in the fourth row of the table show the control box and the slider for setting the initial size in the *red* strategy module. Finally, the fifth row of the table shows the control box and the slider for the initial size of the tree in *your tree* module in the *Treebuilder* domain of abstraction.

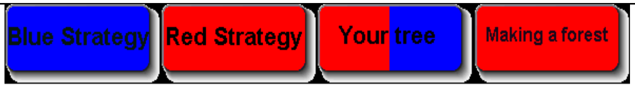
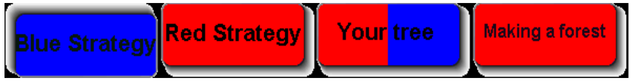
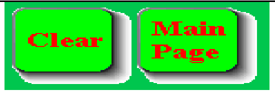
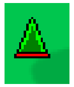
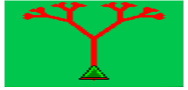
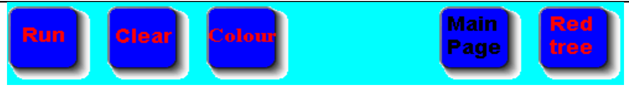
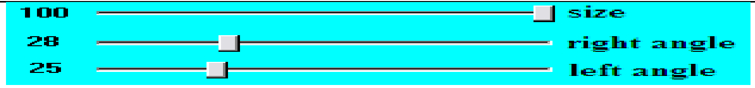
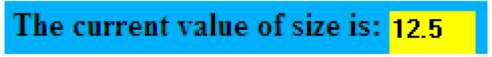
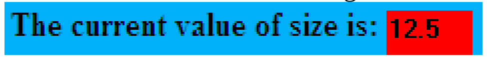
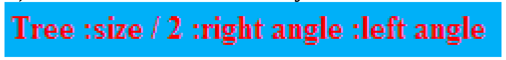
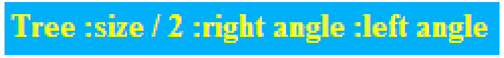
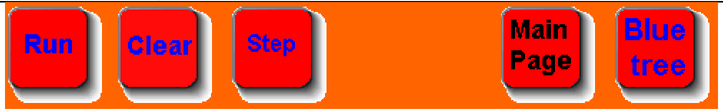
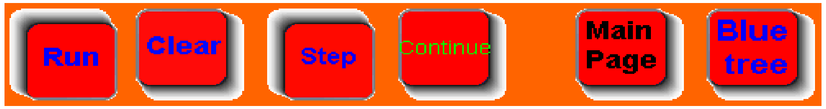
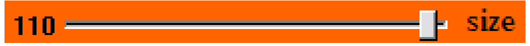
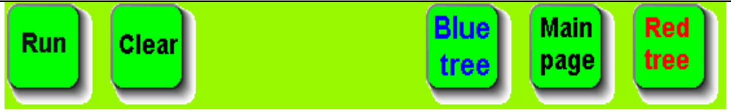
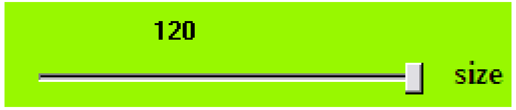
Module	Design features
Main interface	 Blue Strategy Red Strategy Your tree Making a forest  Blue Strategy Red Strategy Your tree Making a forest 1) *blue* strategy, *red* strategy, *your tree*, and *making a forest* 2) *blue* strategy button is activated
Making a forest	 1)The control box of the *making a forest* 2) The *clear* button is activated 3) The *Main page* button is activated   4) The turtle plants a new tree 5) A tree which is planted by the turtle
***blue* strategy**	  1)The control box 2) The sliders in the control box to set angles and size  The current value of size is: 12.5 (yellow background)  The current value of size is: 12.5 (red background) 3)The box for the size of a yellow branch with its colour code 4) The box for the size of a red branch with its colour code  Tree :size / 2 :right angle :left angle  Tree :size / 2 :right angle :left angle 5) The red and yellow colour codes for the first and second recursive calls (branches to the left – red colour and to the right – yellow colour)
***red* strategy**	  1)The control box of the *red* strategy 2) After activating the *step* button  5)The slider for the initial size
your tree	  1) the control box 2) The slider for the initial size

Table 10- The design features of the four modules of the *Treebuilder* domain of abstraction

In the following part of this chapter, the technical aspect of each of the above mentioned modules of the *Treebuilder* domain of abstraction are explained.

6.3.1. The main page of the *Treebuilder* domain of abstraction

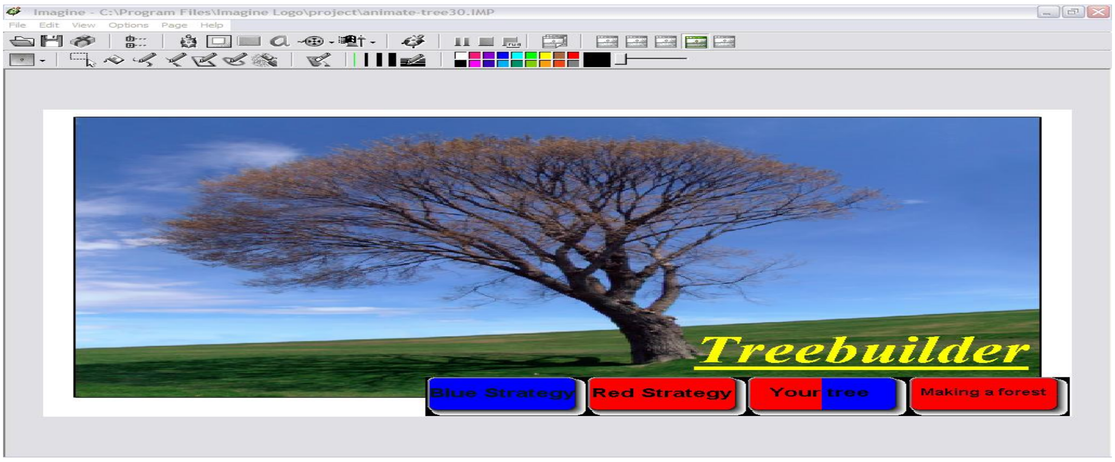
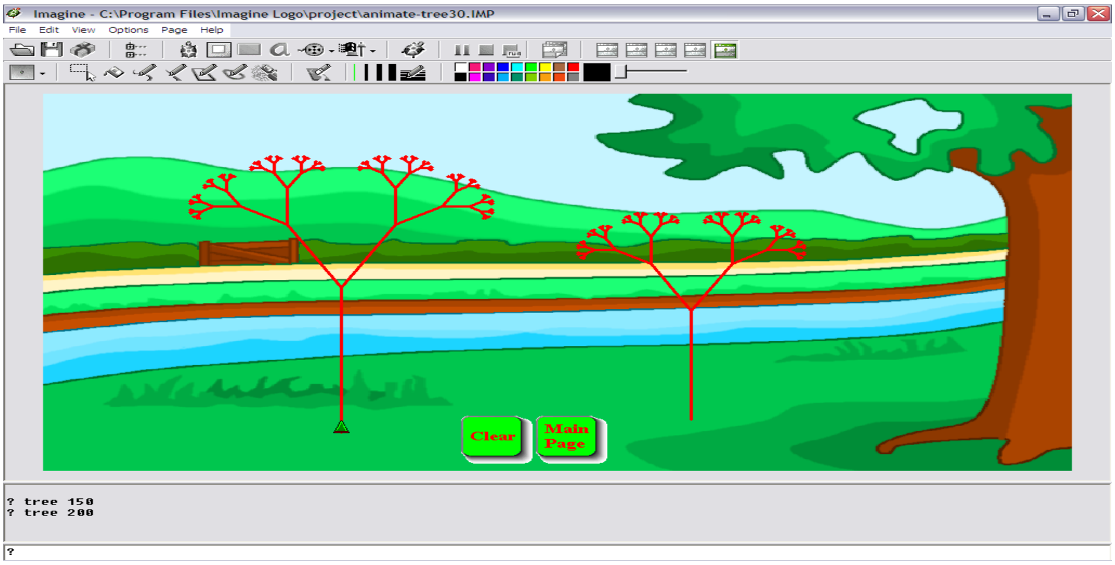


Figure 49- The main page of the *Treebuilder* domain of abstraction

Figure 49 shows the main page of *Treebuilder*. Four buttons were created on this page in accordance with the four modules of this computer-based domain. *Students* were able to move to each one of the quadruplet modules of *Treebuilder*. Technically, both the *blue* and *red* strategies in this domain of abstraction were used to provide an appropriate window for the students to look through and think in-depth about the embedded recursive procedures and see the mechanism of control passing in these procedures. The *your tree* module is a developed version of the *blank-box* module in the second iteration. In the following part of this chapter, the tool development of the modules of the *Treebuilder* is discussed.

The first module that the students were asked to work with was *making a forest*. To activate the *making a forest* module, the students were required to press the Making a forest button. The button Making a forest becomes highlighted as if pressed down after activation. Design features of this module are discussed in the next section of this chapter.

6.3.2. *Making a forest* – *Treebuilder* domain of abstraction


Imagine - C:\Program Files\Imagine Logo\project\animate-tree30.IMP
File Edit View Options Page Help
Clear
Main Page

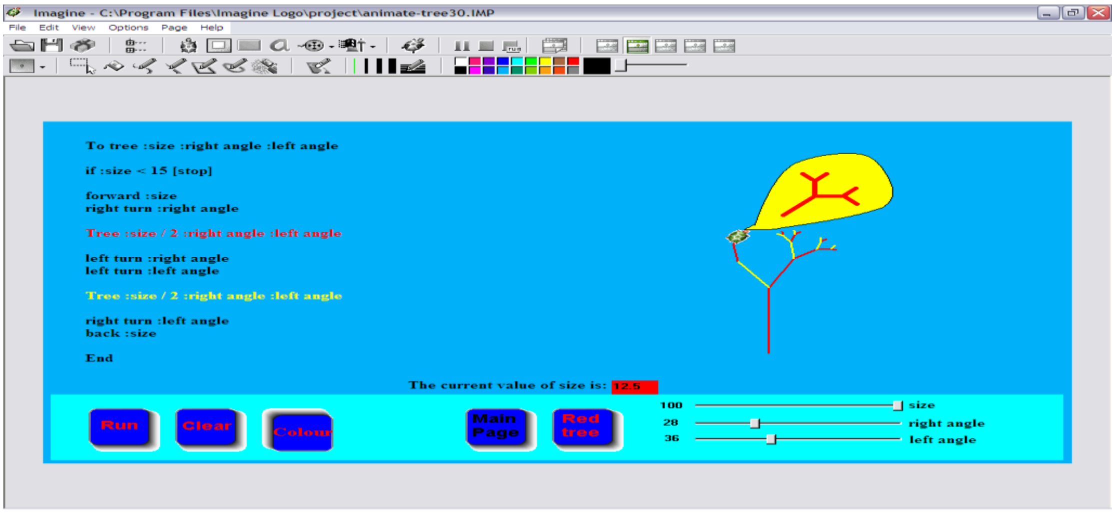
```
? tree 150
? tree 200
?
```

Figure 50- The interface of the *making a forest* module

The interface of the *making a forest* module is shown in the above picture. The students who participated in this module were supposed to type the term ‘tree’ on the command line at the bottom of the screen and press the ‘enter’ button to see an image of a tree on the screen. The design features of this module were realised by addressing the following issues. The first issue was designing a new shape for the turtle ▲ (image (4) second row of Table Two). This new shape for the turtle was designed to be a metaphor representing a seed which will grow in to a tree. The other design issue was the ease of use and movement of the turtle. The students were able to change the turtle's location by clicking on it and dragging it to a new place to draw a new tree. As shown in Table Two of this module, the students were given two *clear* and *main page* buttons. By pressing the *clear* button, they could wipe the screen for another test, and by pressing the *main page*, button they could move to the main page of the *Treebuilder* domain of abstraction.

The purpose of this module from a design perspective, for students, was to provide a window to make some bridge between the functionality i.e. *what* they want to have and the functioning, *how* it will be done, and what are the crucial components of its structure. The purpose of the *making a forest* module from a design perspective for me, as researcher, was also to provide a window into the students' minds to investigate how they think about a recursive structure. I was then able to examine their explanations when they were looking at a recursive structure, a binary tree on the screen, before seeing and knowing anything about the program behind it. The students needed to describe the crucial parts of the binary tree shape that they produced on the screen by typing the term 'tree' on the command line. A full tool use account for this module is discussed in the tool use section of this chapter. The next section concentrates on the *blue* strategy module from a design perspective.

6.3.3. *Blue* strategy – *Treebuilder* domain of abstraction



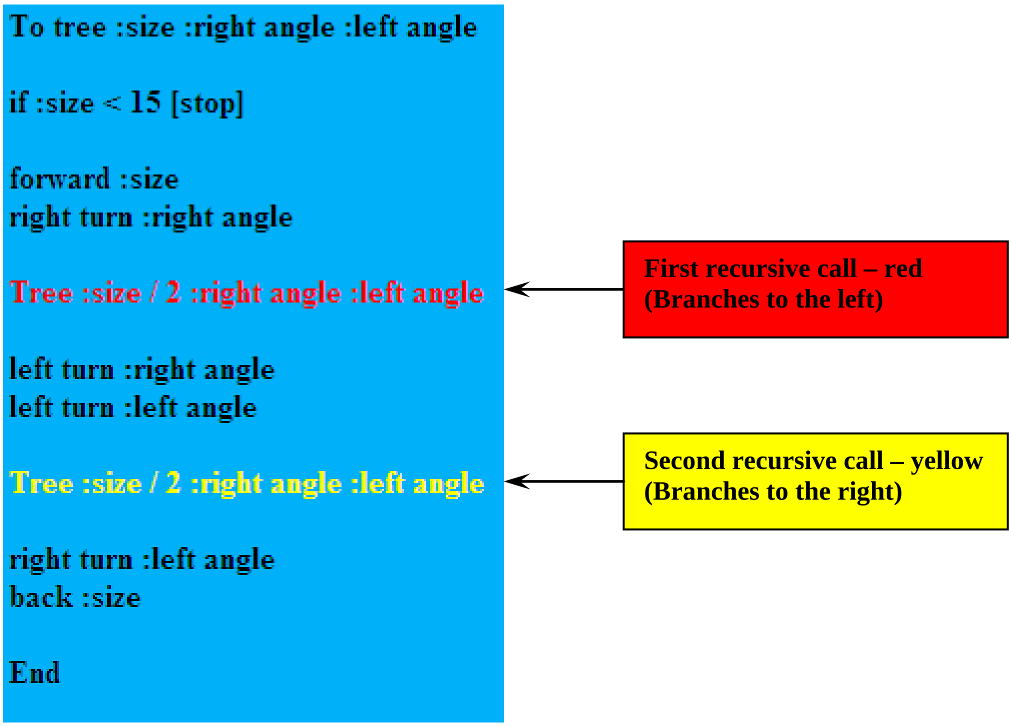
```
To tree :size :right angle :left angle

if :size < 15 [stop]

forward :size
right turn :right angle

Tree :size / 2 :right angle :left angle

left turn :right angle
left turn :left angle

Tree :size / 2 :right angle :left angle

right turn :left angle
back :size

End
```

The current value of size is: 12.5

Imagine - C:\Program Files\Imagine Logo\project\animate-tree30.IMP
File Edit View Options Page Help
Run
Clear
Colour
Main Page
Red tree
100 size
28 right angle
36 left angle

Figure 51- The interface of the *blue* strategy module

Figure 51, shows the main interface of the *blue* strategy. This module provides the students with an embedded recursive procedure with two recursive calls to model a binary tree.



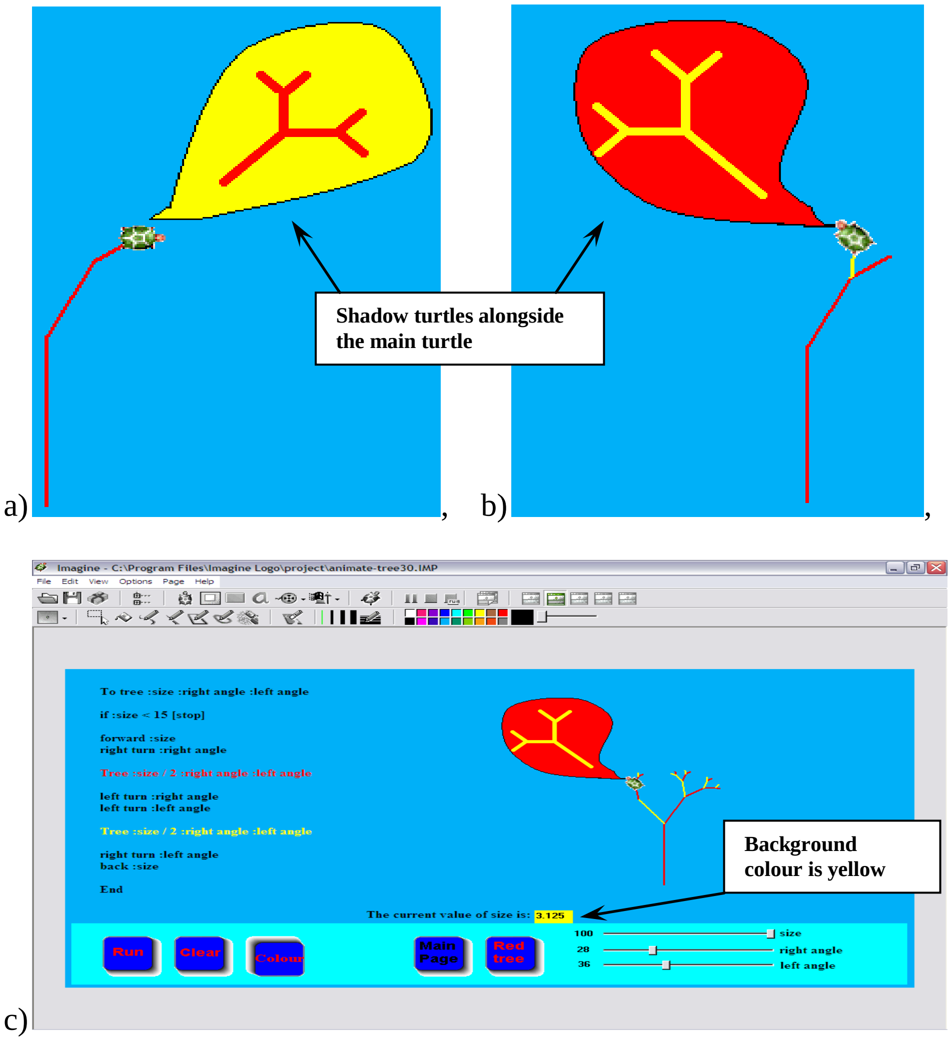
```
To tree :size :right angle :left angle

if :size < 15 [stop]

forward :size
right turn :right angle

Tree :size / 2 :right angle :left angle

left turn :right angle
left turn :left angle

Tree :size / 2 :right angle :left angle

right turn :left angle
back :size

End
```

First recursive call – red (Branches to the left)

Second recursive call – yellow (Branches to the right)

Figure 52- The embedded recursive procedure in the *blue strategy*

In this module, I used the colours red and yellow as colour codes for the two recursive calls respectively (Figure 52). The other substantial design feature for this module was employing the AVDA innovation over the output rather than the procedure. I previously mentioned that the AVDA innovation was invented in the second iteration in which I used animative visualisation to show the generation of the new copies of the original procedure to the students. In this module, I employ the AVDA animative approach on the tree which is being drawn by the turtle rather than the procedure. Therefore, this module is an output-based AVDA approach. The idea emerged from the explanation of one of the students in the previous iteration, '*you are going to look at the whole thing and going to start at the smaller one and then go up*' (Tabby, line 51, Chapter 5). I designed two shadow turtles to move alongside the main turtle over the tree (Figure 53, (a) & (b)).


a)
,
b)
,
Shadow turtles alongside the main turtle
c)
Imagine - C:\Program Files\Imagine Logo\project\animate-tree30.IMP
File Edit View Options Page Help
Background colour is yellow
Run
Clear
Colour
Main Page
Red tree
size
right angle
left angle

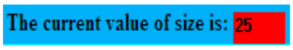
```
To tree :size :right angle :left angle

if :size < 15 [stop]

forward :size
right turn :right angle

Tree :size / 2 :right angle :left angle

left turn :right angle
left turn :left angle

Tree :size / 2 :right angle :left angle

right turn :left angle
back :size

End
```

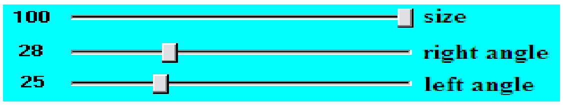
The current value of size is: 3.125

100 size
28 right angle
36 left angle

Figure 53- The shadow turtles alongside the main turtle, (a) when the main turtle started to draw new red branches to the left (red colour), (b) when the main turtle started to draw new branches to the right (yellow colour), (c) the main interface of the *blue* strategy when the turtle was trying to draw some yellow branches. The background colour of the box of the size is yellow

The shadow turtles indicate that the main turtle is about to draw a new tree in either a left or right direction. The directions to the left or right correlate with the recursive calls on the codes of the procedure. The pattern colours of the shadow turtles were also chosen in accordance with the colour codes of those recursive calls in the procedure. The shadow turtle to the left shows a red, lopsided tree to the left, which coincides with the colour code of the first recursive call. The one to the right shows the same thing to the left, in the colour yellow, in accordance with the second recursive call. The shadow turtles were designed in such a way that as soon as the program reaches one of the recursive calls, they appear on the screen alongside the main turtle. The students are thus enticed to launch a new tree which is a whole copy of the procedure but with a smaller initial value for the size, and in a different direction. It was conjectured that they would provide a global picture of the tree and the process of branching to the right and left each time, calling these recursive calls.

Another design characteristic of the *blue* strategy module was contriving a box on the screen to show the length of the branch which is being drawn by the turtle on the screen (Figure 53, (c)). As shown in the picture, the background of this box switches between the colour red and yellow, in accordance with the colour code of the branches. The background colour is red 
The current value of size is: 25
when the main turtle draws a red branch, and is yellow
The current value of size is: 12.5
when the main turtle draws a yellow branch.

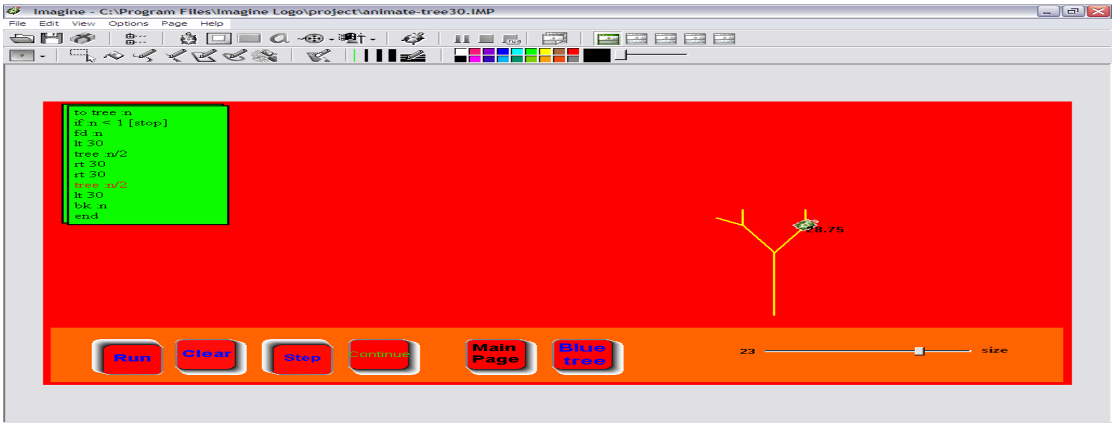
In this module, I contrived three sliders 
100 size
28 right angle
25 left angle
so that the students were able to control the size of the angles to the right and left. To run the procedure, the students were given the opportunity to choose between two modes: colour mode and normal. They could switch the modes by clicking on the buttons labelled *run*

and *colour*

. By clicking on the *run* button, the final output of the procedure, the binary tree, was drawn on the screen without any animation. This mode was designed to provide the students with possible vantage points to bridge their initial embryonic opinion about the tree in the previous module (the *making a forest*) and the written codes behind its exterior output. However, as mentioned above, the *colour* mode was designed based on employing the AVDA approach and colour codes to draw the final output. The students were also able to go to the main page of the *Treebuilder* domain of abstraction or to the next module (the *red* strategy) by clicking on the buttons labelled *main page* Main Page and *red tree* Red tree, respectively.

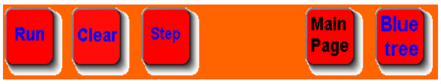
The above-mentioned design features in these modules were specifically designed to facilitate students' being able to work out the control passing mechanism in the embedded recursive procedures. In Chapter Two, reference is made to Kurland and Pea (1985), who distinguish between the iterative control passing from recursive flow of control by introducing *active* and *passive* flow of control. I noticed that, using the term *passive* to describe the complicated mechanism of the control passing in a recursive procedure is not informative enough for that complicated mechanism. In order to avoid the verbal impression of the term *passive*, as well as giving a more descriptive terminology, I decided to call it the '*delegatory*' control passing mechanism instead. Having said that, to provide an efficient window through which students can look into the concept of recursion and its components, I designed another module for the *Treebuilder* domain of abstraction, called the *red* strategy. The next section of this chapter focuses on the design features of the *red* strategy module.

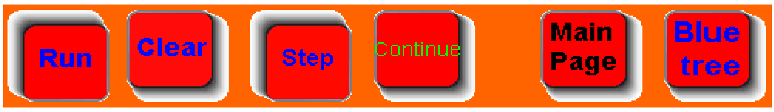
6.3.4. *Red* strategy – *Treebuilder* domain of abstraction


Imagine - C:\Program Files\Imagine Logo\project\animate-tree30.IMP
File Edit View Options Page Help
28.75
Run
Clear
Step
Continue
Main Page
Blue tree
23
size

```
to tree :n
if :n < 1 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

Figure 54- The interface of the *red* strategy module

The main interface of this module is shown in the above picture. In similar fashion to the *blue* strategy, I used AVDA innovation in this module. The *blue* strategy (the previous module) was an output based module in which the focus was on employing the animative techniques of AVDA on the tree which was being drawn by the main turtle and was accompanied by two shadow turtles to show the appropriate colour codes. However, in this module, I principally focused on making use of the AVDA innovation in the original program and using an animative approach to represent the generation of the new copies of the original procedure after each time calling one of the recursive calls. As shown in Table 2, in this module students were given a control box 

, to run the procedure. Also, similar to the previous module, the students had a choice between two modes of execution: step-wise or normal. When they chose to run the procedure in the step-wise mode, a new button labelled *continue* appeared 

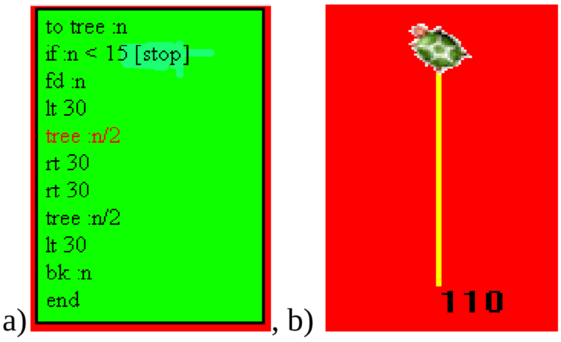
. So, the

students could see each step, each time calling one of the recursive calls, by pressing the *step* button

. The other design feature that was incorporated in this module was showing the length of the current stem which was being drawn by the turtle on the screen alongside the turtle. The students were able to choose the initial length of the main trunk using the slider

, which was contrived on the control box.

As well as using the AVDA visualisation, I also used colour codes for the lines and commands of the procedure. The colour of each command would change to red as it was being executed. It was conjectured that this would facilitate the students' appreciation of the flow of control, as well as providing them with a better means of tracking the delegatory flow. To be more precise, in this way, a numerical label moved alongside the main turtle, to show the current size of the stem which was being drawn. This mechanism provided the students with the precise backwards movement of the last small stems. The following pictures show the connection between the numerical label –showing the length of the stem – and the colour codes over the procedure.

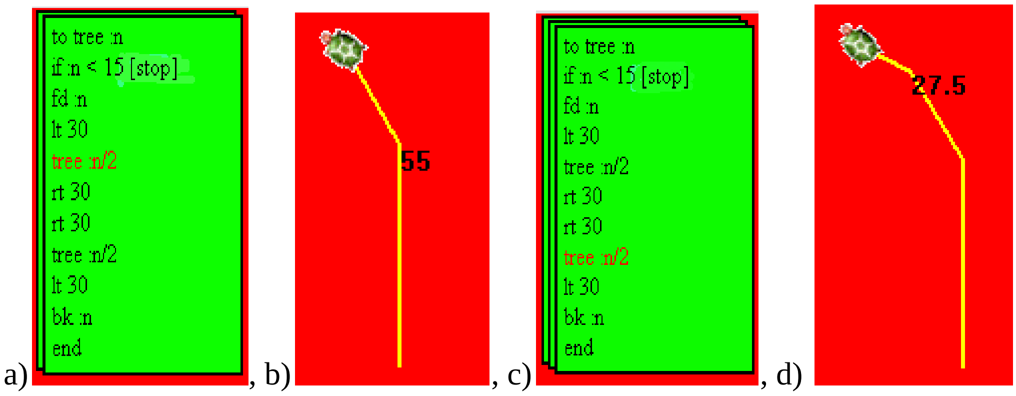

a)

```
to tree :n
if :n < 15 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

, b)

Figure 55- Running the procedure with 110 as an initial value, and the procedure is waiting for the student to press the *continue* button to run the first recursive call

From the pictures above, it can be seen that the first recursive call is shown in red, which means that this is the line which is going to be executed by pressing the *continue* button. When the student clicks the *continue* button, the following sequential results appear on the screen.

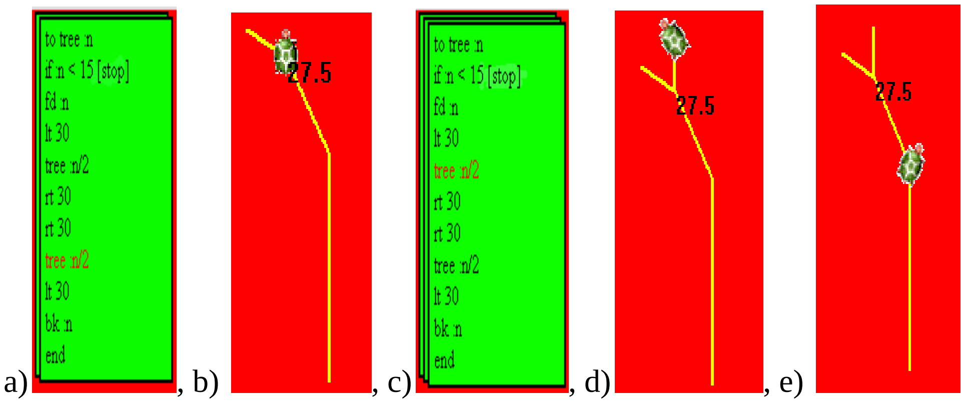


```
to tree :n
if :n < 15 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

```
to tree :n
if :n < 15 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

Figure 56- A new copy of the original procedure is generated and the turtle has drawn the new stems after the *continue* button has been pressed twice

Figure 56(a) shows that the second copy of the original procedure is waiting to run the first recursive call. Figure 56(b) shows the output which is drawn by the turtle. Figure 56(c) shows that the third copy of the original procedure is waiting to run the second recursive call as the turtle reaches the stopping condition and starts to draw the branches into the right side. Figure 56(d) shows that the turtle is heading 30 degrees to the right, which means that it has done two left turns 30, whereas in figure 56(b) the turtle is still heading 30 degrees to the left. The next pictures show the process of calling the first and second recursive calls, the direction of the turtle, and the way that turtle draws the branches.



