InterAcText: Introduction to Java Programming, a Computer-Based Interactive Java Programming Textbook

Charles Bryan
Computing and Software Systems
Institute of Technology
University of Washington Tacoma
cfb3@u.washington.edu

MS CSS Capstone Design Project in Computing and Software Systems
Winter 2010

Committee Chair
Dr. Donald Chinn

Committee Member
Dr. Josh Tenenberg

Abstract

This paper introduces an educational tool used to facilitate learning Java programming. InterAcText is a computer-based Java programming textbook. InterAcText includes several forms of interactive tools that engage readers and encourage them to become part of the learning process. The software for InterAcText is being created using an Iterative Software design approach. This paper discusses the functionality and design of the first release of InterAcText, the results of the first evaluation of the text and accompanying interactive tools, and concludes with a discussion on what features and fixes will be implemented in the next iteration.

1. Introduction

While teaching an introductory Java course, we noticed that students seldom, if ever, executed the given source code examples from the textbook. Many confessed to completely skipping over the text that referred to the examples or even worse, not reading the textbook at all. At the same time, we received positive feedback when source code examples were covered in class. The students were also receptive to active learning exercises performed during class.

This paper describes a tool created to bridge the gap between in-class interactive activities and outside-of-class student reading. *InterAcText: Introduction to Java Programming* is a desktop application that integrates an introductory programming text with interactive tools. The interactive tools were created to mimic an in-class experience. As an all-in-one solution, we believe students will be more likely to explore the code examples presented in the text. We also believe the interactive aspects of the tool will keep the student interest high.

In InterAcText's current form, the two interactive tools are a Programming Visualization (PV), InterActive Code, and a quiz tool, Drill Sergeant. InterActive Code is similar to other program visualizations. A user may step forward and backward through a source code example, and at each step different aspects of the code are graphically represented. In Drill Sergeant, multiple choice questions are presented to the user. Each question reinforces topics previously learned.

The text that accompanies InterAcText was written specifically for the project. In the current form, there are one and a half chapters. The first covers Boolean expressions and if statements. The second chapter covers loops, but only the `for` loop is covered.

Several other types of tools attempt to use interaction to teach novice users how to program. ViLLE [**1**], Jeliot3 [**2**], and BlueJ [**3**] are all forms of PV. Alice and Scratch are graphical programming development environments. Most all of these tools succeed in raising the motivation and learning of novice programmers. Several textbooks, such as those of Lewis & DePasquale [**4**], Barnes & Kolling [**5**], and Herbert [**6**], are intended to be read in conjunction with these interactive tools. However, they do not integrate the reading of the textbook directly with the interactive tool.

The design of InterAcText follows an iterative software development approach. This paper presents the first iteration of the program. To decide what functionality improvements and additions are needed for the next iteration, InterAcText was submitted to a subjective review by first-year programming students at the University of Washington Tacoma. The students were asked to read and review the content as well as work with the interactive tools.

The results of the survey were mostly positive. The students seemed to appreciate Interactive Code and how it explains what is involved in each line of code. The positive results lead us to believe that InterAcText has a future. However, a more in-depth evaluation is needed to determine its true educational value. The evaluation participants also pointed out several areas that need redesign as well as what functionality should be added in the next iteration.

This paper is broken into the following sections. The next section, Section 2, is a review of literature on projects with goals similar to InterAcText. Section 3 summarizes the features of this iteration of InterAcText followed by an explanation of its design. Section 4 discusses both the completed evaluation and future evaluations of InterAcText. Section 5 concludes this paper by presenting a review of InterAcText.

2. Related Work

2.1 Introductory Java Texts and Online Tools

There are several different approaches to teaching Java programming. Textbooks that teach Java programming include Barnes & Kolling [**5**], Horstmann [**7**], Lewis & DePasquale [**4**], Lewis & Loftus [**8**], Liang [**9**], and Reges & Stepp [**10**].

Two of the textbooks, Barnes and Kolling [**5**] and Lewis and DePasquale [**4**], integrate interactive tools to help the reader learn to program. Barnes and Kolling [**5**] is intended to be read to complement BlueJ, a programming visualization tool and functional IDE "…developed and maintained by a joint research group at Deakin University, Melbourne, Australia, and the University of Kent in Canterbury, UK." [**3**] The aspects of BlueJ will be discussed in detail later in this section. Another interesting aspect of the Barnes and Kolling text [**5**] is the objects-first pedagogy. They present an object-oriented paradigm in the beginning of the book and introduce traditional procedural programming later.

In Lewis and DePasquale [**4**], the authors introduce readers to programming through ALICE, an innovative 3D graphical programming environment. This text also uses the objects-first pedagogy, and does not introduce the reader to Java until the seventh chapter. ALICE will also be discussed further in this section.

While both of these texts rely on interactive tools to supplement learning to program, neither one is integrated into the actual tool. They both use traditional print textbooks and require a reader to put the book and tool together, whereas InterAcText merges the interactive aspects and content into one package.

The above textbooks are all examples of traditional printed books. Cengage Learning, a publisher of IT-related content, creates course-specific online textbooks for colleges and universities called Active Readings. Fayetteville Technical Community College offers a course, CIS 111, Basic PC Literacy, that uses the Shelly, Cashman, Vermaat, Paparella, & Simko text [**11**] in Active Readings format.

Shelly et al. [**11**] present the text content in a web application. Multimedia, such as videos and sound clips, as well as review questions and exercises, are integrated directly into the text content. Active Readings are very similar to what InterAcText attempts to achieve. However, Cengage Learning does not have Active Reading content for introductory programming courses.

2.2 Graphical Programming Development Environments

The past two decades have seen the development of two tools aimed at teaching programming to an audience that is more familiar with computer graphics and computer games than previously seen. Both Alice and Scratch (described later) are graphical programming environments to allow users to build programs by dragging and dropping code and graphical objects. The reasoning in this method is twofold. First, a user, most likely a novice programmer, does not need to worry about forming proper syntax. Second, the 2D (Scratch) and 3D (Alice) graphical environments keep users interested and motivated to program.

