StarLogo TNG: An Introduction to Game Development

Andrew Begel
University of California, Berkeley
abegel@cs.berkeley.edu

Eric Klopfer
Massachusetts Institute of Technology
klopfer@mit.edu

December 24, 2004

Abstract

The science of developing computer programs offers a rich educational experience that can help students gain fluency with information technology. Unfortunately, while computers have become commonplace in schools, the practice of teaching programming is being squeezed out of high school and middle school curricula. We believe that programming should be reintroduced to students, and that this can be done by focusing on video game construction, a compelling subject area for many students. Given the current expertise required to create a modern video game, new tools are needed to make this experience accessible to students. We have developed StarLogo TNG, a visual programming- and 3D-based environment that enables students to easily program their own games. It uses graphical programming to ease the learning curve for programming, and 3D graphics to make the developed games more realistic. An initial pilot study has shown that these innovations appeal to students, and in particular appeal to girls.

1 Introduction

In today's world, computer literacy is a required skill, just like the ability to read, write, do arithmetic, and communicate one's ideas to others. In order to prepare children to enter this adult world, it is necessary to provide them instruction in the constructive use of computers. While many children already understand how to consume information with a computer (i.e. listen to audio files, watch movies, or browse the web), they often do not get the chance to produce content (i.e. author their own home page, compose and play their own computer-based music, or produce their own home movie). In other words, when it comes to computers, children know how to "read," but do not often know how to "write." School activities tend to concentrate on obvious job skills, such as training with word processors, spreadsheets and accessing databases. But to gain a real fluency with technology, children require a new set of skills (Murnane & Levy 1996). In 1999 the National Academies of Science outlined a set of FITness skills (National Research Council 1999) that defined the ways in which students will need to be able to use information technologies for their personal and professional lives in the coming years. These skills include:

- **Engage in sustained reasoning** – Students have the ability to plan, design, execute and evaluate solutions to problems.
- **Manage complexity/Expect the unexpected** – Students are able to plan a project, design a solution, respond to unexpected interactions/outcomes and diagnose what is needed for each task.
- **Test a solution** – Students determine the scope, nature and conditions under which a solution is intended to operate. A solution to a problem must be tested to determine that the solution is appropriate to the problem at hand.
- **Organize and navigate information structures and evaluate information** – Students have the ability to locate, assess and use relevant information.

- **Collaborate** – Students have a clear sense for how the various parts of a problem are made to operate together, as well as the expectations for his or her own part, and strategies for ensuring that team members work appropriate parts of the problem.
- **Communicate to other audiences** – Students understand the needs of their audience and use appropriate language and tools to communicate effectively.

These principles center on the idea that we must create a culture in which students can navigate novel problem spaces and collaboratively gather and apply data to solutions. Resnick (Resnick 2002) suggests that an equally important goal is for educational programs to foster creativity and imagination. In other words, students should be acquiring habits of mind that will not only enable them to address today's problems and solutions, but will also allow them to venture into previously unimagined territories. To achieve this, they should have a facility with tools to create manifestations of their ideas so that they can test them and convey their ideas to others.

1.1 Computer Programming

Computer programming is an ideal medium for students to express creativity, develop new ideas, learn how to communicate ideas, and collaborate with others. The merits of learning through computer programming have been discussed over the years (Harel 1990, Papert 1993, Kafai 1995, Kafai 1996). As a design activity, computer programming comes along with the many assets associated with "'constructionist"' (Harel & Papert 1991) learning. Constructionism extends constructivist theories, stating that learning is at its best when people are building artifacts. These benefits of learning through building include (Resnick, Rusk & Cooke 1998):

- Engaging students as active participants in the activity
- Encouraging creative problem solving
- Making personal connections to the materials
- Creating interdisciplinary links
- Promoting a sense of audience
- Providing a context for reflection and discussion

As students learn to conceptualize and develop their own computer programs, this specific type of design and construction activity comes along with many potential learning outcomes. Some of these apply specifically to the opportunities opened by the knowledge of computer programming, which alone can provide access to whole new worlds of exploration (Wolfram 2002). Other learning outcomes from programming are more broadly applicable. Typical programming activities center on the notion of problem-solving, which requires students to develop a suite of skills that may be transferable to other tasks. As students learn to program algorithms they must acquire a systematic reasoning skills. Modeling systems through programs teaches students how to break down systems into their components and understand them. Many problems require students to acquire and apply mathematical skills and analysis to develop their programs and interpret the output. Together these skills comprise a suite of desirable learning outcomes for students, many of which are difficult to engender through typical curriculum. They can also form the cornerstones of true inquiry-based science (American Association for the Advancement of Science 1993), providing students with a set of skills that allows them to form and test hypotheses about systems in which they are personally interested.

1.2 Programming Education

Computer programming was widely introduced to schools in the early 1980s, largely fueled by cheap personal computers, and inspired by Papert's manifesto, Mindstorms (Papert 1980), which advocated that programming in Logo was a cross-cutting tool to help students learn any subject. Logo was popularly used to to enable students to explore differential geometry, art, and natural languages. Microworlds Logo (Logo Computer Systems Inc. 2004) extended this use into middle school where students continued to learn about mathematics, science, reading and storytelling. KidSim/Cocoa/Stagecast Creator (Cypher & Smith 1995)

brought modeling and simulation to children through a simple graphical programming interface, where they could learn thinking skills, creativity, and computer literacy. Subsequent work with Stagecast Creator has shown some success in building students' understand of simple mathematical concepts through building games (Lucena n.d.). Boxer (di Sessa & Abelson 1986) is used to help students in modeling physics and mathematical reasoning, while building understanding of functions and variables. Kafai (Kafai 1996) introduced novel paradigms for helping students learn about mathematics and social and relational skills through programming of games. Through this paradigm, older elementary school students use Logo to develop mathematical games for younger elementary students. It is of note that most of these efforts have been targeted at elementary school aged children. Few efforts have targeted the middle school and high school audience.

Despite many small-scale research efforts to explore programming in schools, Papert's larger vision of pervasive programming in school curricula encountered some serious problems. Programming did not fit into traditional school structures and required teachers to give up their roles as domain experts to become learners on par with the children they were teaching. (Agalianos, Noss & Whitty 2001) Correspondingly, teachers grew disillusioned and tried to marginalize programming in their curricula, going so far as to put it into its own subject, computer science. Computers were used to teach typing skills, and Papert's vision of programming as a vehicle for learning seemed to be dead in the water.

