# Random Sprite Challenge

**Minimum experience:** Grades 3+, 1st year using Scratch, 4th quarter or later

## At a Glance

### Overview and Purpose

Coders create a randomly generate sprite and then review how to add costumes to a sprite to simulate lifelike movements or animations in a scene or short story. The purpose of this project is to learn how to better simulate motion/animations of a newly created sprite.

### Objectives and Standards

| Process objective(s): | Product objective(s): |
|---|---|
| **Statement:**<br>● I will review how to create several costumes in an original sprite to simulate motion.<br>● I will learn how to use a variety of blocks to simulate a newly created sprite's motion.<br>**Question:**<br>● How can we create several costumes in an original sprite that are used to simulate motion?<br>● How can we use a variety of blocks to simulate a newly created sprite's motion? | **Statement:**<br>● I will create a project that simulates lifelike motion of an original sprite (or several sprites).<br>**Question:**<br>● How can we create a project that simulates lifelike motion of an original sprite (or several sprites)? |
| **Main standard(s):** | **Reinforced standard(s):** |
| **1B-AP-10** Create programs that include sequences, events, loops, and conditionals<br>● Control structures specify the order (sequence) in which instructions are executed within a program and can be combined to support the creation of more complex programs. Events allow portions of a program to run based on a specific action. For example, students could write a program to explain the water cycle and when a specific component is clicked (event), the program would show information about that part of the water cycle. Conditionals allow for the execution of a portion of code in a program when a certain condition is true. For example, students could write a math game that asks multiplication fact questions and then uses a conditional to check whether or not the answer that was entered is correct. Loops allow for the repetition of a sequence of code multiple times. For example, in | **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process.<br>● Decomposition is the act of breaking down tasks into simpler tasks. For example, students could create an animation by separating a story into different scenes. For each scene, they would select a background, place characters, and program actions. (source)<br>**1B-AP-15** Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended.<br>● As students develop programs they should continuously test those programs to see that they do what was expected and fix (debug), any errors. Students should also be able to successfully debug simple errors in programs created by others. (source)<br>**1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations. |

a program that produces an animation about a famous historical character, students could use a loop to have the character walk across the screen as they introduce themselves. (source)

**1B-AP-13** Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences.

- Planning is an important part of the iterative process of program development. Students outline key features, time and resource constraints, and user expectations. Students should document the plan as, for example, a storyboard, flowchart, pseudocode, or story map. (source)

- People communicate about their code to help others understand and use their programs. Another purpose of communicating one's design choices is to show an understanding of one's work. These explanations could manifest themselves as in-line code comments for collaborators and assessors, or as part of a summative presentation, such as a code walk-through or coding journal. (source)

## Practices and Concepts

**Source**: K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org.

| Main practice(s): | Reinforced practice(s): |
|---|---|
| **Practice 3: Recognizing and defining computational problems**<br>• "The ability to recognize appropriate and worthwhile opportunities to apply computation is a skill that develops over time and is central to computing. Solving a problem with a computational approach requires defining the problem, breaking it down into parts, and evaluating each part to determine whether a computational solution is appropriate." (p. 77)<br>• **P3.2.** Decompose complex real-world problems into manageable subproblems that could integrate existing solutions or procedures. (p. 77)<br>**Practice 5: Creating computational artifacts**<br>• "The process of developing computational artifacts embraces both creative expression and the exploration of ideas to create prototypes and solve computational problems. Students create artifacts that are personally relevant or beneficial to their community and beyond. Computational artifacts can be created by combining and modifying existing artifacts or by developing new artifacts. Examples of computational artifacts include programs, simulations, visualizations, digital animations, robotic systems, and apps." (p. 80)<br>• **P5.2.** Create a computational artifact for practical intent, personal expression, or to address a societal issue. (p. 80)<br>• **P5.3.** Modify an existing artifact to improve or customize it. (p. 80) | **Practice 6: Testing and refining computational artifacts**<br>• "Testing and refinement is the deliberate and iterative process of improving a computational artifact. This process includes debugging (identifying and fixing errors) and comparing actual outcomes to intended outcomes. Students also respond to the changing needs and expectations of end users and improve the performance, reliability, usability, and accessibility of artifacts." (p. 81)<br>• **P6.1.** Systematically test computational artifacts by considering all scenarios and using test cases." (p. 81)<br>• **P6.2.** Identify and fix errors using a systematic process. (p. 81)<br>**Practice 7: Communicating about computing**<br>• "Communication involves personal expression and exchanging ideas with others. In computer science, students communicate with diverse audiences about the use and effects of computation and the appropriateness of computational choices. Students write clear comments, document their work, and communicate their ideas through multiple forms of media. Clear communication includes using precise language and carefully considering possible audiences." (p. 82)<br>• **P7.2.** Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose. (p. 82) |
| **Main concept(s):** | **Reinforced concept(s):** |
| **Modularity**<br>• "Modularity involves breaking down tasks into simpler tasks and combining simple tasks to create something more complex. In early grades, students | **Algorithms**<br>• "Algorithms are designed to be carried out by both humans and computers. In early grades, students learn about age-appropriate algorithms from the real world. |

