Learning Programming by using Memory Transfer Language (MTL) without the Intervention of an Instructor

Leonard J. Mselle

Dpt. Computer Science
School of Informatics and Virtual Education
University of Dodoma
Mselel@yahoo.com

Keywords: Program visualization, Memory Transfer Language, program parsing.

Abstract

Visualization as a technique used to teach programming is gaining momentum. Memory Transfer Language (MTL) is a programmer-driven visualizer used to learn programming. This article reports on results obtained from a class experiment where MTL was used by non-novices to learn programming. The experiment was carried out to test the effectiveness of MTL in assisting students to learn programming (in a second language) without the intervention of a teacher. Results between the experimental and control group revealed that the group that studied programming using MTL without teachers' intervention performed better than the group that studied programming using conventional approach.

1. Introduction

Since at least the 1980s, computer science education researchers have searched for ways computer science teachers can better support their students (Du Boulay et al. 1981), (Du Boulay 1986). Recently, program visualization in the form of algorithm animations has yielded positive results (Hundhausen and Brown 2007), (Naps et al. 2003). To understand programming, the learner must be able to analyze/parse the program tokens unambiguously (Ben-Ari and Sajaniemi 2003). Program animation is a means to facilitate a learner to visibly parse the code. However, most of the current animators are entirely machine-driven. Helpful as it is, machine-driven animation subordinates the learner to the machine. The side effect of machine-driven animation is the denial of full authority to the learner.

So far, MTL is the only sketch-based language in programming books that can be used as a learner-driven visualizer and parser. MTL is a sketch-based language used by authors, instructors and learners to parse the code as it affects computer memory (Mselle 2011b). MTL is used to describe the program behavior by means of concrete models and visualization (2011c). In other words MTL is a tool for program parsing without actively depending on the machine.

1.2 A brief discussion and illustration of MTL

MTL is a learner-driven compiler or parser whose general framework is as illustrated in Figure 1.

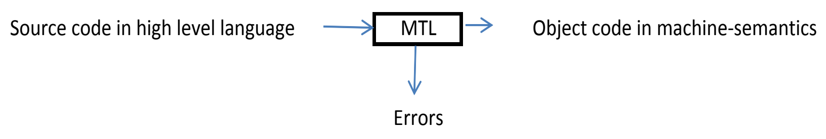
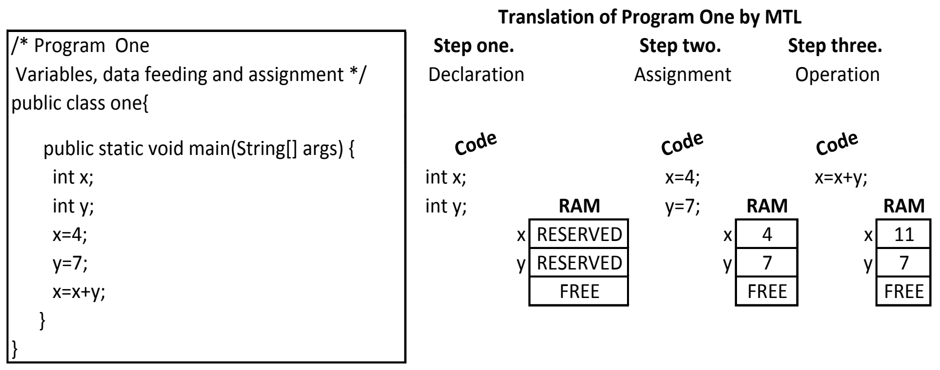

Source code in high level language
MTL
Object code in machine-semantics
Errors

Figure 1- The general framework of MTL

MTL is not machine dependent. It does not demand any knowledge or special rules for the learner to employ MTL. Successful use of MTL is directly tied with understanding the program and nothing else. Examples on the use of MTL to translate different programs and to visualize different programming aspects are demonstrated in Figures 2 through 8.

Figure 2 demonstrates how MTL is used by the learner to parse variable declaration, data feeding and data operation.

