Web Simulator for Service-Oriented Robots

by

Garrett Drown

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2012 by the
Graduate Supervisory Committee:

Wei-Tek Tsai, Co-Chair
Yinong Chen, Co-Chair
David Claveau

ARIZONA STATE UNIVERSITY

December 2012

ABSTRACT

The focus of this document is the examination of a new robot simulator developed to aid students in learning robotics programming and provide the ability to test their programs in a simulated world. The simulator, accessed via a website, provides a simulated environment, programming interface, and the ability to control a simulated robot. The simulated environment consists of a user-customizable maze and a robot, which can be controlled manually, via Web service, or by utilizing the Web programming interface. The Web programming interface provides dropdown boxes from which the users may select various options to program their implementations. It is designed to aid new students in the learning of basic skills and thought processes used to program robots. Data was collected and analyzed to determine how effective this system is in helping students learn. This included how quickly students were able to program the algorithms assigned to them and how many lines of code were used to implement them. Students' performance was also monitored to determine how well they were able to use the program and if there were any significant problems. The students also completed surveys to communicate how well the website helped them learn and understand various concepts. The data collected shows that the website was a helpful learning tool for the students and that they were able to use the programming interface quickly and effectively.

TABLE OF CONTENTS

Page

CHAPTER **Page**

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

Robots that are controlled by Web services become increasingly popular as they enable computational workloads to be processed on a remote machine. These types of robots require less local computing power by offloading the work to the remote machine (similar to cloud computing). Once the remote machine has completed its computations and reached a decision, the system then sends the selected commands to the robot. This enables longer battery life for the robot and possibly the use of a smaller robot. Unfortunately, these types of robots are new and there are limited options available to developers for testing and simulating programs for their robots. The research presented here describes a newly developed simulator for testing these types of robots and to aid in the education of inexperienced programming students.

1.1. Computer Science Teachers Association

The Computer Science Teachers Association (CSTA) provides support for educators teaching in the field of computer science. They supply recommendations, examples, and sample materials to help teachers develop and organize their own curriculum. CSTA is sponsored by Google, Microsoft, Oracle, and others.

Furthermore, students will be engaging in some of the activities recommended in the Career and Technical Education (CTE) Common Core Standards as discussed by CSTA. These include: (1) "[a]pply appropriate academic and technical skills" (Verno and Fuschetto) especially in regards to computer programming. (2) "Demonstrate creativity and innovation" (Verno and Fuschetto). (3) "Utilize critical thinking to make sense of problems and persevere in solving them" (Verno and Fuschetto). (4) "Use technology to enhance productivity" (Verno and Fuschetto).

1.2. Motivation and Research Questions

Robots controlled via Web services are new and there are limited options available to developers for testing and simulating programs for their robots. Similarly, educational institutions have limited options to provide their students with relevant software tools to aid students in these endeavors. The software presented in this paper was developed to aid students in the learning of basic programming skills as well as the writing and testing of robotics applications. Furthermore, this software designed to aid students in the learning of the topics presented by CSTA. It does this in a variety ways as described in Table 2.

The experiment presented herein was designed to determine how well students performed in the areas described by CSTA and to receive feedback regarding the usefulness of the website. Using the CSTA input as guidelines, the following research questions were developed:

1. What is the impact of Web-based visual programming in helping students understand programming logic?
2. Does a user-friendly Web-based interactive tool help students understand human-computer interaction and program construction?
3. Does the Web-based program provide an interesting feedback mechanism from which students can evaluate their software?

The table below demonstrates how these questions relate to the CSTA material.

Table 1: Research Questions and Related CSTA Topics

Research Questions and Related CSTA Topics	
Research Question	**CSTA Topic**
What is the impact of Web-based visual programming in helping students understand programming logic?	Understanding and exploring the problem (Madden et al. 7).
	Design, code, and test a solution (Madden et al. 23).
	"The power of stepwise refinement" (Madden et al. 7).
	"Graphs" (Madden et al. 12).
Does a user-friendly Web-based interactive tool help students understand human-computer interaction and program construction?	"Interactive Programming" (Madden et al. 8).
Does the Web-based program provide an interesting feedback mechanism from which students can evaluate their software?	"Identify elements of user-friendly Web sites" (Madden et al. 14).

To conduct this experiment the students were given two algorithms and asked to program them using the Web programming interface on the website. As the students implemented their individual programs, they were carrying out different actions given in Table 2. For examples of how students may do this during the experiment, see Table 2.

The experiment described in this paper implements some of the ideas provided by CSTA. The following table describes some of the topics recommended by CSTA and how they have been utilized in the experiment. Students are able to learn about and engage in these various topics and activities while using the website.

Table 2: Designing of the Experiment

Designing of the Experiment	
Research Questions	**Experiment Design**
What is the impact of Web-based visual programming in helping students understand programming logic?	Students are given algorithms and asked to implement them using the Web programming interface provided on the website. Through this exercise, students will need to come to understand the steps described by the algorithm to implement a correct working program.
	Students are given algorithms and asked to program them using the Web programming interface provided on the website. Students will need to design a solution for each algorithm and implement the solution via the Web programming interface. After students implement their programs they will need to test each one, identify any bugs, and correct any problems.
	After students are presented with the algorithm, they will need to break the algorithm down into basic steps which can be programmed via the Web programming interface.
	The simulated robot, which the students will be programming, is located in a maze. The algorithms presented to the students are able to successfully lead the robot to the other end of the maze. As the students implement the algorithms provided to them, they will be able observe the robot carry out the algorithms and the different paths taken by each algorithm.
Does a user-friendly Web-based interactive tool help students understand human-computer interaction and program construction?	Students will be utilizing the Web programming interface, which provides them with an interactive development environment (IDE), to program the robot. Additionally, the website enables the students to see the robot in real-time and see how changes to their program affects the behavior of the robot.
Does the Web-based program provide an interesting feedback mechanism from which students can evaluate their software?	After students have completed their assigned tasks, they were asked to rate their experience as well as provide any positive or negative feedback they may have regarding the website. This feedback was used to determine possible future work on the website. The time taken by the students to solve the algorithms can also be used as an indicator as to how well the students were able to utilize the website.

Before the experiment was conducted, the following was hypothesized. (1) Students will find the Web programming interface a useful educational tool and it will aid them in their learning of various programming concepts. (2) Students will be able to efficiently program using the Web programming interface and quickly implement correct

solutions to the programming problems presented to them. (3) Students will find the website helpful, easy to use, and be able to use it with little to no assistance.

1.3. Background

According to the World Wide Web Consortium (W3C), "[a] Web service is a software system designed to support interoperable machine-to-machine interaction over a network" (Web Services Architecture Working Group). Functionally speaking, a Web service is similar to a normal function used in programming. A standard programming function generally has at least one input parameter and an output that it returns. For example, a function that sorts a string of characters to be in alphabetical order would have a string as an input parameter and the function would return the sorted string. However, Web services are not restricted to providing only a single function. Web services are capable of supplying an entire set of functions that may be useful to a developer for a particular use case. In the end, "[a]ll components are wrapped with open standard interfaces to form services or Web services, so that application builders can use those services across the Internet" (Tsai et al., "An Introductory Course on Service-Oriented Computing for High Schools" 318).

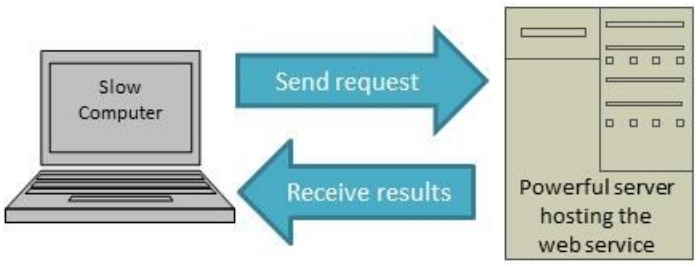

Figure 1: Example Web Service Workflow

Web services are similar to standard programming functions. However, "[t]hese services are platform-independent. They can be published in public or private directories or repositories for software developers to reuse and to compose their applications" (Chen

and Tsai, "Towards Dependable Service-Oriented Computing Systems" 1361). This also enables Web services to be hosted on a server that is more powerful than the client computer. An example of a Web service is one that solves a complex and computationally expensive problem. Additionally, assume that this Web service is hosted on a powerful server. It will not be feasible for a slow computer to perform this computation itself. Instead, by utilizing a Web service the computational workload is passed on from the weak computer to the powerful system, which can compute the solution in substantially less time. Arizona State University hosts a collection of Web services at the ASU Repository of Web Services and Web Applications ("ASU Repository"), which includes a variety of examples of possible uses for Web services.

Using the simulator developed for this thesis project, developers can develop and test robots that are controlled via Web services. Because the simulator can be hosted online, developers are not required to install any new applications on their local computers to begin using the simulator. "Using [service-oriented computing] for robotics has certain inherent advantages over the traditional methods of building a robot. The main benefit is to have a layer of common services with standard interfaces" (Chen and Tsai, *Service-Oriented Computing and Web Software Integration* 395). This is an important characteristic, as it enables developers to use the simulator without requiring significant changes to their code. A developer is only required to modify his/her program's Web services to communicate with the simulator's Web service instead of the Web service hosted on the real physical robot.

Similar work has begun on a Service-Oriented Computing (SOC) environment for developers to generate and execute code developed specifically for smart homes. "Service-oriented computing is considered today as a key technology for the development of robust and high quality intelligent distributed and embedded

applications" ("Call for Papers"). One of the key characteristics of SOC is that "[r]ather than building static applications, SOC approach composes applications by discovering and invoking services based on published interfaces" (Lee et al. 1).

This simulator focuses on service-oriented robots and provides many of the common features provided by other such simulators. For example, it provides the ability to test the robot's programming in a virtual world without endangering people, robots, or other property. It also gives developers the convenience of testing their program on a simulated robot, even before they have a physical robot in their possession.

The simulator also provides a simple graphical programming interface directly on the website. This interface is designed to be an intuitive method for new and inexperienced programmers to be introduced to the field of robotics programming. By using a series of dropdown boxes, the user is able to create simple programs to be carried out by the robot. The dropdown boxes offer simple programming functions, such as move forward, move backward, stop, etc. and are executed when specified logical statements are true. After entering his/her instructions, the user can see the robot in action and, if need be, reset the robot to try again after modifying the program.

As technology becomes more commonplace, pervasive, and familiar to an ever-growing market it is not surprising that computers are becoming popular as an educational tool. "An emerging Web-based platform trend can be seen in the amount of educational materials … currently available via the Web. Students can also take examinations via the Web and work on joint projects using Web-based collaboration tools such as *Google Docs*" (Tsai et al., "Collaborative Learning Using Wiki Web Sites" 114). This trend is due to more than simply the growth and utilization of technology. "Computers are highly interactive and provide a variety of tools to accomplish meaningful tasks. Hence, they are more aligned with the 'learning by doing' view of

education, than with the ‘absorption of cultural knowledge’ view of education that permeates schooling” (Collins and Halverson 20). This ‘learning by doing’ method of education is one of the reasons why computers can be such a valuable tool in the classroom and a part of the motivation behind the development of the programming interface available on the website. This is further seen by the plans being made at Arizona State University to use this simulator in future courses as well as Arizona State University’s Robotics Camp in 2013. The Robotics Camp is designed for middle to high-school students to learn about designing and programming robots. The Web programming interface provides the simple high-level programming instructions that can be useful when teaching new programmers.

