Using Visualization To Teach Novices Recursion

Wanda Dann
Computer Science Dept.
Ithaca College
Ithaca, NY 14850
1-607-274-3602
wpdann@ithaca.edu

Stephen Cooper
Computer Science Dept.
Saint Joseph's University
Philadelphia, PA 19131
1-610-660-1561
scooper@sju.edu

Randy Pausch
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
1-412-268-3579
pausch@cs.cmu.edu

ABSTRACT

This paper describes an approach for introducing recursion, as part of a course for novice programmers. The course is designed to make use of a 3-D animation world-builder as a visualization tool that allows students to see their own programs in action. One of the pedagogical goals of the course is to enable the student to gain an intuitive sense of and mathematical insight into the recursive process. The software, examples of animation using recursion, and some experiences in using this approach are discussed.

1. INTRODUCTION

Recursion is a key computer science concept. As argued by Dan McCracken, "recursion is fundamental in computer science, whether understood as a mathematical concept, a programming technique, a way of expressing an algorithm, or a problem-solving approach." [11]

Laying a firm foundation in recursion in early courses may make topics studied in later courses easier to grasp. Unfortunately, novice programmers seem to struggle with the concept of recursion. Some computer science educators have described the process of teaching recursion as "one of the universally most difficult concepts to teach." [7] Bower [2] states that "(t)o understand the process of recursion and to be able to write a recursive definition or program one must be able to visualize the nature or structure of a problem and how solutions to smaller, similar problems are combined to solve the original problem." This paper presents an approach to introducing recursion to novice programmers with a 3-D animation tool that visualizes the recursive execution process.

2. Approaches To Teaching Recursion

Many approaches have been used for introducing recursion. Wu, Dale, and Bethel [19] summarized five widely used approaches: Russian Dolls, Process Tracing, Stack Simulation, Mathematical Induction, and Structure Templates. Each approach has some complication such as applicability or assumed level of student maturity.

Another approach to teaching recursion employs algorithm animation tools (e.g. XTANGO [18], BALSA [3]). The instructor programs an animation for commonly used algorithms (e.g. a quicksort animation shows little shapes bouncing around from one array slot to another). The student runs the *prepared* animation, observing the animation of the algorithm when using different inputs. The Generalized Algorithm Illustration through Graphical Software (GAIGS) package, developed by Naps [12] offers the ability to rewind a prepared animation to view it again. Algorithm animation tools have contributed to helping students understand algorithm analysis, but students can only view previously prepared animations. Bower [2] writes, "(i)t is important that such simulations are manipulative in order to actively involve the learner rather than...passively watch pre-programmed instruction."

Yet another approach to teaching recursion uses program visualization. Program visualization tools differ from algorithm animation tools in that program visualization directly relates individual lines of a student's own program code to the animation. Some general-purpose program visualization tools have been designed for use by beginners (e.g. *Karel, The Robot* [14]). Karel was originally used to prepare students for success in learning Pascal, and has undergone several updates, the latest being *Karel++* [1]. Karel++ can be used to introduce recursion by writing tail-recursive functions for the robot. Similarly, LOGO [13] can be used to demonstrate some recursive concepts.

Our approach to teaching recursion to novices centers on a software system named Alice, similar in flavor to Karel and LOGO. Alice is a convenient and easy-to-use 3-D graphic animation tool that supports the pedagogical goals of the course, i.e. a fundamental introduction to objects, methods, decision statements, loops and recursion. In terms of recursion, students are encouraged to develop an intuitive sense of recursion and to gain some mathematical insight into the recursive process. The use of 3-D graphics in a first programming course follows in the footsteps of House and Levine[10] who used Jabka to render 3-D models in a computer graphics course for non-majors.

3. What is Alice?

Alice98 (www.alice.org) is a 3-D interactive graphics programming environment for Windows built by the Stage 3 Research Group at Carnegie Mellon University under the direction of Randy Pausch [15]. Alice offers a full scripting and prototyping environment for 3-D object behavior (e.g., animals

and vehicles) in a virtual world. Alice has an object-oriented flavor. By writing simple scripts, Alice users can control object appearance and behavior. Alice is built on top of the Python language (www.python.org) and uses many of Python's features. Readers interested in using Alice may download the free program from www.alice.org. Additional information on programming with Alice is available at www.ithaca.edu/-wpdann/alice1298.