learn that algorithms and programs can be designed by breaking tasks into smaller parts and recombining existing solutions. As they progress, students learn about recognizing patterns to make use of general, reusable solutions for commonly occurring scenarios and clearly describing tasks in ways that are widely usable." (p. 91)

- **Grade 5 -** "Programs can be broken down into smaller parts to facilitate their design, implementation, and review. Programs can also be created by incorporating smaller portions of programs that have already been created." (p. 104)

**Program Development**

- "Programs are developed through a design process that is often repeated until the programmer is satisfied with the solution. In early grades, students learn how and why people develop programs. As they progress, students learn about the tradeoffs in program design associated with complex decisions involving user constraints, efficiency, ethics, and testing." (p. 91)
- **Grade 5 -** "People develop programs using an iterative process involving design, implementation, and review. Design often involves reusing existing code or remixing other programs within a community. People continuously review whether programs work as expected, and they fix, or debug, parts that do not. Repeating these steps enables people to refine and improve programs." (p. 104)

As they progress, students learn about the development, combination, and decomposition of algorithms, as well as the evaluation of competing algorithms." (p. 91)

- **Grade 5 -** "Different algorithms can achieve the same result. Some algorithms are more appropriate for a specific context than others." (p. 103)

**Control**

- "Control structures specify the order in which instructions are executed within an algorithm or program. In early grades, students learn about sequential execution and simple control structures. As they progress, students expand their understanding to combinations of structures that support complex execution." (p. 91)
- **Grade 5 -** "Control structures, including loops, event handlers, and conditionals, are used to specify the flow of execution. Conditionals selectively execute or skip instructions under different conditions." (p. 103)

| Scratch Blocks | |
|---|---|
| **Primary blocks** | Control, Motion, My Blocks |
| **Supporting blocks** | Events, Looks, Sound |

| Vocabulary | |
|---|---|
| **Iterative** | • Involving the repeating of a process with the aim of approaching a desired goal, target, or result (source)<br>• Iteration is a single pass through a group of instructions. Most programs contain loops of instructions that are executed over and over again. The computer iterates through the loop, which means that it repeatedly executes the loop. (source)<br>• The computational practice of developing a little bit, then trying it out, then developing some more. (source) |
| **Modularity** | • The characteristic of a software/web application that has been divided (decomposed) into smaller modules. An application might have several procedures that are called from inside its main procedure. Existing procedures could be reused by recombining them in a new application (source) |
| **Parallel** | • Refers to processes that occur simultaneously. Printers and other devices are said to be either parallel or serial. Parallel means the device is capable of receiving more than one bit at a time (that is, it receives several bits in parallel). Most modern printers are parallel. (source) |

| | |
|---|---|
| | • The computational concept of making things happen at the same time. (source) |
| **Remix** | • The process of creating something new from something old. Originally a process that involved music, remixing involves creating a new version of a program by recombining and modifying parts of existing programs, and often adding new pieces, to form new solutions. (source)<br>• A creative work that is derived from an original work (or from another remix). A remix typically introduces new content or stylistic elements, while retaining a degree of similarity to the original work. (source) |
| **Storyboard** | • Like comic strips for a program, storyboards tell a story of what a coding project will do and can be used to plan a project before coding. |
| **More vocabulary words from CSTA** | • Click here for more vocabulary words and definitions created by the Computer Science Teachers Association |