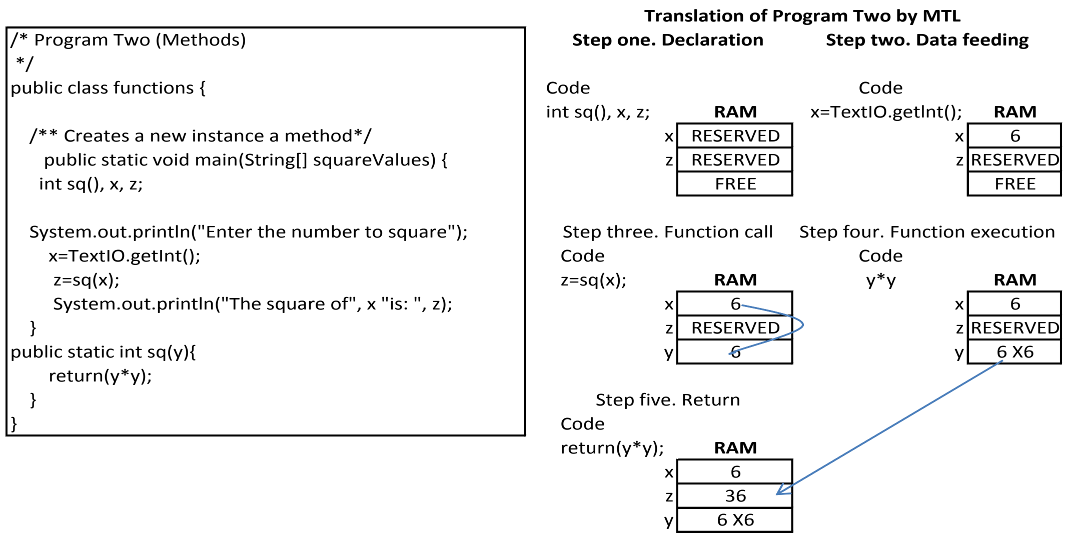


```
/* Program  One
 Variables, data feeding and assignment */
public class one{

    public static void main(String[] args) {
     int x;
     int y;
     x=4;
     y=7;
     x=x+y;
    }
}
```

Figure 2 - Parsing variable declaration, assignment and data operation by MTL

Using MTL, the relationship between variable and computer memory (RAM) is clearly visualized. Concepts of variable declaration, identifiers, data feeding, assignment, data operation such as addition, and data outputting are visualized by simple model which mimics the RAM.

Figure 3 shows how MTL can be used to parse functions/methods.

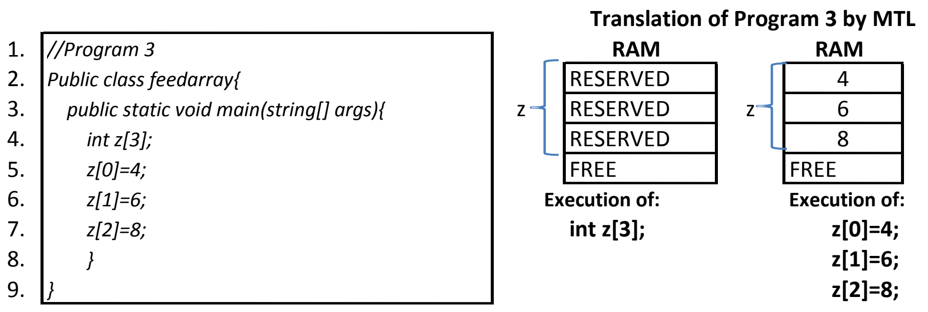


```
/* Program Two (Methods)
 */
public class functions {

   /** Creates a new instance a method*/
     public static void main(String[] squareValues) {
    int sq(), x, z;

   System.out.println("Enter the number to square");
      x=TextIO.getInt();
       z=sq(x);
       System.out.println("The square of", x "is: ", z);
   }
public static int sq(y){
      return(y*y);
   }
}
```

Figure 3- Parsing functions by MTL

Using MTL, abstract concepts such a functions, parameters and *return()* funtion are physically visualized by the same single model.