With the growing popularity of the Internet, many schools have revisited the idea of computers in the classroom and rediscovered their utility in the educational process. Schools now use computers to target job-oriented skills, such as word processing and spreadsheets. Some schools encourage children to create mass media, such as computer-based music, animations or movies. For the last several years, the state of Maine has provided laptops to all of its 7th and 8th grade students (and teachers) to make computers a ubiquitous tool in every subject in school (State of Maine 2004).

Now that computers in school are enjoying a resurgence in popularity, it is time to reconsider programming as a vehicle towards fluency in information technology.

2 Programming Revisited

The benefits of computer programming have inspired us to revisit the use of computer programming in the classroom, and see if there are ways that we can reintroduce programming into the curriculum. One way we feel to increase interest is to move programming in a direction where it would be easy to build video games.

Why should video games be part of a school curriculum? Kafai (Kafai 1995) makes a compelling case for teaching young children to program by using games as a focal piece. The advocacy and research group, Education Arcade (The Education Arcade 2004), has developed pedagogy for integrating games in math, science and humanities curricula to better understand what beneficial roles games might play in educational activities. Games have the potential to entice today's youth into programming. Today's middle and high school students have grown up with video games as a part of their culture. For many it is what entices them to use technology. While boys still currently invest more time in game playing than girls, the gap is rapidly closing and both boys and girls have become avid gamers. Through this familiarity with games, students have begun to develop one half of a video game literacy - they have learned to "read." But they have not yet developed the other half of that literacy - learning to write. But we can use the existing half of their literacy to lead them into writing, thereby connecting their interest and knowledge of games to the practice of programming.

Game design and construction require that students learn programming skills; through careful design of the activity and the environment in which it is enacted it is possible to teach children how to make games and incidentally learn programming. One might ask how learning about games will help at all to learn how to program. Games have a lot of natural connections to programs, for example, most games have animated characters that react to control by human players. These animations and their sequencing and reactions must be scripted to behave properly. Characters' behaviors are all based on models of behavior that are implemented by algorithms and whose state is stored in user-defined data structures maintained by the characters. Choosing data structures wisely is important to making a game play well and

efficiently. Computer-controlled characters incorporate forms of artificial intelligence to decide how to interact with the player's character. Even simply enumerating all the characters' behaviors is similar to planning out the features of a computer application during design.

Storyboarding a game requires knowledge of sequencing and branching which resemble binary decision diagrams used in operations research applications. Game scenery design involves both creating an artistic look and a game environment's functional behavior. Generating realistic-looking flora and landscapes taps into fractal geometry and iterative functions, both of which have very strong ties to the mathematics of recursive functions. Reality-based sporting games often link to large databases of statistics to help specify the virtual characters' behaviors. Simulation games like SimCity are based on mathematical models of city planning, geology or human behavioral systems, all of which require algorithm design and data structures. Games have a significant non-deterministic component (the human player), so they often exhibit unexpected behaviors. Debugging these behaviors requires long hours of testing to identify, diagnose and correct problems.

3 Video Game Construction Kits

The complexity in making professional quality video games has risen steadily since hackers built the first video games (e.g. Spacewar!, Pong) to the point where making games requires an entire Hollywood production crew, with actors, writers, and directors in addition to a large team of programmers. Despite their complexity, these games are not sealed products. A large community of developers and hobbyists have used "modding" tools to customize games using new skins for characters, weapons and environments. While game engines like Quake and Doom make this process relatively straightforward, they all require a fair degree of sophistication and programming expertise to do anything more than import new characters (including adding new behaviors).

This need for programming expertise may appear to be an insurmountable obstacle in enabling children to make professional-quality games, but this has not always been true. Commercial efforts to develop game construction kits for children have successfully employed a simple click-and-drag paradigm to make game construction easy. Some of these kits, which include Klik 'N Play and The Game Factory (Budge 1983, Europress Software Ltd. 2004, Clickteam S.A.R.L. 2004), were used for building real-time strategy games. Unfortunately, while the click-and-drag metaphor lowers the initial barriers to building a game, it lowers the ceiling as well. By contrast, a full-fledged programming environment would allow developers to create virtually limitless types of games.

Many general programming toolkits have been used by instructors as platforms upon which children can build games (Logo Computer Systems Inc. 2004, Ingalls, Kaehler, Maloney, Wallace & Kay 1997, Repenning & Sumner 1992). These toolkits have recently grown more scarce due to a de-emphasis of programming in the curriculum and the sophistication of tools required to create the photo-realistic three-dimensional renderings of modern games. Alice 3D (Conway, Pausch, Gossweiler & Burnette 1994) is one tool that has tried to make this jump. It is a 3D modeling and programming environment, though its level of sophistication makes it most appropriate for advanced high school and early college students.

We are trying to address this scarcity of tools. The way to make video game toolkits that are easy to use by children is to tap into the same increase in computer power that engendered the rise in game complexity. The abundance of CPU power enables us to create a game construction environment that already includes the great graphics, character animations, sounds and storyboarding; all that is left for the child to do is to come up with an idea for a game, design the game levels, create the characters' looks, and program their behavior.

In the next section, we introduce the StarLogo programming environment and discuss how we have enhanced it to be a platform that enables children to create video games.

