LogicCity: On-Line Digital Logic Simulator

Jihad Z. Almahayni
University of Rhode Island

LogicCity:

ON-LINE DIGITAL LOGIC SIMULATOR

BY

JIHAD Z. ALMAHAYNI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

1998

MASTER OF SCIENCE THESIS

OF

JIHAD Z. ALMAHAYNI

APPROVED:

Thesis Committee

Major Professor [illegible]

[illegible]

[illegible]

Thomas J. Rockett

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

1998

Abstract

A primary challenge for science educators is to develop ways to help students understand new ideas that cannot be directly observed. The application of computer simulations is one method instructors may use to address the problem of teaching for conceptual understanding. The use of computer simulations for science teaching is increasing steadily because simulations provide students with the opportunity to witness or "perform" experiments that might otherwise be too expensive, too time consuming, or too dangerous for them to do in the lab. Simulation, in general, is a very effective way for both a student and a professional to visualize concepts and ideas and to check the accuracy of results generated theoretically. Typically, computer simulations represent real world events on the computer and allow students to observe new phenomena by performing various manipulations that affect the on-screen events.

Here, at the University of Rhode Island, introductory computer science courses include computer organization and architecture with concepts that can often be difficult to grasp. This dilemma lays the foundation for the need for educational software "simulation" which better enables students to understand the concepts and ideas presented in class. In this project, we set to design and develop LogicCity, a prototype model of an on-line interactive digital logic design simulator that will be used as a teaching aid by complementing class discussions. LogicCity can be used as a stand-alone application or it can be incorporated into the Web as an applet. Students may use this simulator as a tool to build combinational digital circuits and generate accurate results. It is also designed to work in conjunction with the coaching material of an introductory computer organization course.

Acknowledgments

I would like to thank my advisor, Dr. Gerard Baudet, for suggesting this topic and helping me to see it through. Dr. Baudet has been very patient and showed me a desire to make me understand the things it took me a while to grasp. Thanks to Dr. Joan Peckham for her advice, encouragement and help. Also thanks to Dr. Betty Liu for serving on my committee and to Dr. Norman Finizio for agreeing to chair my defense.

I want to thank my mother and family for their support, and especially my late father, Zuhair Almahayni, who has been my main motivation all along. Thanks dad, you are always in my thoughts. And most of all, I want to thank my wife Dania for proofreading this thesis, for her faith, inspiration, and patience. With love and hope for the future.

Preface

This thesis has been prepared in the **standard format** for theses as given in the *Statement on Thesis and Dissertation Preparation* of the Graduate School of the University of Rhode Island.

This thesis is divided into six chapters. Chapter One is an introduction to the use of software in education. Chapter Two presents background and related research, and a discussion of the general uses of simulation in education. In Chapter Three, logic simulation, algorithms, and available products and their limitations are all introduced. A brief discussion of the Java programming language and the reasons behind our choice of Java for the implementation language are given in Chapter Four. An introduction to my prototype simulator model, LogicCity, and a detailed presentation of its features are also included in Chapter Four. Chapter Five discusses the model's data structures, the various algorithms used, and the design challenges we faced and then overcame. Chapter Six is a conclusion to my thesis and discusses the limitations and potential future enhancements of LogicCity.

Table of Contents

Abstract. ii

Acknowledgments. iii

Table of Contents. iv

List of Tables. vi

List of Figures. vii

Chapter 1 Introduction. 1

Chapter 2 Background & Related Research. 7

2.1 Educational Environments. 8

2.2 Simulation and Education. 10

2.3 Advantages and Disadvantages of Simulation. 12

2.4 Instructional Design for Computer Simulation. 16

Chapter 3 Digital Logic Simulation. 21

3.1 Why Computer Simulation. 21

3.2 How Simulation Work. 23

3.3 Commercial Algorithms. 24

3.4 Limitations of Available Models. 26

Chapter 4 System Specification and Features. 31

4.1 The Java Programming Language. 31

4.2 The System's Web Site. 34

4.3 LogicCity. 36

4.4 Software Features. 38

Chapter 5 Data Structures. 46

5.1 System Design. 46

5.2 System Implementation 49

5.3 System Testing. 61

5.4 Design Challenges. 63

Chapter 6 Conclusion. 65

6.1 Future Work. 68

6.2 System Evaluation. 68

References. 70

Appendix A: Program Listing. 73

Appendix B: System Manual. 142

B.1 How to Build Circuits. 142

B.2 Entering Input Signals / Trapping Output Signals. 143

B.3 How to Edit a Circuit. 144

B.4 How to Save and Open a Circuit. 145

B.5 How to Simulate a Circuit. 146

B.6 How to Change the Background Color. 146

Bibliography. 148

List of Tables

Table 1: Comparison of Applications vs. Applets. 33

List of Figures

Figure 1.1: Place of Computer Simulation as Educational Software. 3

Figure 1.2: Presentation Techniques. 5

Figure 1.3: NEWTON Program. 6

Figure 2.1: Learning Environment. 9

Figure 2.2: An Example Simulation Model (Hysteresis Applet) 11

Figure 2.3: An Example Simulation Model (Harmonic Oscillator). 14

Figure 2.4: An Example Simulation Model (Virtual Memory). 17

Figure 2.5: Complete Simulation Environment. 19

Figure 3.1: Dizzy Program. 27

Figure 3.2: Logg-O Program. 29

Figure 4.1: Main Web Page. 34

Figure 4.2: Simulator Web Page. 35

Figure 4.3: LogicCity's Window. 36

Figure 4.4: Basic Gates. 37

Figure 4.5: Digital Logic Circuit. 40

Figure 4.6: Deletion Confirmation Dialog Box. 41

Figure 4.7: Save File Dialog Box. 42

Figure 4.8: Save Confirmation Dialog Box. 43

Figure 4.9: Simulated Logic Circuit. 44

Figure 4.10: Example Simulated Circuit. 45

Figure 5.1: System Architecture 51

Figure 5.2: Main Attributes of Logic Gate Object 59

Chapter 1

Introduction

Developments in basic electronics and in hardware components of computer and communication systems by major corporations are proceeding very rapidly, independent of any educational considerations. Vast changes in the capability and economics of computing and communication systems imply a number of changes for education. In addition, changes in the efficiency of hardware has increasing impact on all aspects of business, government, industry, media and communication - changes to which education must respond [13].

Educational technologies are not single technologies but complex combinations of hardware and software [21]. Using the term "computer" means more than merely the basic components -- the monitor, keyboard, mouse, etc. It includes multimedia capability that computers can support, and peripheral devices connected with computers like modems. From the standpoint of education, importance is stressed on the material delivered, rather than the hardware delivering it. Currently, computerized education can be divided into four basic groups [13]:

1. Typically a tutorial, presented in a workbook format, controls the material presented to the student.

2. Exploratory education, such as an encyclopedia CD, permits the student to browse and view the information displayed. This kind of application presents complex tasks and engages students in active problem solving.
3. Applications, such as word processors and spreadsheets, provide students with the tools to facilitate writing tasks or analysis of data. These applications also promote higher-order thinking by engaging the students within a collaborative learning context [21].
4. Communication education, allows students to exchange information among themselves through networks such as the Internet. This cooperative network allows long distance learning and gives students access to a broader range of resources.

Computers have taken a major position in the world because they can execute many tasks more effectively and efficiently than humans. Computers play an important role in helping students to learn by communicating information more efficiently as well as by captivating the mind of the student. For computers to reach a higher level in educating, one new element is essential: they must be allowed to teach students without a human intermediary between the student and the computer. The idea is that students learn better when they are forced to take an active constructive role. As tutors, computers can revolutionize education as thoroughly as they have transformed almost every commercial activity in the twentieth century [3]. The technological progress of modern society creates an ever-increasing need for improved training materials. However, producing high-quality material for computer-based education is, and continues to be, very expensive. Widespread use and reuse of high-quality basic learning materials is the most practical way to share the costs for the development of these educational systems. Interactive computer graphics and other multimedia components play a central role in the introduction of tools for higher quality learning material (See Figure 1.1).

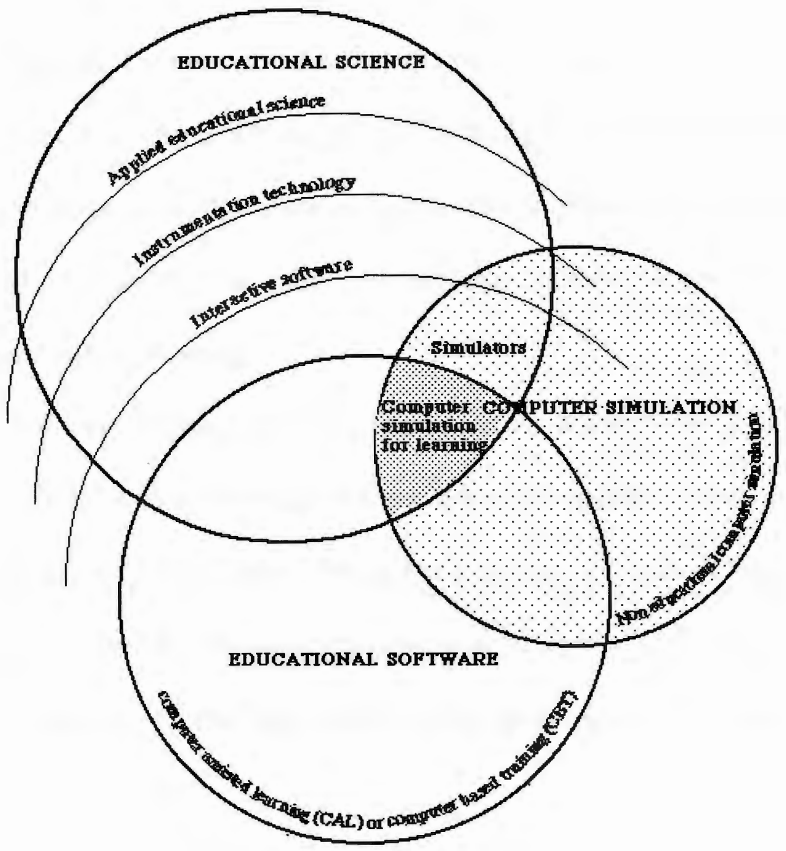

EDUCATIONAL SCIENCE
Applied educational science
Instrumentation technology
Interactive software
Simulators
Computer simulation for learning
COMPUTER SIMULATION
Non-educational computer simulation
EDUCATIONAL SOFTWARE
computer assisted learning (CAL) or computer based training (CBT)

Figure 1.1 - Place of computer simulation as educational software within instrumentation technology and educational science. The area of interest (shaded) is computer simulation. Within this area are to be distinguished: simulation CAL, simulators for training and non-educational computer simulations.

One of the biggest advantages that a computer offers students of all ages is independence. This independence boosts a student's self-esteem and generates a drive that is imperative to learning. After initially grasping a basic understanding as to how a computer operates, students are free to explore a world of new and exciting things. Besides increasing a student's learning opportunities, motivation, and achievement, computerized education helps them acquire skills that are rapidly becoming essential in the work place. In addition, it gives students more control and responsibility of their own learning. In our opinion, it is of the utmost importance to prepare students to be technically literate by the time they are ready to go out to the working world. Technological literacy is defined as the computer skills and the ability to use computers and other technology to improve learning, productivity and performance [19]. Students are entering a world in which 60% of the jobs will require technological competency, and students must continue to update their occupational and technological skills in order to be successful [20]. Technology can enhance traditional classroom presentations and engage students in more active learning.

Since students respond well to visual stimuli, working on a computer is much more fun and effective than having a teacher lecture. Computers allow a student to work and progress at his or her own pace – a teacher's dream. If a teacher feels the majority of the class grasps a concept, s/he must move on and then offer extra help to those who are struggling. A computer on the other hand lets students learn at the pace that is right for them.

The key to educational technology is how teachers and students use the software available to them. Easy to use multimedia-authoring applications, digital media collections, the Internet, and new educationally valid curriculem based software are all making the learning-centered classroom a reality [3]. Presentability is the software's overall appeal to the user (See Figure 1.2). Black and white, two-dimensional flip screen workbooks cannot possibly compete with the exciting, full-color media that students and teachers are exposed to daily. Rich, three-dimensional graphics, enhanced stereo sound, captivating animation and video, and interactive devices that students can control and manipulate are all necessary elements of successful educational software.

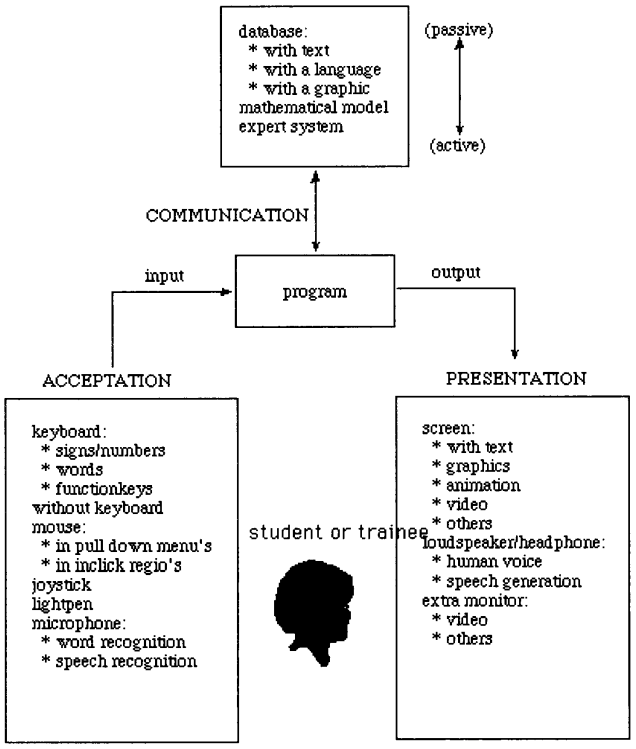


Figure 1.2 - Presentation techniques, acceptation techniques and communication techniques (special media as well as special software procedures) as they occur in educational software.

A good example of educational software with an interactive format that keeps the students attentive and stimulated is NEWTON (See Figure 1.3). This software has been designed and developed at the Computer Science Department at the University of Rhode Island. We have used it in graduate courses as well as undergraduate courses. NEWTON is a package that allows the students to solve mathematical equations. It has a very nice and effective interface that clearly displays the equation being manipulated along with all the intermediate steps taken. Graphs are also an essential part of this system and are drawn in a sophisticated fashion.

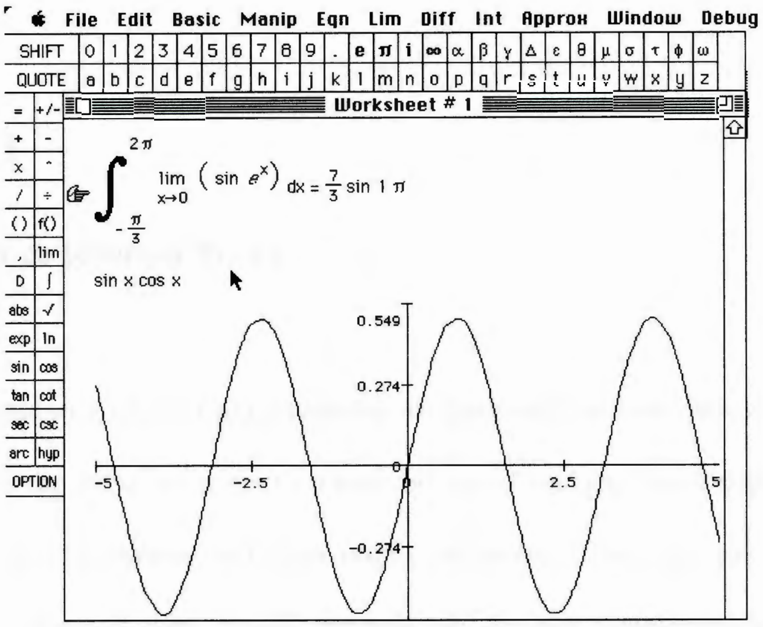

File Edit Basic Manip Eqn Lim Diff Int Approx Window Debug
SHIFT
QUOTE
Worksheet # 1
$\int_{-\frac{\pi}{3}}^{2\pi} \lim_{x\to 0} (\sin e^x)\, dx = \frac{7}{3}\sin 1\,\pi$
sin x cos x
0.549
0.274
-0.274
-5
-2.5
0
2.5
5
OPTION

Figure 1.3 - NEWTON Program

The main contribution of this thesis work is to introduce a new learning environment and a new tool to aid the beginning students in understanding the materials presented in class. "LogicCity" is designed and developed using the Java programming language, thereby it is accessible though the Internet. LogicCity enables students to build on-screen digital logic circuits and to simulate their behavior. This software has an interactive graphical interface format that help stimulate the students in achieving a higher level of conceptual cognition. In chapters 4 and 5, LogicCity is introduced in greater detail and a full discussion of its features and data structures are presented.

Chapter 2

Background & Related Work

The popularity of the Internet among students and teachers has stemmed from its ability to fulfill the needs of users to travel the world seeking knowledge. Through the advances of communications and fiber optics technology, anybody can surf the World Wide Web at a minimal cost. Email, for example, is used extensively between teachers and students. Class notes are no longer limited to paper productions, as the Web has become home to all sorts of publications -- among which are educational material. Learning is no longer restricted to the classroom. An increasing numbers of teachers resort to the Internet for publishing their course materials. Time, energy and resources are greatly reduced with the use of electronic communication. Learning in the digital age is about computer supported education and computer-based simulations in the form of interactive, graphical, and dynamic content.

The Internet is the beginning of a new age in which electronic learning environments will have to try and obtain a firm footing at schools and on the home market. The Internet, therefore gives the electronic learning environment industry and Computer Based Learning a new opportunity. The interactive multimedia has now entered its second stage due to the incorporation of audio and video content. Computer-based simulations have many applications in higher education. There are many different computer-based simulations: patient-simulations, model-driven simulations, animations and simulations in which modeling comes first. Simulation, in general, can be used as a learning tool to gain insight into some phenomena that is hard or may be impossible to demonstrate in a classroom.

2.1 Educational Environments

Learning environments, and simulations in particular, can only be justified as a learning environment when provided with proper instructions adapted to a lesson and with relevant assignments (See Figure 2.1). Assignments and instructional materials should be up-to-date and easy to modify for an individual teacher or school. An important aspect of learning environments is that teachers like to be able to add useful material or they may want to adapt something to the materials in question. It should be possible to make suitable assignments or easily accommodate existing instructional materials. All materials should be embedded into this electronic learning environment. For learning environments with simulations, the interactive simulator should be designed in such a way that modifications are simple and effortless. The instructional materials, and assignments, on the other hand, should be supplied with more flexibility, in order for the teacher to be able to distinguish him/herself, and gear his/her teaching materials and the accompanying materials to the needs of the target group.

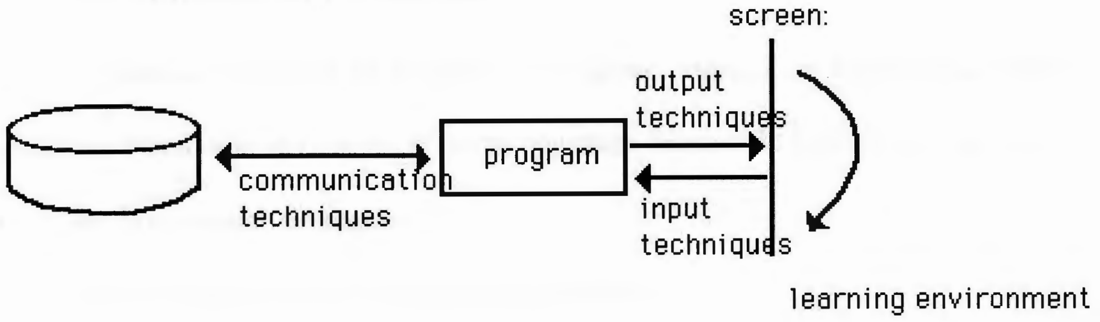


Figure 2.1 - A program is educational software only when much attention has been paid to "presentation" (output), "acceptation" (input) and "communication" techniques.

By using Java as the programming language, software developers can build computer-based simulations as components for teachers. No matter which platform is used to develop the software, applications and applets are platform-independent. These building modules can be embedded into any Web page. Applets are found on virtually any Web page and they are usually interactive, visual and dynamic; making them powerful despite the fact that they have to be interpreted by a browser. This particular form of education provides the most possibilities because an increasing number of applications in applet format are appearing on the Internet. SUN, Microsoft and Apple are working to upgrade their standard programs to Java in order not to lose the battle for the Web. The performance of a simulation program as a built in applet is generally sufficient enough to bring a concept across. A publisher or a Web designer can make an applet and a basic instruction with examples whereas a teacher can decide to do something completely different. However, the Internet does have limitations on the ability of application use. Even though security is not compromised, one of the Internet's greatest disadvantages is that the Web has a very rigid interface concept that results in certain limitations with respect to stand-alone software (applications).

2.2 Simulation and Education

Attention will now be focused on computer simulation with special attention to computer simulation in education. Some questions that are addressed here include:

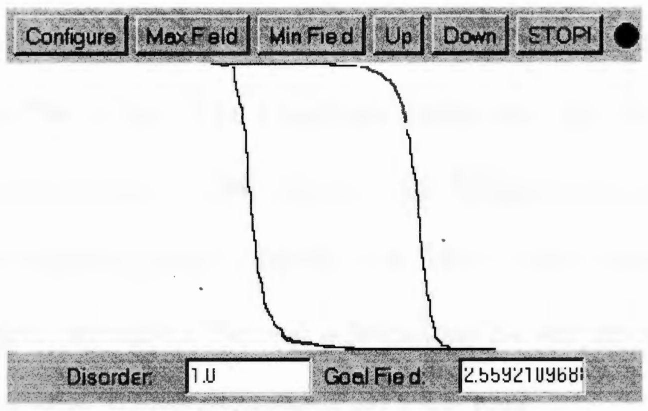
- What is computer simulation?
- How is computer simulation used in education?
- What are the characteristics of a computer simulation?
- What are the advantages of computer simulation?
- What is meant by fidelity in relationship to computer simulation design?

The word simulate means to imitate something. Generally, simulation involves some kind of model or representation. The model imitates important concepts of what is being simulated. A simulation model can be a mathematical model, a computer model, a physical model or even a combination of all [1]. Applications (models) range from biological systems to business and industrial systems. But what exactly is a model?

A model is a carefully structured description of the simulated system. A model describes the state of the system and the possible transitions of the state of the system in the form of rules or equations. There are two types of models in general [14]: (1) qualitative models based on logical and/or conditional relations between variables and, (2) quantitative models based on mathematical equations of the relations between variables. Most computer simulations are based on quantitative models, which may contain parameters with fixed values that represent the properties of the system.

The main purpose of an educational computer simulation program is to provide students with a representation of reality. Students are able to manipulate this representation by changing either the properties or the conditions under which the representation operates. The behavior of this representation as a result of the modifications is similar to the represented part [18]. Computer simulation programs are discovery environments in the sense of experimenting with a model of a system in order to retrieve information about the model and the system. In addition, as mentioned earlier in the introduction, computer simulations are classified as exploratory environments that stimulate learning process (See Figure 2.2). The idea is that the meaningful incorporation of information into the student's cognitive structure becomes easier because the student is forced to take an active constructive attitude [5].

Hysteresis Applet



A 64x64 2-dimensional simulation.

Figure 2.2 - This applet simulates a simple model, which describes the sort of hysteresis seen in a magnetic tape. When you apply an external magnetic field to an initially unmagnetized tape, it becomes magnetized. However, when you remove the external magnetic field, the tape remains partially magnetized. (If not, there would be nothing recorded on our tapes!)

2.3 Advantages and Disadvantages of Simulation

Although the advantages of computer simulations are numerous, they all boil down to one main concept. Computer simulations bring the real world into the classroom and provide a new and modern angle to our methods of education. According to most of the literature, the concept of computer simulation in education has many different names. Examples include educational computer simulation [14], computer-based simulation [17], and instructional simulation [10]. However, they all share the same characteristics, the most common of which are:

1. A model that represents a part of reality,
2. A model that is implemented in a computer program,
3. The user can manipulate or experiment with this computer-implemented model
4. The user explores and discovers the characteristics of the model.

A computer simulation is mainly seen as an exercise or an experiment, which actively involves the student (See Figure 2.3). Computer simulations are claimed to have many advantages. These advantages, which assume that bringing the real world into the classroom promotes learning, are probably the reason why many researchers are investigating computer simulation. Several publications list the advantages of computer simulation. The most often quoted advantages are listed here.

1. Costs: using computer simulation is often cheaper that doing real experiments. Costs of education, including time and resources, can be much higher when doing real lab experiments [14], [18].
2. Scale: some systems are too large or too small to study in reality, but for the purpose of computer simulation they can be scaled [18].

3. Safety: the real system to be studied can be too dangerous [14], [18].

4. Speed: in reality, a system can react too fast or too slow. Computer simulation can slow down or speed up the process by changing the time scale [4].

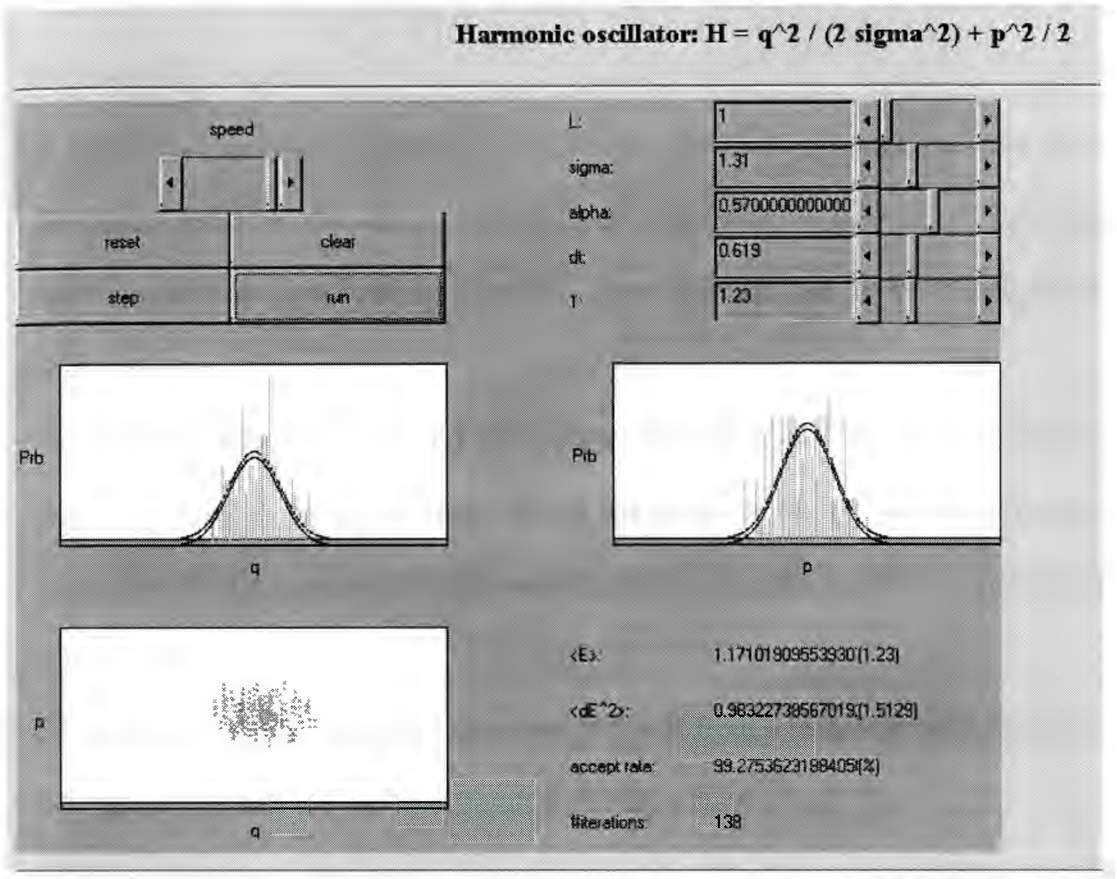
5. Visualization: aspects of the real world can be brought into the classroom in a meaningful way. Abstract concepts can be visualized, which makes it easier for the student to construct a mental model of the system under study [18].

6. Ethics: experiments that are not allowed for ethical reasons can be simulated [18].

7. Didactics: computer simulation is learner-centered. The learner is much more involved in the learning task because s/he can experiment as much as he wants and feedback is instantaneous [5], [14], and [18].

8. Simplification: a computer simulation can be a simplified version of reality since the student is directed to the most important aspects of the system [4].

9. Reality: in reality actions have certain consequences. This can be made clear with computer simulation [14].



Harmonic oscillator: H = q^2 / (2 sigma^2) + p^2 / 2

L: 1
sigma: 1.31
alpha: 0.5700000000000
dt: 0.619
T: 1.23

<E>: 1.17101909553930(1.23)
<dE^2>: 0.98322738567019(1.5129)
accept rate: 99.2753623188405(%)
#iterations: 138

An example simulation model
Figure 2.3

There are not only advantages connected with the use of computer simulation programs in education and training. Limitations are in some cases the result of the wrong or inappropriate use of such programs. Possible limitations of a general and educational kind are [14]:

1. Simulation concerns the manipulation of a number of variables of a model representing a real system. However, manipulation of a single variable often means that the reality of the system as a whole can be lost. Certain systems or components of a realistic situation are not transparent. Some factors have a lot of influence on the

whole, but they have indistinct relations in the whole and can therefore not be represented in a model.

2. A computer simulation program cannot develop the students' emotional and intuitive awareness whereas the use of simulations is specifically directed at establishing relations between variables in a model. This intuition has to be developed in a different way.
3. Computer simulation cannot react to unexpected "sub-goals" which the student may develop during a learning-process. These sub-goals would be brought up during a teacher-student interaction but they remain unsaid during the individual student use of a simulation.
4. Computer simulation programs may function well from a technical point of view, but they are sometimes viewed as difficult to fit into a curriculum.
5. Often a computer simulation program cannot be adapted to take different student levels into account within a group or class. A computer simulation program can certainly be made to adapt to different circumstances if it is designed that way; however, for many computer simulation programs this has not happened.
6. During the experience of interaction with a computer simulation program, the student is frequently asked to solve problems in which creativity is often the decisive factor to success. The fact that this creativity is more present in some pupils than in others is not taken into account by the simulation. Mutual collaboration and discussion among students while using the software could be a solution for this.

2.4 Instructional Design for Computer Simulation

Fidelity is defined as how close a simulation imitates reality [8]. A linear relationship is assumed to exist between fidelity and the transfer of learning -- increase/decrease the fidelity and you increase/decrease the transfer of learning. There is no concrete evidence regarding this assumption because [8]:

- High fidelity means higher complexity which taxes memory and other cognitive abilities, and
- Proven instructional techniques, which improve initial learning, tend to lower fidelity.

These reasons lead to the hypothesis that the relationship between learning and fidelity is non-linear and depends on the instructional (knowledge) level of the student. The amateur student learns best from a relatively low-fidelity simulation, the experienced student learns best from an intermediate-fidelity simulation, and the advanced student learns best from a high-fidelity simulation. Some conflict is created here because increasing fidelity, which theoretically should increase transfer, may inhibit initial learning that in turn will backfire and inhibit transfer. This conflict should be taken into consideration for the inexperienced student.

When working with simulations the student is exploring and discovering new knowledge. This knowledge is best gained when the computer-based simulation that is being used has been properly designed (See Figure 2.4). According to Reigeluth & Schwartz [17], one must consider three major design aspects: (1) a scenario that recreates the real life situation, the scenario determines what happens and how it takes place. It also determines the role of the student and how s/he will interface with the simulation. (2) A mathematical model that reflects the relationships that govern the system and (3) an instructional overlay. The instructional overlay is that part of the program that optimizes learning and motivation. The authors have found that the nature of the content being taught is the major influence on the instructional features a simulation should have.