Figure 4 shows how MTL is used to translate arrays, array declaration and data feeding in arrays.

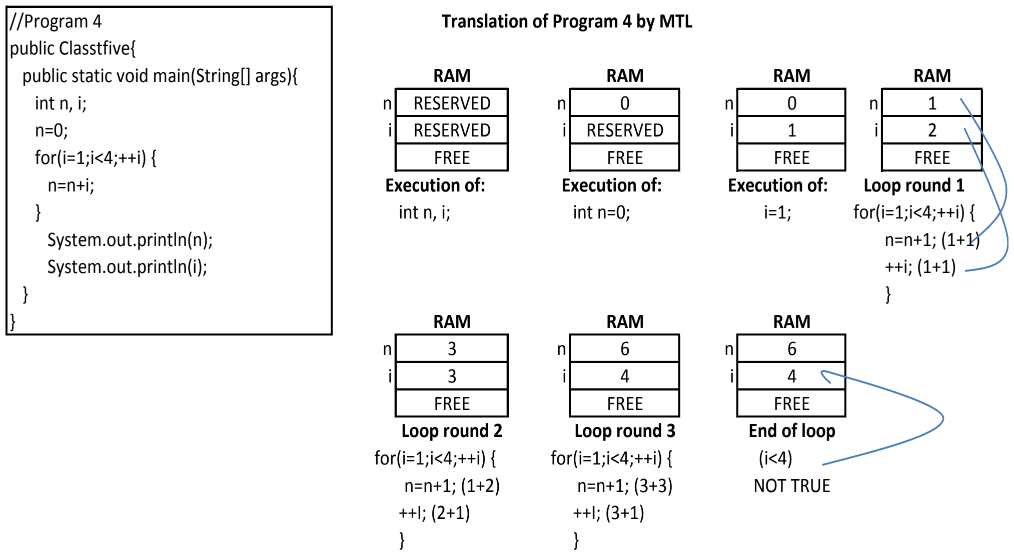


```
//Program 3
Public class feedarray{
   public static void main(string[] args){
      int z[3];
      z[0]=4;
      z[1]=6;
      z[2]=8;
      }
}
```

Figure 4- Parsing array declaration and array feeding by MTL

As demonstrated in Figure 4, the concept of arrays as a data structure different from other simple data types such as integers is clearly visualized. Data feeding in an array using subscripts is clearly visualized.

Figure 5 shows how MTL is used to translate a *for* loop.

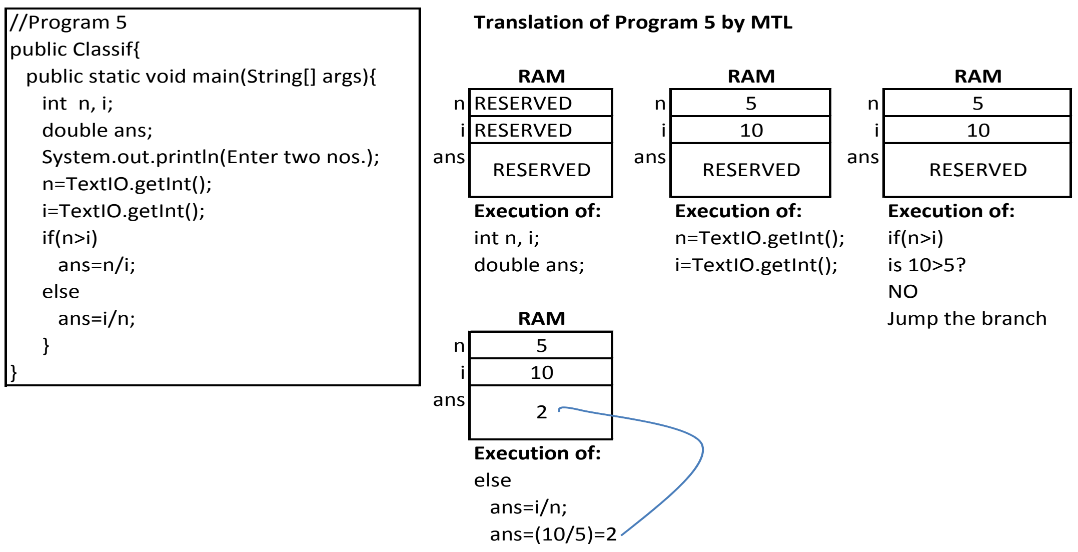


```
//Program 4
public Classtfive{
  public static void main(String[] args){
    int n, i;
    n=0;
    for(i=1;i<4;++i) {
      n=n+i;
    }
      System.out.println(n);
      System.out.println(i);
  }
}
```