CHAPTER 2

EXAMPLES OF ROBOTICS SIMULATORS

Currently, there are many simulators available to developers. Existing robotics simulators generally include a virtual world, physics, and a programmable simulated robot. However, these simulators typically have their own complex application programming interface (API) and interactive development environment (IDE) that the developer must learn in order to use the simulator. These key components provide the user with the ability to simulate and test a robot with their own programs. Below are examples of simulators that are freely available.

2.1. Microsoft Robotics Developer Studio

Robotics Developer Studio (RDS) is a free program developed by Microsoft that includes both an IDE and a simulator. RDS utilizes Microsoft's Visual Programming Language to program the simulated robot. "Visual Programming Language (VPL) provides a relatively simple drag-and-drop visual programming language tool that helps make it easy to create robotics applications" (*Microsoft Robotics Developer Studio*). RDS provides the user with several robots to simulate, such as the iRobot, a robotic arm, and other simple robots. After choosing a particular robot, the user can then program the robot using the tools provided by the Visual Programming Language.

Users can drag-and-drop blocks (i.e. variables, program control flow logic, etc.) and connect them to other blocks to send and receive data to and from other blocks to build their program. RDS refers to these blocks as *activities* or *services*. Services include blocks which contain the configuration and physical details (if applicable) of a robot or robotic component. Services also include user-defined functions which have been consolidated into a single new block. Activities include user-defined activities as well as basic activities which drive the program's control flow or construct data,

variables, or structures. These activities (possibly in conjunction with other services) can be used to construct additional user-defined activities. These user-defined activities are analogous to functions in text-based programming languages, such as C#. Below is an example of a user-defined activity that can be constructed using VPL in RDS.

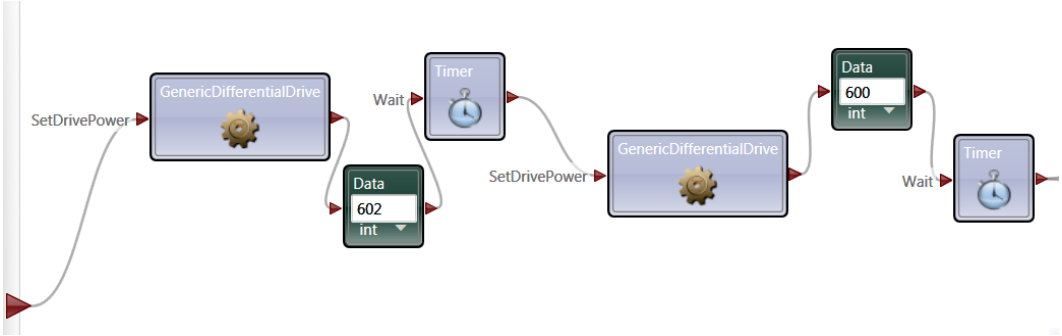

Figure 2: An Example of the Format and Logic Used in RDS Using VPL

The logic in Figure 2 shows the activities and services used in a VPL function to make a 90° right-turn. The program flows from the left to the right, beginning at the large triangle on the far left. The "GenericDifferentialDrive" block is used to control the motors of the simulated robot. The first instruction in this example sets the drive power of the motors. The parameters passed to the block are shown in Figure 3.

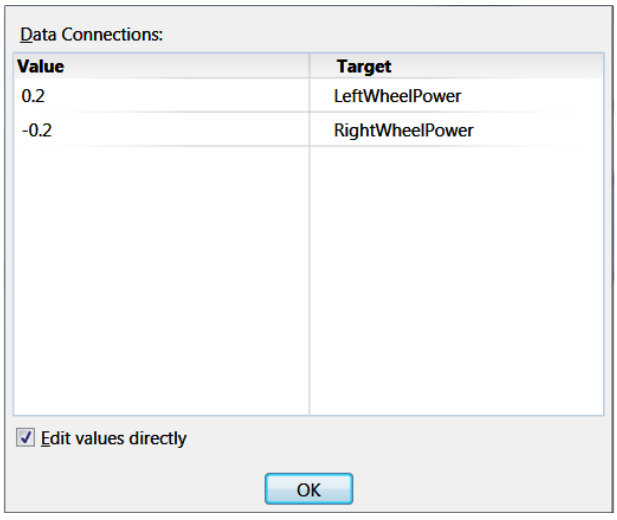

Figure 3: SetDrivePower Parameters

By passing these parameters, the left wheel will turn forward as the right wheel turns backwards, causing the robot to turn right. The value *602* is then passed to a timer block. This causes the program to wait for 602 milliseconds before continuing to the next block of the program. During this time, however, the robot is still turning. After the timer has expired, the robot is commanded to stop by passing zeros to the left and right wheel powers in a similar manner as shown in Figure 3. Another timer is used to make sure the robot comes to a complete halt before returning from the function.

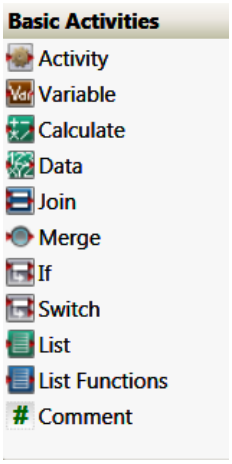

Figure 4: Activities Provided by RDS

Microsoft RDS provides many more blocks than what was shown in the previous example. Figure 4 shows all of the *basic activities* provided by RDS. Note the item named "Activity." This is the object used to create user-defined activities. There are also many more pre-defined services available in RDS. These services range from different robot templates to various types of sensors and pre-defined functions.

2.2. Alice

Alice is a free program developed by Carnegie Mellon University in collaboration with other universities to help students learn and explore programming in a 3-D simulated environment. "In Alice's interactive interface, students drag-and-drop graphic tiles to create a program, where the instructions correspond to standard statements in a

production oriented programming language, such as Java, C++, and C#" (*What is Alice?*). The graphic tiles provide different functions and program control flow operations to the user. Users develop their code by dragging and dropping these tiles into the provided workspace. Below is an example of the pre-defined methods available for *groundRoamer* (a pre-defined object in Alice) and the different control flow tiles available in Alice.

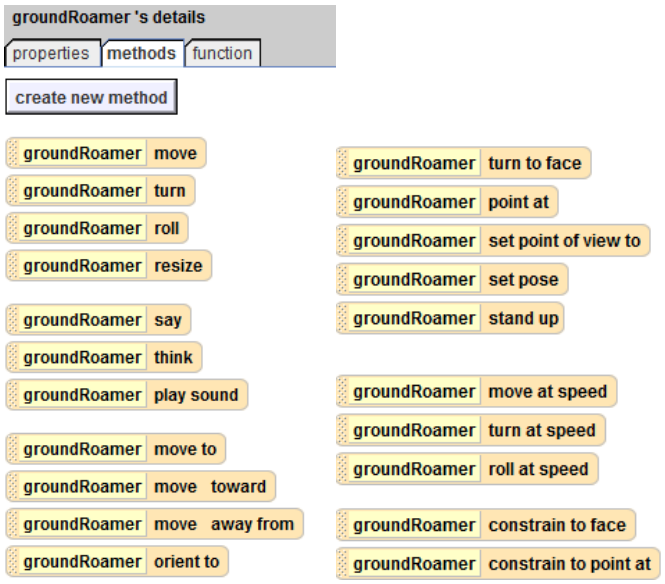

Figure 5: Pre-defined Methods Provided for *groundRoamer*

Figure 6: Control Flow Tiles Available in Alice

Alice also includes a 3-D virtual world. "In Alice, 3-D objects (e.g., people, animals, and vehicles) populate [this] virtual world and students create a program to animate the objects" (*What is Alice?*). The student is able to customize the world by dragging and dropping pre-defined objects into the virtual world workspace. Objects in the virtual world are then able to be referenced in the programming area. For the following example, a robot (called *groundRoamer*) and a fence (named *wall1*) have been placed in

the virtual world. After adding the objects, an algorithm similar to the right-wall-following algorithm was coded using the programming tiles provided by Alice.

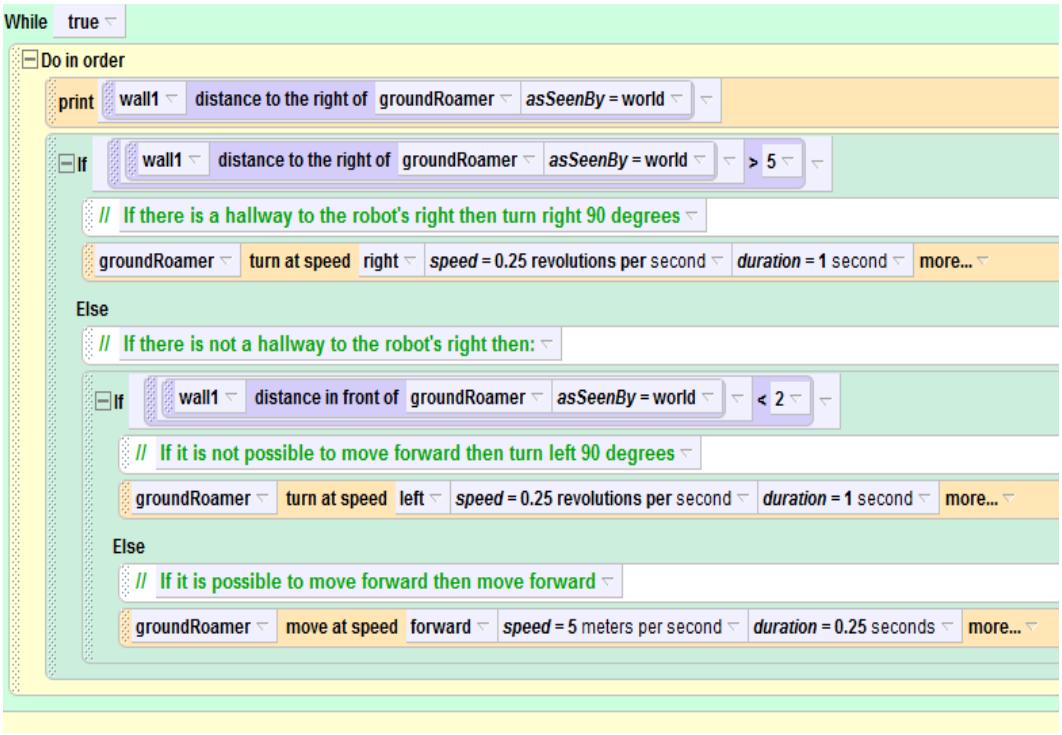

Figure 7: An Example of the Format and Logic Used in Alice

The outermost layer and first line of the logic in Figure 7 is an infinite while-loop. The first if-statement determines if there is a large distance to the right of the *groundRoamer* and *wall1*. If a large distance is detected, then the *groundRoamer* will make a 90° right turn. Otherwise, the nested if-statement will determine if there is some room between the *groundRoamer* and *wall1* so that the robot can continue moving forward. If there is not, then the *groundRoamer* will make a 90° left turn. Otherwise, the robot will move forward some before the program repeats itself (due to the infinite while-loop).