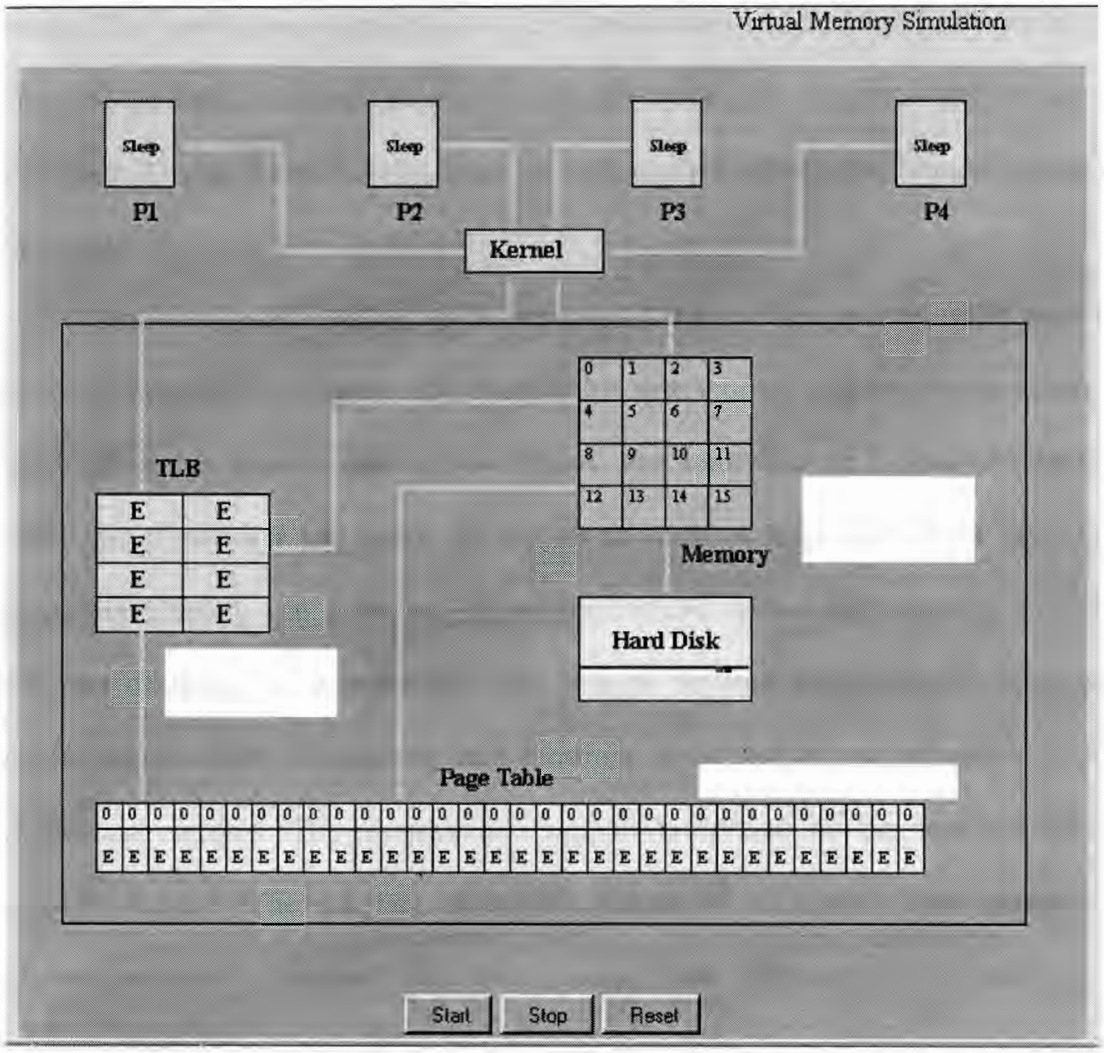


Figure 2.4 - The applet illustrates a simplified simulation of virtual memory implementation in a processor running multiple processes (four, in this example) and scheduling the processes based only on page faults by the processes.

These authors further have identified three phases in the learning process. The learner must first acquire a basic knowledge of the content or behavior. Then s/he must learn to apply this knowledge to the full range of relevant cases or situations. The final stage is an assessment of what has been learned. Therefore, the first set of instructional features in a general model for simulations should be concerned with acquisition of the content, the second set with application of the content, and the third with assessment of learning. The simulation is preceded by an introduction which should present to the student the scenario, goals and objectives, and directions and rules regarding the use of the program. During the acquisition stage the student should develop an understanding of the content.

Simulation-based learning can be improved if the user is supported while working with the simulation (See Figure 2.5). Part of this instructional support can be achieved through off-screen material such as a workbook, and sometimes an individual tutor can be available to monitor and assist the student at work. A large part of the necessary support functionality can, however, be accommodated within the computer program itself. The challenge is to strike the right balance between exploratory freedom and instructional constraint. The instructional constraint is divided into non-directive support and directive support. The non-directive support is defined by the interface of the computer program. The interface is the component of an instructional system that mediates between a student and the system. This interface serves only as a communication aid and has nothing to do with instructional process itself (simulation). As for directive support, it depends on the characteristics of the simulation, the characteristics of the student and the characteristics of the learning goals. The authors have elaborated on the concept of instructional support for simulations. He has tried to derive from cognitive theory, from instructional design theory and from existing exploratory learning environments instructional features (strategies, actions, approaches) that could be relevant to the design of simulations.

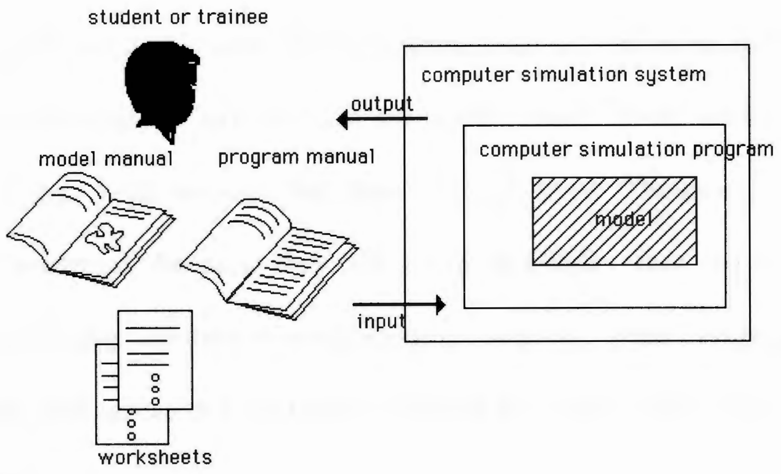


Figure 2.5 - The learning environment for a computer simulation session with a complete computer simulation program, included worksheets, manuals, hardware, software and courseware. The mathematical model is the heart of such computer simulation programs.

Properly designed systems for educational computer simulation programs have several distinct characteristics [18]:

- The operating system of the computer is never shown to the user.
- The underlying model is never shown to the user.
- The only input device for the user is the mouse.
- Some input can be accepted from the user.
- The output of the program can be presented in a highly graphical display.
- There are options with which the user can influence the running of the model.
- Changing model elements is possible.
- Some typical cases can be pre-programmed (sample situations).

- Additional coaching material can be provided on or off-screen.

Research with such simulation programs show that the additional materials were of utmost importance in the instructional aspect for the student [14]. In light of this, the notion of the Parallel Instruction Theory was proposed. The Parallel Instruction Theory states that students are best served when the simulation as well as additional materials in a simulation-based learning environment are within reach. If the additional material is paper-based, it has been noticed that they are not used. Therefore, computer-based materials are preferred, because they are more available. This led to an additional guideline for designing simulation-based learning materials, when designing materials the designer should take notice that all material should be within reach when the material is presented to the student.

Chapter 3

Digital Logic Simulation

Logic simulation is the process of building and exercising a model of a digital circuit on a digital computer. When the user exercises the simulation, the evaluation of signal values in the modeled circuit occurs for applied input variables. There are two main applications for a logic simulator. The first type of application is concerned with the evaluation of a new design. The logic designer is interested in testing for logic correctness, as well as signal propagation characteristics. For VLSI circuits, where design errors are very costly and breadboarding is impractical, logic simulation is an invaluable aid. A second application for logic simulation exists in the area of fault analysis. Here, the test engineer or logic designer may desire information related to faults that are detected by a proposed test sequence, or related to the operational characteristic of the circuit under specific fault conditions. The scope of this thesis will be focused on the former application.

3.1 Why Computer Simulation

Digital logic design is typically taught in a “computer organization” course. Textbooks usually use a series of figures to illustrate the life cycle of a combinational digital circuit. Students see raw numeric output on the screen after a program executes.

Simulation will offer students an alternative method to better grasp ideas presented in a classroom by providing a visual and graphical interface. A more effective way of teaching would be to use graphics to illustrate concepts because:

- Visualization is an extremely powerful form of education.
- Drawing pictures provides an easier way for a better understanding.
- Pictures present concepts better than numbers.
- Graphical interfaces best match the human perception.
- Graphics provide a more productive learning environment and promote the development of more intuitive understanding.

An interactive graphic system allows the user to input commands and see real-time pictorial output on the screen. The use of computer graphics in circuit design eliminates the need to draw and erase wires and gates on paper thereby saving considerable time. The manipulation of a circuit is easily updated graphically and eliminates errors generated by wiring inputs to the wrong gates. A typical student assignment will consist of designing a logic circuit and producing its correct output. Computer simulation enables the students to do their homework quicker and easier by building circuits on screen and verifying the correct output. In addition, it create a stimulating learning environment, better able to interest and educate computer science and computer engineering students. Students can experiment with the simulator at their own pace and investigate various logic circuits to reinforce conceptual understanding independent from the classroom and other students.

3.2 How Simulation Work

The central idea of component-based logic simulation is to build a simulation environment using software components. On modern personal computers or workstations, windows, menus, and file folders are usually represented by software components, which are the breakthrough concept of Object Oriented Programming. Traditionally, logic simulation is first accomplished by mapping a circuit description into a data structure. Then a piece of code called "simulator," which is most commonly event driven, is needed to interpret the data structure and execute the input vector. When implemented using the event driven technique, the simulator is able to exploit the infrequency of logic switching in a digital circuit. Only the logic gates with changing inputs are processed. In the case of event-driven simulation, each component simulates itself and then drives another component into the next cycle thereby executing the whole computation of the circuit. The simulation method that describes the behavior of the component does its interaction with the external world through the input and the output objects.

The event-driven technique is more adaptive, scalable, and efficient to perform simulation for digital systems with various complexities. The structural description might be at gate level, which consists of the topology of the circuit and the circuit element types, along with a list of primary inputs and primary outputs. A high level language is often used for describing input sequence, desired output format, etc. The simulator, therefore, is a program that interprets all the inputs, applies the input data to the digital system and computes the outputs.

3.3 Commercial Algorithms

Most of the commercially available tools that perform digital simulation basically work in the same way. First, a simulator is synthesized by mapping the entire digital system at the gate level into a piece of runable code. Then the simulator is executed to check for the behavior of the design. The fundamental element of a simulation environment is the formation of an object by the encapsulation of the behavior of a functional block within a digital system. The complete simulation process can be characterized as the interaction of many different objects. Each individual object self-describing its behavior is subject to external inputs, initial conditions, and the interchanging of inputs and outputs with each other.

Although we do not intend to compete with the commercially available simulation products, we will give a brief introduction to some of what is available to the public. (In the next chapter, we will introduce more information specific to logic simulation and its terminology).

- Time-Driven (Analog)

  In my undergraduate study in Electrical Engineering, I had the opportunity to use Spice, a VLSI simulation algorithm. Spice basically constructs a sparse matrix and solves design equations at transistor level by manipulating voltage and current. Disadvantages to the Time-Driven system are (1) huge calculations, (2) it is extremely time consuming, and (3) this system requires the user to read an extensive manual before any attempt can be made to use it.

- Event-Driven

Verilog, Vhdl, RSIM (all of which are simulation algorithms) construct an event list using a data structure. This data structure holds all the events generated and executes them in the order in which they are received. These algorithms operate at the gate level, and logic states are used as variables 1 (high) and 0 (low). Although propagation delay time is considered to be a problem, statistics show that for big digital circuits, only 14 percent of the gates will frequently have events, thereby making it an efficient technique.

- Compiled Mode

  Other software algorithms compile all the circuit description into one piece of executable code, and then define the initial and final states. The code is run during the cycle and pointers to the initial states and final states are swapped. This technique allows the next simulation to be run using the previous final state. Only state variables are of concern, and the intermediate values are not of the interest. This cycle based algorithm could be very efficient using native C code.

There are two types of event-driven simulation: discrete-event and continuous-event algorithms [2]. In discrete-event simulation, an event is defined as an incident that causes the system to change its state in some way. For example, a new event is created whenever a simulation component generates an output. A succession of these events provides an effective dynamic model of the system being simulated. The variables of a discrete system jump from one point to another and do not necessarily occupy intermediate values. Therefore, events in a discrete-event simulator can occur only during a distinct unit of time during the simulation. Discrete system simulations are often event oriented in that time is not advanced in uniform increments, and events are not permitted to occur in between time units. In contrast, the variables of a continuous-event system undergo transition from one value to another in a smooth continuous manner, occupying all intermediate values.

In the case of digital simulation of continuous systems, time is usually subdivided into uniform intervals and the simulation is "clocked" in terms of the basic time interval. What separates discrete-event simulation from continuous simulation is the fact that discrete systems are often characterized by difference equations, while continuous systems are described by differential equations. Discrete event simulation is generally more popular than continuous simulation because it is usually faster while also providing a reasonably accurate approximation of a system's behavior.

3.4 Limitations of Available Models

One of the digital logic simulators currently used by the Computer Science Department at the University of Rhode Island is Dizzy (See Figure 3.1) [22]. When the user first opens the software, a menu with several gate options as well as some editing options, is displayed to the side. The editing options include shifting the grid display around using a "hand" icon, deleting an object from the screen using a "zapper" icon, saving a circuit to a file, and opening a previously built circuit. Dizzy also custom saves a random circuit in the form of a black box, which allows for the use of modules. Although Dizzy works fine and allows the user to build many large circuits, it does have some rather serious limitations.

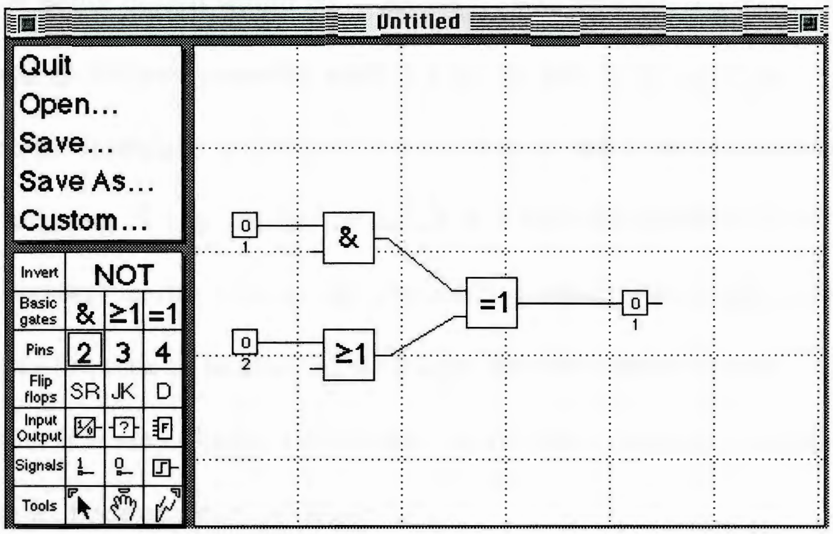


Figure 3.1 - Dizzy program shows the application window and its tool menu. Several objects are displayed that all have the same "square" box shape.

The first major limitation is that when Dizzy initially starts up it opens a fixed window. In particular, the user can not build a circuit larger than the size of the application window or screen thereby eventually limiting the number of gates that can be placed on screen. Secondly, all the gates offered in the menu have the same square box shape. Whether, an AND gate or a NOT gate is used, all objects look alike. This design is very confusing to students, especially students in an introductory level course. Logic gates should be displayed using the conventional logic symbols. In general, a clear graphical presentation is very important when introducing new material, not to mention when the software is used as a teaching tool. Thirdly, although Dizzy does save and open circuits, it does so without any safety feature. That is, Dizzy can save a circuit into a file that has a previous circuit already built in by overriding the existing circuit and saving the new one on top of the old one. Imagine the frustration of a student who has spent a considerable amount of time designing and building a logic circuit, only to have it erased by accident. A better design would be to provide a confirmation dialog box that asks the student to confirm the save operation needed once the save icon is clicked.

A greater limitation to Dizzy's functionality is that it is built only to operate on Macintosh platforms. It can not run on either a Windows platform or on Unix. Apple computers represent about 5% of the computer market worldwide, which gives an indication as to the serious limitations on Dizzy. Students cannot access Dizzy from the Internet because it is very platform dependent, and it cannot be easily embedded in a Web page as an applet because Dizzy is written in C.

Another logic simulator, called Logg-O [23], can be found on the Web (See Figure 3.2). Logg-O also starts up by opening a window with a menu of basic logic gates on one side. An editing menu, which makes correcting easy, is also added at the bottom of the screen. Logg-O allows the user to build a digital circuit on screen and provides a clear output signal in the shape of a light bulb which "glows" if the output evaluates to True. When a simple circuit was constructed on Logg-O, the correct output resulted. However, Logg-O does have limitations as well.

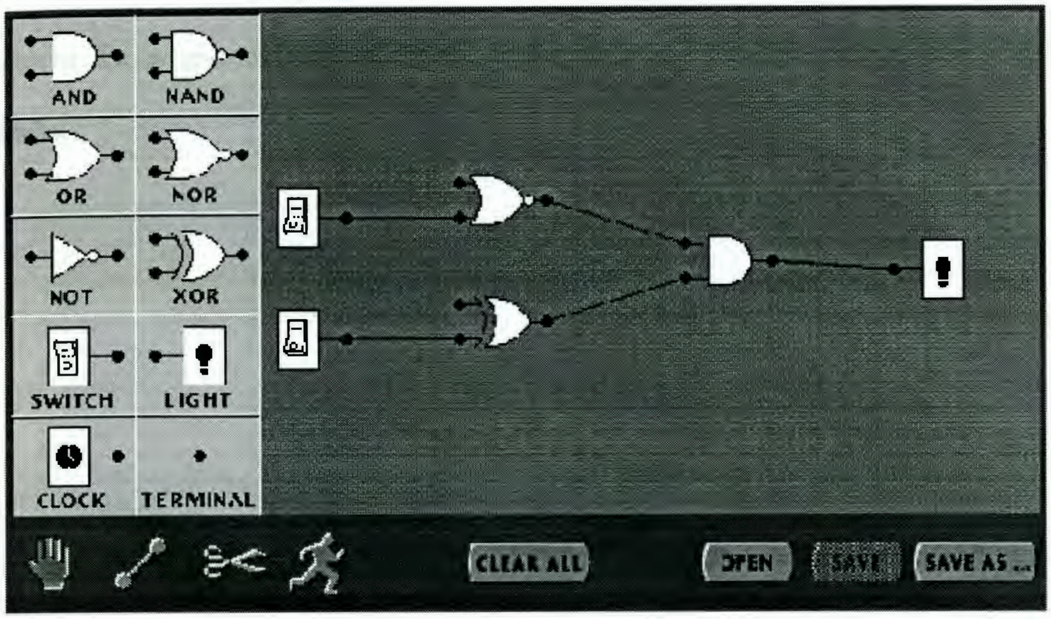


Figure 3.2 - Logg-O Program

When the program starts up, a fixed window is displayed, which, as discussed earlier, limits the size of the circuits built. In addition, the size of a single logic gate is so large that only a handful of gates may be placed on the screen at a time. This is the reason the test circuit used to demo the system is simple. The behavior of the program may be undetermined if a large circuit is constructed. The only way to "test drive" the stability of this system is to construct a large circuit, and this is not feasible. Furthermore, if an input signal is introduced into the system using a considerable circuit, evaluation time may be unacceptably long.

Although both Dizzy and Logg-O produce accurate results and are fully functional, both seem to be unintuitive to an amateur student. An interface design with an intuitive approach enables the user to operate the software in a very timely manner and without much struggle. Students encounter enough challenges with class assignments, never mind spending more time trying to figure out how a program works. It is always imperative for a graphical interface to be as clear as possible and follow industry standards.

Chapter 4

System Specification and Features

One of the objectives of this thesis is to prototype a simulation system, using the Java programming language, that will allow students to access the internet from anywhere to run the simulation program. The development of this digital logic simulation provides a good demonstration of object-oriented technology. In the next few paragraphs, Java is briefly discussed in order to show why we chose to use it, and why it is revolutionizing the Internet. Features and drawbacks of Java are also presented. Then we will introduce LogicCity, our prototype model simulator and instructional software. LogicCity contains many features and functions, which will all be presented in detail. A thorough discussion of the internals of the system will be discussed in the following chapter.

4.1 The Java Programming Language

According to Cornell [3], the promise of Java is to become the universal glue that connects users with information, from Web servers, databases, and any other imaginable source. It is a very well designed language with a solid foundation that incorporates syntax very familiar to programmers. Object-oriented programming is considered the break through in practice, and Java is thoroughly object-oriented even more so than C++. Everything in Java is an object (except for basic types like numbers). Yet like all other programming languages, Java does have its advantages and disadvantages.

A main key advantage is Java's run time library that provides platform independence; that is, you can use a Java program on any other operating system including Windows, Solaris, and Macintosh. The compiler generates a neutral byte code, which is easy to interpret and execute on many processors. Debugging a Java program is much easier than debugging a C++ program. Manual memory allocation and deallocation is eliminated in Java because memory is automatically garbage collected. True arrays that eliminate pointer arithmetic are introduced, and multiple inheritance is replaced with a much easier interface that has the same functionality. Java has an extensive library of methods for dealing with TCP/IP protocols like HTTP and FTP. Java applications can open and access objects across a network via URLs as easily as accessing a local file system. Also since Java is intended to be used in networks and distributed environments, much consideration has been placed on security. Basically, Java's security system prevents a program from corrupting memory outside its own process space and prohibits reading and writing local files.

Java is simple; a user can download Java byte code from the internet and run it on a local machine. Java programs that work on Web pages are called applets. To use an applet, one needs a Java-enabled Web browser that will interpret the byte code downloaded. A Java program can also be run on a stand-alone basis, in this case, it is referred to as an application. Both applets and applications have advantages and disadvantages, which are summarized in Table 1. Following are a few common misconceptions of Java, which are based on a review done by Cornell [3]:

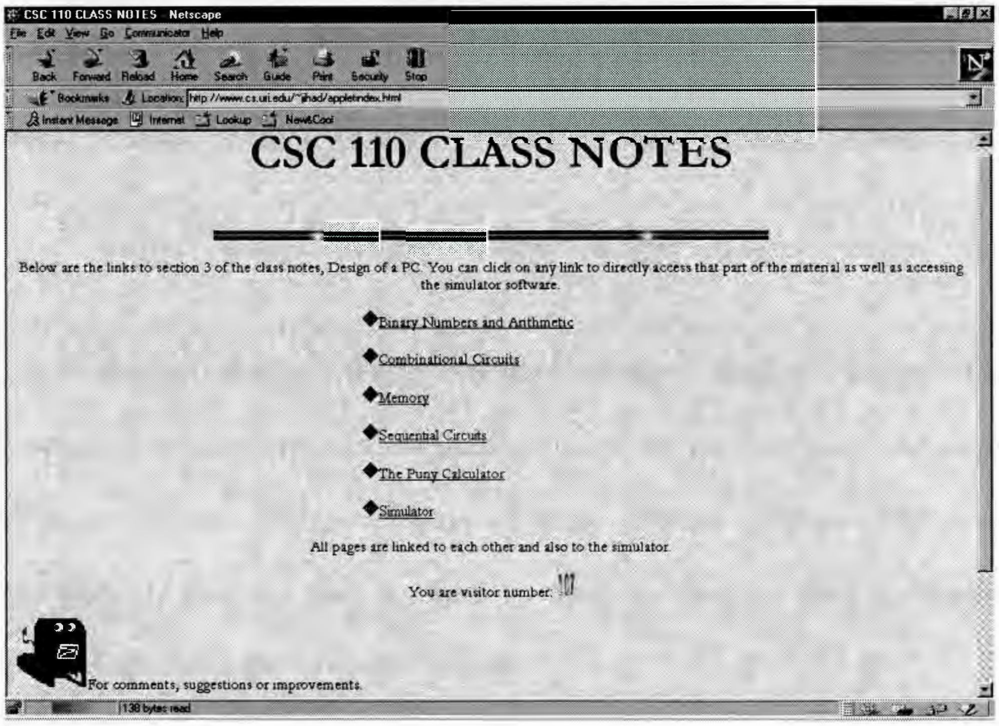
- Java is not an extension of HTML. They have nothing in common except that there are HTML extensions for placing Java applets on a Web page.
- No programming language as powerful as Java is as easy as Java to learn.
- The native Java development environment is not an easy environment to use.
- Java is interpreted, but it is not too slow for serious applications. All programs, regardless of the language used, will detect mouse events in adequate time. There are no serious performance issues once an interpreter is used to translate the byte code.
- Not all Java programs run inside a Web page. All Java applets do run inside a Web page. However, stand-alone programs that run independent of a Web browser cannot always be run in a Web page.

Operation	Applet	Application
Access local files	N	Y
Delete a local file	N	Y
Run another program	N	Y
Find out your name	N	Y
Connect back to the host	Y	Y
Connect to another host	N	Y
Load the Java library	N	Y
Create a pop-up window	Y	Y
Call System to exit	N	Y
Create or list directories	N	Y

Table 1 - Comparison of Applications vs. Applets

4.2 The System's Web Site

LogicCity's design and implementation using this programming language, is based on the many advantages of Java and its ability to use the Internet. LogicCity consists of two integrated parts, suitable text-based material and interactive software. As students access LogicCity's Web page, they are presented with the relevant material for that course (See Figure 4.1). The course material is divided into several parts and each part is linked to the next. A student can browse through the various parts as desired as well as link to the software. At any time, the student is able to intermix reading with using the software provided, which we feel, provides students with the exploratory environment we mentioned earlier.


CSC 110 CLASS NOTES - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://www.cs.uri.edu/~jihad/appletindex.html
Instant Message Internet Lookup New&Cool
138 bytes read

CSC 110 CLASS NOTES

Below are the links to section 3 of the class notes, Design of a PC. You can click on any link to directly access that part of the material as well as accessing the simulator software.

- Binary Numbers and Arithmetic
- Combinational Circuits
- Memory
- Sequential Circuits
- The Puny Calculator
- Simulator

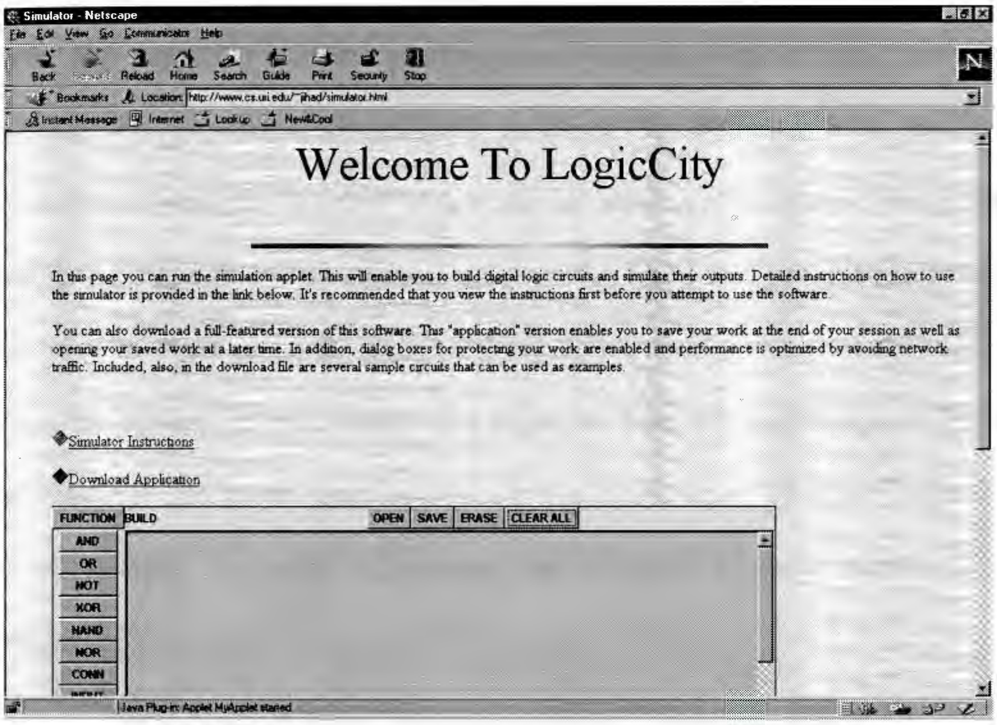
All pages are linked to each other and also to the simulator.

You are visitor number: 107

For comments, suggestions or improvements.

Figure 4.1 - Main Web Page

When accessing the simulator page, students have the option of using the simulator directly while on-line in an "applet" format or downloading the full-featured "application" version (See Figure 4.2).


Welcome To LogicCity

In this page you can run the simulation applet. This will enable you to build digital logic circuits and simulate their outputs. Detailed instructions on how to use the simulator is provided in the link below. It's recommended that you view the instructions first before you attempt to use the software.

You can also download a full-featured version of this software. This "application" version enables you to save your work at the end of your session as well as opening your saved work at a later time. In addition, dialog boxes for protecting your work are enabled and performance is optimized by avoiding network traffic. Included, also, in the download file are several sample circuits that can be used as examples.

◆Simulator Instructions

◆Download Application

Simulator - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://www.cs.uiι.edu/~jihad/simulator.html
Instant Message Internet Lookup New&Cool
FUNCTION BUILD
OPEN SAVE ERASE CLEAR ALL
AND
OR
NOT
XOR
NAND
NOR
CONN
Java Plug-in: Applet MyApplet started

Figure 4.2 - Simulator Web Page

A recommendation is given to read the simulator manual and documentation in the link provided if a question arises regarding how to use the simulator. The zipped file received is quickly and easily decompressed using standard system utilities found on many platforms. In this case, the simulator is considered a stand-alone application and only needs a Java interpreter to run the program.

4.3 LogicCity

Assuming that the application version is downloaded and after the simulator program begins, a new frame (window) with the title "Welcome to LogicCity" appears at the top (See Figure 4.3). This circuit window is to be used for drawing a logic circuit schematic. This frame has standard window functionalities meaning that it can be moved around the screen, minimized, iconized or even closed. The default is set to maximize the window to the screen size in order to provide the greatest workspace. This workspace (window) is not fixed in size so students may use the scroll bars provided to scroll up or down as well as right and left. This scrolling ability increases the magnitude of the workspace by adding several screen size pages to both the height and width of the original window. The extra space allows for the construction of many large circuits, thereby eliminating the worry of running out of room to place additional circuit components.

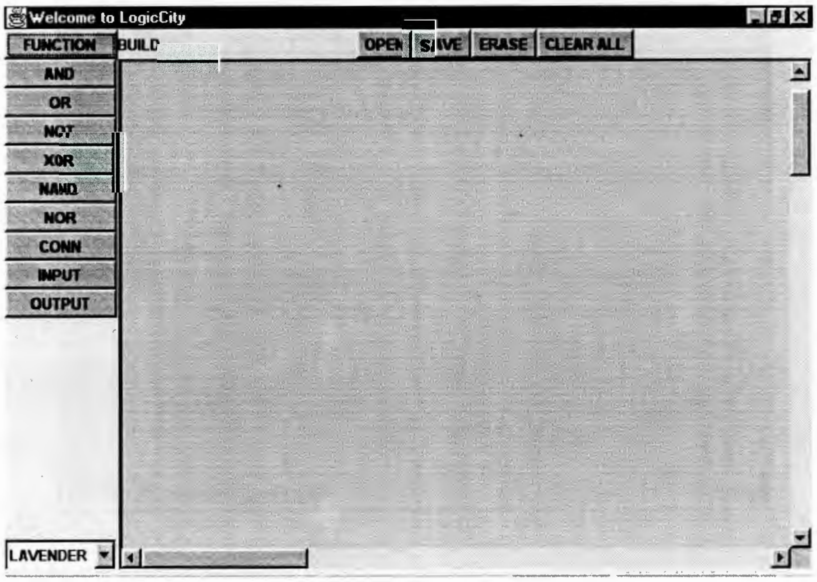


Figure 4.3 - LogicCity's Window

Menus are available at the top and on the left side of the window. The default setting for the simulator is the edit or build mode. In build mode, students can click on any gate from the gate menu and place it on the screen. There are several basic logic gates provided on that menu, namely, AND, OR, NOT, XOR, NAND, NOR. There is also a menu for inputs, outputs and connectors (See Figure 4.4). These are used to input signals into a constructed circuit, and to receive output signals from that circuit. Connectors can be used to tie components together and route wires around in a clean and neat manner. The prototype's graphical interface lets the student select and modify different input scenarios. The user is also given options for the canvas background color. At the top of the simulator window, there is yet another menu for editing which provides facilities for opening a circuit saved to a file, or saving a circuit currently being built. The edit menu also has means for the deletion of a single component from the circuit being built or the option of deleting the whole circuit, thus, clearing the entire screen.

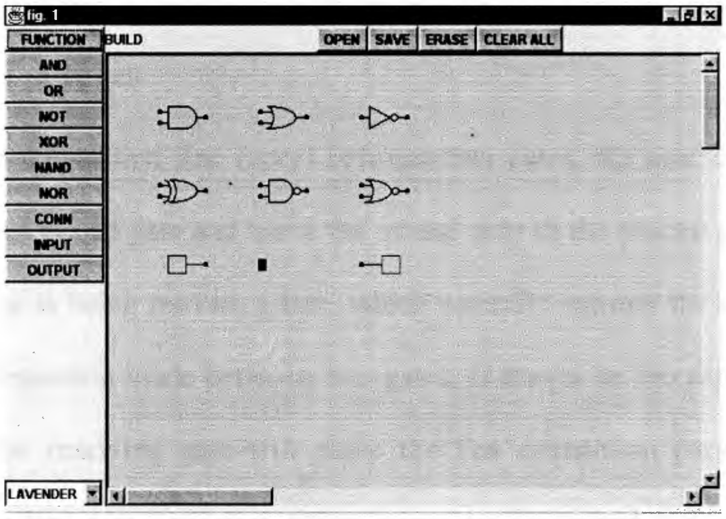


Figure 4.4 - Basic Gates

The intuitive and appealing format of the simulator's graphical interface allows a student to merely look at the window in order to figure out the functionality of the interface. This intuitive design reduces the difficulty the learner will have with the system and allows the student to concentrate on the circuit being designed rather than on how the system works. The simulator interface, which operates via the mouse, is fully interactive. The user need not use a keyboard or any other input device. The consistency of functions makes the system easy to learn and use. This form of interactive interface provides a more dynamic and stimulating environment for the student that encourages learning and exploration. To start building a combinational digital logic circuit, the user just needs to click on the desired gate from the gate menu. The gate menu is composed of buttons so that once a gate button is clicked, the user moves the mouse over to the canvas area and places the selected gate at the desired location with another mouse click. The user can continue to place gates on canvas in the same manner.

