Mission Statement: To provide an easy, interactive way of learning programming without the difficult syntax of typical programming languages.

Team

Eric Krokos: Was the main programming overseer, making sure the different components of the program came together and worked correctly. Wrote the GUI code and wrote part of the paper and video taped the video.

Patricia Sazama: Created and positioned the visual elements in the program. Designed the final GUI interface and wrote part of the paper.

Naveed Khan: Wrote the backend interpreter and error reporting tools in the program. Wrote part of the paper.

Walter Hutchinson: Created the website and edited/compiled the videos together. Wrote part of the paper.
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 [7]. 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 address this problem by creating a way to teach non-technical individuals who are in high school and 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 solution makes use of recognition rather than recall by restricting the user to a set number of predefined structures. In addition, rather than bombarding the user with programming jargon and a complex series of windows, we simplified our design to implement drag and drop functions. We also “gamify” programming with AI oriented logic puzzles to try and make programming as fun as possible. Over time the puzzles grow in complexity, while introducing new commands and concepts to advance the user’s set of logical tools to solve problems.

Past Work

The idea of using video games as a teaching tool has been explored a number of times. The following educational games attempt to combine learning with fun and cover a variety of different subjects. We wanted to use these techniques to introduce non-technical individuals to programming in a fun and non-intimidating way. Three of the the first games we looked at for inspiration on how to create a fun educational video game were Math Blaster [11], Sonic Schoolhouse [6], and Refraction [8].

![Figure 3.1: A screenshot of Math Blaster](image-url)
Math Blaster is a stage progression based game in which players advance by solving math problems [11]. The problems are designed for elementary school children and teach basic math subjects such as addition and multiplication. Figure 3.1 shows an example of one of the stages where the player must first solve the math problem while dodging the debris dropped by the space alien. After getting to the correct answer the user can move on to the next problem. One benefit of this rigid process is that it allows the developers to determine a good learning progression without letting the student dabble in a single level and not learn as much. We included this gradual stage progression approach in Henry the Robot. Refraction, shown in Figure 3.2, also follows a similar level progression strategy [8]. In Refraction, the user is taught the ideas behind fractions by splitting a laser beam into fractions. In this game, unlike Math Blaster, the user is never presented directly with any traditional looking math problems. We thought this made the subject matter less intimidating, and included this approach in Henry the Robot as well.
The two other teaching programs we looked at, Sonic Schoolhouse (Figure 3.3) and Cramster (Figure 3.4), helped us determine what aspects we didn’t want to include in our game. Sonic Schoolhouse is a game with a similar objective and target audience to Math Blasters. The major differences in the games are the increase in subject matter, free range environments, and choice in activity [6]. We thought that this freedom in the game, though appealing to the users, made it too easy to get sidetracked from learning, and that tightly controlled level progression was better for teaching. On the other hand, websites like Cramster focus more on the problem solving and discrete examples. Users can earn karma points by answering questions posted by other people. The level/rank of user is determined based on how many points they have [5]. We thought that the point-earning system alone wasn’t enough to encourage users to want to continue to use the software and felt that having a distinct end of “levels” and “stages” increased the players sense of achievement, so we included that in Henry instead.

After looking at these general educational games, we found several that have been created with the goal of teaching programming. The examples we drew from to create Henry the Robot were Scratch [9], Alice [1], Karel the Robot [10], and Code Academy [4].

Scratch 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 as shown in Figure 3.5. 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 [9]. 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 in Henry by only giving users the commands needed to solve the given puzzle, and introducing new commands incrementally so they’re never overwhelmed.
Carnegie Mellon’s Alice software also uses a drag and drop approach to teach object-oriented programming like Scratch. Its 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, shown in Figure 3.6, 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 with which a user can program a robot on a grid shown in Figure 3.7. 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 [10]. 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.

Figure 3.7: A screenshot of the Karel user interface.

Code Academy approaches teaching programming in a more direct way. The program immediately has the user start writing code and doesn’t have the same game-like aspects of the other three programs discussed, instead offering a similar “track your progress” concept and point system as Cramster. This interface is shown in Figure 3.8. The point system gives the user a sense of achievement as their learning progresses. Their technique of immediately forcing the user to dive into coding seemed like it would be too intimidating for non-technical majors, however, and we chose to avoid that, instead favoring the drag-and-drop puzzle pieces from Scratch and Alice.

Figure 3.8: A screenshot of Cramster.

