

Tech-A-Sketch

Instructor Guide

Grades 3–5

TinkRworks Client Success Team

Contact us: Support@TinkRworks.com

Project Overview

The Tech-A-Sketch project is a comprehensive, hands-on STEAM experience where students in grades 3–5 explore the intersection of computer science, digital art, and engineering. Across 14 lessons, students design, build, and program a hand-held color digital drawing device inspired by the classic Etch A Sketch, using knobs, buttons, and an accelerometer to create original artwork on a color screen.

What Students Will Create

By the end of this project, each student will have a fully functional Tech-A-Sketch that:

- Displays a custom background composed of coded shapes and colors
- Responds to knob movements to move a drawing brush across the screen
- Uses buttons to cycle through brushes, clear the screen and lift the virtual brush
- Shakes to erase—just like the classic Etch A Sketch!
- Features personalized custom brushes and widget tools they programmed themselves

Essential Question

How can we use electronics and coding to create a color digital drawing device?

This question drives the entire project. Students won't just learn how circuits work or how to code. They'll understand why these tools matter and how technology can be designed as a creative instrument.

The Learning Journey

Students begin by exploring X-Y coordinate systems through grid games, then move into the fundamentals of computer programming through algorithm activities. They assemble their Tech-A-Sketch hardware, connect to TinkRcode, and progressively build more sophisticated skills: filling the screen with color, placing rectangles with precise coordinates, creating movable brushes controlled by knobs, adding button functionality, building toolkits with multiple brushes, designing coded backgrounds, adding widgets and erasers, exploring the accelerometer, and finally programming their complete project with all features combined.

Interdisciplinary Learning

Tech-A-Sketch naturally integrates computer science (variables, conditionals, loops, subroutines, debugging), mathematics (X-Y coordinate systems, pixel dimensions, RGB values), visual arts (color theory, composition, design principles), engineering (circuit board components, physical assembly, troubleshooting), and language arts (algorithm writing, technical documentation through the Discovery Journal).

This project meets NGSS Performance Expectations while developing cross-curricular skills that extend well beyond the classroom.

What Makes This Project Special

Unlike projects where students follow predetermined steps, Tech-A-Sketch gives students genuine creative agency. There is no single 'right' background, no required brush shape, no prescribed final product. Each Tech-A-Sketch reflects its creator's artistic vision and programming choices.

The project also introduces programming concepts that transfer to any coding language. When students understand variables, conditionals, and subroutines through a project they care about, those concepts stick. Students aren't just learning to code. They're building real tools they control.

For Grades 3–5 Learners

This project is designed for the curiosity and capability of upper elementary students. Concepts build incrementally, with each lesson creating a foundation for the next. Students who finish early have genuine extension opportunities, and students who need more time can focus on core skills without missing the heart of the project. Creativity is valued alongside technical accuracy throughout.

The Teacher's Role

Your job is to guide, support, and celebrate. The lessons are student-driven, with built-in opportunities for discovery and creative decision-making. You'll facilitate discussions, support troubleshooting, and help students see themselves as programmers and artists. This guide provides comprehensive support for every lesson, including tips for success, common challenges and solutions, and assessment opportunities.

You don't need to be a coding expert. The curriculum is designed so you can learn alongside your students. The slides include presenter notes for each concept, and this guide gives you the context you need to feel confident at every step.

About Lesson Durations

The lesson durations provided are guidelines based on the core slide content and activities. Actual class time will vary based on your students' pace and engagement. Some classes move faster, others slower. Both are perfectly normal.

These durations do not account for Discovery Journal entries, assessment activities, or time for tinkering and exploration. If your students want to experiment further with their brushes, try additional color combinations, or refine their backgrounds, encourage it. The best learning often happens in those unscripted moments. Plan for flexibility, and don't worry if a lesson stretches across two class periods. Deeper engagement is always worth the extra time.

The Transformation

By the end of Tech-A-Sketch, your students won't just have built a project. They'll have transformed how they see themselves and technology. They'll know that screens can be programmed, that knobs translate into code, and that a few lines of logic can create something beautiful and personal. They'll have a hand-held color digital drawing device as proof that they are makers, programmers, and artists.

Let's get started!

LESSON 1: INTRODUCTION TO TECH-A-SKETCH

LESSON OVERVIEW

Students are introduced to the project by exploring what an Etch A Sketch is and how it works, then discovering the digital version they will build. The core activity for this lesson centers on X-Y coordinate systems. Students play grid-based games to build the spatial intuition they will need when placing shapes and brushes on their Tech-A-Sketch screens.

Before they ever write a single line of code, students need to understand the coordinate plane their programs will use. The grid activities make this concrete and fun. Whether playing a coordinate version of Battleship ('Sink a Ship') or creating art with plotted points, students discover that precision matters and that small changes in coordinates produce specific results. This is foundational knowledge they'll draw on throughout every programming lesson.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Explain what makes Tech-A-Sketch similar to and different from a classic Etch A Sketch
- Plot points correctly on an X-Y coordinate grid
- Describe what pixels are and how they relate to screens
- Apply coordinate knowledge in a partner game or activity

LESSON DURATION

30–45 minutes

MATERIALS NEEDED

- Pencils (additional)
- Grid Worksheet printouts (one per student, plus extras)

CLASSROOM SETUP

- Print Grid Worksheets before class
- Arrange seating so students can work with partners for the coordinate games
- Have a projector or whiteboard available for demonstrating how coordinates work
- Consider displaying a photograph of a classic Etch A Sketch as a conversation starter

TIPS FOR SUCCESS

Making Coordinates Click

Some students will grasp X-Y coordinates immediately; others will need more time. Use the phrase 'along the corridor, then up the stairs' to help students remember that X is horizontal (left and right) and Y is vertical (up and down). Drawing a large coordinate grid on the board and having students come up to plot points helps the whole class see the concept before they work independently.

A heads-up for later lessons: the Tech-A-Sketch screen uses a coordinate system where (0,0) is the top-left corner and Y increases downward, the opposite of the standard math graph. You don't need to address this now, but knowing it in advance means you won't be caught off guard when it comes up during programming.

Running the Coordinate Games

Two activity options are built into this lesson. 'Sink a Ship' (a Battleship-style game) is great for competitive motivation and gets students plotting coordinates quickly. The art-replication activity (students draw something, write out coordinates, and swap with a partner to recreate the image) is more creative and builds understanding of precision.

Build in time to debrief after the activity: 'What would have happened if you switched the X and the Y?' This question previews exactly the kind of precision students will need in TinkRcode.

Connecting to the Bigger Picture

Even in this first lesson, you can start planting seeds for the project ahead. If you have a Tech-A-Sketch device on hand, show it to students. Ask: 'Where do you think coordinates are used when you draw on this screen?' Let their curiosity build naturally.

WHAT TO DOCUMENT

- Completed Grid Worksheets showing coordinate plotting
- Photos of students doing the Sink a Ship or art-replication activity
- Student answers to the debrief question about what coordinates are for
- Discovery Journal reflections

EXTENSIONS FOR CURIOUS MINDS

- Challenge students to draw a complex shape on the grid and list all the coordinate points that make it up
- Ask: 'If the screen is 15 squares wide and 8 squares tall, what are the coordinates of the center?'
- Explore where else X-Y coordinates show up in everyday life (maps, video games, GPS navigation, graphs in math class)
- Have early finishers try to describe a classmate's location in the room using coordinate language

COMMON CHALLENGES & SOLUTIONS

Students mix up X and Y:

This is extremely common and completely normal. Keep reinforcing: 'X is the horizontal direction (left-right) and Y is the vertical direction (up-down).' Some students benefit from physical movement: stand up, step left or right for X movement; stand up, step forward or backward for Y movement. This concept will keep coming up in future lessons, so don't worry if everyone doesn't have it perfectly yet.

Students struggle with the top-left origin (0,0):

In most math contexts, students learn (0,0) at the bottom-left. Tech-A-Sketch uses a screen coordinate system where (0,0) is the top-left and Y increases downward. If this comes up, a brief explanation helps: 'Screens work differently from math graphs. We start at the top-left corner.' It will make much more sense once students see it on the actual device.