2.3. Features

Below is a comparison of some of the key features of the programming interfaces provided by the simulators described above and the simulator developed for this project.

Both of the programming methods provided by the simulator are included in the table. These methods include programming the robot utilizing the Web service provided by the simulator and programming the robot using the programming interface on the website.

Table 3: Feature Comparison of Programming Interfaces

Feature Comparison of Programming Interfaces				
	Programming via Project Simulator		**Microsoft Robotics Developer Studio's IDE**	**Alice's IDE**
	C#/Java Web Service	**Web Programming Interface**		
Key Features	Support for simulating robots controlled via Web services. Ability to use the same program to control a physical robot and the simulated robot. Provides an easy-to-use API.	Provides an intuitive and easy to learn interface for new programmers to learn about robotics programming. Straightforward dropdown boxes provide the users with simple instructions used to program the robot. Accessed via Web browser, thus easily accessible and presenting it in an environment familiar to students.	Utilizes Microsoft's Visual Programming Language, enabling users to drag-and-drop blocks (instructions) and connect them together to develop their program. Provides pre-defined templates for some of the most popular robots and robotic components.	The programming interface includes drag-and-drop tiles. Designed to enable programmers to program simple movies or games. The programming tiles provide simplified instructions for easier programming.
Required Software	Any IDE supporting the use of WSDL Web services.	Internet Explorer 9 with Silverlight 5.1 (or later) plug-in installed.	Windows 7 .Net Framework 4.0 Microsoft Robotics Developer Studio	Java Development Kit Alice

2.4. Other Types of Simulators

A wide variety of simulators (not related to robotics) have been developed, spanning a broad range of categories. As computers grow in their ability to solve complex problems, simulators will be used to help model and solve these problems. A few select simulators are listed below which provide a glimpse into the variety of available simulators.

Table 4: Brief Descriptions of Other Select Simulators

Simulator Name	Description	Type
X-Plane 10	"X-Plane 10 Global is the world's most comprehensive and powerful flight simulator for personal computers, and it offers the most realistic flight model available. … Because X-Plane predicts the performance and handling of almost any aircraft, it is a great tool … for engineers to predict how a new airplane will fly, and for aviation enthusiasts to explore the world of aircraft flight dynamics" ("What is X-Plane?").	Flight Simulator
Algodoo	"ALGODOO is a unique 2D-simulation software from Algoryx Simulation AB. Algodoo is designed in a playful, cartoony manner, making it a perfect tool for creating interactive scenes. Explore physics, build amazing inventions, design cool games or experiment with Algodoo in your science classes. Algodoo encourages students and children's own creativity, ability and motivation to construct knowledge while having fun" ("About").	Physics Simulator
The Virtual Cell (VCell)	"The Virtual Cell is a unique computational environment for modeling and simulation of cell biology .[*sic*] It has been specifically designed to be a tool for a wide range of scientists, from experimental cell biologists to theoretical biophysicists. The creation of biological or mathematical models can range from the simple, to evaluate hypotheses or to interpret experimental data, to complex multi-layered models used to probe the predicted behavior of complex, highly non-linear systems. Such models can be based on both experimental data and purely theoretical assumptions" ("Virtual Cell Modeling & Analysis Software").	Living Cell Simulator
LogicWorks 5	"LogicWorks is an interactive circuit design tool intended for teaching and learning digital logic. … The package gives you the power, speed and flexibility to create and test an unlimited number of circuit elements on-screen. This means that you can study advanced concepts much more quickly and clearly using on-screen simulation than you can by spending time wiring up expensive and damage-prone parts in a lab" ("What is LogicWorks 5?").	Circuits Simulator

CHAPTER 3

WEB SIMULATOR FOR SERVICE-ORIENTED ROBOTS

The simulator developed for this project provides many of the features of the simulators described above, including a virtual world, simulated physics, and a programmable object (i.e. a robot) to control and program. However, this simulator provides many new features. For example, the simulator provides the user with the ability to simulate robots controlled via Web services and can be used as a tool for teaching new students how to program robots using the Web programming interface. Additionally, the simulator can be hosted on the Internet and therefore users are not required to install any additional software on their local computer.

The simulator consists of the following three parts: a client application, the Web service, and the Silverlight website. The client application is the term used here to represent any application using a Web service to communicate with a physical robot or the simulator. In the context of the simulator, the client application is the component that can drive the simulator and is to be developed and provided by a developer. This program sends commands and receives data to and from the Web service. The data received from the Web service is used to determine the appropriate commands to send to the robot.

The Web service provides the means for the client application to communicate with and drive the simulator (i.e. the simulated robot). It does this by supplying the client application and the Silverlight website with functions that provide the ability to send and receive sensor updates and commands for the simulated robot. In a non-simulated environment, the Web service is normally hosted on a physical robot.

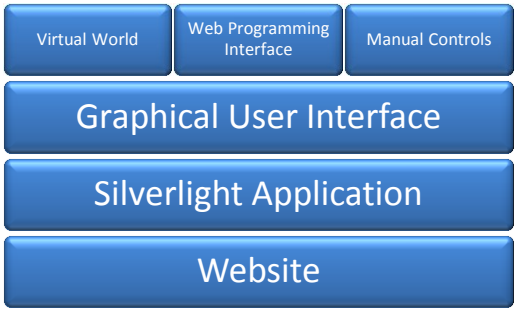

Figure 8: Simulator Component Hierarchy

The third component of the simulator is the Silverlight website, a website which hosts a Silverlight application. By hosting and providing the Silverlight application to clients, the website itself implements client-server architecture, or more specifically thick-client architecture. The Silverlight application is hosted on the website and is downloaded and executed on the client machine via a Web browser.

The Silverlight application is a multithreaded object-oriented application. There are individual threads for sending and receiving updates from the Web service, executing the program written via the Web programming interface, and more. See Figure 9 for an overview of the application's components and how they perform their functions.

Although the Silverlight application utilizes a Web service, it does not offload a computational workload to it. From the simulator's perspective, the Web service is only used as a means of communication for sending data to, and receiving commands from, the client application. The Silverlight application provides the GUI components of the simulator. These components include the virtual world (the maze, robot, physics, sensor data, etc.), manual control interface, sensor readout, and the Web programming interface.

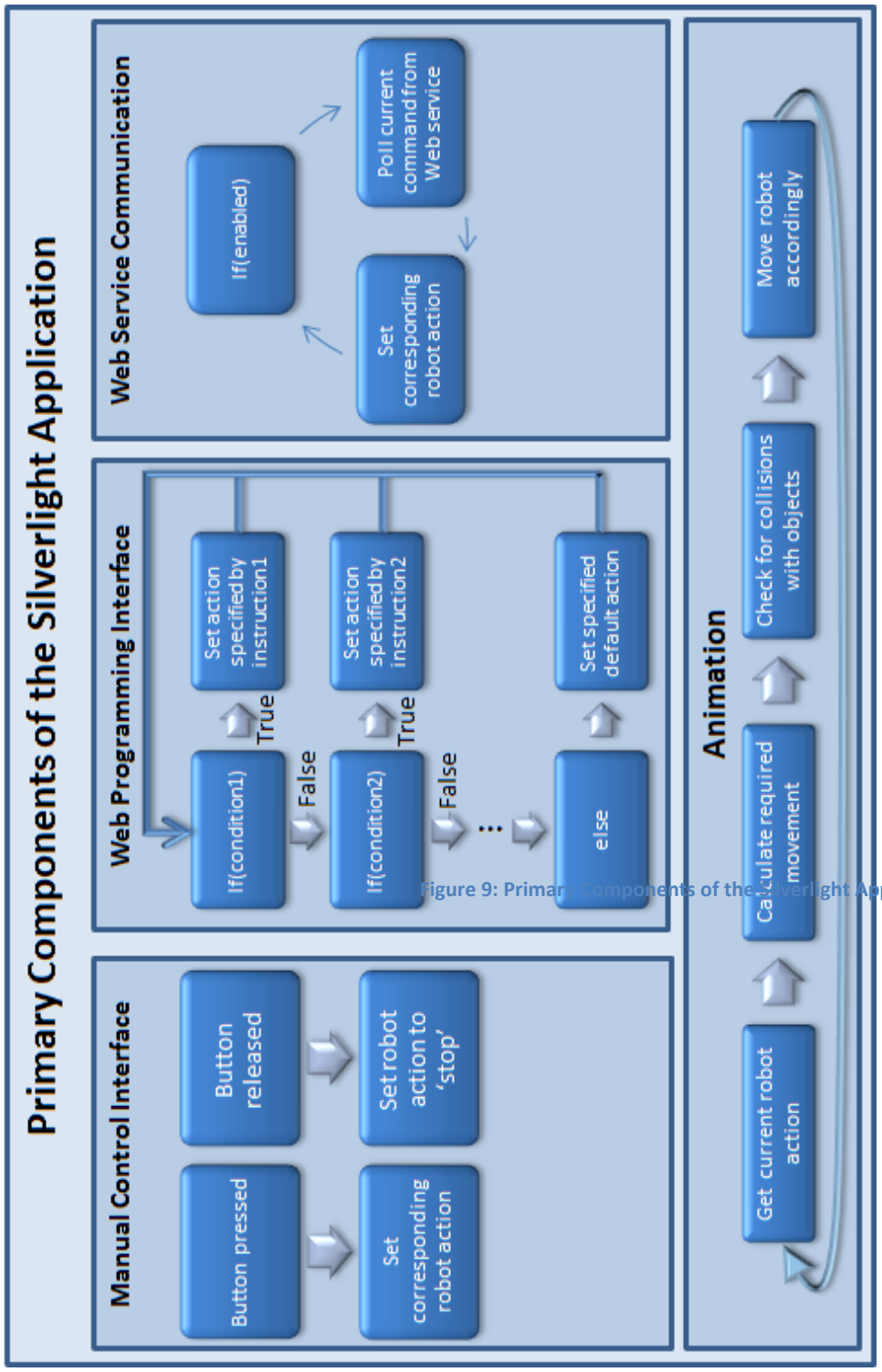

Figure 9: Primary Components of the Silverlight Application

The primary components of the Silverlight application are shown in Figure 9. CRC cards describing these components have been created and can be found in Appendix C. The Silverlight application has a thread which manages the animations that are to take place. These animations are dictated by other components of the simulator setting the current robot action. Based on the currently set robot action, the motion required to animate the action is calculated. After calculating the motion, new location, and orientation of the robot, the physics of the operation are evaluated. If the robot can move to the new location and be re-oriented as necessary without colliding with any objects, then the animation will continue. However, if it is determined that the robot would collide with an object, the robot is not moved. This simulates the robot colliding with an object. This entire process is constantly repeated as quickly as possible to help display a smooth animation, with up to a maximum frame rate of thirty frames per second.

The manual control interface primarily consists of buttons, each representing an action the robot can perform. When a button is pressed, the corresponding action for that button is set within the simulator. Similarly, when a user releases the button the current robot action is set to "stop", halting the robot until the next button is pressed.