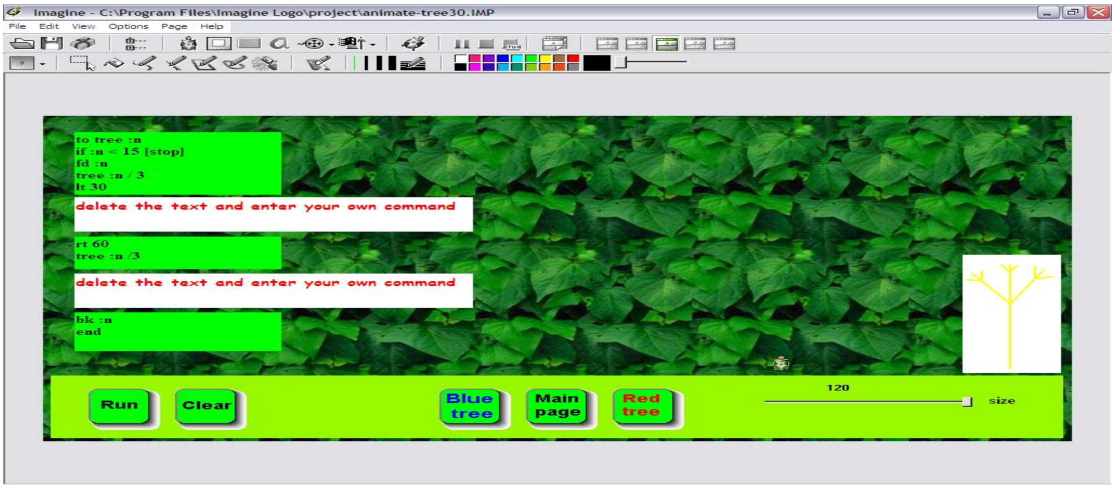
```
to tree :n
if :n < 15 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

```
to tree :n
if :n < 15 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

Figure 57- The AVDA approach to generating the new copies of the original procedure and the way that the turtle draws the branches

The design issues of the *your tree* module of the *Treebuilder* domain of abstraction are considered in the proceeding section.

6.3.5. *Your tree – Treebuilder* domain of abstraction



```
to tree :n
if :n < 15 [stop]
fd :n
tree :n / 3
lt 30
```

delete the text and enter your own command

```
rt 60
tree :n /3
```

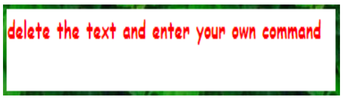
delete the text and enter your own command

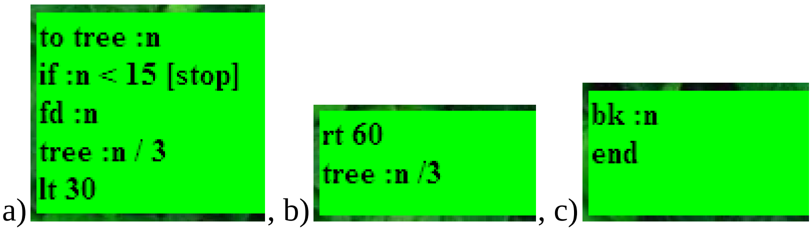
```
bk :n
end
```

Figure 58- The interface of the *your tree* module

The picture above shows the main interface of the *your tree* module. This module is a developed version of the *blank-box* task in the second iteration. Similar to the *blank-box* task, the *your tree* module design was based on completion of an incomplete given embedded recursive procedure to generate a ternary[11] tree.

The students were expected to fill two empty boxes with appropriate commands to complete the procedure. The students were also given some part of the procedure in three groups of commands in three green boxes, which are shown in the pictures below.


delete the text and enter your own command


a)

```
to tree :n
if :n < 15 [stop]
fd :n
tree :n / 3
lt 30
```

, b)

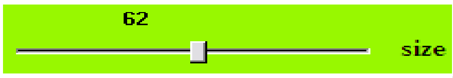
```
rt 60
tree :n /3
```

, c)

```
bk :n
end
```

Figure 59- The commands of the incomplete procedure which were given to the students

The first empty box, as seen in picture 16, was located between Figures 59(a) and 59(b), and the second empty box was located between Figures 59(b) and 59(c). Similar to the other modules of the *Treebuilder* domain, I contrived a control box in this module for students. As shown in the fifth row of Table Two, the control box contains two buttons labelled *run* and *clear* to run the procedure and clear the screen. There are also some buttons to move on to the main page or the other modules. And also, a slider, to set the initial size of the first step in the procedure with a numerical label showing the value of the initial size.



[11] A tree which has three new stems at each branching point.

From a technical point of view, the module was designed in such a way that any Logo codes could be accepted as a missing part of the incomplete procedure. The final output of the procedure was deliberately chosen to be a ternary tree. This planned tactic was aimed at evaluating the students' appreciation of the functioning of the recursive calls as one of the crucial components of the concept of recursion. I wanted to see whether the students were aware of the functioning of the recursive call as triggering a new bunch of branches or a new tree in a smaller scale. To complete the task, the students needed to have an adequate understanding of the state of the delegatory control passing in embedded recursive procedures. Therefore, the results of this task were of importance in testing and evaluating the students' appreciation of the delegatory control passing and functioning of recursive calls. Consequently, this module may be considered as a means of providing a bridge between the functioning and functionality of the indispensable components of the concept of recursion. Thus, this module played a significant role in terms of the concept of functional abstraction.

First and foremost, the *your tree* module was designed to provide an appropriate environment for the students to apply the knowledge and understanding that they would theoretically gain after working with the previous modules of the *Treebuilder* domain. This module opened a window for me as researcher to investigate in close-up the students' thinking and thinking-in-change process regarding embedded recursion. Their responses to this module demonstrated their thought process towards the functioning aspect of the recursive calls, and delegatory control passing. Therefore, I was able to evaluate the influence and efficacy of the AVDA in order to see how and to what extent the students developed and constructed their mental models of the concept of recursion. Additionally, I was able to see to what extent the students could apply their knowledge in problem-solving situations.

Technically, the students were expected to place an additional recursive call to generate the middle branch of the desired ternary tree, as well as inputting *right turn 30* to show their understanding of the delegatory flow throughout the procedure. In the *red* and *blue* strategies, they experienced the binary trees. Based on these former experiments, they were given two recursive calls in the given codes. From a functioning standpoint, the task was designed to test how they used and applied the third recursive call for the third middle branch. Of further importance, was the location of this additional recursive call, as it was conjectured that it would reveal the students' level of appreciation of the delegatory control passing and the functionality dimension of the recursive calls.

6.4. Functioning features of the tool design

It was mentioned earlier that 'distinguishing between functionality and functioning in the concept of recursion' has almost been overlooked in published literature on the matter. One of the major aims of this study is to focus on the functioning features of the concept of recursion and its components throughout the design of purposeful computer-based tools. The functioning aspect, which is mainly related to the *how* part of the mechanism of recursion, has been considered in both the *Spirals* and *Treebuilder* domains of abstraction. The focus of this section is to explain the functioning features of the tools and modules in the third iteration's modules.

6.4.1. Functioning aspects in the *Spirals* domain

The functioning aspects of the design in the *Spirals* domain, was created into the *comparison* module. In this module, the students were able to see the hidden parts of the mechanism of the control passing in the tail recursive procedures by animation and colour coding techniques. The animative visualisation contrived into this module shows new copies of the original procedure which are generated after each successive calling of the recursive call. It was conjectured that, by colour coding the lines that were actively being executed, students would track the flow of the procedure efficiently.

6.4.2. Functioning aspects in the *Treebuilder* domain

The *making a forest* module provided a situation in which the students could express their thinking about the relationships between typing the term *tree* followed by a number and the final output (a tree on the screen). It was conjectured that they would have the opportunity to think about the structure and the way in which the tree would be drawn by the computer. In this way, they were able to bridge *what* they saw on the screen and *how* it was created by the computer.

The functioning aspect of the design of the *blue strategy* was based on revealing *how* the tree was being drawn by the procedure. Focus on the functioning aspect was achieved through the creation of two additional turtles (called *shadow turtles* see Figure 53 (a & b)). These shadow turtles move alongside the main turtle. It was conjectured that these shadow turtles would provide the situation in which the students could see that, after reaching each one of the recursive calls, the turtle was going to draw a new tree, on a smaller scale, to the right or left direction in accord with calling the first or second recursive calls. As mentioned before, the red and yellow colour codes were also used in this module for the branches to the right and left.

Finally, the functioning aspect of the design in the *red strategy* module of the *Treebuilder* domain is exactly like that of the *comparison* module in the *Spirals* domain. It is based on representing the *how* part by using animation over the new copies of the original procedures. From a functioning perspective, the shadow turtles act like those new copies of the original procedure in that they show a new tree is going to be drawn, but with a slightly different length.

6.5. Summary

This chapter focuses on the approach that was chosen to implement the third iteration. The complex control passing process in the tail, embedded, and the control passing between the different recursive calls within an embedded recursive call is depicted in the three flowcharts in this chapter. The chapter ends by describing the design aspects of the modules of *Treebuilder*.

The next chapter of this thesis focuses on the tool use aspects of the third iteration, which mainly concentrates on explaining and discussing the students' accounts.

7. Iteration Three – Tool Use of the *Treebuilder*

This section concentrates on how students would quite literally use the tools within each of the seven modules of the *Spirals* and *Treebuilder* domains. The following diagram outlines the order of the modules which were previously explained in Table 9.

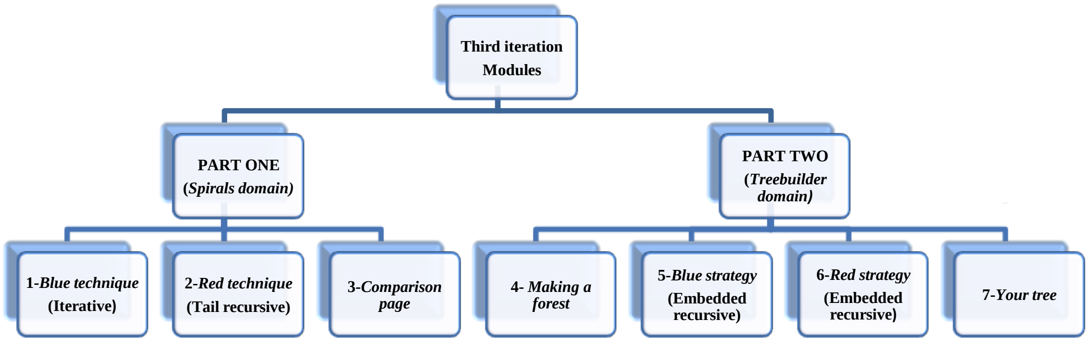

Third iteration Modules
PART ONE (*Spirals domain)*
PART TWO (*Treebuilder domain)*
1-*Blue technique* (Iterative)
2-*Red technique* (Tail recursive)
3-*Comparison page*
4- *Making a forest*
5-*Blue strategy* (Embedded recursive)
6-*Red strategy* (Embedded recursive)
7-*Your tree*

Figure 60- The order of the modules in the third iteration

As shown in the picture above, the students were asked to start the iteration by working with the modules of the *Spirals* domain of abstraction. After completing of those modules, they were invited to work with the modules of the *Treebuilder* domain. Seventeen volunteer students participated in this iteration. Each interview lasted 1.5 hours. Each of the interviews, were recorded using Camtasia screen recorder software. The interviews were fully transcribed and coded. The coded data was analysed and used for extracting the final result of this iteration. In this section of this Chapter three students' accounts are thoroughly examined; those of an individual, and two pairs. These accounts are examined in two major parts as shown in Figure 60, and in accordance with the three tasks of the *Spirals* domain and four modules of the *Treebuilder* domain of abstraction.

The accounts of the students, discussed in this chapter, were chosen in particular because they can be considered as representative of the students who took part in the third iteration collectively. Their responses clearly reflect the rest of students' approaches to these tasks and modules. I have endeavoured to present these three accounts in as much detail as possible in order for you, the reader to see what progress the students made while they were engaged with those domains. The chapter finishes with the findings section for the third iteration and a summary.

7.1. PART ONE – *Spirals* Domain

This section concentrates on the accounts of the five students, Simon, Peter & George, and Andrew & Hayley, who were all studying on a four year degree program, and were training to be primary school teachers. The students' accounts are explicated into two parts. The first part focuses on the students' explanation of the *Spirals* domain tasks and the second part is about their account of the *Treebuilder* domain modules.

7.1.1. Simon's account on the *Spirals* domain – Iteration Three

The first student's account examined in this section is that of Simon. He participated in the interview individually. He started with the *blue* technique and decided to run it in the *step* mode. He checked a few more values to generate spirals of different sizes. Then he moved to the *red* technique and continued to do the same thing that he did in the *blue* technique. He checked a few different values for the ‘n’ and observed the results carefully.

1. Simon: For ‘n’ equals 1 it’s going to be absolutely tiny, isn’t it?
2. My description: [*to be able to see the little spiral under the turtle he hid the turtle and added that*]
3. Simon: Ah, it’s just a point, obviously, yeah! Well it wouldn’t be a point but it would be very tiny!
4. I asked: Can you explain to me what is going to happen at each step?
5. Simon: Yes, I think so. ‘n’ is 150, then to blue 150, well, while 150 is greater than 1, you go forward 150, left 30, make 150, a hundred and fifty divided by 1.1. So, it gets smaller by the ratio of 1.1 each time. So, oh, I see, it goes left 30, smaller by 1.1, left 30, smaller by 1.1, left 30, divided by 1.1, left 30, divided by 1.1, and so on, all the way around to there.
6. My description: [*he was pointing to the end of the spiral on the final output on the screen and added that*]
7. Simon: Ok, that’s good.

I was not sure about what he thought about the stopping condition, so I asked him:

8. I asked: When it is going to stop?
9. Simon: Let’s get rid of the turtle. Well, it won’t ever stop completely. It’s always going to be slightly greater that one and so it’ll actually carry on going and carry on going for ever and ever! Until it converge at a point with no change! Umm, why is that then? It should stop.

Therefore, although in line 5 he stated that ‘*while 150 is greater than 1, you go forward 150*’ also in the same line he added that ‘*it gets smaller by the ration of 1.1*’ but he still showed some difficulties in making a link between what he saw and how it was going to be done. We moved to the recursive procedure (the *red* technique). Simon again decided to run the procedure in the step mode and after observing the execution of the procedure in the step mode he ran it again in the normal mode.

> 10. Simon: So, ‘n’ is less than 1. So, let’s put it back on 50 to see if there is any difference. Let’s change 50 to 100, I should get a bigger one, that’s what I am expecting. And at 150 we should get a bigger one again. Yeah! Perfect.

Simon’s explanation in the above quote shows that he had no conflict in his belief and thinking about the function and functioning of ‘n’ as the initial value of the spiral and its role in having a big or a small spiral. Simon’s first challenge with the *red* technique appeared when he wanted to explain the stopping condition of the recursive procedure.

> 11. Simon: To red 150, 150 is less than one stop! Is it going to stop and then go forward 150, or is that stop completely?
> 12. I said: what do you think?
> 13. Simon: So, can we put on a value of ‘n’ less than 1? Well, we can’t because the minimum value of ‘n’ here on the slider is 1. Oh, it says if ‘n’ is less than one stops if not forward 150 Ok. Let’s take ‘n’ equals two. Ah stopped! Ok, because ‘n’ is 2 and 2 divided by 1.1 is less than 1. So, let’s put it a bit bigger ‘n’ equals 18. That, will also stops won’t it?
> 14. I said; I don’t know! Why not check it in the step mode?
> 15. Simon: Ok, switch, and then start and then step!
> 16. My description: [*Simon kept clicking on the step button until the procedure reached its base case – the stopping condition. Then he added that*]
> 17. Simon: And then it stops! So, the *blue* technique never stops.

His above remark was based on his previous experience with the *blue* technique when he described the *blue* technique in terms of “*it won’t ever stop completely!*” (Simon, line 9). But in the *red* technique (the recursive one), by taking some different values for the initial value ‘n’ and the size reducing factor of 1.1, he concluded that it stops. Although the result that he achieved was not a correct model, it shows that the computer-based environment provided him with a window in which he was able to investigate and examine a more concealed layer of the recursive procedures.

I wanted to explore the way he thought about the stopping condition in those techniques, so I asked Simon, why he thought the *blue* won't stop? He moved back to the *blue* technique and waveringly stated that:

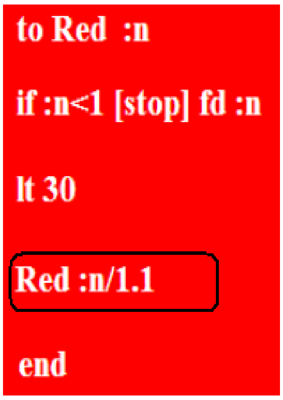
18. Simon: Oh! It does stop, doesn't it?
19. I said: I don't know I just asked you to tell me about it!
20. My description: [*He changed the initial value of 'n' equals 18 as it was in the red, and ran the blue procedure in the step mode and started to count the steps while he was clicking on the step button*]
21. Simon: 1, 2, 3, 4…, 31 steps and stop, and in the *red* technique, 1, 2, 3, 4 …, 31 again, the same. Then I was wrong! So I was wrong. Ok, so, what is the difference?
22. I said: What do you think?
23. Simon: I can't see if there is one.
24. My description: [*He changed the value of 'n' in the red and blue technique to 50 and ran them in the step mode and counted the steps again*]
25. Simon: I have to say I can't. I'm struggling to tell the difference.

In the computer-based environment he could see that the procedures would not work forever. However, he was not able to recognise the difference between the iterative and recursive procedures yet. To ensure that he had given enough attention to the procedures as well as the final output, the image of the spiral that was being drawn by the turtle, I asked him about the written procedures. He moved back on to the *blue* technique and stated:

26. Simon: Ok, all the time, the step is greater than 1, forward 'n', left thirty, and then *make* 'n', 'n' is divided by 1.1, so presumably with 'n' over 1.1 it goes back in to this equation.
27. My description: [*He was pointing to the command 'forward 'n'' and added that*]
28. Simon: And it carries on and carried on and carries on. Then on the *red* technique, if 'n' is less than 1 stop, fine. And then if not,

you are going to go forward ‘n’, left thirty, and this is a bit I do not understand now!

29. My description: [*He was pointing to the recursive call (see Figure 61) and added that*]

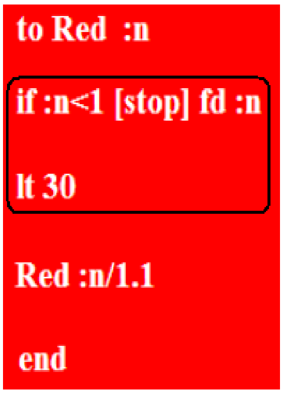


```
to Red  :n
if :n<1 [stop] fd :n
lt 30
Red :n/1.1
end
```

Figure 61- Simon was struggling to describe the recursive call

The following lines of the transcript show the struggle that Simon was beginning to have with the functionality of the recursive call.

30. Simon: Red ‘n’ divided by 1.1 and then finished? It doesn’t go around in a loop.
31. I asked: Is that what you think about it?
32. Simon: Well, I don’t know what that red is? To red, Oh! Presumably it does mean doing this! [*see Figure 62*] Because the procedure *red* is there.

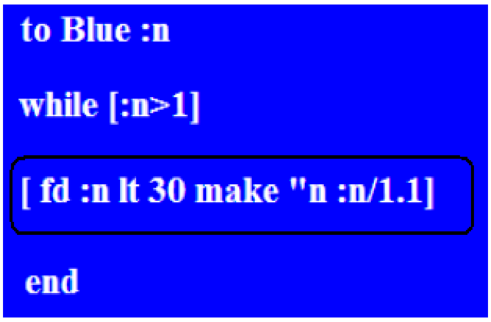


```
to Red  :n
if :n<1 [stop] fd :n
lt 30
Red :n/1.1
end
```

Figure 62- Simon was pointing to those commands that the recursive call was calling

33. Simon: So, therefore that is the new procedure *red* using n over one point one. The only difference between the *red* technique and the *blue* technique is if ‘n’ is less than 1 stopped. That seems to be the only difference.

34. My description: [*he moved back to the blue technique and added that*]
35. Simon: Although, actually while ‘n’ is greater than 1 do that (see Figure 63).



```
to Blue :n
while [:n>1]
[ fd :n lt 30 make "n :n/1.1]
end
```

Figure 63-Simon was pointing to those commands that while ‘n’ was greater than one, were being executed by the procedure

36. Simon: It means there are no instruction commands in there [*Figure 63*] to tell you what happens if ‘n; is less than 1. So, presumably, this means it has to stop! So, actually they are the same!

Simon’s remark in lines 26-36 show his difficulty in recognising the flow of control in the recursive procedure and the functioning of the recursive call. His explanation about the recursive call occurred in line 28 when he stated that, ‘*… this is a bit I do not understand now!*’ then in line 32 when he added that, ‘*I do not know what that red is*’.

According to Kahney’s explanation, at that stage Simon showed possession of a *syntax* model of a recursive procedure. However, lines 33-36 show strong evidence that Simon is in possession of a *return-value* model of recursion. In line 33, he directly pointed out that the recursive call is a “*new procedure using ‘n’ over 1.1*”, then he continued by saying that the only difference in those techniques is the syntactical difference in the stopping conditions. His response showed that at this stage his understanding of the functioning of the recursive call was only as a generator for the new values of ‘n’ over (1.1). He did not realise that this was needed as the initial value of the new copy of the original procedure. For him the functionality of the recursive call was generating a new value for ‘n’, rather than a new copy of the original procedure.

Therefore, before going to the *comparison* module and AVDA innovation, Simon’s mental model of recursion evolved from a loop model in line 23 when he stated “*I can’t see any difference*”, to a combination of syntactical and return-value models. The following diagram shows Simon’s mental model evolution through working with the *red* and *blue* techniques.

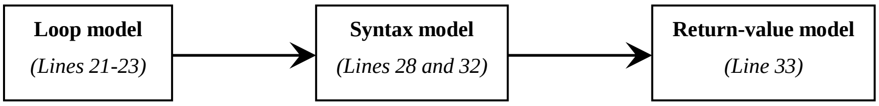
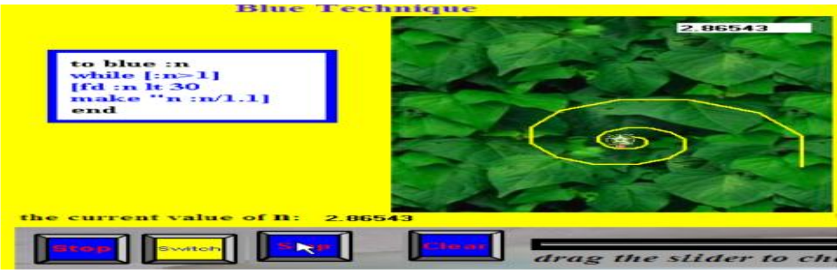


Figure 64- Simon’s evolution of tail recursion mental model after working with the *red* and *blue* techniques (before his experience with the AVDA innovation)

Then we moved into the *comparison* module. Simon began to work with the *comparison* module by running the *blue* technique in the step mode. Then he set the initial value for the ‘n’ equals to 50 and ran the *blue* technique.

37. Simon: So, let’s put ‘n’ 50, which was what we have been using. Ok, let’s start, we have got ‘n’ is greater than 1, so the current value of ‘n’ becomes ‘n’ divided by 1.1 and probably divided by 1.1 again, I’m guessing. And then step, it goes down 31, 28, 25 ok, 21, 19 , 17, … so it goes all the way around until it gets to the point that ‘n’ divided by 1.1 is less than one, and then there is no instructions saying what to do and then it stops there (Figure 65).



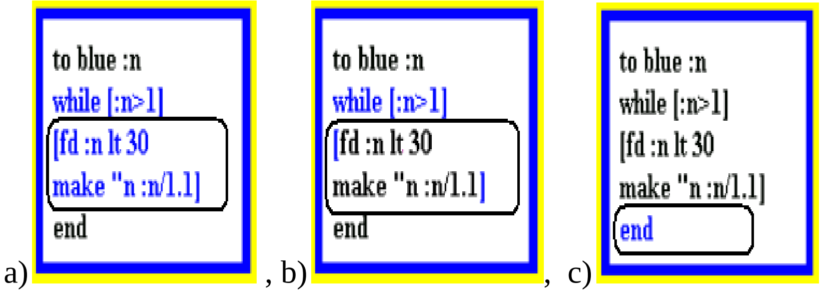
```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

the current value of n: 2.86543

Figure 65- The interface of the *blue* technique in the comparison module, when Simon was running it in the step mode

To ensure that he had paid attention to the written procedure and the flashing blue colour codes for the lines of the procedure I asked his opinion about the written procedure on the left side of the above picture. He ran the *blue* procedure again and tried to describe it for me based on the written commands as follows.

> 38. Simon: Starts, we have got while ‘n’ is greater than 1, the procedure is flashing the next step! ‘n’ is still greater than 1, and do that (Figure 66-a below), and is still greater than 1, and do that, and still do that, and keeps going on and so on. It is still saying that forward ‘n’, left thirty, when you press the button it gets to the end (Figures 66-b) because it has not got instructions when ‘n’ is less than 1.



```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

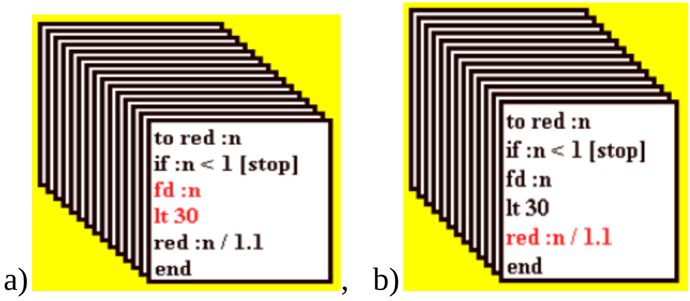
```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

Figure 66- Simon was pointing to the colour codes of the commands of the *blue* technique, which were being executed by the procedure in the *comparison* module

The colour codes which were employed in the *blue* techniques are shown in the figures above. Figures 66(a), and 66(b) above show how the commands between the brackets were flashing blue while ‘n’ was greater than one in the *blue* technique. Figure 66(c) shows that the command *end* becomes blue when ‘n’ gets less than one.

Then Simon moved on to run the *red* technique in the *comparison* module. He ran the *red* technique in the normal mode. When the new generated copies of the original procedure started to move back in the AVDA environment, he responded by saying that:

39. Simon: Presumably, ‘n’ is going to get to less than 1 and stop! It doesn’t! Ah, interesting, interesting! Ok, then on the face of it they look the same but actually they are not! Let’s find out why that is?



a)