Figure 5- Parsing a for loop by MTL

Loops constitute a big hurdle to programming students (Dehnadi and Bornat 2006). With MTL the control mechanisms that are achieved through loops are physically visualized, making it easy for the learner to determine the role(s) of each variable.

Figure 6 shows how MTL is used to demonstrate branching or selection.

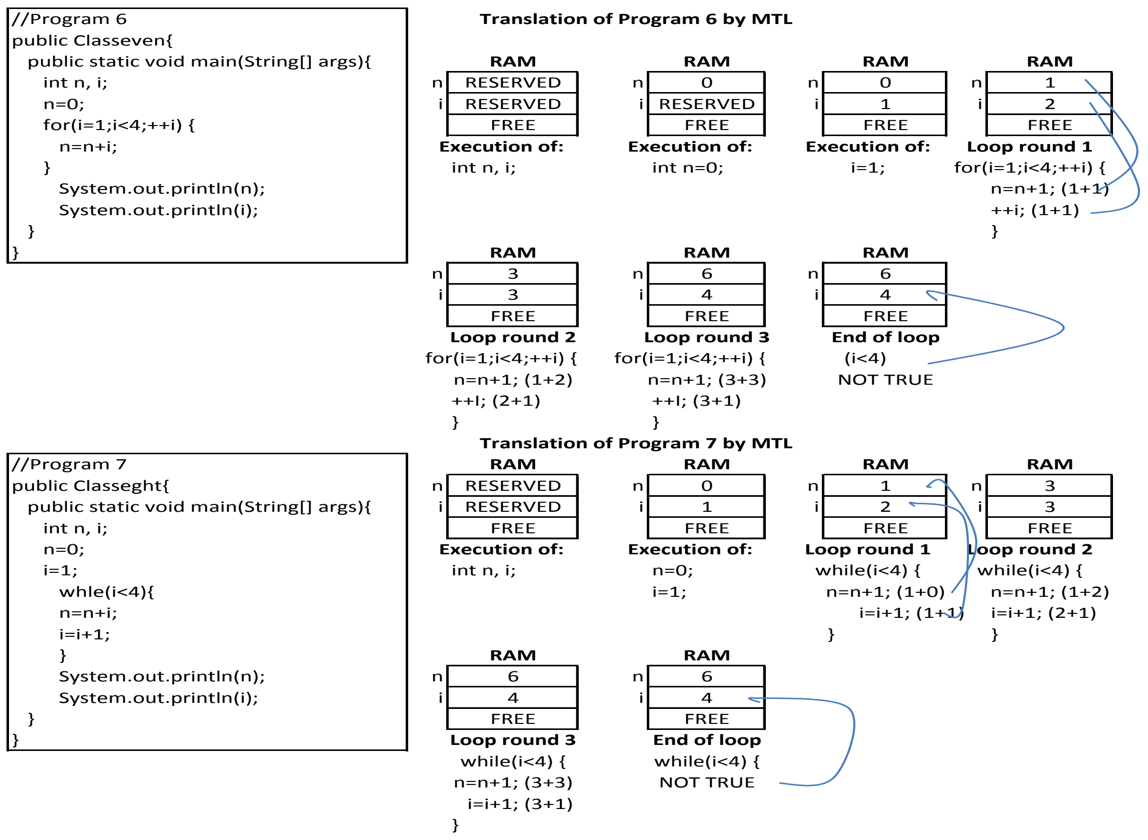


```
//Program 5
public Classif{
   public static void main(String[] args){
      int  n, i;
      double ans;
      System.out.println(Enter two nos.);
      n=TextIO.getInt();
      i=TextIO.getInt();
      if(n>i)
         ans=n/i;
      else
         ans=i/n;
      }
}
```