The game gets too noisy or competitive:

Channel that energy! Set a timer, frame it as a friendly challenge, and debrief with questions about what strategies worked. If needed, shift to the art-replication activity instead, which tends to be quieter and more reflective.

ASSESSMENT OPPORTUNITIES

Look for evidence that students can:

- Navigate an X-Y coordinate grid accurately
- Express a location as an ordered pair (X, Y)

- Explain why the order of the two numbers matters
- Describe what pixels are in relation to a screen

Collect completed Grid Worksheets as evidence of understanding. Listen during the debrief for students who can explain their coordinate choices in their own words.

LOOKING AHEAD

In the next lesson, students will explore the foundations of computer programming: what code is, what algorithms are, and how programmers give computers precise instructions. Lesson 2 is optional, so if you plan to teach it, no additional prep is needed beyond pencils and paper. If you're moving directly to the Build, begin gathering kit components so they're organized and ready.

LESSON 2: INTRODUCTION TO PROGRAMMING

(Optional)

LESSON OVERVIEW

This optional lesson introduces computer programming through hands-on analog activities before students ever open TinkRcode. Students discuss what coding means, explore algorithms as step-by-step instructions, and work through an activity where they write and follow directions to draw a simple illustration. The lesson closes with core programming vocabulary that will appear throughout the project.

Making the algorithm activity physical and partner-based is the key to this lesson's impact. When students discover that 'draw a flower' is too vague for a partner to follow accurately, and then revise their instructions to be more precise, they learn through experience why order and specificity matter in code. That's a lesson no lecture can deliver as effectively.

Lesson 2 is marked optional in the lesson plan. If your students have programming experience or if time is limited, you can move directly to Lesson 3. But for classes with no prior coding background, this foundation is worth every minute.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Explain what coding is in their own words
- Define and give an example of an algorithm
- Write a clear algorithm for a simple drawing task
- Identify common coding concepts: loops, variables, debugging and syntax

LESSON DURATION

30–45 minutes (Optional)

MATERIALS NEEDED

- Paper (additional)
- Pencils (additional)

TIPS FOR SUCCESS

Setting Up the Algorithm Activity

The activity works like this: students draw a simple image (a flower, a sun, a kite, etc.) and write step-by-step instructions as they draw. Then they swap instructions with a partner who follows the written directions to reproduce the image. At the end, both students compare the original drawing to the result.

Almost always, the first attempt reveals significant gaps in the instructions. That's the whole point! Let students experience the frustration briefly, then debrief: 'What did you have to be more specific about?' A second round with improved instructions usually produces much better results. That iteration is the most valuable learning in the lesson.

Introducing Vocabulary After the Activity

Introduce programming vocabulary after the activity, not before. Students understand words like 'algorithm,' 'loop,' and 'debugging' much better when they can connect them to something they just experienced. 'You were debugging your algorithm when you revised your instructions after seeing your partner's drawing!' Building a class vocabulary anchor chart that stays up throughout the project helps students revisit these terms throughout the unit.

Managing the Optional Nature

If some students have coding experience and others don't, this lesson is a natural equalizer. Algorithm thinking isn't the same as syntax knowledge. Students who have coded before may actually struggle with articulating precise written instructions, while students without coding experience may excel at clear step-by-step writing. Let that play out. The activity is valuable for everyone.

WHAT TO DOCUMENT

- First-draft and second-draft algorithm drawings side by side
- Photos of student pairs comparing their original image to the partner's interpretation
- Class vocabulary anchor chart with key programming terms
- Student reflections: 'The most important thing about an algorithm is...'

EXTENSIONS FOR CURIOUS MINDS

- Ask students to write an algorithm for a daily task (making a sandwich, tying a shoe) and test it at home
- Introduce the concept of a loop: 'What if you needed to repeat a step many times? How would you write that more efficiently?'
- Challenge students to rewrite their algorithm using fewer steps that still produce the correct result
- Explore: 'What happens if one step is in the wrong order?'

COMMON CHALLENGES & SOLUTIONS

Instructions are too vague the first time:

This is exactly what should happen. Don't step in to fix it. Let students experience the consequence of vague instructions—the drawing doesn't match—and then figure out how to improve. This is the most valuable moment of the lesson.

Students struggle to write because of handwriting or spelling concerns:

Focus on the thinking, not the writing mechanics. Numbered diagrams, arrows, and labeled sketches are all acceptable. The goal is communicating precise steps, not writing complete sentences.

Students who already know coding feel disengaged:

Challenge them to write the most precise algorithm in the class—one that their partner follows correctly on the very first try. Or ask them to add a 'loop' to their algorithm for any steps that repeat.

LOOKING AHEAD

After this lesson, students are ready to begin building. Gather all kit materials before Lesson 3 and organize them so distribution goes smoothly. Having small bowls or containers for hardware during assembly will help students keep track of small parts.

LESSON 3: BUILD

LESSON OVERVIEW

Students assemble their Tech-A-Sketch hardware while learning about the components on the printed circuit board (PCB). This is one of the most exciting days in the project. Students watch their device come together piece by piece and see the electronic components that will soon respond to their code.

For grades 3–5, the hands-on build experience is incredibly motivating. Students take ownership of their device in a way that makes every subsequent programming lesson feel more meaningful. When they later program buttons and knobs, they'll know exactly which physical components they're talking about because they assembled those components themselves.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Identify the key components of a printed circuit board (PCB)
- Assemble the Tech-A-Sketch hardware following a step-by-step process
- Explain what a microcontroller does
- Recognize and name the major components: display, knobs, buttons, microcontroller and USB port

LESSON DURATION

30–45 minutes

MATERIALS NEEDED

All of the following items are included in the kit:

- PCB
- Back plate
- Face plate
- Bolts
- Spacers
- Lock washers
- Cap nuts
- Knobs
- Knob inserts
- Toggle cap
- Bowls or containers to hold hardware during assembly (additional)

CLASSROOM SETUP

- Set up a clear table surface for each student to work flat
- Pre-sort components by type into shared bowls or containers before students arrive—this saves significant class time
- Set up a demonstration area where all students can see you model the assembly steps
- Have a completed Tech-A-Sketch device to show students what the finished product looks like
- Do a quick practice assembly before teaching so you're comfortable with each step

TIPS FOR SUCCESS

Building Together, Step by Step

This lesson works best when you move through the assembly as a class, completing one step before the next. Resist the temptation to let students race ahead. A student who gets stuck two steps behind feels more frustrated than one who stayed with the group. Move slowly and check in after each step. A useful phrase before moving on: 'Everyone hold up your project and show me what it looks like right now.'

Preventing Lost Hardware

Small bolts and cap nuts have a way of rolling off tables and disappearing. Encourage students to work with the project flat on their table surface, keep hardware in the small bowls until needed, and check the floor around their workspace before moving on. Having a few spare bolts, spacers, and cap nuts on hand will save you when pieces inevitably go missing.

Tightening Guidance

Teach students 'snug but not too tight.' Hardware should be firm but not forced. Over-tightening can strip threads or crack acrylic panels. Model the right amount of pressure before students start, and remind them throughout the assembly.

When Something Doesn't Fit

The most common assembly issue is a panel in the wrong orientation. If something doesn't sit flat or bolts don't align, the piece is probably flipped or rotated. Encourage students to compare their project to your demo model before assuming something is broken. Gentle adjustment almost always solves alignment issues.

WHAT TO DOCUMENT

- Photos of students at each key stage of the assembly
- A photo of each student holding their completed Tech-A-Sketch
- Discovery Journal reflections: 'The component I found most interesting was...'
- Photos of the PCB with components labeled by students

EXTENSIONS FOR CURIOUS MINDS

- Challenge students to identify and label every component on the PCB before beginning assembly
- Ask: 'What do you think the traces on the circuit board do?'
- Have students draw a labeled diagram of the front and back of the PCB from memory after assembly

COMMON CHALLENGES & SOLUTIONS

Panels won't sit flat or align:

The most common cause is an upside-down or backwards panel. Compare to the demo model first. If alignment is still off after reorienting, check that all bolts are going through the correct holes and in the correct order.