4.4 Software Features

To draw a connection line (wire) between two gates, the user can click on the input or output pins of one gate and move the mouse over to the second gate (See Figure 4.5). As the mouse is being moved, a line, which basically mirrors the mouse's motion and shows the connection made between two gates, is drawn on the canvas. A second mouse click at the receiving gate will make the line connection permanent, thereby creating what is called a solder point. Other gates can be created and connected in the same fashion. In order to provide straight routing wires around gates a connector object is created and placed on the canvas in the same manner as gates are. Any gate can send input into the connector object; however, there is no limit as to the number of outputs the connector can send out. The simplicity created here is that one gate can be used to send many outputs to several other gates. As a matter of fact, all gates can equally send multiple output lines to any other object available. Logically, a gate can accept only one input signal at a time, but it does not matter how many output signals are sent out.

Once a logic circuit is designed and constructed, input and output objects are needed to facilitate the introductions of signals into and out of the circuit (See Figure 4.5). For that reason, two more buttons are displayed next to the connector button that allow for the creation of input signals as well as output signals. Input signals are displayed as a square box with an outgoing line and their default value is set to either low or logic zero. A blue color was chosen for this box to visually give the illusion of an initial low (cold) input signal. The output object is displayed in the form of a light gray square box and is initially set to low. The gray color gives the illusion of a low (dimmed) signal to indicate a "dead" output line but when in simulation mode this box glows in red if the output signal evaluates to high.

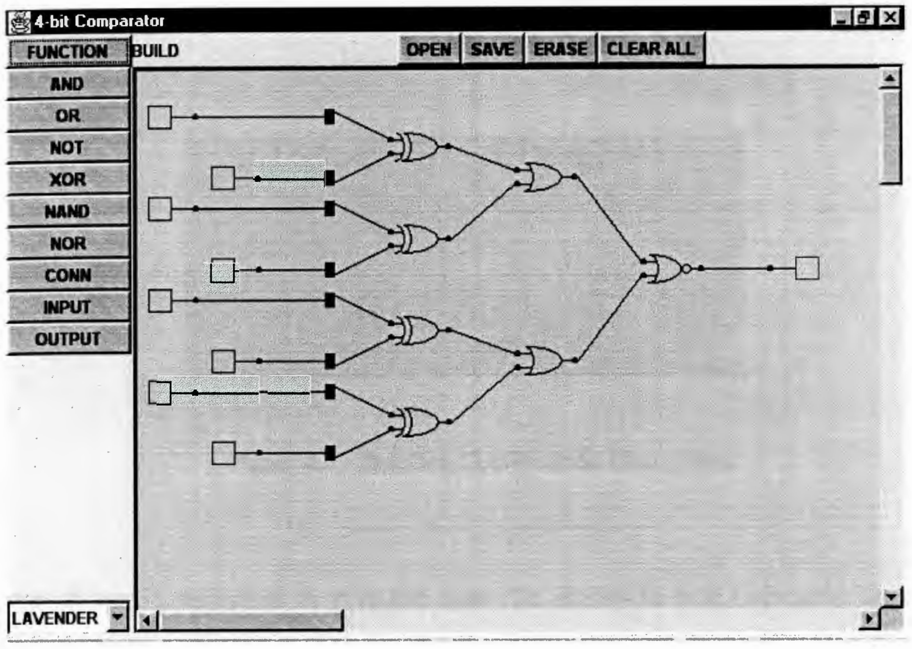
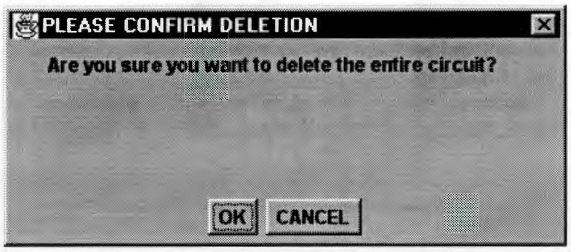


Figure 4.5 - Digital Logic Circuit

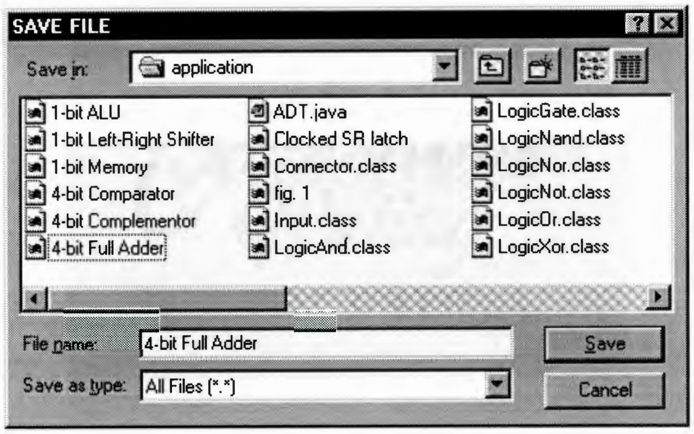
If errors occur while building a circuit, components can be easily erased by simply clicking on the erase button from the edit menu at the top of the screen and then clicking back on the object to be erased. Like the other buttons this erase button is activated only once per mouse click in order to prevent the accidental erasure of more gates. Any time an object is deleted, all the attached wires connected to this object are deleted as well. In addition, all objects that were connected to the deleted component are reset to their initial defaults. If the user wishes to delete everything on the screen s/he simply clicks on the "clear all" button. A warning dialog box (standard window) will then pop up to confirm the deletion of the entire circuit (See Figure 4.6). If the user chooses "cancel", nothing happens and the dialog box disappears, but if the user chooses "ok", then the canvas is cleared and the whole circuit is deleted. The "clear all" button is deactivated if clicked on an empty screen, and control is given back to the user to rebuild a new circuit, go in a different direction, or just close the application.



Are you sure you want to delete the entire circuit?

Figure 4.6 - Deletion Confirmation Dialog Box

A scroll bar is provided to give the user the ability to build circuits larger than the size of the screen. Students can "turn" pages easily while viewing or working on a large circuit design. With several pages of available space in both screen dimensions, logic circuits may grow to be quite complex. This complexity necessitates a save button on the edit menu to allow students to save their work to a file at the end of a session. After a circuit is put together, a student may easily save his/her work by clicking on the save button. A dialog box, which permits the user to enter the name of the file to be saved, pops up (See Figure 4.7). This modal window has the same functions and appearance as in a Windows environment and is provided by Java. Briefly, modal dialog boxes cease the running application until some command or option is entered. This prevents the accidental mistakes that could prove devastating. As soon as a valid name is entered, the simulator saves the current circuit to file and confirms this operation by displaying the new name of this file at the top of the window. If the canvas is empty, that is, there is no circuit displayed on screen, the save button is deactivated.



application

1-bit ALU	ADT.java	LogicGate.class
1-bit Left-Right Shifter	Clocked SR latch	LogicNand.class
1-bit Memory	Connector.class	LogicNor.class
4-bit Comparator	fig. 1	LogicNot.class
4-bit Complementor	Input.class	LogicOr.class
4-bit Full Adder	LogicAnd.class	LogicXor.class

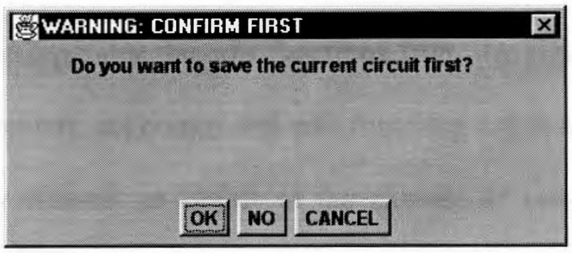
File name: 4-bit Full Adder

Save as type: All Files (*.*)

Figure 4.7 - Save File Dialog Box

Students are able to recover their saved work by clicking on the open button. Then, if the canvas is empty, a window with the standard Windows format appears. (See Figure 4.7). As in the save situation Java provides the functionality of the open dialog box that looks exactly the same as the "save" box.

The name of the file selected will be displayed at the top of the window; the simulator always displays the name of the file or circuit being worked on at the top of the screen. The name of the open file changes appropriately as the student moves from one circuit to another. If there is a circuit currently occupying the canvas at the time the open button is clicked, a safety feature is activated which prevents the destruction of the present circuit and prompts the user to save the existing work. A modal dialog window with the options "ok", "no", or "cancel", pops up (See Figure 4.8). The student is then given the option to either save the current circuit first, or to open a new one on top of it. Students also have the option to cancel the open file call altogether. The circuit recovered at that point is ready either for editing, addition, or just simulation. Before any circuit is opened or placed on the canvas, the entire screen is first cleared to eliminate any confusion or mistakes.



Do you want to save the current circuit first?

Figure 4.8 - Save Confirmation Dialog Box

Another very useful feature provided, is the ability to move objects around the canvas in order to scatter any congestion in any one spot of the screen or to pack together some components to save some room and present them in a distinct manner. A student can click the mouse on a component, hold it down and move the mouse to any coordinates on the canvas. The simulator then automatically updates the canvas by showing the component in the new location. If this component happens to be connected to other objects by means of wires, then all the attached wires are automatically moved along with the component.

Designing and building logic circuits may take many sessions of extended periods of time. Students' eyes often get irritated with the screen glare, especially from applications with the standard white background color. To help minimize this irritation, several "dimmed" colors are provided for background schemes. A student can click on any color item listed in the pull-down menu to instantly change the canvas background. A click on the canvas afterwards activates the new color selected. These colors also aid in the overall appearance of the graphical interface. Pull-down menus provide content and composition support through choices.

All circuit editing is done with the function button set to "build" mode, which acts as a master switch that separates the edit functions from the simulation part. Once the button is clicked to simulate, all menus and edit functions are disabled so that a student may no longer add or connect an object to the canvas, or save and open a file. To simulate a circuit, a simple click on the function button begins the evaluation of all signals entered into the circuit. These signals are propagated from the origin of each input object all the way through to the end of each logic route. State values are then displayed on each part of each object in the circuit (See Figure 4.9). Low state values (zero) are shown in black while high (one) values are shown in red. This visually provides a clear distinction of low and high signal in a circuit. It also enables students to easily track changes in the behavior of the circuit.

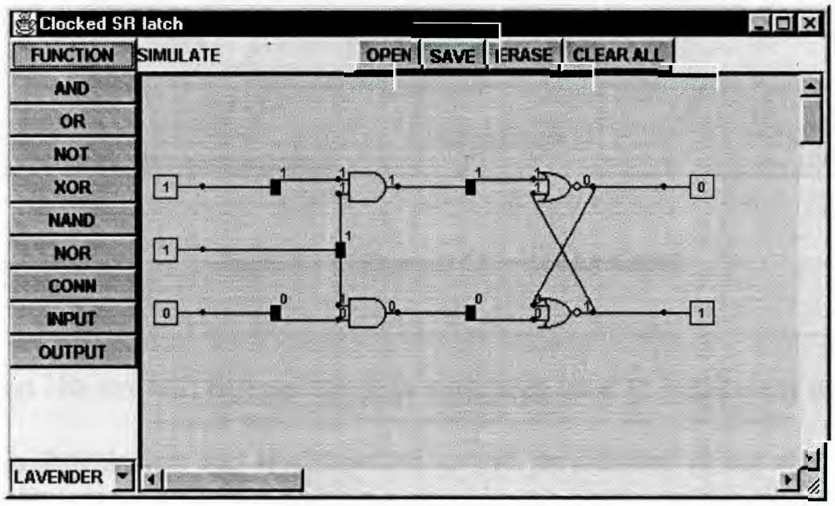


Figure 4.9 - Simulated Logic Circuit

Initially, all input signals are set to low; however circuit design always necessitate the introduction of several high input signals (See figure 4.10). This is accomplished by clicking on the input object desired. A click toggles the state of the input object from low to high; another click on the same object resets it back to low. Only input objects are allowed to change their states when reacting to mouse clicks because any other way would defeat the purpose of the object. Once an input signal is toggled to high, its state value is displayed in red along with the sending wire. The output object (if any) that is getting this high signal will glow in red to visually confirm the "hot" line just received.

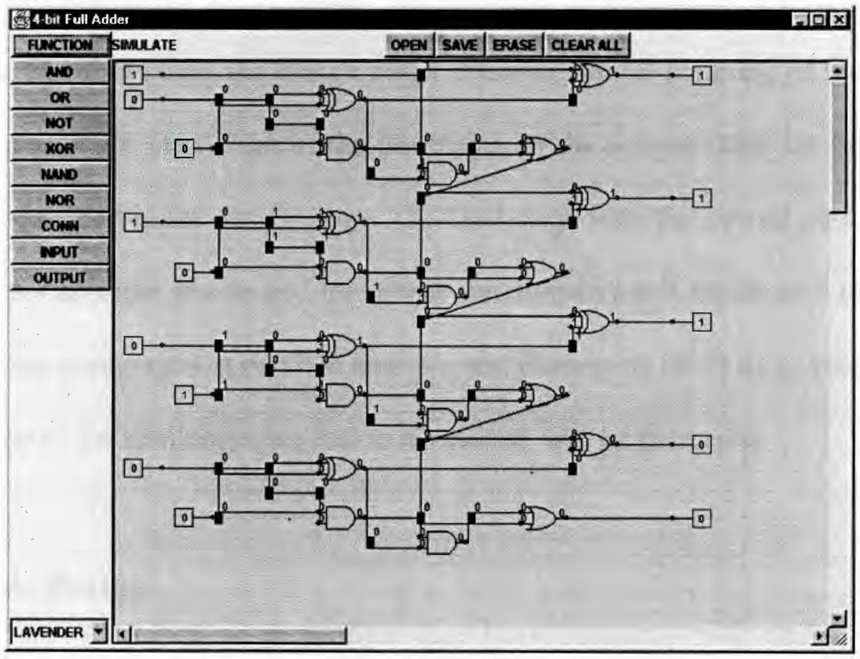


Figure 4.10 - Example Circuit (4-bit Adder)

The next chapter will discuss the data structures used to implement the simulator software. Algorithm design and implementation will be detailed along with challenges that were presented in the implementation phase.

Chapter 5

Data Structures

LogicCity is an easy to use tool for entering a logic circuit schematic and performing simulations of its behavior. The creation of LogicCity was done in three major stages. The first stage, the design stage, entailed careful planning of the necessary data structure used for the design of the prototype. In the second stage we implemented the methods specified in the design stage. The final stage tests the overall performance of the software for accurate results and for proper functionality and appearance on the Web. In the following paragraphs, a detailed analysis and discussion of all three stages, as well as a discussion of the challenges we had to overcome, will be presented.

5.1 System Design

Since there is usually more than one way to go about solving a problem, it is always worthwhile to imagine several solutions and then compare their advantages and disadvantages in order to insure a complete and thorough search. Solutions to more complex problems might typically involve multiple layers of data representations and many iterations of the stepwise refinement process. In addition, an important skill for computer scientists to master is reasoning because it is key when creating and debugging algorithms, when figuring out how to improve an algorithm's performance, and when verifying that methods inside a larger program work correctly.

Deciding on a "good" data structure is crucial because data handling and manipulation can be extremely simplified. We took a divided approach, both "top-down" and "bottom-up" design techniques, to develop the data structure for the simulator application. We eventually arrived at an executable Java program. In the "top-down" approach, we began at the highest conceptual level by imagining a general abstract solution and by sketching a program strategy in outline form before choosing any particular low-level data representations or algorithms. Then, step-by-step in progressive refinements, we filled in the details to implement the parts of the previously sketched solution in higher-level description. In order to implement the parts in parallel, we found it easier to use the "bottom-up" approach, where we first considered what information was needed, and then how to obtain this information. Next, we made a draft of definitions for the methods before using them inside other higher-level methods. This brought us to the details of the bottom layers of the overall program before filling in the higher levels that use the methods we had already written at the lower levels. This approach is not always the best approach for programming because one may reach a point where the methods already defined become inadequate and need to be redefined as the application evolves.

The design process began by developing the model's data structure. This data structure must be able to hold all the information needed to run the simulator application while simultaneously providing the user the ease of running the application. The data structure must have a clear interface while concealing all the details of implementation from the user because in reality a user does not care how a system is built as long as s/he can use it satisfactorily. The second step in the design stage was researching, selecting, and then developing the programming language. One of the big advantages to this simulator is that it is accessible to the public through the Internet, so we chose Java, the Internet language, as the programming language for LogicCity. Since LogicCity is platform-independent, any person with a computer and a Java savvy browser can run the simulator applet. Java is currently the only language that brings dynamic web pages to life and allows for animated graphics and multimedia.

The topmost level of the object-oriented design for the simulator application consists of three interacting Java objects, each of which encapsulates its own hidden, private data fields. These objects interact only by making method calls on each other's "services" (where the services are the publicly available methods in each object's interface). There are no global variables that hold shared values or data structures on which different objects operate. Consequently, the design is an example of pure modular programming in which the modules (the Java objects) interact only in clean simple ways through their interfaces with their internal mechanisms hidden from view.

In the design stage, we had to declare the major objects of the data structure essential to holding the data to be processed. Several drafts were made before a final one was realized. The higher level abstraction of the data structure is divided into three main parts. One part declares the attributes of the basic logic component (gate), the second part declares the operations (methods) invoked on such a component, and the third part builds the actual graphical interface. There are no dependencies on any particular operating system since Java is platform independent. However sufficient memory, which is available in almost all computers today, is required to run the simulation program.

LogicCity is totally object-oriented in design, meaning that an object bundles together data and some behavior. An object's data can be a collection of variables whose values give the object's internal state, whereas an object's behaviors can be a set of operators that change its state. These data and variables are hidden from the user and are encapsulated in a "module" package that a user can have access to only through invoking its available methods. There are a couple of benefits to the encapsulated object format used for this project. One benefit is that the application's GUI (Graphical User Interface) is an event-driven interface in which mouse events (generated by the user clicking the mouse on the screen) must be handled by event-handler code. The application and GUI must be set up to handle mouse events so as to trigger method calls on the problem-solving objects, to receive the computed solutions they return, and to display the results graphically on screen. Another benefit is that the canvas code that translates from mouse events to computed solutions is not completely trivial. The event handling and data manipulation mechanisms are complex enough that using the module as an encapsulated entity saves us from the effort of having to reinvent them and debug them in another program. This benefit creates important enhancements for software productivity for the future.

5.2 System Implementation

The overall organization of the data structure is composed of an abstract object (super class) of a logic gate, and a simulator object that sets up the graphical interface which in turn calls methods for acting on a logic circuit. The simulator object is one of the main components in the data structure that creates the top-level window (See Figure 5.1). This top-level window is derived from the Frame class supplied by Java. The simulator object uses one of the most sophisticated Java supplied layout managers. The layout manager, called Grid Bag Layout, arranges all components in a panel into rows and columns. Each component is told to occupy one or more of the little boxes on the screen. The idea is that this layout manager allows you to align components without requiring that they all be the same size since you are only concerned with which cells they will occupy. Incidentally, the purpose of layout managers is to keep the arrangement of the components reasonable under different font sizes and operating systems. The simulator object controls the master switch that enables the system to either build a logic circuit or simulate its behavior. It also owns the main() method that starts up the entire system. In main(), a simulator object is created, initialized, and displayed to the viewer.

The simulator object is parent to three panel components -- the editMenu, the gateMenu, and thePanel. The editMenu contains circuit edit buttons such as “save”, “open”, “erase” and “clear all”. These buttons invoke actions to be taken on an individual basis when the user triggers specific events. That is, each button calls on its actionPerformed method to process the particular event. The gateMenu contains all circuit parts in a palette format that a user can choose from and consists of several basic logic gates, a connector object, and input and output objects.

The third component that exists in the simulator top level window and created by the simulator object is thePanel object. This one acts only as an intermediary by setting up the stage for the system to initiate the graphical interface by creating theCanvas

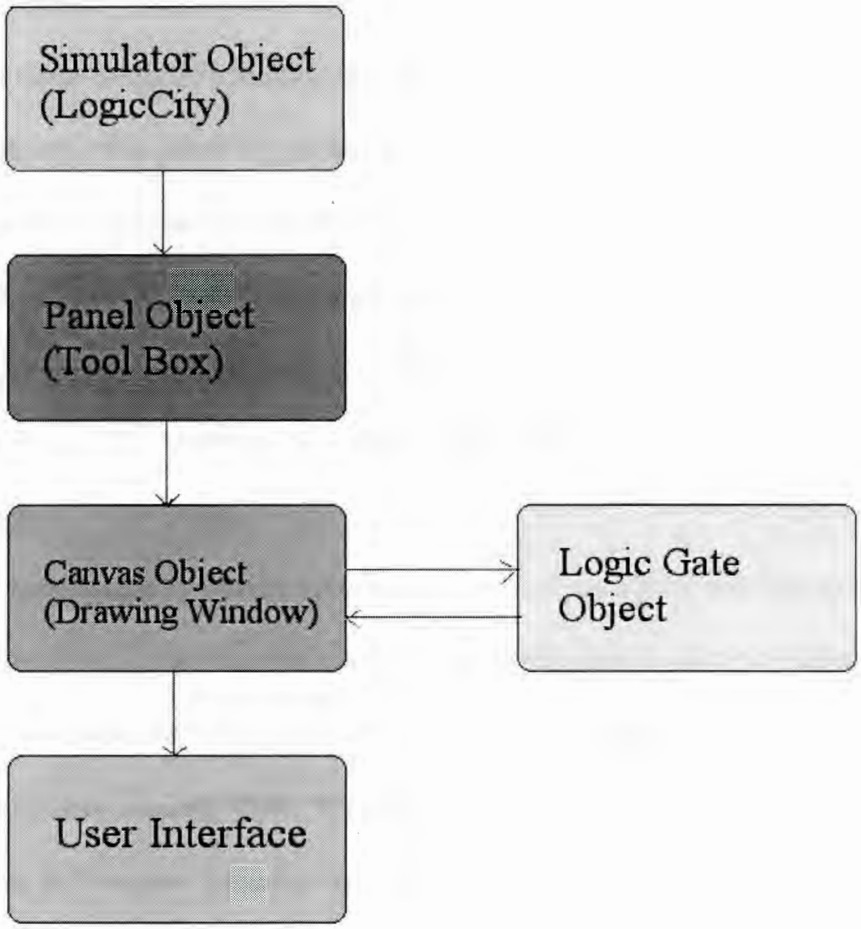


Figure 5.1 System Architecture

object. A canvas is simply a rectangular area in which you can draw, while a panel is a rectangular area into which you can place user interface components. thePanel class is needed to satisfy the Java compiler's requirement to have a panel object create a canvas component. Consequently, this panel object has no other important role except creating the scrollbars object. The scrollbars object has the same hierarchy as the canvas and has to be instantiated by thePanel object. This component allows the user to 'turn pages' on screen by displaying vertical and horizontal scrollbars on each dimension of the canvas window, thereby allowing the creation of circuits larger than the actual size of the application window.

The primary component that thePanel creates is theCanvas object. theCanvas is a component derived (inherited) from the Canvas class that is supplied by Java and accepts input from the user. The main attributes (variables) to theCanvas object are logicCircuit that holds the gate objects created. Here, logicCircuit is implemented using the Java Vector class. Vectors in Java are basically dynamic arrays that expand at run time on demand. We chose to use vectors because they can hold any object type and due to the nature of the logic circuit. Components in a vector are easily and quickly located so that if a method needs to access the ninth gate in the circuit, it can go directly to that position by calling on the array index at that location. This feature simplified the design of LogicCity.

The fact that vectors are dynamic arrays means that a circuit can grow and evolve without limits (assuming sufficient memory is available). Logic gates are initially instantiated and placed in the canvas component in order for the user to visually see them on the screen. Gates are easily manipulated in theCanvas, which offers flexibility since they can be moved around the canvas, erased, or even totally cleared from the screen. Logic gates can be connected (wired) to each other by simply applying the proper mouse clicks and then theCanvas automatically takes over to do the rest of the work. Then, wires are drawn in the proper positions and the data structure that stores the wire connections is adjusted accordingly.

Another attribute of theCanvas is that it has two variables, currentState and currentGate, that indicate the current status of the circuit, as well as a third variable, mouseClickPoint, that holds mouse clicks. Gate positions are also kept track of in xCoordinate and yCoordinate. Because the simulator application is interactive in nature, many responsibilities, such as handling mouse clicks, are placed on theCanvas. The constructor (initializer) of theCanvas object registers the object to receive triggered events of mouse actions from all the buttons in the interface. Each time the mouse button is pressed on a component by the user, a mouse event is generated and is passed as a parameter to its designated listener to be handled. This mouse event parameter is called MouseEvent object and contains data fields giving the x and y coordinates of the "mouse down" point. Events generated by the mouse account for several different events including events triggered when the mouse is clicked, moved, dragged, entered a component, exited a component, or if it is pressed or released. Once an event is trapped, a notification is sent to any registered component in order for the programmed response to take effect.

Since the simulator application is a totally interactive graphical interface, mouse event handlers do most of the necessary work. For example, dragging the mouse around moves a gate around the screen. Therefore the mouseDragged method must be defined in a way that accomplishes this task. In order for theCanvas to handle all mouse events, it must define the interfaces that supply these mouse events. There are three interfaces in theCanvas -- ActionListener, MouseMotionListener, and MouseListener. Combined they provide several mouse methods, all of which must be defined to handle certain conditions of the system. ActionListener has only one method, actionPerformed, to be defined. The objective of actionPerformed is to tell the canvas which button in the system has been clicked. For this method we had to invent analysis techniques to determine which button is clicked since the most current version of Java does not provide facilities for doing this yet. Once a button is identified, this method sets a pointer to the object.

MouseMotionListener is responsible for providing two mouse action methods, mouseMoved and mouseDragged. The objective of the mouseMoved method is to keep track of mouse motion by setting the xCoordinate and the yCoordinate variables accordingly because this method is invoked every time the mouse is moved on screen. The mouseDragged method is called for every time the mouse is dragged on an object on screen and is responsible for locating the dragged component, updating its canvas coordinates, and refreshing the screen display.

MouseListener provides several mouse-related methods, which are mouseEntered, mouseExited, mouse Pressed, mouseReleased, and mouseClicked. The mouseEntered method is utilized to initialize a logic circuit once the mouse has entered the dimensions of the "Function" button that switches to simulate mode. The mouseExited and mouseReleased methods are left undefined at this time. They are included in the implementation however to satisfy the compiler's requirement. The mousePressed method is defined to select one of the several color options provided by the Choice object. The latter is a pull down menu that allows for different canvas background colors. The mouseClicked function is the most important mouse event method. This method is sensitive to the user's communication with the simulator and is called for every time the mouse is clicked on a registered component. The mouseClicked method is responsible for the instantiation of new gate objects and placing them into theCanvas component. Input values to the circuit are toggled with mouse clicks. In addition, all menus on the interface are operated through mouse clicks. It is also responsible for the wiring of the different objects. Moreover, it is through the mouseClicked method that gate objects can be edited or simulated.

The runSimulation method is responsible for performing the actual simulation of the behavior of the logic circuit and displaying the results of this simulation on screen by displaying the logic states of every individual gate object in the circuit. This method displays "0" to denote a logic low or "1" to denote a logic high. The runSimulation method also allows the user to change the states of any arbitrary gate object on the screen depending on the circuit design. By clicking on the input objects to a particular component, the user can easily change the state value of the component.

The canvas also allows for two dialog boxes to be instantiated. These dialog boxes serve as a safety feature against the accidental destruction of an unsaved logic circuit. Input is received form the user through these dialogs and processed accordingly. We accomplished this by creating two new objects, each handling a dialog box separately, which allows for providing a different warning message on each dialog box plus different option buttons.

In Java the various GUI building blocks, such as buttons, input areas for text, and scrollbars, are usually called components. The user interface is constructed using these various building blocks, and one can program the interface to respond to various events. These building blocks are called "controls" in Windows programming and "widgets" in X-Windows programming. To build the user interface, we needed to first decide how the interface should look, particularly what components are needed and how they should appear. Unfortunately since Java lacks a form designer (like Visual Basic has that generates code templates), we had to write code for most everything. We had to write code to make the components in the user interface look the way we want them to look, and we had to layout (position) the user interface components where we want them to be inside the window. We also had to handle user input by programming the components to recognize the events to which we want them to respond.

In order for the system to display the animation resulting from the circuit, the paint method in theCanvas must be overridden. The paint method handles the drawing of the whole simulation in one central location, which eliminates the need to pass the circuit around among the different objects. This also simplifies the implementation of the method since theCanvas holds the information about the circuit. In addition, memory resources and run time are saved which is important for large and intricate circuits. The paint method checks if a gate is being moved around on screen, and if it is, then the gate is removed from its position and visually moved to its new location. The paint method loops through the circuit vector extracting the i^{th} circuit object with a method called logicCircuit.elementAt(i), typecasting it to be a LogicGate object, and finally sending the message displayGate(g) which essentially tells each gate to draw itself into the canvas.

When each individual circuit object receives this displayGate message, it invokes its own custom displayGate method that overrides the abstract displayGate method of the abstract LogicGate class. This is the true power of type polymorphism since each object knows how to draw itself. A variable tempGate, declared to be of abstract type LogicGate, holds an actual object whose type is that of one of the customized subclasses in the LogicGate class hierarchy. When the abstract displayGate message is passed to the object tempGate, it uses its own customized individual displayGate method to do the actual drawing. Thus if the value of tempGate is of type LogicAnd, an AND gate gets drawn, whereas if the type of tempGate is LogicXor, an Xor gate gets drawn, and so forth. This is very powerful, and would require the use of awkward and time consuming techniques to express in non-object oriented language.

The paint method implementation was changed at a later point. It simply passes the responsibility for its action to the update method. The paint method calls the update method when it wants to update those portions of the screen that have changed and need to be redrawn. The update method is responsible for the actual drawing of the gates in the circuit to the screen. The gates are drawn into an offscreen image by sending the displayGate message to each gate on the list and then transferring the completed image of all drawn gates from the offscreen image to the actual screen visible to the user. This way it eliminates the flashing and flickering effect that would have occurred had it drawn the entire list of gates in background-to-foreground order each time the mouse position changes during mouse dragging. The update method also handles conditions where two gates are being connected. A connection line (wire) is drawn as needed at the correct locale between the two gates.

A logic circuit consists of a combination of logic gates, and in turn, a logic gate is an object that is instantiated by the user during the design of a circuit. The way we implemented logic objects is by creating a super (parent) class, which is a stand-alone entity; that is, it is not derived from any existing Java object. The super class, called LogicGate, declares several methods that produce necessary information at different stages of the life cycle of a logic gate. Some of these methods are defined within the super class itself whereas several others are declared as abstract, which is the equivalent of virtual functions in C++. From the super class, many objects are derived as children and inherit the attributes of the parent object. These objects represent the basic logic gates, the connector object, and the input and output objects. Each child derived from the parent class declares additional, exclusive methods.

Careful consideration was given as to what attributes a logic gate must have (See Figure 5.2). The attributes of a logic gate consist of:

1) The gate index that specifies its position in the circuit,
2) A point location represented by x and y coordinates that specifies the gate's location on the canvas,
3) The states of input pins and whether they are connected and to what object,
4) A connectionVector that stores all the wire connections for each gate (all connection information for each gate is stored in this dynamic array), and
5) Boolean variables, which are declared to hold the state values of input pins, and the output pin.

Methods declared for a gate object return important information about the location of the object on canvas and the total environment state at any moment in time. Each gate object

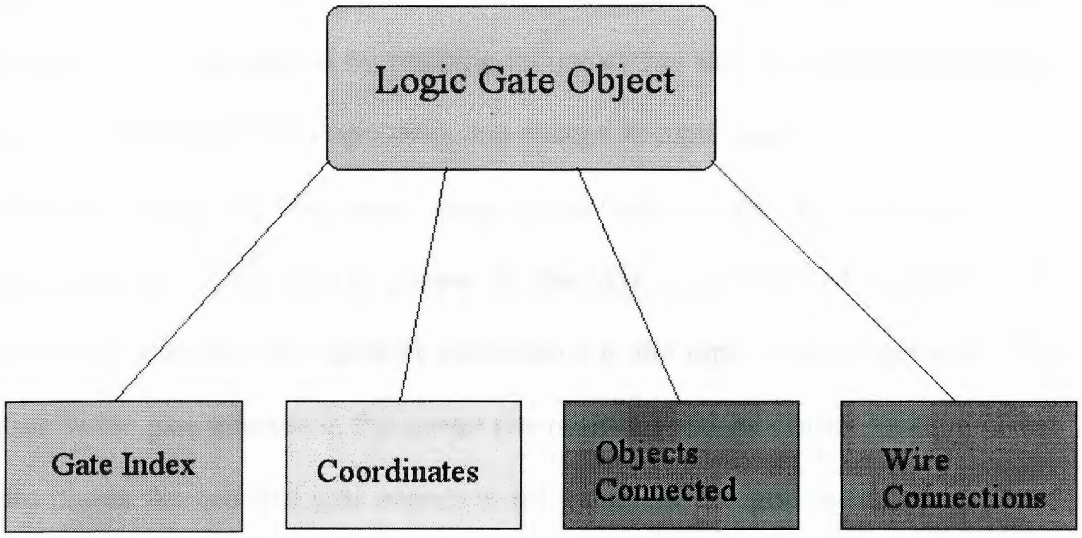


Figure 5.2 Main Attributes of Logic Gate Object

is capable of returning sufficient information to the requester. Information such as the state of the input pins and the output pin. A gate object can also identify itself by returning its type as a reference, its position in the circuit, and its canvas location. A gate can tell what part of it is clicked by trapping mouse events and processing them, and a gate can draw itself, calculate its operation, and change its input states.