Figure 6- Parsing a branch by MTL

Selection in programming is another aspect not easily understood by programming students. MTL enables the learner to verify why a certain branch is taken and another branch is not taken.

Figure 7 shows similarities and differences between a *while* loop and a *for* loop.



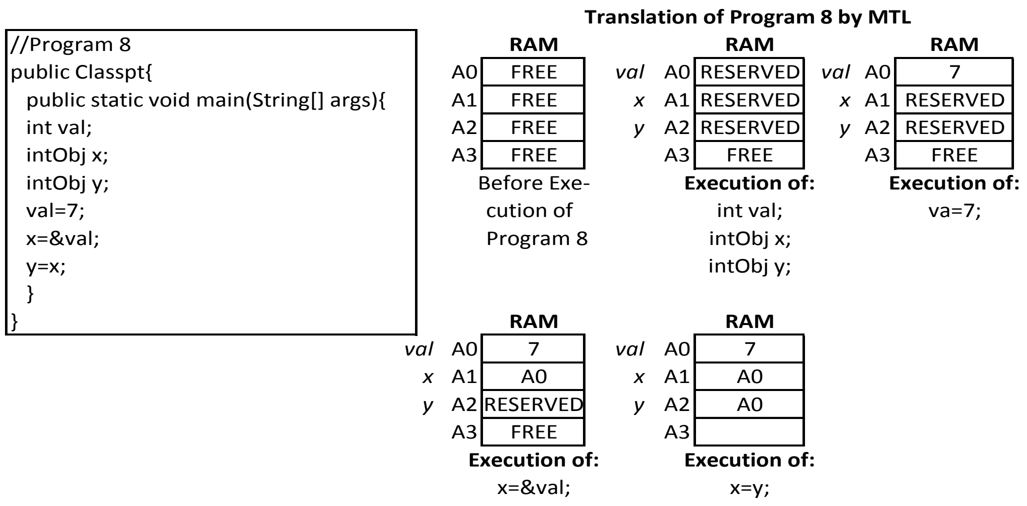
```
//Program 6
public Classeven{
   public static void main(String[] args){
      int n, i;
      n=0;
      for(i=1;i<4;++i) {
         n=n+i;
      }
         System.out.println(n);
         System.out.println(i);
   }
}
```

```
//Program 7
public Classeght{
   public static void main(String[] args){
      int n, i;
      n=0;
      i=1;
         whle(i<4){
         n=n+i;
         i=i+1;
         }
         System.out.println(n);
         System.out.println(i);
   }
}
```

Figure 7- Demonstration of similarities and differences between while and for loop constructs by MTL

Sometimes, having different loop structures which accomplish more or less the same objective is a source of confusion to students. Clear understanding of their similarities and minor differences may mitigate misconceptions. Using MTL as shown in Figure 7 different programming constructs can be compared and contrasted visibly.

Figure 8 shows how MTL is used to illustrate the concept of pointers in programming.



```
//Program 8
public Classpt{
  public static void main(String[] args){
  int val;
  intObj x;
  intObj y;
  val=7;
  x=&val;
  y=x;
  }
}
```

Figure 8- Demonstration of pointers by MTL

In Figure 8, the relationship between pointers and hexadecimal address scheme is visualized. The concept of pointers as variables that store addresses as opposed to other data types is visually demonstrated.