Knobs are hard to attach:

The inserts can be firm to press in. Encourage students to apply steady downward pressure rather than forcing. If an insert is truly stuck, leave it for now and check after class. It's better to skip that step than to damage the knob. These can be managed after the lesson with a bit more leverage.

Students are significantly faster or slower than others:

For fast finishers, ask them to help a neighbor by describing steps without touching their project. For students who are struggling, pair them with a peer or offer one-on-one guidance during the final steps.

Missing hardware:

Check the floor and under desk edges first—small parts love to hide. A few spare bolts in your kit will handle most situations. If a component is genuinely missing or damaged, reach out to support@tinkrworks.com.

COLLECTING PACKAGING FOR REUSE

If you plan to use Tech-A-Sketch kits with another group of students, collect and save all packaging materials as students unpack their kits during this lesson. The anti-static bags (for PCBs), plastic bags, and any foam or cardboard inserts will be needed when you break down the projects at the end of the unit.

Designate a centralized storage space in your classroom now—before anything gets discarded. Having the original packaging makes disassembly, storage and redistribution much easier and keeps sensitive components protected between uses.

Quick reference: save the anti-static bag (for the PCB), the plastic bag (for small hardware), and any box inserts or foam. These become essential when you reach Lesson 14.

LOOKING AHEAD

In the next lesson, students will connect their assembled devices to TinkRcode for the first time and begin programming. Before Lesson 4, confirm that all students have TinkRcode accounts set up (or can create them) and that devices are accessible. A quick test login before class will prevent connection delays during the lesson itself.

LESSON 4: DISPLAY

LESSON OVERVIEW

This lesson marks the first time students connect their Tech-A-Sketch to TinkRcode and write code that makes something happen on their screen. The lesson begins with an exploration of how screens produce color—specifically how combining red, green, and blue light creates every color on a digital display. Then students log into TinkRcode, connect their projects, and program their first command: Fill Screen Color.

This lesson is a genuine milestone. After building their device in Lesson 3, students will see their code create a visible result on that device. When a screen suddenly fills with the color a student chose, the cause-and-effect moment lands in a way no classroom explanation can replicate. It's the foundation of everything that follows: if you change the numbers, you change what happens.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Explain how red, green, and blue light combine to create colors on a screen
- Log into TinkRcode and connect a Tech-A-Sketch device
- Program a Fill Screen Color command
- Describe what pixels are and how they make up a digital display
- Complete 3 programming challenges independently

LESSON DURATION

45–60 minutes (Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Computer or device with TinkRcode access
- USB cable for device connection

CLASSROOM SETUP

- Confirm TinkRcode accounts are set up for all students before class
- Test at least one device connection before the lesson to verify cables and accounts work
- Have student login information accessible
- Set up your demo device with screen sharing so all students can follow along
- Seat students close enough to see your demo clearly and reference it during their own work

TIPS FOR SUCCESS

The Big Picture First

Before diving into TinkRcode, show students your fully programmed Tech-A-Sketch filling the screen with different colors. Let them see the end goal. 'I programmed this to do that. In a few minutes, you'll program yours.' This creates immediate excitement and helps students understand why they're learning each step.

RGB Colors: The Key Concept

The conceptual shift from paint colors to light colors trips up many students. With paint, red and blue make purple. With light, red and blue make magenta. Red and green make yellow. These feel wrong until students see it for themselves.

A free online RGB color mixer is tremendously helpful here. Show students what happens as you drag the red, green, and blue sliders. Watch their faces when green and red together produce yellow—that surprise is the learning. Keep the core explanation simple: 'Screens make colors by mixing three kinds of light—red, green, and blue. Each tiny dot on the screen is called a pixel, and each pixel can be any combination of those three colors.'

First TinkRcode Connection

The first connection can take longer than expected, especially on school networks. Plan for it. Walk through the steps slowly and together: open TinkRcode, log in, connect the USB cable, click Connect and pair the device, then open the Tech-A-Sketch program. Common connection issues and their fixes are listed in the Challenges section below.

Moving Through Challenges Together

The three challenges give students their first independent programming moments. Resist the urge to solve challenges for students. Instead, ask guiding questions: 'What happens if you change the red value? What if you set blue to zero?' Let them discover. Early frustration often leads to the most satisfying breakthroughs.

WHAT TO DOCUMENT

- Photos of students' screens showing their first programmed color
- Student reflections: 'I made [color] by...'
- Screenshots of the TinkRcode workspace with the Fill Screen Color block
- Discovery Journal entries

EXTENSIONS FOR CURIOUS MINDS

- 'Can you make your screen exactly the color of your shirt?'
- 'What RGB values make white? What makes black?'
- Challenge students to create two colors that look similar but use different RGB values
- Ask: 'Why are the primary colors of light different from the primary colors of paint?'

COMMON CHALLENGES & SOLUTIONS

Device won't connect to TinkRcode:

Check that the USB cable is fully inserted in both the device and the computer. Try a different USB port. Make sure the student is logged into the correct TinkRcode account. Some school networks require granting Bluetooth permissions the first time—watch for pop-up prompts. If one device still won't connect, swap the USB cable before assuming a hardware issue.

'My screen didn't change!':

The most common cause is that the student didn't click 'Upload' or 'Run' before disconnecting. Walk through the upload process again: make the code change, click Upload, wait for the success confirmation, then test. Also confirm the device is powered via USB.

RGB values are confusing:

Remind students of the simple range: 0 means that color light is completely off. 255 means that color light is as bright as possible. Numbers in between are partial brightness. A reference chart on the board with common colors and their RGB values (red = 255,0,0; blue = 0,0,255; green = 0,255,0; yellow = 255,255,0) helps students get started and builds intuition quickly.

TinkRcode is slow to load:

This is often a school WiFi issue. Ask your IT department about whitelisting TinkRcode if this is recurring. Having students launch TinkRcode a few minutes early while you're doing the introduction can prevent delays.

ASSESSMENT OPPORTUNITIES

Look for evidence that students can:

- Explain that screens use red, green, and blue light to create color
- Successfully connect to TinkRcode and navigate the interface
- Use a Fill Screen Color block with intentional RGB values
- Predict what a new RGB combination will produce before testing it

LOOKING AHEAD

In the next lesson, students will start placing rectangles on their screen with precise X-Y coordinates. Print the Screen Grid Worksheet before Lesson 5. Students will use it to plan their rectangle positions before coding—a habit that will serve them throughout the project. This is also a great moment to revisit the coordinate work from Lesson 1.

LESSON 5: DRAW RECTANGLE

LESSON OVERVIEW

Students connect coordinate knowledge from Lesson 1 to programming by learning to place colored rectangles at specific positions on their Tech-A-Sketch screen. The Draw Rectangle block requires four arguments: position (X, Y), size (width, height) and color (RGB). Students use the Screen Grid Worksheet to plan their shapes before coding, then test and refine their work through four programming challenges.

This lesson bridges math and coding in a tangible way. When students realize that the X-Y coordinate from their grid game is the same concept that places a shape on a screen, it's a rewarding connection. The Screen Grid Worksheet makes the planning process visual and helps students develop the habit of designing before coding—a skill that will serve them in every lesson ahead.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Use the Draw Rectangle block to place a colored shape on the screen
- Identify the four argument types in the block: position, size and color
- Plan a shape placement using the Screen Grid Worksheet
- Complete 4 programming challenges using coordinates and color

LESSON DURATION

45–60 minutes (Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Screen Grid Worksheet printouts (one per student)
- Computer or device with TinkRcode access

TIPS FOR SUCCESS

Planning on Paper First

Students who skip the Screen Grid Worksheet and jump straight to TinkRcode tend to get more frustrated. They change random values without a clear goal and can't figure out what went wrong. Insist on at least a quick sketch before coding begins. Even placing one planned rectangle on the worksheet dramatically improves results and reduces wasted time.