```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

b)

```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

Figure 67- The colour codes and animation of the commands of the *red* technique in the AVDA environment of the *comparison* module

40. My description: [*the animation in the red technique was quite unexpected when Simon ran the red technique in the step mode and continued that*]
41. Simon: What is going on? So, Ok, if ‘n’ is less than 1 stops, ‘n’ is 50, so that’s fine, ‘n’ is not less than 1! Ah, right now it’s stepping through the procedure. So, the first time you press *step* you get ‘n’ is less than 1 stop, it forwards ‘n’, left 30. Should work now, yeah it does. And then it makes ‘n’, ’n’ divided by 1.1 so then nothing happens there. What is it doing now?
42. My description: [*at this stage, a new copy of the original procedure was being generated and the procedure was waiting for him to press the step button*].
43. Simon: Oh, right, it starts again. Left 30, divided ‘n’ by 1.1, and then goes back to top, alright, that’s right. It does actually cycle around the procedure, which we didn’t have to do in the *blue* technique. It is a much more rigorous procedure. Ok, fine it cycles all around the procedure. Let’s see what will happen when ‘n’ gets down to 1.

44. My description: [*he was keep clicking on the step button to see what happened when 'n' got to less than 1*]
45. Simon: 'n' is less than 1 stop, still is more than 1, carries on, forward 'n', left 30, it does that then *red* 'n' divided by 1.1, now 'n' becomes less than 1, to *red* 'n', if 'n' is less than 1. To *red* 'n', if 'n' is less than 1 stop, so it stops. And then what has it done? Divided 'n' by 1.1 again! Hasn't it? What has it done? If 'n' is less than 1 stop, so it should stop, why is it carrying on?

Simon's remarks are, in the first place, evidence that the AVDA environment allows him to evolve his understanding of the differences between the iterative and tail recursive techniques. In line 39, immediately after seeing the animation in the *red* technique, he stated that, "[…] *on the face of it they look the same but actually they are not!*" which shows that he was thinking that these techniques are the same until seeing the way that they produce the spiral is different. In other words, the AVDA environment provided him with the opportunity to become aware of the functioning aspect of the recursive call and its difference with the '*make*' command in the *blue* technique (line 43). Before beginning to work in the AVDA environment, Simon thought that the *blue* and *red* techniques were the same. Through working in the AVDA environment, he gradually evolved his understanding of the concept and his mental model. Simon's mental model of the concept of recursion before his experience with the AVDA environment - as is shown in Figure 64 - evolved from a loop model to a *syntax* model and then to a *return-value* model.

Simon's remarks while he was working with and experiencing the AVDA environment and the animation and colour codes which were contrived in the comparison module evidenced his possession of some new models of tail recursion. Therefore, the diagram of his mental model's evolution of the concept of tail recursion can be amended as follows: From possession of a *loop* model, to a *syntax* model, and then to a *return-value* model / *step* model, and gradually towards the possession of an incomplete version of the viable copies mental model of recursion which, I called a *quasi-copies* model of the concept of recursion.

A possessor of a *quasi-copies* model of the concept of recursion has knowledge about the generation of the new copies of the procedure after each time of calling the procedure. In addition, he/she knows at each calling of the recursive calls, a new initial value is going to be generated, which is slightly different from the original initial value. The only difference of the *quasi-copies* model compared to the viable *copies* model of the concept of recursion is that the possessor of such a model has no notion of the returning flow of control for the termination of all the generated copies of the original procedure. More evidence of possession of such a model by Simon is shown below.

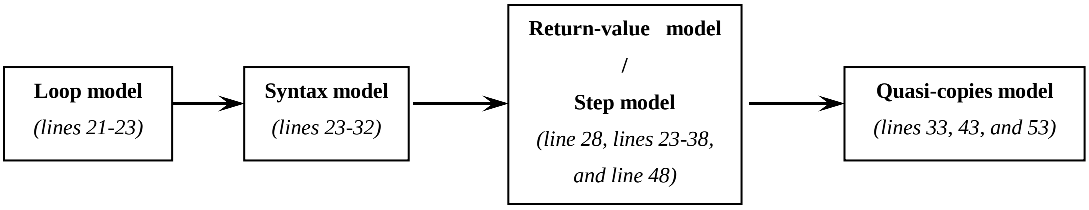


Figure 68- Simon's evolution of the mental models of the concept of tail recursion after working in the AVDA environment

46. I asked: Can you explain the similarities and differences between these two techniques?
47. Simon: Well, they are producing the same things. They have the same aim, they produce the same shape, the idea of both is obviously to create a spiral.

In the above line, Simon mainly focuses on the functionality aspect of the techniques and the '*what*' part, which is more stemmed in the final output of the procedures. Then Simon carried on with more in-depth explanations.

48. Simon: The difference is the *blue* technique is a lot easier, more straightforward, a lot better. The *red* technique is a bit of a struggle because it has to go in a step by step approach and you have to do it normally and then at the end remove all the steps again and it takes you back up to the original value of 'n' which is 50 and it is going to be the same for any other value. So, I would say the *blue* is a lot easier, a lot better if you want to create a spiral. The *red* technique is rather more complicated.
49. I asked: Which parts of those techniques were most difficult to understand?
50. Simon: The *blue* technique is quite straightforward, but it does not make explicit what to do if 'n' is less than 1. So, I just had to make an assumption. But then actually in this written part of the *blue* technique it says *end*. So, that is basically saying what to do if 'n' is less than 1. So, I think it is a very straightforward technique. But, the *red* technique is quite complicated, quite difficult to get your head around.
51. I asked: Why do you think so?
52. My description: [*in response to my question he thoughtfully paused for a while and said:*]
53. Simon: Um … I don't know! It makes a lot less sense! If 'n' is less than 1 stop, forwards 'n' and left 30, and red. I think because you have to remember, I believe you have to remember, you have to keep substituting in 'n' over 1.1 each time for the 'n' you have got in the equations, or in the instructions. And just simply to look at it without writing it all down, writing it as a next step, actually writing it as a new equation, is quite challenging.
54. Simon: I think if you were actually to sit down and write out what the program is doing each time… so, if you write down the next time it does 'n' over 1.1 and then it turns left 30. And then next time it does 'n' over 1.1 squared, left 30, etc. then it is quite easy to understand. But, the fact is you've simply got to remember each time you are substituting a different value. This is tricky a bit.

Simon's above remark in line 53 evidenced his possession of the quasi-copies model of the tail recursive procedures. It is also shows that one of the main reasons that Simon was not able to make a connection with the process of flow in the recursive procedures is the lack of everyday analogies, “[…] *it makes a lot less sense*!” (Simon, the Spiral domain, line 53).

In that line, he also mentioned that “ […] *I believe you have to remember, you have to keep substituting in ‘n’ over 1.1 each time for the ‘n’ you have got in the equations*!”. This shows Simon’s imperfect perception of the flow of control in the recursive procedures. These, and Simon’s explanations in line 33 and 53 about ‘*… new procedure ...*’ (line 33) and ‘*… writing it as a next step, actually writing it as a new equations! ...*’ (line 53), provide evidence of possession of a *step* model and *quasi-copies* model.

Simon’s explanations in line 48 about the *blue* and *red* techniques support the results of the second iteration on the Spiral domain of abstraction. The results of the second iteration reveal that, to create a spiral, students prefer to work with an iterative procedure because it is much easier and more straightforward rather than a complicated and time-consuming recursive algorithm.

7.1.2. George & Peter’s account on the *Spiral* domain – Iteration Three

As mentioned earlier, the second account is about a pair of mathematics specialist students on a four year degree program, George and Peter. After a brief introduction to the software, tasks and modules they started their work with the *blue* technique. The first thing that they did was to try a few different initial values for ‘n’ and observe the output which was a spiral drawn by the turtle on the screen. They set ‘n’ equals 99 as the initial value and decided to run the *blue* procedure in the *step* mode.

55. George: Let’s switch it to the *step* mode. Oh, Ok, so this is the value of ‘n’, we go forward ‘n’.
56. My description: [*Peter interjected and by pointing to the number which was shown by the slider on the screen added that*]
57. Peter: And this is what we say ‘n’ and then turn 30,
58. George [*continued*]: Left turn 30, and then change ‘n’ to ‘n’ over 1.1. Ok, so it gets smaller and smaller.

In line 58, by using the term ‘*smaller and smaller*’, George shows he has considered the mechanism of flow as a repetitive mechanism which is changing the value of ‘n’ by ‘n’ over 1.1 each time.

59. I asked: Can you see any connections between the written commands on the screen and those spirals?
60. George: If ‘n’ is less than 1 stop.
61. Peter [*interjected*]: Forward ‘n’ left 30, Ok, it stepped with ‘n’ over 1.1.
62. George [*continued*]: Yeah, that’s right.

Then they moved to the *red* technique and tried to run it in the *step* mode. George tried a few different initial values for the ‘n’ and said:

63. George: It is the same as the *blue* one!
64. Peter [*agreed with George and said*]: Yes, that’s right!
65. I asked: Can you explain a bit more about your opinion and explain why you think they are the same for me?
66. George [*immediate response*]: They produce the same thing! The same spirals with both the *blue* and *red* techniques.
67. I asked: What do you think Peter?
68. Peter: Um, yes that’s right they are producing the same thing! Shall we go to the comparison module?

George’s remarks on lines 63-66 and Peter’s comment on line 68 show that before experiencing the AVDA innovation in the *comparison* module they were not able to see any differences between those two techniques and their attention was mainly focused on the final output of the procedures. At this stage they showed evidence of possessing an understanding of a *loop* model of recursion.

From the functioning aspect, line 61 can be considered to be evidence of possession of a *syntax* model by George and Peter. In that line, George directly pointed to the syntax of the recursive call. This line can also be considered as a signal of possession of a *step* model because George pointed out that "[…] *it stepped with 'n' over 1.1*" (line 61). However, from the functionality aspect, they still thought that they were the same: "[t]*hey produce the same thing*" (Line 66).

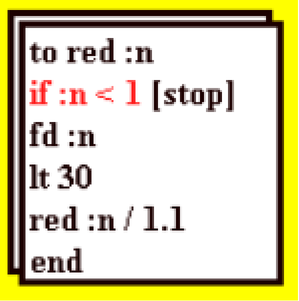
In the *comparison* module they had the opportunity of comparing the *red* and the *blue* techniques in one plus the animative visualisation which was contrived into the *comparison* module. In the comparison, George & Peter ran both procedures in the *step* mode. When they saw the animative visualisation of the new generated copies of the original procedure in the *red* technique (the AVDA environment) George said:

69. George: Oh! They are going back!
70. Peter [*interjected*]: They carried on longer,
71. My description: [*George continued by pointing to the blue technique and then pointing to the red technique and said*]
72. George [*continued*]: Ok, so, this is while 'n' is greater than 1 and this one if 'n' is less than 1, stop! But, it did not stop! Did it stop or didn't it? I thought it was going to stop when 'n' is less than one.

Lines 60 to 72 showed that they were a bit confused by the complicated mechanism of the delegatory flow of control in the recursive procedure. In line

72, George showed his first sign of possession of a syntax model of recursion by pointing to the differences in the stopping conditions between the techniques.

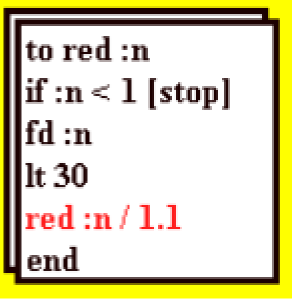
73. George: In the *red*, each time is a sort of check, forward ‘n’, left 30, and then …
74. My description: [*he paused for a while and then continued*]
75. George [*continued*]: … when it gets to ‘n’ is less than 1, it stops and then these go back up again. Why does it go that way?
76. Peter [*interjected*]: So, it goes back on itself. The turtle stopped going, but it is still going back.
77. George: Yeah, with the red, these would go like red
78. My description: [*he was pointing to the three lines of the red procedure if :n<1 [stop], forward :n, left turn 30 and then he continued that*]
79. George [*continued*]: and then that would return red, and then it all shows you what it’s doing each time.
80. My description: [*he was pointing the value of ‘n’ which was showing on the screen and added that:*]
81. George [*continued*]: and this is ‘n’ over 1.1 so, when it goes back on itself …
82. My description: [*then they tried to run the blue technique in the step mode on the comparison module*]
83. George: Ok, each time here, it does this while ‘n’ is greater than 1
84. Peter [*interjected*]: It is showing this is blue!
85. George [*continued*]: And it ends when ‘n’ is less than 1. It is not going to do anything because ‘n’ is less than 1.
86. My description: [*then they tried to run the red technique on the comparison module*]
87. George: If I do the same here, if ‘n’ is less than 1 stop



```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

Figure 69- The stopping condition in the *red* technique

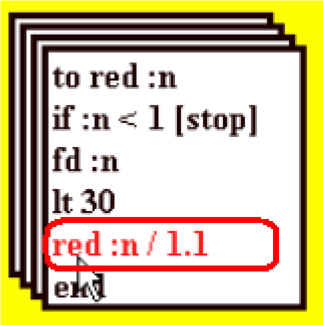
88. [*George moved onto the blue technique and compared the ‘make’ command in the blue technique with the recursive call in the red technique and continued*]: So, this is make ‘n’ that, and this one here red ‘n’ over 1.1



```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

Figure 70- The recursive call in the *red* procedure

89. My description: [*George again moved back on the blue technique and compared it with the red – the recursive call in the red procedure and added that*]
90. George [*continued*]: So, this makes ‘n’, ‘n’ over 1.1 here and this one [*the red technique*] is red [*paused for a while and continued*], so if you got two reds, that’s like a defining this again. And this one is like you have got this within itself!

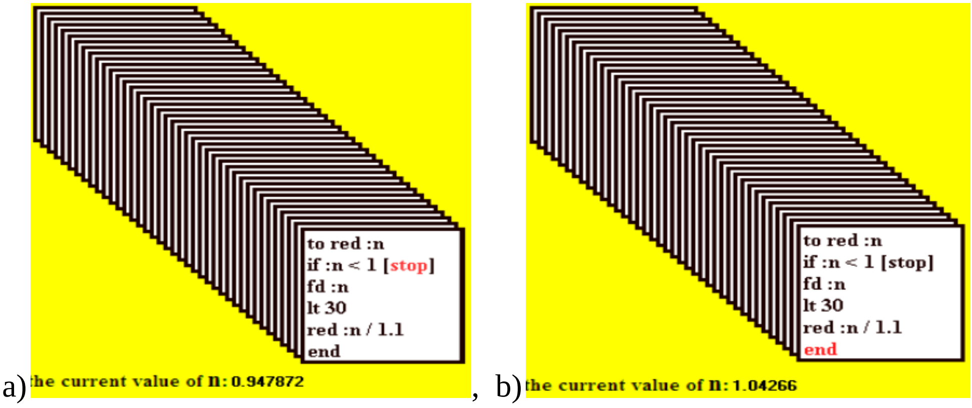


```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

Figure 71- The second *red* within the *red* procedure (two reds)

The following lines show that the AVDA visualisation assisted George & Peter to improve and develop their thinking about the recursive call (lines 91-93).

91. George: So, is that why it goes back on itself! Right, Ok, it does, doesn’t it? Again ‘n’ over 1.1, and it goes again.
92. Peter: Oh, yes! The turtle goes back on itself
93. My description: [*they kept clicking on the step button in the step mode in the red procedure until ‘n’ gets less than 1*]
94. George: So, now ‘n’ is less than 1.
95. Peter [*agreed and added*]: Yeah that’s right
96. George [*continued*]: Stop, if ‘n’ is less than 1 stop. So it goes there.



a)

```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

```
the current value of n: 0.947872
```

, b)

```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

```
the current value of n: 1.04266
```

Figure 72- When the procedure reached its stopping condition and began to end the instantiations

They also developed their understanding of the control passing process. In the lines 97-99 they noticed that the procedure went back on itself.

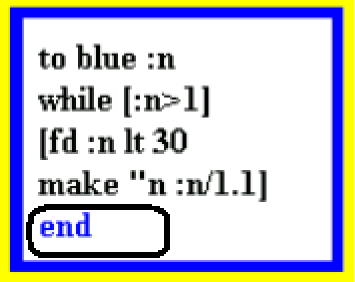
97. George [*continued*]: So, it's going back on itself, because it stops doing it, and it just times' 'n' by 1.1.
98. Peter [*interjected*]: Are there any instructions for the turtle saying it has to stop here and just doing the cancellation?
99. George [*continued*]: No there aren't! It is not going to go forward 'n', left 30 anymore. It just times' the value of 'n' by 1.1 and ends.

Line 99 shows that the visual animative innovation which was employed in the AVDA environment provided them with the opportunity to see the process of finishing the already generated instantiations of the recursive procedure.

100. George and Peter [*both*]: So, it is just reversing the value of 'n' to what it was at the beginning.
101. George: Yeah, and the then ends it! The program is like changing itself after doing different 'n's, and has finished drawing spiral when 'n' is less than 1, and then it goes back. It times' 'n' by 1.1 to get back to what it started with. They are drawing the same thing but with different programs!

After their experience of working with AVDA, George & Peter admitted that the tail recursion technique is more complicated when compared with iteration to generate a spiral (lines 109, 111, and 113).

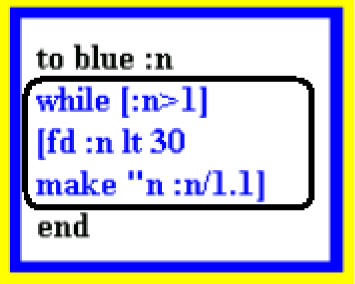
102. I asked: Which one of those techniques would you prefer to use to create a spiral?
103. George [*who was pointing to the blue technique and said*]: This one here is less complicated!
104. Peter [*interjected*]: It is so straightforward.
105. George [*continued*]: While ‘n’ is greater than 1 do this, and do this, and it just checks itself again and then eventually this end [*see below figure*].



```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

Figure 73- The *end* command that George mentioned in line 103

106. My description: [*They tried to run the blue procedure in the step mode again*]
107. George: When you switch to step, so each time it does that [*see below figure*], and then it just checks and changes ‘n’ and then keeps checking while ‘n’ is greater than 1, doing it again and again and again until it is not.



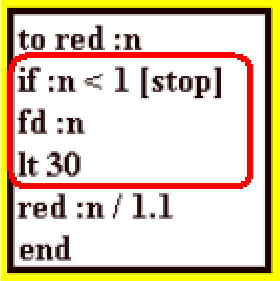
```
to blue :n
while [:n>1]
[fd :n lt 30
make "n :n/1.1]
end
```

Figure 74- The commands that George said are being done while ‘n’ is greater than 1

Then they began to compare the *blue technique* with the *red* one by using the window that the AVDA environment provided them to see the latent layers of the control passing mechanism in those techniques.

108. I asked: Can you see any differences between these techniques?
109. George: Yeah, they are similar things. Just … [*paused a while*] I don’t know, I don’t know! Interesting, so it’s like it starts itself again. To *red*, it’s like, does this [*see below figure*],

and then it starts itself again with a new ‘n’ and starts itself again. So, that’s why the new things pop up, because it keeps restarting itself and then goes back again.



```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

Figure 75- The commands that were being done after each time calling the *red* procedure

At this stage of the activity their responses showed a definite shift in their appreciation and understanding of both processes. This improved perspective differentiated between the iterative and recursive procedures.

110. My description: [*then George moved onto the blue technique and continued*]
111. George: But this one in here doesn’t restart itself and just runs once through.
112. Peter [*interjected*]: And then it goes to the end!
113. George [*pointed to the commands brackets in Figure 75 and continued*]: runs and runs and runs the same thing. [*pointed to the red technique again and added that*] But this one here restarts itself each time.
114. My description: [*and to answer the question 102 when I asked them which one of these techniques they would prefer to use if they wanted to create a spiral, they continued by saying:*]
115. George: I say, in terms of simplicity, probably blue, I would go for blue.
116. Peter: Because, it just seems so straightforward.
117. I asked: What was the most challenging and difficult part of these procedures to understand?
118. George: I suppose, with the red one, why it goes like that, you can’t just look at the lines and see what is it actually doing, what it is telling itself to do! It is difficult to know what it is actually up to!
119. My description: [*then he pointed to the blue technique and continued*]
120. George: But this one here is quite straightforward, you can look at that and see what it is doing. But in the red one in here, all the windows opening here, you can’t see why it is doing that by just looking at it. You can’t see why it is making hundreds of copies of itself and then returning back!

7.1.3. Discussion of George & Peter's account on the *Spirals* – Iteration Three

Line 58 shows that George was thinking about the mechanism of the *blue* technique, the iterative procedure, as a repetitive mechanism (loop). Combine this with his remark on line 63, where he mentions, "*they are the same*" (while he was working with the *red* technique, the recursive procedure) and we can see that he is in possession of a *loop* model for the tail recursive procedure before moving onto the AVDA environment in the *comparison* page. Lines 66 and 68 show that George and Peter built their mental models mainly based on the final output of those procedures – which was the same spiral in both techniques.

In the *comparison* page, the animative visualization in the AVDA environment enabled them to begin the process of thinking-in-change and improving their models. Lines 70-72 show that they were surprised when the *red* procedure started to cancel the already generated copies of the original procedure. That surprise stemmed from the initial mental model that they had for the *red* and the *blue* techniques as *loop* models. The first pointer towards the syntactical differences between those techniques appear in line 72, which can be considered as evidence for a change in their previous thinking about the tail recursive procedure (thinking-in-change process in the AVDA environment) from a loop model to a *syntax* model of recursion.

While they were working with the AVDA environment in the *comparison* page, the thinking-in-change process can be clearly monitored. For instance, when they saw the animative visualization of the tail recursive procedure contrived in the AVDA, George asked, "*Why does it go that way?*" (line 75). Following that question, they tried to carefully observe the way it was working to find it out why the procedure began to cancel the already generated copies after reaching its stopping condition (lines 75-88). Then, in line 88, George pointed to a crucial issue by highlighting the need to have two *red***s** in the *red* procedure. And he concluded that "*that's like defining this again! And this one is like you have got this within itself"* (line 90). He started to compare the '*make*' command in the *blue* technique, the iterative procedure, with the recursive call in the *red* technique and he concluded that the procedure is calling itself from within itself.

Therefore, working with the AVDA environment provided George with the opportunity to think about his previous thinking about the concept of tail recursion. The result, his thoughts evolved from a loop model to an incomplete version of the viable copies model of recursion that I called a *quasi-copies* model. This is where the owner of the model understands the functioning of the recursive call, as it is going to call the original procedure within itself, but still has no idea about the mechanism of delegatory control passing in the recursive procedures.

A viable *copies* model of recursion has two main characteristics: the function of the recursive call as the generator of the original procedure within the main procedure, and the process of delegatory control passing to terminate all the already generated copies to end running of the procedure. In the *quasi-copies* model of recursion, the student appreciates and recognizes the first characteristic, but does not have a complete understanding of the delegatory flow of control in a recursive procedure.

With regard to the case of George & Peter, it become clear that George had gained the knowledge of the function of the recursive call in the tail recursive procedure to generate new copies of the original procedure. But, he did not mention the process of the control passing. He thought that the recursive call defined the procedure within itself again, and that this is the reason for the cancellation of the copies of the procedure (line 91). In line 98, Peter was looking for some instructions for the commands of the procedure to tell the turtle to start canceling the generated instantiations: "*[a]re there any instructions for the turtle saying, it has to stop here and just do the cancellation?*" (line 91).

This remark from Peter is very important because it means that although he reached the point at which the recursive call is redefining the original procedure within itself, he was, however, still looking for some instruction for the strange behaviour of control passing in a delegatory flow in the recursive procedure. Line 91 shows how Peter had difficulty in understanding the declarative nature of the mechanism of the flow in the recursive procedures. At the beginning, before working in the AVDA environment, they both thought that the *red* and the *blue* techniques were the same. But after working with the animative visualization which was contrived into the *comparison* page, they changed their thinking and understanding of those procedures.

George's remark about the techniques in line 109, that "*they are similar things*", is exactly different to the remark that he made before in line 63, saying that they are the same. George in line 109 does not think that the iterative and recursive are the same anymore. Line 109 and lines 111-113 show that, George and Peter significantly changed their thinking about those procedures after working with the AVDA. George described the recursive procedure (the *red* technique) in line 109, stating that "*it's like it starts itself again. ... with a new 'n' and starts itself again*". He continues to discuss the iterative procedure (the *blue* technique) in line 111, saying that it "*doesn't restart itself and just runs once through*" and Peter also in line 113 agreed with him. Lines 114-120 show that to produce a spiral, an iterative structure, they would prefer to use the iterative procedure rather than the tail recursive procedure as they found the iterative one to simpler and more straightforward, where as the tail recursive procedure has a complicated mechanism of control passing.

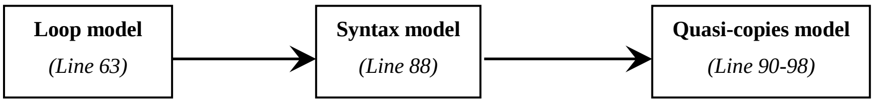


Figure 76- George and Peter's evolution of the tail recursion mental model after working with the *red* and *blue* techniques

Regarding what they explained, the diagram above shows the evolution of George & Peter's mental model for the tail recursive procedure.

7.1.4. Richard and Philip's account on the *Spirals* domain – Iteration Three

Similar to the previous students' accounts, Richard and Peter began their work with the *Spirals* domain after my brief introduction to the procedure. They started with the *blue* technique by testing different initial values in the *red* and the *blue* techniques. They worked with the *blue* technique in the normal mode first and then they moved on to the *red* technique. Richard said:

121. Richard: So, this is just a different colour algorithm? Is that right?
122. I said: What do you mean by a different colour algorithm?
123. Richard: I don't know! I am just asking whether they are just different algorithms or not?
124. I said: What do you think?
125. Richard [*set the initial value 123 and continued*]: So, if you take one like that in the red and one in the blue one can you compare them?
126. I said: You can compare them in the *comparison* page later on, but at the moment can you explain to me whether you can see any similarities or differences between these techniques before going onto that page?
127. Richard: In the wordings or in the results?
128. I said: both aspects!
129. Richard: Umm, can I look at the blue one again.
130. My description: [*he moved to the blue technique*]
131. Richard [*continued*]: while 'n' is greater than 1, forward 'n' amount, left turn 30, and then goes forward with the next value of 'n' which is 'n' over 1.1 Ok, I see.
132. My description: [*then he moved back to the red one*]
133. Richard: And then, this one, if 'n' is less than 1 stop, if not, go forward 'n', left turn 30, red 'n'. That looks like they are almost the same. But, umm, its like, they do this in different steps, and also instead of using the 'make' function, you are naming 'n' with 'red', giving it a name so 'red' is equal to 'n' over 1.1.
134. I asked [*I pointed to the recursive call in the red technique*]: can you explain the function of that line a bit more for me?
135. Richard: That is the same, like, for the next one, use the operation red, but instead of using red, use 'n' over 1.1

Richard's combined remarks in lines 133-135 to show that he possessed a *return-value* model of the tail recursive procedure at that time.

136. Philip [*interjected*]: And how come in one of them ‘n’ is greater than 1 and in another ‘n’ is less than one. Why?
137. Richard: [*pointed to the red technique and said*] That one is just like stop, doing it if ‘n’ is less than one, stop doing it. That is the end! [*then he moved on to the blue technique and added that*] the other one is saying like, make sure the way is positive or something like, only do it while ‘n’ is greater than 1! So, essentially they are doing the same thing really, just in different ways.
138. My description: [*Peter tried to run both of the procedures with the same initial value, Richard suggested taking ‘n’ as equal to 100*]
139. Richard: They are pretty much the same.

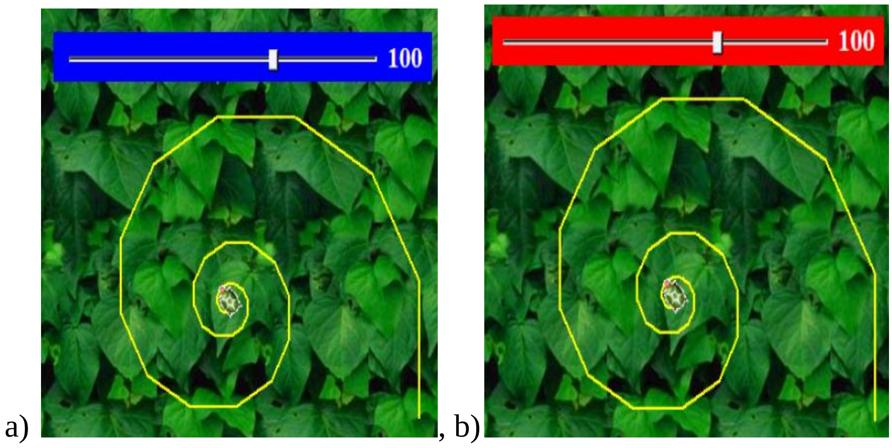
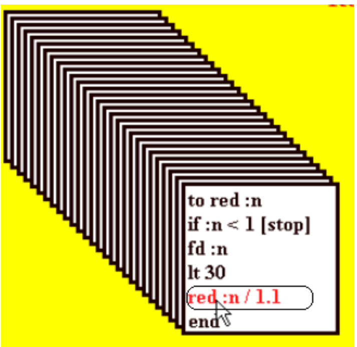


Figure 77- The final output of the *blue* technique (a) and the *red* technique (b) with n = 100

Then they moved onto the *comparison* page to compare both the techniques in one page. They started with the *blue* technique first and then the *red* technique, and ran both procedures in the normal mode. Soon after seeing the animative visualisation in the *red* technique Richard said:

140. Richard: It seems like the difference between these algorithms, the red one and the blue one, is that the blue one does it all in one step, and the red one does lots of repetitions!
141. I asked: What makes those repetitions?
142. Richard: because of the fact that you are redefining red all the time, or redefining ‘n’ as ‘n’ over 1.1



```
to red :n
if :n < 1 [stop]
fd :n
lt 30
red :n / 1.1
end
```

Figure 78- Richard was pointing to the recursive call as re-definer of the value of ‘n’

143. Richard [*pointed to the blue technique and continued*]: rather than here you just say, do it until ‘n’ is less than one or do it while ‘n’ is greater than one. But they are producing the same results. Oh! And then it takes steps back! What is it doing now? Cancelling the steps?

144. I said: What do you think?

145. My description: [*they increased the value of ‘n’ from 50 to 70 and they tried to run both techniques in the step mode*]

146. Richard: So, effectively, doing blue in the step mode is like doing red in normal mode. Is that right?

147. I said: Why do you think so?

148. Richard [*continued*]: Doing blue in step mode is like doing red because you are doing manual repetitions. I think that, because, it seems like that! Because, if you do blue like that, and then you do red normally, it is doing the same thing. Isn’t it? That’s doing automatically what I did over there in blue manually, isn’t it?

149. I said: have you considered the colour codes that were used in the lines of those procedures?

150. My description: [*they tried to do the blue and the red in the step mode again and in the red they said*]

151. Richard: If ‘n’ is greater than 1, go to the next step, forward ‘n’ and left 30 and now redefine ‘n’ [*when it gets to the recursive call*] it goes through and then it goes back up and increase ‘n’ to what it was, 70, so with the blue one we don’t know what the value of ‘n’ was when we started.

152. I asked: If you wanted to create a spiral which one of these techniques would you prefer to use?

153. Richard: I think doing it automatically, I prefer red. But doing it manually I prefer blue, because if you do step with red you have to click a lot! I think the red one is easier because at each point you can see what the value of ‘n’ is, and you get the final value of ‘n’. But I think, with blue is quicker.

154. Philip [*interjected*]: You can see more information from the blue.

155. I asked: Are there any differences between these techniques?

156. Richard: Between the red and the blue? What do you mean? In terms of how to use them? Or, in terms of the results?
157. I said: In any aspects that you think.
158. Richard: Red is probably kind of a more robust algorithm, like if you made it more complicated blue might fail. Because, it is quite simple, go forward, go left, and change it, go forward, go left and change it, etc. But, if you made quite a complicated algorithm, I think the red would cope with a lot more instructions, because it is step by step. It is more like a flowchart. I don't know! This one is more like one operation, but that one is like a flowchart, a kind of questions and answer!

7.1.5. Discussion of Richard and Philip's account on the *Spirals* domain – Third iteration

In line 127, Richard's remark show that he distinguished between the *wording* (the syntactical commands in the given procedures) and the result (the final output of the procedures). This remark evidenced that Richard possessed a *loop* model which had evolved into a *syntax* model. Distinguishing between the syntax and final output of the procedure is called the difference between the process and product. From a syntax view, he pointed to the difference between the stopping conditions in the two techniques in lines 131-133. From the output view, in line 133 Richard mentioned that "[...] *they are almost the same*' which evidenced his possession of a *loop* model of tail recursion.

Richard's utterances in line 133 are very important and give rich insight into how and what he was thinking about the concept of tail recursion. It also showed how he changed his thoughts while he was working with the tools. Indeed, he began with a loop model based on the final output of the procedure and then, based on the syntactical differences, he moved on to a *syntax* model of recursion by saying "[...] *they are almost the same. But* [...] *it's like they do* *it in different steps*'. Then in lines (133-135) he continued towards a *return-value* model by comparing the '*make*' command iterative procedure (the *blue technique*) and the recursive call in the recursive procedure (the *red technique*): "[...] *instead of using 'make' function, you are naming 'n' with 'red'* [...] *so, 'red' is equal to 'n' over 1.1*" (line 133). Therefore, the evolution of Richard and Philip's mental models of the tail recursion before moving on the *comparison* page can be categorised as follows.

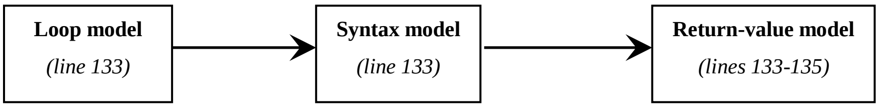


Figure 79- Richard and Philip's evolution of tail recursion mental model after working with the *red* and *blue* techniques (before their experience with the AVDA innovation)

Then they moved on to the *comparison* page. Soon after seeing the AVDA innovation on that page, Richard evolved his model into the *quasi-copies* model of tail recursion as he mentioned that "[...] *the blue one does it all in one step,* [...] *the red one does lots of repetitions*" (line 140). This shows that he was thinking about the flow in a procedural active process and did not have any idea about the delegatory behaviour of flow to the new copies of the original procedure at that stage.