Alice was the first of these two tools created. Created by a team at Carnegie-Mellon University, Alice is a graphical IDE that uses 3D objects and a drag and drop interface. Users can place the objects into a 3D world. The objects correspond to real Java objects, and users are able to access and create methods for the objects. Alice also allows for the more specific object-oriented concepts such as inheritance.

There currently are several pedagogical approaches to using Alice. The first uses Alice as a tool to teach general programming concepts without including an actual programming language. Several textbooks, such as Herbert [**6**] and Gaddis, [**12**] follow this approach. Another approach intertwines learning using Alice and a programming language, usually Java. The Lewis & DePasquale [**4**] and Adams [**13**] textbooks follow this approach. Finally, some studies have attempted to use Alice in small amounts to teach specific programming concepts. Dougherty used Alice animations and vignettes to teach the module on algorithms and programming in a CS0 course. [**14**]

The second graphical programming development environment created was Scratch. Created by researchers at MIT, Scratch is meant to be a "media-rich environment…that lets you create your own animations, games and interactive art." Scratch is aimed at first-time programmers, both computer science majors and non-computer science majors. Scratch is also made social with a web site. The site has a "vibrant online community, with people sharing, discussing, and remixing one another's projects." [**15**]

The programming environment in Scratch is similar to Alice in that users may drag and drop graphical objects into a world. The user may call methods from the objects by dragging and dropping graphical code blocks into an editing area. The order and specifications in these code blocks will then make up the program. It is a syntax-safe environment, meaning that if an action cannot be performed, such as evaluating an operation on incorrect types of data, the environment will alert the user. This removes a common frustration of first-time programmers, debugging syntax. It does not, however, prevent logically incorrect programs.

Several studies have shown that using these graphical programming environments keeps novice programmers motivated and interested in programming. Brown used Alice to teach CS-1 on two occasions. His observed that his "…students using Alice routinely went beyond the bare requirements…adding creative touches and extra features." He also noted that his "…students using Alice spent far less time feeling lost than [his] Java students typically do." Overall, the students were more motivated to work on the interactive Alice assignments than traditional ones. [**16**]

In another example, Sykes compares three classes, one taught using Alice as the language and two control classes. After a statistical analysis of grades among the three groups, "…the Alice group

significantly exceeded the performance of the Comparison Groups." He also compiled students' opinions in a separate qualitative investigation. While it should be noted that the version of Alice used in this survey was older than the current one and contained some severe bugs, the results were mostly positive. A sampling of the opinions include "Alice is quite helpful in problem solving…" and "[Alice made] it really easy to understand the fundamental concepts such as repetition, decision making, and concurrency…" [**17**]

Malan and Leitner used Scratch as a programming tool during a summer version of a course at Harvard University. They show that more than three quarters of the students participating felt Scratch was a positive influence affecting their subsequent experiences with Java. The student comments show the excitement and motivation with using the tool. They include "Scratch was a ton of fun…" and "The experience with Scratch helped me get a general idea of how to think like a programmer." [**15**]

2.3 Programming Visualization Tools

The centerpiece of the interactive content for InterAcText is the InterActive Code example feature. The InterActive Code example feature is a Programming Visualization tool. PVs present a user with source code and provide a graphical display of the code when it is executed. Examples of the visualizations include dynamic memory state diagrams, object diagrams, program output, and textual explanations of each line in the code. PVs usually allow for the user to step through the code, one line at a time.

Several PVs have been developed by Computer Science Education researchers over the past decade. They include ViLLE, Jeliot 3, and, to an extent, BlueJ. ViLLE was created at Turku University. According to its creators, the main research focus of ViLLE is its ability to visualize the same programming concept in more than one programming language. They feel it is important for novice programmers to focus on the programming concept instead of the language syntax. Other features include premade examples, the ability to add new examples, and the ability for different languages to be visualized [**1**].

The visualization features included in ViLLE are the ability to move through the code step by step, forward and backwards. While stepping through the code, ViLLE displays an automatically generated explanation of that step, current variable state, and a call stack for when methods are called. It also has the capability to present questions at specific points in the visualization. All of these features are presented using a single user interface with different windows for each of the features [**18**].

The researchers and creators of ViLLE recently concluded a study in the effectiveness of ViLLE with beginning programming students [**18**]. The study separated the students into two groups, allowing one group to use ViLLE while depriving the other group of it. The authors found many of the results inconclusive and found that the statistical analysis does not rule out their own null hypothesis "that ViLLE does not aid in the learning of Basic programming techniques." The study does support the idea also seen with ALICE and Scratch research that novice programmers benefit more from visualization tools than do experienced programmers.

Another example of a PV is Jeliot 3, the successor to Eliot, Jeliot I, and Jeliot 2000. Jeliot was created to "…involve the students in the construction of their own programs and at the same time examine a visual representation of the programs' execution." [**2**] Jeliot 3's focus is visualizing object-oriented concepts.

Jeliot 3 is a web-based application. The user interface is presented in a standard web browser. Similar to ViLLE, it presents several different areas for the visualizations. Some of the aspects Jeliot 3 represents visually include object instances, expression evaluations, and variable state. A user may step through the source code manually, automatically, forwards, backwards, and skip to a specific step. Jeliot 3 will automatically visualize code entered by a user. [**2**]

A third PV is BlueJ, mentioned earlier. The focus of BlueJ is to be a fully functional Java IDE with a simple interface and to have the ability to visualize class structure. BlueJ displays UML-like diagrams representing the objects and relationships between those objects in a user's program. It also allows users to create objects and interact with using the GUI interface. The idea behind this is for programmers to become familiar with object-oriented concepts without having to focus on the syntax of the language [**5**].

Of the three PV tools discussed, only BlueJ has a dedicated textbook aimed at teaching Java programming associated with it. The other two PVs are standalone projects. While all three are mature and feature-rich, none of them attempt to group all aspects of teaching programming into one package like InterAcText.

3. InterAcText

3.1 Summary of Features

3.1.1 InterAcText User Interface

The main user interface of InterAcText is a desktop application that presents textual content and all the interactive components. Four main interfaces comprise the application: the main Text Viewing interface, the InterActive Code example interface, Drill Sergeant, and the color customization dialog. The text viewing interface and Interactive Code example interface are integrated into the main application, while Drill Sergeant and the color customizer are pop-up dialogs. The following subsections will discuss each of the interfaces.