Explaining Arguments

The concept of 'arguments'—the specific values you pass into a code block to customize its behavior—is new and important. Use an analogy: 'It's like ordering food. You don't just say I want a sandwich. You say what kind of bread, what filling and how big. The rectangle block has arguments for each of those things.' For this block: tell the program WHERE it goes (X, Y), how BIG it is (width, height) and what COLOR it is (R, G, B).

Using the Screen as a Grid

The Tech-A-Sketch screen has specific pixel dimensions. Showing students the Screen Grid with those dimensions labeled helps them understand the relationship between coordinates and visible position. A good discussion prompt: 'If you want a rectangle in the center of the screen, what X and Y values would you use?'

Making the Challenges Progressive

The four challenges build in complexity. Students who move through them quickly can be challenged further: 'Make a house using only rectangles' or 'Make a checkerboard pattern.' Students who struggle with the first challenge may need to revisit coordinate basics with the Screen Grid before continuing.

WHAT TO DOCUMENT

- Completed Screen Grid Worksheets with planned shapes
- Photos of screens showing successfully placed rectangles
- Discovery Journal entries
- Assessment responses about arguments and coordinates

EXTENSIONS FOR CURIOUS MINDS

- 'Can you make a scene using only rectangles? A sunset? A cityscape? A flag?'
- 'What is the smallest rectangle you can make? The largest?'
- Challenge students to create a symmetrical pattern by calculating mirror coordinates
- Ask: 'What would happen if you put a rectangle at (0,0) with a very large width and height?'

COMMON CHALLENGES & SOLUTIONS

Rectangle appears off-screen or in an unexpected location:

Ask the student to look at their X and Y values and compare to the Screen Grid. Common issues: X and Y are swapped (check the coordinate order) or the value is larger than the screen dimensions. Walking students back to their Grid Worksheet usually reveals the problem immediately.

Rectangle is the wrong size:

Remind students that the size arguments represent width and height in pixels—not position. A rectangle at position (50, 50) with size (100, 50) extends from X=50 to X=150 and from Y=50 to Y=100.

Screen fills with one color and no rectangle is visible:

This happens when the fill screen color and the rectangle color are the same. Change one of them to create contrast. Also check that the Fill Screen block comes before the rectangle block in the Setup sequence—shapes placed before the fill will be covered by it.

Students are frustrated by small adjustments:

Celebrate the debugging process. 'You found out what doesn't work—that's exactly how programmers learn.' Guide students to change one value at a time so they can isolate what each change does.

ASSESSMENT OPPORTUNITIES

Look for evidence that students can:

- Identify all the argument types in the Draw Rectangle block
- Accurately predict where a rectangle will appear based on its X and Y values
- Plan a shape on the grid before coding it
- Complete at least two of the four challenges independently

LOOKING AHEAD

Lesson 6 introduces the potentiometer knobs and how to code a movable brush. This is a significant milestone in the project—students will start drawing freely on their screens for the first time. No additional materials are needed, but make sure all assembled Tech-A-Sketch devices are ready to go with programs saved from this lesson.

LESSON 6: SKETCH

LESSON OVERVIEW

This is the lesson where Tech-A-Sketch truly comes to life. Students learn about potentiometers—the electronic components under their knobs—then code a brush that they can move around the screen by turning those knobs. By the end of the lesson, they're drawing freehand on a digital screen and sharing their artwork in the first Gallery Walk of the project.

This lesson is a major payoff moment. Students have been learning concepts and building skills for several lessons. Now those concepts combine into something genuinely fun: a functional drawing tool. When students turn a knob and watch a brush move across their screen, the abstract idea of code controlling hardware becomes real and personal.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Explain how a potentiometer converts knob rotation into an electronic signal
- Code a brush using position, size and color
- Use variables to store X and Y position values from the knobs
- Move a brush across the screen by turning the knobs
- Complete 4 programming challenges and participate in a Gallery Walk

LESSON DURATION

60–75 minutes (Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Computer or device with TinkRcode access

TIPS FOR SUCCESS

Introducing Potentiometers

'Potentiometer' is a wonderful, interesting word that students enjoy saying (pronounced 'puh-TEN-shee-AH-muh-ter'). Make a moment of it! Explain it simply: 'Under each knob is an electronic component that measures how far the knob has been turned. When you turn the knob to the right, more electricity flows through it. When you turn it left, less flows. Your microcontroller reads that amount of electricity as a position number.'

Using Serial Print in TinkRcode to show students the numbers changing as they turn the knobs is a powerful demonstration. Students can see the value go up as they turn right and down as they turn left. The concept of a potentiometer becomes immediately concrete.

Variables as Stored Values

This lesson formally introduces variables. Use a concrete analogy: 'A variable is like a labeled box. You put a value inside and give the box a name. Whenever you need that value, you use the box's name instead of writing the number.' For position tracking, students create an X variable and a Y variable to store the brush's current location. The key insight: when variables store potentiometer readings, the brush position updates automatically as the knobs are turned.

Coding the Brush Together

Walk through coding the brush as a class before students experiment independently. The sequence matters here: create variables for X and Y, read the potentiometer values into those variables, place the brush at the X,Y position, then put this in the Loop so it runs continuously. Getting each step in the right place is important and worth checking before students diverge.

Gallery Walk

Build in time for this! Don't skip it. The Gallery Walk is when students see that their peers made different artistic choices, feel proud of their own work, and get inspired by what's possible. A simple structure works well: 'Walk around, look at everyone's drawing, find one thing you want to try.' Keep it brief but give it space—it's more important than it might seem.

This is also a great time to introduce the Portfolio Folder concept. Students can photograph their drawings with their device camera to create a record of their work throughout the project.

WHAT TO DOCUMENT

- Photos of students' first freehand drawings on their Tech-A-Sketch
- Gallery Walk photos showing multiple projects on display
- Student reflections: 'I'm most proud of...'
- Discovery Journal entries

EXTENSIONS FOR CURIOUS MINDS

- 'Can you change your brush size? What happens when you use a very large brush?'
- Challenge students to add a second brush with a different color and size
- Ask: 'What would happen if you used the same knob for both X and Y?'
- Explore what Serial Print shows when the knobs are turned to their extreme positions

COMMON CHALLENGES & SOLUTIONS

Brush doesn't move when knobs are turned:

Check that the potentiometer reading blocks are connected to the variable and that the loop is running. Use Serial Print to confirm values are changing as the knobs turn. If values aren't changing, verify the pin numbers match the device.

Brush jumps or moves erratically:

This is often caused by the variable reading a range that's too sensitive. Check the sample code for the recommended mapping range, or reach out to TinkRworks support with a screenshot of the code.

Drawing leaves unwanted trails:

That's intentional at this stage! The brush draws continuously as it moves. Lifting the virtual brush (pen up/down) will be introduced in the Buttons lesson. For now, turning the knob back quickly or filling the screen resets the canvas.

Variable concepts are confusing:

Return to the labeled box analogy. Have students physically hold a piece of paper with 'X = ?' written on it and act out changing the value as they turn a knob. Concrete, physical representation helps variables click for students who are struggling with the abstract concept.

ASSESSMENT OPPORTUNITIES

Look for evidence that students can:

- Explain what a potentiometer does in their own words
- Describe why variables are useful—they store values that can change

- Successfully move a brush using knob input
- Explain what happens in the Loop and why it's necessary for continuous drawing

LOOKING AHEAD

Lesson 7 adds button functionality: students will code a clear-screen button and a toggle for picking up and putting down the brush. No additional materials are needed before class. You may want to briefly point out the buttons and toggle on the device so students can identify them quickly when the lesson begins.

LESSON 7: BUTTONS

LESSON OVERVIEW

Students add two powerful new features to their Tech-A-Sketch: a button that clears the screen and a toggle switch that picks up or puts down the virtual brush. To code these features, students learn about conditional statements—one of the most important concepts in all of programming. The lesson ends with two programming challenges and a Gallery Walk where students share a picture drawn using all the features they've coded so far.