As demonstrated in Figures 2-8, MTL enables the author, instructor and the learner to employ the same model and the same symbol-type to show and describe the meaning of different aspects of programming. Questions like; what is a variable, why is it declared, what is the difference between an *int* and a *double* type, what is data feeding, what is data operation, what is an array, what are array subscripts, how are arrays different from simple data types, what is a loop, what mechanisms undergird looping, what are differences and similarities between different loop constructs, what are pointers, how do pointers differ from other types of variables such as integers, etc. are clearly visualized by the means of a single symbol and one model.

As demonstrated in the works of Mselle (2010, 2011a, 2011b, 2011c) and as shown in Figures 2-8, MTL approach reduces ambiguities which are rife in introductory programming.

2. Problem

Although the use of MTL in teaching programming has been reported to produce positive results (Mselle 2010, 2011a), there is no report about experiment conducted on non-novice programmers to find out if students can use MTL to learn a new programming language without an instructor during their second language study.

Many colleges and universities have undergraduate curricula which include learning to program in more than one language. It is a common practice, for example, to find schools or colleges teaching *Programming in C++* during one semester and *Programming in Java* in another semester. The practice of teaching students more than one programming languages is aimed at first making students aware of various languages that can be used in programming and secondly improving students' programming abilities. While the first motive is achieved, the second motive which is essential is hardly achieved (Dehnadi and Bornat 2006). In addition, most research works on program visualization have concentrated on testing the impact of animation tools in circumstances where the instructor is directly involved (Ben Bassat et al. 2001).

3. Objective

The objective of this study was to evaluate the impact of MTL in aiding non-novice programmers to pursue programming studies without the intervention of an instructor during their study of a second language.

To determine the impact of MTL in aiding programmers to pursue their programming lessons without the intervention of an instructor, a class experiment was conducted, where examination results from two groups were statistically compared.

4. The Experiment

To test the hypothesis that MTL can facilitate students pursue their programming classes without the help of an instructor and perform better than the students who pursue the course with the help of an instructor a sample of 108 third year students of the University of Dodoma (UDOM) was used in the experiment.

4.1 The Sample

Students learning “Java Programming” for the first time in the College of Informatics-UDOM, constituted the sample for this study. All students had pursued and passed examination in C++ Programming during their first-year program.

5. Method

A group of 14 students, from the sample of 108 students, who were studying *Java Programming* for the first time, volunteered to pursue the course using MTL. These students (n=14) constituted the experimental group. The rest of students (n=92) constituted the control group. The course syllabus covered; *variable declaration* and *types of variables*, *constants*, *data inputting*, *data manipulation*, *data outputting*, *flow of control*, *functions*, *arrays*, *strings*, *introduction to pointers* and *introduction to file handling, objects, and data encapsulation*. The duration of the course was 52 hours; with 26 hours being used for lectures and 26 hours assigned for laboratory sessions. The lead lecturer for the subject had eleven years experience in teaching programming in Java. For the purpose of this study, the experimental group agreed to use the manual entitled “Java for Novice Programmers” as their main reference book together with any other material while not attending lectures.

5.1 Materials

The materials used included; examination scripts and a programming manual in Java (Mselle 2011b). The programming manual is written to cover introductory programming which includes; variables and variable declaration, data inputting, data processing and outputting. Other topics include; flow of control (bifurcation and looping), arrays, strings, functions, files handling, pointers and objects.

5.2 Procedure

Before the beginning of the course the researcher held a meeting with all students where they were briefed about the experiment and the unique character of the *Java for Novice Programmers* manual. Students who volunteered to pursue the course without attending lectures were given the manual. They were instructed to do all assignments, laboratories and tests as the rest of the class. In the end of semester, the final examination was set by the lecturer who was in-charge of teaching the subject. Questions and solutions were reviewed by an external examiner to ensure adequacy and conformity to the syllabus. Examination scripts (which were all anonymous) were marked by the leading lecturer. Scores from the final examination (60%) were combined with scores from two tests and four assignments (40%). After the examination and publication of results, the researchers performed the statistical analysis to find out the degree of difference on scores between the experimental group and the control group.

