CHAPTER 10

Designing for Tinkerability

MITCHEL RESNICK AND ERIC ROSENBAUM

Introduction

Make magazine. Maker Faires. Makerspaces. Maker clubs. In the past few years, there has been a surge of interest in *making*. A growing number of people are becoming engaged in building, creating, personalizing, and customizing things in the world around them—making their own jewelry, their own furniture, their own robots. The emerging Maker Movement is catalyzed by both technological and cultural trends. New technologies are making it easier and cheaper for people to create and share things, in both the physical world and the digital world. At the same time, the Maker Movement builds upon a broader cultural shift toward a do-it-yourself approach to life, where people take pride and pleasure in creating things personally rather than only consuming mass-produced goods.

Although most people involved in the Maker Movement are not focused explicitly on education or learning, the ideas and practices of the Maker Movement resonate with a long tradition in the field of education—from John Dewey's progressivism (Dewey, 1938) to Seymour Papert's constructionism (Papert, 1980, 1993)—that encourages a project-based, experiential approach to learning. This approach is somewhat out of favor in many of today's education systems, with their strong emphasis on content delivery and quantitative assessment. But the enthusiasm surrounding the Maker Movement provides a new opportunity for reinvigorating and revalidating the progressive-constructionist tradition in education.

To do so, however, requires more than just "making." Too often, we have seen schools introduce making into the curriculum in a way that saps all the spirit from the activity: "Here are the instructions for making your

uncorrected proof pdf
not for distribution

robotic car. Follow the instructions carefully. You will be evaluated on how well your car performs." Or: "Design a bridge that can support 100 pounds. Based on your design, calculate the strains on the bridge. Once you are sure that your design can support 100 pounds, build the bridge and confirm that it can support the weight."

In these activities, students are making something, but the learning experience is limited. Just making things is not enough. There are many different approaches to making things, and some lead to richer learning experiences than others. In this chapter, we focus on a particular approach to making that we describe as *tinkering*. The tinkering approach is characterized by a playful, experimental, iterative style of engagement, in which makers are continually reassessing their goals, exploring new paths, and imagining new possibilities. Tinkering is undervalued (and even discouraged) in many educational settings today, but it is well aligned with the goals and spirit of the progressive-constructionist tradition—and, in our view, it is exactly what is needed to help young people prepare for life in today's society.

Our primary goal in this chapter is to examine strategies for encouraging and supporting a tinkering approach to making and learning. We view this as a design challenge: How can we design technologies and activities for *tinkerability*? We start, in the next two sections, by giving a fuller description of what we mean by tinkering and why we think it is so valuable as part of the learning process. Then we examine specific technologies and activities we have designed in our research group at the MIT Media Lab, discussing our strategies for encouraging and supporting tinkering.

What is Tinkering?

The term *tinkering* means different things to different people. It is not uncommon to hear the term used dismissively—*just tinkering*—in reference to someone working without a clear goal or purpose, or without making noticeable progress. But in our view, *just* and *tinkering* do not belong together. We see tinkering as a valid and valuable style of working, characterized by a playful, exploratory, iterative style of engaging with a problem or project. When people are tinkering, they are constantly trying out ideas, making adjustments and refinements, then experimenting with new possibilities, over and over and over.

Many people think of *tinkering* in opposition to *planning*—and they often view planning as an inherently superior approach. Planning seems more organized, more direct, more efficient. Planners survey a situation, identify problems and needs, develop a clear plan, then execute it. Do it once and do it right. What could be better than that?

volume pre-pdf
not for distribution

The tinkering process is messier. Tinkerers are always exploring, experimenting, trying new things. Whereas planners typically rely on formal rules and abstract calculations (e.g., calculating the optimal position for a supporting beam in a structure), tinkerers tend to react to the specific details of the particular situation (experimentally trying different locations for the supporting beam—or exploring other ways to support the structure). In the words of design theorist Don Schoen, tinkerers have "a conversation with the material" (Schoen, 1983).

