The gap between supply and demand for computer scientists has its roots in children's perception that programming is “hard and boring.”

The problem of making programming both accessible and exciting has its roots in early schooling and is international. In an interview with The Guardian on 10 January 2012, the UK’s Secretary of State for Education, Michael Gove, was highly critical of the current school system and suggested that “instead of children bored out of their minds being taught how to use Word and Excel by bored teachers, we could have 11-year-olds able to write simple 2D computer animations.” But how exciting is programming to students? Asked in the context of a typical computing course, one middle-school student summarized her perception of programming as “hard and boring,” which doesn’t suggest a workable tradeoff but instead a heartbreaking lose-lose proposition.

Various government agencies and private organizations have launched numerous efforts to broaden student participation in computer science, and after many years of failed attempts, general interest is finally beginning to grow. For instance, a YouTube video from code.org that quotes Steve Jobs on the relevance of programming and features interviews with famous computer scientists managed to get an unprecedented 10 million views in a short amount of time. It’s clear that something should be done to make programming accessible and exciting, but the question is how.

**PROJECT GOALS**

Scalable Game Design (SGD; http://scalablegamesdesign.cs.colorado.edu) has the ambitious goal of revolutionizing computer science education in public schools by introducing students to computer science through a combination of game design and science, technology, engineering, and math (STEM) simulation creation integrated into the middle-school curriculum. To date, more than 10,000 students from inner-city, remote rural, and Native American schools have participated in the US’s largest middle-school computer science education study. Researchers at the University of Colorado systematically developed and evaluated this education strategy based on four core principles:

- **Exposure.** Broaden participation and reach every student by injecting an easy-to-teach one-week game design module into existing mandatory keyboarding or MS Office “computing” classes.
- **Motivation.** Motivate students by carefully balancing challenges and skill levels through game design activities in an SGD curriculum that ranges from simple Frogger-like games to advanced Sims-like games.
- **Education.** Build instruments that analyze student projects for critical STEM-skill acquisition so that learning outcomes can be measured objectively. A latent semantic analysis–inspired approach helps determine computational thinking and skill transfer between game design and simulation creation.
- **Pedagogy.** Investigate the interaction of pedagogical approaches and motivational levels across genders and ethnicities. With an optimal pedagogical approach, 35 hours of careful instruction is enough
to train teachers to teach SGD curriculum in a gender-friendly way (45 percent of our participants were female).

Project data suggest that the SGD strategy works extremely well: 74 percent of male participants and 64 percent of female participants wanted to continue with similar courses as electives.

MAKING PROGRAMMING EXCITING

Simply put, creativity and ownership are the keys to making programming exciting. Most students aren’t interested in the act of programming itself—instead, they want to create animations, make stories, or build games. As early as 1991, with our work on AgentSheets, we found that giving students the ability to create their own shapes significantly increased their motivation. Instead of simply replicating an existing project from a tutorial, they took ownership of the creative process by drawing their own shapes and creating their own worlds. Instead of having cars hitting a frog in a Frogger-like game, they might replace the cars with dogs and the frog with a cat to create their own game. Once students create and own their shapes and worlds, they’re much more interested in the idea of using programming as a truly empowering process to bring their creations to life.

If ownership is essential to motivation, how can we advance to more sophisticated levels of creativity, such as the creation of 3D shapes? Most students are intrigued with 3D but only from the consumer viewpoint. Typically, it’s quite difficult to create 3D shapes from scratch, so 3D creativity presents a direct challenge to student ownership and motivation. Existing 3D programming tools generally offer limited options—such as selecting prebuilt 3D shapes from a collection or importing 3D shapes from the Web. We believe that using such prebuilt shapes actually reduces ownership and creativity, so we foster student ownership through a casual design tool that lets users create their own 3D shapes.

The goal of casual 3D design is not to create tools for Pixar-level animators but to reach students with no background in 3D modeling whatsoever. With AgentCubes, students create basic 3D models that represent recognizable 3D shapes such as people, animals, and other objects and use them to construct 3D worlds. Inflatable icons as depicted in Figure 1 instantly engage students by helping them create their first 3D shape in about a minute.

With AgentCubes, students can use sophisticated spatial reasoning to build complex 3D worlds that include layers as well as portals to other worlds. Programming includes camera control in first person and bird’s-eye view. Figure 2 depicts the level of 3D design complexity of a game built by a middle-school student. Projects can be run and authored as desktop applications or as HTML5-based Web applications.
that can run in desktop and mobile browsers without Flash or Java.

**MAKING PROGRAMMING ACCESSIBLE**

A dangerously trivializing perception implies that drag-and-drop programming, which we helped pioneer with AgentSheets, makes programming easy. It does not. To be sure, eliminating the syntactic challenges of traditional programming languages can remove some devastating programming frustrations, but these gains are commonly overestimated. No one seriously claims that spell-checkers, which are syntactic tools for natural languages, empower people to write best-selling novels. It’s just as absurd to harbor similar hopes for syntactic programming tools. Through our decades of training teachers and students around the world to make games and simulations, our SGD project team has found instead that programming’s semantic challenges are considerable and generally not well supported by programming environments. Almost anyone can learn how to write a program, but few learn how to debug it.

Semantic tools help users create programs that actually work. Contemporary computers are enormously powerful and can do much more than just provide syntax support. They can proactively help students comprehend the meaning of a program and visualize the difference between the program they want and the program they have. In AgentCubes, this computational power is employed to have the computer run the user’s program one step into the future to visualize possible consequences. Figure 3 depicts how AgentCubes annotates the program of the selected object. The red/green/neutral annotations reveal which rules will be tested, which one will fire and if the conditions tested are true or false. In the depicted situation the car will be moving up. These proactive annotations help both with program construction and by letting users identify and fix bugs before they turn into actual problems.

Our data shows that the SGD approach works—even in some of the toughest, poorest, and most isolated schools in the nation. Our results consistently exceed Gove’s vision of 11-year-old students building simple 2D computer animations. SGD students not only make simple 2D animations but also create playable games based on sophisticated concepts that include advanced mathematics and artificial intelligence. Significantly younger children at the elementary-school level were able to create
2D and even 3D games. Perhaps most important, many of our SGD students have advanced beyond making games to actually find pleasure in building scientific simulations.

**Acknowledgments**

I thank the National Science Foundation for its help in funding this project.

*Alexander Repenning* is a computer science professor at the University of Colorado and director of the Scalable Game Design project. Contact him at ralex@cs.colorado.edu.

---

Selected CS articles and columns are available for free at http://ComputingNow.computer.org.