The thread that manages the Web programming interface is used primarily for evaluating and executing the instructions specified by the user. Starting with the first instruction given, the condition specified with each instruction is evaluated. If the condition is true, the user-specified action for this instruction is set in the simulator. Furthermore, the subsequent instructions are not evaluated and, instead, the thread begins evaluating each of the instructions from the beginning. However, if an instruction's condition is evaluated to be false, the next instruction given is evaluated.

Yet another thread is used to carry out the communication between the Silverlight application and the Web service. First, the thread checks if the data from the

Web service is being requested by the user. If not, the thread does nothing but wait until this condition is met. If the data from the Web service is needed, then the thread calls one of the Web service's functions, which returns the current robot action requested by the client application. Once this data is received from the Web service, the simulator's current robot action is set as requested by the client application.

3.1. Web Service

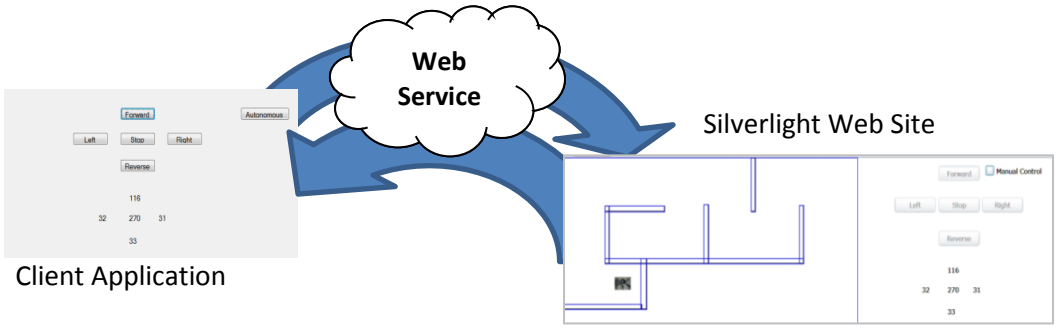

Figure 10: Component Communications

The sole purpose of the Web service is to receive, store, and fulfill requests for data. This is seen in two scenarios.

First, when the client application sends commands to the robot and second when the robot sends data (i.e. sensor readings) back to the client application. In a real world scenario, outside of the simulation environment, the Web service is usually hosted on the robot itself to reduce overhead and increase battery life by removing the need for the robot to regularly push/pull updates to/from the Web service. With the Web service hosted on the robot, as soon as a command is received by the Web service, it can directly call the code required to carry out the requested instruction.

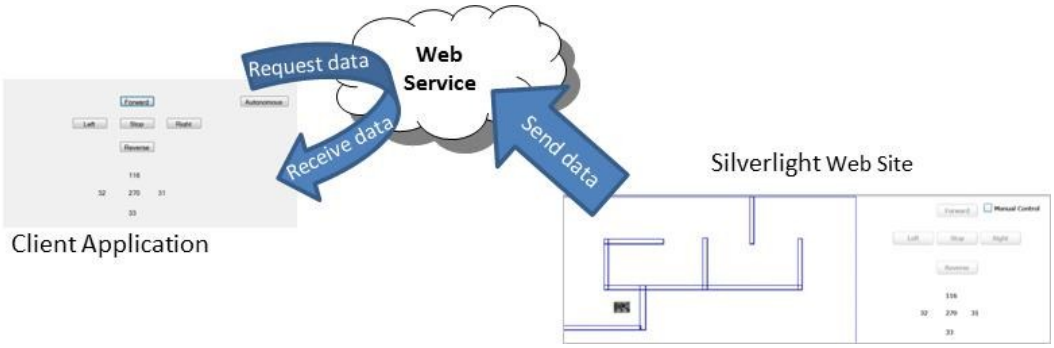

Figure 11: Receiving Data from the Simulator

From the perspective of the Silverlight application developed for this project, the simulated robot's sensor readings are periodically pushed to the Web service by the Silverlight application. When the readings are received by the Web service, the data is available to be pulled by the client application. This is analogous to the program (which hosts the Web service) running on a physical robot. The program is able to poll the robot's sensors and sends these values back to the client application via the return value of a function call.

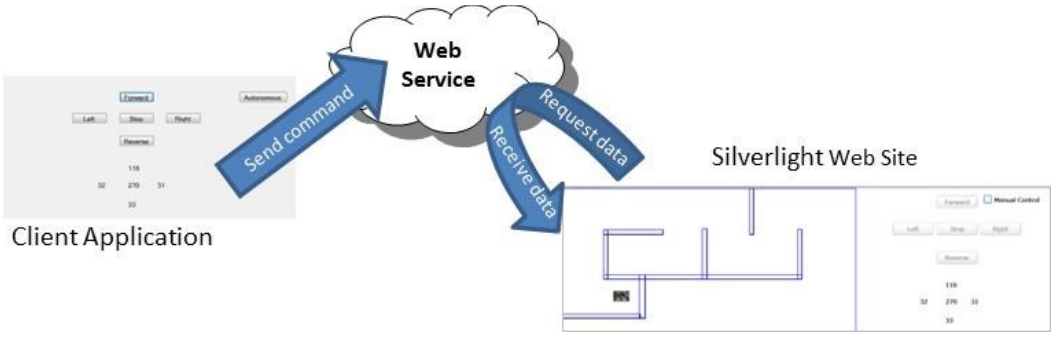

Figure 12: Simulator Receiving Commands from the Client Application

The reverse data path is also possible and is taken when the client application sends a command to the robot. This occurs when the client application makes a function call to a function provided by the Web service. In the case of the simulator, when the client application's function call is received by the Web service with a command for the robot, the command is stored and made available to be pulled by the Silverlight application where it will then executed by the simulated robot. This situation is

equivalent to the client application executing a function call to the Web service hosted on a physical robot. When the command is received by the Web service it can immediately begin executing the command on the robot.

The Web service is a key component of service-oriented robotics and this simulator. "Web services make functional building-blocks accessible over standard Internet protocols independent of platforms and programming languages. These services can represent either new applications or just wrappers around existing legacy systems to make them network-enabled" (Wikipedia contributors). In this case, the Web service is used primarily to wrap the existing functionality of the robot (i.e. movement) into a service that can be accessed by a client application via the Internet. The Web service also provides a means for the client application to receive data from the robot.

3.2. Client Application

The third component of the system is the client application. While the client application is not technically a part of the simulator itself, it is the application that utilizes the Web service provided by the simulator. The client application is what contains the logic and algorithms that drive the (simulated) robot to complete a particular task. The client sends each of its commands to and receives updates (such as sensor readings) from the Web service. As the client application receives updates from the robot it can then perform the (possibly very complex) calculations required to determine what the robot should perform next.

One of the valuable features of the simulator is that it can be simple to switch between controlling the physical robot and the simulated robot in the client application. Because the user's physical robot and the simulator both utilize a Web service to issue commands and receive data from the robot, the user will need to modify which Web service is referenced in their client application. Assuming the Web service hosted on the

user's physical robot uses the same function calls as the simulator's Web service, the user can simply modify the URL used to reference the Web service to switch between controlling the physical robot and the simulated robot. For this same reason, "[t]he languages used in writing the application and the services can be different, which allows us to write robotics applications using high-level languages such as Java and C#..." (Chen, Du, and Garcia-Acosta 152).

3.3. The Silverlight Website

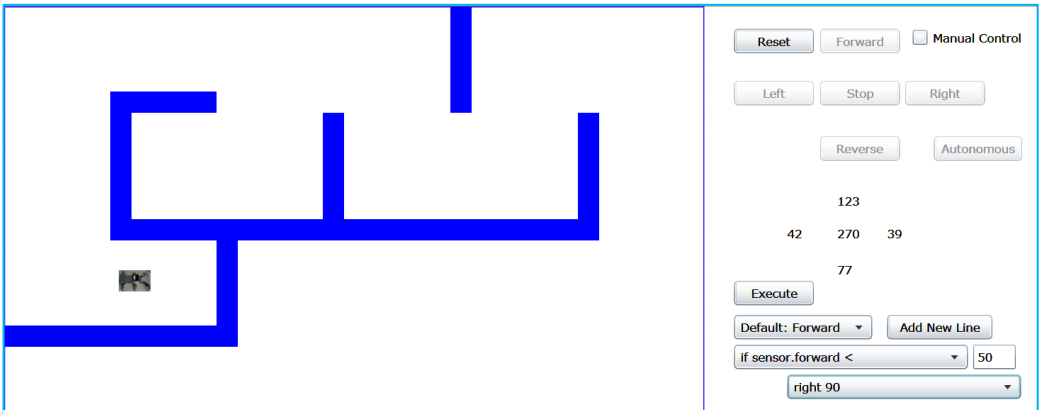

Figure 13: A Screenshot of the Silverlight Website

The website acts as a graphical user interface (GUI) and window into the virtual world of the simulator. The website hosts a Silverlight application which provides the primary functions that are typical in a simulator. The Silverlight application provides all of the simulator-related functionality provided on the website, including the physics engine, maze, robot, programming interface, etc. It enables the user to see the virtual robot, its surroundings, and the values read from the robot's sensors, as well as watch the robot carry out the commands it receives in real-time. Another useful feature is the ability for the user to simply click in the maze to add/remove blocks from the maze to customize it as needed.

3.3.1. Silverlight

The website is powered by a Silverlight application. Silverlight is described as "… a powerful tool for creating and delivering rich Internet applications and media experiences on the Web" (*About | Microsoft Silverlight*). Silverlight is provided as a plug-in to most major browsers and supports many operating systems. Microsoft also provides an SDK for developers to develop Silverlight applications, which can be run via a Web browser or directly on a Windows Phone as a Windows Phone Application. Silverlight's broad compatibility, available features, and development environment provided by Microsoft help make Silverlight a valuable component of this project.

3.3.2. The Maze, Robot, and Physics

The Silverlight application powers the maze, robot, and the physics engine. The physics engine enables the robot to collide with walls and the outer-boundaries of the simulated world. It also is the key component used to provide accurate sensor readings from the robot. The maze provides an area in which the robot can travel and can utilize the developer's programming. The simulator also gives the user the ability to click in the maze to add/remove blocks that constitute the walls of the maze. This enables the user to create a custom maze that may better fit their needs or provide him/her with a more challenging environment. The robot is the programmable entity within the simulation environment. The manual controls, commands received from the Web service, and the Web programming interface can be used to control the robot.

3.3.3. Web Programming Interface

The Web programming interface provides the user with the ability to send instructions to the robot (i.e. "forward," "turn left 90°," etc.). This programming interface is designed to easily enable young engineers to create and run simple programs directly on the website. This provides an intuitive method for new students to be exposed to programming, testing and debugging code, and the concepts of robotics programming. The interface will enable new students to learn the challenges and satisfaction of creating a program that can reliably control a robot and guide it through the maze successfully.