The constructor for the super class, LogicGate, is the one responsible for instantiating all logic objects in the circuit. As theCanvas requests that a specific gate object be created, it passes the necessary information to the super class of that gate. This information is the gate number in the circuit (its position) and its canvas location. Then LogicGate places the selected gate object in the center of the grid specified by these coordinates. LogicGate defines the method whichGate that returns the gate being clicked once a mouse click position is given. After a gate is identified, LogicGate can return the part of the gate selected, which can be either inputs or the output pin. Fine-tuning is further done in the event input is selected in order for the canvas to determine which input pin is clicked. LogicGate can also return a Boolean value of True or False if an output of a gate is clicked. Further more, LogicGate can return to theCanvas the position of the gate selected.

Logic objects derived from the super class, LogicGate, inherit all the attributes and methods of the parent. In addition, each "child" subclass further defines the functionality specific to it such as the logic operation of each gate. All abstract methods that are declared in the parent class must be defined in any inheriting child. However a subclass can of course declare methods that do not exist in the super class. Each subclass object can display itself on screen through a request from theCanvas. Obviously each paint method is different among the logic gates. The paint method draws the object it belongs to and then draws all wire connections to this object. Wire connections for each gate are stored separately in each gate's connectionVector. A logic gate can change its input state with the toggleInput method. This method allows the user to change the input states between 'high' and 'low.' A gate generates its correct output by simply following two rules at all times.

One rule is to always keep the output value consistent with the inputs received on the input side. Performing the appropriate logic operation of the gate on the inputs received validates the consistency. This condition must be true at any time. As one or more input pins changes states, the output consequently must be rechecked and recalculated. Correction is done by changing the value of the output pin in accordance with the result of the logic operation. This leads to the second rule -- if the output changes states then all gates that are connected to it must be reevaluated. The output value of a gate must at all times be consistent with the value of all input pins connected to it.

5.3 System Testing

The final stage of building this prototype's model was to test its performance to assure its correct output in accordance with the specifications theoretically generated. We tested the simulator program to insure its accurateness by using it for designing and simulating many circuits. Initially, each gate object was tested separately for accurate output. Next, simple circuits were constructed and tested, and their results were verified. Finally larger circuits, such as a 4-bit full adder, 1-bit ALU (arithmetic logic unit), and 1-bit left/right shift register, were built. All circuits generated accurate results and performed satisfactorily regardless of their size or complexity. This is important to predict the stability of the system under various circuit designs and sizes.

The next step was to transform the application to an applet format in order to incorporate it into the Web. Several changes had to be made for this transformation. First, an HTML page with an applet tag was constructed. Second, the main() method had to be eliminated since the Web browser does the initiation for the applet automatically. Next, the application title was also removed as it interferes with the browser's naming mechanism. Applets do not have title bars, but one can title the Web page itself using the <title> HTML tag. Then the simulator class had to be modified so that it is derived from the Applet class instead of from the Frame class. Next, the simulator class constructor had to be replaced with a method called init. When the browser creates an object of the applet class, it calls it's init() method. If the application's frame implicitly uses a border layout (for placing components), the layout manager for the applet must be set in the init method. Additionally, all menus must be replaced with buttons since applets cannot have menu bars.

Finally, the instructional materials of the course were transformed into HTML so that they can be viewed on the Web. This way, a student can read the course text and use the simulator in parallel, which stresses conceptual understanding. A manual for the simulator is added on the site through a hyper link. Afterwards, an extensive test is performed on the whole Web site for the following:

- To ensure the readability of the course materials,
- To validate the correct links among the site's many pages,
- To verify the ability of the Web browser to initiate the simulator applet,

- To inspect for accurate functionality of the simulator, and
- To confirm the ability of the Web browser to download the application version of the simulator.

5.4 Design Challenges

We came across many challenges while building the prototype's model. In the beginning, it was difficult to determine what object should have what functionality. Dividing responsibilities among the different classes took some time to establish. Initially, the gate objects super class was given numerous responsibilities that later proved to be inadequate, so LogicGate had to be redefined and carefully redesigned. Thereafter the objective became apparent -- a gate must handle only gate operations. The isolation or abstraction technique proved very effective in designing the simulator as totally object-oriented.

Another challenge we faced in the implementation of the program to learn Java and new Java oriented techniques. Luckily Java does not use explicit pointers, so programming was somewhat simplified as compared to C++ for example. Yet we still had to gain sufficient knowledge of the internals of the Java language and the different libraries it offers in order to use them effectively.

One of the principal challenges that we encountered was how to implement the methods declared for each object. On several occasions we had to re-implement methods that we would later discover did not cover all the necessary bases. We had difficulty deciding how to pass information around from one method to another or between different objects. On occasion we had to decide whether a variable should be declared "global' or should be created as a temporary variable inside a method that uses it. We were able to carefully distinguish when to use each case and why by isolating a variable to a method if it is used in that method alone. If several methods may need such a temporary variable, each method should create its own temporary variable. The variable's scope stays within the method definition and disappears as the method exits. On the other hand if many methods need to have access to the same variable then it is best to declare it globally. Actually the term global does not exist in Java, as the language is completely object-oriented. Instead a global variable is declared at the object level and it can be private, protected, or public.

The single most significant programming challenge that we faced was debugging the application. We spent a great deal of time debugging using the time-honored method of inserting print statements into the code. However, we can tell you that debugging with print statements is not one of life's more joyful experiences. We constantly had to add and remove the print statements, then recompiling the program. Using a debugger is better because a debugger runs a program in full motion until it reaches a breakpoint, and then one can look at everything that is of interest. One can purchase excellent debuggers on most platforms, but if you are on a budget or working on an unusual platform, you may still need to resort to the print statements. The Java development kit includes an extremely primitive command-line debugger, and its user interface is so minimal that you will not find it useful. In my opinion, this debugger is really more a proof of concept than a useful tool.

Chapter 6

Conclusion

Computer simulation is the discipline of designing a model for an actual or theoretical physical system, executing the model on a digital computer, and analyzing the executed output. Simulation embodies the principle of "learning by doing." The use of simulation is an activity that is as natural as a child who role-plays. Children understand the world around them by simulating (with the use of toys and figures) most of their interactions with other people, animals, and objects. To understand reality and all of its complexity, we must build artificial objects and dynamically act out roles with them. Computer simulation is the electronic equivalent of this type of role-playing and it serves to drive synthetic environments and virtual worlds.

Computer simulations are currently used in a wide range of applications in the physical sciences, as well as in the social sciences and economics. For example, much of what is known about the likely behavior of nuclear reactors during accidents is derived from computer simulation models. Needless to say, testing actual reactors or even scaled down models under emergency conditions would involve excessive risks. Thus, computer simulation is of critical importance. Computer simulations are also used in meteorology to forecast the weather. Chemists use computer simulations to explain and view chemical reactions occurring at the molecular level.

Technologies such as simulation will dominate the entertainment and science forefronts well into the next century. While what we may do today may be primitive by standards set in science fiction movies such as “Star Trek” (The Holodeck) and “Lawnmower Man,” the present computer simulation discipline will lead the way to these eventual goals. The key word is "digital," as pointed out by many such as Nicholas Negroponte at the MIT Media Lab in his recent text "*Being Digital*". Scientists want to create digital replicas of everything we see around us. Digital objects may soon be located anywhere on the Internet, and users will be able to use some help tools to locate the building block objects for a digital world. Some of this type of work is being done in Distributed Interactive Simulation, a major project pioneered by the Department of Defense. The implications of these distributed interactive simulations are profound since this idea has enormous potential with the Department of Defense as well as in the industrial and entertainment fields.

The primary purpose of writing simulations in the Java programming language is to allow "live diagrams" to be incorporated into documents. With Java, people can experiment with a working simulation model by clicking on a Web link. This is different from simulations written in a traditional simulation language, C++ for example, where exporting simulation code requires recompilation and installation on each different platform. Java incorporates the language features necessary for simulation, notably objects and threads. Current Java implementations compile down to an intermediate byte code which is then interpreted. Thus, the main disadvantage of using Java, as compared to C++, is longer simulation run times.

The focus of this thesis is on the application of computer simulation for learning purposes. Educational computer simulation programs are available for many subjects such as biology, chemistry, medicine, physics, and economics. Many experiments with high educational value cannot be executed in the classroom. Some systems are too large (planetary systems) or too small (molecular systems) to be studied. Others may involve processes, which are too slow (the growth of biological systems) or too fast (chemical reactions and electronic circuits). In addition, there are experiments that are never executed because they are too dangerous (medical experiments), or unethical, or too costly. Furthermore, some experiments deal with systems that are much too complicated to be fully understood by students. Problems with these complex experiments can be overcome by using educational computer simulation programs.

The objective of this project is to construct a component based logic simulation environment using the Java programming language. This simulation environment will provide electronic designers and students a more scalable, more reliable, and more efficient way of performing verification for digital logic circuits. LogicCity is designed and developed as a computer simulation program that àids students in learning the subject material by enabling them to enter a circuit schematic and run a simulation of its behavior.

As for the computer science field, some computer organization and architecture courses teach digital logic design as an introduction. Some material can be difficult to grasp or visualize, thus necessitating some other form of educational aid to help drive home those tough parts of the subject. Digital logic simulation is an essential tool in helping students emphasize conceptual understanding, while complementing class discussions. LogicCity has a very user friendly, interactive graphical interface that allows students to validate the accuracy of their designs and helps them to do their homework.

6.1 Future Work

LogicCity has many features and benefits as an educational tool but like all programs further improvements are necessary. The architectural structure of the system operates at the gate level. Many arbitrary circuits can be designed and built using the simulator however efficiency might be compromised for very large circuits if they are to be built at the gate level. One can go one abstraction level up by wrapping all the smaller components (gates) to represent the basic building block for a more complex system, thus utilizing the "chip" formats in building more complex circuit modules. Modularity allows for the construction of virtually any electronic circuit with speed and ease.

The graphical interface for the simulator, though intuitive and clear, can still be additionally improved by employing the standard Windows format. Additionally, more sophisticated options can be added in the form of pull down menus that add various useful functionalities.

6.2 System Evaluation

Although the perfect way to evaluate this software would be to have it utilized by actual students for a semester, this was not feasible due to time constraints. However, we did have several graduate and undergraduate students evaluate LogicCity for performance. Some of the key points tested were accuracy of results generated, clarity, intuitiveness, ease of use, and overall presentation. Feedback was very encouraging and positive. The initial success of the simulation program prompted the instructor of an undergraduate computer science course to use LogicCity by incorporating it into the course material.

References

1. Anderson David, Roberts, Nancy, Real Ralph, Garet, Michael, and Shaffer, William. *Introduction to Computer Simulation: A System Dynamics Modeling Approach.* Productivity Press. 1994.
2. Bennett, A. Wayne. *Introduction to Computer Simulation.* pp 429-442. West Publishing Company. 1974.
3. Bennett, Frederick, Ph.D., *Computers as Tutors: Solving the Crisis in Education.* pp 10-34. http://www.cris.com/~Faben1. 1996.
4. Berkum, J.J.A. van, Hijne, H., de Jong, T., van Joolingen, W.R., Njoo, M., "Learning processes, learner attributes and simulations". *Education & Computing*, Volume 6, pp 231-239, 1991.
5. Berkum, J.J.A. van, & de Jong, T., "Instructional environments for simulations". *Education & Computing*, Volume 6, pp 305-358, 1991.
6. Cornell, Gary and Hortsmann, Cay S., *Core Java*, Second Edition. The Sunsoft Press, A Prentice Hall Title, 1997.
7. Ellington, H.I., Addinall, E., Percival, R. *Games and simulations in science education*. London: Kogan Page. 1981.

8. Hays, R.T., and Singer, M.J., *Simulation Fidelity in Training System Design: Bridging the Gap Between Reality and Training*. Springer-Verlag Publishing. 1989.

9. Hoog, R. de, Jong, T. de & Vries, F. de, "Interfaces for instructional use of simulations". *Education & Computing*, Volume 6, pp 359-385. 1991.

10. Joolingen, W.R. van & de Jong, T., "Characteristics of simulations for instructional settings". *Education & Computing*, Volume 6, pp 241-262. 1991.

11. Kennedy, David M., "Interactive Multimedia: Educational Desert or Educational Oasis". Paper, *Multimedia Education Unit*. The University of Melbourne, Australia.

12. McTear, Michael F., *Understanding Cognitive Science*. ", John Wiley & Sons, Inc., Publishers. 1988.

13. Milson, Marliese. "Educational Technology: Hype or Help in Reforming Education". EDUC 420, *The Professional Teaching and American Education*. Mary Washington College.

14. Min, F.B.M., "Parallel Instruction, a Theory for Educational Computer Simulation". *Interactive Learning International*, Volume 8, No. 3, 177-183. 1992.

15. Perkins, David N., Schwartz, Judah L., West, Mary M., and Wiske, Marth S. *Software Goes to School*. pp 106. Oxford University Publishing, 1995.

16. Reigeluth, Charles M., *Instructional-Design Theories and Models: An Overview of their Current Status*. Lawrence Erlbaum associates, Inc. Publishers. 1983.

17. Reigeluth, C.M. & Schwartz, E., "An instructional theory for the design of computer-based simulations". *Journal of computer-based instruction*, 16(1), 1-10. 1989.

18. Schaick Zillesen, P.G. van & Min, F.B.M, "MacTHESIS: a design system for educational computer simulation programs". *Wheels for the mind of Europe*, 2, 23-33. 1987.

19. United States. *U.S. Department of Education.* "Getting America's Students Ready for the 21st Century". Pp. 5-20. U.S. Government Printing Office, 1996.

20. United States. *U.S. Department of Education.* Office of Educational Technology. "Making It Happen". Report of the Secretary's Conference on Educational Technology. Pp. 16-24. U.S. Government Printing Office, 1995.

21. United States. *U.S. Department of Education.* Office of Educational Research and Improvement. "Using Technology to Support Education". Pp. 1-28. U.S. Government Printing Office, 1993.

22. URL for *Dizzy*: By Jim Munki, 1990. ftp://ftp.symantec.com/public/english_us_canada/products/c++/mac/samples/source_code/dizzy.sit.hqx

23. URL for *Logg-O*: http://www.pws.com/aeonline/course/7/2/index.html, Lab 7.2: "Bill's Gates". Rick Decker and Stuart Hirshfield, PWS Publishing Company, 1998.

Appendix A:

Program Listing

```
//*****************************************************************************
//** Application Version.
//** 1998
//*****************************************************************************

import java.awt.*;        //Java's class library for basic GUI programming.
import java.util.*;       // utility library.
import java.awt.event.*;  // library for handling mouse events
import java.awt.Toolkit.*; // to get system properties (i.e. screen size).
import java.io.*;         // for file saving/opening functionality.
import java.lang.*;       // for math functions.


abstract class LogicGate
//*****************************************************************************
//** This is the super class for the logic gates used. It provides
//** functionality that is common to all gates derived. Nine subclasses
//** (gates + others) will be derived with more specific functionality to each gate.
//** A class is defined as abstract if at least one of its methods is abstract.
//** All abstract methods are to be implemented by any derived subclass.
//*****************************************************************************
{
  protected Vector connectionVector = new Vector();  // holds connection data.
  protected LogicGate topPinConnection = null;  // gate connected to this's top input pin.
  protected LogicGate bottomPinConnection = null;  // gate connected to this's top input pin.
  protected int gateIndex = 0;        //position of gate in circuit.
  protected final int gridWidth = 45;     // width of grid that holds the gate.
  protected final int gridHeight = 22;    // height of grid that holds the gate.
  protected int xCoordinate = 0;        // x coords of grid.
  protected int yCoordinate = 0;        // y coords of grid.
  protected final int nietherPin = 0;       // if no input pin was clicked.
  protected final int topPin = 1;          // top pin of a gate.
  protected final int bottomPin = 2;        // bottom pin of a gate.
  protected final int outputPin = 3;        // output pin of a gate.
  protected boolean isNotGate = false;    //a flag to determine when dealing with a NOT gate.
  protected boolean isConnectorObject = false;  //a flag to determine when dealing with a Connector object.
  protected boolean isInputObject = false;  //a flag to determine when dealing with an Input object.
  protected boolean isOutputObject = false;  //a flag to determine when dealing with an Output object.

// Display the logic gate.
 abstract public void displayGate(Graphics g);

// Execute the gate's operation.
 abstract public void calculateGateOperation();

// Toggle the gate input state between high and low.
 abstract public void toggleInput(int whichPin);

// Return the state of the gate's input.
 abstract public boolean getInputState(int whichPin);
```

```
// Return the state of the gate's output.
  abstract public boolean getOutputState();

// Show the gate input and output states at simulation time.
  abstract public void displayStates(Graphics g);

  public LogicGate(int x, int y, int gatePosition)
//** The class constructor positions a new gate into the center of the selected grid.
  {
    gateIndex = gatePosition;  // set gate position into the circuit.
    // x coords for this gate.
    this.xCoordinate = ((x / gridWidth) * gridWidth) + (gridWidth / 2);
    // y coords for this gate.
    this.yCoordinate = ((y / gridHeight) * gridHeight) + (gridHeight / 2);
  }// end constructor

  public LogicGate whichGate(Point point)
//** Given a mouse click position, this method returns the gate being
//** clicked (if valid), null otherwise.
  {
    Point p = returnGatePosition(xCoordinate,yCoordinate);

    if((point.x > p.x) && (point.x < p.x + gridWidth)
      && (point.y > p.y) && (point.y < p.y + gridHeight))
    {
      return this;
    }
    else
    {
      return null;
    }
  }// end whichGate

  public int whichGatePart(Point point)
//** Returns the part of the gate being clicked. Once the gate is determined,
//** fine-tuning is done to find out what part of that gate was clicked.
//** Options are: either one of the two inputs or the output.
  {
    int whichPin = whichInput(point);        // get the pin selected.
    boolean isItOutput = isOutput(point);    // check the output also.

    if(whichPin > 0)
    {
      return whichPin;
    }
    else if(isItOutput)      // output pin was clicked.
    {
      return outputPin;
    }
    else
    {
      return nietherPin;      // nothing valid was clicked.
    }
  }// end whichGatePart

  public int whichInput(Point point)
//** Returns which one of two inputs to a gate has been clicked. It is
//** a helper function to whichGatePart.
  {
    Point position = returnGatePosition(xCoordinate, yCoordinate);
    // is the mouse at the input area?
    if((point.x > position.x) && (point.x < position.x + 9)
      && (point.y > position.y) && (point.y < position.y + gridHeight))
    {
      if(point.y < position.y + 12)     // is it the top pin?
      {
        return topPin;
      }
      else
      {
        return bottomPin;    // else it must be the bottom pin.
      }
    }
    else
    {
      return nietherPin;  // or it could be neither.
    }
  }// end whichInput

   public boolean isOutput(Point point)
//** Returns true if the gate's output has been clicked. This is another helper
//** function to whichGatePart.
  {
    Point position = returnGatePosition(xCoordinate, yCoordinate);

    if((point.x > position.x + 35) && (point.x < position.x + gridWidth)
     && (point.y > position.y + 5) && (point.y < position.y + gridHeight - 5))
    {
      return true;
    }
    else
    {
      return false;     // not at the output pin.
    }
  }// end isOutput

  public Point returnGatePosition(int x, int y)
//** Returns the canvas position of the gate being clicked. The canvas is
//** divided into square grids, once the grid coords are known, the gate
//** position is also identified inside the grid.
  {
    Point point = new Point(((x * gridWidth) / gridWidth) - (gridWidth / 2),
    ((y * gridHeight) / gridHeight) - (gridHeight / 2));

    return point;
  }
}// end LogicGate class

//** LogicAnd class has a detailed documentation. The other eight classes
//** are derived from the same super class have pretty similar functionality
//** and therefore documentation.

class LogicAnd extends LogicGate
//*****************************************************************************
//** This class is derived from its super class LogicGate and implements the
//** abstract methods declared. This class performs the operation of logic AND.
//*****************************************************************************
{
   boolean topInputPin = false;
   boolean bottomInputPin = false;
   boolean OutputPin = false;
```

```
 // The constructor positions a new gate in one of the grids selected.
 public LogicAnd(int x, int y, int gatePosition)
 {
   super(x,y,gatePosition);
   calculateGateOperation();
 }// end constructor

 public void displayGate(Graphics g)
//** Displays the AND gate, along with any wire connections.
 {
   // get location of gate.
   Point p = returnGatePosition(xCoordinate,yCoordinate);

   g.drawLine(p.x + 9,p.y + 22,p.x + 24,p.y + 22); // bottom edge
   g.drawLine(p.x + 9,p.y + 2,p.x + 9,p.y + 22);  // left edge
   g.drawLine(p.x + 9,p.y + 2,p.x + 24,p.y + 2);    // top edge
   g.drawLine(p.x + 4,p.y + 7,p.x + 8,p.y + 7);    // top input pin
   g.drawLine(p.x + 4,p.y + 17,p.x + 8,p.y + 17); // bottom input pin
   g.drawLine(p.x + 35,p.y + 12,p.x + 40,p.y + 12); // output pin
   g.drawArc(p.x + 14,p.y + 2,20,20,90,-180); // right side arc
   g.fillOval(p.x + 0, p.y + 5,4,4);       // bubble on top input pin
   g.fillOval(p.x + 0, p.y + 15,4,4);     // bubble on bottom input pin
   g.fillOval(p.x + 41, p.y + 10,4,4);   // bubble on output pin

   paintConnections(g,p);      // draw all wire connections.
 }// end displayGate

 public void paintConnections(Graphics g, Point p)
//** This method displays all wire connections of a gate.
//** Point p is the position of the gate.
 {
   LogicGate tempGate = null; // gate connected to "this".

   // for all connections to this gate.
   for(int counter = 0; counter < connectionVector.size(); ++counter)
   {
     tempGate = (LogicGate) connectionVector.elementAt(counter);
     Point position = returnGatePosition(tempGate.xCoordinate, tempGate.yCoordinate);

     // firstOccurance is the index of tempGate in the connection array.
     int firstOccurance = connectionVector.indexOf(tempGate);
     if(firstOccurance > -1)  // if the gate is connected.
     {
       //secondOccuance is the second time tempGate appears in the same connection array.
       int secondOccuance = connectionVector.indexOf(tempGate,firstOccurance+1);
if(secondOccuance > firstOccurance) // both inputs of tempGate are connected to "this".
       {
         // these Objects has one input, draw first connection to the top pin of tempGate.
         if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))
         {
           g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
         }
         else          // else it's a top pin
         {
           g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
         }
         //draw second connection to the bottom pin of tempGate.
         g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
       }
       else // else tempGate has only one input from "this".
       {
          // is this (this object) connected to tempGate's top input pin?
          if(tempGate.topPinConnection == this)
          {
            if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject)) // these Objects has
one input.
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
            }
            else          // else it's a top pin
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
            }
          }// end if(tempGate.topPinConnection == this)
          else    // or the bottom pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
          }
        }// end else
      }// end if(firstOccurance > -1)
      tempGate = null;   // reset to null.
    }// end for
  }// end paintConnections

  public void toggleInput(int whichPin)
//** Toggles the gate input state between high and low.
//** It allows the user to change the value of an input pin between 0 and 1.
  {
    if(whichPin == topPin)       // if top input pin being changed
    {
      topInputPin = !topInputPin;   // change state.
    }
    else                  // else bottom input pin being changed
    {
      bottomInputPin = !bottomInputPin;
    }
    calculateGateOperation(); // recompute the gate's logic operation.
  }// end toggleInput

  public void calculateGateOperation()
//** This method computes the logic AND operation for the gate.
//** The result of this calculation is broadcasted on the output pin.
  {
    if(OutputPin != (topInputPin & bottomInputPin))  // if output is incorrect.
    {
      // if output is wrong, change it.
      OutputPin = (topInputPin & bottomInputPin);

      for(int counter = 0; counter < connectionVector.size(); ++counter)
      {
        // for all connections, if the gate's input is connected to "this" gate's output.
        if(((LogicGate) connectionVector.elementAt(counter)).topPinConnection == this)
        {
          // and the input does not match the output coming?
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(topPin) != getOutputState())
          {
            // change the value of the top input pin.
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(topPin);
          }
        }
        else  // do the same for the bottom input pin
        {
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(bottomPin) != getOutputState())
        {
          ((LogicGate) connectionVector.elementAt(counter)).toggleInput(bottomPin);
        }
      }// end else
    }// end for
  }// end if(OutputPin != (topInputPin & bottomInputPin))
 }// end calculateGateOperation

 public boolean getInputState(int whichPin)
//** Returns the state of the gate's input. Given which pin was clicked, this
//** method returns the value of that input pin whether high or low.
//** Returns true if the pin is high , false otherwise.
 {
   if(whichPin == topPin)
   {
     return topInputPin;
   }
   else
   {
     return bottomInputPin;
   }
 }// end getInputState

 public boolean getOutputState()
//** Returns the state of the gate's output. this method returns the value
//** of the output pin whether high or low.
 {
   return OutputPin;
 }// end getOutputState

 public void displayStates(Graphics g)
//** Shows the gate input and output states at simulation time. If an input
//** to a gate is changed, this change is passed throughout the whole circuit.
//** High value is indicated with a red color, black for low value.
 {
   g.setFont(new Font("Helvetica", Font.BOLD,9));
   Point position = returnGatePosition(xCoordinate, yCoordinate);

   if(topInputPin)          // top pin is high
   {
     g.setColor(Color.red);
     g.drawString("1",position.x + 3 , position.y + 6);
   }
   else                     // top pin is low
   {
     g.setColor(Color.black);
     g.drawString("0",position.x + 3 , position.y + 6);
   }
   if(bottomInputPin)       // bottom pin is high
   {
     g.setColor(Color.red);
     g.drawString("1",position.x + 3, position.y + 16);
   }
   else                     // bottom pin is low
   {
     g.setColor(Color.black);
     g.drawString("0",position.x + 3, position.y + 16);
   }
   if(OutputPin)            // output pin is high
   {
     g.setColor(Color.red);
     g.drawString("1",position.x + 37, position.y + 11);
    }
    else              // output pin is low
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 37, position.y + 11);
    }
  }// end displayStates
}// end LogicAnd class

class LogicOr extends LogicGate
//***************************************************************************
//** This class is derived from its super class LogicGate and implements the
//** abstract methods declared. This class performs the operation of logic OR.
//**
//***************************************************************************
{
  boolean topInputPin = false;
  boolean bottomInputPin = false;
  boolean OutputPin = false;

// The constructor positions a new gate in one of the free grids selected.
  public LogicOr(int x, int y, int gatePosition)
  {
    super(x,y,gatePosition);
    calculateGateOperation();
  }// end constructor

  public boolean getInputState(int whichPin)
//** Returns the state of the gate's input. Given which pin was clicked, this
//** method returns the value of that input pin whether high or low.
//** Returns true of the pin is high , false otherwise.
  {
    if(whichPin == topPin)
    {
      return topInputPin;
    }
    else
    {
      return bottomInputPin;
    }
  }// end getInputState

  public boolean getOutputState()
//** Returns the state of the gate's output. this method returns the value
//** of the output pin whether high or low.
  {
    return OutputPin;
  }// end getOutputState

  public void displayGate(Graphics g)
//** Displays the OR gate, along with any wire connections.
  {
    // get location of gate.
    Point p = returnGatePosition(xCoordinate,yCoordinate);

    g.drawLine(p.x + 4, p.y + 7, p.x + 11,p.y + 7);  // top input pin
    g.drawLine(p.x + 4, p.y + 17, p.x + 11, p.y + 17); // bottom input pin
    g.drawLine(p.x + 34, p.y + 12, p.x + 40, p.y + 12);// output pin
    g.drawArc(p.x + 1,p.y + 2,12,20,90,-180);      // left side arc
    g.drawArc(p.x - 17,p.y + 2,50,20,90,-180);     // right side arc
    g.fillOval(p.x + 0, p.y + 5,4,4);  // bubble on top input pin
    g.fillOval(p.x + 0, p.y + 15,4,4);  // bubble on bottom input pin
    g.fillOval(p.x + 41, p.y + 10,4,4);  // bubble on output pin

    paintConnections(g,p);
  }// end displayGate

  public void paintConnections(Graphics g, Point p)
//** This method displays all wire connections of a gate.
  {
    LogicGate tempGate = null;  // gate connected to "this".

    // for all connections to this gate.
    for(int counter = 0; counter < connectionVector.size(); ++counter)
    {
      tempGate = (LogicGate) connectionVector.elementAt(counter);
      Point position = returnGatePosition(tempGate.xCoordinate, tempGate.yCoordinate);

      // firstOccurance is the index of tempGate in the connection array.
      int firstOccurance = connectionVector.indexOf(tempGate);
      if(firstOccurance > -1)
      {
        //secondOccuance is the second time tempGate appears in the same connection array.
        int secondOccuance = connectionVector.indexOf(tempGate,firstOccurance+1);
if(secondOccuance > firstOccurance) // both inputs of tempGate are connected to "this".
        {
          // these Objects has one input, draw first connection to the top pin of tempGate.
          if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
          }
          else          // else it's a top pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
          }
          //draw second connection to the bottom pin of tempGate.
          g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
        }
        else // else tempGate has only one input from "this".
        {
          // is this (this object) connected to tempGate's top input pin?
          if(tempGate.topPinConnection == this)
          {
            if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))  // these Objects has
one input.
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
            }
            else          // else it's a top pin
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
            }
          }// end if(tempGate.topPinConnection == this)
          else    //  or the bottom pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
          }
        }// end else
      }// end if(firstOccurance > -1)
      tempGate = null;    // reset to null.
    }// end for
  }// end paintConnections

  public void toggleInput(int whichPin)
```

```
//** Toggles the gate input state between high and low.
//** It allows the user to change the value of an
//** input pin between 0 and 1.
  {
    if(whichPin == topPin)
    {
      topInputPin = !topInputPin;
    }
    else
    {
      bottomInputPin = !bottomInputPin;
    }
    calculateGateOperation();
  }// end toggleInput

  public void calculateGateOperation()
//** This method computes the logic OR operation between the two gate's
//** input pins. The result of this calculation is broadcasted on the
//** output pin.
  {
    if(OutputPin != (topInputPin | bottomInputPin))
    {
      OutputPin = (topInputPin | bottomInputPin);
      for(int counter = 0; counter < connectionVector.size(); ++counter)
      {
        if(((LogicGate) connectionVector.elementAt(counter)).topPinConnection == this)
        {
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(topPin) != getOutputState())
            {
              ((LogicGate) connectionVector.elementAt(counter)).toggleInput(topPin);
            }
        }
        else
        {
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(bottomPin) != getOutputState())
          {
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(bottomPin);
          }
        }
      }// end for
    }// end if(OutputPin != (topInputPin | bottomInputPin))
  }// end calculateGateOperation

  public void displayStates(Graphics g)
//** Shows the gate input and output states at simulation time.
//** High values are indicated with a red color, black for a low values.
  {
    g.setFont(new Font("Helvetica", Font.BOLD,9));
    Point position = returnGatePosition(xCoordinate, yCoordinate);

    if(topInputPin)         // top pin is high
    {
      g.setColor(Color.red);
      g.drawString("1",position.x + 3 , position.y + 6);
    }
    else               // top pin is low
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 3 , position.y + 6);
    }
    if(bottomInputPin)      // bottom pin is high
    {
      g.setColor(Color.red);
      g.drawString("1",position.x + 3, position.y + 16);
    }
    else              // bottom pin is low
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 3, position.y + 16);
    }
    if(OutputPin)          // output pin is high
    {
      g.setColor(Color.red);
      g.drawString("1",position.x + 37, position.y + 11);
    }
    else              // output pin is low
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 37, position.y + 11);
    }
  }// end displayStates

}// end LogicOr class

class LogicNot extends LogicGate
//****************************************************************************
//** This class is derived from its super class LogicGate and implements the
//** abstract methods declared. This class performs the operation of logic NOT.
//**
//****************************************************************************
{
  boolean topInputPin = false;
  boolean bottomInputPin = false;
  boolean OutputPin = false;


// The constructor positions a new gate in one of the grids selected.
  public LogicNot(int x, int y, int gatePosition)
  {
    super(x,y,gatePosition);
    isNotGate = true;
    calculateGateOperation();
  }// end constructor

  public boolean getInputState(int whichPin)
//** Returns the state of the gate's input.
//** true if the pin is high , false otherwise.
  {
    return topInputPin;
  }// end getInputState

  public int whichInput(Point point)
//** Returns which one of two inputs to a gate has been clicked. It is modified
//** here since a not gate has only one input.
  {
    Point position = returnGatePosition(xCoordinate, yCoordinate);

    // is the mouse at the input area?
    if((point.x > position.x) && (point.x < position.x + 10)
      && (point.y > position.y - 4) && (point.y < position.y + 25))
    {
      return topPin;
    }
    else
    {
      return niether Pin;  // or it could be neither.
    }
  }// end whichInput

  public boolean getOutputState()
//** Returns the state of the gate's output. this method returns the value
//** of the output pin whether high or low.
  {
    return OutputPin;
  }// end getOutputState

  public void displayGate(Graphics g)
//** Displays the NOT gate, along with any wire connections.
  {
    // get location of gate.
    Point p = returnGatePosition(xCoordinate,yCoordinate);
    g.drawLine(p.x + 9,p.y + 22,p.x + 27,p.y + 12); // bottom edge
    g.drawLine(p.x + 9,p.y + 2,p.x + 9,p.y + 22);  // left edge
    g.drawLine(p.x + 9,p.y + 2,p.x + 27,p.y + 12);   // top edge
    g.drawLine(p.x + 4,p.y + 12,p.x + 9,p.y + 12);  // input pin
    g.drawLine(p.x + 34,p.y + 12,p.x + 40,p.y + 12); // output pin
    g.fillOval(p.x + 0, p.y + 10,4,4);      // bubble on input pin
    g.fillOval(p.x + 41, p.y + 10,4,4);   // bubble on output pin
    g.drawOval(p.x + 28,p.y + 9,6,6);      // negation bubble

    paintConnections(g,p);
  }// end displayGate

  public void paintConnections(Graphics g, Point p)
//** This method displays all wire connections of a gate.
  {
    LogicGate tempGate = null; // gate connected to "this".

    // for all connections to this gate.
    for(int counter = 0; counter < connectionVector.size(); ++counter)
    {
      tempGate = (LogicGate) connectionVector.elementAt(counter);
      Point position = returnGatePosition(tempGate.xCoordinate, tempGate.yCoordinate);

      // firstOccurance is the index of tempGate in the connection array.
      int firstOccurance = connectionVector.indexOf(tempGate);
      if(firstOccurance > -1)
      {
        //secondOccuance is the second time tempGate appears in the same connection array.
        int secondOccuance = connectionVector.indexOf(tempGate,firstOccurance+1);
if(secondOccuance > firstOccurance) // both inputs of tempGate are connected to "this".
        {
          // these Objects has one input, draw first connection to the top pin of tempGate.
          if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
          }
          else        // else it's a top pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
          }
          //draw second connection to the bottom pin of tempGate.
          g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
        }
        else // else tempGate has only one input from "this".
        {
```

```
        // is this (this object) connected to tempGate's top input pin?
        if(tempGate.topPinConnection == this)
        {
          if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject)) // these Objects has
one input.
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
          }
          else          // else it's a top pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
          }
        }// end if(tempGate.topPinConnection == this)
        else    // or the bottom pin
        {
          g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
        }
      }// end else
    }// end if(firstOccurance > -1)
    tempGate = null;   // reset to null.
  }// end for
}// end paintConnections

public void toggleInput(int whichPin)
//** Toggles the gate input state between high and low.
//** It allows the user to change the value of an
//** input pin between 0 and 1.
{
  topInputPin = !topInputPin;
  calculateGateOperation();
}// end toggleInput

public void calculateGateOperation()
//** This method computes the logic NOT operation on the gate's
//** input pin. The result of this calculation is broadcasted on the
//** output pin.
{
  if(OutputPin == topInputPin)
  {
    OutputPin = !topInputPin;
    for(int counter = 0; counter < connectionVector.size(); ++counter)
    {
      if(((LogicGate) connectionVector.elementAt(counter)).topPinConnection == this)
      {
        if(((LogicGate) connectionVector.elementAt(counter)).getInputState(topPin) != getOutputState())
        {
          ((LogicGate) connectionVector.elementAt(counter)).toggleInput(topPin);
        }
      }
      else
      {
        if(((LogicGate) connectionVector.elementAt(counter)).getInputState(bottomPin) != getOutputState())
        {
          ((LogicGate) connectionVector.elementAt(counter)).toggleInput(bottomPin);
        }
      }
    }// end for
  } // end if(OutputPin == topInputPin)
}// end calculateGateOperation

public void displayStates(Graphics g)
//** Shows the gate input and output states at simulation time. If the input
```

```
//** to the gate is changed, this change is passed throughout the whole circuit.
//** High value is indicated with a red color, black for low value.
  {
    g.setFont(new Font("Helvetica", Font.BOLD,9));
    Point position = returnGatePosition(xCoordinate, yCoordinate);

    if(topInputPin)
    {
      g.setColor(Color.red);
      g.drawString("1",position.x + 3 , position.y + 11);
    }
    else
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 3 , position.y + 11);
    }
    if(OutputPin)
    {
      g.setColor(Color.red);
      g.drawString("1",position.x + 37, position.y + 11);
    }
    else
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 37, position.y + 11);
    }
  }// end displayStates

}// end LogicNot class

class LogicXor extends LogicGate
//*****************************************************************************
//** This class is derived from its super class LogicGate and implements the
//** abstract methods declared. This class performs the operation of logic XOR.
//*****************************************************************************
{
   boolean topInputPin = false;
   boolean bottomInputPin = false;
   boolean OutputPin = false;

// The constructor positions a new gate in one of the grids selected.
  public LogicXor(int x, int y, int gatePosition)
  {
    super(x,y,gatePosition);
    calculateGateOperation();
  }// end constructor

  public boolean getInputState(int whichPin)
//** Returns the state of the gate's input. Given which pin was clicked, this
//** method returns the value of that input pin whether high or low.
//** Returns true of the pin is high , false otherwise.
  {
    if(whichPin == topPin)
    {
      return topInputPin;
    }
    else
    {
      return bottomInputPin;
    }
  }// end getInputState
  public boolean getOutputState()
//** Returns the state of the gate's output. this method returns the value
//** of the output pin whether high or low.
  {
    return OutputPin;
  }// end getOutputState

  public void displayGate(Graphics g)
//** Displays the XOR gate, along with any wire connections.
  {
    // get location of gate.
    Point p = returnGatePosition(xCoordinate,yCoordinate);
    g.drawLine(p.x + 4, p.y + 7, p.x + 9,p.y + 7);  // top input pin
    g.drawLine(p.x + 4, p.y + 17, p.x + 9, p.y + 17); // bottom input pin
    g.drawLine(p.x + 36, p.y + 12, p.x + 40, p.y + 12);// output pin
    g.drawArc(p.x - 1,p.y + 2,12,20,90,-180);      // left side arc
    g.drawArc(p.x - 15,p.y + 2,50,20,90,-180);     // right side (big) arc
    g.drawArc(p.x + 4,p.y + 2,12,20,90,-180);     // middle arc
    g.fillOval(p.x + 0, p.y + 5,4,4);  // bubble on top input pin
    g.fillOval(p.x + 0, p.y + 15,4,4); // bubble on bottom input pin
    g.fillOval(p.x + 41, p.y + 10,4,4); // bubble on output pin

    paintConnections(g,p);
  }// end displayGate

  public void paintConnections(Graphics g, Point p)
//** This method displays all wire connections of a gate.
  {
    LogicGate tempGate = null; // gate connected to "this".

    // for all connections to this gate.
    for(int counter = 0; counter < connectionVector.size(); ++counter)
    {
      tempGate = (LogicGate) connectionVector.elementAt(counter);
      Point position = returnGatePosition(tempGate.xCoordinate, tempGate.yCoordinate);

      // firstOccurance is the index of tempGate in the connection array.
      int firstOccurance = connectionVector.indexOf(tempGate);
      if(firstOccurance > -1)
      {
        //secondOccuance is the second time tempGate appears in the same connection array.
        int secondOccuance = connectionVector.indexOf(tempGate,firstOccurance+1);
if(secondOccuance > firstOccurance) // both inputs of tempGate are connected to "this".
        {
          // these Objects has one input, draw first connection to the top pin of tempGate.
          if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
          }
          else          // else it's a top pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
          }
          //draw second connection to the bottom pin of tempGate.
          g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
        }
        else // else tempGate has only one input from "this".
        {
          // is this (this object) connected to tempGate's top input pin?
          if(tempGate.topPinConnection == this)
          {
```

```
          if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))  // these Objects has
one input.
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
          }
          else        // else it's a top pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
          }
        }// end if(tempGate.topPinConnection == this)
        else    //  or the bottom pin
        {
          g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
        }
       }// end else
     }// end if(firstOccurance > -1)
     tempGate = null;    // reset to null.
   }// end for
  }// end paintConnections

  public void toggleInput(int whichPin)
//** Toggles the gate input state between high and low.
//** It allows the user to change the value of an
//** input pin between 0 and 1.
  {
    if(whichPin == topPin)
    {
      topInputPin = !topInputPin;
    }
    else
    {
      bottomInputPin = !bottomInputPin;
    }
    calculateGateOperation();
  }// end toggleInput

  public void calculateGateOperation()
//** This method computes the logic XOR operation between the two gate's
//** input pins. The result of this calculation is broadcasted on the
//** output pin.
  {
    if(OutputPin != (topInputPin ^ bottomInputPin))
    {
      OutputPin = (topInputPin ^ bottomInputPin);
      for(int counter = 0; counter < connectionVector.size(); ++counter)
      {
        if(((LogicGate) connectionVector.elementAt(counter)).topPinConnection == this)
        {
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(topPin) != getOutputState())
            {
              ((LogicGate) connectionVector.elementAt(counter)).toggleInput(topPin);
            }
        }
        else
        {
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(bottomPin) != getOutputState())
          {
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(bottomPin);
          }
        }
      }// end for
    }
   }// end calculateGateOperation

   public void displayStates(Graphics g)
//** Shows the gate input and output states at simulation time. If an input
//** to a gate is changed, this change is passed throughout the whole circuit.
//** High value is indicated with a red color, black for low value.
   {
     g.setFont(new Font("Helvetica", Font.BOLD,9));
     Point position = returnGatePosition(xCoordinate, yCoordinate);

     if(topInputPin)          // top pin is high
     {
       g.setColor(Color.red);
       g.drawString("1",position.x + 3 , position.y + 6);
     }
     else              // top pin is low
     {
       g.setColor(Color.black);
       g.drawString("0",position.x + 3 , position.y + 6);
     }
     if(bottomInputPin)       // bottom pin is high
     {
       g.setColor(Color.red);
       g.drawString("1",position.x + 3, position.y + 16);
     }
     else              // bottom pin is low
     {
       g.setColor(Color.black);
       g.drawString("0",position.x + 3, position.y + 16);
     }
     if(OutputPin)            // output pin is high
     {
       g.setColor(Color.red);
       g.drawString("1",position.x + 37, position.y + 11);
     }
     else              // output pin is low
     {
       g.setColor(Color.black);
       g.drawString("0",position.x + 37, position.y + 11);
     }
   }// end displayStates
}// end LogicXor class

class LogicNand extends LogicGate
//*****************************************************************************
//** This class is derived from its super class LogicGate and implements the
//** abstract methods declared. This class performs the operation of logic NAND.
//**
//*****************************************************************************
{
    boolean topInputPin = false;
    boolean bottomInputPin = false;
    boolean OutputPin = false;

// The constructor positions a new gate in one of the grids selected.
   public LogicNand(int x, int y, int gatePosition)
   {
     super(x,y,gatePosition);
     calculateGateOperation();
   }// end constructor

   public boolean getInputState(int whichPin)
```

```
//** Returns the state of the gate's input. Given which pin was clicked, this
//** method returns the value of that input pin whether high or low.
//** Returns true of the pin is high , false otherwise.
  {
    if(whichPin == topPin)
    {
      return topInputPin;
    }
    else
    {
      return bottomInputPin;
    }
  }// end getInputState

  public boolean getOutputState()
//** Returns the state of the gate's output. this method returns the value
//** of the output pin whether high or low.
  {
    return OutputPin;
  }// end getOutputState

  public void displayGate(Graphics g)
//** Displays the NAND gate, along with any wire connections.
  {
    // get location of gate.
    Point p = returnGatePosition(xCoordinate,yCoordinate);
    g.drawLine(p.x + 9,p.y + 22,p.x + 19,p.y + 22); // bottom edge
    g.drawLine(p.x + 9,p.y + 2,p.x + 9,p.y + 22);  // left edge
    g.drawLine(p.x + 9,p.y + 2,p.x + 19,p.y + 2);  // top edge
    g.drawLine(p.x + 4,p.y + 7,p.x + 8,p.y + 7);   // top input pin
    g.drawLine(p.x + 4,p.y + 17,p.x + 8,p.y + 17); // bottom input pin
    g.drawLine(p.x + 37,p.y + 12,p.x + 40,p.y + 12); // output pin
    g.drawArc(p.x + 9,p.y + 2,21,20,90,-180); // right side arc
    g.fillOval(p.x + 0, p.y + 5,4,4);      // bubble on top input pin
    g.fillOval(p.x + 0, p.y + 15,4,4);     // bubble on bottom input pin
    g.fillOval(p.x + 41, p.y + 10,4,4);    // bubble on output pin
    g.drawOval(p.x + 31, p.y + 9,6,6);     // negation bubble

    paintConnections(g,p);
  }// end displayGate

  public void paintConnections(Graphics g, Point p)
//** This method displays all wire connections of a gate.
  {
    LogicGate tempGate = null; // gate connected to "this".

    // for all connections to this gate.
    for(int counter = 0; counter < connectionVector.size(); ++counter)
    {
      tempGate = (LogicGate) connectionVector.elementAt(counter);
      Point position = returnGatePosition(tempGate.xCoordinate, tempGate.yCoordinate);

      // firstOccurance is the index of tempGate in the connection array.
      int firstOccurance = connectionVector.indexOf(tempGate);
      if(firstOccurance > -1)
      {
        //secondOccuance is the second time tempGate appears in the same connection array.
        int secondOccuance = connectionVector.indexOf(tempGate,firstOccurance+1);
if(secondOccuance > firstOccurance) // both inputs of tempGate are connected to "this".
        {
          // these Objects has one input, draw first connection to the top pin of tempGate.
          if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))
```

```
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
            }
            else          // else it's a top pin
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
            }
            //draw second connection to the bottom pin of tempGate.
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
          }
          else // else tempGate has only one input from "this".
          {
            // is this (this object) connected to tempGate's top input pin?
            if(tempGate.topPinConnection == this)
            {
              if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))  // these Objects has
one input.
              {
                g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
              }
              else          // else it's a top pin
              {
                g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
              }
            }// end if(tempGate.topPinConnection == this)
            else    //  or the bottom pin
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
            }
          }// end else
        }// end if(firstOccurance > -1)
        tempGate = null;    // reset to null.
      }// end for
    }// end paintConnections

    public void toggleInput(int whichPin)
//** Toggles the gate input state between high and low.
//** It allows the user to change the value of an
//** input pin between 0 and 1.
    {
      if(whichPin == topPin)
      {
        topInputPin = !topInputPin;
      }
      else
      {
        bottomInputPin = !bottomInputPin;
      }
      calculateGateOperation();
    }// end toggleInput

    public void calculateGateOperation()
//** This method computes the logic NAND operation between the two gate's
//** input pins. The result of this calculation is broadcasted on the
//** output pin.
    {
      if(OutputPin != !(topInputPin & bottomInputPin))
      {
        OutputPin = !(topInputPin & bottomInputPin);
        for(int counter = 0; counter < connectionVector.size(); ++counter)
        {
          if(((LogicGate) connectionVector.elementAt(counter)).topPinConnection == this)
      {
        if(((LogicGate) connectionVector.elementAt(counter)).getInputState(topPin) != getOutputState())
          {
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(topPin);
          }
      }
      else
      {
        if(((LogicGate) connectionVector.elementAt(counter)).getInputState(bottomPin) != getOutputState())
        {
          ((LogicGate) connectionVector.elementAt(counter)).toggleInput(bottomPin);
        }
      }
    }// end for
  }// end if(OutputPin != !(topInputPin & bottomInputPin))
 }// end calculateGateOperation

 public void displayStates(Graphics g)
//** Shows the gate input and output states at simulation time. If an input
//** to a gate is changed, this change is passed throughout the whole circuit.
//** High value is indicated with a red color, black for low value.
 {
   g.setFont(new Font("Helvetica", Font.BOLD,9));
   Point position = returnGatePosition(xCoordinate, yCoordinate);

   if(topInputPin)        // top pin is high
   {
     g.setColor(Color.red);
     g.drawString("1",position.x + 3 , position.y + 6);
   }
   else           // top pin is low
   {
     g.setColor(Color.black);
     g.drawString("0",position.x + 3 , position.y + 6);
   }
   if(bottomInputPin)      // bottom pin is high
   {
     g.setColor(Color.red);
     g.drawString("1",position.x + 3, position.y + 16);
   }
   else           // bottom pin is low
   {
     g.setColor(Color.black);
     g.drawString("0",position.x + 3, position.y + 16);
   }
   if(OutputPin)          // output pin is high
   {
     g.setColor(Color.red);
     g.drawString("1",position.x + 37, position.y + 11);
   }
   else           // output pin is low
   {
     g.setColor(Color.black);
     g.drawString("0",position.x + 37, position.y + 11);
   }
 }// end displayStates

}// end LogicNand class

class LogicNor extends LogicGate
//***************************************************************************
//** This class is derived from its super class LogicGate and implements the
```

```
//** abstract methods declared. This class performs the operation of logic NOR.
//**
//******************************************************************************
{
   boolean topInputPin = false;
   boolean bottomInputPin = false;
   boolean OutputPin = false;

// The constructor positions a new gate in one of the grids selected.
  public LogicNor(int x, int y, int gatePosition)
  {
    super(x,y,gatePosition);
    calculateGateOperation();
  }// end constructor

  public boolean getInputState(int whichPin)
//** Returns the state of the gate's input. Given which pin was clicked, this
//** method returns the value of that input pin whether high or low.
//** Returns true of the pin is high , false otherwise.
  {
    if(whichPin == topPin)
    {
      return topInputPin;
    }
    else
    {
      return bottomInputPin;
    }
  }// end getInputState

  public boolean getOutputState()
//** Returns the state of the gate's output. this method returns the value
//** of the output pin whether high or low.
  {
    return OutputPin;
  }// end getOutputState

  public void displayGate(Graphics g)
//** Displays the NOR gate, along with any wire connections.
  {
    // get location of gate.
    Point p = returnGatePosition(xCoordinate,yCoordinate);
    g.drawLine(p.x + 4, p.y + 7, p.x + 10,p.y + 7);  // top input pin
    g.drawLine(p.x + 4, p.y + 17, p.x + 10, p.y + 17); // bottom input pin
    g.drawLine(p.x + 35, p.y + 12, p.x + 40, p.y + 12);// output pin
    g.drawArc(p.x + 0,p.y + 2,12,20,90,-180);       // left side arc
    g.drawArc(p.x - 17,p.y + 2,45,20,90,-180);    // right side arc
    g.fillOval(p.x + 0, p.y + 5,4,4);  // bubble on top input pin
    g.fillOval(p.x + 0, p.y + 15,4,4);  // bubble on bottom input pin
    g.fillOval(p.x + 41, p.y + 10,4,4); // bubble on output pin
    g.drawOval(p.x + 29, p.y + 9,6,6); // negation bubble on output pin

    paintConnections(g,p);
  }// end displayGate

  public void paintConnections(Graphics g, Point p)
//** This method displays all wire connections of a gate.
  {
    LogicGate tempGate = null; // gate connected to "this".

    // for all connections to this gate.
    for(int counter = 0; counter < connectionVector.size(); ++counter)
    {
      tempGate = (LogicGate) connectionVector.elementAt(counter);
      Point position = returnGatePosition(tempGate.xCoordinate, tempGate.yCoordinate);

      // firstOccurance is the index of tempGate in the connection array.
      int firstOccurance = connectionVector.indexOf(tempGate);
      if(firstOccurance > -1)
      {
        //secondOccuance is the second time tempGate appears in the same connection array.
        int secondOccuance = connectionVector.indexOf(tempGate,firstOccurance+1);
if(secondOccuance > firstOccurance) // both inputs of tempGate are connected to "this".
        {
          // these Objects has one input, draw first connection to the top pin of tempGate.
          if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
          }
          else          // else it's a top pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
          }
          //draw second connection to the bottom pin of tempGate.
          g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
        }
        else // else tempGate has only one input from "this".
        {
          // is this (this object) connected to tempGate's top input pin?
          if(tempGate.topPinConnection == this)
          {
            if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))  // these Objects has
one input.
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
            }
            else          // else it's a top pin
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
            }
          }// end if(tempGate.topPinConnection == this)
          else    //  or the bottom pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
          }
        }// end else
      }// end if(firstOccurance > -1)
      tempGate = null;    // reset to null.
    }// end for
  }// end paintConnections

  public void toggleInput(int whichPin)
//** Toggles the gate input state between high and low.
//** It allows the user to change the value of an
//** input pin between 0 and 1.
  {
    if(whichPin == topPin)
    {
      topInputPin = !topInputPin;
    }
    else
    {
      bottomInputPin = !bottomInputPin;
    }
    calculateGateOperation();
  }// end toggleInput

  public void calculateGateOperation()
//** This method computes the logic NOR operation between the two gate's
//** input pins. The result of this calculation is broadcasted on the
//** output pin.
  {
    if(OutputPin != !(topInputPin | bottomInputPin))
    {
      OutputPin = !(topInputPin | bottomInputPin);
      for(int counter = 0; counter < connectionVector.size(); ++counter)
      {
        if(((LogicGate) connectionVector.elementAt(counter)).topPinConnection == this)
        {
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(topPin) != getOutputState())
            {
              ((LogicGate) connectionVector.elementAt(counter)).toggleInput(topPin);
            }
        }
        else
        {
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(bottomPin) != getOutputState())
          {
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(bottomPin);
          }
        }
      }// end for
    }
  }// end calculateGateOperation

  public void displayStates(Graphics g)
//** Shows the gate input and output states at simulation time. If an input
//** to a gate is changed, this change is passed throughout the whole circuit.
//** High value is indicated with a red color, black for low value.
  {
    g.setFont(new Font("Helvetica", Font.BOLD,9));
    Point position = returnGatePosition(xCoordinate, yCoordinate);

    if(topInputPin)         // top pin is high
    {
      g.setColor(Color.red);
      g.drawString("1",position.x + 3 , position.y + 6);
    }
    else                    // top pin is low
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 3 , position.y + 6);
    }
    if(bottomInputPin)      // bottom pin is high
    {
      g.setColor(Color.red);
      g.drawString("1",position.x + 3, position.y + 16);
    }
    else                    // bottom pin is low
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 3, position.y + 16);
    }
    if(OutputPin)           // output pin is high
    {
      g.setColor(Color.red);
      g.drawString("1",position.x + 37, position.y + 11);
    }
    else                  // output pin is low
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 37, position.y + 11);
    }
  }// end displayStates

}// end LogicNor class

class Connector extends LogicGate
//****************************************************************************
//** This class is derived from its super class LogicGate and implements the
//** abstract methods declared. This class provides a Connector object for
//** the ease and clarity of routing wires around gates.
//****************************************************************************
{
   boolean topInputPin = false;
   boolean bottomInputPin = false;
   boolean OutputPin = false;


  // The constructor positions a new connector in one of the grids selected.
  public Connector(int x, int y, int gatePosition)
  {
    super(x,y,gatePosition);
    isConnectorObject = true;
    calculateGateOperation();
  }// end constructor

  public void displayGate(Graphics g)
//** Displays the connector object.
  {
    // get location of connector object.
    Point p = returnGatePosition(xCoordinate,yCoordinate);
    g.fillRect(p.x, p.y + 7, 7, 11); // connector object.
    paintConnections(g,p);
  }// end displayGate

  public void paintConnections(Graphics g, Point p)
//** This method displays all wire connections of a gate.
  {
    LogicGate tempGate = null;  // gate connected to "this".

    // for all connections to this gate.
    for(int counter = 0; counter < connectionVector.size(); ++counter)
    {
      tempGate = (LogicGate) connectionVector.elementAt(counter);
      Point position = returnGatePosition(tempGate.xCoordinate, tempGate.yCoordinate);

      // firstOccurance is the index of tempGate in the connection array.
      int firstOccurance = connectionVector.indexOf(tempGate);
      if(firstOccurance > -1)
      {
        //secondOccuance is the second time tempGate appears in the same connection array.
        int secondOccuance = connectionVector.indexOf(tempGate,firstOccurance+1);
if(secondOccuance > firstOccurance) // both inputs of tempGate are connected to "this".
        {
          // these Objects has one input, draw first connection to the top pin of tempGate.
          if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))
          {
```

```
          g.drawLine(p.x + 3, p.y + 12, position.x + 3, position.y + 12);
        }
        else        // else it's a top pin
        {
          g.drawLine(p.x + 3, p.y + 7, position.x + 3, position.y + 7);
        }
        //draw second connection to the bottom pin of tempGate.
        g.drawLine(p.x + 3, p.y + 17, position.x + 3, position.y + 17);
      }
      else // else tempGate has only one input from "this".
      {
        // is this (this object) connected to tempGate's top input pin?
        if(tempGate.topPinConnection == this)
        {
          if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject)) // these Objects has
one input.
          {
            g.drawLine(p.x + 3, p.y + 12, position.x + 3, position.y + 12);
          }
          else        // else it's a top pin
          {
            g.drawLine(p.x + 3, p.y + 7, position.x + 3, position.y + 7);