Conditional statements (if this is true, then do this) appear throughout programming. This lesson introduces them in a context students immediately understand: pressing a button. When you press a button, something should happen. When you don't press it, nothing changes. That cause-and-effect relationship makes conditionals intuitive in a way that abstract explanations can't match.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Explain what a conditional statement does
- Code a button to clear the screen when pressed
- Code a toggle switch to pick up or put down the virtual brush
- Describe the difference between a button (momentary) and a toggle (latching)
- Complete 2 programming challenges and participate in a Gallery Walk

LESSON DURATION

60–75 minutes (Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Computer or device with TinkRcode access

TIPS FOR SUCCESS

Introducing Conditionals

The conditional statement is the foundation of interactive programming. Keep the introduction concrete: 'A conditional is like a decision. IF something is true, THEN do this. Otherwise, skip it.' Give everyday examples: 'IF it's raining, THEN bring an umbrella. IF your backpack is empty, THEN pack your books.' Let students generate their own examples before connecting to the Tech-A-Sketch.

Then make it physical: 'IF the button is pressed (that's the condition), THEN clear the screen (that's the action). If the button is NOT pressed, nothing happens.' Acting this out in the classroom helps students feel the logic before they code it.

Buttons vs. Toggles

Help students understand the physical difference before coding. A button bounces back up when released (momentary). The toggle stays down until pressed again (latching). Have students press each one on the physical device before writing any code—the physical difference makes the coding distinction much clearer.

For the clear-screen button: the action should trigger once when pressed, not continuously. For the toggle: it holds its state (up or down) and you use that position to determine whether the brush is drawing or lifted.

Building on Previous Code

This lesson adds to the existing program rather than starting fresh. Help students understand where new blocks should go relative to what they've already built. The conditional for the button belongs in the Loop, after the brush code. Getting the sequence right matters, and it's worth a quick check before students start coding independently.

WHAT TO DOCUMENT

- Short video of a student pressing the clear-screen button and then drawing again
- Gallery Walk photos
- Student explanations of what a conditional statement does
- Discovery Journal entries

EXTENSIONS FOR CURIOUS MINDS

- 'Can you code a second button to do something different? What would you have it do?'
- Challenge: 'What if the button cleared the screen AND changed the background color at the same time?'
- Ask: 'How is the toggle different from the clear button in how it affects your drawing process?'
- Explore: what other everyday devices use toggles vs. momentary buttons?

COMMON CHALLENGES & SOLUTIONS

Button clears screen continuously instead of once:

This usually means the conditional is inside the wrong part of the loop or is checking for a held state rather than a single press. Review where the conditional is placed and how the button state is being read. Comparing to the sample code structure can help identify the issue.

Toggle doesn't seem to work:

Check pin numbers and confirm the toggle block type is correctly connected. Use Serial Print to verify the toggle is reading as 0 (up) or 1 (down) as it's pressed.

Conditional concepts are confusing:

Return to the everyday examples. Act it out: stand up and say 'IF I clap my hands, THEN everyone stand up. Otherwise, stay seated.' Physical enactment of conditionals makes the logic click for students who are struggling with the abstract concept.

'I accidentally cleared my drawing!':

Normalize this. It's a natural consequence of having a clear button. Channel it into a design discussion: 'Is this how you want the button to work? Could you add something to make accidental clearing less likely?' This is a real engineering design consideration.

ASSESSMENT OPPORTUNITIES

Look for evidence that students can:

- Write an if/then statement in plain language describing their button code
- Explain the difference between a button and a toggle
- Correctly place a conditional in the Loop
- Demonstrate the button and toggle working as intended

LOOKING AHEAD

Lesson 8 introduces the toolkit—a way to organize all brushes students have created and cycle through them using a button. Encourage students to think about what kinds of brushes they'd want to have available before next class.

LESSON 8: TOOLKIT

LESSON OVERVIEW

Students learn to organize their brushes into a virtual toolkit that stores all their drawing tools in one place. They create a toolkit variable, add their existing brushes to it, use a button to cycle through tools while drawing, and create new brushes to add to the collection. The lesson ends with two programming challenges and a Gallery Walk.

The toolkit is where Tech-A-Sketch starts feeling like a real, professional drawing tool. Instead of one brush, students now have a collection they can switch between on the fly. The organizational concept—storing related items together under a name—is a simplified form of data structures, and when students see that organizing their brushes makes their drawing program more powerful, the value of structure in code becomes clear.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Create a toolkit variable and add brushes to it
- Understand how variables can store collections of items
- Add at least one new brush to the toolkit independently
- Use button cycling to switch between tools while drawing
- Complete 2 challenges and participate in a Gallery Walk

LESSON DURATION

45–60 minutes (Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Computer or device with TinkRcode access

TIPS FOR SUCCESS

The Toolkit Concept

The toolkit is like a drawer in an artist's desk. Rather than having individual brushes scattered around, they're all organized in one place and you can grab the one you want. In code, this is a variable that holds multiple items. Walk students through the structure: create the toolkit in Setup, add each brush to it, then use a button in the Loop to cycle through.

Creating New Brushes

The 'More Brushes' portion of this lesson lets students work independently to add new tools. Encourage creativity: what size? What shape (rectangle, oval, line, spray)? What color? Remind students they'll be using these tools in Gallery Walks, so making tools they enjoy drawing with is worth the time.

If a student has an idea for a brush but isn't sure how to make it, that's a great coaching opportunity: 'Let's figure it out together. What blocks would you need?'

Portfolio Folder Reminder

This is a good moment to remind students to photograph their drawings before the Gallery Walk. The Portfolio Folder builds throughout the project and becomes a record of their creative and coding

progress. 'Before we do the Gallery Walk, take a photo of your current drawing and save it to your Portfolio Folder.'

WHAT TO DOCUMENT

- Screenshots of students' toolkits in TinkRcode
- Photos of drawings made with multiple brushes
- Gallery Walk photos
- Discovery Journal reflections on what brushes students created and why

EXTENSIONS FOR CURIOUS MINDS

- 'How many different brushes can you fit in your toolkit?'
- Challenge: create one brush for fine details and one for filling large areas
- Ask: 'How is organizing brushes in a toolkit like organizing files in folders on a computer?'
- Create a set of brushes for different artistic purposes: outline, fill, texture, accent

COMMON CHALLENGES & SOLUTIONS

Toolkit not cycling correctly:

Check that all brushes are added to the toolkit in Setup and that the cycling button is correctly connected in the Loop. The cycle follows the order brushes were added, so the sequence matters.

New brush doesn't appear when cycling:

Confirm that the new brush was added to the toolkit variable in Setup, not just defined. A brush needs to be both created and added to the toolkit to appear when cycling.

Students are unsure what to make for a new brush:

Offer specific prompts: 'What size do you want?' 'What if you made a brush that draws a really thin line?' 'What about a big, soft-looking oval?' Sometimes a specific constraint frees students from decision paralysis.

LOOKING AHEAD

Lesson 9 introduces backgrounds—students will use shapes and subroutines to create coded background images that appear before they start drawing. Print the Screen Grid Worksheet before class. Students will use it to plan their background designs, so having one ready per student is important.

LESSON 9: BACKGROUNDS

LESSON OVERVIEW

Students learn to create coded background images using draw shapes blocks (rectangles, ovals, triangles and lines) combined with subroutines. They first explore how to use all available shape types, then learn about subroutines as a way to organize groups of code. Students independently create a snowman scene to practice, then design their own backgrounds. The lesson includes two challenges, a Gallery Walk and background design tips.

This is one of the most creatively expansive lessons in the project. With shapes and color, students can create anything: cityscapes, ocean scenes, outer space, sports fields, winter landscapes. The screen grid and planning-on-paper approach helps students translate their vision into coordinates before coding. Subroutines make complex backgrounds manageable by grouping related code together under a single name.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Use draw shapes blocks to create rectangles, ovals, triangles and lines
- Explain what a subroutine is and why it's useful
- Create an independent snowman scene using multiple shapes
- Design a coded background using the Screen Grid Worksheet for planning
- Complete 2 challenges and participate in a Gallery Walk

LESSON DURATION

60–75 minutes (Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Screen Grid Worksheet printouts (one per student)
- Computer or device with TinkRcode access

CLASSROOM SETUP

- Print Screen Grid Worksheets before class
- Display the Backgrounds Guide tips from the slide deck for students to reference during design time

TIPS FOR SUCCESS

The Value of Planning on Paper

Students who skip the Screen Grid Worksheet and jump straight to coding will spend more time frustrated than those who take five minutes to sketch on paper first. Make this expectation visible and consistent: 'I'm going to give you five minutes with the grid. Use this time.' When students have planned even rough coordinates for their major shapes, the coding goes much faster and the results are more satisfying.