Interactive Prototype

The interactive prototype for Henry the Robot was created using the Java programming language. We divided the program into two separate parts, the graphical frontend, and the interpreter backend. The graphical interface (GUI) was written from the ground up, not using any built-in interface libraries other than the JFrame class to create the window. The buttons, scroll bars, commands, grid, icons were all drawn using the basic Java drawing commands. We decided to go with this approach because we wanted to have the most control over what the GUI looks like and how it interacts with the user, such as clicking and dragging. Everything in
the GUI works except for the save and grid toggle buttons. We have only implemented three levels, so we felt that saving and loading was not an important feature to have for prototype testing. The toggle grid button was more of an option than a core functionality and we felt that this also was not important for testing.

The backend was also written in Java. It was designed to be able to accept a specific object, such as the Henry the Robot or the guards, and a list of commands for that object such as “MoveRight”, “MoveLeft”, etc. Given these objects, the interpreter will execute these commands on the objects and update the game state, such as the positions of the objects on the grid, and alert the user if the level was completed. The interpreter is fully functional, able to execute any set of the built in commands, as long as they are legal commands. If an illegal command is passed, or if the logical structure of the program is not correct, such as a missing “End if” after an “If” statement, the interpreter will throw an error to the user stating the problem. In the situation where the user’s program does not solve the puzzle, after a certain number of iterations of the code, the program will stop and throw an error, stating that the current program does not allow Henry to reach the goal.

We programmed three levels, all are fully functional and contain the commands needed to solve them. The first level is a simple level where all Henry must do is walk right. The user is given one command and are told in the introduction window to drag it to the program list window on the left side of the screen. The user is then instructed to click the Play button and can watch the robot walk over the grid toward the goal. After completing each level, the user is prompted if they want to continue to the next level or continue editing the current level. The second and third levels slowly introduce new programming commands and concepts to the user, challenging them to solve new and harder puzzles while visually showing them the effects of those commands.

Figure 4.1: Final Interface design of Henry the Robot
We changed the paper prototype’s visual layout slightly in the interactive prototype and in the final version (shown in Figure 4.1). We moved the program list scroll bar to the left side of the window from between the program pane and the grid. We did this for enhanced functionality. The new positioning allows the user to simply swing the mouse to the left imprecisely and still get it at the bar (Figure 4.2). We removed the step through commands feature because most of the commands do not visually do anything, such as the if statements and for loops. Only the move statements would do anything, which would be seen when running the program regardless. However, if an error does occur in the code, a popup box will appear and tell the user they have an error in their code and will explain what the error is.

![Figure 4.2: Shows the scroll bar is now on the left side of the program pane and application window.](image)

The three tasks were finding out information about a command (medium), clicking and dragging a command (hard), and running the program (easy). We had the users do these tasks in this order to make the interaction with the program natural, rather than having them do very disjointed tasks in an arbitrary order.

For the first task, finding out information about a command, all the user needed to do was hover over a command or button for one second until a small popup bubble would appear with information regarding that visual element (Figure 4.3).

![Figure 4.3: Shows the bubble box popup help when you hover over a command or visual element.](image)
The second task was dragging a command to the program list pane for the selected object. For the first level, Henry is automatically selected as the editing program object. You move your cursor to the command bar on top of the grid, then click and drag the command you need and over to the program list. It will automatically snap into position in the program pane for you. If there are many commands in the program list already and you wish to insert a command, simply put the new command between the old commands and the list will automatically adjust to the new command (Figure 4.4).

![Figure 4.4](image)

Figure 4.4: Shows an in action picture of moving a command over to the program list panel on the left side of the screen.

For the final task, running the program, the user has already learned what the commands do and has written their first program. To reward them we let them see what Henry will do. To run the puzzle, the user only needs to click the Play button at the upper left side of the screen. To stop the program, they can click the Stop button next to the Play button (Figure 4.5).

![Figure 4.5](image)

Figure 4.5: Shows the play and stop buttons, clicking these will execute the game and run the code that you have put in the henry object.

Testing Method

The interactive prototype was tested on two participants, that were selected based on their field of study. Since our target audience is non-programmers, both participants had to be non-technical majors. Eric, a criminology major with no programming background, was our first participant who we recruited through a friend of a friend. Our second participant, Chris, was an art major, who also had no prior programming experience and was also recruited through a friend of a friend.

The user testing was conducted in an apartment kitchen during the day. The goal was to make user feel comfortable during the testing process. Each participant was given a seat at a
Our interactive prototype was executed using NetBeans, and each participant was presented with a running version of our program. They were observed and filmed by our team as each of them interacted with the program.