Sometimes, tinkerers start without a goal. Instead of the *top-down* approach of traditional planning, tinkerers use a *bottom-up* approach. They begin by messing around with materials (e.g., snapping LEGO bricks together in different patterns), and a goal emerges from their playful explorations (e.g., deciding to build a fantasy castle). Other times, tinkerers have a general goal, but they are not quite sure how to get there. They might start with a tentative plan, but they continually adapt and renegotiate their plans based on their interactions with the materials and people they are working with. For example, a child might start with the goal of building a security system for her bedroom, and then experiment with many different materials, strategies, and designs before coming up with a final version.

There is a long tradition of tinkering in many cultures around the world. In almost all countries, local craft traditions have evolved over centuries, characterized by experimenting and tinkering with indigenous materials. In many places, people develop a do-it-yourself mindset, sometimes out of economic necessity, making use of whatever tools and materials happen to be available to them at a given time. This style of interaction is sometimes described as *bricolage*. Anthropologist Claude Lévi-Strauss (1966) describes how people in many parts of the world, acting as bricoleurs, continually improvise with currently available materials to build or repair objects in their everyday lives. Lévi-Strauss contrasts the bricoleur with the engineer, who systematically develops a plan, then gathers the materials needed to execute it.

Tinkering and bricolage are closely aligned with play. Many people see play as a form of entertainment or fun, but we see it somewhat differently. To us, play is a style of engaging with the world, a process of testing the boundaries and experimenting with new possibilities. We see tinkering as a playful style of designing and making, where you constantly experiment, explore, and try out new ideas in the process of creating something. Tinkering can be hard work, and sometimes it might not seem like play. But there is always a playful spirit underlying the tinkering process.

People often associate tinkering with physical construction—building a castle with LEGO bricks, making a tree house with wood and nails, creating a circuit with electronic components. The Maker Movement has reinforced

Voorbeeldpagina's Noldus Information

this image, since it focuses on making things in the physical world. But we take a broader view of tinkering. We see tinkering as a style of making things, regardless of whether the things are physical or virtual. You can tinker when you are programming an animation or writing a story, not just when you are making something physical. The key issue is the style of interaction, not the media or materials being used.

Why is Tinkering Important?

Tinkering is not a new idea. From the time the earliest humans began making and using tools, tinkering has been a valuable strategy for making things. But tinkering is more important today than ever before. We live in a world that is characterized by uncertainty and rapid change. Many of the things you learn today will soon be obsolete. Success in the future will depend not on what you know, or how much you know, but on your ability to think and act creatively—on your ability to come up with innovative solutions to unexpected situations and unanticipated problems.

In such a fast-changing environment, tinkering is a particularly valuable strategy. Tinkerers understand how to improvise, adapt, and iterate, so they are never stuck on old plans as new situations arise. Tinkering prioritizes creativity and agility over efficiency and optimization, a worthwhile tradeoff for a constantly changing world.

Despite its benefits, tinkering is often undervalued in today's society, particularly in formal education systems. Schools tend to emphasize the value of planning, teaching students to analyze all options, develop a strategy, then carry out the plans. That's why students who are natural planners tend to do well in school. But what about students who are natural tinkerers? They often feel left out or alienated, especially in science, technology, engineering, and math (STEM) classes, which particularly emphasize top-down planning. Thus, many students are turned off to math and science, leading to a less scientifically literate populace and a restricted pipeline to STEM professions.

It does not have to be this way. STEM disciplines are not inherently planning-oriented. In fact, expert practitioners in STEM disciplines typically employ much more tinkering in their work than is common in STEM classroom activities (Brown, Collins, & Duguid, 1989). Many of the greatest scientists and engineers throughout history—from Leonardo da Vinci to Alexander Graham Bell to Barbara McClintock to Richard Feynman—saw themselves as tinkerers. In order to broaden participation and foster innovation in STEM disciplines, we must rethink and revise STEM curricula to become welcoming and appealing to tinkerers, not just planners.

uncorrected proof
not for distribution

Turkle and Papert (1990) argue for "epistemological pluralism"—that is, respecting and valuing multiple styles of learning and multiple ways of knowing. They suggest that logic and planning should be "on tap" (available as needed for particular situations), not "on top" (assumed to be superior). Because the status of logic and planning is privileged, Turkle and Papert worry that many people will be excluded from STEM disciplines not by explicit rules, "but by ways of thinking that make them reluctant to join in." They argue that tinkering and bricolage should be given equal status with logic and planning: "Bricolage is a way to organize work. It is not a stage in a progression to a superior form" (p. 141).