3.1.2 Text Viewing Interface

The Text Viewing Interface is what a user sees when InterAcText is first loaded. The interface contains three distinct areas. The top portion of the interface holds two combo box controls, one used for chapter navigation and another for section navigation. The left combo box contains the chapters of the textual content. The contents of this combo box are static. The right combo box contains the sections in the chapter selected in the chapter combo box. The contents of the section combo box are dynamically linked to the chapter combo box.

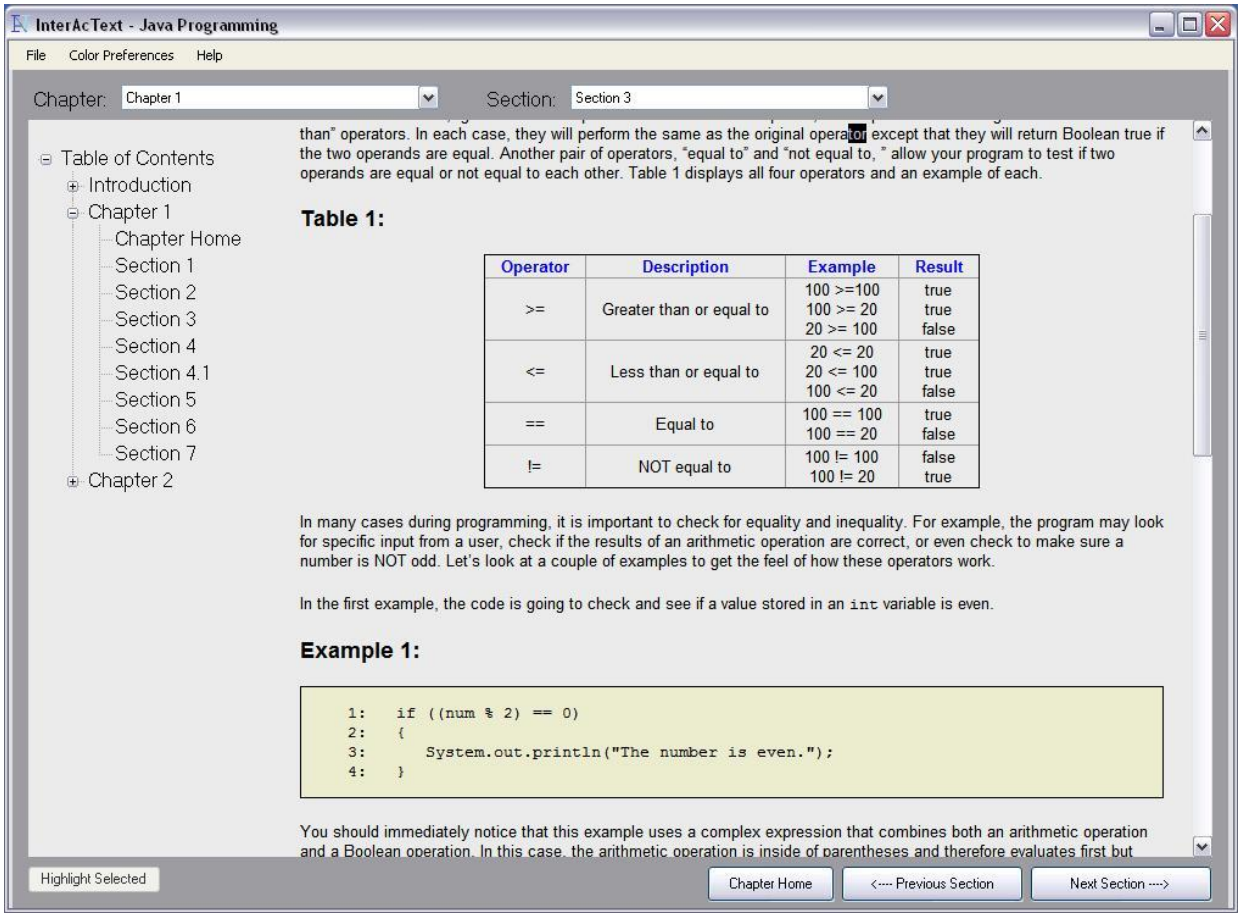

Figure 1 Text Viewing Interface

The middle portion of Text Viewing Interface is the main text presentation area. This area holds a tree-based chapter navigation area on the left. The reader[1] may double-click on sections to directly navigate to them. To the right of the tree navigation is where the actual textual content displayed in InterAcText resides. If the content extends beyond the height of the InterAcText container, a vertical scroll bar appears, allowing the reader to scroll through the content.

Finally, the bottom portion includes three navigation buttons. When pressed, the "Chapter Home" button will navigate the textual content to the beginning of the chapter. The two other buttons, "Previous Section" and "Next Section," navigate the content accordingly. These buttons become disabled if there is no next or previous section to navigate to.

Inside the textual content of the Text Viewing Interface, a user may find three types of interactive learning tools. The first two, InterActive Code Examples and Drill Sergeant, are both started using buttons inside the text and are also discussed in detail further in this section. The third tool is embedded interactive examples.

[1] We will use "reader" when referring to someone reading text in InterAcText. In contrast, we will use "user" when referring to someone using interactive tools in InterAcText.

The embedded interactive examples are found directly in the text and usually include a button to step through different stages of the example. In the current version of *InterAcText: Introduction to Java Programming*, one example steps the reader through source code snippets, while another uses images to step the reader through a memory model example.

3.1.3 InterActive Code Interface

InterActive Code is a programming visualization tool that steps the user through source code examples. This interface, like the Text Viewing Interface, is integrated into the InterAcText application. Each InterActive Code is specific to the topic of the section it was found in. The user needs to click the InterActive Code button found in the text to access the example.

The primary function of the InterActive Code is to step users through source code, allowing them to move forwards and backwards. The user finds two buttons at the bottom of the interface, "Previous Step" and "Start/Next Step" that allow him or her to move through the example.

