

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

Visual Iconic Object Oriented Programming
to Advance Computer Science Education and Novice Programming

A Thesis

by

Nancy Silva Martinez

Submitted to the Graduate School of the
University of Texas – Pan American
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2001

Major Subject: Computer Science

UMI Number: 1405526

Copyright 2001 by
Martinez, Nancy Silva

All rights reserved.

UMI[®]

UMI Microform 1405526

Copyright 2001 by Bell & Howell Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

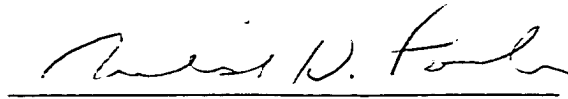
**Visual Iconic Object Oriented Programming
to Advance Computer Science Education and Novice Programming**

A Thesis

by

Nancy Silva Martinez

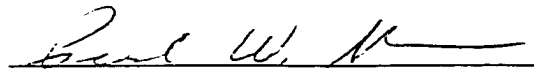
Approved as to style and content by:



Richard H. Fowler, Ph. D.
Chair of Committee



Wendy A. Lawrence Fowler, Ph. D.
Committee Member



Pearl Weaver Brazier, M. S.
Committee Member

August 2001

ABSTRACT

Nancy Silva Martinez, Visual Iconic Object Oriented Programming to Advance Computer Science Education and Novice Programming, Master of Science, Computer Science, August 2001, 33 pages, 9 figures, 50 references, 39 titles, and 11 urls.

Learning how to program a computer is difficult for most people. Computer programming is a cognitively challenging, time consuming, labor intensive, and frustrating endeavor. Years of formal study and training are required to learn a programming language's world of algorithms and data structures. Instructions are coded in advance before the computer demonstrates the desired behavior. Seeing all the programming steps and instruction code is complicated. There exists a tremendous gap between the representations the human brain uses when thinking about a problem and the representations used in programming a computer. Often people are much better at dealing with specific, concrete objects than working with abstract ideas. Concrete and specific programming examples and demonstrations can be very useful. When cleverly chosen and properly used, programming examples and demonstrations help people understand the abstract concepts. Programming by example or demonstration attempts to extend these novel ideas to novice programming.

ACKNOWLEDGEMENTS

The author wishes to express sincere appreciation to professors, Dr. Richard H. Fowler and Dr. Wendy A. Lawrence Fowler, for their assistance in preparing this thesis. Special thanks are due to Dr. Xiannong Meng, Dr. Zhixiang Chen, Professor Pearl Brazier, Dr. Richard K. Fox, and Dr. John P. Abraham, professors and mentors. Special recognition is given to Dr. Jacob Jen-Gwo Chen, former Dean of the College of Science and Engineering, Elvie Davis, Dean of Students, and Dr. Miguel A. Nevarez, President of The University of Texas – Pan American.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
ACKNOWLEDGEMENTS	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	vi
1. Introduction	1
1.1 Initial Ideas	1
1.2 Growth and Development	5
2. Visual Programming	7
2.1 Tools and Techniques	7
2.2 Novice Programming	11
3. Statement of the Problem	16
4. Overview	17
5. Approach	18
6. Implementation.....	19
7. Conclusion.....	20
8. Limitations	21
9. Future Work	22
References	23

LIST OF FIGURES

	Page
FIGURE – 1 LEFT AND RIGHT HEMISPHERES OF THE HUMAN BRAIN	2
FIGURE – 2 GRAPHICAL REPRESENTATIONS OF ALGORITHMS	8
FIGURE – 3 VISUALIZING ALGORITHMS.....	10
FIGURE – 4 GRAPHICAL REPRESENTATIONS OF ALGORITHMS.....	11
FIGURE – 5 VISUALIZING ALGORITHMS.....	12
FIGURE – 6 PROGRAM VISUALIZATION.....	13
FIGURE – 7 INTERACTIVE ANIMATED PROGRAM VISUALIZATION	14
FIGURE – 8 ANIMATION OF ALGORITHMS.....	17
FIGURE – 9 VISUAL ICONIC OBJECT ORIENTED PROGRAMMING.....	19

1. INTRODUCTION

The human visual cortex occupies most of the brain. Icons appeal to users because icons engage brain capacity evolved for other purposes. The benefit is that everyone has the ability to easily understand icons and will evaluate their significance similarly to objects and events in a three dimensional world. The more engaging and life like figures are the more intelligent, conforming, and friendly they will be to the novice programmers [9]. Visual iconic object oriented programming is a pedagogical, integrating, discovery agent exploring human and computer interactions on the Internet. Enhanced perceptions make better experiences. Three dimensional space requires multiple degrees of freedom which personalized perceptual intelligent user interfaces will provide in real time in the future.

1.1 INITIAL IDEAS

Novice programmers can use pictures to communicate with computers and serve as a medium to teach programming in introductory computer science courses. The human brain's left hemisphere is responsible for linguistic ability while the right hemisphere perceives visual patterns. The left hemisphere of the brain thinks analytically and logically while the right hemisphere thinks in an intuitive and artistic sense. Programming has been considered to be an activity of the left hemisphere requiring analytical, logical, and verbal ability. However, visual programming concentrates more on the capabilities of the right hemisphere (Figure 1).

Visual programming is directed towards increasing the productivity of professional programmers and facilitating learning for novice programmers. The phenomenal growth of end user computing necessitates a larger user population and a greater number of novice programmers. The majority of potential computer users are not computer professionals. However, software is still designed as if users are professional programmers. There has been a lack of emphasis on human convenience and a concentration on machine efficiency because computing costs were expensive. Efficiency took precedence over ease of programming. The challenge is to bring computer capabilities usefully and easily to people with no special computer training [1].



Figure 1: Left and Right Hemispheres of the Human Brain.

Icons, pointing devices, and menus have diminished the need to remember commands. However, the tools are limited to a particular predefined category of applications. A radical departure from traditional programming is necessary if programming is to be made more accessible to a large population. As the level of the generations of programming languages has gone up, fewer details have been required from the user. The tradition of linear representations to give instructions to the computer in a statement by statement manner have persisted. The structure of programming languages is still one dimensional and textual.

In visual programming, graphical representations and pictures play a part in the programming process. Pictures concisely convey more meaning than words and are in

some ways easier to understand than text. People understand pictures regardless of the language they speak, and icons can be a means of communicating with computers. People from varied backgrounds have developed visual programming. Visual programming uses meaningful graphic representations in the process of programming. The visual environment deals with the visualization of data or information expressed in graphical form and presented to the user in a spatial framework. Visual systems are devoted to direct manipulation for information retrieval and graphical views for the visualization of the information retrieved. Visualization of software design focuses on a software development environment with requirements, specifications, design decisions, and system structures all in graphical form.

In graphical programming, a user does not need to mentally visualize the effects of instructions while constructing a program. The effects take place on the screen through a programming process relying on interactive graphics with the emphasis on the interaction and not the language. The major areas of visual programming are the visual environment and the visual languages. Visual languages include languages for handling visual information, for supporting visual interaction, and for programming with visual representations.

Visual languages process image information, manipulate and query pictorial data, and allow direct reference to pictures. However, even though the visual language information involves pictures, the visual languages themselves are textual. Icons and graphical objects are being used to communicate with computers. Icons and pictures need software to be animated in the programming world. Visual languages define, create, and manipulate pictorial symbols. Visual representations and visual interactions are supported by visual languages that are themselves textual and not visual. Visual programming languages allow users to program with visual expressions. Pictorial representations are used to carry out programming concepts with graphical representations designed as an integral part of languages. Icons and graphical symbols are deliberately designed to play the central role in programming.

A visual language is a set of iconic sentences constructed with a given syntax and semantics. Icons are applicable to the design of icon oriented user interfaces. A visual language compiler can accept the iconic system, the icon operators, and definition of basic icons as input. The visual language compiler can parse an iconic sentence to determine its syntactic structure and semantic meaning. An icon oriented system is generated by the visual language compiler if the initial design is satisfactory. Iconic dual representation of objects and semantics can be a unified methodology for visual language design and icon oriented system design.

In designing visual, useful, and usable interface systems, the goals are to improve internal and external communication among application users, expedite the decision making process, and improve the productivity of an organization. A visual interface design is the software directly interacting with the users with human behavior being very different from the other system components dealt with by the program. Visual interface

design increases the functions of the design tools and enables the integration of different medium and applications.