The users were asked to perform a series of tasks during the testing process. The tasks were not presented in the order of difficulty, but instead required the user to achieve the ultimate goal of building the program and completing the level.

Our first participant, Eric, was able to complete all three of the tasks with ease. The first task (medium difficulty) required him to find out more information about a command. He was able to quickly accomplish this task by hovering over the command button and noticing the pop up bubble with the information about the command. The next task (hard difficulty) required him to drag a command from the command list to the command pane. He also had no difficulty performing this task as he quickly and intuitively dragged the “MoveRight” command to the program pane until it snapped into place and then released it. He was also able to perform the last task, running the program (easy difficulty), with ease. The combination of these tasks caused the robot on the grid to move towards the treasure chest. He noticed a dialog box showing success as the robot reached the treasure chest, realizing that he had accomplished the goal of the level. Overall, Eric was able to quickly understand the functionality of our program and was able to build and run a program fairly quickly during the testing process.

Our second participant, Chris, had some difficulty during the testing process. When asked to perform the task of finding out more information about a command, his first action was to click on the command. After clicking around for a while, he managed to perform this task without any help. It took him a while to figure out how to drag the command from the command list to the program pane. He also was able to figure out and perform this task with minor difficulty. Running the program proved a challenge for this participant as he clicked the non-interactive start button several times. He eventually noticed the play button and was able to run the program. Overall, frustration was a factor that may have resulted in too many errors for this
The testing procedure involved explanation of tasks, presenting the tasks and then having the participants to complete each of the tasks. Walter had the responsibility of filming the entire testing process. His job was to conceal the identity of the participant (no face shots) while fully capturing their actions. Patricia presented the guidelines and explained how the testing process would proceed. Eric was responsible for explaining each task individually during the testing process and helping the participant when they were completely stuck or were unable to perform the task. Naveed was responsible for recording all the required data for each participant.

The test was used to determine the learning curve of the actual software, and the amount of time it takes a user to fully complete all the tasks. Our data involved the number of errors made by each user for each individual task. Furthermore the time for performing each task was also recorded to analyze the difficulty of performing each task. User reactions such as frustration, were also observed to draw relation between their behavior and ability to perform a task.

**Testing Results and Revisions**

Overall the tests went flawlessly, the people we tested on had little difficulty in performing the tasks we gave them using our interactive prototype. This is a huge leap from the paper prototype’s results. Overall there was less stress in using the program because it was more pleasing to the eye and the interface came off as very simple. The first participant, Eric, even noted that the interface was much simpler than he imagined a programming environment would look like and actually enjoyed using the program. However, there were some problems that we had not thought of due to the fact that the paper prototype was not an actual program.

The first thing we learned was if the user clicked play without any code in the objects, they had no way of knowing if the program was actually running. When you clicked the play button, all of the commands and irrelevant visual elements become unresponsive to your mouse click to keep you from editing your program during execution. This startled the user.
because they did not know what was going on and couldn’t tell if the program was crashing. To resolve this, we added a big banner at the top of the screen that told the user the state of the program, “Program Stopped” (Figure 6.1), and “Program Running” (Figure 6.2). This would visually show them if the program was running so they could take appropriate measures to change the state, such as click play or stop.

![Figure 6.1: Shows the program stopped state, and it is located right at the top middle of the screen to show its importance.](image)

![Figure 6.2: Shows the programming running state, in this state no other buttons except stop can be pressed.](image)

Another change we made was to clearly show the different squares on the grid. Before all the sections of the grid were touching, giving the image of a continuous surface rather than a grid. We changed this and added small spacing between each grid cell to clearly show the cells location and size (Figure 6.3).

![Figure 6.3: Shows the gray spacing between each grid cell so you can more easily tell where and how big a grid cell was.](image)

The last primary change we made to the program after getting the results of our user testing was to rename the Program start point from “Start” to “Program Start Point” (Figure 6.4). During the creation of the program, we all knew that this was the starting point of the code in each object, so we just naturally called it start. It was only until the the other users had tested and pointed out that we had two start points that we changed it. It is interesting because this was a fault noted in an earlier experiment and we had temporarily fixed it and had not realized we reverted back to the old command label.
Summary Discussion

Overall we had a great time developing and refining the project. It was an interesting and enriching experience watching how the initial concept of teaching people programming in an easy way evolved and turned into our final program. We learned that in developing a serious application you must not hold certain initial ideas concrete, that you must be open to new changes and criticism of your design. Not only did we learn how to design the visual and interactive side of the application, but we also learned how to design the backend side of the program and organize it so multiple components can be developed by multiple developers.

