Learning Programming Concepts

With Henry the Robot

Eric Krokos
Patricia Sazama
Naveed Khan
Walter Hutchinson
2012
Henry the Robot

Team
Eric Krokos - Interviewer (Remington)
Patricia Sazama - Interviewer (Katrina)
Naveed Khan - Interviewer (Andrew)
Walter Hutchinson - Interviewer (Evan)

Problem and Solution
According to a poll conducted by the Computer Science Teachers Association, both the number of students enrolled in computer science courses and the number of computer science courses offered in high schools have been decreasing in recent years. In 2005 it was reported that 78% of schools surveyed had an introductory course in computer science, while in 2009 that number had dropped to only 65%. AP computer science courses showed a similar trend, with only 40% of schools offering it in 2005, and 27% offering it in 2009 [3]. This leaves a large number of students in high school and college who may be interested in programming, but who have little or no formal experience in the subject.

Our goal is to help address this problem by creating a way to teach non-technical individuals who are in high school or college how to program in a friendly and non-intimidating way. Studies show that programming is notoriously difficult to teach. In fact, some students seem to be unwilling to try to understand the syntax of programming [2]. Our resulting objective was to remove the difficult syntax of programming languages and make the goals of the program simple and easy to understand. Our proposed solution is to create a simple programming environment with drag and drop functions and logic to solve various puzzles. These puzzles are AI oriented where the goal is to navigate a character to the objective using a set of given functions and logical commands. Over time the puzzles will grow in complexity, while introducing new commands and concepts to advance the user’s set of logical tools to solve problems.

Related Work
There are a number of different pieces of software with the goal of creating a friendly introduction to programming. We drew much inspiration from several of these programs, but each of them had aspects that impeded a new, non-technical user from learning effectively with the program.

The first program we studied was Scratch. It uses a drag and drop approach to teach user objected-oriented programming. The user interface is fairly simple, the commands act as puzzle pieces which the user is required to combine in a certain order to create a valid program. The objective of the program is user defined and many functions are provided in order to achieve that objective. The software is highly interactive and allows the user to quickly view the result of their implementation [4]. Even with the simple user interface there is a steep learning curve.
This is because as soon as the user opens the program they are flooded with every option they can use to make a program, only separated by tabbed categories, which makes it very easy to become overwhelmed. We aimed to solve this problem by only giving users the commands needed to solve the given puzzle, and introducing new commands incrementally so they’re never overwhelmed.

The second piece of software we investigated was Carnegie Mellon’s Alice software. As with Scratch, Alice uses a drag and drop approach to teach object-oriented programming. It’s interactive interface allows the students to immediately see the functionality of their program. The point of the game, however, is closer to simply creating movies and scenes with no real objective guidance in learning to program [1]. Alice allows the user to choose from a massive list of commands for their program, and these commands are also more syntax specific. The negative side of learning syntax is that the programmer is distracted by details, rather than focusing on creating the logical steps used to achieve the objective. This lead us to conclude that the learning curve for this program is much steeper than our desired product.
Karel the Robot is a software that allows a user to learn basics of Java by limiting the number of commands available. The language gives six simple commands (turnOn(), move(), turnLeft(), pickBeeper(), putBeeper(), and turnOff()) with which a user can program a robot on a grid. The user simply give commands to the robot to make him move around and pick up and put down little circles called “beepers” for as long as they choose [5]. Like Scratch and Alice, there is no real goal for the user which we thought would be beneficial to individuals attempting to learn programming. Karel also chose to use actual Java syntax which we thought would be intimidating for a new programmer. Instead, we chose to use drag-and-drop commands described in simple English that are more intuitive to individuals in areas outside of computer science.

A screenshot of the Karel user interface [5].

Contextual Inquiry Participants
We conducted our tests on four non-computer science or technical major students. Their background included little to no programming experience, and each participant expressed varying degrees of interest in learning how to program. These conditions allowed each of our participants to fit our target audience.