## Connections

| | |
|---|---|
| **Integration** | **Potential subjects:** History, language arts, media arts, physical education, science, social studies<br><br>**Example(s):** This project could be modified to animate a historical, contemporary, or fictional sprite in a specific time period or location. For example, animating a sprite or scene from a story read in class. As another example, animating a historical figure or event from a time period and culture coders are learning about. Rather than having the example project give values for the parts of a creature, you could create a project similar to the randomized synthesis project that uses words to describe a sprite's features, location, time period, culture, etc. that a coder will then create and animate in Scratch. Click here to see other examples and share your own ideas on our subforum dedicated to integrating projects or click here for a studio with similar projects. |
| **Vocations** | Scientists and researchers often create models or simulations of environments in order to better understand the processes and systems at play. In this project we are creating a fictional model or simulation of a sprite's movement. Click here to visit a website dedicated to exploring potential careers through coding. |

## Resources

- Example project
- Video walkthroughs
- Quick reference guides
- Project files
- Sample storyboard questions

# Project Sequence

## Preparation (20+ minutes)

| Suggested preparation | Resources for learning more |
|---|---|
| **Customizing this project for your class (10+ minutes):** Remix the project example to include your own parts of an animal, insect, monster, etc. | • BootUp Scratch Tips<br>  ○ Videos and tips on Scratch from our YouTube channel<br>• BootUp Facilitation Tips |

| | |
|---|---|
| **(10+ minutes)** Read through each part of this lesson plan and decide which sections the coders you work with might be interested in and capable of engaging with in the amount of time you have with them. If using projects with sound, individual headphones are very helpful.<br><br>**Download the offline version of Scratch:** Although hopefully infrequent, your class might not be able to access Scratch due to Scratch's servers going down or your school losing internet access. Events like these could completely derail your lesson plans for the day; however, there is an offline version of Scratch that coders could use when Scratch is inaccessible. Click here to download the offline version of Scratch on to each computer a coder uses and click here to learn more by watching a short video. | ○ Videos and tips on facilitating coding classes from our YouTube channel<br>● Scratch Starter Cards<br>○ Printable cards with some sample starter code designed for beginners<br>● ScratchEd<br>○ A Scratch community designed specifically for educators interested in sharing resources and discussing Scratch in education<br>● Scratch Help<br>○ This includes examples of basic projects and resources to get started<br>● Scratch Videos<br>○ Introductory videos and tips designed by the makers of Scratch<br>● Scratch Wiki<br>○ This wiki includes a variety of explanations and tutorials |