Many educators are skeptical about tinkering. There are several common critiques. Some educators worry that tinkerers might succeed at creating things without fully understanding what they are doing. That might be true in some cases, but even in those cases, tinkering provides an opportunity for learners to develop fragments of knowledge that they can later integrate into a more complete understanding (Hancock, 2003). Others worry that tinkering is too unstructured to lead to success. That critique confuses tinkering with random exploration. The bottom-up process of tinkering starts with explorations that might seem rather random, but it does not end there. True tinkerers know how to turn their initial explorations (*bottom*) into a focused activity (*up*). It is the combination of *bottom* and *up* that makes tinkering a valuable process.

Of course, top-down planning can be valuable too. But in many settings, planning is viewed as the correct approach for solving problems, not just one of several alternatives. Our goal is to end the privileged status of planning, and give equal emphasis to tinkering.

Computation + Tinkerability

Many materials—such as wooden blocks and modeling clay—support and encourage tinkering, enabling people to create houses, castles, bridges, sculptures, and other structures. But what if you want to create things that move, sense, react, interact, and communicate? That typically requires computational tools and materials. Computational materials might not seem very amenable to tinkering, since computation is often associated with logic and precision. And, indeed, computational activities (particularly programming) have often been introduced in ways that appeal more to planners than tinkerers (e.g., learning how to efficiently sort a list of numbers).

In our Lifelong Kindergarten research group at the MIT Media Lab, we are trying to change the ways that young people use and think about computation. We have developed a collection of computational construction

Voucher copy of not for distribution

kits and activities that explicitly encourage designing and tinkering with computation. In collaboration with the LEGO Group, for example, we developed the LEGO Mindstorms and WeDo robotics kits, which enable young people to build robotics devices that move, sense, interact, and communicate. In the process, young people learn important mathematical, engineering, and computational ideas. Even more important, they learn to think creatively and work collaboratively, essential skills for active participation in today's society.

In this section, we describe two of our group's computational construction kits, Scratch and MaKey MaKey, that are designed explicitly to engage young people in tinkering. In the next section, we use these two kits as the basis for our analysis of how to design for tinkerability.

Scratch

With Scratch (http://scratch.mit.edu), you can program your own interactive stories, games, animations, and simulations—then share your creations online. To create a program in Scratch, you snap graphical programming blocks together into a *script*, much like snapping LEGO bricks together (Figure 10.1). For each character (or *sprite*) in your Scratch project, you need to assemble a set of scripts to control its behavior (Figure 10.2). For a fish sprite, for example, one script might control the motion of the fish across the screen, while another script tells the fish to change directions if it bumps into coral. From a collection of simple programming blocks, combined with images and sounds, you can create a wide variety of different types of projects. Since the launch of Scratch in 2007, young people around the world have shared more than 3 million projects on the Scratch website (Figure 10.3), including interactive newsletters, science simulations, virtual tours, public service announcements, video games, and animated dance contests (Brennan & Resnick, 2012; Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010; Resnick et al., 2009).

As young people create Scratch projects, they typically engage in an extended tinkering process—creating programming scripts and costumes for each sprite, testing them out to see if they behave as expected, then

Figure 10.1 Scratch programming blocks.

Your copy of the book is not for distribution

Figure 10.2 Scratch programming interface.

revising and adapting them, over and over again. To get a sense of this process, consider the work of a Scratch community member who goes by the username EmeraldDragon. During her first seven months in the community, EmeraldDragon shared 25 projects on the Scratch website. In one of her first projects, EmeraldDragon created a game in which a user can control the movements of an animated dragon. She created 12 images of a dragon, each with the dragon's legs in slightly different positions, then created a programming script that cycled through the images to create the appearance of motion, much like a flipbook.

EmeraldDragon experimented with different versions of the script to make the dragon move in different directions when the user pressed different keys. When EmeraldDragon shared the project on the Scratch website, she included the comment: "I was just tinkering with the scripts in the game and I finally figured out how to make it so you can run back and forth! I'll fix up the game and put out the new and improved still not yet a game version!"