Images are virtual entities [4]. Icons help to achieve instant recognition of familiar symbols. Icons can be classified as pictorial and symbolic icons. Pictorial icons use pictures to represent the abstract and semantic information of operations. Symbolic icons are combined with some characters to help capture semantic information. Iconic interfaces must be consistent and unambiguous. A reasonable visual representation considers application and cultural dependence, easy recognition, distinction from other stylized icons within the system, and consistency in an environment.

A direct manipulation user interface allows the user to operate directly on the objects in the computer rather than carrying on a dialogue about them. Instead of explicitly invoking commands either through a command language or menu selection to specify the operations on the objects, the user manipulates objects visible on the screen to represent the action.

Distance and engagement give a feeling of directness. Distance is the mental effort required to translate thoughts to the physical requirements of the system. In the user interface, this distance involves a relationship between the task the user has in mind and the way the task can be accomplished via the interface. Using graphical representations to depict the objects and actions of the user interface offers the possibility for the user to reduce cognitive effort to accomplish the task. Choosing an appropriate user interface for the objects and actions required is critical to the effective use of direct manipulation. Direct engagement is the sense of manipulating objects directly on a screen rather than conversing about them. There is a feeling of direct involvement with the world of objects. In a direct manipulation interface, the world of interest is explicitly represented and there is no intermediary between the user and the world. The interface is itself a world where the user can act and that changes state in response to actions.

Animation has the ability to present a great deal of information in a short time and ascertains features not obvious in pictures. Color enhances illustrations, distinguishes and differentiates structures, and increases the amount of information presented. Both novices and experts must be accommodated.

An object oriented system provides standard objects for managing the user interface and a framework to guide and help the designer create and modify a specific user interface. The object abstraction allows encapsulation of both static data and object behavior. This approach supports the separation of the interaction between user and application in a natural way and enhances the development of direct manipulation interfaces.

A visual interactive design system uses direct manipulation interfaces as the basis for the construction of interface design tools allowing the designer to create the interaction techniques by demonstrating how the screen should appear and how the end user should interact with the display. Techniques of storyboards, visual programming, and programming by example are used in this interface construction style.

1.2 GROWTH AND DEVELOPMENT

Visual object oriented programming uses graphical techniques in computer programming and models systems in terms of objects. Visual object oriented programming uses an object oriented programming language with a visual syntax and an environment with graphical tools to manipulate programs written in a textual object oriented language. A class is an abstract description of a set of objects providing the same functionality and differing according to the values of their private information. Each class member specifying these values is a class instance. Polymorphism is the ability of different kinds of objects to respond to the same message. Class hierarchies are structured around function similarity with each subclass inheriting the specification of its super classes. The original object serves as a prototype for new objects.

A language grammar with graphical pictures and animations has a visual syntax. A visual syntax may have spatial information, containment, connected relationships, and visual attributes of location and color. A visual environment has tools using graphical techniques for manipulating pictorial elements and for displaying program structure. A visual environment for constructing a program may include point and click for action invocation or selection. A visual environment may also include selecting and relating objects by drawing a line from one object to another [3]. Lively visual programming languages provide immediate feedback.

Visual object oriented systems provide the reuse of a function through classes and their instances. Visual object oriented programming combines the reusability and extensibility of object oriented technology and the accessibility of visual programming enhancing the ability to create visual environments. Every kind of object can have complementary objects providing visualizations of the capability and status of the object. Individual pictorial representations display the functions an object supports and tools are used to draw links among objects.

One of the strengths of object oriented languages is their support for abstract specification in terms of classes and hierarchies of classes. The object oriented approach encourages a style of programming leading to modular architectures promoting reliability and reusability required for large scale programming. The merging of visual and object oriented programming technologies offers the largest possible productivity and reliability gains.

Universal methods may include built in primitives. Class methods may be represented as named icons in a methods window for the class. User defined universal methods may be represented by icons in a universal window. A method may consist of a sequence of cases where each case is a dataflow structure consisting of data inputs, data

outputs, and a set of operations with connections between them. The icon shapes for initialization, extraction, and assignment class methods may indicate the functions of these methods.

Many object oriented languages provide classes describing a collection of objects with common characteristics and objects representing instances of classes. Objects usually contain data and operations manipulating that data including instance variables and methods. Classes provide a means of encapsulation and information hiding like private instance variables and methods in classes. Classes may be described with the physical aggregation of data and methods and with the enclosure of them. A visual programming language supports objects, classes, and all the semantics related to creating, defining, and manipulating objects [6]. A visual programming language supports inheritance, polymorphism, and dynamic dispatch. Object oriented programming languages provide powerful features such as objects, classes, inheritance, polymorphism, and dynamic dispatch to facilitate solving large complex programming challenges.

Real world objects are identified by appearance and interface objects are identified by view. Visual shells are concrete representations giving the illusion of the concrete manipulation of objects. The analogy between an object's appearance and behavior in the physical world and the object's representation and behavior in the interface enables users to transfer knowledge from their real world experience to their computer task. Latency is the time taken by the user interface to respond when the user first initiates an action. Latency is preferably reduced. Feedback bandwidth is the rate at which information is presented to the user during an interaction after the initial latency time. Feedback bandwidth is preferably expedited.

Icons contain a set of graphical primitives describing their appearance. The color of any graphical object can be dependent on the value of another attribute from the same class to which the icon belongs. A strength of object oriented programming is its support for leveraging existing software. Object oriented programming classes can be used unchanged and easily can be extended for reuse in new applications. Visual programming can reduce the time and effort required to write and debug programs and can increase the programmer's accessibility to information about the behavior of the program [5]. Visual programming languages use graphics and other visual techniques to allow a programmer to express the desired logic of a program, to view and trace changes in the program during execution, and to understand how the program works.

Visual specifications can generate textual code. Objects should be visual and interactively accessible. A visual language environment is object focused with objects directly available for interaction and matching view focused environments through object availability and liveliness [2]. Programmers visualize programs in their minds because people deal with the concrete more easily than with the abstract. Program elements of manipulated data and behavior applied are made concrete through object encapsulation. Environmental immediacy and primacy of action present concrete object entities. Objects may be abstracted to reveal differing parts within multiple and separate tools.

View focused environments have interactive tools presenting views of objects. Object focused environments give the programmer direct interactivity with objects of the language. The sense of immediacy with the object semantic is a feeling of direct engagement. View focused environmental direct engagement is achieved by eliminating indirection by view focused tools and by having object focused environments increase direct engagement frequency making objects seem like physical real world things. The object's image and behavior is three dimensional in an artificial environmental reality [31]. Animation gives objects a smooth, lively, engaging, and realistic movement. Tool objects in the interface centralize functionality for activities. Multiple simultaneous views are useful in comparing separate aspects of the object at once. However, concrete objects should be in only one place at a time just like real world objects.

A lively interface allows objects to move, change, transform, and interact on their own. Tools show relationships among objects in view focused environments. Tools gather multiple objects to operate on several objects at once. The enforcement of the unique identities of objects makes them concrete. An object focused programming environment reduces the distance between the programmer's mental model of objects and the environment's representation of the objects.

2. VISUAL PROGRAMMING

Computer science students seem to be more visually oriented than ever before due to the visual influence of the existing era of real time video games and MTV. Students can learn by seeing a concept described with a picture and remember the concept by recalling the picture associated with the concept. Visual programming languages will process information, support human computer communicative interaction, and allow novice programmers with no highly trained programming skills to communicate with computers and program them with visual expressions [10]. Icons, pointing devices, graphics, and programming with interactive graphical support are central to giving programming capabilities to people who are not programmers by profession.

2.1 TOOLS AND TECHNIQUES

Research by M. Raner [33], E. Odekirk, D. Jones, and P. Jensen [34], A. E. Fleury [35], R. Rasala, J. Raab, and V. K. Proulx [36], R. Morelli, R. Walde, and G. Marcuccio [37], suggests icon based programming languages are more effective than text based languages in teaching introductory programming procedural and object oriented concepts to computer science students [40]. Although R. Pattis [38] and R. Rasala [39] state otherwise, emphasizing text based languages. Iconic programming with icons

representing all the major programming constructs and data structures within a syntax directed environment allow students to generate syntactically correct code for any one of several text based languages. The iconic programming language environment allows students to program with icons representing all the major programming constructs including loops and conditional branching within a syntax directed environment generating syntactically correct code for any text based language.