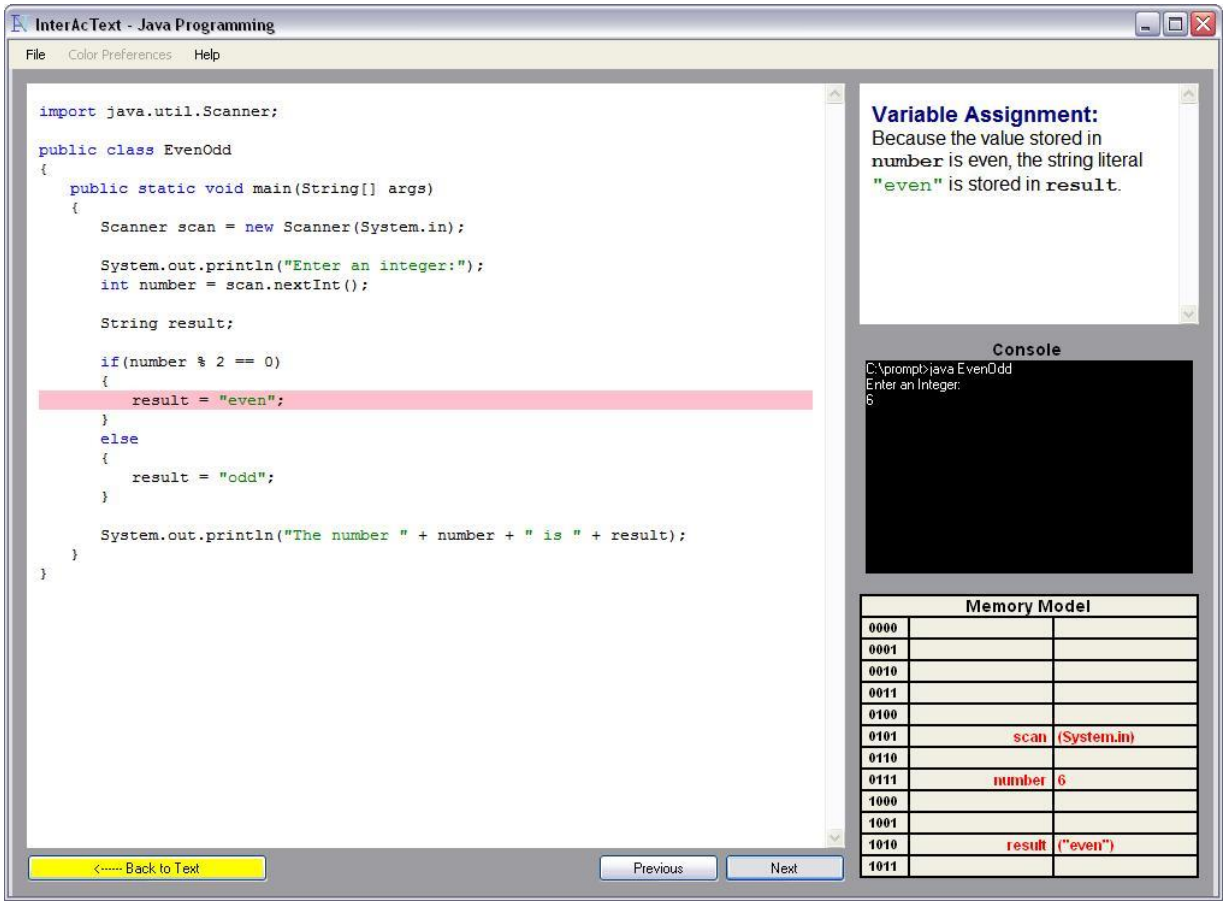

Figure 2

The interface also has four viewing areas: Source Code, Explanation, Console, and Memory Model. Each viewing area is responsible for displaying different information about the current step. The Source Code viewing area is the most prominent viewing area, and it contains the formatted and line-numbered source code for the example. At each step, the source code for that step is highlighted in light red. The

step itself may be a whole line of code or just a portion of a line. For example, the following line would be an entire step:

Step 1: `System.out.println("Enter an integer: ");`

While this next line is broken up into four steps:

Step 1: `for(int i = 0; i < height; i++);`
Step 2: `int i = 0;`
Step 3: `i < height;`
Step 4: `i++;`

The three other viewing areas are roughly the same size and are all found on the right side of the Interface. The top-most one is the Explanation viewing area. This view displays a brief note about each step. Most notes are static, but some are dynamic and display according to different values the user inputs. For example, when a step is a Boolean expression, it will be evaluated in real time using the value the user inputs. Given this, it will display `true` or `false` in the content of the note, depending on the vale.

The Console viewing area mimics a real console and allows output and input. When an example comes to a step that contains a `System.out.println` method call, the viewing area will display accordingly. It is read-only until a step requires input from the user. At that time, a user may enter text, the border around it turns red, and the focus of the interface is put on the viewing area. The output is dynamic to what values the user entered.

The Memory Model viewing area sits below the previous two and dynamically displays variables and their values. When a variable is declared, it is added to the model at a random address location. The values will update dynamically depending on the user input. When a user moves backwards through an example, values will change back to what they previously were. In other words, the InterActive Code keeps a memory of all used values.

3.1.4 Drill Sergeant

InterAcText includes a tool that tests the students' understanding of the material presented in each section. Drill Sergeant presents the reader with challenging multiple-choice questions that allow the reader to practice what he or she has just learned. The questions are formed to reinforce previously learned topics. Several sections include a Drill Sergeant quiz, each with ten questions.

Unlike the InterActive Code, Drill Sergeant opens in a new window. When a question is presented, the Drill Sergeant user interface displays the question text, the source code example, the multiple-choice answers, and three available buttons. The buttons include [Submit], which allows the user to answer the question; [Previous Question], which will present the last question; and [Return to Text], which exits out of Drill Sergeant. After the user selects an answer and presses the [Submit] button, "Correct" or "Incorrect" is displayed accordingly, and the question text is replaced with a short explanation of the correct answer. A fourth button is then available, allowing the user to move to the next question.