I wanted to see what he thought about those repetitions and asked him what makes those repetitions. His responses in lines (142-144) shows he had possession of a mixture of a *quasi-copies* model of tail recursion and the *return-value* model by saying "[...] *you are redefining red all the time, or redefining 'n' as 'n' over 1.1*" (line 142). However, lines 144-148 show that, soon after seeing the cancellation process in the AVDA environment, he abandoned his embryonic quasi-copies model of recursion and moved back to a *step* model of the tail recursion concept by saying "*[s]o, effectively, doing blue in step mode is like doing red in normal mode*" (line 146). Therefore, a diagram of the evolution of Richard and Philip's mental models can be shown as follows.

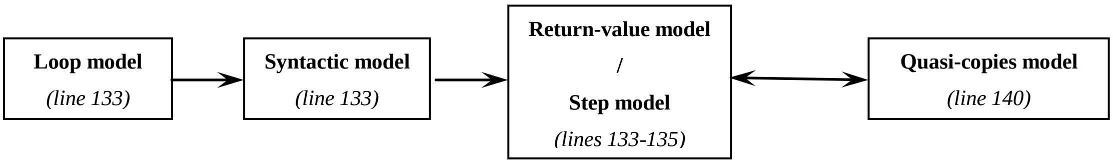


Figure 80- Richard and Philip's evolution of the tail recursion mental model after working with the *red* and *blue* techniques (after their experience with the AVDA innovation)

Finally, in lines 152-158, they compared the two recursive and iterative techniques. Although they did not achieve a viable correct model of the recursion, Richard's remark in line 158 evidenced that they discovered that, compared with the iterative procedure, the recursive procedure is more powerful and able to afford much more difficult situations than drawing a spiral: "*red is probably a kind of robust algorithm, like if you made it more complicated blue might fail.* [...] *I think red would cope with a lot more instructions* […] *it is like a flowchart* […] *a kind of question and answer*" (line 158). This offers very rich information about how he was thinking about the recursive procedure. When he pointed out that a recursive procedure is like a flowchart a kind of question and answer, it can be considered as evidence for his possession of a quasi-copies model. The term "question and answer" which was explained by Richard, can be taken into account as generating the new copies of the original procedure.

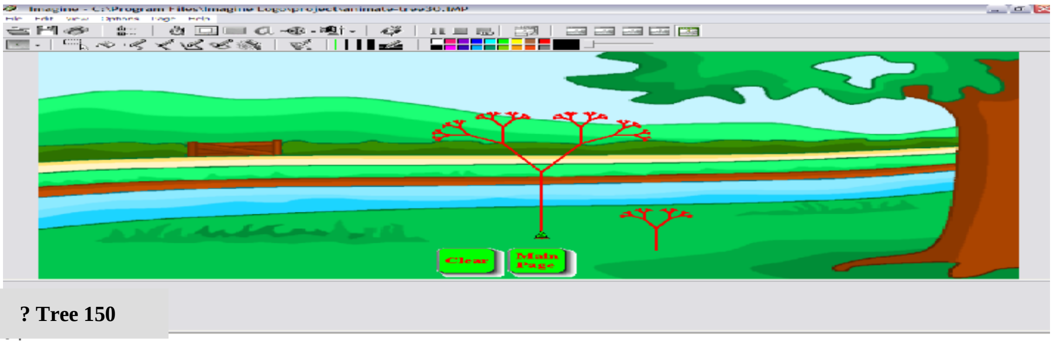
In the next part of this chapter, the main focus is the students' accounts of the four modules of the *Treebuilder* domain of abstraction. In the second part, the accounts of the same students in part one are thoroughly discussed.

7.2. PART TWO – Students' Accounts on the *Treebuilder* domain

In the first part of this chapter, I discuss three students' accounts for the *Spirals* domain. In this part of the chapter, I discuss the accounts of the same students for the *Treebuilder* domain.

7.2.1. Simon's account on the *Treebuilder* domain

Simon started his experiment with the *Spirals* domain by working with the *making a forest* module. It has been mentioned before that the students were asked to type the term '*tree*' followed by a number on the command line and press the button labelled *run*. The purpose of the task was to see how they build a connection between functionality and functioning. What I mean by functionality and functioning is the difference between *what* needs to be done and *how* it will be done.



? Tree 150

Figure 81- The main interface of the *making a forest* module

Simon started to work with this module immediately after finishing his experiment with the tasks in the *Spirals* domain. Hence, he began to work with the module in a systematic style. First he typed the term '*tree*' on the command line and when he was putting a number next to it stated:

1. Simon: So, alright, the initial value is going to be 'n', it is going to be how big the tree is going to be. So, if I drag this …
2. My description: [*He was pointing to the turtle on the main page of the making a forest module*]
3. Simon: I will get a tree initial length 50. Presumably there is a program.
4. My description: [*He examined few other values for 'n'*]
5. I asked: Can you say anything about the structure of the trees that you have made?
6. Simon: Umm, well, I presume in the program you put an angle in, like 30 degrees either way, and each part in the tree is 50% of the length of the previous one, I think. It is like a fractal, isn't it? … I'm sure it has got a limit on the number of steps. They might not have it and the stems are going on and on but I can't see them.
7. Can you see any relationship between this and the previous modules in the *Spirals*?
8. Simon: Um, well, yes! Because each step is reducing the length of 'n' each time. The spiral task was reducing the length of 'n' by a factor of 'n' divided by 1.1. In this task it looks like you're reducing each 'n' by 50% each time. But it might not be quite a percent but it looks like, I'm not sure! I'm sure you could write a program so that reducing it by 0.75 or something like that. So, yeah, it looks like that on each step the angles are preserved. So, you could have said 30 degrees and then each stem on the tree that's 30 degrees to the left, that's 30 degrees to the right!
9. My description: [*He was pointing to the branching points and the new stems to the left and right directions*]
10. Simon [*continued*]: Then here that's 30 and 30 and another 30 and 30. It is a recursive program again. You have probably written something like let re-substitute back in 'n' divided by 2, …
11. My description: [*He was pointing to the first branching point*]
12. Simon [*continued*]: For example, into the original formula for this bit here.

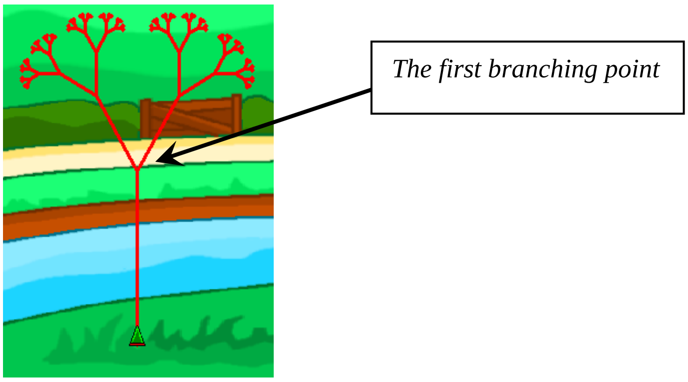


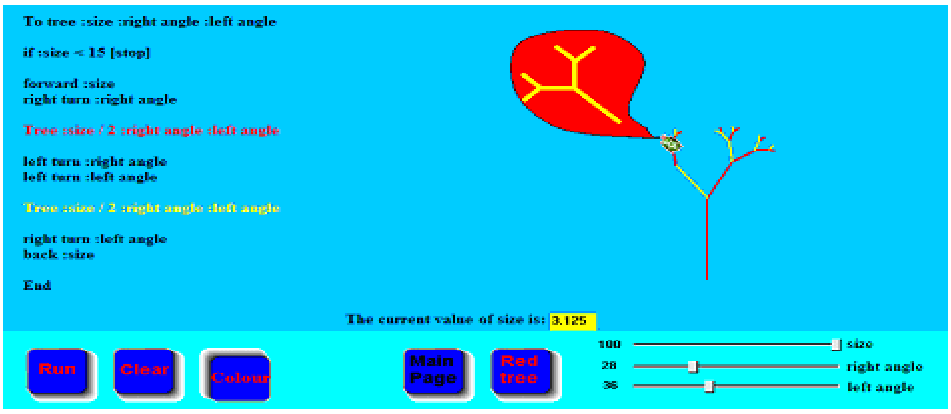
Figure 82-Simon was pointing to the first branching point

Lines 3-12 show that Simon's previous experiment with the *Spirals* domain gave him a good sense of combining functionality and functioning dimensions. Also, by looking at the tree he pointed out that the structure is recursive (line 10). I wanted to see what he really thought when in line 10 he mentioned that the program is recursive! So, I asked him about it.

13. I asked: What do you mean by recursive?
14. Simon: Umm, obviously in the previous one, the spiral one, you wrote a small program for generating one line, and then the second line within your spiral, and then you did the program again. Basically, just plug that back into the equation each time to generate the next step of your spiral. And I imagine that the same is true for the tree. You have got something in there whereby you have created this step here [*see Figure 82*] you have told the program to do this and then you have just recursed it, to set now here, do that the same,
15. My description: [*then he was pointing to the second branching points (see Figure 82) and continued that*]
16. Simon [*continued*]: Well, we have not seen the program so I don't know.

Simon gave a very interesting description of the trees in the *making a forest* module. In line 1, he pointed to 'n' as the initial value for the size of the first branch. He had not seen any syntactical commands at that stage in the module, he was just asked to type the term *tree* followed by a number. From a structural point of view, in lines 6 and 8 he also pointed to the angles and the size of the new stems. Simon showed his appreciation of the stopping condition when he said, "*...it has got a limit on the number of steps*" (line 6). His remarks in lines 8-16 evidenced the influence of the *Spirals* domain on his interpretation of the trees in the *making a forest* module in the *Treebuilder* domain. In lines 10-16, he mentioned that those trees have a recursive program. What he meant by the term recursion was a kind of re-substitution of the value of 'n' with a new value, say 'n' divided by 1.1 as he had seen in the *Spirals* domain (line 10). When in line 14 Simon said "[…] *you wrote a small program for generating one line, and then the second line within your spiral, and then you did the program again*. […] *plug that back into the equation each time to generate the next step of your spiral. And I imagine that the same is true for the tree.* [...] *you have told the program to do this and then you have just recursed it*", it evidenced his possession of a *quasi-copies* model of the recursive procedure, which was mainly imbued by his previous experiment with the spirals in the *Spirals* domain.

After this stage, we moved to the next module, the *blue* strategy. This module, as mentioned before, was mainly designed to present an animative visualisation of the control passing process over the embedded recursive procedure by using two shadow turtles which were moving alongside the main turtle and also two red and yellow colour codes for the first and second recursive calls.



```
To tree :size :right angle :left angle

if :size < 15 [stop]

forward :size
right turn :right angle

Tree :size / 2 :right angle :left angle

left turn :right angle
left turn :left angle

Tree :size / 2 :right angle :left angle

right turn :left angle
back :size

End
```

The current value of size is: 3.125

Figure 83- The written commands for the embedded recursive procedure to generate a binary tree in the *blue* strategy and the colour codes (red and yellow) for the recursive calls

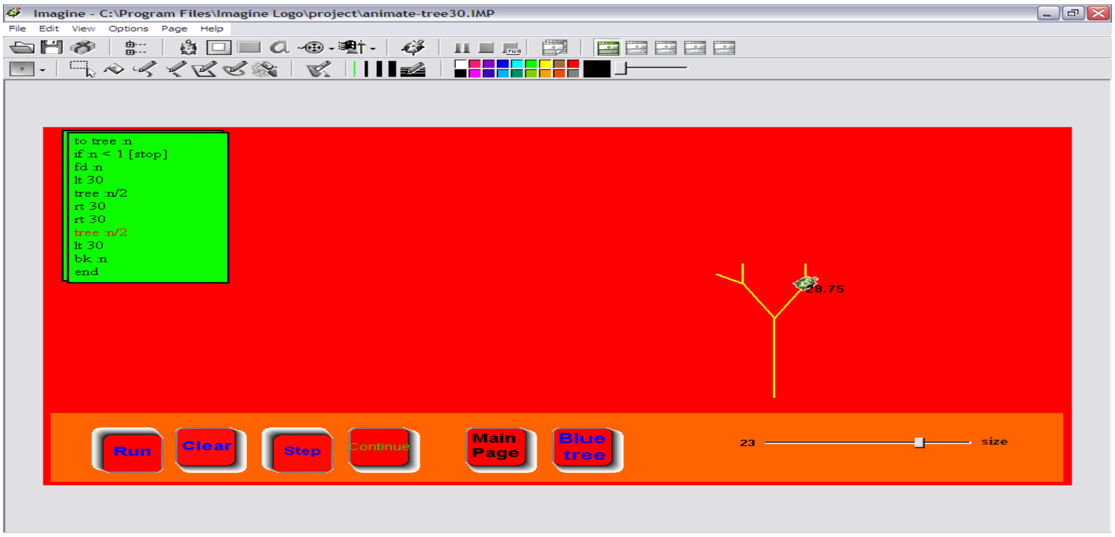
Simon started to check the written commands of the embedded recursive procedure on the screen before running it. His utterances in line 17 evidenced his possession of a *loop* model of the embedded recursive procedure at this stage.

17. Simon: So, if size less than 5 stop. If the size is greater than 5 you carry on, then it forwards the amount of the size. And then you did right turn whatever you said the right angle is, and then you carry on by size divided by 2, same angle each time, the angles are preserved. It does a left turn and a left turn and then you carry on again with tree with size divided by 2 and so on.
18. My description: [*then he clicked on the run button and ran the procedure in the normal mode*]
19. Simon: Now, it doesn't look like it goes more than that, about 3 steps, let's change the angles into what I thought it was 30 and 30 and then run it. Yeah, it had only got 3 steps!

Whilst he was working within the AVDA environment and the animative visualisation in the *blue* strategy, he noticed that the procedure did not behave like normal loop as he expected. Instead, each time it was doing its job "[…] *in a few steps*" (line 19). Then he ran the procedure in the colour mode and thoughtfully watched the animative visualisation, which was contrived into the *blue tree* module by the shadow turtles and colour codes for the recursive calls.

20. Simon [*continued*]: I think it is doing four steps to get the size less than 5 and then stopping. It is going forward 100, 50, 25, 12.5, 6.25 I cannot see the 3.75, oh yes, because when you get at 6.25 you get the size divided by two and you get back into there
21. My description: [*He was pointing to the stopping condition line in the procedure if :size < 5 [stop] and added that:*]
22. Simon: And it becomes less than 5 so stops. The only thing we have had is 1, 2, 3, 4 stems on your tree every time, and then stop. Regardless of how big the size is. If you do the size equals 20, it is going to be tiny.
23. I asked: Why do you think we are going to have only 4 stems each time?
24. Simon: Because, the first which is 20, then 10, 5, and then it stops. If you make it 18, you only will get two stems. Let me hide the turtle.
25. My description: [*Then he hid the turtle to see the little stems*]
26. Simon [*continued*]: Yeah, it has 2 stems to get to 4.5, I like that! Can I look at the *red tree* module? Oh, before going there, let's have a quick look to see what happens if we change the angles. 58 degrees for the angle to the right and 28 degrees for the angle to the left! It just squeezed it one way. Presumably, if we would do that in the other way, 25 and 61 for right and left, it is going to be shaped that way.

Simon's experience with the AVDA helped him to evolve his initial loop model of the embedded recursive procedure (line 19) into a *step* model in line 20. In that line, he also pointed to the syntax of the procedure "[…] *you get the size divided by 2*" (line 20), so his explanation can also be considered as evidence of possession of a *syntax* model. Simon's remarks in lines 20-26 showed he had a good understanding of the parameters of the angles and stopping condition of the embedded procedure. However, he did not show any sign of appreciation of delegatory control passing in the procedure at this point. Next we moved on to the *red* strategy, which was again another version of AVDA innovation based on the visualizing of the copies of the original procedure, similar to the technique I employed in the *Spirals* domain.



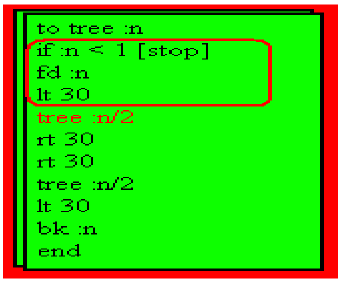
Imagine - C:\Program Files\Imagine Logo\project\animate-tree30.IMP
File Edit View Options Page Help
28.75
Run
Clear
Step
Continue
Main Page
Blue tree
23
size

```
to tree :n
if :n < 1 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

Figure 84- The interface of the *red* strategy after running it in the step mode

Simon was surprised by seeing the main interface of the red strategy before running it, as it does not have anything on the screen until the user presses the *run* button.

27. Simon: Oh! There is no instruction!
28. My description: [*He pressed the run button and ran the procedure in the normal mode and the copies of the embedded procedure appeared on the screen [see Figure 84]*
29. Simon [*continued*]: Oh Ok here they are! This is changing, this is different!
30. My description: [*then he started to run the procedure in the step mode to have a closer look at what was happening on the screen. He pressed the step button, run button and finally the continue button to see the result on the screen in the step mode*]
31. Simon: Ok, so step, and then run, and then continue, Ok, size is 125, 125 is not less than 1 so we carry on, we go forward 125 and left 30, and then it goes back up to here …
32. My description: [*he pointed to the commands above the first recursive call (see Figure 85)*]



```
to tree :n
if :n < 1 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

Figure 85- The commands above the first recursive call that Simon was pointing to are shown in the red box in this picture

Simon's remark on line 31 is an important remark. This is because he noticed the control passing mechanism and its relationship with the stopping condition. Actually, whenever he pressed the *continue* button, the commands above the first recursive call were being executed and the first recursive call, flashing in the colour red, was waiting for him to press continue again. Therefore, the AVDA innovation allowed him to see that, when the procedure reaches its limit for the stopping condition, it started to do the suspended commands below the first recursive call. Simon's remark in line 31 evidenced his possession of a *quasi-copies* model of recursion. He continued:

33. Simon: Oh, Ok, now we do tree 'n' divided by 2, right 30, right 30!
34. My description: [*when he pressed the continue button the procedure got to its second recursive call and so it jumped back up to the original procedure again!*]
35. Simon [*continued*]: Where does that goes? It's over there again
36. My description: [*he was pointing to the commands above the first recursive call*]
37. Simon [*continued*]: Alright, it goes back up to here again, forward 'n' left 30 then it comes back down to this step and then it goes to the right 30!

In lines 33-37, he tries to describe the delegatory control passing process between two recursive calls over the whole procedure. Lines 31-33 evidenced an evolution of Simon's thinking and a transition state from having a *loop* model to a *quasi-copies* model of the embedded recursive. In line 31, he commented that the procedure is going to go back on the previous 4 commands after getting to the first recursive call "*tree 'n' divided by 2*" and also another evidence is his utterance in line 33, when he said: "[…] *now we do tree 'n' divided by 2, right 30, right 30*" (line 33). In line 37, Simon evolved his description by interpreting the recursive call as a *step* which showed his possession of a *step* model of the procedure. He was still expecting that the procedure would pass those steps in a procedural and iterative way. But, control was passing to the top of the procedure after calling each recursive call. So, he said:

> 38. Simon [*continued from line 76*]: No it won't, Oh right, what is it doing? It's building it to the left first! It does all the lefts first. So, we will get another one, and another one, and it will stop. So now it goes to right! Let me see, it's going forward and then left 30, tree 'n' divided by 2, back into tree, back into tree, back into tree, if 'n' is less than 1 stop, go right 30, right 30, to get back on the main part of the tree, it goes tree 'n' divided by 2 and then go left 30 and then go back by the value of 'n'. I see, it is pulling you back.

Simon's remark in line 38 shows that although he could see that after each time of calling the first recursive call, the control is passed to the top of the procedure. The procedure will do this until reaching its stopping condition limit, which is 'n' less than 1. In the AVDA environment he also noticed that after reaching that limit the procedure resumed executing the suspended commands "[…] *go right 30, right 30, to get back on the main part of the tree*" (line 38). However, he had difficulties in describing the similar second recursive call.

39. Simon: It is pulling you back. If we had different colours we could probably see it better when it comes back down the tree and then down to the other spirals!
40. I asked: Which part of the procedure was difficult to understand?
41. Simon: Umm, the part where you've drawn the tree and you have all of the steps up to the left hand side, going left, then because you cannot actually visualize what is going on. Because nothing seems to be happening to the tree, the procedure re-traces the steps back along the tree to get the next bit to carry out, it does that, and comes back again, but it is definitely going back along the tree, it's obvious once you worked out what it is doing, because it's re-tracing the steps.

Simon's explanation on line 41 shows the most difficult part of the procedure for him to trace and understand was the control passing process between the recursive calls and over the whole procedure. The delegatory control passing for him was like a black box as he described it thus: "*[b]ecause nothing seems to be happening to the tree, the procedure re-traces the steps back along the tree to get to the next bit to carry out*" (line 41).

42. I asked: What part or parts of the *blue* strategy was difficult for you to understand and work with?
43. Simon: The *blue* is better because it puts it into different colours. So, we can see, it makes what is it doing a lot clearer, and the turtle is actually moving along the diagrams - you can see it is drawing it out and re-tracing the steps all the time to make sure the correct tree is produced. So I prefer the *blue* one!

Simon's remark in line 43 shows that his main difficulty in understanding and working with the embedded recursive procedure was tracking the flow of control over the procedure. This can be seen from his comment when he pointed out that "[…] *it makes what it is doing a lot clearer* […] *you can see it is drawing it out and re-tracing the steps all the time to make sure the correct tree is produced*" (line 43). Furthermore, his comments in that line also evidenced his *step* model interpretation of the recursive calls. Altogether, before moving on to the final module of the *Treebuilder* domain, I can summarise Simon's evolution of mental model as follows.

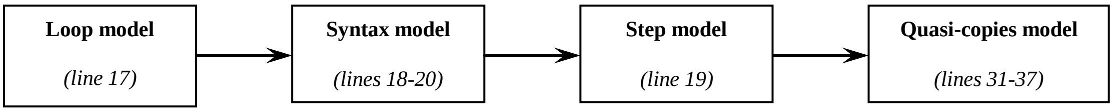
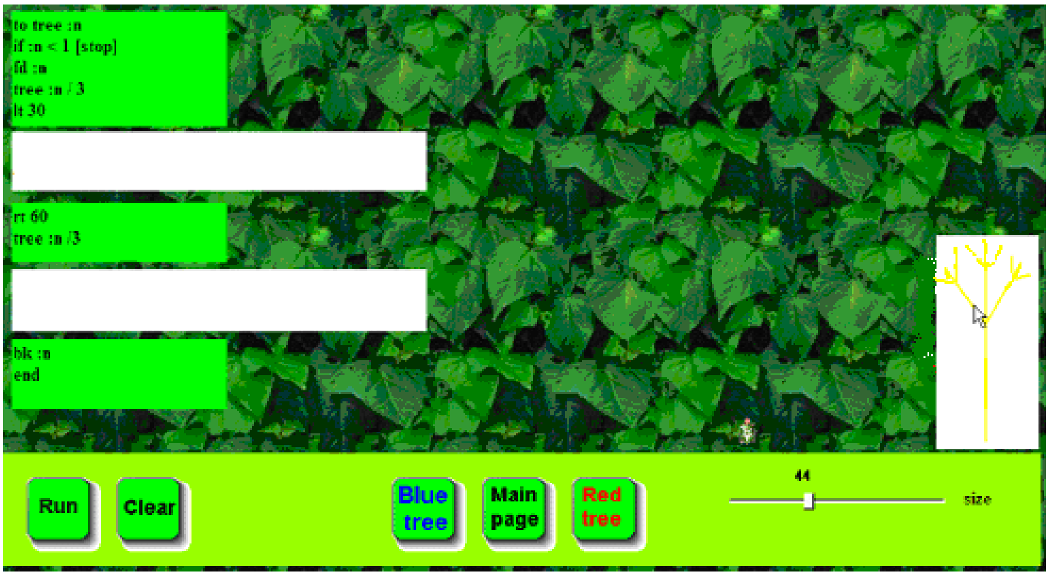


Figure 86- Simon's evolution of embedded recursion mental model after working with the first three modules of the *Treebuilder* domain

Finally we moved on to the last module of the task which was the *your tree* module. In this module the students were asked to complete an incomplete embedded recursive procedure. The main interface of this module is shown in the picture below.



```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30
```

```
rt 60
tree :n /3
```

```
bk :n
end
```

Figure 87- The main interface of the *your tree* module

The following lines show how Simon engaged with the task in the *your tree* module:

44. My description: [*Simon started his work with the your tree module by following the commands in the incomplete embedded recursive procedure and comparing it with the given image of a ternary tree at the bottom right corner of the screen*]

45. Simon: Ok , it is interesting, forward ‘n’ then ‘n’ divided by 3, ‘n’ divided by 3 makes a new ‘n’ so we go forward again up here we stay up there and then we have done left 30

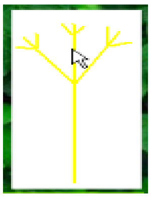
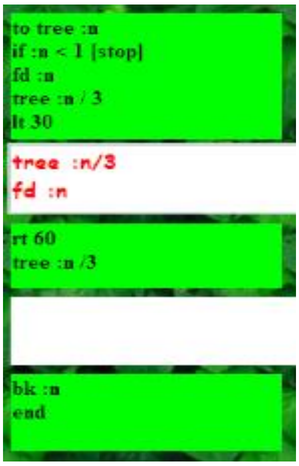


Figure 88- Simon was pointing to the second vertical stem in the ternary tree

46. My description: [*he moved back to the red tree strategy and then typed (tree :n / 3) followed by (forward :n) in the first empty box*]



```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30
```

```
tree :n/3
fd :n
```

```
rt 60
tree :n /3
```

```
bk :n
end
```

Figure 89- Simon’s first attempt to complete the incomplete embedded procedure by putting two lines in the first empty box

47. Simon [*continued*]: Forward ‘n’, and then right 60.
48. My description: [*He was pointing to the last stem to the left in the middle part of the ternary tree and added*]
49. Simon: Forward ‘n’. We are up here.

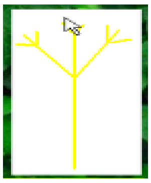


Figure 90-Simon was pointing to the last stem to the left in the middle part of the ternary tree

50. Simon [*continued*]: I’m not sure now. Or, are we up here?

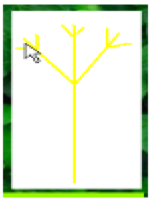
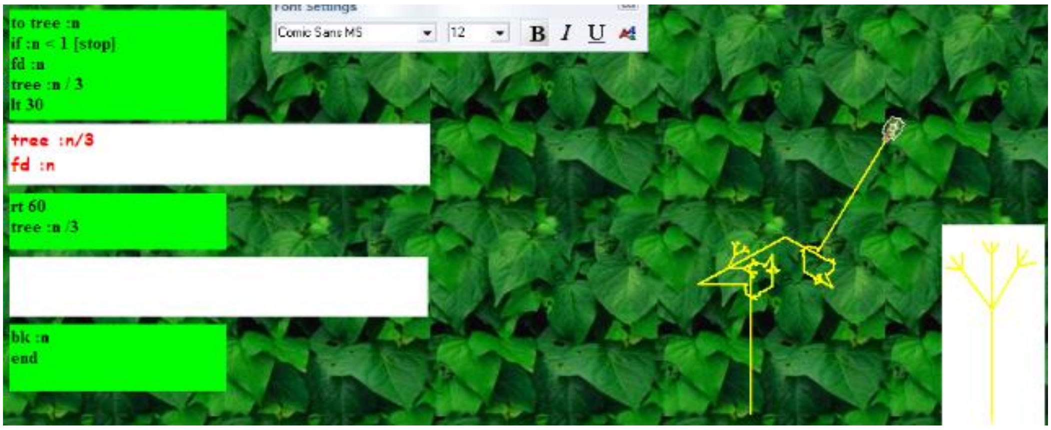


Figure 91- Simon was pointing to the last stem to the left in the left part of the ternary tree

Simon's remarks on lines 47-50 show that he did not have a clear understanding of the delegatory control passing process. He was not able to figure out the functioning of the first recursive call in the embedded procedure. As he mentioned in line 45, the recursive call was a step to change the value of 'n' into 'n' divided by 3, which can be considered to be evidence of possession of a *return-value* model of the embedded recursive procedure by him to complete the procedure. Simon ran his amended procedure, shown in Figure 92, and the result was far from the result that he expected.



```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30
tree :n/3
fd :n
rt 60
tree :n /3
bk :n
end
```

Figure 92- The result of the amended procedure

He laughed at the result and said:

> 51. Simon: I cannot work out where I am! So, I must be here.

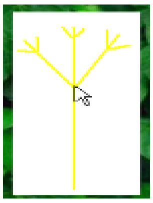


Figure 93- Simon pointed to the first branching point to show the location of the turtle after moving forward ‘n’

52. Simon [*continued*]: Tree ‘n’, forward ‘n’, tree ‘n’ divided by 3, left 30, so, that goes to up there.

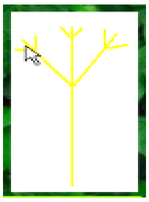
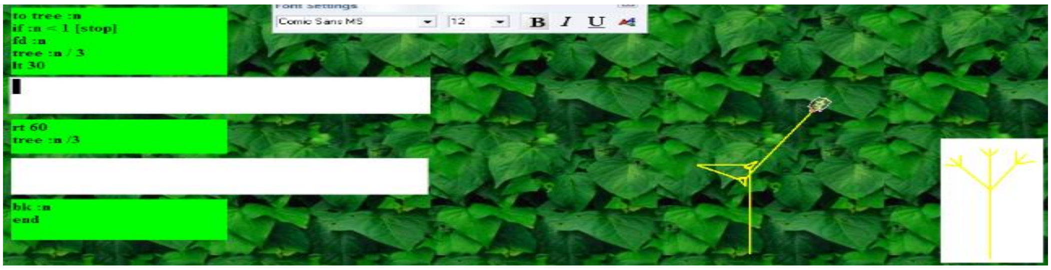


Figure 94-Simon pointed to the new location of the turtle after the left 30 command

He moved back on the *red* strategy and thoughtfully checked the written command of the embedded recursive procedure on that module and said:

53. Simon: Oh, that one does not do that! That one goes straight into the tree ‘n’ over 2!
54. My description: [*actually in the red strategy we were working with recursive procedure to produce a binary tree. However, in the your tree module the students were asked to complete the embedded procedure to produce a ternary tree*]
55. Simon: I am totally stuck!