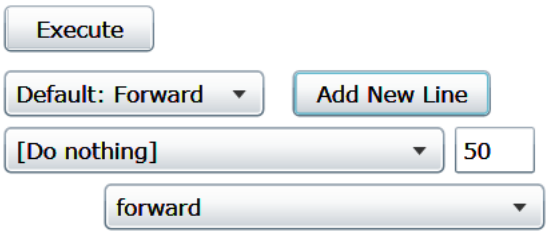

Figure 14: Results of the "Add New Line" Button

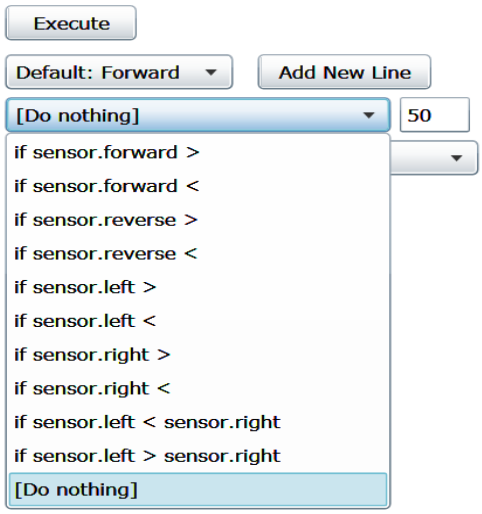

Figure 15: Options Available to Check Various Conditions

The programming interface is designed to be simple and succinct. As seen in Figure 14, the *Add New Line* button provides users with the ability to add as many lines as they need to program what they wish to implement. After clicking the *Add New Line*

button, two new dropdown boxes appear. The first dropdown box contains options for testing various conditions, such as checking whether the forward sensor is reading a value less than a user-specified value. (See Figure 15 for all of the options available.) This dropdown box is used in conjunction with the textbox to its right to form a complete conditional statement. The textbox provides a means for the user to specify particular distances/conditions to be met.

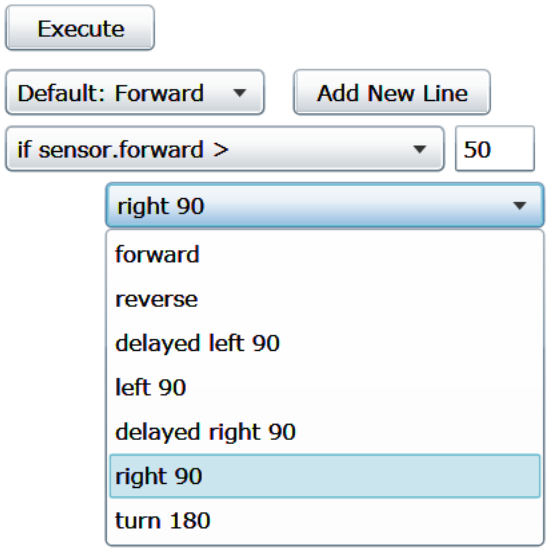

Figure 16: Actions Available to be Executed

The second dropdown box (see Figure 16) contains a list of actions that can be taken when the condition (specified by the first dropdown box and the associated textbox) is met. Each of the actions available is fairly simple and self-explanatory, with the exception of the *delayed*-type actions. The *delayed left 90* and *delayed right 90* actions perform the same actions as *left 90* and *right 90* respectively, but with one slight difference: a delay before executing the turn. These actions are useful, for example, when the robot is travelling forward and detects that it can and should turn right.

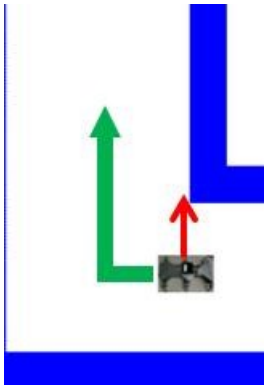

Figure 17: Example D*elayed*-type Action

As seen in Figure 17, turning immediately would cause the robot to collide with the side of the new hallway because the robot did not attempt to center itself, in relation to the new hallway, before entering it. In this example, by utilizing the *delayed right 90* the robot continues moving forward for a short time before turning right. This prevents the robot from colliding with the side of the new hallway.

Each instruction that is added represents an if-then-else statement. By adding multiple instructions, users can create a program with the following structure:

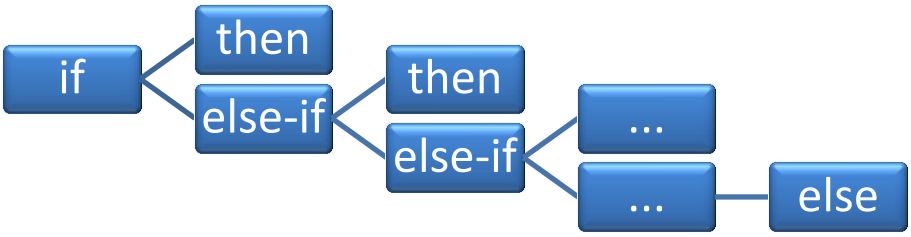

Figure 18: If-then-else Structure of the Web Programming Interface

As a result, if a condition is met, the corresponding action will be taken and none of the following instructions will be evaluated. On the other hand, if none of the listed conditions are met, then the default action will be executed. The program is executed infinitely (unless stopped by the user), as if in an infinite loop. For example, in Figure 19, the program states that if the forward sensor receives a reading less than 50 it is to turn around (i.e. turn 180^{o}). Otherwise, it is to go forward (the default action). As a

result, the robot will travel forward until it reaches a wall and then turn around. This will result in the robot going back and forth, from wall to wall, indefinitely.

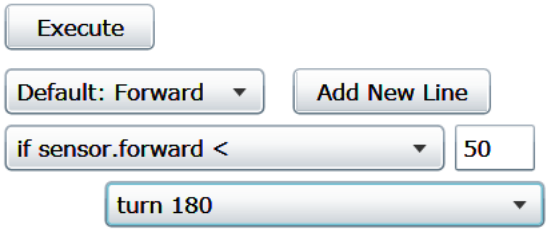

Figure 19: Example Program

An easily overlooked feature of this programming interface is its availability via a Web site. This makes it easy for students to access (from almost anywhere) and from virtually any computer with an Internet connection. It also provides students with a familiar environment by being located on a webpage. Most students will have undoubtedly spent a considerable amount of time on the Internet, so this environment is a natural one to them. The website's programming interface also provides an engaging learning experience, as students are able to execute their program easily on the website and receive immediate feedback as they watch the simulated robot carry out their instructions.

CHAPTER 4

DATA COLLECTION AND ANALYSIS

Data was collected and analyzed to determine how well inexperienced programming students could interact with, and learn from, the programming interface on the website. For the data collection process, a total of 28 students (23 freshman and 5 sophomore) students were studied while completing their assigned tasks. A short presentation (approximately 10 minutes) was given to introduce the website and the programming interface. The students were then asked to program, one at a time, two algorithms: a right-wall following algorithm and a farthest-distance algorithm. Each algorithm was explained to the students during the presentation before they were asked to begin programming.

4.1. Research Objectives

This experiment was conducted to answer the following research questions:

1. What is the impact of Web-based visual programming in helping students understand programming logic?
2. Does a user-friendly Web-based interactive tool help students understand human-computer interaction and program construction?
3. Does the Web-based program provide an interesting feedback mechanism from which students can evaluate their software?

To answer these issues, they were broken down into the following questions. The answers to these questions can be found using data which can be measured and observed during the experiment. The following lists the research questions and how they have been broken down for this research.

1. What is the impact of Web-based visual programming in helping students understand programming logic?

a. Do students place their instructions in the correct order in their solutions?
b. Do students have questions regarding the if-then-else structure of their program?
c. Were there questions regarding the loops in or flow of the program?
d. Was each student able to develop a correct solution?

2. Does a user-friendly Web-based interactive tool help students understand human-computer interaction and program construction?
    a. Did students use proper robot commands to implement their program?
    b. Were *delayed*-type commands used in the proper situations?
    c. Is each student's code correct and complete?
3. Does the Web-based program provide an interesting feedback mechanism from which students can evaluate their software?
    a. Were students able to determine when their code was correct or incorrect?
    b. Were students able to use the programming interface with little to no help from the instructor?
    c. Were the students able to program the algorithms within the given time limits?

4.2. Problems to be Solved by Students

The right-wall following algorithm is a fairly simple algorithm while the farthest-distance algorithm is more complex.

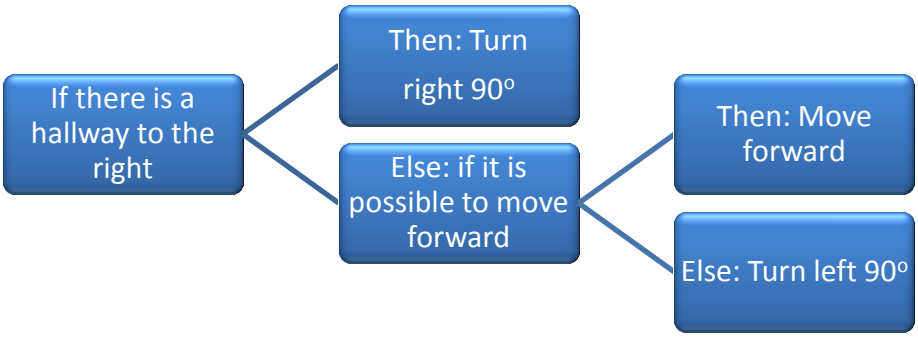

Figure 20: Right-Wall Following Algorithm

The right-wall following algorithm states: 1) if a hallway is detected 90° to the right then turn right 90°. 2) If it possible to move forward then move forward. 3) If none of the other conditions are true then turn left 90°. In other words, go forward while turning right whenever possible, but turning left when it is not possible to go forward any further.

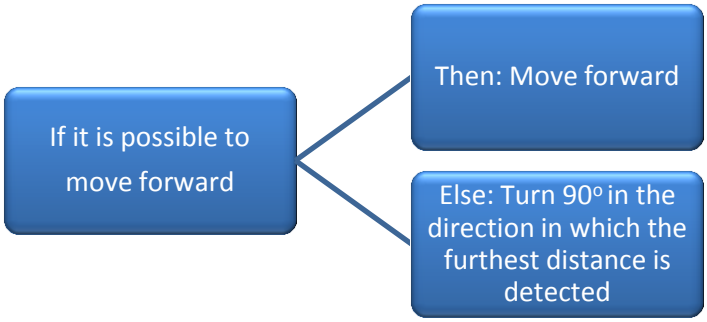

Figure 21: Farthest-Distance Algorithm

The farthest-distance algorithm states: 1) go forward whenever possible. 2) If you cannot go forward then turn in the direction in which you can see the furthest. For example: turn left 90° if the distance to your left is greater than the distance to your right. Otherwise turn right 90°. In other words, go forward until you cannot go forward any further. When you cannot go forward turn left or right, whichever allows you to travel the furthest distance without turning again.

4.3. Types of Data Gathered

As the students programmed via the Web programming interface, the mistakes in incorrect solutions and any questions were noted. The *Evaluation* table provides a list of

various skills, achievements, and specific actions that were to be evaluated by instructor grading. As other notable events occurred, they too were noted by the instructor. When each student completed an algorithm, the following information was recorded: the number of lines of code the student used to implement the algorithm and the time it took him/her to program a correct solution.