The Backgrounds Guide in the slide deck offers specific tips for organizing shapes. Remind students to place larger background shapes first (sky, ground, sea) before adding smaller details (trees, buildings, fish) so nothing important gets covered up.

Introducing Subroutines

A subroutine is a named group of code blocks you can call from anywhere in your program. It's like writing a recipe once and being able to follow it any number of times without rewriting it. Use the snowman example directly: if you want multiple snowmen in your background, you could write the snowman code once (three circles, two stick arms, a hat) and call it multiple times at different positions. That's far more efficient than duplicating every shape.

The Snowman Activity

The snowman is a great independent challenge because it has a clear visual target—students know what a snowman looks like—but no prescribed solution. Every snowman will look different. This is an excellent test of whether students can combine multiple shapes independently.

Circulate during this time and offer specific, technical guidance: 'Your top circle needs to be smaller—what if you change the size from 50 to 25?' Pair encouragement with concrete suggestions.

Background Design Tips to Share

Key tips from the Backgrounds Guide that are worth highlighting with students:

- Layer from back to front—background fill first, foreground details last
- Use contrasting colors so shapes are visible against the background
- Plan at the grid level before coding individual pixel values
- Start with the largest shapes, then add detail

WHAT TO DOCUMENT

- Screen Grid Worksheets with planned backgrounds
- Photos of completed backgrounds on the Tech-A-Sketch screen
- Gallery Walk photos
- Snowman activity results
- Discovery Journal entries

EXTENSIONS FOR CURIOUS MINDS

- Challenge: recreate the background style of a favorite painting or artwork using only shapes
- Ask: 'How would you add a sunset gradient using overlapping colored rectangles with slightly different colors?'
- Create a background for a specific artistic theme: ocean floor, outer space, a sports field
- 'Can you use a subroutine to repeat a design element—like trees—multiple times across your background?'

COMMON CHALLENGES & SOLUTIONS

Background shapes are in the wrong positions:

Ask students to compare their planned coordinates on the Screen Grid Worksheet to what they coded. Most position errors trace back to a missing or transposed coordinate argument. Walking through the grid together almost always reveals the issue.

Shapes are covered by the background color:

Remind students to place their Fill Screen Color block before all shape blocks in Setup. If the fill runs after the shapes, it paints over them.

Subroutine concept is confusing:

Focus on the practical benefit rather than the abstract concept. 'Instead of writing your snowman code five times, you write it once and give it a name. Then you just say draw snowman wherever you want one.' Hands-on experience with the snowman activity makes this click faster than any explanation.

Student designs are too ambitious:

This is wonderful! Help students prioritize: 'What are the three most important parts of your background? Let's start with those.' Complex backgrounds can always be added to across multiple sessions. Remind students that saving their work means they can continue next time.

ASSESSMENT OPPORTUNITIES

Look for evidence that students can:

- Use at least three different shape types in their background
- Explain what a subroutine does and when they'd use one
- Plan a design on the Screen Grid before coding it
- Layer shapes correctly—background fill before shape details

LOOKING AHEAD

Lesson 10 introduces widgets: special tools that modify pixels already on the screen (darkening, lightening, scrambling, texturizing, smudging) plus an eraser tool. No additional materials needed, but encourage students to save a good drawing to their Portfolio Folder before next class, since the eraser will be used during the lesson.

LESSON 10: WIDGETS

LESSON OVERVIEW

Students learn to code widgets: special tools that modify the pixels already on the screen rather than drawing new ones. The five widget types are Darkener, Lightener, Scrambler, Texturizer, and Smudger. Students also code an eraser. One programming challenge and a Gallery Walk close out the lesson.

Widgets add a layer of creative sophistication that surprises and delights students. Drawing is one thing—modifying a drawing in place is another. The Scrambler and Texturizer especially produce unexpected, artistic results that students want to show their classmates immediately. This lesson reinforces that programming can be a creative, exploratory process, not just a procedural one.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Describe what widgets do and how they differ from brushes
- Add at least one widget to the toolkit
- Code an eraser tool
- Complete a programming challenge and participate in a Gallery Walk

LESSON DURATION

30–45 minutes (Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Computer or device with TinkRcode access

TIPS FOR SUCCESS

What Makes Widgets Different

Widgets are distinct from brushes in a key way: brushes add color to the screen, while widgets modify color that's already there. Make this distinction explicit before students start. 'Your brushes paint new colors. Widgets change the colors you already have. Let's experiment to see what each one does.'

Let students try all five widget types without being too prescriptive. The Darkener and Lightener have clear and predictable effects. The Scrambler, Texturizer, and Smudger produce more interesting and surprising results. Give students time to simply play and observe.

The Eraser

The eraser is a specific kind of widget that restores pixels to the background color. Students who want precise control over their drawings will find this especially useful. Help students understand: the eraser removes brush strokes, but it won't erase shapes that were coded in Setup. Those are part of the background program.

Widget Sizes

Widget blocks have size options rather than pixel dimensions because widgets require more processing power than brushes. The five preset size options let students choose a widget size that balances visual effect with device performance. Larger widgets cover more area but can slow the program slightly.

WHAT TO DOCUMENT

- Photos of drawings with widget effects applied
- Gallery Walk photos showing creative widget use
- Student descriptions of what each widget type does
- Discovery Journal entries

EXTENSIONS FOR CURIOUS MINDS

- 'What happens if you use the Scrambler multiple times on the same area?'
- Challenge: use the Darkener to create a shadow effect in a drawing
- Ask: 'Can you use the Smudger to blend two colors together?'
- Create a drawing that uses at least three different widgets as part of the artistic process

COMMON CHALLENGES & SOLUTIONS

Widgets seem to do nothing:

Check that the widget is in the toolkit and connected to a button. Also confirm there's already something drawn on the screen for the widget to modify. A blank screen won't show a visible widget effect.

Eraser removes too much at once:

The eraser size may be too large. Try a smaller size option to give more precision over what gets erased.

Students don't understand the widget size options:

Try each size briefly on screen and show the visual difference. 'Size 1 is like a pencil eraser. Size 5 is like a big sponge.' The visual comparison makes the options intuitive.

ASSESSMENT OPPORTUNITIES

Look for evidence that students can:

- Explain the difference between a brush and a widget
- Successfully add a widget to their toolkit
- Describe what each widget type does to the pixels it touches
- Use the eraser purposefully to modify a specific area

LOOKING AHEAD

Lesson 11 introduces the accelerometer—one of the most memorable features of the project. Students will code their device to detect shaking and tilting, using those physical gestures to interact with their drawing. No extra prep is needed, but you may want to briefly review accelerometer concepts beforehand so you feel comfortable guiding the discussion.

LESSON 11: ACCELEROMETER

LESSON OVERVIEW

Students learn what an accelerometer is and how it detects orientation and motion in 3D space. They then code their Tech-A-Sketch to read accelerometer values, clear the screen when the device is shaken (mimicking the classic Etch A Sketch erase gesture), and partially erase when the device is tilted.

This lesson has a natural 'wow' factor: when students pick up their device and shake it, and their drawing disappears just like the original Etch A Sketch, the room lights up. That connection between physical movement and digital result is powerful. It also introduces an important concept: programs can receive input from many different types of sensors, not just buttons and knobs.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Explain how an accelerometer measures orientation using the force of gravity
- Describe X, Y, and Z dimensions in physical space
- Read accelerometer values using Serial Print
- Code a shake-to-clear-screen feature
- Code a tilt-based partial erase

LESSON DURATION

45–60 minutes (Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Computer or device with TinkRcode access

TIPS FOR SUCCESS

Making the Accelerometer Concrete

The accelerometer concept is one of the more abstract in the project. The 'tiny weight in a box' analogy from the slides is genuinely helpful here. The weight shifts when the box tilts—a sensor detects how much the weight is pressing against each wall.