| **Getting Started (10-15+ minutes)** | |
|---|---|
| **Suggested sequence** | **Resources, suggestions, and connections** |
| **1. Review and demonstration (2+ minutes):**<br>Begin by asking coders to talk with a neighbor for 30 seconds about something they learned last time; assess for general understanding of the practices and concepts from the previous project.<br><br>Explain that today we are going to simulate motion (or an animation) for a randomly generated sprite and then create a dance, story, or game with that sprite. | **Practices reinforced:**<br>● Communicating about computing<br><br>**Video:** Project Preview (0:40)<br>**Video:** Lesson pacing (1:48)<br><br>This can include a full class demonstration or guided exploration in small groups or individually. For small group and individual explorations, you can use the videos and quick reference guides embedded within this lesson, and focus on facilitating 1-on-1 throughout the process.<br><br>**Example review discussion questions:**<br>● What's something new you learned last time you coded?<br>○ Is there a new block or word you learned?<br>● What's something you want to know more about?<br>● What's something you could add or change to your previous project?<br>● What's something that was easy/difficult about your previous project? |
| **2. Discuss (7+ minutes):**<br>Display and demonstrate the challenge project (or your own remixed version). Quickly draw a sprite (use vector mode) with the randomly generated body parts. Have coders talk with each other about how what kind of project they might create with the randomly generated sprite. Indicate that each coder will run the random sprite generator on their own, so we should all end up with different sprites. | **Practices reinforced:**<br>● Communicating about computing<br><br>**Note:** Discussions might include full class or small groups, or individual responses to discussion prompts. These discussions which ask coders to predict how a project might work, or think through how to create a project, are important aspects of learning to code. Not only does this process help coders think logically and creatively, but it does so without giving away the answer. |

| | |
|---|---|
| After the discussion, coders will begin working on their project as a class, in small groups, or at their own pace. | **Example discussion questions:**<br>● What kind of project could we create with this randomly generated sprite?<br>● What kind of blocks might we use to animate this sprite?<br>○ What kind of costumes would the sprite have?<br>○ How would the sprite move?<br>■ What kind of code would we use to simulate this movement? |
| **3. Log in (1-5+ minutes):**<br>If not yet comfortable with logging in, review how to log into Scratch and create a new project.<br><br>If coders continue to have difficulty with logging in, you can create cards with a coder's login information and store it in your desk. This will allow coders to access their account without displaying their login information to others. | **Alternative login suggestion:** Instead of logging in at the start of class, another approach is to wait until the end of class to log in so coders can immediately begin working on coding; however, coders may need a reminder to save before leaving or they will lose their work.<br><br>**Why the variable length of time?** It depends on comfort with login usernames/passwords and how often coders have signed into Scratch before. Although this process may take longer than desired at the beginning, coders will eventually be able to login within seconds rather than minutes.<br><br>**What if some coders log in much faster than others?** Set a timer for how long everyone has to log in to their account (e.g., 5 minutes). If anyone logs in faster than the time limit, they can open up previous projects and add to them. Your role during this time is to help out those who are having difficulty logging in. Once the timer goes off, everyone stops their process and prepares for the following chunk. |

| **Project Work (70-75+ minutes; 2+ classes)** | |
|---|---|
| **Suggested sequence** | **Resources, suggestions, and connections** |
| **4. Create a random sprite (15+ minutes)**<br>***5+ minute review and demonstration***<br>Review the various vector tools in Scratch (see this video) to create a new sprite. Have everyone open up a second tab/window and navigate to the Random Sprite Challenge. Ask coders to click the green flag and then the backdrop, draw the sprite in vector mode in their own project, and then give the sprite a name.<br><br>***9+ minute drawing***<br>Give coders time to create randomly generated sprite. Encourage peer-to-peer assistance and facilitate 1-on-1 as needed. If coders finish their costumes early, encourage them to begin working on their storyboard. | **Standards reinforced:**<br>● **1B-AP-12** Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features<br>● **1B-AP-13** Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences.<br>**Practices reinforced:**<br>● Creating computational artifacts<br>● Recognizing and defining computational problems<br>● Testing and refining computational artifacts<br>**Concepts reinforced:**<br>● Program development<br><br>**Video:** Create a random sprite (1:39)<br>**Quick Reference Guide:** Click here<br><br>**Video:** Image editor: Vector mode (5:00) |