EmeraldDragon named her project *My Dragon Game (NOT finished)*, to make clear that the project was still a work-in-progress (Figure 10.4). In her Project Notes, she wrote: "I am working on being able to run back and forth without the rock disappearing. Any tips or help?" In the project's Comments section, other members of the Scratch community offered

Voorpublicatie
not to be distributed

Figure 10.3 Scratch website and online community.

suggestions. EmeraldDragon soon shared a new version of her project, this time with the name *My Dragon Game (Still NOT finished)*.

EmeraldDragon clearly understood that tinkering is an ongoing process of revision and adaptation. As she wrote in her Project Notes: "This is just a stage in a long process."

MaKey MaKey

With MaKey MaKey (http://makeymakey.com), you can create interfaces to the computer out of any object that conducts electricity (e.g., make a piano keyboard out of pieces of fruit, or use Play-Doh to create a controller for a PacMan game) (Silver & Rosenbaum, 2012). To make a new interface, you connect objects with alligator clips to the MaKey MaKey circuit board (Figure 10.5), which plugs into a computer's USB port. MaKey MaKey pretends it is a computer keyboard, so you can make your own keys. You

Your copy of the draft not for distribution

Figure 10.4 EmeraldDragon's game.

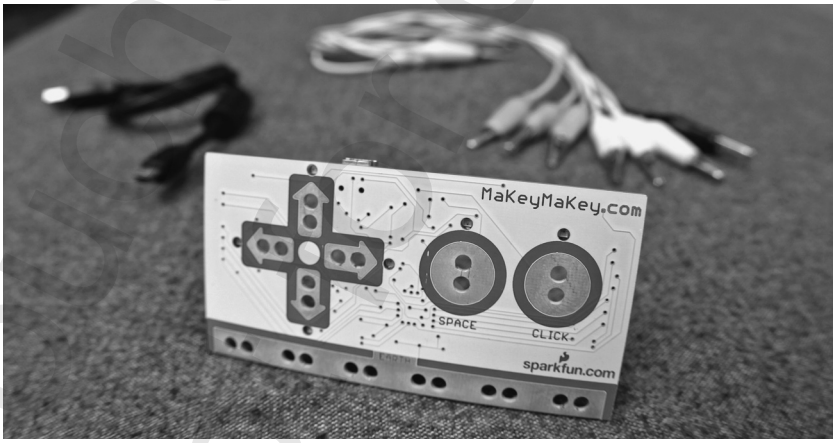

Figure 10.5 MaKey MaKey circuit board, with a USB cable and alligator clips.

version of pdf not for distribution

can replace your space bar, or any other key, with a banana, or with any other object that conducts even a little bit of electricity. MaKey MaKey detects when you touch an object to complete a circuit and sends a signal to the computer that a key has been pressed (Figure 10.6).

Here is a scenario, based on several projects we observed at workshops, that illustrates the workings of MaKey MaKey in more detail. Anna, a 12-year-old girl, is fascinated by a banana piano she saw in a MaKey MaKey video (Figure 10.7), and she decides she wants to make one herself. First, she finds a web page that lets her play a scale on the piano by typing the letters on the home row of her computer keyboard: a, s, d, f, etc. Next, she plugs her MaKey MaKey circuit board into her computer so she can make

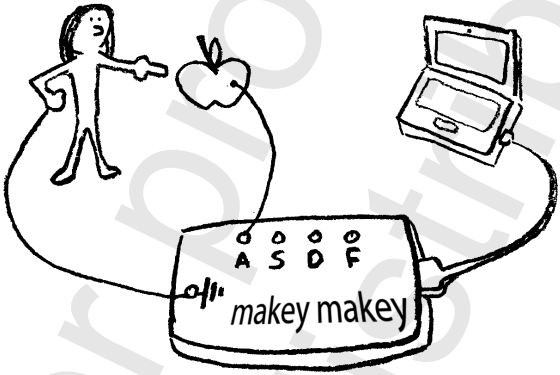

Figure 10.6 A MaKey MaKey circuit.

Figure 10.7 Banana piano.

Version proof pdf
not for distribution

her own keys that trigger piano notes. She lays out a row of bananas and uses alligator clips to connect each one to a different letter on the MaKey MaKey. Then she connects herself to the MaKey MaKey by connecting a lime to "earth" (also known as "ground") on the MaKey MaKey, and holds the lime in her hand. Now, when she touches one of the bananas, a circuit is completed: a tiny amount of current flows from the "a" key input on the MaKey MaKey, through a banana, through her, through the lime, and back to "earth" on the MaKey MaKey. The MaKey MaKey detects the connection, and tells the computer that the "a" key has been pressed, causing the piano web page to play a musical note.

