Racecar Language

White Paper

Team 19

Samuel Kohn	Project Manager
Alexander Fields	Language Guru
Colfax Selby	Verification and Validation
Jeremy Spencer	System Architect
Mason Silber	System Integrator

Due: 27 February 2013
COMS W4115: Programming Languages and Translators
Professor: Alfred Aho
Mentor: Melanie Kambadur

Introduction

Technology education for elementary school students is in its infancy. Teachers are put in the unenviable position of trying to teach children how to harness the potential of computers without actually knowing how computers work and how to write computer programs. As a consequence, many students are never exposed to computer programming and the algorithmic critical thinking style that is critical to writing programs as well as solving many other problems in life. Racecar is designed to solve this problem by providing a language that is *easy to teach*, even for non-experts; *readable*, so parents can easily involve themselves in their children's education; and *engaging* for 8 to 10-year-old children so they are motivated to experiment and learn more about computers and programming, even outside of school.

In order to capture and maintain children's interest in programming, Racecar is designed around a single goal: to write a program that will navigate a car through an obstacle course. Students will learn how to write programs that tell the car to move and turn in specific sequences, a process that allows students to think about concrete objects—an essential requirement for a language designed for 8 to 10-year-olds—while they solve a prototypical problem of the algorithmic style of thinking. More importantly, the programs students write can be run using the accompanying application, which shows the car as it executes the program's instructions and navigates the obstacles. This immediate visual feedback is essential for keeping students on task and excited about their progress.

1. What Problem Does Racecar Solve?

Existing attempts to teach children about computer programming generally utilize languages from one of three categories: graphical “drag-and-drop” programming, simplistic text programming (with graphical output), and conventional programming languages. These various ideas each have their drawbacks, and Racecar is designed to improve on all of the positive aspects while minimizing the effects of the problems with these techniques. In general, these languages fall short in one of three categories: similarity to real-world programming (e.g. program is not a text file), readability for non-experts, and a level of engagement that captures children’s interest. Racecar is designed with these properties in mind, with the goal of making computer programming extremely easy to teach in schools.

2. Who Should Use Racecar?

The intended users of this language are elementary school children around the ages of 8 to 10, and their educators. Given that the purpose of Racecar is to introduce these children to programming, no previous experience is necessary. It is designed to be accessible and engaging to children of all different interests and backgrounds. The scope of the language is relatively small—it is clearly domain-specific—and it is easy for a non-technical adult or teacher to pick up Racecar as well; any elementary school teacher should be able to learn the language quickly and to teach it effectively to others. Lastly, even children’s parents could learn how to write, or at least read, Racecar programs to help on homeworks or independent projects if necessary.

3. Properties of Racecar

Easy to teach

Racecar is syntactically easy to understand so that instructors with minimal knowledge of computer science concepts will be able to teach the accompanying lessons and debug students' programs quickly. The lessons included with the language tutorial build on each other, showing students that a complex task can be accomplished by breaking the problem into manageable parts. Each lesson adds a new movement or programming concept that can be integrated into the previous lesson's program.

Readable

One of the biggest problems in the computer world is readability. At every step of a computer science education, teachers and professors beseech students to include whitespace, avoid long chains of function calls, and comment as often as possible. However, it still seems that people write indecipherable code that even experts have trouble understanding.

Language designers have tried to remedy this problem proactively by writing "readable" languages. Existing readable languages include COBOL and Python. Both contain syntactic constructs that are useful for programmers, yet degrade readability. If you thought "OCCURS 12 TIMES" means "loop/repeat 12 times" in COBOL, you'd be wrong—it's a declaration for a 12-element array! In Python (a vast improvement from COBOL), many keywords and functions such as "def," "len," and "str" are short, making them easy to type, but hard for non-experts to recognize. Racecar strives to be readable even to non-technical students, teachers and parents.

Engaging

The ability to control the outcome of an action, such as driving a car, is engaging and teaches students to think creatively while still conforming to rules. Navigating a car visually may be trivial, but Racecar will show students that a precise definition of the car's movement is necessary to achieve the desired outcome. The goal-oriented nature of the lessons combined with frequent positive feedback coming from the graphical application captures children's attention. Accomplishing the complex goal at the end of the lesson sequence is rewarding and builds confidence to tackle subsequent challenges.

4. Similar Programming Languages

There are a number of technologies available whose goal is to teach children in elementary school to think algorithmically and programmatically. One such "language" is MIT's Scratch platform, which presents a graphics-based language to children; code is constructed using a drag-and-drop interface. However, physically typing commands into a text editor is paramount in internalization of language constructs, programming style, and procedural thinking. Racecar's language and platform combines these approaches, compelling children to write their programs in a normal text editor, but then compiling it and importing it into an application where they can see graphical output of their code.

Other approaches to teaching algorithmic thinking have taken the form of games, completely abstracting away the idea of programming form the user. For example, Armor Games' Light Bot (http://cache.armorgames.com/files/games/light-bot-2205.swf) gives children a

platform on which to build small programs with a limited number of instructions, forcing them to think in terms of subroutines. Again, this platform, while certainly engaging, fails to give children the irreplaceable experience of typing a procedure word for word, the importance of which was discussed earlier.

Older technologies like LOGO incorporate true algorithmic thinking and graphical output. However, LOGO's platform is not as engaging as children have come to expect from modern software. Furthermore, LOGO's language itself lacks human-readability compared to Racecar, a feature that is particularly important in conveying programmatic ideas and facilitating an easy transition into coding for children.

Finally, platforms like Lego Mindstorms give children the ability to program their legos to move, allowing creations like robots, self-driving cars, etc. Hardware-based approaches, however, have fundamental limitations in distribution. One set of legos can create only one robot at a time; there is no such limitation with a purely software-based platform, since, for example, a proud parent can send his child's program to relatives without making them buy physical kits.