Table 5: Evaluation Process During Data Collection

Evaluation Process During Data Collection			
Skill	**Criteria**	**To Observe**	**Analyze**
Program Control Flow Logic	Students recognize and correctly utilize the if-then-else structure, loops, and flow of the program to construct software.	Do students become confused regarding the control flow of the program or write invalid logic by placing their code in the wrong order?	1) Did students place their instructions in the correct order in their solutions?
			2) Did students have questions regarding the if-then-else structure of their program?
			3) Were there questions regarding the loops in or flow of the program?
			4) Was each student able to develop a correct solution?
Commands for Robot Navigation	Students understand the correct uses for the robot's commands.	Did students use an incorrect robot command in an attempt to complete an action?	5) Did students use proper robot commands to implement their program?
			6) Were *delayed*-type commands used in the proper situations?
Evaluating Algorithm Correctness	Students are able to determine when their program is or is not working correctly.	Examine each student's program for correctness when they believe they are finished.	7) Is each student's code correct and complete?
		How many times do students believe he/she has the correct solution but is incorrect?	8) Were students able to determine when their code was correct or incorrect?
Human-Computer Interaction	Students learn to use and interact with the website and are able to quickly understand the feedback provided by it.	Do students struggle to understand the website or the programming interface?	9) Were students able to use the programming interface with little to no help from the instructor?
			10) Were the students able to program the algorithms within the given time limits?

Each student was also evaluated individually by (1) the student's ability to solve the programming problem within the time limit and (2) the time taken to successfully program the assigned algorithm (if applicable). Comments and feedback received from the students were also noted. For additional feedback from the students, they were also asked to complete a survey.

After each of the students finished programming the algorithms, they were given a rating sheet. Using this survey, each student rated how well the website helped them learn and understand various skills and concepts. The rating sheet asked the students to rate these items on a scale from 1 (very little to no help) to 5 (very helpful). The results from the data gathered can be seen in the following tables and graphs. The raw data can be found in Appendix A.

4.4. Possible Outcomes of Student Projects

The table above describes what the instructor should look for during the data collection process. Below are some positive and negative examples that relate to the questions stated in the previous table. Where applicable, figures have been included containing possible implementations that demonstrate the respective example.

4.4.1. Questions 1-4

Positive Examples: Students placed their instructions in the correct order (Figure 22 and Figure 23) without asking for help. Students were able to implement correct solutions for the algorithms (Figure 22 and Figure 23). Students did not ask questions regarding the if-then-else structure, loops, or flow of their program.

Negative Examples: Students asked for help from the instructor to place their instructions in the correct order. Students did not implement a correct solution for an

algorithm within the time limit. Students asked questions regarding the if-then-else structure, loops, or flow of their program.

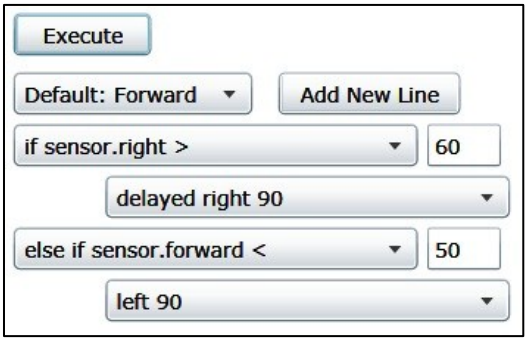

Figure 22: A Correct Solution for the Right-Wall Following Algorithm

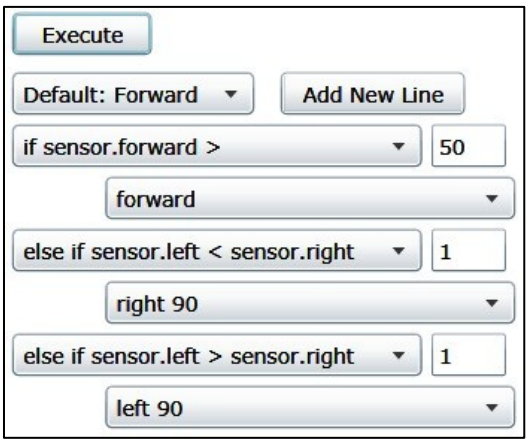

Figure 23: A Correct Solution for the Farthest-Distance Algorithm

4.4.2. Questions 5-6

Positive Examples: Students used the correct robot commands in their programs and utilized the *delayed*-type commands in the correct situations (Figure 22 and Figure 23). Negative Examples: Students' programs were incorrect due to the improper use of robot commands (Figure 24). *Delayed*-type commands were not used by the students in the proper situations (Figure 25).

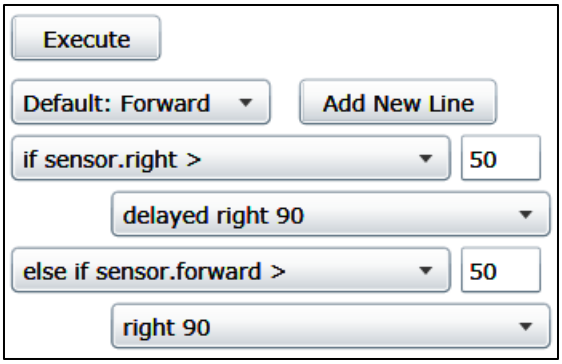

Figure 24: An Incorrect Solution due to Improper Use of the Robot's Commands

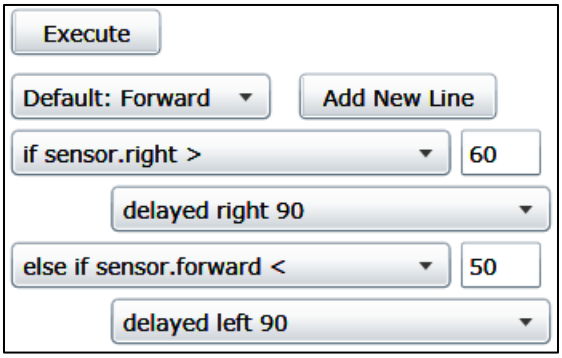

Figure 25: An Incorrect Solution due to Improper Use of a *Delayed*-type Command

4.4.3. Questions 7-8

Positive Examples: Students successfully implemented a correct program for each algorithm (Figure 22 and Figure 23). Students' programs were correct before asking the instructor to verify them. Negative Examples: Students were unable to implement a correct program for an algorithm (Figure 26). The students asked the instructor to verify their solution, but their solution was incorrect.

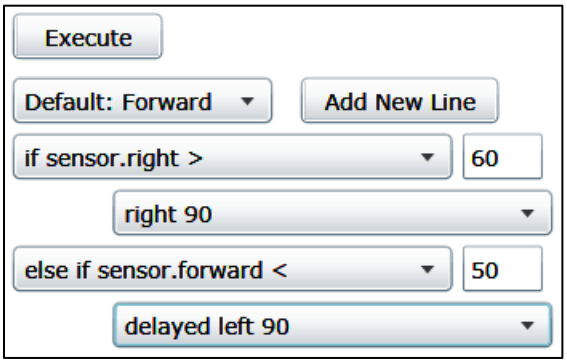

Figure 26: An Incorrect Solution for the Right-Wall Following Algorithm

4.4.4. Questions 9-10

Positive Examples: Students were able to utilize the Web programming interface without receiving help from the instructor. Students were able to implement correct programs for the algorithms (Figure 22 and Figure 23) within the time limit. Negative Examples: Students required help from the instructor to utilize the programming interface. Students were unable to implement a correct solution for an algorithm within the time limit.

4.5. Data Collected

Table 6: Data Collected Per Algorithm

Algorithm	Average Time Taken	Average Lines of Code Used	Student Success Rate
Right-Wall Following Algorithm	5 minutes 17 seconds	5	100%
Furthest-Distance Algorithm	3 minutes 38 seconds	7	100%

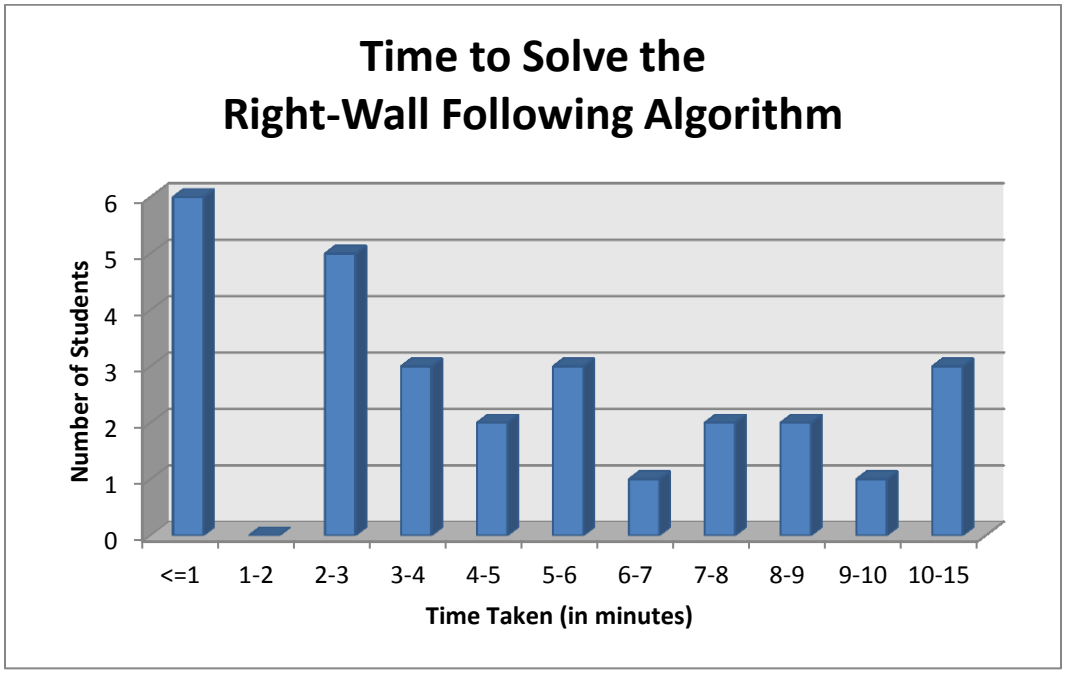

Figure 27: Time to Solve the Right-Wall-Following Algorithm

The average time needed to solve the right-wall following algorithm was 5 minutes and 17 seconds. The standard deviation is 3 minutes and 53 seconds. The maximum time required was 14 minutes, while the minimum time needed was 30 seconds.