| | |
|---|---|
| | **A note on using the "Coder Resources" with your class:** Young coders may need a demonstration (and semi-frequent friendly reminders) for how to navigate a browser with multiple tabs. The reason why is because kids will have at least three tabs open while working on a project: 1) a tab for Scratch, 2) a tab for the Coder Resources walkthrough, and 3) a tab for the video/visual walkthrough for each step in the Coder Resources document. Demonstrate how to navigate between these three tabs and point out that coders will close the video/visual walkthrough once they complete that particular step of a project and open a new tab for the next step or extension. **Although this may seem obvious for many adults, we recommend doing this demonstration the first time kids use the Coder Resources and as friendly reminders when needed**. |
| **5. Create a storyboard (10-15+ minutes):**<br>Walk through the process of creating a storyboard by asking the following questions, then giving coders time to document their answers through physical or digital means:<br>1. What kind of project are you going to create?<br>a. A dance?<br>b. A game?<br>c. A story?<br>d. Something else?<br>2. What kind of animations will your randomly generated sprite have?<br>3. What other sprites and backdrops will you include in your project?<br>a. What will each of these sprites do?<br>i. What algorithms can you create to do that?<br>4. Will users be able to interact with your project?<br>a. If so, how?<br><br>When coders are ready, have them show you their storyboard and ask questions for clarification of their intent (which may change once they start coding and get more ideas). If approved, they may continue on to the next steps (logging in and creating their story); otherwise they can continue to think through and work on their storyboard.<br><br>**Note:** Coders may change their mind midway through a project and wish to rethink through their original storyboard. This is part of the design process and it is encouraged they revise their storyboard to reflect their new ideas. | **Standards reinforced:**<br>● **1B-AP-13** Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences<br>**Practices reinforced:**<br>● Communicating about computing<br>**Concepts reinforced:**<br>● Program development<br>● Modularity<br><br>**Resource:** Example storyboard templates<br>**Resource:** Storyboard questions for displaying<br><br>**Note:** Some coders do really well with open projects, while others thrive within constraints. It may make more sense to suggest a range of sprites and backdrops so coders aren't overwhelmed with possibilities. This can also help with better predicting how long it might take to create the story.<br><br>**Storyboarding Tip:** Coders can color their storyboard (or mark with symbols) what they know, have questions about, and don't know. For example: mark something green if coders know how to create the algorithm for that sprite/action; mark yellow if a coder has questions; mark red if a coder is unsure how to do something.<br><br>**Suggestion:** If coders need additional help, perhaps pair them with someone who might help them with the storyboarding process. Or, you could have coders meet with a peer to discuss their storyboard before asking to share it with yourself. This can be a great way to get academic feedback and ideas from a peer. |
| **6. Create a dance, story, or game (45+ minutes)**<br>Give coders time to animate their dance, game, or short story and ask coders to use My Blocks or message blocks to create functions for each motion or animation for each of their sprites. Encourage peer-to-peer assistance and facilitate 1-on-1 as needed. When necessary, provide friendly reminders | **Standards reinforced:**<br>● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals<br>● **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process.<br>**Practices reinforced:** |

| | |
|---|---|
| that the more costumes there are with small motions, the smoother the animations might appear.<br><br>**Note:** Coders will spend a good amount of time creating several costume variations, so expect to see the image editor up on the screen for much of the class. | • Creating computational artifacts<br>• Recognizing and defining computational problems<br>• Testing and refining computational artifacts<br>**Concepts reinforced:**<br>• Modularity<br><br>**Video:** Create a dance, story, or game (1:33)<br>**Quick Reference Guide:** Click here<br><br>**Video:** Create costumes for animations (4:22)<br>**Quick Reference Guide:** Click here<br><br>**Video:** Image editor: Vector mode (5:00)<br>**Video:** Animated gif (2:38)<br><br>**Facilitation Suggestion:** Some coders may not thrive in inquiry based approaches to learning, so we can encourage them to use the Tutorials to get more ideas for their projects; however, we may need to remind coders the suggestions provided by Scratch are not specific to our projects, so it may create some unwanted results unless the code is modified to match our own intentions. |
| **7. Add in comments (*the amount of time depends on typing speed and amount of code*):**<br>***1 minute demonstration***<br>When the project is nearing completion, bring up some code for the project and ask coders to explain to a neighbor how the code is going to work. Review how we can use comments in our program to add in explanations for code, so others can understand how our programs work.<br><br>Quickly review how to add in comments.<br><br>***Commenting time***<br>Ask coders to add in comments explaining the code throughout their project. Encourage coders to write clear and concise comments, and ask for clarification or elaboration when needed. | **Standards reinforced:**<br>• **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations<br>**Practices reinforced:**<br>• Communicating about computing<br>**Concepts reinforced:**<br>• Algorithms<br><br>**Video:** Add in comments (1:45)<br>**Quick reference guide:** Click here<br><br>**Facilitation suggestion:** One way to check for clarity of comments is to have a coder read out loud their comment and ask another coder to recreate their comment using code blocks. This may be a fun challenge for those who type fast while others are completing their comments. |