4 StarLogo

Over the past years we have been working with StarLogo, a programmable modeling environment that em-

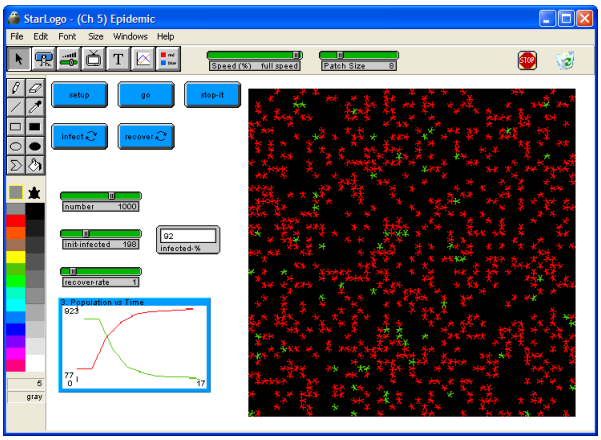

Figure 1: The left-hand panel shows a StarLogo model of an epidemic in action. One can see the interface area, in which users can start and stop the model and adjust parameters, the running model where the different colors code for different states, and the graphs at the bottom tracking model data. The right-hand panel shows the code that was written to generate this model.

braces the constructionist paradigm of learning by building. StarLogo was designed to enable people to build models of complex, dynamic systems. In StarLogo (Resnick 1994), one programs simple rules for individual behaviors of agents that "live" and move in a two-dimensional environment. For instance, a student might create rules for a bird, which describe how fast it should fly and when it should fly towards another bird. Because StarLogo makes use of graphical output, when the student watches many birds simultaneously following those rules, she can observe how patterns in the system, like flocking, arise out of the individual behaviors. Building models from the individual, or "bird" level, enables people to develop a good understanding of the system, or "flock" level behaviors (Figure 1 shows the graphical interface for the growth of an epidemic, another phenomenon that can be studied in terms of complex adaptive systems).

4.1 StarLogo for Games

In the 1990s, a few of our undergraduate researchers working on StarLogo began to build simple video games, such as PacMan, Frogger, and Space Invaders. It turned out that StarLogo's graphical turtle metaphor generalized quite nicely to user-controlled characters that could be programmed to play out the video game's behaviors. Some aspects of game design were not ideal, however. StarLogo's complex text-based language has always skewed it towards high school students and older. Our experiences with these games inspired the development of the Bongo video game construction kit (Begel 1997), which helped scaffold the game design and construction process around three main design features: scene, character and story. Paint tools enabled children to draw the scene and lay out the initial positions of all the characters. Characters and their interactions were designed next, followed by the rules for the game and their enactment in the program. The game's characters and story were still programmed in StarLogo, however, which limited Bongo's accessibility.

In the next section, we discuss a new incarnation of StarLogo which encourages the design of compelling games and makes them easy for children to program.

5 StarLogo: The Next Generation

We are designing the next generation of StarLogo to be a more satisfying game creation environment. The initial goal is to make it suitable for making the 3D equivalent of platform games or "scrollers", games in which a character moves across a scrolling landscape, encounter other characters and sets out on a mission (e.g. Super Mario Brothers). While we expect this environment to be able to create many other kinds of games, given realistic expectations, we do have to constrain the kinds of games that we expect children to build to make sure that the design is programmatically tractable. For example, it is important to have only a few characters, since the player can only easily control one, and artificial intelligence algorithms for controlling game characters are not easily generalized to any arbitrary game a child might want to build. A small world is desired so the child's entire game construction time is not taken by designing the level and drawing in the scenery.

These goals have guided the development of our new programming environment called "StarLogo: The Next

Generation" (StarLogo TNG). StarLogo TNG incorporates two major changes from previous versions of StarLogo – Spaceland,[1] which provides a 3D perspective of the game in action, and StarLogoBlocks, which provides a graphical interface for building games. In this redesign process, knowledge of our audience – primarily middle school through high school and college – and the context in which these tools might be used helped inform us as we made decisions on the level of sophistication we would require of the user. Often this tradeoff involves providing fixed properties or characteristics to the environment, which makes getting going very quick, but in turn can limit choices and flexibility. Things like low-level control, 3D rendering, and the like were fixed in their functionality as a result.

5.1 Spaceland

As people today are inundated with high quality 3D graphics on even the simplest of gaming platforms, students have come to expect such fidelity from their computational experiences. While the 2D "top-down" view of StarLogo is very informative for understanding the dynamics and spatial patterns of systems, it is not particularly enticing for students. Additionally, students often have difficultly making the leap between their own first-person experiences in our kinesthetic learning Activities, and the systems-level view of StarLogo.

Enter "Spaceland," a three dimensional OpenGL-based view of the world. This world can be navigated via keyboard controls (and soon by an on-screen navigator). The default view is similar to the old StarLogo view, being a top-down representation (Figure 2a). However, from this view, the user can zoom in on any portion of the world for a closer look and view the 3D landscape as they fly by. The other available view is the "Turtle's Eye" view (Figure 2b) in this representation, the user sees out of the "eyes" of one of the turtles in the system. As the turtle moves around they see what the turtle sees. This works particularly well in seeing how turtles bump off of objects, or interact with other turtles. Of course, it is not just turtles per se, but shapes can be

[1]The name Spaceland is inspired by Edwin Abbot's "Flatland" where two-dimensional creatures discover the three dimensional world called Spaceland.

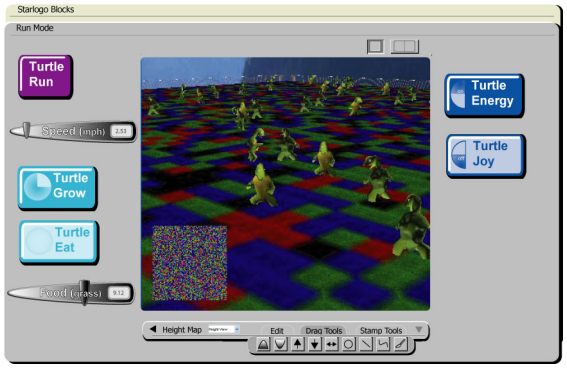

Figure 3: Runtime view

imported so that they agents in the system are turtles or spheres or Mario (Figure 2). The ability to customize the characters provides a sense of personalization, concrete representation, and also of scale. Lending further to the customization capabilities are the abilities to change the "Skybox" background (the bitmap image shown in the distance) as well as the texture, color and topography of the landscape. Since the top-down view is also still quite useful, that 2D view is seen in an inlaid "mini-map" in the lower-left corner of the screen (but may be repositioned as the user sees appropriate).

In runtime mode, users will see not just the 3D representation of the world, but also the tools to edit the landscape and the runtime interface (buttons and sliders) with which a user can interact to start and stop the model, or change parameters on the fly (3).