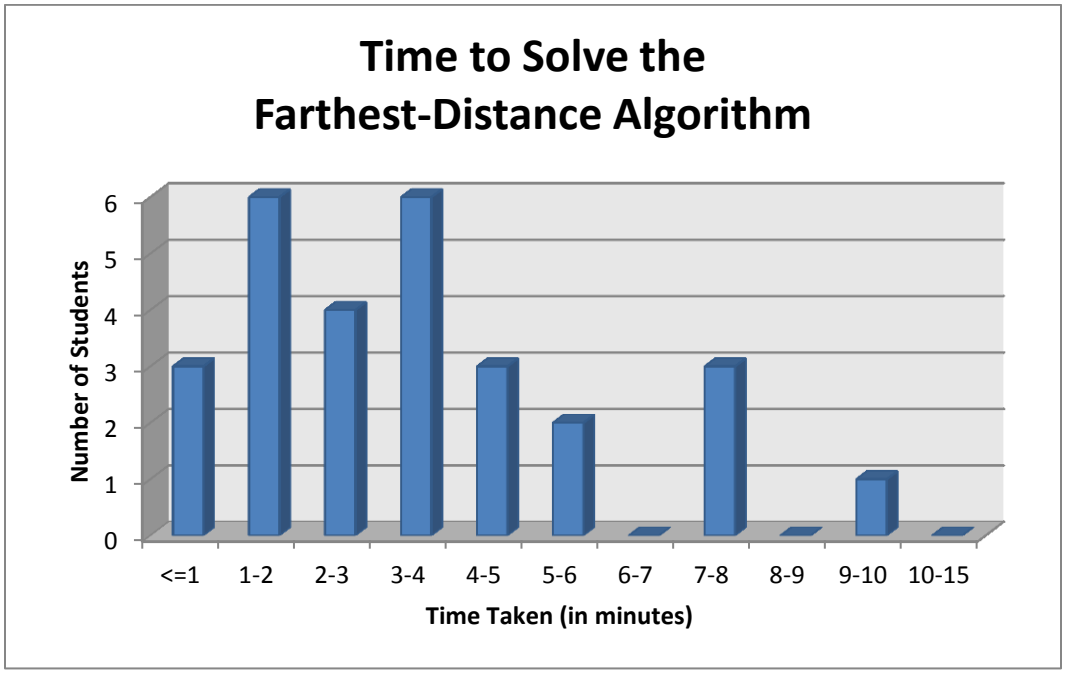

Figure 28: Time to Solve the Farthest-Distance Algorithm

The average time needed to solve the farthest-distance algorithm was 3 minutes and 48 seconds. The standard deviation is 2 minutes and 15 seconds. The maximum time required was 9 minutes and 30 seconds, while the minimum time needed was 45 seconds.

Table 7: Instructor Observations and Analysis

Results			
Skill	**Average Rating by Students*** (1 little to no help – 5 very helpful)	**Observations**	**Analysis**
Program Control Flow Logic	4.0	Students correctly utilized the program's control flow logic. One case was noted where instructions were initially placed in the wrong order. All of the students successfully implemented the algorithms.	Each student was successful in implementing the given algorithms. The average rating received from the students was positive. Only one of twenty-eight (3.6%) students had a fault regarding the program's control flow logic.
Commands for Robot Navigation	4.3	Ultimately, students used the correct commands to implement the algorithms presented to them. Three of twenty-eight (10.7%) students did not initially use a *delayed*-type instruction when needed. Some students had trouble determining the amount of distance represented by the sensor readings and programming inputs.	All of the students successfully developed correct solutions. The average rating received indicates that students found the interface to be helpful to them in this regard. Improvements could be considered to help users understand how the units of measurement relate to the virtual world.
Evaluating Algorithm Correctness	3.9	Ultimately, each student's code for each algorithm was correct. Three of twenty-eight (10.7%) students mistakenly believed they had implemented the correct solution. None of the students repeatedly asked questions regarding the correctness of their program.	Students did not repeatedly mistake an incorrect implementation for a correct solution. All of the students successfully implemented the algorithms. The average rating by students also illustrates that the students found the simulator to be notably helpful when evaluating their algorithms.
Human-Computer Interaction	4.1	Students were quick to begin utilizing the programming interface and write code. No questions were asked regarding the use of the interface or for explanations of the robot's actions.	Students were able to utilize the programming interface without the need for asking questions. Furthermore, students were able to solve each algorithm in less than six minutes on average. The average student rating was positive.
* See the figures below for a breakdown of the ratings received.			

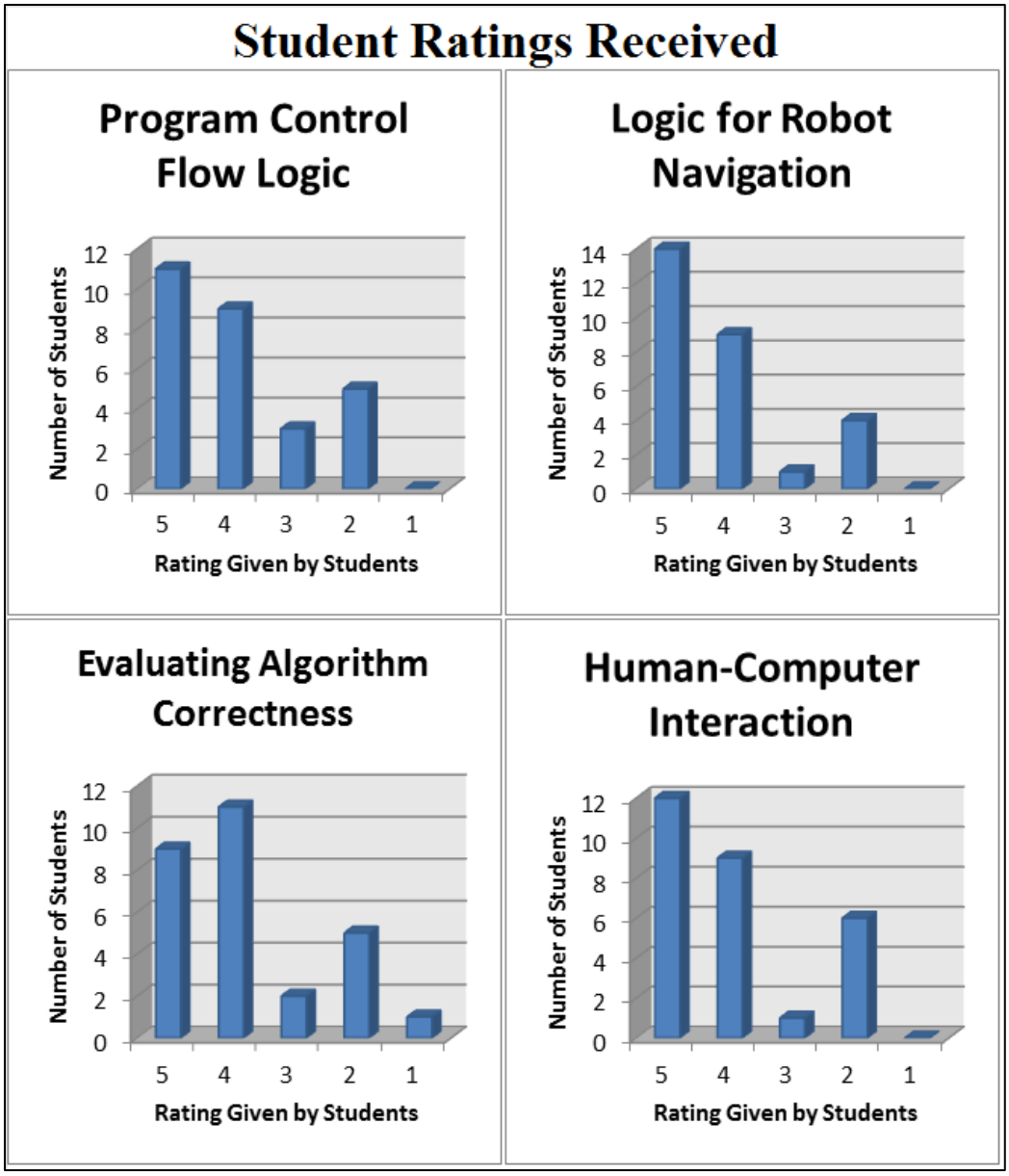

Figure 29: Breakdown of the Student Ratings Received

The research questions were broken down into ten new questions. These questions, and relevant data collected during the research, are presented in the following text. (1) Did students place their instructions in the correct order in their solutions? It was observed, during the research, that the students did accurately use the program's control flow logic. Only one of twenty-eight (3.6%) students believed they had a correct solution when instructions were placed in a wrong order.

(2) Did students have questions regarding the if-then-else structure of their program? (3) Were there questions regarding the loops in or flow of the program? Students did not ask questions regarding these topics. However, as noted above, one student did believe he had a correct solution, but his program contained instructions in the wrong order. This error may have occurred due to an incomplete understanding of the topics presented in questions (2) and (3). However, in regards to question (4) (was each student able to develop a correct solution?); all of the students were able to successfully implement the algorithms presented to them.

(5) Did students use proper robot commands to implement their program? (6) Were *delayed*-type commands used in the proper situations? Three of twenty-eight (10.7%) students did not initially use a *delayed*-type instruction in their solutions when needed. However, these students learned from their mistakes and, in the end, all of the students used the proper commands to implement the algorithms. (7) Is each student's code correct and complete? Ultimately, each student implemented a correct solution for each of the algorithms presented to them.

(8) Were students able to determine when their code was correct or incorrect? During the experiment, none of the students repeatedly believed they had implemented a correct solution when they had not. Students were able to regularly determine when their programs were or were not correct.

(9) Were students able to use the programming interface with little to no help from the instructor? (10) Were the students able to program the algorithms within the given time limits? Students immediately began working with the Web programming interface without any significant problems. None of the students asked questions with regards to interaction with the programming interface. Additionally, students solved each of the algorithms presented to them in less than six minutes on average.

4.6. Evaluation

For this experiment, students were monitored while they utilized the simulator and its Web programming interface. Initially, students were given a short presentation to introduce them to the website, the simulator, and the programming interface. Students were then shown two different algorithms and were asked to program each algorithm using the simulator's Web programming interface. As the students programmed the algorithms, they were supervised and helped answer any questions they had. Other statistics were also recorded, such as how long each student needed to implement a correct solution for each of the given algorithms as well as the number of lines used. From the data gathered and surveys received, the results indicate that the students appreciated the programming interface provided on the website. Few questions were asked by the students regarding the Web programming interface.

A few students had some trouble in a couple of areas. A small number of students communicated that they had trouble determining what part of their code was causing the error they were experiencing. While this type of problem appears in any programming language or environment, improvements to the Web programming interface may be possible to help students locate an error in their code. With regards to *delayed*-type instructions, only a few students used an improper instruction where a *delayed*-type instruction was needed. This may be reduced by placing greater emphasis on these instructions during the presentation used to introduce the students to the website.

In almost every category, the students found the website particularly helpful to them in understanding and learning various principles and skills. Furthermore, many expressed interest in using the software again. Some students said that they enjoyed using the software, a number of them calling it "fun."

The data also reveals that students were able to successfully program the second, and slightly more complex, algorithm in significantly less time, on average, than the first algorithm. This is possibly due to the students becoming more familiar with the programming environment as they completed their first program. As a result, students were able to more efficiently design, implement, and test their second program.

CHAPTER 5

SUMMARY

As service-oriented robots become more popular and are utilized in new applications, there will be a greater need for a simulator which supports such robots. The simulator described in this document not only provides such support, but also includes a Web-accessible programming interface which aids new students in learning topics which are related to robotics programming. To determine the effectiveness of this Web programming interface, students were monitored while programming algorithms using this interface. The results from the data gathered were positive.