Alice serves as a programming language environment where students can immediately see how their programs run. The highly visual feedback allows the student to connect individual lines of code to the animation action. Many of the commands and animation actions have been previously discussed [4, 5] and will not be covered here.

4. Using Recursion in Alice

Alice supports recursion as a means of creating more powerful animations with repeated actions that get closer and closer to completing a task. Below, two different examples of animations that use recursion are illustrated.

4.1 Tail recursion

A typical game-playing example is an animation of a chase scene. In Figure 1, the butterfly moves in a random direction and the rabbit moves in pursuit.

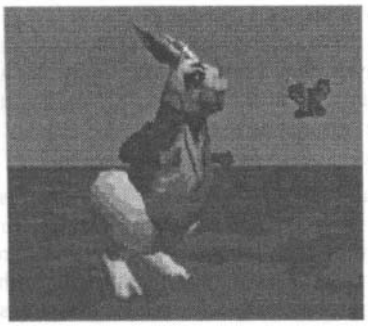

Figure 1. Rabbit and Butterfly Chase

```
def bmove():
    if Rabbit.DistanceTo(butterfly) > 1.0:
 DoTogether(
    DoInOrder(
       Rabbit.PointAt(
          (butterfly.GetPosition()[0], 0,
           butterfly.GetPosition()[2])),
       DoTogether(
          butterfly.MoveTo(
             butterfly.GetPosition()[0]
                 + range(-.25,.25),
             range(0,1),
             butterfly.GetPosition()[2]
                 + range(-.25,.25)),
         Rabbit.Move(Forward,.5)),
      ),
   Alice.SetAlarm(3, do(bmove))
```

If the Rabbit is within one unit of the butterfly, the chase has ended. Otherwise, three actions occur. The Rabbit points itself toward the current location of the butterfly (actually, to the point directly beneath the butterfly). Then, the Rabbit moves towards the butterfly while the butterfly moves in a random direction. The range function generates a random number within a specified range.

Finally, an alarm is set to recursively call the function to do the whole process again. The **SetAlarm** instruction specifies an amount of time to wait before calling a function. In the above code, the alarm waits 3 seconds before the recursive function call. The purpose of the wait is to allow an animation or sequence of animations to complete. If the recursive call were made immediately, the animation would not appear to work properly. The butterfly's height (the second coordinate in the MoveTo instruction) must stay between ground level and 1 unit above the ground because the butterfly could potentially "fly" away, and the recursion would never terminate.

4.2 Multiple recursion

The Towers of Hanoi problem is used as a slightly more complicated example.

```
def towers (n, frompeg, topeg, spare):
  if n == 1:
     moveit(1,frompeg, topeg)
  else:
     towers(n-1, frompeg, spare, topeg)
     Alice.SetAlarm(math.pow(2,(n-1))-1,
       do(moveit,(n,frompeg,topeg)))
     Alice.SetAlarm(math.pow(2,(n-1)),
        do(towers,(n-1, spare,
            topeg,frompeg)))
```

If there is only 1 ring to move, it moves (the base case). Otherwise (the recursive case), n-1 rings move onto the spare peg. An alarm is set so that after the n-1 rings have moved onto the spare peg, the bottom one can move. Finally another alarm is set so that after the bottom ring moves onto the target peg, the n-1 rings can move onto the target peg. An important aspect of this problem is that the student becomes aware that the time required for the animation is directly related to the number of moves needed to move n-1 rings from the original peg to the spare peg. In experiments with different numbers of rings students gradually recognize the pattern (and develop an abstract notion of the recursive decomposition).

Depending on the source and target pegs, a ring move involves an appropriate distance in a specific direction. Figure 2 shows a move in progress. The `moveit()` function accomplishes this task, returning the correct motion animation as a result. (Moving forward a negative amount is the same as moving back.) The `which()` function provides a correlation between the hard-coded name of a ring and its corresponding number.

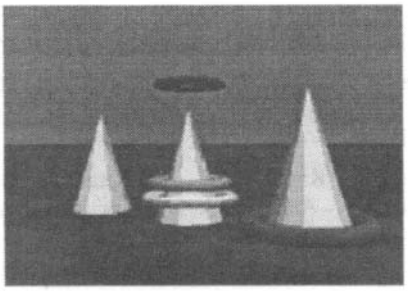

Figure 2. Towers of Hanoi In Progress

```
def moveit (num, frompeg, topeg):
  return which(num).move (forward,
       (topeg-frompeg))
```

While the Towers of Hanoi problem is an example of structural recursion, the rabbit and butterfly example uses generative recursion. Although generative recursion is often not taught to novice programmers, we found students' insight into chase scenes and game-playing provided a context for understanding a generative solution.