Graphical representations of algorithms are an effective methodology for learning programming skills (Figure 2) [8]. Iconic programming was first developed for high level workstations because of their speed and memory requirements. More visually based programming environments are available due to the rapid progress in high speed, high resolution, large random access memory, and inexpensive computers. A graphical representation for the flow of control increases semantic accuracy by accurately conveying to students which variables of the proper type can be used at a particular point in the algorithm. Object oriented constructs define and implement class icons in a visual environment. Attributes and methods are declared in a manner similar to declaring variables and subroutines with attributes and methods being declared as public, private, or protected. Iconic representations support the strengths of the object oriented paradigm.

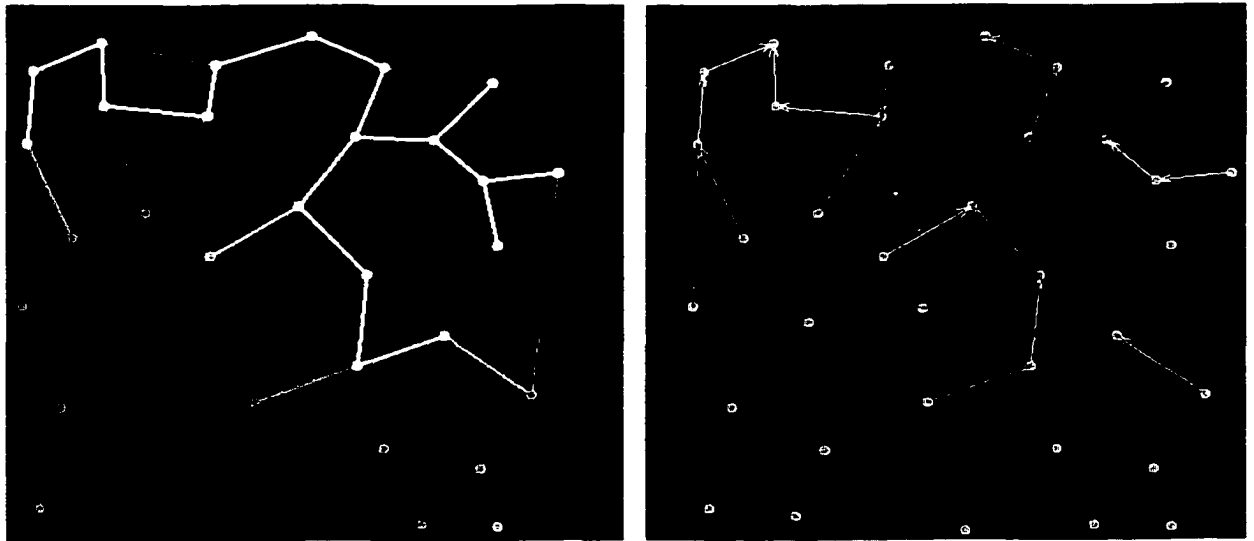


Figure 2: Graphical Representations of Algorithms

Conventional programming environments give feedback to a programmer that does not provide an overall view of the program's behavior [41]. Conventional languages only symbolize ideas and do not visualize or illustrate the ideas. The symbols are arbitrary and do not provide an actual visualization. The key ideas of object orientation need to be visualized. Visualization and illustration are required for the

aspects of encapsulation and abstraction, instantiation of objects, usage of references and pointers, inheritance and its consequences, method overriding, polymorphism, member protection and visibility, abstract classes, and separation of interface and implementation. The visualizations can be animations making use of shades and shadows for emphasis. Students might be taught introductory programming with visual programming environments based on the Java programming language emphasizing building user friendly interfaces and not just algorithms.

Java has many excellent features for teaching sequential programming. The Java development kit for the Java language includes an abundant collection of class libraries application programming interfaces for data structures, input and output, networking, remote procedure calls, and graphics [42]. Students can be introduced to object oriented concepts such as classes, instances, attributes, and methods. Object oriented programming in Java is more complicated than procedural programming and object oriented programming benefits larger programs.

Animation of data structures provides students with an alternative view in understanding a newly presented data structure or algorithm and helps students debug a program using the data structure [12]. An interactive animation allowing students to try different input can be easier to understand and remember than a textual representation. Using animations to debug programs helps students find errors quicker by viewing incorrect movement or seeing pieces incorrectly disconnected. Java and Web based algorithm animations are relatively easy to use architecture independent methods for creating data structure animations over the Web.

Students can write Java animation programs and display the animations with a Web browser supporting Java. Few have leveraged the full capabilities of the Web and Java. The popularity of the Web browser is due to its friendly visual interface [43]. The Web can be used to build courseware modules in order to aid student understanding. Objects that can be manipulated are primitive objects like a geometrical shape and intelligent objects representing data structures with specific commands to perform operations. Animation brings alive programs allowing students to easily see each instruction and its effects. Software visualization systems allow custom animations that capture the dynamics of programs.

Animations or visualizations offer feedback that benefits different learning styles (Figure 3). Visualization and animation can simplify viewing programs by focusing on either objects changing over time or displaying a large amount of data too complex or tedious in a textual manner. Animations and interactive graphics can generate student interest and enthusiasm which can lead to better comprehension and mastery of the material in computer science courses [14]. The visual component of the animations offers another dimension assisting students with the difficult task of debugging. Animations are accessible on all platforms running Java.

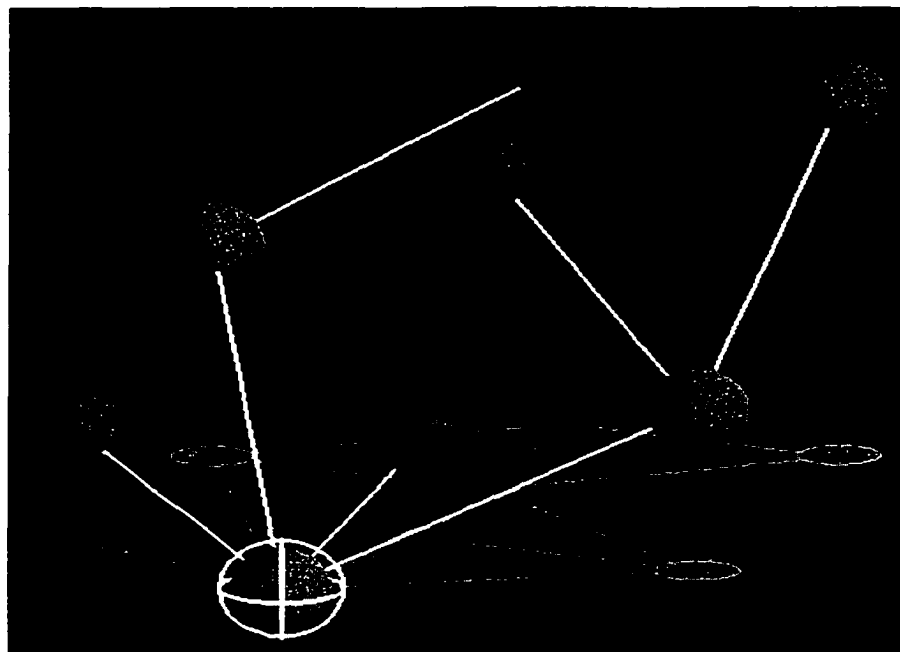


Figure 3: Visualizing Algorithms

Computer science students can benefit from exposure to concurrent programming, event driven programming, graphics management, enhanced human computer interfaces, data compression, image processing, and genetic algorithms. A Java hosted algorithm visualization client and server environment for delivering algorithm visualizations over the Web will contribute to the pedagogical effectiveness of algorithm visualization for students [44]. Showing animations with source code helps keep students on track when students deal with recursive algorithms or data structures because students can lose sight of which interval of arrays is still under execution.