5.1.1 Additional 3D Benefits

Making games is just one reason to switch to a 3D environment from the 2D representation of the existing StarLogo. While making it easier to initially engage students through a more up-to-date look, a 3D representation has additional advantages. A 3D view can make the world feel more realistic. This can actually be a limitation in modeling and simulation, in that students feel that more realistic-looking models actually represent the real world, but in the proper context, a more realistic-looking world can be used to convey useful representations of scale and context – for example, conveying the sense of a landscape or the changing of seasons. Moreover, the

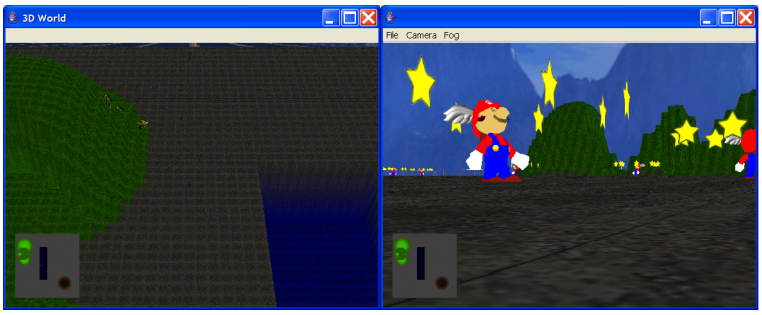

Figure 2: Top-down and turtles-eye views in TNG

3D view of the world is what people are most familiar with from their everyday experiences. Making the leap into programming already comes with a fairly high cognitive load. Providing people with as familiar an environment as possible can lighten that load and make that leap easier. Similarly, a navigable 3D landscape can provide a real sense of scale to a model that otherwise appears arbitrary and abstract to a novice modeler. Finally, with the proliferation of 3D tools (particularly for "modding" games) we can tap into the growing library of ready-made 3D shapes and textures to customize the world.

5.2 StarLogoBlocks

StarLogoBlocks is a visual programming language in which pieces of code are represented by puzzle pieces on the screen. In the same way that puzzle pieces fit together to form a picture, StarLogo blocks fit together to form a program. StarLogoBlocks is inspired by LogoBlocks (Begel 1996), a Logo-based graphical language intended to enable younger programmers to program the Programmable Brick (Resnick, Martin, Sargent & Silverman 1996) (a predecessor to the Lego Mindstorms product).

StarLogoBlocks is an instruction-flow language, where each step in the control flow of the program is represented by a block. Blocks are introduced into the programming workspace by dragging a block from a categorized palette of blocks on the left side of the interface. Users build a program by stacking a sequence of blocks from top to bottom. A stack may be topped by a procedure declaration block, thereby giving the stack a name, as well as enabling recursion. Both built-in and user-defined functions can take argument blocks, which are plugged in on the right. Data types are indicated via the block's plug and socket shapes.

The workspace is a large horizontally-oriented space divided into sections: one for setup code, one for global operations, and one for each breed of turtles. A philosophy we espouse in the programming environment is that a block's location in the workspace should help a user organize their program. While separating code by section appears to the user as a form of object-oriented programming, in reality the system does not enforce this. All code may be executed by any breed of turtle. Fish can swim and birds can fly, but that certainly should not prevent the user from successfully asking a fish to fly.

5.2.1 Graphical Affordances

The "blocks" metaphor provides several affordances to the novice programmer. In the simplest sense, it provides quick and easy access to the entire vocabulary, through the graphical categorization of commands. As programmers are building models they can simply select the commands that they want from the palette and drag them out. Perhaps more importantly though, the shape of the blocks is designed so that only commands that make syntactic sense will actually fit together. That means that the entire syntax is visually apparent to the programmer. Finally, it also provides a graphical overview of the control flow of the program, making apparent what happens when.

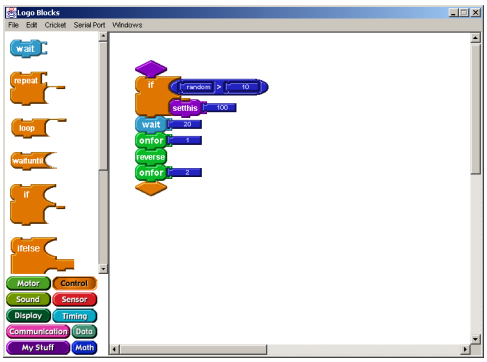

Figure 4: The LogoBlocks interface. A simple program is shown that includes logical statements, random behaviors and movement. The blocks are assembled into a complete procedure, drawing upon the palette of blocks at the left.

For example, the following illustration of StarLogoBlocks code on the left, and StarLogo code on the right for a procedure which has turtles turn around on red patches, decrease their energy, eat, and then die with a 10% chance and continue moving otherwise.

```
to run
  if pc = red
     [ rt 180 ]
  setenergy energy -- 1
  eat
  ifelse (random 100) < 10
     [ die ]
     [ move ]
end
```

While the code in StarLogoBlocks takes up more space, it can be seen that the syntax is entirely visible to the user. In contrast, the StarLogo code has mixtures of brackets [ ], parentheses ( ) and spaces that are often confusing to users. The if-else conditional fully specifies what goes in which socket. The if predicate socket has a rounded edge, which can only fit Booleans, such as the condition shown. Similarly, functions which take integers have triangular sockets. The two other two sockets (for “then” and “else”) are clearly labeled so that a programmer (or reader of a program) can clearly see what they are doing.

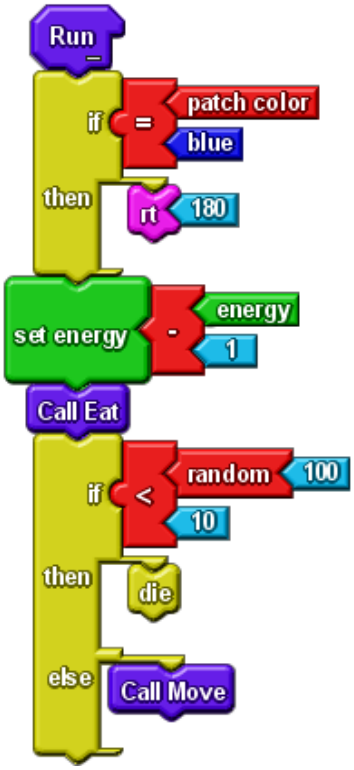

Figure 5: StarLogoBlocks and StarLogo representations of the same code.