The first person we interviewed was Katrina, a Classics and Italian double-major at University of Maryland, College Park. She had no experience using development software and no experience programming. Her major requires heavy use of text books as both learning and reference material. Her computer use focuses on text editing, researching and social networking. Installing software is the highest level of manipulation she has encountered in computing.

The second person we spoke to was Andrew, who is an Economics major here at College Park.
His only experience in programming was constructing and solving equations using MATLAB. The assignments he had in MATLAB provided step by step guides on how to input these equations so there was very little learning involved.

Remington, a Criminology and Psychology double-major, had only previously used development software called SPSS: a data analysis tool that uses basic programming. He also has basic knowledge of C++. His high school curriculum included some programming courses from which he actually retained some information.

Lastly, we tested Evan, a senior Political Science major at University of Maryland. In high school he took a introductory Java course offered through his school. The traditional classroom environment was not conducive to his learning style, and the class was constructed in a way that did not encourage learning but instead focused more on syntax. Assignments required students to copy code rather than implement their own program. As a result, it failed to keep his interest in the subject.

We met with each participant and talked to them about what specifically they would want in software designed to teach them programming. We asked these questions in a way so as not to influence their responses. We wanted them to be creative and imaginative, based on their prior experience on how they would want to be taught programming. We looked at what they had done before, such as their experience with SPSS or with MATLAB and listened to what problems that had or what they liked about using those programs.

**Contextual Inquiry Results**

What we learned from the inquiry was not what we initially expected. We planned on avoiding having a lot of text involved in the program and instead to keeping it very stylish and simple. However, a majority of the subjects actually wanted to have a regular hard textbook or reference guide with lots of examples and code with explanations that could they refer to. After asking why they wanted this they said it was what they were used to from school and knew it would always be there for them to look up information if they did not know how to do something. Also in a book, generally all the necessary information is in a single source and it is easy to look up how to do something using the index.

The books also break down learning into simple chapters, gradually teaching more complex new topics while at the same time giving tutorials and examples to support the new content. Books also often come with CDs with code examples and content that students can run and test on their computers in real time to see the results of what they learned and how it differs from previous chapters.

One thing we noticed among the people we talked to was they all want to interactively learn how to program, rather than just be told how to program. After noting they liked to use code from CDs, they also commented that they liked to change the code given to them to see what would happen and then modify that code to try to do something else. For example, they may try removing code they do not understand and then run the program to see what would happen
without that code in order to derive it’s purpose. The book would act as a foolproof secondary reference in case their experience and tinkering gave them no insight.

Lastly, we noticed that in general they all liked to play video games such as strategy games or role playing games. They were motivated by the idea of making their own video games or perhaps modifying video games to their needs. Remington for example, used what he learned when programming in simple C++ to create a black jack game in the terminal that put the player against the computer. He said that having the ability to understand what exactly was going on behind the scenes for the AI and the processing stages was really interesting and was what drove him to learn more about it.

Most of the users were motivated by the ability to create something interesting and unique to them. They wanted to be able to understand completely what was going on with the program so that they could use that knowledge to create newer and better things. Video games seemed to be the driving goal because it was visually rewarding, can be shown off to friends, and provides a small amount of entertainment while developing the program. Many of them said they would not want to be forced to write a very computational program like a Fibonacci sequence calculator or to create a program like Microsoft Word. In the end, it seemed like short simple programs that can easily be understood and are visually pleasing are what our users are experienced with and motivated by.

Tasks
After our interview with each of our subjects about what they wanted in a program that would help them learn programming concepts, we wanted to observe how they would go about trying to learn these programming concepts currently. What we found after these interviews and observations was that they liked short simple programs that were motivating to program and were visually rewarding. We found that they did not do any lower level programming due to a lack of understanding of how a computer works in terms of memory, etc. Their programs were often very high level and often revolved around games such as Black Jack, Poker, or even Mab Libs.