## Assessment

**Standards reinforced:**
- **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations

**Practices reinforced:**
- Communicating about computing

Although opportunities for assessment in three different forms are embedded throughout each lesson, this page provides resources for assessing both processes and products. If you would like some example questions for assessing this project, see below:

| **Summative**<br>Assessment *of* Learning | **Formative**<br>Assessment *for* Learning | **Ipsative**<br>Assessment *as* Learning |
|---|---|---|

| | | |
|---|---|---|
| The debugging exercises, commenting on code, and projects themselves can all be forms of summative assessment if a criteria is developed for each project or there are "correct" ways of solving, describing, or creating.<br><br>**For example, ask the following after a project:**<br>● Can coders debug the debugging exercises?<br>● Did coders use a variety of block types in their algorithms and can they explain how they work together for specific purposes?<br>● Did coders include descriptive comments for each event in all of their sprites?<br>● Can coders explain how they used broadcast blocks or My Blocks as functions to make their code more organized and easier to read (modularity)?<br>● Can coders explain how their project is similar to their storyboard?<br>● Did coders create at least ## randomly generated sprites with different algorithms?<br>○ Choose a number appropriate for the coders you work with and the amount of time available. | The 1-on-1 facilitating during each project is a form of formative assessment because the primary role of the facilitator is to ask questions to guide understanding; storyboarding can be another form of formative assessment.<br><br>**For example, ask the following while coders are working on a project:**<br>● What are three different ways you could change that sprite's algorithm?<br>● What happens if we change the order of these blocks?<br>● What could you add or change to this code and what do you think would happen?<br>● How might you use code like this in everyday life?<br>● See the suggested questions throughout the lesson and the assessment examples for more questions. | The reflection and sharing section at the end of each lesson can be a form of ipsative assessment when coders are encouraged to reflect on both current and prior understandings of concepts and practices.<br><br>**For example, ask the following after a project:**<br>● How is this project similar or different from previous projects?<br>● What new code or tools were you able to add to this project that you haven't used before?<br>● How can you use what you learned today in future projects?<br>● What questions do you have about coding that you could explore next time?<br>● See the reflection questions at the end for more suggestions. |

# Extended Learning

## Project Extensions

| Suggested extensions | Resources, suggestions, and connections |
|---|---|
| **Add even more (30+ minutes, or at least one class):**<br>If time permits and coders are interested in this project, encourage coders to explore what else they can create in Scratch by trying out new blocks and reviewing previous projects to get ideas for this project. When changes are made, encourage them to alter their comments to reflect the changes (either in the moment or at the end of class). | **Standards reinforced:**<br>● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals<br>**Practices reinforced:**<br>● Testing and refining computational artifacts<br>● Creating computational artifacts<br>**Concepts reinforced:**<br>● Algorithms<br>● Control<br><br>**Facilitation Suggestion:** Some coders may not thrive in inquiry based approaches to learning, so we can encourage them to use the Tutorials to |