Blocks create more obvious indicators of the arguments taken by procedures, as compared to text-based programming. This enables us, as the game construction kit designers, to provide new kinds of control flow and GUIs for the blocks without confusing the programmer. Adding a fifth parameter to a text-based procedure results in children (and teachers) more often consulting the manual, but introducing this fifth parameter as a labeled socket makes the new facility more apparent and easier to use. Graphical programming affords changes in these dimensions because a change in the graphics can be more self-explanatory than in a text-based language.

While StarLogoBlocks and LogoBlocks share the “blocks” metaphor, StarLogoBlocks is presented with much greater challenges due to the relative complexity of the StarLogo environment. LogoBlocks programs draw from a language of dozens of commands, are typ-

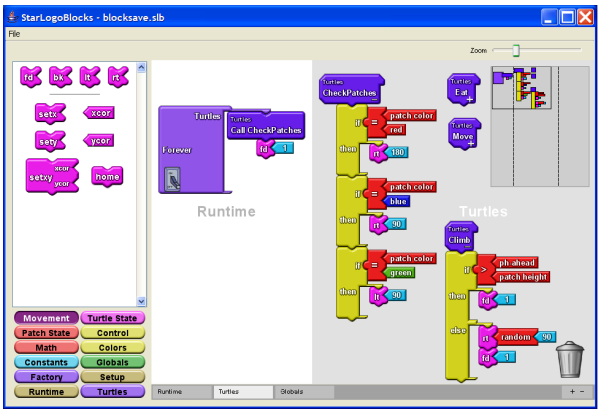

Figure 6: The StarLogoBlocks interface.

ically only 10–20 lines long, have a maximum of two variables, no procedure arguments or return values, and no breeds. StarLogo programs draw from a language with hundreds of commands and can often be a hundred lines or more long. Screen real estate can become a real limit on program size. Driven by this challenge, we created a richer blocks environment with new features specifically designed to manage the complexity and size of StarLogo code.

One of the most important innovations is to incorporate dynamic animated responses to user actions. We use this animation to indicate what kinds of user gestures are proper and improper while the user is performing them. For example, when a user drags a large stack of blocks into an "if" statement block, the "if" block will stretch to accommodate the stack. See the "Climb" procedure in Figure 6. The first part of the if-else command has one command to run (forward 1), while the else has two (right random 90 and forward 1). The blocks expand to fit as many commands as necessary so that the else clause could in theory have dozens of commands.

If a user tries to use a procedure parameter in a different procedure from which the parameter was declared, all of the block sockets in the incorrect stacks of blocks close up. When a user picks up a number block to insert into a list of values (which in Logo may contain values of any type), all block sockets in the list will morph from an amorphous "polymorphic" shape into the triangular shape of a number, to indicate that a number block may be placed in that socket. We plan to continue adding new kinds of animations to help prevent users from making programming errors in the system.

These animations are implemented using vector-based drawing. Vector drawing enables a second important feature: the zoomable interface. Using a zoom slider, the user can zoom the entire interface easily (see the zoom slider in the upper-right of Figure 6) to look closely at a procedure they are writing, or expand their view to see an overall picture of their project.

Another innovation has been "collapsible" procedures. Individual procedures can be collapsed and expanded to see or hide their contents. This allows the programmer to build many small procedures and then "roll up" the procedures that they aren't using and put them on the side. Figure 6 shows the "Eat" and "Move" procedures in their rolled up states. The individual commands are not visible, like they are in the other two procedures, but can be made visible by clicking on the plus sign on the block. Additional advances provide for the easy creation of new procedures, including procedures with an arbitrary number of parameters, and variables.

Together these innovations lower the barrier of entry for programming, thus facilitating model construction/deconstruction in the context of science, math or social science classes.

5.3 StarLogo TNG as Game Builder

The first version of StarLogo TNG will support the development of first- and third-person 3D exploration games. All games consist of three main pieces: characters, scenery, and behavior. Characters are implemented using StarLogo turtles; different categories of characters are defined using StarLogo breeds. The scenery of a game is created in the StarLogo Terrain Editor. Similar to the SimCity city builder editor, our Terrain Editor supports landscaping, drawing, and character placement. The behavior of the game and the characters are programmed using StarLogoBlocks. Our first version of StarLogo TNG will support relatively simple behaviors (such as moving around, jumping and shooting), but because StarLogo TNG is a full-fledged programming environment, we expect people to be able to build more complex character behaviors (incorporating desires like

hunger and greediness, or semi-autonomous motion or intelligence). In fact, while we have designed StarLogo TNG with simple exploration games in mind, building a more complex game with original characters is a great challenge for a designer, especially one we would like to get hooked on programming.

In the next section, we will walk through the creation of a Super Mario Brothers-like video game using the StarLogo TNG toolkit.

6 Building Super Mario Brothers

Super Mario Brothers was one of the most canonical and popular games for the Nintendo Entertainment System, introduced in August of 1985. In this game, two characters, Mario and Luigi (the brothers) are controlled by the players to make their way through the two-dimensional side-scrolling world of the Mushroom Kingdom in an attempt to rescue Princess Toadstool, who has been taken captive by Bowser, the evil King of the Koopas. Along the way, the brothers face Koopa Troopas (turtles that can be stunned by jumping on them), deadly Venus fly traps (must avoid touching them), Goombas (mushrooms which you can shoot with a fireball), and bottomless pits. To help them, there are floating magic boxes; if you hit them from below, out will pop a star (makes you invincible for a short time), a mushroom (makes you twice as bit and resistant to one hit from an enemy), or an extra player mushroom (gives you one more life).

We will build a 3D version of this game, similar to what one would see by playing Super Mario 64, an updated version of the game designed for the Nintendo 64 System. We start our game by designing the characters. To shorten the exposition, we will simplify the game to include only two characters and their interactions.

6.1 Characters

- **Mario** The player character. Mario looks like a plumber and has a red hat and overalls. He can jump, punch upwards, and sometimes shoot fireballs (if he has gotten a fire flower).
- **Koopa Troopa** Evil turtle henchman. Koopa Troopas patrol areas of the screen and crawl like turtles. Koopa Troopas can be stunned if Mario jumps on them. They can be killed if Mario kicks them while they are stunned. Koopa Troopas can kill Mario if they crawl into him.