He then decided to remove the commands that he had put into the first box and ran the procedure with two empty boxes to view the result.



```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30
```

```
rt 60
tree :n /3
```

```
bk :n
end
```

Figure 95- Simon ran the procedure without putting anything into the empty boxes

56. Simon: I am trying to work out what it is doing without my instructions. Forward ‘n’, tree ‘n’ divided by 3, left 30, and then right 60, which is basically right 30, it is going to up there

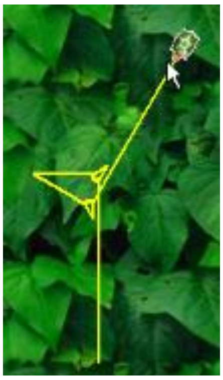
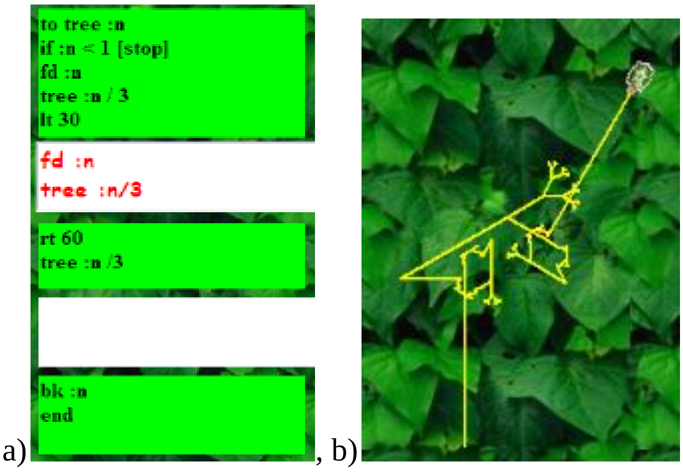


Figure 96- Simon is pointing to the location of the turtle after going left 30 and right 60

He moved back on the *red* strategy again and amended the procedure as follows by typing first forward ‘n’ and then tree ‘n’ over three into the first empty box.



```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30
fd :n
tree :n/3
rt 60
tree :n /3

bk :n
end
```

Figure 97- (a) is the amended procedure by Simon and (b) is the output of the amended recursive procedure

57. My description: [*he tried to describe what is going on by the commands that he had put into the first empty box*].
58. Simon: So, that one goes forward ‘n’, tree ‘n’ divided by 3, left 30, forward ‘n’, tree ‘n’ divided by three, right 60.

Simon's explanation in line 58 shows that he had no imagination of the delegatory flow of control after each time calling recursive calls. I wanted to know what he was actually thinking about the recursive calls.

59. I asked: Can you describe to me what that forward 'n' and tree 'n' over 3 in the box are going to do for you?
60. My description: [*he pointed to the last stem of the left part of the ternary tree (Figure 98) and said*]
61. Simon: Actually at that point we do not need any tree anymore! And the forward is actually because it is like tree forward 'n' to make a trunk. From the beginning you go forward 'n', then go 'n' divided by 3, and then forward 'n; again and you went up there.

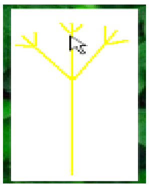


Figure 98- Simon was pointing to the location of the turtle, after his remarks on line 61

62. Simon [*continued*]: Then you go left thirty and that's where I get stuck! Because left thirty means,

At this stage, Simon decided to move on to the *blue* strategy, in which I has employed the colour codes for the stems, yellow to the right and red to the left in accordance with the first and second recursive calls. The AVDA visualisation in the *blue* strategy helped him to change his thinking about the complicated control passing process in the embedded recursive procedure. He described the *blue* strategy as follows.

63. Simon: Right, it is going to do all the right turns first, and then it is doing all the left turns afterwards!
64. My description: [*then he pointed to the shadow turtle in the blue strategy and said*]
65. Simon [*continued*]: So, it is telling it to create a tree on top of a tree that has already been created. I do understand what it is doing. But, it doesn't make it any easier to put the information on to tree. It really does not! To tree 'n', forward 'n', then tree 'n' divided by 3, left 30, it is going to go forward at this point.

66. My description: [*he was pointing to the first branching point*]
67. Simon [*continued*]: at this point the length is going to be ‘n’ divided by 3. So, you go forward ‘n’ divided by 3. That was what I thought originally!
68. My description: [*He typed forward ‘n’ over 3 in the first box and said*]
69. Simon [*continued*]: Ok then, now you have done forward ‘n’, tree ‘n’ divided by 3, left 30, forward ‘n’ divided by 3, then you have got to go the right turns! So, you have to say right 60, and then you do tree ‘n’ divided by 3 again.
70. Simon: Oh! That was wrong. The thing is we only define tree ‘n’ as being forward ‘n’ that’s my problem! In the other one we defined tree ‘n’ as being forward ‘n’, left 30 or whatever.
71. My description: [*Simon was pointing to the difference between the embedded recursive procedures in the your tree module and the red and blue strategies*]
72. Simon: I am really stuck. I have to say I am totally stuck! I can see what is it doing but I just cannot do my own. Because I cannot! The way I do things, the way I always learn stuff and so on is to look at an example and try to apply that example to the one that I am doing. In this case, I just can’t do it. I just can’t get my head around it.

Simon’s remarks on the above lines show that his main problem in completing the incomplete embedded procedure was his inadequate imagination about the delegatory control passing mechanism. However, the animative visualisation of the AVDA in the *red* and *blue* strategies helped him to change his thinking and model about the embedded recursive procedure.

7.2.2. Richard and Philip’s account on the *Treebuilder* domain

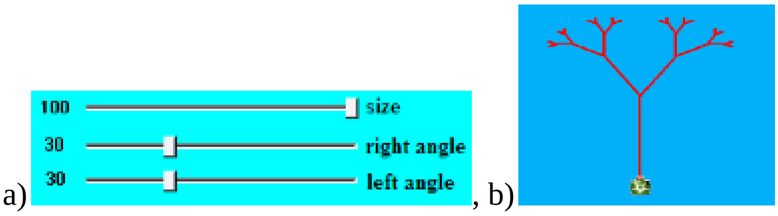
Similar to Simon’s account, Richard and Philip also started the second part of the third iteration by working with the *Treebuilder* modules. They began their experience with the *making a forest* module by testing a few different values. They tried *tree 7, 12, 100, 200* and observed the outputs.

73. I asked: Can you see any relationship between these trees and the spirals that we already had?

74. Richard: It goes up and it turns left and then it goes forward again. But, every time it turns left it also turns right,
75. I asked: What do you think Philip?
76. Philip: Yes, that's right, it goes in both directions!
77. Richard: Yeah, instead of going forward, left, 'n', and then again forward, left, 'n', it goes forward, and then left and right, and then forward, and then left and right.
78. Philip [*interjected*]: It is going forever!
79. I asked: Richard, what do you think? Do you also think that it goes on forever?
80. Richard: No, I think it is going to stop somewhere. Like spirals but, it always goes forward, left and right, goes forward, then left and right, and so on.

Then we moved on to the *blue* strategy, which was designed to employ the AVDA visualization combined with the red and yellow colour codes for the stems to the right and left in accordance with the first and second recursive calls in the given embedded recursive procedure to produce a binary tree. Richard set the initial value of the size at 64 and then they ran the procedure in the normal mode.

81. Richard: You turn 90 degrees to the right, and turn 73 degrees to the left. Let's do 30 and 30, which was the angle of the spiral.
82. My description: [*then they set both angles to the left and right equal to 30*]
83. Richard [*continued*]: That would be a tree like the spiral!

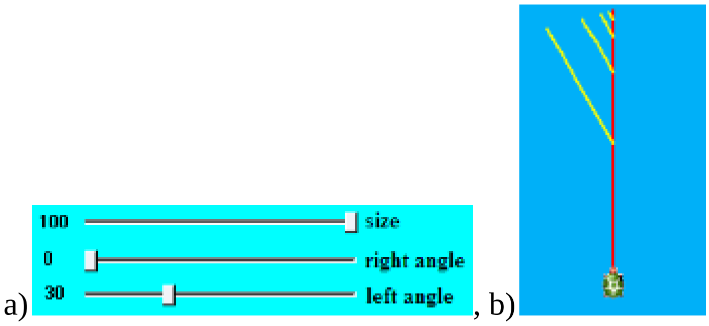


100 size
30 right angle
30 left angle

Figure 99- The tree that Richard made by taking both angles equal to 30

After experiencing the normal mode, they decided to test the colour mode execution of the procedure. The AVDA technique combined with the colour codes caused them to change their thinking about the way that the tree was being made.

84. Richard: It does look like a bit of red and then a bit of yellow
85. I asked: Can you describe it for me, what is it doing?
86. Philip: The red is going to right and the yellow goes to the left. I reckon!
87. Richard: Oh, yes, red is right and yellow is left! Ok, what if I get one angle zero?



100 size
0 right angle
30 left angle

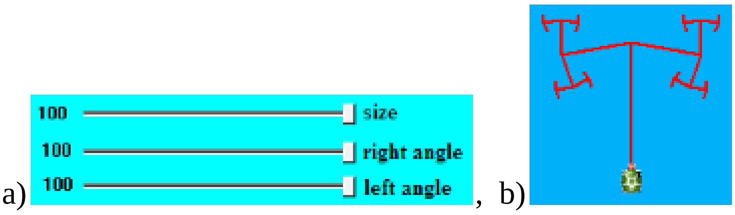
Figure 100-(a) Richard took the angle to the right zero and (b) The output of the procedure with one angle zero

88. Richard [*continued*]: That would be very like a one-sided tree, everything on the left hand side is going to be yellow!

Initially, line 77 showed that Richard was thinking about the binary tree as a *loop* model by saying "[…] *it goes forward, and then left and right*". But, after seeing the AVDA visualisation in the *blue* strategy, he noticed that "*[i]t does look like a bit of red and then a bit of yellow*" (line 84). The AVDA visualisation in the *blue* strategy opened a window for them to look through and change and re-shape their thinking about the concept of recursion. Using this window, Richard changed his interpretation from "*forward, left and right*" to '*it does look like a bit of red and a bit*" which can be considered as a *syntax* model of recursion. The reason for this, is that he noticed the recursive calls which were making the trees to the right and left with different colours.

89. Richard: Can we do it with the maximum value degree like 100?
90. Philip [*interjected*]: It's going to be like an antenna!

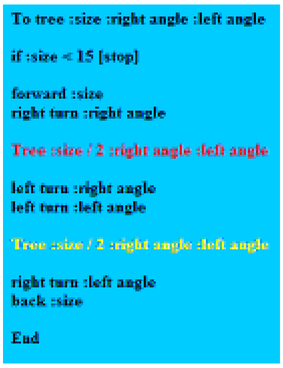
91. Richard: Yeah, that's right, it is like an antenna!
92. My description: [*see the output of the procedure in Figure 101 (a) and (b)*]



```
100  size
100  right angle
100  left angle
```

Figure 101- (a) The maximum values for the angles to the right and left, (b) the output of the procedure with those initial values

93. Philip: Let's put one on 100 and the other on 80 in the colour mode.
94. Richard: It's a kind of parallel and also in the colour mode you can see which is which.
95. I asked: Can you see any relations between theses shapes and the written procedure on the screen?



```
To tree :size :right angle :left angle

if :size < 15 [stop]

forward :size
right turn :right angle

Tree :size / 2 :right angle :left angle

left turn :right angle
left turn :left angle

Tree :size / 2 :right angle :left angle

right turn :left angle
back :size

End
```

Figure 102- The written embedded recursive procedure in the *blue* strategy, the first and second recursive calls are shown by red and yellow colour codes respectively

96. Richard: What do you mean?
97. I said: In what order is the turtle doing those orders?

By asking the question above, I was trying to find out what he really thought about the flow of control over the embedded recursive procedure.

98. Richard: Umm, forward size and turn right, then tree size divided by 2. So, this one is red first and then yellow. It's just a kind of spiral.
99. My description: [*he was pointing to the tasks he has already worked with in the Spirals domain*]

100. Richard [*continued*]: It's just doing a movement and then reducing the size, and then doing a movement and a turn and reducing the size, moving and a turn, reducing the size.

Richard's remark on line 100 showed that although his explanation was far from a delegatory control passing mechanism, it can be considered as a *return-value* model. Moreover, he paid more attention to the execution of the procedure in the colour mode which can be seen in line 101. In that line, Richard pointed to the little box on the screen which was contrived to show the length of the stem which was being draw by the turtle at each step. He also mentioned that "[…] *when it's shooting the yellow part*" (line 101), which evidenced that the AVDA visualisation in the *blue* strategy opened his eyes to the creation of a new tree in different directions by two red or yellow colours.

101. Richard: When it's shooting the yellow part, the box is yellow and when it's shooting the red the box is red. Is that right?
102. I said: What do you think Philip?
103. Philip: Yes, I think so. And also the number in the box shows the length of the stem each time!

Richard also pointed to the shadow turtles and described the relationship between their background colour and the colour of the stems as follows.

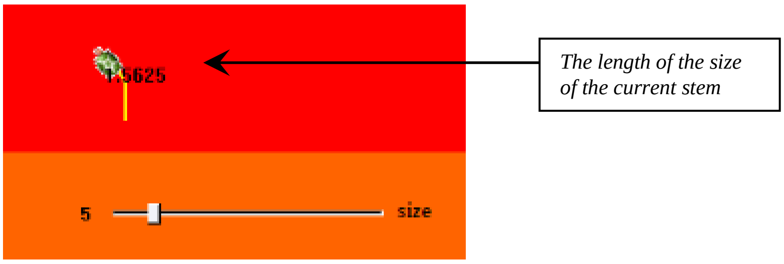
104. Richard: Yeah, and also for those things when the background is red and the tree shape is yellow, it's doing yellow, and when the background colour is yellow and tree shape is red the turtle is drawing red.

At this stage, we moved on to the *red* strategy in which the AVDA visualization was mainly designed to represent the copies of the original procedure. Richard's first idea was to compare it with the AVDA visualisation in the *Spirals* domain.

105. Richard: Oh, this is actually like we were doing with the old spirals in steps, step by step. Because this is ‘n’ less than 1 and the other one is less than 5, it’s going to be a lot more precise! It’s going to take longer.

106. My description: [*he was comparing the stopping condition in the red and blue strategies in the Treebuilder domain*]

107. Richard [*continued*]: So, it’s like doing the first 4 stages automatically and now ‘n’ is 1.56 and it’s still going



The length of the size of the current stem

Figure 103- Richard was pointing to the number moving alongside the turtle to show the length of the stem which was being drawn by the turtle

Richard and Philip kept clicking on the *continue* button and thoughtfully observed the output.

108. I asked: Can you explain what is happening there?

109. Richard: Each time you press *continue* it gets as far down the algorithm as it can and then it goes back again and then starts from the next branch

110. Philip [*interjected*]: If ‘n’ is less than 1, it goes back.

111. Richard [*continued*]: Yeah, if it stops it means ‘n’ is less than 1. Because that one [*he was pointing to Figure 103*] went through like one and a half times, so the tree is ‘n’ over 2, so now it’s going to go back because ‘n’ is 1.56 divided by 2, so ‘n’ is smaller than 1!

Richard and Philip’s remarks in lines 108-111 shows, a significant change in their thinking about the embedded recursive procedure. Initially, they thought about the procedure as a loop, but after their experiences with the AVDA visualisation, they appreciated the process of flow as a back and forward mechanism based on the size of the stems and the stopping condition. So, this can be considered as evidence for possession of a *quasi-copies* model.

However, I was not sure what they thought about the functioning of the recursive call itself. So I asked about it.

> 112. I asked: Can you explain more about the line ‘*tree ‘n’ over 2*’ in the program?
> 113. Philip [*immediately answered*]: Defining and redefining ‘n’
> 114. Richard: Yeah, it’s like defining the length of the next stage as half of the length of the previous one! It’s like the spirals, because that was ‘n; over 1.1 so it is ‘n; over a bigger number.

Philip’s remark in line 113 and Richard’s confirmation in line 114 show that they consider the recursive call to be the generator of the new values ‘n’ over 2 each time. Combining these interpretations with their descriptions in lines 110-111 in which they mention that soon after reaching the stopping condition the procedure goes back and starts another branch, evidenced their possession of a *quasi-copies* model of the embedded recursion.

> 115. I asked: What part or parts of the procedure was the most challenging to work with and to understand?
> 116. Richard: I do get the basic concept of it. I mean it’s quite easy to follow when it’s here, but explaining what its doing is quite difficult!

Richard’s remark in line 116 shows that he had a major difficulty in understanding the complicated delegatory control passing process and the functioning of the recursive calls in the embedded recursive procedure. Therefore, Richard’s evolution of a mental model for the concept of recursion can be shown as follows.

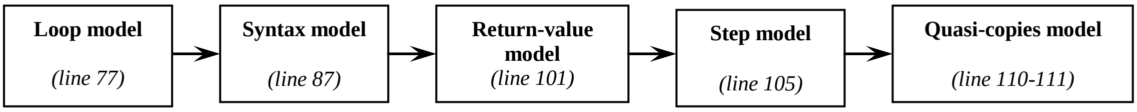
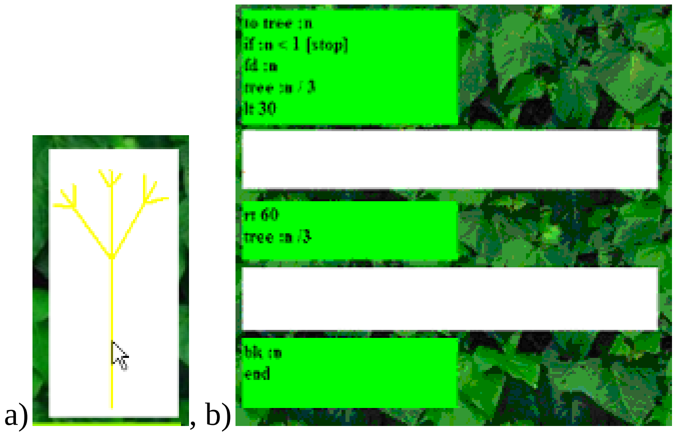


Figure 104- Richard and Philip’s evolution of mental models of recursion after their experience with the first three modules of the *Treebuilder* domain

They moved on to the *your tree* module, which was the last part of the *Treebuilder* domain. As mentioned before, in this model they were asked to complete an incomplete embedded recursive procedure to produce a ternary tree.



```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30

rt 60
tree :n /3

bk :n
end
```

Figure 105- (a) the image of the ternary tree at the bottom right corner of the main interface of the *your tree* module in the *Treebuilder* domain, (b) the given commands and two empty boxes to be filled by them

117. Richard: Is it one instruction per box?
118. I said: Not really, you can enter as many instructions as you want in each one of those boxes.
119. Richard: So, it starts here,
120. My description: [*he was pointing to the slider on the bottom of the screen showing the initial value of the size of the first stem as 'n' and also at the same time he was comparing the commands which were given with the image of the ternary tree*]

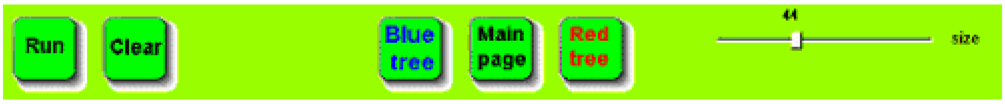
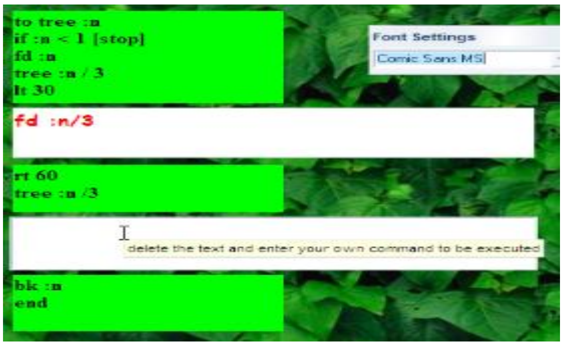


Figure 106- Richard was pointing to the slider showing the initial value of the size

121. Richard [*continued*]: And it goes forward 'n' amount to there and then it goes left 30, and then it's going to go [*paused for a while and continued*] oh , hang on, because the others have two branches and this one has three and each of those angles in there would be 30 and 30 so it goes
122. Philip [*interjected*]: left 30,
123. Richard [*continued*]: Left 30, and then right 60, oh right 60 is already there, so it goes,
124. Philip [*interjected*]: It needs to go forward as well

125. Richard [*continued*]: Yeah, so, that would be, can we just type in something like forward ‘n’ divided by 3 and then it makes it smaller



```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30
```

```
fd :n/3
```

```
rt 60
tree :n /3
```

delete the text and enter your own command to be executed

```
bk :n
end
```

Figure 107- Richard entered the command ‘forward ‘n’ divided by three’ into the first box

His activity, recorded in line 121 showed that he knew what he needed is to create a branch in the middle after placing a stem to the left. The reason that he put ‘*forward ‘n’ divided by 3*’ into the first box was that he did not have any idea of the functioning of the recursive call.

126. Richard [*continued*]: And then it makes it smaller again. What do we need to do after? Basically, what you want to do? You want to go left 30, yeah that’s fine, then you want to really go right 30 because left 30 and then right 60 is doing a sort of like left 30. Left 30 is doing this from here

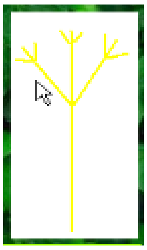


Figure 108- Richard was pointing to the first stem into the left direction

127. My description: [*he was pointing to the first branching point on the image of the ternary tree and moving along the next stem on the left*]

128. Richard [*continued*]: and then from there you go on, right 60 is going to do this.

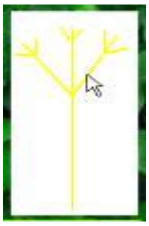
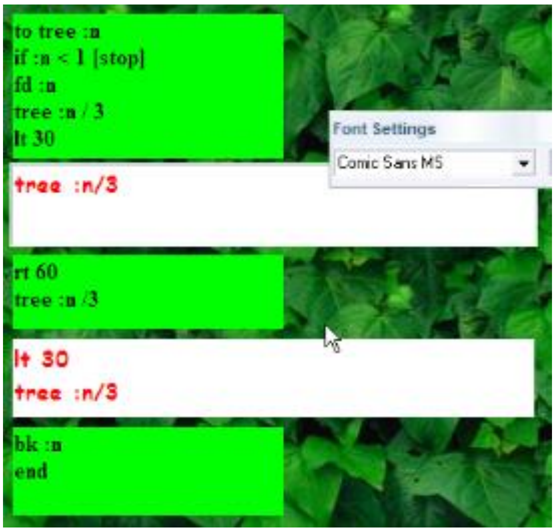


Figure 109- Richard was pointing to the stem in the right direction

129. Philip [*interjected*]: So, you want to do left 30 and then change it?
130. Richard [*continued*]: Yeah,
131. My description: [*they removed forward ‘n’ divided by 3 from the first box and then typed left 30 and tree ‘n’ by 3 into the second box*].
132. Richard: So, we want to go left 30 here.
133. Philip [*interjected*]: And then tree ‘n’ divided by three
134. Richard: Yeah.



```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30
```

```
tree :n/3
```

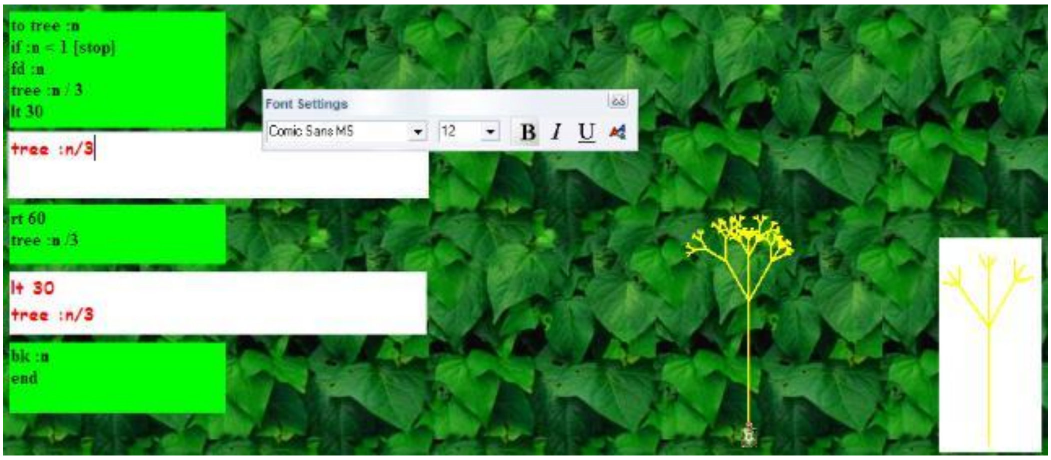
```
rt 60
tree :n /3
```

```
lt 30
tree :n/3
```

```
bk :n
end
```

Figure 110- Richard and Philip’s second attempt at completing the procedure

135. Richard [*pointed to the first empty box and said*]: So, what do we put in here? Do we need a tree ‘n’ by 3 in there? We have got forward ‘n’, tree ‘n’ divided by 3 and then left 30 and then another tree ‘n’ divided by 3, and then right tree divide by tree, left tree divide by three
136. My description: [*then they put tree ‘n; divided by 3 into the first empty box*]
137. Richard: And what is that size value going to be? I don’t want to put a big number, let’s have 60.



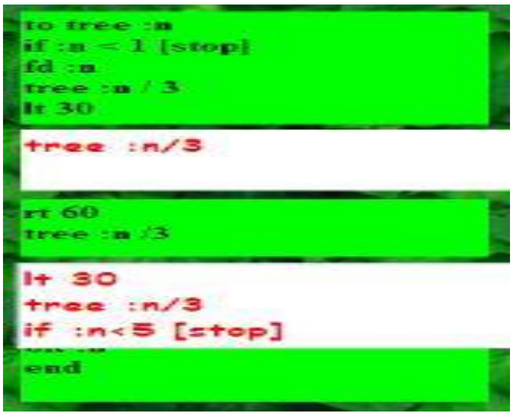
```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30
tree :n/3

rt 60
tree :n /3
lt 30
tree :n/3
bk :n
end
```

Figure 111- The output of the amended recursive procedure

138. Richard: Pretty close! And also once you have done three repetitions you want to stop!
139. Philip: Yeah, because ‘n’ is going to be smaller than one.

Richard’s and Philip’s comments in lines 126-139 showed that the AVDA environment provided them a window in which, they were able to observe the ternary tree that they were asked to make and also see the output of their attempt to amend and complete the incomplete procedure. Richard’s remark on line 135 shows a significant change in his thinking about the procedure. This line showed two important things; first, it showed that Richard’s interpretation of the embedded recursive procedure was procedural (loop model). This was evident when he said “*[w]e have got forward ‘n’, tree ‘n’ divided by 3 and then left 30 and then another tree ‘n’ divided by 3*” (line 135). The second one, which is of importance as well, is that this loop-wise thinking about the embedded procedure is totally different. It can be considered to be an advanced form of procedural thinking about the procedure, which was built in his mind in the AVDA environment. The reason was clear in his remarks when he said “[…] *then left 30 and then another tree ‘n’ divided by 3, and then right tree* *divide by tree, left tree divide by tree* ” (line 135). For Richard to create a new stem in a new direction, he needed to input the term ‘*tree ‘n’ divided by 3*’. His explanation in lines 140-144 below provides more evidence.

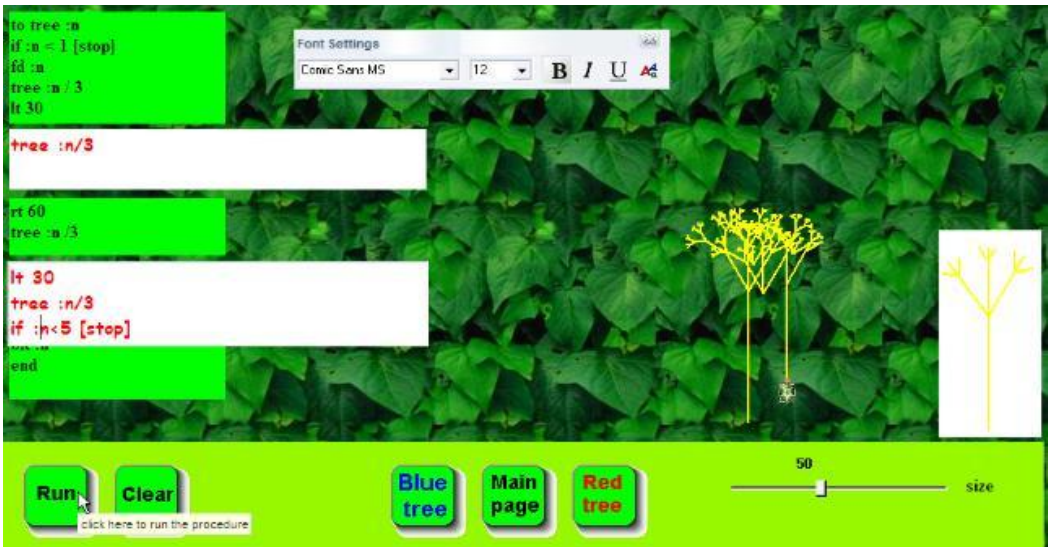
140. I asked: Can you explain to me why you put *tree ‘n’ divided by 3* into those empty boxes?
141. Richard: Well, I thought it’s, like, for each, like, branch, you want to make it a third of the size of the previous one.
142. I asked: For example, what if you wanted to have a tree with 4 branches?
143. Richard [*immediately responded*]: You would need to have it 4 times! But we also need [*paused*]
144. My description: [*he paused a while and started to put a stopping condition into the second box*].
145. Richard [*continued*]: It might be repeated but we want it, umm, so we want, like, if ‘n’ is less than 5 stop really!



```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30
tree :n/3

rt 60
tree :n /3
lt 30
tree :n/3
if :n<5 [stop]
end
```

Figure 112- Richard added a new stopping condition into the second box

146. Richard: So, what happens then? It makes it a little bit too tall as well. Let’s take ‘n’ as 50



```
to tree :n
if :n < 1 [stop]
fd :n
tree :n / 3
lt 30
```

```
tree :n/3
```

```
rt 60
tree :n /3
```

```
lt 30
tree :n/3
if :n<5 [stop]
```

```
end
```

Figure 113- The output of having one more stopping condition in the second box

147. Richard: Oh that's interesting! I don't know! It is kind of seems you've already got it!
148. My description: [*he was pointing to the tree 'n' divided by 3 command that he had typed into the first box and said that*]
149. Richard: It just seems that you have got the forward and then you are reducing the amount by a third of the amount that you moved.
150. Philip [*interjected*]: This 'n' in the next step for the next one is going to be a third of 'n' and then left 30.
151. Richard: Yeah, you go forward and then you reduce the size of 'n' and then turn left.

Richard's and Philip's comments in lines 145-151 show that they have had an inadequate knowledge of the functioning of the recursive calls in the embedded recursive calls. In line 145, Richard decided to add one more stopping condition into the second box which was enough evidence to show that he did not have a clear image of the process of flow of control over the procedure. Lines 145 and 147 show that the AVDA visualisation caused Richard to say that putting a stopping condition might be a repetition of something we have already got in the procedure "[…] *it might be repeated but we want it, umm, so we want like if 'n' is less than 5 stop*" (line 145).