          }
        }// end if(tempGate.topPinConnection == this)
        else    // or the bottom pin
        {
          g.drawLine(p.x + 3, p.y + 17, position.x + 3, position.y + 17);
        }
      }// end else
    }// end if(firstOccurance > -1)
    tempGate = null;    // reset to null.
  }// end for
}// end paintConnections

public void toggleInput(int whichPin)
//** Toggles the gate input state between high and low.
//** It allows the user to change the value of an
//** input pin between 0 and 1.
{
  topInputPin = ! topInputPin; // no change, since this object is just a buffer.
  calculateGateOperation();
}// end toggleInput

public void calculateGateOperation()
//** This method passes the signal through from the connector's
//** input pin. The result of this calculation is broadcasted on the
//** output pin.
{
  if(OutputPin != topInputPin)
  {
    // if output is wrong, change it.
    OutputPin = topInputPin;

    for(int counter = 0; counter < connectionVector.size(); ++counter)
    {
      // for all connections, if the gate's input is connected to this output.
      if(((LogicGate) connectionVector.elementAt(counter)).topPinConnection == this)
      {
        // and the output is inaccurate?
        if(((LogicGate) connectionVector.elementAt(counter)).getInputState(topPin) != getOutputState())
        {
          // change the value.
```

```
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(topPin);
          }
        }
        else  // do the same for the bottom input pin
        {
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(bottomPin) != getOutputState())
          {
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(bottomPin);
          }
        }// end else
      }// end for
    }// end if(OutputPin != topInputPin)
  }// end calculateGateOperation

  public boolean getInputState(int whichPin)
//** Returns the state of the gate's input.
//** true if the pin is high , false otherwise.
  {
    return topInputPin;
  }// end getInputState

  public int whichInput(Point point)
//** Returns which one of two inputs to a gate has been clicked. It is modified
//** here since a connector object has only one input.
  {
    Point position = returnGatePosition(xCoordinate, yCoordinate);

    // is the mouse at the input area?
    if((point.x > position.x - 5) && (point.x < position.x + 5)
      && (point.y > position.y + 2) && (point.y < position.y + gridHeight - 2))
    {
      return topPin;
    }
    else
    {
      return nietherPin;  // or it could be neither.
    }
  }// end whichInput

  public boolean getOutputState()
//** Returns the state of the gate's output. this method returns the value
//** of the output pin whether high or low.
  {
    return OutputPin;
  }// end getOutputState

  public boolean isOutput(Point point)
//** Returns true if the gate's output has been clicked. This is a helper
//** function to whichGatePart.
  {
    Point position = returnGatePosition(xCoordinate, yCoordinate);

    if((point.x > position.x + 4) && (point.x < position.x + 15)
      && (point.y > position.y + 2) && (point.y < position.y + gridHeight - 2))
    {
      return true;
    }
    else
    {
      return false;    // not at the output pin.
    }
  }// end isOutput
  public void displayStates(Graphics g)
//** Shows the gate input and output states at simulation time. If an input
//** to a gate is changed, this change is passed throughout the whole circuit.
//** High value is indicated with a red color, black for low value.
  {
    g.setFont(new Font("Helvetica", Font.BOLD,9));
    Point position = returnGatePosition(xCoordinate, yCoordinate);

    if(topInputPin)
    {
      g.setColor(Color.red);
      g.drawString("1",position.x + 7, position.y + 6);
    }
    else
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 7, position.y + 6);
    }
  }// end displayStates
}// end Connector class

class Input extends LogicGate
//*****************************************************************************
//** This class is derived from its super class LogicGate and implements the
//** abstract methods declared. This class provides an Input object for
//** the ease and clarity of entering a signal into the circuit.
//*****************************************************************************
{
   boolean topInputPin = false;
   boolean bottomInputPin = false;
   boolean OutputPin = false;


  // The constructor positions a new Input object in one of the grids selected.
  public Input(int x, int y, int gatePosition)
  {
    super(x,y,gatePosition);
    isInputObject = true;
    calculateGateOperation();
  }// end constructor

  public void displayGate(Graphics g)
//** Displays the Input object, along with any wire connections.
  {
    // get location of Input object.
    Point p = returnGatePosition(xCoordinate,yCoordinate);

    g.setColor(Color.blue);
    g.drawRect(p.x + 9, p.y + 4,16,16); // input object.
    g.drawLine(p.x + 25, p.y + 12, p.x + 40, p.y + 12);// output pin
    g.fillOval(p.x + 41, p.y + 10,4,4);  // bubble on output pin
    g.setColor(Color.black);

    paintConnections(g,p);
  }// end displayGate

  public void paintConnections(Graphics g, Point p)
//** This method displays all wire connections of a gate.
  {
    LogicGate tempGate = null;  // gate connected to "this".
    // for all connections to this gate.
    for(int counter = 0; counter < connectionVector.size(); ++counter)
    {
      tempGate = (LogicGate) connectionVector.elementAt(counter);
      Point position = returnGatePosition(tempGate.xCoordinate, tempGate.yCoordinate);

      // firstOccurance is the index of tempGate in the connection array.
      int firstOccurance = connectionVector.indexOf(tempGate);
      if(firstOccurance > -1)
      {
        //secondOccuance is the second time tempGate appears in the same connection array.
        int secondOccuance = connectionVector.indexOf(tempGate,firstOccurance+1);
if(secondOccuance > firstOccurance) // both inputs of tempGate are connected to "this".
        {
          // these Objects has one input, draw first connection to the top pin of tempGate.
          if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
          }
          else          // else it's a top pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
          }
          //draw second connection to the bottom pin of tempGate.
          g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
        }
        else // else tempGate has only one input from "this".
        {
          // is this (this object) connected to tempGate's top input pin?
          if(tempGate.topPinConnection == this)
          {
            if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))  // these Objects has
one input.
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 12);
            }
            else         // else it's a top pin
            {
              g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 7);
            }
          }// end if(tempGate.topPinConnection == this)
          else    // or the bottom pin
          {
            g.drawLine(p.x + gridWidth, p.y + 12, position.x, position.y + 17);
          }
        }// end else
      }// end if(firstOccurance > -1)
      tempGate = null;   // reset to null.
    }// end for
  }// end paintConnections

  public void toggleInput(int whichPin)
//** Toggles the gate input state between high and low.
//** It allows the user to change the value of an
//** input pin between 0 and 1.
  {
    topInputPin = ! topInputPin; // change input value.
    calculateGateOperation();
  }// end toggleInput

  public void calculateGateOperation()
//** This method passes the signal through from the connector's
```

```
//** input pin. The result of this calculation is broadcasted on the
//** output pin.
  {
    if(OutputPin != topInputPin)
    {
      // if output is wrong, change it.
      OutputPin = topInputPin;

      for(int counter = 0; counter < connectionVector.size(); ++counter)
      {
        // for all connections, if the gate's input is connected to this output.
        if(((LogicGate) connectionVector.elementAt(counter)).topPinConnection == this)
        {
          // and the output is inaccurate?
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(topPin) != getOutputState())
          {
            // change the value.
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(topPin);
          }
        }
        else  // do the same for the bottom input pin
        {
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(bottomPin) != getOutputState())
          {
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(bottomPin);
          }
        }// end else
      }// end for
    }// end if(OutputPin != topInputPin)
  }// end calculateGateOperation

  public boolean getInputState(int whichPin)
//** Returns the state of the gate's input.
//** true if the pin is high , false otherwise.
  {
    return topInputPin;
  }// end getInputState

  public int whichInput(Point point)
//** Returns which one of two inputs to a gate has been clicked. It is modified
//** here since a connector object has only one input.
  {
    return nietherPin;  // Input object does not receive input.
  }// end whichInput

  public boolean getOutputState()
//** Returns the state of the gate's output. this method returns the value
//** of the output pin whether high or low.
  {
    return OutputPin;
  }// end getOutputState

  public void displayStates(Graphics g)
//** Shows the gate input and output states at simulation time. If an input
//** to a gate is changed, this change is passed throughout the whole circuit.
//** High value is indicated with a red color, black for low value.
  {
    g.setFont(new Font("Helvetica", Font.BOLD,9));
    Point position = returnGatePosition(xCoordinate, yCoordinate);

    if(topInputPin)
    {
      g.setColor(Color.red);
      g.drawLine(position.x + 25, position.y + 12, position.x + 40, position.y + 12);// output pin
      g.fillOval(position.x + 41, position.y + 10,4,4);  // bubble on output pin
      g.drawString("1",position.x + 16, position.y + 16);
    }
    else
    {
      g.setColor(Color.black);
      g.drawString("0",position.x + 16, position.y + 16);
    }
  }// end displayStates
}// end Connector Input

class Output extends LogicGate
//*************************************************************************
//** This class is derived from its super class LogicGate and implements the
//** abstract methods declared. This class provides an Output object for
//** the ease and clarity of exiting a signal from the circuit.
//*************************************************************************
{
  boolean topInputPin = false;
  boolean bottomInputPin = false;
  boolean OutputPin = false;


  // The constructor positions a new Output object in one of the grids selected.
  public Output(int x, int y, int gatePosition)
  {
    super(x,y,gatePosition);
    isOutputObject = true;
    calculateGateOperation();
  }// end constructor

  public void displayGate(Graphics g)
//** Displays the Output object, along with any wire connections.
  {
    // get location of Output object.
    Point p = returnGatePosition(xCoordinate,yCoordinate);

    g.setColor(Color.gray);
    g.drawRect(p.x + 20, p.y + 4,16,16); // output object.
    g.setColor(Color.black);
    g.drawLine(p.x + 4, p.y + 12, p.x + 19, p.y + 12);// output pin
    g.fillOval(p.x + 0, p.y + 10,4,4);  // bubble on output pin

// for all connections to this object.
    for(int counter = 0; counter < connectionVector.size(); ++counter)
    {
      LogicGate tempGate = (LogicGate) connectionVector.elementAt(counter);
      Point position = returnGatePosition(tempGate.xCoordinate, tempGate.yCoordinate);

      // is this (this object) connected to tempGate's input top pin?
      if(tempGate.topPinConnection == this)
      {
        if((tempGate.isConnectorObject) || (tempGate.isNotGate) || (tempGate.isOutputObject))  // these Objects has
one input.
        {
          g.drawLine(p.x + 0, p.y + 12, position.x, position.y + 12);
        }
        else         // else it's a top pin
        {
          g.drawLine(p.x + 0, p.y + 12, position.x, position.y + 7);
        }
      }
      else              //  or the bottom pin
      {
        g.drawLine(p.x + 0, p.y + 12, position.x, position.y + 17);
      }
    }// end for
  }// end displayGate

  public void toggleInput(int whichPin)
//** Toggles the gate input state between high and low.
//** It allows the user to change the value of an
//** input pin between 0 and 1.
  {
    topInputPin = ! topInputPin; // no change, since this object is just a buffer.
    calculateGateOperation();
  }// end toggleInput

  public void calculateGateOperation()
//** This method passes the signal through from the connector's
//** input pin. The result of this calculation is broadcasted on the
//** output pin.
  {
    if(OutputPin != topInputPin)
    {
      // if output is wrong, change it.
      OutputPin = topInputPin;

      for(int counter = 0; counter < connectionVector.size(); ++counter)
      {
        // for all connections, if the gate's input is connected to this output.
        if(((LogicGate) connectionVector.elementAt(counter)).topPinConnection == this)
        {
          // and the output is inaccurate?
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(topPin) != getOutputState())
          {
            // change the value.
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(topPin);
          }
        }
        else   // do the same for the bottom input pin
        {
          if(((LogicGate) connectionVector.elementAt(counter)).getInputState(bottomPin) != getOutputState())
          {
            ((LogicGate) connectionVector.elementAt(counter)).toggleInput(bottomPin);
          }
        }// end else
      }// end for
    }// end if(OutputPin != topInputPin)
  }// end calculateGateOperation

  public boolean getInputState(int whichPin)
//** Returns the state of the gate's input.
//** true if the pin is high , false otherwise.
  {
    return topInputPin;
  }// end getInputState

  public int whichInput(Point point)
//** Returns which one of two inputs to a gate has been clicked. It is modified
//** here since a connector object has only one input.
  {
   Point position = returnGatePosition(xCoordinate, yCoordinate);

   // is the mouse at the input area?
   if((point.x > position.x - 10) && (point.x < position.x + 8)
     && (point.y > position.y + 5) && (point.y < position.y + gridHeight - 5))
   {
     return topPin;
   }
   else
   {
     return nietherPin;  // or it could be neither.
   }
  }// end whichInput

public boolean isOutput(Point point)
//** Returns true if the gate's output has been clicked. This is a helper
//** function to whichGatePart.
  {
   return false; // Output object does not send output signal.
  }// end isOutput
  public boolean getOutputState()
//** Returns the state of the gate's output. this method returns the value
//** of the output pin whether high or low.
  {
   return OutputPin;
  }// end getOutputState

  public void displayStates(Graphics g)
//** Shows the gate input and output states at simulation time. If an input
//** to a gate is changed, this change is passed throughout the whole circuit.
//** High value is indicated with a red color, black for low value.
  {
   g.setFont(new Font("Helvetica", Font.BOLD,9));
   Point position = returnGatePosition(xCoordinate, yCoordinate);

   if(OutputPin)          // output pin is high
   {
     g.setColor(Color.red);
     g.drawRect(position.x + 20, position.y + 4,16,16); // input object.
     g.drawString("1",position.x + 26, position.y + 16);
   }
   else                   // output pin is low
   {
     g.setColor(Color.black);
     g.drawString("0",position.x + 26, position.y + 16);
   }
  }// end displayStates
}// end Output class

public class Simulator extends Frame implements ActionListener, WindowListener
//*****************************************************************************
//** This is the main class that runs the simulation. It inherits from Frame
//** which opens the drawing window. This object is responsible for setting up
//** the graphical interface.
//*****************************************************************************
{
  MyPanel thePanel; //declaration needed here for the paint call in actionPerformed.
  MyCanvas theCanvas ; //declaration needed here for processResult.
  public boolean buildCircuit = true;  // master switch for setting the simulator mode.
  Button functionButton, clearAllButton, eraseButton, saveButton, openButton;
  Button andButton, orButton, notButton, xorButton, nandButton, norButton;
  Button stopButton, connectorButton, inputButton, outputButton;
```

```
private Label currentState; // declaration needed here for the GridBagLayout.
Choice colorChoice; // choice list for canvas background colors.


public Simulator()    // class constructor
{
  setTitle("Welcome to LogicCity");       // window title
  Font font = new Font("Helvetica", Font.BOLD,12); // select font
  setFont(font);
  addWindowListener(this); // allows simulator to detect if the window is manipulated.

  // GridBagLayout arranges all components in rows and columns.
  GridBagLayout gridBagLayout = new GridBagLayout();
  setLayout(gridBagLayout);

  // The GridBagConstraints class specifies constraints for components
  // that are laid out using the GridBagLayout class.
  // Constructor creates a GridBagConstraint object with all of its fields set
  // to their default value.
  GridBagConstraints gridBagConstraints = new GridBagConstraints();

  // Creating the function button.
  functionButton = new Button("FUNCTION");
  functionButton.addActionListener(this);
  // fill indicates the fill behavior of component inside cell,
// one of NONE, BOTH, HORIZONTAL, VERTICAL. This field is used when the
  // component's display area is larger than the component's requested size.
  gridBagConstraints.fill =  GridBagConstraints.HORIZONTAL;
  // weightx and weighty indicates capacity of cell to grow.
  gridBagConstraints.weightx = 0.0;
  gridBagConstraints.weighty = 0.0;
  gridBagConstraints.gridwidth = 1;
  gridBagLayout.setConstraints(functionButton, gridBagConstraints);
  add(functionButton);

  // Creating the label that displays the simulator status.
  currentState = new Label("BUILD");
  gridBagConstraints.weightx = 1.0;
  gridBagConstraints.weighty = 1.0;
  // REMAINDER specifies that this component is the last component in
  // its column or row.
  gridBagConstraints.gridwidth = 1;
  gridBagLayout.setConstraints(currentState, gridBagConstraints);
  add(currentState);

  // Creating the edit menu.
  Panel editMenu = new Panel();
  editMenu.setLayout(new BorderLayout());

  Panel clearAllPanel = new Panel();   // creating the clear all button
  clearAllPanel.setLayout(new BorderLayout());
  clearAllPanel.add(clearAllButton = new Button("CLEAR ALL"));
  editMenu.add("West",clearAllPanel);

  Panel erasePanel = new Panel();        // creating the erase button
  erasePanel.setLayout(new BorderLayout());
  erasePanel.add(eraseButton = new Button("ERASE"));
  clearAllPanel.add("West",erasePanel);

  Panel savePanel = new Panel();         // creating the save button
  savePanel.setLayout(new BorderLayout());
  savePanel.add(saveButton = new Button("SAVE"));
```

```
erasePanel.add("West",savePanel);

Panel openPanel = new Panel();        // creating the open button
openPanel.setLayout(new BorderLayout());
openPanel.add(openButton = new Button("OPEN"));
savePanel.add("West",openPanel);

gridBagConstraints.weightx = 1.0;
gridBagConstraints.weighty = 1.0;
gridBagConstraints.fill =  GridBagConstraints.HORIZONTAL;
gridBagConstraints.gridwidth = GridBagConstraints.REMAINDER;
gridBagLayout.setConstraints(editMenu, gridBagConstraints);
add(editMenu);

// Creating the gates menu.
Panel gateMenu = new Panel();
gateMenu.setLayout(new BorderLayout());

Panel andPanel = new Panel();        // creating the AND button
andPanel.setLayout(new BorderLayout());
andPanel.add(andButton = new Button("AND"));
gateMenu.add("North",andPanel);

Panel orPanel = new Panel();        // creating the OR button
orPanel.setLayout(new BorderLayout());
orPanel.add(orButton = new Button("OR"));
andPanel.add("South",orPanel);

Panel notPanel = new Panel();        // creating the NOT button
notPanel.setLayout(new BorderLayout());
notPanel.add(notButton = new Button("NOT"));
orPanel.add("South",notPanel);

Panel xorPanel = new Panel();        // creating the XOR button
xorPanel.setLayout(new BorderLayout());
xorPanel.add(xorButton = new Button("XOR"));
notPanel.add("South",xorPanel);

Panel nandPanel = new Panel();        // creating the NAND button
nandPanel.setLayout(new BorderLayout());
nandPanel.add(nandButton = new Button("NAND"));
xorPanel.add("South",nandPanel);

Panel norPanel = new Panel();        // creating the NOR button
norPanel.setLayout(new BorderLayout());
norPanel.add(norButton = new Button("NOR"));
nandPanel.add("South",norPanel);

Panel connectorPanel = new Panel();        // creating the NOR button
connectorPanel.setLayout(new BorderLayout());
connectorPanel.add(connectorButton = new Button("CONN"));
norPanel.add("South",connectorPanel);

Panel inputPanel = new Panel();        // creating the INPUT button
inputPanel.setLayout(new BorderLayout());
inputPanel.add(inputButton = new Button("INPUT"));
connectorPanel.add("South",inputPanel);

Panel outputPanel = new Panel();        // creating the OUTPUT button
outputPanel.setLayout(new BorderLayout());
outputPanel.add(outputButton = new Button("OUTPUT"));
inputPanel.add("South",outputPanel);
```

```
    // creating the choice list of background colors.
    colorChoice = new Choice();
    colorChoice.addItem("LAVENDER");
    colorChoice.addItem("WHITE");
    colorChoice.addItem("GREY");
    colorChoice.addItem("BEIGE");
    colorChoice.addItem("OLIVE");
    colorChoice.addItem("BLUE");
    colorChoice.addItem("PINK");
    colorChoice.addItem("GREEN");
    colorChoice.addItem("ORANGE");
    colorChoice.addItem("YELLOW");
    gateMenu.add("South",colorChoice);

    gridBagConstraints.weightx = 1.0;
    gridBagConstraints.weighty = 1.0;
    gridBagConstraints.fill =  GridBagConstraints.VERTICAL;
    gridBagConstraints.gridwidth = 1;
    gridBagLayout.setConstraints(gateMenu, gridBagConstraints);
    add(gateMenu);

    // Creating the drawing window.
    thePanel = new MyPanel(this);
    gridBagConstraints.weightx = 100;
    gridBagConstraints.weighty = 100;
    gridBagConstraints.fill =  GridBagConstraints.BOTH;
    gridBagConstraints.gridwidth = GridBagConstraints.REMAINDER;
    gridBagLayout.setConstraints(thePanel, gridBagConstraints);
    add(thePanel);
  }// end constructor

// The next seven window-related methods are included to satisfy the
// compiler's demand when implementing the interface WindowListener.

  public void windowClosing(WindowEvent event)
// Closes the simulator window and aborts the program.
  {
    System.exit(0);  // terminate the program.
  }

  public void windowClosed(WindowEvent event)
  {
  }

  public void windowIconified(WindowEvent event)
  {
  }

  public void windowDeiconified(WindowEvent event)
  {
  }

  public void windowActivated(WindowEvent event)
  {
  }

  public void windowDeactivated(WindowEvent event)
  {
  }

  public void windowOpened(WindowEvent event)
  {
  }

  public void actionPerformed(ActionEvent evt)
//** This method determines whether to build a circuit or run the simulator.
//** Initially, it's set for building a circuit after which the user can
//** select the mode desired.
  {
    String myString = " ";
    String string = evt.paramString(); // get all info about button clicked.
    StringTokenizer stringTokenizer = new StringTokenizer(string,"=");
    while (stringTokenizer.hasMoreTokens()) // parse the info string.
    {
      myString = stringTokenizer.nextToken(); // get next token in string.
    }// end while

    if(myString.compareTo("FUNCTION") == 0) // is the click on the xor button?
    {
      if(theCanvas.logicCircuit.size() > 0) // if there is a circuit to simulate
      {
        buildCircuit = !buildCircuit;
        if(buildCircuit)        // if in edit mode
        {
          currentState.setText("BUILD");
        }
        else                    // else in simulation mode
        {
          currentState.setText("SIMULATE");
        }
      }
    }

    thePanel.repaint();      // repaint the window
  }// end actionPerformed

  public void processResult(Dialog source, Object object)
//** this method implements the method in the
//** interface resultProcessor. Its purpose is to allow the programmer
//** to reuse a dialog box in other programs.
  {
    if(source instanceof WarningDialog) // is source of type dialog
    {
      if(buildCircuit)   // if in build mode
      {
        setTitle("Welcome to LogicCity");      // window title
        theCanvas.logicCircuit.removeAllElements(); // clear all gates in circuit.
        theCanvas.currentState = theCanvas.createGate; // reset mode to pointing.
        theCanvas.repaint(); // refresh screen.
      }// end if(simulator.buildCircuit)
    }// end if(source instanceof WarningDialog)
  }// end processResult

  public void executeDecision(Dialog source, boolean decision)
//** this method implements the method in the
//** interface resultProcessor. Its purpose is to allow the programmer
//** to reuse a dialog box in other programs.
  {
    if(source instanceof OpenFileDecision) // is source of type dialog
    {
      if(buildCircuit)   // if in build mode
      {
        if(decision)    // if user wants to save current circuit first.
        {
          theCanvas.doSaveFile();   // save circuit first.
          theCanvas.doOpenFile();   // then open desired file.
        }
        else
        {
          theCanvas.doOpenFile();   // if save is not of interest, then just open.
        }
      }// end if(simulator.buildCircuit)
    }// end if(source instanceof WarningDialog)
  }// end processResult

  public static void main(String[] args)
//** This is the main function where it all starts. It's cancelled if the
//** application is converted into an applet.
  {
    Frame simulator = new Simulator();
    //simulator.setSize(Toolkit.getDefaultToolkit().getScreenSize());
    simulator.setSize(550,400);
    simulator.setBackground(new Color(0.98f, 0.95f, 0.60f));
    simulator.show(); //makes a component visible, its typical use is for a Frame.
  }// end main
}// end Simulator class

interface ResultProcessor
//******************************************************************************
//** This is a generic interface to enable the reuse of the dialog box code
//** in another program.
//******************************************************************************
{
  public void processResult(Dialog source, Object object);  // declaration
  public void executeDecision(Dialog source, boolean decision);  // declaration
}// end interface ResultProcessor

class MyPanel extends Panel
//******************************************************************************
//** This class is used as an intermediate step to creating the drawing
//** canvas. It's used to set the GridLayout where this method
//** is defined in class Panel and not in class Canvas. To create a GUI
//** application, place the GUI functionality in a class that extend Panel.
//******************************************************************************
{
  Simulator simulator; // needed for paint.


  public MyPanel(Simulator simulator) // need to pass simulator object.
  {
    super();    // call the parent constructor, must be first line here.
    setLayout(new GridLayout(1,0));
    // Creates a grid layout with the specified number of rows and columns.
    // All components in the layout are given equal size.
    // One, but not both, of rows and cols can be zero, which means that any
    // number of objects can be placed in a row or in a column.
    this.simulator = simulator;    // pass the simulator object.
    simulator.theCanvas = new MyCanvas(simulator); //create drawing canvas.
    simulator.theCanvas.setBackground(new Color(204, 204, 255));
    add(simulator.theCanvas);  // add the drawing canvas to the layout.
    ScrollPane pane = new ScrollPane(); // create scrollable pane.
    setLayout(new BorderLayout());     // select type of layout.
    pane.add(simulator.theCanvas);     // add drawing canvas to the scrollable pane.
    add("Center", pane);               // position the pane into the center of the screen.
    Adjustable horizontal = pane.getHAdjustable(); // get the horizontal scroll object.
    horizontal.setUnitIncrement(40); // change the horizontal scroll increment.
    Adjustable vertical = pane.getVAdjustable(); // get the vertical scroll object.
    vertical.setUnitIncrement(20);  // change the vertical scroll increment.
  }// end constructor

  public void paint(Graphics g)
// paint specifies how object g is to be displayed. All drawing in Java
// must go through a graphics object. note: any time you need to put text or
// graphics into a window, you need to override the paint method from the
// Component class, so you need to write a new class for this that overrides
// the paint method.
  {
    simulator.theCanvas.repaint();
  }// end paint
}// end MyPanel class

class OpenFileDecision extends Dialog implements WindowListener, ActionListener
//*************************************************************************
//** This class is used to create a dialog box that asks the user for
//** confirmation in the event that the open file button is clicked. this
//** confirmation prevents the accidental deletion of the current circuit.
//*************************************************************************
{
  Button okButton, noButton, cancelButton;

  public OpenFileDecision(Simulator parent)  // class constructor.
  {
    super(parent, "WARNING: CONFIRM FIRST", true); // set title and parent.
    Panel p1 = new Panel(); // create a panel to house the two buttons needed.
    p1.add(okButton = new Button("OK")); // create OK button.
    p1.add(noButton = new Button("NO")); // create OK button.
    p1.add(cancelButton = new Button("CANCEL")); // create CANCEL button.
    okButton.addActionListener(this); // register the OK button
    noButton.addActionListener(this); // register the NO button
    cancelButton.addActionListener(this); // register the CANCEL button
    add("South", p1);  // add to bottom of panel.

    Panel p2 = new Panel(); // create a panel to house the warning needed.
    p2.add(new Label("Do you want to save the current circuit first?"));
    add("Center", p2); // add the warning text above the buttons.
    setSize(350, 150);  // set the warning dialog box size.
    addWindowListener(this);  // allows simulator to detect if the window is manipulated.
  }// end constructor

  public void actionPerformed(ActionEvent evt)
//** This method decides what to do if one of the two buttons in the
//** warning dialog box is clicked.
  {
    String myString = " ";
    String string = evt.paramString(); // get all info about button clicked.
    StringTokenizer stringTokenizer = new StringTokenizer(string,"=");
    while (stringTokenizer.hasMoreTokens())  // parse the info string.
    {
      myString = stringTokenizer.nextToken(); // get next token in string.
    }// end while

    if(myString.compareTo("OK") == 0) // is the click on the ok button?
    {
      dispose(); // closes the dialog box.
      ((Simulator) getParent()).executeDecision(this, true); // deal with this event.
    }
    if(myString.compareTo("NO") == 0) // is the click on the ok button?
    {
      dispose(); // closes the dialog box.
      ((Simulator) getParent()).executeDecision(this, false); // deal with this event.
    }
    else if((myString.compareTo("CANCEL") == 0)) // is the click on the cancel button?
    {
      dispose();  // no need to do any thing, just close the dialog box.
    }
  }// end actionPerformed

// The next seven window-related methods are included to satisfy the
// compiler's demand when implementing the interface WindowListener.

  public void windowClosing(WindowEvent event)
// Closes the simulator window and aborts the program.
  {
    dispose();  // enables the user to close the dialog window if the X is clicked.
  }

  public void windowClosed(WindowEvent event)
  {
  }

  public void windowIconified(WindowEvent event)
  {
  }

  public void windowDeiconified(WindowEvent event)
  {
  }

  public void windowActivated(WindowEvent event)
  {
  }

  public void windowDeactivated(WindowEvent event)
  {
  }

  public void windowOpened(WindowEvent event)
  {
  }

}// end class OpenFileDecision

class WarningDialog extends Dialog implements WindowListener, ActionListener
//****************************************************************************
//**  This class is used to create a dialog box that asks the user for
//**  confirmation in the event that the clearAll button is clicked. this
//**  confirmation prevents the accidental deletion of the entire circuit.
//****************************************************************************
{
  Button okButton, cancelButton;

  public WarningDialog(Simulator parent)  // class constructor.
  {
    super(parent, "PLEASE CONFIRM DELETION", true); // set title and parent.
    Panel p1 = new Panel(); // create a panel to house the two buttons needed.
    p1.add(okButton = new Button("OK"));  // create OK button.
    p1.add(cancelButton = new Button("CANCEL")); // create CANCEL button.
    okButton.addActionListener(this); // register the OK button
    cancelButton.addActionListener(this); // register the CANCEL button
```

```
      add("South", p1);  // add to bottom of panel.

      Panel p2 = new Panel(); // create a panel to house the warning needed.
      p2.add(new Label("Are you sure you want to delete the entire circuit?"));
      add("Center", p2); // add the warning text above the buttons.
      setSize(350, 150); // set the warning dialog box size.
      addWindowListener(this); // allows simulator to detect if the window is manipulated.
    }// end constructor

    public void actionPerformed(ActionEvent evt)
  //** This method decides what to do if one of the two buttons in the
  //** warning dialog box is clicked.
    {
      String myString = " ";
      String string = evt.paramString(); // get all info about button clicked.
      StringTokenizer stringTokenizer = new StringTokenizer(string,"=");
      while (stringTokenizer.hasMoreTokens()) // parse the info string.
      {
        myString = stringTokenizer.nextToken(); // get next token in string.
      }// end while

      if(myString.compareTo("OK") == 0) // is the click on the ok button?
      {
        dispose(); // closes the dialog box.
        ((Simulator) getParent()).processResult(this, evt); // deal with this event.
      }
      else if((myString.compareTo("CANCEL") == 0)) // is the click on the cancel button?
      {
        dispose(); // no need to do any thing, just close the dialog box.
      }
    }// end actionPerformed

  // The next seven window-related methods are included to satisfy the
  // compiler's demand when implementing the interface WindowListener.

    public void windowClosing(WindowEvent event)
  // Closes the simulator window and aborts the program.
    {
      dispose(); // enables the user to close the dialog window if the X is clicked.
    }

    public void windowClosed(WindowEvent event)
    {
    }

    public void windowIconified(WindowEvent event)
    {
    }

    public void windowDeiconified(WindowEvent event)
    {
    }

    public void windowActivated(WindowEvent event)
    {
    }

    public void windowDeactivated(WindowEvent event)
    {
    }

    public void windowOpened(WindowEvent event)
  {
  }

}// end class WarningDialog

class MyCanvas extends Canvas implements MouseListener, MouseMotionListener, ActionListener
//*****************************************************************************
//** This class controls the functionality of the simulator.
//** If graphics are used, make an extra class that extend Canvas. This class
//** must provide a paint method and a public method for communication. A
//** canvas can receive input from the user in the form of mouse events.
//*****************************************************************************
{
  protected Vector logicCircuit = new Vector(); // circuit that holds all the gates.
  private int xCoordinate, yCoordinate; // used to keep track of a gate's Coords and mouse motion.
  private int whichInputPin ;      // need declared here for sharing.
  private int currentGate = 0;     // the gate currently selected.
  protected int currentState = 0;    // the current state of the simulator.
  private final int and = 1;        // digit value of the and gate.
  private final int or = 2;        // digit value of the or gate.
  private final int not = 3;        // digit value of the not gate.
  private final int xor = 4;        // digit value of the xor gate.
  private final int nand = 5;       // digit value of the nand gate.
  private final int nor = 6;        // digit value of the nor gate.
  private final int connector = 7;        // digit value of the connector object.
  private final int input = 8;       // digit value of the input object.
  private final int output = 9;        // digit value of the output object.
  protected final int createGate = 0;  // pointing state.
  private final int connectGate = 1;  // connecting state.
  private final int eraseGate = 2;   // delete state.
  private final int clearAll = 3;    // erase entire circuit state.
  private LogicGate theDestinationGate = null; // gate selected as the destination point.
  private LogicGate theOriginatingGate = null;  // gate selected as the originating source.
  private LogicGate logicGate = null;       // current gate being manipulated, declared here for paint.
  protected Point mouseClickPoint = null;  // mouse position at click time.
  protected Simulator simulator;         // declare simulator object for canvas.
  Dimension offScreenDimension;  // size of off-screen images.
  Image offScreenImage;         // to use in double buffering.
  Graphics offScreenGraphics;  // object to eliminate flashing.


  public MyCanvas(Simulator simulator)  // class constructor
  {
    super();    // call the parent constructor, must be first line here.
    this.simulator = simulator;  // pass the simulator to this object.
    addMouseListener(this);     // register class to receive mouse events.
    addMouseMotionListener(this); // register class for mouse motion events.
    simulator.andButton.addActionListener(this); // register the and button
    simulator.orButton.addActionListener(this);  // register the or button
    simulator.notButton.addActionListener(this); // register the not button
    simulator.xorButton.addActionListener(this); // register the xor button
    simulator.nandButton.addActionListener(this);// register the nand button
    simulator.norButton.addActionListener(this); // register the nor button
    simulator.connectorButton.addActionListener(this); // register the connector button
    simulator.openButton.addActionListener(this); // register the open button
    simulator.saveButton.addActionListener(this); // register the save button
    simulator.eraseButton.addActionListener(this); // register the erase button
    simulator.clearAllButton.addActionListener(this); // register the clearAll button
    simulator.inputButton.addActionListener(this); // register the input button
    simulator.outputButton.addActionListener(this); // register the output button
    simulator.functionButton.addMouseListener(this); // register the function button for initial simulation.
    simulator.colorChoice.addMouseListener(this); // register the color menu.
    }// end constructor

    public Dimension getMinimumSize()
//** This method returns the screen minimum size.
    {
      Dimension minimumSize = new Dimension(100,50);
      return minimumSize;
    }// end getMinimumSize

    public Dimension getPreferredSize()
//** This method returns the screen preferred size.
    {
      Dimension preferredSize = new Dimension(2000,2000);
      return preferredSize;
    }// end getPreferredSize

    public void actionPerformed(ActionEvent event)
//** This method determines which button in the tool menu or gate menu is
//** clicked and takes the appropriate action.
    {
      String myString = " ";
      String string = event.paramString(); // get all info about button clicked.
      StringTokenizer stringTokenizer = new StringTokenizer(string,"=");
      while (stringTokenizer.hasMoreTokens())  // parse the info string.
      {
        myString = stringTokenizer.nextToken(); // get next token in string.
      }// end while

      if(myString.compareTo("AND") == 0)   // is the click on the and button?
      {
        currentGate = and;
      }
      else if(myString.compareTo("OR") == 0) // is the click on the or button?
      {
        currentGate = or;
      }
      else if(myString.compareTo("NOT") == 0) // is the click on the not button?
      {
        currentGate = not;
      }
      else if(myString.compareTo("XOR") == 0) // is the click on the xor button?
      {
        currentGate = xor;
      }
      else if(myString.compareTo("NAND") == 0) // is the click on the nand button?
      {
        currentGate = nand;
      }
      else if(myString.compareTo("NOR") == 0) // is the click on the nor button?
      {
        currentGate = nor;
      }
      else if(myString.compareTo("CONN") == 0) // is the click on the connector button?
      {
        currentGate = connector;
      }
      else if(myString.compareTo("INPUT") == 0) // is the click on the input button?
      {
        currentGate = input;
      }
      else if(myString.compareTo("OUTPUT") == 0) // is the click on the output button?
      {
```

```
      currentGate = output;
    }
    else if(myString.compareTo("OPEN") == 0) // is the click on the open button?
    {
      if(simulator.buildCircuit)  // if in build mode
      {
        if(logicCircuit.size() > 0)
        {
          OpenFileDecision decision = new OpenFileDecision(this.simulator);
          decision.show();
        }
        else
        {
          doOpenFile();
        }
        currentState = createGate;
      }// end if(simulator.buildCircuit)
    }// end else if(myString.compareTo("OPEN") == 0)

    else if(myString.compareTo("SAVE") == 0) // is the click on the save button?
    {
      if(simulator.buildCircuit)  // if in build mode
      {
        if(logicCircuit.size() > 0)    // if there is a circuit to be saved.
        {
          doSaveFile();          // call the save method.
          currentState = createGate;  // return to initial state.
        }// end if(logicCircuit.size() > 0)
      }// end if(simulator.buildCircuit)
    }// end else if(myString.compareTo("SAVE") == 0)

    else if(myString.compareTo("ERASE") == 0) // is the click on the erase button?
    {
      currentState = eraseGate;
    }// end else if(myString.compareTo("ERASE") == 0)

    else if(myString.compareTo("CLEAR ALL") == 0) // is the click on the clearAll button?
    {
      if(simulator.buildCircuit)  // if in build mode
      {
        if(logicCircuit.size() > 0)
        {
          WarningDialog warning = new WarningDialog(this.simulator);
          warning.show();
        }// end if(logicCircuit.size() > 0)
      }// end if(simulator.buildCircuit)
    }// end else if(myString.compareTo("CLEAR ALL") == 0)
  }// end actionPerformed

  public void doOpenFile()
//** This method reconstructs a digital logic circuit, that was saved at an
//** earlier time, from a file. The circuit's total environment is restored
//** in the order it was saved.
  {
    LogicGate tempGate = null;
    int andCounter, orCounter, notCounter, xorCounter, nandCounter,
       norCounter, connectorCounter, inputCounter, outputCounter;
    int gateCode = 0;   // integer code of gate found.
    int gateIndex = -1;  // position of gate in circuit.
    String openfileName;       // name of file to be opened.
```

```
FileDialog fileDialog = new FileDialog(simulator,"OPEN FILE",FileDialog.LOAD);
fileDialog.setDirectory(".");   // keep in same directory.
fileDialog.show();              // display the dialog box.
openfileName = fileDialog.getFile(); // get the file name from the user.
simulator.setTitle(openfileName);    // window title
if(simulator.buildCircuit)  // if in build mode
{
  if(openfileName != null)          // if a file is selected.
  {
    try
    {
      if(logicCircuit.size() > 0)  // clear screen first.
      {
        logicCircuit.removeAllElements();
      }
      BufferedReader inputStream = new BufferedReader(new FileReader(openfileName));
      int numberOfLines = Integer.parseInt(inputStream.readLine()); // find out how many lines are there in the file
opened.
      int xcoord = 0; int ycoord = 0;

      for(int i = 0; i < numberOfLines; ++i) //numberOfLines is the number of gates in saved circuit.
      {
        String string = inputStream.readLine(); // read one line from file opened.
        StringTokenizer stringTokenizer = new StringTokenizer(string);
        //gateCode = Integer.parseInt(stringTokenizer.nextToken());
        gateCode = Integer.parseInt(stringTokenizer.nextToken());
        switch (gateCode)
        {
          case 1:       // AND gate is found.
            gateIndex = Integer.parseInt(stringTokenizer.nextToken());
            xcoord = Integer.parseInt(stringTokenizer.nextToken());
            ycoord = Integer.parseInt(stringTokenizer.nextToken());

            tempGate = new LogicAnd(xcoord,ycoord,gateIndex);
            // check to see if on top of another gate (i.e. grid is already occupied).
            for(andCounter = 0; andCounter < logicCircuit.size() &&
              !((((LogicGate) logicCircuit.elementAt(andCounter)).xCoordinate ==
              tempGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
              (andCounter)).yCoordinate == tempGate.yCoordinate)); ++andCounter);
            if(andCounter == logicCircuit.size())  // if grid is free.
            {
              logicCircuit.addElement(tempGate); // add gate to circuit at this grid
              tempGate = null;
            }
            break;

          case 2:    // OR gate is found.
            gateIndex = Integer.parseInt(stringTokenizer.nextToken());
            xcoord = Integer.parseInt(stringTokenizer.nextToken());
            ycoord = Integer.parseInt(stringTokenizer.nextToken());

            tempGate = new LogicOr(xcoord,ycoord,gateIndex);
            // check to see if on top of another gate (i.e. grid is already occupied).
            for(orCounter = 0; orCounter < logicCircuit.size() &&
              !((((LogicGate) logicCircuit.elementAt(orCounter)).xCoordinate ==
              tempGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
              (orCounter)).yCoordinate == tempGate.yCoordinate)); ++orCounter);
            if(orCounter == logicCircuit.size())  // if grid is free.
            {
              logicCircuit.addElement(tempGate); // add gate to circuit at this grid
              tempGate = null;
            }
  break;

case 3:   // NOT gate is found.
  gateIndex = Integer.parseInt(stringTokenizer.nextToken());
  xcoord = Integer.parseInt(stringTokenizer.nextToken());
  ycoord = Integer.parseInt(stringTokenizer.nextToken());

  tempGate = new LogicNot(xcoord,ycoord,gateIndex);
  // check to see if on top of another gate (i.e. grid is already occupied).
  for(notCounter = 0; notCounter < logicCircuit.size() &&
    !((((LogicGate) logicCircuit.elementAt(notCounter)).xCoordinate ==
    tempGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
    (notCounter)).yCoordinate == tempGate.yCoordinate)); ++notCounter);
  if(notCounter == logicCircuit.size())  // if grid is free.
  {
    logicCircuit.addElement(tempGate); // add gate to circuit at this grid
    tempGate = null;
  }
  break;

case 4:   // XOR gate is found.
  gateIndex = Integer.parseInt(stringTokenizer.nextToken());
  xcoord = Integer.parseInt(stringTokenizer.nextToken());
  ycoord = Integer.parseInt(stringTokenizer.nextToken());

  tempGate = new LogicXor(xcoord,ycoord,gateIndex);
  // check to see if on top of another gate (i.e. grid is already occupied).
  for(xorCounter = 0; xorCounter < logicCircuit.size() &&
    !((((LogicGate) logicCircuit.elementAt(xorCounter)).xCoordinate ==
    tempGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
    (xorCounter)).yCoordinate == tempGate.yCoordinate)); ++xorCounter);
  if(xorCounter == logicCircuit.size())  // if grid is free.
  {
    logicCircuit.addElement(tempGate); // add gate to circuit at this grid
    tempGate = null;
  }
  break;

case 5:   // NAND gate is found.
  gateIndex = Integer.parseInt(stringTokenizer.nextToken());
  xcoord = Integer.parseInt(stringTokenizer.nextToken());
  ycoord = Integer.parseInt(stringTokenizer.nextToken());

  tempGate = new LogicNand(xcoord,ycoord,gateIndex);
  // check to see if on top of another gate (i.e. grid is already occupied).
  for(nandCounter = 0; nandCounter < logicCircuit.size() &&
    !((((LogicGate) logicCircuit.elementAt(nandCounter)).xCoordinate ==
    tempGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
    (nandCounter)).yCoordinate == tempGate.yCoordinate)); ++nandCounter);
  if(nandCounter == logicCircuit.size())  // if grid is free.
  {
    logicCircuit.addElement(tempGate); // add gate to circuit at this grid
    tempGate = null;
  }
  break;

case 6:   // NOR gate is found.
  gateIndex = Integer.parseInt(stringTokenizer.nextToken());
  xcoord = Integer.parseInt(stringTokenizer.nextToken());
  ycoord = Integer.parseInt(stringTokenizer.nextToken());

  tempGate = new LogicNor(xcoord,ycoord,gateIndex);
  // check to see if on top of another gate (i.e. grid is already occupied).
  for(norCounter = 0; norCounter < logicCircuit.size() &&
    !((((LogicGate) logicCircuit.elementAt(norCounter)).xCoordinate ==
    tempGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
    (norCounter)).yCoordinate == tempGate.yCoordinate)); ++norCounter);
  if(norCounter == logicCircuit.size())  // if grid is free.
  {
    logicCircuit.addElement(tempGate); // add gate to circuit at this grid
    tempGate = null;
  }
  break;

case 7:    // CONNECTOR object is found.
  gateIndex = Integer.parseInt(stringTokenizer.nextToken());
  xcoord = Integer.parseInt(stringTokenizer.nextToken());
  ycoord = Integer.parseInt(stringTokenizer.nextToken());

  tempGate = new Connector(xcoord,ycoord,gateIndex);
  // check to see if on top of another gate (i.e. grid is already occupied).
  for(connectorCounter = 0; connectorCounter < logicCircuit.size() &&
    !((((LogicGate) logicCircuit.elementAt(connectorCounter)).xCoordinate ==
    tempGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