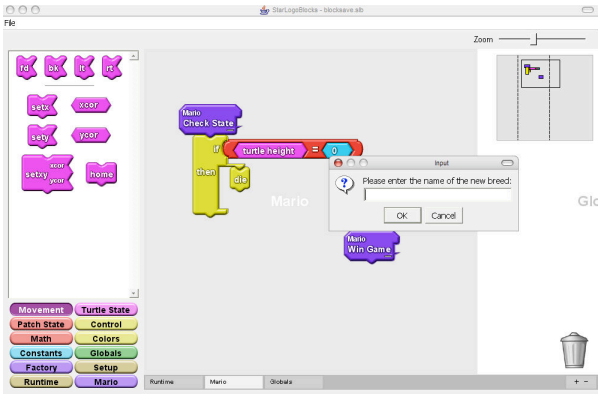

Figure 7: By clicking on the Add Breed (+) button, the user can create a breed of Koopa Troopa turtles.

To create these characters in StarLogo TNG, we create two new breeds by clicking on the Add Breed button in the StarLogoBlocks window (Figure 7). Designers would associate several shapes and animations (3D shapes in StarLogo TNG come with prebuilt animations) with a breed. StarLogo TNG comes with a variety of shapes, and users can add more of their own.

6.2 Scenery

We next move on to describe the scenery of Level 1. Unlike the original Super Mario Brothers, our scenery will be three dimensional. We switch to the StarLogo Terrain Editor and describe our landscape. First, we initialize Spaceland to a flat world covered with a grass texture. Then we grow three dimensional blocks on the surface of the world that Mario can jump on. These blocks will help us create a maze. At one end of the world, we create a flagpole-like extrusion from the world and color it like a metallic flagpole. When Mario reaches this spot, he wins the game. Between a few of the blocks, we push the ground down to create a pit. If Mario falls into the pit, he will die. We make each pit short enough for Mario

to jump over, but leave some longer to make the jumps more challenging.

We design the level to encourage Mario follow a path to the flagpole. Along the path, we place a Koopa Troopa on patrol. We place one Koopa Troopa on every block as well. Sometimes we place more than one in an area to make it more difficult for Mario to get around them. We save this terrain into a terrain block called Level 1. This block will appear in the Block Palette in the StarLogoBlocks window.

6.3 Behavior

Each character has behaviors that must be programmed by the user. Designers could program a funny way to walk, add shooting capability, or the ability to fly. Mario's behaviors are the simplest. He listens to the user's joystick to move him left, right, north and south. In addition, when the user hits the A button, Mario jumps. Mario must also keep track of how many lives he has – when he loses them all, the game is over.

We create one procedure: Respond To Joystick that uses two joystick blocks from the Block Palette. The first joystick block has four slots ready for a block stack, one for each cardinal direction. The second block has two slots, one for each button on the joystick. In each slot, we place code blocks that causes Mario to react to the joystick. For example, if the user moves the joystick left, we want Mario to turn to the left and move forward one step. If the user pushes the A button, we want Mario to jump using a Jump block. To determine if we fallen into a pit, or won the game, we create another procedure stack called Check State (Figure 8).

Koopa Troopas have a very simple behavior. They move back and forth over a certain distance (patrolling the region). This distance may be overridden if they're on a block and reach the edge prematurely. A Koopa has one procedure called Move (Figure 9).

Distance-to-go is a variable that Koopas use to determine how many steps they should patrol. Koopas also need a variable called Hidden? which indicates whether they have been stomped by Mario. If Hidden? is true, they should stand still and not run the Move command.

Finally, we must describe the global behaviors of the game. What happens when each of these characters

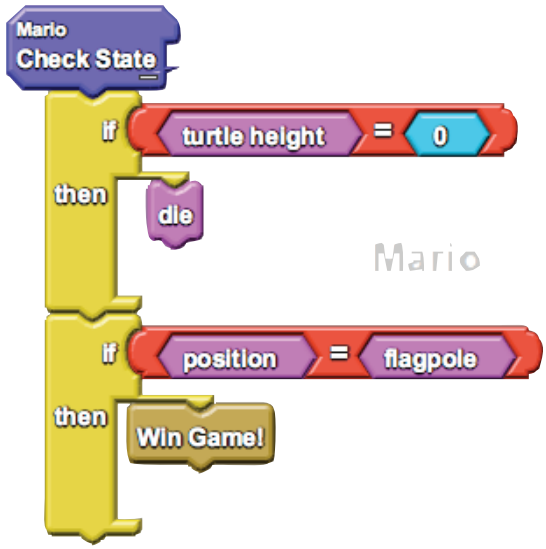

Figure 8: Mario's Check State function.

hit one another? When Mario hits a Koopa Troopa, he might die, or the Koopa might hide. When two Koopas hit one another, they reverse direction. We describe these behaviors with a Collision block that we drag out for each breed. There is one code slot in the Collision block for every breed that our character may hit.

If Mario hits a Koopa Troopa, we run some logic to see what happens (Figure 10). If Mario jumps on a Koopa (i.e. when Mario collides with a Koopa where his height is higher than the Koopa's), the Koopa hides in his turtle shell. If Mario touches the Koopa in any other way, Mario dies.

To finish up the game, we set up the scoring system. The score is a built-in property of all games; we modify the procedures above to increment the score when Mario stomps on a Koopa and when Mario wins by reaching the flagpole.

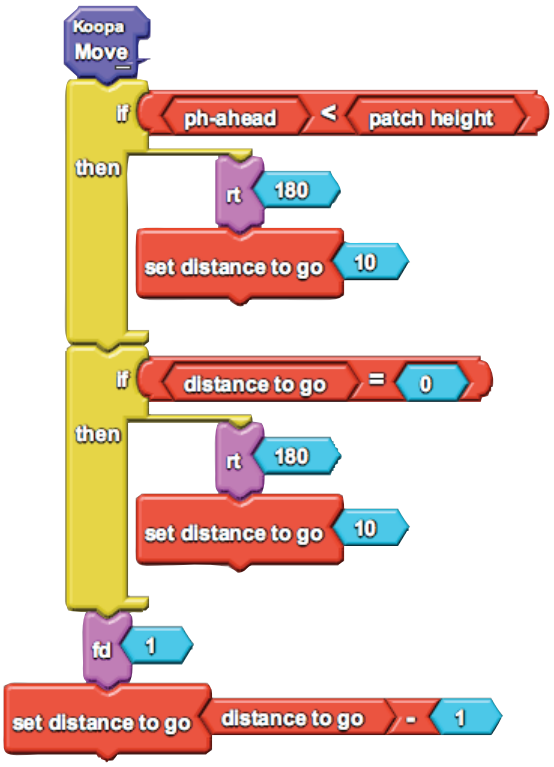

Figure 9: Koopa's Move function.