Richard and Phillip did not show any sign of appreciating the complicated delegatory control passing mechanism in the embedded recursive procedures. However, they developed their initial *loop* model of the procedure to a *step* model and *return-value* model by working at the window of the AVDA domain of abstraction.

7.2.3. George & Peter's account on the *Treebuilder* domain

Similar to the other two accounts, George and Peter started their work with the modules of the *Treebuilder* domain with the *making a forest* module. They typed the term *tree* followed by a few different numbers and observed the output of them on the screen. I asked them about the structure of the trees that they made.

152. George: Tree 200 is bigger. So, this one is twice as big as the previous one.
153. My description: [*he was pointing to the main trunk of the tree and comparing the first stem of it*]
154. Peter [*interjected*]: Probably!
155. George: Maybe, I don't know.
156. Peter: Let's try something in between.
157. My description: [*then they tried tree 150*]
158. I asked: What are you thinking when you type tree?
159. George: So, this obviously knows the program. Knows what tree means. There is program knows that what tree means and then 55 is just …
160. Peter [*interjected*]: That's the length.
161. George [*continued*]: Yeah, so then I mean it goes up and in, up and in, until getting very small little tiny branches
162. I asked: Can you see any relationships between them and the previous tasks you have done?
163. George: So that would be like spirals. So, what would that be like? It keeps draw this particular line.

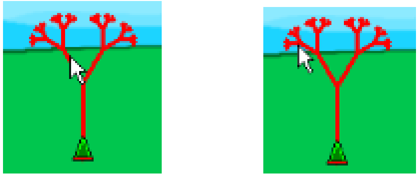


Figure 114- George was pointing to the main trunk and the stems which were drawing to the left direction

164. George [*continued*]: Turns left, draws this particular line, turns left, draws this particular line. So, this bit looks a bit like a spiral

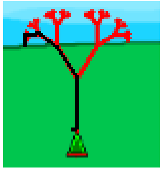


Figure 115- The spiral-like part that George saw in the structure of the tree is shown by the black colour in the above image

165. George: Ok, then it is going around and then
166. Peter [*interjected*]: Then for a certain value comes back on itself!
167. George: And then draws the other of those tiny branches.
168. Peter: This side gets to a certain value and then because they will have the 'n' as less than one, and then it does reverse of the process and then restarts.
169. George [*interjected*]: and it comes back on itself and then do another thing and then come back on itself and do another and again

George's and Peter's remarks in the above lines show that they saw the recursive structure from a functioning perspective in the binary trees pictures. Lines 153-161 show that they appreciated that the number that they typed after the term *tree* can be taken as the initial value of the trunk. Also, they noticed the stopping condition for the procedure; George's remark on line 161 can be considered as evidence for it, when he said that, "[…] *it goes up and in, up and in, until getting very small little tiny branches*".

Then we moved on to the second module of the *Treebuilder* domain in which the students were asked to work with an embedded recursive procedure to generate a binary tree in the AVDA environment combined with the red and yellow colour codes, respectively for the first and the second recursive calls in accordance with the branches to the right and left. George ran the procedure in the normal mode.

170. George: Ok, so let's click the *run* button, oh! That is a tree and just a bit lopsided because the angles are very different
171. My description: [*George changed the values of the angles to the left and right and switched the mode of execution onto colour mode. He was surprised by the shadow turtles which were moving alongside with the main turtle*]

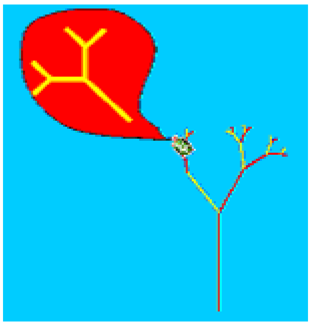
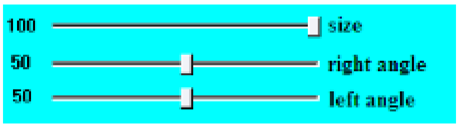


Figure 116- The shadow turtle shows that the main turtle is going to boost new branches to the left with the colour yellow

172. George [*continued*]: Oh, what's he up to?
173. Peter [*interjected*]: What is he doing?
174. George: Red yellow, red yellow, red yellow.
175. Peter: Colouring!
176. George: Red yellow, red yellow, Ok, so if size less than 5 stops. Let us we put these two the same, so it makes it look like the last one in the previous task.
177. My description: [*then he changed the values of the angles to the left and right to 50*]



100 size
50 right angle
50 left angle

Figure 117- The angles to the left and right changed to 50 by George.

178. George [*continued*]: And then it does a little spiral and then it sort of comes back on itself a little bit and then back on itself to make another spiral and then if size less than 5 stop. Otherwise, forward size, right turn, tree size.
179. Peter [*interjected*]: and it does 2 branches.
180. George [*continued*]: Size divided by 2, right angle, left turn 50, and then switches to the yellow part. Ok , and then one more
181. My description: [*he was counting the number of branches as they were being drawn by the turtle*].
182. George [*continued*]: and then goes to stop. So, it does it again, it's, like, running itself within itself again.

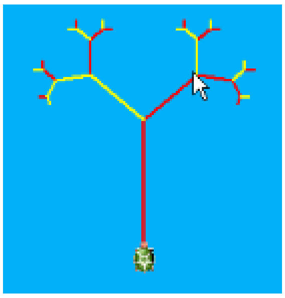


Figure 118- The image of the tree that George made by making both angles to the left and right equal to 50

Experiencing working with the AVDA visualisation combined with the yellow and red colour codes, plus previous experiments with the *Spirals* domain allowed George and Peter to shape their thinking about the embedded recursive procedure. The way that they described the procedure was by simultaneous examination of the written program and the output which was being drawn by the turtle. Lines 178-182 show that they clearly noticed that the procedure was calling itself. In line 178, George described the delegatory flow over the first recursive call and then Peter in line 179 mentioned that, "[…] *it does two branches*", which shows that he was pointing to the second recursive call. Peter's remark in line 179 followed George's description of the second recursive call in line 182 when he said "[…] *and then it goes to stop. So, it does it again, it is like running itself within itself again*".

These comments show their possession of a *quasi-copies* model of recursion.

They continued with the *blue* strategy and further illuminated their thinking.

183. Peter: And then it keeps going.
184. George: Yeah, it keeps going until it gets to 3.125.
185. Peter [*interjected*]: Yes.
186. George [*continued*]: And then left turn and left turn, and then tree size divided by 2, right angle, left angle. And it redraws these bits.

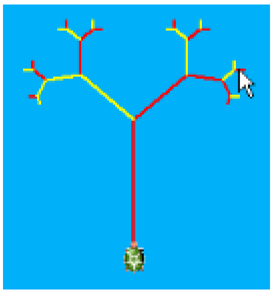
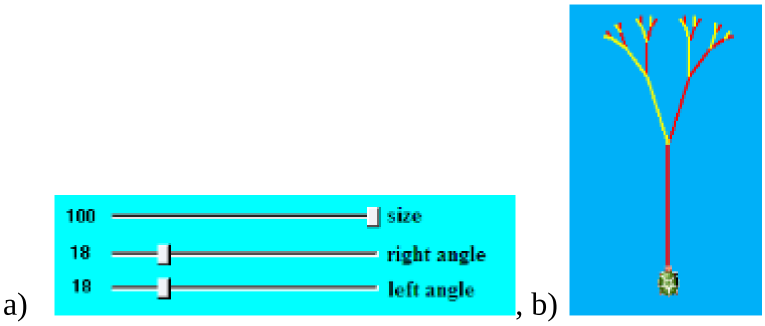


Figure 119- George is pointing to the little stems on the right side of the tree

187. George: If you changed these angles it would be a bit lopsided. … if you have big angles then you've got a bit of a fat tree, and if you have a small numbers like 18, then we've got a poor tree.



100 size
18 right angle
18 left angle

Figure 120- (a) angles to the right and left were chosen 18, (b) shows the final output of the tree

188. George [*continued*]: And then eventually it does right turn, back size, it just eventually reverses itself, eventually back on here, reverse on itself, back on here, and it does another 1 and then one of those over here and then stops, I guess.
189. Peter [*agreed*]: Yeah.

I wanted to see what, the effect of those colour codes was on the way that they shaped their thinking and tracked the control passing mechanism over the procedure. So I asked them about it.

190. I asked: Can you see any relationship between the colours in the procedure and in the tree?
191. My description: [*George was moving the mouse alongside the stems and said*]
192. George: You have got like a red, so red goes up to the right, and then another red up to the right, and here you have got a yellow that goes up to the left, and then again yellow goes up to the left, and then these little branches at the end, yellow and red, yellow and red.
193. Peter [*interjected*]: Then right turn,
194. George [*continued*]: I suppose, because the yellow goes up to the left. So, it does a left turn and then it does a yellow tree and then a right turn and then a red tree. I don't know!

Then they switched their attention to the little box on the middle of the screen which showed the current size of the branch that was being drawn by the turtle

The current value of size is: 3.125

and added that.

195. George: So, it's the current value of the size. And it doesn't get any less than 5!
196. Peter: It was 100 and then 50 and then 25 and 12.5 and 6.25 and then 3.125 and then stop!

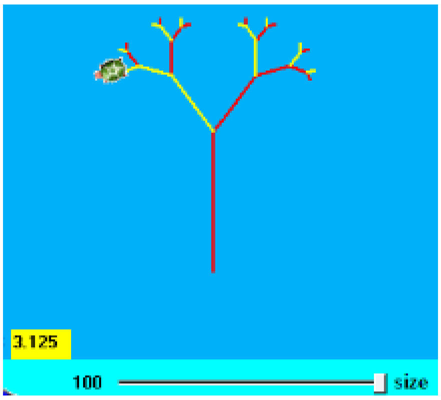
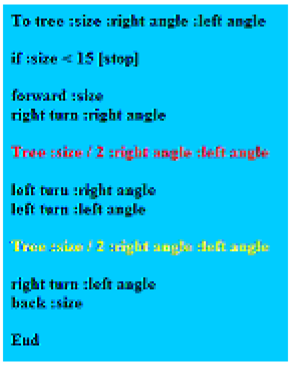


Figure 121- Peter was pointing to the size of the stems until they got less than 5 and stop

197. My description: [*George changed the initial value for 'n' from 100 to 70 and added that*]

198. George: Forward size right turn tree size divided by 2, and then forward half size right turn, and then forward half size right turn, and if size less than 5 stops.

199. My description: [*then he moved on to the commands after the first recursive call*]



```
To tree :size :right angle :left angle

if :size < 15 [stop]

forward :size
right turn :right angle

Tree :size / 2 :right angle :left angle

left turu :right angle
left turn :left angle

Tree :size / 2 :right angle :left angle

right turn :left angle
back :size

Eud
```

Figure 122- George was pointing to the commands after the first recursive call (the red line) after it got to its stopping condition, and he appreciated that the procedure was resuming those suspended commands

George and Peter's remarks in lines 196-199, and especially George's comment in line 198, reveal that the AVDA visualisation allowed them to track the delegatory control passing mechanism for the first recursive call. However, lines 200-205 show that they did not have a clear understanding of this mechanism when it came to the interrelations between two recursive calls. George's remarks in lines 200 and 205 show that he considered the delegatory control passing mechanism for the second recursive to be totally separate from the callings of the first recursive inside it. These comments again support possession of a *quasi-copies* model of embedded recursion by them. George's explanation in line 200 shows that, for him, when the second recursive call got to its stopping condition, it went to do the last 2 commands of the procedure – *right turn and back size*. This means that although on the shape of the tree with the yellow and red branches he described the complicated control passing, he was not able to make a bridge between two recursive calls on the program.

200. George [*continued*]: So, then left turn and left turn, and it does yellow tree until it gets less than 5 and then right turn and back

201. Peter: Yeah, Ok.

202. George: So, then it runs it until it is too small and then it starts off again but not as big as it starts off with another 100.

203. Peter [*interjected*]: It starts off with another half again and then half again.

204. George [*continued*]: So, it is like it reverses back to say here

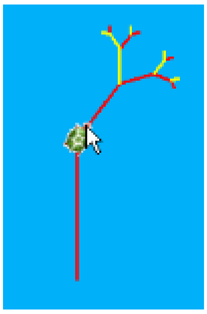


Figure 123- George was pointing to the first branching point as the turtle was about to start drawing the yellow branches to the left

205. George [*continued*]: So it has gone back 50, it goes up 50 and then back and then forward and then eventually it has done all it can do, and reverses itself back down to end.

206. I asked: Which part of the procedure was more challenging to work with?

207. George: In terms of writing or understanding?

208. I said: Explain both please!

209. George: I don't know! Once it has done the last part here

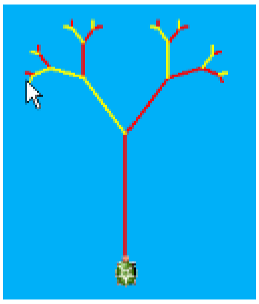


Figure 124- George was pointing to the last stem which was drawn by the turtle with the yellow colour to the left

210. George [*continued*]: Then it starts to draw all over itself again. Once it is there it reverse back on itself and starts again.

211. My description: [*George was pointing to the first few red branches to the left and added that*]

212. George: I suppose, when it does this it does red almost like a red spiral.

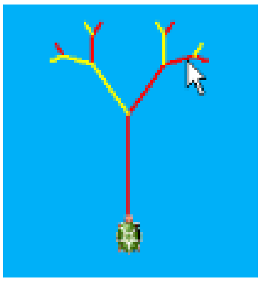


Figure 125- George was pointing to the first red branches to the left and stated that they look like a spiral

213. George [*continued*]: Then it goes back and then it goes yellow and it does like redraw itself, it does yellow and then a red and it does a red and it does a little yellow spiral to the right again starting there

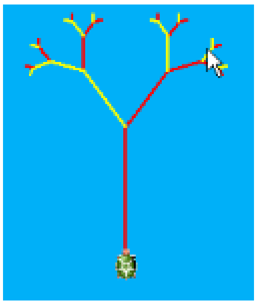


Figure 126- George was pointing to the little yellow branches to the right on the top of the red branches

214. George [*continued*]: And then curving around and doing the same thing until it gets to an end!

George and Peter's remarks on lines 195-214 above show that they evolved their mental models of the embedded recursive procedure in the AVDA environment in the *blue* strategy module. However, they also had difficulties in exchanging the control between the two recursive calls.

Then we moved on to the last module of the *Treebuilder* domain, which was the *red* strategy. The first thing that George pointed to was the difference between the stopping conditions in the *red* and the *blue* strategies. They chose to run the procedure in the normal mode first.

215. George: If ‘n’ is less than 1 stop. So, it does it a lot smaller than ‘n’ is less than 5. So, forward ‘n’, which, was nine.

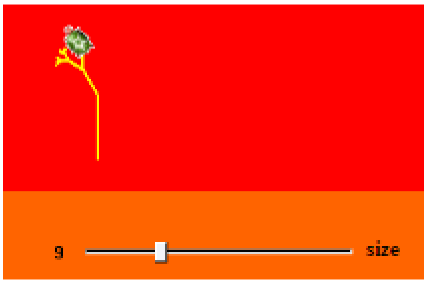
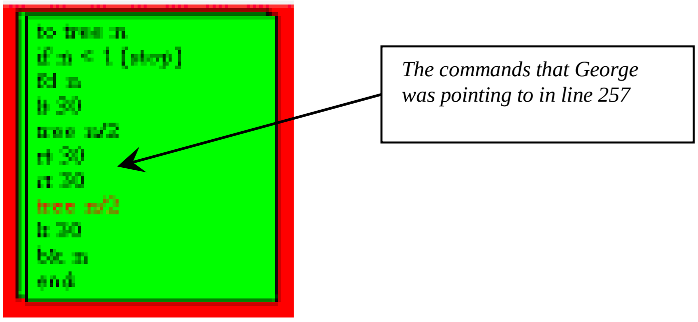


Figure 127- George was pointing to the movements of the turtle in the *red* strategy with the initial value of n = 9

216. George [*continued*]: Left 30 and then do a tree, but run itself with half size and then keep going until ‘n’ is less than 1.



```
to tree :n
if :n < 1 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

The commands that George was pointing to in line 257

Figure 128- The copies of the procedure which were being generated in the AVDA environment in the *red* strategy

217. Peter [*interjected*]: Then it goes back to where it started drawing that branch.

218. George [*continued*]: Right 30, right 30, and then left 30 and then back. And then lots and lots and lots of little ones, until eventually ‘n’ is less than 1.

219. My description: [*George’s above explanation showed that he just followed the commands in a procedural way without paying attention to the delegatory control passing. George changed the value of ‘n’ to 30*].

220. George: So, it keeps going until it gets to ‘n’ is less than 1 again. So you are going to get a little spiral

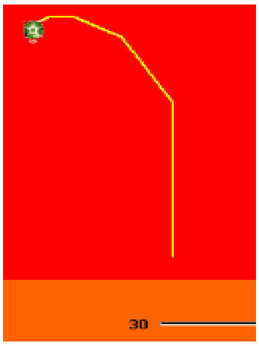


Figure 129- The spiral shape branches that George pointed in the *red* strategy

221. George [*continued*]: So, now it keeps *running* through to the end. Because, here 'n' is 30 and then 15, then 7.5 and here it's doing a little detailed a bit.

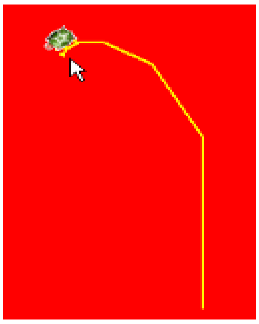


Figure 130- Turtle was drawing the little stems at the end of the branch to the left and George pointed to it as a detailed part of the stem

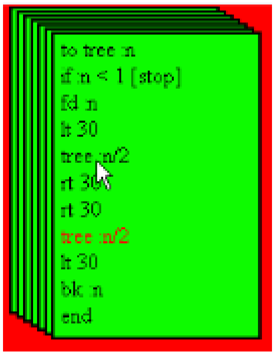
222. George [*continued*]: Probably 'n' is very close to 3 and then one and a bit.
223. Peter [*interjected*]: 'n' is less than 1 and then goes back on itself.
224. George: I think it is a detailed tree and it takes a long time to draw the whole thing. I suppose, with the *blue* one you get the nice tree shape pretty quickly. So here the program sort of runs out a few times and then back on itself a few times, because I suppose it's moving itself back, and then makes a few new ones, and then goes back again, then a few more.

Again, similar to the previous module, George's remark in line 220 shows that he tracked the first recursive call in a delegatory control passing mechanism. The AVDA visualisation in the *red* strategy shows the copies of the original procedure and uses colour code for the recursive calls (each time after reaching the first or second recursive call it was flashing red and was waiting for the user to press the *continue* button). This helped them to change their thinking about the interrelationships between the two recursive calls (lines 221-225).

225. My description: [*George was pointing to the animated visualisation of the* AVDA *over the copies of the original procedure*]
226. I asked: can you see any similarities or differences with the *blue* strategy?
227. George: Well, again it is similar, but the blue one just kept running within itself again.
228. My description: [*they moved back on to the blue strategy and George immediately corrected himself*]
229. George [*continued*]: Oh! No they are the doing the same thing.
230. My description: [*he pointed to the recursive calls in the blue strategy and added that*]
231. George [*continued*]: this running, and rerunning itself again, tree and tree. That's why you don't see the windows that are changing.

That was an interesting remark that George made in the AVDA visualisation by correlating the windows of the new copies of the original procedure in the red strategy with the colour coding of the recursive calls in the *blue* strategy. The other thing is that in lines 227-231 George stated that the procedures in the *blue* and *red* strategies are the same. In the following lines George & Peter were trying to work out the control passing mechanism of the procedure. For instance, in the line 234, they were tracking the angle and direction in which the turtle was heading.

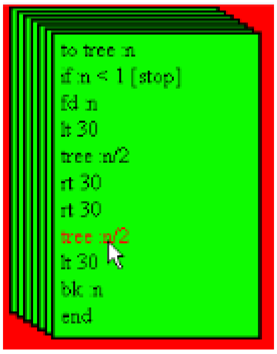
232. Peter: Yes, it is working, so every time it gets to pop up another one and then back again.
233. George: So, you see, that stage flashed up and kept going.



```
to tree :n
if :n < 1 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

Figure 131- When the procedure was doing the second recursive call it is shown that the first recursive call is also called and it helped George to point to it in the above image

234. George [*continued*]: So, now to the right 30, right 30, and now it’s going to start from here, I suppose.



```
to tree :n
if :n < 1 [stop]
fd :n
lt 30
tree :n/2
rt 30
rt 30
tree :n/2
lt 30
bk :n
end
```

Figure 132- As George mentioned, the procedure was calling the second recursive call and he pointed to it in this image

The animative visualisation employed in the AVDA significantly assisted the students to improve their thinking about the control passing mechanism. In the lines 239 and 242, George & Peter noticed that the process goes back on itself.

235. George: And do this and then back,
236. Peter [*interjected*]: Back to where it was left before.
237. George [*continued*]: Yeah, it was less than 1. Let’s hide the turtle to see the little ones.
238. My description: [*he hid the turtle and added that*]
239. George: you see what it’s doing, you see the little lines just appearing and then a few times they are going in different directions. I suppose in a minute it will come back to here.

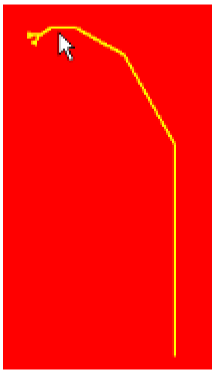


Figure 133- The position in which George predicted that the new stems would be drawn by the turtle

240. George [*continued*]: Yeah, here we go!

241. Peter: And it will go half and will turn left.

242. George: And then again and again and again and then it's going back on itself, you can see what it's doing and then it did go back, back, back, so now would it be here?

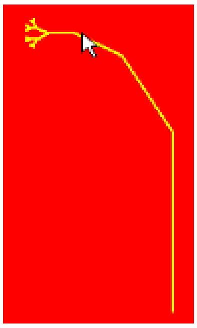
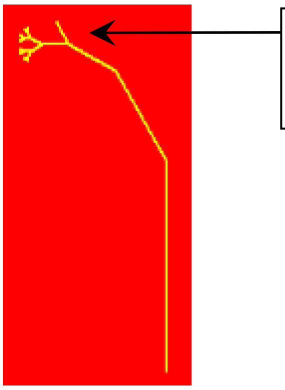


Figure 134- George was pointing to the branching point where he thought the next branch would be drawn

243. George [*continued*]: Yeah it is here.



The new branch just appeared as George predicted in line 282

Figure 135- The branch was drawn by the turtle from the branching point that George predicted

244. George [*continued*]: So then it starts off another one of these.

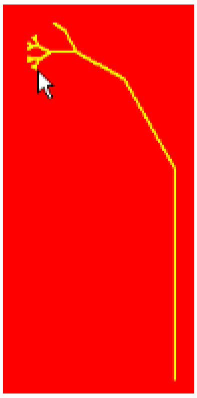


Figure 136- George was pointing to the little stems at one of the end points and explained that the same thing is going to be done by the turtle at the other ending point

Lines 232-244 show George's thinking about the complicated control passing in an embedded recursive procedure and the interrelations between the two recursive calls.

The outcomes of George's and Peter's experiments were similar to the results of the other student participants and were examined and recorded in the previous sections. As such, and to avoid repetition, they are not detailed further.

The next section of this chapter focuses on the findings and results of the third iteration with regards to the research questions of this study.

7.3. Findings and Results of the Third Iteration

Having discussed the three accounts of the students who participated in the third iteration, it is now possible to explain the findings and results of the third iteration.

First and foremost, I would like to reiterate the main research questions that the study was designed to address: *How does the recursive thinking of university students evolve through the use of carefully designed digital tools?* To address this question, some related sub-questions were considered. These related questions were designed to address two aspects of the *tool design* and *tool use* of the tools which were designed and tested in the three iterations. From a *tool design* perspective, the most attention was paid to the role of design to reveal the latent layers of recursion such as the complicated mechanism of delegatory control passing. It was of value to find out, to what extent design can form a bridge between the formal and informal (by the modelling of trees and spirals) to support students to shape and evolve their thinking about recursion. From a *tool use* perspective, the focus was on seeing how the modelling of trees and spirals directed students' thinking about recursion and how their engagement with the purposefully designed tools helped them to construct and modify their mental models of recursion.

The students' accounts in the third iteration revealed that the animative visualisation (AVDA), which was designed and contrived in the modules of the domains of abstractions in this research, successfully helped students to shape and evolve their thinking about the concept of recursion. It also bridged the formality and complexity of the interdisciplinary concept of recursion with the informal examples of everyday life analogies (trees and spirals) (See Chapter 7 - PART ONE; lines 11-20, 48-54, 111, 141-143, and also Chapter 7 – PART TWO; lines 43, 101, 108-14, and 165-169).

Working with some fractals like binary and ternary trees and also some fractal-shaped objects like spirals was easy for them and none of the students who participated in the third iteration had a problem with them. This supports the idea of phenomenalizing the concept of recursion using familiar objects.

From the *tool use* perspective, the students' accounts revealed that to create a spiral (which can be made both recursively and iteratively), all of the students who took part in the third iteration would prefer to use the easier and more straightforward iterative algorithm rather than the complicated and redundant recursive algorithm. However, as in the case of Richard and Philip they mentioned that although they would prefer to use the iterative algorithm, it did seem that the recursive algorithm was more robust and powerful than the iterative one and would probably work better in more complicated situations where the iterative one might fail (See Chapter 7 - PART ONE; lines 48, 53, 102-106, 115-116, 118-120, and 158).

The students' accounts also showed that tracking and understanding of the delegatory control passing mechanism is the most problematic and difficult task in understanding the recursive procedures (both tail and embedded). However, the AVDA visualisation contrived into the modules and tasks of the *Spirals* and *Treebuilder* domains helped them to shape and change their thinking about recursion. The results also revealed that the prior experience with the tail recursive procedures helped students to work with the embedded ones. However, understanding the mechanism of the delegatory control passing between the two recursive calls was still problematic for them. The latter shows that increasing the number of recursive calls makes understanding and tracking the flow of control a very hard task for the students (See Chapter 7 - PART ONE; lines 14, 21, 28-33, 36, 38, 40-45, 63-68, 76-77, 83, 133-135 and also Chapter 7 - PART TWO; lines 29, 33-38, 44-45, 72, 77, 84, 198-111, 196-199, and 231).

Some tools facilitated the students' appreciation of the indispensable components of the recursive procedure (the stopping condition and recursive calls) (See Chapter 7 – PART INE; lines 28, 38, 43, 55, 87, 90, 107-109, 111-113, 135-137, and Chapter 7 – PART TWO; lines 8-12, 17, 145, 147, and 161).

The students' accounts of the re-considerations of the *Spirals* domain and the *Treebuilder* modules in the third iteration also revealed the following pattern for the evolution of their mental models for the (tail and recursive) recursive procedures.

a) Before their experience with the AVDA visualisation:

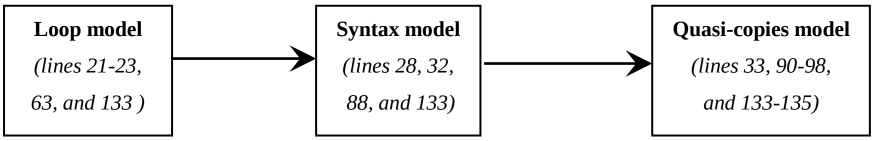


Figure 137-The pattern for the evolution of the students' mental models of recursive procedures in the third iteration before using AVDA

b) After their experience with the AVDA visualisation:

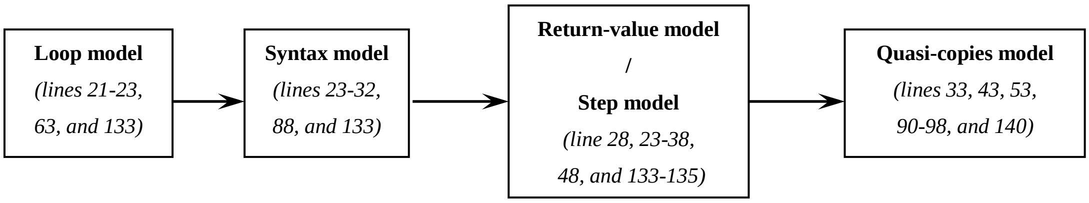

Loop model
(lines 21-23, 63, and 133)
Syntax model
(lines 23-32, 88, and 133)
Return-value model / Step model
(line 28, 23-38, 48, and 133-135)
Quasi-copies model
(lines 33, 43, 53, 90-98, and 140)

Figure 138-The pattern for the evolution of the students' mental model of recursive procedures in the third iteration after using AVDA

7.4. Summary

In summary, this chapter, which is divided into two major parts, forms a continuation of Chapter Six, which mainly concentrated on the *tool design* aspect of the *Treebuilder* domain modules. The first part of chapter seven, discusses the *tool use* of the three students' accounts of the tasks of the *Spirals* domain. These three accounts were chosen as they clearly represented the rest of the students who participated in the third iteration. The second part of the chapter mainly focused on the *tool use* of the same students' accounts of the modules of the *Treebuilder* domain. Those two parts were followed by the results and findings of the third iteration with regards to the research questions that the study was designed to address. Based on the results and findings of the third iteration, a pattern for the evolution of the students' mental models of revolution was also suggested.

The next chapter of the thesis focuses on the final discussion and conclusion of this study.

8. Discussion and Conclusion

8.1. Overview

This chapter discusses the major findings of the research and is divided into two parts. The first part of the chapter presents a summary of the findings of the study. The second concentrates on the importance of the findings, and their relation to prior research conducted in this field. Two perspectives form the basis of this discussion and they are the *knowledge of the concept of recursion (tool use)* and the *design constructs (tool design)* of phenomenalization of the concept of recursion. As the chapter continues the limitations of the study are considered. Then, the possible future developments, and pedagogic suggestions regarding the concept of recursion are discussed. It culminates with a detailed record of my reflections.

8.2. Summary of major findings of the research

My research and the inventing of AVDA *(Animative Visualisation Domain of Abstraction)* enabled me to study and analyse students' *thinking-in-change* and to see how they understand and apply recursion. Principally, my focus was on examining the way that students develop and shape their mental models of recursion. The AVDA visualisation approach supported this as it sheds light on new insights for designing domains of abstraction to introduce and present mathematical concepts, particularly, the concept of recursion.

8.2.1. Students’ Knowledge of Recursion (Thinking-in-change)

The main findings of my research in the AVDA visualisation environment with regards to the students’ knowledge of recursion can be explained as follows.