| | |
|---|---|
| While facilitating this process, monitor to make sure coders don't stick with one feature for too long. In particular, coders like to edit their sprites/backgrounds by painting on them or taking photos, or listen to the built-in sounds in Scratch. It may help to set a timer for creation processes outside of using blocks so coders focus their efforts on coding. | get more ideas for their projects; however, we may need to remind coders the suggestions provided by Scratch are not specific to our projects, so it may create some unwanted results unless the code is modified to match our own intentions.<br><br>**Suggested questions:**<br>● What else can you do with Scratch?<br>● Can you add another random sprite to your project?<br>● What have you learned in other projects that you could use in this project? |
| **Similar projects:**<br>Have coders explore the code of other peers in their class, or on a project studio dedicated to this project. Encourage coders to ask questions about each other's code. When changes are made, encourage coders to alter their comments to reflect the changes (either in the moment or at the end of class).<br><br>Watch this video (3:20) if you are unsure how to use a project studio. | **Standards reinforced:**<br>● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals<br>● **1B-AP-12** Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features<br>**Practices reinforced:**<br>● Testing and refining computational artifacts<br>**Concepts reinforced:**<br>● Algorithms<br><br>**Note:** Coders may need a gentle reminder we are looking at other projects to get ideas for our own project, *not to simply play around*. For example, "look for five minutes," "look at no more than five other projects," "find three projects that each do one thing you would like to add to your project," or "find X number of projects that are similar to the project we are creating."<br><br>**Generic questions:**<br>● How could you add your newly created sprite to this project?<br>a. How would this change the project?<br>● What are some ways you can expand this project beyond what it can already do?<br>● How is this project similar (or different) to something you worked on today?<br>● What blocks did they use that you didn't use?<br>a. What do you think those blocks do?<br>● What's something you like about their project that you could add to your project? |
| **micro:bit extensions:**<br>**Note:** the micro:bit requires installation of Scratch Link and a HEX file before it will work with a computer. Watch this video (2:22) and use this guide to learn how to get started with a micro:bit before encouraging coders to use the micro:bit blocks.<br><br>Much like the generic Scratch Tips folder linked in each Coder Resources document, the micro:bit Tips folder contains video and visual walkthroughs for project extensions applicable to a wide range of projects. Although not required, the micro:bit Tips folder uses numbers to indicate a suggested | **Standards reinforced:**<br>● **1B-AP-09** Create programs that use variables to store and modify data<br>● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals<br>● **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process<br>● **1B-AP-15** Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended<br>**Practices reinforced:**<br>● Recognizing and defining computational problems<br>● Creating computational artifacts<br>● Developing and using abstractions |

| | |
|---|---|
| order for learning about using a micro:bit in Scratch; however, coders who are comfortable with experimentation can skip around to topics relevant to their project. | • Fostering an inclusive computing culture<br>• Testing and refining computational artifacts<br>**Concepts reinforced:**<br>• Algorithms<br>• Control<br>• Modularity<br>• Program Development<br>• Variables<br><br>**Folder with all micro:bit quick reference guides:** Click here<br>**Additional Resources:**<br>• Printable micro:bit cards<br>  ○ Cards made by micro:bit<br>  ○ Cards made by Scratch<br>• Micro:bit's Scratch account with example projects<br><br>**Generic questions:**<br>• How can you use a micro:bit to add news forms of user interaction?<br>• What do the different micro:bit event blocks do and how could you use them in a project?<br>• How could you use the LED display for your project?<br>• What do the tilt blocks do and how could you use them in your project?<br>• How could you use the buttons to add user/player controls?<br>• How might you use a micro:bit to make your project more accessible? |

## Differentiation

| Less experienced coders | More experienced coders |
|---|---|
| If coders struggle with this kind of challenge, pair them with other coders with more experience or understanding. Just make sure the lesser experienced coder "drives" the mouse and the more experienced coder can "navigate." It might also help less experienced coders if they have time to see what others are creating with the blocks; encourage coders to walk around and see what others are doing and then adding similar code in their projects. | If coders are very comfortable with this kind of challenge, pair them with other coders with less experience or understanding. Just make sure the lesser experienced coder "drives" the mouse and the more experienced coder can "navigate." |