As she plugs in more bananas, Anna makes some accidental discoveries. At first, the bananas are in the wrong order, so instead of forming a scale they play a fragment of a familiar melody. This leads to some playful rearranging of bananas as a musical experiment. During this process, Anna notices that two of the bananas are touching, so that when she touches either of them they both trigger notes, playing a chord. Later, Anna's friend Leo drops by, and together they try making a circuit through both of their bodies. If Anna holds the lime and Leo touches a banana, they can trigger a note by giving each other a high-five—or when Anna taps Leo on the nose. This leads to the idea of making a human drum kit: they find a web page that plays different drum sounds when you press keys, then gather some other friends to connect into the circuit, so each can trigger a different sound.

Later Anna and Leo experiment with different materials. They search around the house and find several items that work well, including Jell-O, pennies, Play-Doh, and the graphite in a pencil drawing on paper. Forming

Figure 10.8 Play-Doh game controller.

version proof not for distribution

Play-Doh into different shapes leads to the idea of making a game controller, with Play-Doh buttons. By connecting the buttons to the arrow keys on the MaKey MaKey, they can play a Mario Bros. game (Figure 10.8).

Through these tinkering processes with MaKey MaKey, Anna and Leo are able to quickly try out new ideas, pursue serendipitous discoveries, experiment with a range of different physical materials, and create their own inventions in different genres.

Designing Kits for Tinkerability

How do you design for tinkerability? Reflecting on the construction kits our research group has developed over the years, we identified three core principles guiding our designs: immediate feedback, fluid experimentation, and open exploration.

Immediate Feedback

The tinkering process typically involves a series of quick experiments—and to do quick experiments, you need quick results. In highly tinkerable construction kits, there is a very short interval of time between making a change and seeing its effect. Many physical processes have this property, of course: when you fold a sheet of paper, you do not have to wait to see the crease. But some physical processes—such as baking, metal casting, or gluing—do require a wait, so tinkering with them is more challenging. Some computational construction kits are stuck in a paradigm from the past, when computation was slow and you had to wait to see the results of your program. We design our kits for immediate feedback, so you can quickly see the results of your actions—and also see a representation of the process as it plays out.

See the Result

In Scratch, you can simply click on a programming block and watch what happens. There are no separate steps for compiling, no separate modes for editing. For most blocks, results are immediate, such as a movement, color change, or sound effect. You learn what the blocks do simply by trying them out. You can even click on a block while it is still in the palette, without even dragging it to the scripting area (where you connect blocks together into scripts).

Ideally, tinkering should be a continuous, ongoing process. In Scratch, you can continue to tinker with Scratch scripts even as they are running. For example, you might start by clicking on a Scratch script to make a sprite move across the screen, then click on other scripts to make them run in parallel (e.g., adding a soundtrack or animating the sprite) while the first

voorbeeldpagina pdf
niet voor distribution

script is still running. You can also modify a script while it is running (e.g., changing the number in a *move* block to increase the speed of a sprite) or even insert a new block into the script. This "liveness" allows you to try out many possibilities and see the results immediately, in the context of a running program. The goal is to make people feel they can interact with programming blocks the same way they interact with physical objects.

MaKey MaKey is also designed for immediate feedback. Let's say you make a keypad out of Play-Doh to control a video game character. As soon as you touch the Play-Doh (to complete the circuit), the character on the screen moves. You can also test out the MaKey MaKey circuit board before creating any interface. The board itself has conductive pads that allow you to complete a circuit with your fingers. Touch the "earth" pad (also known as "ground") with one finger and the "space" pad with another finger, and MaKey MaKey will make the computer think that the keyboard spacebar has been pressed.

See the Process

In most programming environments, it is not possible to directly observe the internal properties of the program while it is running. These properties include what the program is doing right now (the current location in its execution) and what the program "knows" (the values of its variables and other data structures). Just as it is sometimes easier to understand what is wrong with a car by looking under the hood (or, at least, by watching the indicator lights on the dashboard), it is easier to fix bugs in a computer program when you can see its internal properties while it is running.