5. Our Experiences

Our course has been used in various formats since 1998 at Ithaca College (gifted and talented special summer programs and a credit-bearing college course for students with no previous programming experience). The course is being conducted again this fall and we continue to collect data for statistical analysis.

The course is recommended to students who have no previous programming, before enrolling in CS1. Recent studies [6, 9] have shown that students without prior programming experience are at a decided disadvantage in being able to complete a computer science degree program (as compared to those students who had prior programming experience). We hope that teaching fundamental concepts in a preliminary course, particularly with regard to the presentation of recursion material, will provide sufficient "prior programming experience". Not enough time has passed to track the degree to which Alice has helped students who went on to be computer science majors.

In conducting the course, recursion was introduced in the 3rd or 4th week. For students, the motivation to learn recursion was to solve a problem, namely the need for an unknown number of repeated actions. (Alice provides a **loop** construct that implements *for* loops but no *while* loop -- although while loops could be implemented by using the underlying Python language.) Students begin by trying to write their recursive programs in a way similar to code presented by the instructor but soon make two kinds of mistakes. The first kind of mistake involves technical issues concerning recursion. These mistakes included not making the sub-problem smaller (the situation became "worse"), missing base cases (the animation kept going when it isn't supposed to), and premature stopping (a base case hits when it should not). Many of these mistakes were quickly and easily identified as students watched their virtual worlds.

The second kind of mistake involved timing issues. Different sequences of actions require different amounts of time to complete the animation. In order to determine the delay time needed for their own animation, students must consider the real-time requirements of a particular sequence of animated actions in order to have the recursive call occur at the right time. This leads to an exploration of how the run-time is related to the number of recursive moves. While students develop an intuitive sense of how long to wait by experimental means, the ultimate proof is an inductive one. Thus, timing the recursive call can be used to provide a basis for gaining an understanding of the run-time of the algorithm.

Student responses on follow-up questionnaires indicate that students had little difficulty understanding recursion and were able to use recursion in writing their own programs. Students demonstrated a high level of involvement. They enjoyed watching their recursive programs *in action*, viewing the impacts of their recursive calls as their worlds responded (not always in an anticipated manner). We emphasize that students did not necessarily "master" the intricate concepts of recursion, but they did experience working with recursion and exhibited some facility with using the technique.

6. But does Alice really provide recursion?

Recursion may be understood from an external, or perhaps abstract perspective. Knuth defines a recurrence to be a situation where "...the solution to each problem depends on the solutions to smaller instances of the same problem." [8] And Roberts states that "...informally, *recursion* is the process of solving a large problem by reducing it to one or more *subproblems* which are (1) identical in structure to the original problem and (2) somewhat easier to solve." [16] By these definitions, Alice does support recursion.

However, recursion in Alice is implemented in a different fashion than in many other languages/environments. The main reason is that a direct recursive call to a Python function does not wait for animated actions in Alice to complete. Since the recursions necessarily involve animation, the SetAlarm method is used to delay the recursive call to allow the animation instructions to complete. In fact, from an implementation point of view, stacking of environments (typically associated with recursive calls) is not involved. The function schedules the recursive call to be made after a specified time delay and then terminates. Since programming in Alice tends to avoid the use of local mutable variables, the issue of repeated instances of a variable does not exist. (See [5] for a deeper discussion of the "state-less" nature of programming in Alice worlds.) The only mutable variables generally needed are the global objects (which inhabit Alice worlds) themselves.

In essence, much of the internal complication involved with recursion, both from the machine standpoint as well as from an explanation standpoint (whether traces or a stack) are not present in Alice. Repeated recursive function calls for the animated actions do not get stacked on top of one another. So, some might argue that Alice animations do not provide true recursion (except in the underlying Python language). While this may or may not be true, our goal is that visualization be used to develop good intuitions about recursion. Thus, we contend that visualization of recursion is more important than whether Alice animations do or do not support true recursion.