## Debugging Exercises (1-5+ minutes each)

| Debugging exercises | Resources and suggestions |
|---|---|
| This project does not have any associated debugging challenges; however, click here for hundreds of debugging exercises. | **Standards reinforced:**<br>• **1B-AP-15** Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended<br>**Practices reinforced:**<br>• Testing and refining computational artifacts<br>**Concepts reinforced:**<br>• Algorithms<br>• Control |

| | |
|---|---|
| | **Suggested guiding questions:**<br>● What should have happened but didn't?<br>● Which sprite(s) do you think the problem is located in?<br>● What code is working and what code has the bug?<br>● Can you walk me through the algorithm (steps) and point out where it's not working?<br>● Are there any blocks missing or out of place?<br>● How would you code this if you were coding this algorithm from Scratch?<br>● Another approach would be to read the question out loud and give hints as to what types of blocks (e.g., motion, looks, event, etc.) might be missing.<br><br>**Reflective questions when solved:**<br>● What was wrong with this code and how did you fix it?<br>● Is there another way to fix this bug using different code or tools?<br>● *If this is not the first time they've coded:* How was this exercise similar or different from other times you've debugged code in your own projects or in other exercises? |

## Unplugged Lessons and Resources

Although each project lesson includes suggestions for the amount of class time to spend on a project, BootUp encourages coding facilitators to supplement our project lessons with resources created by others. In particular, reinforcing a variety of standards, practices, and concepts through the use of unplugged lessons. Unplugged lessons are coding lessons that teach core computational concepts without computers or tablets. You could start a lesson with a short, unplugged lesson relevant to a project, or use unplugged lessons when coders appear to be struggling with a concept or practice.

List of 100+ unplugged lessons and resources

Incorporating unplugged lessons in the middle of a multi-day project situates understandings within an actual project; however, unplugged lessons can occur before or after projects with the same concepts. **An example for incorporating unplugged lessons:**

Lesson 1. Getting started sequence and beginning project work
Lesson 2. Continuing project work
Lesson 3. Debugging exercises and unplugged lesson that reinforces concepts from a project
Lesson 4. Project extensions and sharing

## Reflection and Sharing

| Reflection suggestions | Sharing suggestions |
|---|---|
| Coders can either discuss some of the following prompts with a neighbor, in a small group, as a class, or respond in a physical or digital journal**.** If reflecting in smaller groups or individually, walk around and ask questions to encourage deeper responses and assess for understanding. Here is a sample of a digital journal designed for Scratch (source) and here is an example of a printable journal useful for younger coders.<br><br>**Sample reflection questions or journal prompts:**<br>● How did you use computational thinking when creating your project? | **Standards reinforced:**<br>● **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations<br>**Practices reinforced:**<br>● Communicating about computing<br>● Fostering an inclusive culture<br>**Concepts reinforced:**<br>● Algorithms<br>● Control<br>● Modularity |

- What's something we learned while working on this project today?
    - What are you proud of in your project?
    - How did you work through a bug or difficult challenge today?
- What other projects could we do using the same concepts/blocks we used today?
- What's something you had to debug today, and what strategy did you use to debug the error?
- What mistakes did you make and how did you learn from those mistakes?
- How did you help other coders with their projects?
    - What did you learn from other coders today?
- What questions do you have about coding?
    - What was challenging today?
- Why are comments helpful in our projects?
- How is this project similar to other projects you've worked on?
    - How is it different?
- How do you think the example project uses variables to select the parts of a sprite?
- More sample prompts

- Program development

**Peer sharing and learning video:** Click here (1:33)
At the end of class, coders can share with each other something they learned today. Encourage coders to ask questions about each other's code or share their journals with each other. When sharing code, encourage coders to discuss something they like about their code as well as a suggestion for something else they might add.

**Publicly sharing Scratch projects**: If coders would like to publicly share their Scratch projects, they can follow these steps:

1. **Video:** Share your project (2:22)
    a. Quick reference guide
2. **Video (Advanced):** Create a thumbnail (4:17)
    a. Quick reference guide