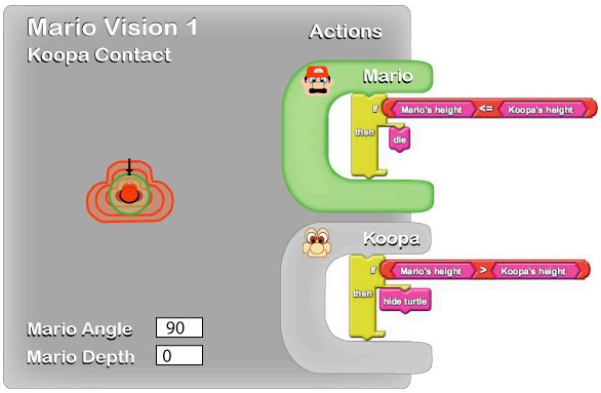

Figure 10: Mario Vision block describes what happens when Mario collides with a Koopa.

7 Field Testing

While StarLogo TNG is still in development, it has reached a point in development where we have been able to do pilot testing with small groups of students, and classroom usability focus groups with teachers. Both of these mechanisms have helped guide the development process, and provide comparative information with the previous version of StarLogo. Below we describe a case study from one of the pilot tests with students, as well as some of the feedback we have received from the teacher focus groups.

7.1 Pilot Study

Our case studies employ comparative tests between the existing StarLogo and StarLogo TNG. We have targeted students who have familiarity with the concepts of StarLogo, but have not done any programming. In one case study we worked with pairs of students at a Boston metropolitan area private school, in which students have used StarLogo models. In a 90 minute session the students were given a simple programming task, along with instructions, in one environment and then the other. The programming task, at its minimum, involved 6–10 lines of code. Students were videotaped during their programming session, and also debriefed afterwards. This particular case was with two tenth grade girls (Alice and Beth) who had never programmed before.

They strongly preferred the StarLogoBlocks programming paradigm to the text based paradigm of the existing StarLogo. Specifically, they pointed out the way in which you could follow the flow of programs visually, and that you didn't need to worry about the syntax.

Alice: It is easier to see the commands too because instead of typing in random things that you don't know what they really mean this is like a puzzle piece and you can kind of put it together.

Beth: You can tell if you are doing it right if the puzzle pieces fit too. Because before I was like questioning myself if I was doing it right like bracket or space. I don't know.

They both expressed satisfaction with the empowerment that programming provided them. Being able to take control over the program was clearly more satisfy-

ing than simply manipulating programs.

Alice: I definitely think this one is a plus, since we can figure out how to program. Since sometimes in a simulation you want to change it. I'm just the kind of person who wants to put their own stuff in there and see what happens. And I haven't been able to do this because I'm not a programmer.

It is of note that they do not consider themselves programmers, even after writing a program. Two boys who were doing the activity simultaneously called themselves programmers at the end, though one had never done any computer programming, and the other had only "programmed" web pages in HTML. In other tests and focus groups we have consistently received highly positive feedback from girls and women who have felt intimidated by programming languages, which to them have been seen as male-oriented.

The 3D environment also appealed to these girls. While they expressed an understanding of the value of the 2D top-down view for getting a sense of the system level dynamics (which we have since given a greater presence in the new version as a result), they liked that you could focus on the interactions of a single turtle or small number of turtles in this version.

Beth: I think what this does with the 3D too, is like you can track one turtle and just track his whole course. You can follow him through. And that can help you understand it.

Alice: With the other one, we've been doing evolution models in class. And we had to see when they reproduced what color came off, like this might be able to help us see which ones thrive and when they reproduce and what happens to them.

Understanding the value and role of the systems-level view and individual view shows that these students have been able to make a huge conceptual leap. Knowing that the large-scale patterns are the results of these individual interactions is a start at overcoming the Centralized Mindset (Resnick 1994). It also allows them to start generating hypotheses about the specifics of how these two scales are connected. But the graphically rich 3D world also makes the world more tangible to them. The 2D world, while illustrative, is limited in its ability to provide a sense of the world. The 3D world has customizable characters, terrains and environmental context. This aspect appealed to them. In the following example, one of the students is commenting on the potential she sees for customizing the backgrounds to convey information about the model.

*B*eth: This could be cool too because you can see the environment in the background. We had a lot of ones in the other one where we had a drought. It would be cool this way since you could see what was happening to the turtles.

The backgrounds, and other graphically rich components of the visualized 3D StarLogo world are an easy for programmers to customize their world, and convey information about the context of the model to their audience. The fact that the background, turtles, or landscape look like something recognizable has tremendous explanatory power.

7.2 Feedback from Teachers

We have also done several focus groups with teachers who have used the existing StarLogo to develop simulations. While a few initially have expressed some reservations about their existing programming skills becoming obsolete, they have embraced the StarLogoBlocks programming metaphor and feel that it will provide the necessary comfort for introducing the students to programming — first by program deconstruction and then through model building. Most of the feedback has been quite positive, while other feedback has allowed us to modify the programming and runtime worlds to better accommodate a variety of programmers and learners.

On the positive side, nearly everyone who has seen the blocks has commented on how much easier it is to see the flow of the programs, and that it relieves the stress of having to remember all of the syntax. We have, however, been asked if it will be possible to "drop down" to the text level after setting things up. Our answer is "no". It is our goal to make it possible to construct sophisticated programs using this paradigm, and we are not treating the blocks as a starting place. Algebraic expressions have been problematic, in that it takes several clicks and drags to write expressions. As a result, we have revised the layout of algebraic expressions to appear less procedural, and will eventually add an algebra-specific mode

that will allow basic mathematical expressions to be entered by keyboard and laid out automatically in blocks.