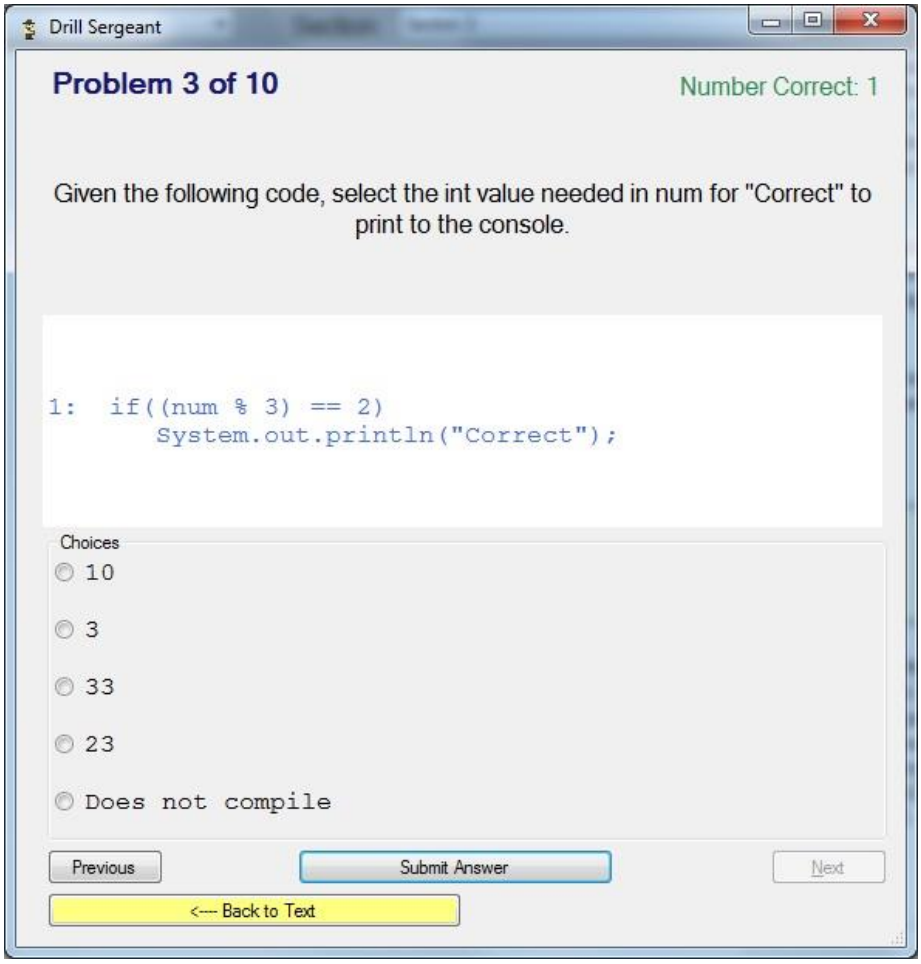

Figure 3 Drill Sergeant Question

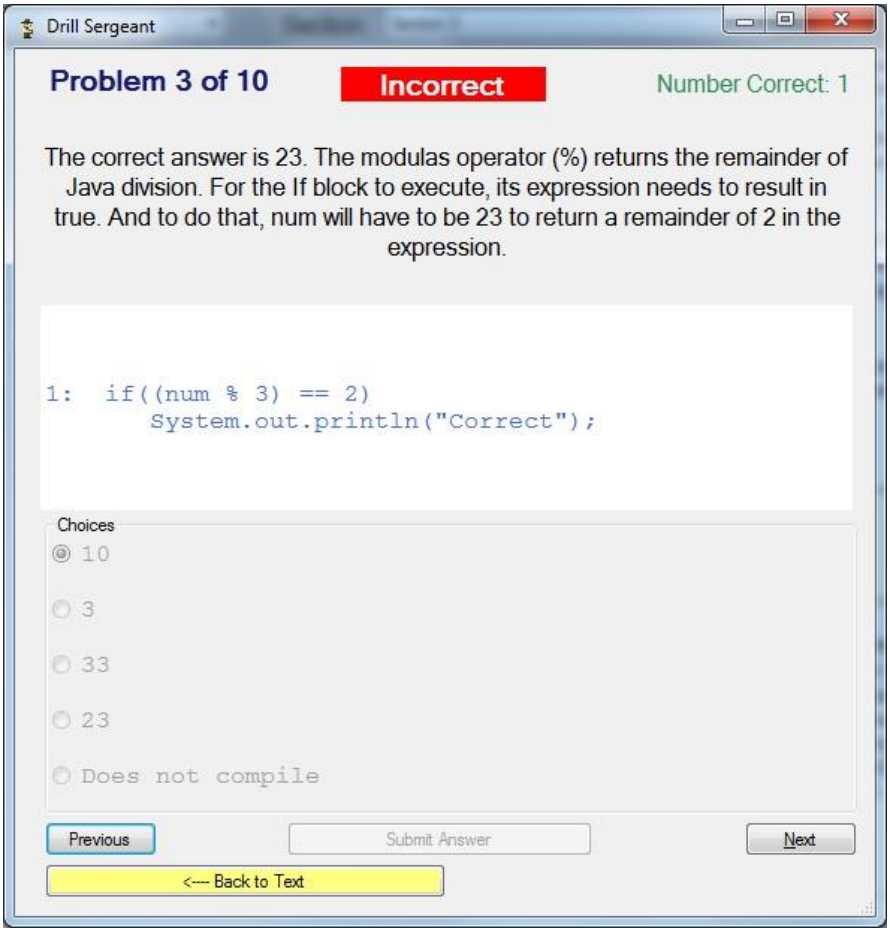

Figure 4 Drill Sergeant Answer

Drill Sergeant keeps track of the number of correct answers given by the user. This is displayed in the main interface. The user has the opportunity to go back to previous questions and answer them again. In this case, a correct answer is not reflected in the total number correct. When all questions have been answered, a final percentage of the number correct is displayed.

3.1.5 Color Selector

Different users require a variety of viewing needs. To address some of these needs, the color scheme in the Text Viewing Interface is fully customizable. The file menu of the Text Viewing Interface contains a button that opens a color chooser dialog box.

The dialog box allows the user to change the color of the heading, subheading, and normal text as well as the background. A preview panel will display any changes a user makes to the colors before they are applied to the actual Text Viewer. The dialog box also includes a button to set all colors back to the original scheme.