Java is a popular first language for university students because of its power as an object oriented language contributing to students' comprehension of construction and use of objects [15]. Teaching can be enhanced when instructors listen to Java students' thoughts and interpret students' actions from the students' perspective. Constructivism is creating new knowledge by combining new experiences with current knowledge and reflecting on the knowledge. All learning involves the interpretation of phenomena, situations, and events. The experience of classroom instruction is interpreted through the perspective of the learner's existing knowledge. The power of object oriented programming is having more than one class declaration and more than one object of one of the classes. Beginning programming students can construct an understanding of the syntactic and semantic rules involving the construction and use of objects in Java through the study of object oriented programming.

2.2 NOVICE PROGRAMMING

Educational software should be expected both to give students quick intuition in areas which are not treated thoroughly in class and to enable students to investigate selected topics in more detail than would be possible with paper and pencil. Different students learn at different rates and through different methods. Self paced learning with the facility for replay for weaker students and with accelerated and exploratory interaction for stronger students is a considerable benefit to teaching novice programmers [45]. A quality education allows computer science students to contribute to their learning and to develop a sense of ownership of the subject. Interactive visualization tools for algorithms and computing models can provide a more compelling means of exploration and feedback than traditional paper and pencil methods in theory and lecture. Students can take advantage of exploratory, supplementary, Web accessible, graphical environments allowing the construction and simulation of algorithms and computing theory through experimental animations (Figure 4).

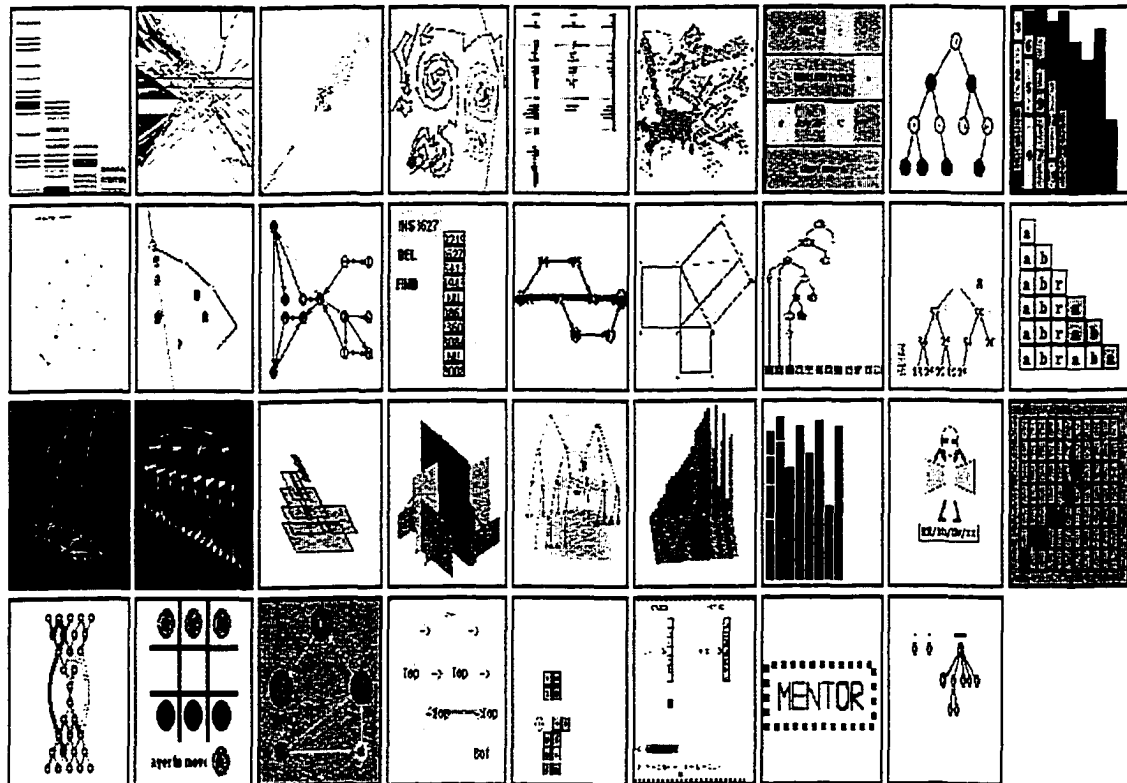


Figure 4: Graphical Representations of Algorithms

University computer science curricula require students to learn multiple programming languages. Students often have difficulty with the multi-lingual nature of the curriculum in learning and using different development environments and programming languages with a compiler or interpreter. The learning curve with the environments, not the languages, is most frustrating to students. Very few language environments provide the user with program development and comprehension aids such as automatic software visualization [46]. Novice programmers may feel they understand an algorithm well; however, the algorithm animation may uncover inaccuracies and misperceptions in their understanding (Figure 5). The animation of algorithms provides a forum encouraging diligence and creativity in students in an engaging manner.



Original array

0	35
1	25
2	6
3	67
4	47
5	73
6	62
7	42
8	28

Figure 5: Visualizing Algorithms

Building an algorithm animation requires identifying the fundamental operations of the algorithm to be portrayed visually [16]. Conceptual, fundamental thought and reasoning must be given to the data manipulated by the algorithm, the interactions occurring in the algorithm, and how the important high level design concepts are manifested. These concepts must be made concrete through graphical depiction by constructing the algorithm to animation mapping and determining what is unique to the algorithm to be communicated visually (Figure 6). A useful way to learn a concept is to have students teach it. Algorithm animation can be valuable due to the enthusiasm and interest inspired in students promoting learning and comprehension.

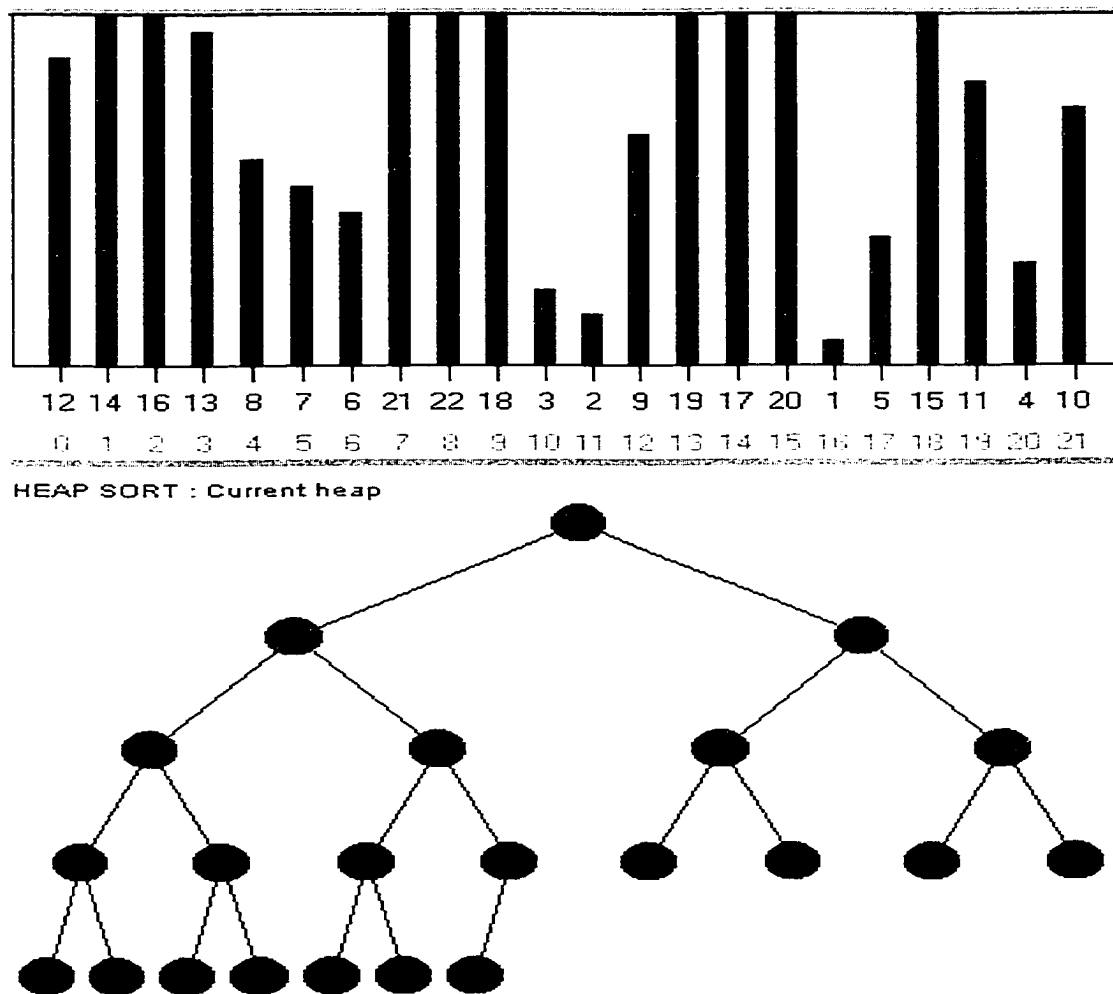


Figure 6: Program Visualization

Interaction and animation engages students in learning (Figure 7). Students are already comfortable with Web browsers. Algorithm animations, program code animations, and animations of other computer science concepts are becoming available on the Web. Animation, sound, and text help students understand the deeper aspects of theory. Java can create a new interactive environment for student program development utilizing the World Wide Web as a substitute for a textbook and for course administration [17]. Java, graphics, and the Web are popular and successful in attracting women to computer science, converting other majors to computer science programs, providing first positive experiences with computing, providing logical progression for students interested in computing and not having any programming experience.