6. Results

Statistics on the final scores for the control and experimental group are summarized in Table 1.

Groups	Number	Total scores	Means	STD
Control	92	5728	53	1.543
Experiment	14	797	56.9	2.764

Table 1- The examination scores summary

Using final scores, between the control and the experimental group, where the experimental group used MTL, to learn programming without teacher's intervention while the control group was instructed through the conventional approach, results suggest a significant difference statically (A one tailed student's T-test, R: t=1.921>1.771, p=0.05). These results agree with claim by Wilson and Moffat (2010) that programming may not be a difficult subject; rather, it is the way it is presented to the learners which breeds confusion leaving the learners with various misconceptions which frustrate the effort to learn. With the MTL, aspects such as variable declaration, assignment, variable overwriting and data operations are made obvious at the very beginning.

As demonstrated in Figures 2 through 8 and in (Mselle 2011b), with MTL, the why variables are declared is made obvious and what happens to each variable during any action is clearly distinguished. What and how various operations are carried out and the meaning of *assignment* and *roles of variables* are clearly demonstrated at every turn of the code. Regarding functions/methods; issues such as *function call* and *parameter passing* are clearly visualized. Confusion and ambiguities are potentially mitigated through visualization with MTL.

7. Discussion

The experimental group used MTL to illustrate the execution of code from the machine point of view. In its core, MTL cultivates the sense of "*I am working the machine*" on the part of the learner. On the other hand the control group had no means to visualize their codes. Lack of a tool to illustrate the effect of each line of code on the machine is a major source of misconceptions (Perkins et al. 1986).

Du Boulay et al. (1981) proposes for a tool representing a notional machine. Du Boulay et al. (1981) advises that such a device should observe simplicity, be small and have few constructs. He argues in favor of implementing a language in such a way that, either pictorial or written traces can be displayed. MTL, is a programmer-driven visualization device which bears most of these characteristics (Mselle 2010, 2011a, 2011b, 2011c), (Samurcay 1985). MTL being programmer-driven, has capabilities to transfer programming authority to the programmer while creating the sense that the machine is not responsible for the mistakes committed by the learner.

7.1 MTL vs. Machine-based Animation

Popular animation tools such as Blue J, Jeliot and Plan Ani have been reported to be effective in enhancing programming comprehension (Ben Bassat et al. 2001), (Kuittinen et al. 2003, 2008). Machine-based animations are suitable for precise close tracking through the machine. In effect, they are a plausible break through. Nevertheless, since they are entirely machine-driven they concentrate most authority to the machine at the expense of the learner. In contrast, MTL is an absolutely learner-driven parse/visualizer. Program parsing by using MTL allows the novice to visually step line-by-line through a piece of code. Using MTL the learner can actively reveal the history of variables, and how the machine reacts when each statement is executed.

Mselle (2010, 2011a, 2011b , 2011c) has shown that MTL allows the learner to play-back the code from machine point of view. MTL is a sketch-based language that provides the learner with the absolute authority over the machine. Since MTL is a learner-driven language, it is unconstrained by the initial design of the code. To its additional credit, MTL can be used in conjunction with other animators and flow charts to capture advantages suggested by Ziegler and Crews (1999). MTL is an instrument for the learner to play the role of compiler outside of machine environment, putting the learner at par with the machine on verification of the correctness or incorrectness of the program (Perkins et al. 1986). Furthermore, MTL can be integrated into the current programming materials (Mselle 2011b) a quality not yet attained by the current visualization tools (Naps et al. 2003). These factors seem to provide explanation why the results observed in the experiment are positive.

8. Conclusion

The objective of this study was to test the impact of MTL when used as a learning tool without the intervention of a lecturer. Specifically, MTL has been proved to be a handy sketch language for learners to parse, track, debug and understand their programs without teachers' or machine intervention. Although the subjects of the experiment were not absolute novices their characteristics do not substantially differ from those of absolute novices because all of them were learning Java for the first time. Initial results are encouraging though far from conclusive.