Physical demonstration helps enormously. Hold a large clear container with a marble inside and tilt it: 'See how the marble moves? The accelerometer has something similar, but microscopic.' This gives students a physical intuition for gravity-based orientation sensing before it appears as a concept in code.

X, Y, and Z Dimensions

Most students understand X (horizontal) and Y (vertical) from previous lessons. Z—depth, the in-and-out direction—is new. Using Serial Print to show live accelerometer values as you tilt and shake the device makes all three dimensions real and visible. 'What direction do the numbers change when I tilt it this way? What about this way?'

The Shake-to-Clear Feature

This is the moment students have been looking forward to. When they shake their Tech-A-Sketch and the drawing vanishes just like the original toy, it's deeply satisfying. Let students enjoy it!

Technically, shake detection reads when accelerometer values exceed a threshold quickly. The threshold may need small adjustments depending on device sensitivity. If students find the screen clears too easily, help them increase the threshold value. If it's not clearing at all, lower it and test again.

Serial Print for Debugging

Before students code their shake threshold, have them shake and tilt the device while Serial Print shows live values. 'What numbers do you see when you shake it hard? What do you see when it's sitting still on the table? Your threshold should be between those values.' This makes threshold-setting feel logical rather than arbitrary.

WHAT TO DOCUMENT

- Short videos of students shaking their Tech-A-Sketch to clear the screen
- Serial Print screenshots showing accelerometer values
- Student reflections on how the accelerometer works
- Discovery Journal entries

EXTENSIONS FOR CURIOUS MINDS

- 'Can you code the device to do something different when you tilt it left vs. right?'
- Challenge: use X, Y, and Z axes together to detect a specific motion and trigger an action
- Ask: 'Where else do devices use accelerometers?' (Phones rotating their screen, step counters, car airbags)
- Explore: 'What if the shake threshold is set very low? Very high? What are the tradeoffs?'

COMMON CHALLENGES & SOLUTIONS

Screen clears too easily or not at all:

Adjust the threshold value. Use Serial Print to check what accelerometer readings a normal shake produces vs. casual movement, and set the threshold between those values. This is a debugging exercise in itself.

Serial Print values are confusing:

Focus on the pattern rather than individual numbers. 'What direction do the numbers go when you tilt left? What about when you shake hard?' Pattern recognition is more useful here than memorizing specific values.

3D dimension concepts are confusing:

Stick to physical demonstrations: point forward/back (Z), left/right (X), up/down (Y). Have students physically act out each axis with arm movements before it appears in code. This makes 3D coordinates feel natural and grounded.

ASSESSMENT OPPORTUNITIES

Look for evidence that students can:

- Explain how the accelerometer detects orientation
- Identify the three dimensions (X, Y, Z) in physical space
- Set an appropriate threshold for shake detection
- Use Serial Print to observe and interpret sensor values

LOOKING AHEAD

Lesson 12 adds two finishing touches: text on the screen and a freeze drawing feature. No extra materials are needed. Encourage students to save a drawing they're proud of to their Portfolio Folder before the next class.

LESSON 12: ADDITIONAL FEATURES

LESSON OVERVIEW

Students add two finishing features to their Tech-A-Sketch: the ability to write text on the screen and a 'freeze drawing' function that temporarily captures and holds their current artwork. The lesson includes two sets of programming challenges (Challenges A and Challenges B) and a Gallery Walk.

These features round out the Tech-A-Sketch experience and give students creative tools to personalize their work. Writing their name or a title on the screen adds an artistic dimension that connects to digital art and design. The freeze feature is particularly satisfying: students can 'save' a snapshot of a drawing they're proud of before continuing to work or clearing the canvas.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Use the Write Text block to display text on the screen
- Explain how text on the screen behaves like a shape (coded into Setup)
- Code a Freeze Drawing feature using a button
- Complete two Challenge A activities and one Challenge B activity
- Participate in a Gallery Walk

LESSON DURATION

30–45 minutes (Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Computer or device with TinkRcode access

TIPS FOR SUCCESS

Text Behaves Like a Shape

The key concept here: text written with the Write Text block is placed in Setup, which means it appears as part of the background. Like shapes, it won't be affected by the eraser but can be drawn over. Help students understand this by having them position text on the screen before they start drawing, then watch what happens as they draw over it. Seeing the behavior is more effective than explaining it in advance.

The Write Text block uses similar arguments to the Draw Rectangle: position (X, Y), what to write, font size and color. Walk through these arguments together before students code independently.

Connecting to the Portfolio Folder

Connect the Freeze Drawing feature to the Portfolio Folder concept students have been building. 'You've been photographing your drawings throughout this project. Now you can also freeze a drawing inside your program to hold it in place while you keep working.' This gives students a sense of how digital tools handle saving and capturing work in progress.

Challenge B Choice

Challenge B offers students a choice among three different options. This is intentional—different students will find different challenges interesting or accessible. Let students choose based on what they want to explore. All three challenges are valid and lead to genuine learning.

WHAT TO DOCUMENT

- Photos of devices showing student names and text on screen
- Freeze Drawing demonstrations (short videos work well)
- Gallery Walk photos
- Portfolio Folder check-ins
- Discovery Journal entries

EXTENSIONS FOR CURIOUS MINDS

- 'Can you create a title screen that shows your name and a short message before you start drawing?'
- Challenge: use text and shapes to create a logo for your Tech-A-Sketch
- Ask: 'What other information might you want to display as text on your screen?'
- Create a drawing 'signature' that always appears in a corner of your canvas

COMMON CHALLENGES & SOLUTIONS

Text isn't visible:

Check the text color against the background color. If both are the same, the text won't show. Change one of them. Also verify that the X, Y position is within the screen dimensions.

Freeze Drawing doesn't work:

Confirm the freeze feature is connected to a button in the Loop and that the button assignment doesn't conflict with another existing function.

Students unsure which Challenge B option to choose:

Offer a brief description of each option and what it explores. Choice with brief context usually resolves the paralysis quickly. All three options are equally valid—there's no 'better' choice.

LOOKING AHEAD

Lesson 13 (Custom Brushes) is optional. If time allows, it's a wonderful creative extension that lets students build brushes made of multiple combined shapes. If you're moving toward the final session, you can go directly to Lesson 14. Encourage students to spend any time between lessons adding to their Portfolio Folders.

LESSON 13: CUSTOM BRUSHES

(Optional)

LESSON OVERVIEW

Students learn to create custom brushes built from multiple shapes. Unlike the basic brushes from earlier lessons (rectangle, oval, line, spray), custom brushes can be any composition of shapes: a flower brush, a star stamp, an emoji, or any design a student imagines. The lesson includes four programming challenges and a Gallery Walk.

This optional lesson is a wonderful creative extension for classes with time to spare. Custom brushes transform the drawing experience: instead of making marks with simple shapes, students now create marks with their own inventions. Gallery Walks after this lesson often feature the most visually interesting student work of the entire project.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Create a custom brush variable
- Build a brush using multiple shapes within a New Brush block
- Understand how relative positions (offset from the brush center) work in custom brushes
- Use transparency to create layered brush effects
- Complete 4 challenges and participate in a Gallery Walk

LESSON DURATION

45–60 minutes (Optional, Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Computer or device with TinkRcode access

TIPS FOR SUCCESS

Planning Custom Brushes on Paper

Custom brushes benefit from paper planning more than any other feature. Students who sketch their brush design first—what shapes, what sizes, what positions relative to the center—have a much clearer starting point than those who try to build directly from scratch in TinkRcode.

Prompt students: 'Draw what you want your brush to look like on paper first. What shapes make it up? What color is each part? How big is the overall brush?'

Relative Positions

The most conceptually challenging part of custom brushes is understanding relative positions. A custom brush defines shapes positioned relative to a center point (0,0 in the brush's own coordinate space). When the brush is placed at X=100, Y=75 on the screen, all shapes inside it shift accordingly.