    (connectorCounter)).yCoordinate == tempGate.yCoordinate)); ++connectorCounter);
  if(connectorCounter == logicCircuit.size())  // if grid is free.
  {
    logicCircuit.addElement(tempGate); // add gate to circuit at this grid
    tempGate = null;
  }
  break;

case 8:    // INPUT object is found.
  gateIndex = Integer.parseInt(stringTokenizer.nextToken());
  xcoord = Integer.parseInt(stringTokenizer.nextToken());
  ycoord = Integer.parseInt(stringTokenizer.nextToken());

  tempGate = new Input(xcoord,ycoord,gateIndex);
  // check to see if on top of another gate (i.e. grid is already occupied).
  for(inputCounter = 0; inputCounter < logicCircuit.size() &&
    !((((LogicGate) logicCircuit.elementAt(inputCounter)).xCoordinate ==
    tempGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
    (inputCounter)).yCoordinate == tempGate.yCoordinate)); ++inputCounter);
  if(inputCounter == logicCircuit.size())  // if grid is free.
  {
    logicCircuit.addElement(tempGate); // add gate to circuit at this grid
    tempGate = null;
  }
  break;

case 9:    // OUTPUT object is found.
  gateIndex = Integer.parseInt(stringTokenizer.nextToken());
  xcoord = Integer.parseInt(stringTokenizer.nextToken());
  ycoord = Integer.parseInt(stringTokenizer.nextToken());

  tempGate = new Output(xcoord,ycoord,gateIndex);
  // check to see if on top of another gate (i.e. grid is already occupied).
  for(outputCounter = 0; outputCounter < logicCircuit.size() &&
    !((((LogicGate) logicCircuit.elementAt(outputCounter)).xCoordinate ==
    tempGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
    (outputCounter)).yCoordinate == tempGate.yCoordinate)); ++outputCounter);
  if(outputCounter == logicCircuit.size())  // if grid is free.
  {
    logicCircuit.addElement(tempGate); // add gate to circuit at this grid
```

```
                tempGate = null;
              }
              break;

            default:
              break;
          }// end switch (myString)
        }// end for
        inputStream.close();
        readFileAgain(openfileName);
      }// end try
      catch(IOException e)
      {
        System.out.print("Error: " + e);
      }
    }// end if(openfileName != null)
  }// end if(simulator.buildCircuit)
  repaint();
}// end doOpenFile

public void readFileAgain(String openfileName)
//** This method reads information from a file in order to reconstruct a
//** digital circuit that has been saved.
{
  int gateCode = 0;    // integer code of gate found.
  int gateIndex = -1;  // position of gate in circuit.

  try
  {
    BufferedReader inputStream2 = new BufferedReader(new FileReader(openfileName));
    int numberOfLines = Integer.parseInt(inputStream2.readLine()); // find out how many lines are there in the file
opened.
    int xcoord = 0; int ycoord = 0; int top = 0; int bottom = 0;
    int topconn = 0; int bottomconn = 0; int out = 0;

    for(int i = 0; i < numberOfLines; ++i) //numberOfLines is the number of gates in saved circuit.
    {
      String string2 = inputStream2.readLine(); // read one line from file opened.
      StringTokenizer stringTokenizer2 = new StringTokenizer(string2);
      gateCode = Integer.parseInt(stringTokenizer2.nextToken());
      switch (gateCode)
      {
        case 1:       // AND gate is found.
          LogicAnd tempGate1 = ((LogicAnd) logicCircuit.elementAt(i)) ;
          while (stringTokenizer2.hasMoreTokens())  //get all info. for this gate..
          {
            gateIndex = Integer.parseInt(stringTokenizer2.nextToken());
            xcoord = Integer.parseInt(stringTokenizer2.nextToken());
            ycoord = Integer.parseInt(stringTokenizer2.nextToken());
            top = Integer.parseInt(stringTokenizer2.nextToken());
            topconn = Integer.parseInt(stringTokenizer2.nextToken());
            bottom = Integer.parseInt(stringTokenizer2.nextToken());
            bottomconn = Integer.parseInt(stringTokenizer2.nextToken());
            out = Integer.parseInt(stringTokenizer2.nextToken());
          }// end while

          if(top == 1)
          {
            tempGate1.topInputPin = true;
          }
          else
          {
```

```
      tempGate1.topInputPin = false;
    }

    if(bottom == 1)
    {
      tempGate1.bottomInputPin = true;
    }
    else
    {
      tempGate1.bottomInputPin = false;
    }

    if(out == 1)
    {
      tempGate1.OutputPin = true;
    }
    else
    {
      tempGate1.OutputPin = false;
    }

    if(topconn != -1)
    {
      whichInputPin = 1;
      theDestinationGate = tempGate1;
      theOriginatingGate = ((LogicGate) logicCircuit.elementAt(topconn));
      propagateState();
    }// end if(topconn != -1)

    if(bottomconn != -1)
    {
      whichInputPin = 2;
      theDestinationGate = tempGate1;
      theOriginatingGate = ((LogicGate) logicCircuit.elementAt(bottomconn));
      propagateState();
    }// end if(bottomconn != -1)
    tempGate1 = null;
    break;

  case 2:     // OR gate is found.
    LogicOr tempGate2 = ((LogicOr) logicCircuit.elementAt(i)) ;
    while (stringTokenizer2.hasMoreTokens())  //get all info. for this gate..
    {
      gateIndex = Integer.parseInt(stringTokenizer2.nextToken());
      xcoord = Integer.parseInt(stringTokenizer2.nextToken());
      ycoord = Integer.parseInt(stringTokenizer2.nextToken());
      top = Integer.parseInt(stringTokenizer2.nextToken());
      topconn = Integer.parseInt(stringTokenizer2.nextToken());
      bottom = Integer.parseInt(stringTokenizer2.nextToken());
      bottomconn = Integer.parseInt(stringTokenizer2.nextToken());
      out = Integer.parseInt(stringTokenizer2.nextToken());
    }// end while

    if(top == 1)
    {
      tempGate2.topInputPin = true;
    }
    else
    {
      tempGate2.topInputPin = false;
    }
```

```
   if(bottom == 1)
   {
     tempGate2.bottomInputPin = true;
   }
   else
   {
     tempGate2.bottomInputPin = false;
   }

   if(out == 1)
   {
     tempGate2.OutputPin = true;
   }
   else
   {
     tempGate2.OutputPin = false;
   }

   if(topconn != -1)
   {
     whichInputPin = 1;
     theDestinationGate = tempGate2;
     theOriginatingGate = ((LogicGate) logicCircuit.elementAt(topconn));
     propagateState();
   }// end if(topconn != -1)

   if(bottomconn != -1)
   {
     whichInputPin = 2;
     theDestinationGate = tempGate2;
     theOriginatingGate = ((LogicGate) logicCircuit.elementAt(bottomconn));
     propagateState();
   }// end if(bottomconn != -1)
   tempGate2 = null;
   break;

 case 3:    // NOT gate is found.
   LogicNot tempGate3 = ((LogicNot) logicCircuit.elementAt(i)) ;
   while (stringTokenizer2.hasMoreTokens()) //get all info. for this gate..
   {
     gateIndex = Integer.parseInt(stringTokenizer2.nextToken());
     xcoord = Integer.parseInt(stringTokenizer2.nextToken());
     ycoord = Integer.parseInt(stringTokenizer2.nextToken());
     top = Integer.parseInt(stringTokenizer2.nextToken());
     topconn = Integer.parseInt(stringTokenizer2.nextToken());
     bottom = Integer.parseInt(stringTokenizer2.nextToken());
     bottomconn = Integer.parseInt(stringTokenizer2.nextToken());
     out = Integer.parseInt(stringTokenizer2.nextToken());
   }// end while

   if(top == 1)
   {
     tempGate3.topInputPin = true;
   }
   else
   {
     tempGate3.topInputPin = false;
   }

   if(bottom == 1)
   {
     tempGate3.bottomInputPin = true;
```

```
  }
  else
  {
    tempGate3.bottomInputPin = false;
  }

  if(out == 1)
  {
    tempGate3.OutputPin = true;
  }
  else
  {
    tempGate3.OutputPin = false;
  }

  if(topconn != -1)
  {
    whichInputPin = 1;
    theDestinationGate = tempGate3;
    theOriginatingGate = ((LogicGate) logicCircuit.elementAt(topconn));
    propagateState();
  }// end if(topconn != -1)

  if(bottomconn != -1)
  {
    whichInputPin = 2;
    theDestinationGate = tempGate3;
    theOriginatingGate = ((LogicGate) logicCircuit.elementAt(bottomconn));
    propagateState();
  }// end if(bottomconn != -1)
  tempGate3 = null;
  break;

case 4:    // XOR gate is found.
  LogicXor tempGate4 = ((LogicXor) logicCircuit.elementAt(i)) ;
  while (stringTokenizer2.hasMoreTokens()) //get all info. for this gate..
  {
    gateIndex = Integer.parseInt(stringTokenizer2.nextToken());
    xcoord = Integer.parseInt(stringTokenizer2.nextToken());
    ycoord = Integer.parseInt(stringTokenizer2.nextToken());
    top = Integer.parseInt(stringTokenizer2.nextToken());
    topconn = Integer.parseInt(stringTokenizer2.nextToken());
    bottom = Integer.parseInt(stringTokenizer2.nextToken());
    bottomconn = Integer.parseInt(stringTokenizer2.nextToken());
    out = Integer.parseInt(stringTokenizer2.nextToken());
  }// end while

  if(top == 1)
  {
    tempGate4.topInputPin = true;
  }
  else
  {
    tempGate4.topInputPin = false;
  }

  if(bottom == 1)
  {
    tempGate4.bottomInputPin = true;
  }
  else
  {
      tempGate4.bottomInputPin = false;
    }

    if(out == 1)
    {
      tempGate4.OutputPin = true;
    }
    else
    {
      tempGate4.OutputPin = false;
    }

    if(topconn != -1)
    {
      whichInputPin = 1;
      theDestinationGate = tempGate4;
      theOriginatingGate = ((LogicGate) logicCircuit.elementAt(topconn));
      propagateState();
    }// end if(topconn != -1)

    if(bottomconn != -1)
    {
      whichInputPin = 2;
      theDestinationGate = tempGate4;
      theOriginatingGate = ((LogicGate) logicCircuit.elementAt(bottomconn));
      propagateState();
    }// end if(bottomconn != -1)
    tempGate4 = null;
    break;

  case 5:    // NAND gate is found.
    LogicNand tempGate5 = ((LogicNand) logicCircuit.elementAt(i)) ;
    while (stringTokenizer2.hasMoreTokens())  //get all info. for this gate..
    {
      gateIndex = Integer.parseInt(stringTokenizer2.nextToken());
      xcoord = Integer.parseInt(stringTokenizer2.nextToken());
      ycoord = Integer.parseInt(stringTokenizer2.nextToken());
      top = Integer.parseInt(stringTokenizer2.nextToken());
      topconn = Integer.parseInt(stringTokenizer2.nextToken());
      bottom = Integer.parseInt(stringTokenizer2.nextToken());
      bottomconn = Integer.parseInt(stringTokenizer2.nextToken());
      out = Integer.parseInt(stringTokenizer2.nextToken());
    }// end while

    if(top == 1)
    {
      tempGate5.topInputPin = true;
    }
    else
    {
      tempGate5.topInputPin = false;
    }

    if(bottom == 1)
    {
      tempGate5.bottomInputPin = true;
    }
    else
    {
      tempGate5.bottomInputPin = false;
    }
    if(out == 1)
    {
      tempGate5.OutputPin = true;
    }
    else
    {
      tempGate5.OutputPin = false;
    }

    if(topconn != -1)
    {
      whichInputPin = 1;
      theDestinationGate = tempGate5;
      theOriginatingGate = ((LogicGate) logicCircuit.elementAt(topconn));
      propagateState();
    }// end if(topconn != -1)

    if(bottomconn != -1)
    {
      whichInputPin = 2;
      theDestinationGate = tempGate5;
      theOriginatingGate = ((LogicGate) logicCircuit.elementAt(bottomconn));
      propagateState();
    }// end if(bottomconn != -1)
    tempGate5 = null;
    break;

  case 6:    // NOR gate is found.
    LogicNor tempGate6 = ((LogicNor) logicCircuit.elementAt(i)) ;
    while (stringTokenizer2.hasMoreTokens())  //get all info. for this gate..
    {
      gateIndex = Integer.parseInt(stringTokenizer2.nextToken());
      xcoord = Integer.parseInt(stringTokenizer2.nextToken());
      ycoord = Integer.parseInt(stringTokenizer2.nextToken());
      top = Integer.parseInt(stringTokenizer2.nextToken());
      topconn = Integer.parseInt(stringTokenizer2.nextToken());
      bottom = Integer.parseInt(stringTokenizer2.nextToken());
      bottomconn = Integer.parseInt(stringTokenizer2.nextToken());
      out = Integer.parseInt(stringTokenizer2.nextToken());
    }// end while

    if(top == 1)
    {
      tempGate6.topInputPin = true;
    }
    else
    {
      tempGate6.topInputPin = false;
    }

    if(bottom == 1)
    {
      tempGate6.bottomInputPin = true;
    }
    else
    {
      tempGate6.bottomInputPin = false;
    }

    if(out == 1)
    {
      tempGate6.OutputPin = true;
  }
  else
  {
    tempGate6.OutputPin = false;
  }

  if(topconn != -1)
  {
    whichInputPin = 1;
    theDestinationGate = tempGate6;
    theOriginatingGate = ((LogicGate) logicCircuit.elementAt(topconn));
    propagateState();
  }// end if(topconn != -1)

  if(bottomconn != -1)
  {
    whichInputPin = 2;
    theDestinationGate = tempGate6;
    theOriginatingGate = ((LogicGate) logicCircuit.elementAt(bottomconn));
    propagateState();
  }// end if(bottomconn != -1)
  tempGate6 = null;
  break;

case 7:    // CONNECTOR object is found.
  Connector tempGate7 = ((Connector) logicCircuit.elementAt(i)) ;
  while (stringTokenizer2.hasMoreTokens())  //get all info. for this gate..
  {
    gateIndex = Integer.parseInt(stringTokenizer2.nextToken());
    xcoord = Integer.parseInt(stringTokenizer2.nextToken());
    ycoord = Integer.parseInt(stringTokenizer2.nextToken());
    top = Integer.parseInt(stringTokenizer2.nextToken());
    topconn = Integer.parseInt(stringTokenizer2.nextToken());
    bottom = Integer.parseInt(stringTokenizer2.nextToken());
    bottomconn = Integer.parseInt(stringTokenizer2.nextToken());
    out = Integer.parseInt(stringTokenizer2.nextToken());
  }// end while

  if(top == 1)
  {
    tempGate7.topInputPin = true;
  }
  else
  {
    tempGate7.topInputPin = false;
  }

  if(bottom == 1)
  {
    tempGate7.bottomInputPin = true;
  }
  else
  {
    tempGate7.bottomInputPin = false;
  }

  if(out == 1)
  {
    tempGate7.OutputPin = true;
  }
  else
  {
    tempGate7.OutputPin = false;
  }

  if(topconn != -1)
  {
    whichInputPin = 1;
    theDestinationGate = tempGate7;
    theOriginatingGate = ((LogicGate) logicCircuit.elementAt(topconn));
    propagateState();
  }// end if(topconn != -1)

  if(bottomconn != -1)
  {
    whichInputPin = 2;
    theDestinationGate = tempGate7;
    theOriginatingGate = ((LogicGate) logicCircuit.elementAt(bottomconn));
    propagateState();
  }// end if(bottomconn != -1)
  tempGate7 = null;
  break;

case 8:     // INPUT object gate is found.
  Input tempGate8 = ((Input) logicCircuit.elementAt(i)) ;
  while (stringTokenizer2.hasMoreTokens())  //get all info. for this gate..
  {
    gateIndex = Integer.parseInt(stringTokenizer2.nextToken());
    xcoord = Integer.parseInt(stringTokenizer2.nextToken());
    ycoord = Integer.parseInt(stringTokenizer2.nextToken());
    top = Integer.parseInt(stringTokenizer2.nextToken());
    topconn = Integer.parseInt(stringTokenizer2.nextToken());
    bottom = Integer.parseInt(stringTokenizer2.nextToken());
    bottomconn = Integer.parseInt(stringTokenizer2.nextToken());
    out = Integer.parseInt(stringTokenizer2.nextToken());
  }// end while

  if(top == 1)
  {
    tempGate8.topInputPin = true;
  }
  else
  {
    tempGate8.topInputPin = false;
  }

  if(bottom == 1)
  {
    tempGate8.bottomInputPin = true;
  }
  else
  {
    tempGate8.bottomInputPin = false;
  }

  if(out == 1)
  {
    tempGate8.OutputPin = true;
  }
  else
  {
    tempGate8.OutputPin = false;
  }
  if(topconn != -1)
  {
    whichInputPin = 1;
    theDestinationGate = tempGate8;
    theOriginatingGate = ((LogicGate) logicCircuit.elementAt(topconn));
    propagateState();
  }// end if(topconn != -1)

  if(bottomconn != -1)
  {
    whichInputPin = 2;
    theDestinationGate = tempGate8;
    theOriginatingGate = ((LogicGate) logicCircuit.elementAt(bottomconn));
    propagateState();
  }// end if(bottomconn != -1)
  tempGate8 = null;
  break;

case 9:    // OUTPUT object gate is found.
  Output tempGate9 = ((Output) logicCircuit.elementAt(i)) ;
  while (stringTokenizer2.hasMoreTokens())  //get all info. for this gate..
  {
    gateIndex = Integer.parseInt(stringTokenizer2.nextToken());
    xcoord = Integer.parseInt(stringTokenizer2.nextToken());
    ycoord = Integer.parseInt(stringTokenizer2.nextToken());
    top = Integer.parseInt(stringTokenizer2.nextToken());
    topconn = Integer.parseInt(stringTokenizer2.nextToken());
    bottom = Integer.parseInt(stringTokenizer2.nextToken());
    bottomconn = Integer.parseInt(stringTokenizer2.nextToken());
    out = Integer.parseInt(stringTokenizer2.nextToken());
  }// end while

  if(top == 1)
  {
    tempGate9.topInputPin = true;
  }
  else
  {
    tempGate9.topInputPin = false;
  }

  if(bottom == 1)
  {
    tempGate9.bottomInputPin = true;
  }
  else
  {
    tempGate9.bottomInputPin = false;
  }

  if(out == 1)
  {
    tempGate9.OutputPin = true;
  }
  else
  {
    tempGate9.OutputPin = false;
  }

  if(topconn != -1)
  {
    whichInputPin = 1;
```

```
          theDestinationGate = tempGate9;
          theOriginatingGate = ((LogicGate) logicCircuit.elementAt(topconn));
          propagateState();
        }// end if(topconn != -1)

        if(bottomconn != -1)
        {
          whichInputPin = 2;
          theDestinationGate = tempGate9;
          theOriginatingGate = ((LogicGate) logicCircuit.elementAt(bottomconn));
          propagateState();
        }// end if(bottomconn != -1)
        tempGate9 = null;
        break;

      default:
        break;
    }// end switch (myString)
  }// end for
  inputStream2.close();
 }// end try
 catch(IOException e)
 {
  System.out.print("Error: " + e);
 }
}// end readFileAgain

public int returnEncodedGate(int gateNumber)
//** Given a gate number (position in array), this method returns a code for
//** the gate being clicked (if valid), -1 otherwise to indicate a null value.
 {
  int gateCode = -1;
  String myString = null;
  String gateString = String.valueOf(((LogicGate) logicCircuit.elementAt(gateNumber)));
  StringTokenizer stringTokenizer = new StringTokenizer(gateString,"@");
  myString = stringTokenizer.nextToken();

  if(myString.compareTo("LogicAnd") == 0)
  {
    gateCode = 1;    // LogicAnd gate has code 1.
  }
  else if(myString.compareTo("LogicOr") == 0)
  {
    gateCode = 2;    // LogicOr gate has code 2.
  }
  else if(myString.compareTo("LogicNot") == 0)
  {
    gateCode = 3;    // LogicNot gate has code 3.
  }
  else if(myString.compareTo("LogicXor") == 0)
  {
    gateCode = 4;    // LogicXor gate has code 4.
  }
  else if(myString.compareTo("LogicNand") == 0)
  {
    gateCode = 5;     // LogicNand gate has code 5.
  }
  else if(myString.compareTo("LogicNor") == 0)
  {
    gateCode = 6;     // LogicNor gate has code 6.
  }
  else if(myString.compareTo("Connector") == 0)
```

```
    {
      gateCode = 7;      // Connector gate has code 7.
    }
    else if(myString.compareTo("Input") == 0)
    {
      gateCode = 8;      // Input gate has code 8.
    }
    else if(myString.compareTo("Output") == 0)
    {
      gateCode = 9;      // Output gate has code 9.
    }
    else
    {
      gateCode = -1;     // else -1 indicates a null value.
    }
    return gateCode;
  }// end returnEncodedGate


  public void doSaveFile()
//** This method saves a digital logic circuit into a file to be retrieved at
//** a later time. The circuit's total environment is saved in the order
//** it was created.
  {
    int topPin = 1;    // digit value of top input pin.
    int bottomPin = 2; // digit value of bottom input pin.
    int outputPin = 3; // digit value of output pin.
    int encodedGate = -1; // integer code of gate (initially set to null).
    String savefileName;

    FileDialog fileDialog = new FileDialog(simulator,"SAVE FILE",FileDialog.SAVE);
    fileDialog.setDirectory("."); // set file to current directory.
    fileDialog.show();            // display the dialog box.
    savefileName = fileDialog.getFile(); // get name of file from the user.
    simulator.setTitle(savefileName);      // window title
    if(savefileName != null)  // if file was found.
    {
      LogicGate tempGate = null;
      try
      {
        PrintWriter outputStream = new PrintWriter(new FileOutputStream(savefileName));
        outputStream.print(logicCircuit.size()); //save number of gates in circuit.
        outputStream.print("\n");

        for(int i = 0; i < logicCircuit.size(); ++i) // for all the gates in a circuit.
        {
          tempGate = ((LogicGate) logicCircuit.elementAt(i)); // find the gate.
          encodedGate = returnEncodedGate(i);   // return this gate's code number.
          outputStream.print(encodedGate); //save the type of gate found.
          outputStream.print(" ");

          outputStream.print(i);    // save gate position as an index.
          outputStream.print(" ");

          outputStream.print(tempGate.xCoordinate); //save gate's x coordinate.
          outputStream.print(" ");

          outputStream.print(tempGate.yCoordinate);//save gate's y coordinate.
          outputStream.print(" ");

          if(tempGate.getInputState(topPin)) // if gate's top pin is high.
          {
```

```
          outputStream.print(1);    //save gate's top pin state as 1.
        }
        else
        {
          outputStream.print(0); //else save gate's top pin state as 0.
        }
        outputStream.print(" ");

        if(tempGate.topPinConnection != null)
        {
          outputStream.print(tempGate.topPinConnection.gateIndex);
        }
        else
        {
          outputStream.print(-1);
        }
        outputStream.print(" ");

        if(tempGate.getInputState(bottomPin)) // if gate's bottom pin is high.
        {
          outputStream.print(1); //save gate's bottom pin state as 1.
        }
        else
        {
          outputStream.print(0); //else save gate's bottom pin state as 0.
        }
        outputStream.print(" ");

        if(tempGate.bottomPinConnection != null)
        {
          outputStream.print(tempGate.bottomPinConnection.gateIndex);
        }
        else
        {
          outputStream.print(-1);
        }
        outputStream.print(" ");

        if(tempGate.getOutputState()) // if gate's output pin is high.
        {
          outputStream.print(1); //save gate's output pin state as 1.
        }
        else
        {
          outputStream.print(0); // else save gate's output pin state as 0.
        }
        outputStream.print("\n");
      }// end for

    outputStream.close();
    }// end try
    catch(IOException e)
    {
      System.out.print("Error: " + e);
    }
  }// end if(savefileName != null)
}// end doSaveFile

public void mouseClicked(MouseEvent e)
//** this method is called every time the mouse is clicked. The simulator
//** is totally run by mouse events.
{
```

```
int andCounter, orCounter, notCounter, xorCounter, nandCounter,
  norCounter, connectorCounter, inputCounter, outputCounter;

if(mouseClickPoint == null)  // get mouse click position.
{
  mouseClickPoint = new Point(e.getPoint());
}
else
{
  mouseClickPoint.x = e.getX();
  mouseClickPoint.y = e.getY();
}

if(simulator.buildCircuit)  // if in build mode
{
  if((currentState == createGate) || (currentState == connectGate))
  {
    if(currentGate > 0)  // if any gate is selected (mouse clicked in gate menu).
    {
      switch (currentGate)   // which gate is clicked.
      {
        case and:        // if and gate is selected, create it
          logicGate = new LogicAnd(mouseClickPoint.x, mouseClickPoint.y, logicCircuit.size());
          // check to see if on top of another gate (i.e. grid is already occupied).
          for(andCounter = 0; andCounter < logicCircuit.size() &&
            !((((LogicGate) logicCircuit.elementAt(andCounter)).xCoordinate ==
            logicGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
            (andCounter)).yCoordinate == logicGate.yCoordinate)); ++andCounter);
          if(andCounter == logicCircuit.size())  // if grid is free.
          {
            logicCircuit.addElement(logicGate); // add gate to circuit at this grid
            logicGate = null;          // deselect gate
          }
          currentGate = 0;      // deselect button clicked
          //theDestinationGate = null; // deselect the receiving gate.
          //theOriginatingGate = null;  // deselect the sending gate.
          break;

        case or:
          logicGate = new LogicOr(mouseClickPoint.x, mouseClickPoint.y, logicCircuit.size());
          // go through all the gates in the circuit.
          for(orCounter = 0; orCounter < logicCircuit.size() &&
            !((((LogicGate) logicCircuit.elementAt(orCounter)).xCoordinate ==
            logicGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
            (orCounter)).yCoordinate == logicGate.yCoordinate)); ++orCounter);
          if(orCounter == logicCircuit.size())
          {
            logicCircuit.addElement(logicGate);
            logicGate = null;
          }
          currentGate = 0;
          //theDestinationGate = null;
         // theOriginatingGate = null;
          break;

        case not:
          logicGate = new LogicNot(mouseClickPoint.x, mouseClickPoint.y, logicCircuit.size());
          for(notCounter = 0; notCounter < logicCircuit.size() &&
            !((((LogicGate) logicCircuit.elementAt(notCounter)).xCoordinate ==
            logicGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
            (notCounter)).yCoordinate == logicGate.yCoordinate)); ++notCounter);
          if(notCounter == logicCircuit.size())
    {
      logicCircuit.addElement(logicGate);
      logicGate = null;
    }
    currentGate = 0;
    theDestinationGate = null;
    theOriginatingGate = null;
    break;

  case xor:
    logicGate = new LogicXor(mouseClickPoint.x, mouseClickPoint.y, logicCircuit.size());
    for(xorCounter = 0; xorCounter < logicCircuit.size() &&
      !((((LogicGate) logicCircuit.elementAt(xorCounter)).xCoordinate ==
      logicGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
      (xorCounter)).yCoordinate == logicGate.yCoordinate)); ++xorCounter);
    if(xorCounter == logicCircuit.size())
    {
      logicCircuit.addElement(logicGate);
      logicGate = null;
    }
    currentGate = 0;
    theDestinationGate = null;
    theOriginatingGate = null;
    break;

  case nand:
    logicGate = new LogicNand(mouseClickPoint.x, mouseClickPoint.y, logicCircuit.size());
    for(nandCounter = 0; nandCounter < logicCircuit.size() &&
      !((((LogicGate) logicCircuit.elementAt(nandCounter)).xCoordinate ==
      logicGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
      (nandCounter)).yCoordinate == logicGate.yCoordinate)); ++nandCounter);
    if(nandCounter == logicCircuit.size())
    {
      logicCircuit.addElement(logicGate);
      logicGate = null;
    }
    currentGate = 0;
    theDestinationGate = null;
    theOriginatingGate = null;
    break;

  case nor:
    logicGate = new LogicNor(mouseClickPoint.x, mouseClickPoint.y, logicCircuit.size());
    for(norCounter = 0; norCounter < logicCircuit.size() &&
      !((((LogicGate) logicCircuit.elementAt(norCounter)).xCoordinate ==
      logicGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
      (norCounter)).yCoordinate == logicGate.yCoordinate)); ++norCounter);
    if(norCounter == logicCircuit.size())
    {
      logicCircuit.addElement(logicGate);
      logicGate = null;
    }
    currentGate = 0;
    theDestinationGate = null;
    theOriginatingGate = null;
    break;

  case connector:       // if connector object is selected, create it
    logicGate = new Connector(mouseClickPoint.x, mouseClickPoint.y, logicCircuit.size());
    // check to see if on top of another gate (i.e. grid is already occupied).
    for(connectorCounter = 0; connectorCounter < logicCircuit.size() &&
      !((((LogicGate) logicCircuit.elementAt(connectorCounter)).xCoordinate ==
          logicGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
          (connectorCounter)).yCoordinate == logicGate.yCoordinate)); ++connectorCounter);
        if(connectorCounter == logicCircuit.size())  // if grid is free.
        {
          logicCircuit.addElement(logicGate); // add gate to circuit at this grid
          logicGate = null;              // deselect gate
        }
        currentGate = 0;      // deselect button clicked
        theDestinationGate = null; // deselect the receiving gate.
        theOriginatingGate = null;  // deselect the sending gate.
        break;

      case input:          // if input object is selected, create it
        logicGate = new Input(mouseClickPoint.x, mouseClickPoint.y, logicCircuit.size());
        // check to see if on top of another gate (i.e. grid is already occupied).
        for(inputCounter = 0; inputCounter < logicCircuit.size() &&
          !((((LogicGate) logicCircuit.elementAt(inputCounter)).xCoordinate ==
          logicGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
          (inputCounter)).yCoordinate == logicGate.yCoordinate)); ++inputCounter);
        if(inputCounter == logicCircuit.size())  // if grid is free.
        {
          logicCircuit.addElement(logicGate); // add gate to circuit at this grid
          logicGate = null;              // deselect gate
        }
        currentGate = 0;      // deselect button clicked
        theDestinationGate = null; // deselect the receiving gate.
        theOriginatingGate = null;  // deselect the sending gate.
        break;

      case output:          // if output object is selected, create it
        logicGate = new Output(mouseClickPoint.x, mouseClickPoint.y, logicCircuit.size());
        // check to see if on top of another gate (i.e. grid is already occupied).
        for(outputCounter = 0; outputCounter < logicCircuit.size() &&
          !((((LogicGate) logicCircuit.elementAt(outputCounter)).xCoordinate ==
          logicGate.xCoordinate) && (((LogicGate) logicCircuit.elementAt
          (outputCounter)).yCoordinate == logicGate.yCoordinate)); ++outputCounter);
        if(outputCounter == logicCircuit.size())  // if grid is free.
        {
          logicCircuit.addElement(logicGate); // add gate to circuit at this grid
          logicGate = null;              // deselect gate
        }
        currentGate = 0;      // deselect button clicked
        theDestinationGate = null; // deselect the receiving gate.
        theOriginatingGate = null;  // deselect the sending gate.
        break;

      default :
        break;

    }// end switch (currentGate)
  }// end if(currentGate > 0)
  else if(logicCircuit.size() > 0) // else if connecting gates (mouse clicked in canvas).
  {
    doConnection();
  }// end if(logicCircuit.size() > 0)
}// end if((currentState == createGate) || (currentState == connectGate))

else if(currentState == eraseGate) // if erase button is clicked.
{
  deleteGate();      // delete the selected gate.
}// end else if(currentState == eraseGate)
repaint();
```

```
    }// end if(simulator.buildCircuit)

    else          // else run the simulation
    {
      runSimulation();
    }// end else if(simulator.buildCircuit)
  }// end mouseClicked

  public void deleteGate()
//** This method when invoked on a gate, erases this gate from the canvas
//** and deletes it also from the data structure. Then it refreshes the screen.
  {
    LogicGate theGateJustErased = null;
    LogicGate theGateConnected = null;
    LogicGate tempGate = null;
    int topIndex = -1;  // position of the gate connected to this gate's top input pin is set to null.
    int bottomIndex = -1;  //position of the gate connected to this gate's bottom input pin is set to null.
    int deletedGateIndex = -1;
    final int topPin = 1;     // digit value of top input pin.
    final int bottomPin = 2;  // digit value of bottom input pin.

    // go through all the gates in the circuit.
    for(int counter = 0; counter < logicCircuit.size(); ++counter)
    {
      // find out the gate just clicked.
      theGateJustErased = ((LogicGate) logicCircuit.elementAt(counter)).whichGate(mouseClickPoint);
      if(theGateJustErased != null)    // if gate is found.
      {
        deletedGateIndex = theGateJustErased.gateIndex;
        if(theGateJustErased.topPinConnection != null)
        {
          topIndex = theGateJustErased.topPinConnection.gateIndex;
          theGateConnected = ((LogicGate) logicCircuit.elementAt(topIndex));
          theGateConnected.connectionVector.removeElement(theGateJustErased);
          theGateConnected = null;
        }// end if(theGateJustErased.topPinConnection != null)

        if(theGateJustErased.bottomPinConnection != null)
        {
          bottomIndex = theGateJustErased.bottomPinConnection.gateIndex;
          theGateConnected = ((LogicGate) logicCircuit.elementAt(bottomIndex));
          theGateConnected.connectionVector.removeElement(theGateJustErased);
          theGateConnected = null;
        }// end if(theGateJustErased.bottomPinConnection != null)
        theGateJustErased.connectionVector.removeAllElements();
        logicCircuit.removeElement(theGateJustErased);
        theGateJustErased = null;
      }// end if(theGateJustErased != null)
    }// end for

    // the following loop is designed to sweep through a circuit and
    // reset any tempGate's information that has been changed after deletion
    // of a gate connected to this tempGate.
    for(int count = 0; count < logicCircuit.size(); ++count)
    {
      // go through all gates in circuit.
      tempGate = ((LogicGate) logicCircuit.elementAt(count));
      if(tempGate.topPinConnection != null) // if top pin is connected.
      {
        // and the gate deleted is the one connected to this tempGate.
        if(tempGate.topPinConnection.gateIndex == deletedGateIndex)
        {
            tempGate.topPinConnection = null; // free tempGate's top pin.
            if(tempGate.getInputState(topPin)) // if pin value is high.
            {
              tempGate.toggleInput(topPin); //reset pin value to low.
            }
          }
        }// end if(theOutputGateConnected.topPinConnection != null)

        if(tempGate.bottomPinConnection != null) // same case for bottom pin.
        {
          if(tempGate.bottomPinConnection.gateIndex == deletedGateIndex)
          {
            tempGate.bottomPinConnection = null;
            if(tempGate.getInputState(bottomPin))
            {
              tempGate.toggleInput(bottomPin);
            }
          }
        }// end if(tempGate.bottomPinConnection != null)
      }// end for

      // the following loop is designed to sweep through a circuit and
      // decrement all gates indices with a value higher than the gate deleted.   // of a gate connected to this tempGate.
      for(int index = 0; index < logicCircuit.size(); ++index)
      {
        tempGate = ((LogicGate) logicCircuit.elementAt(index));
        tempGate.gateIndex = index;
      }
      currentState = createGate;   // return to pointing mode.
      repaint();  // refresh screen display after deletion.
    }// end deleteGate

    public void doConnection()
  //** This method connects two gates together.
  {
    LogicGate theGateClicked = null;

    // go through all the gates in the circuit.
    for(int counter = 0; counter < logicCircuit.size(); ++counter)
    {
      if(theGateClicked == null)
      {
        // find the gate clicked.
        theGateClicked = ((LogicGate) logicCircuit.elementAt(counter)).whichGate(mouseClickPoint);
      }
    }// end for

    if(theGateClicked != null)
    {
      final int topPin = 1;     // digit value of top input pin.
      final int bottomPin = 2;  // digit value of bottom input pin.
      final int outputPin = 3;  // digit value of output pin.

      // find the part of the gate that is clicked.
      int tempGatePart = theGateClicked.whichGatePart(mouseClickPoint);

      switch (tempGatePart)
      {
        case topPin:        // if the top input pin is selected.
          // if this input pin is free (not connected)
          if((theGateClicked.topPinConnection == null)
            // and not connecting an input pin to another.
          && (theDestinationGate == null))
        {
          // set the gate clicked as the receiving gate.
          theDestinationGate = theGateClicked;
          whichInputPin = topPin; // pin selected is the top pin.
          // if no other gate is selected as the sending gate.
          if(theOriginatingGate == null)
          {
            currentState = connectGate; // wait to select another gate
          }
          else      // else there are two gates to be connected together.
          {
            propagateState();  // so, do the connection.
          }
        }
        break;

        case bottomPin:  // same conditions as above.
          if((theGateClicked.bottomPinConnection == null)
            && (theDestinationGate == null))
          {
            theDestinationGate = theGateClicked;
            whichInputPin = bottomPin;
            if(theOriginatingGate == null)
            {
              currentState = connectGate;
            }
            else
            {
              propagateState();
            }
          }
          break;

        case outputPin:
          //if this output pin is free (not connected)
          if(theOriginatingGate == null)
          {
            // set the gate clicked as the gate sending the output.
            theOriginatingGate = theGateClicked;
            // if no other gate is selected as the receiving gate.
            if(theDestinationGate == null)
            {
              currentState = connectGate; // wait to select another gate
            }
            else   // else there are two gates to be connected together.
            {
              propagateState();
            }
          }
          break;

        default :
          theOriginatingGate = null;    // handles a dangling wire.
          theDestinationGate = null;
          break;

    }// end switch(tempGatePart)
  }// end if(theGateClicked != null)
}// end doConnection

  public void initializeCircuit()
```

```
//** This method sets all inputs in a circuit initially to low and
//** propagates the results through out the whole circuit.
  {
    int topPin = 1;    // digit value of top input pin.
    int bottomPin = 2;  // digit value of bottom input pin.
    LogicGate tempGate = null;

    for(int count = 0; count < logicCircuit.size(); ++count)
    {
      tempGate = ((LogicGate) logicCircuit.elementAt(count));
      if(tempGate.topPinConnection != null)
      {
        if(tempGate.getInputState(topPin) != tempGate.topPinConnection.getOutputState())
        {
          // recalculation is simply done by toggling the input pin of the receiving gate.
          ((LogicGate) logicCircuit.elementAt(count)).toggleInput(topPin);
        }
      }// end if
      if(tempGate.bottomPinConnection != null)
      {
        if(tempGate.getInputState(bottomPin) != tempGate.bottomPinConnection.getOutputState())
        {
          // recalculation is simply done by toggling the input pin of the receiving gate.
          ((LogicGate) logicCircuit.elementAt(count)).toggleInput(bottomPin);
        }
      }// end if
    }// end for
    repaint(); // refresh screen display.
  }// end initializeCircuit

  public void runSimulation()
//** This method runs the simulation part if the function button is clicked
//** to simulate mode.
  {
    currentState = createGate; // deselect any menu buttons clicked before simulation.
    LogicGate temp = null;
    final int topPin = 1;    // digit value of top input pin.

    // go through all the gates in the circuit.
    for(int counter = 0; counter < logicCircuit.size(); ++counter)
    {
      temp = ((LogicGate) logicCircuit.elementAt(counter));
      if(temp.isInputObject)
      {
        if((mouseClickPoint.x > temp.xCoordinate - 15)&& (mouseClickPoint.x < temp.xCoordinate + 5)
         && (mouseClickPoint.y > temp.yCoordinate - 7) && (mouseClickPoint.y < temp.yCoordinate + 10))
        {
// toggle the state of this pin, this will cause recalculation of the gate's function.
          ((LogicGate) logicCircuit.elementAt(counter)).toggleInput(topPin);
          repaint();
        }
      }
    }// end for
    initializeCircuit();
  }// end runSimulation

  public void mouseReleased(MouseEvent e)
//** this method is called every time the mouse is released.
  {
    logicGate = null; // deselect gate.
  }// end mouseReleased
  public void mouseEntered(MouseEvent event)
//** This method is called every time the mouse enters a registered component.
//** It is used here to initialize a logic circuit to the correct pin values
//** of each gate.
  {
    String myString = " ";
    String string = event.getComponent().getClass().getName();
    StringTokenizer stringTokenizer = new StringTokenizer(string,".");
    while (stringTokenizer.hasMoreTokens())  // parse the info string.
    {
      myString = stringTokenizer.nextToken(); // get next token in string.
    }// end while

    if(myString.compareTo("Button") == 0)   // is the click on the function button?
    {
      // run initial simulation when function button clicked.
      initializeCircuit();
    }
  }// end mouseEntered

  public void mouseExited(MouseEvent event)
  {
  }// end mouseExited

  public void mousePressed(MouseEvent e)
//** this method is called every time the mouse is pressed down.
  {
    this.xCoordinate = e.getX(); //set the cursor coords to this mouse position.
    this.yCoordinate = e.getY();

    String colorString;
    colorString = simulator.colorChoice.getSelectedItem();

    if(colorString.equals("LAVENDER")) // is the click on the default setting?
    {
      setBackground(new Color(204, 204, 255));
    }
    if(colorString.equals("WHITE")) // is the click on the default setting?
    {
      setBackground(new Color(255, 255, 255));
    }
    if(colorString.equals("GREY")) // is the click on the default setting?
    {
      setBackground(new Color(217, 217, 217));
    }
    if(colorString.equals("BEIGE")) // is the click on the default setting?
    {
      setBackground(new Color(255, 255, 204));
    }
    if(colorString.equals("OLIVE")) // is the click on the default setting?
    {
      setBackground(new Color(66, 99, 66));
    }
    if(colorString.equals("BLUE")) // is the click on the default setting?
    {
      setBackground(new Color(99, 204, 255));
    }
    if(colorString.equals("PINK")) // is the click on the default setting?
    {
      setBackground(new Color(255, 204, 255));
    }
    if(colorString.equals("GREEN")) // is the click on the default setting?
    {
      setBackground(new Color(66, 255, 204));
    }
    if(colorString.equals("ORANGE")) // is the click on the default setting?
    {
      setBackground(new Color(255, 204, 99));
    }
    if(colorString.equals("YELLOW")) // is the click on the default setting?
    {
      setBackground(new Color(255, 255, 66));
    }
    repaint();
  }// end mousePressed

  public void mouseMoved(MouseEvent e)
//** this method is called every time the mouse is moved.
  {
    if(simulator.buildCircuit)
    {
      if(currentState == connectGate)
      {
        this.xCoordinate = e.getX();
        this.yCoordinate = e.getY();
        repaint(); // to show the line connection is following the mouse.
      }
    }// end if(simulator.buildCircuit)
  }// end mouseMoved

  public void mouseDragged(MouseEvent e)
//** this method is called every time the mouse is dragged.
  {
    Point anchorPoint = null;

    if(anchorPoint == null)  // get mouse click position.
    {
      anchorPoint = new Point(e.getPoint());
    }
    else
    {
      anchorPoint.x = e.getX();
      anchorPoint.y = e.getY();
    }
    if(simulator.buildCircuit)
    {
      // go through all the gates in the circuit.
      for(int counter = 0; counter < logicCircuit.size(); ++counter)
      {
        if(logicGate == null)
        {
          // find out the gate just clicked.
          logicGate = ((LogicGate) logicCircuit.elementAt(counter)).whichGate(anchorPoint);
        }
      }// end for
      if(logicGate != null)    // if a gate is found.
      {
        //reposition it at the new location.
        GetUpdatedLocation(e.getX(),e.getY());
        repaint();    // refresh screen display.
      }
    }// end if(simulator.buildCircuit)
  }// end mouseDragged
  private void propagateState()
//** This method is used to connect logically two gates together, by propagating
//** a signal from the output of one gate to an input of another gate.
  {
    final int topPin = 1;     // digit value of top input pin.
    final int bottomPin = 2;     // digit value of bottom input pin.

    if(whichInputPin == topPin)  // if the top pin is selected.
    {
      // set both ends of the line connecting the two gates to be equal.
      theDestinationGate.topPinConnection = theOriginatingGate;
    }
    else if(whichInputPin == bottomPin) // same case as above.
    {
      theDestinationGate.bottomPinConnection = theOriginatingGate;
    }

    // store this connection into the connectionVector.
    theOriginatingGate.connectionVector.addElement(theDestinationGate);
    theDestinationGate = null; // deselect the receiving gate.
    theOriginatingGate = null;  // deselect the sending gate.
    currentState = createGate;
  }// end propagateState

  private void GetUpdatedLocation(int newXCoordinate, int newYCoordinate)
//** this method updates the gate position on the canvas.
  {
    boolean gatePositionChanged = false;

    if(logicGate != null)
    {
      if(((newXCoordinate / logicGate.gridWidth) * logicGate.gridWidth) + (logicGate.gridWidth / 2) !=
logicGate.xCoordinate)
      {
        logicGate.xCoordinate = ((newXCoordinate / logicGate.gridWidth) * logicGate.gridWidth) +
(logicGate.gridWidth / 2);
        gatePositionChanged = true;
      }

      if(((newYCoordinate / logicGate.gridHeight) * logicGate.gridHeight) + (logicGate.gridHeight / 2) !=
logicGate.yCoordinate)
      {
        logicGate.yCoordinate = ((newYCoordinate / logicGate.gridHeight) * logicGate.gridHeight) +
(logicGate.gridHeight / 2);
        gatePositionChanged = true;
      }

      if(gatePositionChanged)
      {
        repaint();
      }
    }// end if(logicGate != null)
  }// end GetUpdatedLocation

  public void paint(Graphics g)
// paint specifies how object g is to be displayed.
  {
    update(g);
  }// end paint

  public void update(Graphics g)
//** this method is used to override the class's update function. it paints
```

```
//** objects to an off-screen image and then displays this image on screen.
//** this way, flashing of moving objects and signals changing is eliminated.
  {

    final int topPin = 1;     // digit value of top input pin.
    final int bottomPin = 2;  // digit value of bottom input pin.
    final int outputPin = 3;  // digit value of output pin.
    Dimension d = getSize(); // get dimensions of the canvas's drawing area.

    // create an off-screen graphics drawing environment if none existed
    // or if the user resized the drawing area to a different size.
    if((offScreenGraphics == null) || (d.width != offScreenDimension.width)
      || (d.height != offScreenDimension.height))
    {
      offScreenDimension = d;
      offScreenImage = createImage(d.width, d.height);
      offScreenGraphics = offScreenImage.getGraphics();
    }

    // erase the previous image.
    offScreenGraphics.setColor(getBackground());
    offScreenGraphics.fillRect(0,0,d.width,d.height);
    offScreenGraphics.setColor(Color.black);

    // paint a border around the drawing area.
    offScreenGraphics.draw3DRect(0,0,d.width - 1,d.height - 1,true);
    offScreenGraphics.draw3DRect(0,0,d.width - 2,d.height - 2,true);

    // draw all gates in the circuit onto the off-screen image.
    for(int counter = 0; counter < logicCircuit.size(); ++counter)
    {
      ((LogicGate) logicCircuit.elementAt(counter)).displayGate(offScreenGraphics);
    }

    if(simulator.buildCircuit)  // if in build mode.
    {
      if(logicGate != null)  // if dragging a gate, display it onto the off-screen image.
      {
        logicGate.displayGate(offScreenGraphics);
      }
      if(currentState == connectGate)  // if connecting up gates.
      {
       if(theDestinationGate != null)  // if there is a gate receiving a connection.
       {
       // if a gate is selected by clicking its top input pin, draw a
       // connection line from that pin to where ever the mouse goes.
         if(whichInputPin == topPin)
         {
           if(theDestinationGate.isConnectorObject)
           {
             offScreenGraphics.drawLine(theDestinationGate.xCoordinate - 24,
             theDestinationGate.yCoordinate, xCoordinate, yCoordinate);
           }
           else if(theDestinationGate.isNotGate)
           {
             offScreenGraphics.drawLine(theDestinationGate.xCoordinate - theDestinationGate.gridWidth / 2 ,
             theDestinationGate.yCoordinate - theDestinationGate.gridHeight / 4 + 5 ,
             xCoordinate,yCoordinate);
           }
           else
           {
             offScreenGraphics.drawLine(theDestinationGate.xCoordinate - theDestinationGate.gridWidth / 2 ,
              theDestinationGate.yCoordinate - theDestinationGate.gridHeight / 4 ,
              xCoordinate,yCoordinate);
            }
          }
          else if(whichInputPin == bottomPin)
          // if a gate is selected by clicking its bottom input pin, draw a
          // connection line from that pin to where ever the mouse goes.
          {
            if(theDestinationGate.isConnectorObject)
            {
              offScreenGraphics.drawLine(theDestinationGate.xCoordinate,
              theDestinationGate.yCoordinate, xCoordinate, yCoordinate);
            }
            else
            {
              offScreenGraphics.drawLine(theDestinationGate.xCoordinate - theDestinationGate.gridWidth / 2,
              theDestinationGate.yCoordinate + theDestinationGate.gridHeight / 4,
              xCoordinate,yCoordinate);
            }
          }
        }// end if(theDestinationGate != null)
        else if(theOriginatingGate != null) // if there is a gate sending a connection.
        {
          // if a gate is selected by clicking its output pin, draw a
          // connection line from that pin to where ever the mouse goes.
          if(theOriginatingGate.isConnectorObject)
          {
            offScreenGraphics.drawLine(theOriginatingGate.xCoordinate - 15,
            theOriginatingGate.yCoordinate, xCoordinate, yCoordinate);
          }
          else
          {
            offScreenGraphics.drawLine(theOriginatingGate.xCoordinate + theOriginatingGate.gridWidth / 2 ,
            theOriginatingGate.yCoordinate ,
            xCoordinate,yCoordinate);
          }
        }// end else if(theOriginatingGate != null)
      }// end if(currentState == connectGate)
    } // end if(simulator.buildCircuit)
    else    // else display the results of running the simulation,
    {       // by showing the states of the pins of all gates in the circuit.
      for(int counter = 0; counter < logicCircuit.size(); ++counter)
      {
        ((LogicGate) logicCircuit.elementAt(counter)).displayStates(offScreenGraphics);
      }
    }// end if(simulator.buildCircuit)

    // paint the off-screen image to the application's viewing window.
    g.drawImage(offScreenImage,0,0,this);
  }// end update
}// end MyCanvas class
```