From the beginning we were concerned about the actual implementation of the interface. We wanted a product that was more than just clicking buttons and to make it as interactive and intuitive as possible. Initially we were daunted by the prospect of having to use a third party library for the graphical interface, having to learn how to integrate it and do what we wanted to do. We also were not sure how much functionality we wanted to put into the program such as the interpreter and level system. After much discussion and a few simple code prototypes we managed to overcome the obstacles and achieved our goal of a high functionality - easy to use application.

We feel like we ironed out most of the usability bugs, such as the conflict between start button and start of program, and the uncertainty of how to write a program and move commands. Of course there is always going to be that uncertainty of what a user may do due to the lack of the number of user tests. We tried to design our tasks to cover the core functionality of the program, but there is always the case of those people who try to go the extra step in programming and try to do actions that may not be supported or may lead the program to crash.

Successes and Failures

The project as a whole seemed to be a success. We feel that it was both fun and gently helped non-technical students learn object-oriented programming concepts. The puzzle-based design created an interesting and engaging environment, with one of our final user testers actually remarking, upon completing the first level that he wanted to beat the rest of the game now. The commands and programs created were clearly introducing the kind of logical thinking used in programming, but in a way that wasn’t overwhelming. The only failure we could really see with our design is that new levels must be hard coded in and so creating more levels can be difficult, but given more time we could create a level-building part of the program to make this process easier.

In the future, we would certainly like to have more user tests and test more than just three levels. For a few of the cases, in which the tester had spare time, we had them just use the program after completing the three tasks just to see what they would do. This gives more
insight into the actual usage of the application. Another change we would make is to test participants in a more comfortable environment, where they’re not so aware that they’re being tested. This would eliminate the anxiety about their performance under surveillance and would give us results more like a regular first time user.

Website

When designing our website, we decided to go with a simple layout with only a few pages. The main page would display information about our project. We took the general design from .calm [3] because it impressed us the most. However, since our group was not as proficient in css as they obviously were, we went for a simpler design. Our website covers the main points outlined in the rubric (about us, content, mission statement) and adds a little bit of information about the benefits of our project. Since our videos are posted on youtube we felt it was absolutely necessary to embed them into our content page to make it more interesting. The website was built using HTML5 (for the basic structure) as well as CSS3 (to prettify it). We also used the following website to generate the CSS for the color fade button-esque text backgrounds [5]. As mentioned previously, we needed help since our CSS knowledge base wasn’t great. Here are two screenshots of the site, excluding our “about us” page as it is visually boring in comparison.

![Figure 7.0: A screenshot of our website.](image)
Figure 7.1: A screen shot of our class deliverables page on the website.
References


Appendix A

Raw Data:

<table>
<thead>
<tr>
<th>Participant Name:</th>
<th>Chris</th>
<th></th>
<th></th>
<th>3 (Running the Program)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task #</td>
<td>1 (Dragging Command)</td>
<td>2 (Obtaining info)</td>
<td></td>
<td>3 (Running the Program)</td>
</tr>
<tr>
<td>Time (sec)</td>
<td>47</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Errors (# of unnecessary clicks)</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Participant Name:</th>
<th>Eric</th>
<th></th>
<th></th>
<th>3 (Running the Program)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task #</td>
<td>1 (Dragging Command)</td>
<td>2 (Obtaining info)</td>
<td></td>
<td>3 (Running the Program)</td>
</tr>
<tr>
<td>Time (sec)</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Errors (# of unnecessary clicks)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Low-Fidelity Paper Prototype of our Design
Mid-Fidelity Prototype based on the Paper Prototype

Design modifications (Indicated in red circles) before high fidelity prototype
Initial organization (division of each section) of the program before actual images (aesthetically appealing GUI) was created. This version was used to experiment with the back end (interpreter and level design).

Final Design (High Fidelity Prototype)
Consent for Video Taping

As part of a human computer interaction study, video surveillance is required.

I, [Name], hereby authorize the use of video surveillance for the purpose of "Heary The Robot" prototype testing.

Signature: [Signature]

Date: [5/11/22]
Consent for Video Taping

As part of a human computer interaction study, video surveillance is required.

I, Christopher Darmody, hereby authorize the use of video surveillance for the purpose of "Henry: The Robot" prototype testing.

Signature: ______________________

Date: May 11, 2012