MaKey Makey has some simple indicators of its internal process: LED lights on the board indicate when a circuit has been completed, so you can debug the board and circuit independent of what is happening on the computer screen.

Scratch has a range of features that allow you to monitor programs as they run. Scratch scripts always highlight while they are running, so you can see which code is being triggered when. If you select *single-stepping* mode, each individual block within a script highlights as it runs, allowing for more fine-grained analysis and debugging.

The Scratch interface also has optional monitors that allow you to see the current values of data stored in variables and lists. In most other programming languages, data structures are hidden from view, but Scratch data structures are visible and manipulable, enhancing tinkerability. As a Scratch program runs, you can watch the variables and lists update in real time. You can also type directly into list monitors, modifying the values of items in a list, even while a program is running. The goal is to allow Scratchers to tinker with data—and to gain a better understanding of how the data relate to the rest of the program.

you are a proof of not for distribution

Fluid Experimentation

The tinkering process is inherently iterative. Tinkerers start by exploring and experimenting, then revising and refining their goals, plans, and creations. Then they are ready to start a new cycle of exploring and experimenting, then revising and refining, over and over. The quicker the iteration, the faster the generation and refinement of ideas. To support this style of interaction, we design our constructions kits so that it is easy for people to get started with experimenting—and then easy to continue experimenting by connecting (and disconnecting and reconnecting) objects within the project.

Easy to Get Started

One of the biggest challenges in tinkering with technological tools is the time it takes to get started. When you have an idea you want to express, or some materials you want to experiment with, you want to dive in right away without spending a lot of time setting up. Electronics projects often require basic infrastructure to be set up, such as wiring on a breadboard, before you can start interacting with new parts. Similarly, in programming, some setup code is often required before you can begin expressing new ideas. In order to support fluid experimentation, we design our kits to minimize these setup processes as much as possible.

When you launch Scratch, you can start trying things right away. There is a default character (the Scratch cat), which already has some media to play with: two images that form a walking animation, and a "meow" sound. You can start programming behaviors for the cat immediately: click the *move* block and the cat moves; click the *next costume* block and the cat animates; click the *play sound* block and the cat meows. The blocks start with reasonable default values for their inputs, so you can start playing with the blocks right away, without filling in any inputs.

MaKey MaKey requires no configuration on the computer because it is a plug-and-play USB device that appears to the computer to be a standard mouse and keyboard. The system works with any software or website that responds to keyboard and mouse commands; no special software is needed. The MaKey MaKey circuit board has a fixed arrangement of inputs mapped to keyboard and mouse events, so it also needs no configuration before you begin. You can plug the board into the computer and start making circuits (and keystrokes) right away. Without any setup, you can start experimenting with different physical materials and make your own physical interface.

Easy to Connect

The tinkerability of a construction kit is determined, to a large degree, by how parts of the construction kit connect with one another. In designing

you can find pdf not for distribution

connectors for our construction kits, we take inspiration from LEGO bricks: they are easy to snap together and also easy to take apart, and they have just the right amount of "clutch power" to make structures sturdy. This carefully chosen compromise is what makes LEGO kits so tinkerable. Plain wooden blocks are easier to stack up but also easier to knock down. At the other extreme, Erector sets are good for making stronger structures, but the structures are more difficult to put together and take apart. LEGO bricks are in between, enabling quick, iterative experimentation.

In the MaKey MaKey kit, alligator clips are used to attach homemade switches to holes in the circuit board. The clips are quick and easy to use, so you can rapidly test different connections and swap them around, but they provide enough mechanical strength to hold together under the strain of gameplay or musical performance. This makes them more tinkerable than other typical electronics connectors such as female headers (small holes that wires can be fed into but easily slip out of) or screw terminals (which hold wires firmly but require a screwdriver to open and close).

In Scratch, the shapes of the programming blocks constrain how the blocks are put together. When you pull blocks from the palette, it is immediately obvious, from the shapes of the blocks, which blocks can be connected with one another. Unlike traditional text-based programming languages, there is no obscure syntax (semicolons, square brackets, etc.) to learn. Instead, the grammar is visual, indicated by the shapes of the blocks and connectors. Blocks snap together only if the combination makes sense. Of course, the blocks might not behave as you intended, but there can be no syntax errors. If blocks snap together, they are sure to run.