Figure 5

3.1.5 Text

The content for *InterAcText: Introduction to Java Programming* was written specifically for InterAcText. Currently there are one and a half chapters of content. The first chapter covers Boolean expressions and if statements; the second chapter covers loops.

Each chapter is broken into individual sections. Each section covers a new topic, and the sections are kept short to keep the amount of material introduced at one time low. As mentioned earlier, a section may include a Drill Sergeant quiz and InterActive Code example. These interactive tools focus on matter introduced in the corresponding section.

3.2 Design and Implementation of InterAcText

3.2.1 InterAcText UI

InterAcText is a desktop application written primarily in C#. The original proposal for InterAcText had it as a web application. Future iterations may still see it migrate to that platform. C# was chosen for its ability to be developed in Visual Studio and its large class library of GUI components. One component in particular, the WebBrowser Control, is used many times throughout the application.

The WebBrowser control is a class in the .NET Framework Class Library. It adds the ability to host XHTML and CSS documents as Web pages in a standard Windows Forms application. This allows InterAcText to use XHTML and CSS to display and format all textual aspects of the written text without requiring a special class to be written for that purpose. WebBrowser controls are used to display the main text in

the Text Viewing Area, the code in the Drill Sergeant questions, and the source code and explanations in the Interactive Code interface.

Methods in the WebBrowser class allow access to the Document Object Model (DOM) for each of the documents displayed inside of it. This is what enables InterAcText to efficiently change color schemes. Using Model View Controller (MVC), an object, acting as the model, is created that holds all of the current colors for the display. The color chooser dialog box acts as the controller. When the user selects and applies a new color in the dialog box, the model is updated with the new color. The model then notifies its views (all components in InterAcText that display the color scheme) and they update themselves. It is at this time that the WebBrowser control in charge of displaying the main text reaches into the DOM and changes the CSS rules for color.

We used the MVC architecture in this case because several components make up the display interface of the text content. It made the design cleaner and easier if each component was responsible for its own display instead of having another object change it.

3.2.2 InterActive Code

The InterActive Code examples rely on the MVC architecture. The views consist of the source code display, explanation display, console display, memory model display, and the previous and next buttons. There are three controls, the previous and next buttons, and the console display. The model is an object created specifically for each example. This object is instantiated from a child class of the parent, `Scenario`.

The `Scenario` class, acting as the model class, holds all data needed for each example. This includes the XHTML file that defines the source code for the example. A special CSS file is used to display this markup to look like code inside an IDE. Scenario objects also contain a `List` of `Step` objects. Each `Step` object holds all the information necessary for each step through the scenario. The information stored in a step includes the explanation for that step and if applicable, console output and variable information. All of this information is stored in an XML file read when each example loads.

Objects from the `Scenario` class also contain several behaviors, setting them apart from traditional models in MVC architecture. They include `NextStep()`, `PreviousStep()`, and `RecieveInput(String input)`. All three of these methods are defined as abstract methods in the `Scenario` class. As mentioned earlier, each example requires its own specifically made `Scenario` object to work and therefore, each example has a subclass written specifically for it.

Each subclass is responsible for defining the behavior specific to that example. This is because when a user clicks the next button, the next step in the List is not always the next step that needs to display. For example, when an `if` statement is encountered, depending of the result of the Boolean expression, different lines of code will execute next. The logic for each example is hard-coded into the implemented methods inherited from the parent class.

The architecture comes together when the user clicks the previous or next button or enters data into the console display. These events will cause the `Scenario` object to change its state accordingly. Once

the state change has completed, the model then notifies all of its views to update accordingly. Through this design, the InterActive code examples are able to mimic program execution moving forwards and backwards.

A disadvantage to this design is that for every example created, the specific `Scenario` subclass has to be coded along with it. To alleviate the overhead of this process, as much code as possible has been moved to the parent `Scenario` class. A template subclass was also created to demonstrate common design themes throughout all of the subclasses.

4. Review/Evaluation

4.1 Overview

InterAcText is being developed using an iterative software design and development approach. In an iterative design method, functionality is added to a software application in phases. At each phase, the program is evaluated and any design changes, bug fixes, or further functionality is then implemented. The process then repeats. This paper describes the first iteration in the development.

The following sub sections describe the evaluation of InterAcText in its current and future forms. They also discuss the appropriate steps needed to implement the suggestions and bug fixes found in each round of evaluation. See Appendix A for a full listing of the evaluation survey results.

4.2 First Iteration Evaluation Setup

Upon the completion of the first phase in development for InterAcText, it was important to fully assess the educational usefulness and functionality of the tool. This section describes the setup of the evaluation, which aims to answer the following questions:

- What needs to be added or fixed in InterAcText for the next iteration?
- Did you feel that the interactive aspects of InterAcText aided in your learning?
- Were any of the interactive aspects of InterAcText confusing?
- What aspects of InterAcText could use improvement?

This evaluation was performed by students taking TCSS 142 (CS1) at the University of Washington Tacoma. Students in this course were introduced to InterAcText during one of their regularly scheduled class meetings. This was followed up by an email giving a short description of InterAcText, the InterAcText install files, and a link to the Catalyst survey used to collect the data and opinions given by the students. The email and Catalyst survey may be found in the Appendix to this paper.

The survey was created on and administered from the University of Washington's Catalyst WebTools. None of the questions prompt the survey taker for personal information, as the survey is anonymous, in accordance to the Certification of Exemption approved by the University of Washington Human Subjects Department.

The survey has seventeen questions that can be split into six groups. The first group includes an introduction to the survey and a single question attempting to gauge the level of programming experience of the survey taker. The second group looks at how the survey taker usually uses source code examples and to what extent they used the InterActive Code examples in InterAcText.

The third group of questions all involve the following individual aspects of the InterActive Code examples, the ability to step though the code example, the display of variables in a memory model, and the corresponding explanations. Each aspect receives two questions. The first asks how well that aspect aided the student in his or her understanding of the topic presented. The second question presents an opportunity to write in any comments regarding that aspect. The survey taker is asked to consider the following question before responding:

- Was it confusing?
- Did it help you understand the content better?
- Could it have done anything differently that may have aided your understanding of the topic?