Other feedback on the programming side has led to collapsible procedures and zooming features that enable the navigation of multiple procedures and lengthy code. As mentioned previously, many users have valued the complimentary perspectives of the 2D and 3D views, prompting us to redesign the runtime interface with the ability to show both 2D and 3D simultaneously, or simply one or the other. The only major concern that teachers have expressed is whether it will run on their computers, due to the 3D graphics card requirements, which are relatively low for 3D games, but may not work on older computers without dedicated graphics cards.

8 Conclusion

The testing and focus groups that we have done so far have suggested that the blocks design will promote the use of StarLogo as a programming tool in the classroom. Teachers feel more secure in teaching this paradigm to their students, and students have been able to easily decode programs written in this way. While it is still early to tell, it seems that it may also provide a more female-friendly programming environment than other tools.

The 3D world is "definitely cool", in the words of many of the students. While teachers have been more lukewarm in their reception of this aspect, they recognized that it will be more attractive to students. It is clear that both 2D and 3D representations will have a place in this environment, but the ability to customize the world and make it look like something recognizable (though not realistic — which is important in conveying that these are simply models) is a big plus for students as they approach modeling for the first time. It also allows students to invest themselves personally in their products, which is empowering.

StarLogo TNG will be in development for some time. Some of the next steps involve adding hooks like network connectivity and joystick control, in order to add to the attraction of those wishing to build games. But we also will be building libraries of blocks for specific academic domains (e.g. ecology, mechanics, etc.) that will allow students to quickly get started building science games using some higher level commands, which can in turn be "opened up" and modified as their skills progress.

Acknowledgments

The authors would like to thank all of the undergraduate researchers who worked on the implementation of StarLogo TNG. We thank Hal Scheintaub for welcoming us into his classroom for our pilot study. The research in this paper was supported in part by a National Science Foundations ITEST Grant (Award #0322573).

References

Agalianos, A., Noss, R. & Whitty, G. (2001), 'Logo in mainstream schools: the struggle over the soul of an educational innovation', *British Journal of Sociology of Education* **22**(4), 479–500.

American Association for the Advancement of Science (1993), *Project 2061: Benchmarks for Science Literacy*, Oxford University Press, New York.

Begel, A. (1996), 'LogoBlocks: A graphical programming language for interacting with the world'. Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science.

Begel, A. (1997), Bongo: A kids' programming environment for creating video games on the web, Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science.

Budge, B. (1983), 'Pinball construction set', Electronic Arts Inc.

Clickteam S.A.R.L. (2004), 'The Games Factory'. http://www.clickteam.com/English/tgf.htm.

Conway, M., Pausch, R., Gossweiler, R. & Burnette, T. (1994), Alice: A rapid prototyping system for building virtual environments, *in* 'Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems', Vol. 2 of *SHORT PAPERS: Designing Interaction Objects*, p. 295.

Cypher, A. & Smith, D. C. (1995), Kidsim: End user programming of simulations, *in* 'Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems', Vol. 1 of *Papers: Programming by Example*, pp. 27–34.

di Sessa, A. & Abelson, H. (1986), 'Boxer: a reconstructible computational medium', *Communications of the ACM* **29**(9), 859–868.

Europress Software Ltd. (2004), 'Klik & Play'. http://www.clickteam.com/English/klilk&play.htm.

Harel, I. (1990), 'Children as software designers: A constructionist approach for learning mathematics', *The Journal of Mathematical Behavior* **9**(1), 3–93.

Harel, I. & Papert, S., eds (1991), *Constructionism*, Ablex Publishing, Norwood, NJ.

Ingalls, D., Kaehler, T., Maloney, J., Wallace, W. & Kay, A. (1997), Back to the future: The story of Squeak, A practical Smalltalk written in itself, *in* 'OOPSLA '97 Conference Proceedings: Object-Oriented Programming Systems, Languages, and Applications', ACM Press, pp. 318–326.

Kafai, Y. B. (1995), *Minds in play: Computer game design as a context for children's learning*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Kafai, Y. B. (1996), 'Software by kids for kids', *Communications of the ACM* **39**(4), 38–39.

Logo Computer Systems Inc. (2004), 'Microworlds Logo'. http://www.microworlds.com.

Lucena, A. T. (n.d.), 'Children's understanding of place value: The design and analysis of a computer game'. http://www.stagecast.com/cgi-bin/templator.cgi?PAGE=Corporate/ presentations/PRESENTATIONS.

Murnane, R. J. & Levy, F. (1996), *Teaching the New Basic Skills, Principles for Educating Children to Thrive in a Changing Economy*, Free Press, New York.

National Research Council (1999), *Being Fluent with Technology*, National Academy Press, Washington, DC.

Papert, S. (1980), *Mindstorms: Children, computers, and powerful ideas*, Basic Books, New York.

Papert, S. (1993), *The Children's Machine: Rethinking School in the Age of the Computer*, Basic Books, New York.

Repenning, A. & Sumner, T. (1992), Agentsheets: A tool for building visual programming environments, *in* 'Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems – Posters and Short Talks', Posters: Helping Users, Programmers, and Designers, p. 30.

Resnick, M. (1994), *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds (Complex Adaptive Systems)*, MIT Press, Cambridge, MA.

Resnick, M. (2002), Rethinking learning in the digital age, *in* G. Kirkman, ed., 'The Global Information Technology Report: Readiness for the Networked World', Oxford University Press, Oxford, UK.

Resnick, M., Martin, F., Sargent, R. & Silverman, B. (1996), 'Programmable bricks: Toys to think with', *IBM Systems Journal* **35**(3-4), 443–452.

Resnick, M., Rusk, N. & Cooke, S. (1998), The Computer Clubhouse, *in* D. Schon, B. Sanyal & W. Mitchell, eds, 'High Technology and Low-Income Communities', MIT Press, Cambridge, MA, pp. 266–286.

State of Maine, D. o. E. (2004), 'Maine learning technology initiative'. http://www.state.me.us/mlte/.

The Education Arcade (2004), 'Education Arcade'. http://www.educationarcade.org.

Wolfram, S. (2002), *A New Kind of Science*, Wolfram Media Inc., Champaign, IL.