Firstly, the students’ results confirmed that, due to the complicated control passing mechanism of recursion, the concept is a difficult concept to teach. Also, due to the inherent complexity of recursion, it can be categorised as a hard concept for students to understand and apply in their problem-solving activities. The results of this research illustrates that the AVDA approach suggests important pedagogical issues to diminish the innate complexities of recursion by employing an animative visualization to reveal the hidden parts of the concept. The results also brought to light the fact that the students’ major problem in understanding and applying recursion was recognising and mastering the complicated mechanism of flow of control in the recursive procedures. This complicated control passing mechanism is here referred to as *Delegatory Control Passing* (*DCP*). The continuous back and forward passing of control over the procedure and between the recursive call(s) confused students. Further outcomes of the research show that, the AVDA visualisation approach had a significant role in eliminating this problem for the students. By showing new generated copies of the original procedure and using colour-codes, AVDA assists the students to improve their thinking about the concept of recursion. However, the students still experienced difficulty when trying to master delegatory control passing. This was especially apparent in the embedded recursive procedures where it became evident that tracking the flow between the two recursive calls was a difficult task.

Another related finding of the research is *Terminating Confusion* (TC). In the stopping condition, the students had difficulty appreciating the difference between the *STOP* command and *END* command. They were unclear if the stop command in the stopping condition totally ended the execution of the procedure, or whether there were still some steps left to be executed.

The results of the research showed clear stages in how the students' thinking-in-change process developed towards a viable mental model of recursion. In particular, the ways in which students developed from having possession of initial naive models, which were highly influenced by the iterative image of recursion as iteration, towards the more sophisticated models. The AVDA visualisation approach revealed a pattern for this transition from the naive unrefined models towards the more sophisticated models. The evolution of the students' mental models, and the order of their emergence (in the students' thinking-in-change) within the AVDA environment, is shown in the below figure (See figures: 76 (George & Peter), 68 and 86 (Simon), 104 (Richard & Philip), and figures 137 & 138).

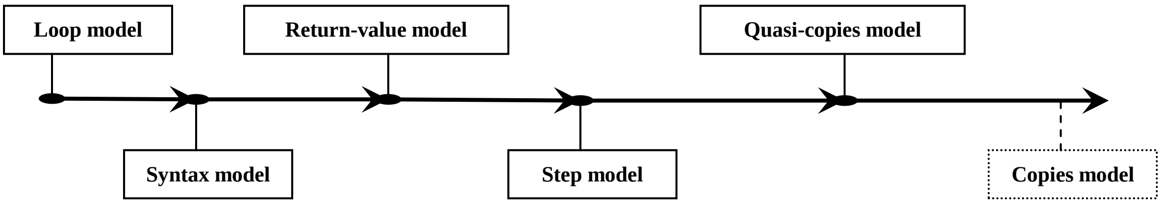


Figure 139- The developing path of evolution of students' mental models of recursion

A related problem that has been revealed by the results of my research is the students' ability to understand and distinguish the differences between *recursion* and *iteration* (*RI*). Students had a strong tendency to conceive of recursion as the familiar iteration concept. However, the results showed that the AVDA visualisation approach played a significant role to improve the students' understanding of recursion by providing visual pictures of the control passing mechanisms in recursion and iteration

However, the above description of thinking-in-change about recursion in Figure 139 is too simple. There are much more complex aspects of the process of thinking-in-change about recursion that are difficult to represent in a basic diagram. Considering functional abstraction and the difference between *what* recursion achieves and *how* it will be operationalised raises the following complication for the students' evolution of mental models. From the functionality perspective of functional abstraction, the students either focused on making sense of the visual output of the procedure or on the semantics of the programming statements. Whereas, from the functioning perspective, they either traced the complex pattern that the turtle took or tracked the statements in the code step by step as they were executed. The students' focus of attention when they were interacting with the AVDA visualisation environment is here referred as the *Mental Compiling Process* (MCP).

8.2.2. Design constructs

In this section, attention is turned to the design issues by considering the influence of the AVDA visualisation approach on thinking-in-change.

The results revealed that the AVDA visualisation approach offered an appropriate situation or environment to bridge formal (the concept of recursion) with informal fractal-shaped objects. For the purpose of this research, I employed spirals and trees as the fractal-shaped objects to phenomenalize the concepts of tail and embedded recursion respectively. This visualisation approach permitted the students to develop their own knowledge about recursion in a dynamic environment. Students were able to observe the immediate feedback of their experience on the screen, which fed new conjectures back into the students' knowledge. In this situated environment, they were able to construct, test, modify, re-construct and develop their knowledge about the concept which was being studied.

The AVDA visualisation environment acted as a window, to reveal the latent layers of recursion and concepts with innate complexity. During the research, the concepts of iteration, tail recursion, and embedded recursion were phenomenalized within the AVDA environment. This visualisation environment provided a window for the students, in which they were able to observe the iterative and recursive control passing mechanisms. The design also provided me, the researcher, opportunity to study students' thinking and thinking-in-change about recursion and its component by observing and analysing the students' experiments with the tools.

In this approach the students were able to work with the recursion concept before knowing anything about it. In fact, the AVDA visualisation offered a situation in which the students were able to shape and evolve their knowledge about the concept of recursion before knowing the functioning and functionality of the main parts of it (Power Principle of Papert, 1980).

Finally, the idea of the functional abstraction informed me regarding the *what* and *how* parts of the students' knowledge of recursion and design construct. In other words, the concerns were about the way that the design process presented the concept, and the way that the students shaped their models for the concept by using that phenomenalization.

8.3. Discussion

The aims of my research principally focused on four areas: 1) The role design to support students' understanding of recursion, 2) The role of design to reveal the latent layers of recursion, and bridging formal and informal, 3) The ways that students shape and develop their mental models of recursion, and 4) The role of design in focus of attention on the functionality and functioning principles of elements of recursion. This detailed discussion, based on the findings of the research is divided into two parts: the knowledge of recursion and design constructs.

8.3.1. Students' thinking-in-change in the AVDA

The findings of the thesis ascertain that AVDA plays a significant role in the students' thinking-in-change process about the concept of recursion. The colour codes and animative techniques which were contrived into the *Spirals* and *Treebuilder* domains offered the students the opportunity of engaging and experiencing the tail recursion-iteration and embedded recursion in dynamic environments. The dynamic environment of the second iteration of my research, the *Spirals* domain, assisted the students to improve and develop their thinking and mental models about the tail recursive procedures and its relationships with iterative processes. The third iteration of my research, the *Treebuilder* domain, allowed the students to work with the embedded recursive procedures in a dynamic environment. Even though most of the students, initially considered recursion as iteration, gradually, by engaging with the animative visualisation provided in the AVDA environment, they improved and developed their understanding of recursion to more sophisticated levels (See figure 139). The next sections of this chapter show how this thinking-in-change process took place during the students' experiments with the modules and tasks of those domains.

8.3.2. Recursion: A difficult concept

The results confirm that the concept of recursion is a difficult concept for the researcher to present. It is also an intricate concept for the students to understand and apply in programming and problem-solving situations. One of the challenging parts of designing the domains of abstraction and the AVDA visualisation approach for me as a researcher and also as designer of the domains was finding appropriate phenomenalization of the concept of recursion. . It is a major issue that people have little experience of the concept of recursion in everyday life. Therefore, trees and spirals were chosen as fractal-shaped objects in order to bridge the formal recursion concept with the informal familiar objects.

It is true to say that most people, and in particular the students who participated in this research, have little experience with recursion in their everyday lives. My research supports the idea of researchers such as Minsky (1988) who suggested that it is hard for the human brain to work with recursion (See Chapter 7- PART ONE; line3 53-54). One of the reasons for this is that we hardly use recursion in our everyday life activities. This makes the concept of recursion unfamiliar for our brain. The findings show that, the students either tried to subjugate the recursive calls or used them stereotypically (See Chapter 7 – PART TWO; lines 38-41, 83-88, and 183-189). The AVDA visualisation approach allowed the students to realise and recognise the syntactical differences between recursion and iteration (Chapter 7 – PART ONE; lines 48-54, 111, 121-135, and Chapter 7 – PART TWO; lines 17-19, 152-163, 188-194, and line 101). However, recognition of the complicated mechanism of the recursive procedure was a very hard task for them. For instance, this is clear in Simon' remark about the recursive call: "[…] *and this is a bit I do not understand now!*" (Chapter 7 – PART ONE; line 28, and Chapter 7 – PART TWO; line 116).

The results show that the students were able to see *what* was happening; this is referred to as the functionality principle. But, they still had difficulties in understanding *how* it was being done by the turtle (the functioning principle). In the *Spirals* domain, the AVDA visualisation approach provided them with a situation in which they could compare iterative and tail recursive procedures to uncover the hidden parts of the mechanism of tail recursion by syntactical and semantic exchanges through observing the animative visualisation. It opened a window for them to look into the latent layers of the mechanism of the recursion and the *how* part (see Chapter 7- PART ONE; lines 89-95 and line 109).

It showed how AVDA provided a situation for the students, in which they evolved and changed their thinking about the concept of recursion. That was a significant achievement for the AVDA approach to reveal the hidden parts of the mechanism of recursive calls in recursive procedures. The dialogues of the participating students mentioned in the research showed that they not only recognised *what* the turtle was doing, but they also started to describe *how* it was being done. The results and findings of the *Spirals* domain also ascertained that the students preferred to use a familiar straightforward iterative approach to create their own spiral rather than the complicated recursive approach. They found it to be a quicker and easier way to create the spiral (Chapter 7- PART ONE; lines 53, 115).

The findings of the research in the *Treebuilder* domain also showed that the students were inclined to use procedural and iterative interpretation for the embedded recursive procedures. Therefore, their tendency to use an iterative approach triggered their difficulties in tracking the complicated flow over the whole embedded recursive procedure. This was especially so when the flow was in transition between two recursive calls, they were not able to understand it. (Chapter 7 – PART TWO; lines 63-68, 133-135, and 178-182)

In the next sections of this chapter, the main components of the students' mental models of the concept of recursion are discussed.

8.3.3. Delegatory Control Passing (DCP)

The results revealed that one of the major difficulties for the students in understanding both tail and embedded recursion was appreciation of the complicated control passing mechanism.

The delegatory control passing mechanism was principally based on a declarative way of thinking, which is about the process of a series of suspending and resuming processes within the original procedure. In contrast, the flow of control in a familiar iterative procedure, which is based on an imperative and procedural way of thinking in which, the commands of the procedure are executed one by one from the beginning of the procedure to its end.

The results and findings show that, this complicated control passing mechanism was a big dilemma for the students to understand and track

(Chapter 7 – PART ONE; lines 41-45, 136-139, and Chapter 7 – PART TWO; lines 17-19, 108-114, 172-188, and 188-194). It has been mentioned previously that, in order to avoid the negative burden of the term *passive flow* introduced and used by Kurland and Pea (1985) which does not provide adequate information about the nature of this sort of advanced control passing, it is rather referred to as the *delegatory control passing* mechanism in my study. The term delegatory shows that the control passing mechanism in this case is not only *passive* but it is a generalised form of active flow in a declarative way. The AVDA environment provides a window for students, through which, they were not only able to see what recursion achieves, but, to picture how it is operationalised (see Chapter 7 – PART ONE; lines 73-82, 97-99, and Chapter 7 – PART TWO; 108-114).

8.3.4. Terminating Confusion (TC)

Terminating confusion is also related to having inadequate knowledge about the delegatory control passing mechanism. After calling each recursive call, when the procedure reached its stopping condition, the students were confused, whether the *STOP* command in the stopping condition terminated the whole execution or if there was still something to be done (For instance see Chapter 7 – PART ONE; lines 72-73). However, the students' experiences with the AVDA visualisation allowed them to see that the *STOP* command in the stopping condition was not going to terminate the whole execution as they could see that the animation was still continuing after that (For instance see Chapter 7 – PART ONE; lines 73-79).

8.3.5. Recursion vs. Iteration (RI)

The results show that confusing the concept of recursion with the familiar iteration concept is a very common problem among the students who participated in my study. This problem is principally related to and originates from having inadequate knowledge about the function and functionality of the recursive calls and in particular delegatory control passing mechanism (Chapter 7 – PART ONE; lines 21-32, 63-68, 69-73, 83-85, and 133-135). The AVDA environment provided the students with a dynamic situation, in which, they were able to see some syntactical and semantic differences between the recursion and iteration, and the difference between mechanisms of control passing in those structures (Chapter 7 – PART ONE; lines 40-43, 109-114, 140-142, and 148-151).

8.3.6. Confluence of Tail and Embedded Recursion

The results of the third iteration, the *Treebuilder* domain, show that having prior experience with tail recursion facilitated the students' later experience with the recursive calls in the embedded recursive procedures. However, the students still showed difficulty in understanding and mastering the complicated delegatory control passing between the two recursive calls, tracking the flow when the first recursive call was called, while the second recursive call was being executed, in the embedded recursive procedures (Chapter 7 – PART TWO; lines 6-8, 14, 38-41, 69-72, 105-111, and 178-182).

8.3.7. Functional abstraction

Functional abstraction has a vital role in the design of computer-based tools and in particular the AVDA environment in my research. By phenomenalization of the concept of recursion using fractals and fractal-shaped objects as well as employing some visual techniques such as animation and colour codes, I tried to work on the functioning mechanism. The students were also able to see *what* recursion achieves (functionality) by observing the final product on the screen.

The following sections of this chapter concentrate on discussing the students' mental models of recursion with focus on the functional abstraction perspective. The above mentioned items of this research are also discussed from both functioning and functionality perspectives. This approach equipped me to suggest the integrated mental model of recursion based on both functioning and functionality later on in this chapter.

8.3.8. Mental Model Evolution (MME)

The findings of my research support the previous works on mental models of recursion that were undertaken by Kahney (1984) and Gotschi *et al*, (2003). It was revealed that, all of the students who participated in the third iteration initially possessed the incorrect loop model for the concept of recursion. Kahney (1984) in his seminal work categorised the students' mental model of the concept of recursion. His work was followed and developed by Gotschi *et al*, (2003). My research not only ascertained additional support of their works, but it also extracted a roadmap of the evolution of mental models based on functioning and functionality principles of the elements of a mental model. These elements will be explained later on in this chapter. The results of the study developed upon the previous research on the categorization of the mental models of recursion by introducing a new model for the concept of recursion which is called *quasi-copies* model. The behaviour of those students who possessed quasi-copies model for the tail and embedded recursive procedures can be explained as follows.

a) In tail recursion procedures: They realize that in the tail recursive procedures, they could see the procedure was calling itself within itself. They also could picture how this process of calling itself within itself stops when the procedure reached its stopping condition. However, they were not able to see that the execution of the procedure totally terminates whenever all the already generated copies terminate one after another;

b) In the embedded recursion procedures: They showed one of the cases below:

    a. The same performance as they showed with the tail recursive procedures with regards to the first or second recursive call,

    b. The same performance as they showed with the tail recursive procedures with regards to the first recursive call and ignoring the second recursive call.

The results show that the students' initial mental model of recursion had evolved and developed in the AVDA environment from both functioning and functionality dimensions (see figures: 137 & 138, 104, 86 & 68, and 76).

By using animation and colour codes within the AVDA visualisation environment, the students had the opportunity of experiencing both functioning and functionality aspects of recursion. As it has been mentioned in the above sections, the students improved their thinking about the following components of recursion: 1) confluences of tail & embedded recursion, 2) recursion vs. Iteration, 3) terminating confusion, and 4) delegatory control passing. These issues demonstrate a gradual progressive development in students' thinking about recursion. It is here referred to as the *spiralling process* between the students' interpretations of functioning and functionality principles of the main components of the concept of recursion, which are discussed later on in this chapter.

The *spiralling process* describes how the students begin to work with recursion using a primitive unsophisticated mental model which was shaped and formed based on the students' previous knowledge and beliefs. This model was not necessarily a sustainable and viable model for recursion. Then they debug the possible errors in their model through interaction with the AVDA environment. Thereupon, they form new models, test and amend them and promote their preconceived model to a higher level model. The students' new models are not robust because they are shaped as a result of the interaction with the tasks and modules of the domains of abstraction. These higher level models are playing a transitional role between the students' preconceived initial models and their possible final viable model of recursion. This final durable mental model of recursion is here referred to as an *integrated mental model of recursion*. This integrated mental model is the ultimate point of the spiralling process of the interactions between the functioning and functionality of principles of the concept of recursion, within the AVDA environment. The students' integrated mental model of recursion is formed by putting all the essential elements of those models that they have already possessed or generated, together into a whole.

The next section of this chapter focuses on the main elements of the students' mental models of recursion.

8.3.9. Mental Compiling Process (*MCP*)

The students' focus of attention while they were working in the AVDA visualisation environment is here referred to as the students' mental compiling process (MCP). Some of the students put their whole concentration into the semantics of the procedure commands, whereas others focused holistically on the visual output of the procedure. MCP describes how the students traced the pattern that was provided on the visual output of the AVDA animative visualisation and that was being drawn by the turtle. It also delineates the step by step tracking of the control passing mechanism of the procedures when they were being executed in the AVDA environment. MCP is more about the students' approaches to the concept of recursion from a problem-solving aspect and from both functioning and functionality perspectives (Chapter 7 – PART ONE; line 127, and Chapter 7 – PART TWO; line 207).

8.3.10. Elements of Mental Model of Recursion

Distinguishing functioning and functionality dimensions for some of the aforementioned issues was not an easy task. The connections between functioning and functionality are examined and developed by the students based on their experiments with the components of the concept of recursion throughout working with the tools. The functioning and functionality perspectives can be considered as two ridges of a mountain which has the integrated mental model of the students' about recursion at its apex. That is one of the crucial aims of the design process, to provide appropriate situations and environments, which enables the students to make these connections between the functioning and functionality aspects to support the development of a viable integrated mental model of the concept of recursion. The following table describes the functioning and functionality aspects of each element of mental models of recursion.

	Functionality dimension	Functioning dimension
Visibility **(*RI*)**	- Don’t use recursion in ***everyday life***; - AVDA visualisation approach ***phenomenalizes*** recursion [design].	- ***Mechanism*** of the concept of recursion is ***obscure / hidden***; - AVDA made the workings ***visible*** [design].
DCP	- Student sees the recursive call as a longer process – ***generalized repeat***; - The recursive call ***embodies*** a ***black box*** routine which is in fact a ***smaller version*** of the main procedure.	- Students attend to the ***syntax*** of ***how*** the repeat iterates; - Focus on the ***details of the copies*** of the procedure, their inputs, and their outputs.
MCP	- Aappreciating ***holistically*** the picture produced; - Attending to the ***semantics*** of the programming code.	- Tracing the ***complex pattern*** that the ***turtle takes***; - Tracing ***step by step*** the statements in the codes as they are ***executed.***
TC	- The ***STOP*** in the main procedure influences the ***‘depth’ of the recursion.***	- STOP ***ends*** the ***whole program***; - STOP ***ends*** the ***current copy*** of the procedure and ***returns control*** to the ***copy construction.***

Table 11- A functional abstraction perspective for the findings of this study on the elements of mental models of recursion

As already mentioned, the students’ prior knowledge about the concept of recursion before starting to work with the *Spirals* and *Treebuilder* domains of abstractions, underpinned their initial mental models of the concept of recursion. For the students, this foundation layer of knowledge constitutes their initial conceptualization of the concept of recursion, which entirely belongs to the functionality principle of recursion. The students' preconceived knowledge about the concept of recursion mostly is not a valid and viable model. The students amended and developed their initial model through their engagement with the AVDA. The process of reconstructing and amending the previous knowledge about the concept towards producing more viable and stable models is considered as the functioning perspective of mental models of the concept of recursion. From this standpoint, the students' previous knowledge and their own new knowledge of recursion developed and evolved towards a viable and stable model, within the AVDA environment. These functioning and functionality aspects of the concept of recursion coordinate a durable meaning for the recursion in the minds of the students.

The next picture describes the model of the evolution of the students' understanding and mental models of the components of recursion from functioning and functionality perspectives. The arrow on the left shows the findings related to the functionality principles of evolution of the students' behaviour within the AVDA environment. The arrow on the right, demonstrates the findings related to the functioning principles. The central arrow shows the coordinated axis for the evolution of the corresponding students' mental models related to those functioning and functionality aspects. This three-arrowed model shows the gradual process towards having a viable integrated model of recursion.

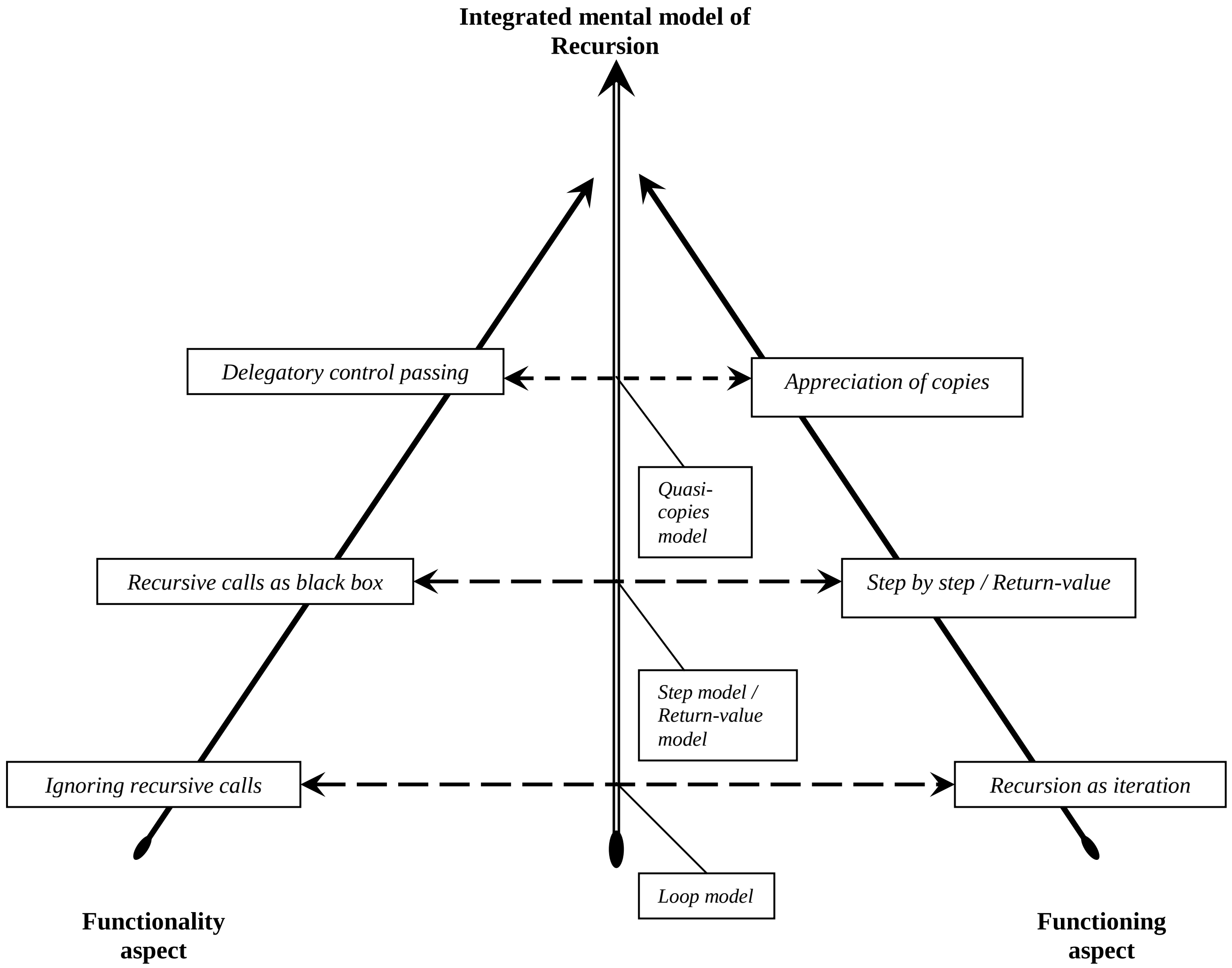

Integrated mental model of Recursion
Delegatory control passing
Appreciation of copies
Quasi-copies model
Recursive calls as black box
Step by step / Return-value
Step model / Return-value model
Ignoring recursive calls
Recursion as iteration
Loop model
Functionality aspect
Functioning aspect

Figure 140- Integrated model of recursion from a functional abstraction standpoint

The interesting point is that the development and evolution of mental models from a functioning aspect can be performed independently of the functionality aspect and vice-versa. This means that one might have a good understanding of the functioning of delegatory control passing, but it can still be located in the low levels of the functionality developing roadmap. Therefore, the development across each one of these roadmaps can be done independently of the other. However, there are still some connections between these roadmaps which were strengthened through students' experiments with AVDA.

Strengthening the connections alongside the evolution of the mental models across the functioning and functionality roadmaps, coordinates the following model for the evolution of the students' mental model of the concept of recursion towards an integrated viable model.

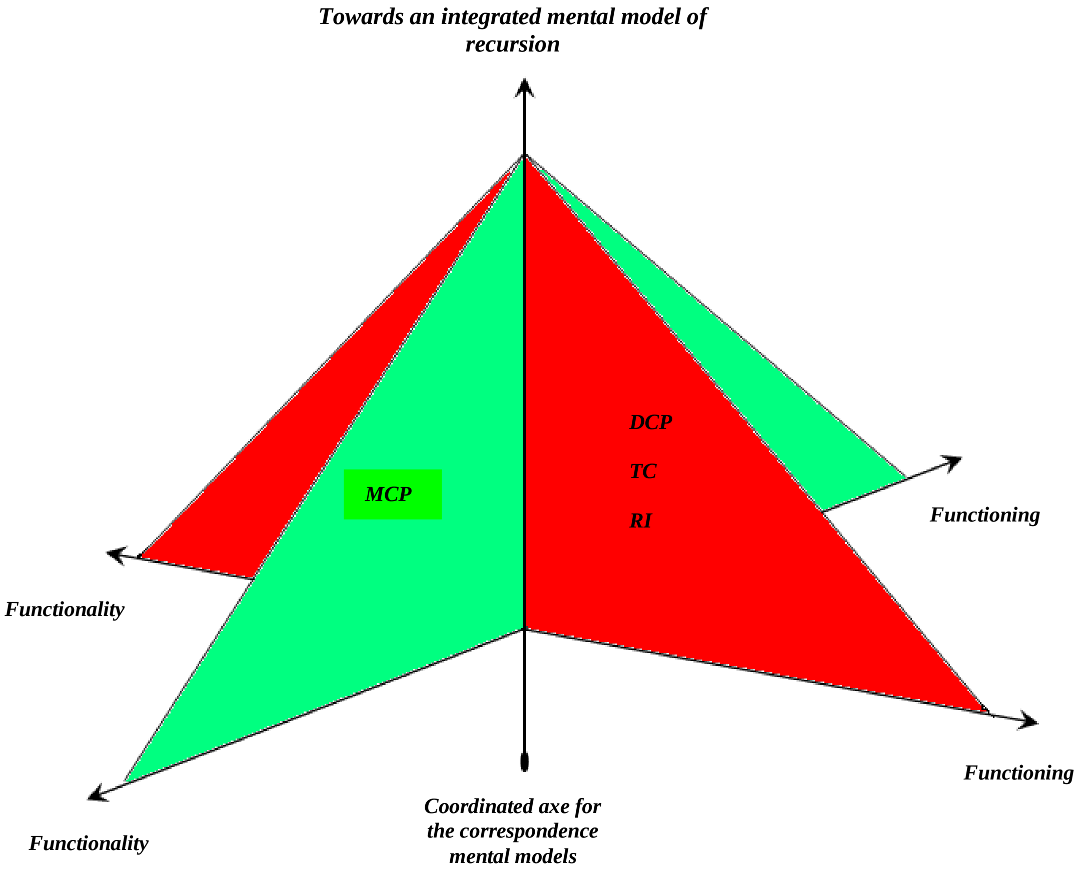

Towards an integrated mental model of recursion
DCP
TC
RI
MCP
Functioning
Functionality
Functioning
Functionality
Coordinated axe for the correspondence mental models

Figure 141- Coordinated Pyramid for Integrated Mental model of Recursion (*CPIM*)

The image above, which is called the *Coordinated Pyramid for Integrated Mental model of Recursion* (CPIM), illustrates the relationships and connections between the functioning and functionality aspects of the elements of mental models of recursion and correspondence model of each one of them which are evolving towards the integrated mental model of recursion. Each axis of the *CPIM* represents the functioning and functionality dimensions of one of the mental models elements as it shown in the above picture. The vertical axis of CPIM shows the evolution of the corresponding mental models of each one of those elements.

These axes are evolving towards a coordinated state of a mental model of recursion on the apex of the CPIM. Previously, most research has placed emphasis on mental models for the functioning dimension of recursion, with the copies model seen as the ultimate achievement. However, possessing such a model is in itself an insufficient achievement, since mastery of recursion requires a developed sense of the functionalities of recursion.

The AVDA visualisation approach embraced both these dimensions by enabling students to employ their current model for the functionality of recursion to support the development of a more sophisticated model of the functioning of recursion. Similarly, AVDA supported the emergence of a more sophisticated model for the functionality of recursion through the use of the students' current model for the functioning of recursion. Thus, whilst in principle functioning and functionalities might develop independently, AVDA set out to internally bridge these two dimensions to pedagogic advantage. The findings of the research ascertained that most of the students who participated in this study developed their mental model of the recursion as the aforementioned CPIM.

8.4. Limitations of the study

To implement this research program, I was faced with a few limitations. These can be categorised as: implementation and contextual limits. Implementation limits, refers to the boundaries that I had relating to designing and implementing the study. Contextual limits, refers to the constraints that I had regarding the generalizations of the results.

Time was one of the major constraints of this study. Based on design based research methodology, I had to design the required computer-based tools through the three iterations within a limited time frame. The design and programming of the domains of abstraction was a very time-consuming part of the research. The other constraint was the time limit for the duration of interviewing the students who participated in my research to test the domains in each of the three iterations. On average, each interview lasted 1 – 1.5 hours. The time that students needed to work with the different modules in each one of the domains was predicted and in total each interview was expected to be completed between 1.15 – 1.30 minutes. Longer time periods were limiting, the main barrier was the availability of the students who participated in the research. A further issue, which was more about the qualitative interviewing and participant observation, was that in this case interviews of a longer time length did not provide further significant findings to this thesis. In contrast, it risked causing the students to get bored and give unattended responses.

Another limitation for this study was finding volunteer students for testing the domains in each of the three iterations. The multidisciplinary nature of the concept of recursion required me to work with students from both mathematics and computer sciences disciplines. This proved to be testing and time-consuming resolved only by offering money to prospective participants..

From a contextual perspective, the sensitivity of the findings of the research to certain features and characteristics of the design is another limitation of this research. Although the AVDA approach revealed significant and common problems with how students understand and applied the concept of recursion, I had to be careful about generalising the findings as they were gained in certain domains of abstractions, under certain AVDA environment conditions and with the particular students who participated in this research.

8.5. Implications

This section focuses on the implications of the study from two perspectives: pedagogic and further research. Implications for further research in this domain are considered as new conjectures that need to be insightfully investigated in the future.