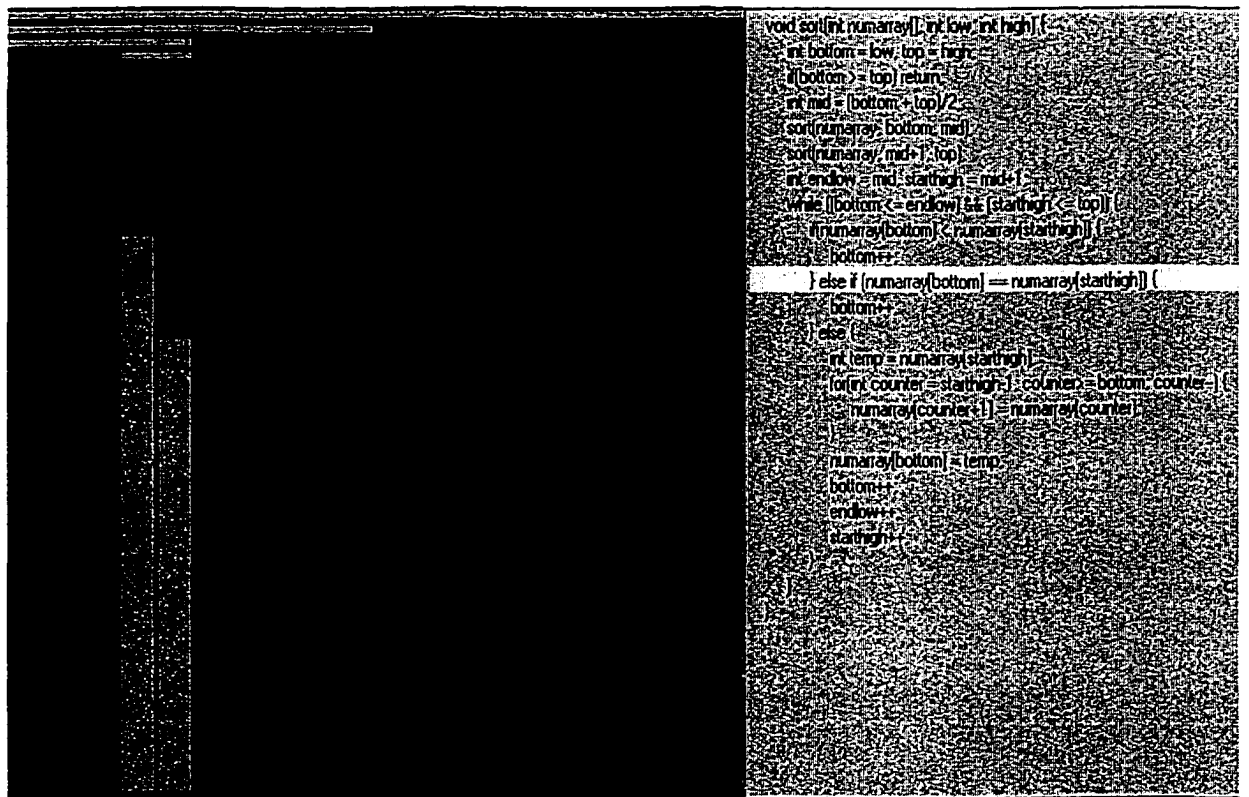


Figure 7: Interactive Animated Program Visualization

Program design is difficult to teach because it is a very abstract topic based on heuristics understood through experience. Video game programming can motivate students. Interactive games serve as software design programming projects solved with abstract data types or object oriented design. Game programming needs to include collections of interacting objects with the right design selected from various ones easing the identification of design abstractions in abstract data types as well as in object oriented

frameworks. One of the most attractive areas of computer science to students is the manipulation of images [47]. Image enhancement applies techniques to emphasize and sharpen image features for display and analysis. Students write generic, reusable code conforming to application programming interfaces to visualize algorithms.

Many educators are promoting the notion of teaching about classes and objects first to help students adopt the object oriented paradigm at an early stage and encourage them to focus on the application structure before beginning coding. Java's clean support for the object oriented paradigm is widely regarded as a suitable choice for introductory teaching [18]. Java is consistently described as an excellent language for teaching the object oriented paradigm. Java provides graphical support for object oriented design, abstracts over files, and abstracts over the operating system. Java provides fully integrated support for designing, editing, compiling, and testing a cycle. Java supports interactive creations of objects and the interactive calling of methods for objects providing support for incremental development which is one of the major advantages of object orientation.

Educators are taking advantage of Java's special features to teach novice programmers. Java syntax is being taught with modern graphical user interface concurrent code with classes [19]. Java allows students to work on interesting problems without having to understand all details of how the code works and receive a practical introduction to the modern programming experience of writing a section of a larger program early emphasizing abstraction. In the transition from procedural to object oriented programming, students learn objects have a state that changes over time as the object is manipulated. In object oriented programming, the user interface code is separated from the domain specific code. Students enjoy object oriented programming more and have more fun filled discussions in class.

Java course work should include interaction and graphical user interfaces, algorithms, object oriented programming, and Java specific issues with themes in creativity and visually interactive methods. Java classes in Java's graphic and interactive capabilities should be suitable for anyone even if they have no prior experience in programming, math, or science. Intuitive interactive graphics are motivating and challenging. Graphical displays allow students to check whether their programs work properly by viewing their program results and determining conformity to their expectations. Java provides exposure to code modifications and experience using the development environment for handling interactions with the user in developing graphical user interfaces. Students can read entire functional blocks of another programmer's code and extend the code to add further functionality. Complicated programs can be studied for completion and future collaborative work.

The focus on creativity results in a learning environment well suited for novice programmers where each student is proud of the individuality of their work motivating students to devote extra time to creative aspects and the exploration for interesting ways to improve their work. Creativity helps personalize the learning experience making it

easier to follow the progress of a particular student. Graphical user interfaces can increase student interest and confidence levels. Java programs can be viewed to see if they respond correctly motivating students to experiment with cause and effect between program statements and applet appearance.

Java's object oriented nature permits students to participate in complex visual programs. Java allows encapsulation, ease of class reuse, and object oriented programming with multiple class declarations and class objects. Emphasis should be placed on abstract comprehension of the entire program not just line by line code comprehension because the way students experience phenomena affects their competence in object oriented programming. Recursion, abstraction, and encapsulation are used to create flexible graphical user interface tools. Object oriented programming is most valuable in large projects making extensive use of adapted and extended libraries.

3. STATEMENT OF THE PROBLEM

Learning how to program a computer is difficult for most people. Computer programming is a cognitively challenging, time consuming, labor intensive, and often frustrating endeavor. Years of formal study and training are required to learn a programming language's world of algorithms and data structures. Instructions are coded in advance before the computer demonstrates the desired behavior [48]. Seeing all the programming steps and instruction code is complicated. There exists a tremendous gap between the representations the human brain uses when thinking about a problem and the representations used in programming a computer.

Using charts and diagrams in association with programming is a manner in which technical details have been presented almost since the start of programming. Flow charts were originally developed as a visual aid for assembly language programmers because assembly languages have no facility to enforce structured programming. Programmers were instructed to draw flow charts to make the program structures clear in their minds to prevent them from being lost and confused with voluminous, low level details.