This group concludes with an open question prompting for comments on the InterActive Code examples as a whole.

The fourth group includes three questions and focuses on Drill Sergeant. The first question attempts to gauge how much the survey taker used Drill Sergeant. The second question is conditional to the response to the first question and asks why a user tried but did not complete a set of Drill Sergeant questions. The final question in this group is an open question prompting for comments on Drill Sergeant.

Three questions make up the fifth group, focusing on the readability of InterAcText, both visual and contextual aspects. The first question asks the survey taker to compare reading the text on the computer screen in InterAcText to reading text from a paper book. The second question looks at how easy or difficult the actual text was to read and understand. The final question is open question prompting for comments on the presentation of text in InterAcText.

The survey concludes with an open question prompting for comments on InterAcText as a whole. The survey taker is asked to consider the following questions before responding:

- What was the most helpful aspect of InterAcText?
- What would you like to see different?
- Would you be more likely to use a tool such as InterAcText to learn programming in the future?

4.3 First Iteration Evaluation Results

We received seven survey responses from the twenty eight students who were sent the project. The overall theme in the results was positive both in the comments and multiple-choice questions. Only one result trended negative in the responses. Unfortunately, the submitter did not leave comments as to his or her reasons for disliking the tool. This subsection will first discuss the results of the evaluation as they

apply to the first five groups introduced in the previous section. This will precede a discussion on the comments of the participants and how they will affect the next iteration of InterAcText.

4.3.1 Evaluation Results

Of the seven responders, four stated they had no previous programming experience. Two of the other three stated they have had experience with scripting languages, while only one responder had experience with a compiled language.

Only one of the responders admitted to skipping the source code examples found in textbooks while five read over them than moved on. A single person claimed to "read over the written example and then compile and execute the program [sic] on computer."

The next group of questions is about the InterActive Code examples. Five participants indicated that they stepped entirely though the examples they looked at. One of the five went so far as to step backwards and forwards through the example to see different execution paths. All five of these participants strongly agreed that the examples aided in their understanding of the topic presented. The last two participants both answered that they exited the example before completion; one was neutral, and the other strongly disagreed that the examples aided in his or her understanding.

Six of the participants left comments for this section. The sole non-commenter unfortunately was the student who did not think the tool helped. The comments were mostly positive. The response included:[2]

> *"It was rather a handy trick to look at were I was going and had been within the program itself rather then just seeing an outcome and hoping that the workings within were doing what I thought they were doing and so I dont find out down the road it was more complicated and I just got lucky with my output."*
>
> *"The program and its examples definitely helped me understand the material much better than the textbook. I enjoyed seeing what ever line of code did."*
>
> *"To see a program think is probably my favorite part of this program."*

The questions regarding Drill Sergeant showed that four of the seven participants, over fifty percent, did not have or take the time to complete the Drill Sergeant sections. Only one answered that he or she went back over the questions for a second time. This may lead to lesser number of questions in each module.

Another responded to not liking the color scheme for Drill Sergeant. He or she said, "Colors sort of hurt the eyes." This should be addressed in the next iteration of InterAcText. The color scheme chooser used for the main text may be implemented into Drill Sergeant.

We were interested in the results from the questions about readability. Four of the participants found reading on the computer screen about the same as reading from a textbook. Only one found it harder to

[2] Quotes are taken directly from the evaluations and have not been edited in any way.

read from the computer screen. The final two responses found reading from the computer was easier. It may be interesting to study the ages of the readers to see if there is a correlation between age and preference to reading apparatus.

We found the readability of the actual text to be a weak point in the project. Only two of the participants strongly agreed that the language was easy to read. Three somewhat agreed while the last two disagreed. This needs to be explored further in a future evaluation. Another way to improve will be to have several experienced programmers read and edit the text of the project.

4.3.2 Evaluation Suggestions

This evaluation of InterAcText produced a few suggestions for improvement. They may be grouped into two categories, one including minor fixes or updates, and a second, which major design changes would be needed to implement. We will not discuss the first category in great detail as they do not effect major changes.

Some suggestions include being more specific in the explanations found in the InterActive code examples such as "instead of saying '`firstNum`' it may be better to say the variable '`firstNum`'." Another suggestion is to change the color of the boxes of the interactive examples found directly in the text to be different than the normal non-interactive examples. Finally, one participant suggested explaining the memory model. This is a good suggestion and would be addressed in a previous chapter on data, variables, and memory management. Unfortunately, this chapter is not yet implemented.

The following suggestions represent issues that will need to be addressed via a major design change or feature implementation. The first involves InterActive Code examples.

> *"Something that could be inproved upon to make InterAcText more comparable to the text book is allowing option for the InterActive Code Example to run completly. It is anyoying to have to go in and out."*

This suggestion relates to functionality found in other PVs. Both ViLLE and Jeliot 3 have the ability to run from start to finish automatically. This feature should be easily implemented with one caveat: the automatic run would have to pause for user input as needed.

The next suggestion alludes to something that the original design of InterAcText was supposed to include.

> *"If it is posible i would allow the window to run in the text or beside it. Going from text to another window and then back is somewhat confusing."*

The decision to run InterActive Code examples in a separate window from the text was made to simplify the user interface. However, this led to some confusion for this participant. One suggestion could be to have the InterActive Code examples expand into the text screen when selected. This way a reader is not navigated away from the text. To determine which method benefits the reader, both ways may be implemented in the next iteration. During the subjective evaluation, participants would then be specifically asked which way they prefer.

The final suggestion involves a major hurdle in the design of the InterActive Code examples.

> *"it was easy to understand. i would like to see the ability to change parts of the code during the example."*

This is an aspect that we would like to see as well. Unfortunately, the current design of InterActive code examples requires each example to be set up before it is included in InterAcText. One possible solution to this would allow the user to select several different pre-written pieces of code to be interchanged in the example. This would not provide a true ability to change the code but mimic it.

It should be noted that we were surprised to see seven participants in this evaluation. One quarter of the students offered the opportunity responded. However, we would like to receive more feedback. Using this design process, feedback and comments are necessary for a successful end product. Future evaluations need to provide more incentive for students to participate. Finding a larger pool of novice programmers may also help provide more results.