Blocks that take inputs have "sockets" of different shapes, indicating what type of block should go inside. For example, the *move* block has an oval socket, indicating that it expects a number as its input. You can insert any oval-shaped block—such as the block that reports the y-position of the mouse cursor. The conditional *if* and *if-else* and *wait-until* blocks have a hexagon-shaped input, indicating that they expect a Boolean (true-false) value as input. For example, you can insert a hexagonal *touching?* block that reports (true or false) whether the sprite is touching another sprite (see Figure 10.1). Some blocks are more tolerant. The *say* block, for example, has a square socket indicating that it can take any type of input (a number, a string, or a Boolean).

The shapes and connectors make it easy to tinker and experiment with the Scratch programming blocks, just as with LEGO bricks. You can snap blocks together, try them out, then disconnect them and try other combinations. The cost of experimentation is low. Also, you can leave some blocks (or stacks of blocks) lying around your workspace (see Figure 10.2), in case you might want to use them later, just as you would with LEGO

Volume proof not for distribution

bricks—unlike most programming environments, where you need to keep your workspace tidy and organized.

Open Exploration

Supporting immediate feedback and fluid experimentation is not enough. It is also important to enable and inspire people to explore a diverse array of possibilities. To do that, construction kits must support a wide variety of materials and a wide variety of genres.

Variety of Materials

Scratch comes bundled with a large library of media intended to spark new project ideas; these media include images and backgrounds, sound effects and music loops, and sprites with programming scripts and media already embedded in them. Scratch also provides several ways to create and import your own media. You can create images with the built-in paint editor, take photos with your computer's camera, or record sounds with the built-in recorder. You can also import images and sounds from your hard disk or the web, simply by dragging and dropping them into Scratch.

Even more important, Scratch provides access to an ever-growing and evolving library of projects created by other members of the Scratch community. You can grab images, sounds, and scripts from other projects and integrate them into your own. Everything shared on the Scratch website is covered by the Creative Commons share-alike license, so community members are free to borrow from one another, as long as they give credit. Roughly one third of the 3 million projects on the Scratch websites are remixes, in which one community member built upon the work of another. The website serves as a continuing source of inspiration, with thousands of new projects shared every day.

Unlike Scratch, MaKey MaKey comes with no materials at all (aside from the circuit board and connectors). Instead, the kit is designed to encourage you to see the world as your construction kit. You can make MaKey MaKey circuits and switches out of anything that conducts electricity—food, plants, pencil drawings, aluminum foil, water, Play-Doh, even your own body. For making structures to support the circuit, you can use any object or material (insulating or conducting), anything from cardboard to beach balls to buckets to hats.

Variety of Genres

When starting out, tinkerers often have no clear idea of what they want to make. So construction kits that support many different genres of projects are useful, providing they offer the flexibility to shift genres as users iteratively adapt and revise their creations over time.

you may not pdf not for distribution

Unlike many software construction kits (such as Gamemaker and Gamestar Mechanic—which, as their names suggest, focus explicitly on games), Scratch enables you to create a wide range of different types of projects, including interactive stories, games, animations, simulations, art, and music. Different parts of the Scratch construction kit are well tuned for different genres. The paint editor and image-effect blocks support animation; *say* and *think* blocks (for making speech balloons) support stories; collision-detection and keyboard-interaction blocks support games; pen blocks (to make sprites draw as they move) support interactive art; mathematical-operation blocks support simulations; and note, instrument, and tempo blocks support music. Because these tools are all in the same kit, a single Scratch project can blend together multiple genres. A tinkerer can start working on one type of project and later decide to morph it into a different genre (or combination of genres) as it develops (e.g., transforming an animation into a game or a simulation into interactive art).

MaKey MaKey also supports creations in a variety of genres, because it can be used to create a controller for any piece of software that can be controlled with a keyboard. For example, you can create a MaKey Makey interface device to control a video game, a sound synthesizer, a video player, a text editor, a web browser, a paint program, or even a Scratch project.

Tinkering with Tinkerability

In this chapter, we have outlined some of our current thinking on how to design construction kits for tinkerability. But designing construction kits is only part of what's needed. Even the most tinkerable construction kit will not be successful unless it is accompanied by the right types of activities, support materials, facilitation, space, and community. In short, designing *contexts for tinkerability* is as important as designing kits for tinkerability.