Often people are much better at dealing with specific, concrete objects than working with abstract ideas. Concrete and specific programming examples and demonstrations can be very useful. When cleverly chosen and properly used, programming examples and demonstrations help people understand the abstract concepts [49]. Programming by example or demonstration attempts to extend these novel ideas to novice programming.

4. OVERVIEW

Computer science students in introductory programming courses are eager to work harder to learn Java and students have a high opinion of Java because they are learning a marketable skill. Students are more motivated to learn a particular programming language if they perceive the programming language as useful for them beyond the immediate learning experience [20]. Java compiles source code for .java files and generates .class files. Java includes the environment, built in types, operators, public static functions, strings, arrays, utilities, toolkits, and applets.

Java simplifies C++ and Ada by removing many features and improving design decisions. Java eliminates C++ features such as templates, operator overloading, pointers, and various subtle C++ constructs. The Java improvement over C++ is the exclusion of the C++ header files and class interface. Also, meaningful documentation page can be extracted from a properly commented Java class through the use of the Javadoc utility.

Java provides a simple mechanism for using graphical user interfaces and drawing graphics more than adequate for use in an early computer science course (Figure 8) [21]. Java defines the Boolean built in type eliminating common ($if(x=y)$) C++ errors. Java has bounds checking for a predefined string object and an array object. Java does not have pointers; however, except for a built in type, an object name is a reference making it a pointer.

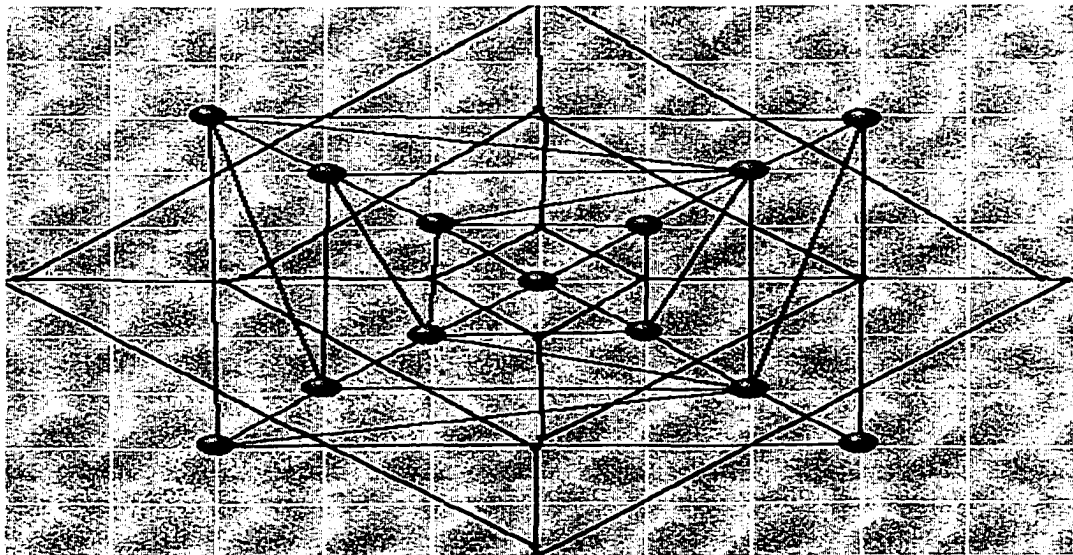


Figure 8: Animation of Algorithms

Java's avoidance of direct pointers will reduce students' programming errors in linked lists and search trees. Java supports garbage collection with no need for delete or delete[] or as in Ada unchecked_deallocation. Java excludes copy constructors, initializing lists, parameter passing mechanisms, return mechanisms, and constant member functions. Java passes and returns all built in types by value and all other object types by reference.

Beginning students are often overwhelmed and confused by having to decide on the correct parameter passing and return type among value, reference, and constant reference when learning to program in C++ [11]. Java's implementation of inheritance with all binding being dynamic by default eliminates difficulties with C++'s virtual functions. The Java interface provides an alternative to multiple inheritance. Java's inheritance handling is among the strongest reasons to choose Java over C++ or Ada.

5. APPROACH

The approach of the thesis in solving the conventional programming problem is through the creation of a visual iconic object oriented programming language. Icons are connected and generate Java code. The program is available through the University of Texas Pan American Computer Science Department Interactive Systems Laboratory at <http://www.cs.panam.edu/TR/cs-tr.html>. The program is written in C++ and generates Java code. Visual programming is encouraged by the fact icons convey meaning more concisely than text, icons are memorable, and icons are understood by people regardless of the language they speak [23]. A visual programming language is created from varied cultural experiences and is designed for different audiences using an interdisciplinary approach drawing on information visualization, graphic design, and cognitive science.

Object oriented programming classes support existing software because the classes are used unchanged and are easily extended for reuse in new applications [25]. Program visualization uses program execution monitors written in the source code. Visual programming increases the understanding of runtime system behavior and provides graphical feedback about program execution allowing people to deal with voluminous data more effectively than with textual techniques.

The goal of visual object oriented programming systems is to combine the advantages of function reusability through classes and their instances, the extensibility of object oriented technology, and the accessibility of visual programming. Object oriented languages support abstract specification in classes and hierarchies of classes. The object oriented approach encourages programming modular architectures promoting reliability and reusability attributes required for large scale programming.

6. IMPLEMENTATION

A typical screen for the visual iconic object oriented programming to advance computer science education and novice programming developed for this thesis is shown below in Figure 9. The icon box allows user to choose a triangle icon to represent an if statement, a rectangle icon to represent a true statement, or a circle icon to represent a false statement. The Java code for the if, true, and false statements is entered in boxes corresponding to the triangle icon for if, the rectangle icon for true, and the circle icon for false. The icon box allows users to choose a triangle icon to represent a while statement, a rectangle icon to represent a condition statement, or a circle icon to represent a block statement. The Java code for the while, condition, and block statements is entered in boxes corresponding to the triangle icon for while, the rectangle icon for condition, and the circle icon for block.