4.4 Future Iteration Evaluation Plans

This section is broken into two parts. The first part describes possible future functionality and design changes. The section part describes the plans for evaluation of future iterations.

4.4.1 Future Work

Besides the three suggestions made in the evaluation for this current iteration, we would like to implement several features, changes, and redesigns into InterAcText for the next iterations. First, the text needs to be expanded upon. The chapter on loops needs to be completed, and a new chapter on data and variables needs to be written. The current chapters need to be re-edited to correct any mistakes or errors that have arisen since the first round of editing.

The next change involves adding questions to the InterActive Code examples. These questions will act as break points in the example. They will involve material covered in the example, and the user will have to answer the question before proceeding further with the example. The user will be able to move on whether or not the question was answered correctly, and both incorrect and correct will produce an explanation as to the answer.

The next suggestion applies to an iteration in the project beyond the next. InterAcText was originally intended to be a Web Application. This would allow students to use it on any Internet-connected computer using a web browser. The interface would require a major redesign for this to be successful. However, most of the code is modular enough to be reused; therefore, the project would not have to be started from scratch.

4.4.2 Future Evaluations

InterAcText is in the middle of an iterative software design process. Therefore, it needs to be tested and evaluated frequently. Each evaluation will test new functionality and fixes as well as seek out errors. We

will also use these evaluations to attempt to determine how students are using the features in InterAcText to learn. Along with student reviews and surveys as described earlier, we propose a more in-depth evaluation.

For this evaluation, we propose a closed environment for participants to use InterAcText. The computers on which InterAcText runs will include a screen capture capability. This will record every action the user performs. We will also capture video of the subject and his or her face while using InterAcText. When all subjects are complete, we will compile the two information streams and attempt to study what aspects of InterAcText aided in learning the most. We will also look at the subconscious body language to attempt to determine what tools need improving.

We will perform this evaluation after the next iteration of InterAcText is complete. These two types of evaluation will continue for several more phases in the design process. This will lead to a final evaluation which will attempt to quantify the educational benefits of InterAcText.

When the full text is implemented and the interactive tools are refined, we propose to test InterAcText in the classroom. We need to find a course that uses Java to teach introductory procedural programming with at least two sections. One section will use InterAcText as the primary teaching tool while the other sections use the textbook and tools previously defined for the course. Throughout the term, we will track and compare the grades, motivation, and overall success of the students. This will lead to the first quantitative analysis of InterAcText's value. We will continue to track the students throughout their programming education to see if there is a correlation between InterAcText's use and long-term success.

5. Conclusion

This paper introduced InterAcText, an educational software tool that integrates the text of introductory Java programming with interactive learning tools. The text was written specifically for the software. The interactive tools include InterActive Code examples, a program visualization tool, and Drill Sergeant, a quiz tool. This is the first iteration of several throughout the iterative software design process.

Previous attempts in using PVs have shown limited success. Moreno and Joy have shown that while they are useful, they do not significantly improve the learning outcome. They also speak of a gap between the course work and the use of Jeliot in their study. We hope that the integration of the PV into the actual text will help bridge this gap and prove more useful than previous work.

Using the iterative software design, this version of InterAcText is not a complete product. The first evaluation proved helpful in finding several areas that require design changes for the next iteration. It also showed the interest the participants had in such a tool. The participants' comments lead us to believe that this type of tool could be helpful for students learning Java programming.

Works Cited

1 Rajala, Teemu, Laakso, Mikko-Jussi, Kaila, Erkki, and Slakoski, Tapio. ViLLE - Multilanguage Tool for Teaching Novice Programming. *TUCS Technical Report*, 827 (June 2007).

2 Moreno, Andres, Myller, Niko, Sutinen, Erkki, and Ben-Ari, Mordechai. Visualizing programs with Jeliot 3. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (Gallipoli, Italy 2004), ACM, 373-376.

3 *What is BlueJ?* http://www.bluej.org/about/what.html, 16 February 2010.

4 Lewis, John and DePasquale, Peter. *Programming with Alice and Java*. Adison-Wesley / Pearson Education, 2009.

5 Barnes, David J. and Kolling, Micheal. *Objects First with Java - A Practical Introduction using BlueJ*. Prentice Hall / Pearson Education, 2006.

6 Herbert, Charles W. *An Introduction to Programming Using Alice*. Thomson Course Technology, 2007.

7 Horstmann, Cay. *Java Concepts*. Wiley, 2008.

8 Lewis, John and Loftus, William. *Java Software Solutions*. Addison Wesley, 2009.

9 Liang, Y. Daniel. *Introduction to Java Programming*. Prentice Hall, 2007.

10 Reges, Stuart and Stepp, Marty. *Building Java Programs*. Addison Wesley, 2008.

11 Shelly, Gary B., Cashman, Thomas J., Vermaat, Misty E., Paparella, Maureen S., and Simko, Eugene S. *CIS 111: Basic PC Lieracy*. Cengage Learning, 2009.

12 Gaddis, Tony. *Starting Out wih Alice: A Visual Introduction to Programming*. Addison-Wesley, 2008.

13 Adams, Joel. *Alice in Action with Java*. Thomson Course Technology, 2008.

14 Dougherty, John P. Concept Visualization in CS0 Using Alice. *Journal of Computing Sciences in Colleges*, 22, 3 (2007), 145-152.

15 Malan, David J and Leitner, Henry H. Scratch for Budding Computer Scientist. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education* (Covington, KY 2007), ACM, 223-227.

16 Brown, Peter H. Some Field Experience With Alice. *Journal of Computing Sciences in Colleges*, 24, 2 (2008).

17 Sykes, Edward R. Determining the Effectiveness of the 3D Alice Programming Environment at the Computer Science I Level. *Journal of Educational Computing Research* (2007).

18 Rajala, Teemu, Laakso, Mikko-Jussi, Kaila, Erkki, and Slakoski, Tapio. Effectiveness of Program Visualization: A Case Study with the ViLLE Tool. *Journal of Information Technology Education*, 7 (2008), 15-32.

19 Resnick, Mitchel. Scratch: Programming for All. *Communications of the ACM*, 52, 11 (2009), 60-67.