A simple way to explain this: 'The brush has its own tiny grid. The center of the brush is (0,0). You position each shape relative to that center point. Then wherever you put the brush on screen, all those shapes move together.'

Transparency

Transparency is introduced in this lesson. Setting a shape's transparency within a custom brush creates layering effects. This is an optional layer of complexity. Students who want to explore it can, and students who are happy with solid colors can skip it and still make excellent custom brushes.

WHAT TO DOCUMENT

- Photos of custom brush designs on paper alongside their digital versions
- Gallery Walk photos showing creative custom brush artwork
- Screenshots of custom brush code
- Portfolio Folder additions
- Discovery Journal entries

EXTENSIONS FOR CURIOUS MINDS

- 'Can you make a brush that looks like a specific emoji?'
- Challenge: create a brush that makes a tile-pattern when used repeatedly
- Ask: 'How would you make a brush that looks like a snowflake?'
- Create brushes that pair thematically with your coded background (ocean scene: wave brush, fish brush, etc.)

COMMON CHALLENGES & SOLUTIONS

Custom brush doesn't look as expected:

Bring students back to their paper plan: 'Where is each shape relative to the center of your brush?' Then translate those positions into code one shape at a time. Most discrepancies trace back to relative position values being too large or in the wrong direction.

Brush looks different from the paper plan:

Sizes may need adjustment. What looks proportional on paper sometimes needs scaling in actual code. Encourage iteration: 'Let's adjust the size of that circle and see what it looks like.' This is normal design iteration, not failure.

Transparency isn't working as expected:

Transparency works on a 0–255 scale (255 = fully opaque, 0 = fully transparent). Start with a middle value around 128 to see a clear blending effect, then adjust from there.

LOOKING AHEAD

Lesson 14 is the final programming session. Students will do a code review, make final decisions about button assignments and toolkit contents, and complete their Tech-A-Sketch programs. Encourage students to think about what they want their final buttons to do before the next class.

LESSON 14: FINAL PROGRAMMING

LESSON OVERVIEW

Students bring everything together in the final programming session. The lesson begins with a code review—students examine their complete program and verify that all major features are working. Then students make their final design decisions: which brushes belong in their toolkit, what each button does, and how their Tech-A-Sketch reflects their creative vision. The project is complete.

This lesson has a different feel from earlier ones. It's quieter, more reflective, and more personal. Students are no longer learning new concepts—they're finalizing creative choices about the project they built and programmed over many sessions. The focus shifts from 'here's how to do this' to 'what do you want this to be?' That shift is meaningful. It marks students' transition from learners following instructions to creators making intentional decisions.

LEARNING OBJECTIVES

By the end of this lesson, students will be able to:

- Review and verify all major program features (brushes, buttons, background, accelerometer)
- Make intentional final decisions about button assignments and toolkit contents
- Complete the Tech-A-Sketch program
- Reflect on the programming concepts learned throughout the project

LESSON DURATION

30–60 minutes (Programming)

MATERIALS NEEDED

- Assembled Tech-A-Sketch (from kit)
- Computer or device with TinkRcode access

TIPS FOR SUCCESS

The Code Review

Begin with a structured code review. Walk through the key sections of the program together: Setup (background, text, toolkit initialization) and Loop (brush movement, button conditionals, accelerometer). For each section, ask: 'Is this working the way you want? Is there anything you want to change?' This review helps students catch lingering issues before they move into final decisions, and it reinforces the overall structure of their program. Many students will be surprised by how much they've built.

Final Button Decisions

Students have 4 buttons and 1 toggle. The toggle is typically assigned to brush up/down. That leaves 4 buttons to assign. Options typically include: clear screen, cycle through toolkit brushes, freeze drawing, and any other coded feature. Encourage students to think about what they actually want while drawing—what's most important to have easy access to? There's no wrong answer here. The goal is intentional, considered choice.

The Final Programming Worksheet (referenced in the email support materials) is a great tool here: students document what each button does so they have a record of their own program.

Defining Done

A Tech-A-Sketch is complete when it works as the student intends it to work. A simpler program that functions perfectly is more successful than a complex program full of bugs. Help students identify their personal definition of done: 'What needs to be working for this to feel complete?' Then hold that as the target.

Portfolio Folder Completion

Before any final showcase or Gallery Walk, have students take a final look at their Portfolio Folders and select their best pieces. This review connects the whole project arc and helps students see and articulate their growth from the first coordinate games to the final programmed drawing device.

WHAT TO DOCUMENT

- Final program screenshots
- Student design documents listing button assignments
- Portfolio Folders with the full collection of drawings and snapshots
- Video of a completed Tech-A-Sketch in action
- Student reflections on the project

EXTENSIONS FOR CURIOUS MINDS

- 'What feature would you add if you had more time?'
- Challenge: write an artist statement explaining your design choices and what you're most proud of
- Ask: 'What programming concept from this project would you most want to use again?'
- Explore: could you recreate a famous artwork digitally using your Tech-A-Sketch as a tool?

COMMON CHALLENGES & SOLUTIONS

Students want to keep adding features and never feel done:

This is a wonderful problem. Set a clear endpoint: 'At the end of today, your project needs to be complete enough to demonstrate to someone who hasn't seen it. What's the minimum that needs to work for that?' Then help them prioritize toward that target.

A key feature has stopped working:

Go back to basics: is the relevant block in the right place (Setup vs. Loop)? Is it connected to the right button? Use Serial Print to check what's happening step by step. Most issues are connection or placement problems, not fundamental errors.

Students feel their project isn't as good as their classmates':

Redirect: 'What are YOU most proud of in your program?' Every Tech-A-Sketch reflects a unique creative vision. A project with a simpler toolkit but a beautiful background is no less valuable than one with complex features. Celebrate what each student did accomplish.

PREPARING FOR REUSE

If you plan to use Tech-A-Sketch with another group of students, now is the time to begin the breakdown and storage process. Refer to the Tech-A-Sketch Reusability Guide (found in your Additional Materials folder) for complete step-by-step instructions on disassembly and storage.

As you break down projects, sort components back into their original packaging using the bags and inserts you saved during the Build lesson. Key storage notes:

- PCB goes back into the anti-static bag

- Small hardware (bolts, spacers, lock washers, cap nuts, knob inserts, knobs, toggle button) goes in the plastic bag
- Top panel and bottom panel go back in the box
- USB-C Power Cable stores separately

All Tech-A-Sketch components are reusable for future classes. Additional student licenses will be needed for each new group. Please reach out to support@tinkrworks.com to prepare for your next delivery.

Taking Inventory: Before storing kits, do a quick component check for each project. If you find missing or damaged parts, contact support@tinkrworks.com now. It's much easier to restock before your next delivery than to discover gaps when you're ready to teach again.

CELEBRATION IDEAS

This lesson marks the completion of the Tech-A-Sketch project. Many teachers follow it with a showcase or gallery experience. Here are some ideas:

- Host a final Gallery Walk where students display and demonstrate their projects
- Invite families in for a showcase where students explain their design choices
- Have students present their Portfolio Folders and share their favorite pieces
- Arrange a visit to another class where students demonstrate and explain their Tech-A-Sketch
- Create a class video where each student shows their device and describes one thing they programmed

REFLECTION QUESTIONS

Help students process and celebrate their learning with questions like:

- Looking at your Portfolio Folder, which piece are you most proud of? Why?
- What was the most challenging part of this project? How did you work through it?
- What's your favorite feature you programmed? Why?
- What surprised you about coding or digital art?
- If you could add one more feature, what would it be?

FINAL THOUGHTS

You've guided your students through an extraordinary learning journey. They arrived not knowing what a pixel was, and they're leaving with a hand-held color drawing device they built and programmed themselves. They understand X-Y coordinates, RGB colors, variables, conditionals, loops, subroutines and accelerometers—not as vocabulary words on a test, but as tools they used to make something real.

Each Tech-A-Sketch is different because each student is different. Their creative choices live in the code they wrote. That's something worth celebrating.

From the entire TinkRworks team, thank you for bringing hands-on STEAM learning to your students. You're making a real difference.