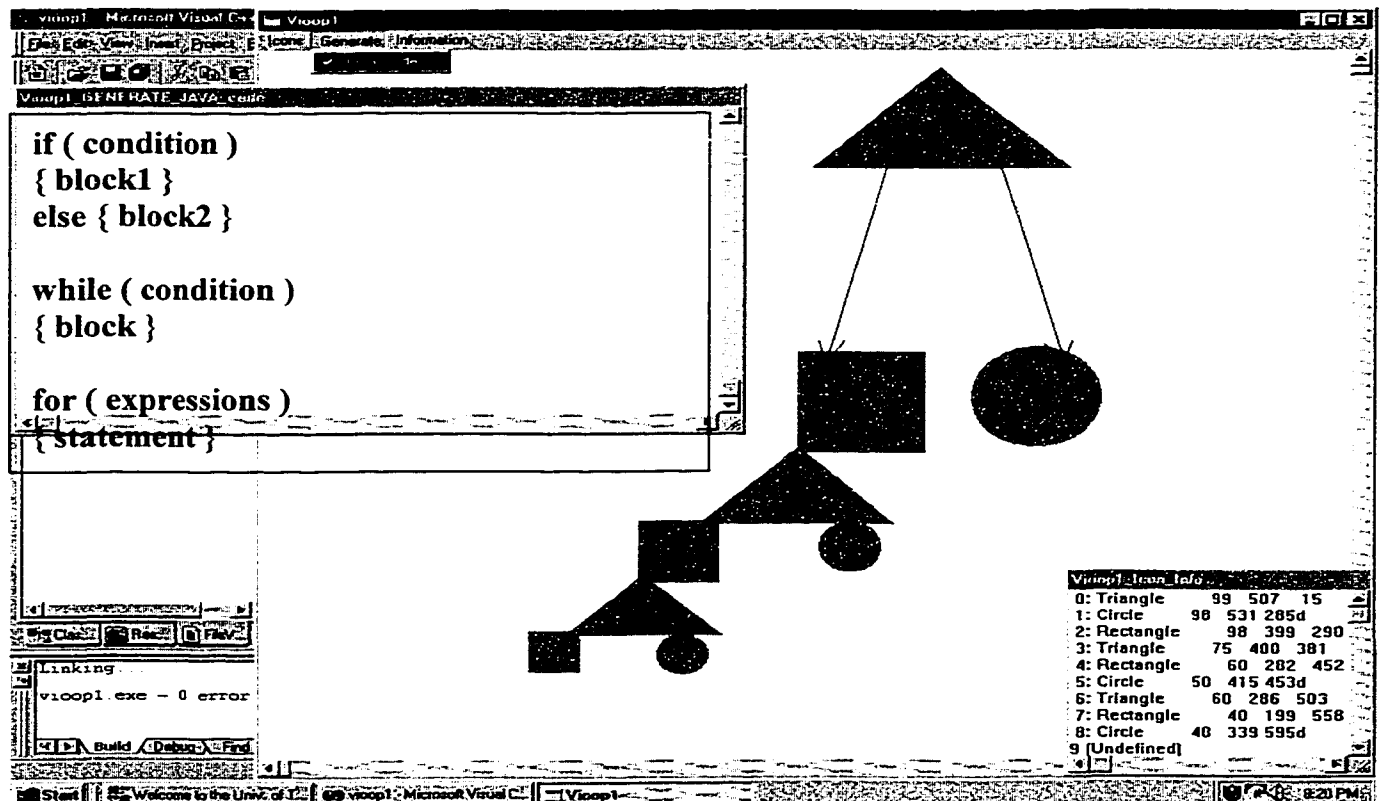


Figure 9: Visual Iconic Object Oriented Programming

Ten icons with the geometrical shapes stated can be selected to generate Java code. Icons are distinguished by color and shape. Icons can be interactively positioned on the main window or deleted. Icons are connected by arrows or by having the icon edges border on each other's shapes. Subwindows display the Java code generated and the icons' information regarding their size, x coordinates, and y coordinates on the main window.

An interactive windows menu uses the mouse to either add, position, or delete a figure by selecting an interactive menu item. While delete is selected, any figure clicked on is deleted. The program checks the menu item which is selected and has the selected item remaining checked until another item is selected. For size specification, an edit field allows the user to enter the size in pixels of the figure.

A window displays the Java code generated with the icons and includes all the statements corresponding to the icons. A second window lists all the figures' shapes, sizes, and positions. A vertical scroll bar is used to view all the information when the window is sized smaller than needed to show all the items at once. The left mouse button is used to both select the figure to be animated and indicate where to move the figure. With a left click in the position mode, positioning an animation will begin with the second left click. The first left click while in position mode, essentially signals that the next left click will be the location to animate.

Computer science students are able to interact with any class visualization through object instantiation and the activation of class methods. The objects are directly manipulated in a class environment. A visual environment enhances student understanding of object oriented concepts [13]. Students visualize the interaction of class components by accepting a class as input and producing a window containing a visualization of the class methods. Objects may be used as arguments for method invocation resulting in the instantiation of further objects. The class being viewed may be changed at any time. The specified class methods will replace the original class methods. Computer science students can easily move among classes. Objects are created in one class and used in methods of another class.

7. CONCLUSION

Novice programmers experience tremendous difficulty when programming. One source of difficulty is a disparity between visual representations used when thinking about a problem and the coded representations of a programming language. Programming is a cognitively challenging task requiring the design of programming

architecture, coding modules, choosing wisely among several data structures and algorithms, constructing the program code, and fixing the inevitable bugs [29]. Visual iconic object oriented programming by demonstration and analogies enables novice programmers to create complex program code [24]. Analogies are powerful cognitive mechanisms people use to construct new knowledge from knowledge they have already acquired and understand. Programming by demonstration enables the construction of arbitrary programs by recording actions on sample data and explicitly removing details from data structures.

Animated programming allows a computer science student to be a character in an animated virtual world in which programming abstractions are replaced by their tangible analogs. The visual properties of the interactive elements are used including their size, shape, color, and appearance to describe the student's intentions [22]. Visual iconic object oriented programming by demonstration enables a system to record actions performed by computer science students in the interface to produce a generalized program code reusable later in analogous examples.

8. LIMITATIONS

Comprehension of visual representations is dependent upon the cultural and prior experiences of students. Program reuse is a software design constraint. As in conventional programming approaches, the effective reuse of new behavior is limited by the underlying representations [30]. A scaling problem poses difficulty in using the same visual language programming development techniques from small programs to larger complex coding applications in visual and multimedia computing.

Personal user interactive interfaces combining the understanding of communicative, cognitive, and perceptual skills with computer input and output devices require the integration at multiple levels of technologies such as speech and sound recognition and generation, computer vision, and graphical animated visualization. Objects need to gain sensors, computational powers, and actuators turning them from static inanimate objects into adaptive, reactive systems making them more friendly, useful, and efficient [28].

Parallel or duplicate functionality is required to accomplish natural goals. Advancing multimodal systems will require multidisciplinary expertise in a variety of areas beyond computer science including speech and hearing, perception and vision, linguistics, psychology, signal processing, pattern recognition, and statistics. Machine vision and image processing at high resolution need to be explored for improved human computer interaction.

9. FUTURE WORK

Visual languages will extend beyond programming into visual communication during human computer interaction. Improvements will be in dialogue analysis, user models, interactive computational models, and visual reasoning. Multimodal interfaces will facilitate visual querying, visual interaction, interactive computing, and multimedia communication [27]. Varying paradigms are required for the visual and multimedia computing life cycle including the integration and synchronization of different media.

The scaling challenge needs to be overcome to apply program development techniques to larger applications for distributed visual and multimedia computing on the Web. Visual programming will improve with the enhancement of visual and spatial reasoning; as well as, with the development of multidimensional, dynamic, multimodal, and graphic languages.

Interactive interfaces will provide users with greater control, communication, learning capabilities, direct manipulation, navigation, immediate feedback, and recovery mechanisms [26]. Users will have full involvement in interface participatory design. Early development will require frequent testing, validation, and assessment of project features for corrections, improvements, and modifications, especially, in the construction of human and computer dialogue.

Future visual languages will be inherently dynamic and interactive. Cognitive models of how humans understand and reason with visualizations and communicative diagrams will be developed. Semantics for visual languages will exist without domain specific knowledge. Multimedia human computer interaction will be enhanced through interdisciplinary diagrammatic reasoning, data visualization, graphic design, and cognitive science [7].

Icons, spatial relationships, and time dimension to demonstrate semantic relationships will be multidimensional. Visual iconic object oriented programming will be via multiple dimensions. Individual audiences will have accessibility to programming, will correctly program, and their programming speed will be improved. Visual programming languages in multiple dimensions will offer particular audiences the opportunity to program promptly and efficiently.

Visual iconic programming languages will emphasize concreteness as opposed to abstractness, will be direct in achieving the goals of the user's actions, will emphasize explicit semantics, and will automatically display edited semantic effects to improve people's ability to program. Natural representations for concepts will innovate communication in visual programming environments. Scalability devices, hybrid visual languages, the integration of environments, and metalanguages that create modeling languages will advance computer science [50]. Visual iconic object oriented communicative computing will efficiently convey information to students in computer science education.