There are, obviously, some shortcomings in this study. The sample size is too small to justify generalization. The population is taken from one university. The students of the experiment group could have been better than those of the control group.

9. Recommendations

More experiments in different settings should be carried out with a much bigger and diverse sample to confirm the effectiveness of MTL. More areas of research on effectiveness of MTL in distance learning, and different age groups are open for future investigation.

References

Ben-Ari, M. and Sajaniemi, J. (2003) Role of Variables from the Perspective of Computer Science Educators. DOI=http://cs.joensuu.fi/pub/Reports/A-2003-6.pdf.

Ben Bassat, L.R., Ben Ari, M. and Uronen, P. (2001) An Extended Experiment with Jeliot 2000. In Proceedings of The First International Program Visualization Workshop. University of Joensuu, Pavoo Finland, September, 2001, 131-140.

Dehnadi, S. and Bornat, R. (2006) The Camel has Two Humps (working title). School of Computing, Middlesex University, UK.

Du Boulay B., O'Shea, T. and Monk, J. (1981) The Black Box Inside the White Box: Presenting Computering Concepts to Novices. *International Journal of Man-Machine Studies*, 14, 237-249.

Du Boulay, B. (1986) Some Difficulties of Learning to program. *Journal of Educational Computing Research*, 2(4), 459-472.

Hundhausen, C. D. and Brown, J. L. (2007) What you See is what you Code: A 'live' Algorithm Development and Visualization Environment for Novice Learners. *Journal of Visual Languages and Computing*, 18(1), 22-47.

Kuittinen, M., Tikansalo, T. and Sajaniemi, J. (2008) A study of the Development of Students' Visualizations of Program State During an Elementary Object-Oriented Programming Course. *ACM Journal of Educational Resources in Computing,* 7(4).

Kuittinen, M. and Sajaniemi, J. (2003). First Results of an Experiment on Using Roles of Variables in Teaching. In M. Petre & D. Budgen (Eds) Proc. Joint Conf. EASE & PPIG 2003.

Mselle, L. (2010) The Impact of RAM Diagrams in Enhancing Comprehension in Programming: Class Experiment. http://www.ppig.org/papers/22nd-Teach-4.pdf

Mselle, L. and Mmasi, R. (2011a) The Impact of MTL on Reducing Misconceptions in Programming. DOI=http://dl.acm.org/citation.cfm?id=1999901&dl=ACM&coll=DL&CFID=38905039&CFTOKEN=61705367.

Mselle, L. (2011b). *Java for Novice Programmers*. LAP LAMBERT Academic Publishing, Berlin.

Mselle, L. (2011c). “Using Formal Logic to Formalize MTL on the Mould of RTL.” PPIG 2011, The University of York, UK.

Naps, T., R¨oßling, G., Almstrum, V., Dann, W. Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. and Vel´azquez-Iturbide, A. (2003). Exploring the Role of Visualization and Engagement in Computer Science Education. *ACM SIGCSE Bulletin*, 35(2), 131–152.

Perkins, D. N., Hobbs, H. R, Martin, F. and Simmons, R. 1986. Conditions of Learning in Novice Programmers. *Journal of Educational Computing Research*, 2(1), 37-55.

Samurcay, R. (1985) The Concept of Variable in Programming: Its Meaning and Use in Problem-Solving by Novice Programmers. *Education Studies in Mathematics*, 16(2), 143-161.

Wilson, A. and Moffat, D. (2010) Evaluating Scratch to Introduce Younger Schoolchildren to Programming. In Proceedings of the 22nd Annual Psychology of Programming Interest Group (Universidad Carlos III de Madrid, Leganés, Spain, September 19-22, 2010). Joseph Lawrence and Rachel Bellamy, editors.

Ziegler, U. and Crews, T. (1999). An Integrated Program Development Tool for Teaching and Learning how to Program. In Proceedings of the 30th SIGCSE Symposium, March 1999, 276-280.