Table 8: Summary

Summary			
Research Question	**CSTA Topic**	**Major Finding**	**Recommendations**
What is the impact of Web-based visual programming in helping students understand programming logic?	Understanding and exploring the problem (Madden et al. 7). Design, code, and test a solution (Madden et al. 23). "The power of stepwise refinement" (Madden et al. 7). "Graphs" (Madden et al. 12).	The website was found to be an effective tool in teaching new students many of the foundational concepts in the field of robotics programming.	To help introduce the basic concepts of programming logic, students and educators should utilize visual programming languages instead of text-based languages.
Does a user-friendly Web-based interactive tool help students understand human-computer interaction and program construction?	"Interactive Programming" (Madden et al. 8).	The students were able to start using the interactive programming interface quickly and program different algorithms, each in a matter of a few minutes on average.	Students and educators should employ an interactive tool which encourages 'learning by doing' to aid students in the understanding of program construction and human-computer interaction.
Is the Web-based program useful in providing an interesting feedback mechanism for students to evaluate their software?	"Identify elements of user-friendly Web sites" (Madden et al. 14).	The website provided clear feedback to the students enabling them to have little trouble while solving the algorithms and correcting any mistakes they made.	Students and educators should consider using a program with apparent and immediate feedback to aid students in software evaluation.

As seen in the above table, by answering the research questions, developed using CSTA guidelines, interesting findings were discovered. The initial question considered was: what is the impact of Web-based visual programming in helping students understand programming logic? By exploring this issue, it was found that the website was an effective tool and aided students in learning programming concepts. As a result, it is recommended that students and educators employ a visual programming language, rather than a text-based language, to aid the learning of basic concepts of programming logic.

The second question asked was: does a user-friendly Web-based interactive tool help students understand human-computer interaction and program construction? By observing the way in which students utilized the interactive programming interface, it was determined that the students were able to quickly begin programming and rapidly implement correct solutions for the algorithms. Consequently, it is recommended that while educating students regarding program construction and human-computer interaction, interactive tools be used which engage students and promote user involvement.

Lastly, the final question considered was: is the Web-based program useful in providing an interesting feedback mechanism for students to evaluate their software? By examining this, it was discovered that students were able to easily solve the algorithms and correct any mistakes. This was promoted by the clear and instant feedback provided by the website. Therefore, when evaluating software, students and educators should select educational programs which employ immediate and understandable feedback.

CHAPTER 6

FUTURE WORK

Development has already begun on a lightweight version of the simulator that runs on Windows Phones. The lightweight version does not provide the programming interfaces available in the full version. The Windows Phone version does, however, provide the user with the ability to modify the maze and manually control the robot. Additionally, this version will provide the user with several pre-programmed algorithms and mazes to choose from. This allows the students to experiment with and explore different scenarios and configurations and learn how these algorithms perform in various situations. Progress been made by Arizona State University students on a similar project that runs on a website and enables students to see how the robot navigates a maze when using any of the four algorithms available ("HexCrawlerSim2").

In the future, additional features could be added to the simulator presented in this paper. By evaluating the feedback received from students, it was determined that it may be beneficial to add a feature that would enable students to see what code is currently being executed. This would provide greater feedback to the student and help them debug their code. The comments received also revealed that some students had trouble relating the scale of the units of measurement to the maze. A scale could be added to the simulator (similar to a legend on a map) that would show the physical representation of the distance depicted by different amounts of measure.

REFERENCES

"About." *About - Algodoo*. Algoryx Simulation AB. 2012. Web. 1 December, 2012. <http://www.algodoo.com/wiki/About>.

"ASU Repository." Arizona State University. n.d. Web. 5 November, 2012. <http://venus.eas.asu.edu/WSRepository/repository.html>.

"Call for Papers." *SOCA 2012*. IEEE International Conference on Service-Oriented Computing and Applications. n.d. Web. 26 November, 2012. <http://conferences.computer.org/soca/2012/cfp.htm>.

Chen, Yinong, Zhihui Du, and Marcos Garcia-Acosta. "Robot as a Service in Cloud Computing." *In Proceedings of the Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE)*. Nanjing: June 4-5, 2010. Pages 151-158. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5570010&isnumber=5569889>.

Chen, Yinong, Wei-Tek Tsai. *Service-Oriented Computing and Web Software Integration*. 3rd edition. Iowa: Kendall Hunt Publishing. 2011.

Chen, Yinong, Wei-Tek Tsai. "Towards Dependable Service-Oriented Computing Systems." *Simulation Modelling Practice and Theory*, vol. 17, 2009. Pages 1361-1366. <http://www.public.asu.edu/~ychen10/fullpaper/ChenTsaiSimpat09.pdf>.

Collins, A., and R. Halverson. "The second educational revolution: rethinking education in the age of technology." *Journal of Computer Assisted Learning*, vol. 26, no.1. Blackwell Publishing Ltd. 12 January, 2010. <http://onlinelibrary.wiley.com.ezproxy1.lib.asu.edu/doi/10.1111/j.1365-2729.2009.00339.x/full>.

"Frequently Asked Questions." *About | Microsoft Silverlight*. n.d. Web. 4 November, 2012. <http://www.microsoft.com/silverlight/what-is-silverlight/>.

"HexCrawlerSim2." Arizona State University. n.d. Web. 5 November, 2012. <http://venus.eas.asu.edu/WSRepository/RaaS/Hexcrawler/>.

Lee, Yann-Hang, et al. "A code generation and execution environment for service-oriented smart home solutions." *Service-Oriented Computing and Applications (SOCA), 2009 IEEE International Conference on*, 14-15 Jan. 2009. Pages 1-8. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5410277&isnumber=5410249>.

Madden, Bill, et al. *A Model Curriculum for K-12 Computer Science Level III Objectives and Outlines*. *Computer Science Teachers Association (CSTA)*. *Computer Science Teachers Association (CSTA)*, 2007. Web. 30 November, 2012. <http://csta.acm.org/Curriculum/sub/CurrFiles/L3-Objectives-and-Outlines.pdf>.

"Microsoft Robotics Developer Studio." n.d. Web. 3 November, 2012. <http://www.microsoft.com/robotics/#Product>.

"Robotics Camp." Arizona State University. n.d. Web. 5 November, 2012. <http://venus.eas.asu.edu/roboticscamp/>.

Tsai, Wei-Tek, et al. "An Introductory Course on Service-Oriented Computing for High Schools." *Journal of Information Technology Education*, vol. 7, 2008. Pages 315-338. <http://jite.org/documents/Vol7/JITEv7p323-346Tsai378.pdf>.

Tsai, Wei-Tek, et al. "Collaborative Learning Using Wiki Web Sites for Computer Science Undergraduate Education: A Case Study," *Education, IEEE Transactions on*, vol.54, no.1. February 2011. Pages 114-124. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5454284&isnumber=5705620>.

Verno, Anita and Brian Fuschetto. *CSTA National Standards and their Impact on the Future of K-12 Computer Education*. *Computer Science Teachers Association (CSTA)*. N.p. n.d. Web. 30 November, 2012. <http://cs4hs.cs.rutgers.edu/Fuschetta_Rutgers-NationalCSTAStandards%20(2).pptx>.

"Virtual Cell Modeling & Analysis Software." *VCell – The Virtual Cell*. University of Connecticut Health Center. n.d. Web. 1 December, 2012. <http://www.nrcam.uchc.edu/>.

Web Services Architecture Working Group. "Web Services Architecture." 11 February, 2004. Web. 3 November, 2012. <http://www.w3.org/TR/ws-arch/>.

"What is Alice?" Carnegie Mellon University. n.d. Web. 3 November, 2012. <http://www.alice.org/index.php?page=what_is_alice/what_is_alice>.

"What is LogicWorks 5?" *Capilano Computing Systems*. Capilano Computing Systems Ltd. 2011. Web. 1 December, 2012. <http://www.capilano.com/lww_5>.

"What is X-Plane?" *Meet X-Plane | X-Plane.com*. N.p. n.d. Web. 1 December, 2012. <http://www.x-plane.com/desktop/meet_x-plane/>.

Wikipedia contributors. "Service-oriented architecture." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 31 Oct. 2012. Web. 6 Nov. 2012.

APPENDIX A

RAW DATA COLLECTED

Table 9: Raw Data - Student Survey Results

Program Control Flow Logic	Logic for Robot Navigation	Evaluating Algorithm Correctness	Human-Computer Interaction
5	5	5	4
5	5	4	3
5	5	4	3
4	4	5	4
4	3	4	4
4	4	4	5
5	5	3	5
5	5	5	5
3	5	3	3
4	3	3	4
5	5	5	5
2	5	4	4
4	2	2	4
2	4	1	5
3	3	3	3
4	4	5	5
3	5	4	5
5	5	4	5
3	4	5	3
4	3	3	4
5	4	4	4
3	4	2	2
4	4	4	3
2	4	4	4
4	5	5	5
5	5	5	5
5	5	5	5
5	5	4	5

Table 10: Raw Data - Time Taken to Solve Each Algorithm (in seconds)

Right-Wall Following Algorithm	Farthest-Distance Algorithm
30	240
30	240
40	360
60	240
60	270
60	240
135	120
150	570
180	70
180	425
180	425
210	100
240	480
240	90
300	90
300	180
330	360
360	180
360	120
420	45
450	45
480	300
540	180
540	300
600	180
780	225
780	60
840	240

APPENDIX B

SAMPLE STUDENT SURVEY

(Note: Some of the topics in the survey were later removed from the scope of this paper.)

(Circle one) Year in school: Freshman Sophomore Junior Senior Graduate

Please feel free to add comments/suggestions in the space provided. Please provide feedback for ratings less than 5.

On a scale from 1 (very little to no help) to 5 (very helpful), how well does this software help you learn or better understand:

1. Program Control Flow Logic (if-then-else, loops)

2. Composing Code

3. Logic for Robot Navigation

4. Evaluating Algorithm Correctness

5. Debugging

6. Human-Computer Interaction

7. Software Engineering Design Process
   (i.e. Define the problem, design a solution, test the solution (repeat))

Figure 30: Student Survey

APPENDIX C

CRC CARDS

Table 11: CRC Card for the Animation Component

Animation Component	
Responsibilities	**Collaborators**
Get the currently assigned robot action/motion	motion
Calculate the requested motion of the robot	theRobot
Perform collision detection	theRobot theWorld
Move the robot on screen	theRobot

Table 12: CRC Card for the Manual Control Interface

Manual Control Interface	
Responsibilities	**Collaborators**
Provide users with buttons from which they can control the simulated robot	forwardButton leftButton rightButton reverseButton stopButton
Set the current robot motion to the action described by the button clicked by the user	motion
Stop the robot when the user releases their currently selected button	motion

Table 13: CRC Card for the Web Programming Interface

Web Programming Interface	
Responsibilities	**Collaborators**
Provide users with an interface from which they can develop a program capable of controlling the robot	executeProgrammingButton addNewLineButton defaultActionComboBox addedDropdowns addedTextBoxes
Execute the program, developed by the user, described in the Web programming interface	motion addedDropdowns addedTextBoxes

Table 14: CRC Card for the executeProgrammingButton

executeProgrammingButton	
Responsibilities	**Collaborators**
Executing the code developed by the users via the Web programming interface	defaultActionComboBox addedDropdowns addedTextBoxes motion

Table 15: CRC Card for the addNewLineButton

addNewLineButton	
Responsibilities	**Collaborators**
Adding a new pair of dropdown boxes for the user to use in the development of their program	addedDropdowns addedTextBoxes

Table 16: CRC Card for the Web Service Communication Component

Web Service Communication Component	
Responsibilities	**Collaborators**
Poll the Web service for up-to-date instructions	Web service
Set the current robot motion in accordance to the instruction received from the Web service	motion