The three tasks we analyzed them performing were looking up information about a particular programming concept, figuring out syntax for a specific function or command, and determining what types of projects they were writing.

When Remington, for instance, was looking up information about a specific concept such as looping, he would google the concept in the language. For example he would type “Looping C++”, and this would return tons of results, usually forums or API websites that often overloaded him with information. If that didn’t work, which happened often, Remington and the others would refer to their books or lecture notes from classes such as those dealing with MATLAB. We noticed this takes up a lot of time and often does not result in a real understanding of the topics, particularly those involving the different types of looping, interactions with functions, etc. We want to minimize the amount of time the users have to spend searching for information on a concept and spend more time showing off what a particular concept can achieve.

We intend to teach new concepts in the program through either a tutorial or a pop-up box
explaining what a command does, or by giving a set of example code for the user to interact and play with. In the beginning, the program would start offering a basic tutorial about how to use the program, so there is no confusion on how it works. After that, the program would teach a very simple concept, such as how to use functions. The scenarios offered would then force the user to use functions by providing the needed functions and explaining what those functions do. As the user advances through the scenarios, they would force the user to use functions in new ways and teach the user more concepts.

The next problem we noticed was understanding the syntax of a function. For example, in C++, the sprintf function is very unclear for those not used to the cryptic syntax of C++. Even in MATLAB, our participants had difficulty understanding what each command did and how to pass arguments to them. This often prevented them from advancing in their program and would often just try to find a different command to use which would later complicate their program.

To fix this problem, we want to completely remove the complex syntax of languages like C++, and to formally introduce the user to functions and syntax of functions as needed. We plan to present the users with all the functions, commands, and logic that they need to complete a scenario. These commands would not be cryptic such as sprintf, but describe what they do such as Print, MoveForward, SetDirection, etc. To prevent typing errors and semantic errors we would have the commands already set up, given the needed arguments or hiding those arguments from the user in the beginning. The user would have a list of the functions given at the start of the scenario and can click and drag those commands into the script for a given entity in the game as needed. Overtime as the user advanced through the scenarios, we would introduce them to new functions and syntax and adding them to the list of available commands for a scenario, if those commands are needed to solve the scenario. We would teach them how to use that command by either a short description if they hover over the command, or given a help dialog if they right click the function.

The last thing we saw them having problems with were picking what type of project to work on. A lot of them had these grand vision of video games or creating huge pieces of software without first understanding how to go about implementing them or what is involved in terms of time, computational power, and skill. This is something we all have dealt with, trying to complete a project that is way beyond either our skill or time allowance. Another problem is a lot of software does not have any immediate feedback about its progress, such as setting up a Database program. You would spend a lot of time setting up data structures without any real feedback about how it is working and you often lose motivation to work on those types of projects more so than those that have immediate feedback on progress.

Our solution is to make the user solve a set of predefine scenarios set up like a video game. The reason for this is is that video games are graphical and any changes to the logic of a video game result in an immediate change in the visualization of that game and allows for errors or solutions to be easily seen. It is much easier as well to simplify the concepts in a video game than those in Microsoft Word. For example, it is easier to understand concepts such as position and game state rather than having to worry about text configuration and different text parameters and features. Also, while programming a video game, if the user does not feel like doing a scenario, they can freely play with the functions and get interesting and instant feedback.
about what they are doing and how that affects the overall project. Lastly, we want to control the scope and size of the programs the user will write. It is much easier to write and understand a small program than a large one, and we can control how long or short a program can be, so that way we can focus the user on learning about logic rather than intimidating them with lines and lines of code.
Brainstorming sketch of how the program could work. In this case commands are in a separate list, with an API-style description to help users learn.
Another brainstorming sketch, this one included only one button besides the list of drag and drop commands which was "GO" to run the program.
Our sketch for our first paper prototype design.
Our first paper prototype, combining the best elements of our individual sketches.
Early concept art for Henry the Robot

References