Appendix B

System Manual

LogicCity is a digital logic simulator used for entering a logic circuit schematic and performing simulations of its behavior. You will only need a few minutes before you begin to use the system. The next few paragraphs detail how to use the simulator software. If you have questions that are left unanswered, please feel free to contact your instructor for further explanations.

B.1 How to Build Circuits

When LogicCity software is started, a circuit window used for drawing a logic circuit schematic opens up. This window (canvas) has a menu on the left side of the canvas composed of buttons with each button representing a different gate or circuit part. To create a gate and display it on the canvas, simply follow these three easy steps.

1. First make sure that the master switch button is set to build mode. The master switch is found at he top left of the screen (labeled Function). A label, displaying the current mode of the switch, is placed next to the master switch.
2. Next, simply click the mouse on the desired gate button from the menu and move the mouse over to the canvas area.
3. Finally, choose a location for your gate and click the mouse again at that position. The selected gate or circuit part will appear at the desired location.

To create additional gates, just follow the above steps as many times as needed.

Once you have loaded a few gates into the canvas, they can easily be wired together.

1. First, decide which two gates are to be connected.
2. Then choose the gate that is sending the output signal and click its output pin.
3. Next, move the mouse over to the second gate to be connected and click the mouse over an input pin to finalize the connection procedure.

The simulator draws a line between the two selected pins. If you want to route wires around the gates to keep the circuit clean and comprehendible, connector objects are available in the parts menu. To place connector objects on the canvas, follow the same steps used to place logic gates. A connector simply acts as a pass through buffer; it has no value to the logic circuit except to make it look cleaner by routing wires around in neat manner.

B.2 Entering Input Signals / Trapping Output Signals

Once you have built a logic circuit, you will need to create one or more input objects depending on your circuit design. Input objects are used to enter input signals into a circuit. Again, placing input objects is done exactly the same way as placing logic gates. After you create an input object:

1. First click the pin used to send the signal out to the circuit
2. Then move the mouse over to the desired location
3. Finally, click the mouse once more to connect the input object to the circuit.

You can continue in this manner to wire other input objects. To obtain an output from a circuit, you will need to create and connect an output object (in the same fashion you created and connected the input object) to the logic circuit. Output signals are provided to present the circuit outputs in a clear visual way.

B.3 How to Edit a Circuit

Making a mistake while building a logic circuit is inevitable. So an edit menu composed of four buttons is provided at the top of the canvas. The four buttons are – "open," "save," "erase," and "clear all." If you wish to erase a gate from the canvas, simply click the "erase" button (this activates the erase mode), and then click the object that you want to delete. The simulator will delete the selected gate by clearing its position on canvas. Every time you need to erase an object from the canvas, you will need to click the erase button first and then the object second. The erase mode is activated only once per click for security reasons to avoid the accidental erasure of an object if the user is not paying attention.

If the entire screen is desired to be cleared, just click on the "clear all" button. A pop up dialog box, with two options, will appear to confirm your request. You have the option to:

- Click on the "cancel" button to get back to the circuit or
- Click on the "ok" button to go ahead and delete the whole circuit. If the circuit is deleted entirely, the canvas is refreshed and displays an empty screen. At that point, the simulator is ready to either build another circuit or open an existing one.

When in the early stages of development, you can never predict how a circuit will actually look. Some components may be too close to each other or in the wrong location, or you may need to scatter some congested areas of the circuit for better clarity. Moving a circuit component around the canvas is easy to do. Just click and hold down the mouse over the desired object and then drag the component to any new location. Gates will easily move around even if they are connected. All wire connections will follow along with the relocated object.

B.4 How to Save and Open a Circuit

Saving a circuit is easily accomplished by clicking on the "save" button after a logic circuit is built. A dialog box with standard Windows format will pop up. Inside the box one can see the file system of the computer used. The user can navigate to the desired directory and then type in a name for the new circuit. If the name given is already used, the system will advise to that fact and prompts again for another file name. Pushing "enter" sends the new name to the simulator program, which then displays the new file name at the top of the canvas.

To open an already existing circuit, just click on the "open" button from the edit menu at the top of the screen. If there is a circuit in the canvas at the time the "open" button is clicked, a dialog box that asks the user what to do with the present circuit appears. At this point, you can:

- First save the circuit before opening another one or
- Ignore the dialog box, which will automatically delete the present circuit once the new circuit appears.

If the canvas is empty at the time the “open” button is clicked, then the “open” dialog box will appear and wait for the user to select a circuit from the computer's filing system. The user can type the name of the desired circuit or just click on its icon to select it. Pushing “enter” sends the message to the simulator, which will search the filing system, locate the selected file, and produce the circuit as a logic schematic on screen.

B.5 How to Simulate a Circuit

When the circuit is ready for simulation, you can click on the master switch to switch from the edit mode to the simulate mode. After this button is clicked, the states of all objects in the circuit are displayed. All input pins and output pins have either a ‘0’ or a ‘1’ next to them. A ‘0’ value represents logic ‘low’ and is displayed in black, while ‘1’ represent logic ‘high’ and is displayed in red. Initially all input signals to the circuit are set by default to ‘low,’ but you can change the input state by toggling the input object. Toggling is achieved by clicking inside the body of the input object. Input objects are initially displayed in blue, but once toggled to ‘high’ its output pin is drawn in red to visually display that a ‘high’ signal is being sent. If an output of the circuit (drawn in light Grey) receives a ‘high’ signal, the whole body of the output object distinctly glows in red. When input objects are toggled, the effect propagates throughout the entire circuit changing the states of all affected gates.

B.6 How to Change the Background Color

The window's background color can easily be changed in either build or simulate modes by using the pull-down menu provided at the bottom left of the canvas. A click on the menu extends the body of the menu to show the ten color selections available. By clicking on the color of interest and then clicking on the canvas once, the new color takes effect immediately.

Bibliography

Amico, Vince, and Clymer A. Ben, "All About Simulators". *Proceedings of the SCS Simulators Conference*. A Publication of the Society for Computer Simulation. 1984.

Anderson David, Roberts, Nancy, Real Ralph, Garet, Michael, and Shaffer, William. *Introduction to Computer Simulation: A System Dynamics Modeling Approach.* Productivity Press. 1994.

Bennett, A. Wayne. *Introduction to Computer Simulation.* pp 429-442. West Publishing Company. 1974.

Bennett, Frederick, Ph.D., *Computers as Tutors: Solving the Crisis in Education.* pp 10-34. http://www.cris.com/~Faben1. 1996.

Bishop, Judy M., *Java Gently, Programming Principles Explained*, Addison-Wesley Publishing Co., 1997.

Berkum, J.J.A. van, Hijne, H., de Jong, T., van Joolingen, W.R., Njoo, M., "Learning processes, learner attributes and simulations". *Education & Computing*, Volume 6, pp 231-239, 1991.

Berkum, J.J.A. van, & de Jong, T., "Instructional environments for simulations". *Education & Computing*, Volume 6, pp 305-358, 1991.

Blease, Derek. *Evaluating Educational Software*. Croom Helm. 1986.

Campione, Mary, and Walrath, Kathy. *The Java Tutorial: Object-Oriented Programming for the Internet*. Addison-Wesley Publishing Co., 1996.

Coburn, Peter, Kelman, Peter, Roberts, Nancy, Snyder, Thomas, F.F., Watt, Daniel H., and Weiner Cheryl. *Practical Guide to Computer Education*. Addison-Wesley Publishing Co., 1982.

Cornell, Gary and Hortsmann, Cay S., *Core Java*, Second Edition. The Sunsoft Press, A Prentice Hall Title, 1997.

Dean, Christopher, and Whitlock, Quentin. *A Handbook of Computer Based Training*, Second Edition. Kogan Page, London/Nichols Publishing Co., 1989.

Deitel, H. M. and Deitel, P. J., *Java, How to Program*, Second Edition. Prentice Hall, 1998.

Doll, Carol A. *Evaluating Educational Software*. American Library Association. 1987.

Ellington, H.I., Addinall, E., Percival, R. *Games and simulations in science education*. London: Kogan Page. 1981.

Foley, J. D., and Van Dam, A., *Fundamentals of interactive Computer Graphics*, Addison-Wesley Publishing Co., 1990.

Foley, J. D., Van Dam, Andries, Feiner, Stephen K., Hughes, John F. and Phillips, Richard L., *Introduction to Computer Graphics*, Second Edition. Addison-Wesley Publishing Co., 1994.

Hamacher, V. Carl, Vranesic, Zvonko, G., Zaky, Safwat G., *Computer Organization*. McGraw-Hill Book Company. 1978.

Hays, R.T., and Singer, M.J., *Simulation Fidelity in Training System Design: Bridging the Gap Between Reality and Training*. Springer-Verlag Publishing. 1989.

Heimler, Charles, Cunningham, James, and nevard Michael. *Authoring Educational Software*. Mitchell Publishing, Inc. 1987.

Hoog, R. de, Jong, T. de & Vries, F. de, "Interfaces for instructional use of simulations". *Education & Computing*, Volume 6, pp 359-385. 1991.

Joolingen, W.R. van & de Jong, T., "Characteristics of simulations for instructional settings". *Education & Computing*, Volume 6, pp 241-262. 1991.

Kain, Richard Y., *Computer Architecture: Software and Hardware*. Prentice Hall, Inc., 1989.

Kennedy, David M., "Interactive Multimedia: Educational Desert or Educational Oasis". Paper, *Multimedia Education Unit*. The University of Melbourne, Australia.

Korn, Granino A., *Interactive Dynamic System Simulation*. McGraw-Hill Book Company. 1989.

Lewis, John and Loftus, William. *Java Software Solutions*, Preliminary Edition. Addison Wesley Longman, Inc., 1998.

McTear, Michael F., *Understanding Cognitive Science*, John Wiley & Sons, Inc., Publishers. 1988.

Milson, Marliese. "Educational Technology: Hype or Help in Reforming Education". EDUC 420, *The Professional Teaching and American Education*. Mary Washington College.

Min, F.B.M., "Parallel Instruction, a Theory for Educational Computer Simulation". *Interactive Learning International*, Volume 8, No. 3, pp 177-183. 1992.

Nievergelt, Jay, Ventura, Andrea, and Hinterberger, Hans. *Interactive Computer Programs for Education: Philosophy, Techniques, and Examples.* Addison-Wesley Publishing Co., 1986.

Perkins, David N., Schwartz, Judah L., West, Mary M., and Wiske, Marth S. *Software Goes to School.* pp 106. Oxford University Publishing, 1995.

Reigeluth, Charles M., *Instructional-Design Theories and Models: An Overview of their Current Status.* Lawrence Erlbaum associates, Inc. Publishers. 1983.

Reigeluth, C.M. & Schwartz, E., "An instructional theory for the design of computer-based simulations". *Journal of computer-based instruction*, Volume 16, No. 1, pp 1-10. 1989.

Romiszowski, A.J., "Designing Instructional Systems: Decision Making in Course Planning and Curriculum Design". *Kogan Page*, London/Nichols Publishing Co., 1981.

Schaick Zillesen, P.G. van & Min, F.B.M, "MacTHESIS: a design system for educational computer simulation programs". *Wheels for the mind of Europe*, pp 23-33. 1987.

Schofield, Janet Ward. *Computers and Classroom Culture*. Cambridge University Press. 1995.

Sewell, David F., *New Tools for New Minds*. St. Martin's Press. 1990.

Standish, Thomas A., *Data Structures in Java*, Addison-Wesley Publishing Co., 1998.

Tanenbaum, Andrew S., *Structured Computer Organization*, Third Edition. Prentice Hall, 1990.

United States. *U.S. Department of Education.* "Getting America's Students Ready for the 21st Century". pp 5-20. U.S. Government Printing Office, 1996.

United States. *U.S. Department of Education*. Office of Educational Technology. "Making It Happen". Report of the Secretary's Conference on Educational Technology. pp 16-24. U.S. Government Printing Office, 1995.

United States. *U.S. Department of Education*. Office of Educational Research and Improvement. "Using Technology to Support Education". pp 1-28. U.S. Government Printing Office, 1993.

Dizzy: By Jim Munki, 1990. URL: ftp://ftp.symantec.com/public/english_us_canada/products/c++/mac/samples/source_code /dizzy.sit.hqx

Logg-O URL: http://www.pws.com/aeonline/course/7/2/index.html, Lab 7.2: "Bill's Gates". Rick Decker and Stuart Hirshfield, PWS Publishing Company, 1998.

Weiss, Mark Allen, *Data Structures & Problem Solving Using Java*. Addison-Wesley Publishing Co., 1998.

Whicker, Marcia Lynn, and Sigelman, Lee. *Computer Simulation Applications: An Introduction*. Sage Publications. 1991.

Williams, Frederick and Williams, Victoria. *Success with Educational Software*. Praeger Publishers. 1985.

Zeigler Bernard P., *Theory of Modeling and Simulation*, John Wiley & Sons, Inc., Publishers. 1976.

Zobrist, George W., and Leonard, James V., *Progress in Simulation*, Volume One. Ablex Publishing Corp. 1992.