It is beyond the scope of this paper to examine these issues in depth. But, in closing, we share a short summary of some key lessons we have learned in designing contexts for tinkerability. These ideas can be useful for educators who want to support young people in the process of designing and tinkering.

- *Emphasize process over product.* While making something is an important part of the tinkering process, too much emphasis on the final product can undermine the experimentation that is at the heart of tinkering. To engage people in thinking about the tinkering process, encourage them to document and discuss intermediate stages, failed experiments, and sources of inspiration.

your proof pdf not for distribution

- *Set themes, not challenges.* Rather than posing challenges to solve (as is typical in many design workshops), propose themes to explore. Select workshop themes that are broad enough to give everyone freedom to work on projects that they care about, but specific enough to foster a sense of shared experience among participants (Rusk, Resnick, Berg, & Pezalla-Granlund, 2008). For example, we might ask workshop participants to design an interactive card for a holiday celebration.
- *Highlight diverse examples.* Show sample projects that illustrate the wide diversity of what is possible, provoking people to think divergently. Keep examples and documentation on display for continuing inspiration.
- *Tinker with space.* Consider how you might rearrange or relocate, to open new possibilities for exploration and collaboration. For example, how can the arrangement of tables and screens help people see each other's work? How can the arrangement of materials encourage clever and unexpected combinations?
- *Encourage engagement with people, not just materials.* In addition to having a "conversation with the material," tinkerers also benefit from having conversations (and collaborations) with other people.
- *Pose questions instead of giving answers.* Resist the urge to explain too much or fix problems. Instead, support tinkerers in their explorations by asking questions, pointing out interesting phenomena, and wondering aloud about alternative possibilities.
- *Combine diving in with stepping back.* While it is valuable for tinkerers to immerse themselves in the process of making, it is also important for them to step back and reflect upon the process.

Our goal is to provide everyone—of all ages, backgrounds, and interests—with new opportunities to learn through tinkering. To do this well, we ourselves need to remain engaged as tinkerers (and meta-tinkerers), playfully experimenting with new ways to design for tinkerability. We see this chapter as just a start. We plan to continue to tinker with and iterate ideas and technologies, continuing to refine our thinking about the nature of tinkering and strategies for enhancing it.

Acknowledgments

Many members of the Lifelong Kindergarten research group at the MIT Media Lab contributed to the ideas and projects discussed in this paper. We are grateful to Seymour Papert and Sherry Turkle, who have deeply influenced the ways we think about tinkering and learning.

Voucher proof pdf
not for distribution

References

Brennan, K., & Resnick, M. (2012). Imagining, creating, playing, sharing, reflecting: How online community supports young people as designers of interactive media. In N. Lavigne, & C. Mouza (Eds.), *Emerging technologies for the classroom: A learning sciences perspective*. New York: Springer, pp. 253–268.

Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational Researcher*, *18*(1), 32–42.

Dewey, J. (1938). *Experience and education*. New York: Simon & Schuster.

Hancock, C. (2003). *Real-time programming and the big ideas of computational literacy*. Doctoral dissertation. MIT Media Lab, Cambridge, MA.

Lévi-Strauss, C. (1966). *The savage mind*. Chicago, IL: University of Chicago Press.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 1–15.

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

Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. New York: Basic Books.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67.

Rusk, N., Resnick, M., Berg, R., & Pezalla-Granlund, M. (2008). New pathways into robotics: Strategies for broadening participation. *Journal of Science Education and Technology*, *17*(1), 59–69.

Silver, J., & Rosenbaum, E. (2012, February 19–22). Makey Makey: Improvising tangible and nature-based user interfaces. Paper presented at the Sixth International Conference on Tangible, Embedded and Embodied Interaction, Queen's Human Media Lab, Kingston, Ontario. *Proceedings of the International Conference on Tangible, Embedded and Embodied Interaction*. New York: ACM.

Schoen, D. (1983). *The reflective practitioner: How professionals think in action*. London, UK: Maurice Temple Smith.

Turkle, S., & Papert, S. (1990). Epistemological pluralism. *Signs*, *16*(1), 128–157.

voucher proof pdf
not for distribution