7. Conclusion

Using 3-D animations for program visualization offers computer science instructors an approach to introducing fundamental concepts such as recursion to novice programmers. Introducing recursion using 3-D animations employs a combination of visualization, experimentation, and mathematical explanation. Benefits of this approach include a high level of student involvement and the ability to develop an intuitive understanding of basic concepts in a visual feedback environment, where students see their own programs in action. Recursion can be presented as a means to create more powerful (and interesting) animations with repeated actions that get closer and closer to completing the task. Visualizing recursion allowed students to quickly identify and learn from mistakes.

In the future, a follow-up study will assess whether experience with Alice will help students when they learn more about recursion in CS1/CS2. While our experiences involved using 3-D animations to introduce fundamental programming concepts in a pre-CS1 course, we anticipate that this approach could also be used during the first couple of weeks of a CS1 class. This would follow the approach presented by Scragg et al.[17] who recommended introducing students to basic computer science concepts before launching into the traditional CS1.

8. ACKNOWLEDGEMENTS

Alice and the Stage3 Research Group are sponsored by DARPA, NSF, Intel, Chevron, Advanced Network & Services, Inc., Microsoft Research, PIXAR, and NASA.

9. REFERENCES

[1] Bergin, J., Stehlik, M., Roberts, J., and Pattis, R., *Karel++, A Gentle Introduction to the Art of Object-Oriented Programming*. New York: Wiley, 1997.

[2] Bower, R.W. *An investigation of a manipulative simulation in the learning of recursive programming*. PhD thesis. Iowa State University, 1998.

[3] Brown, M.H., *Algorithm Visualization*. Cambridge, MA: M.I.T. Press, 1988.

[4] Cooper, S., Dann, W., and Pausch, R. Alice: a 3-D tool for introductory programming concepts. In *Proceedings of the 5th Annual CCSC Northeastern Conference*, Mahwah, NJ, (2000), 107-116.

[5] Dann, W., Cooper, S., and Pausch, R. Making the connection: programming with animated small worlds. In *[20]*, 41-44.

[6] Davy, J.D., Audin, K., Barkham, M. and Joyner, C. Student well-being in a computing department. In *[20]*, 136-139.

[7] Gal-Ezer, J. and Harel, D. What (Else) Should CS Educators Know? *Communications of the ACM* 41, 9 (September 1998), 77-84.

[8] Graham, R.L., Knuth, D. E., and Patashnik, O. *Concrete Mathematics*. Addison-Wesley Pub. Co., Reading, MA 1989.

[9] Hagan, D., and Markham, S. Does it help to have some programming experience before beginning a computing degree program? In *[20]*, 25-28.

[10] House, D. and Levine, D. The Art and Science of Computer Graphics: A Very Depth-First Approach to the Non-majors Course. In *Proceedings of the 25th SIGCE Technical Symposium*, Phoenix, March, 1994.

[11] McCracken, D.D. Ruminations on Computer Science Curricula. *Communications of the ACM*. 30, 1: (January 1987), 3-5.

[12] Naps, T. and Swander, B. An object-oriented approach to algorithm visualization – easy, extensible, and dynamic. In *Proceedings of the 25th SIGCE Technical Symposium*, Phoenix, March, 1994.

[13] Papert, S., *MindStorms: Children, Computers, and Powerful Ideas*. New York: Basic Books, 1980.

[14] Pattis, R., *Karel the Robot*. New York: Wiley, 1981.

[15] Pausch, R. (head), Burnette, T., Capeheart, A.C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S., White, J. Alice:rapid prototyping system for virtual reality, *IEEE Computer Graphics and Applications*, May, 1995.

[16] Roberts, E.S. *Thinking Recursively*. John Wiley & Sons, Inc., New York, 1986.

[17] Scragg, G., Baldwin, D., and Koomen, H. Computer science needs an insight-based curriculum. In *Proceedings of the 25th SIGCSE Technical Symposium*, Phoenix, (1994), 150-154.

[18] Stasko, J.T., Dominque, J., Brown, M. and Price, B., eds. Software Visualization, Programming as a Multimedia Experience. Cambridge: MIT Press, 1998.

[19] Wu, C., Dale, N.B., and Bethel, L.J. Conceptual Models and Cognitive Learning. Styles in Teaching Recursion in *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*. Atlanta, 292-296.

[20] *Proceedings of the 5th Annual Conference on Innovation and Technology in Computer Science Education*, Helsinki, Finland, (2000).