REFERENCES

- [1] Nan C. Shu, IBM Los Angeles Scientific Center, *Visual Programming*, Van Nostrand Reinhold Company, Inc., 1988.
- [2] Margaret M. Burnett, Adele Goldberg, and Ted G. Lewis, *Visual Object Oriented Programming: Concepts and Environments*, Manning Publications Company, 1995.
- [3] Tadao Ichikawa, Erland Jungert, and Robert R. Korfhage, *Visual Languages and Applications*, Plenum Publishing Corporation, 1990.
- [4] Shi Kuo Chang, University of Pittsburgh and Knowledge Systems Institute, *Principles of Visual Programming Systems*, Prentice Hall, Inc., 1990.
- [5] Clinton L. Jeffery, Department of Computer Science, University of Nevada, Las Vegas, *Program Monitoring and Visualization: An Exploratory Approach*, Springer Verlag New York, Inc., 1999.
- [6] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization, Using Vision to Think*, Morgan Kaufmann Publishers, 1999.
- [7] S. K. Chang, M. M. Barnett, S. Levialdi, K. Marriott, J. J. Pfeiffer, and S. L. Tanimoto, The Future of Visual Languages, in *IEEE Proceedings of Visual Languages, IEEE Symposium on Visual Languages, Pages: 58-61, 1999*.
- [8] R. J. Wolfe, *3D Graphics: A Visual Approach*, Oxford University Press, 2000.
- [9] Nan C. Shu, A Visual Programming Language Designed for Automatic Programming, in *System Sciences, Volume 2, Software Track, Proceedings of the Twenty-Fifth Annual Hawaii International Conference on Visual Programming Languages, Pages: 662-671, 1988*.
- [10] Barry Burd, Wayne Spies, Lee Wittenberg, and Robert Workman, Visual Programming Tools in the Computer Science Curriculum, in *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education, Volume 29, Issue 1, Pages: 388-389, February 27 - March 1, 1997*.
- [11] Frank Wester, Marleen Sint, and Peter Kluit, Visual Programming with JAVA: An Alternative Approach to Introductory Programming, in *Integrating Technology into Computer Science Education, Volume 29, Issue 3, Pages: 57-58, June 2-4, 1997*.
- [12] Thomas L. Naps, Algorithm Visualization on the World Wide Web: The Difference Java Makes, in *Integrating Technology into Computer Science Education, Volume 29, Issue 3, Pages: 59-61, June 2-4, 1997*.
- [13] Herbert L. Dershem and James Vanderhyde, Java Class Visualization for Teaching Object Oriented Concepts, in *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education, Pages: 53-57, February 25 - March 1, 1998*.

- [14] Christopher M. Boroni, Frances W. Goosey, Michael T. Grinder, and Rockford J. Ross, A Paradigm Shift: The Internet, the Web, Browsers, Java, and the Future of Computer Science Education, in *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education*, Pages: 145-152, February 25 – March 1, 1998.
- [15] Christopher M. Boroni, Frances W. Goosey, Michael T. Grinder, Jessica L. Lambert, and Rockford J. Ross, Tying It All Together Creating Self Contained, Animated, Interactive, Web Based Resources for Computer Science Education, in *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, Pages: 7-11, March 24 to March 28, 1999.
- [16] Arturo I. Concepcion, Lawrence E. Cummins, Ernest J. Moran, and Man M. Do, Algorithm 98: An Algorithm Animation Project, in *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, Pages: 301-305, March 24 to March 28, 1999.
- [17] Linda Stern, Herald Sondergaard, and Lee Naish, A Strategy for Managing Content Complexity in Algorithm Animation, in *Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, Pages: 127-130, June 27 – July 1, 1999.
- [18] Steve Cunningham, Powers of 10: The Case for Changing the First Course in Computer Graphics, in *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education*, Pages: 46-49, March 8 to March 12, 2000.
- [19] Thomas L. Naps, James R. Eagan, and Laura L. Norton, Have an Environment to Actively Engage Students in Web Based Algorithm Visualizations, in *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education*, Pages: 109-113, March 8 to March 12, 2000.
- [20] Guido Robling and Bernd Freisleben, Experience in Using Animations in Introductory Computer Science Lectures, in *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education*, Pages: 134-138, March 8 to March 12, 2000.
- [21] Judith Bishop and Nigel Bishop, Object Orientation in Java for Scientific Programmers, in *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education*, Pages: 357-361, March 8 to March 12, 2000.
- [22] Matthew Turk and George Robertson, Perceptual User Interfaces, in *Communications of the ACM*, Volume 43, Issue 3, Pages: 33-34, March 2000.
- [23] Alex Pentland, Perceptual Intelligence, in *Communications of the ACM*, Volume 43, Issue 3, Pages: 35-44, March 2000.
- [24] Henry Lieberman, Programming by Example, in *Communications of the ACM*, Volume 43, Issue 3, Pages: 73-74, March 2000.

- [25] David Canfield Smith, Allen Cypher, and Larry Tesler, Novice Programming Comes of Age, in *Communications of the ACM, Volume 43, Issue 3, Pages: 75-81, March 2000.*
- [26] Brad A. Myers, Richard McDaniel, and David Wolber, Intelligence in Demonstrational Interfaces, in *Communications of the ACM, Volume 43, Issue 3, Pages: 82-89, March 2000.*
- [27] Alexander Repenning and Corrina Perrone, Programming by Analogous Examples, in *Communications of the ACM, Volume 43, Issue 3, Pages: 90-97, March 2000.*
- [28] Mathias Bauer, Dietmar Dengler, Gabriele Paul, and Markus Meyer, Programming by Demonstration for Information Agents, in *Communications of the ACM, Volume 43, Issue 3, Pages: 98-103, March 2000.*
- [29] Ken Kahn, Generalizing by Removing Detail, in *Communications of the ACM, Volume 43, Issue 3, Pages: 104-106, March 2000.*
- [30] Robert St. Amant, Henry Lieberman, Richard Potter, and Luke Zettlemoyer, Visual Generalization in Programming by Example, in *Communications of the ACM, Volume 43, Issue 3, Pages: 107-114, March 2000.*
- [31] Alan Watt, *3D Computer Graphics*, Addison-Wesley, London, England Pearson Education, 2000.
- [32] Edward Angel, *Interactive Computer Graphics: A Top-Down Approach with OpenGL*, Addison-Wesley Longman, Inc., 2000.
- [33] Mirko Raner, Teaching Object Orientation with the Object Visualization and Annotation Language (OVAL), in *SIGCSE Bulletin Inroads: Providing the Way Towards Excellence in Computing Education, Volume 32, Issue 2, Pages: 45-48, June 2000.*
- [34] Elizabeth Odekirk, Dominic Jones, and Peter Jensen, Three Semesters of CS0 Using Java: Assignments and Experiences, in *SIGCSE Bulletin Inroads: Providing the Way Towards Excellence in Computing Education, Volume 32, Issue 2, Pages: 144-147, June 2000.*
- [35] Ann E. Fleury, Encapsulation and Reuse as Viewed by Java Students, in *SIGCSE Bulletin Inroads: Providing the Way Towards Excellence in Computing Education, Volume 32, Issue 4, Pages: 189-193, December 2000.*
- [36] Richard Rasala, Jeff Raab, and Viera K. Proulx, Java Power Tools: Model Software for Teaching Object Oriented Design, in *SIGCSE Bulletin Inroads: Providing the Way Towards Excellence in Computing Education, Volume 32, Issue 4, Pages: 297-301, December 2000.*
- [37] Ralph Morelli, Ralph Walde, Gregg Marcuccio, A Java API for Historical Ciphers: An Object Oriented Design Project, in *SIGCSE Bulletin Inroads: Providing the Way Towards Excellence in Computing Education, Volume 32, Issue 4, Pages: 307-311, December 2000.*

- [38] Richard Pattis, Teaching OOP in C++ Using an Artificial Life Framework, in *Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education, Volume 29, Issue 1, Pages: 39-43, February 27–March 1, 1997.*
- [39] Richard Rasala, Automatic Array Algorithm Animation in C++, in *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education, Pages: 257-260, March 24 to March 28, 1999.*

URL REFERENCES

- [40] Programming Visualization: <http://gais.cmsc.lawrence.edu/cmsc34/AVClient.html>
- [41] Programming Visualization: <http://www.cs.brockport.edu/cs/java/apps/sorters/heapsortaniminp.html>
- [42] Java in Computer Science Education: <http://www.cs.rit.edu/~ncs/tinkerToys/tinkerToys.html>
- [43] Algorithm Visualization: <http://www.research.compaq.com/SRC/zeus/gallery.html>
- [44] Algorithm Visualization: <http://swarm.cs.wustl.edu/cgi-bin/gal>
- [45] Algorithm Animation: <http://www.cc.gatech.edu/gvu/softviz/parviz/polkaanim.html>
- [46] Geometric Animation: <http://loki.cs.brown.edu:8080/pages/Delaunay.html>
- [47] Programming Visualization: http://www.cs.princeton.edu/~ah/alg_anim/version0/Graham.html
- [48] Programming Visualization: <http://www.cs.hope.edu/~algaanim/animator/Animator.html>
- [49] Geometric Animation: <http://www.latech.edu/~watson/graphics.html>
- [50] Programming Visualization: http://www.cs.princeton.edu/~ah/alg_anim/version1/Animator.html