The first implication for further studies is the need for a comprehensive definition for the concept of recursion. There is an evident lack of an all-inclusive definition of recursion in existing literature and in most mathematics and computer science text books. This affects the level of prior academic knowledge that university level students possess in this field. The findings of this research ascertained that one of the problems of the students in understanding and using recursion was, not having a clear knowledge about its crucial components.

In computer sciences and in mathematics the concept of recursion requires two different definitions. This is a very important step which makes teaching and understanding this concept easier. It also allows students not to be confused by the different functions of the components of the concept of recursion in those disciplines. For instance, the base case, which is one of the main components of recursion, from a mathematical analysis perspective is a *starting point*, whereas, from a computational perspective is a *stopping condition*. Secondly, the findings of this research are extracted based on the computational aspect of the concept of recursion. Presumably there are similar problems with regards to understanding and applying recursion from a mathematical analysis point of view, which also need to be investigated.

Finally, the AVDA visualisation approach demonstrated that animative visualisation has a significant role in revealing the hidden mechanism of the declarative nature of the concept of recursion. Based on these results, my conjecture is that designing a smart and precise animative visualisation for embedded recursion would be an effective and useful method of assisting students to see the mechanism of delegatory flow between the recursive calls and over the whole procedure. The design process must be developed in such a way that it provides more control for the user to engage with it constructively. In other words, move from domains of abstractions towards microworlds.

The research findings also suggest some pedagogic implications for teaching and understanding the concept of recursion. Firstly, due to the inherent complexities of the concept of recursion, in order to uncover and make those complexities clear, using technology and computer-based tasks is of vital importance to introduce the concept of recursion. To open such a window for the students using traditional methods is a very difficult, if not impossible task. Secondly, avoiding the stereotypical examples to teach and introduce the concept of recursion is extremely advisable. The risk of using only very particular examples is subjective. It is possible that the students might simply echo the teachers' language and encapsulation of them without appreciating the mechanism of recursion. A further danger in using such examples is that the students might be unable to apply recursion in different and more robust problem-solving situations. Thirdly, the AVDA visualisation approach ascertained that using animation has a significant role in revealing the hidden layers of the concept of recursion, especially the flow of control and delegatory control passing in recursive procedures. And finally, the results and findings demonstrated that the students who participated in this research showed no sign of any difficulties with the phenomenalization of the concept of recursion by using spirals and binary trees. Therefore, the final pedagogic implication of this study suggests that using everyday analogies does encourage students to understand and think about the components of the concept of recursion more easily.

The next and final section of this chapter concentrates on the final reflections of this study.

8.6. Final reflections

The last section of this chapter summarises the account of how the research was shaped and discusses its contributions to increased knowledge in the domain of the concept of recursion.

Fractals have had a profound effect on my life and way of thinking. At the time of my study for a Masters degree a single fractal image created a pivotal moment in my professional career. Upon seeing the fractal image I realised that they could be used as a vehicle in an educational setting, to inspire others to experience the joy and beauty of mathematics. By investigating the main characteristics of fractals, I noticed that they can only be defined recursively. The characteristics of these fascinating geometrical objects, distinguishes them from other similar objects.

This inherent characteristic combined with my initial interest in fractals persuaded me to focus on the concept of recursion. Reviewing the literature revealed to me that recursion, despite its simple appearance, is a very complicated interdisciplinary concept to understand and apply. After reviewing more literature and related works on the concept of recursion, I found a few crucial gaps in the literature. The main gap in the literature was the separation of the functioning and functionality aspects of the concept of recursion and its components. Also, the literature was silent about the separation of the tail and embedded recursive procedures and their confluences on each other. Another gap in the literature was about the way that students' mental models about the concept of recursion, evolve from naive and unsophisticated models towards more viable and sophisticated models. To tackle those gaps and to see how students think about the concept of recursion, I decided to use computer assisted tasks and modules to uncover the hidden parts of the complicated mechanism of delegatory control passing.

The Logo-based tools designed in this study mainly represent a viable copies model of tail and embedded recursive procedures. They were designed to act as a window to represent and introduce the crucial components of the concept of recursion and the mechanism of control passing in the interactive environment of AVDA visualisation. This design abstraction provided me with deeper insights to investigate the concept of recursion from a functional abstraction view point.

The findings of this research provide the following contributions to the body of knowledge about the concept of recursion. Firstly, the findings support the work of Kurland and Pea (1985) on the flow of control and develop it by introducing *delegatory control passing* mechanism as the generalisation of what they called *active flow*. Secondly, the results and findings support the seminal work of Kahney (1983) and his followers Gotschi *et al*, (2003) on the categorization of the mental models of recursion. This study develops their work by introducing a new kind of mental model for the concept of recursion which is called a *quasi-copies* model. The final contribution of the research is to unfold the road maps of the students' understanding of the concept of recursion from functioning and functionality perspectives, which led me to sketch the CPIM model to show how the students' mental models of recursion evolve throughout active engagement with AVDA.

References

Aho, A. V., and Ulman, J. D. (1992), *Foundations of Computer Science*, New York, NY: W.H. Freeman and Company.

Ainley, J. (1996), ‘Purposeful contexts for formal notation in a spreadsheet environment’, *Journal of Mathematical Behaviour*, 15(4), pp. 405-422.

Ainley, J., and Pratt, D. (1995) ‘Planning for Portability’, in L. Burton and B. Jaworski (eds.), *Technology and Mathematics Teaching: A Bridge Between Teaching and Learning*, Bromley: Chartwell Bratt, 1995, pp. 435-448.

Ainley, J., and Pratt, D. (2002), ‘Purpose and Utility in Pedagogic Task Design’, in A. Cockburn & E. Nardi (Eds.), *Proceedings of the 26th Annual Conference of the International Group for the Psychology of Mathematics Education*, Vol. 2, pp. 17-24, Norwich, UK: PME.

Ainley, J., and Pratt, D. (2006), ‘Design and Understanding, Digital Technologies and Mathematics Teaching and Learning: Rethinking the Terrain’, *The Seventeenth International Commission on Mathematical Instruction Study,* Vietnam: Hanoi University of Technology.

Ainley, J., Pratt, D., and Hansen, A. (2006), ‘Connecting Engagement and Focus in Pedagogic Task Design’, *British Educational Research Journal* 32(1), 21-36.

Allen, B., and Johnston-Wilder, S. (Eds.) (2004), *Mathematics Education, Exploring the Culture of Learning*, Routledge Falmer, London: The Open University.

Ammari-Allahyari, M. (2005), ‘Exploring Students’ Understanding of the Relationship between Recursion and Iteration, *Proceeding of the British Society for Research into Learning Mathematics,* 25(2), pp. 1-7.

Ammari-Allahyari, M. (2006), ‘The Role of Aesthetics in Mathematics Education’, *Proceeding of the British Society for Research into Learning Mathematics,* 26 (1), pp. 7-11.

Anazi, Y., and Uesato, Y. (1982a), ‘Is recursive computation difficult to learn?’, *CIP paper No. 439, Dept. of Psychology,* Carnegie – Mellon University..

Anazi, Y., and Uesato, Y. (1982b), ‘Learning recursive procedures by middle school children’, *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, Ann Arbor, Michigan, pp. 100-102.

Anderson, J. R., Pirolli, P., and Farrell, R. (1999), ‘Learning to program recursive functions’, in Chi M. T., R. Glaser, and M. J. Farr (Eds.), Hillsdale, NJ: Lawrence Erlbaum Associates, Inc., pp 151-183.

Anderson, J. R., Farrell, R., and Sauers, R. (1984), 'Learning to Program LISP', *Cognitive Science*, Vol. 8, pp. 87-129.

Barab, S., and Squire, K. (2004), 'Design-Based Research: Putting a Stake in the Ground'", *The Journal of The Learning Sciences*, 13(1), pp. 1-14.

Bhuiyan, S. H. (1992), 'Identifying and Supporting Mental Methods of Recursion in a Learning Environment', unpublished PhD thesis, Dept. of Computational Science, University of Saskatchewan, Saskatchewan, Canada.

Bloch, E. D. (2000), *Proof and Fundamentals*, Boston: Birkhauser.

Brown, A. L. (1992), 'Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings', *The Journal of The Learning Sciences*. 2(2), pp.141–178.

Bryman, A. (1988), *Quality and Quantity in Social Research*, London: Unwin Hyman.

Bryman, A. (2001), *Social Research Methods*, Oxford University Press: Oxford.

Burnett, J. D. (1982), *LOGO: An Introduction For Teachers, Students and Other Computer Users New to the Philosophy and Methodology of Logo*, New Jersey: Creative Computing Press.

Carey, S. (1988), ‘Conceptual differences between children and adults’, *Mind and Language,* 3, pp. 167-181.

Cobb, P., Confrey, J., diSessa A. A., Lehrer, R., and Schauble, L. (2003), ‘Design Experiments in Educational Research’, *Educational Researcher,* Vol. 32, No. 1, pp. 9-13.

Collins, A. (1988), ‘Cognitive Apprenticeship and Instructional technology’, (Technical Report No. 6899), Cambridge, MA: BBN Labs Inc.

Cotton, J. (1995a), *The Theory of Learning Strategies: An Introduction*. London: Kogan Page.

Cotton, J. (1995b), *The Theory of Learning: An Introduction*. London: Kogan Page.

Craik, K. (1943), *The Nature of Explanation*, Cambridge: Cambridge University Press.

Dale, N. B., and Weems, C. (1991) *Pascal* (3rd ed.), Lexington, MA: D.C. Heath.

Davis, R. B. (1984), *Learning Mathematics*. London: Croom Helm Ltd.

Design-Based Research Collective (2003), Design-based research: An emerging paradigm for educational inquiry, *Educational Researcher,* 32(1), 5-8.

Dewey, J. (1938), *Experience and Education,* New York: Macmillan.

diSessa, A. A., and Sherin, B. L. (1998), 'What changes in conceptual change?', *International Journal of Science Education*. Vol. 20, Issue 10, pp. 1155-1191.

diSessa, A. A., and Cobb, P. (2004), 'Ontological innovation and the role of theory in design experiments', *Journal of the Learning Sciences,* 13, pp. 77-103.

Dubinsky, E. (1994), 'Reflective abstraction in advanced mathematical thinking', in Tall, D. (Ed.), *Advanced mathematical Thinking*, Dordrecht: Kluger Academic Press, pp. 95-123.

Edelson, D. C. (2002), 'Design research: what we learn when we engage in design', *Journal of the Learning Sciences*, 11(1), pp. 105-121.

Edwards, L. (1995), 'Microworlds as Representations', in *Computers and exploratory learning*, diSessa, A., C. Hoyles, R. Noss, With L. Edwards (Eds.), Berlin: Springer Verlag, pp. 127-154.

Enderton, H. B. (1972), *A Mathematical Introduction to Logic*, New York: Academic Press.

Evans, J. (1989), *Bias in Human Reasoning, Causes and Consequences*, London: Lawrence Erlbaum.

Fegers, V. G., and Johnson, M. B. (2002), ‘Fractals – Energizing the Mathematics Classroom’, in *Fractals, Graphics, & Mathematics Education*, M. L. Frame & B. B. Mandelbrot, New York: The Mathematical Association of America, pp. 69-105.

Fey, J. T. (1989), ‘Technology and Mathematics Education: A survey of recent developments and important problems’, *Educational Studies in Mathematics*, 20, pp. 237-272.

Frame, M. L., and Mandelbrot, B. B. (2002), *Fractals, Graphics, & Mathematics Education*, USA: The Mathematical Association of America.

Gans, H. J. (1968), ‘The Participant – observer as Human Being: Observations on the Personal Aspects of Field Work’, in H. S. Becker, *Institutions and the Person*, Papers presented to Everett C. Hughes, Aldine, Chicago, pp. 300-317.

Gentner, D. (1983), ‘Structure-mapping: A theoretical framework for analogy, *Cognitive Science* 7, pp. 155-170.

Gentner, D., Brem, S., Ferguson, R. W., Markman, A. B., Levidow, B. B., Wolff, P., and Forbus, K. D. (1997), 'Analogical reasoning and conceptual change: A case study of Johannes Kepler', *The Journal of the Learning Sciences*. 6(1), pp. 3-40.

Gentner, D., and Stevens, A. L. (1983), *Mental Models*, Hillsdale, NJ: Lawrence Erlbaum Associates.

George, C. E. (2000), 'Experiences with Novices: The Importance of Graphical Representations in Supporting Mental Models', in A. F. Blackwell, and E. Bilotta (Eds.) *Proceedings of Psychology of Programming Interest Group* 12, 2000, pp. 33-44, Italy.

Gersting, J. L. (2007), *Mathematical Structures for Computer Sciences: A Modern Approach to Discrete Mathematics* (6th ed.), New York: W. H. Freeman and Company.

Gibbs, G. (2007), *Analysing Qualitative Data*, Los Angeles; London: Sage Publications.

Ginat D., and Shifroni, E. (1999), 'Teaching Recursion in a Procedural Environmental - How much should we emphasize the Computing Model?', *Technical Symposium on Computer Science Education*, pp. 127-131.

Gold, R. L. (1969), *Roles in Sociological Field observations*, in G. J. McCall and J. L. Simmons (Eds.), *Issues in Participant Observation*, pp. 30-39, Reading, MA: Addison-Wesley.

Goldin, G. (1991), ‘The IGPME working group on representations’, in F. Furinghetti (ed.) *Proceedings of the XV Conference of the International Group for the Psychology of Mathematics Education* (Volume 1, p. xxii) Assisi, Italy.

Goodyear, P. (1984), *Logo: A Guide to Learning Through Programming*, London: Ellis Horwood Publishers Ltd.

Gotschi, T., Sanders, I., and Galpin, V. (2003), ‘Mental Models of Recursion’, *ACM SIGCSE Bulletin*, Vol. 35, Issue 1, pp. 346-350.

Greer, J. E. (1987), ‘Empirical Comparison of Techniques for Teaching Recursion in Introductory Computer Science’, unpublished PhD dissertation, University of Texas at Austin.

Guin D., Rithven, K., and Trouche, L. (2005), *The Didactical Challenge of Symbolic Calculators Turning a Computational Device into a Mathematical Instrument*, USA: Springer.

Haberman, B., and Averbuch, H. (2002), 'The case of Base Cases: Why are they so Difficult to Recognize? Student Difficulties with Recursion', *ITiCSE'02, June 24-26*, pp. 84-88, Denmark: Arahus.

Harel, I., and Papert, S. (1993), *Constructionism*, Norwood: Ablex Publishing Corporation.

Harvey, B. (1997), *Iteration, Control Structures, Extensibility. Computer Science Logo Style*, *volume 2: Advanced Techniques* 2/e, Cambridge, MA: The MIT Press.

Harvey, B., and Wright, M. (1994), *Simply Scheme*. Cambridge, MA: The MIT Press.

Harvey, B. (1985) *Computer Science Logo Style, Volume 1: Intermediate Programming*. Cambridge, MA: The MIT Press.

Henderson, P. B., and Romero, F. J. (1989), 'Teaching Recursion as a Problem-Solving Tool Using Standard ML', *ACM SIGCSE*, Vol.21, Issue 1, pp. 27-31.

Hoyles, C. and Noss, R. (1987), 'Children Working in Structured Logo Environment: From Doing to Understanding, *Researches on Didactiques des Mathematiques*, 8(12), pp. 131-174.

Hoyles, C., and Sutherland, R. (1989), *Logo Mathematics in the Classroom*, London: Routledge.

Hoyles, C., and Noss, R. (1992), *Learning Mathematics and Logo*, Cambridge, MA: The MIT Press.

Jagcinski, R. L., and Miller, R. A. (1978), 'Describing the human operator's internal model of a dynamic system', *Human Factors,* 20(4), pp. 425-433.

Johnson–Laird, P. N. (1983), *Mental Models: Towards a Cognitive Science of Language, Interface and Consciousness*. Cambridge: Cambridge University Press.

Johnson – Laird, P. N. (2004), 'The History of Mental Models', in K., Manktelow, and M. C. Chung, (Eds.), *Psychology of Reasoning: Theoretical and Historical Perspectives*, pp. 179-212, New York: Psychology Press.

Johnston-Wilder, S. J., Johnston-Wilder, P., Pimm, D., and Westwell, J. (2005), *Learning to Teach Mathematics in Secondary School,* London: Routledge Falmer.

Kahney, H. (1983), 'What do novice programmers know about recursion?', *Proceedings of the CHI '83 Conference on Human factors in Computer Systems,* pp. 235-239, Boston, MA.

Kann, C., Lindenman, R., and Heller, R. (1997), 'Integrating algorithm animation into a learning environment', *Computers Education,* 28 (4), pp. 223-228.

Kessler, A., and Anderson, J. (1986), 'Learning Flow of Control: Recursive and Iterative Procedures', *Human – Computer Interaction,* 2, pp. 135-166.

Kieras, D., and Bovair, S. (1984), 'The role of mental models in learning to operate a device', *Cognitive Science*. 8, pp. 255-273.

Kilpatrick, J., Hoyles, C., and Skovsmos, O. (Eds.) (2005), *Meanings in Mathematics Education*, USA: Springer.

Kim, Daniel H. (1993), 'The link between Individual and Organizational Learning', *Sloan Management Review*. pp. 37-50.

Kim, S., and Lee, H. (2006), 'The Impact of Organizational Context and Information Technology on Employee Knowledge-Sharing Capabilities', *Public Administration Review,* 66(3), pp. 370-385.

Koffman, E. B. (1992), *Pascal* (4th ed.). Reading, MA: Addison Wesley.

Kolb, D. A. (1984), *Experimental Learning – Experience as the Source of Learning and Development,* Englewood Cliffs, NJ: Prentice Hall.

Kurland, D. M., and Pea, R. D. (1985), ‘Children’s mental models of recursive LOGO programs’, *Proceedings of the 5th Annual Conference of the Cognitive Science Society*, session 4, pp. 1-5, New York, NY: Rochester.

Kvale, S. (1996), *Interviews: An Introduction to Qualitative Research Interviewing*, Thousand Oaks, California: Sage.

Lakoff, G., and Nunez, R. E. (2000), *Where Mathematics Comes From: The Embodied Mind Brings Mathematics into Being*, New York: Basic Books.

Larkin, J. H. (1983), ‘The role of problem representation in physics’, in D. Gentner and A. L. Stevens (Eds.), *Mental Models*, pp. 75-98, Hillsdale, NJ: Lawrence Erlbaum Associates.

Lave, J. (1988), *Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life*, Cambridge: Cambridge University Press.

Lave, J., and Wenger, E. (1991), *Situated Learning Legitimate Peripheral Participation*, Cambridge: Cambridge University Press.

Leron, U., and Zazkis, R. (1985), ‘Mathematical Induction and Computational Recursion, *Proceedings of the First International Conference for Logo and Mathematics Education*, Hoyles, C. & Noss, R. (Eds.), London.

Levy, D., and Lapidot, T. (1993) ‘Shared Terminology, Private Syntax: The case of recursive descriptions’, *Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education,* Denmark pp. 89-93.

Masingila, J. O. (1993), ‘Learning from mathematics practice in out-of-school situations’, *For the Learning of Mathematics*, 13(2), pp. 18-22.

Mandelbrot, B. (1982), *The Fractal Geometry of Nature*, San Francisco: W. H. Freeman.

May, T. (2001), *Social Research Issues, Methods and Process*, Great Britain: Open University Press.

McCracken, D. D., and Salmon, W. S. (1987), *A Second Course in Computer Science with Modula-2 [data structures]*, New York: Wiley.

Medin, D. L., Goldstone, R. L., and Gentner, D. (1990), ‘Similarity involving attribute and relations: judgments of similarity and differences are not inverses’, *Psychological Science,* 1, pp. 64-69.

Minsky, M. (1988), *The Society of Mind*. London: Picador in association with Heinemann.

Montovani, G. (1996), ‘Social Context in HCI: A New Framework for Mental Models, Co-operation and Communication’, *Cognitive Science Vol. 20, No. 2,* pp. 237-269.

Muramatsu, J., and Pratt, W. (2001), ‘Transparent Queries: investigation user’s mental models of search engines’, *Proceedings of the 24th annual international ACM SIGIR conference on R&D in informal retrieval*, pp. 217-224.

Norman, D. A. (1983), ‘Some Observations on Mental Models, in Gentner, D. and A. L. Stevens (Eds.), *Mental Models*, London: Erlbaum.

Noss, R. (1984), *Children Learning Logo Programming, Interim report No. 2 of The Children Logo Project*, UK: Microelectronics Education Programme.

Noss, R., and Hoyles, C. (1996), *Windows on Mathematical Meaning: Learning Cultures and Computers,* London: Kluwer Academic Publisher.

Nuffield Mathematics Project (1967), *I Do, and I Understand*, Edinburgh, London, New York: Chambers; John Murray; Wiley.

Nunez, T., Schliemann, A. D., and Carraher, D. (1993), *Street Mathematics and School Mathematics,* Cambridge: Cambridge University Press.

Orton, A. (2004), *Learning Mathematics, Issues, Theory and Classroom Practice,* London: Continuum.

Papert, S. (1954), *The Construction of Reality in the Child*, New York: Ballantine Books.

Papert, S. (1980), *Mindstorms: Children, Computers and Powerful Ideas*, New York: Basic Books.

Papert, S. (1990), ‘Constructionist Learning’, in I. Harel, and S. Papert (Eds.), *Constructionism*, Cambridge, MA: MIT Media Laboratory.

Papert, S. (1993), *The Children’s Machine: Rethinking School in the Age of the Computer*, New York: Basic Books.

Papert, S. (1996), ‘An Exploration in the Space of Mathematics Education’, *International Journal of Mathematical Learning*, Vol. 1, No. 1, pp. 95-123.

Peitgen, H. O., Jurgens, H., and Saupe, D. (1992), *Chaos and Fractal: New Frontiers of Science*, New York: Springer Verlag.

Pettigrew, A. M. (1997), ‘What is a Processual Analysis?’, *Scandinavian Journal of Management*, 13(4), pp. 337-348.

Pirolli, P. L., and Anderson, J. R. (1985), ‘The role of learning from examples in the acquisition of recursive programming skills’, *Canadian Journal of Psychology* 39, pp. 240-272.

Posner, G. J., Strike, K. A., Hewson, P. W., and Gertzog, W. A. (1982), ‘Accommodation of a scientific conception: toward a theory of conceptual change’, *Science Education*. 66(2), pp. 211-227.

Pratt, D. (1998), ‘The Construction of Meanings In and For a Stochastic Domain of Abstraction’, Unpublished PhD thesis, London: University of London.

Pratt, D. (2000), ‘Making Sense of the Total of Two Dice’, *Journal for Research in Mathematics Education*, 31(5), pp. 602-625.

Pratt, D., Noss, R., and Jones, I. (2008), ‘Designing for mathematical abstraction’, (in press), *The Journal of the Learning Sciences*.

Rasmussen, J. (1979), ‘On the structure of knowledge – A morphology of mental models in a man-machine system context (Risø-M-2192)’, *Risø National Laboratory, Electronic Department, Denmark, Roskilde*

Reeves, T. (2006), ‘Design research from a technology perspective’, in J. V. D. Akker, K. Gravemeijer, S. McKenney, and N. Nieveen (eds.), *Educational design research*, pp. 52–66. New York: Routledge.

Richardson, S., Dohrenwend, B. S., and Klein, D. (1965), *Interviewing, Its Forms and Functions*, New York: Basic Books.

Sandoval, W. A. (2004), ‘Developing Learning Theory by Refining Conjectures Embodied in Educational Designs’, *Educational Psychologist*, Vol. 39, No. 4, pp. 213-223.

Schon, D. (1983), *The reflective practitioner*, New York: Basic Books.

Senge, P. M. (1990), *The Fifth Discipline: The Art and Practice of the Learning Organization*, London: Doubleday.

Senge, P. M. (1992), ‘Mental models’, *Planning Review*. Vol. 20, No.2, pp. 4-44.

Sfard, A. (1991), ‘On the dual nature of mathematical conceptions: Reflections on processes and objects as different sides of the same coin’. *Educational Studies in Mathematics* 22, pp. 1-36.

Skemp, R. R. (1971), *The Psychology of Learning Mathematics*, Harmondsworth: Penguin.

Skemp, R. R. (1976), ‘Relational understanding and instrumental understanding’, *Mathematics Teaching*, 77, pp. 20-26.

Sooriamurthi, R. (2001), 'Problems in comprehending recursion and suggested solutions', *Proceedings of the 6th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pp. 25-28.

Spicer, D. P. (1998), 'Linking Mental Models and Cognitive Maps as an Aid to Organisational Learning, *Career Development International 3(3)*, pp. 125-132.

Stake, R. E. (1981), 'The Art of Progressive Focussing', *The 65th Annual Meeting of the American Educational Research Association*, Los Angeles, CA.

Tall, D. (1994), (Ed.) *Advanced Mathematical Thinking*, Dordrecht: Kluger Academic Publisher.

Tall, D., and Thomas, M. (2002), *Intelligence, Learning and Understanding in Mathematics: A tribute to Richard Skemp*, Flaxton, Australia: Post Pressed.

Tall, D., and Vinner, S. (1981), 'Concept image and concept definition in mathematics with particular reference to limits and continuity', *Educational Studies in Mathematics, 12,* pp. 151-169.

Tikly, C., and Wolf, A. (2000), *The Maths We Need Now: Demands, Deficits and Remedies*, London: Institute of Education, University of London.

Tung, S. H., Chang, C., Wong, W. K., and J. C. J. Jehng (2001), ‘Visual representation for recursion’, *Int. J. Human – Computer studies*, 54, pp. 285-300.

Turkle, S., and Papert, S. (1991), Epistemological Pluralism and the Revaluation of the Concrete, in I. Harel and S. Papert, (Eds.), *Constructionism: Research Reports and Essays 1985-90*, Norwood, NJ, Ablex Publishing, pp. 161-192.

Turbak, F., Royden, C., Stephan, J., and Herbst, J. (1999), ‘Teaching Recursion Before Loops in CS1’, *Journal of Computing in small colleges*, Vol. 14, No 4, pp. 86-101.

Van der Veer, G. C., and Felt, M. A. M. (1988), ‘Development of mental models of an office system: A field study on an introductory course’, in G. C. van der Veer and G. Mulder (Eds.) *Human-Computer Interaction: Psychonomic Aspects*, New York: Springer-Verlag, pp. 251-272.

Velazquez, J. A. (2000), ‘Recursion is Gradual steps (is recursion really that difficult?)’, *ACM SIGCSE Bulletin*, Vol. 32, Issue 1, pp. 310-314.

Welman, C., Kauger, F., and Mitchell, B. (2005), *Research Methodology*, Oxford: Oxford University Press.

Wiedenbeck, S. (1988), ‘Learning recursion as a concept and as a programming technique’, *ACM SIGCSE Bulletin*, 20 (1), pp. 275-278.

Wilensky, U. (1993), ‘Connected Mathematics: Building Concrete Relationships with Mathematical Knowledge’, unpublished PhD dissertation, Cambridge, MA: MIT Media Laboratory

Wilensky, U. (1991), ‘Abstract meditations on the concrete and concrete implications for mathematics education’, in Harel I. and S. Papert (Eds.) *Constructionism*, Norwood, NJ: Ablex Publishing Corporation, pp. 193-203.

Wilcocks, D., and Sanders, I. (1994), ‘Animating recursion as an aid to instruction’, *Computer Education,* 23 (3), pp. 221-226.

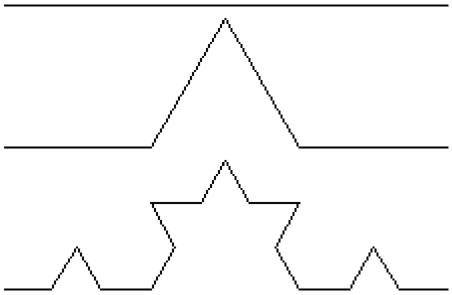
Wilkinson, D. (1998), *The Researcher’s Toolkit*, London: RoutledgeFalmer.

Wu, C. C., Dale N. B., and Bethel, L. J. (1998), ‘Conceptual Models and Cognitive Learning Styles in Teaching Recursion’, *Proceedings of the 29th SIGCSE technical symposium on computer science education*, Atlanta, Georgia, USA, pp. 292-296.

Appendix A

The semi-structured interview guides sample:

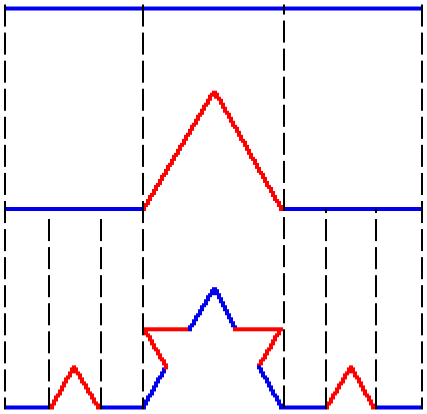
Second task of Iteration one:



1-How are these levels related to each other?

- **How is each level constructed?**
- **How can you make level 2 from level 1?**

2- Write a procedure to model the above image.



Task 1-Iteration 2

Theme:

Describing a given photograph of a spiral

Aim:

In this task I am going to examine students' *major problems in working with iteration as a programming technique as well as a problem solving strategy*.

Implementation:

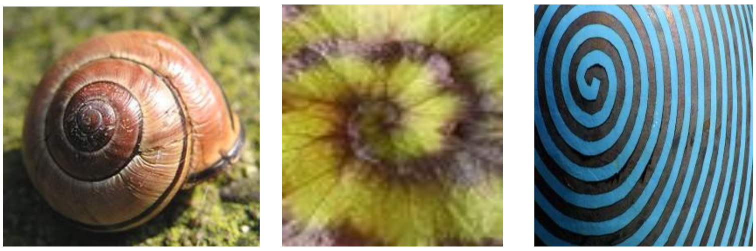
To achieve this goal, students will first be given a photograph of a spiral and then asked to describe it. Their possible answers are a spectrum from very general ideas about a spiral to a very sophisticated description of a spiral.

General opinion ↔ **sophisticated description**

The task will be performed in the format of a semi-structured interview. Questions have been designed to be initially general and gradually more specific. Most of them are open ended questions, which will be helpful in maintaining the desired direction of the interview.

My role as a participant observer:

Depending on the participants' responses, my role will be different. As a participant observer, in different situations on the above spectrum, I will encourage them to give more detail about the structure, shape, size, and angle, etc. or I will just be an active listener. I will also help them in the case that they seem to be having difficulty and are not able to progress. Under any circumstances as a participant observer, I will avoid giving direct references to iteration or recursion. The students will also be assisted by an explanation or through some necessary hints when they are confused or stuck. My role is to *note down and record their efforts*.



1. **Can you describe the above image?**
2. **What can you say about the shape, structure, size, angle, …**
3. **Can you draw it in the blank space below?**
4. **What are the essential features of it when you want to draw it?**
    a. **Structure?**
    b. **Shape? (dimension, colour, …)**
    c. **Angle?**

5. **How would you go about using a computer program to draw it? (preferably using Logo)**